A MATLAB Script that Demonstrates Aerospace Trajectory Optimization Using Direct Transcription and Collocation

This brief report describes a technical approach and MATLAB script 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, $\alpha(t)$, 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 "near" Mars. The orbits of the planets are assumed to be circular and coplanar, and the total transfer time is about 193 days.

Mathematical background

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

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

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:

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

where **p** is a vector of problem *parameters* that are time independent.

The initial dynamic variables 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

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

and the control variables

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

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

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

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 can be 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.

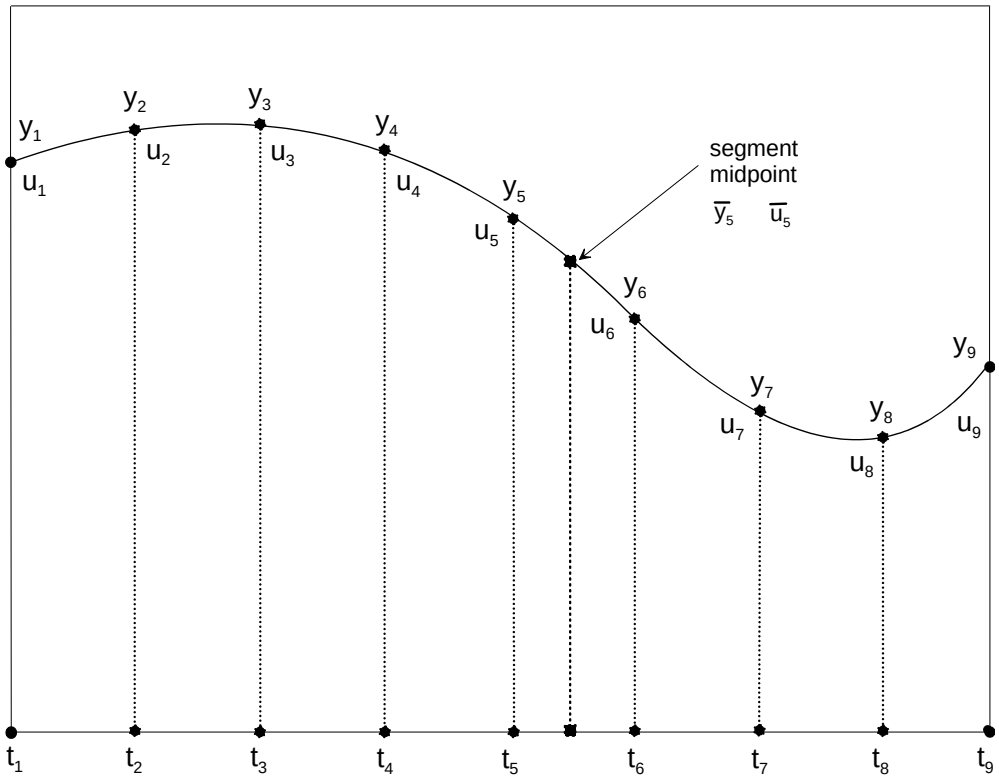


Figure 1 Trajectory and Control Discretization

Collocation

For the *trapezoidal* discretization method, the NLP variables are

$$\mathbf{x} = \left[\mathbf{y}_0, \mathbf{u}_0, \mathbf{y}_1, \mathbf{u}_1, \ldots, \mathbf{y}_f, \mathbf{u}_f, \mathbf{p}, t_0, t_f \right]^T$$

The state equations are approximately satisfied by setting the *defects*

$$\zeta_k = \mathbf{y}_k - \mathbf{y}_{k-1} - \frac{h_k}{2}\left[\mathbf{f}_k + \mathbf{f}_{k-1} \right]$$

to zero for $k = 1, \ldots, n_s$. The step size is denoted by $h_k \equiv t_k - t_{k-1}$, and the right hand side of the differential equations are given by $\mathbf{f}_k \equiv \mathbf{f}\left[\mathbf{y}(t_k), \mathbf{u}(t_k), \mathbf{p}, t_k \right]$.

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

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

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

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

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

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

and

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

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

In these equations h_k is the time interval between segments. For *equal duration* segments h_k is equal to $\left(t_f - t_i\right)/\left(n_N - 1\right)$ 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 trapezoidal approximation to the actual trajectory as defined by the *right-hand-sides* of the equations of motion. Furthermore, driving the vector of defects to zero enforces continuity at the trajectory nodes.

Implicit integration with the trapezoidal method

The general form for integration from time t_n to time t_{n+1} is given by

$$x_{n+1} = x_n + \int_{t_n}^{t_{n+1}} \dot{x} dt = x_n + \int_{t_n}^{t_{n+1}} f\left(x\right) dt$$

The trapezoidal approximation can be written as

$$x_{n+1} = x_n + \frac{\dot{x}_{n+1} + \dot{x}_n}{2} \Delta t$$

The defect condition is given by

$$x_{n+1} - x_n - \frac{\dot{x}_{n+1} + \dot{x}_n}{2} \Delta t = 0$$

which can also be expressed as

$$\Delta_n = x_{n+1} - x_n - \frac{f\left(x_{n+1}\right) + f\left(x_n\right)}{2} \Delta t = 0$$

In these equations, $f(x)$ is the first order equation of motion and $\Delta t = t_{n+1} - t_n$ is the time increment between nodes.

For example, a problem with 8 nodes (7 segments) can be written compactly as follows:

$$\begin{Bmatrix} \Delta_1 \\ \Delta_2 \\ \Delta_3 \\ \Delta_4 \\ \Delta_5 \\ \Delta_6 \\ \Delta_7 \end{Bmatrix} = \begin{bmatrix} -1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & -1 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & -1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & -1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & -1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & -1 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & -1 & 1 \end{bmatrix} \begin{Bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \\ x_6 \\ x_7 \\ x_8 \end{Bmatrix} + \frac{\Delta t}{2} \begin{Bmatrix} f(x_1) + f(x_2) \\ f(x_2) + f(x_3) \\ f(x_3) + f(x_4) \\ f(x_4) + f(x_5) \\ f(x_5) + f(x_6) \\ f(x_6) + f(x_7) \\ f(x_7) + f(x_8) \end{Bmatrix}$$

Chebyshev-Gauss-Lobatto grid points

The location of the grid points in this MATLAB script is determined using a Chebyshev-Gauss-Lobatto (CGL) technique. The CGL algorithm locates points evenly distributed in the interval $\tau \in [-1,+1]$ according to

$$\tau_k = -\cos\left(\frac{\pi k}{N}\right) \qquad k = 0,\ldots,N$$

where *N* is the total number of nodes.

The actual time at individual grid points is determined from

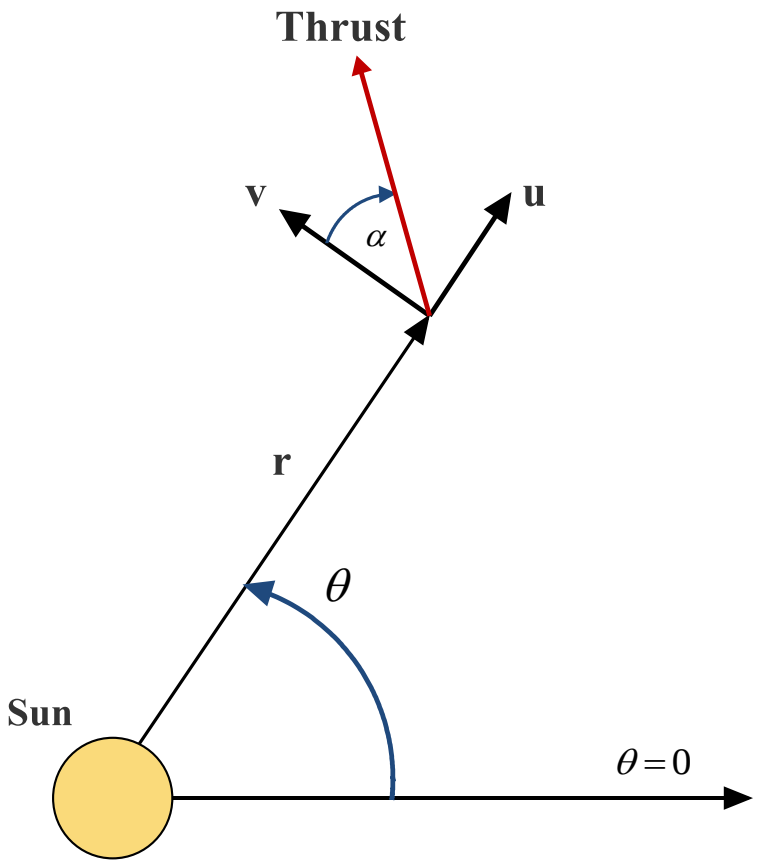
$$t_k = \frac{1}{2}\left\{\left(t_f - t_0\right)\tau_k + \left(t_f + t_0\right)\right\}$$

where t_0 is the initial time and t_f is the final time.

Bryson-Ho problem description

This trajectory optimization problem is modeled in a two-dimensional polar coordinate system. This coordinate system is heliocentric (sun-centered) and all motion is confined to the ecliptic or fundamental plane. The following diagram illustrates the geometry of this coordinate system along with the orientation of the steering angle.

In this diagram, **r** is the heliocentric distance, θ is the polar angle, **v** is the transverse or tangential component of the velocity, **u** is the radial component of the velocity and α is the steering or thrust orientation angle. The steering angle is measured relative to the instantaneous tangential direction and is positive in the clockwise direction. The direction of the propulsive thrust is denoted by the red vector.



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

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

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

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

where

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

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

The spacecraft mass at any elapsed time *t* is determined from

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

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

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

$$r(0) = r_0$$

$$u(0) = u_0 = 0$$

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

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

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

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

The first equation states that the radial velocity should be zero and the second equation is a boundary condition that forces the final velocity to be equal to the *local circular velocity* at the final radial distance.

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

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

Dimensional analysis

To "streamline" the numerical calculations, the fundamental distance, velocity and time in the equations of motion are normalized. In this MATLAB script, all heliocentric distances are normalized with respect to the Astronomical Unit which is equal to `149597870.691` kilometers. Likewise, all velocity values are normalized with respect to the "local circular velocity" at the heliocentric distance of the Earth's circular orbit, r_0. Therefore, the velocity unit is $\sqrt{\mu_S / r_0} = \sqrt{\mu_S}$ since r_0 is equal to 1 AU.

Furthermore, all time values are normalized with respect to $\sqrt{r_0^3/\mu_S} = \sqrt{1/\mu_S}$ since again r_0 is equal to 1 AU. In these equations, μ_S is the gravitational constant of the Sun. The corresponding value for this astronomical constant is equal to `0.00029591220828559l2` AU^3/day^2.

Finally, the acceleration is formulated as the ratio of the acceleration due to propulsive thrust and the acceleration due to the point-mass gravity of the Sun evaluated at a distance of 1 AU. Therefore, the acceleration ratio $\tilde{\alpha}$ is equal to $a_T/a_\oplus$ where a_T is the propulsive acceleration and $a_\oplus$ is the solar acceleration. The acceleration due to the Sun is equal to μ_S/r_0^2.

For this problem, the *non-dimensional* acceleration ratio is given by

$$\tilde{a} = \frac{T/m_0}{\mu_s/r_0^2} = \frac{0.001 \bullet \dfrac{3.781\ \text{N}}{4535.9\ \text{kg}}}{\dfrac{132712441933\,\text{km}^3/\text{sec}^2}{(149597870.691\,\text{km})^2}} = 0.1405$$

The *non-dimensional* flight time is given by t_f/t_{cf} where t_f is the total flight time in days and t_{cf} is the time conversion factor which is equal to $\sqrt{1/\mu_s}$ = `58.13244`.

If we normalize the dimensional propellant flow rate by the initial mass and apply the time conversion factor, the *non-dimensional* propellant flow rate is given by

$$\tilde{\dot{m}} = \frac{\dot{m}}{m_0} t_{cf} = \frac{5.85\ \text{kg/day}}{4535.9\ \text{kg}} \bullet 58.13244 = 0.074974$$

This interplanetary mission requires about 1129 kilograms of propellant.

With these conversions, the two-dimensional equations of motion are modeled as

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

$$\dot{u} = \frac{du}{dt} = \frac{v^2}{r} - \frac{1}{r^2} + a\sin\alpha$$

$$\dot{v} = \frac{dv}{dt} = -\frac{uv}{r} + a\cos\alpha$$

In the second and third equations, $a = \tilde{a}/(1 - \tilde{\dot{m}}t)$.

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 MATLAB script called `dto_trap` 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, control variables and differential equations. The following is the *problem definition* for this example.

```
% number of differential equations

ndiffeq = 3;

% number of control variables

ncv = 1;

% number of discretization nodes

nnodes = 50;

% number of state nlp variables

nlp_state = ndiffeq * nnodes;

% number of control nlp variables

nlp_control = ncv * nnodes;

% total number of nlp variables

nlpv = nlp_state + nlp_control;

% number of state vector defect equality constraints

nc_defect = nlp_state - ndiffeq;

% number of auxilary equality constraints (boundary conditions)

nc_aux = 2;

% total number of equality constraints

nc_total = nc_defect + nc_aux;
```

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 variable at each node is simply set to zero.

The NLP problem is solved by calling a MATLAB interface routine to the SNOPT algorithm. The following is the syntax for this operation.

```
[x, f, inform, xmul, fmul] = snopt(xg, xlb, xub, flow, fupp, 'dtofunc_trap');
```

In the calling arguments `dtofunc_trap` is a user-supplied function that evaluates the objective function and equality constraints for any given trajectory condition.

MATLAB versions of SNOPT for several computer platforms can be found at Professor Philip Gill's web site which is located at http://scicomp.ucsd.edu/~peg/.

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

```
function [f, g] = dtofunc_trap (x)

% objective function and equality constraints

% trapezoidal collocation method

% inputs

%  x = current nlp variable values

% outputs

%  f = vector of equality constraints and
%      objective function evaluated at x

% Orbital Mechanics with MATLAB

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

global nc_defect ndiffeq nnodes nlp_state ncv tarray

% compute state vector defect equality constraints

for k = 1:1:nnodes - 1

    % time elements

    tk = tarray(k);

    tkp1 = tarray(k + 1);

    % state vector elements

    if (k == 1)

       % first node

       nks = 0;

       nkp1s = ndiffeq;
```

```
    else
       % reset to previous node
       nks = (k - 1) * ndiffeq;
       nkp1s = nks + ndiffeq;
    end
    for i = 1:1:ndiffeq
        xk(i) = x(nks + i);
        xkp1(i) = x(nkp1s + i);
    end
    % control variable elements
    if (k == 1)
       % first node
       nkc = nlp_state;
       nkp1c = nkc + ncv;
    else
       % reset to previous node
       nkc = nlp_state + (k - 1) * ncv;
       nkp1c = nkc + ncv;
    end
    for i = 1:1:ncv
        uk(i) = x(nkc + i);
        ukp1(i) = x(nkp1c + i);
    end
    % compute state vector defects for current node
    reswrk = defect_trap(tk, tkp1, xk, xkp1, uk, ukp1);
    % load defect array for this node
    for i = 1:1:ndiffeq
        resid(nks + i) = reswrk(i);
    end
end
% set active defect constraints
% (offset by 1)
for i = 1:1:nc_defect
    f(i + 1) = resid(i);
```

```
end

% current final state vector

xfinal(1) = x(nlp_state - 2);

xfinal(2) = x(nlp_state - 1);

xfinal(3) = x(nlp_state);

% objective function (maximize final radius)

f(1) = -xfinal(1);

% compute auxillary equality constraints
% (final boundary conditions)

f(nc_defect + 2) = xfinal(2);

f(nc_defect + 3) = xfinal(1) * xfinal(3)^2 - 1;

% transpose

f = f';

% no derivatives

g = [];
```

The following is the source code for the MATLAB function that calculates the state defect vector for the trapezoidal collocation algorithm.

```
function resid = defect_trap(tk, tkp1, xk, xkp1, uk, ukp1)

% state vector defects - trapezoid 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

% output

%  resid = state defect vector for node k

% Orbital Mechanics with MATLAB

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

global ndiffeq

% compute delta time

hk = tkp1 - tk;

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

xdk = dto_rhs (tk, xk, uk);
```

```
xdkp1 = dto_rhs (tkp1, xkp1, ukp1);

% compute state vector defect for this node

for i = 1:1:ndiffeq
    resid(i) = xkp1(i) - xk(i) - 0.5d0 * hk * (xdk(i) + xdkp1(i));
end
```

The right-hand-side of the equations of motion is defined in a function called `dto_rhs.m`. Here is the MATLAB source code for this function.

```
function xdot = dto_rhs (t, x, u)

% first-order equations of motion

% required by dto_trap.m

% input

%  t = current time

%  x = current state vector

%  x(1) = r, x(2) = u, x(3) = v

%  u = current control vector

% output

%  xdot = rhs equations of motion

%  xdot(1) = r dot, xdot(2) = u dot, xdot(3) = v dot

% Orbital Mechanics with MATLAB

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

global acc beta

% current control variable

theta = u(1);

% current thrust acceleration

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

% evaluate equations of motion at current conditions

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);
```

Simulation Results

The following is the NLP algorithm header and the last iteration for this example.

```
Major Minors     Step   nCon Feasible  Optimal  MeritFunction    nS Penalty
    44      1  1.0E+00     48 (6.7E-10)(1.5E-06)-1.5247152E+00    48 3.2E+02   R
```

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.

```
          program dto_trap

    trajectory optimization using
 direct transcription and collocation


 initial state vector

 radius               =   1.00000000

 radial velocity      =   0.00000000

 transverse velocity  =   1.00000000


final state vector

 radius                =   1.52446193

 radial velocity       =   0.00000000

 transverse velocity   =   0.80991923
```

The following are plots of the in-plane thrust angle, several other flight parameters and the heliocentric planet orbits and transfer trajectory. Please note that the radial distance and velocities are non-dimensional. For this example the MATLAB script used 50 discretization nodes. In these plots, the asterisks represent the locations of the Chebyshev-Gauss-Lobatto nodes.

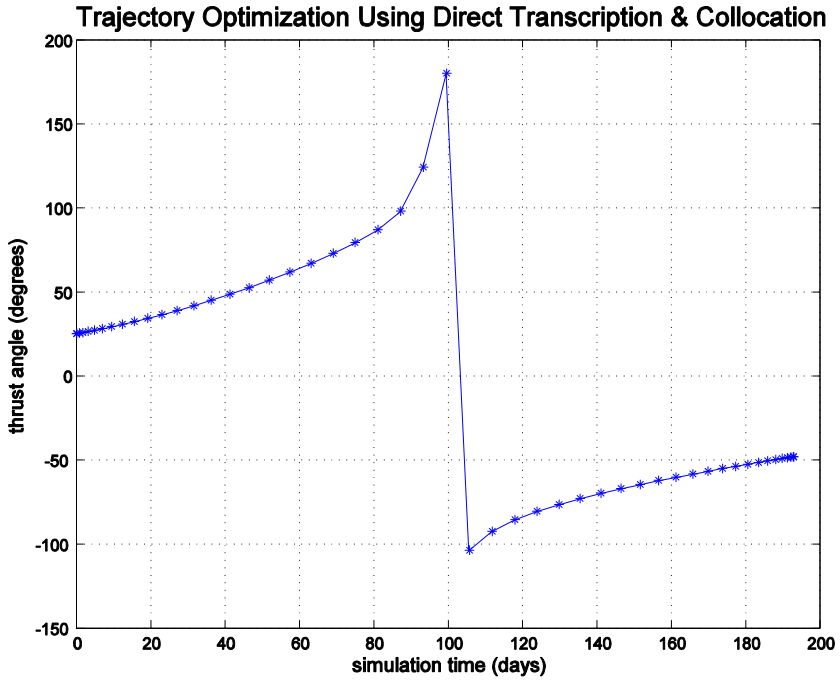


Figure 2. Thrust Angle versus Simulation Time

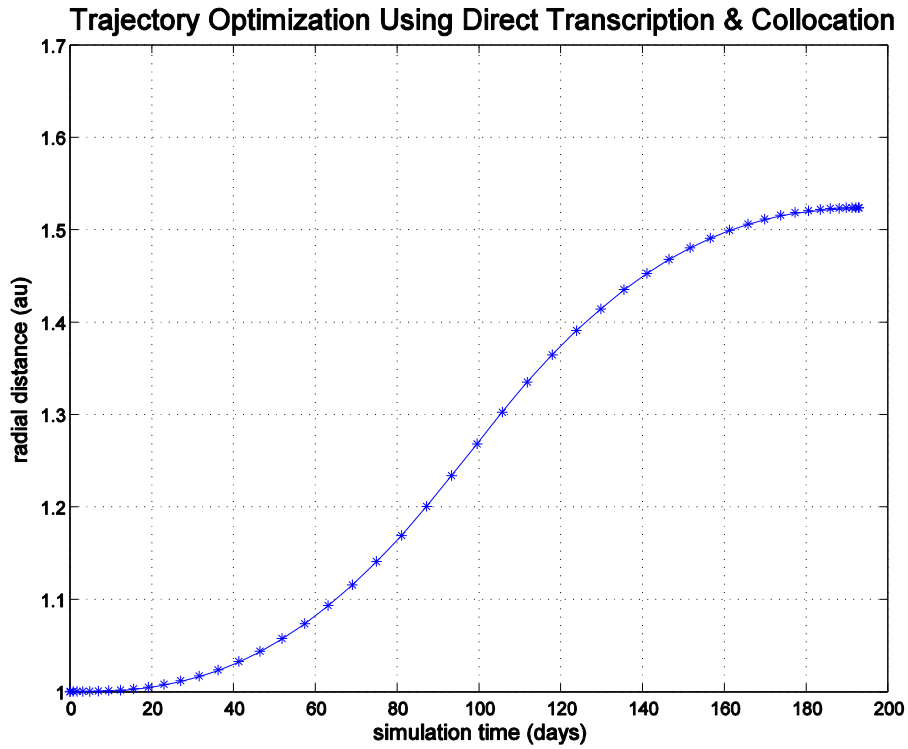


Figure 3. Radial Distance versus Simulation Time

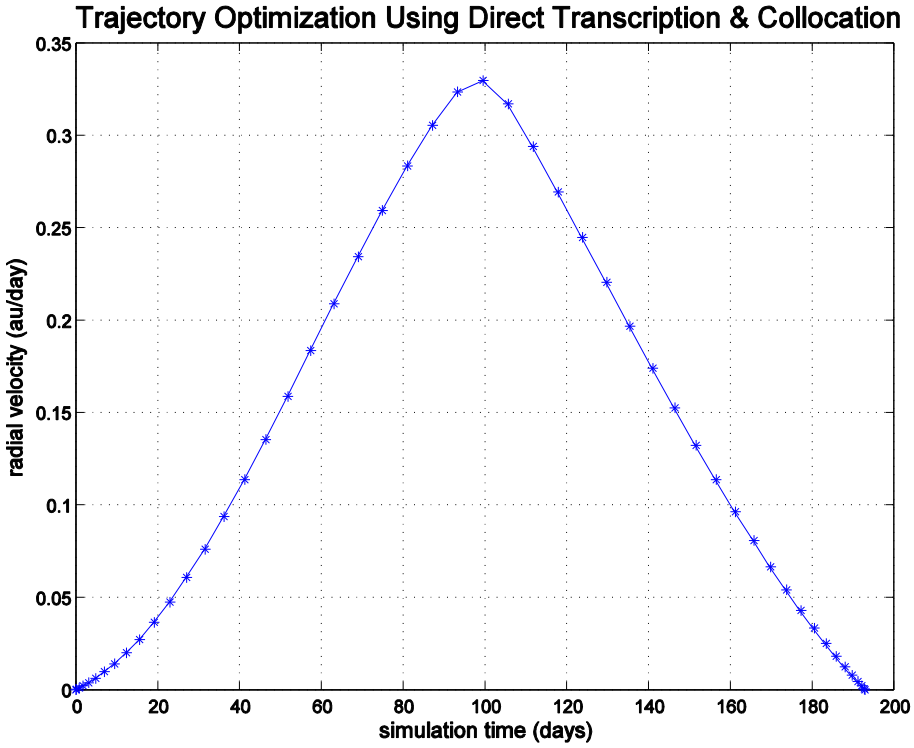


Figure 4. Radial Velocity versus Simulation Time

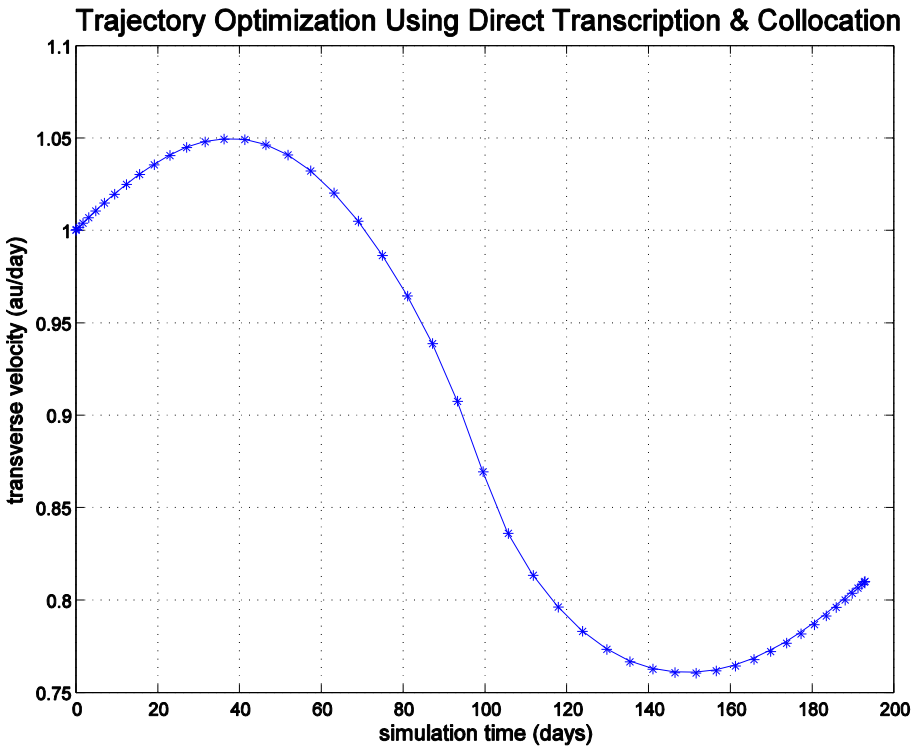


Figure 5. Transverse Velocity versus Simulation Time

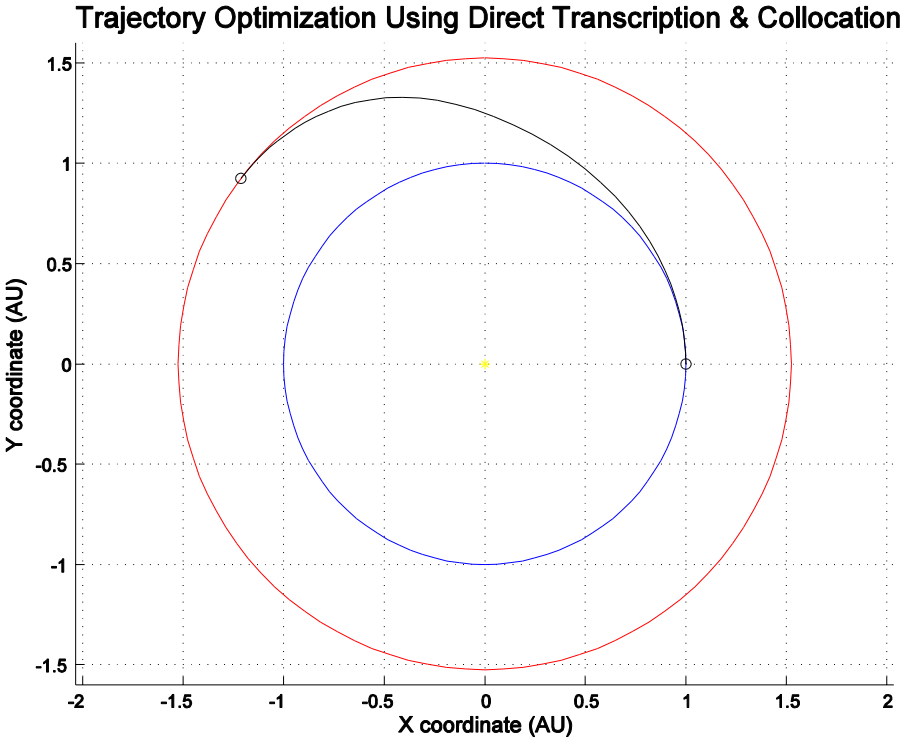


Figure 6. Heliocentric planet orbits and transfer trajectory

The `dto_trap` MATLAB script will create color Postscript disk files of these graphic images. These images include a TIFF preview and are created with MATLAB code similar to

```
print -depsc -tiff -r300 vradial.eps
```

Algorithm resources

"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 and Estimation Using Nonlinear Programming, John T. Betts, Society for Industrial and Applied Mathematics, Philadelphia, 2010.