

## MCOLL: MONTE COLLOCATION TRAJECTORY DESIGN TOOL

Daniel J. Grebow\* and Thomas A. Pavlak\*

In this paper we describe a prototype low-thrust optimization software being developed at JPL. The software tool is based on a collocation algorithm where a trajectory discretization is fitted and adjusted until the underlying dynamics equations of motion are satisfied. The resulting large scale non-linear programming problem may either be optimized with IPOPT or KNITRO. The user specifies path constraints, boundary constraints, and objectives. We describe the collocation algorithm as well as various mesh refinement strategies, and apply the software tool to solve various example problems.

### INTRODUCTION

This paper details our progress in developing a prototype trajectory design software tool that has resulted from a multi-year research and development effort at JPL. The tool, called MColl (MONTE Collocation), relies on a collocation algorithm for integrating trajectories while leveraging JPL's high-fidelity Mission Design & Navigation Software MONTE (Mission-design and Operations Navigation Toolkit Environment).<sup>1</sup> The goal for this software is to enable rudimentary capability of low-thrust trajectory optimization within MONTE. Currently MColl can compute complex ballistic trajectories, as well as optimize high-fidelity low-thrust trajectories, where the propulsive force could either be a solar-electric propulsion (SEP) system or a solar sail. User-defined path constraints, boundary constraints, and objectives may also be specified.

The underlying algorithm in MColl is collocation. Basic to all collocation schemes is a discretization of a trajectory approximation defining nodes of polynomials. At every node, or collocation point, the derivatives of the polynomials are forced to match the governing system differential equations. After the collocation problem is solved, the segment boundary times of the polynomials and/or the degree needs to be adjusted. Possibly segments need to be added or subtracted to meet a user-specified tolerance. This process is called mesh-refinement.

Collocation algorithms have recently received significant attention due to their ability to solve difficult problems in engineering and mathematics. The benefits of collocation strategies are numerous, but perhaps the most significant advantage is the wide radius of convergence these methods enjoy. The software packages COLSYS,<sup>2</sup> COLDAE,<sup>3</sup> AUTO,<sup>4,5</sup> OTIS,<sup>6,7</sup> DIDO,<sup>8</sup> DIRCOL,<sup>9</sup> PROPT,<sup>10</sup> and GPOPS-II<sup>11</sup> all rely on collocation in some capacity. In the literature, collocation strategies differ in (i) the number of segments (one, a few, or many), (ii) the degree of the interpolating polynomial, and (iii) the placement of nodes. There are also differences in the polynomial bases selected for defining the polynomial(s), affecting only the behavior of convergence of the method (but not the final converged solution).

---

\*Mission Design Engineer, Mission Design and Navigation Section, Jet Propulsion Laboratory, California Institute of Technology, 4800 Oak Grove Dr., Pasadena, CA 91109.

The first year of this research and development effort was spent surveying the literature on the topics of collocation and mesh refinement, and comparing the performance of various methods. The important historical contributions to this rich area of research are summarized in sections of the paper below. A generalized collocation algorithm is also provided. The algorithm selected for MColl is an arbitrary, odd-degree, multi-segment strategy with nodes specified at Legendre-Gauss-Lobatto (LGL) points. The derivatives of the collocation constraints are computed using MONTE's automatic differentiation capability. Of the various mesh refinement options investigated, ultimately, three were selected for implementation, including a method that controls error with JPL's standard propagator DIVA.<sup>12</sup> The current version of MColl allows optimization with IPOPT<sup>13</sup> or KNITRO,<sup>14</sup> as specified by the user. In this paper we justify our selection of the algorithms that we selected for implementation and present them in greater detail while highlighting our own contributions. The paper closes with several example problems and comparisons to JPL low-thrust optimization software MALTO<sup>15</sup> or Mystic,<sup>16,17</sup> when possible.

## COLLOCATION

Collocation is a numerical method that enables computation of solutions to ordinary differential equations by fitting piecewise continuous polynomials to a discretization of a trajectory. This idea was perhaps first introduced by de Boor<sup>18</sup> in 1966 for solving boundary value problems (BVPs) for linear differential equations, and later extended by Russell and Shampine<sup>19</sup> to apply more generally to ordinary differential equations. In a 1974 paper, Weiss<sup>20</sup> showed that collocation is equivalent to implicit Runge-Kutta schemes. Both COLSYS<sup>2</sup> (1979) and AUTO<sup>4</sup> were developed for solving BVPs. AUTO<sup>5</sup> is well-known and widely used today.

Historically, optimal control problems are transformed into BVPs using the calculus of variations. Therefore collocation naturally lends to solving problems with control. In the mid-1980s, Hargraves and Paris<sup>21</sup> showed how the optimal control problem can be solved straightforwardly with collocation by casting it as a nonlinear programming (NLP) problem and directly minimizing some objective using sequential quadratic programming. This approach is significantly easier than indirect approaches because it does not require an initial guess for costates (which are often non-intuitive), nor does it involve deriving complicated boundary conditions that are problem specific. There is less numerical sensitivity on the initial guess and path constraints are straightforward to apply, both challenges for indirect methods based on shooting. Because of the sparsity pattern of the Jacobian, Hargraves and Paris leveraged powerful third party optimizers to solve the NLP efficiently. Their software package Optimal Trajectories by Implicit Simulation (OTIS<sup>6</sup>) was first distributed in 1988. In 2008 OTIS won NASA's Software of the Year Award and it is presently maintained by the Glenn Research Center.<sup>22</sup>

Collocation methods continued to gain traction in the astrodynamics community in the 1990s. Betts and Huffman<sup>23,24</sup> published papers on this topic and together they created Sparse Optimal Control Software (SOCS<sup>25</sup>). Enright and Conway<sup>26</sup> showed how piecewise continuous polynomials can be used to fit a discretization where the derivatives are forced to match the vector field at LGL points. This method has been extended to degree seven by Herman<sup>27</sup> and Herman and Conway,<sup>28</sup> and later generalized to any degree by Williams.<sup>29</sup> During this time the processes of solving the NLP problem with optimal control using collocation became widely known as *direct transcription*. Enright and Conway<sup>26</sup> attribute the phrase's first use to Canon.<sup>30</sup>

More recently, so-called pseudospectral methods have received attention. Spectral methods find their origins in solving partial differential equations, particularly complex fluid dynamics problems,

because they outperform both finite differencing and finite element methods. Historically these methods rely on one or a several high-degree polynomials to approximate the dynamics, rather than many piecewise-continuous polynomials. Fahroo and Ross<sup>31,32</sup> describe a pseudospectral method where nodes are located at Chebyshev-Gauss-Lobatto (CGL) points. More recently Ross has developed the software DIDO<sup>8</sup> for solving astrodynamics problems with optimal control using pseudospectral collocation. Research continues for pseudospectral methods where currently Legendre-Gauss-Radau (LGR) points or Legendre-Gauss (LG) points are favored.<sup>33</sup> The software DIRCOL,<sup>9</sup> PROPT,<sup>10</sup> and GPOPS-II<sup>11</sup> all utilize pseudospectral collocation in some capacity.

We refer the reader to the following resourceful survey papers: Betts,<sup>34</sup> Conway,<sup>35</sup> and Topputo and Zang.<sup>36</sup> For more information on direct transcription, see Betts<sup>37</sup> and Conway.<sup>38</sup>

## Collocation Schemes

Perhaps the simplest collocation approach is the 1<sup>st</sup> order Euler method in which the solution is approximated by a series of  $N - 1$  linear segments where  $N$  is the number of discretized nodes along the trajectory. In this approach, the solution is governed by the well-known Euler integration rule

$$\mathbf{x}_{i+1} = \mathbf{x}_i + \Delta t_i \mathbf{f}[t_i, \mathbf{x}_i] \quad (1)$$

where  $\mathbf{x}_i$  denotes the  $i^{\text{th}}$  discrete state along the trajectory,  $\Delta t_i$  represents the time step along the  $i^{\text{th}}$  segment, i.e.,  $t_{i+1} - t_i$ , and  $\mathbf{f}[t_i, \mathbf{x}_i]$  is the vector field evaluation at point  $\mathbf{x}_i(t_i)$ . Thus, the ‘‘slope’’ of each segment is based on the vector field at the current point,  $\mathbf{x}_i$ . Continuity in position and velocity is enforced between adjacent trajectory segments via a series of  $N - 1$  *defect* constraints,  $\Delta_i$ , given by

$$\Delta_i = \mathbf{x}_i - \mathbf{x}_{i+1} + \Delta t_i \mathbf{f}[t_i, \mathbf{x}_i] = \mathbf{0} \quad (2)$$

The Euler collocation constraint, Equation 2, is equivalent to a forward finite differencing scheme.

The trapezoidal collocation method is similar to the Euler approach in that both algorithms use a series of linear trajectory segments. However, the trapezoidal method achieves 2<sup>nd</sup> order accuracy by approximating the solution via the trapezoidal integration rule, yielding defect constraints of the form

$$\Delta_i = \mathbf{x}_i - \mathbf{x}_{i+1} + \frac{\Delta t_i}{2} \{ \mathbf{f}[t_i, \mathbf{x}_i] + \mathbf{f}[t_{i+1}, \mathbf{x}_{i+1}] \} = \mathbf{0} \quad (3)$$

Higher-order collocation rules are achieved by increasing the degree of the interpolating polynomial. In each Gauss-Lobatto collocation formulation, there are  $n$  total state and defect points associated with each  $n^{\text{th}}$  degree polynomial. For each segment, the times associated with the nodes are normalized on the interval  $[-1, 1]$  and are placed via Gauss-Lobatto integration rules. The 3<sup>rd</sup> degree method with 4<sup>th</sup> order of accuracy represents each segment of the discretized trajectory as a Hermite cubic polynomial. The ODE solution is approximated via Simpson’s rule which is a 3<sup>rd</sup> degree Gauss-Lobatto integration method. Thus, the 3<sup>rd</sup> degree Gauss-Lobatto collocation approach is often called the Hermite-Simpson method. The polynomial is constructed from state and derivative information at the endpoints of each trajectory segment and the defects are computed at the midpoint of each trajectory segment, that is,

$$\Delta_i = \mathbf{x}_i - \mathbf{x}_{i+1} + \frac{\Delta t_i}{6} \{ \mathbf{f}[t_i, \mathbf{x}_i] + \mathbf{f}[t_c, \mathbf{x}_c] + \mathbf{f}[t_{i+1}, \mathbf{x}_{i+1}] \} = \mathbf{0} \quad (4)$$

where the midpoint state,  $\mathbf{x}_c$ , is interpolated from the polynomial by the expression

$$\mathbf{x}_c = \frac{1}{2}(\mathbf{x}_i - \mathbf{x}_{i+1}) + \frac{\Delta t_i}{8} \{ \mathbf{f}[t_i, \mathbf{x}_i] - \mathbf{f}[t_{i+1}, \mathbf{x}_{i+1}] \} = \mathbf{0} \quad (5)$$

The 5<sup>th</sup> and 7<sup>th</sup> degree Gauss-Lobatto collocation methods (of order 8 and 12, respectively) are higher-order analogues of the Hermite-Simpson rule. The defect constraints for these methods are detailed by Herman and Conway.<sup>27,28</sup>

## A Generalized Algorithm

For the generalized method presented here, a polynomial segment  $\mathbf{p}_i$  is defined as an  $n^{\text{th}}$  degree polynomial of the following matrix form

$$\mathbf{p}_i(\tau) = \mathbf{C}_i \times [ 1 \quad \tau \quad \tau^2 \quad \dots \quad \tau^n ]^T \quad (6)$$

where, for numerical reasons and simplicity, the time domain  $\tau$  for all segments spans the interval  $[-1, 1]$ . Note that if there are  $l$  state variables, the polynomial  $\mathbf{p}_i$  is an  $l$ -dimensional vector, and the matrix of coefficients  $\mathbf{C}_i$  is of dimension  $l \times (n + 1)$ .

The points where states are collocated are selected corresponding to the roots of a particular representation of Legendre polynomials  $P_n(\tau)$ . Generally the following types of node placement are considered in the literature (although some methods use roots of Chebyshev polynomials, this is less common):

**Legendre-Gauss-Lobatto (LGL)** points are collocated at  $-1$  and  $1$  and at the roots of  $\dot{P}_{n-1}(\tau)$ .

**Legendre-Gauss-Radau (LGR)** points are collocated at the roots of  $P_{n-1}(\tau) + P_n(\tau)$ .

**Legendre-Gauss-Radau reversed (LGRr)** points are the same as LGR but with opposite sign.

**Legendre-Gauss (LG)** points are collocated at the roots of  $P_{n-1}(\tau)$ .

See Figure 1 for a comparison of the node placement strategies for 5<sup>th</sup> degree methods. For this approach, we assume that each segment polynomial is uniquely determined by the states and derivatives at the variable nodes, or the odd nodes in Figure 1. The state and derivatives at the constrained nodes are computed from the segment polynomials, evaluated at the even nodes. Therefore the degree of the polynomial is equal to the total number of nodes desired per segment.

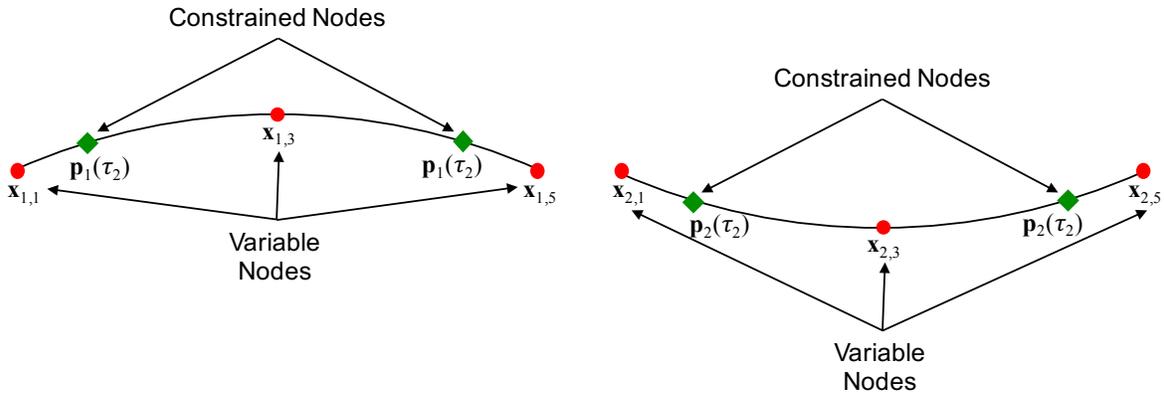
The order of accuracy depends on which type of node placement is selected. LGL has an order of accuracy  $2n - 2$ , while LGR(r) and LG methods have orders of accuracy  $2n - 1$  and  $2n$ , respectively. Although LG is a higher-order method, LG nodes do not include the endpoints  $-1$  and  $1$ , making them slightly more complicated to join segments, or to enforce boundary condition constraints. LGR(r) has a node at only one endpoint and is, therefore, slightly less complicated than LG. (Note also that the nodes for LGR(r) are collocated asymmetrically about 0.) In Figure 1(a), if  $\mathbf{x}_{1,5} = \mathbf{x}_{2,1}$ , it becomes apparent that across two segments, for LG points there are 10 nodes, whereas for LGL there are only 9 (even though the degree of the polynomial is the same). This is why the order of accuracy is higher for LG methods. In general, however, an arbitrary accuracy can be achieved with any node placement strategy by adjusting the degree of the polynomial and/or number of segments.

Depending on the strategy selected for the node locations (LGL, LGR(r), or LG), all methods enforce the following constraints at every node  $j$  of every segment  $i = 1, \dots, m$

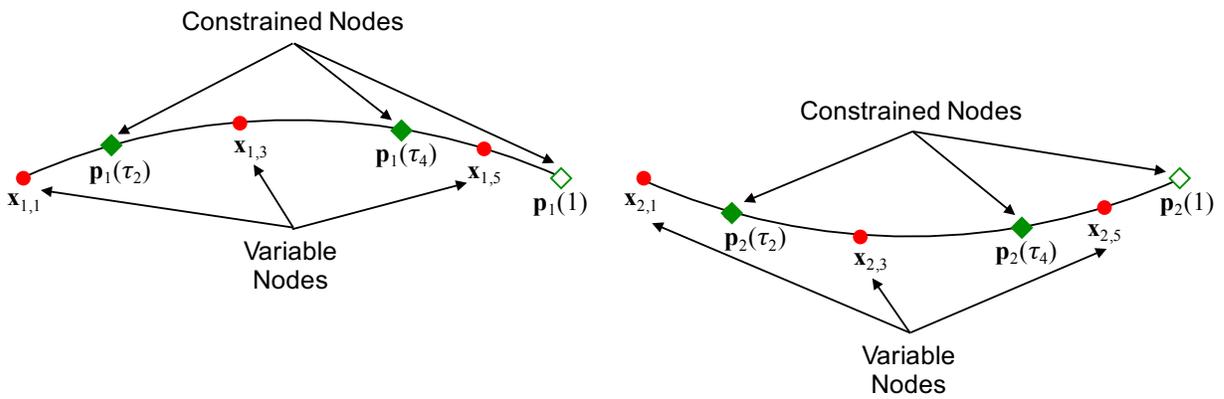
$$\begin{aligned} \mathbf{p}_{i,j} &= \mathbf{x}_{i,j} \\ \dot{\mathbf{p}}_{i,j} &= \dot{\mathbf{x}}_{i,j} \end{aligned} \quad j = 1, \dots, n \quad (7)$$

where  $\mathbf{p}_{i,j} = \mathbf{p}_i(\tau_j)$ ,  $\dot{\mathbf{p}}_{i,j} = d\mathbf{p}_i(\tau_j)/d\tau$ , and  $\tau_j$  is the node time somewhere on the interval  $[-1, 1]$ . The term  $\dot{\mathbf{x}}_{i,j}$  is given by the RHS of the ODEs, i.e.,

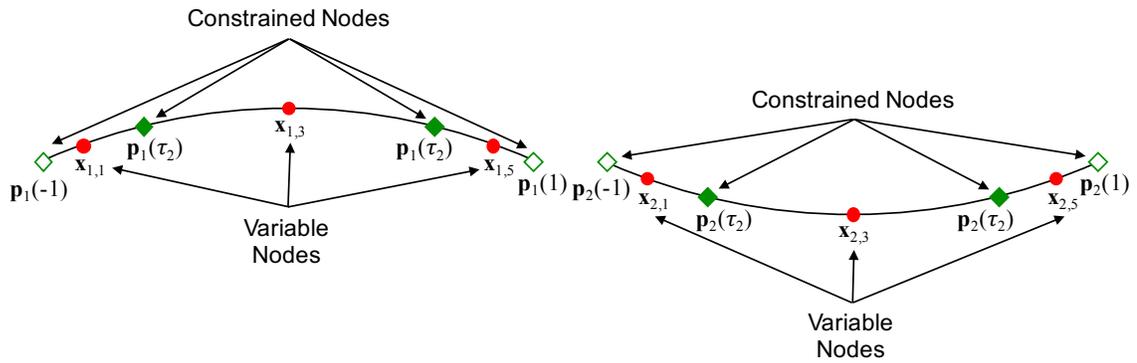
$$\dot{\mathbf{x}}_{i,j} = \frac{\Delta t_i}{2} \mathbf{f}[\tau_j, \mathbf{x}_{i,j}] \quad (8)$$



(a) Legendre-Gauss-Lobatto (LGL)



(b) Legendre-Gauss-Radau (LGR)



(c) Legendre-Gauss (LG)

**Figure 1. Comparison of Collocation Node Placement Strategies**

We can substitute Equations 6 and 8 into 7 and express the system constraints in matrix form as

$$\mathbf{C}_i \times \begin{bmatrix} \boldsymbol{\tau} & \dot{\boldsymbol{\tau}} \end{bmatrix} = \begin{bmatrix} \mathbf{x}_{i,1} & \mathbf{x}_{i,2} & \dots & \mathbf{x}_{i,n} & \dot{\mathbf{x}}_{i,1} & \dot{\mathbf{x}}_{i,2} & \dots & \dot{\mathbf{x}}_{i,n} \end{bmatrix} \quad (9)$$

where

$$\boldsymbol{\tau} = \begin{bmatrix} 1 & 1 & 1 \\ \tau_1 & \tau_2 & \tau_n \\ \tau_1^2 & \tau_2^2 & \dots & \tau_n^2 \\ \vdots & \vdots & & \vdots \\ \tau_1^n & \tau_2^n & & \tau_n^n \end{bmatrix}, \quad \dot{\boldsymbol{\tau}} = \begin{bmatrix} 0 & 0 & 0 \\ 1 & 1 & 1 \\ 2\tau_1 & 2\tau_2 & \dots & 2\tau_n \\ \vdots & \vdots & & \vdots \\ n\tau_1^{n-1} & n\tau_2^{n-1} & & n\tau_n^{n-1} \end{bmatrix} \quad (10)$$

As posed, the system of Equations 9 represents  $2ln$  constraints with  $2ln + l$  unknowns: all the coefficients  $\mathbf{C}_i$  and each node state  $\mathbf{x}_{i,j}$ . Typically the coefficients  $\mathbf{C}_i$  are eliminated as variables by assuming some basis for the polynomials, such as Lagrange interpolation, B-splines, Chebyshev polynomials, etc. The collocation problem is solved when Equations 9 is satisfied. Here we will eliminate the coefficients  $\mathbf{C}_i$  by assuming that the polynomials satisfy state and derivative information at the odd nodes. If an even number of nodes is desired, then the state at the final node may also be used when to construct the polynomial.

For an odd number of nodes, we introduce the constant matrix  $\mathbf{A}$  as follows

$$\mathbf{A} = \begin{bmatrix} 1 & 1 & 1 & \vdots & 0 & 0 & 0 \\ \tau_1 & \tau_3 & \tau_n & \vdots & 1 & 1 & 1 \\ \tau_1^2 & \tau_3^2 & \dots & \tau_n^2 & 2\tau_1 & 2\tau_3 & \dots & 2\tau_n \\ \vdots & \vdots & & \vdots & \vdots & \vdots & & \vdots \\ \tau_1^n & \tau_3^n & & \tau_n^n & n\tau_1^{n-1} & n\tau_3^{n-1} & \dots & n\tau_n^{n-1} \end{bmatrix} \quad (11)$$

The matrix  $\mathbf{A}$  is square and nonsingular. (Note that all subscripts in the previous equation should be odd.) Then the coefficients  $\mathbf{C}_i$  for each segment are uniquely determined from

$$\mathbf{C}_i = \begin{bmatrix} \mathbf{x}_{i,1} & \mathbf{x}_{i,2} & \dots & \mathbf{x}_{i,n} & \dot{\mathbf{x}}_{i,1} & \dot{\mathbf{x}}_{i,2} & \dots & \dot{\mathbf{x}}_{i,n} \end{bmatrix} \times \mathbf{A}^{-1} \quad (12)$$

Now that  $\mathbf{C}_i$  is fully determined, we can compute the polynomial time derivatives at the even nodes. To do so, we introduce the following matrices  $\mathbf{B}$  and  $\mathbf{D}$  which allow us to interpolate the polynomials at any given time  $\tau$  (notice that the matrix  $\mathbf{D}$  corresponds to the time derivative of  $\mathbf{B}$ )

$$\mathbf{B} = \begin{bmatrix} 1 & 1 & 1 & & 1 & 1 \\ -1 & \tau_2 & \tau_4 & & \tau_{n-1} & 1 \\ 1 & \tau_2^2 & \tau_4^2 & \dots & \tau_{n-1}^2 & 1 \\ \vdots & \vdots & \vdots & & \vdots & \vdots \\ -1 & \tau_2^n & \tau_4^n & & \tau_{n-1}^n & 1 \end{bmatrix}, \quad \mathbf{D} = \begin{bmatrix} 0 & 0 & 0 \\ 1 & 1 & 1 \\ 2\tau_2 & 2\tau_4 & \dots & 2\tau_{n-1} \\ \vdots & \vdots & & \vdots \\ n\tau_2^{n-1} & n\tau_4^{n-1} & & n\tau_{n-1}^{n-1} \end{bmatrix} \quad (13)$$

Since states at the even nodes are variables, it is unnecessary to include them as variables in the NLP problem. Instead, their values are interpolated directly from the polynomial representations:

$$\begin{bmatrix} \mathbf{x}_{i,0} & \mathbf{x}_{i,2} & \mathbf{x}_{i,4} & \dots & \mathbf{x}_{i,n-1} & \mathbf{x}_{i,f} \end{bmatrix} = \mathbf{C}_i \times \mathbf{B} \quad (14)$$

We have included the initial and final states on the segment when constructing the matrix  $\mathbf{B}$  so that the initial and final state from the polynomial interpolation are on-hand if they are needed

for additional constraints, such as segment continuity constraints. The subscript ‘0’ refers to the segment initial state and ‘ $f$ ’ corresponds to the final state. For LGL points, the segment initial and final states are collocation points, whereas for LG points, these quantities must be interpolated from the polynomial representations. Once the values of the states at the even nodes are determined, then the constraints are given by

$$\Delta_i = [C_i \times D - [x_{i,2} \ x_{i,4} \ \dots \ x_{i,n-1}]] \times W = 0 \quad (15)$$

The matrix  $W$  is a diagonal matrix that contains the quadrature weights for the even nodes as determined by the node placement strategy (e.g., LGL, LGR(r), or LG).<sup>39</sup> The matrices  $A^{-1}$ ,  $B$ , and  $D$  are all constants and can be pre-computed and stored in memory.

For an even number of nodes, the process is similar but state information at the final node must also be included when constructing the polynomial. Therefore, for an even number of nodes, there is an extra column in the matrices  $A$ ,  $B$ , and  $D$  corresponding to the time  $\tau_n$ .

Suppose we want to solve an initial value problem with  $m$  segments using an odd  $n^{\text{th}}$  degree polynomial. Then, the NLP variables are given by the matrix

$$\mathbf{X} = \begin{bmatrix} x_{1,1} & x_{1,3} & \dots & x_{1,n} \\ x_{2,1} & x_{2,3} & \dots & x_{2,n} \\ \vdots & \vdots & \ddots & \vdots \\ x_{m,1} & x_{m,3} & \dots & x_{m,n} \end{bmatrix} \quad (16)$$

and the constraints are

$$\mathbf{F} = \begin{bmatrix} x_{1,0} - x_{0,des} \\ x_{2,0} - x_{1,f} \\ \vdots \\ x_{m,0} - x_{m-1,f} \\ \Delta_1 \\ \Delta_2 \\ \vdots \\ \Delta_m \end{bmatrix} = 0 \quad (17)$$

where the initial state is  $x_{0,des}$ . This NLP has  $lmn$  variables and  $lmn$  constraints. Recall from Figure 1 that for LGL points there are no diamonds at the segment boundaries; therefore the boundaries between the segments may initially assumed to be equal, yielding a problem with  $l(m-1)$  fewer variables and constraints. We call this the ‘‘simplified LGL’’ method because  $x_{i,n}$  is not included as variables in  $\mathbf{X}$  for the first  $m-1$  segments, and, furthermore, it does not require the  $l(m-1)$  segment continuity constraints in the top partition of  $\mathbf{F}$ .

## MESH REFINEMENT

After the collocation problem is solved, the segment boundary times need to be adjusted and segments may need to be added or subtracted to meet a user-specified tolerance. This process is called mesh refinement, and it is equivalent in principle to allowing for adaptive step-size in explicit propagation schemes. Thus, mesh refinement is an essential part of the computational process for collocation.

There are various approaches for mesh refinement, many of which are described by Russell and Christiansen.<sup>40</sup> (See also Betts and Huffman.<sup>23</sup>) The software AUTO relies on the mesh refinement algorithms presented in Russell and Christiansen's paper. Most of these strategies first work to equally distribute the error across all segments. Once the error is sufficiently equally distributed, the number of segments is updated. Some researchers<sup>41</sup> have suggested meeting error tolerances by adjusting the degree of the polynomial for a given segment instead of adjusting the segment size. Another possibility is to use a Sundman transformation.<sup>42</sup> The Sundman transformation describes the change in time with respect to a new independent time variable that, when integrated along with the system differential equation, equally distributes error. There are some Sundman transformations that are known to work well for particular EOMs, however the strategy is not easily generalized to any set of dynamical equations. Another option that has been utilized with some success in engineering circles is to control the error with an explicit propagator after solving the NLP. (For example, see Dickmans<sup>43</sup> and Hargraves and Paris.<sup>21</sup>) If the error exceeds a user-specified value for a segment, then the segment is split into two segments. Once every segment has been adjusted in this way, the NLP is re-solved.

### Mesh Refinement Strategies

Suppose we have solved the collocation problem with  $n^{\text{th}}$  degree polynomials using the following mesh:

$$\Pi : t_1 < t_2 < \dots < t_m < t_{m+1} \quad (18)$$

Recall that  $m$  is the number of segments and  $n$  is the degree of the polynomial. Since  $\Pi$  is initially a guess, often there is wildly varying error over the initial mesh.

*Method from de Boor.* Carl de Boor's method<sup>44</sup> has been used for mesh refinement both by Russell and Christiansen<sup>40</sup> and Ozimek et al.<sup>45</sup> Initially the times in Equation 18 are adjusted to equally distribute error across each segment while keeping the number of segments fixed. The error is usually expressed as an expansion of  $\Delta t_i = t_{i+1} - t_i$ :

$$e_i = C\Delta t_i^{n+1}\xi_i + O(\Delta t_i^{n+2}) \quad (19)$$

where  $C$  is a computable constant that depends on the degree of the polynomial (see Appendix of Russell and Christiansen<sup>40</sup>). According to de Boor,  $\xi_i$  can be estimated as the  $n^{\text{th}} + 1$  derivative of the solution. This derivative is approximated with the polynomials of the converged solution and the following differencing scheme:

$$\xi \approx \begin{cases} 2\max \left[ \frac{|\mathbf{p}_1^{(n)} - \mathbf{p}_2^{(n)}|}{\Delta t_1 + \Delta t_2} \right], & \text{on } (t_1, t_2) \\ \max \left[ \frac{|\mathbf{p}_{i-1}^{(n)} - \mathbf{p}_i^{(n)}|}{\Delta t_{i-1} + \Delta t_i} \right] + \max \left[ \frac{|\mathbf{p}_{i+1}^{(n)} - \mathbf{p}_i^{(n)}|}{\Delta t_{i+1} + \Delta t_i} \right], & \text{on } (t_i, t_{i+1}) \ i = 2, \dots, m-1 \\ 2\max \left[ \frac{|\mathbf{p}_{m+1}^{(n)} - \mathbf{p}_m^{(n)}|}{\Delta t_{m+1} + \Delta t_m} \right], & \text{on } (t_m, t_{m+1}) \end{cases} \quad (20)$$

where  $\mathbf{p}_i^{(n)}$  is the  $n^{\text{th}}$  derivative with respect to the non-normalized time of the polynomials for segment  $i$ . The new segment boundary times can be updated to asymptotically equally distribute

the error across each segment. The new times are

$$t_{i+1} = I^{-1} \left[ \frac{iI(t_{m+1})}{m} \right], \quad i = 1, \dots, m-1 \quad (21)$$

where

$$I(t) = \int_{t_1}^t \xi(s)^{\frac{1}{n+1}} ds \quad (22)$$

Since  $\xi_i$  is piecewise constant, Equations 21 and 22 are trivial to compute, since the integral can be solved exactly with the rectangle rule. Updating  $\Pi$  with these equations, reconverging, and iterating this process eventually leads to a mesh with error equally distributed. Guesses for the new node states are always computed by interpolating points with the converged solution of the existing mesh. After the error is equally distributed across each segment to within a user-specified tolerance, the number of segments may be updated according to

$$m_{j+1} = \text{round} \left[ m_j \left( \frac{10e_i}{\text{tol}} \right)^{\frac{1}{o+1}} + 5 \right] \quad (23)$$

where  $o$  is the order of the collocation scheme. The entire process repeats until the error tolerances are satisfied.

*Higher-Order Method.* An alternative approach is to use a higher-order method and control the error by comparing to a high-order solution. Russell and Christiansen say that this approach is inefficient because the additional expense of computing the higher order solution: “The approach only seems reasonable if no other reliable estimates [for the error] are available” (p. 67). However we consider it here because it is similar to what occurs with an  $n^{\text{th}}$  order explicit Runge-Kutta scheme and  $n^{\text{th}} + 1$  error control. The error for the  $i^{\text{th}}$  segment can also be written

$$e_i = \|\mathbf{x} - \mathbf{p}_i\| \quad (24)$$

In Equation 24,  $\mathbf{x}$  is the (unknown) solution and  $\mathbf{p}_i$  is the polynomial solution computed with the initial mesh over the interval  $(t_i, t_{i+1})$ . The higher-order approach also uses Equations 21 and 22, but  $\xi_i$ , instead, is approximated with a higher-order solution  $\mathbf{p}_i^*$ . The average error for the  $i^{\text{th}}$  segment is

$$e_i \approx \bar{e}_i = \frac{1}{\Delta t_i} \int_{t_i}^{t_{i+1}} \|\mathbf{p}_i^* - \mathbf{p}_i\| dt \quad (25)$$

Setting Equation 25 to Equation 19 and solving for  $\xi_i$  yields

$$\xi_i \approx \frac{\bar{e}_i}{C \Delta t_i^{n+1}} \text{ on } (t_i, t_{i+1}), \quad i = 1, \dots, m \quad (26)$$

Similar to de Boor’s method, Equation 26 gives  $\xi_i$  that is piecewise constant. The new times are computed using Equations 21 and 22. Once the error is sufficiently equally distributed, the number of segments is updated with Equation 23. As with de Boor’s scheme, the entire process repeats until the error tolerances are met.

*Control with Explicit Propagation (CEP).* Another option for mesh refinement is to check the solution with explicit propagation. If the error over a segment exceeds a user-specified tolerance, the segment is split in two. After all segments have been checked, the NLP is re-solved and the process iterates until a tolerance is met. It is better to start by first removing unnecessary segments because it reduces the size of the NLP. Therefore we propose to add a first step before this task that seeks to *remove segments* if the error from integrating over segment pairs is sufficiently small. The algorithm runs as follows: After the collocation problem is initially solved, pairs of segments are explicitly propagated and the difference between the solutions is computed. If the error is larger than a user-specified tolerance, the segment boundaries are frozen for all subsequent iterations. Otherwise the node point at the interface is removed, i.e., the two segments become one. After all segment pairs have been examined, the NLP is re-solved. The process repeats until all possible segment boundaries are removed, i.e., until the difference between the solution and the result from explicit propagation is larger than some tolerance for all segments. Then the algorithm switches to subdividing. For subdividing, individual segments are explicitly propagated and the difference between the two solutions is computed. If the error is larger than a user-specified tolerance, the segment is split in half, otherwise the segment is unaltered. The NLP is re-solved and segments are once again compared with explicit propagation and subdivided as necessary. The process repeats until the difference between the solution and the results from explicit propagation is below a user-specified tolerance for all segments. As with de Boor’s scheme and the higher-order method, guesses for the new node states are always computed by interpolating points with the converged solution of the existing mesh.

## PROTOTYPE SOFTWARE MCOLL

In this section we summarize the various collocation and mesh refinement strategies that were chosen for implementation in MColl. We also describe some of the features and capability of the software. The section ends with some example problems solved with MColl.

### Methods Chosen & Justification

We decided to avoid implementing even degree algorithms because they have the same number of variables and constraints per segment as the same method but one degree higher, rendering even degree methods useless for numerical applications. A lower-degree method may be faster to converge, but it will require many more segments to achieve the same numerical accuracy as a higher degree method. Since the number of variables and constraints per segment increases at a rate of  $n/2$  and the order of accuracy increases by  $2n$ , algorithms that utilize higher degree polynomials will generally achieve the same accuracy more quickly than the lower degree methods. This rule applies up to the point where the order of accuracy meets the number of digits required for double precision. It is difficult to see the advantages of using methods with  $n > 9$  with double precision computations.

For the various test problems investigated, the simplified LGL method usually outperformed the LG collocation method, even though more segments were required for the LGL method to achieve the same accuracy. We attribute the better performance to the smaller problem size for the simplified LGL method. Therefore, MColl utilizes Herman and Conway’s simplified LGL approach, with a user-defined odd-degree polynomial with  $n = 3, 5, 7, 9$ . (For simplicity, the degree is assumed to be the same across all segments.) The software defaults to  $n = 7$  since, for the test problems we investigated, we observed the best performance for polynomials of degree 7.

For mesh-refinement we found that the higher-order method was typically less efficient than both de Boor and CEP. Explicitly integrating the EOMs for mesh refinement occurs only after solving the collocation problem (not during iteration), therefore, collocation may be the preferable method for any iterative scheme. For ballistic propagations of an initial state, controlling the error with an explicit propagation scheme removes the advantage of using collocation. Compared to CEP, de Boor mesh refinement is faster, and it ends with a solution where the error is equally distributed. However we found that CEP is much more robust and usually requires fewer segments. Furthermore, in CEP the error is controlled directly whereas for de Boor the error is only based on approximations. Since CEP does not initially equally distribute the error, it could potentially require many iterations ( $> 100$ ) more than de Boor for subdividing an initially coarse mesh. This problem can be alleviated with a hybrid de Boor-CEP method. CEP has an advantage of validating an MColl solution against external, reliable software.

The options for mesh refinement are (i) no mesh refinement, (ii) de Boor, (iii) CEP, or (iv) hybrid de Boor-CEP method. For CEP, the user may also indicate if segments should first be removed from the mesh. The hybrid option starts with de Boor’s method for equally distributing the error, and then switches to CEP once error across the segments is equal to a certain tolerance.

### Description of Software

MColl requires an initial guess for the entire trajectory to exist as a trajectory object within MONTE’s binary object archive (BOA). The guess may originate from propagating with MONTE’s propagator DIVA (with or without a control law), or it can be constructed in an external software (e.g., in MATLAB) and imported into MColl. Once an initial guess for the trajectory exists in the BOA, MColl decomposes the trajectory into segments for collocation based on a user-defined initial mesh.

Wrappers have been created for optimization with IPOPT and KNITRO. If the trajectory is ballistic (i.e., uncontrolled), MColl can compute a minimum-norm solution, or it can invoke IPOPT or KNITRO without an objective. If there is control, then MColl also requires an initial guess in the BOA for the control direction profile. Then, when MColl decomposes the trajectory into segments it concatenates to the design vector  $\mathbf{X}$  (recall Equation 16) the control for each segment  $\mathbf{u}_i$ . Additionally the unit magnitude constraint

$$\|\mathbf{u}_i\|^2 = 1 \quad (27)$$

is added to the constraint vector  $\mathbf{F}$  for each segment (recall Equation 17). The control direction is assumed to be fixed for a given segment.

The gravity force model is specified by the user and can be either point-mass, or MONTE full body gravity fields. Solar radiation pressure (SRP) and drag can also be activated. User-defined thruster(s) can be created with solar array and bus power models similar to those available in MALTO and Mystic.

If the user specifies a low-thrust engine, the segment start mass  $m_{0,i}$  and end mass  $m_{0,f}$  are added as variables to the design vector. A “throttle” constraint function  $s_i$  is introduced such that

$$0 \leq s_i \leq 1 \quad (28)$$

where

$$s_i = \frac{m_{0,i} - m_{f,i}}{\dot{m}_{max,i} \Delta t_i} \quad (29)$$

Then the thrust  $T_i$  and mass flow rate  $\dot{m}_i$  for a given segment are

$$T_i = s_i T_{max,i} \quad (30)$$

$$\dot{m}_i = s_i \dot{m}_{max,i} \quad (31)$$

The values  $T_{max,i}$  and  $\dot{m}_{max,i}$  are computed from  $(\mu, \tau)$  thrust-power curves (i.e., polynomial representations of the thrust and mass-flow rate as a function of input power).

MColl can also be used in a solar sailing mode, where the sail model is created from a MONTE-based spacecraft shape model—such as a flat plate—with user-determined characteristics for the area and reflectivity properties. For solar sailing, the control direction  $\mathbf{u}_i$  defines the normal of the spacecraft’s reflective surface. The user also inputs a maximum pitch angle  $\alpha_{max}$  for pitching the sail normal away from the sun direction  $\hat{\mathbf{r}}_s$ . Then the constraints

$$\hat{\mathbf{r}}_s^T \mathbf{u}_i \geq \cos(\alpha_{max}) \quad (32)$$

are enforced at every segment midpoint.

Additional constraints can also be specified at the boundaries, or along the path. Boundary constraints may be specified either at the beginning or end of the trajectory. The constraint can be fixing a MONTE state coordinate to a particular value, or bounding it within a range, as indicated by the user. Constraints for any MONTE computable coordinate can be introduced, including Cartesian coordinates in a particular frame and with respect to a user-specified central body, as well as MONTE spherical and/or Keplerian coordinates. If the user desires to enforce a constraint that is not available in MONTE, a user-defined custom constraint can be created. Constraints can also be applied at a particular epoch along the trajectory, or across all segments (for example, such as a path constraint on minimum radius from a particular body).

The objective function is set similarly to a constraint. It can be any MONTE computable quantity, or it may be determined by evaluating a function supplied by the user. The derivatives for the constraints and objective are computed using MONTE’s built-in automatic differentiation capability. Since non-adjacent segments are independent, all derivatives may be computed efficiently by computing them on a per segment basis.

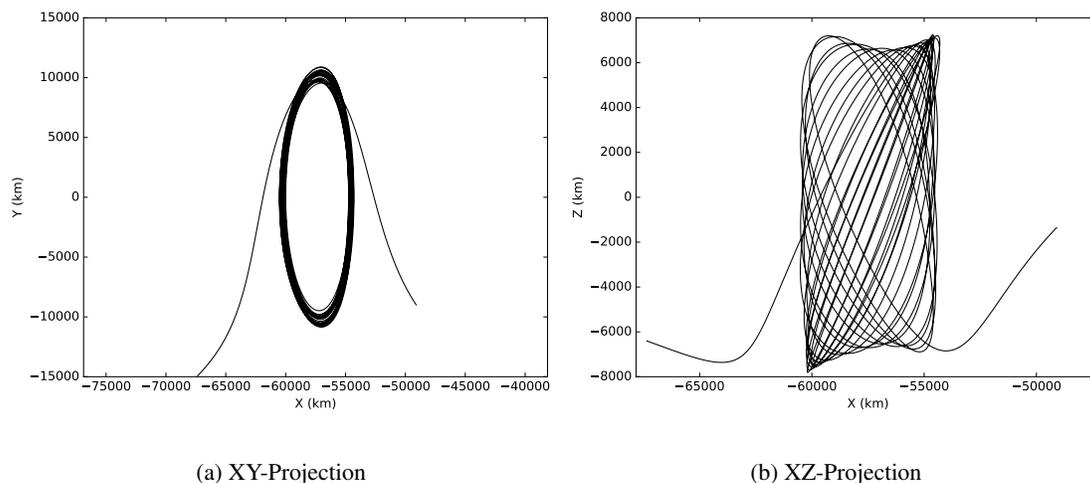
MColl also includes rudimentary support for solving “multi-leg” collocation problems consisting of one or more interrelated sub-problems, i.e. legs. The multi-leg architecture is particularly useful for problems involving time-varying engine parameters (duty cycle changes, forced coast/thrust arcs), close flybys of gravitational bodies, and multi-spacecraft rendezvous. Initially, each sub-problem is created independently with its own MONTE BOA, dynamical model, engine/sail model, control law, constraints, etc. Additional constraints can be imposed to enforce position, velocity, and/or mass continuity between subsequent legs. A multi-leg objective is specified as in the single leg case.

Once a solution is computed, an MColl utility propagates each segment with JPL’s DIVA propagator. Using the same error metric used in CEP, the utility verifies that the final state error for all the segments is below a desired tolerance, thereby validating MColl’s final solution with DIVA.

## Example Problems

*Lissajous Orbit.* In this example a ballistic  $L_1$  Lissajous orbit is computed in a full ephemeris model, starting with an initial guess from the Earth-Moon Circular Restricted 3-Body Problem

(CR3BP). Lissajous orbits have been computed the full ephemeris model before with multiple shooting or the two-level differential corrections scheme.<sup>46</sup> In this example problem we pass to both MColl and MONTE's built-in two-level differential corrector the same initial guess from the CR3BP. The two-level corrector converges in 5.9 seconds and MColl converges in 2.7 seconds on the same computer. CEP was used to mesh refine the MColl trajectory, highlighting the important point that even if an explicit propogator is used to control the error, collocation may still outperform a differential corrector.

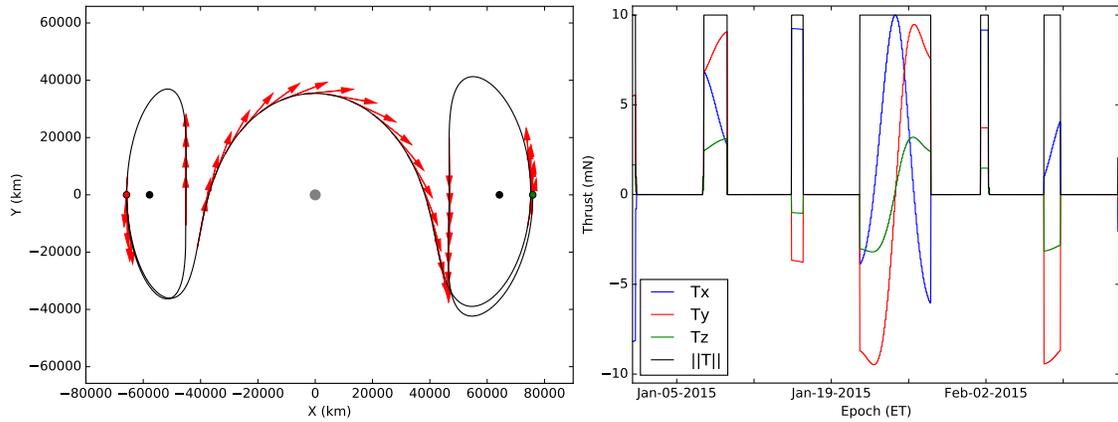


**Figure 2. High-Fidelity Earth-Moon Lissajous**

*Lyapunov-to-Lyapunov Low-Thrust Transfer.* For this example problem, MColl is used to optimize a low-thrust transfer trajectory from an  $L_1$  Lyapunov orbit to an  $L_2$  Lyapunov orbit in the Earth-Moon CR3BP. The  $L_1$  and  $L_2$  orbits have a slightly different Jacobi constant value. Although there is no ballistic connections between the unstable and stable manifolds of the orbits, the invariant manifolds were used as an initial guess for this transfer. The example assumes a constant thrust of 10 mN and the mass-flow rate (independent from power, for this example) is  $\dot{m} = T/I_{sp}g_0$ , where  $I_{sp} = 2,000$  sec. The initial spacecraft mass is 500 kg and the final optimal mass is 499.46 kg. The total CPU time for this example problem is 21.3 seconds.

*ARRM Earth-to-Asteroid Transfer.* A slightly more complicated low-thrust example optimizes an Earth-to-asteroid transfer for the ARRM trajectory. The example starts from a fixed six-state near the Earth and rendezvous with the asteroid 2008 EV5. The total transfer time is 390 days. The spacecraft uses 3 HERMeS high-efficiency thrusters operating at 90% duty cycle. Thrust and  $\dot{m}$  are polynomial functions of input power where the coefficients of the polynomials are determined by the performance of the HERMeS thrusters. The array power is an inverse-square model with a reference power of 47 kW. The power diverted to the spacecraft bus is 500 W. The initial spacecraft mass is 9,945 kg and the optimal final mass is 8,423 kg. The same problem was optimized in JPL's low-thrust software Mystic and the same final mass was computed with a nearly identical thrust profile presented in Figure 4(b). Both Mystic and MColl take approximately 25 seconds to optimize this trajectory starting from a ballistic initial guess.

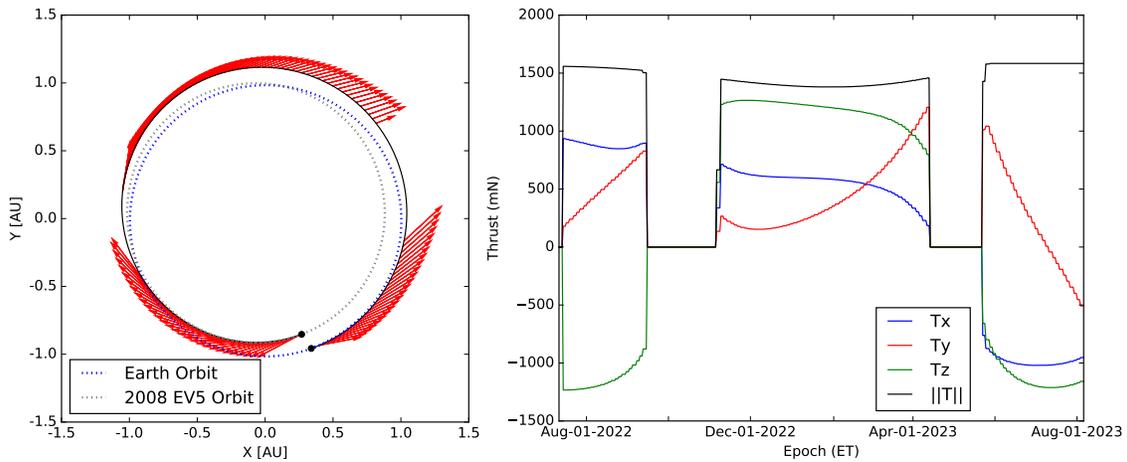
*Solar Sail Solar System Escape.* This problem assumes as spacecraft has been transported to perihelion where a solar sail is deployed for optimal solar system escape. In MColl, solar sailing is



(a) Low-Thrust Transfer

(b) Optimal Thrust Profile

**Figure 3. Mass-Optimal Earth-Moon Lyapunov Transfer**



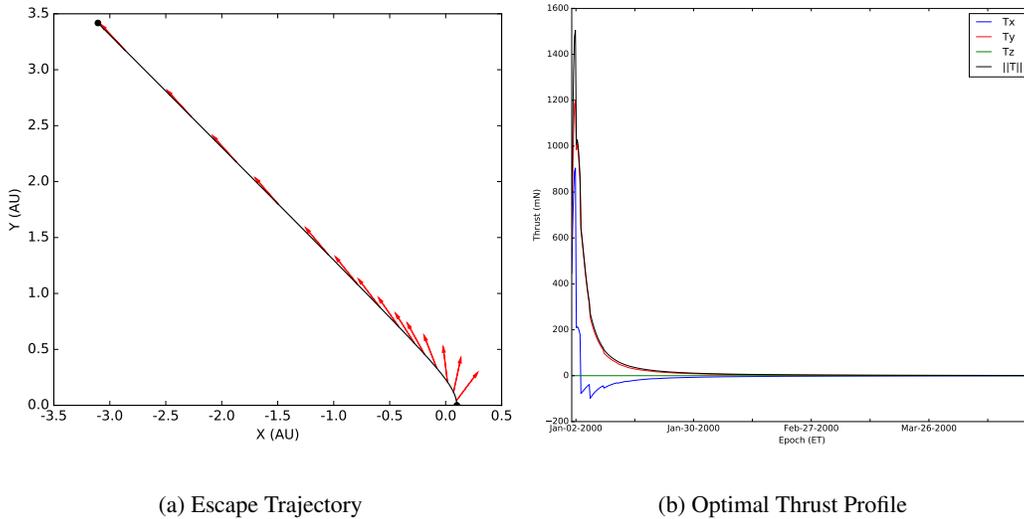
(a) Rendezvous Trajectory

(b) Optimal Thrust Profile

**Figure 4. Mass-Optimal ARRM Outbound Trajectory**

accomplished via a solar radiation pressure model, where the pressure is determined by MONTE by specifying a particular spacecraft shape model with specific reflective properties. For this problem, the reflective properties correspond to an ideal sail where photons are perfectly reflected. The spacecraft mass is 20 kg, and the total sail area is 2,500 m<sup>2</sup>. The objective function is maximize the  $C^3$ . As with the previous examples, the initial guess is ballistic starting from parabolic conditions at perihelion. The compute time is 82 sec, and the final optimal  $C^3$  is 4,381 km<sup>2</sup>/sec<sup>2</sup>. The converged trajectory is shown in Figure 5. (Note that the thrust profile is jagged because MColl assumes that the pitch of the sail is inertially fixed for each segment.)

*NEAScout Asteroid Rendezvous.* MColl also has the capability of optimizing solar sail trajectories with high-fidelity sail models. In this example we solve for asteroid rendezvous with the asteroid



**Figure 5. Maximum- $C_3$  Solar Sail Escape Trajectory**

1991 VG, the target body of interest for the NEAScout mission. The NEAScout spacecraft mass is assumed to be 14 kg. The total area for of the sail is  $86 \text{ m}^2$  and the diffuse and specular reflectivity factors are  $\kappa_\nu = 9.7562e-4$  and  $\kappa_\mu = 0.4277$ , respectively. Furthermore, a 90% duty cycle is imposed on the sail for operations margin. The sail normal cannot be pitched further than  $50^\circ$  away from the Sun line. From MColl, the optimal time of flight is 795.5 days. The converged solution is shown in Figure 6, including a plot of the angle between the sail normal and the Sun direction (see Figure 6(c)). Small violations in the  $\alpha_{max}$  around pitch angles of  $50^\circ$  are present because Equation 32 is only enforced at the segment midpoint. The number of segments can be increased for a smoother thrust profile if desired. This example problem took 27 minutes to converge.

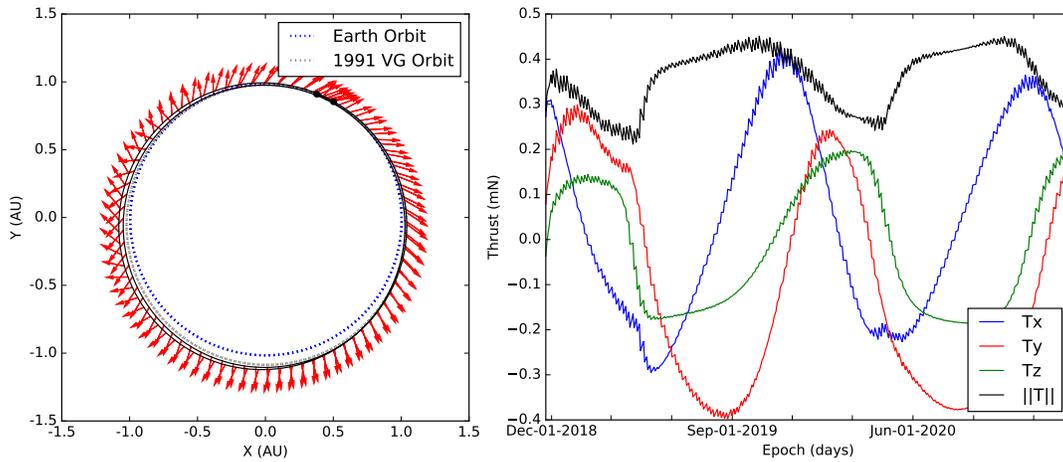
## CONCLUSION

In this paper we describe a prototype low-thrust software MColl that is being developed at JPL. We detail the results of our investigation of collocations methods as well as our justification for selecting the various algorithms we have chosen to implement in MColl. To date, the software can be used to compute ballistic trajectories, such as the L1 Lissajous example, as well as high-fidelity low-thrust optimal trajectories, e.g., the ARRM (SEP) and NEAScout (sail) transfers. The user can specify their own trajectory constraints and/or objective functions for optimization with either IPOPT or KNITRO. For the example problems presented in this paper, we have found the performance of MColl to be comparable to other JPL low-thrust software tools Mystic and MALTO.

## ACKNOWLEDGMENTS

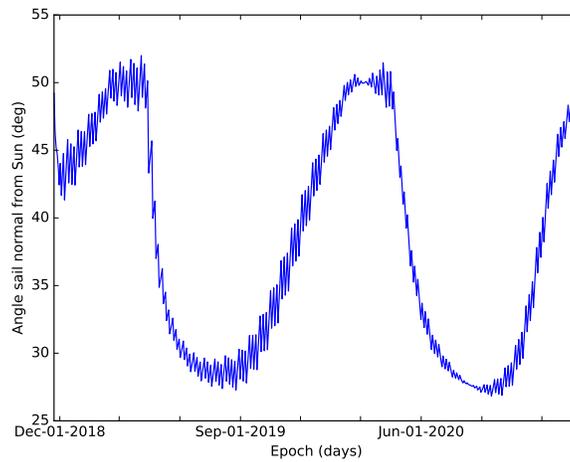
The authors thank Juan Arrieta for creating the MONTE Python interface with IPOPT. The authors also thank Robert Pritchett and Rolfe Power for testing MColl and exercising the software with the example problems presented in this paper.

This work was funded by NASA's Advanced Multi-Mission Operations System (AMMOS). The work described in this paper was performed at the Jet Propulsion Laboratory, California Institute



(a) Rendezvous Trajectory

(b) Optimal Thrust Profile



(c) Angle Sail Normal from Sun line

**Figure 6. NEAScout Trajectory**

of Technology, under contract with the National Aeronautics and Space Administration. Copyright 2017 California Institute of Technology. Government sponsorship acknowledged.

## REFERENCES

- [1] R. Sunseri, H.-C. Wu, J. Evans, T. Drain, and M. Guevara, "Mission Analysis, Operations, and Navigation Toolkit Environment (MONTE) Version 040," *NASA Tech Briefs*, Vol. 36, No. 9, 2012.
- [2] U. Ascher, J. Christiansen, and R. Russell, "COLSYS-A Collocation Code for Boundary-Value Problems," *Codes for Boundary Value Problems in Ordinary Differential Equations* (G. d. Pillo and M. Roma, eds.), Springer-Verlag, 2006.
- [3] U. Ascher and R. Spiteri, "Collocation Software for Boundary Value Differential-Algebraic Equations," *SIAM Journal on Scientific Computing*, Vol. 15, 1994, pp. 938–952.
- [4] E. Doedel, "AUTO, a Program for the Automatic Bifurcation Analysis of Autonomous Systems," *Congressus Numerantium*, Vol. 30, 1981, pp. 265–384.

- [5] E. Doedel and B. Oldeman, "AUTO-07P: Continuation and Bifurcation Software for Ordinary Differential Equations," Concordia University, Montreal, Canada, Jan. 2012.
- [6] C. Hargraves and S. Paris, "Optimal Trajectories by Implicit Simulation (OTIS)," Air Force Wright Aeronautical Lab., AFWAL-TR-88-3057, Wright Patterson AFB, OH, Nov. 1988.
- [7] NASA Glenn Research Center, "OTIS: Optimal Trajectories by Implicit Simulation," <http://otis.grc.nasa.gov/request.html>. [Online; accessed Sep. 25, 2015].
- [8] I. M. Ross, "A Beginner's Guide to DIDO (Ver. 7.3), A MATLAB Application Package for Solving Optimal Control Problems," Elissar, LLC, 2007.
- [9] O. von Stryk, "User's Guide for DIRCOL (Version 2.1): A Direct Collocation Method for the Numerical Solution of Optimal Control Problems," Fachgebiet Simulation und Systemoptimierung (SIM), Technische Universität Darmstadt, 2000.
- [10] P. Rutquist and M. Edvall, "PROPT - Matlab Optimal Control Software," 1260 SE Bishop Blvd Ste E, Pullman, WA, 2008.
- [11] M. Patterson and A. Rao, "GPOPS - II: A MATLAB Software for Solving Multiple-Phase Optimal Control Problems Using hp-Adaptive Gaussian Quadrature Collocation Methods and Sparse Nonlinear Programming," *ACM Transactions on Mathematical Software*, Vol. 39, No. 3, 2013, pp. 1843–1851.
- [12] F. Krogh, "DIVA/SIVA, Chapter 14.1 of MATH77, Release 4.0," Jet Propulsion Laboratory, California Institute of Technology, Pasadena, CA, JPL D-1341, Rev. C, May 1992.
- [13] A. Wächter and L. Biegler, "On the Implementation of a Primal-Dual Interior Point Filter Line Search Algorithm for Large-Scale Nonlinear Programming," *Mathematical Programming*, Vol. 106, No. 1, pp. 25–57.
- [14] R. Byrd, J. Nocedal, and R. Waltz, "KNITRO: An Integrated Package for Nonlinear Optimization," *Large-Scale Nonlinear Optimization*, Vol. 76, Berlin: Springer-Verlag, 1979.
- [15] J. Sims, P. Finlayson, M. Vavrina, and T. Kowalkowski, "Implementation of a Low-Thrust Trajectory Optimization Algorithm for Preliminary Design," *AIAA/AAS Astrodynamics Specialist Conference, Paper No. AIAA-2006-6746*, Keystone, CO, Aug. 21-24 2006.
- [16] G. Whiffen. "Static/Dynamic Control for Optimizing a Useful Objective," United States Patent No. 6,496,741. Issued Dec. 17, 2002, Filed Mar. 25, 1999.
- [17] G. Whiffen, "Mystic: Implementation of the Static Dynamic Optimal Control Algorithm for High-Fidelity, Low-Thrust Trajectory Design," *AIAA/AAS Astrodynamics Specialist Conference, Paper No. AIAA 2006-6741*, Keystone, CO, Aug. 21-24 2006.
- [18] C. de Boor, *The Method of Projections as Applied to the Numerical Solution of Two Point Boundary Value Problems using Cubic Splines*. PhD thesis, University of Michigan, Ann Arbor, Michigan, 1966.
- [19] R. Russell and L. Shampine, "A Collocation Method for Boundary Value Problems," *Numerische Mathematik*, Vol. 19, No. 3, 1972, pp. 1–28.
- [20] R. Weiss, "The Application of Implicit Runge-Kutta and Collocation Methods to Boundary Value Problems," *Mathematics of Computation*, Vol. 28, No. 126, 1974, pp. 449–464.
- [21] C. Hargraves and S. Paris, "Direct Trajectory Optimization Using Nonlinear Programming and Collocation," *J. Guid. Control Dyn.*, Vol. 10, No. 4, 1987, pp. 338–342.
- [22] See <http://www.techbriefs.com/component/content/article/5625>, 2009. [Online; accessed Sep. 7, 2017].
- [23] J. Betts and P. Huffman, "Application of Sparse Nonlinear Programming to Trajectory Optimization," *J. Guid. Control Dyn.*, Vol. 15, No. 1, 1992, pp. 198–206.
- [24] J. Betts and P. Huffman, "Mesh Refinement in Direct Transcription Methods for Optimal Control," *Optimal Control Applications AMPERSAND Methods*, Vol. 19, 1998, pp. 1–21.
- [25] J. Betts and P. Huffman, "Sparse Optimal Control Software SOCS," The Boeing Co., Rept. MEA-LR-085, Seattle, WA, Jul. 1997.
- [26] P. Enright and B. Conway, "Discrete Approximations to Optimal Trajectories Using Direct Transcription and Nonlinear Programming," *J. Guid. Control Dyn.*, Vol. 15, No. 4, 1992, pp. 994–1002.
- [27] A. Herman, *Improved Collocation Methods with Applications to Direct Trajectory Optimization*. PhD thesis, Department of Aerospace Engineering, University of Illinois at Urbana-Champaign, Urbana, Illinois, Sep. 1995.
- [28] A. Herman and B. Conway, "Direct Optimization Using Collocation Based on High-Order Gauss-Lobatto Quadrature Rules," *J. Guid. Control Dyn.*, Vol. 19, No. 3, 1996, pp. 592–599.
- [29] P. Williams, "Hermite-Legendre-Gauss-Lobatto Direct Transcription in Trajectory Optimization," *J. Guid. Control Dyn.*, Vol. 32, No. 4, 2009, pp. 1392–1395.
- [30] M. Canon, C. Cullum, and E. Polak, *Theory of Optimal Control and Mathematical Programming*. New York: McGraw-Hill, 1970.

- [31] F. Fahroo and I. Ross, "Trajectory Optimization by Indirect Spectral Collocation Methods," *Proceedings of the AIAA/AAS Astrodynamics Conference*, No. AIAA-2000-4028, Denver, CO, August 2000.
- [32] F. Fahroo and I. Ross, "Direct Trajectory Optimization by a Chebyshev Pseudospectral Method," *J. Guid. Control Dyn.*, Vol. 25, No. 1, 2002, pp. 160–166.
- [33] D. Garg, M. Patterson, W. Hager, A. Rao, D. Benson, and G. Huntington, "A Unified Framework for the Numerical Solution of Optimal Control Problems using Pseudospectral Methods," *Automatica*, Vol. 46, 2010, pp. 1843–1851.
- [34] J. Betts, "Survey of Numerical Methods for Trajectory Optimization," *J. Guid. Control Dyn.*, Vol. 21, No. 2, 1998, pp. 193–207.
- [35] B. Conway, "A Survey of Methods Available for the Numerical Optimization of Continuous Dynamical Systems," *Journal of Optimization Theory and Applications*, Vol. 52, No. 2, 2012, pp. 271–306.
- [36] F. Topputo and C. Zhang, "Survey of Direct Transcription for Low-Thrust Space Trajectory Optimization and Applications," *Abstract and Applied Analysis*, Vol. 2014, No. Article ID 851720, 2014.
- [37] J. Betts, *Practical Methods for Optimal Control using Nonlinear Programming*. Philadelphia: Society for Industrial and Applied Mathematics, 2 ed., 2010.
- [38] B. Conway, *Spacecraft Trajectory Optimization*. Cambridge: Cambridge University Press, 2010.
- [39] For example, for LGL nodes, see Greg von Winckel's MATLAB code `lglnodes.m`, <http://www.mathworks.com/matlabcentral/fileexchange/4775-legendre-gauss-lobatto-nodes-and-weights/content/lglnodes.m>, 2004. [Online; accessed Sep. 25, 2015].
- [40] R. Russell and J. Christiansen, "Adaptive Mesh Selection Strategies for Solving Boundary Value Problems," *SIAM J. Numer. Anal.*, Vol. 58, No. 1, 1978, pp. 353–370.
- [41] C. Darby, W. Hager, and A. Rao, "An hp-Adaptive Pseudospectral Method for Solving Optimal Control Problems," *Optim. Control Appl. Meth.*, Vol. 32, No. 4, 2011, pp. 476–502.
- [42] B. Leimkuhler and S. Reich, *Simulating Hamiltonian Dynamics*. Cambridge University Press, 2004.
- [43] E. Dickmanns, "Efficient Convergence and Mesh Refinement Strategies for Solving General Ordinary Two-Point Boundary Value Problems with Collocated Hermite Approximation," *2nd IFAC Workshop on Optimization*, Oberpfaff-Enhofen, FRG, Sept. 1980.
- [44] C. de Boor, "Good Approximation by Splines with Variable Knots. II," *Conference on the Numerical Solution of Differential Equations, Lecture Notes in Mathematics*, Vol. 363, New York: Springer, 1973.
- [45] M. Ozimek, D. Grebow, and K. Howell, "A Collocation Approach for Computing Solar Sail Lunar Pole-Sitter Orbits," *Open Aerospace Eng. J.*, Vol. 3, No. 1, 2010, pp. 65–75.
- [46] K. Howell and H. Pernicka, "Numerical Determination of Lissajous Trajectories in the Restricted Three-Body Problem," *Celestial Mechanics*, Vol. 41, 1988, pp. 107–124.