

## Program escape

### Earth Escape Trajectory Optimization with SOCS

This document is the user's manual for a Fortran computer program called `escape` that uses the Sparse Optimal Control Software (SOCS) object code library developed by Boeing Phantom Works ([www.boeing.com/phantom/socs/](http://www.boeing.com/phantom/socs/)) to solve the finite-burn, Earth escape trajectory optimization problem. The software models the trajectory as a single, finite-burn propulsive maneuver. This computer program attempts to maximize the final spacecraft mass. Since this simulation involves a single continuous maneuver, this is equivalent to minimizing the required propellant mass.

The important features of this scientific simulation are as follows:

- single, continuous thrust orbital maneuver
- variable attitude steering
- constant propulsive thrust magnitude
- modified equinoctial equations of motion with oblate Earth gravity model
- user-defined final orbital energy and inclination

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 `escape` 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

The SOCS software 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 `escape` software is “data-driven” by a user-created text file. The following is a typical input file used by this computer program. In the following discussion the actual input file contents are in courier font and all explanations are in *times italic* font. This example attempts to optimize the maneuver required to perform a low-thrust escape from a 1000 kilometer circular Earth orbit.

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.*

```
*****
** earth escape trajectory optimization with SOCS
** single phase, continous-thrust maneuver
** program escape - leo-to-escape orbit transfer
** leo2esc.in - February 9, 2006
*****
```

*The first three inputs provide the initial spacecraft mass, thrust magnitude (assumed constant) and the specific impulse (also assumed constant) of the propulsion system.*

```
initial spacecraft mass (kilograms)
1000.0

thrust magnitude (newtons)
5.0

specific impulse (seconds)
3300.0
```

*The next two inputs define the circular orbit altitude and inclination of the initial Earth orbit.*

```
*****
* INITIAL ORBIT *
*****

altitude (kilometers)
1000.0

orbital inclination (degrees)
28.5
```

*The next two inputs specify the orbital inclination and specific orbital energy of the final escape trajectory.*

```
*****
* FINAL ORBIT *
*****

orbital inclination (degrees)
10.0
```

```
specific orbital energy (km/sec)**2
0.0
```

*This next input defines the method used to estimate the transfer time.*

```
*****
* type of initial guess for transfer time *
-----
1 = numerical integration
2 = user-defined
-----
1
```

*For option 2 above, the next input is the user's initial guess for the transfer time.*

```
*****
* user-defined initial guess for transfer time (hours) *
*****
480
```

*The next integer input defines the type of initial guess to use for the trajectory.*

```
*****
* initial guess/restart option *
*****
1 = integrated with tangential thrust
2 = binary data file
-----
1
```

*This input defines the name of a restart or initial guess binary data file.*

```
name of initial guess/restart input data file
leo2esc.rsbin
```

*The next input allows the creation of a binary file of the solution. This binary file can also be used as an initial guess for other simulations.*

```
*****
* restart and solution data file options *
*****

create/update binary restart data file (yes or no)
no
```

*The name of the binary data file to read or create is specified in this next input.*

```
name of binary restart data file
leo2esc.rsbin
```

*This next input specifies the type of solution data file to create.*

```
*****
* 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 input defines either the number of data points or the time step size of the data output in the solution file.*

```
number of user-defined nodes or print step size in solution data file  
25
```

*The name of the comma-separated-variable solution data file is defined in this next line.*

```
name of solution output file  
leo2esc.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 SOCOOUT 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

```

## **SOCS solution and graphics**

The following is the SOCS solution for this example. This simulation used the numerical integration method for estimating the transfer time, and the gravity-turn steering method to create the initial guess for the transfer trajectory. The output includes the orbital characteristics at the beginning and end of the propulsive maneuver.

```

program escape
-----
numerical integration initial guess

```

-----  
beginning of finite burn  
-----

sma (km)	eccentricity	inclination (deg)	argper (deg)
0.737814000000D+04	0.333066907388D-15	0.285000000000D+02	0.000000000000D+00
raan (deg)	true anomaly (deg)	arglat (deg)	period (min)
0.000000000000D+00	0.000000000000D+00	0.000000000000D+00	0.105118713217D+03
rx (km)	ry (km)	rz (km)	rmag (km)
0.737814000000D+04	0.000000000000D+00	0.000000000000D+00	0.737814000000D+04
vx (kps)	vy (kps)	vz (kps)	vmag (kps)
-.656097256141D-22	0.645942676649D+01	0.350718257926D+01	0.735013767191D+01

-----  
end of finite burn  
-----

sma (km)	eccentricity	inclination (deg)	argper (deg)
-.235592481724D+16	0.100000000006D+01	0.100000000146D+02	0.310806461100D+03
raan (deg)	true anomaly (deg)	arglat (deg)	period (min)
0.346515474360D+03	0.823719380820D+02	0.331783991818D+02	0.000000000000D+00
rx (km)	ry (km)	rz (km)	rmag (km)
0.224183187388D+06	0.784786902914D+05	0.226740474196D+05	0.238602428244D+06
vx (kps)	vy (kps)	vz (kps)	vmag (kps)
0.663271551844D+00	0.167403869139D+01	0.314312335743D+00	0.182787497827D+01

*The following program output is the final spacecraft mass, the propellant mass consumed, the actual thrust duration for the maneuver, the accumulated delta-v and the final specific energy.*

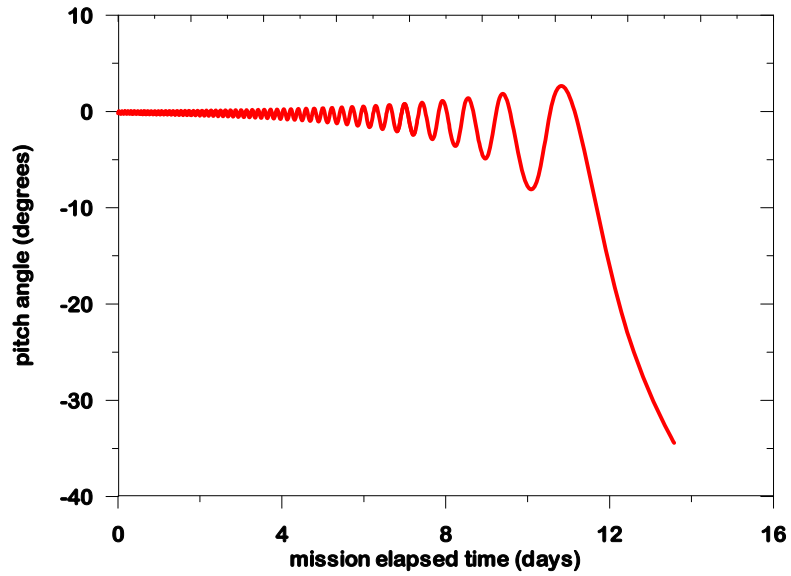
final mass	818.781389041040	kilograms
propellant mass	181.218610958960	kilograms
thrust duration	325.810373379458	hours
	13.5754322241441	days
delta-v	6470.38758377112	meters/second
specific energy	1.691908835255163E-010	km**2/sec**2

The delta-v is estimated using a cubic spline method that integrates the thrust acceleration at each and every grid point of the solution. Notice that the final orbital eccentricity is equal to 1 which confirms that a parabolic escape orbit has been created.

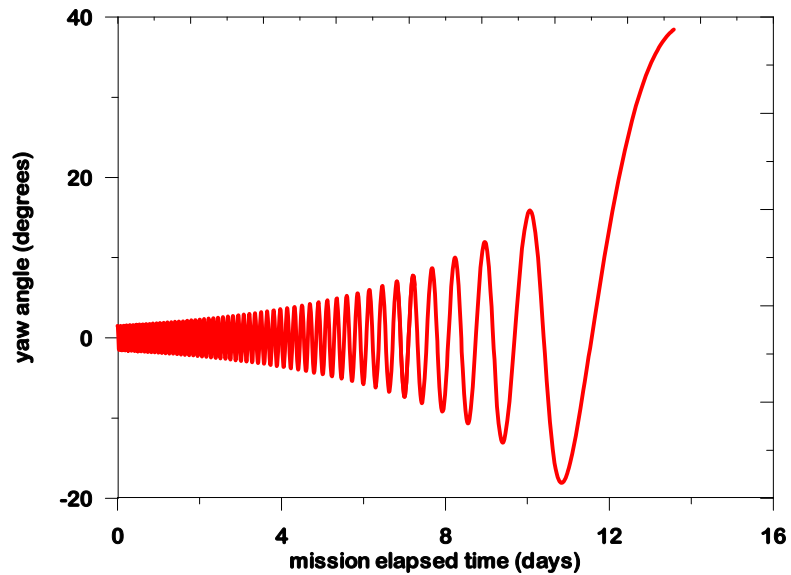
The `escape` computer program will create a comma-separated-variable (csv) data file containing the SOCS solution. This data file can be used to create graphic displays of important trajectory characteristics. Additional information about the contents of this file can be found in Appendix B.

The following plots illustrate the evolution of the pitch and yaw steering angles during this finite-burn maneuver.

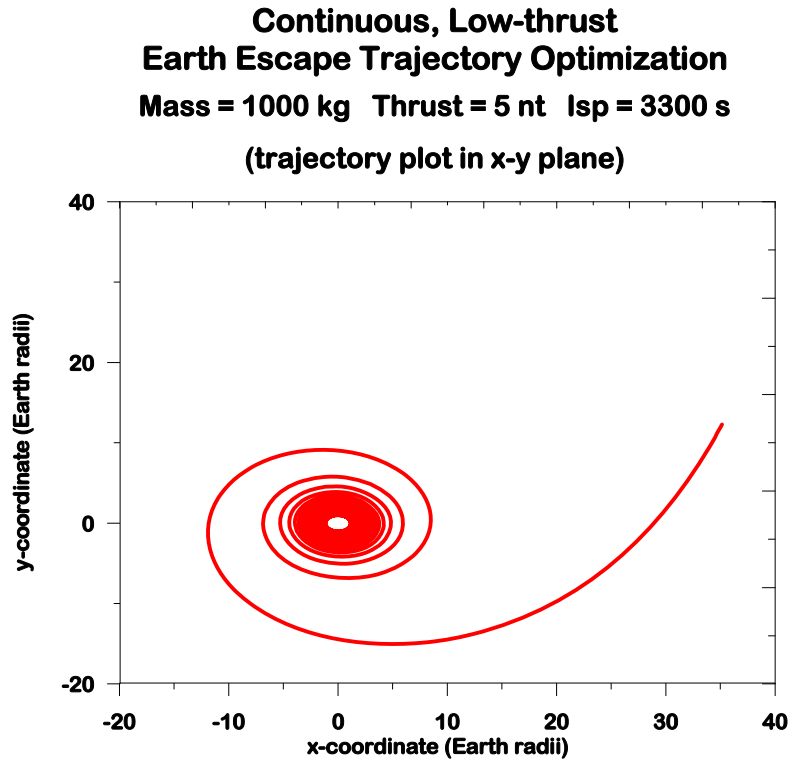
**Continuous, Low-thrust  
Earth Escape Trajectory Optimization**  
Mass = 1000 kg Thrust = 5 nt Isp = 3300 s  
(pitch angle vs mission elapsed time)



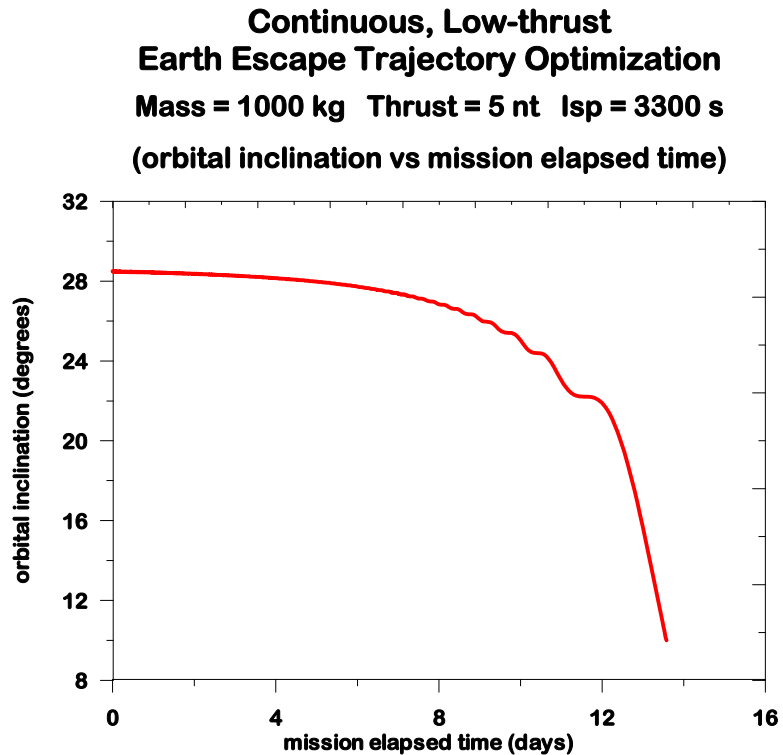
**Continuous, Low-thrust  
Earth Escape Trajectory Optimization**  
Mass = 1000 kg Thrust = 5 nt Isp = 3300 s  
(yaw angle vs mission elapsed time)



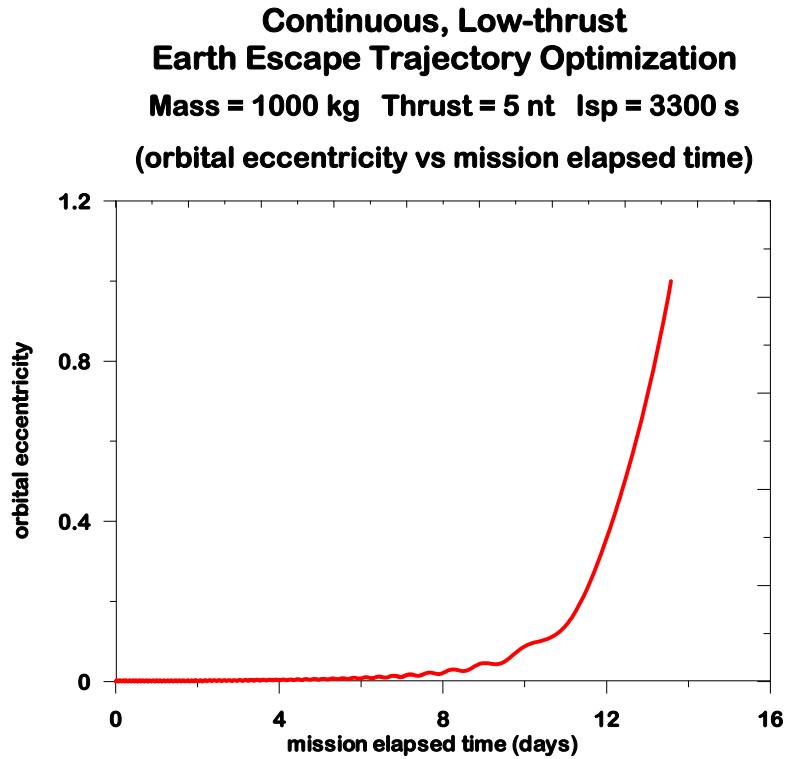
The following is a graphics display of the transfer trajectory in the x-y (equatorial) plane.



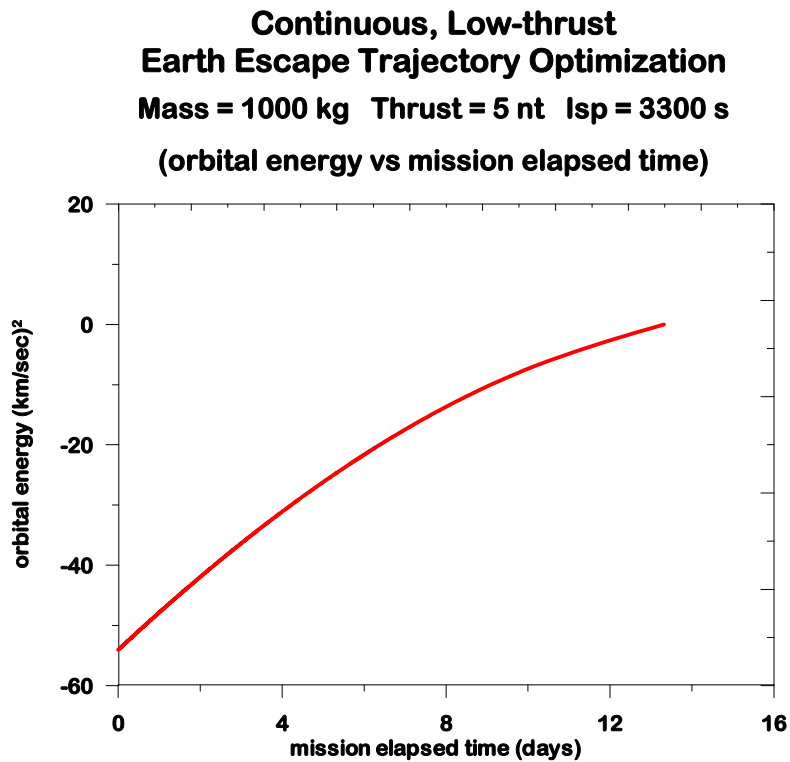
This next plot illustrates the behavior of the orbital inclination during the propulsive maneuver.



This plot depicts the time evolution of the orbital eccentricity.



Finally, the following plot illustrates the behavior of the specific orbital energy.



## Problem setup for SOCS

This section provides additional details about the software implementation. It explains such things as point and path constraints, the performance index and the numerical technique used to create an initial guess for the software.

### (1) Initial orbit constraints

The lower and upper bounds for all modified equinoctial elements are set equal to the initial user-defined modified equinoctial orbital elements (subscript  $i$ ) as follows:

$$p_L = p_U = p_i$$

$$f_L = f_U = f_i$$

$$g_L = g_U = g_i$$

$$h_L = h_U = h_i$$

$$k_L = k_U = k_i$$

$$L_L = L_U = L_i$$

The initial time is also constrained to a value of zero. In SOCS terminology, these derived constraints or boundary conditions are called *point functions*.

### (2) Performance index – minimize transfer time

The objective function or performance index  $J$  for this simulation is the total maneuver time. This is simply

$$J = t_f$$

Since this simulation involves a single continuous maneuver, this is equivalent to minimizing the required propellant mass. The value of the `maxmin` indicator in SOCS tells the software whether the user is minimizing or maximizing the performance index.

### (3) Path constraint – unit thrust vector scalar magnitude

The scalar magnitude of the components of the unit thrust vector at any time during the simulation is constrained as follows:

$$|\mathbf{u}_T| = \sqrt{u_{T_r}^2 + u_{T_t}^2 + u_{T_n}^2} = 1$$

This formulation avoids numerical problems that may occur when using thrust steering angles as control variables.

#### (4) Point functions – final mission orbit constraints

The final, user-defined orbit inclination is constrained by enforcing

$$\sqrt{h^2 + k^2} = \tan\left(\frac{i}{2}\right)$$

where  $i$  is the mission orbit inclination.

The final, user-defined specific orbital energy  $C_3$  is constrained with the following calculation

$$C_3 = v^2 - 2\frac{\mu}{r}$$

where  $v$  is the scalar velocity,  $r$  is the geocentric distance and  $\mu$  is the Earth's gravitational constant, all evaluated at the final condition.

#### Bounds on the dynamic variables

The following lower and upper bounds are applied to the modified equinoctial orbital elements *during* the orbital transfer.

$$10p_f \leq p \leq 0.9p_i$$

$$-1 \leq f \leq +1$$

$$-1 \leq g \leq +1$$

$$-1 \leq h \leq +1$$

$$-1 \leq k \leq +1$$

The true longitude is unbounded. The spacecraft mass at the initial time is fixed to the user-defined initial value.

The three components of the unit thrust vector are constrained as follows:

$$-1.1 \leq u_r \leq +1.1$$

$$-1.1 \leq u_t \leq +1.1$$

$$-1.1 \leq u_n \leq +1.1$$

#### Creating an initial guess for SOCS

The software implements a simple numerical method for estimating the total maneuver time. The program simply integrates the equations of motion until the specific orbital energy becomes positive. This approach assumes a circular or elliptical initial orbit.

### *Dynamic variables initial guess*

The initial guess for the dynamic variables at each grid point are determined by SOCS by setting the initial guess option `INIT(1) = 6` with `INIT(2) = 2`. For tangential thrusting the unit thrust vector in the modified equinoctial coordinate system at all times is simply  $\mathbf{u}_T = [0 \ 1 \ 0]^T$ . Please note that this type of steering creates a *coplanar* initial guess.

The `escape` computer program can also use a binary file created from a previous simulation as the initial guess.

### **Technical Discussion**

The modified equinoctial orbital elements are a set of orbital elements that are useful for trajectory analysis and optimization. They are valid for circular, elliptic, and hyperbolic orbits. These equations exhibit no singularity for zero eccentricity and orbital inclinations equal to 0 and 90 degrees. However, two components of the orbital element set are singular for an orbital inclination of 180 degrees.

The relationship between direct modified equinoctial and classical orbital elements is defined by the following definitions

$$p = a(1 - e^2)$$

$$f = e \cos(\omega + \Omega)$$

$$g = e \sin(\omega + \Omega)$$

$$h = \tan(i/2) \cos \Omega$$

$$k = \tan(i/2) \sin \Omega$$

$$L = \Omega + \omega + \theta$$

where

$p$  = semiparameter

$a$  = semimajor axis

$e$  = orbital eccentricity

$i$  = orbital inclination

$\omega$  = argument of periapsis

$\Omega$  = right ascension of the ascending node

$\theta$  = true anomaly

$L$  = true longitude

The relationship between classical and modified equinoctial orbital elements is summarized as follows:

semimajor axis

$$a = \frac{p}{1 - f^2 - g^2}$$

orbital eccentricity

$$e = \sqrt{f^2 + g^2}$$

orbital inclination

$$i = 2 \tan^{-1} \left( \sqrt{h^2 + k^2} \right)$$

argument of periapsis

$$\omega = \tan^{-1} (g/f) - \tan^{-1} (k/h)$$

right ascension of the ascending node

$$\Omega = \tan^{-1} (k/h)$$

true anomaly

$$\theta = L - (\Omega + \omega) = L - \tan^{-1} (g/f)$$

The mathematical relationships between an inertial state vector and the corresponding modified equinoctial elements are summarized as follows:

position vector

$$\mathbf{r} = \begin{bmatrix} \frac{r}{s^2} (\cos L + \alpha^2 \cos L + 2hk \sin L) \\ \frac{r}{s^2} (\sin L - \alpha^2 \sin L + 2hk \cos L) \\ \frac{2r}{s^2} (h \sin L - k \cos L) \end{bmatrix}$$

velocity vector

$$\mathbf{v} = \begin{bmatrix} -\frac{1}{s^2} \sqrt{\frac{\mu}{p}} (\sin L + \alpha^2 \sin L - 2hk \cos L + g - 2f h k + \alpha^2 g) \\ -\frac{1}{s^2} \sqrt{\frac{\mu}{p}} (-\cos L + \alpha^2 \cos L + 2hk \sin L - f + 2ghk + \alpha^2 f) \\ \frac{2}{s^2} \sqrt{\frac{\mu}{p}} (h \cos L + k \sin L + fh + gk) \end{bmatrix}$$

where

$$\alpha^2 = h^2 - k^2$$

$$s^2 = 1 + h^2 + k^2$$

$$r = \frac{p}{w}$$

$$w = 1 + f \cos L + g \sin L$$

The system of first-order modified equinoctial equations of orbital motion are given by

$$\dot{p} = \frac{dp}{dt} = \frac{2p}{w} \sqrt{\frac{p}{\mu}} \Delta_t$$

$$\dot{f} = \frac{df}{dt} = \sqrt{\frac{p}{\mu}} \left[ \Delta_r \sin L + [(w+1) \cos L + f] \frac{\Delta_t}{w} - (h \sin L - k \cos L) \frac{g \Delta_n}{w} \right]$$

$$\dot{g} = \frac{dg}{dt} = \sqrt{\frac{p}{\mu}} \left[ -\Delta_r \cos L + [(w+1) \sin L + g] \frac{\Delta_t}{w} + (h \sin L - k \cos L) \frac{g \Delta_n}{w} \right]$$

$$\dot{h} = \frac{dh}{dt} = \sqrt{\frac{p}{\mu}} \frac{s^2 \Delta_n}{2w} \cos L$$

$$\dot{k} = \frac{dk}{dt} = \sqrt{\frac{p}{\mu}} \frac{s^2 \Delta_n}{2w} \sin L$$

$$\dot{L} = \frac{dL}{dt} = \sqrt{\mu p} \left( \frac{w}{p} \right)^2 + \frac{1}{w} \sqrt{\frac{p}{\mu}} (h \sin L - k \cos L) \Delta_n$$

where  $\Delta_r, \Delta_t, \Delta_n$  are *non-two-body* perturbations in the radial, tangential and normal directions, respectively. The radial direction is along the radius vector of the spacecraft measured positive in a

direction away from the gravitational center, the tangential direction is perpendicular to this radius vector measured positive in the direction of orbital motion, and the normal direction is positive along the angular momentum vector of the spacecraft's orbit.

The equations of orbital motion can also be expressed in vector form as follows:

$$\dot{\mathbf{y}} = \frac{d\mathbf{y}}{dt} = \mathbf{A}(\mathbf{y})\mathbf{P} + \mathbf{b}$$

where

$$\mathbf{b} = \left[ 0 \quad 0 \quad 0 \quad 0 \quad 0 \quad \sqrt{\mu p} \left( \frac{w}{p} \right)^2 \right]^T$$

and

$$\mathbf{A} = \left\{ \begin{array}{ccc} 0 & \frac{2p}{w} \frac{\sqrt{p}}{\sqrt{\mu}} & 0 \\ \sqrt{\frac{p}{\mu}} \sin L & \sqrt{\frac{p}{\mu}} \frac{1}{w} \{(w+1)\cos L + f\} & -\sqrt{\frac{p}{\mu}} \frac{g}{w} \{h \sin L - k \cos L\} \\ -\sqrt{\frac{p}{\mu}} \cos L & \sqrt{\frac{p}{\mu}} \{(w+1)\sin L + g\} & \sqrt{\frac{p}{\mu}} \frac{f}{w} \{h \sin L - k \cos L\} \\ 0 & 0 & \sqrt{\frac{p}{\mu}} \frac{s^2 \cos L}{2w} \\ 0 & 0 & \sqrt{\frac{p}{\mu}} \frac{s^2 \sin L}{2w} \\ 0 & 0 & \sqrt{\frac{p}{\mu}} \{h \sin L - k \cos L\} \end{array} \right\}$$

The total *non-two-body* acceleration vector is given by

$$\mathbf{P} = \Delta_r \hat{\mathbf{i}}_r + \Delta_t \hat{\mathbf{i}}_t + \Delta_n \hat{\mathbf{i}}_n$$

where  $\hat{\mathbf{i}}_r$ ,  $\hat{\mathbf{i}}_t$  and  $\hat{\mathbf{i}}_n$  are unit vectors in the radial, tangential and normal directions. These unit vectors can be computed from the inertial position vector  $\mathbf{r}$  and velocity vector  $\mathbf{v}$  according to

$$\hat{\mathbf{i}}_r = \frac{\mathbf{r}}{|\mathbf{r}|}$$

$$\hat{\mathbf{i}}_n = \frac{\mathbf{r} \times \mathbf{v}}{|\mathbf{r} \times \mathbf{v}|}$$

$$\hat{\mathbf{i}}_t = \hat{\mathbf{i}}_n \times \hat{\mathbf{i}}_r = \frac{(\mathbf{r} \times \mathbf{v}) \times \mathbf{r}}{|\mathbf{r} \times \mathbf{v}| |\mathbf{r}|}$$

For *unperturbed* two-body motion,  $\mathbf{P} = 0$  and the first five equations of motion are simply  $\dot{p} = \dot{f} = \dot{g} = \dot{h} = \dot{k} = 0$ . Therefore, for two-body motion these modified equinoctial orbital elements are constant. The true longitude is often called the *fast variable* of this orbital element set.

### Non-spherical Earth Gravity

The non-spherical gravitational acceleration vector can be expressed as

$$\mathbf{g} = g_N \hat{\mathbf{i}}_N - g_r \hat{\mathbf{i}}_r$$

where

$$\hat{\mathbf{i}}_N = \frac{\hat{\mathbf{e}}_N - (\hat{\mathbf{e}}_N^T \hat{\mathbf{i}}_r) \hat{\mathbf{i}}_r}{\|\hat{\mathbf{e}}_N - (\hat{\mathbf{e}}_N^T \hat{\mathbf{i}}_r) \hat{\mathbf{i}}_r\|}$$

and

$$\hat{\mathbf{e}}_N = [0 \quad 0 \quad 1]^T$$

In these equations, the north direction component is indicated by subscript  $N$  and the radial direction component is subscript  $r$ .

The contributions due to the *zonal* gravity effects of  $J_2, J_3, J_4$  are as follows:

$$g_N = -\frac{\mu \cos \phi}{r^2} \sum_{k=2}^4 \left( \frac{R_e}{r} \right)^k P_k J_k$$

$$g_r = -\frac{\mu}{r^2} \sum_{k=2}^4 (k+1) \left( \frac{R_e}{r} \right)^k P_k J_k$$

where

- $\mu$  = gravitational constant
- $r$  = geocentric distance of the spacecraft
- $R_e$  = equatorial radius of the Earth
- $\phi$  = geocentric latitude
- $J_k$  = zonal gravity coefficient
- $P_k$  =  $k^{\text{th}}$  order Legendre polynomial

For a zonal-only Earth gravity model, the east component is identically zero.

Finally, the zonal gravity perturbation contribution is

$$\mathbf{a}_g = \mathbf{Q}^T \mathbf{g}$$

where  $\mathbf{Q} = [\hat{\mathbf{i}}_r \quad \hat{\mathbf{i}}_t \quad \hat{\mathbf{i}}_n]$ .

For  $J_2$  effects only, the three components are as follows:

$$\Delta_{J_{2r}} = -\frac{3\mu J_2 R_e^2}{2r^4} \left[ 1 - \frac{12(h \sin L - k \cos L)^2}{(1 + h^2 + k^2)^2} \right]$$

$$\Delta_{J_{2t}} = -\frac{12\mu J_2 R_e^2}{r^4} \left[ \frac{(h \sin L - k \cos L)(h \cos L + k \sin L)}{(1 + h^2 + k^2)^2} \right]$$

$$\Delta_{J_{2n}} = -\frac{6\mu J_2 R_e^2}{r^4} \left[ \frac{(1 - h^2 - k^2)(h \sin L - k \cos L)}{(1 + h^2 + k^2)^2} \right]$$

These are the equations modeled in this computer program,

### Propulsive Thrust

The acceleration due to propulsive thrust can be expressed as

$$\mathbf{a}_T = \frac{T}{m(t)} \hat{\mathbf{u}}_T$$

where  $T$  is the thrust magnitude,  $m$  is the spacecraft mass and  $\hat{\mathbf{u}}_T = [u_{T_r} \quad u_{T_t} \quad u_{T_n}]^T$  is the unit pointing thrust vector expressed in the spacecraft-centered radial-tangential-normal coordinate system.

The spacecraft mass at any mission elapsed time  $t$  is given by  $m(t) = m_{sc_i} - \dot{m} t$  where  $m_{sc_i}$  is the initial mass of the spacecraft and  $\dot{m}$  is the propellant flow rate.

The propellant mass flow rate is determined from

$$\dot{m} = \frac{dm}{dt} = \frac{T}{g I_{sp}}$$

where  $g$  is the acceleration of gravity and  $I_{sp}$  is the specific impulse of the propulsive system. The product  $g I_{sp}$  is also called the *exhaust velocity*.

Finally, the pitch and yaw angles can be determined from the components of the unit thrust vector according to

$$\theta = \sin^{-1}(u_{T_r})$$

$$\psi = \tan^{-1}(u_{T_n}, u_{T_r})$$

The pitch angle is positive above the “local horizontal” and the yaw angle is positive in the direction of the angular momentum vector.

## References and Bibliography

“A Set of Modified Equinoctial Orbital Elements”, M. J. H. Walker, B. Ireland and J. Owens, *Celestial Mechanics*, Vol. 36, pp. 409-419, 1985.

“Using Sparse Nonlinear Programming to Compute Low Thrust Orbit Transfers”, John T. Betts, *The Journal of the Astronautical Sciences*, Vol. 41, No. 3, July-September 1993, pp. 349-371.

“Equinoctial Orbit Elements: Application to Optimal Transfer Problems”, Jean A. Kechichian, AIAA 90-2976, AIAA/AAS Astrodynamics Conference, Portland, OR, 20-22 August 1990.

“Survey of Numerical Methods for Trajectory Optimization”, John T. Betts, *AIAA Journal of Guidance, Control and Dynamics*, Vol. 21, No. 2, March-April 1998.

# APPENDIX A

## Compiling and Running the Software

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

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

```
df /arch:host escape.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:

```
escape leo2esc1.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 escape           *
*                                     *
*       Earth escape trajectory       *
*       optimization with SOCS       *
*                                     *
*           February 9, 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 command line 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 escape           *'
print *, ' *                                           *'
print *, ' *           Earth escape trajectory       *'
print *, ' *           optimization with SOCS        *'
print *, ' *                                           *'
print *, ' *           February 9, 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.

## 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 escape software.

The simulation summary screen display contains the following information:

**sma (km)** = semimajor axis in kilometers

**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 (min)** = orbital period in minutes

**rx (km)** = x-component of the spacecraft's position vector in kilometers

**ry (km)** = y-component of the spacecraft's position vector in kilometers

**rz (km)** = z-component of the spacecraft's position vector in kilometers

**rmag (km)** = scalar magnitude of the spacecraft's position vector in kilometers

**vx (km/sec)** = x-component of the spacecraft's velocity vector in kilometers per second

**vy (km/sec)** = y-component of the spacecraft's velocity vector in kilometers per second

**vz (km/sec)** = z-component of the spacecraft's velocity vector in kilometers per second

**vmag (km/sec)** = scalar magnitude of the spacecraft's velocity vector in kilometers per second

**final mass** = final spacecraft mass in kilograms

**propellant mass** = expended propellant mass in kilograms

**thrust duration** = maneuver duration in seconds

**delta-v** = scalar magnitude of the maneuver in meters/seconds

The delta-v is determined using a cubic spline integration of the thrust acceleration data at each collocation node.

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

**time (hrs)** = simulation time since ignition in hours  
**time (days)** = simulation time since ignition in days  
**semimajor axis** = semimajor axis in kilometers  
**eccentricity** = orbital eccentricity (non-dimensional)  
**inclination** = orbital inclination in degrees  
**argument of perigee** = argument of perigee in degrees  
**raan** = right ascension of the ascending node in degrees  
**true anomaly** = true anomaly in degrees  
**period (min)** = orbital period in minutes  
**mass (kg)** = spacecraft mass in kilograms  
**thracc** = thrust acceleration in meters/second squared  
**yaw** = thrust vector yaw angle in degrees  
**pitch** = thrust vector pitch angle in degrees  
**perigee altitude** = perigee altitude in kilometers  
**apogee altitude** = apogee altitude in kilometers  
**ut-radial** = radial component of unit thrust vector  
**ut-tangential** = tangential component of unit thrust vector  
**ut-normal** = normal component of unit thrust vector  
**semi-parameter** = orbital semiparameter in kilometers  
**f equinoctial element** = modified equinoctial orbital element  
**g equinoctial element** = modified equinoctial orbital element  
**h equinoctial element** = modified equinoctial orbital element  
**k equinoctial element** = modified equinoctial orbital element  
**true longitude** = true longitude in degrees  
**rx (er)** = x-component of the spacecraft's position vector in Earth radii  
**ry (er)** = y-component of the spacecraft's position vector in Earth radii  
**rz (er)** = z-component of the spacecraft's position vector in Earth radii  
**fpa (deg)** = flight path angle in degrees  
**energy** = specific orbital energy in (kilometers per second) squared

## APPENDIX C

### Fortran Functions and Subroutines

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

- escape.f** - SOCS main executive program
- atan3.for** - four quadrant inverse tangent function
- csint.for** - cubic spline integration of tabular data subroutine
- display.for** - subroutine that displays solution information
- eci2mee.for** - convert eci position and velocity vectors to modified equinoctial orbital elements subroutine
- eci2orb.for** - convert eci position and velocity vectors to classical orbital elements subroutine
- linput1.for** - read and echo a line of text from an input file subroutine
- mee2eci.for** - convert modified equinoctial orbital elements to geocentric position and velocity vectors subroutine
- meeeqms2.for** - modified equinoctial orbital elements equations of motion subroutine - used to create initial guess for transfer time
- odeigs.for** - SOCS initial guess 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
- 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
- rkf78.for** - Runge-Fehlberg-Kutta (RKF78) numerical integration subroutine
- rkf78cn.for** - evaluate RKF78 integration coefficients subroutine
- 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 escape software. This subroutine is the point function routine required by the SOCS software.

```
      subroutine odepf(ipphase, iphend, time, y, ny, p, np,
&                    fptf, nf, iferr)
c     point functions
c     *****
      implicit double precision (a-h, o-z)
      include 'socscom1.inc'
      dimension y(ny), fptf(nf), p(np), reci(3), veci(3)
      iferr = 0
c     load equinoctial elements into local variables
      pmee = y(1)
      fmee = y(2)
      gmee = y(3)
      hmee = y(4)
      xkmee = y(5)
      xlmee = y(6)
c     compute eci state vector
      call mee2eci(pmee, fmee, gmee, hmee, xkmee, xlmee, reci,
&                veci, smovrp, tani2s, cosl, sinl, wmee, radius,
&                hsmks, ssqrd)
c     radius magnitude (kilometers)
      rmag = vecmag(reci)
      vms = veci(1)**2 + veci(2)**2 + veci(3)**2
c     twice specific orbital energy (km/sec)**2
      c33 = vms - 2.0d0 * emu / rmag
      if (ipphase .eq. 1 .and. iphend .eq. +1) then
c         enforce at end of phase 1
```

```
fptf(1) = c33
c      final orbit inclination constraint
      fptf(2) = sqrt(hmee * hmee + xkmee * xkmee)
end if
return
end
```