# The Impact of Electric Propulsion in the Design and Performance of Spacecraft

Jack Tyler

# Contents

Chapter 1

# Ideal Flight Performance and Types of Electric Propulsion System

Despite electric propulsion being developed since the inception of spaceflight, only now has an adoption for commercial, scientific and military missions been seen, after the transfer of technology from the former Soviet Union and the rise of desirable power systems in the mid-1990s. An electric propulsion system uses electrical energy, along with reaction mass (propellant), to change the velocity of a spacecraft, and unlike chemical propellant, the energy for the propulsion system is independent of the reaction mass. Indeed, this circumvents the limitations of chemical systems by ensuring that the thermodynamic relationships in chemical energy and propellant flow do not limit the overall power of the rocket. Electric propulsion systems are often largely far more efficient than their chemical counterparts, with specific impulse values typically ranging from 200s-5000s, due to their high exhaust velocity, an integral part to the operation of rocket systems.

## 1.1 Deriving Ideal Flight Performance

Specific impulse of a system is often noted by $I_{sp}$ and can be thought of as the the force with respect to the amount of propellant used per unit time, or:

$$I_{sp} = \frac{v_e}{g_0} \tag{1.1.1}$$

Along with specific impulse, performance is analysed in terms of component masses, where $m_0$ is the total mass of the craft:

$$m_0 = m_{powerplant} + m_{payload} + m_{propellant}$$

and power-plant mass is related to an electrical power output by an inverse specific power $\alpha$, which depends on the configuration of the system and power conditioning equipment. Modern, lightweight solar arrays typically hold values of 20kg/kW;

$$M_{powerplant} = \alpha W$$

values of 500-2000kg/kW are currently being researched. Also, since power is converted by the thruster into the kinetic energy of the exhaust, and allowing for losses due to an efficiency $\eta$:

$$P_e = \alpha m_{powerplant} = \frac{\frac{1}{2}\dot{m}V_e^2}{2t_b\eta} \tag{1.1.2}$$

since

$$\eta = \frac{power_{jet}}{input} = \frac{\dot{m}V_e^2}{2P_e} = \frac{\dot{w}g_0 I_{sp}}{2P_e} = \frac{F I_{sp} g_0}{2P_e} \tag{1.1.3}$$

And by combining (1.1.4) with the rocket equation:

$$e^{\frac{\Delta u}{c}} = \frac{1}{\mathbf{MR}} = \frac{m_0}{m_f}$$

an equation for the reciprocal payload mass fraction can be found, assuming a gravity and drag-free flight.

$$\frac{m_0}{m_f} = \frac{e^{\frac{\Delta u}{c}}}{1 - \left(e^{\frac{\Delta u}{c}}\right) v^2/(2\alpha t_b \eta)} \tag{1.1.4}$$

Furthermore, by grouping all parameters with units of speed, a 'characteristic velocity' can be found, where $t_b$ is equal to the burn time, and defines the speed the power-plant would have if its output were converted into kinetic energy of its own mass. A plot of this characteristic velocity, $v_c^2 = 2\alpha t_b \eta$, can then be made:

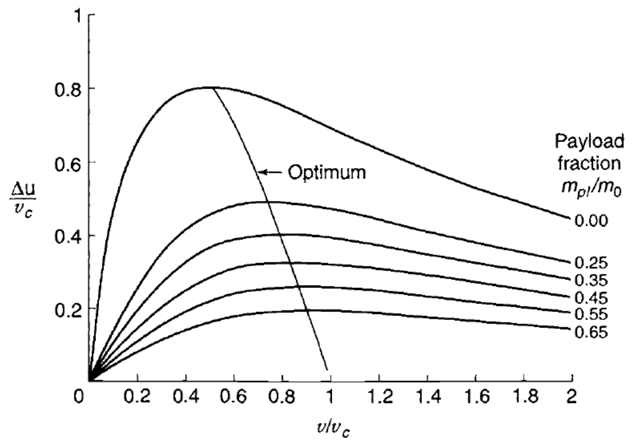


Figure 1.1: *Normalized vehicle velocity increment as a function of normalized exhaust velocity for various payload fractions.[15]*

The presence of optima shows that for a given mission, there is an optimum range of specific impulse and that, for any electric system, the optimum specific impulse is proportional to the velocity increment required. It is evident, therefore, that the selection of systems and propellant is varied –

indeed, more so than its chemical counterparts. Furthermore, since it is of obvious benefit to operate near optimum, where $v/v_c$ is approximately 1, a large value of $2t_b/\alpha$ is desirable and hence, large burn times and small values of specific power are necessary. Indeed, provided that the mission is not time critical and large burn times are acceptable, electric propulsion systems carry large improvements, particularly in overcoming translational and rotational orbit perturbations, and long-duration deep space missions, where the prospect of carrying the required propellant is deemed too costly or inefficient. Near-Earth applications are also suited to electric propulsion, given that a lower propellant requirement can reduce launch savings dramatically.

Further savings could be made, given that numerous electric propulsion systems are in operation. These systems fall under three categories: electrothermal, electrostatic, and electromagnetic.

## 1.2 Electrothermal Thrusters

Electrothermal thrusters are often regarded as the simplest type of electric propulsion. Indeed, they operate much like a chemical system, and are seen as 'true' rockets: a propellant is heated electrically, and then expanded through a nozzle to achieve a high exhaust velocity. Since thermodynamics is exploited, the thrusters can still be analysed as their chemical counterparts are:[15, 5]

$$v_e = C_f C^* \tag{1.2.1}$$

where $C_f$ is the thrust coefficient, and $C^*$ is the characteristic velocity, which is defined as follows.

$$C^* = \left\{ \gamma(\frac{2}{\gamma+1})^{\frac{\gamma+1}{\gamma-1}} \frac{M_r}{RT_c} \right\}^{-\frac{1}{2}} \tag{1.2.2}$$

Here $\gamma$ is the nozzle expansion ratio, R is the universal gas constant, $M_r$ is the molecular mass of the gas and $T_c$ is the operating temperature. For all types of Electrothermal thrusters, specific impulses, compared to other electrical counterparts, are more modest, ranging from 100s to around 2000s, but their relative simplicity is appealing to mission analysers when such high efficiencies are not required.

### Resistojet

Resistojets, as the name suggests, rely on ohmic heating through a filament to heat a propellant to high temperatures. Because of this, exhaust velocity is a function of temperature, and hence, the lifetime and performance is limited by the heating element and its degradation. Often, the heating element employs Tungsten or Ytterbium and Zirconium stabilised Platinum filaments to maximise lifetimes.[5] Since high exhaust velocities are desired, a high temperature and high thruster efficiency are required. Yet, although efficiency has been demonstrated to be close to 90%, losses to due heat radiation and improper heating are commonplace, especially when direct coupled heating – where the filament is in direct contact with the gas – is used. To overcome these, indirect heating is used, with a heat exchanger of material grain stabilised platinum, where the grain stabiliser is typically zirconium oxide. Furthermore, the heating chambers are often lined with foil to prevent radiation losses. It must be noted, however, that when indirect heating is used, less performance is achieved, although greater lifetimes emerge.

Propellant choices for resistojets are varied: any gas can be used, although to achieve the highest characteristic velocity, hydrogen would naturally be the choice due to its low molecular mass. However, cryogenic storage for hydrogen and its disassociation can be troublesome for satellites with tight mass restrictions, despite its high specific temperature and non-corrosive nature. Hence, nitrogen

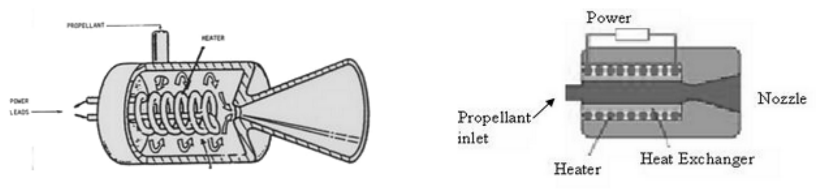


Figure 1.1: *Comparison of direct coupled and indirect heating in resistojets. Direct coupled often results in greater performances, whilst indirect yields longer mission lifetimes.[18]*

and ammonia are also regularly employed on missions, where they can be used in a backup monopropellant system or as part of the attitude control system. The use of resistojets in long-term manned missions is also a possibility, since waste $CO_2$ and water can be used in the propulsion system: Surrey small satellites tested a $CO_2$based system in 1999 which achieved 90mN of thrust at a specific impulse of 138s.[11] Multiple propellants can be used with the same design.

More recently, hydrazine has been exploited as a propellant in Power-Augmented Electrothermal Hydrazine thrusters. Hot gases from the catalytic decomposition of hydrazine reduces necessary power input – which can range from a watt to several kilowatts – due to a 35-40% increase in performance. However, consideration for the extra gases from the decomposition of hydrazine by the following equation may be needed:

$$3\,N_2H_4 \longrightarrow 4\,(1-\mathrm{x})NH_3 + (1+2x)N_2 + 6\,xH_2$$

where $x$ is the degree of ammonia disassociation, and depends on catalyst type and dwell time, among other things. The chamber pressure of a resistojet also needs to be considered, as although high pressures reduce gas disassociation losses, improve heat exchanger performance and reduce required chamber size, higher heat transfer losses and nozzle throat erosion occurs, which may limit the lifetime of the system. Compromise pressures of 15-200 psi are common. Furthermore, the temperature varies inversely with mass flow rate, and although a low flow rate is desired for minimal fuel use, the high temperatures that this causes could destroy the system, limiting the exhaust velocity of resistojets to approximately 10km$s^{-1}$. The mass flow rate can be found if the input power and exhaust velocity are known.

$$\dot{m} = \frac{2\eta P_{input}}{V_e^2} \tag{1.2.3}$$

Despite these issues, resistojets are being regularly employed on Lockheed-Martin satellites [15] and on smaller systems, since no special need for power conditioning equipment is required and high efficiencies of 70-90% are superior to other types of electrothermal and electric systems. Resistojets are most attractive for missions where low-to-mid velocity increments are required, and power limits, thrusting time and plume effects are the main drivers.

## Arcjets

### Arcjet Basics

Although arcjets and resistojets both produce thrust using thermodynamic effects, their method of doing so varies. Arcjets - hence the name - use an electric arc to heat a neutral propellant gas. By having an anode and cathode held together by a resistive material, often the boron nitride found in resistojet insulation [15], and applying a voltage that is higher than the breakdown voltage and a current higher than the holdover current, an inherently unstable arc forms. Once this occurs, electrons flow from the cathode to the anode as the gas is ionised, and, through collisions with these electrons, the gas is heated. The anode itself is often shaped to the form of a nozzle to provide the expansion.

Since the temperature of the arc can range from 10,000-20,000K, a tungsten alloy is often needed, and an azimuthal swirl injection is used, where the propellant surrounds the cathode and electric arc, preventing the arc from kinking and melting the path between the positive and negative electrodes. Furthermore, since only a small amount of the propellant is in contact with the gas when this method is used, the cool air mixture keeps the walls from achieving excessive temperatures that could shorten the lifetime of the system. Typically, the lifetime of an arcjet system depends on electrode erosion and has been noted to be between 800 and 1500 operational hours[21]. However, unlike the resistojet, the arcjet suffers from efficiencies below 50%, and has been recorded as being as low as 35%. Here, the culprit is the residual non-kinetic energy of the exhaust, although losses of 10-20%, depending on the configuration, can be attributed to the heat dissipation that is associated with all thermal systems.[19]

There are advantages of an arcjet over a resistojet, however: most notably, the exhaust velocity, using hydrogen as a propellant, can be as high as 20km$s^{-1}$, as a higher power input can be achieved, due to the gas itself being heated and not a resistance element. Furthermore, the specific impulse of such a system is approximately 1500s, far superior to the resistojet. Nonetheless, a more complex power system is required to deal with the added complexity of the arc itself, although less propellant is needed due to the higher specific impulse and thus allowing the mass gained and lost to balance.

Arcjets are less commonly found than resistojets due to their lower efficiencies and more complex nature, but can be found on systems where mid-level velocity increments with lower fuel limits are the main drivers.

## Analysis of Arcjets using a generalised Ohm's law

Since arcjets operate using a plasma, their analysis revolves around plasma physics. Gaseous conductors obey a modified version of Ohm's law in an assumption of a uniform medium:[15]

$$V = IR = (I/A) \cdot (AR/d) \cdot (d) \tag{1.2.4}$$

Where $A$ is the area, $R$ the resistance, $I$ the current and $d$ the distance by virtue of a voltage drop $V$. Furthermore, the electric field, current density and electrical conductivity can be defined as follows:

$$Electric\,Field = E = \frac{V}{d} \tag{1.2.5}$$

$$Current\ Density = j = \frac{I}{A} \tag{1.2.6}$$

$$Electrical\ conductivity = \sigma = \frac{d}{AR} \tag{1.2.7}$$

by combining all these equations, it becomes possible to write Ohm's law simply as:

$$j = \sigma E \tag{1.2.8}$$

noting that it is in scalar form. However, since arc currents are often influenced by external fields, a generalised Ohm's law in vector form is required:

$$\mathbf{j} = \sigma[\mathbf{E} + \mathbf{v} \times \mathbf{B} - (\beta/\sigma \mathrm{B}) \cdot (\mathbf{j} \times \mathbf{B})] \tag{1.2.9}$$

Here, $\mathbf{v}$ is the velocity, $\mathbf{B}$ the magnetic induction field, $\mathbf{E}$ the electric field and $\beta$ the hall parameter, which is the electron cyclotron frequency multiplied by the mean time taken for an electron to lose its momentum due to collisions, or $\omega\tau$. Furthermore, $\sigma$ is the plasma electrical conductivity, defined by:

$$\sigma = \frac{e^2 n_e \tau}{\mu_e} \tag{1.2.10}$$

where $e$ is the electron charge, $n_e$ the number of electrons, $\tau$ the mean time between collisions and $\mu_e$ electron mass. [17]

Note the last two terms in (1.2.9) are the hall electric field, which is perpendicular to the current vector and the magnetic induction field and hence the cross-product is used.

## 1.3 Electrostatic Thruster

Electrostatic systems, unlike chemical and electrothermal systems, obtain their thrust from relying on Coulomb forces to accelerate charged particles in a near-vacuum. Although electrons would seem a desirable candidate for electrostatic engines due to their relative ease of production, even at speeds approaching the speed of light their momentum is negligible and hence the thrust from an electron beam per unit area is low. Instead, heavy molecular mass atoms that have been ionised to form positive ions are used, with research ongoing into the use of “charged colloids”, with 10000 times the mass of atomic particles, although the stability of such large ions is challenging. Much like electrothermal systems, electrostatic systems can be divided further, depending on their method of ionisation:[5]

- Electron bombardment thrusters
  - Positive ions from a monatomic gas are produced by bombarding gas – typically Xenon or Mercury – with electrons emitted from a heated cathode.
- Ion contact thrusters
  - Positive ions are produced by passing propellant vapour, typically Caesium, through an 1100 °C Tungsten contact ioniser.
- Field emission/colloid thrusters
  - Droplets of propellant are charged, either positively or negatively, as they pass through an intense electric discharge.

### Deriving Relationships for Electrostatic Devices

Independent of type, an electrostatic thruster consists of various subsystems:[17, 15, 21]

- Propellant source

- A means of generation or inputting electric power
- Ionisation chamber
- Accelerating region
- A method of avoiding space-charge build-up by neutralisation of the exhaust, typically by the insertion of electrons downstream.

Exhaust velocity is a function of voltage over the accelerating chamber, $V_{acc}$, the mass of the particle $\mu$, and its charge, $e$. It is assumed the particles are singly charged. Hence, the kinetic energy of the particle is equal to the electrical energy gained through the field, on another assumption of no collisions:

$$\frac{1}{2}\mu v^2 = eV_{acc} \tag{1.3.1}$$

And hence

$$v_e = \sqrt{(}\frac{2eV_{acc}}{\mu}) \tag{1.3.2}$$

In ideal conditions, the current $I$ is equal to the sum of propellant mass carried per second by the accelerated ions:

$$I = \dot{m}(\frac{e}{\mu}) \tag{1.3.3}$$

and since thrust is equal to[15]:

$$F = \dot{m}v_{exhaust} + (\rho_{exhaust} - \rho_{ambient})A_{exhaust}$$

the thrust of an electrostatic device can be expressed as

$$F = I\sqrt{(}2\mu V_{acc}/e) \tag{1.3.4}$$

since $\rho$ is extremely low and can thus be omitted. Hence, it is evident that for a fixed voltage and current, the thrust of such a device depends on the $\frac{mass}{charge}$ ratio for the ion. Furthermore, by combining equations 1.3.4, 1.1.5, where here $\eta$ is 1, gives:

$$\frac{P}{F} = \frac{1}{2}g_0 I_{sp}$$

and along with

$$\eta_t = \frac{P_{jet}}{P_{input}}$$

the power of the accelerator region can be expressed as follows:

$$P_{acc} = IV_{acc} = \frac{[(\frac{1}{2}) \cdot (\dot{m}v^2)]}{\eta_t} \tag{1.3.5}$$

The overall efficiency of a thruster depends on not only the efficiency of the accelerator region, but also on other sources, including the energy expended in ionisation, a non-recoverable input. The energy required for ionisation is found by:

$$E_{ionisation} = \varepsilon_{ionisation} \cdot I_{flow} \tag{1.3.6}$$

Where $\varepsilon_{ionization}$ is the ionisation potential, specific to each propellant. For the reason of conserving irrecoverable losses, and also for its easy containment properties, Xenon is emerging as the prime propellant for electrostatic systems.[21]

## The Accelerator Grid and Current Density

The current density $j$ that arises from a charged particle beam has a saturation value, due to the ion cloud opposing the accelerator electric field when too many particles of the same charge pass through it. A saturation current can be derived for a plane-geometry electrode configuration[13]: here, current density in terms of the space charge density is:

$$j = \rho_e v \tag{1.3.7}$$

Furthermore, using Poisson's equation[22], the voltage in a one-dimensional space-charge region can be found:

$$\frac{\delta^2 V}{\delta x^2} = \frac{\rho_e}{\varepsilon_0} \tag{1.3.8}$$

Here, $x$ is the distance and $\varepsilon_0$ the permittivity of free space. By solving equations 1.3.2, 1.3.8 and 1.3.9 whilst applying proper boundary conditions, the Child-Langmuir law is obtained:

$$j = \frac{4\varepsilon_0}{9} \cdot \sqrt{\frac{2e}{\mu}} \cdot \frac{V_{acc}^{3/2}}{d^2} \tag{1.3.9}$$

Where $d$ is the accelerator inter-electrode distance. For Xenon, typical current density values of between 2 and 10mA/cm$^2$exist. As well as this, by assuming that the cross section is circular, such that

$$I = (\frac{\pi D^2}{4})j$$

And by combining with 1.3.10 and 1.3.5, the thrust can be written as:

$$F = (\frac{2}{9})\pi\varepsilon_0 D^2 V_{acc}^2 / d^2 \tag{1.3.10}$$

$\frac{D}{d}$ is the ratio of exhaust beam emitted diameter to accelerator electrode grid spacing, and is referred to as the *aspect ratio* of an accelerator region. Note that if multiple grids exist, the diameter D is of individual perforation holes and the distance d is the mean spacing between grids. Furthermore, $\frac{D}{d}$ cannot take a value greater than one, so stubby engine designs, with many holes, are desirable.

## Ionization Methods

While the acceleration of ions in the system is similar across all types of electrostatic thruster, the method of ionisation varies. Most systems rely on DC ionisation methods, whereas some use RF discharges. Regardless of method, the ionisation chamber is responsible for most of the mass of the device.

Ionisation through electron bombardment is a well established technology. Thermionic emission occurs at a cathode and these electrons are forced to interact with the propellant flow, at a typical

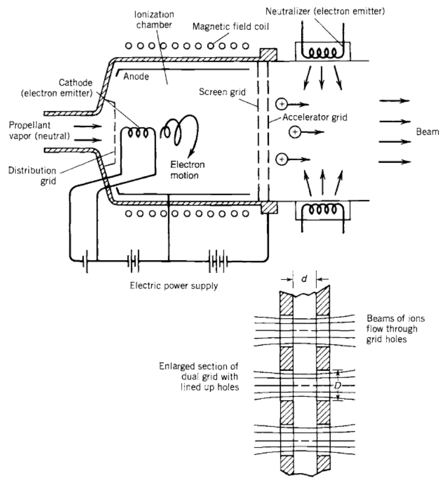

Ionization chamber
Magnetic field coil
Neutralizer (electron emitter)
Anode
Cathode (electron emitter)
Screen grid
Accelerator grid
Propellant vapor (neutral)
Beam
Distribution grid
Electron motion
Electric power supply
d
Beams of ions flow through grid holes
Enlarged section of dual grid with lined up holes
D

Figure 1.1: Schematic of an electron bombardment ion thruster, with enlarged accelerator grid shown. [15]

chamber pressure of 0.134Pa, or $10^{-3}$ torr. These emitted electrons are then attracted to the cylindrical anode, but an axial magnetic field causes them to spin and enhance collisions per electron. A radical electric field and axial electric field work in unity to remove electrons from the chamber and move ions towards the accelerator grid respectively, where the ions are electrostatically accelerated. The electrons that strike the cylindrical anode are routed through another circuit to a hot cathode at the exhaust to neutralise the beam.

Ionisation can also be done by field emission, where positive or negative ions are generated when colloids pass through a corona discharge. Alternatively, RF ionisation uses an RF discharge that, pending further research, produces a high specific impulse.

Lastly, ion contact thrusters produce ions by surface ionisation. The major boundary for this, however, is that the work function of the surface must be higher than the ionisation potential of the propellant.

## 1.4 Electromagnetic Thrusters

Electromagnetic thrusters exploit the acceleration of a propellant gas that has been heated to a plasma, a mixture of positive ions, electrons and neutral atoms. Carrying a thrust per unit area of magnitude 10-100 times higher than electrostatic systems, the Lorentz force is at the heart of the operation of such thrusters. Note that electromagnetic theory denotes that, when a current is carried perpendicular to a magnetic field, a body force perpendicular to both is created. Hence, the plasma is accelerated without need for area changes. All systems of this type follow a similar method of operation: a plasma is created, followed by a high current, often by means of an applied electric field. Finally, the plasma is accelerated to a high velocity in the direction of the thrust vector using an intense magnetic field. Although many different types of electromagnetic systems exist, the scope of this work is limited to MPD, PPT and Hall Effect thrusters.

### Magnetoplasmadynamic and Pulsed Plasma Thrusters

Based on Faraday accelerators, MPD thrusters carry a current in the direction of an electric field yet perpendicular to that of a magnetic field, providing a thrust normal to both[14]. Recall the generalised Ohm's law from equation 1.2.9, noting that it can be distinguished into a Cartesian coordinate system, with 'mass-mean velocity' in $x$, electric field in $y$, and the magnetic field in $z$, with a negligible hall parameter, $\beta$.

$$j_y = \sigma(E_y - v_x B_z) \tag{1.4.1}$$

And hence the Lorentz force becomes:

$$\bar{F_x} = j_y B_z = \sigma(E_y - v_x B_z) B_z \tag{1.4.2}$$

note that $\bar{F_x}$ does not suggest thrust; instead, it carries units N$m^{-3}$, hence "force density" inside the accelerator. Thrust is instead represented by the mass flow rate multiplied by the maximum axial velocity, $v_x$, which increases along the length of the accelerator. Indeed, since $v_x$ increases along the length of the accelerator, by equation 1.4.2, it is evident that the acceleration force decreases also, due to the $v_x B_z$ term that subtracts from the electric field. It would seem wise, then, to increase the electric field through the length of the accelerator, although optimising for peak performance would require long accelerator lengths, and practicality therefore limits $v_e$ to far less than the optimal value of $E_y/B_z$. Furthermore, MPD thrusters have an issue due to their use of constant area accelerators, which choke if the plasma velocity does not equal a precise value of[2]:

$$[(k-1)/k]/(E/B) \tag{1.4.3}$$

at the sonic location of the accelerator, where $k$ is the ratio of specific heat. In an inert gas Mach one occurs at approximately 1000m$s^{-1}$ and hence many thrusters can be limited. As well as this, due to their extremely high power requirements for optimum performance, MPD thrusters have not been explored as fully as other systems.

Instead, PPT systems are often employed, and have been used since the Soviet satellites Zond 2 and 3 in 1964[12]. Here, a plasma is accelerated after being struck between two rail electrodes. Upon doing so, the plasma created effectively completes the circuit, and the flow of the electrons causes a strong electromagnetic field, exerting a Lorentz force on the plasma. The current is generated from a capacitor, which reduces necessary power input, but requiring a recharging of the plates. However, the pulsing is usually extremely fast, and so an almost continuous thrust is created. Since the electrical energy required is stored in a capacitor, the thrust and power draw can be varied by varying the time between each discharge.

Although similar to an MPD system, the total internal accelerating force is defined differently since the Lorentz force accelerates the plasma along the rails, and is of magnitude:

$$F = sIB \tag{1.4.4}$$

Where $s$ is the rail width. For PPT, efficiency is often low. Despite having such low efficiencies, its counterpart MPD engine has flown in space only once, and operational efficiency data is as yet unknown.

### Hall Effect Thrusters

Hall effect thrusters can be described as either an electrostatic or electromagnetic device. An electrostatic potential is used to accelerate ions to high speeds, much like electrostatic engines, but the source of ionisation comes from electrons swirling azimuthally in a Hall current at the mouth of the thruster. An electron gun injects electrons into an electric and magnetic field at the open end of the thruster, creating the Hall current and subsequent electron plasma. A propellant, typically Xenon, due to its low ionisation potential and high molecular mass, is fed through an anode that also acts as a gas distributor. Xenon atoms then collide with the electrons swirling azimuthally, subsequently being ionised, and are then accelerated by the potential difference. Furthermore, because the ions pull an equal amount of electrons from the electron gun upon exiting, the exhaust plume is neutral, removing the need for added mass to counter the space-charge build-up that can occur with traditional electrostatic systems.

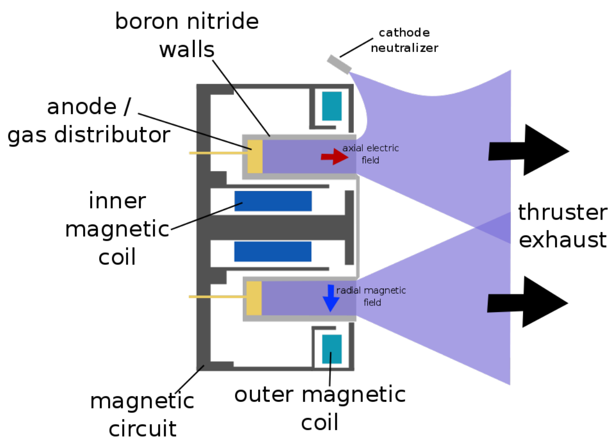


Figure 1.1: Cross-section of a symmetric Hall thruster

Note that the radial magnetic field is calibrated to a strength, typically a few hundred Gauss, such that the electrons are deflected and kept in the Hall Current, but the far higher mass Xenon ions are largely unaffected. In fact, since most electrons are trapped in the hall current, they possess a long residence time, ionising almost all of the Xenon propellant, leading to 90-99% propellant utilisation. However, despite most of the propellant remaining neutral through the operation of the thruster - as only small amounts of ionisation is needed to create the required exhaust velocity - 20-30% of the output is an electron current, which does not produce thrust. Hence, efficiencies typically reside around the 60-70% mark, with more modern thrusters being able to produce efficiency values up to 75%[7]. Hall effect thrusters have been widely used since their inception in the 1960s by the Soviet Union, and were used on the ESA mission SMART-1 to enter orbit around the moon[4], with over 200 such systems being flown in the past 30 years.

# Chapter 2

# Power Systems

Evolving trends show that, despite the increases in power system efficiency, power system requirements have been increasing. Defined at the end of life of the satellite, or EOL, average and peak power requirements are taken into account, often leaving a surplus of available power at the beginning of life - BOL - of the spacecraft. Indeed, the electric power system, henceforth EPS, must provide, but also store and distribute, any power required by the spacecraft or payload. A typical power system consists of a primary source of power, which converts 'fuel' into electrical power, coupled with a secondary source to provide power when the primary source is unavailable, such as in periods in shadow when solar panels are employed. Charge control is also a requirement of power systems.

## 2.1 Primary Power Systems

### Solar Arrays

#### Solar Array Basics

Widely regarded as the most popular primary power source for Earth-orbiting, and many interplanetary, spacecraft, they consist of a p-n junction, a boundary between two types of semiconductor. Here, $p$ is electron deficient, and $n$ is electron excess. Hence, when illuminated, photons with a suitable energy create electron-hole pairs and cause a potential to flow, with DC outputs of up to 300kW being observed. However, photons with excess energy dissipate this as heat, reducing the efficiency of solar arrays to some 12-14% for Silicon cells.[21] Maximum power-point tracking has been used on three-axis stabilised craft to maximise power output and mitigate the effects of lower efficiencies.

Despite their relatively low cost and ease of use, they suffer from radiation damage and comparatively large lifetime degradation. In most solar arrays, the n-type material is placed uppermost due to its extra radiation tolerances, and the use of Si cells with base resistivity higher than 10Ωcm help to reduce the effect of radiation-induced degradation. Furthermore, the use of a cover glass provides radiation protection, the effectiveness of which depends on an exponential absorption:[5]

$$I \sim I_0 e^{-R\rho x} \tag{2.1.1}$$

Where $x$ is the depth into glass traversed, $I_0$ the radiation fluence at $x = 0$, $\rho$ the density of the cover glass and $R$ an energy-dependent absorption coefficient. Both front and back cover glass must be considered if both side of a solar array are to be exposed. A UV coating may be applied to reject UV radiation if Cerium doping is not used in the cell itself. Furthermore, connectors between cells must be able to withstand the thermal cycling associated with shadow and daylight periods, and consideration for low-orbit drag must be taken.

Yet, with some of the highest specific power (W/kg) and lowest specific cost values ($/W)[21], solar arrays are the most popular type of power source. They do, however, become less attractive for missions that travel significant distances away from the sun where the intensity of light is diminished.

### Solar Array Sizing and Power

To configure the required size of a solar array, many factors, such as average and peak power, as well as the configuration of the spacecraft, must be considered. For basic estimates of the required solar array power and size, the following may be used[21].

$$P_{sa} = \left\{ \frac{\frac{P_e T_e}{X_e} + \frac{P_d T_d}{X_d}}{T_d} \right\} \tag{2.1.2}$$

Here, $P_{sa}$ is the power of the solar array, $P_e$ the power requirements during an eclipse, $P_d$ the requirements during daylight and $T_d$ and $T_e$ the time in daylight and eclipse respectively. Also, $X_e$ represents the efficiency of a current travelling from the array to the battery, and then onto the load. $X_d$ is the efficiency of a current travelling from array to load. Furthermore, the power of the spacecraft at BOL can be determined via:

$$P_{BOL} = P_o I_d cos\theta \tag{2.1.3}$$

where $P_0$ is the ideal cell output per unit area, found via reference tables, $I_d$ the inherent degradation and $\theta$ the angle between the normal to the array surface and the sun line, known as the cosine loss. Given factors leading to degradation, such as MMOD damage and thermal cycling, the lifetime degradation, $L_d$, can be expressed mathematically,

$$L_d = (1 - degradation/year)^{lifetime} \tag{2.1.4}$$

noting that the units are all years. Hence, by manipulating equations 2.1.3 and 2.1.4, the power output at EOL can be found:

$$P_{EOL} = P_{BOL} L_d \tag{2.1.5}$$

And hence, the required solar array area, $A_{sa}$ can be found;

$$A_{sa} = \frac{P_{sa}}{P_{EOL}} \tag{2.1.6}$$

However, solar array sizing is often more complex than this, when varying geometry, changing $\theta$ and inclinations are present. Hence, a ‘worst case’ scenario is often constructed that allows for suitable array sizing.

## Radioisotope Thermal Generators

### Introduction

Further away than Jupiter, Radioisotope Thermal Generators become the power system of choice, due to the loss of light intensity associated with travelling such a distance from the sun.

The RTG is based on the thermoelectric effect discovered by Seebeck, where a voltage is generated between two conductors if a temperature difference across them is maintained. Here, the temperature difference is created by the decay of a radioactive isotope. For such a static power system, the output is a function of absolute temperature of the hot junction and other material properties. Device efficiencies are low, often 5-10%, and disposing of excess heat can be an issue. Nonetheless, for missions into deep space or distances where solar panels are not a viable option due to distances, RTGs are the power source of choice.[1]

### Isotope Choice

The process of choosing which radioactive substance to use in the RTG revolves around four key criteria:

1. There should be a high energy release per decay.
2. Radiation must be easily absorbed. Hence, beta radiation, with Bremsstrahlung secondary radiation production, as well as gamma, is often avoided.
3. The half-life must be long enough to provide power for the length of the mission, but short enough to be able to provide suitable power levels; the longer the half-life, the lower the energy per unit time output. Indeed, for an isotope with half-life $\tau_{1/2}$, the power lost at a time $t$ is:

$$P_t = P_0 exp\left(\frac{-0.693}{\tau_{1/2}}t\right) \tag{2.1.7}$$

4. The chosen substance must produce a high amount of power per unit mass and volume.

Plutonium-238 and Strontium-90 are the most used isotopes, although many, including Caesium-137 and Americium-241, have been studied.

Where $^{238}$Pu is attractive is in its low shielding requirements and long half-life, where often the casing of the RTG is enough shielding. Often found in the form of Plutonium Oxide, $PuO_2$, it has a half-life of 87.7 years and low gamma radiation levels. However, Plutonium-238 reserves are low, with some 16.3kg remaining in the US as of September 2013. [1]

Strontium-90, although it decays by $\beta$ emission, has a negligible gamma output. However, its half-life is much shorter than Plutonium-238, at 28.8 years, with a lower decay energy. Hence, lower temperatures, and therefore lower efficiencies, are experienced. It must be noted, however, that Strontium-90 is a common product of nuclear fission and thus available in large quantities.

### Safety concerns

RTGs post a threat of contamination in the event of a fuel leak. For space applications, the largest concern is the effect on the environment that would accompany a failure of the containment system in the RTG. However, given that the risk of an accident on launch, later in ascent and in flight for the Cassini-Huygens probe were 1 in 1400, 1 in 476 and less than 1 in a million respectively, the likelihood of such an accident is low[9]. Furthermore, the likelihood of contamination given a catastrophic failure stood at 1 in 10. Fuel is also stored in modular units, surrounded by iridium and graphite blocks protected by an aero shell to prevent disintegration on re-entry.

| **Advantages** | **Disadvantages** |
|---|---|
| Power independent of orientation | Affects the radiation environment of spacecraft |
| Power independent of distance from sun | Careful handling procedures required |
| Low-power output is sustainable for long periods of time | High temperature operation needed for high efficiencies |
| Radiation damage in Van Allen belts not an issue | Interference with plasma diagnostic systems |
| Suitable for long eclipse periods | Concerns over inclusion of radioactive sources in spacecraft |

## Fuel Cells

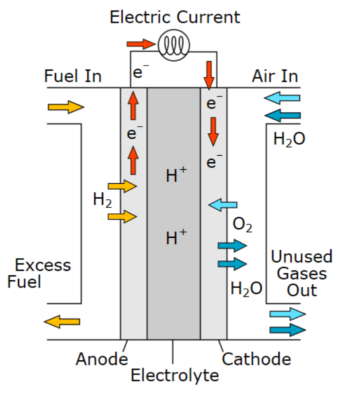


Figure 2.1: Operation principles of a fuel cell. Source: Wikimedia

Fuel cells have been the primary power source of choice for all manned US spacecraft. Ideal for low-duration, high-power missions of up to 50kW[21], they utilise the conversion of chemical energy from an oxidation reaction into electrical energy. Few unmanned missions have used fuel cells due to their relatively low operation times, but for manned missions, the ‘alkaline’ fuel cell, a $H_2/O_2$ mix (called alkaline due to the potassium hydroxide electrolyte) is most commonly employed due to its water by-product. However, regenerative fuel cells, where the water by-product is electrolysed back into its respective elements and then reused, are currently being researched. Such a technology would prove promising for future lunar bases. Voltages vary from 0.2-50kW, and the voltage at an electrode is equal to:

$$E_r = \frac{-\Delta G}{nF} \tag{2.1.8}$$

where $\Delta G$ is the change of Gibbs free energy, $n$ the number of electrons transferred in the reaction and $F$ the Faraday constant, equal to $9.65\times10^4$C/mol. Yet, this maximum voltage would rarely be observed in operation due to polarisation losses. Activation polarisation losses occur when a current is first drawn, as the reactants must be chemically absorbed onto the electrode, requiring energy to break and reform bonds and thus leading to a drop in voltage. This loss is of a quantifiable magnitude, defined by

$$\Delta V = a + b\ln J \tag{2.1.9}$$

here, $a$ and $b$ are temperature dependent constants for the reaction surface, and $J$ the current density. The current density is limited by concentration polarisation, where, at a high current, problems due to the transport of reactants to the reaction sites occur. Furthermore, since species concentrations are not uniform, a backwards EMF can be created. However, efficiencies of 50-80% are present, and operation times of up to 40000 hours are being investigated. [21]

### Dynamic Solar Heat Systems

Although not yet proven in-flight, solar heat systems that use heat generated from the Sun to drive a thermodynamic energy converter, based upon the Brayton cycle engine, have been envisioned, with power levels of up to 300kW.[10] A concentrator focuses energy onto a receiver, boiling a fluid. The heated gas then drivers a turbine, in turn driving an electrical generator. With energy conversion rates up to twice as high as photovoltaics, the required area can be reduced and thus orbit drag minimised. Latent heat also provides the energy storage for such a system, reducing the need for secondary systems. [10]

The maximum usable energy depends on the Carnot cycle efficiency, denoting the maximum thermodynamic conversion:

$$\eta = \frac{T_{hot} - T_{cold}}{T_{hot}} \tag{2.1.10}$$

Note that the temperatures are in Kelvin. Little degradation occurs when small amounts of thermal storage occurs. Such a system was considered for the space station with a 200-500kW power range.

## 2.2 Secondary systems

### Batteries

Batteries are often use to provide power for short missions or backup power for longer missions. The number of cells required is determined by the required bus voltage, and the power stored is measured in Amp or Watt-hours. The duration for which batteries need to be able to provide power varies with orbital parameters, with geostationary Earth orbit satellites experiencing up to 1.2 hours, and LEO satellites often around 40 minutes in a regular cycle. Hence, GEO satellites require fewer, deeper discharges, and LEO satellites frequent, shallow discharges. Indeed, it is the charge-discharge rate, depth of discharge and the extent of overcharging that determines the battery type. Most common is the Nickel-Cadmium cell, but Lithium-ion is quickly emerging for all applications. To determine the battery capacity, the following formula may be used:[21]

$$Cr = \frac{P_e T_e}{(DOD)Nn} W/hr \tag{2.2.1}$$

here, $P_e$ and $T_e$ is the power required during, and time of, eclipse. $(DOD)$ is the depth of discharge as a decimal. $N$ is the number of batteries that have a transmission efficiency of $n$. The capacity in A/hr can be found by dividing by the voltage.

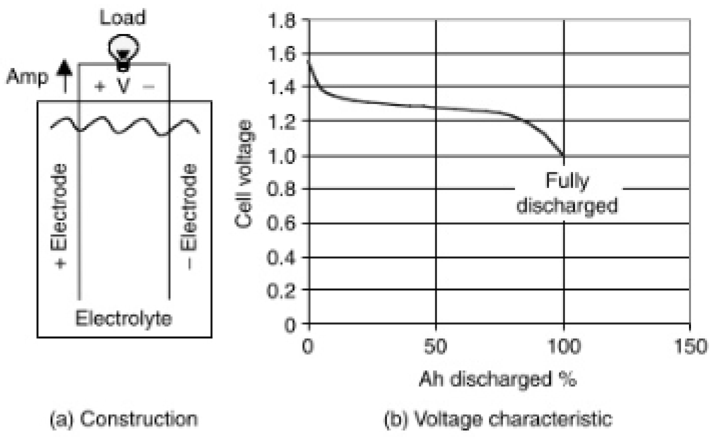


Figure 2.1: Simplified schematic and voltage characteristic of a battery. [10]

Two types of batteries exist: the primary battery and the secondary battery. Put simply, primary batteries are not rechargeable, most suited to low-power memory backups, whereas secondary cells can be recharged.

# Chapter 3

# Orbital Manoeuvres

Perhaps the greatest impact on the mission design arises from the impact of electric propulsion on orbital manoeuvres. Due to the low thrust available from such propulsion systems, traditional near-impulsive manoeuvres at apogee or perigee are not available. Instead, electrically powered spacecraft have to exploit spiralling trajectories for orbit raising, and continuous thrust for the duration of the burn. Hence, transfer times are often extremely long, rendering electric propulsion inappropriate for low-duration missions. However, over a long duration of time, the velocity increment that can be imparted onto a spacecraft by such a system can exceed that of traditional chemical systems. In this chapter two situations will be explored: the transfer of a satellite to geostationary Earth orbit, from both a geostationary transfer orbit and Low-Earth orbit, as well as the role of electric propulsion in an interplanetary transfer, in comparison to a chemical system. The MATLAB code for these can be found in the appendix.

## 3.1 Getting to Geostationary Orbit

Geostationary is the orbit of choice for communications satellites. Here, the orbital period is the same as the time required for a rotation of the Earth, some 42,164km high[20]. Both of the analyses that follow will assume an inclination change of five degrees to be undertaken. Furthermore, both systems will perform a 90 degree phasing manoeuvre over 10 orbits once in GEO.

### Chemical Propulsion

For chemical systems, getting to GEO requires insertion into a Geostationary Transfer Orbit, a highly elliptical orbit with its apogee at 42,164km and an argument of perigee that ensures apogee occurs on or near the equator. Assuming an Ariane 5 launch from French Guiana, assumptions will be made based on orbital insertion parameters from the ASTRA 5B mission. Noting that the ASTRA 5B mission was based on the Eurostar 3000 platform, which utilises chemical bipropellant and masses of 4500-6000kg[16], a craft of 5000kg utilising an $N_2O_4$/MMH mix at an exhaust velocity of 2827m$s^{-1}$will be assumed.[15]

The satellite was inserted into an orbit 42164x(249.4+6378)km, inclined at five degrees. Assuming an impulsive manoeuvre at apogee, the $\Delta$V required can be easily calculated. The inclination change will be incorporated into the impulsive apogee manoeuvre.

Taking the transfer orbit as $O_1$, at an altitude of 42164x6627.4km, and GEO as $O_2$, at 42164x42164km, the angular momenta for the orbits can be calculated:

$$h = \sqrt{2\mu}\sqrt{\frac{r_{apogee} \cdot r_{perigee}}{r_{apogee} + r_{perigee}}} \tag{3.1.1}$$

Hence

$$h_1 = \sqrt{(2 \cdot 398600)}\sqrt{\frac{42164 \cdot 6627.4}{42164 + 6627.4}} = 67570kms^{-2}$$

$$h_2 = \sqrt{\mu r} = 129640kms^{-2}$$

Therefore, the speed at apogee for $O_1$and the speed of $O_2$can be found.

$$v_1 = \frac{h_1}{r_a} = \frac{67570}{42164} = 1.6026km^{-1}$$

$$v_2 = \frac{h_2}{r} = \frac{129640}{42164} = 3.0747kms^{-1}$$

Given that the combined ΔV for a circularization and inclination change is equal to

$$\Delta V = \sqrt{V_{t,a}^2 + V_{GEO}^2 - 2V_{t,a}V_{GEO}cos\Delta i} \tag{3.1.2}$$

Which results in

$$\Delta V = \sqrt{1.6026^2 + 3.0747^2 - 2 \cdot 1.6026 \cdot 3.0747 \cdot cos5} = 1.4848kms^{-1}$$

Furthermore, to perform the phasing manoeuvre of 90 degrees over ten orbits, the following formula is needed;

$$\omega_E(10T_2) = 10 \cdot 2\pi + \Delta\Lambda \tag{3.1.3}$$

Note that $\omega$ is the angular velocity of the Earth. Substituting in the correct values yields;

$$T_2 = \frac{1}{10}\frac{90 \cdot \frac{\pi}{180} \cdot +20\pi}{72.922 \times 10^{-6}} = 88317s$$

This is the period of the slower phasing orbit that the satellite will enter into. Knowing the period can result in obtaining the semimajor axis, followed by the radius, and then onto the previous calculations of angular momentum and speed.

$$a = \left(\frac{T_2\sqrt{\mu}}{2\pi}\right)^{2/3} = \left(\frac{88317\sqrt{398600}}{2\pi}\right)^{2/3} = 42,476km$$

$$2a = r_{phasing} + r_{GEO} \Rightarrow r_{phasing} = 2 \cdot 42476 - 42164 = 42788km$$

$$h = \sqrt{2\mu}\sqrt{\frac{r_{GEO} \cdot r_{phasing}}{r_{GEO} + r_{phasing}}} = 892.86 \cdot 145.73 = 130115kms^{-2}$$

$$v_{at\,intersect} = \frac{130115}{42164} = 3.0859kms^{-2}$$

Therefore, the total $\Delta v$ required for the phase change is

$$\Delta v_{phase} = \mid 3.0859 - 3.0747 \mid + \mid 3.0747 - 3.0859 \mid = 22.4ms^{-1}$$

Finally, by use of the ideal rocket equation, the amount of propellant required for these manoeuvres can be found:

$$\frac{\Delta m}{m} = 1 - e^{\frac{-\Delta V}{v_e}} \Rightarrow \Delta m = 5000 \cdot \left(1 - e^{\frac{-1484.8+(-22.4)}{2827}}\right) = 2066.2kg$$

It is clear that the amount of propellant required for such a manoeuvre approaches half the launch mass of the craft. Given \$/kg costs to orbit are around \$13000[3], such a requirement is extremely expensive. The advantage, however, is that all of these manoeuvres take a small time when compared to electric propulsion systems.

### Electric Propulsion

Since electric propulsion systems utilises such low thrusting times, near-impulsive manoeuvres of such magnitude are impossible. Hence, for analysing the orbital transfer, an algorithm is required. The code for the following algorithm can be found at the back of this piece.

Using *LTOT.m*, and inputting information regarding the thruster input power, specific impulse and desired altitude and inclination changes, information regarding the manoeuvre is outputted. Given that the Eurostar-3000 platform also carries a Xenon Hall Effect thruster, albeit for North-South stationkeeping, data relating to the Hall Effect thruster was used. Furthermore, instead of the transfer orbit envisaged for chemical propulsion, the electrically-propelled spacecraft will begin from a circular LEO of 300x300km at an input power of 6.5kW.[1]

| | **Input Data** | | **Output Data** |
|---|---|---|---|
| Initial Orbit Altitude AMSL | 300km | Initial orbit velocity | 7725.7591 meters/second |
| Final Orbit Altitude AMSL | 35,786km | Final orbit velocity | 3074.6614 meters/second |
| Initial Inclination | 5 degrees | Thrust | 0.2651N |
| Final inclination | 0 degrees | $\Delta$m | 639.7008kg |
| Input Power | 6.5kW | Total inclination change | 5.0000 degrees |
| Thruster Efficiency | 0.7 | Total $\Delta v$ | 4698.7614 meters/second |
| Specific Impulse | 3500s | Thrust Duration | 1025.6212 days |
| Initial Spacecraft mass | 5000kg | Thrust acceleration | 0.000053 meters/second^2 |

Indeed, it is evident that the timescales involved are great, although from LEO to GEO represents a 'worst case' scenario, with a more likely scenario being injection into a MEO or higher. However, the propellant mass savings are vast, equating to several million pounds in launch costs alone. Furthermore, the operational lifetime of the satellite on station can be improved dramatically, provided long transfer times are acceptable. There are reliability issues associated with the operation of the engine for this long, however, so a more likely candidate for widespread integration of this technology into spacecraft would be the PAEHT (page 6), where the hydrazine can be used as a monopropellant system for limited manoeuvring in case of failure. However, the NASA NEXT Hall Effect thruster operated continuously for 1.3 years without incident.[6]

## 3.2 Interplanetary Missions

Given that, over long periods of time, electric systems can reach far higher velocities than their chemical counterparts, it makes sense that they be used on deep-space missions, notwithstanding

[1]The Eurostar-3000 platform is calibrated for 9-12kW of power at EOL.[16]

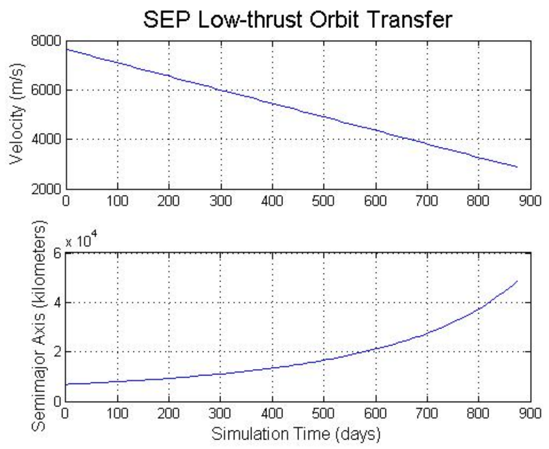


Figure 3.1: Plot of time against velocity and semi-major axis

their evident mass savings. Hence, follows an analysis of an interplanetary mission to Mars using both chemical and electric propulsion systems. Both analyses rely on a patched conics problem, and assume starting from a 300km Earth orbit (or launch hyperbola altitude for the chemical system) launched from Cape Canaveral, aboard the same Altas V 551 variant that launched MAVEN. By using MAVEN, which also launched to escape, a reliable mass for analysis can be found, at 2500kg[8]. However, for the electric system, it will be assumed that the craft is launched into a MEO of 20,200x20,200km, typical of MEO satellites, and then burn to escape. Both systems will attempt circularise into a 5000km circular orbit at Mars, and the required velocity increment then used to calculate the propellant mass required.

## Ballistic Chemical Transfer

The chemical transfer has been optimised to take advantage of the 2016 launch window to Mars. It is assumed that the Centaur upper stage has sufficient velocity increment available to perform the TMI burn, but that the satellite itself must perform the orbital insertion burn at Mars. Again, hypergolic fuels are assumed here, with the same exhaust velocity and propellant as before. Raw output and MATLAB codes for this analysis can be found later in the text. Computing departure $\Delta$v requirements yields the following:

```
departure hyperbola

-------------------

c3 11.367664 km^2/sec^2

v-infinity 3371.596588 meters/second

asymptote right ascension 251.424656 degrees

asymptote declination 9.687020 degrees
```

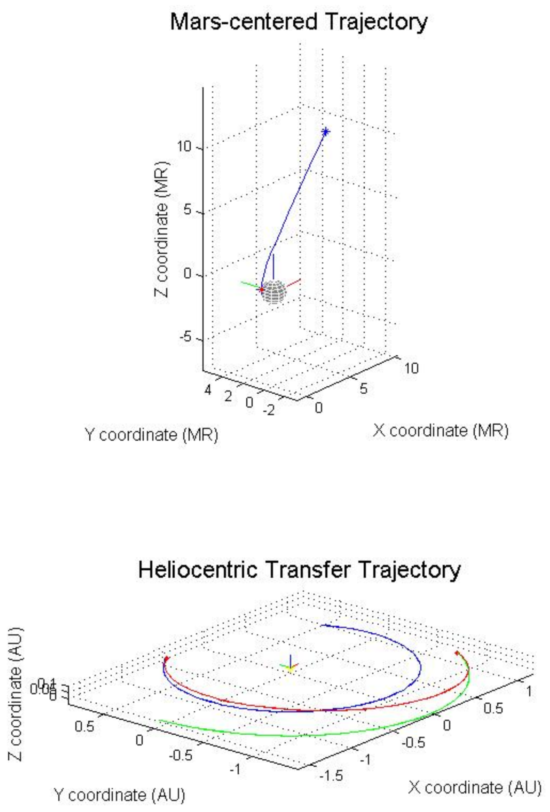


Figure 3.1: Heliocentric and Mars-centered trajectories for a chemical transfer

```
UTC calendar date 15-Jan-2016

UTC time 19:19:54.435

UTC Julian Date 2457403.30625717



hyperbolic injection delta-v vector and magnitude (Earth mean equator and equinox of J2000)

-----------------------------------------

x-component of delta-v -3336.855556 meters/sec

y-component of delta-v -1222.837781 meters/sec

z-component of delta-v -1059.748011 meters/sec
```

```
delta-v magnitude 3708.504158 meters/sec

transfer time 277.159688 days
```

This provides a hyperbolic excess velocity, $v_\infty$, of 3744.134051 meters/second. Thus, knowing $v_\infty$, the required velocity increment to enter into a circular orbit can be found via

$$\Delta v = v_{p_{hyperbola}} - v_{p_{capture}} = \sqrt{v_\infty^2 + \frac{2\mu}{r_p}} - \sqrt{\frac{\mu(1+e)}{r_p}} \tag{3.2.1}$$

Noting that $e$ is the eccentricity, which here is 1. Therefore, the required velocity increment for insertion is:

$$\Delta v = \sqrt{3744.034051^2 + \frac{2 \cdot 42828}{3493.076940}} - \sqrt{\frac{42828}{3493.076940}} = 3740.53578ms^{-1}$$

Here, two mass requirement calculations will be performed; one will assume a half-full Centaur upper stage to perform the initial injection of 3708.504158 meters/second, and another of the satellite performing the Mars injection burn at $3740.53578ms^{-1}$.

$$\Delta m_{centaur} = m\left(1 - e^{\frac{-\Delta V}{I_{sp} g_0}}\right) = 10415 \cdot \left(1 - e^{\frac{-3708.504158}{464 \cdot 9.8066}}\right) = 5804.952kg$$

$$\Delta m_{payload} = m\left(1 - e^{\frac{-\Delta V}{v_e}}\right) = 2500\left(1 - e^{\frac{-3740.53578}{2827}}\right) = 1834.259kg$$

At \$/kg to orbit values of \$13,182 on the Atlas V, some \$100.7m is spent on fuel costs alone. Clearly, this represents a large investment for any potential mission.

## Electric Burn-to-escape and Low-thrust transfer from MEO

Assuming that the electrically-powered satellite is placed into a 20,200km circular orbit, a simple burn-to-escape analysis will be performed, assuming that the Hill Sphere of the Earth is the highest stable orbit possible, at 1,500,000km. From then on, a low-thrust manoeuvre to intercept Mars will occur. A satellite of mass 2500kg, with a Hall Effect thruster at a specific impulse of 3600s and input power of 1kW are also assumed[2].

### Burn to Escape

Calculation of the inital inclination of the orbit given a launch azimuth of 93 ° at a latitude of 28.5 ° north, can be done thusly:

$$cos i = cos\phi sin\beta$$

$$\therefore i = cos^{-1}\left(cos28.5sin93\right) = 28.64°$$

Using *ltot.m*, the following burn-to-escape data is achieved:

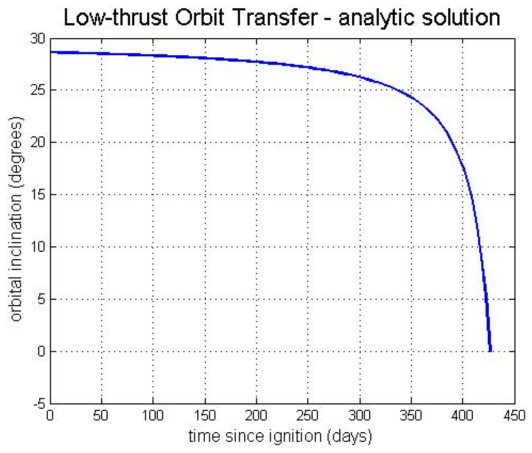
```
Low-thrust Orbit Transfer Analysis
initial orbit altitude 20212.0000 kilometers
```

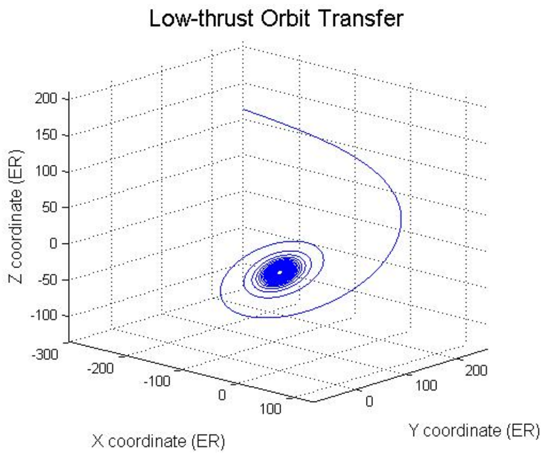
[2]MAVEN was due to provide 1.2kW at EOL.[8]

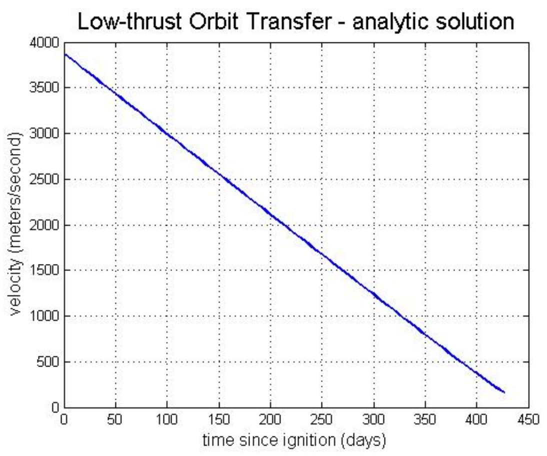
```
initial orbit inclination 28.6400 degrees
initial orbit velocity 3871.7616 meters/second
final orbit altitude 15000000.0000 kilometers
final orbit inclination 0.0000 degrees
final orbit velocity 162.9787 meters/second
total inclination change 28.6400 degrees
total delta-v 3758.2598 meters/second
thrust duration 426.4547 days
initial yaw angle 1.7568 degrees
thrust acceleration 0.102000 meters/second^2
```

By use of the ideal rocket equation, this equates to a mass change of:

$$\Delta m = 2500 \cdot \left(1 - e^{\frac{-3758.2598}{3600 \cdot 9.8066}}\right) = 252.461kg$$







Output plots for a low-thrust burn to escape

**Interplanetary Trajectory**

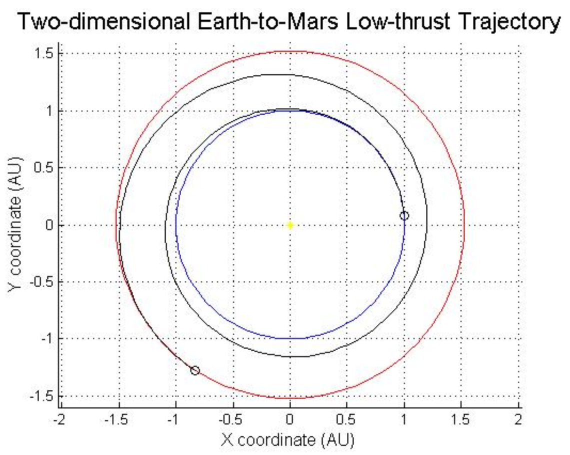
Hence, taking 2247.539 to be the new mass of the vehicle, the low-thrust interplanetary manoeuvre can be calculated using *ilt_ opt.m:*

```
throttle setting 0.734476

propellant mass 369.770623 kilograms

final mass 1877.768377 kilograms
```

Summing all mass requirements for electric propulsion yields a usage of 622.23kg. It is obvious that there are large mass savings to be had in interplanetary missions through the use of electric propulsion. By use of $/kg values as before, some $92 million can be saved in launch costs alone. It makes sense, therefore, that such a system would be favoured if the long transfer times are acceptable.



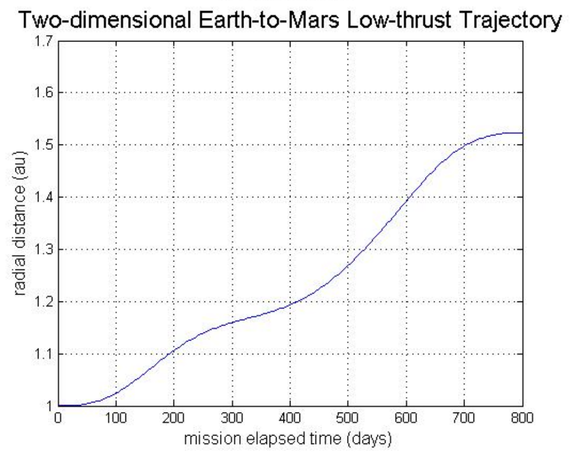


Figure 3.2: Output data of a low-thrust mission to Mars utilising electric propulsion

# Chapter 4

# Conclusions

It is evident, following the analysis set out in this piece, that electric propulsion is a valuable technology in alternative propulsion systems. However, increased subsystem requirements are present, in increased needs for power particularly, as well as the requirement for alternative orbital manoeuvres. Yet, the mass savings that electric propulsion can provide often compensates far beyond the added mass that arises from extra subsystems, and results often in decreased launch costs. It is of personal opinion that electric propulsion offers far higher versatility and ability than the more traditional chemical systems, despite evident extra time requirements. Nonetheless, this work touches little on the intricacies behind each method of propulsion, and features only simple analysis of orbital manoeuvres, areas of which further study is required. Research is ongoing into electric systems, however, and they are emerging as a promising technology for both deep space and more regular operations.

# Appendix A

# MATLAB code

The following MATLAB code was used in the analysis of orbital transfers for Chapter 3. The various codes that follow often call other scripts, including the SNOPT 7 package. The programs feature heavily code from the MATLAB file exchange.

## A.1 Solar-Electric Propulsion Transfer for LEO to GEO

Analysis of the use of solar-electric propulsion for a transfer from LEO to GEO, taking data on power input, specific impulse and inclination changes from user interactions.

```
% sep_ltot.m

% low-thrust orbit transfer between
% non-coplanar circular orbits using
% solar-electric propulsion (sep)

% Edelbaum equations

% Orbital Mechanics with Matlab

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% conversion factors

dtr = pi/180;                % degrees to radians

rtd = 180/pi;                % radians to degrees

% astrodynamic constants

req = 6378.14;               % Earth equatorial radius (kilometers)

mu = 398600.5;               % Earth gravitational constant (km^3/sec^2)

% request inputs

clc; home;

fprintf('\n   SEP Low-thrust Orbit Transfer Analysis\n');

while (1)
    fprintf('\n\nplease input the initial altitude (kilometers)\n');

    alt1 = input('? ');

    if (alt1 > 0)
```

```
        break;
    end
end

while (1)
    fprintf('\nplease input the final altitude (kilometers)\n');

    alt2 = input('? ');

    if (alt2 > 0)
        break;
    end
end

while (1)
    fprintf('\nplease input the initial orbital inclination (degrees)');
    fprintf('\n(0 <= inclination <= 180)\n');

    inc1 = input('? ');

    if (inc1 >= 0 && inc1 <= 180)
        break;
    end
end

while (1)
    fprintf('\nplease input the final orbital inclination (degrees)');
    fprintf('\n(0 <= inclination <= 180)\n');

    inc2 = input('? ');

    if (inc2 >= 0 && inc2 <= 180)
        break;
    end
end

while (1)
    fprintf('\nplease input the initial spacecraft mass (kilograms)\n');

    xmass0 = input('? ');

    if (xmass0 > 0)
        break;
    end
end

while (1)
    fprintf('\nplease input the SEP propulsive efficiency (non-dimensional)\n');

    teff = input('? ');

    if (teff > 0 && teff <= 1)
        break;
    end
end

while (1)
    fprintf('\nplease input the SEP input power (kilowatts)\n');

    tpower = input('? ');

    if (tpower > 0)
        break;
    end
end

while (1)
    fprintf('\nplease input the SEP specific impulse (seconds)\n');

    xisp = input('? ');
    if (xisp > 0)
        break;
    end
end

% compute thrust (newtons)

thrust = 1000.0d0 * (2.0d0 * teff * tpower / (9.80665d0 * xisp));

% compute the thrust acceleration to km/sec^2

thracc = (thrust / xmass0) / 1000;

% convert inclinations to radians

inc1 = inc1 * dtr;

inc2 = inc2 * dtr;

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% solve the orbit transfer problem %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% calculate total inclination change

dinct = abs(inc2 - inc1);

% check for coplanar orbits

if (dinct == 0)
    dinct = 1.0e-8;
end

% compute geocentric radii of initial and final orbits (kilometers)

r1 = req + alt1;

r2 = req + alt2;

% compute "local circular velocity" of initial and final orbits

v1 = sqrt(mu / r1);

v2 = sqrt(mu / r2);

% initial yaw angle

beta0 = atan(sin(0.5 * pi * dinct) / ((v1/v2) - cos(0.5 * pi * dinct)));

% delta-v

dvt = v1 * cos(beta0) - v1 * sin(beta0) / tan(0.5 * pi * dinct + beta0);

% thrust duration

tdur = dvt / thracc;

if (tdur < 3600)
    % minutes
    tdflag = 1;
    tdur = tdur / 60;
elseif (tdur < 86400)
    % hours
    tdflag = 2;
    tdur = tdur / 3600;
else
    % days
    tdflag = 3;
    tdur = tdur / 86400;
end
```

```
dtstep = tdur / 100;

tsim = - dtstep;

for i = 1:1:101
    tsim = tsim + dtstep;

    if (tdflag == 1)
        tsec = 60 * tsim;
    elseif (tdflag == 2)
        tsec = 3600 * tsim;
    else
        tsec = 86400 * tsim;
    end

    t(i) = tsim;

    beta(i) = rtd * atan(v1 * sin(beta0) / (v1 * cos(beta0) ...
        - thracc * tsec));

    tmp1 = atan((thracc * tsec - v1 * cos(beta0))/(v1 * sin(beta0)));

    dinc = rtd * (2/pi) * (tmp1 + 0.5 * pi - beta0);

    inc(i) = rtd * inc1 - dinc;

    v(i) = 1000 * sqrt(v1 * v1 - 2 * v1 * thracc * tsec * cos(beta0) ...
        + thracc * thracc * tsec * tsec);

    sma(i) = 1.0e6 * mu / (v(i) * v(i));
end

xmprop = xmass0 * (1.0d0 - exp(-1000.0d0 * dvt / (9.80665d0 * xisp)));

xmassf = xmass0 - xmprop;

% print results

clc; home;

fprintf('\n   SEP Low-thrust Orbit Transfer Analysis \n\n\n');

fprintf('initial orbit altitude      %10.4f kilometers \n\n', alt1);

fprintf('initial orbit inclination   %10.4f degrees \n\n', inc1 * rtd);

fprintf('initial orbit velocity      %10.4f meters/second \n\n\n', 1000 * v1);

fprintf('final orbit altitude        %10.4f kilometers \n\n', alt2);

fprintf('final orbit inclination     %10.4f degrees \n\n', inc2 * rtd);

fprintf('final orbit velocity        %10.4f meters/second \n\n', 1000 * v2);


fprintf('\npropulsive efficiency      %10.4f \n\n', teff);

fprintf('input power                 %10.4f kilowatts\n\n', tpower);

fprintf('specific impulse            %10.4f seconds\n\n', xisp);

fprintf('thrust                      %10.4f newtons\n\n', thrust);

fprintf('initial spacecraft mass     %10.4f kilograms\n\n', xmass0);

fprintf('final spacecraft mass       %10.4f kilograms\n\n', xmassf);

fprintf('propellant mass             %10.4f kilograms\n\n', xmprop);


fprintf('\ntotal inclination change    %10.4f degrees\n\n', rtd * dinct);
fprintf('total delta-v               %10.4f meters/second \n\n', 1000 * dvt);

if (tdflag == 1)
    fprintf('thrust duration             %10.4f minutes \n\n', tdur);
elseif (tdflag == 2)
    fprintf('thrust duration             %10.4f hours \n\n', tdur);
else
    fprintf('thrust duration             %10.4f days \n\n', tdur);
end

fprintf('initial yaw angle           %10.4f degrees \n\n', rtd * beta0);

fprintf('thrust acceleration         %10.6f meters/second^2 \n\n', 1000 * thracc);

keycheck;

% display graphics

subplot(2,1,1);

plot(t, beta);

title('SEP Low-thrust Orbit Transfer', 'FontSize', 16);

ylabel('Yaw Angle (deg)', 'FontSize', 12);

grid;

subplot(2,1,2);

plot(t, inc);

if (tdflag == 1)
    xlabel('Simulation Time (minutes)', 'FontSize', 12);
elseif (tdflag == 2)
    xlabel('Simulation Time (hours)', 'FontSize', 12);
else
    xlabel('Simulation Time (days)', 'FontSize', 12);
end

ylabel('Inclination (deg)', 'FontSize', 12);

grid;

print -depsc -tiff -r300 sep_ltot1.eps

keycheck;

subplot(2,1,1);

plot(t, v);

title('SEP Low-thrust Orbit Transfer', 'FontSize', 16);

ylabel('Velocity (m/s)', 'FontSize', 12);

grid;

subplot(2,1,2);

plot(t, sma);

if (tdflag == 1)
    xlabel('Simulation Time (minutes)', 'FontSize', 12);
elseif (tdflag == 2)
    xlabel('Simulation Time (hours)', 'FontSize', 12);
else
    xlabel('Simulation Time (days)', 'FontSize', 12);
end
```

```
ylabel('Semimajor Axis (kilometers)', 'FontSize', 12);

grid;

print —depsc —tiff —r300 sep_ltot2.eps
```

## A.2 Low-thrust Orbital Transfers for Burn-to-escape

Analysis of low-thrust orbital transfers between two non-coplanar circular orbits for electric propulsion was achieved through use of *ltot.m*. Data on altitude and specific impulse, as well as inclination changes, arises from user interaction.

```
% ltot.m           October 22, 2013

% low—thrust orbit transfer between non—coplanar circular orbits

% integrated solution with Edelbaum yaw steering

% Orbital Mechanics with MATLAB

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

clear all;

global req mu dtr rtd rkcoef beta0

global v1 tdflag thracc

% initialize rkf78 method

rkcoef = 1;

% read astrodynamic constants and conversion factors

om_constants;

% request inputs

clc; home;

fprintf('\nLow—thrust Orbit Transfer Between Non—coplanar Circular Orbits\n');

while (1)

    fprintf('\n\nplease input the initial altitude (kilometers)\n');

    alt1 = input('? ');

    if (alt1 > 0.0)
        break;
    end

end

while (1)

    fprintf('\nplease input the final altitude (kilometers)\n');

    alt2 = input('? ');

    if (alt2 > 0.0)
        break;
    end

end
```

```
while (1)

    fprintf('\nplease input the initial orbital inclination (degrees)');
    fprintf('\n(0 <= inclination <= 180)\n');

    inc1 = input('? ');

    if (inc1 >= 0.0 && inc1 <= 180.0)
        break;
    end

end

while (1)

    fprintf('\nplease input the final orbital inclination (degrees)');
    fprintf('\n(0 <= inclination <= 180)\n');

    inc2 = input('? ');

    if (inc2 >= 0.0 && inc2 <= 180.0)
        break;
    end

end

while (1)

    fprintf('\nplease input the initial RAAN (degrees)');
    fprintf('\n(0 <= RAAN <= 360)\n');

    raan0 = input('? ');

    if (raan0 >= 0.0 && raan0 <= 360.0)
        break;
    end

end

while (1)

    fprintf('\nplease input the thrust acceleration (meters/second^2)\n');

    tacc = input('? ');

    if (tacc > 0)
        break;
    end

end

% convert thrust acceleration to km/sec^2

thracc = tacc / 1000.0;

% convert inclinations to radians

inc1 = inc1 * dtr;

inc2 = inc2 * dtr;

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% solve the orbit transfer problem %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% calculate total inclination change

dinct = abs(inc2 - inc1);

% check for coplanar orbits
```

```
if (dinct == 0)

    dinct = 1.0e-8;

end

% compute geocentric radii of initial and final orbits (kilometers)

r1 = req + alt1;

r2 = req + alt2;

% compute "local circular velocity" of initial and final orbits

v1 = sqrt(mu / r1);

v2 = sqrt(mu / r2);

% initial yaw angle

beta0 = atan3(sin(0.5 * pi * dinct), (v1/v2) - cos(0.5 * pi * dinct));

% delta-v

dvt = v1 * cos(beta0) - v1 * sin(beta0) / tan(0.5 * pi * dinct + beta0);

% thrust duration

tdur = dvt / thracc;

thrdur_sec = tdur;

if (tdur < 3600.0)

    % minutes

    tdflag = 1;

    tdur = tdur / 60.0;

elseif (tdur < 86400.0)

    % hours

    tdflag = 2;

    tdur = tdur / 3600.0;

else

    % days

    tdflag = 3;

    tdur = tdur / 86400.0;

end

dtstep = tdur / 100.0;

tsim = -dtstep;

for i = 1:1:101

    tsim = tsim + dtstep;

    if (tdflag == 1)

        tsec = 60.0 * tsim;

    elseif (tdflag == 2)
```

```
        tsec = 3600.0 * tsim;

    else

        tsec = 86400.0 * tsim;

    end

    t(i) = tsim;

    beta(i) = rtd * atan3(v1 * sin(beta0), (v1 * cos(beta0) ...
        - thracc * tsec));

    tmp1 = atan((thracc * tsec - v1 * cos(beta0))/(v1 * sin(beta0)));

    dinc = rtd * (2/pi) * (tmp1 + 0.5 * pi - beta0);

    inc(i) = rtd * inc1 - dinc;

    v(i) = 1000.0 * sqrt(v1 * v1 - 2.0 * v1 * thracc * tsec * cos(beta0) ...
        + thracc * thracc * tsec * tsec);

    sma(i) = 1.0e6 * mu / (v(i) * v(i));

end

% print analytic results

clc; home;

fprintf('\n   Low-thrust Orbit Transfer Analysis \n\n');

fprintf('initial orbit altitude      %10.4f kilometers \n\n', alt1);

fprintf('initial orbit inclination   %10.4f degrees \n\n', inc1 * rtd);

fprintf('initial RAAN                %10.4f degrees \n\n', raan0);

fprintf('initial orbit velocity      %10.4f meters/second \n\n\n', 1000.0 * v1);

fprintf('final orbit altitude        %10.4f kilometers \n\n', alt2);

fprintf('final orbit inclination     %10.4f degrees \n\n', inc2 * rtd);

fprintf('final orbit velocity        %10.4f meters/second \n', 1000.0 * v2);

fprintf('\ntotal inclination change    %10.4f degrees\n\n', rtd * dinct);

fprintf('total delta-v               %10.4f meters/second \n\n', 1000.0 * dvt);

if (tdflag == 1)

    fprintf('thrust duration             %10.4f minutes \n\n', tdur);

elseif (tdflag == 2)

    fprintf('thrust duration             %10.4f hours \n\n', tdur);

else

    fprintf('thrust duration             %10.4f days \n\n', tdur);

end

fprintf('initial yaw angle           %10.4f degrees \n\n', rtd * beta0);

fprintf('thrust acceleration         %10.6f meters/second^2 \n\n', 1000.0 * tacc);

%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% create integrated solution
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% initial semimajor axis (kilometers)

oev_initial(1) = r1;

% initial orbital eccentricity (nd)

oev_initial(2) = 0.0;

% initial orbital inclination (radians)

oev_initial(3) = inc1;

% initial argument of periapsis (radians)

oev_initial(4) = 0.0 * dtr;

% initial RAAN (radians)

oev_initial(5) = raan0 * dtr;

% initial true anomaly (radians)

oev_initial(6) = 0.0;

% convert user-defined classical orbital elements to eci state vector

[reci_initial, veci_initial] = orb2eci(mu, oev_initial);

% convert eci state vector to modified equinoctial orbital elements

mee_initial = eci2mee(mu, reci_initial, veci_initial);

% data generation step size (seconds)

deltat = 1800.0d0;

% final simulation time (seconds)

tfinal = thrdur_sec;

% number of differential equations

neq = 6;

% truncation error tolerance

tetol = 1.0d-12;

tf = 0.0d0;

nplot = 1;

[reci, veci] = mee2eci(mu, mee_initial(1:6));

oev_wrk = eci2orb1 (mu, reci, veci);

%%%%%%%%%%%%%%%%%%%%%%%
% initial graphics data
%%%%%%%%%%%%%%%%%%%%%%%

% simulation time (seconds)

xplot(nplot) = 0.0;

% x position coordinate (kilometers)

yplot1(nplot) = reci(1) / req;

% y position coordinate (kilometers)
```

```
yplot2(nplot) = reci(2) / req;

% z position component (kilometers)

yplot3(nplot) = reci(3) / req;

% yaw angle (degrees)

yplot4(nplot) = rtd * beta0;

% semimajor axis (kilometers)

yplot5(nplot) = oev_wrk(1);

% orbital eccentricity (nd)

yplot6(nplot) = oev_wrk(2);

% orbital inclination (degrees)

yplot7(nplot) = rtd * oev_wrk(3);

% RAAN (degrees)

yplot8(nplot) = rtd * oev_wrk(5);

% integrate equations of motion and create data arrays

while (1)

    h = 10.0d0;

    ti = tf;

    tf = ti + deltat;

    if (tf > tfinal)

        tf = tfinal;

    end

    mee_final = rkf78 ('meeeqm', neq, ti, tf, h, tetol, mee_initial);

    [reci, veci] = mee2eci(mu, mee_final(1:6));

    oev_wrk = eci2orb1 (mu, reci, veci);

    pmee = mee_final(1);
    fmee = mee_final(2);
    gmee = mee_final(3);
    hmee = mee_final(4);
    xkmee = mee_final(5);
    xlmee = mee_final(6);

    % current yaw angle (radians)

    beta_tmp = atan3(v1 * sin(beta0), (v1 * cos(beta0) - thracc * tf));

    % current argument of latitude (radians)

    arglat = mod(oev_wrk(4) + oev_wrk(6), 2.0 * pi);

    if (arglat >= 0.0 && arglat < 0.5 * pi)

        beta_wrk = -beta_tmp;

    end

    if (arglat > 0.5 * pi && arglat < pi)
```

```
        beta_wrk = beta_tmp;

    end

    if (arglat > pi && arglat < 1.5 * pi)

        beta_wrk = beta_tmp;

    end

    if (arglat > 1.5 * pi && arglat < 2.0 * pi)

        beta_wrk = -beta_tmp;

    end

    % load data arrays for plotting

    nplot = nplot + 1;

    % simulation time (seconds)

    xplot(nplot) = ti;

    % x position coordinate (Earth radii)

    yplot1(nplot) = reci(1) / req;

    % y position coordinate (Earth radii)

    yplot2(nplot) = reci(2) / req;

    % z position component (Earth radii)

    yplot3(nplot) = reci(3) / req;

    % yaw angle (degrees)

    yplot4(nplot) = rtd * beta_wrk;

    % semimajor axis (kilometers)

    yplot5(nplot) = oev_wrk(1);

    % orbital eccentricity (nd)

    yplot6(nplot) = oev_wrk(2);

    % orbital inclination (degrees)

    yplot7(nplot) = rtd * oev_wrk(3);

    % RAAN (degrees)

    yplot8(nplot) = rtd * oev_wrk(5);

    if (tf == tfinal)

        break

    end

    % reload integration vector

    mee_initial = mee_final;

end

% print results
```

```
fprintf('\nfinal classical orbital elements and state vector — integrated solution');
fprintf('\n————————————————————————————————————————————————————————————————————————\n');

oeprint1(mu, oev_wrk, 2);

[reci_final, veci_final] = orb2eci(mu, oev_wrk);

svprint(reci_final, veci_final);

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% display analytic solution graphics
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

figure(1);

plot(t, beta, 'LineWidth', 1.5);

title('Low—thrust Orbit Transfer — analytic solution', 'FontSize', 16);

if (tdflag == 1)

    xlabel('time since ignition (minutes)', 'FontSize', 12);

elseif (tdflag == 2)

    xlabel('time since ignition (hours)', 'FontSize', 12);

else

    xlabel('time since ignition (days)', 'FontSize', 12);

end

ylabel('|yaw angle| (degrees)', 'FontSize', 12);

grid;

print —depsc —tiff —r300 ltot1.eps

figure(2);

plot(t, inc, 'LineWidth', 1.5);

title('Low—thrust Orbit Transfer — analytic solution', 'FontSize', 16);

if (tdflag == 1)

    xlabel('time since ignition (minutes)', 'FontSize', 12);

elseif (tdflag == 2)

    xlabel('time since ignition (hours)', 'FontSize', 12);

else

    xlabel('time since ignition (days)', 'FontSize', 12);

end

ylabel('orbital inclination (degrees)', 'FontSize', 12);

grid;

print —depsc —tiff —r300 ltot2.eps

figure(3);

plot(t, v, 'LineWidth', 1.5);

title('Low—thrust Orbit Transfer — analytic solution', 'FontSize', 16);
```

```
if (tdflag == 1)

    xlabel('time since ignition (minutes)', 'FontSize', 12);

elseif (tdflag == 2)

    xlabel('time since ignition (hours)', 'FontSize', 12);

else

    xlabel('time since ignition (days)', 'FontSize', 12);

end

ylabel('velocity (meters/second)', 'FontSize', 12);

grid;

print -depsc -tiff -r300 ltot3.eps

figure(4);

plot(t, sma, 'LineWidth', 1.5);

title('Low-thrust Orbit Transfer - analytic solution', 'FontSize', 16);

if (tdflag == 1)

    xlabel('time since ignition (minutes)', 'FontSize', 12);

elseif (tdflag == 2)

    xlabel('time since ignition (hours)', 'FontSize', 12);

else

    xlabel('time since ignition (days)', 'FontSize', 12);

end

ylabel('semimajor axis (kilometers)', 'FontSize', 12);

grid;

print -depsc -tiff -r300 ltot4.eps

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% display integrated solution graphics
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

figure(5);

plot(xplot / 86400.0, yplot4, 'LineWidth', 1.5);

title('Low-thrust Orbit Transfer - integrated solution', 'FontSize', 16);

xlabel('time since ignition (days)', 'FontSize', 12);

ylabel('yaw angle (degrees)', 'FontSize', 12);

grid;

print -depsc -tiff -r300 ltot5.eps;

figure(6);

plot(xplot / 86400.0, yplot5, 'LineWidth', 1.5);

title('Low-thrust Orbit Transfer - integrated solution', 'FontSize', 16);

xlabel('time since ignition (days)', 'FontSize', 12);
```

```
ylabel('semimajor axis (kilometers)', 'FontSize', 12);

grid;

print —depsc —tiff —r300 ltot6.eps;

figure(7);

plot(xplot / 86400.0, yplot6, 'LineWidth', 1.5);

title('Low—thrust Orbit Transfer — integrated solution', 'FontSize', 16);

xlabel('time since ignition (days)', 'FontSize', 12);

ylabel('orbital eccentricity', 'FontSize', 12);

grid;

print —depsc —tiff —r300 ltot7.eps;

figure(8);

plot(xplot / 86400.0, yplot7, 'LineWidth', 1.5);

title('Low—thrust Orbit Transfer — integrated solution', 'FontSize', 16);

xlabel('time since ignition (days)', 'FontSize', 12);

ylabel('orbital inclination (degrees)', 'FontSize', 12);

grid;

print —depsc —tiff —r300 ltot8.eps;

figure(9);

plot(xplot / 86400.0, yplot8, 'LineWidth', 1.5);

title('Low—thrust Orbit Transfer — integrated solution', 'FontSize', 16);

xlabel('time since ignition (days)', 'FontSize', 12);

ylabel('RAAN (degrees)', 'FontSize', 12);

grid;

print —depsc —tiff —r300 ltot9.eps;

figure(10);

% create axes vectors

xaxisx = [1 1.5];
xaxisy = [0 0];
xaxisz = [0 0];

yaxisx = [0 0];
yaxisy = [1 1.5];
yaxisz = [0 0];

zaxisx = [0 0];
zaxisy = [0 0];
zaxisz = [1 1.5];

hold on;

grid on;

% plot earth
```

```
[x y z] = sphere(24);

h = surf(x, y, z);

colormap([127/255 1 222/255]);

set (h, 'edgecolor', [1 1 1]);

% plot coordinate system axes

plot3(xaxisx, xaxisy, xaxisz, '-g', 'LineWidth', 1);

plot3(yaxisx, yaxisy, yaxisz, '-r', 'LineWidth', 1);

plot3(zaxisx, zaxisy, zaxisz, '-b', 'LineWidth', 1);

% plot transfer orbit

plot3(yplot1, yplot2, yplot3, '-b', 'LineWidth', 1);

xlabel('X coordinate (ER)', 'FontSize', 12);

ylabel('Y coordinate (ER)', 'FontSize', 12);

zlabel('Z coordinate (ER)', 'FontSize', 12);

title('Low-thrust Orbit Transfer', 'FontSize', 16);

axis equal;

view(50, 20);

rotate3d on;

print -depsc -tiff -r300 ltot10.eps;
```

## A.3 Maximum-mass Transfer to Mars for Chemical Systems

Used for calculating the optimal trajectory for a chemical transfer to Mars, this code takes input from a *.dat* file and is designed to maximise mass given an initial launch date guess and propulsion characteristics data.

```
% e2m_matlab.m        December 24, 2012

% Earth-to-Mars trajectory optimization

% JPL DE421 ephemeris and SNOPT NLP algorithm

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

clear all;

global aunit iephem km ephname dtr rtd

global emu smu mmu pmu xmu itar_type rkcoef

global j2 req req_mars rsoi

global jdtdb0 jdtdb_tip jdtdb_soi jdtdb_ca

global otype rito vito rm2sc vm2sc rfto vfto

global ip1 ip2 dv1 dv2 rpt xinct fpa_tar rmag_tar

global xinc_po rpmag_po rhyper vhyper
global vinf rla dla rsc_soi vsc_soi rsc_ca vsc_ca

global bdott_user bdotr_user rgeo_soi vgeo_soi

% read leap seconds data file

readleap;

% angular conversion factors

rtd = 180.0d0 / pi;

dtr = pi / 180.0d0;

atr = dtr / 3600.0d0;

% de421 value for astronomical unit (kilometers)

aunit = 149597870.699626200;

% gravitational constant of the Earth (km^3/sec^2)

emu = 398600.4415;

% equatorial radius of the Earth (kilometers)

req = 6378.1378;

% j2 gravity coefficient of the Earth

j2 = 0.00108263;

% sphere—of—influence of the Earth (kilometers)

rsoi = 925000.0d0;

% gravitational constant of the moon (km^3/sec^2)

mmu = 4902.800238d0;

% gravitational constant of mars (km^3/sec^2)

pmu = 42828.376212;

% radius of mars (kilometers)

req_mars = 3396.2;

% initialize de421 ephemeris

ephname = 'de421.bin';

iephem = 1;

km = 0;

% initialize RKF7(8) integration method

rkcoef = 1;

% define "reference" julian date (1/1/2000)

jdtdb0 = 2451544.5;

%**********************************************
% DE421 gravitational constants (au**3/day**2)
%**********************************************

% earth—moon mass ratio

emrat = 0.813005674615454410D+02;
```

```
% sun

xmu(1) = 0.295912208285591149D-03;

smu = xmu(1) * aunit^3 / 86400.0d0^2;

% mercury

xmu(2) = 0.491254745145081187d-10;

% venus

xmu(3) = 0.724345233302178764D-09;

% earth

xmu(4) = 0.899701152970881167D-09;

xmu(4) = emu * 86400.0d0^2 / aunit^3;

% mars

xmu(5) = 0.954954891857520080D-10;

% jupiter

xmu(6) = 0.282534585557186464D-06;

% saturn

xmu(7) = 0.845972727826697794D-07;

% uranus

xmu(8) = 0.129202491678196939D-07;

% neptune

xmu(9) = 0.152435890078427628D-07;

% read simulation definition data file

[filename, pathname] = uigetfile('*.in', 'Please select the input file to read');

[fid, otype, jdtdb_tip, ddays1, jdtdb_arrival, ddays2, alt_po, aziml, ...
    xlatgs, rpt, xinct, fpa_tar, alt_tar, itar_type, ...
    bdott_user, bdotr_user] = e2m_readdata(filename);

% set target radius at Mars (kilometers)

rmag_tar = req_mars + alt_tar;

% begin simulation

clc; home;

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% step 1 - solve two-body lambert problem
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% departure and arrival planets

ip1 = 3;

ip2 = 4;

% posigrade transfer

direct = 1;
```

```
% less than one rev transfer

revmax = 0;

mu = smu;

if (otype < 4)

   %%%%%%%%%%%%%%%%%%%%%%%%%
   % find optimal solution %
   %%%%%%%%%%%%%%%%%%%%%%%%%

   xg(1) = jdtdb_tip - jdtdb0;

   xg(2) = jdtdb_arrival - jdtdb0;

   xg = xg';

   % bounds on control variables

   xlwr(1) = xg(1) - ddays1;

   xupr(1) = xg(1) + ddays1;

   xlwr(2) = xg(2) - ddays2;

   xupr(2) = xg(2) + ddays2;

   xlwr = xlwr';

   xupr = xupr';

   % bounds on objective function

   flow(1) = 0.0d0;

   fupp(1) = +Inf;

   % find optimum

   snscreen on;

   [x, f, inform, xmul, fmul] = snopt(xg, xlwr, xupr, flow, fupp, 'e2m_deltav');

   jdtdb_tip = x(1) + jdtdb0;

   jdtdb_arrival = x(2) + jdtdb0;

   % transfer time (days)

   taud = jdtdb_arrival - jdtdb_tip;

   % evaluate current solution

   [f, g] = e2m_deltav(x);

   dvm1 = norm(dv1);

else

   %%%%%%%%%%%%%%%%%
   % no optimization
   %%%%%%%%%%%%%%%%%

   x(1) = jdtdb_tip - jdtdb0;

   x(2) = jdtdb_arrival - jdtdb0;

   [f, g] = e2m_deltav(x);

   taud = jdtdb_arrival - jdtdb_tip;
   dvm1 = norm(dv1);

end

%%%%%%%%%%%%%%%%%
% print results %
%%%%%%%%%%%%%%%%%

% convert solution julian dates to calendar dates and utc

jdutc1 = tdb2utc (jdtdb_tip);

[cdstr1, utstr1] = jd2str(jdutc1);

fprintf('\nEarth-to-Mars mission design\n');

fprintf('\n=========================');
fprintf('\ntwo-body Lambert solution');
fprintf('\n=========================');

switch otype

case 1

  fprintf('\n\nminimize departure delta-v\n');

case 2

  fprintf('\n\nminimize arrival delta-v\n');

case 3

  fprintf('\n\nminimize total delta-v\n');

case 4

  fprintf('\n\nno optimization\n');

end

fprintf('\ndeparture heliocentric delta-v vector and magnitude');
fprintf('\n(Earth mean equator and equinox of J2000)');
fprintf('\n-------------------------------------------\n');

fprintf('\nx-component of delta-v      %12.6f  meters/second', 1000.0 * dv1(1));

fprintf('\ny-component of delta-v      %12.6f  meters/second', 1000.0 * dv1(2));

fprintf('\nz-component of delta-v      %12.6f  meters/second', 1000.0 * dv1(3));

fprintf('\n\ndelta-v magnitude           %12.6f  meters/second\n\n', 1000.0 * norm(dv1));

fprintf('\narrival heliocentric delta-v vector and magnitude');
fprintf('\n(Earth mean equator and equinox of J2000)');
fprintf('\n-------------------------------------------\n');

fprintf('\nx-component of delta-v      %12.6f  meters/second', 1000.0 * dv2(1));

fprintf('\ny-component of delta-v      %12.6f  meters/second', 1000.0 * dv2(2));

fprintf('\nz-component of delta-v      %12.6f  meters/second', 1000.0 * dv2(3));

fprintf('\n\ndelta-v magnitude           %12.6f  meters/second\n', 1000.0 * norm(dv2));

% print orbital elements of the earth at departure

fprintf('\n\nheliocentric coordinates of the Earth at departure');
fprintf('\n(Earth mean equator and equinox of J2000)');
fprintf('\n-------------------------------------------\n');
fprintf('\nUTC calendar date    ');

disp(cdstr1);

fprintf('\nUTC time             ');

disp(utstr1);

fprintf('\nUTC Julian Date      %12.8f\n', jdutc1);

svearth = jplephem(jdtdb_tip, ip1, 11);

r = svearth(1:3);

v = svearth(4:6);

oev = eci2orb1(smu, aunit * r, aunit * v / 86400.0);

oeprint2(smu, oev);

svprint(aunit * r, aunit * v / 86400.0);

% print orbital elements of the heliocentric transfer orbit

fprintf('\nspacecraft heliocentric coordinates after the first impulse');
fprintf('\n(Earth mean equator and equinox of J2000)');
fprintf('\n-----------------------------------------\n');

fprintf('\nUTC calendar date    ');

disp(cdstr1);

fprintf('\nUTC time             ');

disp(utstr1);

fprintf('\nUTC Julian Date      %12.8f\n', jdutc1);

oev = eci2orb1(smu, rito, vito');

oeprint2(smu, oev);

svprint(rito, vito');

fprintf('\nspacecraft heliocentric coordinates prior to the second impulse');
fprintf('\n(Earth mean equator and equinox of J2000)');
fprintf('\n-----------------------------------------\n');

jdutc2 = tdb2utc (jdtdb_arrival);

[cdstr2, utstr2] = jd2str(jdutc2);

fprintf('\nUTC calendar date    ');

disp(cdstr2);

fprintf('\nUTC time             ');

disp(utstr2);

fprintf('\nUTC Julian Date      %12.8f\n', jdutc2');

% propagate transfer orbit

[rfto, vfto] = twobody2 (smu, 86400.0 * taud, rito, vito);

oev = eci2orb1(smu, rfto, vfto);

oeprint2(smu, oev);

svprint(rfto, vfto);
```

```
fprintf('\nspacecraft heliocentric coordinates after the second impulse');
fprintf('\n(Earth mean equator and equinox of J2000)');
fprintf('\n------------------------------------------\n');

fprintf('\nUTC calendar date    ');

disp(cdstr2);

fprintf('\nUTC time             ');

disp(utstr2);

fprintf('\nUTC Julian Date      %12.8f\n', jdutc2');

rscf = rfto;

vscf = vfto + dv2;

oev = eci2orb1(smu, rfto, vfto);

oeprint2(smu, oev);

svprint(rscf, vscf);

% print orbital elements of Mars at arrival

fprintf('\nheliocentric coordinates of Mars at arrival');
fprintf('\n(Earth mean equator and equinox of J2000)');
fprintf('\n------------------------------------------\n');

fprintf('\nUTC calendar date    ');

disp(cdstr2);

fprintf('\nUTC time             ');

disp(utstr2);

fprintf('\nUTC Julian Date      %12.8f\n', jdutc2');

svmars = jplephem(jdtdb_arrival, ip2, 11);

r = svmars(1:3);

v = svmars(4:6);

oev = eci2orb1(smu, aunit * r, aunit * v / 86400.0);

oeprint2(smu, oev);

svprint(aunit * r, aunit * v / 86400.0);

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% compute orientation of the departure hyperbola
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

decl1 = 90.0d0 - rtd * acos(dv1(3) / dvm1);

rasc1 = rtd * atan3(dv1(2), dv1(1));

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% step 2 - compute characteristics of launch hyperbola
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

rpmag_po = req + alt_po;

xinc_po = acos(cos(xlatgs) * sin(aziml));

% check for invalid orbital inclination
```

```
if (abs(xinc_po) < abs(dtr * decl1))

   fprintf('\npark orbit error!!\n');

   fprintf('\n|inclination| must be > |asymptote declination|');

   fprintf('\n\npark orbit inclination   %12.6f  degrees', rtd * xinc_po);

   fprintf('\n\nasymptote declination    %12.6f  degrees\n\n', decl1);

   break;

end

[rhyper, vhyper] = launch(emu, dvm1, dtr * decl1, dtr * rasc1, rpmag_po, xinc_po);

oev1 = eci2orb1 (emu, rhyper, vhyper);

% compute orbital elements of circular park orbit

for i = 1:1:6

    oev2(i) = oev1(i);

end

oev2(1) = rpmag_po;

oev2(2) = 0.0;

oev2(4) = 0.0;

oev2(6) = oev1(4);

[rpo, vpo] = orb2eci (emu, oev2);

% compute injection delta-v

for i = 1:1:3

    deltav(i) = vhyper(i) - vpo(i);

end

decl1 = 90.0d0 - rtd * acos(dv1(3) / norm(dv1));

rasc1 = rtd * atan3(dv1(2), dv1(1));

fprintf('\npark orbit and departure hyperbola characteristics');
fprintf('\n(Earth mean equator and equinox of J2000)');
fprintf('\n---------------------------------------------------\n');

fprintf('\npark orbit');
fprintf('\n------------\n');

oeprint1(emu, oev2);

svprint (rpo, vpo);

fprintf('\ndeparture hyperbola');
fprintf('\n--------------------\n');

fprintf('\nc3                          %12.6f  km^2/sec^2\n', dvm1 * dvm1);

fprintf('\nv-infinity                  %12.6f  meters/second\n', 1000.0 * dvm1);

fprintf('\nasymptote right ascension   %12.6f  degrees\n', rasc1);

fprintf('\nasymptote declination       %12.6f  degrees\n', decl1);

fprintf('\nperigee altitude            %12.6f  kilometers\n', alt_po);
fprintf('\nlaunch azimuth              %12.6f  degrees\n', rtd * aziml);

fprintf('\nlaunch site latitude        %12.6f  degrees\n\n', rtd * xlatgs);

fprintf('\nUTC calendar date    ');

disp(cdstr1);

fprintf('\nUTC time             ');

disp(utstr1);

fprintf('\nUTC Julian Date      %12.8f\n', jdutc1);

oeprint1(emu, oev1);

svprint (rhyper, vhyper);

fprintf('\nhyperbolic injection delta-v vector and magnitude');
fprintf('\n(Earth mean equator and equinox of J2000)');
fprintf('\n-------------------------------------------------\n');

fprintf('\nx-component of delta-v      %12.6f  meters/sec', 1000.0 * deltav(1));

fprintf('\ny-component of delta-v      %12.6f  meters/sec', 1000.0 * deltav(2));

fprintf('\nz-component of delta-v      %12.6f  meters/sec', 1000.0 * deltav(3));

fprintf('\n\ndelta-v magnitude           %12.6f  meters/sec', 1000.0 * norm(deltav));

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% solve b-plane targeting problem using a simple shooting method
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

fprintf('\n\nplease wait, solving b-plane targeting problem ...\n');

% initial guess for launch vinf, rla and dla

xg(1) = dvm1;

xg(2) = dtr * rasc1;

xg(3) = dtr * decl1;

if (otype == 4)

   % transpose

   xg = xg';

end

% define lower and upper bounds for vinf, rla and dla

xlwr(1) = xg(1) - 0.05;
xupr(1) = xg(1) + 0.05;

xlwr(2) = xg(2) - 10.0 * dtr;
xupr(2) = xg(2) + 10.0 * dtr;

xlwr(3) = xg(3) - 1.0 * dtr;
xupr(3) = xg(3) + 1.0 * dtr;

if (otype == 4)

   % transpose bounds

   xlwr = xlwr';

   xupr = xupr';
```

```
end

% bounds on objective function

flow(1) = 0.0d0;

fupp(1) = +Inf;

% bounds on final b-plane/orbital element equality constraints

flow(2) = 0.0d0;
fupp(2) = 0.0d0;

flow(3) = 0.0d0;
fupp(3) = 0.0d0;

flow = flow';

fupp = fupp';

snscreen on;

[x, f, inform, xmul, fmul] = snopt(xg, xlwr, xupr, flow, fupp, 'e2m_shoot');

snprintfile('off');

snsummary('off');

% evaluate final solution

[f, g] = e2m_shoot(x);

% compute orbital elements of launch hyperbola

oev1 = eci2orb1 (emu, rhyper, vhyper);

% compute orbital elements of circular park orbit

for i = 1:1:6

    oev2(i) = oev1(i);

end

oev2(1) = rpmag_po;

oev2(2) = 0.0;

oev2(4) = 0.0;

oev2(6) = oev1(4);

[rpo, vpo] = orb2eci (emu, oev2);

% compute injection delta-v (km/sec)

for i = 1:1:3

    deltav(i) = vhyper(i) - vpo(i);

end

decl1 = 90.0d0 - rtd * acos(deltav(3) / norm(deltav));

rasc1 = rtd * atan3(deltav(2), deltav(1));

fprintf('\n========================');
fprintf('\noptimal n-body solution');
fprintf('\n========================\n');
```

```
if (itar_type == 1)

   fprintf('\nB-plane targeting\n');

end

if (itar_type == 2)

   fprintf('\norbital element targeting\n');

end

if (itar_type == 3)

   fprintf('\nEI conditions targeting\n');

end

fprintf('\npark orbit and departure hyperbola characteristics');
fprintf('\n(Earth mean equator and equinox of J2000)');
fprintf('\n------------------------------------------\n');

fprintf('\npark orbit');
fprintf('\n----------\n');

oeprint1(emu, oev2);

svprint (rpo, vpo);

fprintf('\ndeparture hyperbola');
fprintf('\n-------------------\n');

vinf = f(1);

fprintf('\nc3                          %12.6f  km^2/sec^2\n', vinf * vinf);

fprintf('\nv-infinity                  %12.6f  meters/second\n', 1000.0 * vinf);

fprintf('\nasymptote right ascension   %12.6f  degrees\n', rtd * rla);

fprintf('\nasymptote declination       %12.6f  degrees\n\n', rtd * dla);

fprintf('\nUTC calendar date    ');

disp(cdstr1);

fprintf('\nUTC time             ');

disp(utstr1);

fprintf('\nUTC Julian Date      %12.8f\n', jdtdb_tip);

oeprint1(emu, oev1);

svprint (rhyper, vhyper);

fprintf('\nhyperbolic injection delta-v vector and magnitude');
fprintf('\n(Earth mean equator and equinox of J2000)');
fprintf('\n------------------------------------------\n');

fprintf('\nx-component of delta-v      %12.6f  meters/sec', 1000.0 * deltav(1));

fprintf('\ny-component of delta-v      %12.6f  meters/sec', 1000.0 * deltav(2));

fprintf('\nz-component of delta-v      %12.6f  meters/sec', 1000.0 * deltav(3));

fprintf('\n\ndelta-v magnitude           %12.6f  meters/sec\n', 1000.0 * norm(deltav));

fprintf('\ntransfer time               %12.6f  days\n', jdtdb_ca - jdtdb_tip);

fprintf('\ntime and conditions at Mars closest approach');
fprintf('\n(Mars mean equator and IAU node of epoch)');
fprintf('\n-------------------------------------------\n');

jdutc_ca = tdb2utc(jdtdb_ca);

[cdstr_ca, utstr_ca] = jd2str(jdutc_ca);

fprintf('\nUTC calendar date    ');

disp(cdstr_ca);

fprintf('\nUTC time             ');

disp(utstr_ca);

fprintf('\nUTC Julian Date      %12.8f\n', jdutc_ca);

% orbital elements at closest approach

oev = eci2orb1(pmu, rm2sc, vm2sc);

oeprint1(pmu, oev);

svprint(rm2sc, vm2sc);

% compute b-plane coordinates

[bplane, rv, tv, ibperr] = rv2bp2(pmu, rm2sc, vm2sc);

% print results

fprintf('\nB-plane coordinates at Mars closest approach');
fprintf('\n(Mars mean equator and IAU node of epoch)');
fprintf('\n-------------------------------------------\n');

fprintf ('\nb-magnitude                %12.6f  kilometers', sqrt(bplane(7)^2 + bplane(8)^2));

fprintf ('\nb dot r                    %12.6f  kilometers', bplane(8));

fprintf ('\nb dot t                    %12.6f  kilometers', bplane(7));

fprintf ('\nb-plane angle              %12.6f  degrees', rtd * bplane(6));

fprintf ('\nv-infinity                 %12.6f  meters/second', 1000.0d0 * bplane(1));

fprintf ('\nr-periapsis                %12.6f  kilometers', bplane(5));

fprintf ('\ndecl-asymptote             %12.6f  degrees', rtd * bplane(2));

fprintf ('\nrasc-asymptote             %12.6f  degrees\n', rtd * bplane(3));

% sine of flight path angle at closest approach

sfpa = rm2sc' * vm2sc /(norm(rm2sc) * norm(vm2sc));

fprintf ('\nflight path angle          %12.6f  degrees\n', rtd * asin(sfpa));

fprintf('\nspacecraft heliocentric coordinates at closest approach');
fprintf('\n(Earth mean equator and equinox of J2000)');
fprintf('\n-------------------------------------------\n');

fprintf('\nUTC calendar date    ');

disp(cdstr_ca);

fprintf('\nUTC time             ');

disp(utstr_ca);

fprintf('\nUTC Julian Date      %12.8f\n', jdutc_ca);
```

```
oev_ca = eci2orb1 (smu, aunit * rsc_ca, aunit * vsc_ca / 86400.0);

oeprint1(smu, oev_ca);

svprint(aunit * rsc_ca, aunit * vsc_ca / 86400.0);

fprintf('\nheliocentric coordinates of Mars at closest approach');
fprintf('\n(Earth mean equator and equinox of J2000)');
fprintf('\n------------------------------------------\n');

fprintf('\nUTC calendar date    ');

disp(cdstr_ca);

fprintf('\nUTC time             ');

disp(utstr_ca);

fprintf('\nUTC Julian Date      %12.8f\n', jdutc_ca);

svmars = jplephem(jdtdb_ca, ip2, 11);

r = svmars(1:3);

v = svmars(4:6);

oev = eci2orb1(smu, aunit * r, aunit * v / 86400.0);

oeprint2(smu, oev);

svprint(aunit * r, aunit * v / 86400.0);

fprintf('\nspacecraft geocentric coordinates at the Earth SOI');
fprintf('\n(Earth mean equator and equinox of J2000)');
fprintf('\n------------------------------------------\n');

jdutc_soi = tdb2utc(jdtdb_soi);

[cdstr_soi, utstr_soi] = jd2str(jdutc_soi);

fprintf('\nUTC calendar date    ');

disp(cdstr_soi);

fprintf('\nUTC time             ');

disp(utstr_soi);

fprintf('\nUTC Julian Date      %12.8f\n', jdutc_soi);

oev_soi = eci2orb1 (emu, rgeo_soi, vgeo_soi);

oeprint1(emu, oev_soi);

svprint(rgeo_soi, vgeo_soi);

fprintf('\nspacecraft heliocentric coordinates at the Earth SOI');
fprintf('\n(Earth mean equator and equinox of J2000)');
fprintf('\n------------------------------------------\n');

fprintf('\nUTC calendar date    ');

disp(cdstr_soi);

fprintf('\nUTC time             ');

disp(utstr_soi);

fprintf('\nUTC Julian Date      %12.8f\n', jdutc_soi);

oev_soi = eci2orb1 (smu, aunit * rsc_soi, aunit * vsc_soi / 86400.0);
```

```
oeprint1(smu, oev_soi);

svprint(aunit * rsc_soi, aunit * vsc_soi / 86400.0);

while(1)

   fprintf('\n\nwould you like to create trajectory graphics (y = yes, n = no)\n');

   slct = input('? ', 's');

   if (slct == 'y' || slct == 'n')

      break;

   end

end

if (slct == 'y')

   % create trajectory graphics

   marsplot;

end
```

## A.4 Optimal Low-thrust Martian transfer for Electric Propulsion

Used for calculating the optimal trajectory for an electrical transfer to Mars, this code takes input from user interaction, given an initial launch date guess and propulsion characteristics data.

```
% ilt_opt.m           May 6, 2014

% two-dimensional low-thrust Earth-to-Mars trajectory optimization

% minimize transfer time or maximize final mass
% with piecewise-linear steering

% time and state variables

%  t    = simulation time
%  y(1) = radial distance (r)
%  y(2) = radial component of velocity (u)
%  y(3) = tangential component of velocity (v)
%  y(4) = polar angle (theta)

% Orbital Mechanics with MATLAB

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

clear all;

global xmu aunit rkcoef tfactor nsegments thrmag mass0 iopt tof

global acc_thrust mdot mp_dot alpha_wrk xinitial xfinal

% conversion factors - radians to/from degrees

rtd = 180.0 / pi;

dtr = pi / 180.0;

% astronomical unit (kilometers)

aunit = 149597870.691;
```

```
% gravitational constant of the sun (au^3/day^2)

smu = 0.2959122082855912D-03;

% gravitational constant of the sun (kilometer^3/second^2)

xmu = smu * aunit^3 / 86400.0^2;

% non-dimensional to dimensional time conversion factor

tfactor = sqrt(1.0 / smu);

% initialize rkf78 algorithm

rkcoef = 1;

clc; home;

fprintf('\n\nprogram ilt_opt - Earth-to-Mars low-thrust trajectory\n');

% request name of simulation definition data file

[filename, pathname] = uigetfile('*.dat', 'Please select the input file to read');

% read data file

[fid, iopt, nsegments, mass0, thrmag, xisp, time_g, time_lb, ...
    time_ub, throttle_g, throttle_lb, throttle_ub, alpha_g, ...
    alpha_lb, alpha_ub, jd_guess] = read_ilt(filename);

% compute non-dimensional thrust acceleration ratio

acc_thrust = (0.001 * thrmag / mass0) / (xmu / aunit^2);

% compute normalized propellant flow rate

mdot = (thrmag / (9.80665 * xisp)) * 86400.0;

mp_dot = (mdot / mass0) * tfactor;

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% initial and final times and states
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% define state vector at initial time

xinitial(1) = 1.0;

xinitial(2) = 0.0;

xinitial(3) = 1.0;

xinitial(4) = 0.0;

% final conditions

xfinal(1) = 1.52368;

xfinal(2) = 0.0;

xfinal(3) = sqrt(1.0 / xfinal(1));

if (iopt == 1)

    % initial guess for non-dimensional transfer time

    xg(1) = time_g / tfactor;

else

    % initial guess for throttle setting
    xg(1) = throttle_g;

    tof = time_g / tfactor;

end

% initial guess for steering angles (radians)

xg(2:nsegments + 1) = alpha_g;

% transpose initial guess

xg = xg';

if (iopt == 1)

    % upper and lower bounds for non-dimensional transfer time

    xlb(1) = time_lb / tfactor;

    xub(1) = time_ub / tfactor;

else

    % upper and lower bounds for throttle setting

    xlb(1) = throttle_lb;

    xub(1) = throttle_ub;

end

% upper and lower bounds for steering angles (radians)

xlb(2:nsegments + 1) = alpha_lb;

xub(2:nsegments + 1) = alpha_ub;

% transpose bounds

xlb = xlb';

xub = xub';

% define lower and upper bounds on objective function (transfer time)

flow(1) = 0.0;

fupp(1) = +Inf;

% define bounds on final state vector equality constraints

flow(2) = xfinal(1);
fupp(2) = xfinal(1);

flow(3) = xfinal(2);
fupp(3) = xfinal(2);

flow(4) = xfinal(3);
fupp(4) = xfinal(3);

flow = flow';

fupp = fupp';

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% solve low-thrust shooting problem
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

snscreen('on');
```

```
[x, f, inform, xmul, fmul] = snopt(xg, xlb, xub, flow, fupp, 'ilt_shoot');

% print results

fprintf('\n\nprogram ilt_opt - Earth-to-Mars low-thrust trajectory\n');

if (iopt == 1)

    fprintf('\nminimize transfer time\n');

else

    fprintf('\nmaximize final mass\n');

end

fprintf ('\nnumber of segments  %2i \n', nsegments);

fprintf('\ninitial mass            %12.6f kilograms\n', mass0);

fprintf('\nthrust magnitude        %12.6f newtons\n', thrmag);

fprintf('\nspecific impulse        %12.6f seconds\n', xisp);

fprintf('\ninitial state vector \n');

fprintf('\n  radius                %12.6f (AU)\n', xinitial(1));
fprintf('\n  radial velocity       %12.6f (AU/day)\n', xinitial(2));
fprintf('\n  transverse velocity   %12.6f (AU/day)\n', xinitial(3));

fprintf('\n\nfinal state vector \n');

fprintf('\n  radius                %12.6f (AU)\n', f(2));
fprintf('\n  radial velocity       %12.6f (AU/day)\n', f(3));
fprintf('\n  transverse velocity   %12.6f (AU/day)\n\n', f(4));

fprintf('\ntransfer time           %12.6f (non-dimensional)\n', tof);
fprintf('\ntransfer time           %12.6f days \n', tfactor * tof);

if (iopt == 1)

    % minimize transfer time (fixed throttle setting)

    fprintf('\nthrottle setting        %12.6f \n', throttle_g);

else

    % maximize final mass (optimal throttle setting)

    fprintf('\nthrottle setting        %12.6f \n', x(1));

end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% create control, state and trajectory graphics
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% compute duration of each time interval

if (iopt == 1)

    % minimize transfer time option

    deltat = x(1) / nsegments;

else

    % maximize final mass option

    deltat = tof / nsegments;
```

```
end

% specify number of differential equations

neq = 6;

% truncation error tolerance

tetol = 1.0e-10;

% initialize initial time

ti = -deltat;

% set initial conditions

yi(1) = xinitial(1);

yi(2) = xinitial(2);

yi(3) = xinitial(3);

yi(4) = xinitial(4);

yi(5) = 0.0;

yi(6) = 0.0;

% load initial graphics data

npts = 1;

x1(npts) = 0.0;

x2(npts) = yi(1);

x3(npts) = yi(2);

x4(npts) = yi(3);

x5(npts) = x(2);

x6(npts) = yi(4);

% step size guess (non-dimensional time)

h = (1200.0 / 86400.0) / tfactor;

% integrate for all segments

for i = 1:1:nsegments

    % current steering angle

    alpha_wrk = x(i + 1);

    % increment initial and final times

    ti = ti + deltat;

    tf = ti + deltat;

    % integrate from current ti to tf

    yfinal = rkf78('ilt_eqm', neq, ti, tf, h, tetol, yi);

    % create graphics data

    npts = npts + 1;
    x1(npts) = tf;

    x2(npts) = yfinal(1);

    x3(npts) = yfinal(2);

    x4(npts) = yfinal(3);

    x5(npts) = alpha_wrk;

    x6(npts) = yfinal(4);

    % reset integration vector

    yi = yfinal;

    % check for end of simulation

    if (tf >= tof)

        break;

    end

end

fprintf('\npropellant mass         %12.6f kilograms\n', mass0 * yfinal(5));

fprintf('\nfinal mass              %12.6f kilograms\n\n', ...
    mass0 * (1.0 - yfinal(5)));

% determine departure and arrival calendar dates

transfer_time = tfactor * tof;

transfer_angle = rtd * x6(end);

[jd_depart, tsynodic] = departure(jd_guess, transfer_time, transfer_angle);

[cd_depart, ut_depart] = jd2str(jd_depart);

fprintf('\ndeparture calendar date      ');

disp(cd_depart);

jd_arrive = jd_depart + transfer_time;

[cd_arrive, ut_arrive] = jd2str(jd_arrive);

fprintf('\narrival calendar date        ');

disp(cd_arrive);

fprintf('\ntransfer angle          %12.6f degrees\n', transfer_angle);

fprintf('\nsynodic period          %12.6f days', tsynodic);
fprintf('\n                        %12.6f years\n\n', tsynodic / 365.25);

%%%%%%%%%%%%%%%%%%%%%%%%%
% create graphic displays
%%%%%%%%%%%%%%%%%%%%%%%%%

t = (jd_depart - 2451545.0) / 36525.0;

t2 = t * t;

% celestial longitude of the Earth at departure (radians)

theta_earth =  mod(dtr * (100.466449 + 35999.3728519 * t - 0.00000568 * t2), 2.0 * pi);

figure(1);
```

```
x(end+1) = x(end);

stairs(tfactor * x1(1:end), rtd * x(2:end));

title('Two—dimensional Earth—to—Mars Low—thrust Trajectory', 'FontSize', 16);

xlabel('mission elapsed time (days)', 'FontSize', 12);

ylabel('steering angle (degrees)', 'FontSize', 12);

grid;

print —depsc —tiff —r300 control.eps

figure(2);

plot(tfactor * x1, x2);

title('Two—dimensional Earth—to—Mars Low—thrust Trajectory', 'FontSize', 16);

xlabel('mission elapsed time (days)', 'FontSize', 12);

ylabel('radial distance (au)', 'FontSize', 12);

grid;

print —depsc —tiff —r300 rdistance.eps

figure(3);

plot(tfactor * x1, x3);

title('Two—dimensional Earth—to—Mars Low—thrust Trajectory', 'FontSize', 16);

xlabel('mission elapsed time (days)', 'FontSize', 12);

ylabel('radial velocity (au/day)', 'FontSize', 12);

grid;

print —depsc —tiff —r300 vradial.eps

figure(4);

plot(tfactor * x1, x4);

title('Two—dimensional Earth—to—Mars Low—thrust Trajectory', 'FontSize', 16);

xlabel('mission elapsed time (days)', 'FontSize', 12);

ylabel('transverse velocity (au/day)', 'FontSize', 12);

grid;

print —depsc —tiff —r300 vtransverse.eps

figure(5);

plot(tfactor * x1, rtd * (x6 + theta_earth));

title('Two—dimensional Earth—to—Mars Low—thrust Trajectory', 'FontSize', 16);

xlabel('mission elapsed time (days)', 'FontSize', 12);

ylabel('polar angle (degrees)', 'FontSize', 12);

grid;

print —depsc —tiff —r300 polar_angle.eps
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%
% create trajectory display
%%%%%%%%%%%%%%%%%%%%%%%%%%%

figure(6);

hold on;

axis([−1.7 1.7 −1.6 1.6]);

plot (0, 0, 'y*');

% create array of polar angles (radians)

t = 0: pi / 50.0: 2.0 * pi;

% plot Earth orbit

plot(xinitial(1) * sin(t), xinitial(1) * cos(t), 'Color', 'b');

% plot destination planet orbit

plot(f(2) * sin(t), f(2) * cos(t), 'Color', 'r');

% plot and label transfer orbit

plot(x2.* cos(x6 + theta_earth), x2.* sin(x6 + theta_earth), 'Color', 'k');

plot(x2(1) * cos(x6(1) + theta_earth), x2(1) * sin(x6(1) + theta_earth), 'ko');

plot(x2(end) * cos(x6(end) + theta_earth), x2(end) * sin(x6(end) + theta_earth), 'ko');

% label plot and axes

xlabel('X coordinate (AU)', 'FontSize', 12);

ylabel('Y coordinate (AU)', 'FontSize', 12);

title('Two−dimensional Earth−to−Mars Low−thrust Trajectory', 'FontSize', 16);

grid;

axis equal;

zoom on;

print −depsc −tiff −r300 trajectories.eps
```

# Bibliography

[1] World Nuclear Association. Nuclear reactors and radioisotopes for space. http://world-nuclear.org/info/Non-Power-Nuclear-Applications/Transport/Nuclear-Reactors-for-Space/, 2014.

[2] Jr E L Resler and W R Sears. Prospects of magneto-aerodynamics. *Journal of Aeronautical Sciences*, pages 235–246, 1958.

[3] Stack Exchange. What is the current cost-per-pound to send something into LEO? http://space.stackexchange.com/questions/1989/what-is-the-current-cost-per-pound-to-send-something-into-leo.

[4] Bernard H Fong. Europe at the moon: Smart-1 highlights. Technical report, European Space Agency, 2006.

[5] Peter Fortescue, John Stark, and Graham Swinerd. *Spacecraft Systems Engineering*. Wiley, 3rd edition, 2003.

[6] Daniel A Herman, George C Soulas, and Michael J Patterson. Next long-duration test after 11,570h and 237kg of xenon processed. Technical report, NASA Glenn Research Centre, 2007.

[7] Richard R Hofer. Development and characterization of high-efficiency, high-specific impulse xenon hall thrusters. Technical report, University of Michigan, 2004.

[8] NASA JPL. Maven, the 2013 mission to mars. Powerpoint presentation.

[9] NASA. Cassini final supplemental environmental impact statement. Appendix D.

[10] Mukund R Patel. *Spacecraft Power Systems*. CRC Press, 1st edition, 2004.

[11] Surrey Small Satellites. Water resistojet for small satellites .pdf. http://microsat.sm.bmstu.ru/e-library/SSTL/Subsys$_h20rjet_HQ.pdf$.

[12] Peter Vallis Shaw. *Pulsed Plasma Thrusters for Small Satellites*. PhD thesis, University of Surrey, 2011.

[13] A Spitzer. Near optimal transfer orbit trajectory using electric propulsion. *American Astronautical Society*, 1995.

[14] G W Sutton and A Sherman. *Engineering Magnetohydrodynamics*. Dover Civil and Mechanical Engineering, 1st edition, 1965.

[15] George P Sutton and Oscar Biblarz. *Rocket Propulsion Elements*. John Wiley  Sons, 1st edition, 2010.

[16] Airbus Space Systems. http://www.space-airbusds.com/en/programme/eurostar-series-czw.html.

[17] Martin J L Turner. *Rocket and Spacecraft Propulsion: Principles, Practice and New Developments.* Springer-Praxis, 2nd edition, 2000.

[18] Tu Delft University.

[19] TU Delft University. http://www.lr.tudelft.nl/en/organisation/departments/space-engineering/space-systems-engineering/expertise-areas/space-propulsion/propulsion-options/thermal-rockets/arcjets/.

[20] Loo Kang Wee and Giam Hwee Goh. Geostationary earth orbit satellite model using easy java simulation. http://arxiv.org/ftp/arxiv/papers/1212/1212.3863.pdf, 2012.

[21] James R Wertz and Wiley J Larson. *Space Mission Analysis and Design.* Microcosm/Springer, 3rd edition, 1999.

[22] Equation World. Poisson equation. http://eqworld.ipmnet.ru/en/solutions/lpde/lpde302.pdf.