Aerospace Trajectory Optimization Using Direct Transcription and Collocation

This brief report describes a technical approach and computer program that demonstrate the use of *direct transcription* and *collocation* to solve aerospace optimal control problems. The computer program solves the problem described on pages 66-69 of the classic text, *Applied Optimal Control*, by Arthur E. Bryson, Jr. and Yu-Chi Ho.

This problem is summarized in the text as follows:

"Given a constant-thrust rocket engine, T = thrust, operating for a given length of time, t, we wish to find the thrust-direction history, $\phi(t)$, to transfer a rocket vehicle from a given initial circular orbit to the largest possible circular orbit."

In the language of optimal control theory, this problem is a "continuous system with functions of the state variables prescribed at a fixed terminal time". The numerical example given in the text is a continuous, low-thrust coplanar orbit transfer from Earth to Mars. The orbits of the planets are assumed to be circular and coplanar, and the total transfer time is about 193 days.

Mathematical Formulation

A typical optimal control trajectory problem can be described by a system of *dynamic variables*

$$\mathbf{z} = \begin{bmatrix} \mathbf{y}(t) \\ \mathbf{u}(t) \end{bmatrix} \tag{1}$$

consisting of the *state variables* $\mathbf{y}$ and the *control variables* $\mathbf{u}$ for any time t. In this discussion vectors are denoted in bold.

The system dynamics are defined by a vector system of ordinary differential equations called the *state equations* that can be represented as follows:

$$\dot{\mathbf{y}} = \frac{d\mathbf{y}}{dt} = \mathbf{f}\left[\mathbf{y}(t), \mathbf{u}(t), \mathbf{p}, t\right] \tag{2}$$

where $\mathbf{p}$ is a vector of problem *parameters* that are time independent.

The initial flight conditions at time t_0 are defined by $\boldsymbol{\psi}_0 \equiv \boldsymbol{\psi}\left[\mathbf{y}(t_0), \mathbf{u}(t_0), t_0\right]$ and the terminal conditions at the final time t_f are defined by $\boldsymbol{\psi}_f \equiv \boldsymbol{\psi}\left[\mathbf{y}(t_f), \mathbf{u}(t_f), t_f\right]$. These conditions are called the *boundary values* of the trajectory problem.

The problem may also be subject to *path constraints* of the form $\mathbf{g}\left[\mathbf{y}(t), \mathbf{u}(t), t\right] = 0$.

For any mission time *t* there are also simple bounds on the state variables

$$\mathbf{y}_l \le \mathbf{y}(t) \le \mathbf{y}_u \quad (3)$$

and the control variables

$$\mathbf{u}_l \le \mathbf{u}(t) \le \mathbf{u}_u \quad (4)$$

The basic optimal control problem is to determine the control vector history that minimizes the scalar performance index or objective function given by

$$J = \phi\left[\mathbf{y}(t_0), t_0, \mathbf{y}(t_f), t_f, \mathbf{p}\right] \quad (5)$$

while satisfying all the user-defined mission constraints.

Direct Transcription

The basic idea behind direct transcription involves *discretizing* the state and control representation of a continuous aerospace trajectory. This technique allows the Optimal Control Problem (OCP) to be "transcribed" into a Nonlinear Programming Problem (NLP). The OCP can be thought of as an NLP with an infinite number of controls and constraints.

Each phase of an aerospace trajectory is divided into segments or intervals such that

$$t_I = t_1 < t_2 < \ldots < t_n = t_F$$

where t_I is the initial time and t_F is the final mission time. The individual time points are called *node*, *mesh* or *grid* points. The value of the state vector at a grid point or node is $\mathbf{y}_k = \mathbf{y}(t_k)$ and the control vector is $\mathbf{u}_k = \mathbf{u}(t_k)$.

In the direct transcription method we treat the values of the states and controls at the nodes as a set of NLP variables. Furthermore, the differential equations of the problem are represented by a system of *defect* equality constraints that are enforced at each discretization node. In the direct transcription method the state and control variable bounds become simple bounds on the NLP variables. The defect constraints and variable bounds are imposed at all the grid points. If we represent the state defects by a *Hermite-Simpson* discretization method, these constraints and bounds are also imposed at the *midpoints* of each trajectory segment.

The following diagram illustrates the geometry of trajectory and control discretization. For this simple example we have divided the trajectory into 8 segments. These 8 segments can be represented by 9 discrete nodes with their corresponding times t_i, states $\mathbf{y}_i$ and controls $\mathbf{u}_i$. A typical segment midpoint is also illustrated.

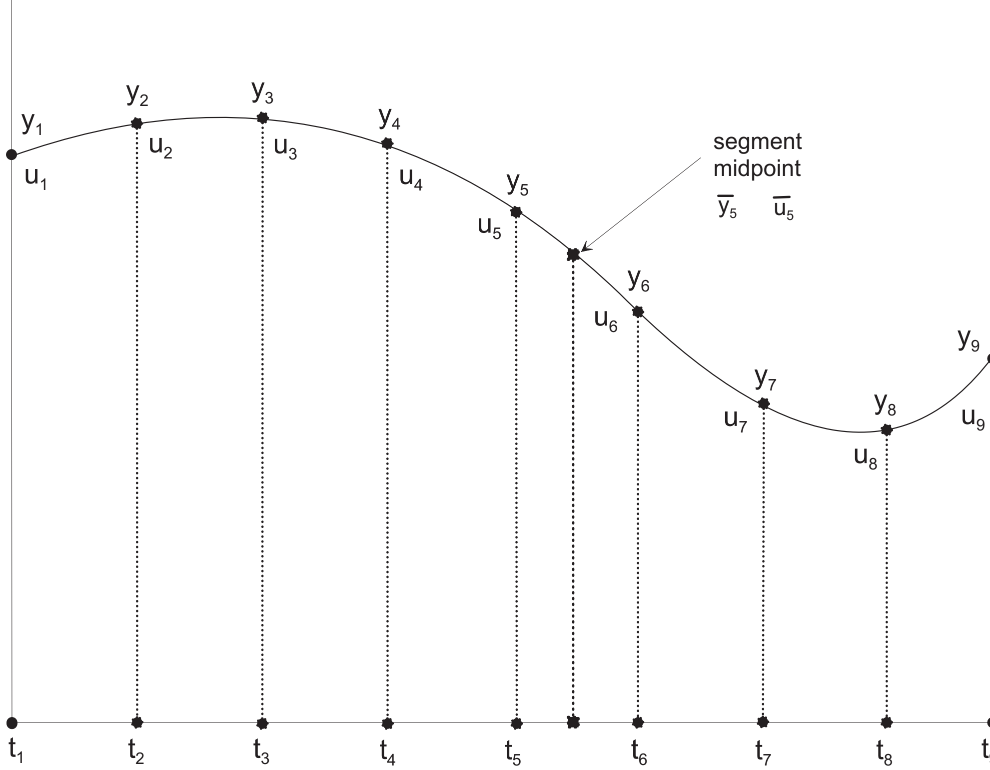

Figure 1 Trajectory and Control Discretization

Collocation

For the *compressed Hermite-Simpson* discretization or collocation method, the NLP variables are

$$\mathbf{x} = \left[\left(\mathbf{y}, \mathbf{u}, \mathbf{u}_m \right)_i, \left(\mathbf{y}, \mathbf{u}, \mathbf{u}_m \right)_{i+1}, \ldots, \left(\mathbf{y}, \mathbf{u} \right)_f, t_0, t_f \right]^T \tag{6}$$

where $\mathbf{u}_{m_1}, \mathbf{u}_{m_2}, etc.$ are values of the controls at the *midpoints* of the discretization segments.

The *defects* for this discretization algorithm are given by

$$\zeta_k = \mathbf{y}_{k+1} - \mathbf{y}_k - \frac{h_k}{6} \left[\mathbf{f}\left(\mathbf{y}_{k+1}, \mathbf{u}_{k+1} \right) + 4\mathbf{f}\left(\mathbf{y}_{m_k}, \mathbf{u}_{m_k} \right) + \mathbf{f}\left(\mathbf{y}_k, \mathbf{u}_k \right) \right] \tag{7}$$

where $\mathbf{f}\left(\mathbf{x}, \mathbf{u} \right)$ represents the equations of motion evaluated at the nodes and midpoints.

The state vector and equations of motion at the *midpoint* of a segment are given by

$$\mathbf{y}_{m_k} = \frac{1}{2} \left[\mathbf{y}_k + \mathbf{y}_{k+1} \right] + \frac{h_k}{8} \left[\mathbf{f}\left(\mathbf{y}_k, \mathbf{u}_k \right) - \mathbf{f}\left(\mathbf{y}_{k+1}, \mathbf{u}_{k+1} \right) \right] \tag{8}$$

and

$$\mathbf{f}_{m_k} = \mathbf{f}\left[\mathbf{y}_{m_k}, \mathbf{u}_{m_k}, \mathbf{p}, t_k + \frac{h_k}{2}\right] \tag{9}$$

for $k = 1, \ldots, n_N - 1$

In these equations h_k is the time interval between segments. For *equalduration* segments h_k is equal to $(t_f - t_i)/(n_N - 1)$ where n_N is the number of nodes and $n_N - 1$ is the number of segments.

By treating the state vector defects as equality constraints, the NLP algorithm attempts to converge the Hermite-Simpson approximation to the actual trajectory as defined by the *right-hand-sides* of the equations of motion. Furthermore, setting the vector of defects defined by Eq. (7) to zero enforces continuity at the trajectory nodes.

Bryson-Ho Problem Description

The first-order, two-dimensional equations of motion for the Bryson-Ho Earth-to-Mars orbit transfer problem are given by

$$\dot{r} = \frac{dr}{dt} = u$$

$$\dot{u} = \frac{du}{dt} = \frac{v^2}{r} - \frac{\mu}{r^2} + \frac{T}{m}\sin\phi \tag{10}$$

$$\dot{v} = \frac{dv}{dt} = -\frac{uv}{r} + \frac{T}{m}\cos\phi$$

where

$r =$ radial position
$u =$ radial velocity
$v =$ transverse velocity
$T =$ propulsive thrust
$m =$ spacecraft mass
$\phi =$ thrust angle
$\mu =$ gravitational constant

The thrust angle is defined relative to the "local horizontal" or tangential direction at the spacecraft's position. It is measured positive above the local horizontal plane and negative below. The in-plane thrust angle ϕ is the single control variable for this problem.

The spacecraft mass at any elapsed time t is determined from

$$m(t) = m_0(1 - \dot{m}t) \tag{11}$$

where $\dot{m}$ is the propellant flow rate of the propulsive device and m_0 is the initial mass.

The flight conditions of the spacecraft in its *initial circular orbit* are as follows:

$$r(0) = r_0$$

$$u(0) = u_0 = 0 \tag{12}$$

$$v(0) = v_0 = \sqrt{\frac{\mu}{r_0}}$$

The *boundary conditions* that create a *circular orbit* at the final time t_f are given by

$$u(t_f) = u_f = 0$$

$$v(t_f) - \sqrt{\frac{\mu}{r(t_f)}} = 0 \tag{13}$$

The initial mass and propulsive characteristics for the Earth to Mars orbit transfer are as follows:

- initial thrust $T = 3.781$ newtons
- initial spacecraft mass $m_0 = 4535.9$ kilograms
- propellant flow rate $\dot{m} = 5.85$ kilograms/day

The *non-dimensional* acceleration due to thrusting is given by

$$\frac{T/m_0}{\mu/r_0^2} = 0.1405 \tag{14}$$

The *non-dimensional* acceleration unit is μ/r_0^2 and the *non-dimensional* total flight time is given by

$$\frac{t_f}{\sqrt{r_0^3/\mu}} \tag{15}$$

The *non-dimensional* time unit used in the software is $\sqrt{r_0^3/\mu}$. The *non-dimensional* value of the gravitational constant μ is 1. The dimensional value of the Sun's gravitational constant is $132712441933\ \text{km}^3/\text{sec}^2$. The *non-dimensional* value of the initial distance r_0 is 1. Finally, the dimensional value of the radius of the Earth's *circular* orbit is 149597870.691 kilometers or one Astronomical Unit.

From the dimensional propellant flow rate and Equation (11) we can determine the *non-dimensional* propellant flow rate which is equal to -0.07487. This interplanetary mission requires about 1129 kilograms of propellant.

For this problem we would like to *maximize* the radius of the final circular orbit. Therefore the objective function is $J = -r_f$.

Numerical Solution of the Optimal Control Problem

In order to transcribe the optimal control problem (OCP) into a nonlinear programming problem (NLP) for computer solution, the user must provide an initial guess for the state and control vectors at the nodes and the following information and software components:

(1) problem definition (number of trajectory states, number of discretization nodes, number of control variables, initial and final conditions, etc.)

(2) right-hand-side of the equations of motion

(3) state vector defect equality constraints

(4) collocation method

(5) scalar object function

To demonstrate this process a Fortran 90 computer program was created to solve the Bryson-Ho example. This section documents the construction and operation of this software.

The software begins by defining such things as the total number of nodes and control variables with Fortran `parameter` statements. The following is the *problem definition* for this example.

```
! number of differential equations

parameter (nstates = 3)

! number of control variables

parameter (ncv = 1)

! number of discretization nodes

parameter (nnodes = 48)

! number of state nlp variables

parameter (nlps = nstates * nnodes)
```

```
! number of control nlp variables

parameter (nlpc = 2 * ncv * nnodes)

! total number of nlp variables

parameter (nlpv = nlps + nlpc)

! number of state vector defect equality constraints

parameter (ndeq = nlps - nstates)

! number of auxilary equality constraints

parameter (neqaux = 2)

! neq = total number of equality constraints

parameter (neq = ndeq + neqaux)
```

The next part of the computer program calculates such things as the time values at each node, the initial and final state and control guesses, and the lower and upper boundaries for the NLP variables. For this example the initial guess for the state vector at each node is set to the value of the initial state vector. The initial guess for the control at each node is simply set to zero.

The following is part of the Fortran source code that performs these operations.

```
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
! initial and final conditions and times
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!

! initial simulation time

tinitial = 0.0d0

! final simulation time

tfinal = 3.32d0

! define state vector at initial time

xinitial(1) = 1.0d0

xinitial(2) = 0.0d0

xinitial(3) = 1.0d0

!!!!!!!!!!!!!!!!!!!!!!!!!!!!
! create time values at nodes
!!!!!!!!!!!!!!!!!!!!!!!!!!!!

tarray(1) = tinitial

tarray(nnodes) = tfinal

deltat = (tfinal - tinitial) / (nnodes - 1)

do i = 2, nnodes - 1
   tarray(i) = (i - 1) * deltat
end do
```

```
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
! upper and lower bounds for nlp state variables
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!

do i = 1, nlps
   xlb(i) = -1000.0d0

   xub(i) = +1000.0d0
end do

! constrain initial boundary conditions
! to be the initial state vector

do i = 1, nstates
   xlb(i) = xinitial(i)

   xub(i) = xinitial(i)
end do

!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
! upper and lower bounds for nlp control variables
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!

do i = nlps + 1, nlpv
   xlb(i) = -pi2

   xub(i) = +pi2
end do

!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
! initial guess for nlp state variables
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!

j = 0

do i = 1, nlps
   j = j + 1

   xguess(i) = xinitial(j)

   if (j .eq. nstates) j = 0
end do

!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
! initial guess for nlp control variables
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!

do i = nlps + 1, nlpv
   xguess(i) = 0.0d0
end do

!!!!!!!!!!!!!!!!!!!!!!!!!!!!
! nlp variable scale factors
!!!!!!!!!!!!!!!!!!!!!!!!!!!!

do i = 1, nlpv
   xscale(i) = 1.0d0
end do
```

The NLP problem is solved by calling a Fortran subroutine that is part of the optimization section of the IMSL numerical analysis library. The following is the syntax for this operation.

```
call nconf (fcn, m, me, n, xguess, ibtype, xlb, xub, xscale, iprint, &
            maxitn, x, fvalue)
```

In the calling arguments `fcn` is a user-supplied subroutine that evaluates the objective function for any given trajectory conditions. The syntax for this subroutine is as follows:

```
call fcn (m, me, n, x, active, f, g)
```

where

`m`	total number of constraints (input)
`me`	number of equality constraints (input)
`n`	number of variables (input)
`x`	the point at which the functions are evaluated (input)
`active`	logical vector of length mmax indicating the active constraints (input)
`mmax`	max(1, m)
`f`	the computed function value at the point x (output)
`g`	vector of length mmax containing the values of constraints at point x (output)
`xguess`	vector of length n containing an initial guess of the computed solution (input)
`ibtype`	scalar indicating the types of bounds on variables (input)
`xlb`	vector of length n containing the lower bounds on variables (input)
`xub`	vector of length n containing the upper bounds on variables (input)
xscale	vector of length n containing the diagonal scaling matrix for the variables (input)
`iprint`	parameter indicating the desired output level (input)
`maxitn`	maximum number of iterations allowed (input)
`x`	vector of length n containing the computed solution (output)
`fvalue`	scalar containing the value of the objective function at the computed solution (output)

The source code for the Fortran subroutine that calculates the current value of the objective function and the equality constraints (defects) for this example is as follows:

```
subroutine dtofcn (nct, n1, n2, x, active, f, g)

! objective function and constraints

! inputs

!  nct    = total number of constraints
!  n1     = number of equality constraints
!  n2     = number of nlp variables
!  x      = current nlp variable values
```

```
!  active = constraint active flag

! outputs

!  f = current value of scalar objective function
!  g = vector of constraints evaluated at x

!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!

use simdef

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

! dimension arrays

dimension x(nlpv), xk(nstates), xkp1(nstates), xfinal(nstates)

dimension g(*), reswrk(nstates), resid(ndeq)

dimension uk(ncv), ukp1(ncv), umid(nstates)

logical active(*)

! current final state vector

xfinal(1) = x(nlps - 2)

xfinal(2) = x(nlps - 1)

xfinal(3) = x(nlps)

! current value of objective function
! (maximize radial distance)

f = -xfinal(1)

! compute state vector defect equality constraints

do k = 1, nnodes - 1

   ! time elements

   tk = tarray(k)

   tkp1 = tarray(k + 1)

   ! state vector elements

   if (k .eq. 1) then
      ! first node
      nks = 0
      nkp1s = nstates
   else
      ! reset to previous node
      nks = (k - 1) * nstates
      nkp1s = nks + nstates
   end if

   do i = 1, nstates
      xk(i) = x(nks + i)

      xkp1(i) = x(nkp1s + i)
```

```
      end do

      ! control variable elements

      if (k .eq. 1) then
         ! first node
         nkc = nlps
         nmid = nkc + 1
         nkp1c = nmid + 1
      else
         ! reset to previous node
         nkc = nkc + ncv + 1
         nmid = nkc + ncv
         nkp1c = nmid + ncv
      end if

      do i = 1, ncv
         uk(i) = x(nkc + i)

         umid(i) = x(nmid + i)

         ukp1(i) = x(nkp1c + i)
      end do

      ! compute state vector defects for current node

      call defect(tk, tkp1, xk, xkp1, uk, ukp1, umid, reswrk)

      ! load defect array for this node

      do i = 1, nstates
         resid(nks + i) = reswrk(i)
      end do
   end do

   ! set active defect constraints

   do i = 1, ndeq
      if (active(i)) g(i) = resid(i)
   end do

   ! compute auxillary equality constraints
   ! (final boundary conditions)

   if (active(ndeq + 1)) g(ndeq + 1) = xfinal(2)

   if (active(ndeq + 2)) g(ndeq + 2) = xfinal(1) * xfinal(3)**2 - 1.0d0

   return
   end
```

The following is the source code of the Fortran subroutine that calculates the state defect vector for the *compressed Hermite-Simpson* collocation algorithm.

```
   subroutine defect(tk, tkp1, xk, xkp1, uk, ukp1, umid, resid)

   ! state vector defects subroutine

   ! Compressed Hermite-Simpson method
```

```
! input

!  tk   = time at node k
!  tkp1 = time at node k + 1
!  xk   = state vector at node k
!  xkp1 = state vector at node k + 1
!  uk   = control variable vector at node k
!  ukp1 = control variable at node k + 1
!  umid = control variable at segment midpoint

! output

!  resid = state defect vector for node k

!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!

use simdef

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

dimension resid(nstates)

dimension xk(nstates), xkp1(nstates), xmid(nstates)

dimension uk(ncv), ukp1(ncv), umid(nstates)

dimension xdk(nstates), xdkp1(nstates), xdmid(nstates)

! compute delta time

hk = tkp1 - tk

! evaluate equations of motion
! at beginning and end of segment

call rhs (tk, xk, uk, xdk)

call rhs (tkp1, xkp1, ukp1, xdkp1)

! state vector at midpoint

do i = 1, nstates
   xmid(i) = 0.5d0 * (xkp1(i) + xk(i)) &
             + (hk / 8.0d0) * (xdk(i) - xdkp1(i))
end do

! equations of motion at midpoint

call rhs (tk + 0.5d0 * hk, xmid, umid, xdmid)

! compute state vector defect for this node

do i = 1, nstates
   resid(i) = xkp1(i) - xk(i) &
              - (hk / 6.0d0) * (xdk(i) + 4.0d0 * xdmid(i) + xdkp1(i))
end do

return
end
```

The right-hand-side of the equations of motion is defined in a subroutine called `rhs`. Here is the Fortran source code for this subroutine.

```
subroutine rhs (t, x, u, xdot)

! equations of motion

! input

!  t = current time
!  x = current state vector
!  u = current control vector

! output

!  xdot = equations of motion

!!!!!!!!!!!!!!!!!!!!!!!!!!!!!

use simdef

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

dimension x(nstates), xdot(nstates), u(ncv)

! current control variable

theta = u(1)

! current thrust acceleration

accm = acc / (1.0d0 - beta * t)

! evaluate equations of motion

xdot(1) = x(2)

xdot(2) = (x(3) * x(3) - 1.0d0 / x(1)) / x(1) + accm * sin(theta)

xdot(3) = -x(2) * x(3) / x(1) + accm * cos(theta)

return
end
```

Simulation Results

The following is the NLP algorithm header and the first 20 NLP iterations for this example.

```
---------------------------------------------------------------------------
     START OF THE SEQUENTIAL QUADRATIC PROGRAMMING ALGORITHM
---------------------------------------------------------------------------

     PARAMETERS:
        MODE =  0
        ACC =   0.1490E-07
        SCBOU =   0.1000E+04
        MAXFUN =  5
        MAXIT =  500
        IPRINT =   2
```

```
OUTPUT IN THE FOLLOWING ORDER:
   IT    - ITERATION NUMBER
   F     - OBJECTIVE FUNCTION VALUE
   SCV   - SUM OF CONSTRAINT VIOLATION
   NA    - NUMBER OF ACTIVE CONSTRAINTS
   I     - NUMBER OF LINE SEARCH ITERATIONS
   ALPHA - STEPLENGTH PARAMETER
   DELTA - ADDITIONAL VARIABLE TO PREVENT INCONSISTENCY
   DLAN  - MAXIMUM NORM OF LAGRANGIAN GRADIENT
   KT    - KUHN-TUCKER OPTIMALITY CRITERION
```

```
IT         F           SCV     NA  I    ALPHA    DELTA      DLAN      KT
--------------------------------------------------------------------------------
 1 -0.10000000D+01  0.54D+00 143  0  0.00D+00  0.00D+00  0.69D+03  0.27D+03
 2 -0.12508748D+01  0.48D+00 143  2  0.23D+00  0.00D+00  0.70D+03  0.85D+02
 3 -0.13533963D+01  0.34D+00 143  2  0.28D+00  0.00D+00  0.39D+03  0.68D+02
 4 -0.14323666D+01  0.25D+00 143  1  0.10D+01  0.00D+00  0.28D+03  0.13D+02
 5 -0.12736672D+01  0.34D-01 143  1  0.10D+01  0.00D+00  0.69D+01  0.85D-01
 6 -0.12501491D+01  0.92D-03 143  1  0.10D+01  0.00D+00  0.64D+01  0.38D-02
 7 -0.12528730D+01  0.20D-04 143  1  0.10D+01  0.00D+00  0.63D+01  0.16D-01
 8 -0.12691047D+01  0.61D-03 143  1  0.10D+01  0.00D+00  0.58D+01  0.74D-01
 9 -0.13427471D+01  0.13D-01 143  1  0.10D+01  0.00D+00  0.67D+01  0.17D+00
10 -0.14984896D+01  0.81D-01 143  1  0.10D+01  0.00D+00  0.61D+01  0.18D+00
11 -0.14248866D+01  0.63D-02 143  1  0.10D+01  0.00D+00  0.46D+01  0.49D-02
12 -0.14250937D+01  0.55D-03 143  1  0.10D+01  0.00D+00  0.28D+01  0.58D-02
13 -0.14302719D+01  0.21D-02 143  1  0.10D+01  0.00D+00  0.33D+01  0.31D-02
14 -0.14293934D+01  0.23D-03 143  1  0.10D+01  0.00D+00  0.35D+01  0.35D-02
15 -0.14326524D+01  0.84D-03 143  1  0.10D+01  0.00D+00  0.34D+01  0.76D-02
16 -0.14395305D+01  0.24D-02 143  1  0.10D+01  0.00D+00  0.37D+01  0.13D-01
17 -0.14508342D+01  0.46D-02 143  1  0.10D+01  0.00D+00  0.43D+01  0.21D-01
18 -0.14682534D+01  0.86D-02 143  1  0.10D+01  0.00D+00  0.27D+01  0.16D-01
19 -0.14775650D+01  0.59D-02 143  1  0.10D+01  0.00D+00  0.20D+01  0.92D-02
20 -0.14752891D+01  0.59D-03 143  1  0.10D+01  0.00D+00  0.45D+01  0.27D-02
```

After the NLP algorithm has converged, the software will display a printout of the initial and final conditions. The following is the screen display for this example.

```
initial state vector

radius              =    1.00000000000000
radial velocity     =   0.000000000000000E+000
transverse velocity =    1.00000000000000

final state vector

radius              =    1.52524615470846
radial velocity     =  -1.801473704575790E-024
transverse velocity =   0.809710983907160
```

The following are plots of the thrust angle at the segment midpoints and several other flight parameters. Please note that the radial distance and velocities are non-dimensional. For this example the computer program used 48 discretization nodes.

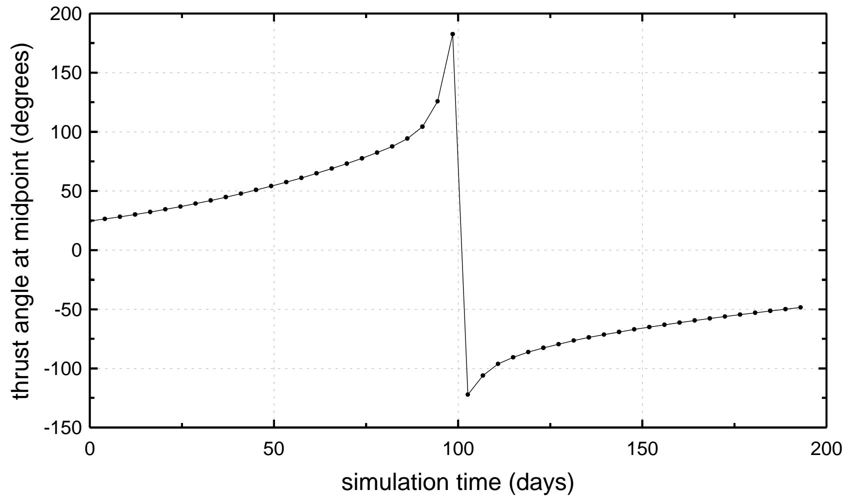

Figure 1. Thrust Angle versus Simulation Time

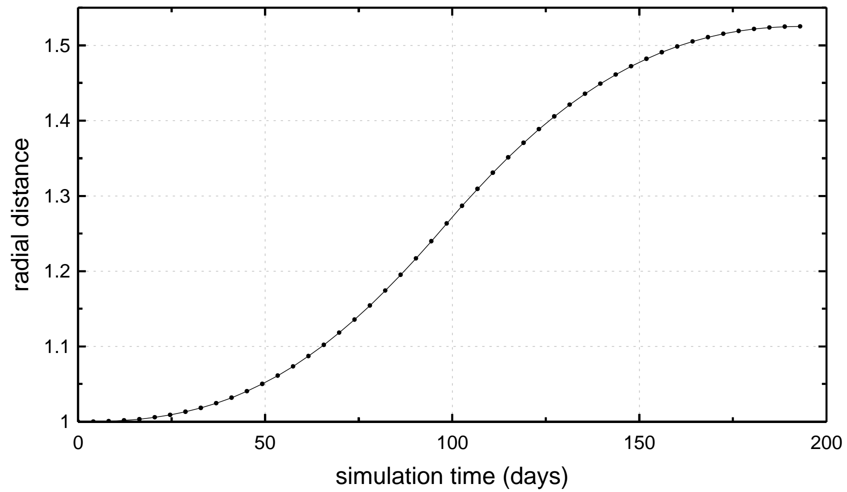

Figure 2. Radial Distance versus Simulation Time

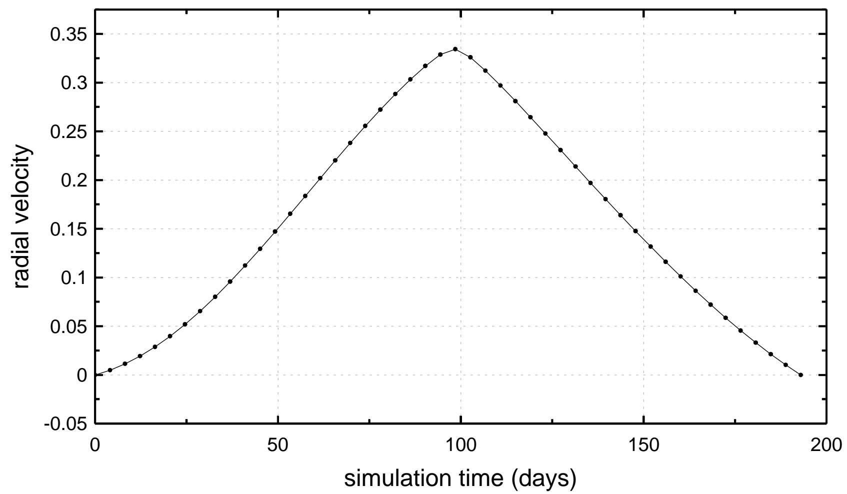

Figure 3. Radial Velocity versus Simulation Time

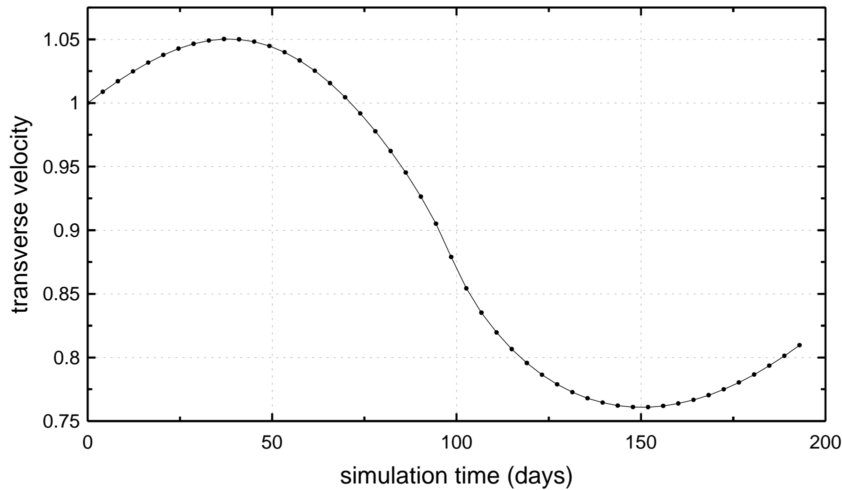

Figure 4. Transverse Velocity versus Simulation Time

References

"Optimal Finite-Thrust Spacecraft Trajectories Using Direct Transcription and Nonlinear Programming", Paul J. Enright, Ph.D. Thesis, University of Illinois at Urbana-Champaign, 1991.

"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.

"Optimal Interplanetary Orbit Transfers by Direct Transcription", John T. Betts, *The Journal of the Astronautical Sciences*, Vol. 42, No. 3, July-September 1994, pp. 247-268.

"Improved Collocation Methods with Application to Direct Trajectory Optimization", Albert L. Herman, Ph.D. Thesis, University of Illinois at Urbana-Champaign, 1995.

"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.

"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.

Practical Methods for Optimal Control Using Nonlinear Programming, John T. Betts, Society for Industrial and Applied Mathematics, Philadelphia, 2001.