

Program ipto_soc

Ballistic Interplanetary Trajectory Optimization with SOCS

This document is the user's manual for a Fortran computer program called `ipto_soc` that uses the Sparse Optimal Control Software (SOCS) object code library developed by Boeing (www.boeing.com/phantom/socs/) to solve the classic one impulse flyby and two-impulse rendezvous interplanetary trajectory optimization problems. The software attempts to minimize the launch delta-v, the arrival delta-v or the total delta-v for the interplanetary transfer. The type of trajectory optimization is specified by the user.

The important features of this scientific simulation are as follows:

- one impulse, *patched-conic* interplanetary *flyby* trajectory modeling
- two impulse, *patched-conic* interplanetary *rendezvous* trajectory modeling
- heliocentric, inertial cartesian equations of motion with point-mass planetary perturbations
- elliptical, non-coplanar planetary and asteroid/comet orbits
- user-selected Meeus, SLP96 or DE405 planetary ephemeris model

SOCS is a *direct transcription method* that can be used to solve a variety of trajectory optimization problems using the following combination of numerical methods:

- collocation and *implicit* integration
- adaptive mesh refinement
- sparse nonlinear programming

Additional information about the mathematical techniques and numerical methods used in SOCS can be found in the book, "Practical Methods for Optimal Control Using Nonlinear Programming" by John. T. Betts, SIAM, 2001.

The `ipto_soc` software consists of Fortran routines that perform the following tasks:

- main program that sets algorithm control parameters and calls the SOCS transcription/optimal control subroutine
- define problem definition and perform initialization related to scaling, lower and upper bounds, initial conditions, etc.
- evaluate the *right-hand-side* differential equations
- define and compute any point and path constraints
- display the optimal solution results

SOCS will use this information to *automatically* transcribe the user's problem and perform the optimization using a sparse nonlinear programming method. The software allows the user to select the type of collocation method and other important algorithm control parameters.

Typical Input File

The `ipto_soc`s computer program is “data-driven” by a simple user-created text file. The following is a typical input or “simulation definition” file used by the software. This example is an Earth-to-Mars rendezvous trajectory that minimizes the total delta-v for the mission. Please note for a flyby trajectory input file (`trajectory type = 1`), the only valid optimization option is `1 = minimize launch delta-v`.

In the following discussion the actual input file contents are in *courier* font and all explanations are in *times italic* font.

Each data item within an input file is preceded by one or more lines of *annotation* text. Do not delete any of these annotation lines or increase or decrease the number of lines reserved for each comment. However, you may change them to reflect your own explanation. The annotation line also includes the correct units and when appropriate, the valid range of the input. ASCII text input is not case sensitive but must be spelled correctly.

The first six lines of any input file are reserved for user comments. These lines are ignored by the software. However the input file must begin with six and only six initial text lines.

```
*****
** impulsive delta-v interplanetary trajectory optimization
** patched-conic heliocentric motion - ipto_soc
** Mars '03 - mars03.in
** February 15, 2006
*****
```

The first program input is an integer that defines the type of delta-v optimization. Please note that option 4, no optimization simply solves the orbital two-point boundary value problem.

```
*****
* simulation type *
*****
1 = minimize launch delta-v
2 = minimize arrival delta-v
3 = minimize total delta-v
4 = no optimization
-----
3
```

*The next input is an integer that specifies the type of planetary ephemeris model to use during the simulation. **Important note:** the Meeus algorithm (option 1) does not include an ephemeris for Pluto.*

```
ephemeris type (1 = Meeus, 2 = SLP96, 3 = DE405)
3
```

The next input is an integer that tells the simulation what type of trajectory to model.

```
trajectory type (1 = flyby, 2 = rendezvous)
2
```

The next three inputs are the user's initial guess for the launch calendar date. Be sure to include all four digits of the calendar year.

```
launch calendar date initial guess (month, day, year)
6,1,2003
```

The software allows the user to specify an initial guess for the launch and arrival calendar dates and lower and upper bounds on the actual dates found during the optimization process. For any guess for launch time t_L and user-defined launch time lower and upper bounds Δt_l and Δt_u , the launch time t is constrained as follows:

$$t_L - \Delta t_l \leq t \leq t_L + \Delta t_u$$

Likewise, for any guess for arrival time t_A and user-defined arrival time bounds Δt_l and Δt_u , the arrival time t is constrained as follows:

$$t_A - \Delta t_l \leq t \leq t_A + \Delta t_u$$

For fixed launch and/or arrival times, the lower and upper bounds are set to 0.

The next two inputs are the lower and upper bounds for the launch calendar date search interval. These values should be input in days.

```
launch date search boundary (days)
-30, +30
```

The next program input is an integer that specifies the launch planet.

```
*****
* launch planet *
*****
1 = Mercury
2 = Venus
3 = Earth
4 = Mars
5 = Jupiter
6 = Saturn
7 = Uranus
8 = Neptune
9 = Pluto
-----
3
```

The next set of inputs defines the user's initial guess for the arrival calendar date, the search interval and the arrival planet/comet/asteroid.

```
arrival calendar date initial guess (month, day, year)
12,1,2003

arrival date search boundary (days)
-30, +30

*****
* arrival celestial body *
*****
```

```

1 = Mercury
2 = Venus
3 = Earth
4 = Mars
5 = Jupiter
6 = Saturn
7 = Uranus
8 = Neptune
9 = Pluto
0 = asteroid/comet
-----

```

4

The next series of inputs include the name and classical orbital elements of a comet or asteroid (arrival celestial body = 0). Please note that the angular orbital elements must be specified with respect to a heliocentric, Earth mean ecliptic and equinox of J2000 coordinate system.

```

*****
* asteroid/comet classical orbital elements *
* (heliocentric, Earth mean ecliptic J2000) *
*****

asteroid/comet name
Tempel 1

calendar date of perihelion passage (month, day, year)
7, 5.3153, 2005

perihelion distance (au)
1.506167

orbital eccentricity (non-dimensional)
0.517491

orbital inclination (degrees)
10.5301

argument of perihelion (degrees)
178.8390

longitude of the ascending node (degrees)
68.9734

```

This next input specifies the type of comma-delimited or comma-separated-variable (CSV) solution data file to create. Option 1 will create a solution file at each collocation point or node determined by SOCS. Options 2 and 3 allow the user to specify either the number of nodes (option 2) or time step size of the data file (option 3).

```

*****
* type of comma-delimited solution data file *
*****
1 = SOCS-defined nodes
2 = user-defined nodes
3 = user-defined step size
-----

```

1

For options 2 or 3, this next input defines either the number of data points (option 2) or the time step size of the data output in the solution file (option 3).

number of user-defined nodes or print step size in solution data file
1

The name of the solution data file is defined in this next line. Please consult Appendix B of this document for a description of the information written to this file.

name of solution output file
mars03.csv

The next series of program inputs are algorithm control options and parameters for the SOCS software. The first input is an integer that specifies the type of collocation method to use during the solution process. For most simulations, the trapezoidal method is recommended.

```
*****  
* algorithm control parameters *  
*****  
  
discretization/collocation method  
-----  
1 = trapezoidal  
2 = separated Hermite-Simpson  
3 = compressed Hermite-Simpson  
4 = Runge-Kutta 4-stage  
-----  
1
```

The next input defines the relative error in the objective function.

relative error in the objective function (performance index)
1.0d-5

The next input defines the relative error in the solution of the differential equations.

relative error in the solution of the differential equations
1.0d-7

The next input is an integer that defines the maximum number of mesh refinement iterations.

maximum number of mesh refinement iterations
20

The next input is an integer that defines the maximum number of function evaluations.

maximum number of function evaluations
50000

The next input is an integer that defines the maximum number of algorithm iterations.

maximum number of algorithm iterations
10000

The level of output from the SOCS NLP algorithm is controlled with the following integer input.

```
*****  
sparse NLP iteration output  
-----  
1 = none  
2 = terse  
3 = standard  
4 = interpretive
```

```
5 = diagnostic
-----
2
```

The level of output from the SOCS optimal control algorithm is controlled with the following integer input. Please note that option 4 will create lots of information.

```
*****
optimal control output
-----
1 = none
2 = terse
3 = standard
4 = interpretive
-----
1
```

The level of output from the SOCS differential equations algorithm is controlled with the following integer input. Please note that option 5 will create lots of information.

```
*****
differential equation output
-----
1 = none
2 = terse
3 = standard
4 = interpretive
5 = diagnostic
-----
1
```

The level of output can be further controlled by the user with this final text input. This program option sets the value of the SOCOUT character variable described in the SOCS user's manual. To ignore this special output control, input the simple character string no.

```
*****
user-defined output
-----
input no to ignore
-----
a0b0c0d0e0f0g0h0i0j2k0l0m0n0o0p0q0r0
```

Optimal Control Solution and Trajectory Plot

The following is the program output created by the `ipto_soc`s simulation for this example. The first part of the solution display summarizes important information about the type of trajectory, the ephemeris used, the type of optimization and the launch conditions. The second part of the display summarizes the arrival conditions and characteristics of the planetary and transfer orbits. Please see Appendix B of this document for additional details about this information.

```
program iptoc_soc
*****
RENDEZVOUS TRAJECTORY
*****
```

minimize total delta-v

DE405 ephemeris

LAUNCH CONDITIONS
=====

calendar date 06/06/2003
ephemeris time 08:18:17
julian date 2452796.846029010135680
launch delta-vx 2.900355257132376 km/sec
launch delta-vy -6.167147321180659E-001 km/sec
launch delta-vz -4.017140829781655E-002 km/sec
launch delta-v 2.965469848174399 km/sec

launch hyperbola characteristics
(Earth mean equator and equinox of J2000)

right ascension 349.265306600026236 degrees
declination -5.460089518808275 degrees
launch energy 8.794011420431495 (km/sec)**2

heliocentric orbital elements and state vector of the departure planet
(Earth mean ecliptic and equinox of J2000)

sma (au) eccentricity inclination (deg) argper (deg)
0.1000231777D+01 0.1633787507D-01 0.3276443858D-03 0.2718842281D+03
raan (deg) true anomaly (deg) arglat (deg) period (days)
0.1902737401D+03 0.1530415749D+03 0.6492580297D+02 0.3653838927D+03
rx (km) ry (km) rz (km) rmag (km)
-.3877866494D+08 -.1467666162D+09 0.7862744466D+03 0.1518032426D+09
vx (kps) vy (kps) vz (kps) vmag (kps)
0.2832127058D+02 -.7711224239D+01 0.7227420337D-04 0.2935229713D+02

heliocentric orbital elements and state vector of the spacecraft
(Earth mean ecliptic and equinox of J2000)

sma (au) eccentricity inclination (deg) argper (deg)
0.1259528055D+01 0.1943611189D+00 0.7110274174D-01 0.1789332195D+03
raan (deg) true anomaly (deg) arglat (deg) period (days)
0.7543868349D+02 0.8276398802D+00 0.1797608594D+03 0.5163095942D+03
rx (km) ry (km) rz (km) rmag (km)
-.3877866494D+08 -.1467666162D+09 0.7862744462D+03 0.1518032426D+09

vx (kps)	vy (kps)	vz (kps)	vmag (kps)
0.3122162584D+02	-.8327938971D+01	-.4009913409D-01	0.3231324954D+02

ARRIVAL CONDITIONS
=====

calendar date	12/27/2003
ephemeris time	16:57:02
julian date	2453001.206273247953504
arrival delta-vx	2.021800006131718 km/sec
arrival delta-vy	-1.614337031143630 km/sec
arrival delta-vz	-7.794055761964022E-001 km/sec
arrival delta-v	2.702079267364651 km/sec
arrival energy	7.301232367121890 (km/sec)**2
total delta-v	5.667549115539050 km/sec
transfer time	204.360244237817824 days

heliocentric orbital elements and state vector of the planet/asteroid/comet
(Earth mean ecliptic and equinox of J2000)

sma (au)	eccentricity	inclination (deg)	argper (deg)
0.1523678538D+01	0.9354080420D-01	0.1849371902D+01	0.2865170861D+03
raan (deg)	true anomaly (deg)	arglat (deg)	period (days)
0.4954092163D+02	0.7248486344D+02	0.3590019496D+03	0.6869710761D+03
rx (km)	ry (km)	rz (km)	rmag (km)
0.1454913687D+09	0.1646992409D+09	-.1235318254D+06	0.2197580342D+09
vx (kps)	vy (kps)	vz (kps)	vmag (kps)
-.1723360237D+02	0.1810899919D+02	0.8028149607D+00	0.2501154562D+02

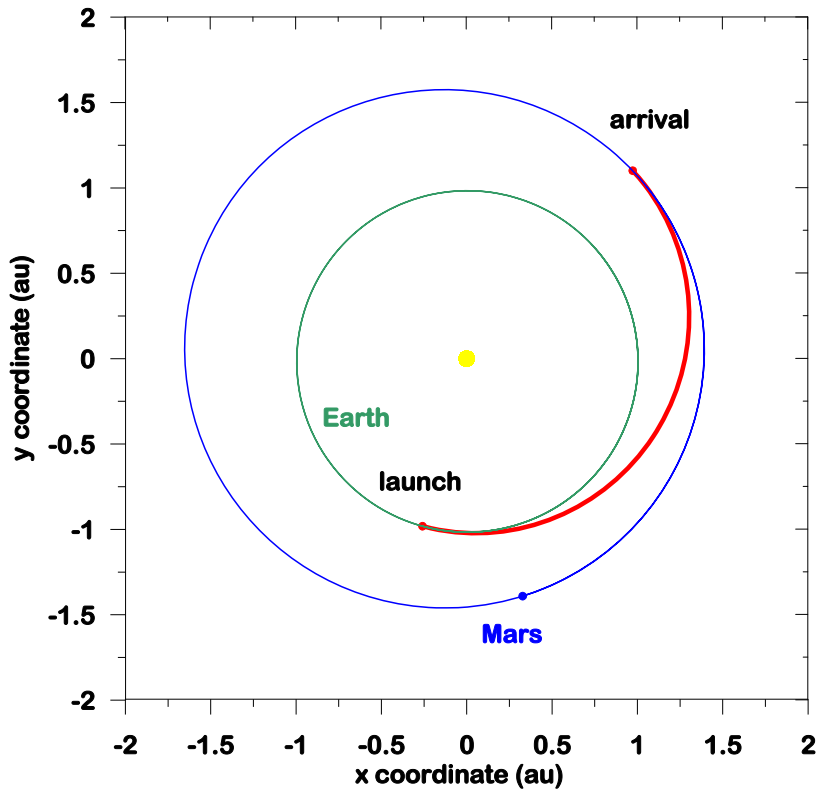
heliocentric orbital elements and state vector of the spacecraft
(Earth mean ecliptic and equinox of J2000)

sma (au)	eccentricity	inclination (deg)	argper (deg)
0.1523678538D+01	0.9354080429D-01	0.1849371902D+01	0.2865170863D+03
raan (deg)	true anomaly (deg)	arglat (deg)	period (days)
0.4954092163D+02	0.7248486326D+02	0.3590019496D+03	0.6869710765D+03
rx (km)	ry (km)	rz (km)	rmag (km)
0.1454913687D+09	0.1646992410D+09	-.1235318254D+06	0.2197580342D+09
vx (kps)	vy (kps)	vz (kps)	vmag (kps)
-.1723360237D+02	0.1810899919D+02	0.8028149607D+00	0.2501154562D+02

The `ipto_soc`s software will create two comma-separated-variable (csv) output files. The first file contains the heliocentric, ecliptic state vector of the spacecraft and the second file (`planets.csv`) contains the state vectors of the planet and the destination celestial body.

The following is the transfer trajectory for this example. It is a view of the trajectory and planetary orbits from the north pole of the ecliptic looking down on the ecliptic plane.

Interplanetary Trajectory Analysis Earth-to-Mars Minimum Total Delta-V



Problem setup for SOCS

This section provides additional details about the software implementation. For good scaling during the optimization, the time unit used in all internal calculations is days, position is expressed in astronomical units, and the velocity and delta-v unit is astronomical units per day.

(1) Launch and arrival time bounds

The software allows the user to specify an initial guess for the launch and arrival calendar dates and bounds on the actual dates found during the optimization process. For any guess for launch time t_L and user-defined launch time search bound Δt_L , the launch time t is constrained as follows:

$$t_L - \Delta t_L \leq t \leq t_L + \Delta t_L$$

Likewise, for any guess for arrival time t_A and user-defined arrival time bound Δt_A , the arrival time t is constrained as follows:

$$t_A - \Delta t_A \leq t \leq t_A + \Delta t_A$$

For fixed launch and/or arrival times, the lower and upper bounds are set to 0.

(2) Performance index – minimize delta-v

The objective function or performance index J for this simulation is one of three possible delta-v's. For this classic trajectory optimization problem, this index is simply

$$J = \Delta V$$

where ΔV is either the launch delta-v, arrival delta-v or the total delta-v. The value of the `maxmin` indicator in SOCS tells the software whether the user is minimizing or maximizing the performance index. The type of delta-v is selected by the user.

(3) Point functions – position and velocity vector “matching” at launch

For any launch time t_L , the optimal solution for a rendezvous trajectory must satisfy the following state vector boundary conditions (equality constraints) at launch:

$$\begin{aligned} \mathbf{r}_{s/c}(t_L) - \mathbf{r}_p(t_L) &= 0 \\ \mathbf{v}_{s/c}(t_L) - \{\mathbf{v}_p(t_L) + \Delta \mathbf{v}(t_L)\} &= 0 \end{aligned}$$

where $\mathbf{r}_{s/c}$ and $\mathbf{v}_{s/c}$ are the heliocentric, inertial position and velocity vectors of the spacecraft at the launch time t_L , \mathbf{r}_p and \mathbf{v}_p are the heliocentric, inertial position and velocity vectors of the departure planet at the launch time, and $\Delta \mathbf{v}$ is the *impulsive* delta-v vector required at launch.

(4) Point functions – position and/or velocity vector “matching” at arrival

For any arrival time t_A , the optimal solution for a rendezvous trajectory must satisfy the following state vector boundary conditions at arrival:

$$\begin{aligned} \mathbf{r}_{s/c}(t_A) - \mathbf{r}_p(t_A) &= 0 \\ \mathbf{v}_{s/c}(t_A) - \{\mathbf{v}_p(t_A) + \Delta \mathbf{v}(t_A)\} &= 0 \end{aligned}$$

where $\mathbf{r}_{s/c}$ and $\mathbf{v}_{s/c}$ are the heliocentric, inertial position and velocity vectors of the spacecraft at the arrival time t_A , and \mathbf{r}_p and \mathbf{v}_p are the heliocentric, inertial position and velocity vectors of the destination planet at the arrival time, and $\Delta \mathbf{v}$ is the *impulsive* delta-v vector required at arrival..

This system of launch and arrival state vector equality constraints ensures a rendezvous mission. For a flyby mission, the velocity vector point functions at arrival are not enforced.

The components of the departure and arrival impulsive delta-v's are the optimization parameters used by SOCS to solve this trajectory problem.

Initial Guess

An initial guess for the launch and arrival impulsive delta-v vectors can be determined from the solution of the Lambert two-point boundary-value problem (TPBVP). Lambert's Theorem states that the time to traverse a trajectory depends only upon the length of the semimajor axis a of the transfer trajectory, the sum $r_i + r_f$ of the distances of the initial and final positions relative to a central body, and the length c of the chord joining these two positions.

The Lambert solution that initializes the `ipto_soc`s software uses the user's initial guess for launch and arrival dates. The dynamic variables at each grid point are determined by SOCS by setting the initial guess option `INIT(1) = 6` with `INIT(2) = 4`.

The algorithm used in this computer program is based on the method described in "A Practical Note on the Use of Lambert's Equation" by Geza Gedeon, *AIAA Journal*, Volume 3, Number 1, 1965, pages 149-150. This iterative solution is valid for elliptic, parabolic and hyperbolic transfer orbits which may be either posigrade or retrograde, and involve one or more revolutions about the central body. Additional information can also be found in G. S. Gedeon, "Lambertian Mechanics", Proceedings of the 12th International Astronautical Congress, Vol. I, 172-190.

The *elliptic* form of the general Lambert Theorem is

$$t = \sqrt{\frac{a^3}{\mu}} \left[(1-k)m\pi + k(\alpha - \sin \alpha) \mp (\beta - \sin \beta) \right]$$

where k may be either +1 (posigrade) or -1 (retrograde), and m is the number of revolutions about the central body.

The Gedeon algorithm introduces the following variable

$$z = \frac{s}{2a}$$

and solves the problem with a Newton-Raphson procedure. In this equation, a is the semimajor axis of the transfer orbit and

$$s = \frac{r_1 + r_2 + c}{2}$$

This algorithm also makes use of the following constant:

$$w = \pm \sqrt{1 - \frac{c}{s}}$$

The function to be solved iteratively is given by:

$$N(z) = \frac{1}{z|z|^{1/2} 2^{1/2}} \left\{ \frac{1-k}{2} m\pi + k \left[|z|^{1/2} - |z|^{1/2} (1-z)^{1/2} \right] - \left[w|z|^{1/2} - w|z|^{1/2} - w|z|^{1/2} (1-w^2 z)^{1/2} \right] \right\}$$

The Newton-Raphson algorithm also requires the derivative of this equation given by

$$N'(z) = \frac{dN}{dz} = \frac{1}{|z| 2^{1/2}} \left\{ \frac{k}{(1-z)^{1/2}} - \frac{w^3}{(1-w^2 z)^{1/2}} - \frac{3N(z)}{2^{1/2}} \right\}$$

The iteration for z is as follows:

$$z_{n+1} = z_n - \frac{N(z_n)}{N'(z_n)}$$

The heliocentric state vector during the interplanetary cruise part of the mission is computed using Sheppard's two-body orbit propagation algorithm. The derivation of this algorithm is described in "Universal Keplerian State Transition Matrix", *Celestial Mechanics*, **35** (1985) 129-144. This numerical method uses a combination of universal variables and continued fractions to propagate two-body orbits.

Technical Discussion

The general vector equation for *point-mass* perturbations such as the Moon or planets is given by

$$\ddot{\mathbf{r}} = - \sum_{j=1}^n \mu_j \left[\frac{\mathbf{d}_j}{d_j^3} + \frac{\mathbf{s}_j}{s_j^3} \right]$$

In this equation, \mathbf{s}_j is the vector from the primary body to the secondary body j , μ_j is the gravitational constant of the secondary body and $\mathbf{d}_j = \mathbf{r} - \mathbf{s}_j$, where \mathbf{r} is the position vector of the spacecraft relative to the primary body.

To avoid numerical problems, use is made of Battin's $F(q)$ function given by

$$F(q_k) = q_k \left[\frac{3 + 3q_k + q_k^2}{1 + (\sqrt{1 + q_k})^3} \right]$$

where

$$q_k = \frac{\mathbf{r}^T (\mathbf{r} - 2\mathbf{s}_k)}{\mathbf{s}_k^T \mathbf{s}_k}$$

The third-body acceleration can now be expressed as

$$\ddot{\mathbf{r}} = -\sum_{k=1}^n \frac{\mu_k}{d_k^3} [\mathbf{r} + F(q_k)\mathbf{s}_k]$$

The first-order system of equations required by SOCS can be created from the second-order system by the method of *order reduction*. With the following definitions,

$$\begin{aligned} y_1 &= r_x & y_2 &= r_y & y_3 &= r_z \\ y_4 &= v_x & y_5 &= v_y & y_6 &= v_z \end{aligned}$$

where v_x, v_y, v_z are the velocity vector components of the spacecraft, the first-order system of differential equations is given by

$$\begin{aligned} \dot{y}_1 &= v_x \\ \dot{y}_2 &= v_y \\ \dot{y}_3 &= v_z \\ \dot{y}_4 &= -\mu_s \frac{r_x}{r^3} + a_x \\ \dot{y}_5 &= -\mu_s \frac{r_y}{r^3} + a_y \\ \dot{y}_6 &= -\mu_s \frac{r_z}{r^3} + a_z \end{aligned}$$

In these equations, μ_s is the gravitational constant of the sun, and a_x, a_y and a_z are the x, y and z gravitational contributions of the planets. All planetary perturbations except those due to the launch and arrival planets are included in the equations of motion. The equations described here are coded in the right-hand-side subroutine required by SOCS.

The software models the planetary coordinates using either a Meeus algorithm, the DE405 model from JPL or the SLP96 algorithm from the Bureau of Longitudes. The planetary ephemerides used in this software are mean orbital elements relative to the Earth ecliptic and equinox of J2000.

The Meeus ephemeris option is based on the algorithm described in chapter 30 of *Astronomical Algorithms* by Jean Meeus. Each orbital element is represented by a cubic polynomial of the form

$$a_0 + a_1T + a_2T^2 + a_3T^3$$

where the fundamental time argument T is given by

$$T = \frac{JD - 2451545}{36525}$$

In this expression JD is the Julian date.

The orbital elements of an asteroid or comet relative to the ecliptic and equinox of J2000 coordinate system must be provided by the user. These elements can be obtained from the ASTCOM database at JPL (<http://ssd.jpl.nasa.gov>), the MPC database at Harvard (<http://cfa-www.harvard.edu>) or the Bureau of Longitudes in Paris (<http://www.bdl.fr>).

These orbital elements consist of the following items:

- calendar date of perihelion passage
- perihelion distance (AU)
- orbital eccentricity (non-dimensional)
- orbital inclination (degrees)
- argument of perihelion (degrees)
- longitude of ascending node (degrees)

The software determines the mean anomaly of the asteroid or comet at any simulation time using the following equation:

$$M = \sqrt{\frac{\mu_s}{a^3}} t_{pp} = \sqrt{\frac{\mu_s}{a^3}} (JD - JD_{pp})$$

where μ_s is the gravitational constant of the sun, a is the semimajor axis of the celestial body, and t_{pp} is the time since perihelion passage.

The semimajor axis is determined from the perihelion distance r_p and orbital eccentricity e according to

$$a = \frac{r_p}{(1 - e)}$$

This solution of Kepler's equation in this computer program is based on a numerical solution devised by Professor J.M.A. Danby at North Carolina State University. Additional information about this algorithm can be found in "The Solution of Kepler's Equation", *Celestial Mechanics*, **31** (1983) 95-107, 317-328 and **40** (1987) 303-312.

The initial guess for Danby's method is

$$E_0 = M + 0.85 \text{sign}(\sin M) e$$

The fundamental equation we want to solve is

$$f(E) = E - e \sin E - M = 0$$

which has the first three derivatives given by

$$f'(E) = 1 - e \cos E$$

$$f''(E) = e \sin E$$

$$f'''(E) = e \cos E$$

The iteration for an updated eccentric anomaly based on a current value E_n is given by the next four equations:

$$\Delta(E_n) = -\frac{f}{f'}$$

$$\Delta^*(E_n) = -\frac{f}{f' + \frac{1}{2}\Delta f''}$$

$$\Delta_n(E_n) = -\frac{f}{f' + \frac{1}{2}\Delta f'' + \frac{1}{6}\Delta^2 f'''}$$

$$E_{n+1} = E_n + \Delta_n$$

This algorithm provides quartic convergence of Kepler's equation. This process is repeated until the following convergence test involving the fundamental equation is satisfied:

$$|f(E)| \leq \varepsilon$$

where ε is the convergence tolerance. This tolerance is hardwired in the software to $\varepsilon = 1.0\text{e-}10$. Finally, the true anomaly can be calculated with the following two equations

$$\sin \theta = \sqrt{1 - e^2} \sin E$$

$$\cos \theta = \cos E - e$$

and the four quadrant inverse tangent given by

$$\theta = \tan^{-1}(\sin \theta, \cos \theta)$$

If the orbit is hyperbolic, the initial guess is

$$H_0 = \log\left(\frac{2M}{e} + 1.8\right)$$

where H_0 is the hyperbolic anomaly. The fundamental equation and first three derivatives for this case are as follows:

$$f(H) = e \sinh H - H - M$$

$$f'(H) = e \cosh H - 1$$

$$f''(H) = e \sinh H$$

$$f'''(H) = e \cosh H$$

Otherwise, the iteration loop which calculates Δ, Δ^* , and so forth is the same. The true anomaly for hyperbolic orbits is determined with this next set of equations:

$$\sin \theta = \sqrt{e^2 - 1} \sinh H$$

$$\cos \theta = e - \cosh H$$

The true anomaly is then determined from a four quadrant inverse tangent evaluation of these two equations.

The ΔV 's required at launch and arrival are simply the differences between the velocity on the transfer trajectory determined by the solution of Lambert's problem and the heliocentric velocities of the two planets. If we treat each planet as a point mass and assume *impulsive* maneuvers, the *planet-centered* magnitude and direction of the required maneuvers are given by the two vector equations:

$$\Delta \mathbf{V}_L = \mathbf{V}_{T_L} - \mathbf{V}_{P_L}$$

$$\Delta \mathbf{V}_A = \mathbf{V}_{T_A} - \mathbf{V}_{P_A}$$

where

\mathbf{V}_{T_L} = heliocentric velocity vector of the transfer trajectory at launch

\mathbf{V}_{T_A} = heliocentric velocity vector of the transfer trajectory at arrival

\mathbf{V}_{P_L} = heliocentric velocity vector of the launch planet

\mathbf{V}_{P_A} = heliocentric velocity vector of the arrival planet

The scalar magnitude of each maneuver is also called the “hyperbolic excess velocity” or V_∞ at launch and arrival. The hyperbolic excess velocity is the speed of the spacecraft relative to each planet at an *infinite* distance from the planet. Furthermore, the *energy* or C_3 at launch or arrival is equal to V_∞^2 for the respective maneuver. C_3 is also equal to twice the orbital energy per unit mass (the specific orbital energy).

The orientation of the departure hyperbola is specified in terms of the right ascension and declination of the asymptote. These coordinates can be calculated using the components of the V_∞ velocity vector.

The right ascension of the asymptote is determined from

$$\alpha = \tan^{-1}(\Delta V_y, \Delta V_z)$$

and the geocentric declination of the asymptote is given by

$$\delta = 90^\circ - \cos^{-1}(\Delta \hat{V}_z)$$

where $\Delta \hat{V}_z$ is z-component of the unit ΔV vector. The right ascension is computed using a four quadrant inverse tangent function.

In this computer program the heliocentric planetary coordinates and therefore the ΔV vectors are computed in the Earth mean ecliptic and equinox of J2000 coordinate system. In order to determine the orientation of the departure and arrival hyperbolas, these ΔV vectors must be transformed to the equatorial frame.

The required transformation is given by

$$\Delta \mathbf{V}_{eq} = \begin{bmatrix} 1 & -0.000000479966 & 0 \\ 0.000000440360 & 0.917482137087 & 0.397776982902 \\ -0.000000190919 & -0.397776982902 & 0.917482137087 \end{bmatrix} \Delta \mathbf{V}_{ec}$$

where $\Delta \mathbf{V}_{ec}$ is the delta-velocity vector in the ecliptic frame, and $\Delta \mathbf{V}_{eq}$ is the delta-velocity vector in the equatorial frame.

APPENDIX A

Compiling and Running the Software

This appendix describes how to compile and run the `ipto_soc`s computer program. This software was created using version 6.4.3 of SOCS and Compaq Visual Fortran.

A DOS/Windows version of `ipto_soc`s using Compaq Visual Fortran version 6.6C can be created with the following command:

```
f132 /arch:host ipto_soc.f *.for c:\socs\socs643.lib advapi32.lib
```

This command assumes the SOCS library is located in the subdirectory `c:\socs`.

An input file created by the user can be run from the command line or a simple batch file with a statement similar to the following:

```
ipto_soc mars03.in
```

If the software is executed without an input file on the command line, the computer program will display the following information screen and file name prompt:

```
*****
*           Program ipto_soc           *
*                                       *
*       interplanetary trajectory      *
*       optimization with SOCS         *
*                                       *
*           February 15, 2006          *
*****
```

```
please input the name of the simulation definition file
```

The source code that reads the name of an input file included on the command line is

```
c   if present, use commandline argument #1 for input file
    call getarg(1, inputfname$, istatus)
```

The source code that creates the file name input prompt is as follows:

```
c   clear screen

    isys = system("cls")

    if (istatus .eq. -1) then
c   *****
c   input filename not on command line
c   request name of simulation definition input file
c   *****
```

```

print *, ' '
print *, ' '

print *, ' *****'
print *, ' *          Program ipto_socs          *'
print *, ' *          *          *'
print *, ' *          interplanetary trajectory      *'
print *, ' *          optimization with SOCS         *'
print *, ' *          *          *'
print *, ' *          February 15, 2006              *'
print *, ' *****'
print *, ' '
print *, ' '

print *,
&      'please input the name of the simulation definition file'

      read (*, *) inputfname$
end if

```

If your compiler does not accept input from a command line, you will have to modify this source code for your particular Fortran compiler. You may also choose to eliminate the code that accepts a command line input file. Please note also that your compiler may have a different command to clear the screen.

Important Note

The binary ephemeris files provided with this computer program were created for use on PC-compatible computers. For other platforms, you will need to create binary files specific to that system. Information and computer programs for creating these files can be found at the JPL solar system FTP site located at <ftp://ssd.jpl.nasa.gov/pub/eph/export/>.

APPENDIX B

Contents of the Simulation Summary and CSV Files

This appendix is a brief summary of the information contained in the simulation summary screen displays and the CSV data files produced by the `ipto_socs` software. All output except the coordinates of the launch hyperbola is computed and displayed in a heliocentric, Earth mean ecliptic and equinox of J2000 coordinate system.

The simulation summary screen display contains the following information:

`calendar date` = calendar date of trajectory event
`ephemeris time` = ephemeris time of trajectory event
`julian date` = julian date of trajectory event
`right ascension` = right ascension of the launch hyperbola in degrees
`declination` = declination of the launch hyperbola in degrees
`sma (au)` = semimajor axis in astronomical unit
`eccentricity` = orbital eccentricity (non-dimensional)
`inclination (deg)` = orbital inclination in degrees
`argper (deg)` = argument of perigee in degrees
`raan (deg)` = right ascension of the ascending node in degrees
`true anomaly (deg)` = true anomaly in degrees
`arglat (deg)` = argument of latitude in degrees. The argument of latitude is the sum of true anomaly and argument of perigee.
`period (days)` = orbital period in days
`delta-vx` = x-component of the impulsive velocity vector in kilometers/second
`delta-vy` = y-component of the impulsive velocity vector in kilometers/second
`delta-vz` = z-component of the impulsive velocity vector in kilometers/second
`delta-v` = scalar magnitude of the impulsive delta-v in kilometers/seconds
`energy (km/sec)` = twice specific orbital energy in $\text{km}(\text{sec})^{**2}$

The comma-separated-variable disk file is created by the `odeprt` subroutine and contains the following information:

`time (days)` = simulation time since launch in days
`rx (au)` = x-component of spacecraft position vector in astronomical units
`ry (au)` = y-component of spacecraft position vector in astronomical units
`rz (au)` = z-component of spacecraft position vector in astronomical units

rmag (au) = heliocentric radius magnitude of spacecraft in astronomical units
vx (km/sec) = x-component of spacecraft velocity vector in kilometers per second
vy (km/sec) = y-component of spacecraft velocity vector in kilometers per second
vz (km/sec) = z-component of spacecraft velocity vector in kilometers per second
vmag (km/sec) = heliocentric velocity of spacecraft in kilometers per second
semimajor axis (au) = semimajor axis in astronomical units
eccentricity = orbital eccentricity (non-dimensional)
inclination (deg) = orbital inclination in degrees
arg of perigee (deg) = argument of perigee in degrees
raan (deg) = right ascension of the ascending node in degrees
true anomaly (deg) = true anomaly in degrees
period (days) = orbital period in days

The `planets.csv` file contains the following information:

time (days) = simulation time since launch in days
rp1-x (au) = x-component of the launch planet position vector in astronomical units
rp1-y (au) = y-component of the launch planet position vector in astronomical units
rp1-z (au) = z-component of the launch planet position vector in astronomical units
rp2-x (au) = x-component of the destination body position vector in astronomical units
rp2-y (au) = y-component of the destination body position vector in astronomical units
rp2-z (au) = z-component of the destination body position vector in astronomical units

APPENDIX C

Fortran Functions and Subroutines

This appendix is a brief summary of the major Fortran functions and subroutines included in the `ipto_soc`s computer program.

- `ipto_soc`s.f - SOCS main executive program
- `atan3`.for - four quadrant inverse tangent function
- `eci2orb`.for - convert eci position and velocity vectors to classical orbital elements subroutine
- `gdate`.for - compute calendar date from Julian date subroutine
- `jd2str`.for - from a Julian date, print the character representation of calendar date and time subroutine
- `jpleph`.for - subroutine that reads and interpolates a JPL ephemeris file
- `julian`.for - subroutine to convert calendar date to Julian date
- `kepler1`.for - solve Kepler's equation using Danby's method subroutine
- `linput`.for - read and echo a line of text from an input file subroutine
- `lambfunc`.for - solve Lambert's problem using Geodeon's method subroutine
- `odeinp`.for - SOCS simulation input subroutine
- `odepf`.for - SOCS point functions subroutine
- `odeprt`.for - SOCS print subroutine - creates comma-separated-variable file
- `oderhs`.for - SOCS subroutine that evaluates the equations of motion and any algebraic equations
- `oeprint`.for - subroutine that displays classical orbital elements
- `orb2eci`.for - convert classical orbital elements to position and velocity vectors subroutine
- `p2000`.for - subroutine that returns a planet's or asteroid/comet position and velocity vectors in kilometers and kilometers/second
- `readfpn`.for - read and echo floating point number from an input file subroutine
- `readint`.for - read and echo an integer from an input file subroutine
- `readtext`.for - read and echo text from an input file subroutine
- `slp96`.for - read and interpolate SLP96 ephemeris subroutine
- `svprint`.for - subroutine that displays position and velocity vectors

twobody2.for - subroutine that solves the two-body initial value problem
utility.for - number and text manipulation functions and subroutines
uvector.for - unit vector subroutine
vcross.for - vector cross product subroutine
vdot.for - vector dot product subroutine
vecmag.for - vector scalar magnitude function
xmod.for - modulo 2 pi function

APPENDIX D

Example Fortran Subroutine

This appendix contains the source code for a single Fortran 77 routine and illustrates typical programming conventions used in the `ipto_soc`s software. This subroutine is the point function routine required by the SOCS software.

```
      subroutine odepf(iphase, iphend, time, ydyn, nydyn, parm,
&                    nparm, ptf, nptf, iferr)

c     evaluate "position & velocity matching" point functions
c     and scalar delta-v objective function

c     *****

      implicit double precision (a-h, o-z)

      include 'socscopl.inc'

      parameter (zero = 0.0d0, one = 1.0d0)

      dimension ydyn(nydyn), parm(nparm), ptf(nptf)

      dimension rp(3), vp(3)

      if (iphase .eq. 1 .and. iphend .eq. -1) then
c     *****
c     "position & velocity match" at beginning of phase
c     for either rendezvous or flyby program option
c     *****

      xjdate = time + xjdateil

      call p2000(ip1, xjdate, rp, vp)

c     position match

      ptf(1) = ydyn(1) - rp(1) / aunit

      ptf(2) = ydyn(2) - rp(2) / aunit

      ptf(3) = ydyn(3) - rp(3) / aunit

c     velocity match

      ptf(4) = ydyn(4) - (vmult * vp(1) + parm(1))

      ptf(5) = ydyn(5) - (vmult * vp(2) + parm(2))

      ptf(6) = ydyn(6) - (vmult * vp(3) + parm(3))

      if (iopt .eq. 1 .or. iopt .eq. 3) then
c     *****
c     launch delta-v contribution to objective function
c     *****
```

```

        ptf(7) = sqrt(parm(1)**2 + parm(2)**2 + parm(3)**2)

    end if

end if

if (iphase .eq. 1 .and. iphend .eq. +1) then
c *****
c "position and/or velocity match" at end of phase
c *****

    xjdate = time + xjdatei1

    call p2000(ip2, xjdate, rp, vp)

c    position match

    ptf(1) = ydyn(1) - rp(1) / aunit

    ptf(2) = ydyn(2) - rp(2) / aunit

    ptf(3) = ydyn(3) - rp(3) / aunit

    if (itraj .eq. 2) then
c -----
c rendezvous option - include velocity match
c -----

        ptf(4) = ydyn(4) - (vmult * vp(1) + parm(4))

        ptf(5) = ydyn(5) - (vmult * vp(2) + parm(5))

        ptf(6) = ydyn(6) - (vmult * vp(3) + parm(6))
    end if

    if (iopt .eq. 2 .or. iopt .eq. 3) then
c *****
c arrival delta-v contribution to objective function
c *****

        ptf(7) = sqrt(parm(4)**2 + parm(5)**2 + parm(6)**2)

    end if

end if

return
end

```