

## Program lvsocs

### Lunar Ascent Trajectory Optimization with SOCS

This document is the user's manual for a Fortran computer program called `lvsocs` 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 single-stage, finite-burn lunar ascent trajectory optimization problem. The trajectory is modeled as a single optimized phase with a user-defined vertical rise, pitch over and initial and final boundary conditions. This computer program attempts to maximize the final mass placed into lunar orbit or minimize the time to ascend to the final orbit. The lunar mission orbit can be circular or elliptical.

The important features of this scientific simulation are as follows:

- user-defined vertical rise and pitch over phases
- angle-of-attack, bank angle and throttle control variables
- 3-DOF flight path equations of motion relative to a spherical, rotating Moon
- user-defined vehicle mass and propulsion properties
- user-defined mission orbit semimajor axis, eccentricity 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 `lvsocs` 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 `lvsocs` 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 data file defines a typical simulation with throttling enabled and a circular final mission orbit with an inclination constraint. For this simulation, the initial azimuth is bounded so that the software can determine the correct azimuth after the pitch over maneuver.

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.*

```
*****
** lunar launch vehicle simulation
** lvl.in
** November 15, 2005
** maximize payload to orbit
*****
```

*The first input is an integer that defines the type of simulation. Option 2 is equivalent to a full throttle or constant thrust simulation.*

```
type of trajectory optimization
*****
1 = maximize final mass
2 = minimize flight time
-----
1
```

*The next three inputs allow the user to specify the duration of the vertical rise and pitch over maneuvers along with the total pitch angle.*

```
vertical rise time (seconds)
10

pitch over time duration (seconds)
30

pitch angle change during pitch over (degrees)
1.25
```

*These next two inputs define the propulsion characteristics for the simulation. Here the user can specify the maximum thrust and specific impulse.*

```
*****
propulsion characteristics
*****
```

```
maximum thrust (newtons)
5000.0

specific impulse (seconds)
350.0
```

*The initial vehicle mass and a guess for the final mass are defined by the next two data inputs.*

```
*****
vehicle mass characteristics
*****

initial vehicle mass (kilograms)
3000.0

initial guess for final vehicle mass (kilograms)
1500.0
```

*The following series of data items are reserved for user-defined initial conditions. To fix one or more initial conditions, the user should input identical lower and upper bounds. To free or unconstrain one or more initial states, set the lower and/or upper bounds to 1.0d99. Please note the units and valid data range for each item.*

```
*****
initial flight conditions and bounds
*****
NOTE 1: set upper and lower bounds to
the initial value to constrain or
"fix" the flight azimuth.
NOTE 2: set bound to 1.0d99 to ignore
-----

initial flight azimuth (0 <= azimuth <= 360; degrees)
90.0

upper bound for initial flight azimuth (0 <= azimuth <= 360; degrees)
360.0

lower bound for initial flight azimuth (0 <= azimuth <= 360; degrees)
0.0

initial selenocentric latitude (-90 <= latitude <= + 90; degrees)
20.0

initial selenographic east longitude (0 <= longitude <= 360; degrees)
40.0
```

*The following series of data items allow the user to define the final flight conditions. To fix one or more conditions, the user should input identical lower and upper bounds. To free or unconstrain one or more final states set the lower and/or upper bounds to 1.0d99. Please note the units and valid data range for each item.*

```
*****
initial guess for final flight conditions and bounds
*****
NOTE 1: set upper and lower bounds
to the final value to constrain or
"fix" a flight condition.
NOTE 2: set bound to 1.0d99 to ignore
-----
```

final time (seconds)  
800.0

upper bound for final time (seconds)  
1.0d99

lower bound for final time (seconds)  
1.0d99

final altitude (meters)  
100000.0

upper bound for final altitude (meters)  
120000.0

lower bound for final altitude (meters)  
10000.0

final velocity (meters/second)  
1600.0

upper bound for final velocity (meters/second)  
2000.0

lower bound for final velocity (meters/second)  
1000.0

final flight path angle (-90 <= fpa <= +90; degrees)  
0.0

upper bound for final flight path angle (-90 <= fpa <= +90; degrees)  
+90.0

lower bound for final flight path angle (-90 <= fpa <= +90; degrees)  
-90.0

final flight azimuth (0 <= azimuth <= 360; degrees)  
90.0

upper bound for final flight azimuth (0 <= azimuth <= 360; degrees)  
360.0

lower bound for final flight azimuth (0 <= azimuth <= 360; degrees)  
0.0

final declination (-90 <= declination <= +90; degrees)  
0.0

upper bound for final declination (-90 <= declination <= +90; degrees)  
+90.0

lower bound for final declination (-90 <= declination <= +90; degrees)  
-90.0

final east longitude (0 <= longitude <= 360; degrees)  
55.0

upper bound for final east longitude (0 <= longitude <= 360; degrees)  
360.0

lower bound for final east longitude (0 <= longitude <= 360; degrees)  
0.0

*The next series of data inputs define lower and upper bounds on the state variables during the ascent-to-orbit phase. To free or unconstrain one or more states, set the lower and/or upper bounds to the value 1.0d99.*

\*\*\*\*\*  
upper and lower bounds on the flight conditions during ascent  
\*\*\*\*\*  
NOTE: set bound to 1.0d99 to ignore  
-----

upper bound for altitude (meters)  
120000.0

lower bound for altitude (meters)  
1.0

upper bound for velocity (meters/second)  
2000.0

lower bound for velocity (meters/second)  
0.1

upper bound for flight path angle (-90 <= fpa <= +90; degrees)  
+90.0

lower bound for flight path angle (-90 <= fpa <= +90; degrees)  
-90.0

upper bound for flight azimuth (0 <= azimuth <= 360; degrees)  
360.0

lower bound for flight azimuth (0 <= azimuth <= 360; degrees)  
0.0

upper bound for declination (-90 <= declination <= +90; degrees)  
+90.0

lower bound for declination (-90 <= declination <= +90; degrees)  
-90.0

upper bound for east longitude (0 <= longitude <= 360; degrees)  
360.0

lower bound for east longitude (0 <= longitude <= 360; degrees)  
0.0

*This next section of the input file defines the initial guesses and bounds for the control variables. To free or unconstrain one or more control variables set the lower and/or upper bounds to 1.0d99.*

\*\*\*\*\*  
initial flight controls and bounds

```

*****
NOTE 1: set upper and lower bounds
to the initial value to constrain
or "fix" a flight control.
NOTE 2: set bound to 1.0d99 to ignore
-----

initial angle-of-attack (degrees)
0.0

upper bound for initial angle-of-attack (degrees)
+90.0

lower bound for initial angle-of-attack (degrees)
-90.0

initial bank angle (degrees)
0.0

upper bound for initial bank angle (degrees)
+90.0

lower bound for initial bank angle (degrees)
-90.0

initial throttle setting
1.0

upper bound for initial throttle setting
1.0

lower bound for initial throttle setting
0.5

```

*This section of the input file defines the final guesses and bounds for the control variables. To free one or more control variables set the lower and/or upper bounds to 1.0d99.*

```

*****
final flight controls and bounds
*****
NOTE 1: set upper and lower bounds
to the final value to constrain
or "fix" a flight control.
NOTE 2: set bound to 1.0d99 to ignore
-----

final angle-of-attack (degrees)
10.0

upper bound for final angle-of-attack (degrees)
+90.0

lower bound for final angle-of-attack (degrees)
-90.0

final bank angle (degrees)
0.0

upper bound for final bank angle (degrees)
+90.0

```

lower bound for final bank angle (degrees)  
-90.0

final throttle setting  
0.75

upper bound for final throttle setting  
1.0

lower bound for final throttle setting  
0.5

*This section of the input file defines the lower and upper bounds for the control variables during the ascent-to-orbit phase. To free or un-constrain one or more control variables, set the lower and/or upper bounds to 1.0d99.*

```
*****  
upper and lower bounds on the flight controls during phase  
*****
```

NOTE: set bound to 1.0d99 to ignore  
-----

upper bound for angle-of-attack (degrees)  
+90.0

lower bound for angle-of-attack (degrees)  
-90.0

upper bound for bank angle (degrees)  
+90.0

lower bound for bank angle (degrees)  
-90.0

upper bound for throttle setting  
1.0

lower bound for throttle setting  
0.5

*These next three inputs allow the user to specify the final orbit characteristics. Please note that an input of 1.0d99 for the final orbital inclination will tell the software to ignore the orbital inclination constraint.*

```
*****  
final orbit constraints  
*****
```

final semimajor axis (kilometers)  
1838.0

final orbital eccentricity (non-dimensional)  
0.0d0

final orbital inclination (degrees; enter 1.0d99 to ignore this constraint)  
25.0

*The software also creates a comma-separated-variable (csv) ascii data file that contains the optimal control solution and other flight parameters. This next integer 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 next 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
1.0
```

*The name of this output file is specified in the next line of information. Please consult Appendix B for information about the contents of this file.*

```
name of comma-delimited solution data file
lv1.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.*

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

*The next integer defines the number of initial grid points to use in the collocation modeling of the ascent trajectory.*

```
number of initial guess grid points to use
25
```

*The next series of program inputs are algorithm control options and parameters for the SOCS software.*

```
*****
algorithm control parameters
*****
```

*This 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  
100000

*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
```

## SOCS solution and graphics

Before attempting to optimize the trajectory, the software will display the conditions at the beginning and end of the vertical rise as well as the flight conditions at the end of the pitch over maneuver. The following is the program output for this example.

```
liftoff
-----

time          = 0.000000000000000E+000 seconds
altitude      = 0.000000000000000E+000 meters
longitude     = 40.0000000000000 degrees
declination   = 20.0000000000000 degrees
velocity      = 0.000000000000000E+000 meters/second
fpa           = 0.000000000000000E+000 degrees
azimuth       = 0.000000000000000E+000 degrees
mass          = 3000.00000000000 kilograms

end of vertical rise
-----

time          = 10.0000000000000 seconds
altitude      = 2.31424039066769 meters
longitude     = 40.0000000000000 degrees
declination   = 20.0000000000000 degrees
velocity      = 0.476409333678228 meters/second
mass          = 2985.43262552889 kilograms

end of pitch over
-----

time          = 40.0000000000000 seconds
altitude      = 43.5612765820697 meters
longitude     = 40.0001936048218 degrees
declination   = 20.0000000000000 degrees
velocity      = 2.45921128623709 meters/second
fpa           = 76.9988433932694 degrees
azimuth       = 89.9999999999999 degrees
aoa           = 11.7511565715146 degrees
pitch         = 1.250000000000000 degrees
mass          = 2941.73050211555 kilograms
```

After convergence, the software will display the flight path conditions immediately after the pitch over and the flight path conditions and classical orbital elements at mission orbit injection. The following is the program output for this example.

```
program lvsocs
-----

maximize final spacecraft mass
```

flight path coordinates after pitch over

-----

time	40.0000000000000	seconds
altitude	43.5612765818369	meters
longitude	40.0001936048218	degrees
declination	20.0000000000000	degrees
relative velocity	2.45921128623710	meters/second
relative fpa	76.9988433932694	degrees
relative azimuth	105.172802478125	degrees
vehicle mass	2941.73050211555	kilograms

final flight path coordinates

-----

time	1274.72461779489	seconds
altitude	100000.000000000	meters
longitude	67.6367285613016	degrees
declination	10.5949167724820	degrees
relative velocity	1628.80391248496	meters/second
relative fpa	5.723002084051430E-016	degrees
relative azimuth	112.841287734027	degrees
vehicle mass	1448.95056465587	kilograms

classical orbital elements - final orbit

-----

sma (km)	eccentricity	inclination (deg)	argper (deg)
0.1838000000000D+04	0.737680639419D-16	0.250000000000D+02	0.000000000000D+00
raan (deg)	true anomaly (deg)	arglat (deg)	period (min)
0.271285983313D+03	0.154210911681D+03	0.154210911681D+03	0.117848695428D+03
rx (km)	ry (km)	rz (km)	rmag (km)
0.687395682825D+03	0.167078609594D+04	0.337942298127D+03	0.183800000000D+04
vx (kps)	vy (kps)	vz (kps)	vmag (kps)
-.134839994404D+01	0.680465282475D+00	-.621489625991D+00	0.163323751027D+01

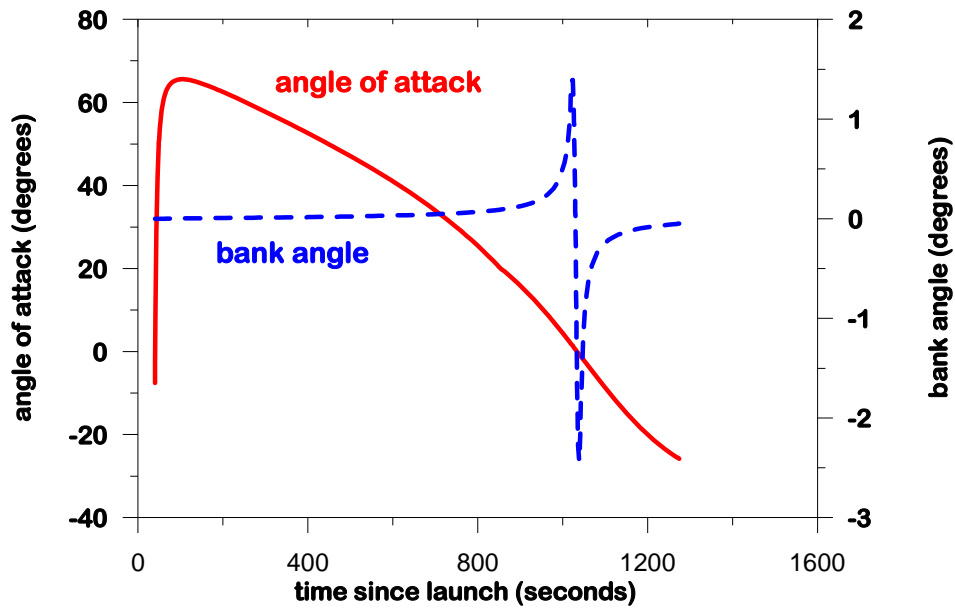
The following are trajectory characteristics plots created from the user-defined summary file. The first plot illustrates the behavior of the angle of attack and bank angle during the ascent trajectory.

Notice the bank reversal as the angle of attack passes through zero. This behavior is explained on page 1192 of “Advanced Launch System Trajectory Optimization”, by P. Daniel Burkhart and David G. Hull, AAS 96-168. An examination of the azimuth differential equation given in the Technical Discussion later in this document will help explain this behavior.

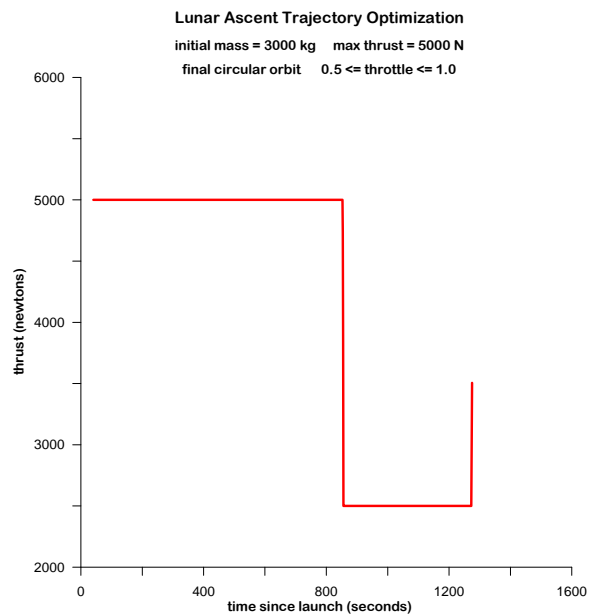
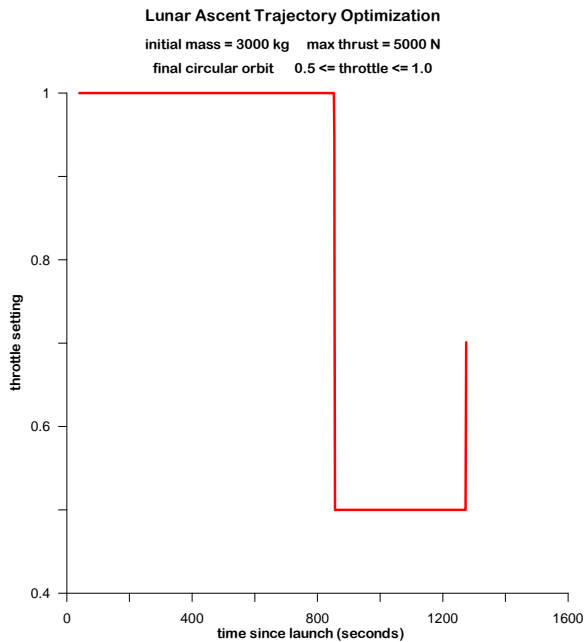
## Lunar Ascent Trajectory Optimization

initial mass = 3000 kg    max thrust = 5000 N

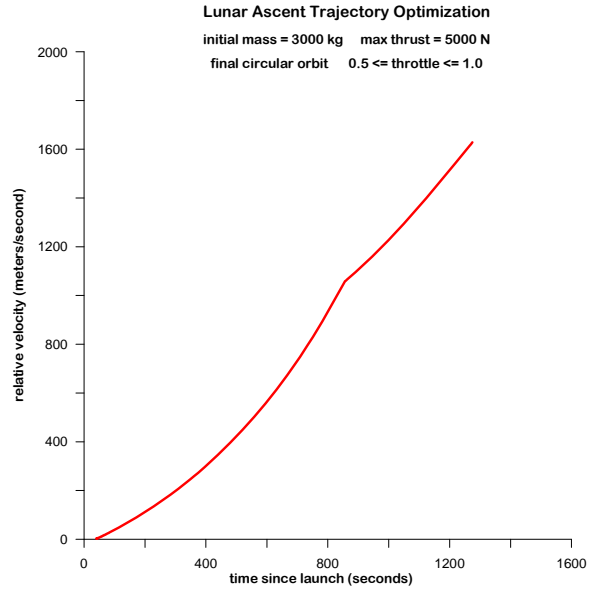
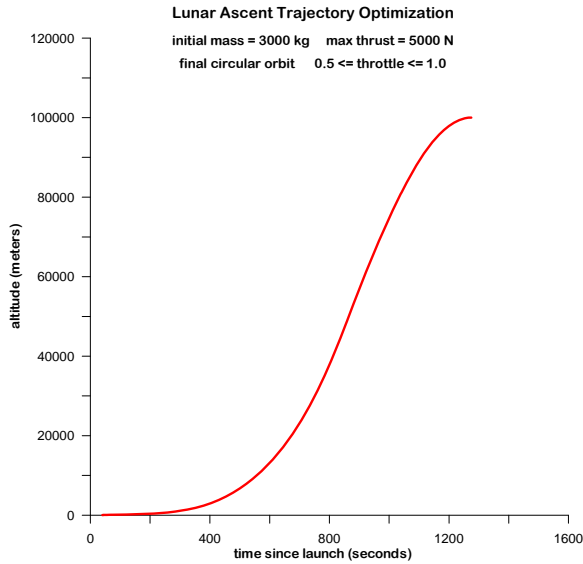
final circular orbit     $0.5 \leq \text{throttle} \leq 1.0$



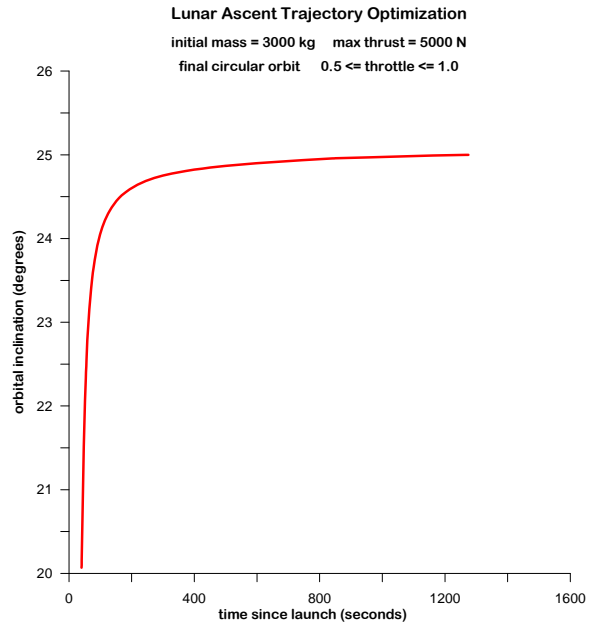
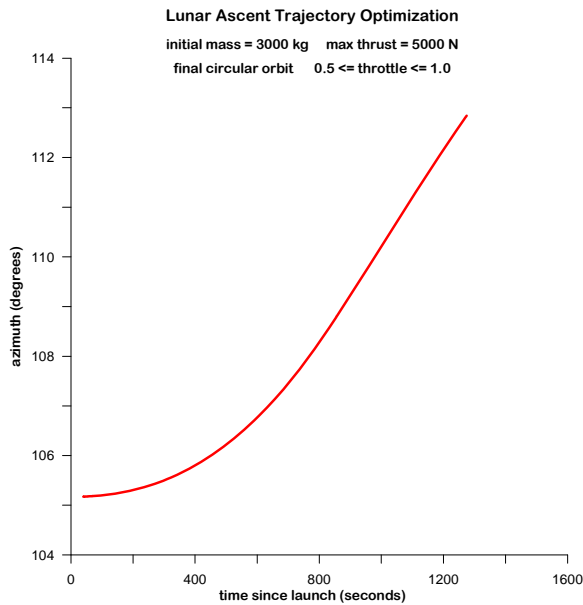
The next two plots show the variation of throttle setting and thrust during the ascent.



The altitude and velocity are illustrated in the next two plots.



The final two plots are the azimuth and orbital inclination for this example.



### Problem setup for SOCS

This section provides additional details about the SOCS software implementation. It briefly explains such things as point constraints and the performance index options. The `lvsocs` simulation consists of a vertical rise followed by a pitch over maneuver and a final, optimized ascent-to-orbit phase. The software uses a linear interpolation guess based on the flight conditions after the pitch over maneuver and final flight conditions provided by the user.

## (1) Performance index

The performance index for the *maximize final mass* option is the final spacecraft mass given by

$$J = m_f = m(t_f)$$

where  $m_f$  is the spacecraft mass at the final time.

For the *minimize flight time* program option, the performance index is

$$J = t_f$$

where  $t_f$  is the simulation time at mission orbit injection.

The value of the `maxmin` indicator in SOCS tells the software whether the user is minimizing or maximizing the performance index.

## (2) Path constraints

The path constraints enforced by the software are based on the lower and upper bounds for the dynamic variables provided by the user.

## (3) Point constraints

The number and type of point functions enforced by the software is a function of the orbital elements of the final mission orbit. This section discusses the point constraints or final boundary conditions implemented in the `lvsoes` computer program.

### *Circular orbit*

For a final circular mission orbit, the user provides the semimajor axis of this orbit. The software then uses the following three point constraints to “target” a final circular orbit.

$$\begin{aligned}r_f &= a_u \\v_f &= \sqrt{\frac{\mu}{a_u}} \\ \sin \gamma_f &= \frac{\mathbf{r}_f \bullet \mathbf{v}_f}{|\mathbf{r}_f \bullet \mathbf{v}_f|} = 0\end{aligned}$$

If the user also specifies an orbital inclination for the final mission orbit, the software enforces the following point constraint at the final time

$$\sin i_f = \frac{h_{fz}}{h_f} = \frac{(\mathbf{r}_f \times \mathbf{v}_f)_z}{|\mathbf{r}_f \times \mathbf{v}_f|} = \sin i_u$$

In these equations, the  $f$  subscript refers to mission characteristics computed by the software at the final time and the  $u$  subscript refers to user-provide final or “target” orbital conditions.

### *Elliptical orbit*

For a user-defined elliptical mission orbit, the software constrains the final specific orbital energy and final angular momentum magnitude according to

$$E_f = v_f^2 - \frac{2\mu}{r_f} = E_u = v_u^2 - \frac{2\mu}{r_u}$$

$$h_f = \frac{\mathbf{r}_f \times \mathbf{v}_f}{|\mathbf{r}_f \times \mathbf{v}_f|} = h_u = \frac{\mathbf{r}_u \times \mathbf{v}_u}{|\mathbf{r}_u \times \mathbf{v}_u|}$$

If the user also specifies an orbital inclination for the final mission orbit, the software enforces the following point constraint

$$\sin i_f = \frac{h_{fz}}{h_f} = \frac{(\mathbf{r}_f \times \mathbf{v}_f)_z}{|\mathbf{r}_f \times \mathbf{v}_f|} = \sin i_u$$

In these equations, the  $f$  subscript refers to mission characteristics computed by the software at the final time and the  $u$  subscript refers to user-provided final orbital conditions. The user-defined position vector  $\mathbf{r}_u$  and velocity vector  $\mathbf{v}_u$  are determined from the semimajor axis, orbital eccentricity and inclination input by the user. All other angular orbital elements are set to 0.

## **Technical Discussion**

This section describes the equations of motion implemented in this computer program along with other important numerical algorithms.

### *Flight path equations of motion*

The first-order flight path equations of motion relative to a rotating, spherical Moon are as follows:

selenocentric radius

$$\dot{r} = \frac{dr}{dt} = V \sin \gamma$$

selenographic longitude

$$\dot{\lambda} = \frac{d\lambda}{dt} = V \frac{\cos \gamma \sin \psi}{r \cos \delta}$$

selenocentric declination

$$\dot{\delta} = \frac{d\delta}{dt} = V \frac{\cos \gamma \cos \psi}{r}$$

speed

$$\dot{V} = \frac{dV}{dt} = \frac{T \cos \alpha}{m} - g \sin \gamma + \omega^2 r \cos \delta (\sin \gamma \cos \delta - \sin \delta \cos \gamma \cos \psi)$$

flight path angle

$$\begin{aligned} \dot{\gamma} = \frac{d\gamma}{dt} = & \frac{V}{r} \cos \gamma + \left( \frac{T \sin \alpha}{mV} \right) \cos \beta - \frac{g \cos \gamma}{V} + 2\omega \sin \psi \cos \delta \\ & + \omega^2 \frac{r}{V} \cos \delta (\cos \psi \sin \gamma \sin \delta + \cos \gamma \cos \delta) \end{aligned}$$

flight azimuth

$$\begin{aligned} \dot{\psi} = \frac{d\psi}{dt} = & \frac{V}{r} \tan \delta \sin \psi \cos \gamma + \left( \frac{T \sin \alpha}{mV \cos \gamma} \right) \sin \beta \\ & + 2\omega (\sin \delta - \cos \psi \cos \delta \tan \gamma) + \frac{r}{V \cos \gamma} \omega^2 \sin \psi \cos \delta \sin \delta \end{aligned}$$

where

$r$  = selenocentric radius

$V$  = relative speed

$\gamma$  = flight path angle

$\delta$  = selenocentric declination

$\lambda$  = selenographic longitude (+ east)

$\psi$  = flight azimuth (+ clockwise from north)

$\beta$  = bank angle (+ for a right turn)

$\alpha$  = angle of attack

$r_{eq}$  = lunar equatorial radius

$\omega$  = lunar inertial rotation rate

$\mu$  = lunar gravitational constant

$g$  = acceleration of gravity =  $\mu/r^2$

$T$  = propulsive thrust

$m$  = spacecraft mass

## Vertical rise phase

During the vertical rise part of the trajectory, the software uses a subset of the full equations of motion. This set of equations includes  $\dot{r}, \dot{v}, \dot{\delta}, \dot{\lambda}$  with  $\dot{\gamma} = \dot{\psi} = 0$ . This is necessary in order to avoid singularities in the flight path and azimuth differential equations. Furthermore, during the numerical integration of the vertical rise differential equations,  $\alpha = 0^\circ, \beta = 0^\circ, \gamma = 90^\circ, \psi = 0^\circ$  and the throttle setting is equal to 1.

## Pitch over phase

During the pitch over part of the trajectory, the software also uses a subset of the full equations of motion. This set of equations includes  $\dot{r}, \dot{v}, \dot{\delta}, \dot{\lambda}, \dot{\gamma}$  with  $\dot{\psi} = 0$ . This is necessary because azimuth is poorly defined until the end of the pitch over part of the ascent trajectory. Furthermore, during the numerical integration of the pitch over differential equations,  $\alpha = 0^\circ, \beta = 0^\circ, \gamma = 90^\circ, \psi = 0^\circ$  and the throttle setting is equal to 1.

The angle of attack during the pitch over phase is determined from

$$\alpha = 90^\circ - \gamma + \dot{\theta}(t - t_p)$$

where  $\dot{\theta}$  is the pitch rate,  $t$  is the time since launch and  $t_p$  is the time since launch at which the pitch over maneuver begins. The constant pitch rate is determined from the pitch angle and pitch over maneuver duration provided by the user.

For a realistic and contiguous ascent trajectory, the launch vehicle should pitch over to the azimuth direction determined by the ascent-to-orbit optimization.

## Crossrange and downrange calculations

The crossrange angle is determined from the following expression which can be derived from spherical trigonometry relationships:

$$\sin \nu = -\sin \psi_1 \sin \phi_2 \cos \phi_1 \cos \Delta\lambda - \cos \psi_1 \cos \phi_1 \sin \Delta\lambda + \sin \psi_1 \cos \phi_2 \sin \phi_1$$

The downrange angle is determined from the following three equations:

$$\sin \mu = -\cos \psi_1 \sin \phi_2 \cos \phi_1 \cos \Delta\lambda + \sin \psi_1 \cos \phi_1 \sin \Delta\lambda + \cos \psi_1 \cos \phi_2 \sin \phi_1$$

$$\cos \mu = \cos \phi_2 \cos \phi_1 \cos \Delta\lambda + \sin \phi_2 \sin \phi_1$$

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

where

$\phi_1$  = selenocentric declination of the initial point

$\psi_1$  = flight azimuth at the initial point

$\phi_2$  = selenocentric declination of the final point

$$\Delta\lambda = \lambda_2 - \lambda_1$$

$\lambda_1$  = east longitude of the initial point

$\lambda_2$  = east longitude of the final point

Please note that the inverse tangent above is a four quadrant form.

### Conversion of flight path coordinates to inertial state vector

The software uses the following expressions to calculate the selenocentric, inertial position and velocity vectors of the spacecraft at the final time:

$$r_x = r \cos \delta \cos \lambda$$

$$r_y = r \cos \delta \sin \lambda$$

$$r_z = r \sin \delta$$

$$v_x = v \left[ \cos \lambda (-\cos \psi \sin \beta \sin \delta + \cos \beta \cos \delta) - \sin \psi \sin \beta \sin \alpha \right]$$

$$v_y = v \left[ \sin \lambda (-\cos \psi \sin \beta \sin \delta + \cos \beta \cos \delta) + \sin \psi \sin \beta \cos \alpha \right]$$

$$v_z = v (\cos \psi \cos \delta \sin \beta + \cos \beta \cos \delta)$$

where the correction for lunar rotation is

$$v_x = v_x - \omega r_y$$

$$v_y = v_y + \omega r_x$$

The inertial speed can also be computed from the following expression

$$v_i = \sqrt{v^2 + 2vr\omega \cos \gamma \sin \psi \cos \delta + r^2 \omega^2 \cos^2 \delta}$$

and the inertial flight path angle from

$$\cos \gamma_i = \sqrt{\frac{v^2 \cos^2 \gamma + 2vr\omega \cos \gamma \cos \psi \cos \delta + r^2 \omega^2 \cos^2 \delta}{v^2 + 2vr\omega \cos \gamma \cos \psi \cos \delta + r^2 \omega^2 \cos^2 \delta}}$$

where all coordinates on the right-hand-side of these equations are relative to a rotating Moon.

## References and Bibliography

- (1) "Direct Trajectory Optimization Using Nonlinear Programming and Collocation", C. R. Hargraves and S. W. Paris, *AIAA Journal of Guidance, Control and Dynamics*, Vol. 10, No. 4, July-August, 1987, pp. 338-342.
- (2) "Optimal Finite-Thrust Spacecraft Trajectories Using Direct Transcription and Nonlinear Programming", Paul J. Enright, Ph.D. Thesis, University of Illinois at Urbana-Champaign, 1991.
- (3) "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.
- (4) "Optimization of Thrust Direction Histories and Vehicle Parameters for Three-dimensional Ascent Trajectories", C. Tucker Battle and Robert G. Gottlieb, AIAA 64-663.
- (5) "Improved Collocation Methods with Application to Direct Trajectory Optimization", Albert L. Herman, Ph.D. Thesis, University of Illinois at Urbana-Champaign, 1995.
- (6) "Optimal Low Thrust Interplanetary Trajectories by Direct Method Techniques", Craig A. Kluever, *The Journal of the Astronautical Sciences*, Vol. 45, No. 3, July-September 1997, pp. 247-262.
- (7) "Survey of Numerical Methods for Trajectory Optimization", John T. Betts, *AIAA Journal of Guidance, Control and Dynamics*, Vol. 21, No. 2, March-April 1998, pp. 193-207.
- (8) "Design and Evaluation of a 3-D Optimal Ascent Guidance Algorithm", Anthony J. Calise, Nahum Melamed and Seungjae Lee, AIAA 97-3707.
- (9) "Optimization of Launch Vehicle Ascent Trajectories with Path Constraints and Coast Arcs", C. P. Gath and A. J. Calise, *AIAA Journal of Guidance, Control and Dynamics*, Vol. 24, No. 2, March-April, 2001, pp. 296-304.
- (10) "Advanced Launch System Trajectory Optimization Using Suboptimal Control", Douglas A. Shaver and David G. Hull, AIAA 90-3413.
- (11) "Solving the Optimal Control Problem Using a Nonlinear Programming Technique, Part 2: Optimal Shuttle Ascent Trajectories", T. Bauer, J. Betts, W. Huffman and K. Zondervan, AIAA 84-2038.
- (12) "Semi-Analytical Formulas for Launcher Performance Evaluation", Emanuele Di Sotto and Paolo Teofilatto, *AIAA Journal of Guidance, Control and Dynamics*, Vol. 25, No. 3, May-June, 2002, pp. 538-545.

# APPENDIX A

## Compiling and Running the Software

This appendix describes how to compile and run the `lvsocs` computer program. This software was created using version 6.3.3 of SOCS and Compaq Visual Fortran.

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

```
df /arch:host lvsocs.f *.for c:\socs\socs633.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:

```
lvsocs lv1.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 lvsocs           *
*                                     *
*       lunar ascent trajectory       *
*       optimization with SOCS       *
*                                     *
*           November 15, 2005        *
*****
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 lvsocs          *'
print *, ' *          *          *'
print *, ' *      lunar ascent trajectory      *'
print *, ' *      optimization with SOCS        *'
print *, ' *          *          *'
print *, ' *          November 15, 2005          *'
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 CSV File

This appendix is a brief summary of the information contained in the CSV data file produced by the lvsocs software. The comma-separated-variable disk file is created by the odeprt subroutine and contains the following information:

**time (seconds)** = simulation time since launch in seconds

**altitude (meters)** = altitude relative to a spherical Moon in meters

**velocity (mps)** = Moon-relative velocity in meters per second

**flight path angle (d)** = Moon-relative flight path angle in degrees

**azimuth (deg)** = Moon-relative azimuth angle in degrees

**declination (deg)** = selenocentric declination in degrees

**longitude (deg)** = selenographic east longitude in degrees

**angle of attack (deg)** = angle-of-attack in degrees

**bank angle (deg)** = bank angle in degrees

**throttle** = throttle setting

**thrust (newtons)** = thrust in newtons

**load factor (g)** = load factor in Earth g's

**crossrange (km)** = crossrange distance in kilometers

**downrange (km)** = downrange distance in kilometers

**vehicle mass (kg)** = vehicle mass in kilograms

**fuel flowrate (kg/s)** = fuel flowrate in kilograms /second

**rmag (meters)** = selenocentric distance in meters

**vmag (mps)** = selenocentric speed in meters/second

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

**eccentricity** = orbital eccentricity (non-dimensional)

**inclination (deg)** = orbital inclination in degrees

**arg of periapsis (deg)** = argument of periapsis in degrees

**raan (deg)** = right ascension of the ascending node in degrees

**true anomaly (deg)** = true anomaly in degrees

**periapsis radius (km)** = selenocentric periapsis radius in kilometers

**apoapsis radius (km)** = selenocentric apoapsis radius in kilometers

**eci fpa (deg)** = inertial flight path angle in degrees

## APPENDIX C

### Fortran Functions and Subroutines

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

<b>lvsocs.f</b>	- SOCS main executive program
<b>atan3.for</b>	- four quadrant inverse tangent function
<b>crdr.for</b>	- subroutine that calculates crossrange and downrange
<b>eci2orb.for</b>	- subroutine that converts an inertial position and velocity vector to classical orbital elements
<b>fpc2car.for</b>	- subroutine that converts flight path coordinates to inertial position and velocity vectors
<b>fpeqms1.for</b>	- subroutine that evaluates flight path equations of motion during the vertical rise phase
<b>fpeqms2.for</b>	- subroutine that evaluates flight path equations of motion during the pitch over phase
<b>odeinp.for</b>	- SOCS simulation input subroutine
<b>odepf.for</b>	- SOCS point functions subroutine
<b>odeprt.for</b>	- SOCS print subroutine - creates comma-separated-variable file
<b>oderhs.for</b>	- SOCS subroutine that evaluates the equations of motion and any algebraic equations
<b>readfpn.for</b>	- read and echo floating point number from an input file subroutine
<b>readint.for</b>	- read and echo an integer from an input file subroutine
<b>readtext.for</b>	- read and echo text from an input file subroutine
<b>rkf78.for</b>	- Runge-Fehlberg-Kutta (RKF78) numerical integration subroutine
<b>rkf78cn.for</b>	- evaluate RKF78 integration coefficients subroutine
<b>utility.for</b>	- number and text manipulation functions and subroutines
<b>uvector.for</b>	- unit vector subroutine
<b>vcross.for</b>	- vector cross product subroutine
<b>vdot.for</b>	- vector dot product subroutine
<b>vecmag.for</b>	- vector scalar magnitude function
<b>xmod.for</b>	- 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 `lvsocs` software. This subroutine is the differential-algebraic equations routine required by the SOCS software.

```
      subroutine oderhs(iphase, t, ydyn, ny, p, np, frhs, nf, iferr)

c     computes the right hand sides of the
c     differential-algebraic (dae) equations

c     input

c     state variables

c     rmag      = selenocentric radius (meters)           = ydyn(1)
c     elon     = selenographic longitude (radians)       = ydyn(2)
c     dec      = selenocentric declination (radians)     = ydyn(3)
c     vrel     = relative velocity (meters/second)       = ydyn(4)
c     fpa      = relative flight path angle (radians)    = ydyn(5)
c     azim     = relative azimuth (radians)              = ydyn(6)
c     mass     = vehicle mass (kilograms)                = ydyn(7)
c     alfa     = angle of attack (radians)               = ydyn(8)
c     beta     = bank angle (radians)                   = ydyn(9)
c     throttle = throttle setting (nd)                  = ydyn(10)

c     output

c     frhs = right hand sides of differential-algebraic system

c     *****

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

      include 'socscom1.inc'

      dimension ydyn(ny), p(np), frhs(nf), xtmp(2)

      iferr = 0

c     selenocentric radius (meters)

      rmag = ydyn(1)

c     selenographic longitude (radians)

      elon = ydyn(2)

c     selenocentric declination (radians)

      dec = ydyn(3)

c     relative speed (meters/second)
```

```

vrel = ydyn(4)

c flight path angle (radians)

fpa = ydyn(5)

c flight azimuth (radians)

azim = ydyn(6)

c current vehicle mass (kilograms)

xmass = ydyn(7)

c angle of attack (radians)

alfa = ydyn(8)

c bank angle (radians)

bank = ydyn(9)

c throttle setting (non-dimensional)

throttle = ydyn(10)

c compute gravitational acceleration (meters/second**2)

agrav = 1.0d9 * xmu / rmag**2

c *****
c propulsion properties
c *****

c propellant flow rate (kilograms/second)

xmdot = throttle * thrmax / vexhaust

c propulsive thrust (newtons)

thrust = throttle * thrmax

c *****
c equations of motion
c *****

c compute common trigonometric terms

sfpa = sin(fpa)
cfpa = cos(fpa)

sazim = sin(azim)
cazim = cos(azim)

salfa = sin(alfa)
calfa = cos(alfa)

```

```

sbank = sin(bank)
cbank = cos(bank)

sdec = sin(dec)
cdec = cos(dec)

c *****
c state differential equations
c *****

c no aerodynamic lift or drag

xlift = 0.0d0

drag = 0.0d0

rdot = vrel * sfpa

elondot = vrel * cfpa * sazim / (rmag * cdec)

decdot = vrel * cfpa * cazim / rmag

tmp1 = omega**2 * rmag * cdec * (sfpa * cdec
&      - sdec * cfpa * cazim)

tmp2 = (thrust * calfa - drag) / xmass

tmp3 = -agrav * sfpa

vdot = tmp1 + tmp2 + tmp3

tmp1 = vrel * cfpa / rmag

tmp2 = 2.0d0 * omega * sazim * cdec

tmp3 = omega**2 * (rmag / vrel) * cdec * (cazim
&      * sfpa * sdec + cfpa * cdec)

tmp4 = (thrust * salfa + xlift) * cbank / (xmass * vrel)

tmp5 = -agrav * cfpa / vrel

gamdot = tmp1 + tmp2 + tmp3 + tmp4 + tmp5

tmp1 = (vrel / rmag) * tan(dec) * sazim * cfpa

tmp2 = 2.0d0 * omega * (sdec - cazim * cdec * tan(fpa))

tmp3 = (rmag / (vrel * cfpa)) * omega**2
&      * sazim * cdec * sdec

tmp4 = ((thrust * salfa + xlift) * sbank) / (xmass * vrel * cfpa)

azidot = tmp1 + tmp2 + tmp3 + tmp4

```

```
c *****
c first-order equations of motion
c *****

frhs(1) = rdot
frhs(2) = elondot
frhs(3) = decdot
frhs(4) = vdot
frhs(5) = gamdot
frhs(6) = azidot
frhs(7) = -xmdot

return
end
```