Implicit Formulations of Bounded-Impulse Trajectory Models for Preliminary Interplanetary Low-Thrust Analysis

Robert D Falck* and Steven L. McCarty† and Jeffrey Pekosh‡ and Kaushik Ponnapalli§
NASA Glenn Research Center, Cleveland, OH, 44135

The bounded-impulse approach to low-thrust interplanetary trajectory optimization is widely used. In an effort to efficiently implement this approach using NASA's OpenMDAO optimization software, the authors have implemented implicit formulations of the forward shooting/backwards-shooting methods commonly used in bounded-impulse models. These implicit approaches allow for vectorization of the underlying calculations which can significantly reduce runtime in interpreted languages. An implicit approach may be either converged by using an underlying nonlinear solver to converge the state propagation, or as a constraint in an optimizer-driven multiple-shooting approach. Significant computational efficiency gains are realized through the utilization of the modular approach to unified derivatives. Further computational efficiency is achieved by capitalizing on the sparsity of the constraint Jacobian matrix. This work demonstrates that a vectorized multiple-shooting approach for propagating a state-time history is superior in terms of computational efficiency as the number of segments in the state-propagation is increased.

I. Nomenclature

χ	=	generalized Sundman anomaly
ϕ	=	thrust elevation angle
θ	=	thrust right-ascension angle
$C3$	=	hyperbolic excess energy
m	=	mass
$\mathbf{r}$	=	position vector
$\mathbf{R}(\mathbf{x},\mathbf{y})$	=	residual function for inputs $\mathbf{x}$ and implicit outputs $\mathbf{y}$
t	=	time
T	=	thrust magnitude
$\mathbf{u}$	=	thrust vector
$\mathbf{v}$	=	velocity vector
v_∞	=	hyperbolic excess velocity

Subscripts

$\{\}_0$	=	initial value at departure
$\{\}_f$	=	final value at arrival
$\{\}_i$	=	value in i^{th} segment
$\Delta\{\}_{mid}$	=	error at matchpoint

II. Introduction

Bounded impulse trajectory models have been a valuable tool for preliminary low-thrust mission design. Sims and Flanagan developed a parameterization for low-thrust trajectory optimization which discretizes a continuous

*Aerospace Engineer, Mission Architecture and Analysis Branch, robert.d.falck@nasa.gov, AIAA Member.
†Aerospace Engineer, Mission Architecture and Analysis Branch, steven.mccarty@nasa.gov.
‡Aerospace Engineer, Mission Architecture and Analysis Branch, jeffrey.d.pekosh@nasa.gov, AIAA Student Member.
§Aerospace Engineer, Vantage Partners LLC, kaushik.s.ponnapalli@nasa.gov, AIAA Member.

low-thrust trajectory arc into a series of ballistic arcs and associated impulsive maneuvers [1, 2]. More recently, Ellison, Englander, and Conway improved the algorithm by reformulating it to use the Sundman anomaly as the independent variable of integration - referred to as the Multiple Gravity-Assist Low-Thrust Sundman (MGALTS) transcription [3].

As typically posed in the literature, these methods use two legs, one forward-propagating and one backward-propagating with constraints imposed to guarantee continuity at a matchpoint. Each leg is generally propagated using an explicit single-shooting technique. In an effort to utilize the MGALTS transcription as part of a low-thrust spacecraft multidisciplinary design optimization, the authors have implemented MGALTS in the OpenMDAO framework[4]. OpenMDAO is a software framework that enables efficient multidisciplinary analysis, primarily through gradient-based optimization. Other authors have demonstrated that the use of analytic gradients significantly improves the computational efficiency of bounded-impulse methods[5, 6]. In particular, in Reference 5, Ellison et.al demonstrate a significant performance improvement when using a chain-rule approach to applying analytic derivatives through an explicit propagation approach. In this paper, the authors demonstrate the potential for further efficiency gains by reformulating the propagation across a single *leg* of a trajectory implicitly. These implicit approaches improve performance through three mechanisms: vectorization, a more efficient approach to derivative calculation across iterative systems, and greater sparsity of the constraint Jacobian matrix. The following sections will demonstrate three ways of implementing propagation of the trajectory state within a leg, and the impact of each formulation on the calculation of the derivatives for the optimizer.

III. Formulation

Bounded-impulse low-thrust trajectory models typically model a single body-to-body transfer as two *legs*. One leg starts at the departure body and propagates forward in time. The other leg starts at the arrival body and propagates backwards in time. Each leg consists of one or more segments which are analytically propagated. Using more segments in a leg tends to increase the accuracy of the transcription at some computational expense. Following the analytic propagation of each segment, the velocity is changed impulsively based on the direction of a thrust vector, which is assumed to be fixed for that segment. The optimizer is then responsible for modifying the initial and final mass, the initial and final velocity, the thrust vector along the trajectory, and the total duration of the trajectory such that the legs are continuous in state and time. This is achieved by ensuring that the final position, velocity, mass, and time of the two legs are within an acceptable tolerance at the *matchpoint*. In the MGALTS transcription, the duration of the propagation adjusted by setting the total change in Sundman anomaly (χ) over the entire *phase* (both forward and backwards legs). Figure 1 graphically depicts a typical phase in a bounded-impulse method.

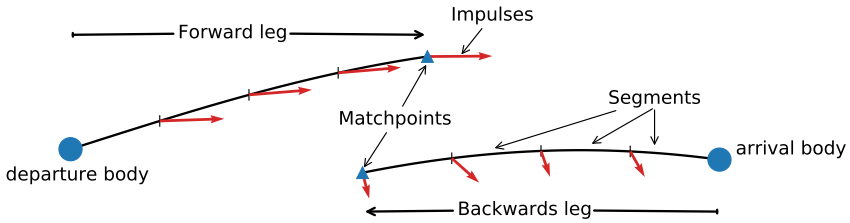


Fig. 1 A graphical depiction of a single bounded-impulse phase using the MGALTS transcription.

This work is primarily concerned with finding the most efficient way of performing the propagation across each leg. First, the propagation is formulated using an explicit stepping approach. To reduce overhead and improve computational efficiency within OpenMDAO, the authors next posed the leg propagation as an implicit algorithm. This implicit approach allows the algorithm to evaluate the trajectory simultaneously at all points, and this vectorization provides a significant improvement in performance. The generality of OpenMDAO allows the implicit formulation to be approached in two ways. In the first approach, a Newton solver is used to converge the state-time history of the spacecraft. While this approach capitalizes on vectorization, as far as the optimizer is concerned each leg is a fully-propagated trajectory and thus the optimization problem is the same as that in the explicitly propagated case. In the second approach, the optimizer is used to find the optimal trajectory using a true multiple-shooting approach. The following sections summarize each of these formulations. All approaches use the algorithm from Reference 3 to propagate the spacecraft state analytically across a single segment. The extended design structure matrix for this system is shown in Figure 2 [7].

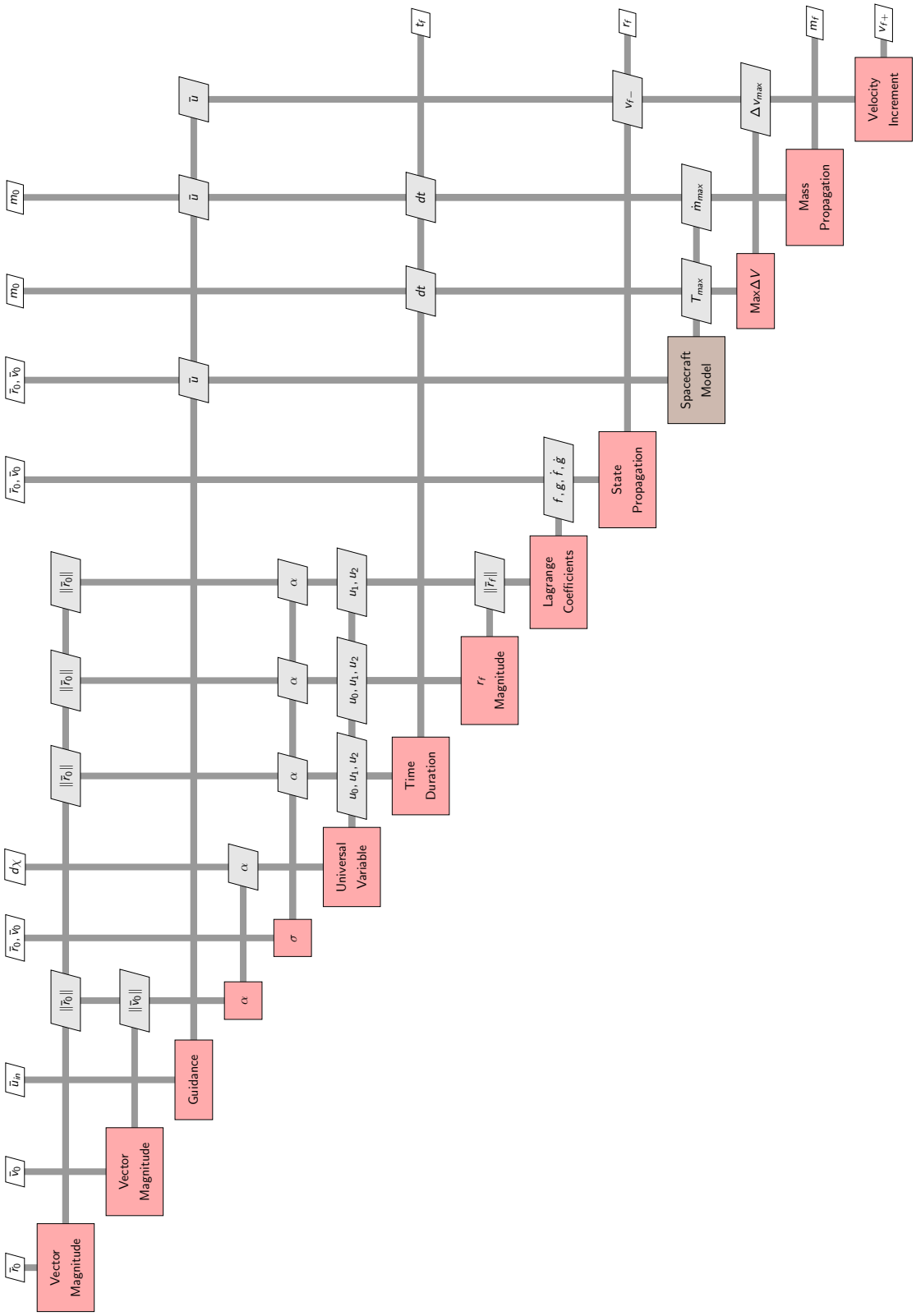


Fig. 2 Extended design structure matrix (XDSM) for a single MGALTS segment propagation, as described by Ellison, Englander, and Conway[3].

A. The explicit single-shooting formulation of the propagation of an MGALTS leg

An explicit propagation scheme is arguably the easiest to conceptualize. While time-stepping an integration across fixed time intervals leads to inaccuracy during periapsis passage of eccentric orbits, the use of Sundman anomaly as the independent variable alleviates this by automatically shortening the time duration of segments during periapsis passage.

To step across an entire leg of the trajectory, multiple segments are simply chained together in series. This is depicted in Figure 3.

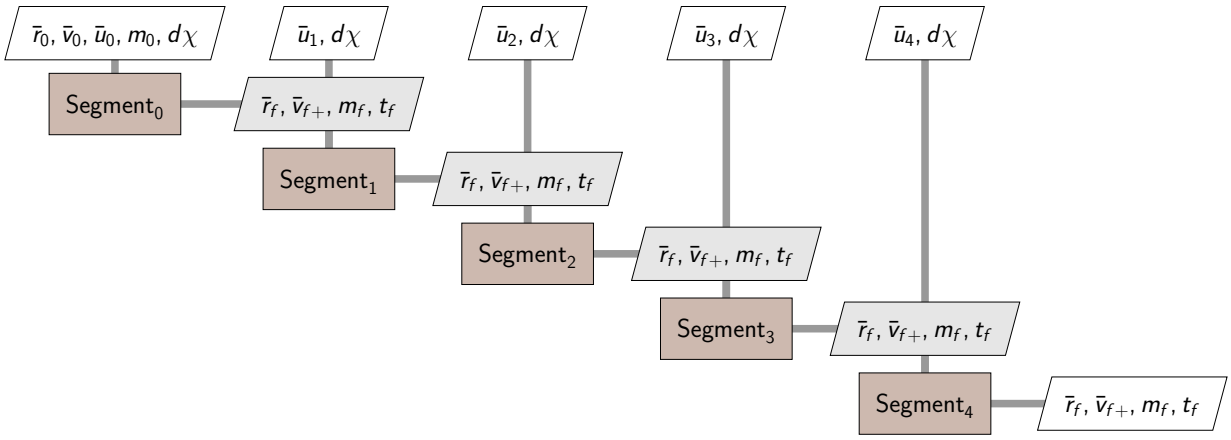


Fig. 3 Extended design structure matrix (XDSM) for an explicit MGALTS leg of five segments.

One advantage of this approach over implicit approaches is that a simple evaluation of the system always yields a valid trajectory. The method does not rely on the convergence of an underlying solver or optimizer to propagate the state time history.

Computing derivatives for the explicit propagation approach is a matter of chaining together partial derivatives from each evaluation of the system in Figure 2. This can be accomplished using either a forward chain rule or a reverse (adjoint) differentiation technique. However, since the number of computational systems in the model increases as the number of segments increases, this poses a computational hindrance. Not only must the algorithm loop over many more systems when propagating the state of the spacecraft, but the same is true when evaluating the derivatives for the optimizer.

B. A solver-driven single-shooting formulation for the propagation of a MGALTS leg

To overcome some of the computational deficiencies of chaining segments together explicitly, an implicit formulation in which a Newton-solver is used to converge the trajectory during each iteration of the optimizer is next posed. In this solver-driven formulation, the initial position, velocity, and mass at the start of leg (or half-phase, to use the nomenclature of Ellison, Englander, and Conway) are inputs, as they are for the explicit formulation. However, a vectorized form of the segment depicted in Figure 2 is now posed to simultaneously propagate the state across all segments in a leg. The entire trajectory state-history (position, velocity, and mass) is provided as implicit variables whose associated residuals are to be driven to zero by the solver. Starting with a guess of the initial position, velocity, and mass in each segment of a leg, the algorithm in Figure 4 is used to compute the final position, velocity, and mass across each segment.

The residuals for each state are the difference between the computed value of the state at the end of one segment and the (given) initial value of the state at the start of the subsequent segment, as calculated with Equations 1, 2, and 3.

$$\mathbf{R}(\mathbf{r}_{[1:]}) = \mathbf{r}_{[1:]} - \mathbf{r}_f \tag{1}$$

$$\mathbf{R}(\mathbf{v}_{[1:]}) = \mathbf{v}_{[1:]} - \mathbf{v}_f \tag{2}$$

$$\mathbf{R}(m[1:]) = m_{[1:]} - m_f \tag{3}$$

The residual associated with the first value of a state in each leg is the initial value of the state in that leg, as given by Equations 4, 5, and 6.

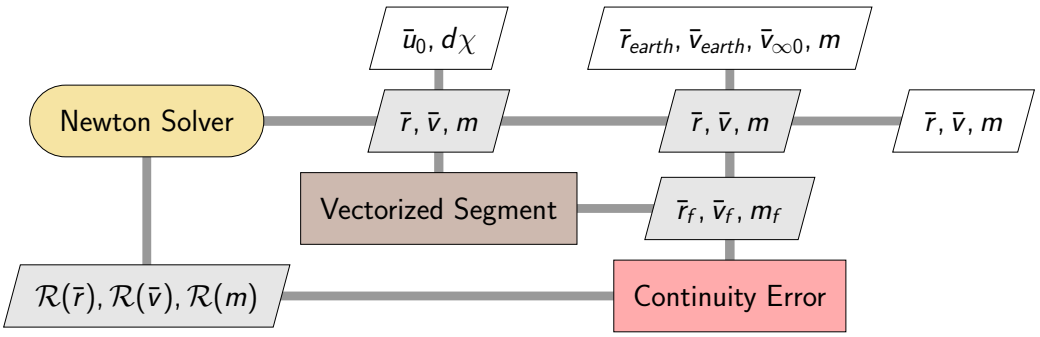


Fig. 4 Extended design structure matrix (XDSM) for a solver-driven implicit leg.

$$\mathbf{R}(\mathbf{r}[0,:]) = r[0,:] - \mathbf{r}_0 \tag{4}$$
$$\mathbf{R}(\mathbf{v}[0,:]) = v[0,:] - \mathbf{v}_0 \tag{5}$$
$$\mathbf{R}(m[0]) = m[0] - m_0 \tag{6}$$

In the case of position, the initial position of a leg is the position of the departure gravitational body (or arrival body for the backwards-propagated leg). Initial velocity is the velocity of the arrival or departure body, plus the departure/arrival excess velocity. Finally, the initial mass of a leg is an independent variable and is treated as a design variable. This is effectively a multiple-shooting approach, but the continuity between the segments is being enforced as residuals for the solver rather than as constraints for the optimizer. From the optimizer's perspective, it still sees the same optimization problem as that posed when using the explicit propagation approach.

One potential downside to this implicit scheme is that it relies on the convergence of a nonlinear solver to provide a valid trajectory. In practice, this issue can be mitigated by using a simple ballistic propagation of the initial conditions as the initial guess for the nonlinear solver if the residuals are too large.

This approach has several computational advantages over the explicit formulation. First, vectorization of the calculations provides a significant speedup in higher level languages such as Python, in which OpenMDAO is written. This reduces the time required to evaluate the outputs of the model. Additional gains are achieved by capitalizing on the way OpenMDAO computes total derivatives across a model. When chaining derivatives together, the derivatives would have to be chained through each iteration of the nonlinear solver. However, OpenMDAO uses a unique method for computing derivatives based on the Modular Approach to Unified Derivatives (MAUD), originally developed by developed by Martins and Hwang[4, 8].

Suppose the model consists of some combination of implicit and explicit calculations. When evaluating the model, the residuals associated with the implicit functions are driven to zero. Thus, when the values of some set of independent variables ($\mathbf{x}$) are changed, the implicit and explicit outputs of a system ($\mathbf{y}$) are changed such that the residuals through the entire system are zero.

$$R(\mathbf{x}, \mathbf{y}) = 0 \tag{7}$$
$$G(\mathbf{x}, \mathbf{y}) = g(x, \mathbf{y}) \tag{8}$$

In this instance, the implicit variables consist of the initial value of the states in each segment. The independent variables consists of the prescribed initial mass at the start of the leg, the change in Sundman anomaly over the leg, V_∞, and the control vector in each segment.

$$\mathbf{x} = [m_0^*, d\chi, V_{\infty 0}, V_{\infty f}, \mathbf{u}] \tag{9}$$
$$\mathbf{y} = [\mathbf{r}_0, \mathbf{v}_0, m_0] \tag{10}$$

Now consider the calculation of the total derivative Jacobian across an arbitrarily complex model. The total derivatives of the objective and constraints ($\mathbf{G}$) with respect to the design variables ($\mathbf{x}$) are:

$$\frac{d\mathbf{G}}{d\mathbf{x}} = \frac{\partial \mathbf{G}}{\partial \mathbf{x}} + \frac{\partial \mathbf{G}}{\partial \mathbf{y}} \frac{d\mathbf{y}}{d\mathbf{x}} \tag{11}$$

The partial derivatives on the right-hand-side of Equation 11 can be computed for a single computational *system* with finite differences, complex step, automatic differentiation, or user-provided analytic derivatives. However, the final term on the right-hand-side of Equation 11 accounts for the *total* change in the implicit outputs ($\mathbf{y}$) with respect to the design variables ($\mathbf{x}$), and is expensive to compute. This final term also appears in the total derivative of Equation 7. Since the solver always reduces the residuals to zero when the design variables (x) are changed, the total derivative of $\mathbf{R}$ with respect to $\mathbf{x}$ is zero.

$$\frac{d\mathbf{R}}{d\mathbf{x}} = \frac{\partial \mathbf{R}}{\partial \mathbf{x}} + \frac{\partial \mathbf{R}}{\partial \mathbf{y}} \frac{d\mathbf{y}}{d\mathbf{x}} = 0 \tag{12}$$

Rearranging Equation 12 provides a linear system which **must be solved once for each design variable** to compute the total derivative $\frac{d\mathbf{y}}{d\mathbf{x}}$.

$$\frac{d\mathbf{y}}{d\mathbf{x}} = -\left[\frac{\partial \mathbf{R}}{\partial \mathbf{y}}\right]^{-1} \frac{\partial \mathbf{R}}{\partial \mathbf{x}} \tag{13}$$

Once Equation 13 has been computed, the results can be plugged back into Equation 11 to compute the total derivatives across the model. This is the forward chain rule.

The reverse method of differentiation can be derived by combining Equation 13 and Equation 11 as follows:

$$\frac{d\mathbf{G}}{d\mathbf{x}} = \frac{\partial \mathbf{G}}{\partial \mathbf{x}} - \underbrace{\frac{\partial \mathbf{G}}{\partial \mathbf{y}} \left[\frac{\partial \mathbf{R}}{\partial \mathbf{y}}\right]^{-1}}_{\psi^T} \frac{\partial \mathbf{R}}{\partial \mathbf{x}} \tag{14}$$

$$\psi^T = -\left[\frac{\partial \mathbf{R}}{\partial \mathbf{y}}^T\right]^{-1} \frac{\partial \mathbf{G}}{\partial \mathbf{x}}^T \tag{15}$$

Now Equation 15 **must be solved once for each objective or constraint** to compute each element of ψ^T, the adjoint vector. Each solution to Equation 15 is plugged into Equation 14 to compute the total derivatives across the model. Since reverse differentiation requires one solution per objective or constraint, it is generally preferable when the number of constraints in a problem is small.

OpenMDAO handles the assembly of total derivatives automatically for the user, the benefits of accurate analytic derivatives are realized without getting bogged down by the complexity of implementing them. Other authors have cited this complexity as a reason for preferring approaches like automatic differentiation or complex-step[9]. While OpenMDAO allows these various techniques to be used for providing the partial derivatives of a particular computation, the use of MAUD to assemble the total derivatives from these partials can be significantly faster. This approach to computing derivatives is advantageous because it does not require a nonlinear solver to reconverge the system with each derivative evaluation, as would be the case with finite differencing. Since the only requirement to compute the partials with this method is that the model residuals are zero, this method also does not rely on the need to propagate derivatives through each iteration of the nonlinear solver, as would be the case with automatic differentiation approaches. Finally, complex-step is somewhat slower than a pure-analytic approach and fails when models involve non-analytic functions.

C. An optimizer-driven implicit formulation for the propagation of an MGALTS leg.

In developing the solver-driven implicit approach, the building blocks needed for a full-blown, multiple-shooting transcription of MGALTS were largely in place. Only a few modest modifications are necessary. Again, the segment propagation model (Figure 2) is vectorized. The residuals from Equations 1-3 are now posed as equality constraints

for the optimizer, and the optimizer is given the initial state in each segment as design variables. Since the initial state in each leg is now a design variable in the optimization, the initial position is forced to be coincident with the departure/arrival body via a nonlinear constraint instead of enforcing it as a residual as in Equation 4. Likewise, the velocity at the start of the first segment in a leg is forced to be compatible with the planet velocity and V_∞ via nonlinear constraint. Finally, if the mass at the start of the leg is fixed, this too must be implemented as a nonlinear constraint in this implementation.

It is debatable whether the matchpoint joining the forward-shooting and backwards-shooting legs of the trajectory is necessary. After all, each segment is now effectively a propagation connected to its adjacent legs by matchpoint constraints. For the purpose of this work, the forward-shooting/backward-shooting paradigm has been maintained to keep as much commonality with the single-shooting approaches as possible. The implementation of a single leg (or half-phase) of the optimizer-driven multiple-shooting approach is shown in Figure 5.

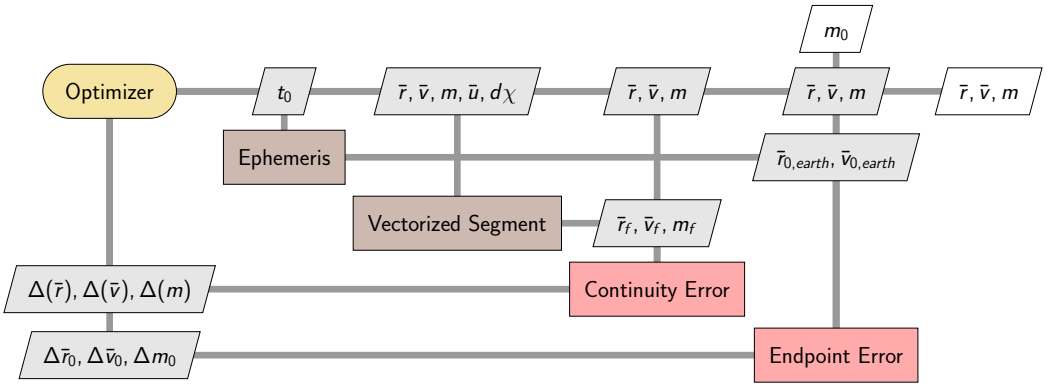


Fig. 5 Extended design structure matrix (XDSM) for an optimizer-driven implicit leg. Note that this diagram only captures the behavior of a single leg of the bounded-impulse model for the sake of simplicity.

It might seem counter-intuitive to improve performance by posing a problem with far more variables and constraints. In general, though, multiple-shooting approaches have a few key benefits over single-shooting methods. First, each individual segment is a valid propagation, but the entire trajectory is not physically realizable until the constraints which govern state continuity at segment bounds have been satisfied. This means the optimizer has more freedom to move through the infeasible space to find an optimal trajectory. Secondly, the state continuity constraints at segment bounds are only directly impacted by input variables pertaining to those adjacent segments. This means that, while the Jacobian matrix has far more rows and columns, the vast majority of entries will be zero. This *sparsity* of the Jacobian can be leveraged to dramatically reduce the time required to compute total derivatives across the model. OpenMDAO can automatically detect this sparsity pattern and use it to simultaneously compute multiple rows or columns of the Jacobian using forward or reverse (adjoint) differentiation, or even both. OpenMDAO's use of graph-coloring algorithms to efficiently compute derivatives is discussed in more detail in Reference 4.

Like the solver-driven implicit single-shooting form, the optimizer-based approach requires a reasonably accurate initial guess for the states throughout the leg. This can be achieved by starting with a Lambert-method to obtain the velocity at the beginning of a given leg, and then using the explicit form of MGALTS to propagate the states through each segment in the leg.

IV. Results

To assess the performance of varying implementations of the state propagation through a leg of the MGALTS algorithm, we apply these three approaches to a simple Earth-Mars transfer; reference mission 1 from a series of low-thrust trajectory optimization examples by Polsgrove et al[10]. The simplicity of this reference mission, showin in Figure 6 is due to a few factors. First, the launch date, arrival date, and departure mass are fixed. The departure V_∞ vector is a design variable but its magnitude is constrained. The second factor which simplifies this case is that the available power never drops below the maximum power of the thruster; thus it can be run with a constant power, constant I_{sp} propulsion model. Reference 10 provides full details for the case, including the solar array and thruster models used.

This simple problem is used to demonstrate the comparative cost of the methods developed in this paper: explicitly

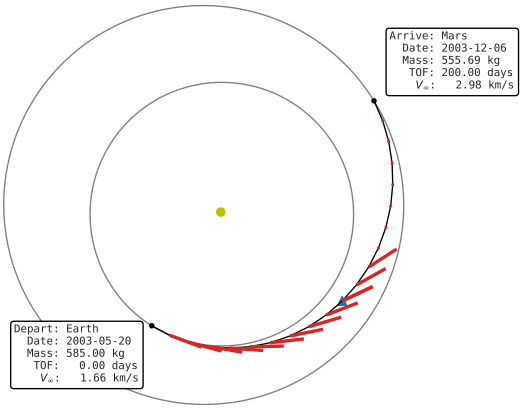


Fig. 6 Solution diagram for reference mission 1 from Reference 10.

propagated shooting, implicit solver-driven shooting, and optimizer-driven multiple-shooting. Table 1 provides the NLP transcription for the problem. Again, the explicitly propagated form and the solver-driven implicit form, which will collectively be referred to as the single-shooting methods, both pose the same problem for the optimizer. Any differences in performance are typically due to the ability to vectorize the dynamics. The size of the problem is shown as a function of the number of segments n. The optimizer-driven multiple-shooting approach has several times as many variables and constraints as the single-shooting method.

Table 1 Earth-Mars flyby problem NLP formulation for n segments using different transcriptions.

(a) Single-shooting

Objective	Maximize m_f	1
Design variables	$10kg \leq m_f \leq 600kg$	1
	$50^o \leq \Delta\chi \leq \infty$	1
	$-\infty \leq \mathbf{v}_{\infty 0} \leq \infty$	3
	$-\infty \leq \mathbf{v}_{\infty f} \leq \infty$	3
	$[-1,-1,-1] \leq \mathbf{u}_i \leq [1,1,1]$	$3n$
	Total design variables	**3n + 8**
Constraints	$\Delta\mathbf{r}_{mid} = 0$	3
	$\Delta\mathbf{v}_{mid} = 0$	3
	$\Delta t_{mid} = 0$	1
	$\Delta m_{mid} = 0$	1
	$C3_0 \leq 1.66^2$	1
	$\|\mathbf{u}_i\| \leq 1$	n
	Total constraints	**n + 9**

(b) Multiple-shooting

Objective	Maximize m_f	1
Design Variables	$10kg \leq \mathbf{m}_0 \leq 600kg$	n
	$-\infty \leq \mathbf{r}_0 \leq \infty$	$3n$
	$-\infty \leq \mathbf{v}_0 \leq \infty$	$3n$
	$50^o \leq \Delta\chi \leq \infty$	1
	$-\infty \leq \mathbf{v}_{\infty 0} \leq \infty$	3
	$-\infty \leq \mathbf{v}_{\infty f} \leq \infty$	3
	$[-1,-1,-1] \leq \mathbf{u}_i \leq [1,1,1]$	$3n$
	Total design variables	**10n + 7**
Constraints	$\Delta\mathbf{r}_{mid} = 0$	3
	$\Delta\mathbf{v}_{mid} = 0$	3
	$\Delta t_{mid} = 0$	1
	$\Delta m_{mid} = 0$	1
	$\mathbf{r}_0 = \mathbf{r}_{earth}$	3
	$\mathbf{v}_0 = \mathbf{v}_{earth} + \mathbf{v}_{\infty 0}$	3
	$\mathbf{r}_f = \mathbf{r}_{mars}$	3
	$\mathbf{v}_f = \mathbf{v}_{mars} - \mathbf{v}_{\infty f}$	3
	$\mathbf{r}_0[1:] - \mathbf{r}_f[:-1] = 0$	$3n-6$
	$\mathbf{v}_0[1:] - \mathbf{v}_f[:-1] = 0$	$3n-6$
	$m_0[1:] - m_f[:-1] = 0$	$n-2$
	$C3_0 \leq 1.66^2$	1
	$m_0[0] = 585kg$	1
	$\|\mathbf{u}_i\| \leq 1$	n
	Total constraints	**8n + 8**

This problem was solved with varying numbers of segments, from ten to one-hundred. Figure 7 shows the average cost of objective and derivative evaluations for the three formulations. In both plots, the time required to evaluate the objective or derivatives is normalized by dividing by the time required to compute them for the fastest case - multiple shooting. The cost of computing both the objective and derivatives for the explicitly propagated legs increases significantly as the number of segments is increased. While the cost of evaluating the objective implicit formulations are both insensitive to the number of segments, the use of a Newton solver to converge the residuals in the solver-driven case does add some expense relative to the optimizer-driven method. There is a slightly upward trend in compute time for the solver-driven case due to the increased size of the vectors involved.

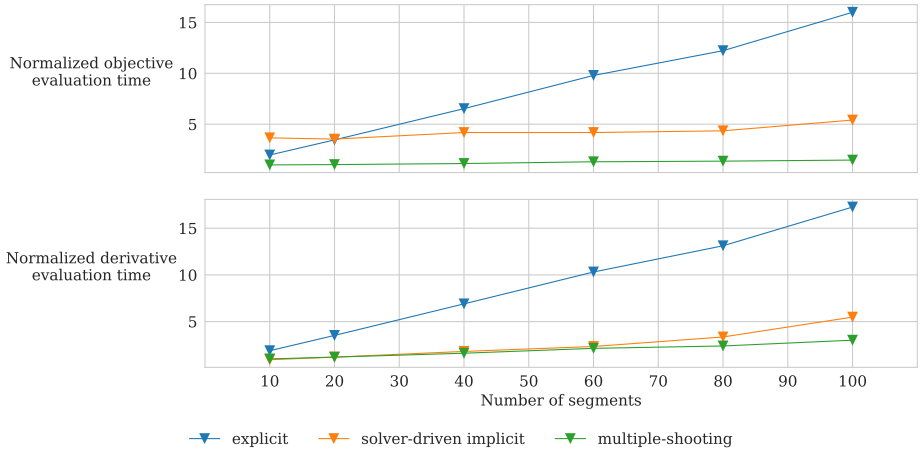


Fig. 7 Performance results showing the cost of the objective and derivative evaluations vs. the number of segments used in the transcription. Times are relative to the cost of the objective and derivative evaluation for the 10-segment multiple-shooting case.

If we look at the normalized total-time required to optimize the problem (Figure 8), we see the cost of the MGALTS solution with explicitly propagagated segments exponentially increasing with time. The iteration count profile is the same for both the explicit and solver-driven implicit formations. This is to be expected since both formulations are presenting the same form of the problem to the optimizer. Note that the multiple-shooting case had some difficulty in converging the 60-segment case. Despite using far more iterations than the explicit formulation it is still nearly twice as fast.

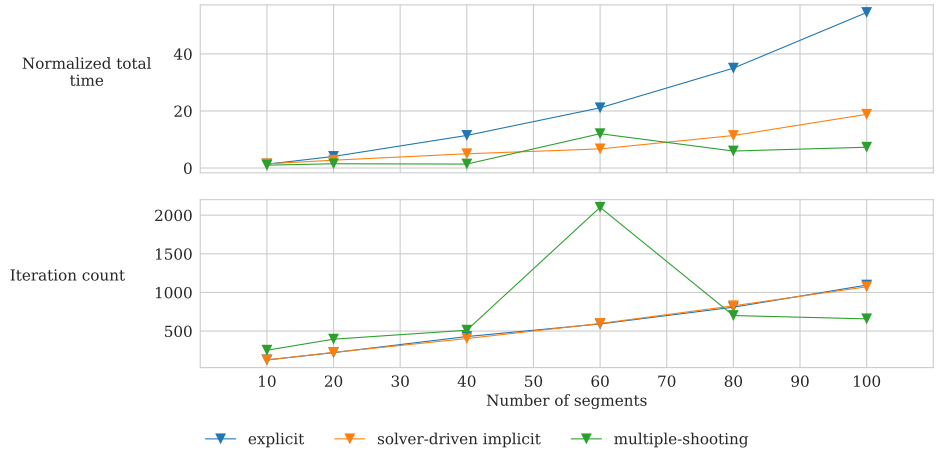
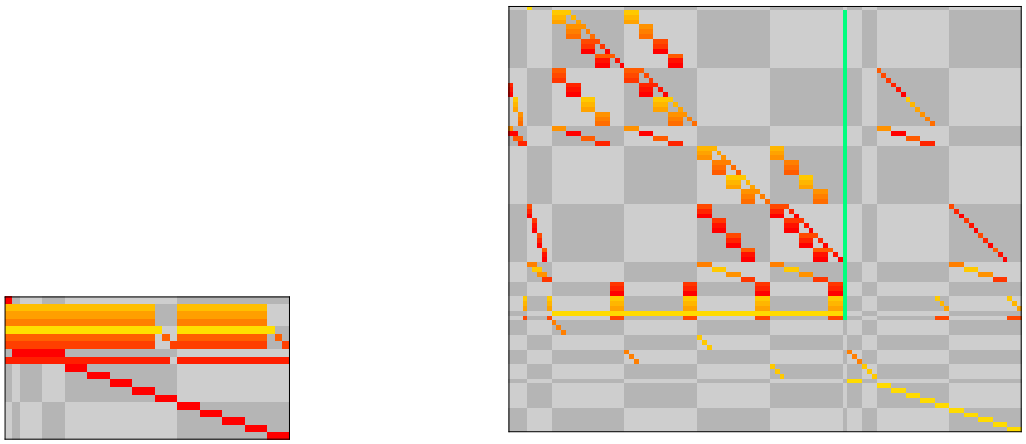


Fig. 8 Performance results showing the normalized total time and iterations required for the optimization vs. the number of segments used in the transcription. Times are relative to the cost of the 10-segment multiple-shooting case.

Both the solver-driven and optimizer-driven implicit approaches to propagation show good performance in computing derivatives. To understand how the cost of the derivative evaluation is kept so low, consider the *sparsity* of the total constraint Jacobian. Figure 9a shows the total constraint Jacobian for the single-shooting approach and the multiple-shooting with ten total segments. For the single-shooting case, the derivatives of the matchpoint constraints form a series of dense rows in the constraint Jacobian. Since the states and time are both propagated forward/backward across some change in Sundman anomaly, these quantities at the matchpoint are dependent on the entirety of the state time-history. The impulse magnitude constraints are block diagonal, since the magnitude constraint in a given segment is only a function of the input control vector in the same segment. While these add a substantial number of constraints, OpenMDAO uses a graph-coloring algorithm to determine which elements of the Jacobian matrix can be computed simultaneously[4]. Due to their block-diagonal nature, the entire bottom half of the single-shooting Jacobian matrix can be computed simultaneously using reverse (adjoint) differentiation.

Figure 9b shows the constraint Jacobian for the multiple-shooting formulation. These matrices are significantly larger than their single-shooting counterparts but also significantly more sparse. The structure of their sparsity pattern allows many elements of the Jacobian to be solved simultaneously. The number of times Equations 13 or 15 must be solved is independent of the number of segments used, even in the presence of path constraints. This last fact makes the multiple-shooting approach superior to the single-shooting method in this regard, since the number of solutions required in the presence of general path constraints will increase for the single-shooting methods.



Size: 19 x 38 Sparsity: 37.81% Solves required: 8

(a) Single-shooting

Size: 88 x 106 Sparsity: 8.02% Solves required: 15

(b) Multiple-shooting

Fig. 9 Graphic form of the total-derivative Jacobian for single-shooting and multiple-shooting formulations. Elements with the same color are able to be computed simultaneously. Each figure also includes the total Jacobian size, the percentage of nonzeros in the sparsity pattern, and the number of solutions to Equations 13 or 15 required to compute the total derivatives.

It is important to note the presence of the dense row and the dense column in the multiple-shooting constraint matrices. The dense row, shown in yellow, is due to the matchpoint time continuity constraint. This constraint is dependent upon the entire time history of the position and velocity in both the forward and backwards-propagated leg. The dense column, shown in green, is due to the dependence of all matchpoint constraints and segment continuity constraints on the given total change in Sundman anomaly. Typically, the presence of both a dense row and a dense column would require the entire upper-left block of the Jacobian to be computed one variable at a time (in forward mode) or one constraint at a time (in reverse mode). However, OpenMDAO's bidirectional derivative capability allows it to compute these derivatives much more efficiently. Most elements of the constraint Jacobian are computed simultaneously in reverse mode, while the elements of the green column are computed in a single forward-mode derivative evaluation. Without this bidirectional derivative capability, the computational efficiency gained with this multiple shooting formulation would not be achievable.

V. Conclusions

In this work ways were demonstrated to increase the computational efficiency of a bounded-impulse low-thrust trajectory optimization approach through the use of implicit formulations. Using the Multiple Gravity Assist Low-Thrust Sundman (MGALTS) algorithm as the basis for the transcriptions, the equivalent trajectory optimization problem was constructed in three ways. A naive, purely explicit single-shooting approach in OpenMDAO increases the complexity of the model as the number of segments used in the MGALTS transcription is increased. An implicit single-shooting approach in which the time-history of the trajectory is controlled by an underlying Newton solver shows considerable improvement over the purely explicit approach. Due to OpenMDAO's unique approach to derivative calculation, the cost of derivative calculations is nearly constant with increasing segment counts under this approach - although there is some increase due to the increase size of the nonlinear system. Finally, a purely optimizer-driven multiple-shooting approach to propagating an MGALTS leg was examined. Under this approach, all of the segment continuity constraints are handled by the optimizer. By leveraging OpenMDAO's bidirectional, sparse differentiation capability, the computational effort needed to compute derivatives under this approach is almost independent of the number of segments involved. The lack of an iterative solver in the underlying model kept the cost of objective function calculations essentially constant when the segment count is increased.

References

[1] Sims, J., and Flanagan, S., "Preliminary Design of Low-Thrust Interplanetary Missions," *AAS Astrodynamics Specialist Conference, Girdwood, Alaska*, 1999. AAS 99-338.

[2] Sims, J., Finlayson, P., Rinderle, E., Vavrina, M., and Kowalkowski, T., "Implementation of a low-thrust trajectory optimization algorithm for preliminary design," *AIAA/AAS Astrodynamics specialist conference and exhibit*, 2006, p. 6746.

[3] Ellison, D. H., Englander, J. A., and Conway, B. A., "A Time-Regularized Multiple Gravity-Assist Low-Thrust Bounded-Impulse Model for Trajectory Optimization," *27th AAS/AIAA Space Flight Mechanics Meeting, San Antonio, Texas*, 2017. AAS 17-429.

[4] Gray, J. S., Hwang, J. T., Martins, J. R., Moore, K. T., and Naylor, B. A., "OpenMDAO: An open-source framework for multidisciplinary design, analysis, and optimization," *Structural and Multidisciplinary Optimization*, 2019, pp. 1–30.

[5] Ellison, D., Conway, B., Englander, J., and Ozimek, M., "Analytic Gradient Computation for Bounded-Impulse Trajectory Models Using Two-Sided Shooting," *Journal of Guidance, Control, and Dynamics*, Vol. 41, 2018, pp. 1–14. doi:10.2514/1.G003077.

[6] Ellison, D., Conway, B., Englander, J., and Ozimek, M., "Application and Analysis of Bounded-Impulse Trajectory Models with Analytic Gradients," *Journal of Guidance, Control, and Dynamics*, Vol. 41, 2018, pp. 1–15. doi:10.2514/1.G003078.

[7] Lambe, A. B., and Martins, J. R. R. A., "Extensions to the Design Structure Matrix for the Description of Multidisciplinary Design, Analysis, and Optimization Processes," *Structural and Multidisciplinary Optimization*, Vol. 46, 2012, pp. 273–284. doi:10.1007/s00158-012-0763-y.

[8] Martins, J. R., and Hwang, J. T., "Review and unification of methods for computing derivatives of multidisciplinary computational models," *AIAA Journal*, Vol. 51, No. 11, 2013, pp. 2582–2599.

[9] Pellegrini, E., and Russell, R. P., "On the computation and accuracy of trajectory state transition matrices," *Journal of Guidance, Control, and Dynamics*, 2016, pp. 2485–2499.

[10] Polsgrove, T., Kos, L., Hopkins, R., and Crane, T., "Comparison of Performance Predictions for New Low-Thrust Trajectory Tools," *AIAA/AAS Astrodynamics Specialist Conference and Exhibit*, 2006. doi:10.2514/6.2006-6742, URL `https://arc.aiaa.org/doi/abs/10.2514/6.2006-6742`, aIAA 2006-6742.