To the Graduate Council:

I am submitting herewith a thesis written by Andrew Allen McCarley entitled "A Propulsive Comparison Study Between Chemical Rockets, Nuclear Thermal Rockets, and Solar Sails for a Manned Mars Exploration Mission." I have examined the final copy of this thesis for form and content and recommend that it be accepted in partial fulfillment of the requirements for the degree of Master of Science, with a major in aerospace engineering.

Dr. Gary A. Flandro, Major Professor

We have read this thesis
and recommend its acceptance:

Accepted for the Council:

Associate Vice Chancellor
and Dean of the Graduate School

STATEMENT OF PERMISSION TO USE

In presenting this thesis in partial fulfillment of the requirements for a Master's degree at The University of Tennessee, Knoxville, I agree that the Library shall make it available to borrowers under rules of the Library. Brief quotations from this thesis are allowable without special permission, provided that accurate acknowledgment of the source is made.

Permission for extensive quotation from or reproduction of this thesis may be granted by my major professor, or in his absence, by the Head of Interlibrary Services when, in the opinion of either, the proposed use of the material is for scholarly purposes. Any copying or use of the material in this thesis for financial gain shall not be allowed without my written permission.

Signature A. Allen McCarley

Date July 29, 1993

A PROPULSIVE COMPARISON STUDY BETWEEN CHEMICAL ROCKETS, NUCLEAR THERMAL ROCKETS, AND SOLAR SAILS FOR A MANNED MARS EXPLORATION MISSION

A Thesis

Presented for the

Master of Science

Degree

The University of Tennessee, Knoxville

Andrew Allen McCarley

August 1993

Dedication

This thesis is dedicated to my parents:

T. Leo McCarley

and

Darothy P. McCarley

Acknowledgments

I would like to thank Brian K. Funk for his assistance in writing the code for the impulsive thrust trajectory program and graphics. Both H. Dan Thomas and Brian K. Funk worked in writing the solar sail trajectory optimization program with me. I would also like to thank H. Dan Thomas for allowing me to use his Iris graphics procedures during my oral defense. Carl Sauer, who twice took time out of his schedule to talk to me on the phone, answered many questions about forcing the solar sail program towards convergence. A special vote of thanks goes to my advisor, Dr. G. Flandro, whose support and encouragement made this project possible. Finally, I would like to thank Dr. Robert Roach and Dr. Roy Schulz for serving on my thesis committee.

Abstract

In order to make interplanetary exploration and exploitation feasible, the cost of interplanetary flight needs to be reduced. Two possible means of reducing this cost are nuclear thermal propulsion and solar sails. The primary purpose of this thesis is to develop optimal missions for chemical, nuclear thermal, and solar sail propulsion systems and then perform a comparison in order to demonstrate the need to develop these technologies.

In order to model these systems, two programs were written for use with high end Macintosh desktop computers and Silicone Graphics Iris workstations that modeled the interplanetary flight (in three dimensions) of spacecraft using these propulsion systems. The impulsive thrust program utilizes Battin's modified Gauss algorithm to solve Lambet's problem with a patched conic approximation. The low thrust program utilizes the iterative procedures of the method of steepest descent and Newton's method to solve the two-point boundary problem that arises from the application of Pontryagin's Maximum Principle to optimize solar sail trajectories. Methods of low thrust trajectory optimization are not available in standard texts, therefore the theory and application of this second program are given in great detail.

The criteria of comparison between the various vehicles were initial mass in low earth orbit (IMLEO) and mission duration. The mission was defined as the transport of 100 MT of usable payload from earth to Mars and the return of 50 MT with a minimum stay on the Martian surface of 30 days. The impulsive thrust vehicles are found to have a 427 day sprint mission with minimum energy cost on April 13, 2018. The IMLEO for the chemical mission is 19, 036.05 MT, a highly restrictive quantity. An acceptance of the health and reliability hazards of longer missions or development of additional technology such as aerobraking would be required to make this option viable. The IMLEO for the nuclear thermal system with the same mission is 2, 013.77 MT; this is

10.58% of the mass needed for the chemical system. The solar sail was found to have a time-optimal 923 day mission opportunity on June 1, 2018. The IMLEO for this solar sail mission with a chemical booster stage to provide earth escape is 452.24 MT. This is 22.46% of the mass required by NTP and only 2.38% of the mass required by chemical propulsion. An added benefit is that all of the mass excepting the chemical booster is returnable and presumably reusable. However, the lengthy mission time required by the sail indicates that it would be best used in support of a permanent exploitation of Mars (a "cycler" cargo ferry) or in an effort to reduce IMLEO of a chemical or nuclear thermal exploration mission (a cargo vessel carrying all supplies not needed to sustain the crew).

Because of the tremendous opportunities for cost savings presented by nuclear thermal propulsion and solar sails, research and development of both technologies should be restarted. Interplanetary travel via chemical propulsion is not currently a viable option and existence of these systems could be the key to making exploration and exploitation of Mars an achievable goal.

Table of Contents

List of Tables

List of Figures

List of Symbols and Acronyms

a	Semi major axis
a	Acceleration vector
a_O	Solar sail characteristic acceleration
AU	Astronomical Unit
a_r	Acceleration in the radial direction
a_Φ	Acceleration in the tangential direction
a_Ψ	Acceleration in the normal direction
C3	Square of the hyperbolic excess velocity
e	Eccentricity
F_r	Engine thrust in the radial direction
$F\Phi$	Engine thrust in the tangential direction
$F\Psi$	Engine thrust in the normal direction
g	Gravity at the earth's surface (9.81 km/s^2)
GM	Universal gravitational constant times the mass of the Sun
G_π	Payload mass
h	Angular momentum
H	Hamiltonian
HEO	High Earth Orbit
i	Inclination
$\hat{i}$	Unit vector in the direction of incident sunlight
IMLEO	Initial Mass in Low Earth Orbit
Isp	Specific impulse
JPL	Jet Propulsion Laboratories
LEO	Low Earth Orbit

m	Spacecraft mass
M_1	Mass of spacecraft before a propulsive maneuver
M_2	Mass of spacecraft after a propulsive maneuver
M_{Dry}	Rocket stage mass after expending all usable fuel
M_{Engine}	Engine mass
M_{Frig}	Mass of sorption refrigeration system
M_P	Propellant mass
MT	Metric ton (1,000 kg)
n	Mean motion
$\hat{n}$	Unit vector normal to the sail's surface
NASA	National Aeronautics and Space Administration
NERVA	Nuclear Engine for Rocket Vehicle Application
NERVA-ENABLER	An advanced form of NERVA engine
NTP	Nuclear Thermal Propulsion
p	Orbital parameter
P_A	Solar pressure per unit area one AU from the sun
P_r	Auxiliary variable conjugate to radial position
P_Φ	Auxiliary variable conjugate to celestial longitude
P_Ψ	Auxiliary variable conjugate to celestial latitude
P_{V_r}	Auxiliary variable conjugate to radial velocity
P_{V_Φ}	Auxiliary variable conjugate to tangential velocity
P_{V_Ψ}	Auxiliary variable conjugate to normal velocity
r	Radial distance
$\mathbf{R}$	Gravitational potential field of the sun
R_f	Reflectivity of the sail material
R_o	Reference distance of 1 AU from the sun

S_o	Sail surface area
TEI	Trans. Earth Injection
TMI	Trans. Mars Injection
UCLA	Universtiy of California at Los Angeles
V_r	Velocity in the radial direction
V_Φ	Velocity in the tangential direction
V_Ψ	Velocity in the normal direction
W_{Cool}	Total cooling load in Watts
$\mathbf{x}$	Initial conditions for sail trajectory optimization
$\mathbf{y}$	Results of an iteration during sail trajectory optimization
β	Sail clock angle
ΔV	Required velocity change for a propulsive maneuver
δ	Sail material thickness
ρ	Sail material density (with a factor for support structures)
μ	Gravitational parameter of the attracting body
σ	Ratio of spacecraft mass after/before propulsive maneuver
θ	Sail cone angle
ω	Argument of the periapsis
Φ	Celestial longitude
Λ	Sail areal density (including support structures)
Ψ	Celestial Latitude
Ω	Longitude of the ascending node

I. Introductory Remarks

1.1 Statement of the Problem

Despite the demise of the manned Mars mission sponsored by President Bush, strong interest remains for the exploration of the red planet. The human drive to discover and learn, our increasing need for raw materials, and the expansion of the species are all factors which compel this undertaking. However, increasing budgetary constraints continue to plague exploration efforts. With these constraints in mind, inexpensive means need to be developed to carry out this venture.

Two possible propulsion schemes that could potentially lower the cost of interplanetary missions are nuclear thermal rockets and solar sails. Neither of these are new technologies, but neither have been given a great deal of development. The primary purpose of this thesis is to examine the performance in delivering interplanetary payloads of nuclear thermal and solar sail spacecraft alongside a more conventional chemical propulsion system in an effort to demonstrate the need to develop these technologies. In order to meet this goal, computational procedures have been developed to model the interplanetary flight of these systems in three dimensions and determine the most efficient flight times and dates of launch.

The two criteria used to compare the missions achievable by the three propulsion schemes are initial mass in low earth orbit (IMLEO) and total mission duration. Both of these parameters need to be minimized in order to keep the mission as inexpensive and safe as possible. Technology development costs, which would be required for the chemical mission as well as the competing schemes, are not included in this comparison. It would be unfair to assign the entire development cost to a single undertaking when the

resulting technology would be used in denumerable space ventures for scores of years.

The IMLEO is a measure of the expense of the mission. It represents the amount of mass that must be hauled up from the earth's surface and, for the chemical and nuclear thermal spacecraft, it is also an indirect measure of how much fuel must be used to accelerate the vehicle out of and into its parking orbits around the earth and Mars. (Indirect because fuel is considered within IMLEO.) IMLEO will be particularly expensive if the shuttle is used to lift mass from the earth's surface rather than some yet to be designed heavy lift launch system.

Total mission duration is a measure of the safety of the mission and also enters into the calculation of mission cost. The longer the mission, the longer the astronauts will be subjected to the debilitating effects of zero gravity and cosmic radiation and the longer that mission hardware must be relied upon. Longer missions also add the cost of an increased amount of life support materials that must be carried. This latter effect can either increase IMLEO or, in a fixed payload system, subtract from the amount of scientific equipment that can be carried to perform the mission.

1.2 Overview of Chemical Propulsion

Chemical rockets have been in existence for centuries. The Chinese developed the first solid fueled rockets around 1232 AD as an extension of the incendiary devices, "firearrows", that they had previously attached to arrows. Typical fuel in this period was charcoal, saltpeter, and sulfur. Herman Oberth and Robert Goddard both began experimenting with liquid fueled rockets in the early twentieth century. The Germans produced the first modern liquid fueled rocket, the A-4, in the early 1940's under the leadership of Werner von Braun and General Dr. Walter Dornberger. Once this rocket began to be used for military purposes in World War II it received the designation V-2. The Germans produced approximately 3000 V-2 rockets during the war, and were

proceeding to bigger, 4-stage rockets for the intercontinental bombardment of North America. Known as the A-10 series, these rockets were in the process of being fabricated when the war ended [1]. Following the war, von Braun came to work for the American rocket program and provided much of the leadership for the Mercury, Gemini, and Apollo projects. These programs had been spurred into existence by the launching of the first artificial earth satellite, Sputnik, and the first man into orbit, Yuri Gagarin, both by the Soviet Union. (The Soviets also placed the first woman into orbit, Valentina Tereshkova, and the first space walker, Aleksei Leonov.) The "classic" means of space transportation, chemical rockets are, in fact, the only means that have ever been used for manned space exploration.

For the purposes of this study, a cryogenic O_2/H_2 chemical system was assumed. Active refrigeration, utilizing a sorption compressor, prevents boiloff of the propellants during the long coast periods. The sorption compressor works by first absorbing the gas onto a suitable medium and then desorbing the gas at high temperature to produce a high pressure gas. The gas is then cooled by expansion through a Joule-Thomson valve. This system is similar to that modeled by Frisbee in a recent JPL study [2], and is assumed to produce an Isp of 470 s.

1.3 Overview of Nuclear Thermal Propulsion

Nuclear thermal propulsion has long been considered for the manned Mars mission and offers much higher values for Isp than is attainable with chemical rockets. This propulsion scheme heats the propellant (usually hydrogen) by passing it through a fission fuel core. Two classes of nuclear thermal engines are known as solid core, using conventional nuclear technology, and gas core, in which the propellant is heated with the radiant energy of a high-temperature fissioning plasma. While actual design and testing

has been performed for solid core rockets, the gas core nuclear engine is a far-term project which still faces significant engineering challenges [3].

In the late sixties and early seventies efforts were made to develop a large, high thrust to weight solid core engine for the manned Mars mission. This engine was based on hydrogen-cooled graphite reactor technology and became the Nuclear Engine for Rocket Vehicle Application (NERVA). This engine achieved operation at 1500 MW with a thrust of 333 kN and was designed for a 10-hour life and 60 operating cycles. [3]

The Isp of NERVA was 825 s, but advances in reactor technology since the early seventies would allow a modern solid core nuclear thermal rocket to attain more efficient performance. The Isp achievable by a given solid core rocket does have an upper limit as the operating temperature of the working fluid must be less than the melting point of the fuel, moderator, and core structural materials. (This is the reason behind the desire to develop gaseous core technology.) This requirement restricts the attainable value to around 1000 s. [3] The upgraded NERVA-ENABLER concept was cited by Clark to offer an Isp within the range of 925 s to 1080 s [4]. For the purposes of this study, a solid core engine with an Isp of 925 s was assumed.

1.4 Overview of Solar Sails

Solar Sails gain momentum by the reflection of photons. Their thrust depends upon the photon flux or photon intensity in the spatial location of the vehicle and upon the orientation of the sail. In this manner, they allow one to fly in space free of propellant or mechanical aid and thus bring great reduction in IMLEO. Solar sailing is a very old idea. Two Russians, Konstantin Tsiolkovsky and Friderickh Arturovich Tsander, were the first scientists to mention propulsion by solar pressure. In 1924 Tsander wrote, "For the purposes of flight in interplanetary space I am working on the idea of flying, using

tremendous mirrors of very thin sheets, capable of achieving favorable results." In this same paper Tsander proposed earth-orbiting space stations (5).

The first serious technical paper on solar sails was published by Carl Wiley. Entitled "Clipper Ships of Space," this nonfiction article was published in the May 1951 edition of *Astounding Science Fiction.* At this time Wiley used the pen name Russell Sanders because he did not wish to jeopardize his credibility by discussing far-fetched ideas in a science-fiction magazine. The first professional article dealing with solar sails was published in *Jet Propulsion* in 1958 by Dr. Richard Garwin. This article included preliminary calculations of sail vehicle performance (5).

In 1958 Ted Cotter came up with the idea of spinning the sail for stability and wrote a short article describing the technical aspects of this feat. *Time* magazine mentioned his work in the 1958 editorial "Trade Winds in Space," and Ted Villers wrote his masters thesis at MIT on the topic in 1960. In 1961, a short course was offered on solar sails at UCLA. Richard MacNeal and John Hedgepath invented a helicopter-like spinning sail called the heliogyro between 1965 and 1967 (5).

NASA became interested in the technology during the mid 1960's, but interest died with shrinking budgets after the Apollo program and by the early 1970's no development of solar sails was being performed at all. Interest was renewed when Jerome Wright designed a solar sail rendezvous mission to Halley's comet with a four year flight time. Prior to this time, it was believed that the only way to rendezvous with Halley was utilizing advanced solar or nuclear electric propulsion with a ten year flight time. (The extreme difficulty of Halley rendezvous missions is caused by the comet's retrograde orbit.) Wright's mission design prompted the 1977-1978 JPL solar sail design study, and the result of this preliminary examination was that the solar sail concept was found to be feasible for Halley rendezvous. However, NASA Lewis was proposing a solar-electric rendezvous at this time, and the solar sail option became caught up in a

competition with this technology. After the competition solar sails were deemed by NASA officials to be insufficiently "mature," and the solar-electric alternative was chosen for the Halley mission. This project was later scrapped, however, and the United States ended up in possession of a space program with no Halley mission at all(5).

Currently there is no NASA funded work in solar sails. After the competition with solar-electric propulsion, several JPL scientists formed the World Space Foundation and continue to seek public funding for research into the topic. This group has built a prototype sail and conducted a demonstration of its deployment.(5) In 1993, the Russian space program deployed a large reflecting mirror and demonstrated solar sail techniques in operation.

1.5 Assumptions Common to All Propulsive Methods

In all cases within this paper a patched conic approximation will be assumed for trajectory calculations. That is, throughout the course of the spacecraft's interstellar flight the vehicle will be assumed to be outside the sphere of influence of all celestial bodies except the sun. During escape and rendezvous maneuvers, the vehicle will be assumed to reside only within the spheres of influence of the departure and rendezvous planets, respectively. Thus at all times the body centered gravity forces have been simplified from an n-body problem to a two-body problem.

The equations of motion for an accelerating spacecraft within a two-body gravitational field are as follows:

$$\frac{d}{dt}(m\dot{r}) - mr\left(\dot{\Phi}^2\cos^2(\Psi) + \dot{\Psi}^2\right) = F_r - \frac{GMm}{r^2} \tag{1.1}$$

$$\frac{d}{dt}\left(mr^2\cos^2(\Psi)\dot{\Phi}\right) = r\cos(\Psi)F_\Phi \tag{1.2}$$

$$\frac{d}{dt}\left(mr^2\dot{\Psi}\right) + mr^2\cos(\Psi)\sin(\Psi)\dot{\Phi}^2 = rF_\Psi \tag{1.3}$$

Here m is the mass of the spacecraft, r is the distance from the center body, G is the universal gravitational constant, and M is the mass of the central body. These equations are valid within the sphere of influence of the earth, during interplanetary flight, and within the sphere of influence of Mars using the relevant value of GM (the gravitational parameter) for each case. Were the patched conic approximation not being used, vector summations of the relevant gravitational parameters would appear in the above equations rather than the single gravitational term in equation (1.1).

The coordinate system used by (1.1)-(1.3) is defined by Fig. 1-1. The symbols F_r, F_Φ, and F_Ψ are the acceleration forces produced by the spacecraft propulsion system. Equations (1-1)-(1.3) can be used for all spacecraft with the proper substitutions being made for the F terms depending upon the propulsion system being used. These equations become especially simple to use in the case of the solar sail because m is constant.

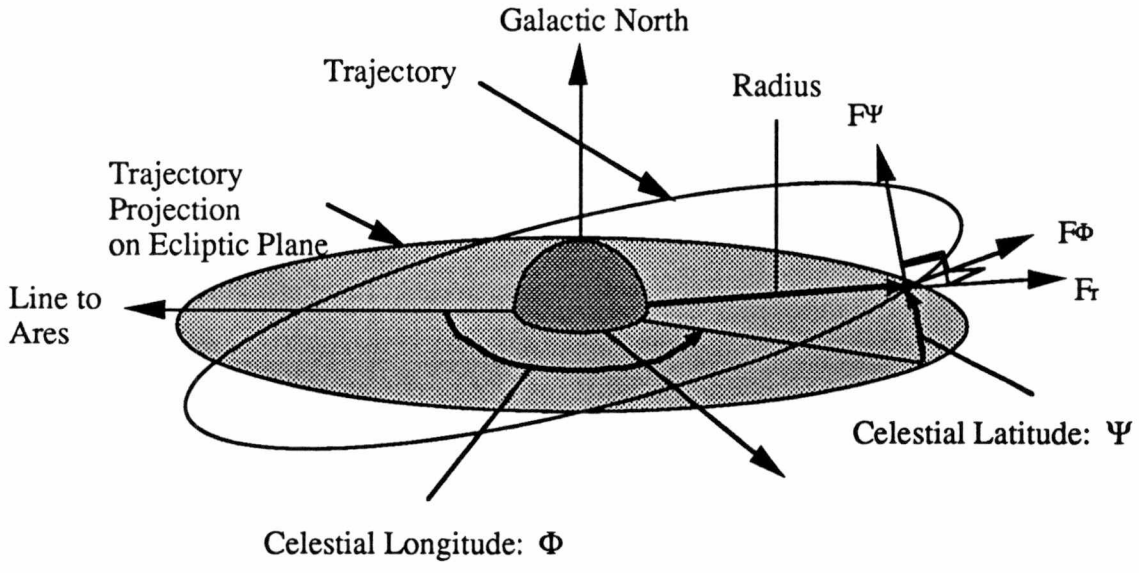

Fig. 1-1: Definition of the r,Φ,Ψ Coordinate System

All propulsive systems are assumed to transport a total payload of 100 metric tons[6] and all spacecraft are required to have a stay time of at least 30 days in Martian orbit[7]. One half of this payload (Martian lander, habitat, exploration vehicles, life support supplies, etc.) is assumed discarded at Mars or depleted by the time the return trip to earth is undertaken. Departure from the earth is assumed to be from a typical space station orbit of 500 km altitude with a 28.5 degree inclination. For the solar sail option, this requires a chemical booster to break free of the earth's gravity. Martian parking orbits are assumed to have an altitude of 6000 km. While this is very near to the orbit of Phobos, no examination has been undertaken within this study of using Phobos for a staging area or refueling station. The parking orbits were selected to make this thesis compatible with recent studies undertaken at JPL [2,3].

II. Chemical Rocket Propulsion

2.1 Definition and History of Lambert's Problem

Impulsively thrusted space faring vehicles are by far the most familiar variety at the present time, and their principles of movement are the best understood. In order to conduct an interplanetary mission utilizing this method of propulsion, a burst of thrust is applied at the departure planet to escape its the sphere of influence. This burst also places the vehicle into a predetermined conic orbit about the sun which will carry it near to the destination planet. The vehicle then coasts for the duration of its interplanetary journey, kept to its orbit by the laws of celestial mechanics. Once near the destination planet a second burst of thrust places the vehicle into a convenient parking orbit. For the return trip this entire process is repeated.

An interesting methodology is utilized when determining trajectories with this style of propulsion. Since the vehicle is coasting for the duration of its trip between the launch and destination bodies, it's required heliocentric orbit is completely determined by the position of the two planets at launch and rendezvous and the time of flight desired to traverse this distance. This process of determining the orbit of the vehicle based upon two positions and a flight time is called Lambert's problem.

In the days of its origin, the solution of Lambert's problem was essential to gathering the orbital elements of celestial bodies that astronomers wished to observe. Today it is also used to target spacecraft and missiles on missions such as the one described in this work. "Over the years a variety of techniques for solving this problem has been developed. Each is characterized by a particular choice of independent variable to be used in an iterative algorithm to determine the orbital elements." (Battin, p. 295)

Carl Frederick Gauss is credited with the first real progress in the solution of this problem, and for this reason what has been described within this work as Lambert's problem is referred to in some sources as "the Gauss problem." In 1801 Gauss, like many of his contemporaries, was fascinated by a new heavenly body, named Ceres, discovered by Piazzi on January 1, 1801. Piazzi was only able to observe this body for about one month before it was blotted out of his sight by glare from the sun. The challenge of reestablishing observation of Ceres once it passed beyond the sun's glare grasped Gauss' attention (8).

Gauss tells of his discovery in the preface of his book *Theoria Motus Corporum Coelestium in Sectionibus Conicus Solem Ambientium*--Theory of the Motion of the Heavenly Bodies Moving about the Sun in Conic Sections. The following quote from that preface is taken from Battin (Battin, pp. 295-296):

> "To determine the orbit of a heavenly body, without hypothetical assumption, from observations not embracing a great period of time, and not allowing a selection with a view to the application of special methods, was almost wholly neglected up to the beginning of the present century; or, at least, not treated by anyone in a manner worthy of its importance; since it assuredly commended itself to mathematicians by its difficulty and elegance, even if its great utility in practice were not apparent. An opinion had universally prevailed that a complete determination from observations embracing a short interval of time was impossible, -- an ill-founded opinion, -- for it is now clearly shown that the orbit of a heavenly body may be determined quite nearly from good observations embracing only a few days; and this without any

hypothetical assumption.

"Some ideas occurred to me in the month of September of the year 1801, engaged at the time on a very different subject, which seemed to point to the solution of the great problem of which I have spoken. Under such circumstances we not unfrequently, for fear of being too much led away by an attractive investigation, suffer the associations of ideas, which, more attentively considered, might have proved most fruitful in results, to be lost from neglect. And the same fate might have befallen these conceptions, had they not happily occurred at the most propitious moment for their preservation and encouragement that could have been selected. For just about this time the report of the new planet, discovered on the first day of January of that year with the telescope at Palermo, was the subject of universal conversation; and soon afterwards the observations made by that distinguished astronomer Piazzi [Giuseppe Piazzi (1746-1826)] from the above date to the eleventh of February were published. Nowhere in the annals of astronomy do we meet with so great an opportunity, and a greater one could hardly be imagined, for showing most strikingly, the value of this problem, than in this crisis and urgent necessity, when all hope of discovering in the heavens this planetary atom, among innumerable small stars after the lapse of nearly a year, rested solely upon a sufficiently approximate knowledge of its orbit to be based upon these very few observations. Could I ever have found a more seasonable opportunity to test the practical value of my conceptions, than now in employing them for the determination of the orbit of the

> planet Ceres, which during these forty-one days had described a geocentric arc of only three degrees, and after the lapse of a year must be looked for in a region of the heavens very remote from that in which it was last seen? This first application of the method was made in the month of October, 1801, and the first clear night, when the planet was sought for [by Baron Franz Xaver von Zach on December 7, 1801] as directed by the numbers deduced from it, restored the fugitive to observation. Three other new planets, subsequently discovered, furnished new opportunities for examining and verifying the efficiency and generality of the method."

The "planet" Ceres mentioned by Gauss in the above quote was actually an asteroid, the first of hundreds discovered in the gap between Mars and Jupiter. It is interesting to include at this point another quote:

> It is ironic that the discovery of Ceres coincided with the publication of the famous philosopher Hegel of a vitriolic attack on astronomers for wasting their time in search for an eighth planet. If they paid some attention to philosophy, Hegel asserted, they would see immediately that there can be precisely seven planets, no more, no less. This slight lapse on Hegel's part no doubt has been explained by his disciples even if they cannot explain away the hundreds of minor planets which mock his philosophical ban. (Bate, p.228)

Because Lambert's problem is so well established and widely published, a cursory glimpse at a number of references will lead one to a convenient algorithm for solution.

Such references noted in this work are Bate, Mueller, and White[9] and Battin [8]. The algorithm used by the author follows closely to that presented in Battin. This method consists of a slightly modified version of the original Gauss method with a continued fraction scheme introduced into several of the subroutine loops to assure convergence. The exact algorithm is presented as Appendix A.

2.2 Selection of Mission Class

Due to the nature of impulsive thrust, it is possible to achieve virtually any flight time to the desired destination from virtually any launch date. This flexibility requires only that one accept what may be unreasonable energy costs, or perhaps even energy costs impossible by current technology, in order to achieve all of these flight possibilities. In order to optimize an interplanetary trajectory with impulsive thrust, one must look for opportunities when the launch and destination planets are so aligned as to minimize the energy costs required to achieve the desired mission. This approach is the one that was used during this investigation.

As stated in Chapter 1, it is desired to keep flight time, and thus the astronauts' exposure to the debilitating effects of zero gravity and cosmic radiation, to a minimum. For this reason, a sprint class mission was selected as the mission profile of choice in this investigation. In this class of mission, the spacecraft departs earth when earth and Mars are near opposition, travels to Mars on a sprint trajectory, remains for the duration of the on-site mission (30 days in this investigation), and then returns to earth. The astronauts are not required to wait on the Martian surface for the full 2.13 year synodic period (defined below) of Mars as is required by missions seeking solely to minimize energy expenditure.

Effort began in finding optimal launch windows for the impulsive thrust case by using a typical sprint mission profile, presented by Mark and Smith [10], as a probe.

Mark's 150 day outbound, 30 day stay, 240 day return mission was run using the program given in Appendix A in an iterative manner throughout the 2000-2020 period. Initial launch date from earth was advanced by thirty days for each iteration.

Selection of favorable mission windows was based upon the summation of the C3's required for all of the propulsion maneuvers required for each mission opportunity. C3 is the square of the hyperbolic excess velocity and is thus a convenient quantity for measuring the energy costs of a mission. The summation of C3 required for launch and rendezvous on the outbound leg, for launch and rendezvous on the return leg, and for the total mission for the period 2000-2010 are presented in Fig. 2-1. The same data for the period of 2010-2020 is presented in Fig. 2-2.

The synodic period of two planets is defined as the time between two successive alignments of those planets such that the angular distance between their radial lines to the sun is the same. The synodic period of earth and Mars is approximately 2.13 years (Bate, p.368). Thus it is that the local minimae for total C3 in Figs. 2-1 and 2-2 occur at this frequency. However, since neither the orbit of the earth nor of Mars is perfectly circular and since the orbits are inclined relative to each other, each of these local minimae differ in magnitude. As can be seen by examining these two graphs, the most favorable opportunity for a sprint mission occurs at the local minimum for total C3 in 2018. A close-up of the curves in the region of this favorable mission opportunity is presented as Fig. 2-3.

Other favorable opportunities are found to exist in 2003, 2005, and 2016 as seen in Fig. 2-1 and Fig. 2-2. The opportunity in 2003 is the one investigated by Mark and Smith. However, since the purpose of this study is to select best case scenarios for comparison of propulsion systems, those opportunities are not investigated further here.

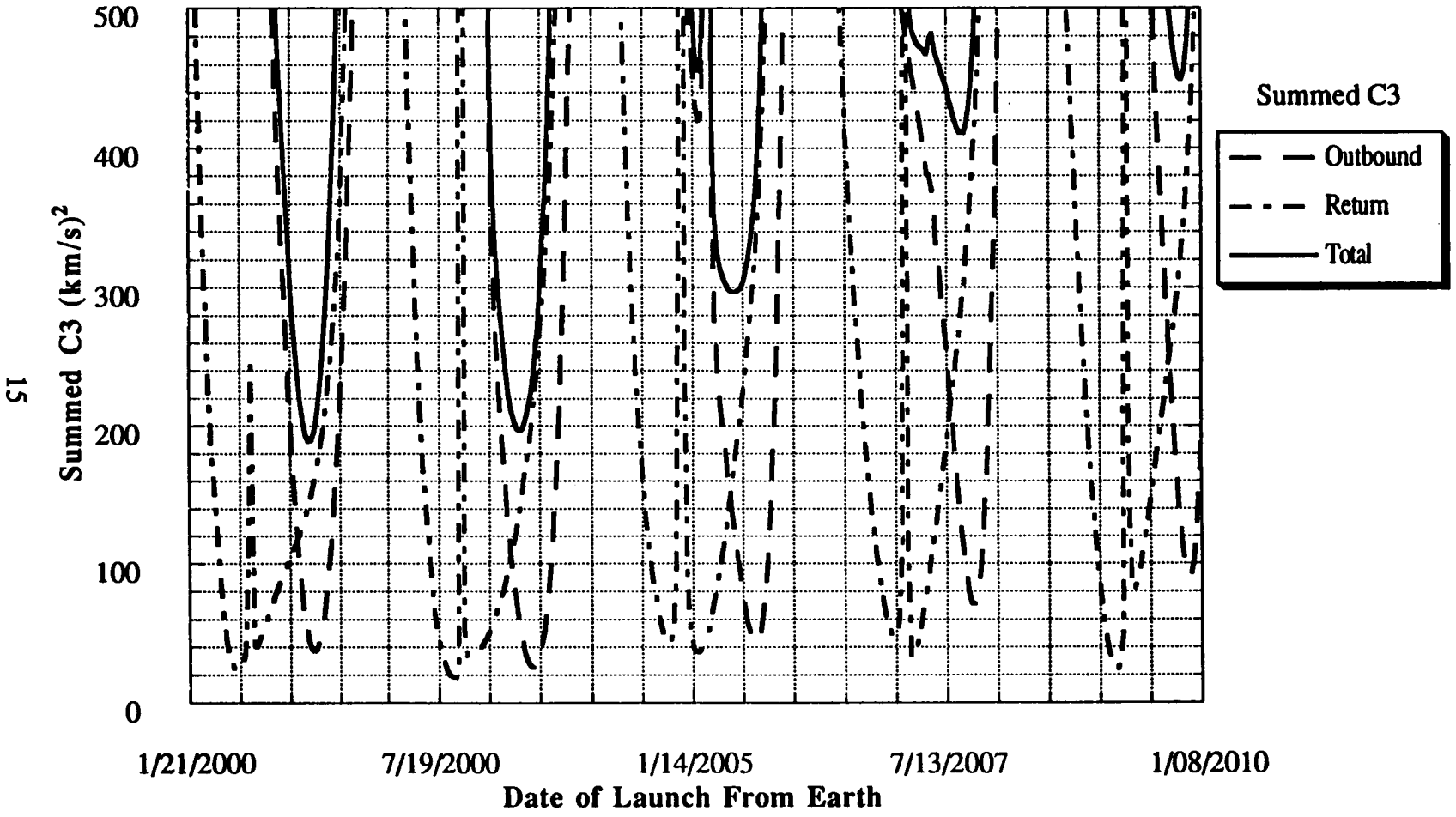

Fig. 2-1: C3 Required for Probe-Case Mars Sprint Mission: 2000-2010
(150 Days to Mars, 30 Day Loiter, 240 Day Return Trip)

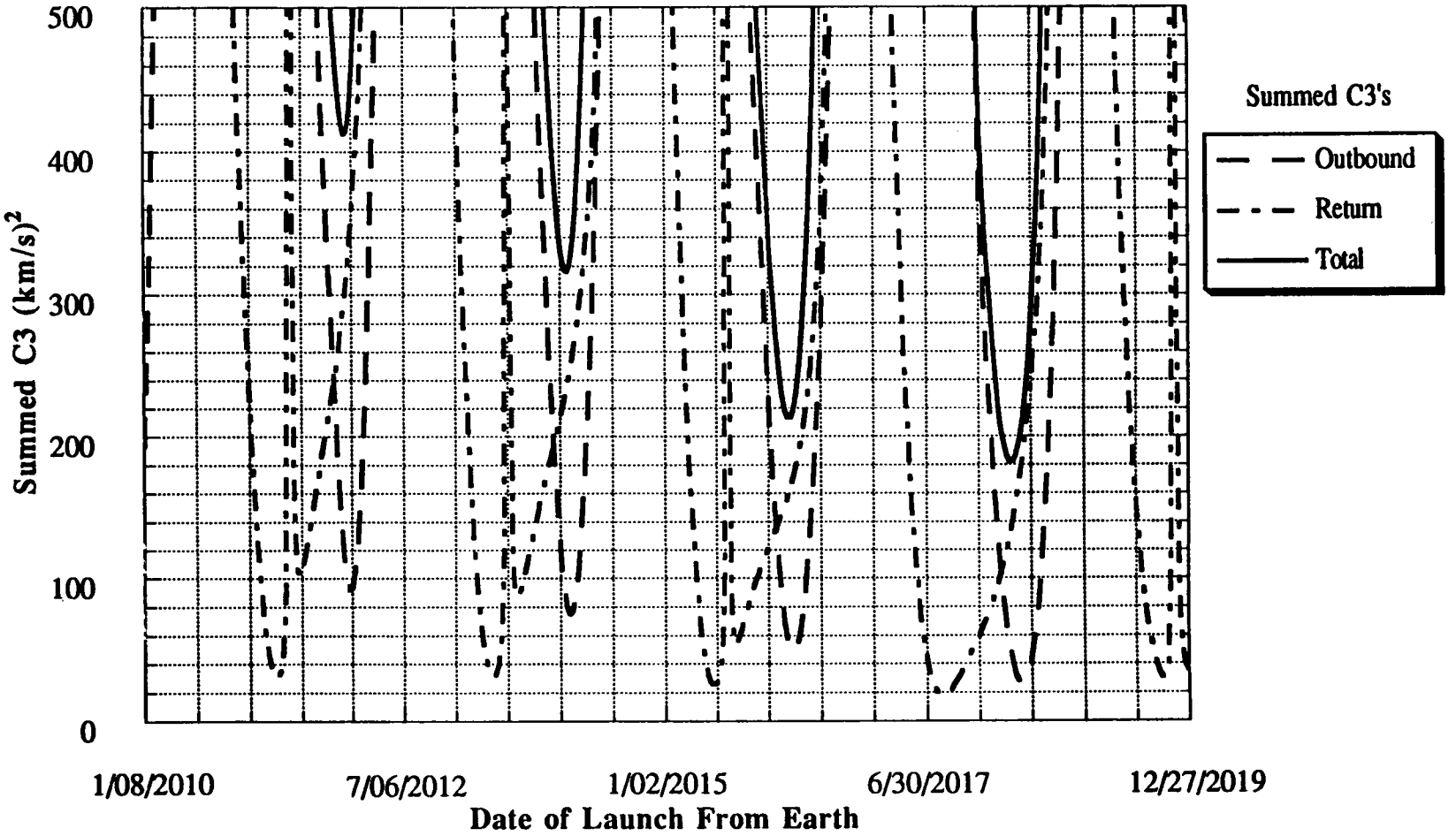

Fig. 2-2: C3 Required for Probe-Case Mars Sprint Mission: 2010-2020
(150 Days to Mars, 30 Day Loiter, 240 Day Return Trip)

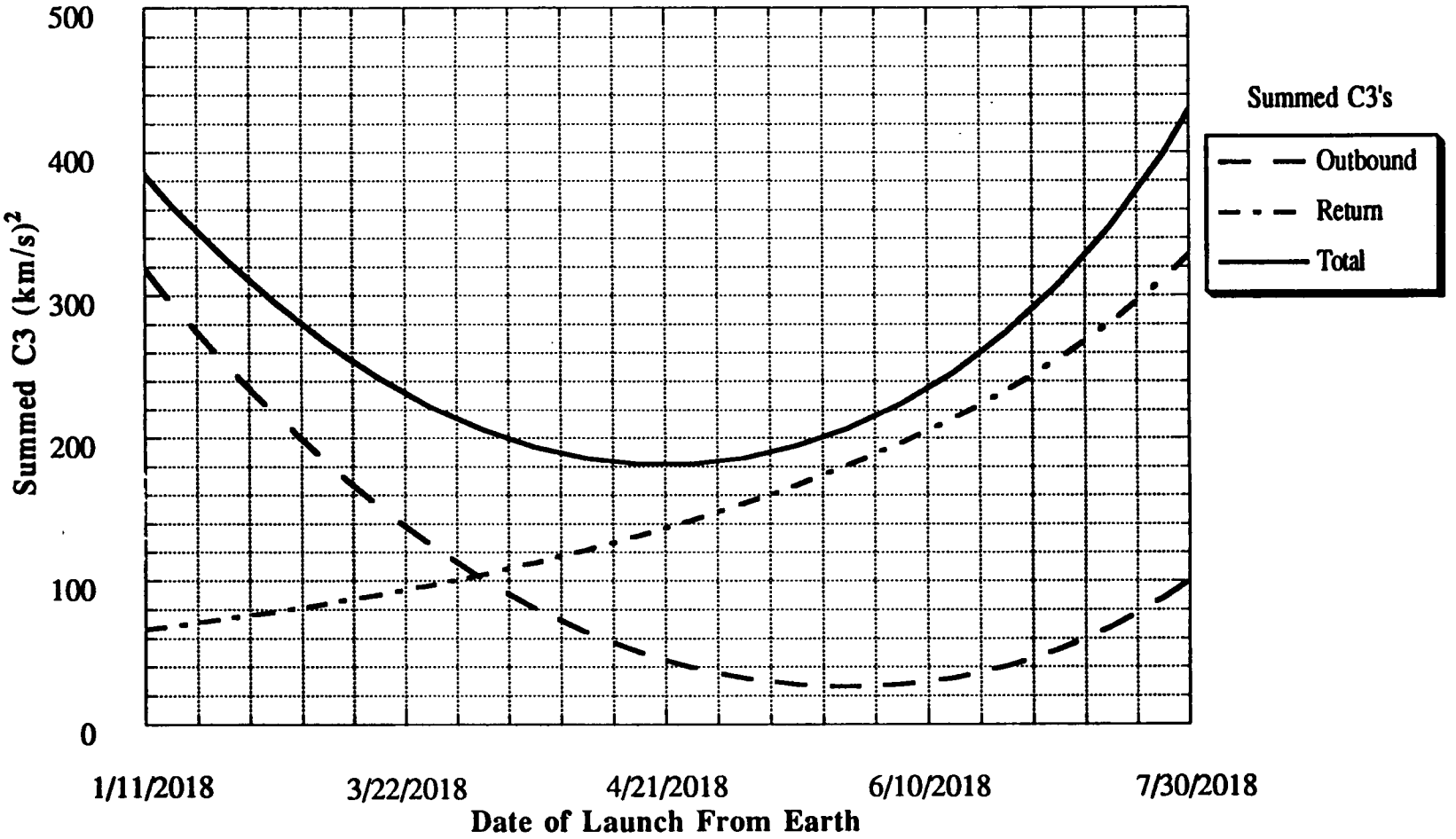

Fig. 2-3: C3 Required for Best Probe-Case Mars Sprint Mission Opportunity
(150 Days to Mars, 30 Day Loiter, 240 Day Return Trip)

2.3 Trajectory Optimization and Launch Windows

The 150/30/240 day mission profile discussed above served to locate favorable launch opportunities, but the duration of the outbound and return legs were chosen in a rather arbitrary method. In order to select the optimal mission it is necessary to examine the variance of total mission C3 required with changing launch date and mission leg length. Again, the computational procedure given in Appendix A was executed in an iterative manner to generate this required data.

Constant C3 curves for both mission legs are presented as Figs. 2-4 and 2-5. These graphs show the energy required for mission legs of varying duration launched from varying dates. The curves for Fig. 2-4 show a double loop phenomenon, demonstrating the effect of rendezvousing with Mars before or after it crosses the 180° phase angle line with the earth's position at launch. (i.e. the spacecraft traverses less or more than 180°.) The former is called a type I orbit, while that latter is referred to as a type II orbit. The curves for Fig. 2-5 are much more well behaved.

It is impossible to select the mission profile by simply choosing the two points of minimum energy required on these graphs whether the type I or II outbound trajectory is chosen. In either case the astronauts would be required to depart from Mars before arriving there. Instead, the initial launch date from earth and flight time of each leg must be chosen so as to present the best possible tradeoff between the energy requirements for the two legs.

Note that the elliptical curves in Fig. 2-5 lie rather flat in the horizontal plane. Thus, regardless of the date of launch from Mars, the minimum energy required by the return leg will be very nearly achieved by selecting a flight time of 240 days. With this flight time selected, and holding the desired on-site mission time to 30 days (a longer time would only move the mission to a higher energy curve on Fig. 2-5), it becomes a simple matter to run the program given in appendix A once again in an iterative fashion

and generate curves of constant total mission C3 for varying earth launch date and outbound flight time.

The graph of these results is presented as Fig. 2-6. As can be seen, the curves of Fig. 2-6 consist of a set of reasonably well behaved concentric ellipsoids. The point of minimum energy required lies at an initial earth launch date of April 13, 2018 with a total mission time of 427 days. This translates into a 157 day outbound leg, a 30 day stay on Mars, and then a 240 day return trip to the earth. The spacecraft thus makes its final rendezvous with earth on June 14, 2019. A schematic of this mission profile is given as Fig. 2-7. A complete list of data defining a mission using this optimal launch opportunity is presented in Table 2-1. Data listed includes propulsive energy requirements, planetary positions at relevant launch and rendezvous times, and heliocentric conic orbital elements.

In Figs. 2-4 and 2-5 typical NASA 30 day launch windows have been superimposed over the launch energy curves. In an actual mission plan, it would be a serious mistake to base the fuel requirement and vehicle sizing calculations upon the data presented in Table 2-1. Instead, these calculations should be based upon the worst case scenario within the launch window. Again, however, the purpose of this study is a comparison of best case scenarios between differing propulsive options. Thus the adverse affects of a launch within the mission window at a date other than the optimal are not considered.

2.4 Mission Mass Schedule

The spacecraft used for the chemical mission is as described in Chapter I. It is a two stage O_2/H_2 vehicle with three sets of tanks in the second stage. A sorption compressor is used to prevent boiloff of the cryogenic propellants. The fuel each tank

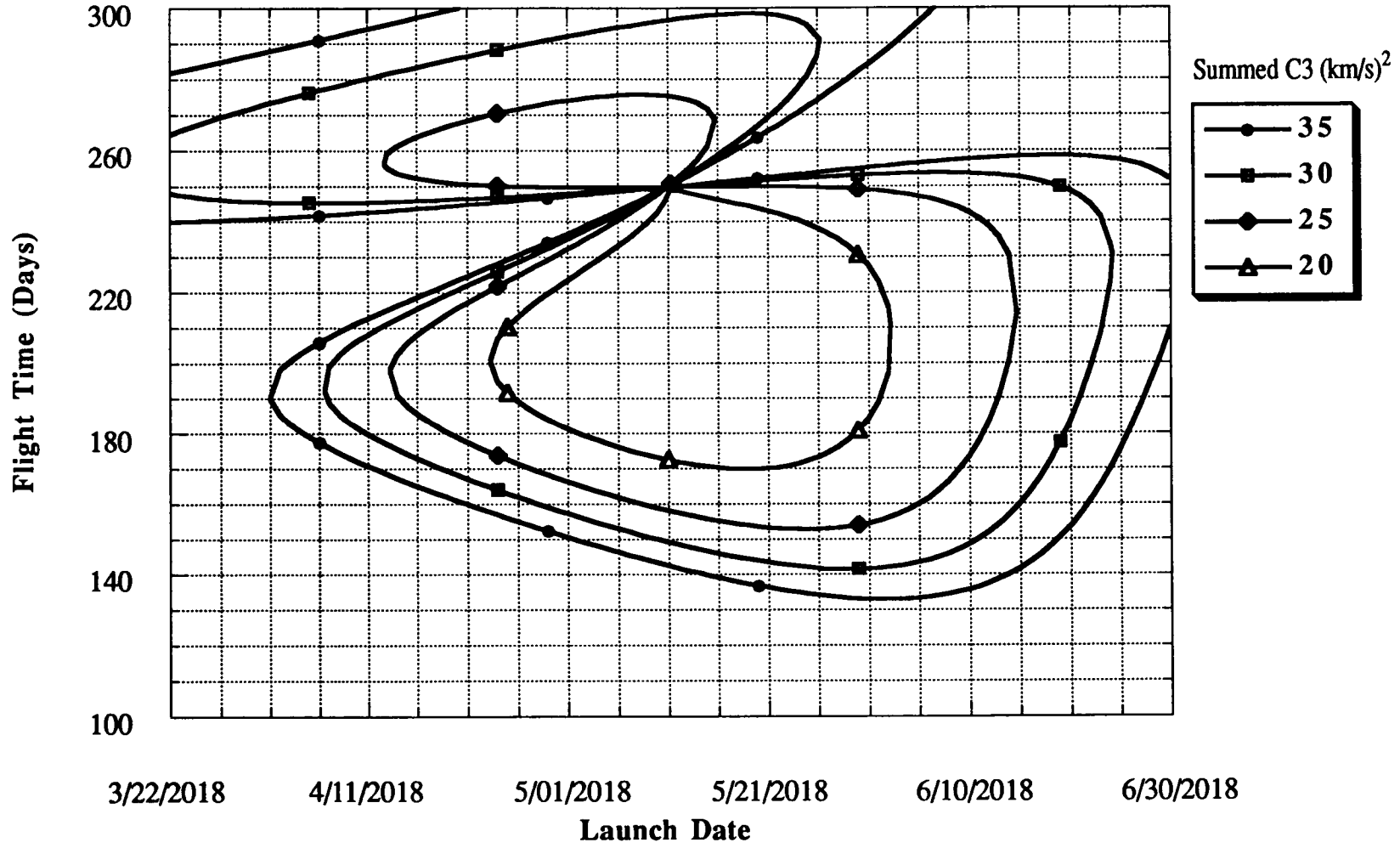

Fig. 2-4: Earth to Mars Constant Summed C3 for Impulsive Thrust Missions Near Optimum

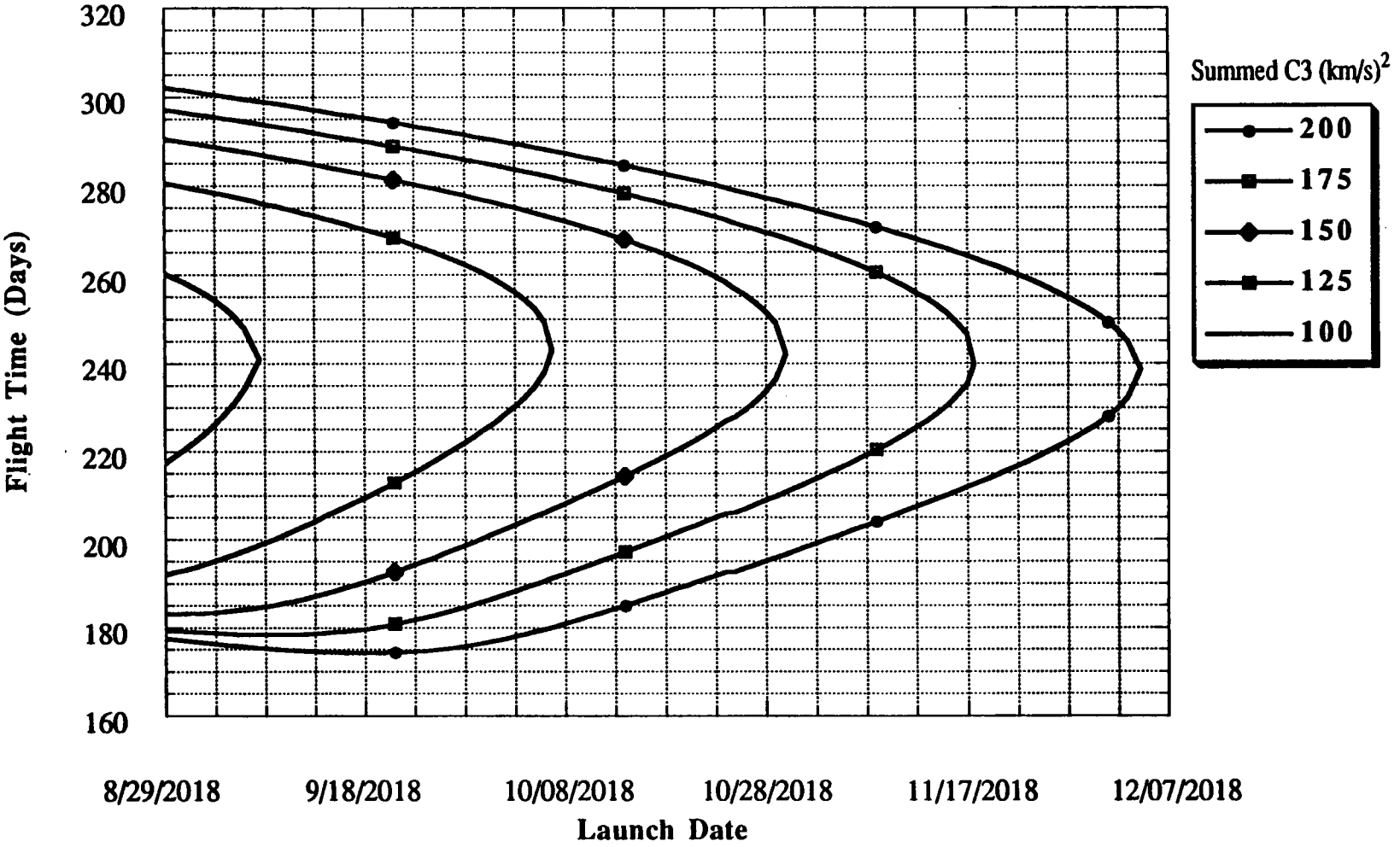

Fig. 2-5: Mars to Earth Constant Summed C3 for
Impulsive Thrust Missions Near Optimum

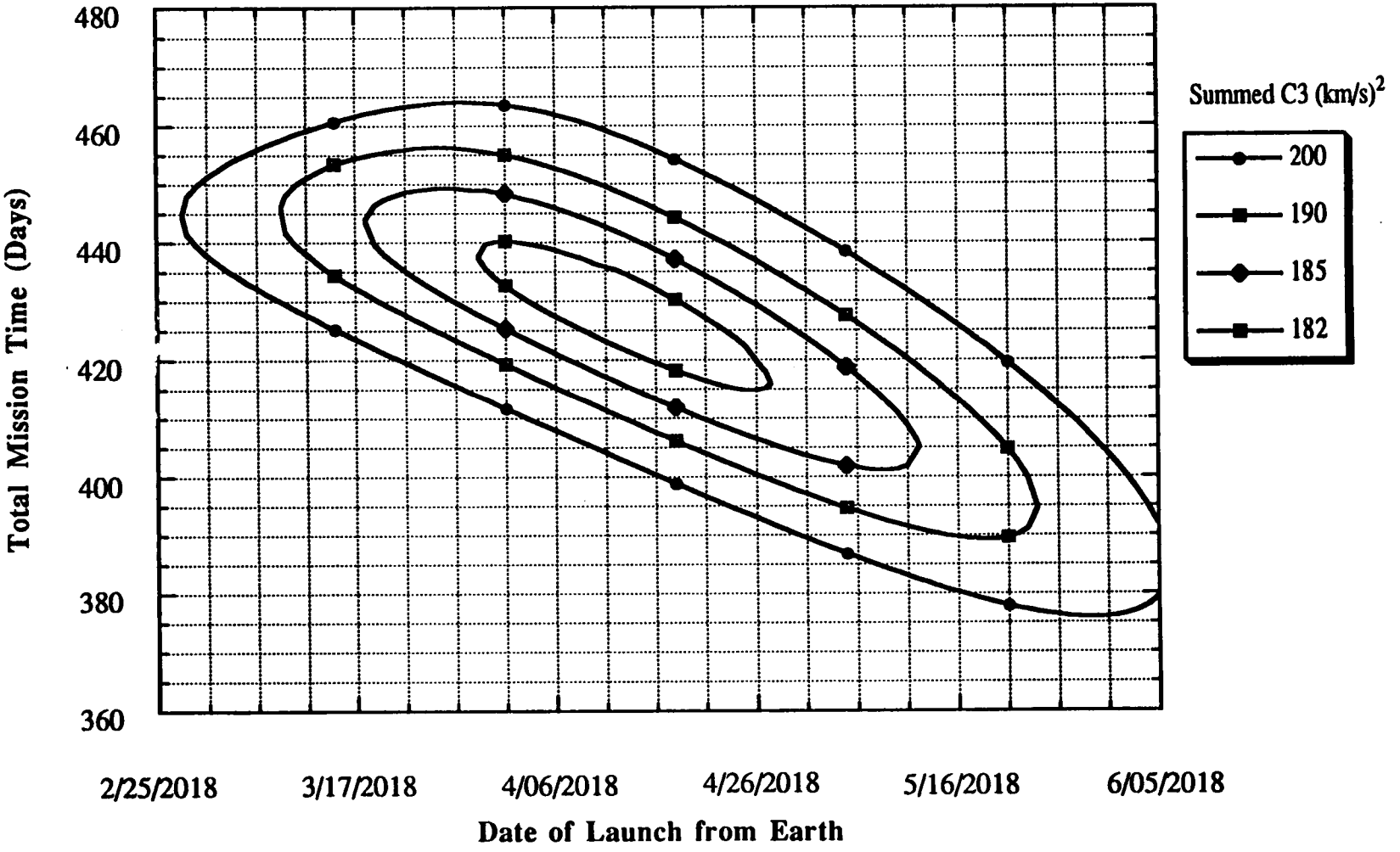

Fig. 2-6: Roundtrip Mars Mission Summed C3 for Impulsive Thrust Missions Near Optimum

Table 2-1: Optimal Impulsive Thrust Mission Opportunity

QUANTITY	OUTBOUND LEG	RETURN LEG
LAUNCH		
Departure Planet	earth (500 km Parking Orbit)	Mars (6000 km Parking Orbit)
Launch Date	April 13, 2018	October 17, 2018
Distance from Sun	1.002 AU	1.388 AU
Celestial Longitude	202.540°	355.425°
Celestial Latitude	0.000°	-1.5017°
Launch True Anomaly	349.457°	194.941°
Heliocentric Velocity	32.789 km/s	21.054 km/s
Flight Path Angle	-1.876°	-7.062°
C3 Required at Launch	13.620 $(km/s)^2$	41.468 $(km/s)^2$
ΔV Required at Launch	3.7682 km/s	4.9726 km/s
ORBITAL ELEMENTS		
a	1.277	1.062
e	0.2172	0.3282
p	1.2165	0.9478
h	4.91452×10^9 km^2/s	4.3379×10^9 km^2/s
n	0.6832 °/day	0.9004 °/day
Ω	22.5401°	262.2149
ω	190.5426°	-101.7317
i	2.4572°	1.5041°
RENDEZVOUS		
Arrival Planet	Mars (6000 km Parking Orbit)	earth (500 km Parking Orbit
Time of Flight	157 days	240 days
Arrival Date	September 17, 2018	June 14, 2019
Distance from Sun	1.381 AU	1.016 AU
Celestial Longitude	336.433°	262.215°
Celestial Latitude	-1.7713°	0.0000°
Arrival True Anomaly	123.324°	101.7317°
Heliocentric Velocity	24.282 km/s	30.198 km/s
Flight Path Angle	11.646°	18.997°
C3 Required to Rendezvous	31.665 $(km/s)^2$	94.317 $(km/s)^2$
ΔV Required to Rendezvous	4.2472 km/s	6.8864 km/s

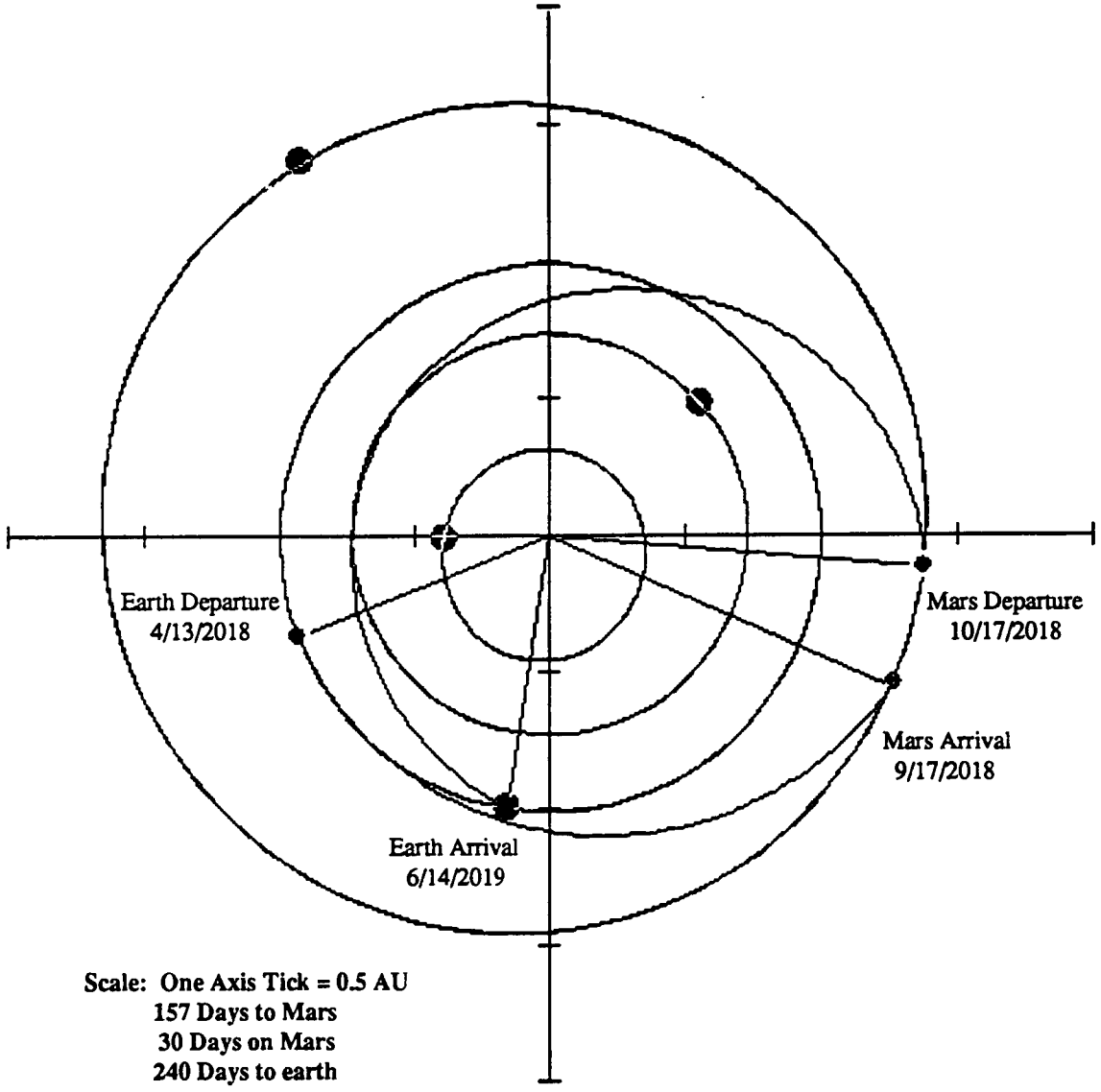

Fig. 2-7: Ecliptic Projection of Optimal Impulsive Thrust Option
(Planets Shown in Final Positions)

contains is assumed 1% unusable. As the propellant in each tank in the second stage is consumed, these tanks are discarded. This action removes the need to expend energy to accelerate inert material in successive burns.

The first stage provides a 100 m/s ΔV for earth orbit corrections, docking etc., the TMI, and is then jettisoned. The second stage provides for a 100 m/s outbound midcourse correction capability, the Mars rendezvous maneuver, a 100 m/s ΔV for Martian orbit operations, the TEI, a return trip 100 m/s midcourse correction capability, the earth rendezvous maneuver, and a 100 m/s ΔV for final earth orbit operations capability. The ΔV required for each of these maneuvers is summarized in Table 4-2. The Isp for each of these maneuvers is assumed to be 470 [2].

2.5 Mission Mass Statement

In order to determine the required mass for the chemical rocket spacecraft, the following factors must be considered: payload mass, bus vehicle mass, and propulsion system mass. The payload mass is set at 100 MT as stated in chapter 1. The mass of the bus vehicle, including active refrigeration system, is calculated below using scaling equations taken from Frisbee [2]. The propellant mass is determined from the energy requirements of placing the vehicle and payload into the proper trajectories.

The mass scaling equation for each of the O_2/H_2 stages is [2]:

$$M_{Dry} = 1000.0 + 0.115(M_P) \qquad \text{[all masses in kg]} \qquad (2\text{-}1)$$

M_{Dry} is the dry mass of the O_2/H_2 stage and M_P is the propellant mass. The value of 1000 kg is the fixed mass of the hardware (engines, etc.) and the value of 0.115 is a tankage factor for those components that scale with the propellant mass (tanks, insulation, structure, etc.). To M_{Dry} above must be added the mass of the active refrigeration

system. The scaling equation for a 3-stage (thermoelectric cooler, O_2 sorption cooler, H_2 sorption cooler) sorption refrigerator for liquid hydrogen temperatures (20 K) is [2]:

$$M_{Frig}[20K] = 45.9 + 21.1(W_{Cool}) \qquad \text{[all masses in kg]} \tag{2-2}$$

where M_{Frig} is the mass of the sorption refrigerator and W_{Cool} is the total cooling load in Watts. The scaling equation for the total cooling load for a 4-tank O_2/H_2 stage is [2]:

$$W_{Cool} = 0.04252(M_p)^{2/3} \qquad \text{[all masses in kg]} \tag{2-3}$$

Combining equations (2-1), (2-2), and (2-3), the complete scaling equation for the bus vehicle is [2]:

$$\begin{aligned} M_{Dry} &= 1045.9 + 0.115(M_P) + 0.8972(M_P)^{2/3} \quad \text{[all masses in kg]} \\ &= 1.0459 + 0.115(M_P) + 0.8972(M_P)^{2/3} \quad \text{[all masses in MT]} \end{aligned} \tag{2-4}$$

The equation relating the mass fraction of the vehicle at each propulsive burn is found using the following formula:

$$\sigma = \frac{M_1}{M_2} = e^{(\Delta V/gIsp)} \tag{2-5}$$

Here, σ is the mass fraction, M_1 is the mass of the vehicle before the burn, M_2 is the mass of the vehicle after the burn, ΔV is the velocity change required, and g is 9.81 m/s^2. As M_1 for each burn contains the fuel and tankage mass for all successive burns, it is

necessary to calculate the mass ratio for each propulsive maneuver in reverse chronological order.

The wet mass for each stage must be solved in an iterative manner as M1 in (2-5) depends upon the tankage factor in (2-4) which itself is unknown until (2-5) is solved. One method of solving this problem is to make an initial guess for tankage factor mass and thus allow an initial fuel requirement approximation to be calculated. This fuel requirement, adjusted for 1% unusable fuel, in turn requires a certain mass of tankage and refrigerant capacity as shown in (2-4). The original tankage factor "guess" is adjusted repeatedly until it agrees with the resulting tankage factor requirement. This procedure was performed for both stages of the vehicle and the results are summarized in Table 2-2.

2.6 Discussion of Chemical Mission Results

Chemical propulsion is a relatively "mature" technology that allows manned Mars exploration to be carried out with an acceptable mission duration and efforts could begin on such a mission almost immediately. However, as seen in Table 2-2, the IMLEO (and thus expense) of the chemical mission option is prohibitive. This mission requires the lofting of 19, 036.05 MT of equipment and propellant into LEO and 95% of this IMLEO is non recoverable.

Different missions profiles would have to be employed or new technologies would have to be developed in order to make chemical rockets a viable option. Lower energy orbits or Venus swingby maneuvers could be used to reduce the fuel costs, and thus IMLEO. Both of these options would significantly increase interplanetary flight time, however, and thus as stated in Chapter I would complicate health risks and reliability concerns. Another possibility would be to develop aerobraking technology alongside the rocket technology advances that would be needed. Future trade studies

Table 2-2: Chemical Mission Mass Statement

Element		Mass (Metric Tons)	ΔV (km/s)
Payload	Outbound	100.00	
	Return	50.00	
Stage 1:	M_{Dry}	1, 249.47	
	M_P earth orbit operations	408.42	0.1
	M_P earth departure	10, 401.03	3.7682
	Stage1/Stage2 Adaptor (2.5% (Stage2+Payload))	170.17	
	Total Stage 1	12,229.09	
Stage 2:	M_{Dry}	831.06	
tank 1:	M_P Midcourse Correction	144.97	0.1
tank 1:	M_P Mars Rendezvous	3, 980.05	4.2472
tank 1:	M_P Orbit Operations	56.47	0.1
tank 2:	M_P Mars Departure (Less 50 MT Payload)	1, 302.95	4.9726
tank 2:	M_P Midcourse Correction	14.41	0.1
tank 3:	M_P Earth Rendezvous	372.24	6.8864
tank 3:	M_P Final Orbit Operations	2.31	0.1
	Stage2/Payload Adapter (2.5% Payload mass)	2.50	
	Total Stage 2	6, 706.96	
Total Stage1+Stage2		18, 936.05	
Total IMLEO		19,036.05	

should consider each of these techniques.

As these techniques could also be used for the other propulsion options in this comparison (with the possible exception of aerobraking for the solar sail) they will not be considered further here. The mission described above will be used as the baseline for comparison with NTP and the solar sail.

III. Nuclear Thermal Propulsion

3.1 Mission Design

The principals behind designing a mission for NTP are the same as those for designing the chemical option. The only differences are the size and Isp of the spacecraft. Thus, the same mission opportunity presented in Chapter 2 will also be used here. The reader should refer to Fig. 2-1 through Fig. 2-6 for mission selection, Fig. 2-7 for a mission schematic, and Table 2-1 for trajectory data and propulsive requirements.

3.2 Mission Mass Schedule

As described in Chapter 1, the vehicle assumed utilizes an advanced NERVA class engine. It is a two stage vehicle with three sets of tanks in the second stage. A sorption compressor is used to prevent boiloff of the cryogenic propellant (liquid hydrogen). The fuel each tank contains is assumed 1% unusable. As the propellant in each tank in the second stage is consumed, these tanks are discarded. This action removes the need to expend energy to accelerate inert material in successive burns.

The first stage provides a 100 m/s ΔV for earth orbit corrections, docking etc. This stage then provides trans Mars injection maneuver and is subsequently jettisoned. The second stage provides for a 100 m/s outbound midcourse correction capability, the Mars rendezvous maneuver, a 100 m/s ΔV for Martian orbit operations, the trans earth injection maneuver, a return trip 100 m/s midcourse correction capability, the earth rendezvous maneuver, and a 100 m/s ΔV for final earth orbit operations capability. The ΔV required for each of these maneuvers is summarized in Table 4-1. The Isp for each of these maneuvers is assumed to be 925 s [3].

3.3 Mission Mass Statement

In order to determine the required mass for the NTP spacecraft, the following factors must be considered: payload mass, bus vehicle mass, and propulsion system mass. The payload mass is set at 100 MT as stated in chapter 1. The mass of the bus vehicle, including active refrigeration system, is calculated below using scaling equations taken from Frisbee [3]. The propellant mass is determined from the energy requirements of placing the vehicle and payload into the proper trajectories.

The mass scaling equation for each of the NTP stages is [3]:

$$M_{Dry} = M_{Engine} + 1573 + 0.2584(M_P) \qquad \text{[all masses in kg]} \qquad (3\text{-}1)$$

M_{Dry} is the dry mass of the NTP stage and M_P is the propellant mass. The value of 1573 kg is the fixed mass of the hardware and the value of 0.2584 is a tankage factor for those components that scale with the liquid hydrogen propellant mass (tanks, insulation, structure, etc.). M_{Engine} is the mass of the advanced NERVA engine, which was set to 10.4 MT [3]. To M_{Dry} above must be added the mass of the active refrigeration system as was done with the baseline chemical engine. The scaling equation for a 1 tank sorption refrigerator for liquid hydrogen temperatures (20 K) is [3]:

$$M_{Frig}[20K] = 45.9 + 1.736(M_p)^{2/3} \qquad \text{[all masses in kg]} \qquad (3\text{-}2)$$

where M_{Frig} is the mass of the sorption refrigerator.

As in Chapter 2, the equation relating the mass fraction of the vehicle at each propulsive burn is found using the following formula:

$$\sigma = \frac{M_1}{M_2} = e^{(\Delta V/gIsp)} \qquad (3\text{-}3)$$

Here, σ is the mass fraction, M_1 is the mass of the vehicle before the burn, M_2 is the mass of the vehicle after the burn, ΔV is the velocity change required, and g is 9.81 m/s^2. As M_1 for each burn contains the fuel and tankage mass for all successive burns, it is necessary to calculate the mass ratio for each propulsive maneuver in reverse chronological order.

The wet mass for each stage must be solved in an iterative manner as M_1 in (3-3) depends upon the tankage factors in (3-1) and (3-2) which themselves are unknown until (3-3) is solved. The same method presented earlier in Chapter 2 can be used here to solve this problem. An initial guess for tankage factors is made allowing the fuel requirement to be calculated. This fuel requirement, adjusted for 1% unusable fuel, in turn requires a certain mass of tankage and refrigerant capacity as shown in (3-1) and (3-2). The tankage factor "guess" is adjusted repeatedly until it agrees with the resulting tankage factor requirement. This procedure was performed for both stages of the vehicle and the results are summarized in Table 4-1.

3.4 Comparison between NTP and Chemical Propulsion Results

As shown in Table 4-1, NTP provides significant IMLEO savings over the similar chemically powered system while performing the same mission in the same amount of time. A total of 2, 013.77 MT must be lofted into LEO of which 83% is non recoverable. This figure represents 10.58% of the IMLEO for the chemical case. Thus it is seen that NTP does much to bring the cost of manned Mars exploration within reach

Table 4-1: NTP Mission Mass Statement

Element		Mass (Metric Tons)	ΔV (km/s)
Payload	Outbound	100.00	
	Return	50.00	
Stage 1:	M_{Dry}	156.94	
	M_P earth orbit operations	22.07	0.1
	M_P earth departure	676.85	3.78
	Stage1/Stage2 Adapter (2.5% (Stage2+Payload))	28.24	
	Total Stage 1	884.10	
Stage 2:	M_{Dry}	283.68	
tank 1:	M_P Midcourse Correction	11.83	0.1
tank 1:	M_P Mars Rendezvous	399.14	4.28
tank 1:	M_P Orbit Operations	7.33	0.1
tank 2:	M_P Mars Departure (Less 50 MT Payload)	205.97	4.96
tank 2:	M_P Midcourse Correction	3.09	0.1
tank 3:	M_P Earth Rendezvous	115.02	6.86
tank 3:	M_P Final Orbit Operations	1.11	0.1
	Stage2/Payload Adapter (2.5% Payload mass)	2.50	
	Total Stage 2	1,029.67	
Total Stage1+Stage2		1,913.77	
Total IMLEO		2,013.77	

As with the chemical case, certain steps could be taken to further reduce the IMLEO. Lower energy orbits or Venus swingby maneuvers could be used to reduce the fuel costs, and thus IMLEO. Both of these options, however, would significantly increase overall mission time and thus, as stated in Chapter I, would complicate health risks and reliability concerns. Another possibility would be to develop aerobraking technology alongside NTP technology. Trade studies should be performed to consider the benefits and risks presented by these options

IV. Solar Sail Propulsion

4.1 The Equations of Motion for a Solar Sail

The motion of a solar sail spacecraft is governed by inverse-squared gravity fields of surrounding celestial bodies and the force imparted upon the sail by the reflection of photons. Thus, equations 1.1-1.3 can be used for the solar sail with the quantities F_r, F_Φ, and F_Ψ defined by the strength of incident light falling upon the sail and by the orientation of the sail's surface. The value of m becomes constant since no propellant is expended. The acceleration of the spacecraft caused by the force of the impinging photons is determined in the following paragraphs. The following development of the equations of motion and the notation used generally follows that of Grodsovskii (11) except that the equations have been extended to three dimensions.

The acceleration due to solar pressure can be broken into three component vectors (See Fig. 4-1):

$$a_r = a\cos(\theta) \tag{4.1}$$

$$a_\Phi = a\sin(\theta)\sin(\beta) \tag{4.2}$$

$$a_\Psi = a\sin(\theta)\cos(\beta) \tag{4.3}$$

In these equations, The thrust vector, **a**, is always considered to be oriented normally to the sail surface. The control variables, θ and β, are the thrust vector cone and clock angles, respectively.

The equation of motion for a solar sail spacecraft is:

$$\mathbf{a} = \frac{P_A S_o R_f}{S_o \rho \delta + G_\pi / g}\left(\frac{R_o}{r}\right)^2 \left(\hat{n} \bullet \hat{i}\right)^2 \hat{n} + \mathbf{R} \tag{4.4}$$

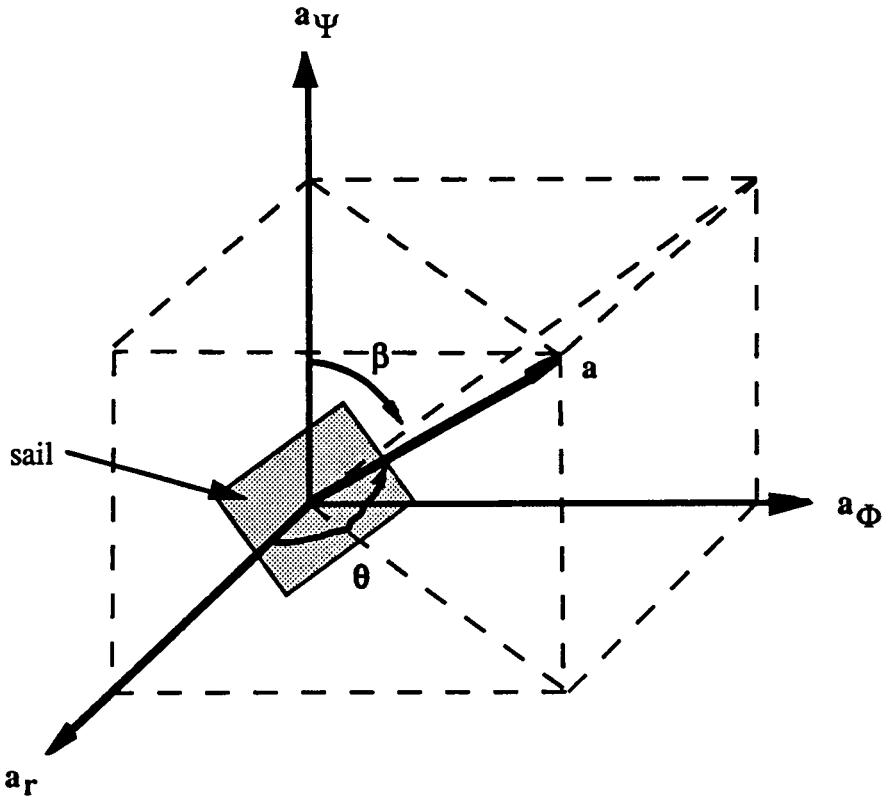

Fig. 4-1: Definition of Sail Thrust Vector Cone and Clock Angles

Here, $\hat{n}$ is a unit normal to the shaded side of the sail surface, $\hat{i}$ is a unit vector in the direction of the incident sunlight. P_A is solar pressure per unit sail area on a sail one AU from the sun and set perpendicular to incident light, S_o is the sail surface area, R_f is the reflectivity of the sail material, $G\pi$ is the mass of the payload, ρ is the sail material density (including a factor for support structures), and δ is the sail material thickness. R_o is the reference distance of one AU and r is the actual distance of the spacecraft from the sun. **R** represents the gravity field of the sun. The factor of R_f does not appear in Grodzovskii [11] as he assumes a reflectivity of unity.

Here, it is convenient to introduce the notion of the characteristic acceleration, a_o. The characteristic acceleration is defined as that acceleration communicated to a sail set

perpendicular to incident light at a distance from the sun of one AU.

$$a_o = \frac{P_A S_o R_f}{S_o \rho \delta + G_\pi / g} \leq \frac{P_A}{\rho \delta} \tag{4.5}$$

Thus, (4.4) can be rewritten as (4.6).

$$\mathbf{a} = a_o \left(\frac{R_o}{r} \right)^2 \left(\hat{n} \bullet \hat{i} \right)^2 \hat{n} + \mathbf{R} \tag{4.6}$$

Utilizing (4.6) the equations of thrust for the solar sail as given in equations (4.1)-(4.3) can now be rewritten as equations (4.7)-(4.9).

$$a_r = a_o \left(\frac{R_o}{r} \right)^2 \cos^3(\theta) \tag{4.7}$$

$$a_\Phi = a_o \left(\frac{R_o}{r} \right)^2 \cos^2(\theta) \sin(\theta) \sin(\beta) \tag{4.8}$$

$$a_\Psi = a_o \left(\frac{R_o}{r} \right)^2 \cos^2(\theta) \sin(\theta) \cos(\beta) \tag{4.9}$$

The above three equations, (4.7)-(4.9) can be combined with equations (1.1)-(1.3) in order to obtain the complete equations of motion for a solar sail. Before doing this, however, it is desired to nondimensionalise all variables as done by Lebedev[12]. This nondimensionalization will make the equations of motion easier to work with during trajectory optimization. Conversion to nondimensionalized variables is achieved with the following formulae:

$$r = \frac{r^*}{R_o} \tag{4.10}$$

$$V_i = \frac{V_i^*}{\sqrt{\frac{\mu}{R_o}}} \tag{4.11}$$

$$a_i = \frac{a_i^*}{\mu} \tag{4.12}$$

$$t = \frac{t^*}{\sqrt{\frac{R_o}{\mu}}} \tag{4.13}$$

Here the non-starred variables represent the nondimensionalized variables, the starred variables represent the dimensional variables, R_o represents the distance of one AU, and μ is the gravitational parameter of the attracting body. Now equations (1.1)-(1.3) and (4.7)-(4.9) can be combined into the complete, nondimensionalized equations of motion for a solar sail:

$$\frac{dr}{dt} = V_r \tag{4.14}$$

$$\frac{d\Phi}{dt} = \frac{V_\Phi}{r\cos(\Psi)} \tag{4.15}$$

$$\frac{d\Psi}{dt} = \frac{V_\Psi}{r} \tag{4.16}$$

$$a_r = \frac{V_\Phi{}^2 + V_\Psi{}^2}{r} - \frac{1}{r^2} + a_o\left(\frac{1}{r}\right)^2 \cos^3(\theta) \tag{4.17}$$

$$a_\Phi = \frac{V_\Phi V_\Psi \tan(\Psi)}{r} - \frac{V_\Phi V_r}{r} + a_o\left(\frac{1}{r}\right)^2 \cos^2(\theta)\sin(\theta)\sin(\beta) \tag{4.18}$$

$$a_\Psi = -\frac{V_r V_\Psi}{r} - \frac{V_\Phi^2 \tan(\Psi)}{r} + a_o\left(\frac{1}{r}\right)^2 \cos^2(\theta)\sin(\theta)\cos(\beta) \tag{4.19}$$

As characteristic acceleration is a constant for any given sail, the only control functions in the equations of motion are the sail's thrust vector cone and clock angles, θ

and β. These will be the variables which will later be used to ensure minimum flight time for a given sail to a given target.

4.2 The Pontryagin Maximum Principle

The problem of minimum flight time for a solar sail spacecraft is much less widely understood and less well documented than is Lambert's problem. The program used to solve the problem in this study represents original work by the author, Brian K. Funk, and H. Dan Thomas. This procedure was derived from the basic mathematical principles laid out by Pontryagin[13]. Such an application is not found in standard texts, and therefore deserves much more attention here than was given to Lambert's problem in Chapters 2 and 3.

In order to calculate the local minimum flight time for a solar sail interplanetary trajectory, use is made of the Pontryagin Maximum Principle. This principle is a mathematical tool for finding the optimal control law for any controlled process during which it is desired to minimize a given quantity (flight time, fuel expenditure, etc.). This principle requires the development of a set of auxiliary variables in addition to the equations of motion as well as the introduction of a quantity known as the Hamiltonian of the system. The development that follows springs from Pontryagin [13], and his notation has been maintained wherever possible.

Consider a system of first order differential equations obtained from the equations of motion for the solar sail:

$$\frac{dx^i}{dt} = f^i(\mathbf{x},\mathbf{u}), \quad i = 0,1,2,\ldots,n \tag{4.20}$$

In this case, x^i represents state variables and **u** represents a set of control variables. In the case of the solar sail, the set of control variables consists of the sail thrust vector cone and clock angels. Consider also a system of equations in auxiliary variables, P_i:

$$\frac{dP_i}{dt} = -\sum_{j=0}^{n} \frac{\partial f^j(\mathbf{x},\mathbf{u})}{\partial x^i} P_j, \quad i = 0,1,...,n \tag{4.21}$$

If some acceptable set of control variables has been chosen as a function of time, $\mathbf{u} = \mathbf{u}(t)$ where $t_0 \leq t \leq t_1$, and there has been generated a trajectory, $\mathbf{x}(t)$, corresponding to this control function with the initial condition $\mathbf{x}(t_0)=\mathbf{x}_0$, then system (4.21) takes the form:

$$\frac{dP_i}{dt} = -\sum_{j=1}^{n} \frac{\partial f^j(\mathbf{x}(t),\mathbf{u}(t))}{\partial x^i} P_i, \quad i = 0,1,2,...,n \tag{4.22}$$

This system of equations is both linear and homogenous. Therefore, for any given set of initial conditions for P_i there is a unique solution for system (4.22) defined throughout the time interval $t_0 \leq t \leq t_1$ on which the control functions $\mathbf{u}(t)$ and their corresponding trajectory $\mathbf{x}(t)$ are defined. Like the solution for system (4.20), the sail trajectory, the solution for system (4.22), the auxiliary variables, consists of continuous functions $P_i(t)$, having continuous derivatives with respect to t at all points except those finite number of points corresponding to points of discontinuity in the sail angle control functions. Every solution of system (4.22) (with given initial conditions) is in fact the solution to system (4.21) corresponding to the chosen sail angle control functions, $\mathbf{u}(t)$, and sail trajectory, $\mathbf{x}(t)$.

The systems of equations (4.20) and (4.21) are now combined into a function known as the Hamiltonian, H. This Hamiltonian is a function of the state variables, the auxiliary variables, and the chosen sail angle control functions.

$$H(\mathbf{P},\mathbf{x},\mathbf{u}) = H(\mathbf{P},\mathbf{f}(\mathbf{x},\mathbf{u})) = \sum_{i=0}^{n} P_i f^i(\mathbf{x},\mathbf{u}) \tag{4.23}$$

With the aid of the Hamiltonian function the systems of equations given in (4.20) and (4.21) can be combined into the following Hamiltonian system:

$$\frac{dx^i}{dt} = \frac{\partial H}{\partial P_i}, \quad i = 0,1,...,n \tag{4.24}$$

$$\frac{dP_i}{dt} = -\frac{\partial H}{\partial x^i}, \quad i = 0,1,...,n \tag{4.25}$$

Using (4.23), the Hamiltonian for the solar sail traveling in three dimensions is expressed as:

$$H = P_r\left(\frac{dr}{dt}\right) + P_\phi\left(\frac{d\phi}{dt}\right) + P_\Psi\left(\frac{d\Psi}{dt}\right) + P_{V_r}\left(\frac{dV_r}{dt}\right) + P_{V_\phi}\left(\frac{dV_\phi}{dt}\right) + P_{V_\Psi}\left(\frac{dV_\Psi}{dt}\right) \tag{4.26}$$

The maximum principle states that for the control functions **u**(t) to ensure the transfer of the solar sail from point one (coincident with the launch planet) to point two (coincident with the destination planet) with a minimum time, then **u**(t) must provide an absolute maximum for the Hamiltonian [13].

4.3 The Hamiltonian for the Solar Sail

Referring back to the equations of motion for the solar sail, H can be rewritten as:

$$H = P_r(Vr) + P_\Phi\left(\frac{V_\Phi}{r\cos(\Psi)}\right) + P_\Psi\left(\frac{V_\Psi}{r}\right) + P_{V_r}\left(\frac{V_\Phi^2 + V_\Psi^2}{r} - \frac{1}{r^2} + a_o\left(\frac{1}{r}\right)^2\cos^3(\theta)\right)$$
$$+ P_{V_\Phi}\left(\frac{V_\Phi V_\Psi \tan(\Psi)}{r} - \frac{V_\Phi V_r}{r} + a_o\left(\frac{1}{r}\right)^2\cos^2(\theta)\sin(\theta)\sin(\beta)\right)$$
$$+P_{V_\Psi}\left(-\frac{V_r V_\Psi}{r} - \frac{V_\Phi^2\tan(\Psi)}{r} + a_o\left(\frac{1}{r}\right)^2\cos^2(\theta)\sin(\theta)\cos(\beta)\right) \quad (4.27)$$

In accordance with (4.24) and (4.25), the following system of equations thus arises for the auxiliary variables:

$$\frac{dP_r}{dt} = -\frac{\partial H}{\partial r} = P_\Phi\left(\frac{V_\Phi}{r^2\cos(\Psi)}\right) + P_\Psi\left(\frac{V_\Psi}{r^2}\right) + P_{V_r}\left(\frac{V_\Phi^2 + V_\Psi^2}{r^3} - a_o\left(\frac{2}{r^3}\right)\cos^3(\theta)\right)$$
$$+P_{V_\Phi}\left(\frac{V_\Phi V_\Psi\tan(\Psi)}{r^2} - \frac{V_\Phi V_r}{r^2} + a_o\left(\frac{2}{r^3}\right)\cos^2(\theta)\sin(\theta)\sin(\beta)\right)$$
$$+P_{V_\Psi}\left(-\frac{V_r V_\Psi}{r^2} - \frac{V_\Phi^2\tan(\Psi)}{r^2} + a_o\left(\frac{2}{r^3}\right)\cos^2(\theta)\sin(\theta)\cos(\beta)\right) \quad (4.28)$$

$$\frac{dP_\Phi}{dt} = -\frac{\partial H}{\partial \Phi} = 0 \Rightarrow P_\Phi = \text{constant} \quad (4.29)$$

$$\frac{dP_\Psi}{dt} = -\frac{\partial H}{\partial \Psi} = -P_\Phi\left(\frac{V_\Phi\sin(\Psi)}{r\cos^2(\Psi)}\right) - P_{V_\Phi}\left(\frac{V_\Phi V_\Psi}{r\cos^2(\Psi)}\right) + P_{V_\Psi}\left(\frac{V_\Phi^2}{r\cos^2(\Psi)}\right) \quad (4.30)$$

$$\frac{dP_{V_r}}{dt} = -\frac{\partial H}{\partial V_r} = -P_r + P_{V_\Phi}\left(\frac{V_\Phi}{r}\right) + P_{V_\Psi}\left(\frac{V_\Psi}{r}\right) \quad (4.31)$$

$$\frac{dP_{V_\Phi}}{dt} = -\frac{\partial H}{\partial V_\Phi} = -P_\Phi\left(\frac{1}{r\cos(\Psi)}\right) - P_{V_r}\left(\frac{2V_\Phi}{r}\right) - P_{V_\Phi}\left(\frac{V_\Psi\tan(\Psi)}{r} - \frac{V_r}{r}\right)$$
$$+P_{V_\Psi}\left(\frac{2V_\Phi\tan(\Psi)}{r}\right) \quad (4.32)$$

$$\frac{dP_{V_\Psi}}{dt} = -\frac{\partial H}{\partial V_\Psi} = -P_\Psi\left(\frac{1}{r}\right) - P_{V_r}\left(\frac{2V_\Psi}{r}\right) - P_{V_\Phi}\left(\frac{V_\Phi\tan(\Psi)}{r}\right) + P_{V_\Psi}\left(\frac{V_r}{r}\right) \quad (4.33)$$

For the control functions, **u**(t), to give an optimal time transfer, then they must provide a maximum for the Hamiltonian as mentioned above. For this to be the case:

$$\frac{\partial H}{\partial \theta} = 0 \tag{4.34}$$

$$\frac{\partial H}{\partial \beta} = 0 \tag{4.35}$$

Taking the derivative of the Hamiltonian with respect to θ and β in turn, it is thus found that the optimum sail-orientation program is found from:

$$\tan(\beta) = \frac{P_{V_\Phi}}{P_{V_\Psi}} \tag{4.36}$$

$$\tan(\theta) = -\frac{3P_{V_r} + \sqrt{9P_{V_r}^2 + 8\left(P_{V_\Phi}\sin(\beta) + P_{V_\Psi}\cos(\beta)\right)^2}}{4P_{V_\Phi}\sin(\beta) + 4P_{V_\Psi}\cos(\beta)} \tag{4.37}$$

If (4.36) and (4.37) are substituted into the six equations of motion for the solar sail and the six equations for the auxiliary variables given above, then a system of eleven first order differential equations results. (Only eleven because the equation for P_Φ is constant and zero.)

4.4 Optimal Trajectories: The Two Point Boundary Value Problem

Now the two point boundary value problem can be solved to determine the time optimal trajectory from the launch planet to the rendezvous planet for a given payload. In this analysis, the assumption of zero hyperbolic excess speeds is made. Therefore, the initial position and velocity components of the spacecraft are set equal to those of the

launch planet. For rendezvous, the required conditions are that the final position and velocity components of the spacecraft exactly match those of the target. These are the two boundary points.

In order to have a properly specified problem, the six rendezvous conditions (position and velocity) and the transversality condition, to be discussed later, require seven unknown initial conditions. These quantities are the initial values of P_r P_{V_r}, P_{V_Φ}, P_Ψ, and P_{V_Ψ}, the date of launch, and the time of flight. Together, these values will be known collectively as **x**.

The eleven first-order differential equations are placed in two iterative procedures which accept a first guess for **x**, utilize a fourth-order Runge-Kutta[16] to integrate the equations of motion and generate the trajectory for these values, and then alter the values to obtain a improved result for the next iteration. The first of these procedures is the method of steepest descent, and the second is Newton's method [16]. In the algorithm developed for this work inputs for these iterative procedures are:

$$
\begin{aligned}
x_1 &= \text{Date of Launch} \\
x_2 &= \text{Flight Time} \\
x_3 &= P_r \\
x_4 &= P_{V_r} \\
x_5 &= P_{V_\Phi} \\
x_6 &= P_\Psi \\
x_7 &= P_{V_\Psi}
\end{aligned}
\tag{4.38}
$$

The outputs of these procedures for each iteration are:

$$y_1 = r_s - r_R$$

$$y_2 = \Phi_s - \Phi_R$$

$$y_3 = \Psi_s - \Psi_R$$

$$y_4 = V_{r_s} - V_{r_R}$$

$$y_5 = V_{\Phi_s} - V_{\Phi_R}$$

$$y_6 = V_{\Psi_s} - V_{\Psi_R}$$

$$y_7 = H - 1 \qquad (4.39)$$

Here, the subscript s indicates the value for the spacecraft and the subscript R indicates the value for the rendezvous planet. The objective of both the method of steepest descent and the Newton's method iterative procedures is to drive **y** to zero. When this objective has been met rendezvous is said to have occurred. Both the spacecraft and the destination planet are, in the heliocentric system, mathematically in the same position and traveling with the same velocity.

The convergence of Newton's method is quadratic, while the method of steepest descent converges to a solution only linearly. However, Newton's method is extremely limited in that the initial guesses for **x** must be very near the optimum values for the method to converge. The method of steepest descent, while it converges much more slowly, allows for a much greater variance in the initial guesses for **x**. Thus the method of steepest descent is utilized first in order to provide the initial conditions for Newton's method.

While the first six values for **y** in (4.39) are derived from obvious considerations, the seventh is the aforementioned transversality condition and requires further discussion. In the calculus of variations, traansversality conditions are those end point conditions that must hold due to the physics of the problem being considered. In the case of finding

solar sail time-optimal trajectories, the transversality condition is that the Hamiltonian of the system be equal to a constant [14, 15].

Recall that equations (4.34) and (4.35) require the value of H to be a constant for the optimal trajectory. Exactly which constant is largely immaterial as can be seen by examining (4.26). The choice of differing constant values for H would only require the scaling of the auxiliary variables which serve in (4.26) as Lagrange undetermined multipliers. However, the value of unity is a convenient value and Pontryagin states that optimal motions can always be accomplished in such a way that H=1. (Pontryagin, p. 70) From this reasoning, the equation for y_7 is chosen as given in (4.39).

4.5 Vehicle Sizing

The nature of finding optimal trajectories for a solar sail differs from that of impulsive thrust vehicles in more ways than the mathematical procedures used to determine interplanetary trajectories. The optimal trajectory a given solar sail can achieve depends upon the characteristic acceleration that the sail possesses, and this in turn depends upon the sail reflective area and payload. Thus in the case of the solar sail it is required to size the vehicle before the optimal trajectory can be found. This order is the exact reverse of the path followed in outlining the chemical and nuclear thermal missions.

The required size of the solar sail depends upon the desired a_o of the sail and the thickness of the sail material (which determines the sail mass) and is in turn limited by the need to deploy the sail after the firing of the chemical booster. Due to shadowing, an a_o of 0.6 mm/s^2 represents a lower limit for maneuvering near a planet [2]. Freidman [5] states that an a_o of 1mm/s^2 is desirable for interplanetary travel. Garvey states that a sail thickness of 1 micron is required for deployable sails [17]. This thickness requirement is to ensure that the sail material can survive folding and unfolding. Staehle has considered

sails with a thickness of 2.5 microns [2]. With this information, equation (4.5) allows one to size the sail needed for a given mission.

Using the minimum possible a_o of 0.6 mm/s^2, a sail thickness of 2.5 microns and the desired payload mass of 100 metric tons, the sail is sized by solving (4.5) for sail area as given in (4.40). The value for P_A is 9 Newtons/km^2.

$$S_o = \frac{a_o\left(G_\pi/g\right)}{P_A R_f - a_o \Lambda} \tag{4.40}$$

Here, ρ and δ have been combined into the single variable quantity of areal density, Λ. Areal density is a commonly used variable in solar sail literature. Typical deployable sails, such as those studied by Staehle, have an areal density of 4.8 g/m^2. This results in a required S_o of 10.58 km^2.

MacNeal [18] presents a scheme for deploying sails of this size that even includes a form of artificial gravity for the astronauts. Note, however, that 0.6 is the minimum a_o that can be considered. The primary limiting factor for solar sails in the comparison criteria used in this study is time of flight, and thus it is desired to obtain a higher value for a_o; a value on the order of Freidman's recommended 1.0 mm/s^2. It is also desire to keep the size of this higher performance sail within the realms of construction and deployability. These desires mean that a more clever scheme must be employed than simply hanging one large payload on a single, heavy sail.

First, a more modern technology is assumed than that utilized in the Staehle study. Drexler demonstrates that sail material with a thickness as low as 0.1 microns (Friedman p. 28) are achievable. However, this study assumes the Garvey deployable sail minimum thickness of 1 micron and thus brings the areal density down from 4.8 g/m^2 to 1.92 g/m^2. The second assumption made is that two sails are to be deployed. One sail would thus

carry the crew and MTV while the other, which could rendezvous with Mars long before the crew even leaves earth, would carry the MEV and remaining cargo.

While this second assumption might at first seem an unfair attempt to make the sail option competitive, it does not work against the sail as much as might first seem. As sails require no fuel, two sails will be not breed a redundancy of tankage factor weights into the IMLEO. Nor will there need to be construction of multiple engines. Construction of two sails requires little more than the assemblage of the same amount of material that would be required in order to loft a single sail of their combined area, and in fact this construction of multiple smaller sails may prove easier to perform than that of their single large cousin.

There is also the advantage of a second sail in Mars orbit should anything go wrong with the first. Since no fuel is used, the cargo pallet vehicle could be used to propel the crew habitat module back to earth. Von Braun himself, in his famous Mission to Mars [19], envisioned a fleet of no less than ten ships to perform the first manned Martian exploration. Stating the advantages of multiple vessels in his own words:

> In 1492 Columbus knew far less about the far Atlantic than we do about the far heavens, yet he chose not to sail with a flotilla of less than three ships, and history tends to prove that he might never have returned to Spanish shores with his report of discoveries had he entrusted his fate to a single bottom. So it is with interplanetary exploration: it must be done on the grand scale. (Braun, p.2)

The payload is thus divided into two equal parts of 50 MT per sail. Note that this saddles the sail with the same payload on both manned legs of its trip as was assumed in

Table 4-1 Mission Mass Statement for Deployable, Dual Solar Sail Option

Sail Area, S_o	7.54 km^2 (each)
Areal Density, Λ	1.92 g/m^2
Sail Thickness	1 micron
Reflectivity	0.95
Characteristic Acceleration, a_o	1 mm/s^2
Sail Mass	14.48 MT (each)
Payload Mass	50 MT (each)
Manned Sail Chemical Booster Mass	3.1524 km/s $\Rightarrow$ 161.64 MT (each)
Total IMLEO (Both Sails)	452.24 MT

the impulsive thrust cases' return leg. With these assumptions in mind, the sail sizing calculation is repeated using (4.40) utilizing an a_o of 1mm/s2, a payload mass of 50 MT and a Λ of 1.92 g/m^2. These values result in a required S_o of 7.54 km^2. The sizing data for this sail is given in easy reference form as table 4-1. The calculation of mass for the manned sail chemical booster will be shown later but the value is included here for completeness.

4.6 Selection of the Optimal Trajectory and Calculation of Booster Mass

With the characteristic acceleration of the solar sail determined, computational procedures can be used to determine the optimal flight opportunity of the manned portion of the mission. An analysis of the 2001-2020 period was examined for a solar sail with an a_o of 1 mm/s^2. The code (see Appendix B) drives the date of launch and flight time from the original user input to those corresponding to the local minimum flight time for the current synodic period between the earth and Mars and determines the needed time history for the control variables.

Due to the eccentricities in the orbits of the earth and Mars and the inclination of the Martian orbit, each local minimum is different from the others. The local minimums for the outbound and return flights within the period of 2001 to 2020 are presented in table 4-2. The number of days between arrival at Mars on an optimal outbound trajectory and the occurrence of an optimal return trajectory is called the "stay" time. As can be seen by examining Table 4-2, the solar sail mission option using optimal outbound and return trajectories varies in duration from 923 to 950 days for the 2000-2020 period with stay times ranging from 63 to 207 days. It would be possible for the solar sail to depart Mars after a loiter of shorter or longer duration than the stay time specified in Table 4-2, but it would have to use a non time-optimal return trajectory. (Or possibly a non optimal

Table 4-2: Solar Sail Mars Mission Profiles with Time-Optimal Interplanetary Trajectories (2000 - 2020)

Outbound Launch	Flight Time (Days)	Stay (Days)	Return Launch	Flight Time (Days)	Total Mission (Days)
4/13/2001	361	192	10/18/2002	371	924
6/29/2003	372	188	1/09/2005	365	925
8/20/2005	433	109	2/13/2007	393	935
9/27/2007	456	69	3/05/2009	420	945
10/31/2009	449	59	3/23/2011	442	950
12/07/2011	430	64	4/14/2013	454	948
1/18/2014	403	93	5/30/2015	444	940
3/15/2016	372	161	8/31/2017	395	928
6/01/2018	357	207	12/17/2019	359	923
8/02/2020	412	136	2/02/2022	383	931

outbound trajectory.) A detailed examination of the tradeoffs between non-optimal trajectories and stay time has not been carried out in this study.While it is desirable to select a mission opportunity with relatively short total mission time, longer stay times are attractive as stay time must encompass the time required for the sail to spiral in and out of its Martian parking orbit. Sauer has reported times of around 100 days for achievement of a 6000 km parking orbit using a sail with an a_0 of 1 mm/s^2 (2). It would of course be possible for the crew to utilize a different parking orbit or even to depart for the Martian surface before the sail achieved its final orbit.

In accordance with the above criteria, the 2018 mission opportunity was selected as the solar sail mission example for comparison with competing propulsion schemes. Schematics of this mission are presented in Figs. 4-2 and 4-3. Time histories of the control angles and auxiliary variables needed to obtain the required trajectories are presented as Figs. 4-4 through 4-7. Time histories of the state variables for the sail are given in Figs. 4-8 through 4-15.

The solar sail is not capable of direct ascension to Mars from LEO and, in fact, is incapable of operation below an earth altitude of 2000 km due to atmospheric drag and shading (2). This operational restriction requires the use of either an orbital ferry vehicle or a booster stage. For the purposes of this study, a disposable chemical booster with Isp of 470 s was assumed. The booster is assumed to break the vehicle out of low earth orbit and place it into a parabolic escape trajectory. At this point the booster is jettisoned, the sail is deployed, and the vehicle is assumed to meet the initial boundary condition of matching the position and velocity of the launch planet. Orbits around Mars are high enough to allow direct sail operations (spiraling as described above). The mass of the booster was determined using the same procedure as outlined in Chapters 2 and 3 and is given in Table 4-1.

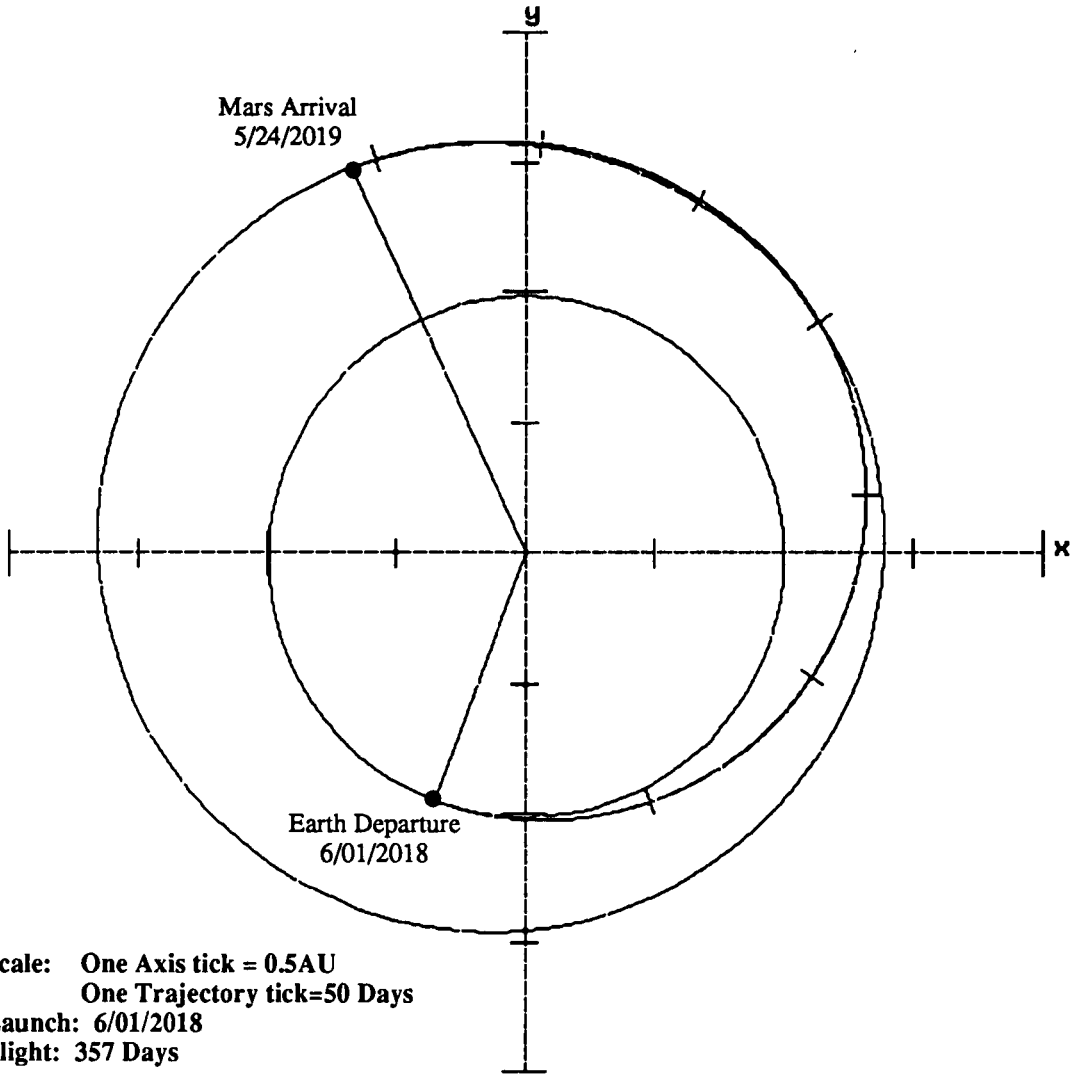

Fig. 4-2: Ecliptic Projection of Earth to Mars Leg of Optimal Solar Sail Trajectory

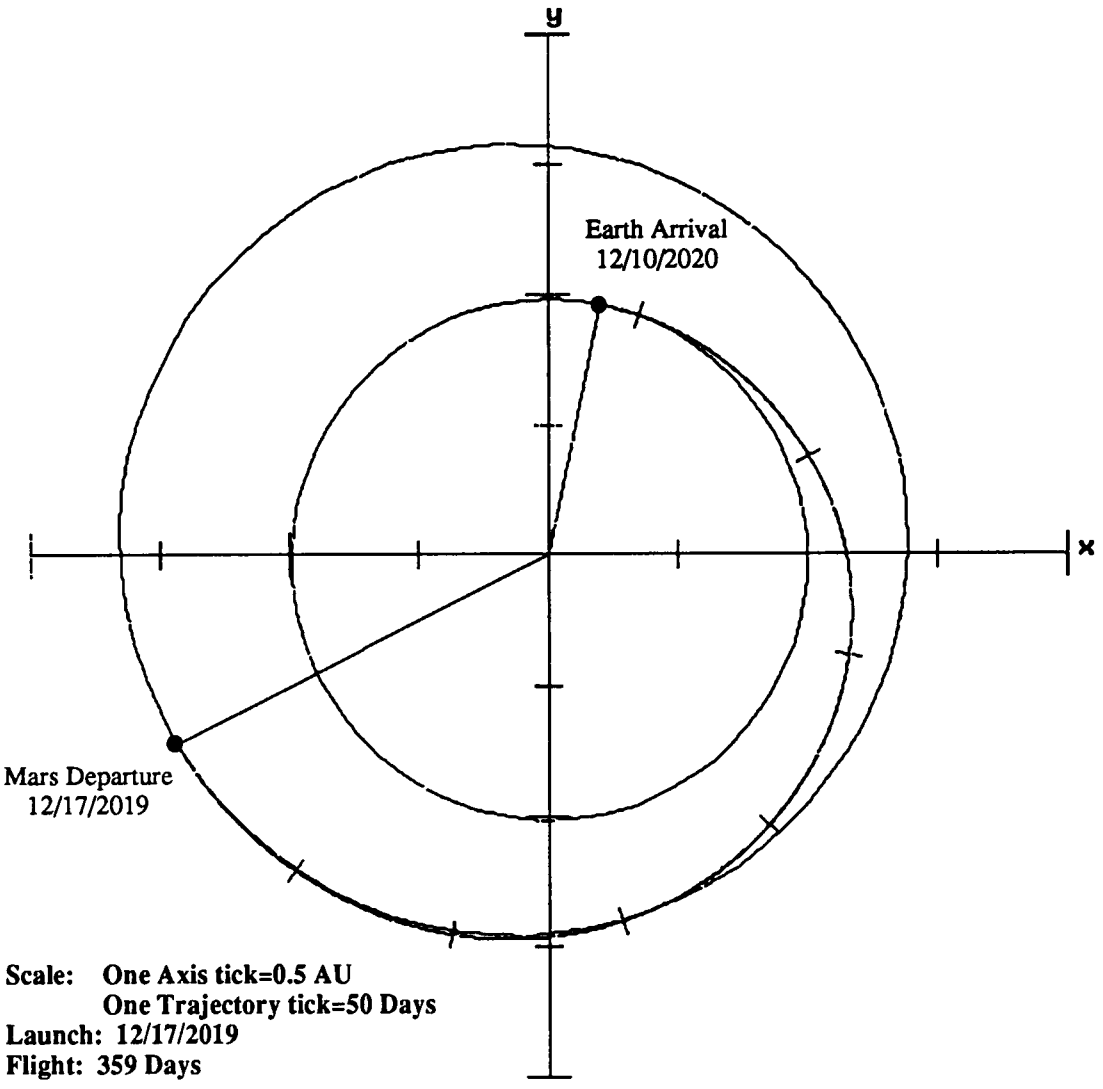

Fig. 4-3: Ecliptic Projection of Mars to Earth Leg of Optimal Solar Sail Trajectory

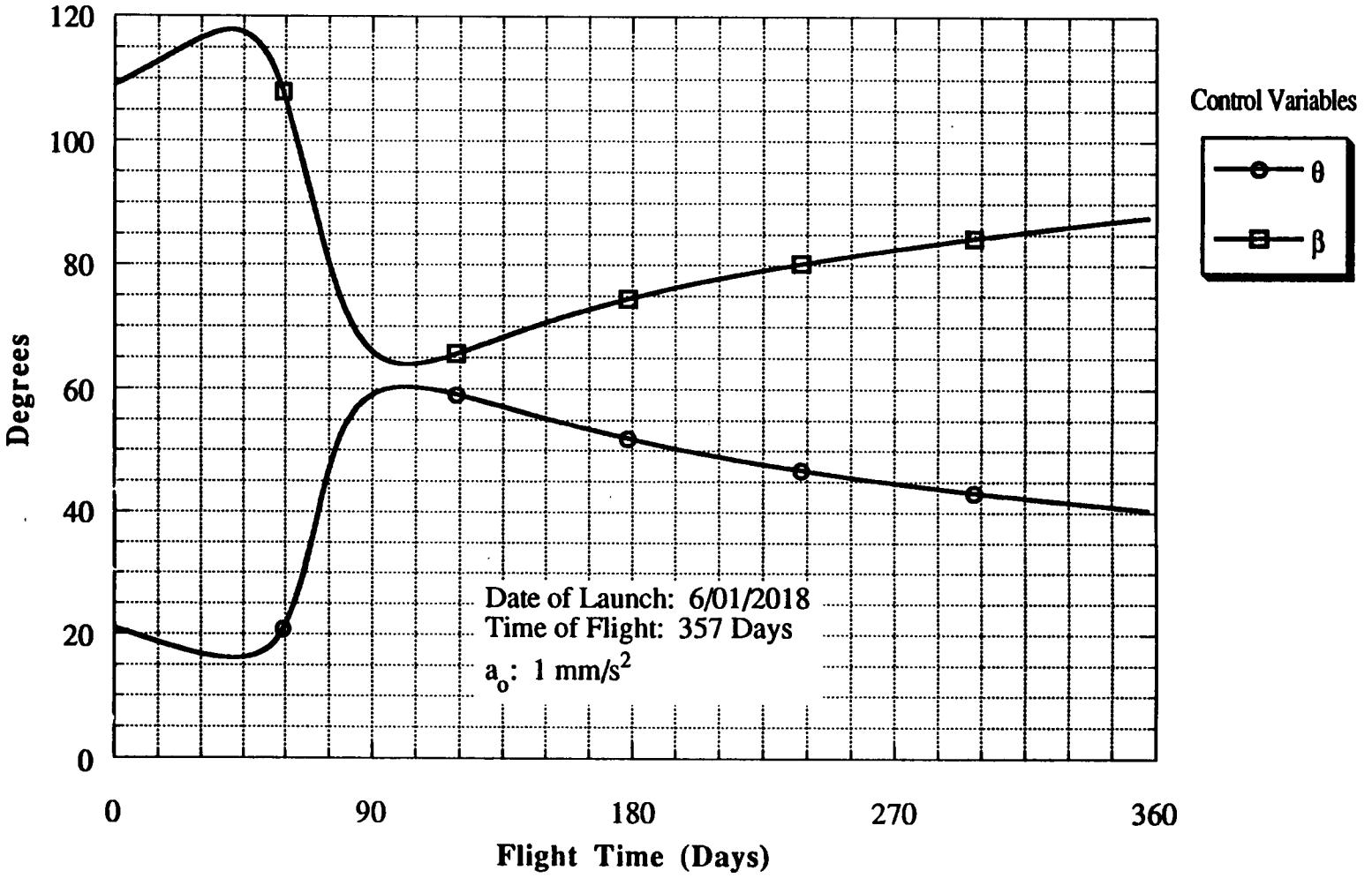

Fig. 4-4: Solar Sail Control Variable Time Histories
for Optimal 2018 Earth to Mars Trajectory

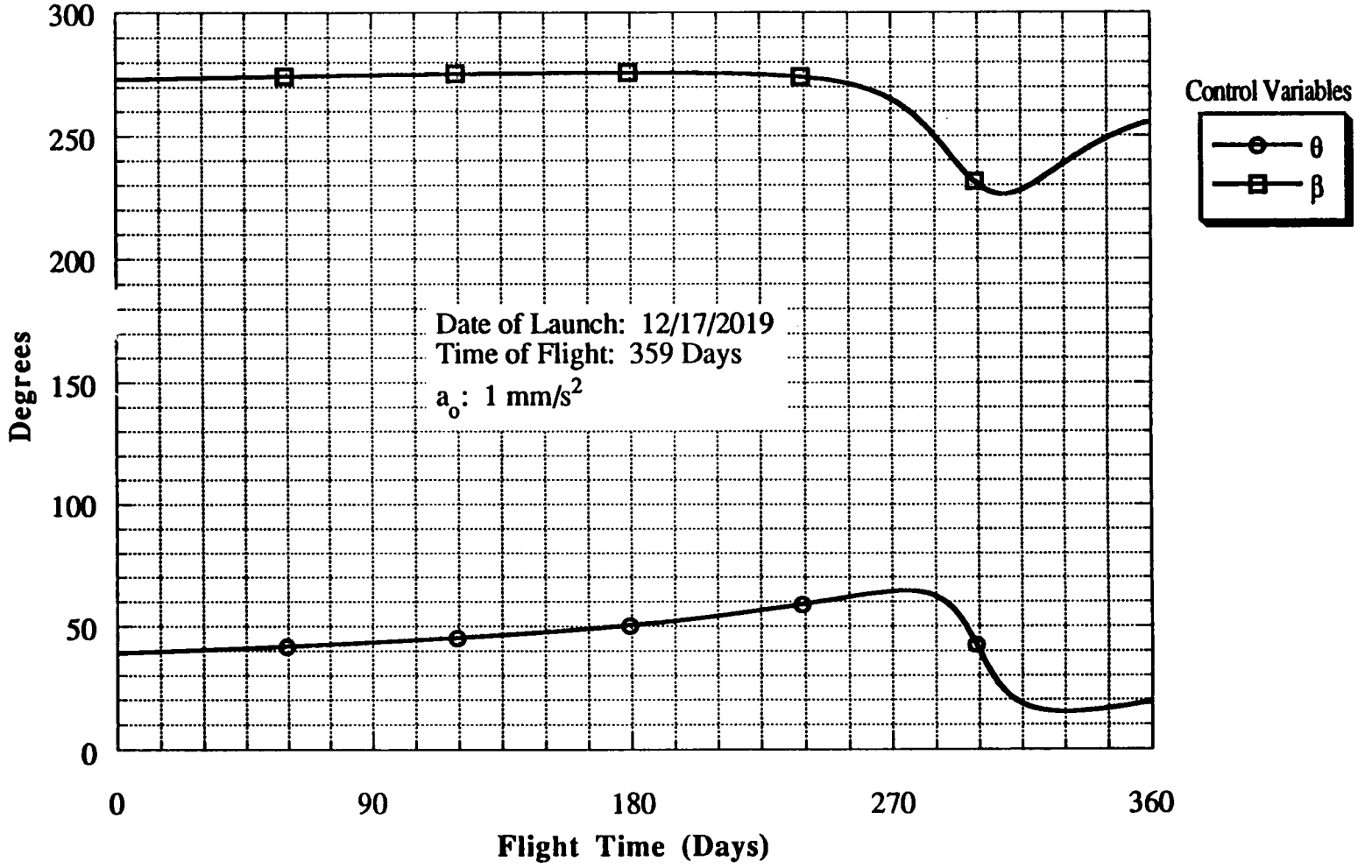

Fig. 4-5: Solar Sail Control Variable Time Histories
for Optimal 2019 Mars to Earth Trajectory

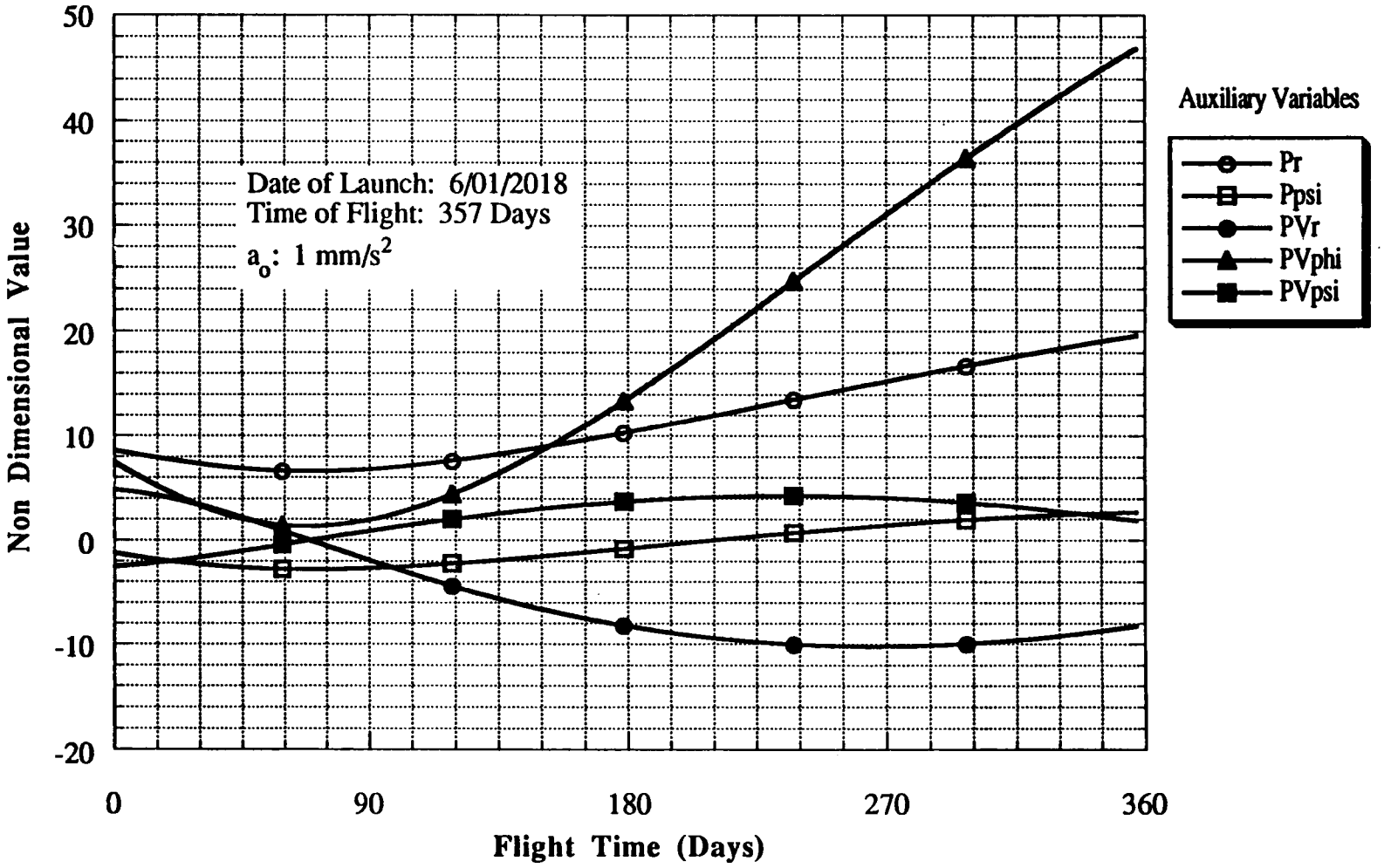

Fig. 4-6: Solar Sail Auxiliary Variable Time Histories
for Optimal 2018 Earth to Mars Trajectory

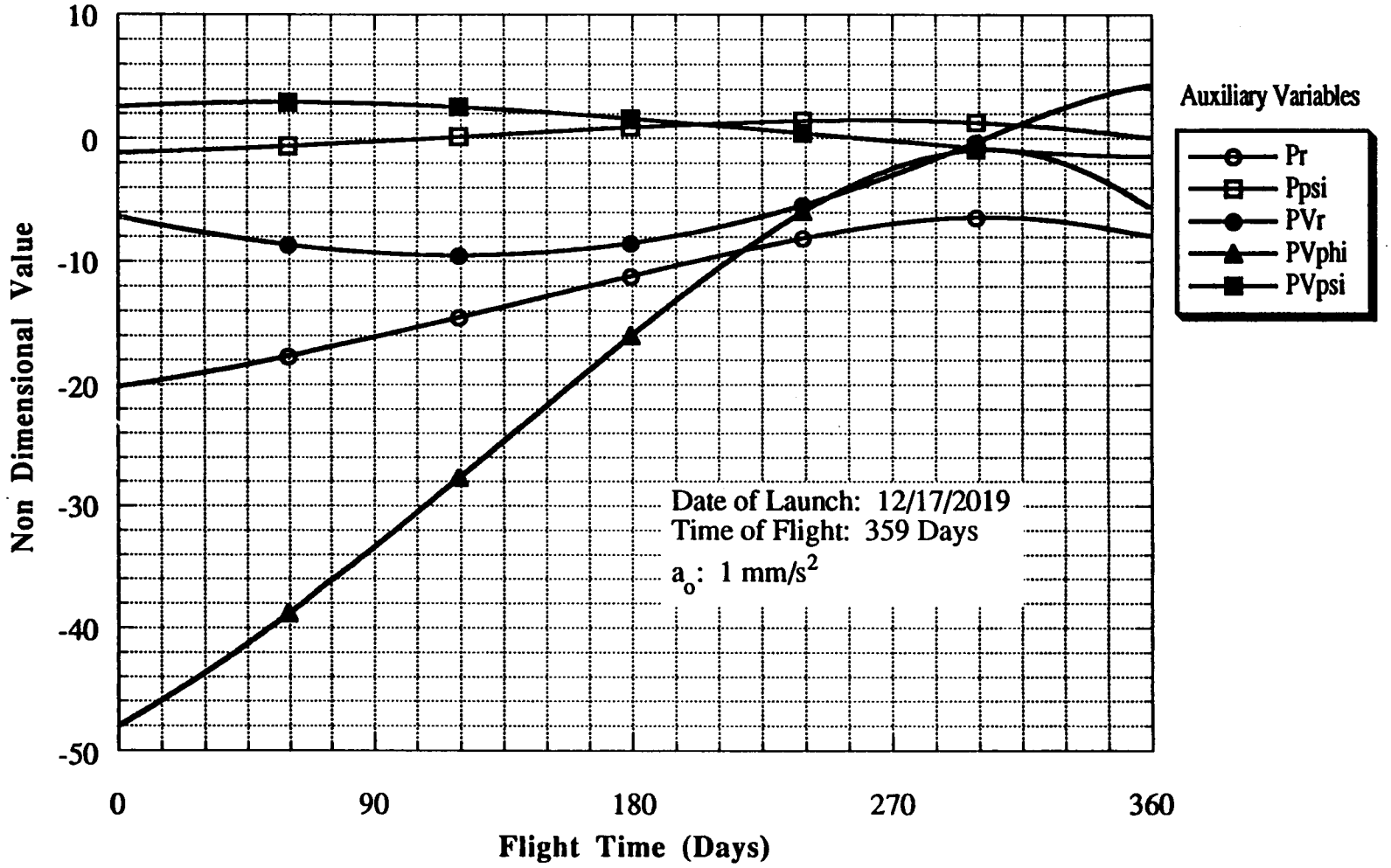

Fig. 4-7: Solar Sail Auxiliary Variable Time Histories
for Optimal 2019 Mars to Earth Trajectory

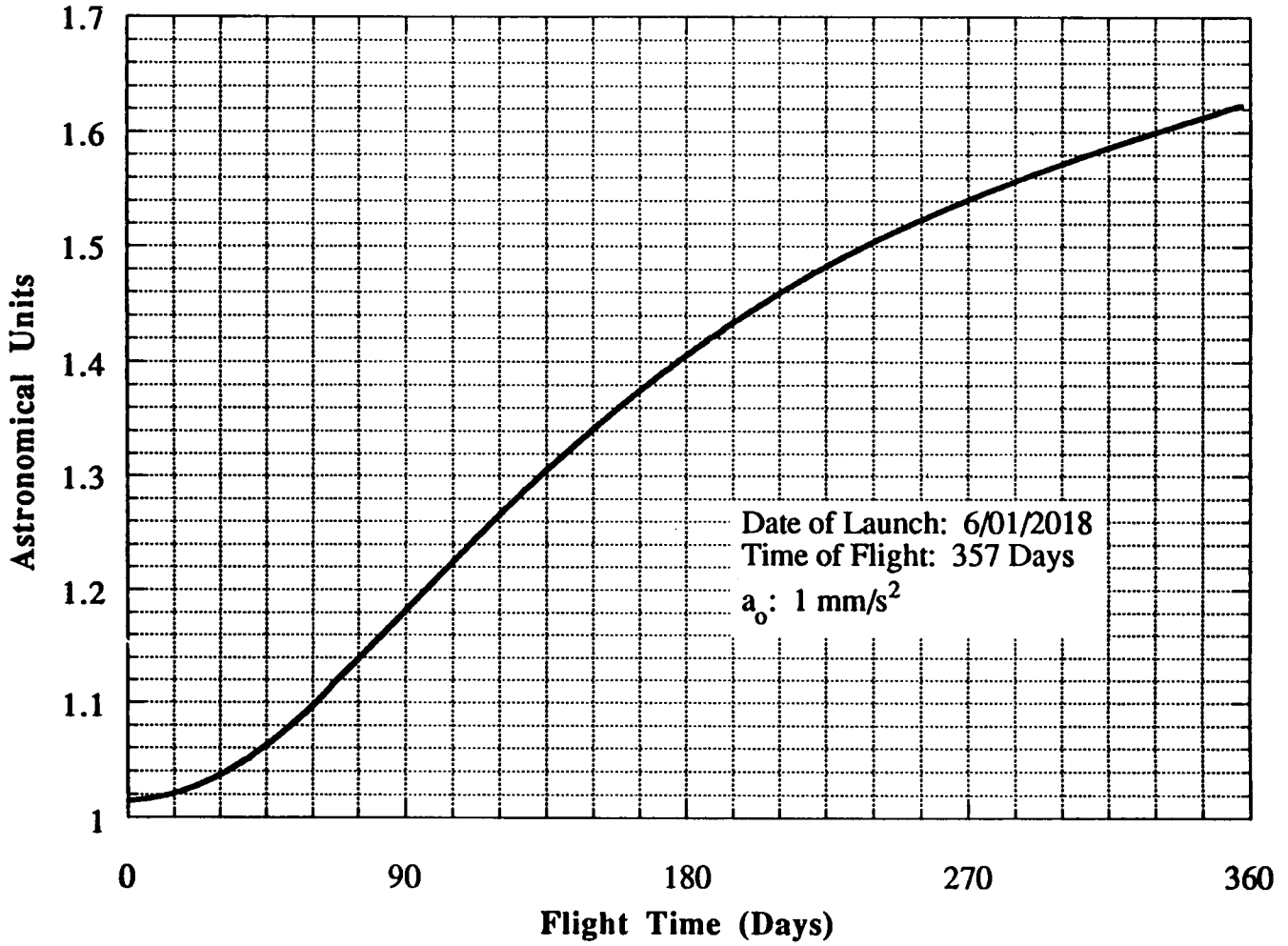

Fig. 4-8: Solar Sail Radial Distance from Sun Time History
for Optimal 2018 Earth to Mars Trajectory

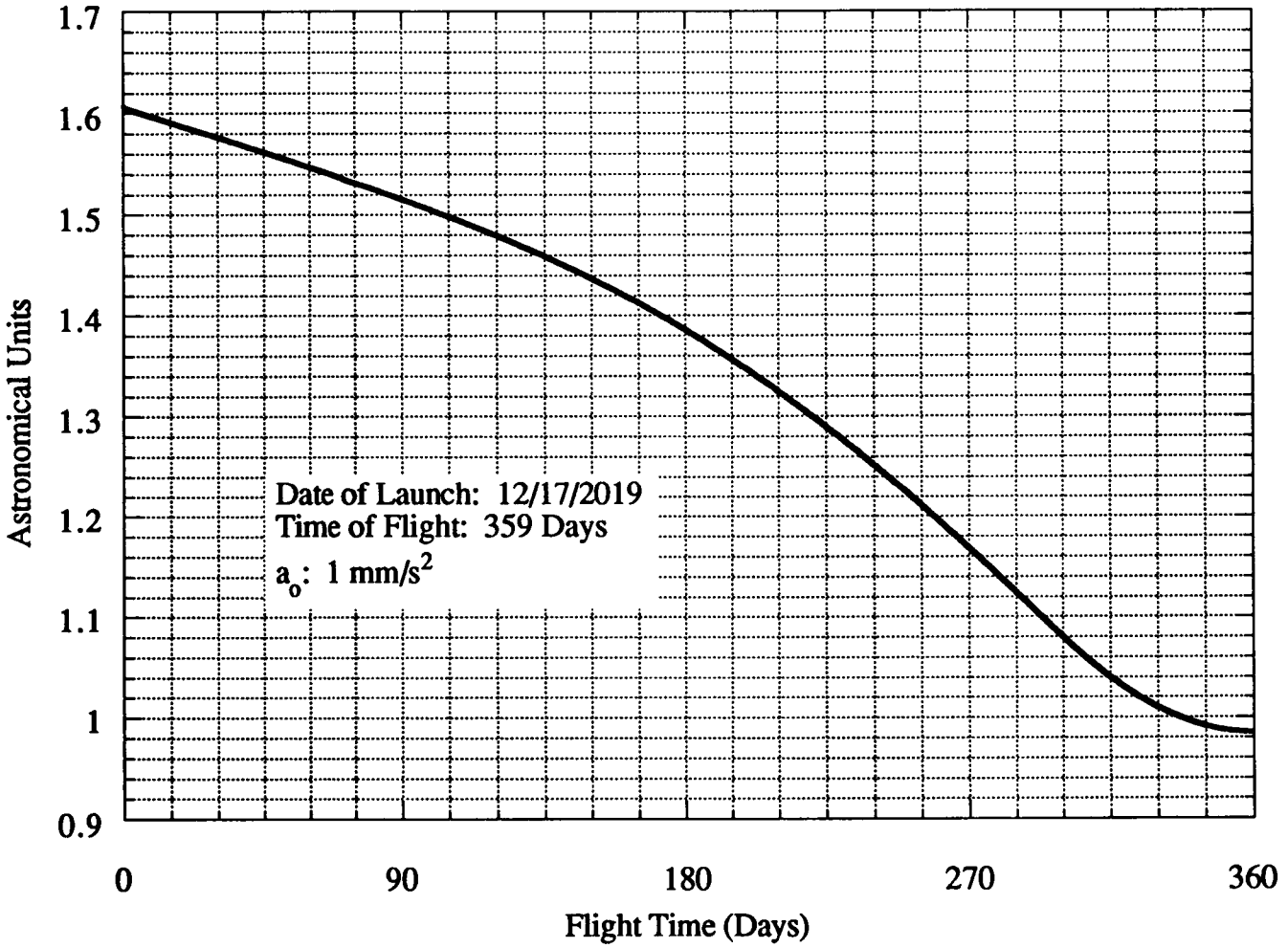

Fig. 4-9: Solar Sail Radial Distance from Sun Time History for Optimal 2019 Mars to Earth Trajectory

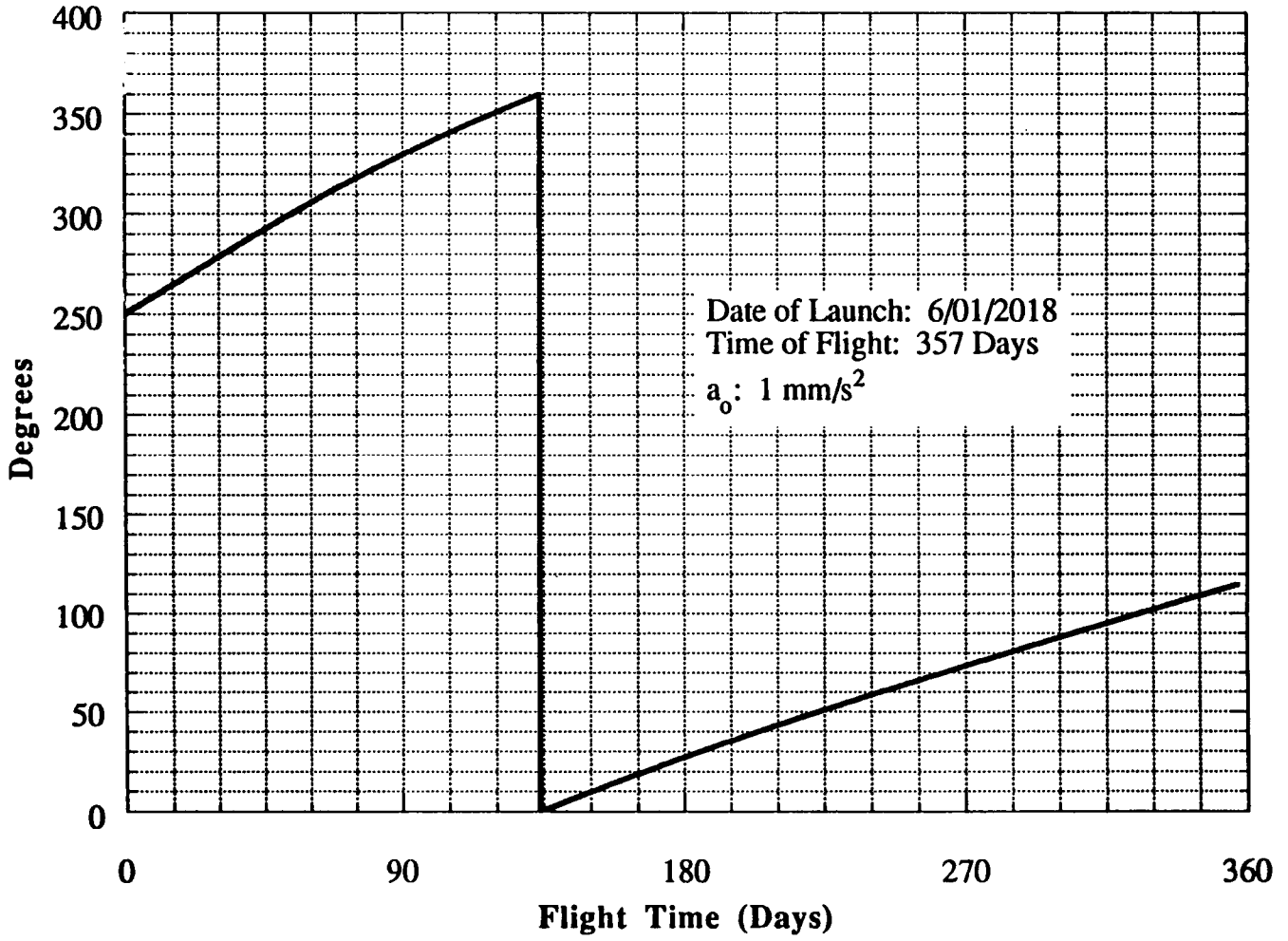

Fig. 4-10: Solar Sail Celestial Longitude Time History
for Optimal 2018 Earth to Mars Trajectory

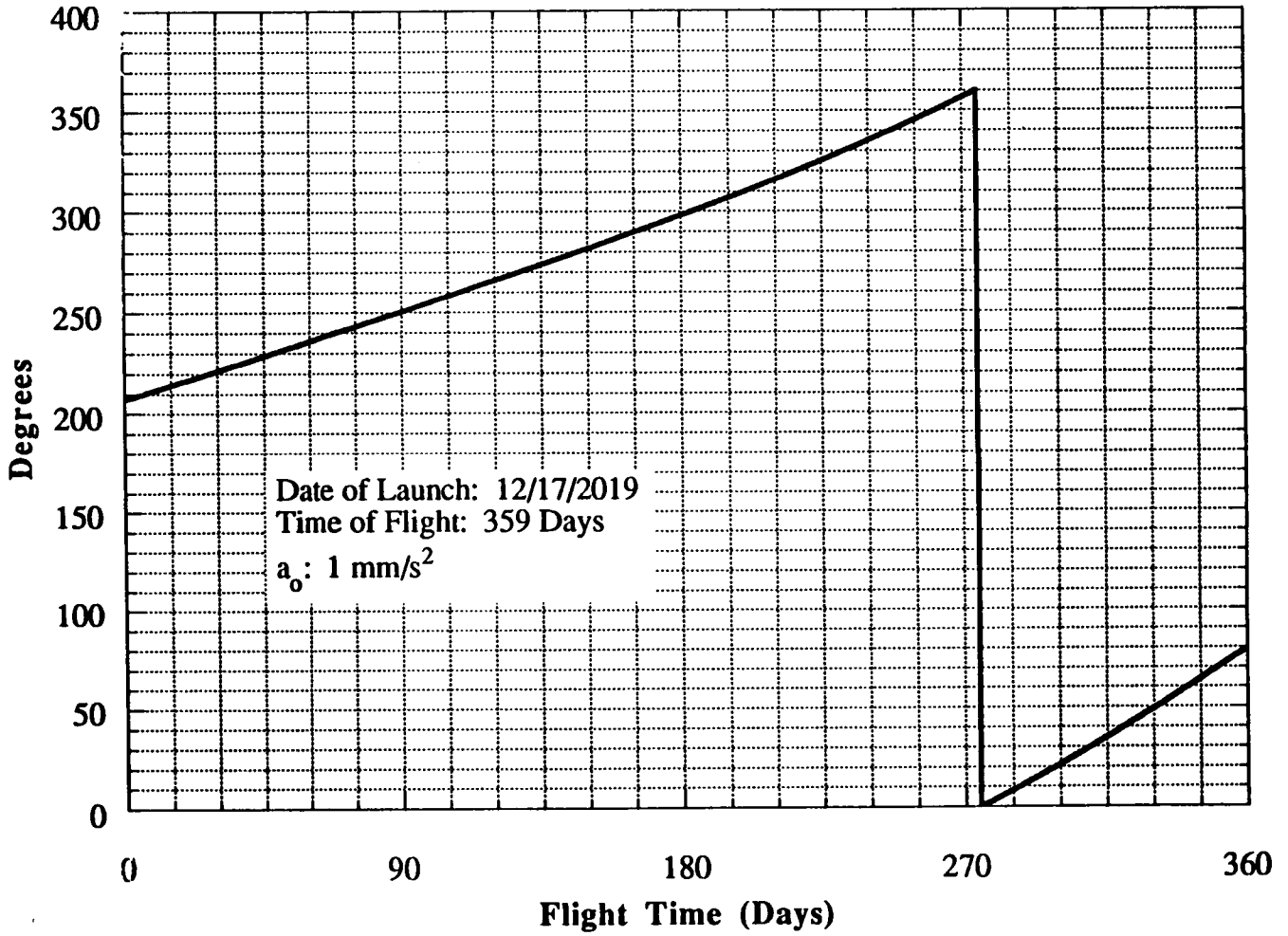

Fig. 4-11: Solar Sail Celestial Longitude Time History
for Optimal 2019 Mars to Earth Trajectory

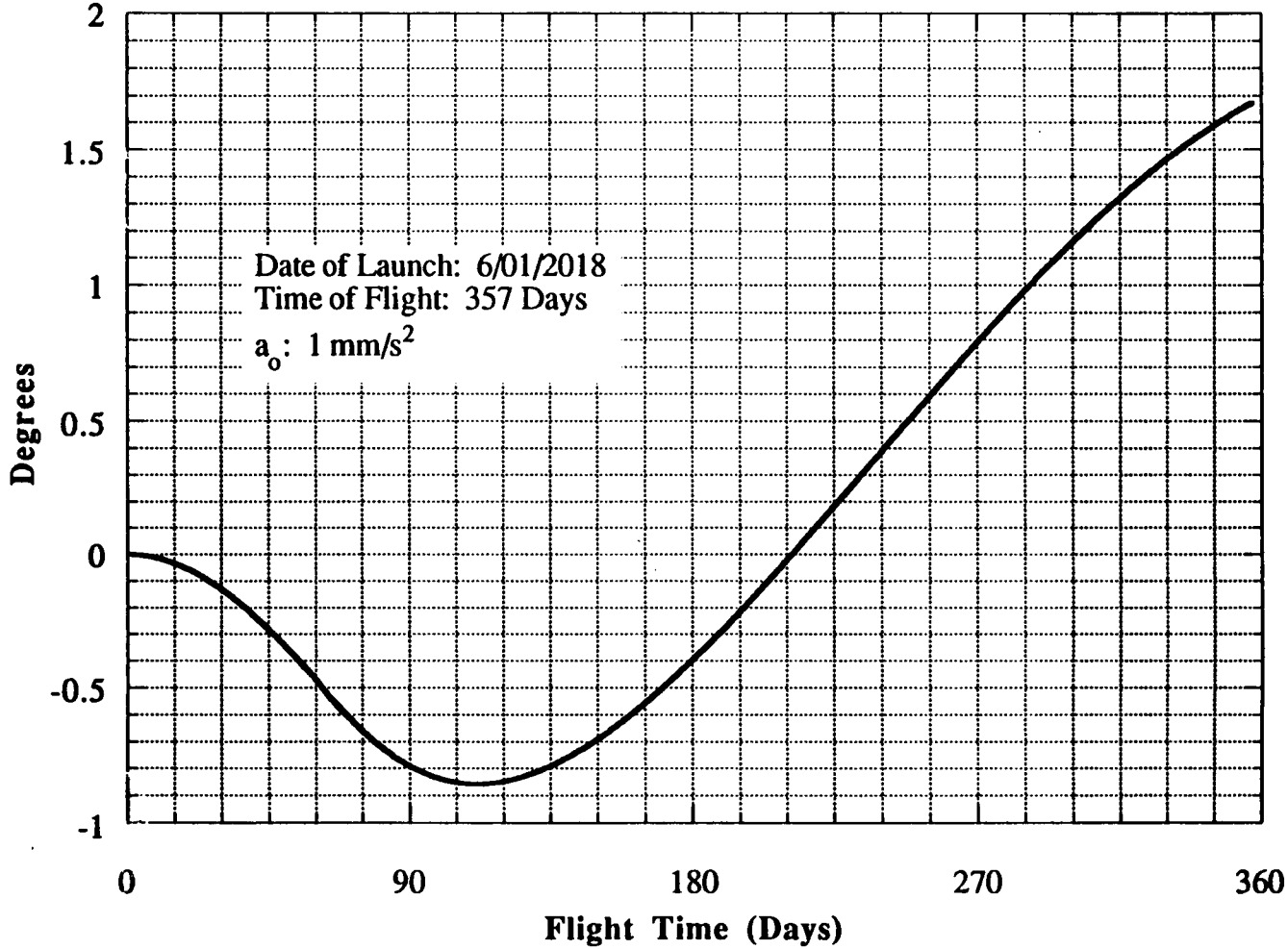

Fig. 4-12: Solar Sail Celestial Latitude Time History
for Optimal 2018 Earth to Mars Trajectory

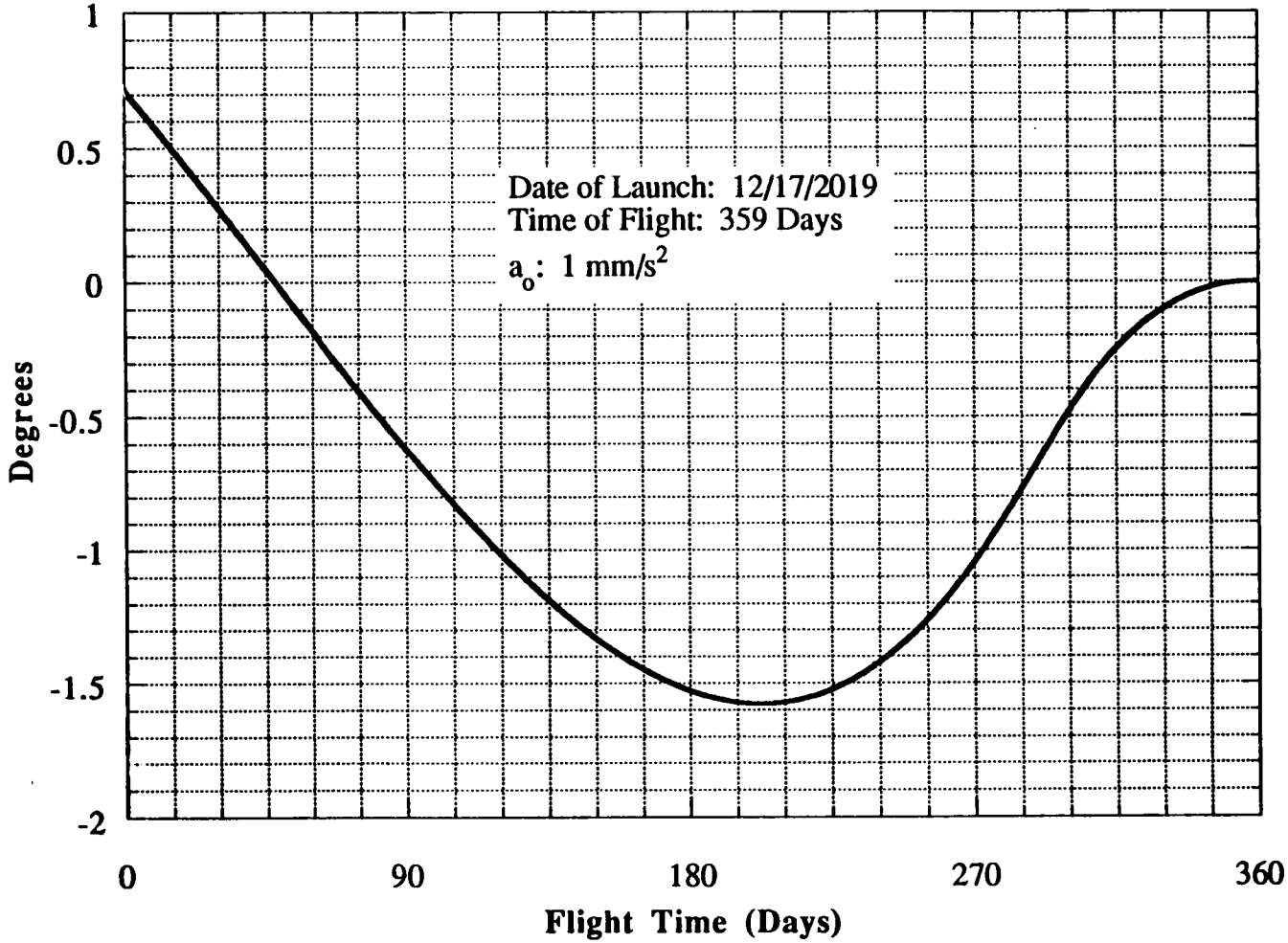

Fig. 4-13: Solar Sail Celestial Latitude Time History
for Optimal 2019 Mars to Earth Trajectory

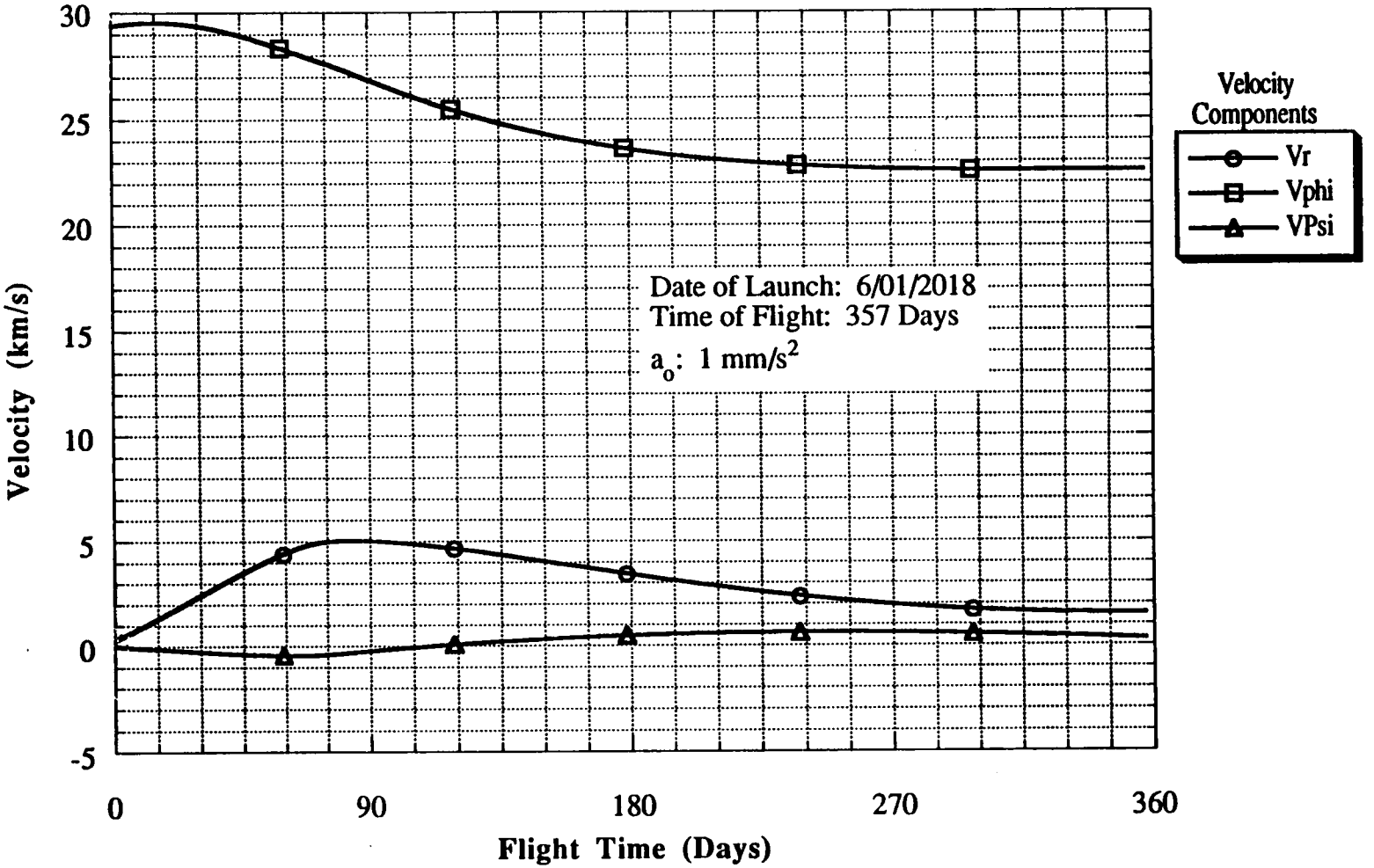

Fig. 4-14: Solar Sail Velocity Component Time Histories
for Optimal 2018 Earth to Mars Trajectory

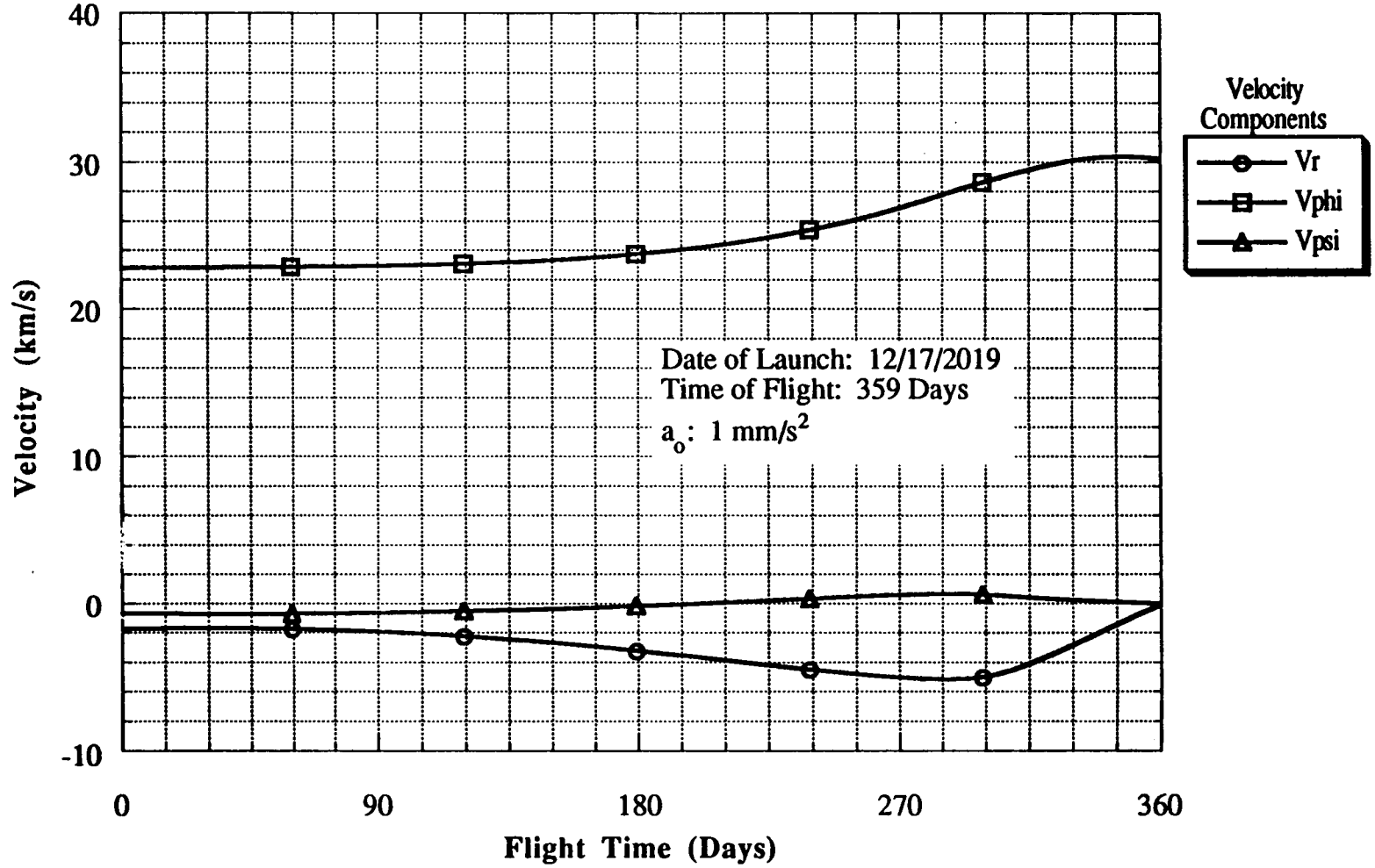

Fig. 4-15: Solar Sail Velocity Component Time Histories
for Optimal 2019 Mars to Earth Trajectory

4-7 Comparison with Chemical and Nuclear Thermal Propulsion

The solar sail presents considerable IMLEO savings over both the chemical and nuclear thermal propulsion systems. As shown in Table 4-1, only 452.24 MT need to be lofted from the earth's surface. This is 22.46% of the mass required by NTP and only 2.38% of the mass required by chemical propulsion. An added benefit is that all of the mass excepting the chemical booster is returnable and presumably reusable. (With a LEO to HEO ferry established the booster would become unnecessary for future missions.) This makes the solar sail an ideal propulsion system for continuous operations between the earth and Mars.

The major drawback of the solar sail is the extensive time required in order to complete the mission. A total of 923 days (roughly 2 1/2) is required, or 216% of the time required for the chemical and NTP options. As stated in Chapter 1, this increases the health and safety risks of the astronauts and aggravates hardware reliability concerns. This length of time also raises questions concerning the psychological health of the crew. While the Russian space program has proven that man can endure space for periods approaching those of the impulsive propulsion mission presented here, no one has ever remained in space for time on the order of that required by the sail. Further studies should be conducted to study man's resilience in space.

However, the low cost of the system makes solar sails appear ideal for robotically controlled cargo missions to Mars. These missions could be done in support of the chemical or NTP options examined above. Placing all equipment required for exploration of Mars but not for sustenance of the crew on a solar sail could greatly reduce the costs of these impulsive mission options. Trade studies should be conducted to examine the cost benefits of such a hybrid mission.

V. Conclusions

The human drive to discover and learn, our increasing need for raw materials, and the expansion of the species are all factors which compel us to strive towards the exploration of Mars. However, given the current budgetary climate, interplanetary exploration will prove impossible if means are not found to make such missions less expensive. Nuclear Thermal Propulsion and Solar Sails are technologies that could make a journey to the red planet possible and are compared to a baseline chemical system in this report. Computational procedures were developed to model the interplanetary flight of these systems. Mission duration and initial mass in low earth orbit (IMLEO) were used as the criteria for comparison. The purpose of each mission was to transport 100 MT of usable cargo to Mars for an exploration mission of at least 30 days. Only 50 MT were assumed returned. The remaining 50 MT were considered consumed (life support supplies) or left on Mars (surface habitat, exploration vehicles, etc.). The final results of the IMLEO and mission duration comparisons are tabulated in Table 5-1.

Table 5-1: IMLEO and Mission Duration Requirements of Competing Systems

Vehicle	IMLEO	Total Mission Duration
Chemical	19,036.05 MT	427 Days
Nuclear Thermal	2,013.77 MT	427 Days
Solar Sail (Two Vehicles)	452 MT	923 Days

For the two impulsive thrust missions (baseline chemical and nuclear thermal) mission opportunities were selected by finding the launch dates and flight times that provided for a type I sprint trajectory with minimum energy cost. The optimal mission

was found to occur on April 13, 2018 with an outbound flight time of 157 days, a 30 day loiter time at Mars, and a 240 day return trip flight time.

For the chemical propulsion case, a two stage O_2H_2 vehicle was assumed with three sets of tanks in the second stage. Propulsive correction maneuvers were assumed to produce a fuel reserve and each tank was assumed to have 1% unusable fuel. Sorption refrigerators were added to each tank to prevent boil-off of the cryogenic propellant. As each set of tanks depleted its fuel it was jettisoned to prevent the necessity of accelerating inert matter during successive propulsive maneuvers. IMLEO for this vehicle was 19,036.05 MT. This figure is excessive. Extended missions comprised of lower energy trajectories and or Venus swingby maneuvers would have to be accepted or new technologies would have to be developed (such as aerobraking) in order to make chemical propulsion viable.

For the nuclear thermal propulsion case, a two stage vehicle utilizing an advanced NERVA engine was assumed. The propellant was liquid hydrogen. As with the chemical case sorption refrigerators were used, corrective maneuvers were allowed for, 1% unusable fuel was assumed, and tanks were jettisoned after they were depleted. IMLEO for this vehicle was found to be 2,013.77 MT. This mass represents 10.58% of the mass required by chemical rocket for the same mission. The extended missions and new technologies mentioned for the chemical case could also be used for this vehicle should additional IMLEO savings be required. With such substantial cost savings, this vehicle should be given serious consideration for any future interplanetary missions.

For the solar sail mission, two sails of 7.54 km^2 and a characteristic acceleration of 1 mm/s^2 were assumed. Each carried one half of the required payload with one carrying that material needed for crew sustenance during interplanetary flight and return cargo and the other carrying that cargo needed for planetary habitation and exploration. A chemical booster stage was assumed to launch the sails from LEO to a parabolic escape

trajectory. Because no fuel is consumed, the sole design criteria for this mission is time of flight. The optimal mission was found to occur on June 1, 2018 with an outbound flight time of 357 days, a 207 day loiter time at Mars, and a return flight time of 359 days. The IMLEO for this system was found to be 452.24 MT. This is 22.46% of the mass required for the nuclear thermal option and only 2.38% of the mass needed for the chemical mission. The sail presents the added benefit of being reusable. However, the time required is 216% of that needed with these two previous systems. With these facts in mind, the sail seems best suited for cargo missions to Mars. These cargo missions could either be continuous "cycling" in support of some permanent exploitation of the Martian surface, or transport of exploration hardware needed for the above chemical and nuclear systems. This latter option would greatly reduce the cost of either impulsive thrust system.

Because of the tremendous opportunities for cost savings presented by nuclear thermal propulsion and solar sails, research and development of both technologies should be restarted. Interplanetary travel via chemical propulsion is not currently a viable option and existence of these systems could be the key to making exploration and exploitation of Mars an achievable goal.

LIST OF REFERENCES

List of References

1) Schultz, Roy J., Lecture Notes in Rocket Propulsion, Vol. I, pp. 4-10, UT Space Institute, Tullahoma, TN.

2) Frisbee, R. H., et al., "Advanced Propulsion Options for the Mars Cargo Mission", JPL D-6620 Rev. A, Jet Propulsion Laborartory, California Institute of Technology, Pasadena, CA, September 1989.

3) Frisbee, R. H., et al., "A Comparison of Chemical Propulsion, Nuclear Thermal Propulsion, and Multimegawatt Electric Propulsion for Mars Missions", JPL D-8023, Jet Propulsion Laborartory, California Institute of Technology, Pasadena, CA, October 1990.

4) Clark, J. S., "Nuclear Propulsion Technology Development", p. 229, Mars: Past, Present, and Future, ed. by E. Brian Pritchard, Progress in Astronautics and Aeronautics, vol. 145, AIAA, 1992.

5) Friedman, Louis, Starsailing: Solar Sails and Interstellar Travel, John Wiley & Sons, Inc., New York, NY, 1991.

6) Orth, C., et al., "Transport Vehicle for Manned Mars Missions Powered by Inertial Confinement Fusion," AIAA Paper 87-1904, AIAA/SAE/ASME 23rd Joint Propulsion Conference, San Diego, CA, June29-July 2, 1987.

7) "Report on the 90 Day Study on Human Exploration of the Moon and Mars," NASA TM--102999, November, 1989.

8) Battin, Richard H., An Introduction to the Mathematics and Methods of Astrodynamics, AIAA, Inc., New York, NY, 1987.

9) Bate, Roger R, Donald D. Mueller, Jerry E. White, Fundamentals of Astrodynamics, Dover Publications, Inc., New York, 1971.

10) Mark, Hans and Harlan J. Smith, "Fast Track to Mars," *Aerospace America*, August 1991, pp. 36-41.

11) Grodzovskii, G. L. et al., Mechanics of Spaceflight Low-Thrust, Israel Program for Scientific Translations, Jerusalem, 1969.

12) Lebedev, V. N.,"Calculation of the Motion of a Low Thrust Spacecraft," NASA TT F-586, Washington D. C., November 1969.

13) Pontryagin, L. S., V. G. Boltyanskii, R. V. Gamkrelidze, and E. F. Mishchenko, The Mathematical Theory of Optimal Processes, The Macmillan Company, New York, 1964.

14) Sauer, Carl G., Personal Communications, September 11, 1992.

15) Sauer, Carl G., "Optimum Solar-Sail Interplanetary Trajectories," AIAA No. 76-792, AIAA/AAS Astrodynamics Conference, San Diego, California, August 18-20, 1976.

16) Burden, Richard L. and J. Douglas Faires, Numerical Analysis, PWS-Kent Publishing Company, Boston, 1989.

17) Garvey, J.M., "Space Station Options for Constructing Advanced Solar Sails Capable of Multiple Mars Missions," AIAA No. 87-1902, AIAA/SAE/ASME/ASEE 23rd Joint Propulsion Conference, San Diego, CA, June 29-July 2, 1987.

18) MacNeal, R. H., The Heliogyro, An Interplanetary Flying Machine, NASA Contractor's Report CR 84460, June 1967.

19) von Braun, Wernher, The Mars Project, University of Illinois Press, Urbana, 1962.

APPENDICES

Appendix A: Impulsive Thrust Program

```
program Battin (input, output, data, PhysicalConstants);
{_____________________________________________________________________}
{*********************************************************************}
{Program:          Intrerplanetary Trajectories                       }
{                                                                     }
{By:               A. Allen McCarley and  Brian K. Funk               }
{                                                                     }
{Date:             May 21, 1990                                       }
{                                                                     }
{Last Revised:     October 2,  1991                                   }
{                                                                     }
{Description:      This program calculates the three dimensional conic trajectory
{                  between the earth and Mars using Battin's modified
{                  Gauss method.  A patched conic approximation is assumed.
{                  Launch Dates, flight times, and desired parking orbits are the
{                  only  required inputs.  This program was designed to use the
{                  earth as either the launch or rendezvous planet.  If one of the
{                  planets considered is not the earth,
{                  minor changes will have to be made.
{                                                                     }
{*********************************************************************}
{---------------------------------------------------------------------}

uses    Ephemeris, Graphical, Indirect, MarsReturn, Quadrant;


type
        name = array[1..5] of string;
        vector = array[1..3] of extended;
        planet = array[1..7] of extended;
        storage = array[1..5] of extended;
        matrix = array[1..6] of array[1..3] of extended;
        placement = array[1..6] of extended;
        vec = array[1..20] of extended;


const {From USAFA Text by Bate, Mueller and White}
        GMearth = 398600;                       {km3/s2}
        GMmars = 42906;                         {km3/s2}
        AU = 1.4959965E+8;                      {km/au}
        GM = 3.96396415708E-14;                 {au3/s2}
        R = 57.29577951;                        {deg/rad}
        root = 1.990970657E-7;
        stepsize = 2;


var
        path,                   {the path of the transport vehicle}
```

```
spacecraft,          {the transport vehicle}
other,               {Mercury}
flyby,               {Venus}
departure,           {The planet of departure}
target: planet;      {The target planet}
x,                   {The Battin independant variable}
L,                   {Another Battin variable}
m,                   {Yet one more Battin variable}
Rop,                 {Radius to mean point between r1 and r2}
c,                   {Cord between endpoints of r1 and r2}
s,                   {Semi-perimeter of triangle r1,r2,c}
a,                   {Semi major axis}
ecc,                 {Eccentricity of the transfer trajectory}
e1,                  {Two more Battin variables}
e2,                  { "  "   "   "   "  }
eccentric1,          {the eccentric anomoly at r1}
eccentric2,          {the eccentric anomoly at r2}
eccalt,              {holding variable for drawing the ships trajectory}
Xi,                  {Battin continued fraction}
K,                   {Continued fraction used to solve the cubic}
p,                   {Transfer orbit parameter}
angmom,              {Transfer orbit angular momentum}
n,                   {Mean daily motion of the spacecraft}
COne,                {Square of the hyperbolic excess speed at launch}
CTwo,                {Square of the hyperbolic excess speed at target}
VOne,                {Heliocentric speed at departure}
VTwo,                {Heliocentric speed at target}
gamma1,              {Spacecraft path angle at launch}
gamma2,              {Spacecraft path angle at approach}
H,                   {Heliocentric angle traverse parameter}
launchT,             {The date of departure}
arriveT,             {The outbound trip time of flight}
arriveT2,            {The return trip time of flight}
JlaunchT,            {The Julian date of departure from earth}
JLaunchT2,           {The Julian date of departure from Mars}
Jcentury1,           {Julian centuries from Jan. 1, 1900 to launch}
JarriveT,            {The Julian date of arrival at Mars}
JarriveT2,           {The Julian date of arrival at earth}
Jcentury2,           {Julian centuries from Jan. 1, 1900 to arrival}
Jcentury,            {Julian centuries from Jan. 1,1900 to considered date}
duration,            {The date of arrival at the target planet}
nu1,                 {True anomaly at time of launch}
nu2,                 {True anomaly at time of landing}
nualt,               {Holding variable for nu1 or nu2 while drawing the path}
inclination,         {inclination of the transfer plane}
xpo,                 {spacecraft's x position}
ypo,                 {spacecraft's y position}
RPark1,              {Radius of the parking orbit around the departure planet}
RPark2,              {Radius of the parking orbit around the target planet}
DeltaV1,             {Delta V to leave the departure planet}
DeltaV2,             {Delta V to enter the parking orbit around the target planet}
```

```
    Days,                   {counting variable that adds up towards the time of flight}
    dt: extended;           {Time of flight}
    CelesLat,               {The celestial latitude}
    radius,                 {Radius of the planetary orbit}
    Vplanet,                {Planet velocity}
    phi,                    {Planet flight path angle}
    E,                      {eccentric anomaly}
    Nu,                     {true anomaly}
    SMinAx,                 {Semi Minor Axis}
    Vpx,                    {Planet x-axis velocity component}
    Vpy,                    {Planet y-axis velocity component}
    long: storage;          {Longitude}
    check,                  {Like Days}
    i: integer;             {Indexing variables}
    place: name;            {Launch and target planets, enterred from keyboard}
    EcliptVpx,              {Planetary velocity in the ecliptic x direction}
    EcliptVpy,              {Planetary velocity in the ecliptic y direction}
    EcliptVPz: storage;     {Planetary velocity in the ecliptic z direction}
    PlPort: Grafport;
    MyWindow2: GrafPtr;{Graphical tools}
    MyWindow1: WindowPtr;   { " " }
    wMgrPort: GrafPtr;      {" "}
    MyRect1,                {" "}
    MyRect2: Rect;          {" "}
    event: EventRecord;
    ihor,                   {" "}
    iver: integer;          {" "}
    scra,                   {" "}
    scrb,                   {" "}
    scrc,                   {" "}
    scrd: extended;         {" "}
    y: vec;                 {" "}
    rock1,                  {The graphical region for the departure planet}
    rock2,                  {The graphical region for the departure planet}
    rock3,                  {The graphical region for Venus}
    rock4,                  {The graphical region for Mercury}
    rock5: RgnHandle;       {The graphical region for the spacecraft}
    xlast,                  {last x position in local graphport coordinate system}
    ylast: placement;       {last y position in local graphport coordinate system}
    Trans: matrix;          {The transformation matrix}
    t,                      {time since perifocal passage}
    newtime,                {one day (step size)}
    w,                      {mean motion}
    delta: extended;        {Newton's method error term}

{*************************************************************************}
{******************************** Functions ******************************}
{*************************************************************************}
function tan (arguement: extended): extended;

                { This function returns the tangent of the input angle }
```

```
begin                                                                    {tan}
        tan := sin(arguement) / cos(arguement);
end;                                                                     {tan}

{=============================================================}
function asin (arguement: extended): extended;

                {This function returns the arcsine value of the input angle in radians}

begin                                                                    {asin}
        asin := ArcTan((arguement) / (sqrt(1 - arguement * arguement)));
end;                                                                     {asin}

{=============================================================}
function acos (arguement: extended): extended;

                {This function returns the arcosine value of the input angle in radians}

begin                                                                    {acos}
        acos := pi / 2 - ArcTan((arguement) / (sqrt(1 - arguement * arguement)));
end;                                                                     {acos}

{*************************************************************************}
{***************************** Procedures ********************************}
{*************************************************************************}
procedure CaseEntry1 (var departure, target: planet; var launchT, JlaunchT, arriveT,
JarriveT, dt,                   Jcentury1, Jcentury2: extended; var place: name);

                        { This procedure prompts for the names of the    }
                        { target and launch planets and the dates of these }
                        { events.  It also calls the procedure           }
                        { ChoosePlanet to read the planetary orbital     }
                        { elements and the procedure Convert to convert  }
                        { all dates to the Julian calendar.              }

var     i: integer;                     { An indexing variable }

begin                                                                    {CaseEntry}
        writeln('Enter the launch date as mm.ddyyyy or as a Julian date.');
        readln(launchT);
        JlaunchT := launchT;
        if ((JlaunchT / 1000) < 1) then
                Convert(JlaunchT);
        Jcentury1 := (JlaunchT - 2415021) / (36525);
        place[1] := 'earth';
        for i := 1 to 7 do
                departure[i] := 0;
        earth(departure, Jcentury1);
        writeln('Enter the desired time of flight to Mars in days.');
        readln(arriveT);
```

```
        JarriveT := JlaunchT + arriveT;
        Jcentury2 := (JarriveT - 2415021) / (36525);
        place[2] := 'Mars';
        for i := 1 to 7 do
                target[i] := 0;
        Mars(target, Jcentury2);
        dt := (JarriveT * 24 * 3600) - (JlaunchT * 24 * 3600);
        place[3] := 'Venus';
        for i := 1 to 7 do
                flyby[i] := 0;
        Venus(flyby, Jcentury1);
        place[4] := 'Mercury';
        for i := 1 to 7 do
                other[i] := 0;
        Mercury(other, Jcentury1);
end;                                                    {CaseEntry}

{================================================================}
procedure GiveOutput1 (var place: name; var H, launchT, JlaunchT, arriveT, JarriveT,
                        VOne, VTwo, COne, CTwo, ZOne, ZTwo, a, ecc, p, angmom, n,
                        nu1, nu2, DeltaV1, DeltaV2: extended; radius, Long: storage;
                        spacecraft: planet);

                        { This procedure writes the output to the screen      }
                        { and to the external file, data.                     }

begin                                                   {GiveOutput}
        writeln;
        writeln;
        writeln('Departure Planet:  ', place[1]);
        writeln('Launch Date:  ', launchT : 1 : 6);
        writeln('Julian Launch Date = ', JlaunchT : 1 : 0);
        writeln('R1 = ', radius[1] : 1 : 3, ' AU          Long = ', Long[1] * R : 1 : 3, '°');
        writeln('Celestial Latitude at time of launch = ', CelesLat[1] * R : 1 : 4);
        writeln('Launch True Anomaly = ', R * Nu1 : 1 : 6, '°');
        writeln('V1 = ', VOne : 1 : 3, ' km/s' : 5, '  F1 = ', R * ZOne : 1 : 3, '°');
        writeln('C3 = ', COne : 1 : 3, ' (km/sec)^2     DeltaV1 = ', DeltaV1 : 1 : 4);
        writeln('Mlong: ', departure[2] - trunc(departure[2] / (360)) * 360 : 1 : 5);
        writeln;
        writeln('HELIOCENTRIC CONIC:');
        writeln('a = ', a : 1 : 3, '  e = ', ecc : 1 : 4);
        writeln('HCA = ', R * H : 1 : 2);
        writeln('p = ', p : 1 : 4, ' h = ', angmom : 1 : 4, ' n = ', n * 180 / pi * 24 *
                3600 : 1 : 4, '°/day');
        writeln('sc omega = ', spacecraft[7] : 1 : 6);
        writeln('sc pi = ', spacecraft[3] : 1 : 6);
        writeln('sc SmOmega = ', spacecraft[3] - spacecraft[7] : 1 : 6);
        writeln('sc inclination = ', spacecraft[6] : 1 : 6);
        writeln;
        writeln('Arrival Planet:  ', place[2]);
        writeln('Time of Flight:  ', arriveT : 1 : 0, ' days');
```

```
        writeln('Julian Arrival Date:  ', JarriveT : 1 : 0);
        writeln('R2 = ', radius[2] : 1 : 3, ' AU          Long = ', Long[2] * R : 1 : 3, 'º');
        writeln('Celestial Latitude at time of arrival = ', CelesLat[2] * R : 1 : 4);
        writeln('Arrival True Anomaly = ', R * Nu2 : 1 : 6, 'º');
        writeln('V2 = ', VTwo : 1 : 3, ' km/s', '  F2 = ', R * ZTwo : 1 : 3, 'º');
        writeln('C3 = ', CTwo : 1 : 3, ' (km/sec)^2     DeltaV2 = ', DeltaV2 : 1 : 4);
end;                                                                    {GiveOutput}

{=============================================================}
procedure EarthParking (var Rpark1, COne, DeltaV1: extended);

                        { This procedure allows the user to select the      }
                        {  altitude of the circular parking orbit at earth  }

var     Vreq1,                          {Required departure velocity}
        Vcirc1: extended;               {Circular parking orbit velocity}
begin                                                                   {EarthParking}
        writeln('Enter the desired circular earth parking orbit altitude in km.');
        readln(Rpark1);
        Vreq1 := sqrt(2 * GMearth / (Rpark1 + 6378.145) + COne);
        Vcirc1 := sqrt(GMearth / (Rpark1 + 6378.145));
        DeltaV1 := Vreq1 - Vcirc1;
end;                                                                    {EarthParking}

{=============================================================}
procedure MarsParking (var Rpark2, CTwo, DeltaV2: extended);

                        { This procedure allows the user to select the      }
                        {altitude of the circular parking orbit at Mars.    }

var     Vreq2,                          {Required departure velocity}
        Vcirc2: extended;               {Circular parking orbit velocity}

begin                                                                   {MarsParking}
        writeln('Enter the desired circular Mars parking orbit altitude in km.');
        readln(Rpark2);
        Vreq2 := sqrt(2 * GMmars / (Rpark2 + 3310) + CTwo);
        Vcirc2 := sqrt(GMmars / (Rpark2 + 3310));
        DeltaV2 := Vreq2 - Vcirc2;
end;                                                                    {MarsParking}

{=============================================================}
procedure Mercury (var choice: planet; time: extended);

                        { This procedure determines the orbital     }
                        { elements for the planet Mercury at        }
                        { the date chosen.                          }

begin                                                                   {Mercury}
        choice[1] := (14732.4197380 - 0.000001355 * time);
        choice[2] := (178 + 10/60 + 44.68/3600) + (538106654.80/3600)*time +
```

```
            (1.084/ 3600)*time*time;
        choice[3] := 75 + 53/60 + 58.91/3600 + (5599.76/3600)*time +
(1.061/3600)*time * time;
        choice[4] := 0.20561421 + 0.00002046 * time - 0.000000030 * time * time;
        choice[5] := 0.38709860;
        choice[6] := 7 + 10.37 / 3600 + (6.699 / 3600) * time - (0.066 / 3600) * time *
time;
        choice[7] := 47 + 8/60 + 45.4/3600 + (4266.75/3600)*time +
(0.626/3600)*time*time;
end;                                                        {Freddy Mercury}

{============================================================}
procedure Venus (var choice: planet; time: extended);

                        { This procedure determines the orbital     }
                        { elements for the planet Venus at the      }
                        { date chosen.                              }

begin                                                               {Venus}
        choice[1] := (5767.6697692 + 0.0000002628 * time);
        choice[2] :=
342+46/60+1.39/3600+(210669162.88/3600)*time+(1.1148/3600)*time*time;
        choice[3] := 130 + 9/60 + 49.8/3600 + (5068.93/3600)*time -
(3.515/3600)*time*time;
        choice[4] := 0.00682069 - 0.00004774 * time + 0.000000091 * time * time;
        choice[5] := 0.7323162;
        choice[6] := 3 + 23/60 + 37.07/3600+(3.621/3600) * time -
(0.0035/3600)*time*time;
        choice[7] := 75 + 46/60 + 46.73/3600 + (3239.46/3600) * time +
(1.476/3600)*time*time;
end;                                                                {Venus}

{============================================================}
procedure earth (var choice: planet; time: extended);

                        { This procedure determines the orbital     }
                        { elements for the planet earth at the      }
                        { date chosen.                              }

begin                                                               {earth}
        choice[1] := (3548.1928323 - 0.000001103 * time);
        choice[2] := 99+41/60+48.04/3600+(129602768.13/3600)*time+(1.089/3600)
                *time* time;
        choice[3] := 101 + 13 / 60 + 15 / 3600 + (6189.03 / 3600) * time + (1.63 / 3600) *
                time * time + (0.012 / 3600) * time * time * time;
        choice[4] := 0.01675104 - 0.00004180 * time - 0.000000126 * time * time;
        choice[5] := 1.00000023;
        choice[6] := 0.0;
        choice[7] := 0.0;
end;                                                                {earth}
```

```
{=============================================================}
procedure Mars (var choice: planet; time: extended);

                    { This procedure determines the orbital      }
                    { elements for the planet Mars at the        }
                    { date chosen.                               }

begin                                                            {Mars}
        choice[1] := (1886.5186207 + 0.000000463 * time);
        choice[2] := 293+44/60+51.46/3600+(68910103.83/3600)*time+(1.1184/3600)*
                time*time;
        choice[3] := 334 + 13/60 + 5.53/3600 + (6626.73/3600) * time + (0.4675/3600) *
                time * time - (0.0043 / 3600) * time * time * time;
        choice[4] := 0.09331290 + (0.000092064) * time - (0.000000077) * time * time;
        choice[5] := 1.52368840;
        choice[6] := 1 + 51 / 60 + 1.2 / 3600 - (2.43 / 3600) * time + (0.0454 / 3600) *
                time * time;
        choice[7] := 48 + 47 / 60 + 11.19 / 3600 + (2775.57 / 3600) * time - (0.005 /
                3600) * time * time - (0.0192 / 3600) * time * time * time;
end;                                                    {Mars bar (with almonds)}

{=============================================================}
procedure position (var EA, Nu: extended; a, ecc: extended; body: planet; ship: boolean);

var
        verify,                 {verifies the quadrant of nu}
        Mo,                     {Previous value for Mean in iteration}
        E,                      {temporary holding variable for EA}
        rotations,              {The number of 2*pi multiples in E}
        term,                   {just a silly simplification term}
        Mean: extended;         {mean anomaly}
        counter: integer;       {counter}

begin                                                            {position}
        if (ship = TRUE) then
                begin                                                    {if}
                        t := (sqrt(a * a * a) / root) * (EA - ecc * sin(EA));
                        newtime := t + stepsize * 86400;
                        w := sqrt(GM) / sqrt(a * a * a);
                        Mean := w * newtime;
                        counter := 0;
                        repeat
                                counter := counter + 1;
                                delta := (-EA + ecc * sin(EA) + Mean) / (1 - ecc *cos(EA));
                                EA := EA + delta;
                        until ((delta < 0.0000000000001) or (counter = 40));
                        term := (cos(EA) - ecc) / (1 - ecc * cos(EA));
                        Nu := pi / 2 - ArcTan(term / (sqrt(1 - term * term)));
                        verify := EA - trunc(EA / (2 * pi)) * 2 * pi;
                        if (verify > pi) then
                                Nu := 2 * pi - Nu;
```

```
            end                                                    {if}
        else
            begin                                                  {else}
                Mean := pi / 180 * (body[2] - body[3]);
                if (Mean < 0) then
                    Mean := Mean + (2 * 3.141592654);
                E := Mean + ecc * (sin(Mean) + ecc * sin(2 * Mean) / 2);
                repeat
                    Mo := E - ecc * sin(E);
                    delta := (Mean - Mo) / (1 - ecc * cos(E));
                    E := E + delta;
                until (abs(delta) <= 1E-14);
                Nu := 2*ArcTan(sqrt((1 + ecc)/(1 - ecc))*((sin(E / 2) / cos(E/2))));
                if (Nu < 0) then
                    Nu := Nu + 2 * pi;
                rotations := E / (2 * pi);
                EA := E - trunc(rotations) * 2 * pi;
            end;
                                                                   {else}
end;                                                               {position}

{================================================================}
                    { The next several procedures (all beginning with a "g")   }
                    { are tools used in creating the real (compressed) time    }
                    { graphics for a Macintosh computer. They may be           }
                    { if some other sort of computer is being used.            }
{================================================================}

procedure Gscale (Xmin, Xmax, Ymin, Ymax: extended);

begin                                                              {Gscale}
    scra := ihor / (Xmax - Xmin);
    scrb := -scra * Xmin;
    scrc := iver / (Ymin - Ymax);
    scrd := -scrc * Ymax
end;                                                               {Gscale}
{================================================================}
procedure Gmove (x, y: extended);

begin                                                              {Gmove}
    MoveTo(trunc(scra * x + scrb), trunc(scrc * y + scrd))
end;                                                               {Gmove}

{================================================================}
procedure Gdraw (x, y: extended);

begin                                                              {Gdraw}
    LineTo(trunc(scra * x + scrb), trunc(scrc * y + scrd))
end;                                                               {Gdraw}

{================================================================}
```

```
procedure Gxaxis (Ycoord, TickSpace, Origin: extended);

var
        iy: integer;
        xi, x, dx, s: extended;

begin                                                                   {Gxaxis}
        iy := trunc(scrc * Ycoord + scrd);
        MoveTo(0, iy);
        Line(700, 0);
        if TickSpace > 0.0 then
                begin                                                   {if}
                        xi := Origin * scra + scrb;
                        dx := TickSpace * scra;
                        MoveTo(trunc(xi), iy - 8);
                        Line(0, 16);
                        s := -1;
                        repeat
                                x := xi;
                                repeat
                                        x := x + s * dx;
                                        MoveTo(trunc(x), iy - 4);
                                        Line(0, 8);
                                until (x > 700.0) or (x < 0.0);
                                s := s + 2;
                        until s > 1;
                end;                                                    {if}
end;                                                                    {Gxaxis}

{=================================================================}
procedure Gyaxis (Xcoord, TickSpace, Origin: extended);

var
        ix: integer;
        yi, y, dy, s: extended;
begin                                                                   {Gyaxis}
        ix := trunc(scra * Xcoord + scrb);
        MoveTo(ix, 0);
        Line(0, 500);
        if TickSpace > 0.0 then
                begin                                                   {if}
                        yi := Origin * scrc + scrd;
                        dy := TickSpace * scrc;
                        MoveTo(ix - 8, trunc(yi));
                        Line(16, 0);
                        s := -1;
                        repeat
                                y := yi;
                                repeat
                                        y := y + s * dy;
                                        MoveTo(ix - 4, trunc(y));
```

```
                        Line(8, 0);
                    until (y > 500.0) or (y < 0.0);
                    s := s + 2;
                until s > 1;
            end;                                                    {if}
end;                                                                {Gyaxis}

{=================================================================}
procedure Gerase;

var
    box: rect;

begin                                                               {Gerase}
    box.top := 0;
    box.left := 0;
    box.bottom := iver;
    box.right := ihor;
    eraserect(box);
end;                                                                {Gerase}

{=================================================================}
procedure Gwindow1 (left, top, right, bottom: integer; name: string);

begin                                                               {Gwindow1}
    ihor := right - left;
    iver := bottom - top;
    InitGraf(@ThePort);
    GetWMgrPort(wMgrPort);
    MyRect1 := wMgrPort^.portRect;
    SetRect(MyRect1, left, top, right, bottom);
    MyWindow1 := NewWindow(nil, MyRect1, name, TRUE, 0, POINTER(-1),
        FALSE, 0);
    SetPort(MyWindow1);
    Gscale(0, 1, 0, 1);
end;                                                                {Gwindow1}

{=================================================================}
procedure Gimove (ix, iy: integer);

begin                                                               {Gimove}
    MoveTo(ix, iy)
end;                                                                {Gimove}

{=================================================================}
procedure Gidraw (ix, iy: integer);

begin                                                               {Gimove}
    LineTo(ix, iy)
end;                                                                {Gimove}
```

```
{============================================================}
procedure Gpause;

var
        dummy: boolean;

begin                                                   {Gpause}
        repeat
                dummy := GetNextEvent(everyevent, event);  { This allows screenshots }
        until button;
        DisposeWindow(MyWindow1);
end;                                                    {Gpause}

{============================================================}
procedure BuildTransform (var body: planet; var Trans: matrix);

                        { This procedure builds the matrix which serves     }
                        { as the transformation matrix between the          }
                        { ecliptic and pericentric coordinate system        }

var
        SmOmega: extended;              {arguement of periapsis}

begin                                                   {BuildTransform}
        SmOmega := body[3] - body[7];
        Trans[1, 1] := cos(body[7] / R) * cos(SmOmega / R) - sin(body[7] / R) *
                sin(SmOmega/R) * cos(body[6] / R);
        Trans[1, 2] := -cos(body[7] / R) * sin(SmOmega / R) - sin(body[7] / R) *
                cos(SmOmega/R) * cos(body[6] / R);
        Trans[1, 3] := sin(body[7] / R) * sin(body[6] / R);
        Trans[2, 1] := sin(body[7] / R) * cos(SmOmega / R) + cos(body[7] / R) *
                sin(SmOmega/R) * cos(body[6] / R);
        Trans[2, 2] := -sin(body[7]/R) * sin(SmOmega / R) + cos(body[7] / R) *
                cos(SmOmega/R)* cos(body[6] / R);
        Trans[2, 3] := -cos(body[7] / R) * sin(body[6] / R);
        Trans[3, 1] := sin(SmOmega / R) * sin(body[6] / R);
        Trans[3, 2] := cos(SmOmega / R) * sin(body[6] / R);
        Trans[3, 3] := cos(body[6] / R);
end;                                                    {BuildTransform}

{============================================================}
procedure CoordConvert (var body:planet;a,E,b,nu:extended;var Eclipt,Peri,Trans:matrix;
                        i:integer);

                { This procedure converts from pericentric to ecliptic      }
                { coordinates using the transformation matrix built         }
                { BuildTransform.                                           }

begin                                                   {CoordConvert}
        Peri[i, 1] := (body[5] * (1 - body[4] * body[4])/(1 + body[4] * cos(nu))) * cos(nu);
        Peri[i, 2] := (body[5] * (1 - body[4] * body[4])/(1 + body[4] * cos(nu))) * sin(nu);
```

```
        Peri[i, 3] := 0;
        Eclipt[i, 1] :=Trans[1, 1]*Peri[i, 1]+Trans[1, 2]*Peri[i, 2] + Trans[1, 3]*Peri[i, 3];
        Eclipt[i, 2] := Trans[2, 1]*Peri[i, 1]+Trans[2, 2]*Peri[i, 2]+Trans[2, 3]*Peri[i, 3];
        Eclipt[i, 3] :=Trans[3, 1]*Peri[i, 1]+Trans[3, 2]*Peri[i, 2]+Trans[3, 3 ]* Peri[i, 3];
end;                                                                    {CoordConvert}

{=============================================================}
procedure PlanetIcon1 (x1, y1: extended; var rock1: RgnHandle);

                { This procedure draws icons for the launch planets }

var
        angle, rd, x, y: extended;
        Leftx, Lefty, Rightx, Righty: integer;
        tempRect: Rect;
        response: integer;

begin                                                                   {PlanetIcon}
        angle := 0;
        rd := 0.05;
        x := x1 + rd * cos(angle);
        y := y1 + rd * sin(angle);
        xlast[1] := scra * x1 + scrb;
        ylast[1] := scrc * y1 + scrd;
        GMove(x, y);
        rock1 := NewRgn;
        OpenRgn;
        repeat
                angle := angle + 5 * 0.017453292;
                x := x1 + rd * cos(angle);
                y := y1 + rd * sin(angle);
                GDraw(x, y);
        until (angle >= (6.28));
        CloseRgn(rock1);
        FrameRgn(rock1);
        eraseRgn(rock1);
        InvertRgn(rock1);
end;                                                                    {PlanetIcon}

{=============================================================}
procedure PlanetIcon2 (x1, y1: extended; var rock2: RgnHandle);


var
        angle, rd, x, y: extended;  {angle,radius,horizontal position,vertical position}

begin                                                                   {PlanetIcon2}
        angle := 0;
        rd := 0.05;
        x := x1 + rd * cos(angle);
```

```
        y := y1 + rd * sin(angle);
        xlast[2] := scra * x1 + scrb;
        ylast[2] := scrc * y1 + scrd;
        GMove(x, y);
        rock2 := NewRgn;
        OpenRgn;
        repeat
                angle := angle + 5 * 0.017453292;
                x := x1 + rd * cos(angle);
                y := y1 + rd * sin(angle);
                GDraw(x, y);
        until (angle >= (6.28));
        CloseRgn(rock2);
        InvertRgn(rock2);
end;                                                    {PlanetIcon2}

{=============================================================}
procedure PlanetIcon3 (x1, y1: extended; var rock3: RgnHandle);


var
        angle, rd, x, y: extended;  {angle,radius,horizontal position,vertical position}

begin                                                   {PlanetIcon3}
        angle := 0;
        rd := 0.05;
        x := x1 + rd * cos(angle);
        y := y1 + rd * sin(angle);
        xlast[3] := scra * x1 + scrb;
        ylast[3] := scrc * y1 + scrd;
        GMove(x, y);
        rock3 := NewRgn;
        OpenRgn;
        repeat
                angle := angle + 5 * 0.017453292;
                x := x1 + rd * cos(angle);
                y := y1 + rd * sin(angle);
                GDraw(x, y);
        until (angle >= (6.28));
        CloseRgn(rock3);
        InvertRgn(rock3);
end;                                                    {PlanetIcon3}

{=============================================================}
procedure PlanetIcon4 (x1, y1: extended; var rock4: RgnHandle);


var
        angle, rd, x, y: extended;  {angle,radius,horizontal position,vertical position}

begin                                                   {PlanetIcon4}
```

```
        angle := 0;
        rd := 0.05;
        x := x1 + rd * cos(angle);
        y := y1 + rd * sin(angle);
        xlast[4] := scra * x1 + scrb;
        ylast[4] := scrc * y1 + scrd;
        GMove(x, y);
        rock4 := NewRgn;
        OpenRgn;
        repeat
                angle := angle + 5 * 0.017453292;
                x := x1 + rd * cos(angle);
                y := y1 + rd * sin(angle);
                GDraw(x, y);
        until (angle >= (6.28));
        CloseRgn(rock4);
        InvertRgn(rock4);
end;                                                    {PlanetIcon4}

{============================================================}
procedure ShipIcon1 (x1, y1: extended; var rock5: RgnHandle);

                        { This procedure draws icons for the spacecraft }

var
        angle, rd, x, y: extended;  {angle,radius,horizontal position,vertical position}

begin                                                   {ShipIcon}
        angle := 0;
        rd := 0.03;
        x := x1 + rd * cos(angle);
        y := y1 + rd * sin(angle);
        xlast[5] := scra * x1 + scrb;
        ylast[5] := scrc * y1 + scrd;
        GMove(x, y);
        rock5 := NewRgn;
        OpenRgn;
        repeat
                angle := angle + 5 * 0.017453292;
                x := x1 + rd * cos(angle);
                y := y1 + rd * sin(angle);
                GDraw(x, y);
        until (angle >= (6.28));
        CloseRgn(rock5);
        InvertRgn(rock5);
end;                                                    {ShipIcon}

{============================================================}
procedure GoGraphics (var a: extended; var b, E, nu: extended; i: integer; var body:
                        planet; animate: boolean; var Trans: matrix; ship: boolean);
```

```
var
        Peri,                   {The pericentral coordinate system}
        Eclipt: matrix;         {The ecliptical plane coordinate system projection}
        final: extended;        {The final eccentric anomoly after a 2*pi rotation}

begin                                                           {GoGraphics}
        BuildTransform(body, Trans);
        CoordConvert(body, a, E, b, nu, Eclipt, Peri, Trans, i);
        GMove(Eclipt[i, 1], Eclipt[i, 2]);
        position(E, nu, body[5], body[4], body, ship);
        CoordConvert(body, a, E, b, nu, Eclipt, Peri, Trans, i);
        if (animate = FALSE) then
                Gdraw(Eclipt[i, 1], Eclipt[i, 2]);
        if (animate = TRUE) then
                case i of
                        1:
                                begin                                   {case 1}
                                        InvertRgn(rock1);
                                        OffsetRgn(rock1, trunc((scra * Eclipt[i, 1] + scrb)) -
                                                trunc(xlast[i]), trunc((scrc * Eclipt[i, 2] +
                                                scrd)) - trunc(ylast[i]));
                                        InvertRgn(rock1);
                                        xlast[i] := scra * Eclipt[i, 1] + scrb;
                                        ylast[i] := scrc * Eclipt[i, 2] + scrd;
                                end;                                    {case 1}
                        2:
                                begin                                   {case 2}
                                        InvertRgn(rock2);
                                        OffsetRgn(rock2, trunc((scra * Eclipt[i, 1] + scrb)) -
                                                trunc(xlast[i]), trunc((scrc * Eclipt[i, 2] +
                                                scrd)) - trunc(ylast[i]));
                                        InvertRgn(rock2);
                                        xlast[i] := scra * Eclipt[i, 1] + scrb;
                                        ylast[i] := scrc * Eclipt[i, 2] + scrd;
                                end;                                    {case 2}
                        3:
                                begin                                   {case 3}
                                        InvertRgn(rock3);
                                        OffsetRgn(rock3, trunc((scra * Eclipt[i, 1] + scrb)) -
                                                trunc(xlast[i]), trunc((scrc * Eclipt[i, 2] +
                                                scrd)) - trunc(ylast[i]));
                                        InvertRgn(rock3);
                                        xlast[i] := scra * Eclipt[i, 1] + scrb;
                                        ylast[i] := scrc * Eclipt[i, 2] + scrd;
                                end;                                    {case 3}
                        4:
                                begin                                   {case 4}
                                        InvertRgn(rock4);
```

```
                        OffsetRgn(rock4, trunc((scra * Eclipt[i, 1] + scrb)) -
                              trunc(xlast[i]), trunc((scrc * Eclipt[i, 2] +
                              scrd)) - trunc(ylast[i]));
                        InvertRgn(rock4);
                        xlast[i] := scra * Eclipt[i, 1] + scrb;
                        ylast[i] := scrc * Eclipt[i, 2] + scrd;
                    end;                                                {case 4}
                5:
                    begin                                               {case 5}
                        InvertRgn(rock5);
                        OffsetRgn(rock5, trunc((scra * Eclipt[i, 1] + scrb)) -
                              trunc(xlast[i]), trunc((scrc * Eclipt[i, 2] +
                              scrd)) - trunc(ylast[i]));
                        InvertRgn(rock5);
                        xlast[i] := scra * Eclipt[i, 1] + scrb;
                        ylast[i] := scrc * Eclipt[i, 2] + scrd;
                    end;                                                {case 5}
            end;
end;                                                                    {GoGraphics}

{============================================================}
                {This completes the graphics procedures }
{============================================================}
procedure QuadrantCheck (var Omega: extended; delLong, Lat1, Lat2, L1, L2:extended);

            { This procedure utilizes geometric arguements within      }
            { the celestial sphere to ensure that Omega has been       }
            { assigned in the correct quadrant                         }

begin                                                                   {QuadrantCheck}
    delLong := delLong * R;
    Lat1 := Lat1 * R;
    Lat2 := Lat2 * R;
    L1 := L1 * R;
    L2 := L2 * R;
    if (delLong <= 180) then
        begin                                                           {if #1}
            if ((Lat1 >= 0) and (Lat2 >= 0)) then
                begin                                                   {if #2}
                    if ((L2<=180)and((L2<Omega)and
                      (Omega<(L2+180)))) then
                        Omega := Omega + 180;
                    if ((L2>180) and (((L2<=Omega) and
                      (Omega<=360)) or  ((0<=Omega) and
                      (Omega<(L2+180-360)))))then
                        Omega := Omega + 180;
                end;                                                    {if #2}
            if ((Lat1 <= 0) and (Lat2 <= 0)) then
                begin                                                   {if #3}
                    if ((L2 <= 180) and not ((L2 <= Omega) and
                      (Omega <  (L2 + 180)))) then
```

```
                                Omega := Omega + 180;
                            if ((L2 > 180)and not(((L2<=Omega) and
                              (Omeg<360))or((0<=Omega)and
                              (Omega<(L2+180-360))))) then
                                Omega := Omega + 180;
                    end;                                                {if #3}
                if ((Lat1 > 0) and (Lat2 > 0)) then
                    begin                                               {if #4}
                            if ((L1<=180)and((L1<=Omega)and
                              (Omega<(L1+180)))) then
                                Omega := Omega + 180;
                            if ((L1>180) and (((L1<=Omega) and
                              (Omega<=360)) or ((0 <= Omega) and
                              (Omega < (L1 + 180 - 360))))) then
                                Omega := Omega + 180;
                    end;
                                                                        {if #4}
                if ((Lat1 < 0) and (Lat2 > 0)) then
                    begin                                               {if #5}
                            if((L1<=180) and not((L1<=Omega)
                              and(Omega<(L1 + 180)))) then
                                Omega := Omega + 180;
                            if ((L1>180) and not(((L1<=Omega)and
                              (Omega<= 360)) or ((0<=Omega) and
                              (Omega < (L1 + 180 - 360))))) then
                                Omega := Omega + 180;
                    end;                                                {if #5}
        end                                                             {if #1}
else
        begin                                                           {else}
                if ((Lat2 >= 0) and (Lat1 >= 0)) then
                    begin                                               {if #6}
                            if ((L1<=180)and((L1<Omega)and
                              (Omega<(L1+180)))) then
                                Omega := Omega + 180;
                            if ((L1>180) and (((L1<=Omega) and
                              (Omega<=360)) or ((0<=Omega) and
                              (Omega<(L1 + 180 - 360))))) then
                                Omega := Omega + 180;
                    end;                                                {if #6}
                if ((Lat2 <= 0) and (Lat1 <= 0)) then
                    begin                                               {if #7}
                            if ((L1<=180)and not((L1<=Omega) and
                              (Omega<(L1+ 180)))) then
                                Omega := Omega + 180;
                            if ((L1>180)and not(((L1<=Omega)and
                              (Omega<=360)) or ((0<=Omega) and
                              (Omega<(L1 + 180 - 360))))) then
                                Omega := Omega + 180;
                    end;                                                {if #7}
                if ((Lat2 > 0) and (Lat1 > 0)) then
```

```
                        begin                                                {if #8}
                                if ((L2<=180)and((L2<=Omega) and
                                 (Omega<(L2+180)))) then
                                        Omega := Omega + 180;
                                if ((L2>180) and (((L2<=Omega) and
                                 (Omega<=360)) or ((0<=Omega) and
                                 (Omega<(L2 + 180 - 360))))) then
                                        Omega := Omega + 180;
                        end;                                                 {if #8}
                if ((Lat2 < 0) and (Lat1 > 0)) then
                        begin                                                {if #9}
                                if ((L2<=180) and not ((L2<=Omega) and
                                 (Omega<(L2+ 180)))) then
                                        Omega := Omega + 180;
                                if ((L2>180)and not(((L2<=Omega)and
                                 (Omega<=360))or ((0<=Omega) and
                                 (Omega<(L2 + 180 - 360))))) then
                                        Omega := Omega + 180;
                        end;                                                 {if #9}
        end;                                                                 {else}
end;                                                                 {QuadrantCheck}

{=============================================================}
procedure PieQuadrant (var Pie: extended; CelesLat, long: storage; inclination, Omega:
extended);

var
        PieSin,                         {The sine of the sum of Omega and SmOmega}
        PieCos: extended;       {The cosine of the sum of Omega and SmOmega}

begin                                                                {PieQuadrant}
        PieSin := sin(CelesLat[2]) / sin(inclination);
        if (Celeslat[2] = 0) then
                PieSin := sin(CelesLat[1]) / sin(inclination);
        PieCos := cos(Long[2] - Omega / R) * cos(CelesLat[2]);
        if (Celeslat[2] = 0) then
                PieCos := cos(Long[1] - Omega / R) * cos(CelesLat[1]);
        if ((PieSin > 0) and (PieCos < 0)) then
                Pie := pi - Pie;
        if ((PieSin < 0) and (PieCos < 0)) then
                Pie := pi - Pie;
end;                                                                 {PieQuadrant}

{=============================================================}
procedure Verify (a, dt, H, p, ecc: extended; var eccentric1, eccentric2: extended; var
radius:                         storage; var nu1, nu2: extended);

        {This procedure verifies that the value for nu1 has been assigned    }
        {to the proper quadrant in the case of a nonhyperbolic trajectory.   }
        {The time of flight is solved for using Kepler's equations, and the  }
        {quadrant of nu1 is alterred until this time matches the input time  }
```

```
        {of flight.  nu2 is assigned, dependant upon the value of nu1.        }

var
        time1,                  {initial time since perigee passage}
        time2,                  {final time since perigee passage}
        time,                   {time of flight}
        Period,                 {orbit period}
        w,                      {mean motion}
        arg1,                   {cosine of eccentric anomoly at r1}
        arg2,                   {cosine of eccentric anomoly ar r2}
        check1,
        check2: extended;       {comparison values for time of flight}
        loop: integer;          {an indexing variable}

begin                                                                   {Verify}
        loop := 0;
        nu1 := acos((p / radius[1] - 1) / ecc);
        repeat
                loop := loop + 1;
                case loop of                                            {case}
                        1:      nu1 := nu1;
                        2:      nu1 := 2 * pi - nu1;
                        3:      nu1 := 2 * pi - nu1;
                end;                                                    {case}
                nu2 := nu1 + H;
                if (nu2 > 2 * pi) then
                        nu2 := nu2 - 2 * pi;
                arg1 := (ecc + cos(Nu1)) / (1 + ecc * cos(Nu1));
                arg2 := (ecc + cos(Nu2)) / (1 + ecc * cos(Nu2));
                eccentric1 := acos(arg1);
                eccentric2 := acos(arg2);
                if (Nu1 > pi) then
                        eccentric1 := 2 * pi - eccentric1;
                if (Nu2 > pi) then
                        eccentric2 := 2 * pi - eccentric2;
                Period := (2 * pi * sqrt(a * a * a)) / (sqrt(GM));
                w := 2 * pi / Period;
                time1 := (eccentric1 - ecc * sin(eccentric1)) / w;
                time2 := (eccentric2 - ecc * sin(eccentric2)) / w;
                time := time2 - time1;
                if (eccentric1 > eccentric2) then
                        time := Period + time;
                check1 := time / 3600 / 24;
                check2 := dt / 3600 / 24;
        until ((abs(check1 - check2) < 0.001) or (loop = 3));
        if (loop = 3) then
                writeln('Not!!!!!!!(????)');
end;                                                                    {Verify}

{=============================================================}
```

```
procedure HyperVerify (var nu1, nu2: extended; radius: storage; ecc, H, p, a, dt:
extended);

        { This procedure will use time of flight computations in      }
        { order to verify that the correct sign of nu1 has been       }
        { selected in the case of a hyperbolic trajectory.            }
        { Corrections are made if  they are needed as done in         }
        { Verify, above.  As no hyperbolic cases are considered       }
        { in this study, this procedure has NOT been completed.       }

var
    loop: integer;          {an indexing variable}
    F1,                     {the initial hyperbolic eccentric anomaly}
    F2,                     {the final hyperbolic eccentric anomaly}
    t1,                     {the initial time past perigee}
    t2,                     {the final time past perigee}
    time,                   {the time of flight}
    Numax,                  {The maximum true anomaly possible in the hyperbola}
    arguement: extended; {Holding value for arcsinh terms}

begin                                                          {HyperVerify}
    loop := 0;
    nu1 := pi/ -ArcTan(((p/radius[1]-1)/ecc)/(sqrt(1-((p/radius[1]-1)/ecc*(p radius[1]-
        1)/ecc))));
    Numax := pi / 2 - ArcTan((-1 / ecc) / (sqrt(1 - ((-1 / ecc) * (-1 / ecc)))));
    if ((nu1 + H) > Numax) then
        begin                                                      {if}
            nu1 := -nu1;
            nu2 := nu1 + H;
        end                                                        {if}
    else
        begin                                                      {else}
            repeat
                loop := loop + 1;
                case loop of                                       {case}
                    1:    nu1 := nu1;
                    2:    nu1 := -nu1;
                    3:    nu1 := -nu1;
                end;                                               {case}
                nu2 := nu1 + H;
                arguement:=(sqrt(ecc * ecc - 1) * sin(nu1)) / (1 + ecc *
                    cos(nu1));
                F1 := ln(arguement + sqrt(arguement * arguement + 1));
                t1 := (ecc*((exp(F1) - exp(-F1))/2) -
                    F1)/(sqrt(GM/(abs(a*a*a))));
                arguement := (sqrt(ecc*ecc - 1) * sin(nu2)) / (1 + ecc *
                    cos(nu2));
                F2 := ln(arguement + sqrt(arguement * arguement + 1));
                t2 := (ecc * ((exp(F2)-exp(-F2))/2) -
                    F2)/(sqrt(GM/(abs(a*a*a))));
                time := t2 - t1;
```

```
                until ((abs(dt - time) < 0.00001) or (loop = 3));
                if (loop = 3) then
                        writeln('Not!!!!!!!(????)');
        end;                                                    {else}
end;                                                            {HyperVerify}

{================================================================}
procedure Convert (var date: extended);

        {This procedure converts any date enterred in       }
        {the standard calender into a Julian date.          }


var
        Month,          {The portion of the enterred date that represents the month}
        Day,            {The portion of the enterred date that represents the day}
        Year: extended; {The protion of the enterred date that represents the year}

begin                                                           {Convert}
        Month := trunc(date);
        Day := trunc(100 * (date - Month));
        Year := Round((1000000 * date) - (Month * 1000000) - (Day * 10000));
        if (Month > 2) then
                Month := Month + 1
        else
                begin                                           {else}
                        Month := Month + 13;
                        Year := Year - 1;
                end;                                            else}
        date := (365.25 * Year) - (Year / 100) + (Year / 400) + (30.60001 * Month) + Day
                - 478164 + 2199160;
        date := trunc(date);
end;                                                            {Convert}

{================================================================}
procedure CaseEntry2 (var departure, target: planet; var duration, JlaunchT2, arriveT2,
                JarriveT2, dt, Jcentury1, Jcentury2: extended; var place: name;
                JarriveT: extended);

                { This procedure prompts for the names of the       }
                { target and launch planets and the dates of        }
                { these events.  It also calls the procedure        }
                { ChoosePlanet to read the planetary orbital        }
                { elements and the procedure Convert to             }
                { convert all dates to the Julian calendar.         }

var
        i: integer;

begin                                                           {CaseEntry}
        writeln('Enter the time spent at Mars in days.');
```

```
        readln(duration);
        JlaunchT2 := JarriveT + duration;
        Jcentury1 := (JlaunchT2 - 2415021) / (36525);
        place[1] := 'Mars';
        for i := 1 to 7 do
                departure[i] := 0;
        Mars(departure, Jcentury1);
        writeln('Enter the desired return trip time of flight.');
        readln(arriveT2);
        JarriveT2 := JlaunchT2 + arriveT2;
        Jcentury2 := (JarriveT2 - 2415021) / (36525);
        place[2] := 'earth';
        for i := 1 to 7 do
                target[i] := 0;
        earth(target, Jcentury2);
        dt := (JarriveT2 * 24 * 3600) - (JlaunchT2 * 24 * 3600);
        place[3] := 'Venus';
        for i := 1 to 7 do
                flyby[i] := 0;
        Venus(flyby, Jcentury1);
        place[4] := 'Mercury';
        for i := 1 to 7 do
        other[i] := 0;
        Mercury(other, Jcentury1);
end;                                                    {CaseEntry}

{================================================================}
procedure ComputePositions (var body: planet; time: extended; index: integer; var check:
integer;                        var long, CelesLat, radius, phi, Vplanet, EA, nu,
SMinAx, VPr,                            VPt: storage; var EcliptVPx, EcliptVPy,
EcliptVPz: storage);

                { This procedure finds the positions of each of      }
                { the planets within their orbits on the dates       }
                { specified.                                         }


var
        Trans: matrix;          {a temporary transformation matrix}
        Peri,                   {The pericentral coordinate system}
        Eclipt: matrix;         {The ecliptical plane coordinate system projection}
        n,                      {mean motion}
        M,                      {mean anomoly}
        Mo,                     {storage variable}
        D,                      {slope}
        x,                      {position variable}
        rotations,              {number of body rotational periods since epoch}
        E,                      {Eccentric Anomaly}
        SmOmega,                {arguement of the periapsis}
        q,                      {intersection with the auxilliary circle}
        final,                  {final nu angle}
```

```
        initx,                  {initial x}
        inity,                  {initial y}
        arg: extended;          {a holding variable for arcsin arguement}

begin                                                   {ComputePositions}
        check := check + 1;
        n := body[1];
        q := trunc(n / 360);
        n := n - (q * 360);
        M := pi / 180 * (body[2] - body[3]);
        if (M < 0) then
                M := M + (2 * 3.141592654);
        E := M + body[4] * (sin(M) + body[4] * sin(2 * M) / 2);
        repeat
                Mo := E - body[4] * sin(E);
                D := (M - Mo) / (1 - body[4] * cos(E));
                E := E + D;
        until (abs(D) <= 1E-14);
        nu[index] := 2 * ArcTan(sqrt((1 + body[4])/(1 - body[4])) * ((sin(E/
                2)/cos(E/2))));
        if (nu[index] < 0) then
                nu[index] := nu[index] + 2 * pi;
        SmOmega := (body[3] - body[7]) / R;
        CelesLat[index] := ArcTan((sin(SmOmega + nu[index]) * sin(body[6] / R)) /
                                (sqrt(1 - (sin(SmOmega + nu[index]) * sin(body[6] / R)) *
                                (sin(SmOmega + nu[index]) * sin(body[6] / R)))));
        if body[6] = 0 then
                CelesLat[index] := 0;
        radius[index] := body[5] * (1 - body[4] * body[4])/(1 + body[4] * cos(nu[index]));
        x := radius[index] / body[5];
        phi[index] := pi/2 - ArcTan((sqrt((1 - body[4] * body[4])/(x * (2 - x))))/sqrt(1 -
                                (sqrt((1 - body[4] * body[4]) / (x * (2 - x)))) * (sqrt((1 -
                                body[4] * body[4]) / (x * (2 - x))))));
        Vplanet[index] := sqrt(GM * (2 - x) / radius[index]) * AU;
        if (nu[index] < pi) then
                phi[index] := -phi[index];
        SMinAx[index] := body[5] * sqrt(1 - body[4] * body[4]);
        rotations := E / (2 * pi);
        EA[index] := E - trunc(rotations) * 2 * pi;
        if ((check = 1) or (check = 2)) then
                begin                                           {if #1}
                        BuildTransform(body, Trans);
                        CoordConvert(body, body[5], EA[index], SMinAx[index],
                                                nu[index], Eclipt, Peri, Trans, index);
                        GMove(0, 0);
                        Gdraw(Eclipt[index, 1], Eclipt[index, 2]);
                end;                                            {if #1}
        if (check = 1) then
                begin                                           {if #2}
                        initx := Eclipt[index, 1];
                        inity := Eclipt[index, 2];
```

```
                GMove(Eclipt[index, 1], Eclipt[index, 2]);
                final := nu[index] + 2 * pi;
                repeat
                        nu[index] := nu[index] + 5 * 0.017453292;
                        CoordConvert(body, body[5], EA[index],SMinAx[index],
                                nu[index], Eclipt, Peri, Trans, index);
                        Gdraw(Eclipt[index, 1], Eclipt[index, 2]);
                until (nu[index] >= final);
                Eclipt[index, 1] := initx;
                Eclipt[index, 2] := inity;
                xlast[1] := Eclipt[index, 1];
                ylast[1] := Eclipt[index, 2];
                PlanetIcon1(Eclipt[index, 1], Eclipt[index, 2], rock1);
                Eclipt[5, 1] := Eclipt[index, 1];
                Eclipt[5, 2] := Eclipt[index, 2];
                ShipIcon1(Eclipt[5, 1], Eclipt[5, 2], rock5);
                InvertRgn(Rock5);
        end;                                                    {if #2}
if (check = 5) then
        begin                                                   {if #3}
                BuildTransform(body, Trans);
                CoordConvert(body, body[5], EA[index], SMinAx[index],
                                nu[index], Eclipt, Peri, Trans, index);
                initx := Eclipt[index, 1];
                inity := Eclipt[index, 2];
                GMove(Eclipt[index, 1], Eclipt[index, 2]);
                final := nu[index] + 2 * pi;
                repeat
                        nu[index] := nu[index] + 5 * 0.017453292;
                        CoordConvert(body, body[5], EA[index],SMinAx[index],
                                nu[index], Eclipt, Peri, Trans, index);
                        Gdraw(Eclipt[index, 1], Eclipt[index, 2]);
                until (nu[index] >= final);
                Eclipt[index, 1] := initx;
                Eclipt[index, 2] := inity;
                xlast[2] := Eclipt[index, 1];
                ylast[2] := Eclipt[index, 2];
                PlanetIcon2(Eclipt[index, 1], Eclipt[index, 2], rock2);
        end;                                                    {if #3}
if (check = 3) then
        begin                                                   {if #4}
                BuildTransform(body, Trans);
                CoordConvert(body, body[5], EA[index], SMinAx[index],
                                nu[index], Eclipt, Peri, Trans, index);
                initx := Eclipt[index, 1];
                inity := Eclipt[index, 2];
                GMove(Eclipt[index, 1], Eclipt[index, 2]);
                final := nu[index] + 2 * pi;
                repeat
                        nu[index] := nu[index] + 5 * 0.017453292;
                        CoordConvert(body, body[5], EA[index],SMinAx[index],
```

```
                                        nu[index], Eclipt, Peri, Trans, index);
                        Gdraw(Eclipt[index, 1], Eclipt[index, 2]);
                until (nu[index] >= final);
                Eclipt[index, 1] := initx;
                Eclipt[index, 2] := inity;
                xlast[3] := Eclipt[index, 1];
                ylast[3] := Eclipt[index, 2];
                PlanetIcon3(Eclipt[index, 1], Eclipt[index, 2], rock3);
        end;                                                    {if #4}
    if (check = 4) then
        begin                                                   {if #5}
                BuildTransform(body, Trans);
                CoordConvert(body, body[5], EA[index], SMinAx[index],
                                nu[index], Eclipt, Peri, Trans, index);
                initx := Eclipt[index, 1];
                inity := Eclipt[index, 2];
                GMove(Eclipt[index, 1], Eclipt[index, 2]);
                final := nu[index] + 2 * pi;
                repeat
                        nu[index] := nu[index] + 5 * 0.017453292;
                        CoordConvert(body, body[5], EA[index],SMinAx[index],
                                nu[index], Eclipt, Peri, Trans, index);
                        Gdraw(Eclipt[index, 1], Eclipt[index, 2]);
                until (nu[index] >= final);
                Eclipt[index, 1] := initx;
                Eclipt[index, 2] := inity;
                xlast[4] := Eclipt[index, 1];
                ylast[4] := Eclipt[index, 2];
                PlanetIcon4(Eclipt[index, 1], Eclipt[index, 2], rock4);
        end;                                                    {if #5}
    long[index] := ArcTan(Eclipt[index, 2] / Eclipt[index, 1]);
    if ((Eclipt[index, 1] < 0) and (Eclipt[index, 2] > 0) and (long[index] < 0)) then
        long[index] := long[index] + pi;
    if ((Eclipt[index, 1] < 0) and (Eclipt[index, 2] < 0) and (long[index] > 0)) then
        long[index] := long[index] + pi;
    if ((Eclipt[index, 1] > 0) and (Eclipt[index, 2] < 0) and (long[index] < 0)) then
        long[index] := long[index] + 2 * pi;
    BuildTransform(body, Trans);
    VPx[index] := (-body[5] * sin(E) * sqrt(GM) / (sqrt(body[5] * body[5] * body[5])
                * (1 - body[4] * cos(E)))) * AU;
    VPy[index] := (SMinAx[index] * cos(E) * sqrt(GM) / (sqrt(body[5] * body[5] *
                        body[5]) * (1 - body[4] * cos(E)))) * AU;
    EcliptVPx[index] := Trans[1, 1] * VPx[index] + Trans[1, 2] * VPy[index];
    EcliptVPy[index] := Trans[2, 1] * VPx[index] + Trans[2, 2] * VPy[index];
    EcliptVPz[index] := Trans[3, 1] * VPx[index] + Trans[3, 2] * VPy[index];

end;                                                    {ComputePositions}

{=============================================================}
procedure Parameters (var H, c, s, Rop, m, L, e1, e2: extended; var CelesLat, long:
                storage; radius: storage);
```

```
                { This procedure defines several parameters depending      }
                { upon whether the problem being solved is that for a      }
                { body in heliocentric or geocentric orbit. The distance   }
                { units are astronomical units or kilometers, respectively. }

var
        x, y, z: storage;                       {planetary position in the ecliptic}
        H_One,                                  {preliminary HCA variable}
        k0, k1, k2, k3, k4, k5, k6, k7: extended;   {computational parameters}

begin                                                           {Parameters}
        x[1] := cos(CelesLat[1]) * cos(long[1]);
        x[2] := cos(CelesLat[2]) * cos(long[2]);
        y[1] := cos(CelesLat[1]) * sin(long[1]);
        y[2] := cos(CelesLat[2]) * sin(long[2]);
        z[1] := sin(CelesLat[1]);
        z[2] := sin(CelesLat[2]);
        inclination := arctan((z[2]-z[1])/(sqrt((x[2]-x[1])*(x[2]-x[1]))+((y[2] -
                y[1])*(y[2]-y[1]))));
        H_One := pi / 2 - ArcTan((x[1] * x[2] + y[1] * y[2] + z[1] * z[2]) / (sqrt(1 -
                sqr(x[1] * x[2] + y[1] * y[2] + z[1] * z[2]))));
        if ((sin(long[2] - long[1])) >= 0) then
                H := H_One
        else
                H := 2 * pi - H_One;
        c := sqrt(radius[1] * radius[1] + radius[2] * radius[2] - 2 * radius[1] * radius[2] *
                cos(H));
        s := (radius[1] + radius[2] + c) / 2;
        Rop := (radius[1] + radius[2] + 2 * sqrt(radius[1] * radius[2]) * cos(H / 2)) / 4;
        m := (GM * (dt) * (dt)) / (8 * Rop * Rop * Rop);
        k0 := radius[2] / radius[1];
        k1 := sqrt(k0);
        k2 := (k0 - 1);
        k3 := k2 * k2 / (4 * (k1 + k0 * (2 + k1)));
        k4 := (sin(H / 4)) * (sin(H / 4)) + k3;
        k5 := (cos(H / 4)) * (cos(H / 4)) + k3;
        k6 := cos(H / 2);
        k7 := 4 * k0 * (sin(H / 2) * sin(H / 2));
        e1 := k2 * k2;
        e2 := k7;
        if ((0 < H) and (H < pi)) then
                L := k4 / (k4 + k6)
        else
                L := (k5 - k6) / k5;
end;                                                            {Parameters}

{=============================================================}
procedure XiFunction (x: extended; var Xi: extended);

                { This procedure uses the continued fraction        }
```

```
                { scheme to approximate the function Xi.          }

var
        eta, c, c2, A, A1, A2, B,
        B1, B2, d, Xilast: extended;  {calculation parameters}
        counter: integer;             {counts the number of repeat loop iterations}
        exit: boolean;                {determines when to exit the procedure}


begin                                                          {XiFunction}
        eta := x / ((sqrt(1 + x) + 1) * (sqrt(1 + x) + 1));
        c2 := 8 * (sqrt(1 + x) + 1);
        A1 := 5 + eta;
        B1 := 1;
        A2 := 1;
        B2 := 0;
        counter := 0;
        exit := FALSE;
        Xi := 0;
        repeat
                counter := counter + 1;
                if (counter = 1) then
                        c := 9 * eta / 7
                else
                        c :=eta*((counter+2)*(counter+2))/((2*counter+5)*(2*counter+3));
                d := 1;
                A := d * A1 + c * A2;
                B := d * B1 + c * B2;
                A1 := A1 / B;
                B1 := B1 / B;
                A := A / B;
                A2 := A1;
                B2 := B1;
                A1 := A;
                B1 := 1;
                Xilast := Xi;
                Xi := A;
                if (abs(Xi) = abs(Xilast)) then
                        begin                                          {if}
                                Xi := c2 / (3 + (1 / Xi));
                                exit := TRUE;
                        end;                                           {if}
        until ((counter = 100) or (exit = TRUE));
end;                                                           {XiFunction}

{================================================================}
procedure KFunction (u: extended; var K: extended);

                { This procedure uses a continued fraction scheme to   }
                { approximate the function K. This function is in turn  }
                { used to solve the cubic equation of y.                }
```

```
var
        exit: boolean;                  {determines when to exit procedure}
        a, c, d, Klast: extended;       {calculation parameters}
        coefficient: extended;          {coefficient of u in the continued fraction}
        n: extended;                    {the independant variable in the coeff eqn's}
        counter: integer;               {counts the number of iterations}
        index: extended;                {half-step counter that advances n}

begin                                                                   {KFunction}
        a := 1 / 3;
        d := 1;
        c := a;
        K := c;
        index := 0;
        exit := FALSE;
        counter := -1;
        repeat
                counter := counter + 1;
                index := index + 0.5;
                n := trunc(index);
                if ((counter mod 2) = 0) then
                        coefficient := (2 *(3*n + 2)*(6 * n + 1))/(9*(4*n + 1)*(4 n + 3))
                else
                        coefficient := (2*(3 * n + 1)*(6 * n - 1))/(9*(4 * n - 1)*(4*n + 1));
                a := coefficient * u;
                d := 1 / (1 + a * d);
                c := c * (d - 1);
                Klast := K;
                K := K + c;
                if ((abs(K) = abs(Klast)) or (n = 100)) then
                        exit := TRUE;
        until (exit = TRUE);
end;                                                                    {KFunction}

{=================================================================}
procedure Cubic (Xi, L, m: extended; var y, K: extended);

                { This procedure calls the KFunction procedure      }
                { in order to solve the cubic equation for y.       }

var
        u, B, h1, h2: extended;         {calculation parameters}

begin                                                                   {Cubic}
        h1 :=((L + x)*(L + x)*(1 + 3 * x + Xi)) / ((1 + 2 * x + L)*(4 * x + Xi * (3 + x)));
        h2 := (m * (x - L + Xi)) / ((1 + 2 * x + L) * (4 * x + Xi * (3 + x)));
        B := (27 * h2) / (4 * (1 + h1) * (1 + h1) * (1 + h1));
        if ((1 + B) < 0) then
                writeln('These dates crash the program. Ignore the output data.');
        if ((1 + B) > 0) then
```

```
            begin                                                    {if}
                  u := B / (2 * (sqrt(1 + B) + 1));
                  KFunction(u, K);
                  y := ((1 + h1) / 3) * (2 + (sqrt(1 + B)) / (1 + 2 * u * K * K));
            end;                                                     {if}
end;                                                                 {Cubic}

{=================================================================}
procedure TrajectoryComputations (var x, Xi, K, dt, H, c, s, Rop, m, L, e1, e2,
                                  inclination:extended;var radius,CelesLat,long: storage);


var
      xlast,          {Holding variable for x}
      y: extended;    {Battin dependant variable}

begin                                                    {TrajectoryComputations}
      x := 100;
      Parameters(H, c, s, Rop, m, L, e1, e2, CelesLat, long, radius);
      repeat
            XiFunction(x, Xi);
            Cubic(Xi, L, m, y, K);
            xlast := x;
            x := sqrt(((1 - L) / 2) * ((1 - l) / 2) + m / (y * y)) - (1 + L) / 2;
      until ((abs(x - xlast)) < 1.0E-15);
end;                                                     {TrajectoryComputations}
{=================================================================}
procedure ComputeOrbit (var L, a, ecc, eccentric1, eccentric2, p, angmom, n, gamma1,
                        gamma2, COne, CTwo, nu1, nu2: extended; Rop, e1, e2, c, s, H,
                        dt, inclination: extended; var radius, SMinAx, CelesLat, long:
                        storage; var spacecraft: planet; var Trans: matrix; var EcliptVpx,
                        EcliptVpy, EcliptVpz: storage);

                              { This procedure calculates the orbital      }
                              { parameters of the spacecraft               }

var
      V1t,              {Tangential velocity at departure}
      V2t,              {Tangential velocity at arrival}
      V1r,              {Radial velocity at departure}
      V2r,              {Radial velocity at arrival}
      V1x,              {x- axis velocity at departure}
      V1y,              {y-axis velocity at departure}
      V2x,              {x-axis velocity at arrival}
      V2y,              {y-axis velocity at arrival}
      Vinfinity,        {Hyperbolic excess velocity}
      SmOmega,          {arguement of the periapsis for the spacecraft}
      NuAlt,            {true anomaly reduced to a single quadrant}
      LongAlt,          {longitude}
      delLong,          {difference :  longitudes of launch and target planets}
      delLat,           {difference:  latitudes of launch and target planets}
```

```
        snapperhead,        {Hypotenuse of spherical triangle}
        Period: extended;   {The period of the orbit ('nuff said!)}
        descend: boolean;   {checks if orbit climbs or descends in CelesLat}
        i, j: integer;      {indexing variables}
        EcliptVel: matrix;  {velocity in the ecliptic plane}

begin                                                          {ComputeOrbit}
        descend := FALSE;
        j := 1;
        a := Rop * (1 + x) * (L + x) / (2 * x);                {au}
        ecc := sqrt((e1 + e2 * ((L - x) / (L + x)) *
                ((L - x) / (L + x))) / (e1 + e2));             {unitless}
        p := a * (1 - ecc * ecc);                              {au}
        angmom := sqrt(GM * p) * AU * AU;                      {km2/s}
        n := sqrt((GM) / abs(a * a * a));                      {rad/s}
        VOne := sqrt(GM * (2 / radius[1] - 1 / a)) * AU;       {km/s}
        VTwo := sqrt(GM * (2 / radius[2] - 1 / a)) * AU;       {km/s}
        V1t := angmom / (radius[1] * AU);                      {km/s}
        V2t := angmom / (radius[2] * AU);                      {km/s}
        V1r := sqrt(VOne * VOne - V1t * V1t);                  {km/s}
        V2r := sqrt(VTwo * VTwo - V2t * V2t);                  {km/s}
        gamma1 := arctan(V1r / V1t);                           {rad}
        gamma2 := arctan(V2r / V2t);                           {rad}
        if (a > 0) then
                begin                                                  {if #1}
                        Period := n / (2 * pi);                        {s}
                        Verify(a, dt, H, p, ecc, eccentric1, eccentric2, radius, nu1, nu2);
                        if (nu1 > pi) then
                                gamma1 := -gamma1;
                        if (nu2 > pi) then
                                gamma2 := -gamma2;
                end;
                                                                       {if #1}
        if (a < 0) then
                begin                                                  {if #2}
                        writeln('This is a hyperbolic orbit');
                        Hyperverify(nu1, nu2, radius, ecc, H, p, a, dt);
                        if (nu1 < 0) then
                                gamma1 := -gamma1;
                        if (nu2 < 0) then
                                gamma2 := -gamma2;
                        Vinfinity := sqrt(-GM / a) * AU;               {km/s}
                                writeln('V infinity = ', Vinfinity : 12 : 4, ' km/s' : 5);
                end;                                                   {if #2}
        spacecraft[4] := ecc;
        spacecraft[5] := a;
        delLong := long[2] - long[1];
        spacecraft[7] := ArcTan((((tan(Celeslat[1]) / tan(Celeslat[2])) * sin(long[2])) -
                                sin(long[1])) / (((tan(Celeslat[1]) / tan(Celeslat[2])) *
                                cos(long[2])) - cos(long[1])));
        if (Celeslat[2] = 0) then
```

```
            spacecraft[7] := ArcTan((((tan(Celeslat[2]) / tan(Celeslat[1])) *
                        sin(long[1])) - sin(long[2])) / (((tan(Celeslat[2]) /
                        tan(Celeslat[1])) * cos(long[1])) - cos(long[2])));
        spacecraft[7] := spacecraft[7] * R;
        if (spacecraft[7] < 0) then
            spacecraft[7] := spacecraft[7] + 360;
        QuadrantCheck(spacecraft[7], delLong, CelesLat[1], CelesLat[2], long[1],
                long[2]);
        if (spacecraft[7] > 360) then
            spacecraft[7] := spacecraft[7] - 360;
        if (Celeslat[j] = 0) then
            j := 2;
        spacecraft[6] := ArcTan((tan(Celeslat[j])) / (sin(long[j] - spacecraft[7] / R)));
        SMinAx[5] := spacecraft[5] * sqrt(1 - spacecraft[4] * spacecraft[4]);
        snapperhead := asin(sin(CelesLat[j]) / sin(spacecraft[6]));
        PieQuadrant(snapperhead, CelesLat, long, spacecraft[6], spacecraft[7]);
        if (snapperhead < 0) then
            snapperhead := snapperhead + 2 * pi;
        spacecraft[3] := snapperhead * R - nu2 * R + spacecraft[7];
        if CelesLat[2] = 0 then
            spacecraft[3] := snapperhead * R - nu1 * R + spacecraft[7];
        if (spacecraft[6] < 0) then
            spacecraft[6] := -spacecraft[6];
        spacecraft[6] := spacecraft[6] * R;
        V1x := (-a * sin(eccentric1)*sqrt(GM)/(sqrt(a * a * a) * (1 - ecc *
            cos(eccentric1)))) * AU;
        V1y := (SMinAx[5]*cos(eccentric1)*sqrt(GM)/(sqrt(a*a*a)*(1-
            ecc*cos(eccentric1))))* AU;
        V2x := (-a * sin(eccentric2)*sqrt(GM)/(sqrt(a * a * a) * (1 - ecc *
            cos(eccentric2)))) * AU;
        V2y := (SMinAx[5]*cos(eccentric2) * sqrt(GM)/(sqrt(a*a*a)*(1 -
            ecc*cos(eccentric2))))* AU;
        BuildTransform(spacecraft, Trans);
        EcliptVel[1, 1] := Trans[1, 1] * V1x + Trans[1, 2] * V1y;
        EcliptVel[1, 2] := Trans[2, 1] * V1x + Trans[2, 2] * V1y;
        EcliptVel[1, 3] := Trans[3, 1] * V1x + Trans[3, 2] * V1y;
        EcliptVel[2, 1] := Trans[1, 1] * V2x + Trans[1, 2] * V2y;
        EcliptVel[2, 2] := Trans[2, 1] * V2x + Trans[2, 2] * V2y;
        EcliptVel[2, 3] := Trans[3, 1] * V2x + Trans[3, 2] * V2y;
        COne := (EcliptVel[1, 1]-EcliptVPx[1])*(EcliptVel[1, 1] EcliptVPx[1]) +
            (EcliptVel[1, 2]-EcliptVPy[1])*(EcliptVel[1, 2]EcliptVPy[1])+
            (EcliptVel[1, 3]-EcliptVPz[1])*(EcliptVel[1, 3]- EcliptVPz[1]);
        CTwo := (EcliptVel[2, 1]-EcliptVPx[2])*(EcliptVel[2, 1]-EcliptVPx[2])+
            (EcliptVel[2, 2] -EcliptVPy[2])*(EcliptVel[2, 2]-EcliptVPy[2])+
            (EcliptVel[2, 3]-EcliptVPz[2])*(EcliptVel[2, 3]- EcliptVPz[2]);
end;                                                        {ComputeOrbit}

{============================================================}
procedure GiveOutput2 (var place: name; var H, duration, JlaunchT, arriveT, JarriveT2,
                    VOne, VTwo, COne, CTwo, ZOne, ZTwo, a, ecc, p, angmom, n,
                    nu1, nu2: extended; radius: storage; spacecraft: planet);
```

```
                    { This procedure writes the output to the screen      }
                    { and to the external file, data.                     }

begin                                                                {GiveOutput2}
        writeln;
        writeln;
        writeln('Departure Planet:  ', place[1]);
        writeln('Mission duration:  ', duration : 1 : 6);
        writeln('Julian Launch Date = ', JlaunchT : 1 : 0);
        writeln('R1 = ', radius[1]:1:3, ' AU              Long = ', Long[1] * R:1:3, 'º');
        writeln('Launch True Anomaly = ', R * Nu1 : 1 : 6, 'º');
        writeln('V1 = ', VOne : 1 : 3, ' km/s' : 5, '  F1 = ', R * ZOne : 1 : 3, 'º');
        writeln('C3 = ', COne : 1 : 3, ' (km/sec)^2              DeltaV1 = ', DeltaV1 : 1 : 4);
        writeln;
        writeln('HELIOCENTRIC CONIC:');
        writeln('a = ', a : 1 : 3, '  e = ', ecc : 1 : 4);
        writeln('HCA = ', R * H : 1 : 2);
        writeln('p = ', p :1:4, ' h = ', angmom: 1:4,' n = ',n*180/pi*24*3600: 1:4, 'º/day');
        writeln('sc omega = ', spacecraft[7] : 1 : 6);
        writeln('sc pi = ', spacecraft[3] : 1 : 6);
        writeln('sc SmOmega = ', spacecraft[3] - spacecraft[7] : 1 : 6);
        writeln('sc inclination = ', spacecraft[6] : 1 : 6);
        writeln;
        writeln('Arrival Planet:  ', place[2]);
        writeln('Return Time of Flight:  ', arriveT : 1 : 6);
        writeln('Julian Arrival Date:  ', JarriveT2 : 1 : 0);
        writeln('R2 = ', radius[2]:1:3,' AU                    Long = ',Long[2]*R:1:3, 'º');
        writeln('Arrival True Anomaly = ', R * Nu2 : 1 : 6, 'º');
        writeln('V2 = ', VTwo : 1 : 3, ' km/s', '  F2 = ', R * ZTwo : 1 : 3, 'º');
        writeln('C3 = ', CTwo : 1 : 3, ' (km/sec)^2              DeltaV2 = ', DeltaV2 : 1 : 4);
end;                                                                 {GiveOutput2}

{=============================================================}
procedure ReturnTrip (var JarriveT: extended);

                    { This procedure repeats the steps of the      }
                    { outbound case for the return trip.           }
var
        j: integer;

begin                                                                {ReturnTrip}
        check := 0;
        CaseEntry2(departure, target, duration, JlaunchT2, arriveT2, JarriveT2, dt,
                Jcentury1, Jcentury2, place, JarriveT);
        InvertRgn(rock1);
        InvertRgn(rock2);
        InvertRgn(rock3);
        InvertRgn(rock4);
```

```
ComputePositions(departure, JlaunchT2, 1, check, long, CelesLat, radius, phi,
                 VPlanet, E, nu, SMinAx, VPx, VPy, EcliptVPx, EcliptVPy,
                 EcliptVPz);
ComputePositions(target, JarriveT2, 2, check, long, CelesLat, radius, phi,
                 VPlanet, E, nu, SMinAx, VPx, VPy, EcliptVPx, EcliptVPy,
                 EcliptVPz);
ComputePositions(flyby, JlaunchT2, 3, check, long, CelesLat, radius, phi,
                 Vplanet, E, nu, SMinAx, VPx, VPy, EcliptVPx, EcliptVPy,
                 EcliptVPz);
ComputePositions(other, JlaunchT2, 4, check, long, CelesLat, radius, phi,
                 Vplanet, E, nu, SMinAx, VPx, VPy, EcliptVPx, EcliptVPy,
                 EcliptVPz);
TrajectoryComputations(x, Xi, K, dt, H, c, s, Rop, m, L, e1, e2, inclination,
                 radius, CelesLat, long);
ComputeOrbit(L, a, ecc, eccentric1, eccentric2, p, angmom, n, gamma1, gamma2,
                 COne, CTwo, nu1, nu2, Rop, e1, e2, c, s, H, dt, inclination, radius,
                 SMinAx, CelesLat, long, spacecraft, Trans, EcliptVpx, EcliptVpy,
                 EcliptVpz);
MarsParking(Rpark1, COne, DeltaV1);
EarthParking(Rpark2, CTwo, DeltaV2);
GiveOutput2(place, H, duration, JlaunchT2, arriveT2, JarriveT2, VOne, VTwo,
                 COne, CTwo, gamma1, gamma2, a, ecc, p, angmom, n, nu1, nu2,
                 radius, spacecraft);
place[2] := 'earth';
for j := 1 to 7 do
        target[j] := 0;
earth(target, Jcentury1);
ComputePositions(target, JlaunchT, 2, check, long, CelesLat, radius, phi, VPlanet,
                 E, nu, SMinAx, VPx, VPy, EcliptVPx, EcliptVPy, EcliptVPz);
eccalt := eccentric1;
nualt := nu1;
for j := 1 to 7 do
        path[j] := spacecraft[j];
SMinAx[6] := SMinAx[5];
Days := JlaunchT2;
repeat
        Days := Days + stepsize;
        GoGraphics(path[5],SMinAx[6],eccalt,nualt,6,path,FALSE, Trans,TRUE);
until (Days >= JarriveT2);
Days := Jlauncht2;
repeat
        Days := Days + stepsize;
        Jcentury := (Days - 2415021) / (36525);
        GoGraphics(spacecraft[5],SMinAx[5],eccentric1,nu1,5,spacecraft,TRUE,
                 Trans, TRUE);
        Mars(departure, Jcentury);
        GoGraphics(departure[5], SMinAx[1], E[1], nu[1], 1, departure, TRUE,
                 Trans, FALSE);
        earth(target, Jcentury);
        GoGraphics(target[5], SMinAx[2], E[2], nu[2], 2, target, TRUE, Trans,
                 FALSE);
```

```
                Venus(flyby, Jcentury);
                GoGraphics(flyby[5], SMinAx[3], E[3], nu[3], 3, flyby, TRUE, Trans,
                        FALSE);
                Mercury(other, Jcentury);
                GoGraphics(other[5], SMinAx[4], E[4], nu[4], 4, other, TRUE, Trans,
                        FALSE);
        until (Days >= JarriveT2);
end;                                                            {ReturnTrip}

{*************************************************************************}
{***************************** Main Program ******************************}
{*************************************************************************}

begin                                                           {Main}
        Gwindow1(10, 10, 350, 350, 'x,y');
        Gscale(-2, 2, -2, 2);
        Gxaxis(0, 1, 0);
        Gyaxis(0, 1, 0);
        check := 0;
        CaseEntry1(departure, target, launchT, JlaunchT, arriveT, JarriveT, dt, Jcentury1,
                Jcentury2, place);
        ComputePositions(departure, JlaunchT, 1, check, long, CelesLat, radius, phi,
                VPlanet, E, nu, SMinAx, VPx, VPy, EcliptVPx, EcliptVPy, EcliptVPz);
        ComputePositions(target, JarriveT2, 2, check, long, CelesLat, radius, phi,
                VPlanet, E, nu, SMinAx, VPx, VPy, EcliptVPx, EcliptVPy, EcliptVPz);
        ComputePositions(flyby, JlaunchT, 3, check, long, CelesLat, radius, phi, Vplanet,
                E, nu, SMinAx, VPx, VPy, EcliptVPx, EcliptVPy, EcliptVPz);
        ComputePositions(other, JlaunchT, 4, check, long, CelesLat, radius, phi, Vplanet,
                E, nu, SMinAx, VPx, VPy, EcliptVPx, EcliptVPy, EcliptVPz);
        TrajectoryComputations(x, Xi, K, dt, H, c, s, Rop, m, L, e1, e2, inclination,
                radius, CelesLat, long);
        ComputeOrbit(L, a, ecc, eccentric1, eccentric2, p, angmom, n, gamma1, gamma2,
                COne, CTwo, nu1, nu2, Rop, e1, e2, c, s, H, dt, inclination, radius,
                SMinAx, CelesLat, long, spacecraft, Trans, EcliptVpx, EcliptVpy,
                EcliptVpz);
        EarthParking(Rpark1, COne, DeltaV1);
        MarsParking(Rpark2, CTwo, DeltaV2);
        GiveOutput1(place, H, launchT, JlaunchT, arriveT, JarriveT, VOne, VTwo,
                COne, CTwo, gamma1, gamma2, a, ecc, p, angmom, n, nu1, nu2,
                DeltaV1, DeltaV2, radius, Long, spacecraft);
        place[2] := 'Mars';
        for i := 1 to 7 do
                target[i] := 0;
        Mars(target, Jcentury1);
        ComputePositions(target, JarriveT2, 2, check, long, CelesLat, radius, phi,
                VPlanet, E, nu, SMinAx, VPx, VPy, EcliptVPx, EcliptVPy, EcliptVPz);
        eccalt := eccentric1;
        nualt := nu1;
        for i := 1 to 7 do
                path[i] := spacecraft[i];
        SMinAx[6] := SMinAx[5];
```

```
        Days := JlaunchT;
        repeat
                Days := Days + stepsize;
                GoGraphics(path[5],SMinAx[6],eccalt,nualt,6,path,FALSE,Trans,TRUE);
        until (Days >= JarriveT);
        Days := JlaunchT;
        repeat
                Days := Days + stepsize;
                Jcentury := (Days - 2415021) / (36525);
                GoGraphics(spacecraft[5], SMinAx[5], eccentric1, nu1, 5, spacecraft,
                        TRUE, Trans, TRUE);
                earth(departure, Jcentury);
                GoGraphics(departure[5], SMinAx[1], E[1], nu[1], 1, departure, TRUE,
                        Trans, FALSE);
                Mars(target, Jcentury);
                GoGraphics(target[5],SMinAx[2],E[2],nu[2],2,target,TRUE,Trans,
                        FALSE);
                Venus(flyby, Jcentury);
                GoGraphics(flyby[5], SMinAx[3], E[3], nu[3], 3, flyby, TRUE, Trans,
                        FALSE);
                Mercury(other, Jcentury);
                GoGraphics(other[5], SMinAx[4], E[4], nu[4], 4, other, TRUE, Trans,
                        FALSE);
        until (Days >= JarriveT);
        SaveDrawing('orbit');
        ReturnTrip(JarriveT);
        Gpause;
end.                                                                    {Main}
```

Appendix B: Solar Sail Program

```
program Grod_Sail (input, output, SSail3dOut, SMom3dOut);
{_____}
{**********************************************************************}
{Program: 3-D Solar Sail Program                                       }
{                                                                      }
{By:    Brian K. Funk, A. Allen McCarley, H. Dan Thomas                }
{                                                                      }
{Date:  11/23/92                                                       }
{                                                                      }
{Last Revised:  11/23/92                                               }
{                                                                      }
{Description:       This program uses the calculus of variations and Pontryagin's }
{                   Maximum Principle to calculate the minimum time Solar Sail 3-D }
{                   trajectory between the earth and Mars.             }
{**********************************************************************}
{----------------------------------------------------------------------------------------}
type
        vec = array[1..20] of extended;
        mat = array[1..20,1..20] of extended;

const   {Astronautical constants taken from U.S.A.F. Academy text}
        AU = 1.4959965E+8;                  {km/au}
        GM = 3.96396415708E-14;             {au3/s2}
        GMearth = 398600;                   {km3/s2}
        RAD = 57.29577951;
        dt = 0.005;
        pi = 3.1415926535897;
        inf = 1e+15;

var     JlaunchT,       {The launch date on the Julian calender}
        Tfinal,         {The date of arrival at the target planet}
        tstar,          {Non-dimensionalizing factor}
        JarriveT,       {Tfinal in the Julian calender}
        Jcentury1,      {Launch date in Julian Centuries}
        Jcentury2,      {Arrival date in Julian Centuries}
        Vr,             {Radial velocity}
        Vphi,           {Tangential velocity}
        Vpsi,           {Normal velocity}
        r,              {Radius}
        Phi,            {Celestial Latitude in 2-D spherical}
        Psi,            {Celestial Longitude in 2-D spherical}
        PVr,            {Lagrange multiplier conjugate to Vr}
        PVphi,          {Lagrange multiplier conjugate to Vphi}
        PVpsi,          {Lagrange multiplier conjugate to Vpsi}
        Pr,             {Lagrange multiplier conjugate to radius}
        Pphi,           {Lagrange multiplier conjugate to Phi}
        Ppsi,           {Lagrange multiplier conjugate to Psi}
        rearth,         {Initial radius of earth}
```

```
        rfinal,          {Final radius}
        Phiearth,        {Initial Phi of earth}
        Phifinal,        {Final Phi of solar sail}
        Psiearth,        {Initial Psi of earth}
        Psifinal,        {Final Psi}
        Vrearth,         {Initial Radial Velocity of earth}
        Vrfinal,         {Final Radial Velocity}
        Vphiearth,       {Initial Tangental Velocity of earth}
        Vphifinal,       {Final Tangental Velocity}
        Vpsiearth,       {Initial Normal Velocity of earth}
        Vpsifinal,       {Final Normal Velocity}
        Theta,           {The sail thrust cone angle}
        Beta,            {The sail thrust clock angle}
        Hamil,           {The Hamiltonian}
        t,               {Time}
        inc,             {earth's tilt}
        Parki,           {Inclination of Cape Canaveral parking orbit}
        rPark,           {Parking orbit radius}
        VcPark,          {Parking orbit velocity}
        Vpara,           {Parabolic velocity}
        DeltaV,          {Velocity increment required for parabolic escape}
        Caldate,         {Launch date in calendar form}
        ao: extended;    {Characteristic acceleration}
        f1,              {External file reference variable}
        f2: text;        {External file reference variable}
        departure,       {Stores the orbital elements of the departure planet}
        target,          {Stores the orbital elements of the target planet}
        Phiplanet,       {Celestial Latitude of the planets}
        Psiplanet,       {Celestial Longitude of the planets}
        Rplanet,         {Radius of the Planets}
        Vrplanet,        {Radial Velocity of the Planets}
        Vphiplanet,      {Tangential Velocity of the Planets}
        Vpsiplanet,      {Normal Velocity of the Planets}
        x,               {Initial guess conditions}
        y: vec;          {Final conditions to be met}
        k1: integer;
        doit: boolean;

{*************************************************************************}
{************************** FUNCTIONS ************************************}
{*************************************************************************}

function tan (arguement: extended): extended;

        { This function returns the tangent of the input angle in radians}
begin                                                                {tan}
        tan := sin(arguement) / cos(arguement);
end;                                                                 {tan}

{=========================================================================}
function asin (arguement: extended): extended;
```

```
        {This function returns the arcsine value of the input angle in radians}

begin                                                                   {asin}
        asin := ArcTan((arguement) / (sqrt(1 - arguement * arguement)));
end;                                                                    {asin}

{=================================================================}
function acos (arguement: extended): extended;

        {This function returns the arcosine value of the input angle in radians}

begin                                                                   {acos}
        acos := pi / 2 - ArcTan((arguement) / (sqrt(1 - arguement * arguement)));
end;                                                                    {acos}

{=================================================================}

function CalcBeta (arg10, arg11: extended): extended;

var     Betac,                  {The cosine of Beta}
        Betas:  extended;       {The sine of Beta}

begin                                                                   {CalcBeta}
        Betac := arg11 / (sqrt(sqr(arg10) + sqr(arg11)));
        Betas := arg10 / (sqrt(sqr(arg10) + sqr(arg11)));
        CalcBeta := acos(Betac);
        if (Betas < 0) then
                CalcBeta := 2*pi - acos(Betac);
end;                                                                    {CalcBeta}

{=================================================================}
function CalcTheta (arg9, arg10, arg11, Beta: extended): extended;

var     B1:  extended;

begin                                                                   {CalcTheta}
        B1 := arg10*sin(Beta)+arg11*cos(Beta);
        CalcTheta := ArcTan((sqrt(9*arg9*arg9 + 8*sqr(B1)) - 3*arg9)/(4*B1));
end;                                                                    {CalcTheta}

{=================================================================}

{************************ Differential Equations ************************}
function Dr (arg4: extended): extended;

begin                                                                   {Dr}
        Dr := arg4;
end;                                                                    {Dr}

{=================================================================}
```

```
function Dphi (arg1, arg3, arg5: extended): extended;

begin                                                                    {Dphi}
        Dphi := arg5/(arg1*cos(arg3));
end;                                                                     {Dphi}

{================================================================}
function Dpsi (arg1, arg6: extended): extended;

begin                                                                    {Dpsi}
        Dpsi := arg6/arg1;
end;                                                                     {Dpsi}

{================================================================}
function DVr (arg1, arg5, arg6, ao, Theta: extended): extended;

begin                                                                    {DVr}
        DVr := (sqr(arg5)+sqr(arg6))/arg1 +
                (ao*cos(Theta)*cos(Theta)*cos(Theta) - 1)/(arg1*arg1);
end;                                                                     {DVr}

{================================================================}
function DVphi (arg1, arg3, arg4, arg5, arg6, ao, Theta, Beta: extended): extended;

begin                                                                    {DVphi}
        DVphi := -(arg4*arg5/arg1) + (arg5*arg6*tan(arg3)/arg1) +
                (ao*sin(Theta)*cos(Theta)*cos(Theta)*sin(Beta))/(arg1*arg1);
end;                                                                     {DVphi}

{================================================================}
function DVpsi (arg1, arg3, arg4, arg5, arg6, ao, Theta, Beta: extended): extended;

begin                                                                    {DVpsi}
        DVpsi := -(arg4*arg6/arg1) - (arg5*arg5*tan(arg3)/arg1) +
                (ao*sin(Theta)*cos(Theta)*cos(Theta)*cos(Beta))/(arg1*arg1);
end;                                                                     {DVpsi}

{================================================================}
function DPvr (arg1, arg5, arg6, arg7, arg10, arg11: extended): extended;

begin                                                                    {DPvr}
        DPvr := arg10*arg5/arg1 + arg11*arg6/arg1 - arg7;
end;                                                                     {DPvr}

{================================================================}
function DPvphi (arg1, arg3, arg4, arg5, arg6, arg9, arg10, arg11: extended): extended;

begin                                                                    {DPvphi}
        DPvphi := -(2*arg9*arg5/arg1) + arg4*arg10/arg1 - arg10*arg6*tan(arg3)/arg1 +
                        2*arg11*arg5*tan(arg3)/arg1;
end;                                                                     {DPvphi}
```

```
{=============================================================}
function DPvpsi (arg1, arg3, arg4, arg5, arg6, arg8, arg9, arg10, arg11: extended):
                extended;

begin                                                          {DVphi}
        DPvpsi := -(2*arg9*arg6/arg1) + arg4*arg11/arg1 - arg10*arg5*tan(arg3)/arg1 -
                        arg8/arg1;
end;                                                           {DVphi}

{=============================================================}
function DPr (arg1, arg3, arg4, arg5, arg6, arg8, arg9, arg10, arg11, ao,
                Theta, Beta: extended): extended;

var     C1, C2, C3, C4, C5, C6, C7, C8, C9, C10:  extended;

begin                                                          {DPr}
        C1 := arg8*arg6/(sqr(arg1));
        C2 := arg9*(sqr(arg5)+sqr(arg6))/(arg1*arg1);
        C3 := 2*arg9/(arg1*arg1*arg1);
        C4 := 2*ao*arg9*cos(Theta)*cos(Theta)*cos(Theta)/(arg1*arg1*arg1);
        C5 := arg10*arg4*arg5/(arg1*arg1);
        C6 := arg10*arg5*arg6*tan(arg3)/(arg1*arg1);
        C7 :=2*arg10*ao*cos(Theta)*cos(Theta)*sin(Theta)*sin(Beta)/(arg1*arg1*arg1);
        C8 := arg11*arg4*arg6/(sqr(arg1));
        C9 := arg11*arg5*arg5*tan(arg3)/(sqr(arg1));
        C10 := 2*arg11*ao*cos(Theta)*cos(Theta)*sin(Theta)*cos(Beta)/
                (arg1*arg1*arg1);
        DPr := C1 + C2 - C3 + C4 - C5 + C6 + C7 - C8 - C9 + C10;
end;                                                           {DPr}

{=============================================================}
function DPpsi (arg1, arg3, arg5, arg6, arg10, arg11:extended): extended;

begin                                                          {DPsi}
        DPpsi := -(arg10*arg5*arg6/(arg1*cos(arg3)*cos(arg3))) +
                        arg11*sqr(arg5)/(arg1*cos(arg3)*cos(arg3));
end;                                                           {DPsi}

{*************************************************************}
{************************** PROCEDURES ***********************}
{*************************************************************}
procedure Mercury (var choice: vec; time: extended);

                        { This procedure determines the orbital elements    }
                        { for the planet Mercury at the date chosen.        }

begin                                                          {Mercury}
        choice[1] := (14732.4197380 - 0.000001355 * time);
        choice[2] := (178 + 10 / 60 + 44.68 / 3600) + (538106654.80 / 3600) * time +
                        (1.084 / 3600) * time * time;
```

```
        choice[3] := 75 + 53 / 60 + 58.91 / 3600 + (5599.76 / 3600) * time +
                        (1.061 / 3600) * time * time;
        choice[4] := 0.20561421 + 0.00002046 * time - 0.000000030 * time * time;
        choice[5] := 0.38709860;
        choice[6] := 7 + 10.37 / 3600 + (6.699 / 3600) * time -
                        (0.066 / 3600) * time * time;
        choice[7] := 47 + 8 / 60 + 45.4 / 3600 + (4266.75 / 3600) * time +
                        (0.626 / 3600) * time * time;
end;                                                        {Freddy Mercury}

{=============================================================}
procedure Venus (var choice: vec; time: extended);

                        { This procedure determines the orbital elements    }
                        { for the planet Venus at the date chosen.          }

begin                                                                {Venus}
        choice[1] := (5767.6697692 + 0.0000002628 * time);
        choice[2] := 342 + 46 / 60 + 1.39 / 3600 + (210669162.88 / 3600) * time +
                        (1.1148/3600) * time * time;
        choice[3] := 130 + 9 / 60 + 49.8 / 3600 + (5068.93 / 3600) * time -
                        (3.515 / 3600) * time * time;
        choice[4] := 0.00682069 - 0.00004774 * time + 0.000000091 * time * time;
        choice[5] := 0.7323162;
        choice[6] := 3 + 23 / 60 + 37.07 / 3600 + (3.621 / 3600) * time -
                        (0.0035 / 3600) * time * time;
        choice[7] := 75 + 46 / 60 + 46.73 / 3600 + (3239.46 / 3600) * time +
                        (1.476 / 3600) * time * time;
end;                                                                 {Venus}

{=============================================================}
procedure earth (var choice: vec; time: extended);

                        { This procedure determines the orbital elements    }
                        { for the planet earth at the date chosen.          }

begin                                                               {earth}
        choice[1] := (3548.1928323 - 0.000001103 * time);
        choice[2] := 99 + 41 / 60 + 48.04 / 3600 + (129602768.13 / 3600) * time +
                        (1.089 / 3600) * time * time;
        choice[3] := 101 + 13 / 60 + 15 / 3600 + (6189.03 / 3600) * time +
                        (1.63 / 3600) * time * time + (0.012 / 3600) * time * time * time;
        choice[4] := 0.01675104 - 0.00004180 * time - 0.000000126 * time * time;
        choice[5] := 1.00000023;
        choice[6] := 0.0;
        choice[7] := 0.0;
end;                                                                {earth}

{=============================================================}
procedure Mars (var choice: vec; time: extended);
```

```
                    { This procedure determines the orbital elements    }
                    { for the planet Mars at the date chosen.           }

begin                                                                   {Mars}
        choice[1] := (1886.5186207 + 0.000000463 * time);
        choice[2] := 293 + 44 / 60 + 51.46 / 3600 + (68910103.83 / 3600) * time +
                        1.1184 * time * time / 3600;
        choice[3] := 334 + 13 / 60 + 5.53 / 3600 + (6626.73 / 3600) * time +
                        (0.4675 / 3600) * time * time -
                        (0.0043 / 3600) * time * time * time;
        choice[4] := 0.09331290 + (0.000092064) * time - (0.000000077) * time * time;
        choice[5] := 1.52368840;
        choice[6] := 1 + 51 / 60 + 1.2 / 3600 - (2.43 / 3600) * time +
                        (0.0454 / 3600) * time * time;
        choice[7] := 48 + 47 / 60 + 11.19 / 3600 + (2775.57 / 3600) * time -
                        (0.005 / 3600) * time * time -
                        (0.0192 / 3600) * time * time * time;
end;                                                    {Mars Bar (with Almonds)}

{=============================================================}
procedure Convert (var date: extended);

                    { This procedure converts any date enterred in the  }
                    { standard calender into a Julian date. Taken from  }
                    { Astronomical Formulae for Calculators, Meeus,     }
                    { p. 24.                                            }

var     Month,              {The portion of the enterred date that represents the month}
        Day,                {The portion of the enterred date that represents the day}
        Year: extended;     {The protion of the enterred date that represents the year}
        a,b: extended;      {Temporary holding variables}

begin                                                                   {Convert}
        Month := trunc(date);
        Day := trunc(100 * (date - Month));
        Year := Round((1000000 * date) - (Month * 1000000) - (Day * 10000));
        if (Month <= 2) then
                begin                                                           {if}
                        Year := Year - 1;
                        Month := Month + 12;
                end;                                                            {if}
        a := Year/100;
        b := 2 - a + a/4;
        date := trunc(365.25 * Year) + trunc(30.60001*(Month + 1.0)) + Day +
                1720994.5 + b;
end;                                                                    {Convert}

{=============================================================}
procedure InvConvert (var date: extended);
```

```
                    { This procedure converts any Julian date    }
                    { into a calendar date.                      }
                    { Taken from Astronomical Formulae for       }
                    { Calculators, Meeus, p. 24.                 }

var
        Month,              {The portion of the enterred date that represents the month}
        Day,                {The portion of the enterred date that represents the day}
        Year: extended;     {The protion of the enterred date that represents the year}
        a,b,c,d,e,
        f: extended;        {Temporary holding variables}

begin                                                            {InvConvert}
        f := date + 0.5;
        a := trunc(f);
        f := f - a;
        if (a >= 2299161) then
                begin                                                    {if #1}
                        b := (a - 1867216.25)/36524.25;
                        a := a + (1 + b - b/4);
                end;                                                     {if #1}
        b := a + 1524;
        c := trunc((b - 122.1)/365.25);
        d := trunc(365.25*c);
        e := trunc((b - d)/30.6001);
        Day := b - d - trunc(30.6001*e) + f;
        if (e <= 13) then
                Month := e - 1
        else
                Month := e - 13;
        if (Month <= 2) then
                Year := c - 4715
        else
                Year := c - 4716;
        date := trunc(Month) + 0.01*round(Day) + 0.000001*trunc(Year);

end;                                                             {InvConvert}

{=================================================================}
procedure CaseEntry (var JlaunchT, tstar, ao: extended; var x:vec);

                    { This procedure reads then characteristic   }
                    { acceleration, a0, and the inital guesses for }
                    { launch date and time of flight. It also    }
                    { calls the procedure Convert to convert the }
                    { launch date to the Julian calendar.        }


begin                                                            {CaseEntry}
        writeln('Enter the initial guess launch date as mm.ddyyyy or as a Julian date.');
        readln(JlaunchT);
```

```
        if ((JlaunchT / 1000) < 1) then
                Convert(JlaunchT);
        x[1] := JlaunchT;
        writeln('Enter the initial guess for time of flight in days.');
        readln(x[2]);
        x[2] := x[2]/tstar;
        writeln('Enter the characteristic acceleration in (mm/s^2).');
        readln(ao);
        ao := ((0.000001 * ao)/AU)/GM;
end;                                                        {CaseEntry}

{============================================================}
procedure BuildTrans1 (var SmOmega: extended; var body: vec; var Transec: mat);

                { This procedure builds the matrix which serves as  }
                { the transformation matrix between the ecliptic    }
                { and pericentric coordinate systems.               }

begin                                                       {BuildTrans1}
        Transec[1, 1] := cos(body[7] / RAD) * cos(SmOmega / RAD) - sin(body[7] /
                         RAD) *sin(SmOmega / RAD) * cos(body[6] / RAD);
        Transec[1, 2] := -cos(body[7] / RAD) * sin(SmOmega / RAD) - sin(body[7] /
                         RAD) *cos(SmOmega / RAD) * cos(body[6] / RAD);
        Transec[1, 3] := sin(body[7] / RAD) * sin(body[6] / RAD);
        Transec[2, 1] := sin(body[7] / RAD) * cos(SmOmega / RAD) + cos(body[7] /
                         RAD) *sin(SmOmega / RAD) * cos(body[6] / RAD);
        Transec[2, 2] := -sin(body[7] / RAD) * sin(SmOmega / RAD) + cos(body[7] /
                         RAD) *cos(SmOmega / RAD) * cos(body[6] / RAD);
        Transec[2, 3] := -cos(body[7] / RAD) * sin(body[6] / RAD);
        Transec[3, 1] := sin(SmOmega / RAD) * sin(body[6] / RAD);
        Transec[3, 2] := cos(SmOmega / RAD) * sin(body[6] / RAD);
        Transec[3, 3] := cos(body[6] / RAD);
end;                                                        {BuildTrans1}

{============================================================}
procedure BuildTrans2 (var Phiplanet, Psiplanet: vec; index:integer; var Transsp: mat);

                        { This procedure builds the matrix which serves as  }
                        { the transformation matrix between the spherical   }
                        { and ecliptic coordinate systems.                  }

begin                                                       {BuildTrans2}
        Transsp[1, 1] := cos(Phiplanet[index])*cos(Psiplanet[index]);
        Transsp[1, 2] := sin(Phiplanet[index])*cos(Psiplanet[index]);
        Transsp[1, 3] := sin(Psiplanet[index]);
        Transsp[2, 1] := -sin(Phiplanet[index]);
        Transsp[2, 2] := cos(Phiplanet[index]);
        Transsp[2, 3] := 0;
        Transsp[3, 1] := -sin(Psiplanet[index])*cos(Phiplanet[index]);
        Transsp[3, 2] := -sin(Psiplanet[index])*sin(Phiplanet[index]);
        Transsp[3, 3] := cos(Psiplanet[index]);
```

```
end;                                                    {BuildTrans2}

{=============================================================}
procedure ComputePositions (var body, Rplanet, Phiplanet, Vrplanet, Vphiplanet: vec;
                            index: integer);

var     Ma,
        Mo,
        Ea,
        D,
        SminAx,
        SmOmega,
        Perix,
        Periy,
        Vpx,
        Vpy,
        Vearthx,
        Vearthy,
        Vearthz,
        B,
        Omeg1,
        Omeg2,
        relinc,
        Vel: extended;
        nu,
        Ecliptx,
        Eclipty,
        Ecliptz,
        EcliptVx,
        EcliptVy,
        EcliptVz: vec;
        Transec,
        Transsp: mat;

begin                                                   {ComputePositions}
        Ma := pi / 180 * (body[2] - body[3]);
        if (Ma < 0) then
                Ma := Ma + (2 * 3.141592654);
        Ea := Ma + body[4] * (sin(Ma) + body[4] * sin(2 * Ma) / 2);
        repeat
                Mo := Ea - body[4] * sin(Ea);
                D := (Ma - Mo) / (1 - body[4] * cos(Ea));
                Ea := Ea + D;
        until (abs(D) <= 1E-10);
        nu[index] := 2 * ArcTan(sqrt((1+body[4]) / (1-body[4])) *((sin(Ea/2) /
                cos(Ea/2))));
        if (nu[index] < 0) then
                nu[index] := nu[index] + 2 * pi;
        SminAx := body[5]*sqrt(1 - body[4]*body[4]);
        SmOmega := body[3] - body[7];
        Rplanet[index] := body[5] * (1-body[4]*body[4]) / (1+body[4]*cos(nu[index]));
```

```
Perix := Rplanet[index] * cos(nu[index]);
Periy := Rplanet[index] * sin(nu[index]);
VPx := (-body[5] * sin(Ea) / (sqrt(body[5] * body[5] * body[5]) *
        (1 - body[4] * cos(Ea))));
VPy := (SminAx * cos(Ea)/ (sqrt(body[5] * body[5] * body[5]) *
        (1 - body[4] * cos(Ea))));
Vel := sqrt(2/Rplanet[index]-1/body[5]);
BuildTrans1 (SmOmega,body,Transec);
Ecliptx[index] := Transec[1, 1] * Perix + Transec[1, 2] * Periy;
Eclipty[index] := Transec[2, 1] * Perix + Transec[2, 2] * Periy;
Ecliptz[index] := Transec[3, 1] * Perix + Transec[3, 2] * Periy;
EcliptVx[index] := Transec[1, 1] * VPx + Transec[1, 2] * VPy;
EcliptVy[index] := Transec[2, 1] * VPx + Transec[2, 2] * VPy;
EcliptVz[index] := Transec[3, 1] * VPx + Transec[3, 2] * VPy;
Phiplanet[index] := ArcTan(Eclipty[index] / Ecliptx[index]);
if ((Ecliptx[index] < 0) and (Eclipty[index] > 0) and (Phiplanet[index] < 0)) then
        Phiplanet[index] := Phiplanet[index] + pi;
if ((Ecliptx[index] < 0) and (Eclipty[index] < 0) and (Phiplanet[index] > 0)) then
        Phiplanet[index] := Phiplanet[index] + pi;
if ((Ecliptx[index] > 0) and (Eclipty[index] < 0) and (Phiplanet[index] < 0)) then
        Phiplanet[index] := Phiplanet[index] + 2 * pi;
Psiplanet[index] := asin(sin(SmOmega/RAD+nu[index]) * sin(body[6]/RAD));
BuildTrans2 (Phiplanet, Psiplanet, index, Transsp);
Vrplanet[index] := Transsp[1, 1]*EcliptVx[index] +
        Transsp[1,2]*EcliptVy[index] +Transsp[1, 3]*EcliptVz[index];
Vphiplanet[index] := Transsp[2, 1]*EcliptVx[index] +
        Transsp[2, 2]*EcliptVy[index] + Transsp[2, 3]*EcliptVz[index];
Vpsiplanet[index] := Transsp[3, 1]*EcliptVx[index] +
        Transsp[3, 2]*EcliptVy[index] + Transsp[3, 3]*EcliptVz[index];
if (index = 1) and (doit = TRUE) then
        begin                                                    {if #1}
                Vearthx := -EcliptVx[index]*sin(Phiplanet[index]) +
                        (EcliptVy[index]*cos(inc) - EcliptVz[index]*sin(inc))*
                        cos(Phiplanet[index]);
                Vearthy := -EcliptVx[index]*cos(Phiplanet[index]) +
                        (- EcliptVy[index]*cos(inc) + EcliptVz[index]*sin(inc))*
                        sin(Phiplanet[index]);
                Vearthz := EcliptVy[index]*sin(inc) + EcliptVz[index]*cos(inc);
                B := acos(Vearthz/Vel);
                if ((B + Parki)<pi/2) then
                        begin                                    {if #2}
                                writeln('Nontangential injection.');
                                relinc := asin((cos(B+Parki))/(cos(89.9999/RAD)));
                                writeln('relinc: ',relinc*RAD:1:6);
                        end;                                     {if #2}
                if ((B + Parki)>=pi/2) then
                        begin                                    {if #3}
                                writeln('Tangential injection.');
                                writeln('No plane change required.');
                                Omeg1 := pi + asin(1/(tan(B)*tan(Parki)));
                                Omeg2 := 2*pi - asin(1/(tan(B)*tan(Parki)));
```

```
                                writeln;
                                writeln('Omega 1: ',Omeg1*RAD:1:6);
                                writeln('Omega 2: ',Omeg2*RAD:1:6);
                                write('(Omega is measured relative to the earth"s');
                                writeln(' velocity vector.)');
                            end;                                              {if #3}
                        Vearthx := Vearthx*sqrt(GM)*AU;
                        Vearthy := Vearthy*sqrt(GM)*AU;
                        Vearthz := Vearthz*sqrt(GM)*AU;
                        EcliptVx[index] := EcliptVx[index]*sqrt(GM)*AU;
                        EcliptVy[index] := EcliptVy[index]*sqrt(GM)*AU;
                        EcliptVz[index] := EcliptVz[index]*sqrt(GM)*AU;
                end;                                                          {if #1}
end;                                                          {ComputePositions}

{=============================================================}
procedure InitialConditions (var JlaunchT, Jcentury1, Jcentury2,J arriveT, t, ao, Vr, Vphi,
                             r, phi, PVr,PVphi,Pr,Tfinal,Phiearth,Phifinal: extended;
                             var x, departure, target, Phiplanet: vec);

var     i: integer;     {an indexing variable}

begin                                                         {Initial Conditions}
        JlaunchT := x[1];
        Jcentury1 := (JlaunchT - 2415021) / (36525);
        for i := 1 to 7 do
                departure[i] := 0;
        earth(departure, Jcentury1);
        JarriveT := x[1] + x[2] * tstar;
        Jcentury2 := (JarriveT - 2415021) / (36525);
        for i := 1 to 7 do
                target[i] := 0;
        Mars(target, Jcentury2);
        ComputePositions (departure, Rplanet, Phiplanet, Vrplanet, Vphiplanet, 1);
        ComputePositions (target, Rplanet, Phiplanet, Vrplanet, Vphiplanet, 2);
        rearth := Rplanet[1];
        rfinal := Rplanet[2];
        Phiearth := Phiplanet[1];
        Phifinal := Phiplanet[2];
        if (Phiearth >= 2 * pi) then
                Phiearth := Phiearth - 2 * pi;
        if (Phifinal >= 2 * pi) then
                Phifinal := Phifinal - 2 * pi;
        Psiearth := Psiplanet[1];
        Psifinal := Psiplanet[2];
        Vrearth := Vrplanet[1];
        Vrfinal := Vrplanet[2];
        Vphiearth := Vphiplanet[1];
        Vphifinal := Vphiplanet[2];
        Vpsiearth := Vpsiplanet[1];
        Vpsifinal := Vpsiplanet[2];
```

```
        t := 0.0;
        Vr := Vrearth;
        Vphi := Vphiearth;
        Vpsi := Vpsiearth;
        r := rearth;
        phi := Phiearth;
        Psi := Psiearth;
        Tfinal := x[2];
        Pr := x[3];
        PVr := x[4];
        PVphi := x[5];
        Ppsi := x[6];
        PVpsi := x[7];
        Pphi := 0.0;
end;                                                    {Initial Conditions}

{================================================================}
procedure RungeKutta (var r, Phi, Psi, Vr, Vphi, Vpsi, PVr, PVphi, PVpsi, Pr, Ppsi,
                      Theta, Beta, ao, Hamil: extended);

var     K1, K2, K3, K4, L1, L2, L3, L4, M1, M2, M3, M4, N1, N2, N3, N4: extended;
        O1, O2, O3, O4, P1, P2, P3, P4, Q1, Q2, Q3, Q4, R1, R2, R3, R4: extended;
        S1, S2, S3, S4, U1, U2, U3, U4, W1, W2, W3, W4: extended;

begin                                                   {RungeKutta}
        Beta := CalcBeta (PVphi, PVpsi);
        Theta := CalcTheta (PVr, PVphi, PVpsi, Beta);
        K1 := dt*Dr(Vr);
        L1 := dt*Dphi(r, Psi, Vphi);
        M1 := dt*Dpsi(r, Vpsi);
        N1 := dt*DVr(r, Vphi, Vpsi, ao, Theta);
        O1 := dt*DVphi(r, Psi, Vr, Vphi, Vpsi, ao, Theta, Beta);
        P1 := dt*DVpsi(r, Psi, Vr, Vphi, Vpsi, ao, Theta, Beta);
        Q1 := dt*DPvr(r, Vphi, Vpsi, Pr, PVphi, PVpsi);
        R1 := dt*DPvphi(r, Psi, Vr, Vphi, Vpsi, PVr, PVphi, PVpsi);
        S1 := dt*DPvpsi(r, Psi, Vr, Vphi, Vpsi, Ppsi, PVr, PVphi, PVpsi);
        U1 := dt*DPr(r, Psi, Vr, Vphi, Vpsi, Ppsi, PVr, PVphi, PVpsi, ao, Theta, Beta);
        W1 := dt*DPpsi(r, Psi, Vphi, Vpsi, PVphi, PVpsi);

        Beta := CalcBeta (PVphi+R1/2.0, PVpsi+S1/2.0);
        Theta := CalcTheta(PVr + Q1/2.0, PVphi + R1/2.0, PVpsi+S1/2.0, Beta);
        K2 := dt*Dr(Vr+N1/2.0);
        L2 := dt*Dphi(r+K1/2.0, Psi+M1/2.0, Vphi+O1/2.0);
        M2 := dt*Dpsi(r+K1/2.0, Vpsi+P1/2.0);
        N2 := dt*DVr(r+K1/2.0, Vphi+O1/2.0, Vpsi+P1/2.0, ao, Theta);
        O2 := dt*DVphi(r+K1/2.0, Psi+M1/2.0, Vr+N1/2.0, Vphi+O1/2.0, Vpsi+P1/2.0,
                       ao, Theta, Beta);
        P2 := dt*DVpsi(r+K1/2.0, Psi+M1/2.0, Vr+N1/2.0, Vphi+O1/2.0, Vpsi+P1/2.0,
                       ao, Theta, Beta);
        Q2 := dt*DPvr(r+K1/2.0, Vphi+O1/2.0, Vpsi+P1/2.0, Pr+U1/2.0, PVphi+R1/2.0,
                       PVpsi+S1/2.0);
```

```
R2 := dt*DPvphi(r+K1/2.0, Psi+M1/2.0, Vr+N1/2.0, Vphi+O1/2.0, Vpsi+P1/2.0,
                PVr+Q1/2.0, PVphi+R1/2.0, PVpsi+S1/2.0);
S2 := dt*DPvpsi(r+K1/2.0, Psi+M1/2.0, Vr+N1/2.0, Vphi+O1/2.0, Vpsi+P1/2.0,
                Ppsi+W1/2.0, PVr+Q1/2.0, PVphi+R1/2.0, PVpsi+S1/2.0);
U2 := dt*Dpr(r+K1/2.0, Psi+M1/2.0, Vr+N1/2.0, Vphi+O1/2.0, Vpsi+P1/2.0,
                Ppsi+W1/2.0, PVr+Q1/2.0, PVphi+R1/2.0, PVpsi+S1/2.0,
                ao, Theta, Beta);
W2 := dt*DPpsi(r+K1/2.0, Psi+M1/2.0, Vphi+O1/2.0, Vpsi+P1/2.0,
       PVphi+R1/2.0, PVpsi+S1/2.0);

Beta := CalcBeta (PVphi+R2/2.0, PVpsi+S2/2.0);
Theta := CalcTheta(PVr + Q2/2.0, PVphi + R2/2.0, PVpsi+S2/2.0, Beta);
K3 := dt*Dr(Vr+N2/2.0);
L3 := dt*Dphi(r+K2/2.0, Psi+M2/2.0, Vphi+O2/2.0);
M3 := dt*Dpsi(r+K2/2.0, Vpsi+P2/2.0);
N3 := dt*DVr(r+K2/2.0, Vphi+O2/2.0, Vpsi+P2/2.0, ao, Theta);
O3 := dt*DVphi(r+K2/2.0, Psi+M2/2.0, Vr+N2/2.0, Vphi+O2/2.0, Vpsi+P2/2.0,
                ao, Theta, Beta);
P3 := dt*DVpsi(r+K2/2.0, Psi+M2/2.0, Vr+N2/2.0, Vphi+O2/2.0, Vpsi+P2/2.0,
                ao, Theta, Beta);
Q3 := dt*DPvr(r+K2/2.0, Vphi+O2/2.0, Vpsi+P2/2.0, Pr+U2/2.0, PVphi+R2/2.0,
                PVpsi+S2/2.0);
R3 := dt*DPvphi(r+K2/2.0, Psi+M2/2.0, Vr+N2/2.0, Vphi+O2/2.0, Vpsi+P2/2.0,
                PVr+Q2/2.0, PVphi+R2/2.0, PVpsi+S2/2.0);
S3 := dt*DPvpsi(r+K2/2.0, Psi+M2/2.0, Vr+N2/2.0, Vphi+O2/2.0, Vpsi+P2/2.0,
                Ppsi+W2/2.0, PVr+Q2/2.0, PVphi+R2/2.0, PVpsi+S2/2.0);
U3 := dt*Dpr(r+K2/2.0, Psi+M2/2.0, Vr+N2/2.0, Vphi+O2/2.0, Vpsi+P2/2.0,
                Ppsi+W2/2.0, PVr+Q2/2.0, PVphi+R2/2.0, PVpsi+S2/2.0,
                ao, Theta, Beta);
W3 := dt*DPpsi(r+K2/2.0, Psi+M2/2.0, Vphi+O2/2.0, Vpsi+P2/2.0,
                PVphi+R2/2.0, PVpsi+S2/2.0);

Beta := CalcBeta (PVphi+R3, PVpsi+S3);
Theta := CalcTheta(PVr + Q3, PVphi + R3, PVpsi+S3, Beta);
K4 := dt*Dr(Vr+N3);
L4 := dt*Dphi(r+K3, Psi+M3, Vphi+O3);
M4 := dt*Dpsi(r+K3, Vpsi+P3);
N4 := dt*DVr(r+K3, Vphi+O3, Vpsi+P3, ao, Theta);
O4 := dt*DVphi(r+K3, Psi+M3, Vr+N3, Vphi+O3, Vpsi+P3, ao, Theta, Beta);
P4 := dt*DVpsi(r+K3, Psi+M3, Vr+N3, Vphi+O3, Vpsi+P3, ao, Theta, Beta);
Q4 := dt*DPvr(r+K3, Vphi+O3, Vpsi+P3, Pr+U3, PVphi+R3, PVpsi+S3);
R4 := dt*DPvphi(r+K3, Psi+M3, Vr+N3, Vphi+O3, Vpsi+P3,PVr+Q3,
                PVphi+R3, PVpsi+S3);
S4 := dt*DPvpsi(r+K3, Psi+M3, Vr+N3, Vphi+O3, Vpsi+P3,Ppsi+W3, PVr+Q3,
                PVphi+R3, PVpsi+S3);
U4 := dt*Dpr(r+K3, Psi+M3, Vr+N3, Vphi+O3, Vpsi+P3, Ppsi+W3, PVr+Q3,
                PVphi+R3, PVpsi+S3, ao, Theta, Beta);
W4 := dt*DPpsi(r+K3, Psi+M3, Vphi+O3, Vpsi+P3, PVphi+R3, PVpsi+S3);

{********************** Final Runge Kutta Equations ********************}
```

```
        r := r + (1.0/6.0)*(K1 + 2.0*K2 + 2.0*K3 + K4);
        Phi := Phi + (1.0/6.0)*(L1 + 2.0*L2 + 2.0*L3 + L4);
        if (Phi >= 2 * pi) then
                Phi := Phi - 2 * pi;
        Psi := Psi + (1.0/6.0)*(M1 + 2.0*M2 + 2.0*M3 + M4);
        Vr := Vr + (1.0/6.0)*(N1 + 2.0*N2 + 2.0*N3 + N4);
        Vphi := Vphi + (1.0/6.0)*(O1 + 2.0*O2 + 2.0*O3 + O4);
        Vpsi := Vpsi + (1.0/6.0)*(P1 + 2.0*P2 + 2.0*P3 + P4);
        PVr := PVr + (1.0/6.0)*(Q1 + 2.0*Q2 + 2.0*Q3 + Q4);
        PVphi := PVphi + (1.0/6.0)*(R1 + 2.0*R2 + 2.0*R3 + R4);
        PVpsi := PVpsi + (1.0/6.0)*(S1 + 2.0*S2 + 2.0*S3 + S4);
        Pr := Pr + (1.0/6.0)*(U1 + 2.0*U2 + 2.0*U3 + U4);
        Ppsi := Ppsi + (1.0/6.0)*(W1 + 2.0*W2 + 2.0*W3 + W4);
        Beta := CalcBeta (PVphi, PVpsi);
        Theta := CalcTheta(PVr, PVphi, PVpsi, Beta);

        Hamil := Pr*Vr + Ppsi*Vpsi/r;
        Hamil := Hamil + PVr*((sqr(Vphi)+sqr(Vpsi))/r - 1/(r*r)) +
                        PVr*(ao*cos(Theta)*cos(Theta)*cos(Theta)/(r*r));
        Hamil := Hamil + PVphi*(-Vr*Vphi/r + Vphi*Vpsi*tan(Psi)/r) +
                        PVphi*(ao*sin(Theta)*cos(Theta)*cos(Theta)*sin(Beta)/(r*r));
        Hamil := Hamil + PVpsi*(-Vr*Vpsi/r - Vphi*Vphi*tan(Psi)/r) +
                        PVpsi*(ao*sin(Theta)*cos(Theta)*cos(Theta)*cos(Beta)/(r*r));
end;                                                                    {RungeKutta}

{================================================================}
procedure Solution (var r, Phi, Psi, Vr, Vphi, Vpsi, PVr, PVphi, PVpsi, Pr, Ppsi,
                        Theta, Beta, ao: extended);

var     time1: extended;

begin                                                                   {Solution}
        InitialConditions(JlaunchT, Jcentury1, Jcentury2, JarriveT, t, ao, Vr, Vphi, r, phi,
                        PVr, PVphi, Pr, Tfinal, Phiearth, Phifinal, x, departure, target,
                        Phiplanet);
        time1 := 0.0;
        while (t <= Tfinal) do
                begin                                                   {while}
                        RungeKutta(r, Phi, Psi, Vr, Vphi, Vpsi, PVr, PVphi, PVpsi, Pr,
                                        Ppsi, Theta, Beta, ao, Hamil);
                        t := t + dt;
                        time1 := t * tstar;
                        if (time1 > 1000) then
                                return;
                end;                                                    {while}
end;                                                                    {Solution}

{================================================================}
procedure Vecset (var x,y: vec);

begin                                                                   {Vecset}
```

```
        Solution (r,Phi,Psi,Vr,Vphi,Vpsi,PVr,PVphi,PVpsi,Pr,Ppsi,Theta,Beta,ao);
        y[1] := r - rfinal;
        y[2] := Phi - Phifinal;
        y[3] := Psi - Psifinal;
        y[4] := Vr - Vrfinal;
        y[5] := Vphi - Vphifinal;
        y[6] := Vpsi - Vpsifinal;
        y[7] := Hamil - 1;
end;                                                                    {Vecset}

{================================================================}
procedure GaussianElim (var A: mat; var b,x: vec; var n: integer);

var     SltnCheck: boolean;


procedure AugMatrix (var A: mat; var b: vec);

var     i: integer;

begin                                                                   {AugMatrix}
        for i := 1 to n do
                A[i,n+1] := b[i];
end;                                                                    {AugMatrix}

procedure PartialPiv (var A: mat; var n, k: integer);

var     row, i, j: integer;
        max, temp: extended;

begin                                                                   {PartialPiv}
        max := Abs(A[k,k]);
        row := k;
        for i := k+1 to n do
                begin                                                   { for #1}
                        if (Abs(A[i,k]) > max) then
                                begin                                   {if #1}
                                        row := i;
                                        max := Abs(A[i,k]);
                                end;                                    {if #1}
                end;                                                    {for #1}
        if (row <> k) then
                begin                                                   {if #2}
                        for j := k to n+1 do
                                begin                                   {for #2}
                                        temp := A[k,j];
                                        A[k,j] := A[row,j];
                                        A[row,j] := temp;
                                end;                                    {for #2}
                end;                                                    {if #2}
```

```
end;                                                        {PartialPiv}

procedure BackSubst (var A: mat; var n: integer; var x: vec);

var     i,j: integer;
        sum: extended;

begin                                                       {BackSubst}
        x[n] := A[n,n+1]/A[n,n];
        for i := n-1 downto 1 do
                begin                                           {for #1}
                        sum := 0.0;
                        for j := i+1 to n do
                                sum := sum + A[i,j]*x[j];
                        x[i] := (A[i,n+1] - sum)/A[i,i];
                end;                                            {for #1}
end;                                                        {BackSubst}

procedure Gauss (var A: mat; var n: integer; var SltnCheck: boolean);

var     i, j, k:  integer;
        m, tol:  extended;

begin                                                           {Gauss}
        tol := 1.0e-16;
        for k := 1 to n-1 do
                begin                                           {for #1}
                        if (SltnCheck = TRUE) then
                                begin                           {if #1}
                                        k1 := k;
                                        PartialPiv (A, n, k1);
                                        if (Abs(A[k,k]) > tol) then
                                                begin           {if #2}
                                                        for i := k+1 to n do
                                                        begin (* i *)
                                                        m := A[i,k]/A[k,k];
                                                        for j := k+1 to n+1 do
                                                                A[i,j] := A[i,j] - m*A[k,j]
                                                        end; (* i *)
                                                        for i := k+1 to n do
                                                                A[i,k] := 0.0;
                                                end;            {if #2}
                                end                             {if #1}
                        else
                                SltnCheck := FALSE;
                end;                                            {for #1}
        if (Abs(A[n,n]) < tol) then
                SltnCheck := FALSE;
end;                                                            {Gauss}
        tol := 1.0e-16;
```

```
begin                                                          {GaussianElim}
        SltnCheck := TRUE;
        AugMatrix(A,b);
        Gauss(A,n,SltnCheck);
        if (SltnCheck = TRUE) then
                BackSubst(A, n, x);
end;                                                           {GaussianElim}

{================================================================}
procedure Steep(var x, ybase: vec; ndim: integer);

var     y, xbase, z: vec;
        jacob, jacobT: mat;
        tol1: extended;
        g0, g1, g2, g3, h1, h2, h3: extended;
        a0, a1, a2, a3, z0, gmin, amin, delx: extended;
        i, j, k, w, q: integer;

begin                                                          {Steep}
        tol1 := 1e-15;
        for w := 1 to 15 do
                begin                                          {for #1}
                        writeln;
                        writeln('Steepest descent:');
                        writeln;
                        writeln('w = ',w:2);
                        writeln;
                        Vecset(x, ybase);
                        for i := 1 to ndim do
                                xbase[i] := x[i];

                        { Calculate Jacobian }

                        writeln;
                        for j := 1 to ndim do
                                begin                          {for #2}
                                        delx:=x[j]/100;
                                        if (j=1) then delx := 5;
                                        x[j] := x[j] + delx;
                                        write('Calculating Jacobian:');
                                        writeln(' varying initial condition ',j:1);
                                        Vecset(x,y);
                                        for i := 1 to ndim do
                                                jacob[i,j] := (y[i] - ybase[i])/delx;
                                        x[j] := x[j] - delx;
                                end;                           {for #2}
                        writeln;
                        g1 := 0.0;
                        for i := 1 to ndim do
                                g1 := g1 + ybase[i] * ybase[i];
```

```
{ Find Transpose of Jacobian }

for i := 1 to ndim do
        begin                                                   {for #3}
                for j := 1 to ndim do
                        jacobT[i,j] := jacob[j,i];
        end;                                                    {for #3}

{ Obtain 2*jacobT*ybase (gradient of g) }

for j := 1 to ndim do
        begin                                                   {for #4}
                z[j] := 0.0;
                for k := 1 to ndim do
                        z[j] := z[j] + 2 * jacobT[j,k] *ybase[k];
        end;                                                    {for #4}

z0 := 0.0;
for i := 1 to ndim do
        z0 := z0 + z[i] * z[i];
if (z0 < tol1 / 2) then
        begin                                                   {if #1}
                writeln('exit Steepest Descent');
                readln;
                return;
        end;                                                    {if #1}
for i := 1 to ndim do
        z[i] := z[i] / z0;
a1 := 0.0;
a3 := 1.0;
for i := 1 to ndim do
        x[i] := xbase[i] - a3 * z[i];
Vecset(x,y);
g3 := 0.0;
for i := 1 to ndim do
        g3 := g3 + y[i] *y[i];
while ((abs(g3) - abs(g1)) > tol1 / 10) do
        begin                                                   {while #1}
                a3 := a3 / 2;
                for i := 1 to ndim do
                        x[i] := xbase[i] - a3 * z[i];
                Vecset(x,y);
                g3 := 0.0;
                for i := 1 to ndim do
                        g3 := g3 + y[i] *y[i];
                writeln('Steep');
                for i := 1 to ndim do
                        writeln('x[',i:1,'] = ', x[i]:1:15);
                writeln('g1 = ',g1:1:15);
                writeln('g3 = ',g3:1:15);
```

```
                writeln('g3 - g1 = ',abs(g3) - abs(g1));
                writeln;
                if (a3 < 1e-15) then
                        return;
        end;                                                    {while #1}
a2 := a3 / 2;
for i:= 1 to ndim do
        x[i] := xbase[i] - a2 * z[i];
Vecset(x,y);
g2 := 0.0;
for i := 1 to ndim do
        g2 := g2 + y[i] * y[i];
h1 := (g2 - g1) / a2;
h2 := (g3 - g2) / (a3 - a2);
h3 := (h2 - h1) / a3;
writeln('g1 = ',g1:1:15);
writeln('g2 = ',g2:1:15);
writeln('g3 = ',g3:1:15);
a0 := 0.5 * (a2 - h1/h3);
for i := 1 to ndim do
        x[i] := xbase[i] - a0 * z[i];
Vecset(x,y);
g0 := 0.0;
for i := 1 to ndim do
        g0 := g0 + y[i]*y[i];
gmin := g0;
amin := a0;
if (g1 < gmin) then
        begin                                                   {if #2}
                gmin := g1;
                amin := a1;
        end;                                                    {if #2}
if (g2 < gmin) then
        begin                                                   {if #3}
                gmin := g2;
                amin := a2;
        end;                                                    {if #3}
if (g3 < gmin) then
        begin                                                   {if #4}
                gmin := g3;
                amin := a3;
        end;                                                    {if #4}
for i := 1 to ndim do
        x[i] := xbase[i] - amin*z[i];
writeln('gmin = ',gmin);
for i := 1 to ndim do
        writeln('xbest[',i:1,'] = ',x[i]:1:15);
for i := 1 to 3 do
        writeln;
if (gmin < tol1) then
        begin                                                   {if #5}
```

```
                                writeln('exit Steep');
                                readln;
                                return;
                            end;                                        {if #5}
                    end;                                                {for #2}
        writeln('Steepest descent did not work');
        readln;
end;                                                                    {Steep}

{================================================================}
procedure Vecroot(var x, ybase: vec; ndim: integer);
var     y,
        x1,
        xbest:  vec;            {The value of x that gives the least error}
        jacob:  mat;            {The Jacobian matrix}
        errlast,                {Error on the previous iteration}
        err,                    {Error on the current iteration}
        delx:   extended;       {The change in x for partial differentials}
        i, j, k, w,             {Counter and indexing variables}
        bad:  integer;          {The successive increases in error}

begin                                                                   {Vecroot}
        writeln;
        writeln('ybase at entry:');
        for i:= 1 to ndim do
                writeln('ybase[',i:1,'] = ':4,ybase[i]);
        writeln;
        errlast := inf;
        for k := 1 to ndim do
                xbest[k] := x[k];
        for w := 1 to 25 do
                begin                                                   {for #1}
                        writeln('Vecroot');
                        writeln;
                        write('Phifinal = ',Phifinal:1:4);
                        writeln(' or ',Phifinal*180/pi:1:4,' deg.');
                        writeln;
                        for i := 1 to ndim do
                                writeln('x[', i:1, '] = ', x[i]:1:15);
                        Vecset(x, ybase);
                        writeln;
                        writeln('ybase after first Vecset; w = ',w:1);
                        for i:= 1 to ndim do
                                writeln('ybase[',i:1,'] = ':4,ybase[i]);
                        writeln;
                        err := 0;
                        for i := 1 to ndim do
                                err := err + sqr(ybase[i]);
                        err := sqrt(err);

                        writeln;
```

```
writeln('Vecroot Error: ', err:1:19);
writeln;
if (errlast > err) then
        begin                                                   {if #1}
                for j := 1 to ndim do
                        xbest[j] := x[j];
        end;                                                    {if #1}
if (err >= 10.0) then
        begin                                                   {if #2}
                for j := 1 to ndim do
                        x[j] := xbest[j];
                writeln('Error too large: Aborted.');
                return;
        end;                                                    {if #2}

if (w > 5) and (err >= errlast) then
        begin                                                   {if #3}
                bad:=bad+1;
                if (bad=2) then
                        begin                                   {if #4}
                                for j := 1 to ndim do
                                        x[j] := xbest[j];
                                for i := 1 to ndim do
                                        begin                   {for #2}
                                                write('xbest[', i:1, '] =' );
                                                  writeln(x[i]:1:15);
                                        end;                    {for #2}
                                readln;
                                return;
                        end;                                    {if #4}
        end                                                     {if #3}
else bad:=0;
if (errlast > err) then
        errlast := err;
writeln;
for j := 1 to ndim do
        begin                                                   {for #3}
                delx:=x[j]/100;
                if (j=1) then delx:=1;
                x[j] := x[j] + delx;
                write('Calculating Jacobian: ');
                writeln('varying initial conditions = ',j:1);
                Vecset(x, y);
                for i := 1 to ndim do
                        jacob[i,j] := (y[i] - ybase[i])/delx;
                x[j] := x[j] - delx;

        end;                                                    {for #3}
writeln;
for i:= 1 to ndim do ybase[i]:=-ybase[i];
GaussianElim(jacob, ybase, x1, ndim);
```

```
                    for i := 1 to ndim do
                        x[i] := x[i] + x1[i];
                end;                                                    {for #1}
        writeln('Solution not found in "Vecroot"');
end;                                                                    {Vecroot}

{*************************************************************************}
{************************** Main Program *********************************}
{*************************************************************************}
begin                                                                   {Main}
        rewrite(f1,'SSail3dOut');
        rewrite(f2,'SMom3dOut');
        x[3] := {assigned by user};
        x[4] := {assigned by user};
        x[5] := {assigned by user};
        x[6] := {assigned by user};
        x[7] := {assigned by user};
        doit := FALSE;
        inc := 23.443597/RAD;
        Parki := 28.5/RAD;
        rPark := 6378.145 + 500;
        Vcpark := sqrt(GMearth/rPark);
        Vpara := sqrt(2*GMearth/rPark);
        DeltaV := (Vpara-Vcpark);
        tstar := sqrt(1.0/GM)/86400.0;
        ao := ((0.000001)/AU)/GM;
        CaseEntry (JlaunchT, tstar, ao, x);
        Steep(x, y, 7);
        Vecroot(x, y, 7);
        writeln;
        doit := TRUE;
        InitialConditions(JlaunchT, Jcentury1, Jcentury2, JarriveT, t, ao, Vr, Vphi, r, phi,
                          PVr, PVphi, Pr, Tfinal, Phiearth, Phifinal, x, departure,
                          target, Phiplanet);
        Beta := CalcBeta (PVphi, PVpsi);
        Theta := CalcTheta(PVr, PVphi, PVpsi, Beta);
        Caldate := JlaunchT;
        InvConvert (Caldate);
        Hamil := Pr*Vr + Ppsi*Vpsi/r;
        Hamil := Hamil + PVr*((sqr(Vphi)+sqr(Vpsi))/r - 1/(r*r)) +
                         PVr*(ao*cos(Theta)*cos(Theta)*cos(Theta)/(r*r));
        Hamil := Hamil + PVphi*(-Vr*Vphi/r + Vphi*Vpsi*tan(Psi)/r) +
                         PVphi*(ao*sin(Theta)*cos(Theta)*cos(Theta)*sin(Beta)/(r*r));
        Hamil := Hamil + PVpsi*(-Vr*Vpsi/r - Vphi*Vphi*tan(Psi)/r) +
                         PVpsi*(ao*sin(Theta)*cos(Theta)*cos(Theta)*cos(Beta)/(r*r));
        writeln('Launch Date: ',Caldate:1:6);
        writeln('Julian Launch Date: ',trunc(JlaunchT));
        writeln;
        writeln('Required Delta V: ',DeltaV:1:5,' km/s');
        writeln;
        writeln('Tfinal = ',Tfinal:1:5, ' = ',Tfinal*tstar:1:5);
```

```
        writeln(f1,'Time', 'radius':10, 'Phi':12, 'Psi':12,'Vr':10, 'Vphi':10, 'Vpsi':10);
        write(f2,'Theta', 'Beta':10, 'Pr':10, 'Ppsi':10,'PVr':11, 'PVphi':11);
        writeln(f2,'PVpsi':10,'H':7);
        writeln('Theta:', Theta*RAD:1:2, '  Beta: ', Beta*RAD:1:2, '  H:',Hamil:1:5);
        readln;
        write('Time:', (t*tstar):1:4, '  r:', r:1:6, '  Phi:', Phi*RAD:1:2);
        write('  Psi:', Psi*RAD:1:2,'  Vr:', Vr:1:4, '  Vphi:', Vphi:1:4);
        writeln('  Vpsi:', Vpsi:1:4, '  H:',Hamil:1:5);
        write(f1, (t*tstar):3:2, r:10:6, Phi*RAD:12:6, Psi*RAD:12:6, Vr:10:6, Vphi:10:6);
        writeln(f1, Vpsi:10:6);
        write(f2, Theta*RAD:2:6, Beta*RAD:10:6, Pr:10:6, Ppsi:10:6);
        writeln(f2, PVr:10:6, PVphi:10:6, PVpsi:10:6, Hamil:8:2);
        t:=0.0;
        while (t <= Tfinal) do
                begin                                                    {while #1}
                        RungeKutta(r,Phi,Psi,Vr,Vphi,Vpsi,PVr,PVphi,PVpsi,Pr,Ppsi,
                                Theta,Beta,ao,Hamil);
                        write('Time:', (t*tstar):1:4, '  r:', r:1:6, '  Phi:', Phi*RAD:1:2);
                        write('  Psi:', Psi*RAD:1:2,'  Vr:', Vr:1:4, '  Vphi:', Vphi:1:4);
                        writeln('  Vpsi:', Vpsi:1:4, '  H:',Hamil:1:5);
                        write(f1, (t*tstar):3:2, r:10:6, Phi*RAD:12:6, Psi*RAD:12:6,
                                Vr:10:6);
                        writeln(f1, Vphi:10:6, Vpsi:10:6);
                        write(f2, Theta*RAD:2:6, Beta*RAD:10:6, Pr:10:6, Ppsi:10:6);
                        writeln(f2, PVr:10:6, PVphi:10:6, PVpsi:10:6, Hamil:8:2);
                        t := t + dt;
                end;                                                     {while #1}
        write('R Final: ',rfinal:1:5, '  Phi Final: ',Phifinal*RAD:1:5);
        writeln('  Psi Final: ',Psifinal*RAD:1:5);
        write('Vr Final: ',Vrfinal:1:5 , '     Vphi Final: ',Vphifinal:1:5);
        writeln('  Vpsi Final:  ', Vpsifinal:1:5);
        close(f1);
        close(f2);
end.                                                                     {Main}
```

Vita

Allen McCarley was born on December 19, 1966 in Honolulu, Hawaii. At this time his father was finishing an enlistment in the United States Marine Corp. Soon thereafter his parents returned to South Carolina where he spent most of his youth. With formative years during the height of the Apollo program, Allen soon developed an interest in aerospace. After high school he entered a dual degree program in place between Erskine College and the Georgia Institute of Technology. In 1991 he earned his Bachelor of Science, *cum Laude*, in physics from Erskine and his Bachelor of Aerospace Engineering, with honor, from Georgia Tech. Allen enrolled in graduate school at the University of Tennessee Space Institute immediately after graduation and began attendance after a six month period of employment as a mechanical engineer. During his enrollment Allen worked as a graduate research assistant for Dr. Gary A. Flandro. His primary research area was interplanetary trajectory optimization and mission planning. Allen earned his Master of Science in aerospace engineering in August of 1993.