Higher Order Sensitivities for Solving Nonlinear Two-Point Boundary-Value Problems

D. Todd Griffith
Texas A&M University, College Station, Texas 77843-3141

James D. Turner
Amdyn Systems, White GA 30184

S. R. Vadali and John. L. Junkins
Texas A&M University, College Station, Texas 77843-3141

In this paper, we consider new computational approaches for solving nonlinear Two-Point Boundary-Value Problems. The sensitivity calculations required in the solution utilize the automatic differentiation tool OCEA (Object Oriented Coordinate Embedding Method). OCEA has broad potential in this area and many other areas since the partial derivative calculations required for solving these problems are automatically computed and evaluated freeing the analyst from deriving and coding them. In this paper, we demonstrate solving nonlinear Two-Point Boundary Value Problems by shooting and direct methods using automatic differentiation. We demonstrate standard first-order algorithms and higher-order extensions. Additionally, automatic generation of co-state differential equations and second- and higher-order midcourse corrections are considered. Optimization of a sample Low-thrust, Mars-Earth trajectory is considered as an example. Computational issues related to domain of convergence and rate of convergence will be detailed.

I. Introduction

Applications which require the solution to a Two-Point Boundary-Value Problem (TPBVP) occur in many engineering disciplines. The solution of the TPBVP determines states or functions of the states at two points, usually the initial and final times, which satisfy the boundary conditions for a given mathematical problem. For linear problems, analytical solutions exist for determining the unknown initial or final conditions; however, for nonlinear problems we must resort to iterative methods. Many methods have been developed to solve nonlinear TPBVPs dealing with a wide range of issues, including stiffness of differential equations of motion, reduction in sensitivity, and speed of convergence.

One particular solution is obtained by the method of differential corrections. Here a guess is made for the unknown states, the state differential equations are integrated until the final time, and the boundary conditions are checked to see if they are satisfied. When the constraint conditions are not satisfied, corrections to the unknown states are computed and the process is repeated until convergence criteria are met. Typically, these corrections are computed by using first order sensitivity calculations or at best, second order calculations using an approximate Hessian.

In this paper, we study the automatic generation of sensitivity equations for solving nonlinear TPBVPs. The Fortran 90 extension OCEA (Object Oriented Coordinate Embedding Method) will be used to perform the sensitivity calculations. OCEA has been used to solve problems in optimization, estimation, and generation of equations of motion. OCEA has broad potential in many areas since it is extremely useful for computing partial derivatives of scalar, vector, matrix, and higher order tensor functions.

Optimization of a sample Low-thrust, Mars-Earth trajectory will be considered as one example. Direct optimization using Differential Inclusions will be considered in a second example demonstrating OCEA’s
capabilities. First through fourth order sensitivity calculations will be considered in each case. A number of results detailing performance will be presented.

II. Overview of OCEA

This approach uses the previously developed OCEA (Object Oriented Coordinate Embedding Method) extension for FORTRAN90 (F90).\textsuperscript{3,4} The OCEA package is an object-oriented automatic differentiation equation manipulation package. Computer implementation of differentiation is typically accomplished by two distinct approaches – automatic differentiation and symbolic differentiation.\textsuperscript{8} The primary distinction of these approaches is that automatic differentiation invokes the chain automatically and takes place in the background. This allows for time and memory optimized differentiation. On the other hand, symbolic differentiation requires more user intervention. Typically, derivatives are computed using a symbolic differentiation program, such as Macsyma or Mathcad, and the derivatives then need to be either hand typed in a computer program or saved in a file that typically requires editing and compilation. There is an advantage here in that the computed derivative expressions can be viewed by the analyst; however, the resulting mathematical expressions can be so large that no physical insight can be gained from the effort.

OCEA defines \textit{embedded variables} that represent abstract data types, where hidden dimensions (background arrays) are used for storing and manipulating partial derivative calculations. OCEA replaces each scalar variable in the problem with a differential n-tuple consisting of the following variables (second-order OCEA method):

\[
\begin{bmatrix}
 f \\
 \nabla f \\
 \nabla^2 f
\end{bmatrix}
\]  

(1)

where \( \nabla \) and \( \nabla^2 \) denote symmetric first- and second-order gradient tensors with respect to a user-defined set of independent variables. The introduction of the abstract differential n-tuple allows the computer to continue to manipulate each scalar variable as a conventional scalar variable, even though the first and higher-order partial derivative are attached to the scalar variable in a hidden way. The individual objects are extracted, using OCEA's adopted notation, as follows: \( f = f \% E \), \( \nabla f = f \% V \), and \( \nabla^2 f = f \% T \). The automatic computation of the partial derivatives is achieved by operator-overloading methodologies that redefine the intrinsic mathematical operators and functions using the rules of calculus. For example, addition and multiplication are redefined as follows.

\[
(a + b) := \begin{bmatrix}
 a + b \\
 \nabla a + \nabla b \\
 \nabla^2 a + \nabla^2 b
\end{bmatrix}
\]  

(2)

\[
(a \ast b) := \begin{bmatrix}
 a \ast b \\
 \partial_i (a \ast b) \\
 \partial_j \partial_i (a \ast b)
\end{bmatrix}
\]  

(3)

Thus the “+” and “\ast” operators are overloaded so that coding the left side expressions of Eqs. (2) and (3) causes all the right side computations to be carried out. More subtly, if \( z_1 = a + b \) and \( z_2 = a \ast b \), then computing \( z_3 = z_1 + z_2 \) causes the results of Eqs. (2) and (3) to be propagated efficiently in the background to compute:

\[
\begin{bmatrix}
 a + b + a \ast b \\
 \partial_i (z_3) \\
 \partial_j \partial_i (z_3)
\end{bmatrix}
\]  

(4)

Additional operations for the standard mathematical library functions, such as trigonometric and exponential functions, are redefined to account for the known rules of differentiation. In essence, this approach pre-codes, once and for all, all of the partial derivatives required for any problem, and the chain rule is implemented automatically in background operations that the user neither derives nor codes. At compile time, and without user intervention, the OCEA-based approach links the subroutines and functions required for evaluating the partial derivative models.

III. The Optimal Control Formulation

In this section, we overview the formulation of the necessary conditions for the Optimal Control problem. One of the problems we consider in this paper is a minimum time problem. Therefore, we also examine converting the minimum time problem to one in which the final time is fixed, and the unknown final time becomes a free parameter to be optimized.
A. General Formulation

Consider the minimization of the following function where \( t_f \) is the final time:

\[
J = \int_{0}^{t_f} L(x(t), u(t), t) dt
\]

subject to the following dynamics:

\[
H = L(x(t), u(t), t) + \lambda^T f(x(t), u(t), t)
\]

In general, the Hamiltonian for the system is written as

\[
H = L(x(t), u(t), t) + \lambda^T f(x(t), u(t), t)
\]

For unconstrained control, the well-known necessary conditions along the trajectory required to minimize \( J \) are given by Eqs. (8-10)

\[
\dot{x} = \frac{\partial H}{\partial \lambda}
\]

\[
\dot{\lambda} = -\frac{\partial H}{\partial x}
\]

\[
\frac{\partial H}{\partial u} = 0
\]

Furthermore, we consider the initial and final boundary conditions on the states and co-states. The necessary boundary condition for an unspecified initial state, \( x_i(t_0) \), are \( \lambda_i(t_0) = 0 \), whereas initial co-states are unknown and must be solved numerically when the corresponding initial state is specified. The problems considered in this paper are of the type in which the initial states are known and the desired terminal states are specified. Therefore, the objective of one of the numerical solutions, the Low-thrust Mars-Earth trajectory, is to determine the unknown initial co-states. Also, when the final time is free we have \( H(t_f) = 0 \) as an additional necessary condition.

B. Converting to Fixed Final Time Free Parameter Type Problem

In this section, we show how a free final time problem can be converted to fixed final time problem with \( t_f \) included in the formulation as a free parameter. A fixed final time problem has the obvious advantage of fixed time limits for integrating any differential equations. The numerical algorithm for solving the TPBVP, which is outlined in subsequent sections, benefits since the unknown final time parameter appears explicitly in the state and co-state equations and ultimately appears explicitly in the sensitivity calculations.

Consider a new time variable, \( \tau \), defined as

\[
\tau = \frac{t}{t_f}; \quad 0 \leq \tau \leq 1
\]

Differentiation of the states with respect to the new time variable (denoted with a prime) can be written as
\[
x' = \frac{dx}{d\tau} = \frac{dx}{dt} \frac{dt}{d\tau} = t_f \dot{x}
\]

Therefore, in producing the new state equations for the fixed time problem we simply multiply Eq. (6) by the parameter \( t_f \). Additionally, new co-state equations for the fixed time problem are computed in the same manner.

The cost function is rewritten as \( J = \int_0^1 t, d\tau \), and the Hamiltonian is defined as

\[
H = t_f + t_f \dot{x}^T f(x(t), u(t), t) = t_f \left[ 1 + \dot{x}^T f(x(t), u(t), t) \right]
\]

Equations (8-10) are applied to the Hamiltonian in Eq. (13) to produce the fixed time problem necessary conditions along the trajectory. However, here we must consider one additional necessary condition on the free parameter \( t_f \):

\[
\int_0^1 \frac{\partial H}{\partial t_f} d\tau = 0
\]

In the next section, we describe the numerical procedures for solving the TPBVP.

**IV. Development of Numerical Algorithms**

In this section, we present the first- through fourth-order extension of the update equation required in the numerical solution. Additionally, we overview two particular methods for solving nonlinear TPBVP using OCEA. These methods include the method of differential corrections and solution by Differential Inclusions. Many approaches have been developed to solve TPBVPs. Here we show that the automatic differentiation capability of OCEA is very efficient in solving TPBVPs by these methods, solving optimization problems in general, and extending the methods to higher-order.

**A. Update Equation: Reversion of Series Solution**

In this section, we summarize the reversion of series solution for the update equation. More details can be found in Reference 4. This solution provides the correction to be applied to the current guess of the unknown parameters in the iterative numerical procedure. The update equation is given by

\[
x \approx x_{\text{guess}} + \delta = x_{\text{guess}} - \frac{dx}{ds_{\text{g}}} + \frac{1}{2!} \frac{d^2 x}{ds^2_{\text{g}}} - \frac{1}{3!} \frac{d^3 x}{ds^3_{\text{g}}} + \frac{1}{4!} \frac{d^4 x}{ds^4_{\text{g}}}
\]

where

\[
\frac{dx}{ds}_{\text{g}} = -\left( \nabla G \right)^{-1} h(x_{\text{guess}})
\]

\[
\frac{d^2 x}{ds^2}_{\text{g}} = -\left( \nabla G \right)^{-1} \nabla^2 G \cdot \frac{dx}{ds}_{\text{g}} \cdot \frac{dx}{ds}_{\text{g}}
\]
\[
\frac{d^3 x}{ds^3} \bigg|_g = -\left(\nabla G\right)^{-1} \left\{ \nabla^3 G \cdot \frac{dx}{ds} \bigg|_g \cdot \frac{dx}{ds} \bigg|_g \cdot \frac{dx}{ds} \bigg|_g + 2\nabla^2 G \cdot \frac{d^2 x}{ds^2} \bigg|_g \cdot \frac{dx}{ds} \bigg|_g \right\} + \\
\left\{ \nabla^2 G \cdot \frac{dx}{ds} \bigg|_g \cdot \frac{d^2 x}{ds^2} \bigg|_g \right\}
\]

where \(h(x_{\text{guess}})\) is the error in the terminal boundary states and \((\nabla G, \nabla^2 G, \nabla^3 G, \text{ and } \nabla^4 G)\) are symbolically the first- through fourth-order sensitivities.

In general, the solution given in Eqs. (15-19) provides a means for solving iterative problems in optimization and estimation. When only considering the first-order term in Eq. (15), we have the multilinear version of Newton’s Method. Equation (15) can therefore be considered as the higher-order extension of Newton’s Method. The gradient terms \((\nabla G, \nabla^2 G, \nabla^3 G, \text{ and } \nabla^4 G)\) are understood to be taken with respect to the unknown parameters to be estimated. The correction term in Eq. (15) can be accomplished once the desired order of terms coming from Eqs. (16-19) are computed.

B. The Method of Differential Corrections

The method of differential corrections is a shooting method. The numerical process entails guessing the unknowns initial conditions, integrating the state and co-state equations along with the state transition matrix differential equations until the final time is reached, and then evaluating how well the terminal boundary conditions are met. If the terminal conditions are satisfied to some acceptable error, then we accept the estimate for the unknown initial conditions as the solution. Otherwise, we utilize sensitivity calculations (Eqs. 16-19) to update the guess for the unknowns at the initial time and repeat the process until the terminal conditions are satisfied.

When the objective is to reach a specified set of terminal states, the sensitivity matrices required to solve the problem by the method of differential corrections using Eq. (15) are simply the state transition matrices that relate the initial and terminal states. For illustration we show that, to first order, the relationship is written as

\[
\delta x(t_f) = \Phi(t_f, t_0) \delta x(t_0); \quad \Phi(t_f, t_0) = \frac{\partial \xi(t_f)}{\partial \xi(t_0)}
\]

where \(\delta x(t_0)\) and \(\delta x(t_f)\) represent small state departures from the optimal solution and \(\Phi(t_f, t_0)\) is the first-order state transition matrix that relates the departure state at the two times. Considering the procedure outlined above, it
can be seen that Eq. (20) can be solved for $\delta \dot{x}(t_0)$ in order to update the estimate for the initial states. This solution corresponds to the first-order term given in Eq. (16).

We can easily extend the procedure to include second- and higher-order terms. For example, considering up to second-order terms in Eq. (15) we have as an update equation

$$\delta \dot{x}(t_0) = \Phi(t_f, t_0)^{-1} \delta \dot{x}(t_f) - \frac{1}{2} \Phi(t_f, t_0)^{-1} \left\{ \Phi(t_f, t_0)^{-1} \delta \dot{x}(t_f) \Phi(t_f, t_0)^{-1} \delta \dot{x}(t_f) \right\}$$ (21)

where “·” represents a dot product operation and the second-order state transition matrix is defined as

$$\Phi(t_f, t_0, t_0) = \frac{\delta^2 \dot{x}(t_f)}{\delta \dot{x}^2(t_0)}.$$  

The first- through fourth-order state transition matrix algorithms have been developed in Reference 5. Appendix A contains the first- through fourth-order state transition matrix differential equations. Note in the Appendix that automatic differentiation enables the analyst to completely avoid the most time-consuming and error prone task involved in solving the state transition matrix differential equations since OCEA computes and evaluates the partial derivatives of the function $\Phi(x(t), t)$ automatically. The benefits are readily seen for the first-order case. Extension to higher-order is easily accomplished since the partial derivatives are automatically computed and evaluated.

C. Differential Inclusions

The method of Differential Inclusions \(^7\) is a direct optimization approach for generating finite-dimensional approximations to the solutions of optimal control problems. This method is used for rapid trajectory generation, and has the advantage that controls can be eliminated from the problem formulation. The differential equations are integrated implicitly resulting in a parameter optimization problem. There is no need to derive co-state and state transition matrix differential equations. Additionally, this method provides a means for approximating unknown parameters which can improve the starting guess for another solution effort, such as the method of differential corrections.

In this paper, we consider solving problems using Differential Inclusions of the following type

$$\dot{x} = f(x(t), t) + u(t)$$ (22)

$$J = \frac{1}{2} \int_0^T \left( x^T Q x + u^T R u \right) dt$$ (23)

where the final time is fixed.

The procedure entails discretizing the equations of motion. For the \(i^{\text{th}}\) equation of Eq. (22) we have

$$\frac{x_{k+1}^{(i)} - x_k^{(i)}}{\Delta^{(i)}} - f \left( \frac{x_k^{(i)} + x_{k+1}^{(i)}}{2} \right) = u_k^{(i)}$$ (24)

where $\Delta^{(i)} = (t_f-t_0)/N^{(i)}$, and $N^{(i)}$ is the number equally spaced time intervals chosen. The performance index of Eq. (23) is also rewritten as

$$J = \frac{1}{2} \sum_{i=1}^{m} \sum_{j=1}^{m} \left( \sum_{k=1}^{N^{(i)}} q^{(i,j)} x_k^{(i)} x_k^{(j)} + r^{(i,j)} u_k^{(i)} u_k^{(j)} \right)$$ (25)

American Institute of Aeronautics and Astronautics
where \( m \) is the number of states \( (x \in \mathbb{R}^m) \).

The expression on the left-hand side of Eq. (24) can be substituted into the performance index of Eq. (25) which eliminates the control from the formulation. Therefore, the unknowns to be solved for are the discretized state coordinates defined as \( x_k(t) = x(i)(k\Delta^i) \). This can be accomplished by minimization of the nonlinear function that results from substituting the control into Eq. (25).

The capability of OCEA to automatically compute the partial derivatives \( J \) makes this problem readily solvable. The gradient of \( J \) produces the necessary conditions for a minimum. The Hessian of \( J \) provides the required sensitivity matrix used to iteratively solve for the roots of the nonlinear equations defined by \( \nabla J = 0 \). These roots are the solution for the unknown discretized states. OCEA also enables higher-order optimization methods to be employed here by utilizing higher-order gradients of \( J \).

Given the solution for the states, we can substitute them into Eq. (24) and solve for the control. With the control and state time histories, we can then compute the initial co-states for the Differential Inclusions solution. These can be used as an improved starting guess for another solution method.

V. Midcourse Corrections

An issue that must be addressed in any trajectory design is performing midcourse corrections when the vehicle deviates from the reference path. Many times this correction is computed using a first-order state transition matrix time history that is stored onboard. In order to illustrate the idea, a departure from the reference path at time \( t^* \) will result in a first-order correction of the terminal error as given in the following expression:

\[
delta x(t^*) = \frac{\delta r(t^*)}{\delta v(t^*)} = \Phi(t^*, t_f) \left[ \begin{array}{c} \delta r(t_f) \\ \delta v(t_f) \end{array} \right]
\]

where

\[
\Phi(t^*, t_f) = \begin{bmatrix}
\frac{\partial r(t^*)}{\partial r(t_f)} & \frac{\partial r(t^*)}{\partial v(t_f)} \\
\frac{\partial v(t^*)}{\partial r(t_f)} & \frac{\partial v(t^*)}{\partial v(t_f)}
\end{bmatrix}
\]

The procedure is given by Battin in Reference 9. Suppose we wish to maintain the reference path, then we choose \( \delta r(t_f) = 0 \) and solve Eq. (26) for \( \delta v(t^*) \), the instantaneous velocity correction to compensate for the position error, \( \delta r(t^*) \), at that time. The state transition matrix required in Eq. (26) can be computed by utilizing the special "group" property of the first-order state transition matrices which hold in the general case:

\[
\Phi(t_3, t_1) = \Phi(t_1, t_2) \Phi(t_2, t_3) \\
= \Phi(t_1, t_2) \Phi(t_2, t_3)^{-1}
\]

Therefore, we can compute \( \Phi(t^*, t_f) = \Phi(t^*, t_0) \Phi(t_f, t_0)^{-1} \) using the forward integrated, stored history of state transition matrices described by \( \Phi(t, t_0) \) for \( t_0 \leq t \leq t_f \) and then solve Eq. (26). This property and the symplectic property, which allows for matrix inversion to be replaced by a transpose operation and pre- and post-multiplication by an orthogonal matrix, of first-order state transition matrix are well known.

Now suppose that we desire to perform a second-order midcourse correction. Thus we write

\[
delta x(t^*) = \Phi(t^*, t_f) \delta x(t_f) + \frac{1}{2} \Phi(t^*, t_f, t_f) \delta x(t_f) \delta x(t_f)
\]
where $\Phi(t^*, t_f, t_f)$ is a second order state transition matrix and is given by

$$
\Phi(t^*, t_f, t_f) = \begin{bmatrix}
\frac{\partial^2 r(t^*)}{\partial r^2(t_f)} & \frac{\partial^2 r(t^*)}{\partial v^2(t_f)} \\
\frac{\partial^2 v(t^*)}{\partial r^2(t_f)} & \frac{\partial^2 v(t^*)}{\partial v^2(t_f)} \\
\frac{\partial^3 r(t^*)}{\partial r^3(t_f)} & \frac{\partial^3 v(t^*)}{\partial v^3(t_f)} \\
\frac{\partial^3 v(t^*)}{\partial r^3(t_f)} & \frac{\partial^3 v(t^*)}{\partial v^3(t_f)}
\end{bmatrix}
$$

Again, choosing $\delta r(t_f) = 0$, we can solve Eq. (28) for $\delta v(t_f)$

$$
\delta v(t_f) = \left[ \frac{\partial r(t^*)}{\partial \delta v(t_f)} \right]^{-1} \delta r(t^*)
$$

$$
-\frac{1}{2} \left[ \frac{\partial r(t^*)}{\partial \delta v(t_f)} \right]^{-1} \left\{ \left[ \frac{\partial^2 r(t^*)}{\partial r^2(t_f)} \right] \left[ \frac{\partial r(t^*)}{\partial \delta v(t_f)} \right]^{-1} \delta r(t^*) \right\} \left[ \frac{\partial^2 r(t^*)}{\partial v^2(t_f)} \right]^{-1} \delta r(t^*) \right\}
$$

and eliminate it from expression in order to compute $\delta v(t^*)$:

$$
\delta v(t^*) = \frac{\partial v(t^*)}{\partial \delta v(t_f)} \delta v(t_f) + \frac{1}{2} \frac{\partial^2 v(t^*)}{\partial \delta v^2(t_f)} \delta v(t_f) \delta v(t_f)
$$

Equation (30) represents the velocity change required to maintain the reference path including up to second-order terms. We stop at second-order for no particular reason. Higher-order terms can easily be included if desired.

In the above developments it was shown that the first-order state transition matrix in Eq. (28) is readily computed from the onboard stored data. It is worth noting here that the second- and higher-order state transition matrices do not share the “group” property of first-order state transition matrices. This was verified for the second-order case and is clearly follows for higher-order as well. On the other hand, it can be seen that second- and higher-order guidance laws can be accomplished by integrating backward from time $t_f$ in order to compute $\Phi(t^*, t_f, t_f)$ in Eq. (28). Higher-order guidance laws will increase the computations cost to some extent; however, the significance of a more accurate dynamical prediction cannot be overlooked since there is potential for reducing the magnitude and/or frequency of velocity corrections that are required to meet mission objectives.

VI. Automatic Generation of Co-state Equations

The OCEA automatic differentiation facility enables the automatic generation of co-state equations. It can be seen from Eq. (9) that by simply differentiating the Hamiltonian with respect to the states, we can produce the co-state differential equations. Additionally a numerical integration routine can readily solve the automatically generated co-state equations. Although a considerable amount of simplification can be performed in some optimal control problems (e.g. some co-states may be known to be zero for all time if a terminal state is a free unspecified parameter), the impact of automatically generated co-state equations results in time and programming savings especially for systems containing many states and/or complicated equations of motion.

VII. Numerical Examples

A. Low-thrust Mars-Earth Transfer by Differential Corrections

In this example, we demonstrate two developments enabled by OCEA: 1) the automatic generation of co-state equations, and 2) higher-order sensitivity calculations. The problem to be considered is a minimum time Low-thrust Mars-Earth transfer. The performance index and the polar heliocentrical equations of motion are given by Eqs. (31-35). These equations are in the form of a fixed time problem with the final time as a free parameter.
\[ J = \int_0^1 t_f \, d\tau \]  
(31)

\[ r' = t_f \, u \]  
(32)

\[ \theta' = t_f \frac{v}{r} \]  
(33)

\[ u' = t_f \left( \frac{v^2}{r} - \frac{\mu}{r^2} + \frac{2\varepsilon P}{I_{sp} g (m_0 + m \tau)} \sin \alpha \right) \]  
(34)

\[ v' = t_f \left( -uv \frac{r}{r} + \frac{2\varepsilon P}{I_{sp} g (m_0 + m \tau)} \cos \alpha \right) \]  
(35)

where the mass flow rate is given by

\[ m' = -t_f \frac{2\varepsilon P}{I_{sp} g} \]  

and \( r \) is a radial distance from the sun, \( \theta \) is the angle measured from a reference line, and \( \alpha \) is the control angle. The variables \( u \) and \( v \) are respectively, the velocities in radial and tangential directions. The initial mass is \( m_0 \), \( \mu \) is the gravitational parameter of the sun, \( g \) is the gravitational acceleration at sea level on Earth, \( P \) is the power, \( \varepsilon \) is the engine efficiency, and \( I_{sp} \) is the specific impulse.

Equation (10) results in the following optimal control law

\[ \alpha = \tan^{-1} \left( \frac{-\lambda_u}{-\lambda_v} \right) \]  
(36)

From Eqs (31-35), the Hamiltonian for the system can be defined as

\[ H = t_f \left[ 1 + \lambda_r (u) + \lambda_u \left( \frac{v}{r} \right) + \lambda_v \left( \frac{v^2}{r} - \frac{\mu}{r^2} + \gamma \sin \alpha \right) + \lambda_v \left( \frac{-uv}{r} + \gamma \cos \alpha \right) \right] \]  
(37)

The co-state differential equations are automatically generated through Eq. (9) by differentiating the Hamiltonian given in Eq. (37). The system is comprised of 10 total equations including Eqs. (32-35), the four co-state differential equations, and Eqs. (38-39). Here, \( t_f \) is considered as a free parameter; therefore, we must satisfy an additional necessary condition as described in Eq. (14). Eq. (39) is the differential equation representation of this integral boundary condition.

\[ t_f' = 0 \]  
(38)
\[ z = \frac{\partial H}{\partial t_f} \]  

(39)

For these solutions, the initial mass is 842.5 kg, the power is 10 kW, the engine efficiency is 0.661, and the specific impulse is 3945 sec.

The desired initial and terminal boundary conditions are given in Table 1. For this problem we are using astronomical units for distance and non-dimensional time units where 1 AU = 1.49599965e11 m and 1 TU = 5.02267576e6 sec.

<table>
<thead>
<tr>
<th>Initial Time (( \tau = 0 ))</th>
<th>Final Time (( \tau = 1 ))</th>
</tr>
</thead>
<tbody>
<tr>
<td>( r(AU) )</td>
<td>1.524</td>
</tr>
<tr>
<td>( \theta )</td>
<td>0</td>
</tr>
<tr>
<td>( u(AU/TU) )</td>
<td>0</td>
</tr>
<tr>
<td>( v(AU/TU) )</td>
<td>0.81</td>
</tr>
<tr>
<td>( z )</td>
<td>0</td>
</tr>
</tbody>
</table>

Table 1. Initial and Final Conditions

We could eliminate \( \lambda_j \) from the Hamiltonian since it is zero for all times as a result of \( \theta(1) \) being a free terminal state. This would result in lower dimension solution; however, we retain this variable in the formulation for completeness in these developments.

In summary, we have four unknowns and four boundary and necessary conditions to satisfy in order to solve the problem. The unknowns are the three initial unknown co-states and \( t_f \). The conditions to be met at the final time are the four terminal conditions listed in Table 1. We begin the solution process by guessing the three unknown co-states and \( t_f \). We have chosen to solve this problem by first- and second-order methods, and for two cases of initial conditions. For Case I, we start with a relatively good initial guess and for Case II we start with a poor initial guess for the unknowns. Arising from the form of the optimal control, a good guess essentially implies that the ratio of the co-states are suitably scaled. For each case, we choose as a stopping criterion that the largest terminal error condition be less than 1e-12. This results in errors of the order of 10 centimeters and 10 centimeters/sec.

1. **Case I: Good starting guess**

Initial guess for unknowns: \( \{ \lambda_r, \lambda_u, \lambda_v, t_f \} = \{1.0, 1.0, 1.7, 4.4\} \)

<table>
<thead>
<tr>
<th>Iteration</th>
<th>( \delta r(\tau = 1) )</th>
<th>( \delta u(\tau = 1) )</th>
<th>( \delta v(\tau = 1) )</th>
<th>( \delta z(\tau = 1) )</th>
<th>( \varepsilon = \delta^T \delta )</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0.124285D-01</td>
<td>-0.412110D-01</td>
<td>-0.455149D-01</td>
<td>-0.850729D+00</td>
<td>0.727664D+00</td>
</tr>
<tr>
<td>1</td>
<td>0.552611D-02</td>
<td>-0.454643D-01</td>
<td>-0.232267D-01</td>
<td>0.668859D-02</td>
<td>0.268176D-02</td>
</tr>
<tr>
<td>2</td>
<td>-0.102705D-01</td>
<td>-0.102573D-01</td>
<td>0.458271D-02</td>
<td>0.155596D-02</td>
<td>0.234119D-03</td>
</tr>
<tr>
<td>3</td>
<td>0.551949D-03</td>
<td>0.248241D-03</td>
<td>-0.464508D-03</td>
<td>0.303631D-04</td>
<td>0.582962D-06</td>
</tr>
<tr>
<td>4</td>
<td>0.243315D-06</td>
<td>0.528947D-06</td>
<td>-0.263844D-06</td>
<td>0.518276D-02</td>
<td>0.411287D-12</td>
</tr>
<tr>
<td>5</td>
<td>-0.131073D-11</td>
<td>-0.239765D-12</td>
<td>0.113398D-11</td>
<td>0.916281D-13</td>
<td>0.306981D-23</td>
</tr>
<tr>
<td>6</td>
<td>0.888178D-15</td>
<td>0.891648D-15</td>
<td>-0.222045D-15</td>
<td>-0.393782D-15</td>
<td>0.178827D-29</td>
</tr>
</tbody>
</table>

Table 2. First-Order Solution Terminal Errors

First-order solution: \( \{ \lambda_r, \lambda_u, \lambda_v, t_f \} = \{6.822434207, 7.020330890, 10.852450714, 4.591579951\} \)
Tables 2 and 3 show the results for the errors in the terminal boundary conditions and a norm error measure of these errors for a first-order method and a first- and second-order teaming method, respectively. In Table 2, bold indicates second-order. Rapid convergence is found when utilizing second-order sensitivity calculations after two iterations as shown in Table 3. The time required for the solutions in Tables 2 and 3 require 24 second and 1 min 50 seconds, respectively. Both solutions agree to 8 digits. The time of flight is 267 days.

2. Case II: Poor starting guess

Initial guess for unknowns: \( \{\lambda_1, \lambda_2, \lambda_3, t_f\} = \{1.0, 1.0, 1.0, 4.0\} \)

Tables 4 and 5 show the results for the errors in the terminal boundary conditions for a first-order method and a first- and second-order teaming method, respectively. Improved convergence is found when utilizing second-order
sensitivity calculations after seven iterations as shown in Table 5. The time required for the solutions in Tables 4 and 5 require 42 second and 1 min 41 seconds, respectively.

3. Domain of Convergence Study

In this section we study the domain of convergence of the first- and second-order algorithms. Here, we simply vary the initial guess by scaling the known optimal solution. The results are given in Table 6. In these results, we choose either the first-order or second-order algorithm with no teaming solution as was done in the earlier test cases.

<table>
<thead>
<tr>
<th>$\alpha \cdot (x_{\text{guess}} = \alpha x_{\text{optimal}})$</th>
<th>First-order iteration count</th>
<th>Second-order iteration count</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.50 Does not converge</td>
<td></td>
<td>Does not converge</td>
</tr>
<tr>
<td>0.60</td>
<td>7</td>
<td>Does not converge</td>
</tr>
<tr>
<td>0.70</td>
<td>6</td>
<td>5</td>
</tr>
<tr>
<td>0.80</td>
<td>6</td>
<td>4</td>
</tr>
<tr>
<td>0.85</td>
<td>5</td>
<td>4</td>
</tr>
<tr>
<td>0.90</td>
<td>5</td>
<td>3</td>
</tr>
<tr>
<td>0.95</td>
<td>4</td>
<td>3</td>
</tr>
<tr>
<td>0.98</td>
<td>4</td>
<td>3</td>
</tr>
<tr>
<td>0.99</td>
<td>3</td>
<td>2</td>
</tr>
<tr>
<td>0.995</td>
<td>3</td>
<td>2</td>
</tr>
<tr>
<td>1.0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1.005</td>
<td>3</td>
<td>2</td>
</tr>
<tr>
<td>1.01</td>
<td>3</td>
<td>2</td>
</tr>
<tr>
<td>1.02</td>
<td>4</td>
<td>3</td>
</tr>
<tr>
<td>1.05</td>
<td>4</td>
<td>3</td>
</tr>
<tr>
<td>1.10</td>
<td>5</td>
<td>3</td>
</tr>
<tr>
<td>1.15</td>
<td>6</td>
<td>6</td>
</tr>
<tr>
<td>1.20</td>
<td>7</td>
<td>7</td>
</tr>
<tr>
<td>1.30</td>
<td>9</td>
<td>Does not converge</td>
</tr>
<tr>
<td>1.40</td>
<td>Does not converge</td>
<td>Does not converge</td>
</tr>
<tr>
<td>1.50</td>
<td>Does not converge</td>
<td>Does not converge</td>
</tr>
</tbody>
</table>

Table 6. Domain of Convergence Studies

The results of Table 6 detail the domain of convergence for the first- and second-order algorithms. We have investigated an interval of +/- 50% of the optimal solution. As expected, the second-order algorithm shows significant improvement in rate of convergence when near the solution, and the first-order algorithm has a larger domain of convergence although it is not significantly larger. These results are consistent with those results from the teaming solutions in Cases I and II. That is, the larger domain of convergence of the first-order algorithms provides a means to reach the domain of convergence of the second order algorithm. Once in this region, the second-order algorithm provides a superior rate of convergence.

4. Results Summary

It should be kept in mind when examining these solutions that no partial derivatives were computed and coded by hand in producing the co-state differential equations and the state transition matrix differential equations. Extending the method to second-order increased the computational burden; however, there is no increase in analyst effort in going to higher-order since no hand calculations or additional coding, other than the algorithm itself which is coded once and for all, are needed. The decrease in time for analytical developments more than offsets the increase in computational cost for the second-order algorithm. The OCEA automatic differentiation environment provides a means for focusing solely on new algorithm development – in this case, the evaluation of higher-order methods.

With the method of differential corrections, computation of second-order sensitivities results in an increase in the number of equations by a factor equal to the number of differential equations to be integrated (10 in this case). This is a drawback of the method since higher-order methods require integration of a larger number of equations. Due to coupling, the entire 10x10 first-order state transition matrix must be computed although only a 4x4 partition is needed to solve by the method of differential corrections. Significant reductions in computational cost related to
symmetry and sparsity have not yet been exploited, and no advanced optimization techniques, incorporating line searching, has been attempted in these solutions.

The results for each solution are equivalent to eight digits. Therefore, we show plots for the motion only once. Figure 1 shows the Mars-Earth optimal trajectory. The velocity states and co-states are plotted in Figures (2) and (3) respectively. Figure 4 shows the optimal thrust angle control.
B. An Example by Differential Inclusions

Here we consider a simple nonlinear scalar system in order to illustrate the OCEA method in a direct optimization example. Consider

$$\dot{x} = -x + \varepsilon x^3 + u$$

with performance index to be minimized given by

$$J = \frac{1}{2} \int_0^5 u^2 dt$$

The discretized form of the equations of motion and performance index are then given by

$$\frac{x_{k+1} - x_k}{\Delta} + \frac{x_{k+1} + x_k}{2} + \varepsilon \left( \frac{x_{k+1} + x_k}{2} \right)^3 = u_k$$

and

$$J = \frac{1}{2} \sum_{k=1}^{10} u_k^2$$

Here we choose $\varepsilon = 0.01$ and, approximate the system by conveniently choosing 10 time intervals over the 5 second, thus $\Delta = 0.5$ seconds. We choose as boundary conditions $x_1 = 3$ and $x_{11} = 0$.

Once the relation for the control given in Eq. (42 is substituted into Eq. (43) we have the cost as a function of the unknown states as

$$J = J(x_k); \quad k = 2, 3, \ldots, 10$$

A standard optimization procedure is employed now to solve for the nine unknown states. As mentioned earlier, $\nabla J = \mathbf{0}$ defines a set of nonlinear equations ($\mathbf{h}(x_k) = \mathbf{0}$) which must be solved iteratively for the unknown states. The Hessian of $J$, $\nabla^2 J$, provides the matrix of first-order sensitivities ($G = \nabla^2 J$) required for the solutions. OCEA’s automatic differentiation capability enables an efficient tool for computing and evaluating these partial derivatives, and higher-order partials if desired. A first-order correction to the unknown states can be written as
\[ \Delta x = (G^T G)^{-1} G^T h(x) \] (45)

We choose to solve this example using Matlab’s Symbolic Toolbox in order to compare the effort required in using OCEA and Matlab. Essentially, this is a test of symbolic and automatic differentiation methods. The difference comes when we need to compute and evaluate the derivatives of Eq. (44). With Matlab, we must type a command for each partial derivative required. In computing first- through third-order partials of the cost function, many ten’s of these commands are needed, at best, using Matlab. OCEA is ideally suited for solution of direct optimization problems such as the minimization of the expression given in Eq. (44). No coding effort is required beyond specifying the cost function since all required partial derivatives are automatically computed and evaluated. The implementation of higher-order solutions for direct optimization problems results in relatively small increase in the computational cost and no additional effort to derive or code partials for the higher-order terms using OCEA.

Table 7 shows a comparison of solving the problem by first- and second-order methods. More rapid convergence is found for the second-order method.

<table>
<thead>
<tr>
<th>Iteration count</th>
<th>First-order error</th>
<th>Second-order error</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1.4313e+002</td>
<td>1.4313e+002</td>
</tr>
<tr>
<td>2</td>
<td>2.1830e-001</td>
<td>1.8792e-002</td>
</tr>
<tr>
<td>3</td>
<td>2.9154e-007</td>
<td>4.0911e-012</td>
</tr>
<tr>
<td>4</td>
<td>2.6237e-018</td>
<td>8.6737e-030</td>
</tr>
<tr>
<td>5</td>
<td>8.3470e-030</td>
<td></td>
</tr>
</tbody>
</table>

Table 7. Differential Inclusions Solution Errors

A comparison of the optimal solutions and that by Differential Inclusions is shown in Figure 5.

As expected, the Differential Inclusions solution provided a starting guess for the initial co-state that resulted in convergence to the optimal solution in only two iterations using Matlab’s \texttt{fmincon} function. On the other hand, blind guessing resulted in many unsuccessful attempts to choose a suitable starting guess. A check of the performance index of these two solutions shows that the approximate Differential Inclusions solution is sub-optimal as anticipated. Of course a larger number of time intervals can be chosen; however, the objective of this exercise is to demonstrate the utility of a computational tool which can reduce the time involved and coding effort involved in this and many other optimization applications.
VI. Conclusion

In this paper, we demonstrated by several examples the solution of nonlinear Two-point Boundary Value Problems using automatic differentiation. The automatic differentiation tool, OCEA, was utilized to compute the partial derivatives required in the solution of state transition matrix differential equations and the sensitivities required for solving discrete optimization problems. OCEA has the capability to compute higher-order sensitivities which enable extending traditional algorithms to higher-order for improving rate of convergence. Further uses of automatic differentiation result in automatically generating co-state differential equations and enabling second- and higher-order guidance laws.

VII. References

APPENDIX A: State Transition Matrix Differential Equations

The state transition differential equations are here written in indicial notation. Note: All indices run from 1 to \( n_s \) where \( n_s \) is the number of states. Initial conditions are the identity matrix for the first order state transition matrix differential equations, and zeros for second and higher order state transition matrix differential equations.

First order:

\[
\dot{\Phi}_{ij} = f_{i,s} \Phi_{sj} \\
\Phi_{sj} = \Phi_{ij}(t, t_o) = \frac{\partial x_i(t)}{\partial x_j(t_o)}
\] (A.1) (A.2)

Second order:

\[
\dot{\Phi}_{ijk} = f_{i,s} \Phi_{sjk} + f_{i,st} \Phi_{ik} \Phi_{sj} \\
\Phi_{ijk} = \Phi_{ijk}(t, t_o, t_o) = \frac{\partial^2 x_i(t)}{\partial x_j(t_o) \partial x_k(t_o)}
\] (A.3) (A.4)

Third order:

\[
\dot{\Phi}_{ijkl} = f_{i,s} \Phi_{sjkl} + f_{i,su} \Phi_{ul} \Phi_{sjk} + f_{i,st} \Phi_{ik} \Phi_{sjl} \\
+ f_{i,st} \Phi_{ik} \Phi_{sml} + f_{i,stu} \Phi_{ul} \Phi_{ik} \Phi_{sjl} \\
\Phi_{ijkl} = \Phi_{ijkl}(t, t_o, t_o, t_o) = \frac{\partial^3 x_i(t)}{\partial x_j(t_o) \partial x_k(t_o) \partial x_l(t_o)}
\] (A.5) (A.6)

Fourth order:

\[
\dot{\Phi}_{ijklm} = f_{i,s} \Phi_{sjklm} + f_{i,suv} \Phi_{vm} \Phi_{sijkl} \\
+ f_{i,su} \Phi_{ul} \Phi_{sjkm} + f_{i,su} \Phi_{ulm} \Phi_{sjk} + f_{i,suv} \Phi_{vm} \Phi_{ul} \Phi_{sijk} \\
+ f_{i,st} \Phi_{ik} \Phi_{sjlm} + f_{i,st} \Phi_{ik} \Phi_{sjm} + f_{i,suv} \Phi_{vm} \Phi_{ul} \Phi_{sijkl} \\
+ f_{i,stu} \Phi_{ik} \Phi_{sml} + f_{i,stu} \Phi_{ik} \Phi_{sml} + f_{i,stu} \Phi_{ul} \Phi_{ik} \Phi_{sjl} \\
\Phi_{ijklm} = \Phi_{ijklm}(t, t_o, t_o, t_o, t_o) = \frac{\partial^4 x_i(t)}{\partial x_j(t_o) \partial x_k(t_o) \partial x_l(t_o) \partial x_m(t_o)}
\] (A.7) (A.8)