High-Order State and Parameter Transition Tensor Calculations

James D. Turner*, Manoranjan Majji†, and John L. Junkins‡
Texas A&M University, College Station, Texas 77843-3141

Most mathematical models governing physical systems are described by first-order ordinary differential equations. The local phase space flows of such nonlinear differential equations are described by state and parameter transition tensors. This paper presents a mixed numeric/symbolic algorithm for automating the derivation, coding, and computation of tensor-based generalizations of such flows. Automatic differentiation algorithms provide the partial derivative models required for assembling the tensor differential equations for the state and parameter transition tensors. The tensor equations are described by implicit differential equations. Complicated analytic models are available through the use of Faà di Bruno’s formula for differentiation of composite functions. A simple vector generalization for Faà di Bruno’s formula is presented that builds on a recursive integer algorithm. An array-of-arrays data structure is introduced for (1) tracking the derivative terms arising during a derivation of the tensor necessary conditions, and (2) assembling a generalized state space model for integrating the state and tensor differential equation models. Pell exponential numbers model the total number of implicit derivative calculations performed at each expansion order. The integer codes that describe the derivative derivation for the tensors are transformed automatically into FORTRAN by a Fortran-based string manipulation routine. The goal of the analysis is to generate a perturbation model, based on the local flow of solutions for the nonlinear system, for general-purpose applications in uncertainty analysis, quantification of state estimate uncertainty for use in navigation algorithms and construction of neighboring extremal solutions in guidance problems. The new algorithms presented herein are expected to expedite the computation of the perturbation descriptions of such departure motion nonlinear systems by a systematic exploitation of the structure and symmetry of the tensor fields.

I. Introduction

Sensitivity calculations begin with a differential equation that describes the response of a system to external loads, where the linear differential equation model is assumed to be given by \( \dot{x} = f(x, p, t) \), \( x \in \mathbb{R}^N \) denotes the state vector, and \( p \in \mathbb{R}^M \) denotes the parameter state vector. Two classes of simulation are important: (1) prediction of the future state motion, and (2) motion uncertainty predictions due to the variations in the problem initial conditions and model parameter values. The tensor transition methods presented in this paper are ideally suited for understanding and manipulating such probabilistic models for simulating uncertainty in the state evolution dynamics.

* Research Professor, Director Of Operations for Consortium for Autonomous Space Systems, Aerospace Engineering Department, 3141 TAMU, Associate Fellow, AIAA.
† Graduate Research Assistant, Aerospace Engineering Department, 3141 TAMU, Student Member, AIAA.
‡ Regents Professor, Distinguished Professor of Aerospace Engineering, Holder of the Royce E. Wisenbaker Chair in Engineering, Aerospace Engineering Department, 3141 TAMU, Fellow, AIAA.
Classical state transition matrix calculations describe the first-order sensitivity of the final state value to variation in either initial conditions or model parameters [1]. Higher-order generalizations of transition matrices are required for handling nonlinear applications where uncertainty propagation is of interest [2,3]; including, but are not limited to Monte Carlo simulation and filter design [3-5, 25,26]. Tensor-based models are required for 2nd order sensitivity models and beyond. Two critical issues are addressed in this paper for handling this class of problems: (1) computer-automated derivation and coding for the transition tensors, and (2) a generalized scalar data structure is introduced for numerically integration the generalized state space that consists of the state and transition tensor terms.

As shown in Figure 1, the goal of this paper is to present an algorithm for computer aided derivation and coding of an executable FORTRAN subroutine for evaluating the state as well as its initial condition and parameter transition tensors. This is a multi-step process. First, the chain rule of calculus for implicit functions is expressed in the form of a recursive integer program that automates the derivation for the multivariable tensor derivative model. The integer codes capture the unique mathematical description for each derivative operation performed in a tensor Taylor series expansion for a first order differential equation. Second, after completing the tensor derivative derivation calculations, an efficient FORTRAN string manipulation program transcribes the integer codes for the tensor derivative models into an executable program. This step involves symbolically generating the subroutine calls for implementing the implied tensor contraction operations, as well as building the software interfaces for passing the required data for performing the calculations. Third, the state and transition tensor model is numerically integrated by defining a scalar abstract compound data object, that uses operator overloading features of modern compilers to automate derivation, saving, and processing the state and tensor differential equations [2,3]. The analyst only defines the desired tensor order; no further analyst intervention is required for generating an executable FORTRAN 95/2003 subroutine that implements the tensor differential equations. Repeated packing and unpacking for the tensor differential equations is avoided altogether by defining operator overloading commands for the basic math operations required in the numerical integration algorithm. This overloading allows the user to simply write code appropriate for a generalized scalar object to handle the integration process for a generalized state space variable. All chain-rule differentiation processes are automated in a recursive fashion. The algorithm scales to arbitrary order tensor expansions. A complete simulation tool is obtained by linking the string-generated subroutine for computing the tensor differential equations with Turner’s Object-Oriented Coordinate Embedding Algorithm (OCEA) computational differentiation tool for providing the high-order differential equation gradients required in the tensor models [2,3]. The paper is divided into four major sections. First section introduces the state and parameter transition tensor differential equations governing the departure motion of nonlinear dynamical systems.

II. Mathematical Model

The classical first-order state and parameter transition matrix models are derived and used as the starting point for generating the higher-order algorithms. The parameter transition matrix equation is shown to be simplified by defining an augmented state space that combines both the state and parameter vector. The celebrated Faà di Bruno’s formula [6-9] is presented for computing arbitrary order implicit rate calculations for scalar applications. The challenges encountered with applying di Bruno’s formula for vector systems are reviewed [10-12].

A. First Order State and Parameter Sensitivity Models.

The governing equations for the state transition matrix models are derived by observing that the math model for the system dynamics is cast in the form

$$\dot{x} = f(x,p,t) \quad x(t_0) = x_0$$

where $x \in \mathbb{R}^n$ denotes the state vector, $p \in \mathbb{R}^m$ denotes the parameter vector, and $t$ denotes time. The functional form for the solution for Eq. (1) is given by

$$x(t) = x(t_0) + \int_{t_0}^t f(x(\tau),p,\tau) d\tau$$

First-order differential equations are developed for both the state and parameter sensitivity models. Second- through fourth-order models are presented in Appendix A.
1. Initial Condition State Transition Matrix

The initial condition state transition matrix is computed by differentiating Eq. (2) w.r.t. \(x(t_0)\), leading to

\[
\frac{\partial x(t)}{\partial x(t_0)} = I_{nxn} + \int_{t_0}^{t} \frac{\partial f(x(\tau), p; \tau)}{\partial x(\tau)} \frac{\partial x(\tau)}{\partial x(t_0)} d\tau
\]

This is further differentiated w.r.t. time to yield the differential equation

\[
\frac{d}{dt} \left( \frac{\partial x(t)}{\partial x(t_0)} \right) = \frac{\partial f(x(t), p; t)}{\partial x(t)} \frac{\partial x(t)}{\partial x(t_0)} + \frac{\partial f(x(t), p; t)}{\partial p} \frac{\partial x(t)}{\partial p} \bigg|_{t=t_0} = I_{nxn}
\]

(3)

The derivation of the higher order initial condition partial derivative is simplified by introducing the variables

\[
\frac{d\Phi^{(1)}(t,t_0)}{dt} = \nabla f. \Phi^{(1)}(t,t_0); \quad \Phi^{(1)}(t,t_0) = I_{nxn}
\]

(4)

where \(\Phi^{(1)}(t,t_0) = \frac{\partial x(t)}{\partial x(t_0)}\) denotes the first order state transition tensor and \(\nabla f = \frac{\partial f(x(t), p; t)}{\partial x(t)}\) is the jacobian of the vector nonlinear function appearing in the mathematical model for the system. During transition tensor calculations Turner’s OECA [2,3] computational differentiation software provides the numerical values required for \(\nabla f, \nabla^2 f, \ldots\).

2. Parameter State Transition Matrix

The parameter state transition matrix is computed by differentiating Eq. (2) w.r.t. \(p\), leading to

\[
\frac{\partial x(t)}{\partial p} = \int_{t_0}^{t} \left[ \frac{\partial f(x(\tau), p, \tau)}{\partial x(\tau)} \frac{\partial x(\tau)}{\partial p} + \frac{\partial f(x(\tau), p, \tau)}{\partial p} \right] d\tau
\]

This is further differentiated w.r.t. time to yield the differential equation

\[
\frac{d}{dt} \left( \frac{\partial x(t)}{\partial p} \right) = \frac{\partial f(x(t), p; t)}{\partial x(t)} \frac{\partial x(t)}{\partial p} + \frac{\partial f(x(t), p; t)}{\partial p} \bigg|_{t=t_0} = 0_{nxm}
\]

(5)

The derivation of the higher order parameter partial derivatives is simplified by introducing the variables

\[
\frac{d\Theta^{(1)}}{dt} = \nabla f. \Theta^{(1)} + \frac{\partial f(x(t), p; t)}{\partial x(t)} \bigg|_{t=t_0} = 0_{nxm}
\]

(6)

where \(\Theta^{(1)} = \frac{\partial x(t)}{\partial p}\) and \(\nabla f = \frac{\partial f(x(t), p; t)}{\partial x(t)}\).

B. Augmented State Space Form for the Parameter Tensor Calculation

The parameter tensor calculation is greatly simplified by re-defining the governing differential equation of Eq. (1) as the augmented system [13]

\[
\begin{bmatrix}
\dot{x} \\
\dot{p}
\end{bmatrix} = \begin{bmatrix}
f(x(t), p; t) \\
0
\end{bmatrix} = F(x, p, t); \quad \begin{bmatrix} x(t_0) \\ p(t_0) \end{bmatrix} = F(x_0, p)
\]

(7)

This assumes that the parameters are constant. Introducing the augmented vector

\[
\dot{z} = F(z, t); \quad z = \begin{bmatrix} x \\ p \end{bmatrix}
\]

(8)

one obtains the functional form of the integrated solution given by

\[
z = z_0 + \int_{t_0}^{t} F(z, \tau) d\tau
\]

(9)
The parameter partial derivative of Eq. (11) follows as

\[ \frac{\partial z}{\partial z_0} = \frac{\partial z_0}{\partial z_0} + \int_{\tau_0}^{\tau} \left( \frac{\partial F}{\partial z} \right) \left( \frac{\partial z}{\partial z_0} \right) d\tau \]

\[ = I_{(n+m)\times(n+m)} + \int_{\tau_0}^{\tau} \nabla_z F^* \left( \frac{\partial z}{\partial z_0} \right) d\tau; \quad \frac{\partial z}{\partial z_0} \bigg|_{\tau=\tau_0} = I_{(n+m)\times(n+m)} \] (12)

The augmented parameter transition tensor differential equations follows by differentiating w.r.t. time, yielding

\[ \frac{d}{dt} \frac{\partial z}{\partial z_0} = \nabla_z F \frac{\partial z}{\partial z_0}; \quad \frac{\partial z}{\partial z_0} \bigg|_{\tau=\tau_0} = I_{(n+m)\times(n+m)} \] (13)

where the differential equation now has the same structure as the initial condition transition tensor presented in Eq. (4). As a result, the integer code algorithm presented in this paper works for both the initial condition and parameter sensitivity calculations presented for Eqs. (4) and (13), respectively. Let us now perform a simple demonstration of Eq. (13) to show that it is same as Eqs. (7)(or (8)) and (4) (or (5)) grouped together. Considering the expression for

\[ \nabla_z F = \frac{\partial F}{\partial z} = \begin{bmatrix} \frac{\partial f(x(t), p; t)}{\partial x} & \cdots & \frac{\partial f(x(t), p; t)}{\partial p} \\ \vdots & & \vdots \\ 0 & \cdots & 0 \end{bmatrix} \] (14)

together with the definition of the augmented state transition matrix given by

\[ \frac{\partial z}{\partial z_0} = \begin{bmatrix} \frac{\partial x}{\partial x_0} & \frac{\partial x}{\partial p} \\ \frac{\partial p}{\partial x_0} & I_{\text{non}} \end{bmatrix} \] (15)

in Eq. (13) we have that

\[ \frac{d}{dt} \left( \frac{\partial z}{\partial z_0} \right) = \frac{d}{dt} \begin{bmatrix} \frac{\partial x}{\partial x_0} & \frac{\partial x}{\partial p} \\ \frac{\partial p}{\partial x_0} & I_{\text{non}} \end{bmatrix} \begin{bmatrix} \frac{\partial f(x(t), p; t)}{\partial x} & \cdots & \frac{\partial f(x(t), p; t)}{\partial p} \\ 0 & \cdots & 0 \end{bmatrix} \] (16)

Introducing more manipulations on the block-system we have that
\[
\begin{align*}
\frac{d}{dt} \left( \frac{\partial x}{\partial x_0} \right) &= \left[ \frac{\partial f(x(t), p; t)}{\partial x} + \frac{\partial f(x(t), p; t)}{\partial p} \right] \cdot \left[ \frac{\partial x}{\partial x_0}, \frac{\partial x}{\partial p} \right] - \frac{\partial p}{\partial x_0} \cdot \left[ \frac{\partial p}{\partial x}, \frac{\partial p}{\partial p} \right] \\
\frac{d}{dt} \left( \frac{\partial p}{\partial x_0} \right) &= \left[ \frac{\partial f(x(t), p; t)}{\partial x} + \frac{\partial f(x(t), p; t)}{\partial p} \right] \cdot \left[ \frac{\partial x}{\partial x_0}, \frac{\partial x}{\partial p} \right] - \frac{\partial p}{\partial x_0} \cdot \left[ \frac{\partial p}{\partial x}, \frac{\partial p}{\partial p} \right] \\
\end{align*}
\]

(17)

Where the functional dependencies of \( f(.) \) are dropped with no loss in clarity and the sensitivity \( \frac{\partial p}{\partial p} \) which is obviously an identity matrix is retained to explicitly show that the augmented state transition tensor equations give us this solution automatically. To realize this and other claims, we compare the block elements in the left and right hand sides of the matrix differential equation (17) and obtain the following four independent equations.

\[
\begin{align*}
\frac{d}{dt} \left( \frac{\partial x}{\partial x_0} \right) &= \left[ \frac{\partial f(x(t), p; t)}{\partial x} + \frac{\partial f(x(t), p; t)}{\partial p} \right] \cdot \left[ \frac{\partial x}{\partial x_0} \right] - \frac{\partial p}{\partial x_0} \cdot \left[ \frac{\partial p}{\partial x} + \frac{\partial p}{\partial p} \right] \\
\frac{d}{dt} \left( \frac{\partial x}{\partial p} \right) &= \left[ \frac{\partial f(x(t), p; t)}{\partial x} + \frac{\partial f(x(t), p; t)}{\partial p} \right] \cdot \left[ \frac{\partial x}{\partial p} \right] - \frac{\partial p}{\partial x_0} \cdot \left[ \frac{\partial p}{\partial p} \right] \\
\frac{d}{dt} \left( \frac{\partial p}{\partial x_0} \right) &= 0 \\
\frac{d}{dt} \left( \frac{\partial p}{\partial p} \right) &= 0
\end{align*}
\]

(18) (19) (20) (21)

The last two differential Eqs. ((20) and (21)) together with their initial conditions, \( \frac{\partial p}{\partial p} \bigg|_{t=0} = I_{m,n} \) and \( \frac{\partial p}{\partial x_0} \bigg|_{t=0} = 0 \), imply that the state transition tensors, \( \frac{\partial p}{\partial p} = I_{m,n} \), \( \frac{\partial p}{\partial x_0} = 0 \) for all time, which as advertised, comes out to hold automatically. Using these relations in Eqs. (18) and (19), we obtain Eqs. (7) and (5). The generalization of Eq. (13) to higher order is straightforward and the augmented state space approach yields tensors of all combinations and mixed types. However, we pay a large price for this convenience by increasing the dimensionality due to this padding of the state space with the parameter vector. Practical implementations require a careful attention to handling the intrinsic sparsity of the products appearing in Eq. (13) and its higher order generalizations. Alternatively, we can simply recover equations analogous to Eqs. (20) and (21), implicitly tracking pointers that “know” about this structure. These advances are a subject of discussions for a different future communication.

C. High-Order Derivatives of a scalar Composition of Functions

For more than a century Faá di Bruno’s formula [6-9] has provided a closed-form solution for evaluating scalar composite function calculations for implicit rate calculations. His formula is expressed in the form: The \( n \)th \( x \)-derivative of a function \( u = \phi(y) \), where \( y = \psi(x) \), is written as
\[
\frac{d^n}{dx^n} \phi(\psi(x)) = \\
\sum_{b_1, b_2, \ldots, b_n} \frac{n!}{b_1!b_2!\cdots b_n!} \phi^{(k)}(\psi(x)) \left( \frac{\psi'}{1} \right)^{b_1} \left( \frac{\psi''}{1 \cdot 2} \right)^{b_2} \left( \frac{\psi'''}{1 \cdot 2 \cdot 3} \right)^{b_3} \cdots \left( \frac{\psi^{(n)}}{1 \cdot 2 \cdots n} \right)^{b_n}
\]

where the summation extends over all the positive integer solutions of the constraint equations

\[
b_1 + b_2 + b_3 + \cdots + b_n = k \\
b_1 + 2b_2 + 3b_3 + \cdots + nb_n = n
\]

The scalar nature of Eq. (22) allows the formula to be written a very compact form. Vector generalizations of Faá di Bruno’s formula are extremely complicated to implement [10-12]. This paper replaces the combinatorial calculation for Faá di Bruno’s formula with a mixed symbolic-numeric algorithm. Integer computational data structures are presented that permit a very simple integer recursive algorithm for evaluating vector generalizations of Eq.(22). The proposed computer-aided recursive algorithm closely mimics a hand-based derivation approach.

### III. Recursive Integer Algorithm for the Multivariable Vector-Valued Implicit Rate Calculations

The Faá di Bruno algorithm of Eqs. (22) and (23) is only useful for scalar applications. A new algorithm is required for the vector system generalizations considered in this paper. To this end, a recursive integer array notation is introduced that directly mimics how analysts derive implicit derivative equations. The algorithm is greatly simplified by introducing an array-of-arrays data structure for supporting sequential processing for the derivative calculation. A multi-component integer algorithm is used to define the derivative calculation: three operational rules govern the transformation of the integer code into a new code for handling the next higher-order derivative calculation. No simplifications are made to exploit potential opportunities for efficiency because the entire calculation is a one-time pre-processing step, where simplicity and robustness are far more important than optimizing computer memory utilization.

#### A. Recursive Integer Algorithm for High-Order Implicit Derivative Calculations

The goal of the algorithm is to develop (1) an integer code that specifies recursively the individual terms in each derivative calculation, and (2) rules for processing individual terms in derivative expansions. A simple calculation is presented to highlight the key ideas. To this end, consider the calculation for the chain rule derivative of the second-order initial condition tensor that, upon taking total derivatives with respect to initial conditions and using the implicit function theorem, yields the third-order initial condition tensors as

\[
f_{ij} = f_{ijr}x_{r,ij} \\
f_{ijk} = f_{ijkp}x_{p,ijk} + f_{ijs}x_{s,ijk} + f_{irs}x_{r,ijk} + f_{rj}x_{r,ijk} + f_{r,s}x_{s,ijk}
\]

The derivative notation, for initial condition sensitivity problems, is interpreted to mean \( f_{ys} = \partial^2 f / \partial x_y \partial x_s \) and \( x_{p,k} = \partial x_p / \partial x_{ok} \). Only two unique codes appear at second order. The first term consists of the product of three terms and yields three terms at third order; whereas, the second term consists of the product of two terms and yields two terms at third order. The following three rules enable the transformation of (n-1)-order results to n-order results.
The calculations of Eq. (24) are easily replaced with the following integer codes:

\[
f_2 := \{ [2 \ 1 \ 1 \ 0], \ [1 \ 2 \ 0 \ 0] \}
\]

\[
f_3 := \{ [3 \ 1 \ 1 \ 1], \ [2 \ 2 \ 1 \ 0], \ [2 \ 1 \ 2 \ 0]; \ [2 \ 2 \ 1 \ 0], \ [1 \ 3 \ 0 \ 0] \}
\]

where RULES 1,2,3 have been applied. These observations suggest a very simple algorithm for building arbitrarily high order sensitivity models. The integer calculation is presented as the flow diagram for manipulating the sub-array components.

The quantity \( P_{s(i)}(K) \) denotes the \( K^{th} \) sub-array of the integer codes for derivative order \( i-1 \). The integer function routine Locate_Zero returns the location of the first zero value encountered when testing left-to-right, this information defines the complexity of the current term in the expansion. For example, assuming that the integer array is dimensioned as \( a(0:4) \), the Function routine returns the following value for the input array \([5,1,4,0,0,...]\):
and the 0 location denotes the function gradient order. The general flow diagram for the overall algorithm follows as

1. **Integer Data Structure**

   An array-of-arrays data structure is introduced to capture the number of codes generated at each tensor order.

<table>
<thead>
<tr>
<th>Expansion Order</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sub-Array Dimension</td>
<td>1</td>
<td>2</td>
<td>5</td>
<td>15</td>
<td>552</td>
<td>203</td>
<td>877</td>
<td>4140</td>
</tr>
</tbody>
</table>

   The number of sub-arrays required at each expansion order is governed by an integer sequence known as Bell numbers [14-20]. The Bell numbers describe the number of ways a set with \( n \) elements can be partitioned into disjoint, non-empty subsets. The following closed-form equation is available for predicting the number of Bell numbers in the form of a truncated sum of Stirling numbers of the second kind [21-23]:

   \[
   s(n) = \sum_{i=1}^{7} \frac{n^n}{i!} = \frac{7^n}{5040} + \frac{5^n}{240} + \frac{4^n}{72} + \frac{3^n}{16} + \frac{2^n}{60} + \frac{5^n}{144}
   \]  

   (25)

   where \( n \) denotes the order of the expansion. The integer codes for an \( n^{th} \) order expansion are stored in an array-of-arrays data structure of the form

   \[
   P = \left\{ \left[ \begin{array}{c} \text{codes of order } n(1) \\ \text{codes of order } n(2) \\ \text{codes of order } n(3) \\ \cdots \\ \text{codes of order } n(n) \end{array} \right] \right\}
   \]

   where the array data structure for the \( k^{th} \) order expansion terms are given by

   \[
   \left[ \begin{array}{c} \text{codes of order } k(1) \\ \text{codes of order } k(2) \\ \text{codes of order } k(3) \\ \cdots \\ \text{codes of order } k(m) \end{array} \right] = \left[ \begin{array}{c} i_1 \\ \cdots \\ j \\ \cdots \\ m \end{array} \right]_{s(k)}
   \]

   Then the \( j^{th} \) sub-array data structure of \( \left[ \begin{array}{c} \text{codes of order } k(1) \\ \text{codes of order } k(2) \\ \text{codes of order } k(3) \\ \cdots \\ \text{codes of order } k(m) \end{array} \right] \) is given by
\[ [ ]_j = \left[ \nabla^l f \text{ or } \nabla^{l-1} f \text{ or } \cdots \text{ or } \nabla f, \frac{\partial^l x}{\partial \xi^l}, \frac{\partial^{l-1} x}{\partial \xi^{l-1}}, \cdots, \frac{\partial x}{\partial \xi} \right]; \sum_{r=1}^{l} k_r = l \] (26)

where the dimension of each sub-array is \( nx(l+1) \).

Many repetitions of terms appear in the sub-arrays because all terms in the derivative expansion individually handled. This algorithm memory trade is important because it eliminates the requirement for solving the vector-based integer constraint generalization of Eq. (23). This calculation is a pre-processing step that is only preformed one time. After completing the derivative calculations, a simple sorting routine identifies the unique integer codes and tracks the number of repetitions for each code. After identifying the unique codes at each expansion level the number of unique terms is found to scale as \( 2^{(\text{order}-1)} \). For example, at fourth order the algorithm finds the repeat factors and unique codes presented in Figure 2.

<table>
<thead>
<tr>
<th>Repeat Factor</th>
<th>Integer Codes</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \nabla^l f )</td>
<td>( k_1 ) ( k_2 ) ( k_3 ) ( k_4 )</td>
</tr>
<tr>
<td>1</td>
<td>4</td>
</tr>
<tr>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>1</td>
<td>3</td>
</tr>
<tr>
<td>3</td>
<td>2</td>
</tr>
<tr>
<td>3</td>
<td>2</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

Figure 2: Unique 4th Order Codes

### IV. String Manipulation Subroutine Generation

The implied tensor contraction operations are implemented by introducing a Fortran-based string manipulation program that translates the transition tensor sub-codes into FORTRAN 95/2003 software for numerical calculations. For example, the transformation from analytic model follows the path defined by

\[
2\nabla^2 f \cdot \frac{\partial^2 x}{\partial p \partial p} \cdot \frac{\partial x}{\partial p} \Rightarrow 2[2 2 1 0] \Rightarrow 2*(f\%t2 .dot. x\%t2) .dot. x\%t1
\] (27)

Analytic Model Integer Code String Code Generated Fortran

The objects defined by the integer code are tensors of order one higher than the integer code, because the equation of motion is a vector of dimension \( nx1 \). The string code variables have the FORTRAN definitions: \( f\%t2 \) denotes the 3rd order tensor \( \nabla^2 f \); \( x\%t2 \) denotes the 3rd order tensor \( \frac{\partial^2 x}{\partial p \partial p} \); and \( x\%t1 \) denotes the 2nd order tensor \( \frac{\partial x}{\partial p} \). The .dot. operator is user defined and allows tensor computations for tensors of different orders, this capability is provided in the OCEA computational differentiation package [2,3]. The string manipulation program writes the repeat factor and integer codes to string variables using character variables.

Future program enhancements will consider array algebra extensions, as well as opportunities for exploiting the block structures appearing in \( \nabla f, \nabla^2 f, \cdots \) and \( \nabla F, \nabla^2 F, \cdots \).

No analyst intervention is required for modeling arbitrarily high-order tensor transition differential equations, subject to execution time and memory limitations. Only the three rules following Eq. (24) are required. An
executable FORTRAN subroutine is automatically generated, see Figure 2. This is particularly important for higher-order problems where many terms are required for building the complete tensor differential equation model. While the curse of dimensionality has not been completely eliminated, it has been largely negated from the point of view of user derivation, coding, and debugging.

A. Language Impacts for Higher-Order Problems

A near-term limitation of this approach is that standard FORTRAN only supports seven dimensions, which effectively limits the expansion order to sixth order. One dimension is lost to the basic requirement that the equation of motion is a vector. Workstations provide additional array dimension extensions that can increase the expansion order if required. FORTRAN 2008 eliminates these constraint and increases the number of arrays dimensions to thirteen. Future experience in applying related uncertainty propagation algorithms will establish the need for array dimensions beyond the currently available seven. Obviously, many practical problems can be accommodated with dimensions of 13 or less.

V. Generalized Scalar Integration Variable

An $n^{th}$ order expansion requires that the state and $(n-1)$ transition tensor differential equations be numerically integrated. The standard approach maps every vector and tensor to a vector array, which requires many wasted packing and unpacking operations. A simple object-oriented solution for this problem consists of defining a user-defined data type consisting of a scalar that is defined as an abstract compound data object. The packing/unpacking operations are eliminated by introducing the following scalar variable

$$\alpha^1_{|x|1} = \left\{ t \left\langle \frac{x_0}{n_{x_1}} \right\rangle_{n_{x_0}} \Phi_1 \left\langle \frac{0}{n_{x_{x_0}}} \right\rangle_{n_{x_{x_0}}} \cdots \Phi_n \left\langle \frac{0}{n_{x_{x_{n-1}}}} \right\rangle_{n_{x_{x_{n-1}}}} \right\}$$

which contains hidden values for the state and transition tensors. The definition of a scalar compound data structure simplifies data passing and processing within the numerical integration algorithm. Operator-Overloading is used to define multiplication by a scalar and scalar addition the user-defined data type; thereby, enabling standard integration algorithm operations. The integrator is passed the time derivative of this equation as

$$\dot{\alpha^1}_{|x|1} = \left\{ t \left\langle \frac{f}{n_{x_1}} \right\rangle_{n_{x_0}} \Phi_1 \left\langle \frac{0}{n_{x_{x_0}}} \right\rangle_{n_{x_{x_0}}} \cdots \Phi_n \left\langle \frac{0}{n_{x_{x_{n-1}}}} \right\rangle_{n_{x_{x_{n-1}}}} \right\}$$

The state and transition tensor values are stored in Eq. (29) by introducing using the component selector character percent % symbol as follows

$$\alpha\% f = f ; \quad \alpha\% p1 = \Phi_1 ; \quad \alpha\% p2 = \Phi_2 ; \cdots ; \alpha\% pn = \Phi_n$$

The initial condition is given by

$$\alpha(t_0)_{|x|1} = \left\{ t_0 \left\langle \frac{x_0}{n_{x_1}} \right\rangle_{n_{x_0}} \left\langle \frac{I}{n_{x_{x_0}}} \right\rangle_{n_{x_{x_0}}} 0 \cdots 0 \right\}$$

Leading to a scalar integral of the form

$$\alpha = \alpha_0 + \int_0^T \dot{\alpha}(\tau) d\tau$$

VI. Applications

As example demonstrations of the state and parameter state transition tensor computations we consider two sets of problems from optimal control theory. The first set involves sensitivity calculations for the state feedback solutions to the finite time optimal control problems. Examples in this class of problems have been discussed in a recent paper by the authors [24]. We revisit the examples in this context to demonstrate the utility of the state parameter transition tensor algorithm presented here in with the more advanced data structures and array handling. The second set involves the sensitivity calculations of an open loop optimal control problem. Sensitivity calculations of the transition tensors for the open loop problem lead to a classical method of solution to the two point boundary value
A. Optimal Closed Loop Control and the State Feedback Gain Sensitivity Calculations

Consider the scalar problem of minimizing the cost function [27-29]

$$\min J = \frac{1}{2} s, x^2 (t_f) + \frac{1}{2} \int_{t_0}^{t_f} q x^2 (\tau) + r u^2 (\tau) \, d\tau$$

subject to:

$$\dot{x}(t) = a^0 x(t) + bu(t)$$

(31)

together with \(x(0)\) given, \(t_f\) fixed. The solution to this problem is a linear state feedback control signal, \(u(t) = -\frac{b}{r} s(a^0, t)x(t)\), where the time varying state feedback gain is calculated by solving the Riccati differential equation backwards in time.

$$\dot{s} = -2a^0 s + s^2 \frac{b^2}{r} - q$$

(32)

with final conditions \(s(a^0, t_f) = s_f\). The gains calculated via the solution of this famous nonlinear differential equation are typically stored for use by the controller to operate the plant in forward time. Notice however that, even for this simple problem, with a small change in plant parameters (say \(a = a^0 + \delta a\)), the state feedback gains need to be recomputed if the performance of the system cannot be sacrificed. Alternatively consider an implicit function approach to the problem, where the state-feedback gain is an implicit function of the varying plant parameters and so is the optimal state evolution, and therefore can be expanded in a Taylor series about the nominal trajectory (in this case the optimal path for the nominal parameter value) as

$$s(a, t) = s(a^0, t) + \left( \frac{\partial s}{\partial a} \right)_{a=a^0} \delta a + \frac{1}{2!} \left( \frac{\partial^2 s}{\partial a^2} \right)_{a=a^0} (\delta a)^2 + ...$$

$$x(a, t) = x(a^0, t) + \left( \frac{\partial x}{\partial a} \right)_{a=a^0} \delta a + \frac{1}{2!} \left( \frac{\partial^2 x}{\partial a^2} \right)_{a=a^0} (\delta a)^2 + ...$$

(33)

To determine the coefficients required in the above expansion, the differential equations governing the closed loop together with the gain differential equations need to be considered. We demonstrate the utility of such a calculation with an example problem with the nominal parameters:

$$a^0 = 1, b = 1, r = 2, q = 20$$

$$t_f = 3 \text{sec}, s_f = 20$$

(34)

Experiencing a possible variation in the plant parameters in the range, \(\delta a \in [-0.4, 0.5]\) about \(a^0 = 1\). The gains predicted from the sensitivity calculations presented are plotted in Fig. 1 along with the exact solutions calculated for the corresponding parameters. It can be noted that the predictions and the exact calculations agree to plotting accuracy. Thus, in lieu of solving many Riccati equations for each \(a\), a single solution of the Ricatti equation and a single computation of the derivatives of the gain solution with respect to \(a\) provides all the neighboring optimal feedback gains. Of course the justification for the new ideas runs much deeper than scalar optimal feedback control gain computation.
Figure 2. State Feedback Gain Variation Predictions Using Implicit Function Theorem (Fixed Time Problem)

For the general case of a multivariable system, the equations governing the evolution of the gain sensitivities with respect to the plant parameters \( \frac{\partial s}{\partial a_{\omega,\eta}} \) are more involved. However, for demonstration purposes, let us now look at the sensitivity differential equations of first order governing the evolution of this variation of the state feedback gain matrices with respect to the matrix of plant parameters. The Riccati differential equation in this case (vector state space) is given as a matrix differential equation,

\[
\dot{S} = -A^T S - SA + SBR^{-1}B^T S - Q
\]

(35)

together with the boundary condition \( S(t_f) = S_f \) which penalizes the state at final time (soft constraint). It is well known that the optimal state feedback gain matrix \( S(t) \) obtained from the differential equation above stabilizes the vector linear system,

\[
\dot{x} = A_0 x + Bu
\]

(36)

with the linear state feedback control law, \( u(t) = -R^{-1}B^T S(t)x(t) \). Using the \( \text{Vec}(\cdot) \) operator which essentially stacks all the columns of a matrix into a giant array (a recent paper by authors presents yet another method of performing these transition tensor calculations based on this representation of tensors; refer [25]) and the identities involving kronecker products [28, 32],

\[
\frac{d \text{Vec} S}{dt} = -\left[I \otimes A_0^T\right]\text{Vec} S - \left[A_0^T \otimes I\right]\text{Vec} S + \left[I \otimes SBR^{-1}B^T\right]\text{Vec} S + \text{Vec} Q
\]

(37)

\[
= -\left[S \otimes I\right]T_N \text{Vec} A_0 - \left[I \otimes S\right]\text{Vec} A_0 + \left[I \otimes SBR^{-1}B^T\right]\text{Vec} S + \text{Vec} Q
\]

where \( T_N \) is a transpose-like linear transformation such that \( \text{Vec} A_0^T = T_N \text{Vec} A_0 \). Although the general high order sensitivities of the state feedback gain matrix with respect to variations in the plant parameters are too complicated to write in a structured fashion, we can immediately observe the similarity of the above vector differential equation to the system in (1) and the parameter transition tensor differential equation can be written immediately using (8) as
\[
\frac{d}{dt} \left( \frac{\partial \text{Vec} S}{\partial \text{Vec} A} \right) = \left( \left[ I \otimes A^T \right] + \left[ A^T \otimes I \right] \right)_{A=A_0 \atop S=S(A_0)} \left( \frac{\partial \text{Vec} S}{\partial \text{Vec} A} \right) - \left( [S \otimes I]T_N + [I \otimes S] \right)_{A=A_0 \atop S=S(A_0)} \tag{38}
\]

The higher order sensitivity calculations lack the obvious structure of the first order equations, and for the sake of compactness of the paper, we do not present the more extensive expressions involving the second order sensitivities associated with the optimal closed loop state feedback gains. However, the utility of the presented sensitivity expressions is obvious for application problems. The neighboring extremal gains need not be determined by solving the Riccati differential equation backwards in time. Gain determination for a neighboring trajectory can be done using the apriori calculated sensitivities with an apriori determined loss of optimality and performance. The implications in practical applications are immediate and obvious. Let us now briefly discuss the sensitivity calculations associated with the open loop optimal control problem.

B. Open Loop Optimal Control Problems: Sensitivity Calculations

We now consider a problem where the phasing information and the departure and arrival times are important to account for. Preliminary results obtained are discussed as follows. The target orbit was obtained by rotating the orbit plane of the asteroid Apophis [30, 31] to be co-planar with the Eccliptic (for simplistic illustration in the current discussion). The earliest launch time frame of interest was assumed to be in January of 2011. To gain initial insight into the physics of the problem and to ascertain convergence uniformly with a single initial guess on the initial costate, we report the results obtained by considering a launch during the first 150 days, starting January 1, 2011, and times of flight to Apophis ranging from 130 through 200 days. The transfer maneuver is considered to be a low thrust transfer, but with allowable impulses at initial and final time. Consequently, we seek a solution that minimizes the impulse requirements at initial and final time and also thrust to steer the spacecraft matching the fixed boundary conditions. The problem considered can be formally stated as,

\[
\min \frac{1}{2} \left( \left( u(t_f) - u_{\text{Asteroid}} \right)^2 + \left( v(t_f) - v_{\text{Asteroid}} \right)^2 \right) + \frac{1}{2} \left( \left( u(t_f) - u_{\text{Earth}} \right)^2 + \left( v(t_f) - v_{\text{Earth}} \right)^2 \right) 
\]

subject to:

\[
\begin{align*}
\dot{r} &= u, \quad \dot{\theta} = \frac{v}{r} \\
\dot{u} &= \frac{v^2}{r} - \frac{\mu}{r^2} + \frac{T \sin(\phi)}{m_0 - \mu} \\
\dot{v} &= -\frac{uv}{r} + \frac{T \cos(\phi)}{m_0 - \mu}
\end{align*}
\tag{39}
\]

and

\[
\Psi_1(t_f) = \theta_{\text{Asteroid}}; \quad \Psi_2 = r(t_f) = r_{\text{Asteroid}}
\]

with initial conditions

\[
\begin{align*}
r(t_0) &= r_{\text{Earth at } t_0}, \theta(t_0) = \theta_{\text{Earth at } t_0}, u(t_0) = 0, v(t_0) = \frac{\mu}{\sqrt{r(t_0)}}
\end{align*}
\]

The necessary conditions for optimality yield:
Hamiltonian

\[ H = \dot{\lambda}_u + \lambda_v \frac{v}{r} + \bar{\lambda}_\omega \left( \frac{v^2}{r^3} + \frac{\mu}{r^2} + \frac{T \sin \phi}{m} \right) + \lambda_{\bar{\omega}} \left( -\frac{uv}{r} + \frac{T \cos(\phi)}{m} \right) \]

Co-state Equations

\[
\begin{align*}
\dot{\lambda}_u &= -\lambda_v \left( \frac{v^2}{r^3} + \frac{2\mu}{r^2} \right) - \lambda_{\bar{\omega}} \frac{uv}{r^2} + \lambda_{\bar{\omega}} \frac{v}{r}, \\
\dot{\lambda}_\omega &= -\lambda_v + \lambda_{\bar{\omega}} \frac{v}{r}, \\
\dot{\lambda}_\bar{\omega} &= -\lambda_v + \lambda_{\bar{\omega}} \frac{2v}{r} + \lambda_{\bar{\omega}} \frac{u}{r} - \frac{\lambda_{\bar{\omega}}}{r}, \\
\end{align*}
\]

Optimal Control

\[
\phi(t) = \tan^{-1} \left( \frac{\lambda_v}{\lambda_{\bar{\omega}}} \right)
\]

along with the boundary conditions, \( r(t_f) = r_{\text{Asteroid}}, \theta(t_f) = \theta_{\text{Asteroid}} \) and the conditions for costates at final time \( \lambda_u(t_f) = u(t_f) - u_{\text{Asteroid}}, \lambda_{\omega}(t_f) = v(t_f) - v_{\text{Asteroid}} \). A nominal solution to the problem along with the simulation parameters is plotted in Fig. 3.

**Figure 3. A Nominal Solution and Simulation Parameters**

Once the optimal control law is determined, the state co-state system of equations together form the set of nonlinear equations required to compute the optimal control. We now list the first order partials required for the computation of the state and parameter transition tensors.
\begin{align*}
\dot{x}_1 &= x_3 \\
\dot{x}_2 &= \frac{x_4}{x_1} \\
\dot{x}_3 &= \frac{x_4^2}{x_1} - \frac{\mu}{x_1^3} + \frac{T s \phi}{(m_0 - \bar{m} t)} \\
\dot{x}_4 &= -\frac{x_1 x_4}{x_1} + \frac{T c \phi}{(m_0 - \bar{m} t)} \\
\dot{x}_5 &= -x_1 \left( -\frac{x_4^2}{x_1^3} + \frac{2 \mu}{x_1^2} \right) - x_8 \frac{x_3 x_4}{x_1^2} + x_6 \frac{x_4}{x_1^2} \\
\dot{x}_6 &= 0 \\
\dot{x}_7 &= -x_6 + x_8 \frac{x_4}{x_1} \\
\dot{x}_8 &= -x_7 + x_1 \frac{2 x_4}{x_1} + x_8 \frac{x_4}{x_1} - x_6 \frac{x_4}{x_1} \\
\end{align*}

(41)

with the understood definitions of the state costate variables in the augmented state space (8 dimensional) as 
\( x_{1\rightarrow4} = [r, \theta, u, v], x_{5\rightarrow8} = [\lambda_r, \lambda_{\theta}, \lambda_u, \lambda_v] \), together with the \( 8 \times 1 \) right hand side nonlinear vector function 
\( f(x_{1\rightarrow8}) \in \mathbb{R}^8 \). This enables us to directly use the transition tensor differential equations presented in the previous sections (ref. equations (18)).

For demonstration purpose, we now formally lay out the second order parameter transition tensor differential equations including the state parameter interaction terms which are integrated to aid in the solution of neighboring open loop problems. The first type of such mixed sensitivity is given by the general expression.

\[
\frac{d}{dt} \left[ \frac{\partial^2 x_k}{\partial p_h \partial p_j} \right] = \frac{\partial f_i}{\partial x_k} \frac{\partial^2 x_i}{\partial x_k \partial x_l} + \frac{\partial^2 f_i}{\partial p_h \partial x_k} \frac{\partial x_k}{\partial x_l} + \frac{\partial^2 f_i}{\partial p_h \partial p_j} \frac{\partial x_k}{\partial x_l} \frac{\partial x_l}{\partial p_j} \tag{42}
\]

Together with the second order sensitivity with respect to pure parameters given by

\[
\frac{d}{dt} \left[ \frac{\partial^2 x_k}{\partial p_h \partial p_j} \right] = \frac{\partial f_i}{\partial x_k} \frac{\partial^2 x_i}{\partial x_k \partial x_l} + \frac{\partial^2 f_i}{\partial p_j \partial x_k} \frac{\partial x_k}{\partial x_l} + \frac{\partial^2 f_i}{\partial p_j \partial p_j} \frac{\partial x_k}{\partial x_l} \frac{\partial x_l}{\partial p_j} + \frac{\partial^2 f_i}{\partial p_h \partial p_j} \frac{\partial x_k}{\partial x_l} \frac{\partial x_l}{\partial p_j} \tag{43}
\]

Numerical simulations are carried out using a Newton method that incorporates the state parameter transition tensors to solve the two point boundary value problem. To reduce the length of this manuscript, the details of the formulation of this modified Newton’s method are ruled beyond the scope of the current paper and will be reported separately. Sample solutions (initial and final impulse requirements) of trajectories departing January 1st of 2011 through the first 250 days for flight times ranging from 130 up-until 200 days are plotted in Figs. 4 (Departure Impulse required) and 5 (Arrival Impulse required).
Figure 4: Departure Impulse Requirement for a Hybrid Rendezvous to an elliptical orbit
Figure 5: Arrival Impulse Requirements for the Hybrid Rendezvous to an elliptical orbit

In the plots, the darker region indicates higher than a cutoff impulse requirement. Sample plot of implicit approximated averaged errors (for a detailed discussion on the weighted averaging concept, please refer to discussion in [24]) is provided in Fig. 7. Fig. 6 plots a time of flight (treated as parameter) first order sensitivity extrapolated trajectory and compares it with the true optimal solution. The time of flight difference between the trajectories was 1 day and it can be seen that the trajectories are visually identical. However averaging as in the case of the thrust parameter increases the number of accurate significant digits by over 2. The shaded region near the top of Figures, 4,5 is very interesting. Thinking about the physics of this problem, it is apparent that a region must exist for each prescribed thrust magnitude, where zero initial and final impulses are required. Since our performance is
the norm of the initial and final impulses, we might expect some difficulty when we attempt to minimize a cost function that has a finite region wherein the cost function can be driven to zero exactly. Put another way, a unique solution does not result from this formulation when zero impulses at both ends are possible.

**Figure 6: First order sensitivity extrapolation (w.r.t time of flight parameter) solution versus the true optimal solution: Agreement to plotting accuracy.**
Figure 7: Errors Incurred due to Implicit Approximation of the neighboring extremal Trajectories for variations in the Thrust parameter (First order Taylor expansion of the implicit derivatives used. With out averaging errors of initial costates and corresponding velocity mismatches ~ 1e-3, With averaging (presented in [24] errors ~ 1e-5). In both cases the “Pork Chop” plots for the 1% perturbed thrust values are recovered to within plotting accuracy.

Figure 8: Initial Co-state Extremal Field Maps.
Figure 9: Costate First Order Sensitivities (With Respect to the Thrust Parameter)

Figure 10: Costate Sensitivities with respect to the Time of Flight (parameter)
VII. Conclusion

The paper provides strategies to perform efficient computations of the state and parameter transition tensors of arbitrarily high orders. Several tools to make the efficient computations possible are reviewed and the limitations are outlined. A repetition counting strategy based on a hybrid string integer array manipulation scheme, bypassing the vector versions of the Faà di Bruno’s formula (which was shown to be more expensive) is presented. An array-of-arrays data structure is introduced to track the derivative terms arising during an automatic population of the tensor differential equations, while simultaneously assembling a generalized state space model for integrating them. Pell exponential numbers keep track of the total number of implicit derivative calculations performed at each expansion order in tandem with the repetition counts provided by the Faà di Bruno’s formula. Example applications to sensitivity calculations of open loop and feedback solutions to optimal control problems conclude the paper. The results obtained and the algorithms discussed in the paper form a basis for optimism as the next generation tools in solution of modern guidance navigation and control problems involving high nonlinearity, large uncertainty and high dimensionality.
Appendix A
Second through fourth order initial condition and parameter state transition tensors.

Higher-order analytic versions of Eqs. (4) and (7) are presented using tensor index notation. The initial condition state transition tensors follow as:

\[ \dot{\Phi}_1 = f \cdot \Phi_1 \]
\[ \dot{\Phi}_2 = f \cdot \Phi_2 + 2f \cdot \Phi_1 \cdot \Phi_1 \]
\[ \dot{\Phi}_3 = f \cdot \Phi_3 + 2f \cdot \Phi_2 \cdot \Phi_1 + 2f \cdot \Phi_1 \cdot \Phi_2 + 3f \cdot \Phi_1 \cdot \Phi_1 \cdot \Phi_1 \]
\[ \dot{\Phi}_4 = f \cdot \Phi_4 + 3f \cdot \Phi_3 \cdot \Phi_1 + 2f \cdot \Phi_2 \cdot \Phi_2 + 3f \cdot \Phi_1 \cdot \Phi_3 + 3f \cdot \Phi_1 \cdot \Phi_1 \cdot \Phi_2 + 3f \cdot \Phi_1 \cdot \Phi_1 \cdot \Phi_1 \cdot \Phi_1 \]

(44)

The parameter state transition tensors follow as:

\[ \dot{\Theta}_1 = \nabla f \cdot \Theta_1 + f_p \]
\[ \dot{\Theta}_2 = \nabla f \cdot \Theta_2 + \nabla^2 f \cdot \Theta_1 \cdot \Theta_1 + 2\nabla f_p \cdot \Theta_1 + f_{pp} \]
\[ \dot{\Theta}_3 = \nabla f \cdot \Theta_3 + 2\nabla^2 f \cdot \Theta_2 \cdot \Theta_1 + \nabla^2 f \cdot \Theta_1 \cdot \Theta_2 + 3\nabla f_p \cdot \Theta_2 + \nabla^3 f \cdot \Theta_1 \cdot \Theta_1 \cdot \Theta_1 + 3\nabla f_p \cdot \Theta_1 + 3f_{ppp} \]
\[ \dot{\Theta}_4 = \nabla f \cdot \Theta_4 + 3\nabla^2 f \cdot \Theta_3 \cdot \Theta_1 + \nabla^2 f \cdot \Theta_1 \cdot \Theta_3 + 4\nabla f_p \cdot \Theta_3 + 3\nabla^2 f \cdot \Theta_2 \cdot \Theta_2 + 3\nabla^3 f \cdot \Theta_2 \cdot \Theta_1 \cdot \Theta_1 + 2\nabla^3 f \cdot \Theta_1 \cdot \Theta_2 \cdot \Theta_1 + \nabla^3 f \cdot \Theta_1 \cdot \Theta_1 \cdot \Theta_2 + 4\nabla^2 f_p \cdot \Theta_1 \cdot \Theta_2 + 8\nabla^2 f_p \cdot \Theta_2 \cdot \Theta_1 + 6\nabla f_{ppp} \cdot \Theta_2 + \nabla^4 f \cdot \Theta_1 \cdot \Theta_1 \cdot \Theta_1 \cdot \Theta_1 + 4\nabla^3 f_p \cdot \Theta_1 \cdot \Theta_1 \cdot \Theta_1 + 6\nabla^2 f_{ppp} \cdot \Theta_1 + 4\nabla f_{pppp} \cdot \Theta_1 + f_{ppppp} \]

(45)

Note that the special parameter state transition tensor, being shown in (45) is for illustrative purposes only. In general, this is contained in the augmented state (parameter augmented) transition tensors of the type (44).

Two assumptions have guided this derivation of the transition tensors. First, the order dependence of the contraction operations is important. Second, mixed state and parameter partials are possible. In both cases it follows that the number of terms increases rapidly. Equation (42) is an example of the mixed state and parameter
partials and the systematic definition of the augmented state variable and looking at the state transition tensors in this general setting (the cascade of differential equations (44), as was pointed out earlier includes all these terms).

References

13. Junkins, J. L., Lecture Notes on AERO – 624, Celestial Mechanics, Center for Mechanics and Control, Texas A&M University, College Station, TX, 77840.

American Institute of Aeronautics and Astronautics