NEW METHODS FOR ESTIMATION, MODELING AND VALIDATION OF DYNAMICAL SYSTEMS USING AUTOMATIC DIFFERENTIATION

A Dissertation

by

DANIEL TODD GRIFFITH

Submitted to the Office of Graduate Studies of
Texas A&M University
in partial fulfillment of the requirements for the degree of

DOCTOR OF PHILOSOPHY

December 2004

Major Subject: Aerospace Engineering

NEW METHODS FOR ESTIMATION, MODELING AND VALIDATION OF DYNAMICAL SYSTEMS USING AUTOMATIC DIFFERENTIATION

A Dissertation

by

DANIEL TODD GRIFFITH

Submitted to Texas A&M University
in partial fulfillment of the requirements
for the degree of

DOCTOR OF PHILOSOPHY

Approved as to style and content by:

John L. Junkins (Chair of Committee)	
Srinivas R. Vadali (Member)	John E. Hurtado (Member)
Alan B. Palazzolo (Member)	Walter E. Haisler (Head of Department)

December 2004

Major Subject: Aerospace Engineering

ABSTRACT

New Methods for Estimation, Modeling and Validation of Dynamical

Systems Using Automatic Differentiation. (December 2004)

Daniel Todd Griffith, B.S., Morehead State University;

B.S., University of Kentucky;

M.S., University of Kentucky

Chair of Advisory Committee: Dr. John L. Junkins

The main objective of this work is to demonstrate some new computational methods for estimation, optimization and modeling of dynamical systems that use automatic differentiation. Particular focus will be upon dynamical systems arising in Aerospace Engineering. Automatic differentiation is a recursive computational algorithm, which enables computation of analytically rigorous partial derivatives of any user-specified function. All associated computations occur, in the background without user intervention, as the name implies. The computational methods of this dissertation are enabled by a new automatic differentiation tool, OCEA (Object oriented Coordinate Embedding Method). OCEA has been recently developed and makes possible efficient computation and evaluation of partial derivatives with minimal user coding. The key results in this dissertation details the use of OCEA through a number of computational studies in estimation and dynamical modeling.

Several prototype problems are studied in order to evaluate judicious ways to use OCEA. Additionally, new solution methods are introduced in order to ascertain the extended capability of this new computational tool. Computational tradeoffs are studied in detail by looking at a number of different applications in the areas of estimation, dynamical system modeling, and validation of solution accuracy for complex dynamical systems. The results of these computational studies provide new insights and indicate the future potential of OCEA in its further development.

To my parents

Dan and Barbara

and

my brothers and sisters

Christie, Ryan and Andy

ACKNOWLEDGMENTS

Setting upon any of life's journeys requires the support and guidance of many people. Many people have shared equally the successes and disappointments which have arisen while on the path to completing this dissertation. Of course, there are many who have made technical contributions that aided me in meeting my educational objectives and, more specifically to completing this work, many have made meaningful contributions. I have benefited greatly, as well, from the support of many people in my personal life who have done so much to shape me into who I am today.

I am greatly indebted to my graduate advisor, Dr. John Junkins. I thank him for welcoming me to study with him at Texas A&M in August 2000. I, of course, acknowledge the technical contributions he has made toward completing this dissertation. He has been a source of knowledge on many subjects, and has given me a glimpse into the fundamental principles which unify many technical areas. He has been a source of inspiration to me whether it be in the classroom, in a research meeting or in passing in the hallway, and gave his time freely to me whenever it was needed. I especially appreciate the guidance and words of encouragement he provided during my toughest times and most misguided moments. I appreciate the financial support he provided over my first year of study and during the latter months while I completed this dissertation. My earnest hope is that I take a great deal from my experiences with Dr. Junkins to my professional life since I feel the impact he has had on me is significant in many ways, not only for the technical knowledge and the large-scale perspective I have gained from him, but also for the passion he has for his work and students. I could not be more pleased with the time I spent with him at Texas A&M.

My parents, Dan and Barbara, have always been my biggest asset in their constant support, by the example they set, and for the drive and confidence they have instilled in me my entire life. They have, most importantly, tempered my ambition by demonstrating the balance in life that I strive for in order to be truly happy and completely successful.

My most joyous moments have come since meeting the most special person in my life, my fiancé Loraine. Her support has inspired me to work diligently and focus on the completion of this dissertation. She has sacrificed a great deal by understanding the importance of this dissertation to me and has supported me in more ways than I could ever expect or deserve. I could not have completed this without her love and support.

I acknowledge the tremendous impact that Dr. James Turner has had on technical merit of this dissertation. I thank him for the countless instances he has given his time to share his technical expertise. I appreciate his sense of humor and the passion he has for his work. It was entirely my pleasure to have the opportunity to work with him.

I am very thankful for the insights and valuable time of my committee members Dr S. Rao Vadali, Dr. John E. Hurtado, and Dr. Alan Palazzolo. Dr. Vadali has provided me with a fine example in the thoughtfulness of his classroom teaching. I am thankful for the time he devoted to me as I learned much about optimization. I am indebted to Dr. Hurtado for many inspiring lectures and conversations regarding advanced topics in dynamics. I am thankful for his practical, informal style of teaching and mentoring.

I am thankful for the support of the American Society for Engineering Education in awarding three years of financial assistance to support my doctoral studies through the National Defense Science and Engineering Fellowship Program from 2001-2004.

The Aggie campus is unlike any other in this nation. I appreciate the caring nature, togetherness and values of the people of Texas A&M and College Station, Texas. In my time in College Station, I am quite thankful for the community of St. Mary's Catholic Church, especially Father Mike Sis, for providing me with great words of wisdom and perspectives on living. I shared many good times with many new friends and colleagues at Texas A&M. I am thankful to have had two wonderful friends as office mates. Chad Searcy has been and is a friend who has truly enhanced my experience at Texas A&M by showing me the ropes when I arrived in College Station. I appreciate his kindness and generosity, and our many discussions on politics. Rajnish Sharma has been an exceptional person to share a workspace with in my last days at Texas A&M. I appreciate his warm personality and have taken a great deal from his relaxed attitude regarding life and work.

I would like to acknowledge two friends who have made the times away from my desk very enjoyable. Lisa Biggs has been a great friend, and someone who has many times given me a good perspective on what is truly important. I thank her for her friendship. I am also very thankful for the times I have spent with Andy Sinclair. I had the pleasure of enjoying many discussions with him on Southeastern Conference (SEC) sports. Although he leans toward the University of Florida Gators (and I the University of Kentucky Wildcats), I am very happy for the sense of home I received from our discussions. I am also very happy for the significant technical discussions we had, many of which led us to hours of research on the dynamics encountered in the game of golf. The match play feature of the Tiger Woods golf game executed on the PlayStation II game system proved to be an ideal testbed for this study.

I would also like to thank Art and Rocio Fano for welcoming me into their family. Art and Rocio have made the last year of my studies very enjoyable through their warmth and thoughtful attention.

There are many others who have I have not mentioned here by name who I would like to thank. I sincerely thank them as well.

TABLE OF CONTENTS

LIST OF TABLES

TABLE Page

LIST OF FIGURES

FIGURE Page

FIGURE Page

CHAPTER I

INTRODUCTION

The development of new technology is driven by a number of highly coupled factors. These include development of new theory and ideas, advances in the ability to better observe the characteristics of our natural surroundings and our man-made devices, and development of new computational algorithms, software, and hardware. These are intimately coupled since advances in one area can impact one or all of the others by enabling a solution or providing the stimulus for the development of a new technology area. Certainly, the invention and subsequent development of the computer; for example, has given scientists and engineers an incredible range of possibilities for testing new ideas on any scale whether it be very small or very large, very simple or very complex, while vastly reducing the computational time needed for solving a given problem.

The focus of this work is new solution approaches which are enabled by the development of a new computational tool. New ideas have been implemented through the use of automatic differentiation for solving problems pertinent to many disciplines of science and engineering, in particular those arising in Aerospace Engineering. Automatic differentiation is a computational approach which enables partial derivatives of any user-specified function to be computed in the background without user

The journal model is the *Journal of Guidance, Control, and Dynamics*.

intervention, as the name implies. A new automatic differentiation tool, OCEA (Object oriented Coordinate Embedding Method), has been recently developed which shows promise for efficient computation and evaluation of partial derivatives. For a rather arbitrary sequentially substituted set of functions, coded in FORTRAN 90, OCEA invokes operator overloading to perform up to fourth-order partial differentiation with automatic background computations. The key results in this work are computational approaches for typical problems in modeling, estimation an control of dynamical systems enabled by OCEA. Both the algorithms and computational results are presented.

Several standard problems in the field are studied in order to establish judicious ways to use OCEA. Motivated by the ease of automatic differentiation, several new approaches for these problems are developed and evaluated. Important computational issues are studied in detail by looking at a number of different applications in the areas of modeling, estimation, control, and validation of solution accuracy for complex dynamical systems. The results of these computational studies provide new insights and indicate the potential of OCEA in its further development.

The need for computing partial derivatives abounds in all areas of dynamical systems. The task of computing partial derivatives is almost never the focus of a scientific or engineering study, it is usually considered merely a mathematical exercise that must be completed in order to reach the objectives of a particular analysis. However, as model complexity and dimensionality increases, obviously the volume of algebra, calculus, coding and validation poses a severe practical obstacle. If we can be freed from concern about deriving, programming and computing partial derivatives, a scientist or engineer can focus directly on what is most important – analysis and development of new ideas. The acceleration implicit in not having to worry about the partial derivatives is enormous. These qualitative observations provide part of the motivation for this dissertation.

This dissertation is divided into six chapters. This first chapter provides an overview of the main developments. In the second chapter, the state of the art in computerized differentiation is described along with the key features of the new computational tool OCEA.

In Chapter III, we present new ideas related to estimation, optimization and control of dynamical systems. Several novel algorithms are presented. A key result is the introduction of the higher-order state transition matrix and associated differential equations, for a class of nonlinear dynamical systems. In the area of trajectory optimization some new ideas are presented for automating the process of deriving co-state differential equations. Additionally, higher-order algorithms for computing midcourse corrections are introduced.

In Chapter IV, some new insights into modeling of dynamical systems are presented. Producing dynamical models in the form of coupled nonlinear differential equations is a frequent first step for analysis, estimation and control of dynamical systems. The main result of this chapter is a new method for automatically generating and integrating differential equations using automatic differentiation and analytical continuation. The method is applied to linked mechanical systems comprised of rigid or flexible elements. An OCEA enabled approach to solving these differential equations is presented.

In Chapter V, we focus on validating solution accuracy for approximate flexible body dynamical system models. We show that for complex many body systems, exact PDE/ODE equations of motion and the corresponding boundary conditions can be automatically produced using automatic differentiation. Furthermore, we show how to use the method of manufactured solutions, along with OCEA to construct "exact" solutions to the PDE/ODE systems. These special case exact solutions can be used to validate discretized numerical solutions. This development extends the capability of validation efforts by rapidly producing exact representations and special case exact solutions for multibody distributed parameter systems with significant complexity.

Concluding remarks including a description of future work is given in Chapter VI.

CHAPTER II

OVERVIEW OF COMPUTERIZED DIFFERENTIATION

Computer implementation of differentiation is typically accomplished by two distinct approaches – automatic differentiation (AD) and symbolic differentiation. The primary distinction between these approaches is that automatic differentiation invokes the chain rule automatically and partial derivatives are recursively simultaneously derived and numerically evaluated *in the background.* Whereas, symbolic differentiation is performed using symbolic programs in which the derivatives are computed and displayed symbolically. The symbolically derived derivative expressions can, of course, be subsequently coded and used for numerical computation.

2.1 SYMBOLIC DIFFERENTIATION

Most scientists and engineers are very familiar with symbolic differentiation programs. Today's university students routinely use symbolic manipulation programs such as Matlab, Maple, Mathcad, and Macsyma to manipulate equations including automated derivation of partial derivatives[1-4]. These programs are very useful when dealing with complex equations, especially for analytical studies that require viewing the equations. In order to perform a numerical analysis, partial derivatives computed using a symbolic differentiation program must be either hand typed in a computer program or saved in a file. While some degree of automation in programming these automatically derived equations has been achieved, the files typically require editing prior to compilation. One advantage of symbolic differentiation is that the derivative expressions can be viewed by the analyst; however, the resulting mathematical expressions are frequently so large that no physical insight can be gained from the effort.

2.2 AUTOMATIC DIFFERENTIATION

On the other hand, automatic differentiation, especially using the OCEA approach, offers a time and memory optimized differentiation approach where less user intervention is required. In the traditional AD approach, the automatically derived partial derivative equations are output in a machine readable code which is then compiled along with another code that "uses" the derivatives. Research on AD methods has been active since the 1980's. Previously developed AD tools include: 1) ADIFOR, 2) AD01, 3) ADOL-C, 4) ADMIT-1, 5) AUTODERIVE, and 6) OCEA. The first five AD tools typically require interfacing with other programs[5-8], e.g. ADMIT-1 requires interfacing with the Matlab environment. Also, most of these only compute 1st order partial derivatives. A comparison of symbolic and automatic differentiation approaches can be found in Reference 9. The motivation for OCEA was two-fold: (1) How to eliminate the cumbersome file manipulations and interfacing between AD programs and the codes that use the derivatives, and (2) How to achieve generalization to higher-order derivatives.

2.3 OVERVIEW OF OCEA

OCEA[10-15] bypasses the overhead of pre-existing AD approaches and offers tremendous potential since the operator-overloading approach makes it possible to implement the AD approach simply as an extension of any of the existing popular programming languages such as FORTRAN. Additionally, OCEA currently offers 1st through 4th order partial derivative capabilities. OCEA is currently implemented as a FORTRAN90 (F90) extension. The OCEA package is an object-oriented automatic differentiation equation manipulation package. OCEA defines *embedded variables* that represent abstract data types, where hidden dimensions (background arrays) are used for storing and manipulating partial derivative calculations. The chain rule for differentiation has itself been coded using operator-overloading and operates in the background of the user's software. Given a FORTRAN90 code for evaluating virtually any set of differentiable functions, OCEA replaces each scalar variable in the program with a differential n-tuple consisting of the following variables (for a second-order OCEA implementation):

$$f := \begin{bmatrix} f & \nabla f & \nabla^2 f \end{bmatrix} \tag{2.1}$$

where ∇ and ∇^2 denote first- and symmetric 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 derivatives are attached to the scalar variable in a hidden way. The chain rule operates recursively on the n-tuples in the background. The individual objects can be extracted, when needed, 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, overloaded addition and multiplication of the functions $a = a(x)$ and $b = b(x)$ are redefined as follows.

$$a + b := \left[a + b \quad \nabla a + \nabla b \quad \nabla^2 a + \nabla^2 b \right] \tag{2.2}$$

$$a * b := \left[a * b \quad \partial_i \left(a * b \right) \quad \partial_j \partial_i \left(a * b \right) \right] \tag{2.3}$$

Thus the "+" and "*" operators are overloaded so that coding the left side expressions of Eqs. (2.2) and (2.3) causes all the right side computations to be carried out. The underlying vector x is defined (see subsequent discussion) in a way that OCEA knows x is the vector of independent variables and ∇a denotes, for example $\left[\frac{\partial a}{\partial x_1} \cdots \frac{\partial a}{\partial x_n} \right]$. More subtly, if $z_1 = a + b$ and $z_2 = a * b$, then computing $z_3 = z_1 + z_2$ causes the results of Eqs. (2.2) and (2.3) to be propagated efficiently in the background to compute $z_3 = \left[a+b+a*b \quad \partial_i(z_3) \quad \partial_j \partial_i(z_3) \right]$ where $\partial_i(z_3) = \frac{\partial z_3}{\partial x_i}$ and $\partial_i \partial_j (z_3) = \frac{\partial^2 z_3}{\partial x_j \partial x_i}$.

The chain rule operates recursively in the background. Given the function $g = g(x)$, whose partial derivatives are known a priori, OCEA computes the partial derivatives of the function f, which is a function of g, by the following composite function chain rule of differentiation.

$$f(g) := f\left(\left[g, \nabla g, \nabla^2 g\right]\right) := \left[f(g), \frac{\partial f}{\partial g}\nabla g, \frac{\partial^2 f}{\partial g^2}\nabla g(\nabla g)^T + \frac{\partial f}{\partial g}\nabla^2 g\right] \tag{2.4}$$

OCEA utilizes the chain rule of this form to recursively, numerically compute partial derivatives. The function f and its partial derivatives are computed with the a priori known partial derivatives of the function $g = g(x)$, and, of course, the partials of f with respect to g. To illustrate the point, consider a special case in which the function g is defined simply as a set of independent variables ($g = x_1, x_2, \cdots x_n$) then Eq. (2.4) illustrates how to compute the partial derivatives of the function $f(x)$. In this case, the gradient of the ith element of g is defined by

$$\nabla g_i = \nabla x_i = \begin{bmatrix} \delta_{1i} \\ \vdots \\ \delta_{ni} \end{bmatrix} \tag{2.5}$$

where δ_{ij} denotes the standard kronecker delta for *i* = 1, 2, …*n*, and, of course, the Hessian of the ith element of g is given by

$$\nabla^2 g_i = \nabla^2 x_i = \begin{bmatrix} 0 & 0 & \cdots & 0 \\ 0 & 0 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & 0 \end{bmatrix} \tag{2.6}$$

Of course, when using Eq. (2.4) to generate the partials of any nonlinear function then Eqs. (2.5) and (2.6) simply serve to initialize the recursive chain rule operations. Additional details on OCEA can be found in Appendix A.

We now discuss in more detail some issues related to programming in FORTRAN with OCEA. A FORTRAN subroutine enhanced by the OCEA automatic differentiation capability requires that a special statement be typed in the subroutine. This statement is responsible for linking the user subroutine with OCEA, and indicates that the subroutine contains functions which are to be differentiated. Secondly, a new variable type declaration must be utilized in order to specify the problem independent and dependent variables. These are the only two considerations that must be made when "promoting" a standard FORTRAN subroutine to an OCEA-FORTRAN subroutine. The elegance of the OCEA approach lies in the fact that minimal coding is required to promote the standard FORTRAN subroutine. We demonstrate the two aforementioned considerations in the following example. In Fig. 2.1 we show an OCEA-enhanced FORTRAN90 subroutine for computing the function F_3 and its partial derivatives, where the function is defined as $F_1 = x_1^3 + x_2 \quad F_2 = x_2^2 \sin(x_1), \quad and \quad F_3 = F_1 + F_2$.

In Fig. 2.1, we note that the USE statement ("USE EB_HANDLING") links the subroutine to OCEA at compile time. The type declaration ("TYPE(EB)") declares X and F3 as the problem independent and dependent variables. By linking this subroutine with OCEA, the partial derivatives of the function F3 are computed automatically at compile time. When the subroutine is executed, the enhanced OCEA-FORTRAN subroutine evaluates the function F3 and its analytical partial derivatives for a specified numerical value of X. Standard FORTRAN programming, of course, only evaluates the function itself. In this example, we demonstrate that the dependent variables (or equivalently functions) F1, F2, and F3 are coded in a standard way. Additionally, we define double precision (DP) arrays which are used to store the numerically evaluated function F3 and its first and second partials (S_F3, JAC_F3, and HES_F3). This demonstrates the operator-overloading which has been performed on the "=" sign, and all other operators automatically. We set each of these double precision arrays equal to F3, and OCEA automatically detects the dimension of the array on the left hand side. Thus the appropriate function or partial derivative information is easily extracted.

```
SUBROUTINE NONLINEAR_FUNCTION( X, F3 )
     USE EB_HANDLING

!..... ARGUMENT LIST VARIABLES
     TYPE(EB), DIMENSION(2), INTENT(IN):: X
     TYPE(EB), DIMENSION(1), INTENT(OUT):: F3

!.....LOCAL VARIABLES
     REAL(DP), DIMENSION(1):: S_F3
     REAL(DP), DIMENSION(2):: JAC_F3
     REAL(DP), DIMENSION(2,2):: HES_F3

     F1 = X(1)**3.0D0 + X(2)
     F2 = ( X(2)**2.0D0 )*SIN( X(1) )
     F3 = F1 + F2

     S_F3 = F3          ! Extract function
     JAC_F3 = F3        ! Extract Jacobian (2x1)
     HES_F3 = F3        ! Extract Hessian (2x2)

END SUBROUTINE NONLINEAR_FUNCTION
```

Figure 2.1. Example OCEA-FORTRAN subroutine

Additional operations for the standard mathematical library functions, such as trigonometric and exponential functions, are redefined to account for the known rules of differentiation. A list of these functions is given in Table 2.1. In essence, this approach pre-codes, once and for all, all of the partial derivatives for the elementary functions required for any problem, and the chain rule is implemented automatically in background operations that the user neither derives nor codes to complete the simultaneous derivation and evaluation of all needed derivatives. Hidden operator-overloading tools completely free the analyst from the time consuming and error prone tasks of deriving, coding, and validating analytical partial derivative models. Thus, an arbitrary FORTRAN code, by simply adding data type declarations and compiling linked to OCEA, is "promoted" to a generalized code that automatically computes the corresponding arrays of partial derivatives.

Table 2.1. List of overloaded OCEA functions

Common name	**FORTRAN function**
Polynomial to the nth power	x**n
Square root	Sqrt(x)
Sine	Sin(x)
Cosine	Cos(x)
Tangent	Tan(x)
Arc sine	Asin(x)
Arc cosine	Acos(x)
Arc tangent	Atan(y/x)
Arc tangent	Atan2(y,x)
Exponential	Exp(x)
Logarithm	Log(x)
Hyperbolic sine	Sinh(x)
Hyperbolic cosine	Cosh(x)
Hyperbolic tangent	Tanh(x)
Absolute value	Abs(x)
Floor	Floor(x)
Ceiling	Ceil(x)

The functions listed in Table 2.1 account for all of the standard FORTRAN mathematical functions. This list is shown in order to illustrate that all of these standard functions have been redefined, or to put it another way, overloaded in the OCEA environment in order to enable the automatic computation of the partial derivatives of these elementary functions. Any function which is composed of these overloaded elementary functions can now be coded in OCEA, and the resulting partial derivatives of these composite functions are automatically computed and evaluated. It is worth noting that additional high-level operators are also enabled in the OCEA environment including special operators such as the matrix transpose (.T.) and dot product (.DOT.). These operators are extremely useful when defining embedded functions which are composed of vector/matrix terms since these functions can be coded in a direct vector/matrix fashion. These types of expressions are prevalent in the study of dynamical systems, with the additional feature that differentiation of such vector/matrix functions is also enabled.

CHAPTER III

ESTIMATION AND CONTROL OF DYNAMICAL SYSTEMS[*]

In this chapter, new solution approaches are presented for solving problems in estimation and control of dynamical systems. We present new methods for solving problems encountered in parameter optimization, orbit determination, trajectory optimization, and guidance.

The major contributions in this chapter include the utilization of automatic differentiation in deriving and evaluating the necessary partial derivative calculations, and the development of higher-order extensions of the conventional algorithms. These contributions are detailed for implementing the Nonlinear Least Squares and Gaussian Least Squares Differential Correction (GLSDC) algorithms, solving Two-point Boundary Value Problems (TPBVPs), and computing midcourse corrections.

The first section of this chapter contains a review of first-order algorithms. Here we review the computation of first-order sensitivities, in general, for the Nonlinear Least Squares algorithm, and specifically the computation of sensitivities utilizing state transition matrix calculations for solving GLSDC/orbit determination problems and TPBVPs.

In the second section, we detail the calculation of higher-order sensitivities. The concept of higher-order state transition matrix is introduced along with their governing

[*] Portions of this chapter published in References 16 and 17.

differential equations. Advanced optimization algorithms are considered that make use of first- through fourth-order generalized state transition matrix algorithms. We discuss the multilinear reversion of series solution which establishes higher-order mappings and inverse mappings between the state variables. This capability provides the means to develop the higher-order algorithms.

In the third section, new algorithms are presented as higher-order extensions of the first-order methods. Two methods for solving TPBVPs encountered in optimal control are presented. These include a state transition matrix approach and a direct optimization approach. Some related topics such as automatic co-state equation generation are also presented. Higher-order laws for computing midcourse corrections are introduced.

Several examples are presented to demonstrate these methods for each of the algorithms in the fourth section. These examples include a ballistic projectile identification problem, an orbit determination problem, and a trajectory optimization problem.

3.1 REVIEW OF FIRST-ORDER ALGORITHMS

3.1.1 Nonlinear Least Squares

It is a fact of life that most estimation problems are nonlinear. A description of the Nonlinear Least Squares algorithm can be found in many books on estimation[18]. In summary, given a set of observations or measurements and a model for these measurements, the task is to estimate a set of measurement model parameters which best fit the observations. For a nonlinear problem, an iterative solution procedure must be employed. First, a starting guess for the unknown model parameters is supplied, and one iteration of the algorithm produces a correction to the starting guess. This process repeats until the estimate for the model parameters has converged. Of course, the algorithm is considered to have failed if the residual errors increase during the iteration process or the residual error remains essentially unchanged for many iterations of the algorithm.

Typically, one begins in finding the Least Squares estimate $\hat{\boldsymbol{x}}$ which minimizes the following cost function

$$J = \frac{1}{2}\left(\tilde{\boldsymbol{y}} - \boldsymbol{h}(\hat{\boldsymbol{x}})\right)^T W \left(\tilde{\boldsymbol{y}} - \boldsymbol{h}(\hat{\boldsymbol{x}})\right) \tag{3.1}$$

where $\tilde{\boldsymbol{y}}$ is the vector of measurements, $\boldsymbol{h}(\hat{\boldsymbol{x}})$ is an algebraic model for the measurements, and W is an assumed weighting matrix. Taking the gradient of Eq. (3.1) w.r.t. the unknown state $\hat{\boldsymbol{x}}$ results in the necessary condition for minimizing the cost function, leading to:

$$\left(\frac{-\partial \boldsymbol{h}}{\partial \boldsymbol{x}}\right)^T W \left(\tilde{\boldsymbol{y}} - \boldsymbol{h}\left(\hat{\boldsymbol{x}}\right)\right) = 0 \tag{3.2}$$

which essentially defines a root-solving problem. Linearization of the measurement model produces $\boldsymbol{h}(\hat{\boldsymbol{x}}_{k+1}) = \boldsymbol{h}(\hat{\boldsymbol{x}}_{\boldsymbol{k}}) + \left.\frac{\partial \boldsymbol{h}}{\partial \boldsymbol{x}}\right|_{\hat{x}_k} \Delta \boldsymbol{x}_{\boldsymbol{k}}$, which when substituted into the necessary condition produces the well known normal equations.

$$\Delta\hat{\boldsymbol{x}}_k = (H^T W H)^{-1} H^T W \Delta\boldsymbol{y}_k \tag{3.3}$$

where $H = \left.\frac{\partial \boldsymbol{h}}{\partial \boldsymbol{x}}\right|_{\hat{x}_k}$ and $\Delta \boldsymbol{y}_k = \tilde{\boldsymbol{y}} - \boldsymbol{h}(\hat{\boldsymbol{x}}_k)$; solution for $\Delta \hat{\boldsymbol{x}}_k$ based upon local linearization of $\boldsymbol{h}$ thus leads to an iteration algorithm (Gaussian Least Squares Differential Correction). This is the standard method for solution of the nonlinear least squares problem.

The automatic differentiation capability of OCEA is well suited for computing the sensitivities for the Nonlinear Least Squares algorithm. These sensitivities are the partial derivatives of the measurement model with respect to the unknown model parameters. The benefit for the analyst is quite significant when considering the time saved in computing these sensitivities by hand or by symbolic manipulation. As well, the analyst is freed from validating and hard coding these partial derivative expressions. In essence, OCEA simultaneously derives and evaluates the derivative expressions, so they do not need to be viewed, coded, or validated. This capability is particularly advantageous when there are a large number of model parameters to be estimated. For example, given a scalar measurement model with n unknown model parameters, n partials are required for a first-order correction model. If m measurement equations depending on n variables, the Jacobian of first partials requires the derivation and coding of $m*n$ equations (if OCEA is not used). When going to higher-order, the number of partial derivatives to be computed explodes to $m*n^p$ where p is the order of the correction model. Higher-order extensions are discussed in more detail later in this chapter. Changes in the assumed form of the chosen mathematical model (the measurement model) do not result in a significant increase in effort beyond merely specifying the new model since the necessary revisions of partial derivatives are automatically derived and evaluated for each model. This generality is obviously an additional benefit. Furthermore, it motivates the development of general purpose codes for the most common nonlinear estimation problems.

3.1.2 Gaussian Least Squares Differential Correction

The problem of determining the orbits of the heavenly bodies has been studied in great detail for many hundreds of years. Since the 1500's orbital models have been conjectured and fit to measurements. Kepler developed three laws that captured the main features consistent with the precision possible during the early 1600's. Just over 200 years ago, Gauss devised a method for solving the complete orbit determination problem which bears his name, Gaussian Least Squares Differential Correction or GLSDC. This algorithm is of broad utility and is in fact the most commonly applied nonlinear estimation method[18]. The essence of this method, in the context of Gauss' problem, involves estimating the position and velocity (or six other "elements") at some time, usually the initial time in which the first measurement was viewed. Along with a model for the dynamics of the body of interest, the complete best estimate of the orbit can be reconstructed from the estimated initial state. In addition, uncertain model parameters can be estimated. These parameters can include force model parameters such as drag, solar radiation pressure, and gravitational perturbation model constants.

The conventional GLSDC algorithm is reviewed in this section. The first-order GLSDC algorithm is a standard topic in many textbooks on estimation; however, it is below presented in order to proceed logically to the second- and higher-order GLSDC algorithms presented later.

In contrast to Nonlinear Least Squares problems, we choose to fit the measurements to a dynamical model as opposed to a set of algebraic equations. The dynamical model and measurement models are written in general in Eqs. (3.4) and (3.5), respectively.

$$\dot{\boldsymbol{x}}(t) = \boldsymbol{f}(t, \boldsymbol{x}(t)); \quad \boldsymbol{x}_0 = \boldsymbol{x}(t_0) \tag{3.4}$$

$$\boldsymbol{y}(t_j) = \boldsymbol{h}(\boldsymbol{x}(t_j)) \qquad j = 1, 2, ...m \tag{3.5}$$

Here, we consider the state vector, $\boldsymbol{x}$, to contain the position and velocity states as well as any additional unknown model parameters such as those arising from external forces.

Arising from the nonlinear nature of the dynamical and measurement models, an iterative procedure must be employed in determining the unknown parameters. In solving an algebraic parameter optimization (e.g. Nonlinear Least Squares) problem, the unknowns appear explicitly in the sensitivity calculations. However, with GLSDC this is not the case. Here, the sensitivities are computed at each measurement time as follows

$$\begin{aligned} H_j &= \frac{\partial \boldsymbol{h}(\boldsymbol{x}(t_j))}{\partial \boldsymbol{x}(t_0)} \\ &= \frac{\partial \boldsymbol{h}(\boldsymbol{x}(t_j))}{\partial \boldsymbol{x}(t_j)} \frac{\partial \boldsymbol{x}(t_j)}{\partial \boldsymbol{x}(t_0)} \\ &= \frac{\partial \boldsymbol{h}(\boldsymbol{x}(t_j))}{\partial \boldsymbol{x}(t_j)} \Phi(t_j, t_0) \qquad j = 1, 2, ...m \end{aligned} \tag{3.6}$$

and the complete first-order sensitivity matrix is formed by

$$H = \begin{bmatrix} H_1 \\ H_2 \\ \vdots \\ H_m \end{bmatrix} \tag{3.7}$$

where $\Phi(t,t_0)$ is the state transition matrix (first-order). This sensitivity calculation is used in Eq. (3.3) to compute the state update. Now, we discuss how to compute the state transition matrix.

Typically, we compute the state transition matrix by integrating the state transition matrix differential equation. Proceeding toward the classical development for the first-order state transition matrix, we look at linearizing the dynamical model about some reference trajectory

$$\delta\dot{\boldsymbol{x}}(t) = \frac{\partial \boldsymbol{f}(t,\boldsymbol{x}(t))}{\partial \boldsymbol{x}(t)} \delta\boldsymbol{x}(t) \tag{3.8}$$

And we assume that the departure motion is governed by

$$\delta\boldsymbol{x}(t) = \Phi(t,t_0)\delta\boldsymbol{x}(t_0) \tag{3.9}$$

We can substitute Eq. (3.9) into Eq. (3.8) in order to derive the following first-order state transition matrix differential equation.

$$\dot{\Phi}(t,t_o) = \left[\frac{\partial \boldsymbol{f}(t,\boldsymbol{x}(t))}{\partial \boldsymbol{x}}\right]\Phi(t,t_o) \tag{3.10}$$

whose initial condition, following from Eq. (3.9), is the identity matrix.

Alternatively, we can derive this state transition differential equation by beginning with the integral form of Eq. (3.4). We will proceed in this manner in deriving the second- and higher-order state transition matrix differential equations later in this chapter.

The utility of automatic differentiation is even more profound for the GLSDC algorithm. Here, in computing the sensitivities the measurement model must be differentiated as well as the dynamical equations, which must be done in order to derive the state transition matrix differential equations.

The GLSDC algorithm can be summarized as follows:

1. Given measurements, $\tilde{\boldsymbol{y}}$, and an initial guess for the state, $\hat{\boldsymbol{x}}(t_0)$.
2. Integrate the state transition differential equations along with the dynamical equations until the next measurement is available (at time t_j): Eqs. (3.10) and (3.4).
3. Compute the sensitivities for each measurement time: Eq. (3.6).
4. Once final measurement time is reached, compute differential correction: $\Delta\hat{\boldsymbol{x}}_k$.
5. Update state: $\hat{\boldsymbol{x}}_{k+1}(t_0) = \hat{\boldsymbol{x}}_k(t_0) + \Delta\hat{\boldsymbol{x}}_k$.
6. Check convergence.
7. If not converged, then repeat steps 2-6.
8. If converged, then done.

3.2 HIGHER-ORDER GENERALIZED SENSITIVITY CALCULATIONS

The sensitivity calculations needed for the first-order algorithms are typically computed by hand and hard-coded. This path of solving these problems is time-consuming and error-prone. In this section, we present higher-order sensitivity calculations. Without automated methods for deriving partial derivative expressions, we would typically not even consider implementing these solutions due to the sheer number of partial derivative calculations required.

For parameter optimization problems, calculation of sensitivities is straightforwardly accomplished by taking partial derivatives of a set of algebraic equations. On the other hand, calculation of sensitivities for problems such as orbit determination and trajectory optimization, additionally require state transition matrix calculations as was detailed earlier for the first-order GLSDC algorithm. Therefore, in this section we focus on new developments in higher-order generalizations of the first-order state transition matrix concept for use in computing higher-order sensitivities.

3.2.1 Higher-order State Transition Matrix Concepts

First, we begin by focusing on the sensitivity calculations. As was shown earlier, the first-order sensitivities are given are computed for each measurement time by

$$\begin{aligned} H_j &= \frac{\partial \boldsymbol{h}(\boldsymbol{x}(t_j))}{\partial \boldsymbol{x}(t_0)} \\ &= \frac{\partial \boldsymbol{h}(\boldsymbol{x}(t_j))}{\partial \boldsymbol{x}(t_j)} \frac{\partial \boldsymbol{x}(t_j)}{\partial \boldsymbol{x}(t_0)} \\ &= \frac{\partial \boldsymbol{h}(\boldsymbol{x}(t_j))}{\partial \boldsymbol{x}(t_j)} \Phi(t_j, t_0) \qquad j = 1, 2, \ldots m \end{aligned} \tag{3.11}$$

Naturally, the second-order sensitivities are computed as follows:

$$\begin{aligned}\nabla H_j &= \frac{\partial^2 \boldsymbol{h}(\boldsymbol{x}(t_j))}{\partial \boldsymbol{x}^2(t_0)} \\ &= \frac{\partial^2 \boldsymbol{h}(\boldsymbol{x}(t_j))}{\partial \boldsymbol{x}(t_j)^2} \otimes \frac{\partial \boldsymbol{x}(t_j)}{\partial \boldsymbol{x}(t_0)} \otimes \frac{\partial \boldsymbol{x}(t_j)}{\partial \boldsymbol{x}(t_0)} + \frac{\partial \boldsymbol{h}(\boldsymbol{x}(t_j))}{\partial \boldsymbol{x}(t)} \otimes \frac{\partial^2 \boldsymbol{x}(t_j)}{\partial \boldsymbol{x}(t_0)^2} \\ &= \frac{\partial^2 \boldsymbol{h}(\boldsymbol{x}(t_j))}{\partial \boldsymbol{x}(t_j)^2} \otimes \Phi(t_j,t_0) \otimes \Phi(t_j,t_0) + \frac{\partial \boldsymbol{h}(\boldsymbol{x}(t_j))}{\partial \boldsymbol{x}(t)} \otimes \Phi^{(2)}(t_j,t_0)\end{aligned} \tag{3.12}$$

where the symbol $\Phi(t_j,t_0)$ represents a first-order state transition matrix and $\Phi^{(2)}(t_j,t_0)$ denotes a second-order state transition matrix, which we define below. As can be seen in Eqs. (3.11) and (3.12), in calculating these sensitivities we require computing a number of partial derivatives including the first- and second-order partials as shown here. These sensitivities are written here symbolically where the symbol "$\otimes$" represents a dyadic product. A more thorough interpretation of how to carry out the implied products for the first- through fourth-order sensitivity calculations is given in Appendix B in indicial notation.

Now we develop the necessary state transition matrix differential equations. For first- through fourth-order generalizations[16], we must introduce a number of state transition matrices, and we adopt the following notation:

First-order: $\Phi(t,t_0) = \dfrac{\partial \boldsymbol{x}(t)}{\partial \boldsymbol{x}(t_0)}$

Second-order: $\Phi^{(2)}(t,t_0) = \dfrac{\partial^2 \boldsymbol{x}(t)}{\partial \boldsymbol{x}(t_0)^2}$

Third-order: $\Phi^{(3)}(t,t_0) = \dfrac{\partial^3 \boldsymbol{x}(t)}{\partial \boldsymbol{x}(t_0)^3}$

Fourth-order: $\Phi^{(4)}(t,t_0)=\dfrac{\partial^4 \boldsymbol{x}(t)}{\partial \boldsymbol{x}(t_0)^4}$

k$_{\text{th}}$-order: $\Phi^{(k)}(t,t_0)=\dfrac{\partial^k \boldsymbol{x}(t)}{\partial \boldsymbol{x}(t_0)^k}$

We adopt this notation as a natural extension of the standard first-order notation. Furthermore, it should be noted that the second- and higher-order state transition matrices, as we are calling them, are actually third- and higher-order tensor arrays. Thus, we name them higher-order state transition "matrices" since they are an extension of the first-order state transition matrix concept.

All the higher-order state transition matrices are computed at each instant in time by solving differential equations. As will be evident, these differential equations assume a cascade form that facilitates computation. Previously, we developed the first-order state transition matrix differential equation through the linearized equations of motion. Here we proceed in a more general fashion by beginning with the integral form of Eq. (3.4).

$$\boldsymbol{x}(t)=\boldsymbol{x}(t_o)+\int_{t_o}^{t}\boldsymbol{f}(\tau,\boldsymbol{x}(\tau))d\tau \tag{3.13}$$

We then differentiate Eq. (3.13) with respect to the initial state to compute

$$\Phi(t,t_o)=\frac{\partial \boldsymbol{x}(t)}{\partial \boldsymbol{x}(t_o)}=I+\int_{t_o}^{t}\left[\frac{\partial \boldsymbol{f}(\tau,\boldsymbol{x}(\tau))}{\partial \boldsymbol{x}(\tau)}\right]\left[\frac{\partial \boldsymbol{x}(\tau)}{\partial \boldsymbol{x}(t_o)}\right]d\tau \tag{3.14}$$

The first-order state transition matrix differential equation is obtained upon time differentiation of Eq. (3.14).

$$\dot{\Phi}(t,t_o)=\left[\frac{\partial \boldsymbol{f}(t,\boldsymbol{x}(t))}{\partial \boldsymbol{x}(t)}\right]\Phi(t,t_o) \tag{3.15}$$

Upon differentiating Eq. (3.14) once again with respect to the initial state, and then time differentiating this expression, we arrive at the following second-order state transition matrix differential equation

$$\dot{\Phi}^{(2)}(t,t_o)=\left[\frac{\partial \boldsymbol{f}(t,\boldsymbol{x}(t))}{\partial \boldsymbol{x}(t)}\right]\otimes\Phi^{(2)}(t,t_o)+\left[\frac{\partial^2 \boldsymbol{f}(t,\boldsymbol{x}(t))}{\partial \boldsymbol{x}(t)^2}\right]\otimes\Phi(t,t_o)\otimes\Phi(t,t_o) \quad (3.16)$$

with zero valued initial conditions.

Again, it is obvious that the state transition matrix differential equations can be extended to any order desired by continuing along this path. The above equations are symbolic shorthand, not matrix expressions. The actual equations required for computation are most easily developed in indicial equations. These developments are given for first- through fourth-order state transition matrix differential equations in Appendix C. It should be noted that in deriving state transition matrix differential equations for a particular problem, some number of partial derivatives of the dynamical equations are required. Using automatic differentiation, we can code all state transition matrix differential equations once and for all since the required partials are automatically derived and evaluated in the background using OCEA. Additionally, the dimensions of the state transition matrices can be arbitrarily declared, thus creating the framework of a general algorithm for computing state transition matrix differential equations for an arbitrary dimensioned problem. Any integration scheme can be wrapped around these equations in order to solve them.

Furthermore, we note that with these developments of higher-order state transition matrix concepts, that we can look to write the departure motion about some nominal state path as follows:

$$\begin{aligned}\delta\boldsymbol{x}(t) = {} & \Phi(t,t_0)\delta\boldsymbol{x}(t_0) + \tfrac{1}{2}\Phi^{(2)}(t,t_0)\otimes\delta\boldsymbol{x}(t_0)\otimes\delta\boldsymbol{x}(t_0) \\ & + \tfrac{1}{6}\Phi^{(3)}(t,t_0)\otimes\delta\boldsymbol{x}(t_0)\otimes\delta\boldsymbol{x}(t_0)\otimes\delta\boldsymbol{x}(t_0) \\ & + \tfrac{1}{24}\Phi^{(4)}(t,t_0)\otimes\delta\boldsymbol{x}(t_0)\otimes\delta\boldsymbol{x}(t_0)\otimes\delta\boldsymbol{x}(t_0)\otimes\delta\boldsymbol{x}(t_0) \\ & + \text{ H.O.T.}\end{aligned} \tag{3.17}$$

Equation (3.17) represents a fourth-order power series of the departure motion in terms of the initial departure motion. Of course, by adding terms to this expression we can expect to more accurately predict the true departure motion due to changes in the initial state. We are not limited to first-order methods for studying; for example, the sensitivity of differential equations to changes in initial conditions. Again, the above Eq. (3.17) is a symbolic shorthand for the corresponding indicial equation; for example, the second-order term in Eq. (3.17) can be written in indicial notation as $\frac{1}{2}\Phi^{(2)}_{ist}\delta x_t \delta x_s$. Only the first term can be computed by matrix product.

Here, though, we are primarily focused on higher-order estimation algorithms. We can think of Eq. (3.17) as a fourth-order mapping of small departure motion between the initial time and any other time. However, what we typically require in solving an estimation problem is the inverse mapping, i.e., a solution for the initial departure state in terms of the departure state at some arbitrary time. This solution is presented later in this section.

3.2.2 Properties of Higher-order State Transition Matrices

The properties of and methods for computing the first-order state transition matrix have been studied in great detail. The first-order state transition matrix can be computed by utilizing the group chain rule property which holds in the general case:

$$\begin{aligned}\Phi(t_2,t_1) &= \Phi(t_2,t_0)\Phi(t_0,t_1) \\ &= \Phi(t_2,t_0)\Phi(t_1,t_0)^{-1}\end{aligned} \tag{3.18}$$

Therefore, we can compute $\Phi(t_2,t_1)$ using the forward integrated, stored history of state transition matrices described by $\Phi(t,t_0)$ for $t_0 \le t \le t_f$. This is a well known property of the first-order state transition matrix which allows for computing the state transition matrix which maps the departure motion between any times. It follows from the chain rule for partial differentiation, because it is easy to prove that $\partial x(t_j)/\partial x(t_i) \equiv \Phi(t_j,t_i)$. Another special property of the first-order state transition matrix is the symplectic property. This allows for matrix inversion of the first-order state transition matrix to be replaced by a transpose operation and pre- and post-multiplication by an orthogonal matrix[19] for the special case of conservative natural dynamical systems. Furthermore, the inverse mapping of the first-order state transition matrix can be computed via matrix inversion and results in the following:

$$\Phi(t_1,t_2) = \Phi(t_2,t_1)^{-1} \tag{3.19}$$

Now we must ask whether these special properties also hold for the second- and higher-order state transition matrices. A simple examination of the second-order state transition matrix shows that the group chain rule property does not hold. This clearly follows for the higher-order state transition matrices as well. Furthermore, the matrix inversion property is not valid for the higher-dimension tensor objects. The inverse mapping; however, can at least be computed for the second- and higher-order state transition matrices by integrating backward from some time t_2 in order to compute $\Phi^{(2)}(t,t_2)$. However, an alternative structure of augmented differential equations in a higher dimensional space allows us to see that analogous group properties do in fact exist.

We can augment the state vector $\boldsymbol{x}$ by the elements of the first-order state transition matrix $\Phi(t,t_0)$. In order to accomplish this we must perform a "vec" operation on the first-order state transition to form the following vector containing the elements of the first-order state transition matrix.

$$\boldsymbol{\phi} \equiv column(\Phi(t,t_0)) = \begin{Bmatrix} \Phi_{11}(t,t_0) \\ \Phi_{12}(t,t_0) \\ \vdots \\ \Phi_{nn}(t,t_0) \end{Bmatrix} \tag{3.20}$$

where $\Phi_{ij}(t,t_0)$ is an element of the first-order state transition matrix. Thus we can form a new state vector by augmenting the state $\boldsymbol{x}$ with $\boldsymbol{\phi}$:

$$\boldsymbol{y} = \begin{bmatrix} \boldsymbol{x} \\ \text{----} \\ \boldsymbol{\phi} \end{bmatrix} \tag{3.21}$$

The differential equations governing $\boldsymbol{y}$ are formed using Eq. (3.4) and the "vec" form of the first-order state transition matrix differential equations given by Eq. 3.15.

$$\dot{\boldsymbol{y}} = \boldsymbol{F}(t, \boldsymbol{y}) = \left\{ \begin{array}{c} \boldsymbol{f} \\ \hdashline \dfrac{d\boldsymbol{\phi}}{dt} \end{array} \right\} \tag{3.22}$$

where $\dfrac{d\boldsymbol{\phi}}{dt}$ is a function of $t, \boldsymbol{x},$ and $\boldsymbol{\phi}$.

We can now derive a state transition matrix differential equation for the augmented state as given in Eq. (3.23).

$$\dot{\Psi}(t,t_0) = \left[\frac{\partial \boldsymbol{F}(t)}{\partial \boldsymbol{y}(t)} \right] \Psi(t,t_0) \tag{3.23}$$

where

$$\Psi(t,t_0) = \left[\frac{\partial \boldsymbol{y}(t)}{\partial \boldsymbol{y}(t_0)} \right] \tag{3.24}$$

We note that $\Psi(t,t_0)$ does satisfy the state transition matrix group chain rule property, and is in fact a higher-order state transition matrix which contains all second partials. Equation (3.25) illustrates this point.

$$\Psi(t,t_0) = \left[\frac{\partial \boldsymbol{y}(t)}{\partial \boldsymbol{y}(t_0)} \right] = \left[\begin{array}{c:c} \dfrac{\partial \boldsymbol{x}(t)}{\partial \boldsymbol{x}(t_0)} & \dfrac{\partial \boldsymbol{x}(t)}{\partial \boldsymbol{\phi}(t_0)} \\ \hdashline \dfrac{\partial \boldsymbol{\phi}(t)}{\partial \boldsymbol{x}(t_0)} & \dfrac{\partial \boldsymbol{\phi}(t)}{\partial \boldsymbol{\phi}(t_0)} \end{array} \right] \tag{3.25}$$

We identify the *n* by *n* matrix $\dfrac{\partial \boldsymbol{x}(t)}{\partial \boldsymbol{x}(t_0)}$ as $\Phi(t,t_0)$, which is the first-order state transition matrix associated with the state $\boldsymbol{x}$, and the *m* by *n* matrix $\dfrac{\partial \boldsymbol{\phi}(t)}{\partial \boldsymbol{x}(t_0)}$ as the "vec" form of $\Phi^{(2)}(t,t_0) \equiv \dfrac{\partial^2 \boldsymbol{x}(t)}{\partial \boldsymbol{x}(t_0)^2}$, which is the second-order state transition matrix associated with the state $\boldsymbol{x}$, where $m = n^2$. Thus, by this augmented state formulation, we can compute the first- and second-order state transition matrices simultaneously. We also note in Eq. (3.25) that all elements of the n by m matrix $\dfrac{\partial \boldsymbol{x}(t)}{\partial \boldsymbol{\phi}(t_0)}$ are zero. The matrix $\dfrac{\partial \boldsymbol{\phi}(t)}{\partial \boldsymbol{\phi}(t_0)}$ has dimension m by m.

We now investigate the group chain rule property for the matrix $\Psi(t,t_0)$. Equation (3.18) results in

$$\Psi(t_2,t_1) = \Psi(t_2,t_0)\Psi(t_1,t_0)^{-1} \tag{3.26}$$

where

$$\Psi(t_1,t_0) = \left[\begin{array}{c|c} \dfrac{\partial \boldsymbol{x}(t_1)}{\partial \boldsymbol{x}(t_0)} & 0_{nxm} \\ \hline \dfrac{\partial \boldsymbol{\phi}(t_1)}{\partial \boldsymbol{x}(t_0)} & \dfrac{\partial \boldsymbol{\phi}(t_1)}{\partial \boldsymbol{\phi}(t_0)} \end{array}\right] \tag{3.27}$$

and

$$\Psi(t_2,t_0) = \left[\begin{array}{c|c} \dfrac{\partial \boldsymbol{x}(t_2)}{\partial \boldsymbol{x}(t_0)} & 0_{nxm} \\ \hline \dfrac{\partial \boldsymbol{\phi}(t_2)}{\partial \boldsymbol{x}(t_0)} & \dfrac{\partial \boldsymbol{\phi}(t_2)}{\partial \boldsymbol{\phi}(t_0)} \end{array}\right] \tag{3.28}$$

As indicated in Eq. (3.26), we must invert $\Psi(t_1,t_0)$. In order to accomplish this, we turn to a well known relationship[20] for inverting block matrices of the form:

$$\mathrm{A}^{-1} = \left[\begin{array}{c:c} A & B \\ \hdashline C & D \end{array}\right]^{-1} = \left[\begin{array}{c:c} A^{-1} + A^{-1}B\Delta^{-1}CA^{-1} & -A^{-1}B\Delta^{-1} \\ \hdashline -\Delta^{-1}CA^{-1} & \Delta^{-1} \end{array}\right] \tag{3.29}$$

where

$$\Delta = D - CA^{-1}B$$

In applying Eq. (3.29) to $\Psi(t_1,t_0)^{-1}$ we find

$$\Psi(t_1,t_0)^{-1} = \left[\begin{array}{c:c} \left[\dfrac{\partial \boldsymbol{x}(t_1)}{\partial \boldsymbol{x}(t_0)}\right]^{-1} & 0_{nxm} \\ \hdashline -\left[\dfrac{\partial \boldsymbol{\phi}(t_1)}{\partial \boldsymbol{\phi}(t_0)}\right]^{-1}\left[\dfrac{\partial \boldsymbol{\phi}(t_1)}{\partial \boldsymbol{x}(t_0)}\right]\left[\dfrac{\partial \boldsymbol{x}(t_1)}{\partial \boldsymbol{x}(t_0)}\right]^{-1} & \left[\dfrac{\partial \boldsymbol{\phi}(t_1)}{\partial \boldsymbol{\phi}(t_0)}\right]^{-1} \end{array}\right] \tag{3.30}$$

Equation (3.26) results in the following equation given Eq. (3.30)

$$\Psi(t_2,t_1) = \left[\begin{array}{c:c} \left[\dfrac{\partial \boldsymbol{x}(t_2)}{\partial \boldsymbol{x}(t_0)}\right]\left[\dfrac{\partial \boldsymbol{x}(t_1)}{\partial \boldsymbol{x}(t_0)}\right]^{-1} & 0_{nxm} \\ \hdashline \begin{array}{l} \left[\dfrac{\partial \boldsymbol{\phi}(t_2)}{\partial \boldsymbol{x}(t_0)}\right]\left[\dfrac{\partial \boldsymbol{x}(t_1)}{\partial \boldsymbol{x}(t_0)}\right]^{-1} \\ -\left[\dfrac{\partial \boldsymbol{\phi}(t_2)}{\partial \boldsymbol{\phi}(t_0)}\right]\left[\dfrac{\partial \boldsymbol{\phi}(t_1)}{\partial \boldsymbol{\phi}(t_0)}\right]^{-1}\left[\dfrac{\partial \boldsymbol{\phi}(t_1)}{\partial \boldsymbol{x}(t_0)}\right]\left[\dfrac{\partial \boldsymbol{x}(t_1)}{\partial \boldsymbol{x}(t_0)}\right]^{-1} \end{array} & \left[\dfrac{\partial \boldsymbol{\phi}(t_2)}{\partial \boldsymbol{x}(t_0)}\right]\left[\dfrac{\partial \boldsymbol{\phi}(t_1)}{\partial \boldsymbol{\phi}(t_0)}\right]^{-1} \end{array}\right] \tag{3.31}$$

where

$$\Psi(t_2,t_1) = \left[\begin{array}{c|c} \dfrac{\partial \boldsymbol{x}(t_2)}{\partial \boldsymbol{x}(t_1)} & 0_{nxm} \\ \hline \dfrac{\partial \boldsymbol{\phi}(t_2)}{\partial \boldsymbol{x}(t_1)} & \dfrac{\partial \boldsymbol{\phi}(t_2)}{\partial \boldsymbol{\phi}(t_1)} \end{array}\right] \tag{3.32}$$

We identify by comparing Eqs. (3.31) and (3.32) that

$$\frac{\partial \boldsymbol{x}(t_2)}{\partial \boldsymbol{x}(t_1)} = \left[\frac{\partial \boldsymbol{x}(t_2)}{\partial \boldsymbol{x}(t_0)}\right]\left[\frac{\partial \boldsymbol{x}(t_1)}{\partial \boldsymbol{x}(t_0)}\right]^{-1} \tag{3.33}$$

$$\frac{\partial \boldsymbol{\phi}(t_2)}{\partial \boldsymbol{x}(t_1)} = \left[\frac{\partial \boldsymbol{\phi}(t_2)}{\partial \boldsymbol{x}(t_0)}\right]\left[\frac{\partial \boldsymbol{x}(t_1)}{\partial \boldsymbol{x}(t_0)}\right]^{-1} - \left[\frac{\partial \boldsymbol{\phi}(t_2)}{\partial \boldsymbol{\phi}(t_0)}\right]\left[\frac{\partial \boldsymbol{\phi}(t_1)}{\partial \boldsymbol{\phi}(t_0)}\right]^{-1}\left[\frac{\partial \boldsymbol{\phi}(t_1)}{\partial \boldsymbol{x}(t_0)}\right]\left[\frac{\partial \boldsymbol{x}(t_1)}{\partial \boldsymbol{x}(t_0)}\right]^{-1} \tag{3.34}$$

and

$$\frac{\partial \boldsymbol{\phi}(t_2)}{\partial \boldsymbol{\phi}(t_1)} = \left[\frac{\partial \boldsymbol{\phi}(t_2)}{\partial \boldsymbol{x}(t_0)}\right]\left[\frac{\partial \boldsymbol{\phi}(t_1)}{\partial \boldsymbol{\phi}(t_0)}\right]^{-1} \tag{3.35}$$

Thus, by this augmented state vector approach, we show (once again) that the group chain rule property holds for the first-order state transition matrix as shown in Eq. (3.33). Most importantly, we show that the second-order state transition matrix has this property as well and is computed by Eq. (3.34). Of course, we must perform an inverse "vec" operation to compute

$$\Phi^{(2)}(t_2,t_1) \equiv \frac{\partial \boldsymbol{\phi}(t_2)}{\partial \boldsymbol{x}(t_1)} \tag{3.36}$$

By augmenting the state vector with the elements of the first-order state transition matrix, we show that the group chain rule property holds for first- and second-order state transition matrices. Of course, we could continue along this path for the third- and higher-order state transition matrices. We now turn our attention to computing the state transition matrices.

Many methods have been developed to solve for the first-order state transition matrices. One summary is given in Reference 21. The most direct method for solving the first-order state transition matrices is integrating the differential equation; however, some analytical methods have been developed for the first-order case. No attempt has been made in this work to develop analogous exact analytical methods other than explicit numerical integration for solving for the second- and higher-order state transition matrices; however, this does not imply that other methods do not exist.

3.2.3 Reversion of Series Solution

In this section, we summarize the reversion of series solution. In the most general sense, the reversion of series provides the solution for the state update equation for the estimation algorithms. More specifically, it provides the inverse mapping for Eq. (3.17). More details on the original formulation can be found in Reference 14. In the subsequent sections, we detail the general form and the state transition matrix specific inverse mapping form.

3.2.3.1 General Form

Here, we show the reversion of series solution in its most general form. This form applies to any iterative estimation process considering very general sensitivity calculations. This solution provides the correction to be applied to the current guess of the unknown parameters in the estimation problem, where $g(x) = 0$ defines the necessary condition for the root of the equation.

In developing the reversion of series solution, Turner redefines the necessary condition of the root solving problem in the following parameter embedding problem

$$G(x(s),s) = g(x(s)) - sg(x_{guess}) = 0 \tag{3.37}$$

where s is a scalar embedding parameter, x_{guess} is the starting guess, and $x = x(s)$.

The initial and final states of the embedding function are $s = s_0 = 1$ and $s = s_f = 0$. Thus, $G(x(s),s)\big|_{s=1} = 0$ because $x(s=1) = x_{guess}$ and $G(x(s),s)\big|_{s=0} = g(x) = 0$ where the original function is returned. The solution to the original problem is obtained by choosing s_f such that the original necessary condition is satisfied. The value of s is varied from 1 to 0 in order to determine how x changes along the curve defined by G(x(s),s).

Along the homotopy path (i.e. analytical continuation path) defined by Eq. (3.37), the solution for $x(s)$ is generated by a Taylor series

$$x(s_f) = x_{guess} + \left.\frac{dx}{ds}\right|_{s_f} \Delta s + \frac{1}{2!}\left.\frac{d^2x}{ds^2}\right|_{s_f} \Delta s^2 + \frac{1}{3!}\left.\frac{d^3x}{ds^3}\right|_{s_f} \Delta s^3 + \ldots \tag{3.38}$$

where $\Delta s = s_f - s_0 = s_f - 1$. By setting $s_f = 0$ in Eq. (3.38), the artificial independent variable is eliminated from the series solution and the original problem dimension is restored, yielding the reversion of series solution given by

$$x \approx x_{guess} + \delta = x_{guess} - \left.\frac{dx}{ds}\right|_{s=1} + \frac{1}{2!}\left.\frac{d^2x}{ds^2}\right|_{s=1} - \frac{1}{3!}\left.\frac{d^3x}{ds^3}\right|_{s=1} + \ldots \tag{3.39}$$

where the differential rates appearing in Eq. (3.39) are obtained repeatedly differentiating G(*x*(*s*),*s*) with respect to *s*. For example, the first partial derivative of G(*x*(*s*),*s*) = 0 leads to

$$\begin{aligned}\frac{dG}{ds} &= \frac{\partial G}{\partial s} + \frac{\partial G}{\partial x}\frac{dx}{ds} \\ &= \frac{\partial G}{\partial s} + \nabla G \frac{dx}{ds}\end{aligned} \tag{3.40}$$

which can be solved for the differential rate given in Eq. (3.41). The developments leading to computing the additional rates are presented in Reference 14; however, the results clearly follow by continuing to differentiate Eq. (3.40) with respect to *s* and solving for the differential rate. The first- through fourth-order terms are given by

$$\left.\frac{dx}{ds}\right|_{s=1} = \left(\nabla G\right)^{-1} g(x_{guess}) \tag{3.41}$$

$$\left.\frac{d^2x}{ds^2}\right|_{s=1} = -\left(\nabla G\right)^{-1}\left(\nabla^2 G \otimes \left.\frac{dx}{ds}\right|_{s=1} \otimes \left.\frac{dx}{ds}\right|_{s=1}\right) \tag{3.42}$$

$$\left.\frac{d^3x}{ds^3}\right|_{s=1} = -\left(\nabla G\right)^{-1}\left\{\begin{array}{l} \nabla^3 G \cdot \left.\frac{dx}{ds}\right|_{s=1} \otimes \left.\frac{dx}{ds}\right|_{s=1} \otimes \left.\frac{dx}{ds}\right|_{s=1} + \\ 2\nabla^2 G \otimes \left.\frac{d^2x}{ds^2}\right|_{s=1} \otimes \left.\frac{dx}{ds}\right|_{s=1} + \\ \nabla^2 G \otimes \left.\frac{dx}{ds}\right|_{s=1} \otimes \left.\frac{d^2x}{ds^2}\right|_{s=1} \end{array}\right\} \tag{3.43}$$

$$\left.\frac{d^4x}{ds^4}\right|_{s=1} = -\left(\nabla G\right)^{-1}\left\{\begin{array}{l} \nabla^4 G\otimes\left.\frac{dx}{ds}\right|_{s=1}\otimes\left.\frac{dx}{ds}\right|_{s=1}\otimes\left.\frac{dx}{ds}\right|_{s=1}\otimes\left.\frac{dx}{ds}\right|_{s=1}+ \\ 3\nabla^3 G\otimes\left.\frac{d^2x}{ds^2}\right|_{s=1}\otimes\cdot\left.\frac{dx}{ds}\right|_{s=1}\otimes\left.\frac{dx}{ds}\right|_{s=1}+ \\ 2\nabla^3 G\otimes\left.\frac{dx}{ds}\right|_{s=1}\otimes\left.\frac{d^2x}{ds^2}\right|_{s=1}\otimes\left.\frac{dx}{ds}\right|_{s=1}+ \\ \nabla^3 G\cdot\left.\frac{dx}{ds}\right|_{s=1}\otimes\left.\frac{dx}{ds}\right|_{s=1}\otimes\left.\frac{d^2x}{ds^2}\right|_{s=1}+3\nabla^2 G\otimes\left.\frac{d^3x}{ds^3}\right|_{s=1}\otimes\left.\frac{dx}{ds}\right|_{s=1}+ \\ 3\nabla^2 G\otimes\left.\frac{d^2x}{ds^2}\right|_{s=1}\otimes\left.\frac{d^2x}{ds^2}\right|_{s=1}+\nabla^2 G\otimes\left.\frac{dx}{ds}\right|_{s=1}\otimes\left.\frac{d^3x}{ds^3}\right|_{s=1} \end{array}\right\} \quad (3.44)$$

where $g(x_{guess})$ is, in a very general sense, the measurement model error. The symbol "$\otimes$" represents a dyadic product. For the GLSDC problem, this is the error between the observations and the predictions of the measurement model. In the case of a TPBVP, this measurement model error is the error in the terminal boundary states. $\nabla G, \nabla^2 G, \nabla^3 G$, and $\nabla^4 G$ are symbolically the first- through fourth-order sensitivities.

In general, the solution given in Eqs. (3.41-3.44) provides a means for solving iterative problems in optimization and estimation. When only considering the first-order term in Eq. (3.39), we have the multilinear version of Newton's Method. Equation (3.39) can therefore be considered as the higher-order extension of Newton's Method. The gradient terms ($\nabla G, \nabla^2 G, \nabla^3 G$, and $\nabla^4 G$) are understood to be taken with respect to the elements of the vector of unknown parameters to be estimated. The state update in Eq. (3.39) can be accomplished once the desired order of terms coming from Eqs. (3.41-3.44) are computed.

3.2.3.2 State Transition Matrix Form

We can utilize Eq. (3.97) to compute the inverse mapping of Eq. (3.17) when the sensitivity calculations are computed solely by state transition matrix calculations. That is, $\nabla G, \nabla^2 G, \nabla^3 G$, and $\nabla^4 G$ are symbolically the first- through fourth-order state transition matrices.

We simply substitute the state transition matrices into Eq. (3.39) in order to compute the state update, including up through second-order terms in this illustration we produce the inverse mapping of Eq. (3.17):

$$\begin{aligned}\delta\boldsymbol{x}(t_0) &= \Phi(t,t_0)^{-1}\delta\boldsymbol{x}(t) \\ &\quad -\tfrac{1}{2}\Phi(t,t_0)^{-1}\left\{\Phi^{(2)}(t,t_0)\otimes\left(\Phi(t,t_0)^{-1}\delta\boldsymbol{x}(t)\right)\otimes\left(\Phi(t,t_0)^{-1}\delta\boldsymbol{x}(t)\right)\right\}\end{aligned} \tag{3.45}$$

This mapping can be easily extended to include third- and higher-order terms using Eqs. (3.43) and (3.44); however, this is easily shown with only the second-order illustration.

The developments of this section provide the ingredients required for developing estimation and optimization algorithms which utilize higher-order sensitivity calculations. The reversion of series solution provides the means to compute the state update, and the development of the higher-order state transition matrix generalizations enables the computation of the required sensitivity calculations for many of the methods developed in the next section.

3.3 HIGHER-ORDER ALGORITHMS

The developments of the previous section provide the general framework for the new methods detailed in this section. Here, we provide the essential details for implementation of new solutions for Nonlinear Least Squares and GLSDC algorithms. Additionally, we present some new ideas for solving trajectory optimization problems and computing midcourse corrections.

Generally, these algorithms fall into two categories: direct optimization type algorithms and state transition matrix type algorithms. Nonlinear Least Squares problems fall into the direct optimization category. Here, the sensitivities are explicit partial derivative calculations. GLSDC algorithms fall into the state transition matrix category due to the definition of the sensitivities. Simply put, we need the state transition matrix calculations for solving this type of problem. The sensitivity calculations are computed as they are in Eqs. (3.11) and (3.12) which require explicit partials of the measurement model and the state transition matrices. Regardless of the type of problem, many times we wish to weight the measurements in some fashion. A description of weight matrix issues when considering higher-order generalizations is given in the next section.

3.3.1 Weight Matrix Issues

We note here that some special attention must be given to weighting observations for higher-order methods. For uncorrelated measurements, the optimal choice for weighting is $w_i = \sigma_i^{-2}$ where σ_i^2 is the variance of the ith measurement. This weighting approach results in a diagonal weight matrix, W, which can be factored by the Cholesky Decomposition as $W = LL^T$. Thus, for the optimal choice, $L = L^T = diag(1/\sigma_i)$. The simplest way to implement weighted observations is to pre-multiply the observations and the measurement model predictions, which in this work includes the evaluations of the measurement model and its first- and higher-order partial derivatives, by L^T. This can be easily illustrated by considering direct optimization of the cost function. Obviously, here we include the weight matrix in the formulation of the cost function, and subsequent sensitivity calculations automatically include the weighting effect regardless of the order of the solution.

3.3.2 Trajectory Optimization Algorithms

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.

Here, we study the automatic generation of sensitivity equations for solving nonlinear TPBVPs. The type of TPBVP we consider is a trajectory optimization problem, e.g. orbital transfer. In this work, we investigate two methods for solving TPBVPs including optimal solutions by the method of differential corrections[22,23] and near-optimal solutions using the method of differential inclusions[24].

One particular solution approach is 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. Direct optimization using differential inclusions will be considered as a second approach, demonstrating OCEA's AD capabilities in solving TPBVPs. First- through fourth-order sensitivity calculations will be considered in each case. A number of results detailing performance will be presented.

3.3.2.1 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 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.

Consider the minimization of the following function where t_f is the final time

$$J = \int_0^{t_f} L(\boldsymbol{x}(t), \boldsymbol{u}(t), t) dt \tag{3.46}$$

subject to the following dynamics

$$\dot{\boldsymbol{x}} = \boldsymbol{f}(t, \boldsymbol{x}(t)) \tag{3.47}$$

In general, the Hamiltonian for the system is written as

$$H = L(\boldsymbol{x}(t), \boldsymbol{u}(t), t) + \boldsymbol{\lambda}^T \boldsymbol{f}(\boldsymbol{x}(t), \boldsymbol{u}(t), t) \tag{3.48}$$

For unconstrained control, the well-known necessary conditions along the trajectory required to minimize J are given by Eqs. (3.49-3.51)

$$\dot{\boldsymbol{x}} = \frac{\partial H}{\partial \boldsymbol{\lambda}} \tag{3.49}$$

$$\dot{\boldsymbol{\lambda}} = -\frac{\partial H}{\partial \boldsymbol{x}} \tag{3.50}$$

$$\frac{\partial H}{\partial \boldsymbol{u}} = 0 \tag{3.51}$$

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. Also, when the final time is free we have $H(t_f) = 0$ as an additional necessary condition.

Now, we consider converting the free final time problem into a fixed final time problem. With the fixed final time problem, t_f appears 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, τ , defined as

$$\tau = \frac{t}{t_f}; \qquad 0 \le \tau \le 1 \tag{3.52}$$

Differentiation of the states with respect to the new time variable (denoted with a prime) can be written as

$$\boldsymbol{x}' = \frac{d\boldsymbol{x}}{d\tau} = \frac{d\boldsymbol{x}}{dt}\frac{dt}{d\tau} = t_f \dot{\boldsymbol{x}} \tag{3.53}$$

Therefore, in producing the new state equations for the fixed time problem we simply multiply Eq. (3.49) 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 $J = \int_0^1 t_f d\tau$, and the Hamiltonian is defined as

$$\begin{aligned} H &= t_f + t_f \boldsymbol{\lambda}^T \boldsymbol{f}(\boldsymbol{x}(\tau), \boldsymbol{u}(\tau), \tau) \\ &= t_f \left[1 + \boldsymbol{\lambda}^T \boldsymbol{f}(\boldsymbol{x}(\tau), \boldsymbol{u}(\tau), \tau)\right] \end{aligned} \tag{3.54}$$

Equations (3.49-3.51) are applied to the Hamiltonian in Eq. (3.54) to produce the fixed time problem necessary conditions along the trajectory. However, here we must consider one additional necessary condition[22] on the free parameter t_f

$$\int_0^1 \frac{\partial H}{\partial t_f} d\tau = 0 \tag{3.55}$$

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

3.3.2.2 Differential Corrections

The method of differential corrections is a shooting type method. The numerical process entails guessing the unknown 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 initial conditions as the solution. Otherwise, we introduce a local Taylor series of the terminal constraints as a function of initial conditions and utilize sensitivity calculations (Eqs. 3.41-3.44) to iterate the guess for the unknowns at the initial time and repeat the process until the terminal conditions are satisfied. Of course, implicit in this process are difficult issues on the domain of attraction that govern convergence. We do not explicitly consider the conditions for convergence in this study.

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. (3.39) 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

$$\delta \boldsymbol{x}(t_f) = \Phi(t_f, t_0)\delta \boldsymbol{x}(t_0); \qquad \Phi(t_f, t_0) = \frac{\partial \boldsymbol{x}(t_f)}{\partial \boldsymbol{x}(t_0)} \tag{3.56}$$

where $\delta \boldsymbol{x}(t_0)$ and $\delta \boldsymbol{x}(t_f)$ represent small state departures from the optimal solution at the initial and final times 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. (3.56) can be solved for $\delta \boldsymbol{x}(t_0)$ in order to update the estimate for the initial states. This solution corresponds to the first-order term given in Eq. (3.39).

We can easily extend the procedure to include second- and higher-order terms. For example, considering up to second-order terms in Eq. (3.39) we have as an update equation:

$$\begin{aligned}\delta \boldsymbol{x}(t_0) &= \Phi(t_f,t_0)^{-1}\delta \boldsymbol{x}(t_f) \\ &\quad -\tfrac{1}{2}\Phi(t_f,t_0)^{-1}\left\{\Phi^{(2)}(t_f,t_0)\otimes\left(\Phi(t_f,t_0)^{-1}\delta \boldsymbol{x}(t_f)\right)\otimes\left(\Phi(t_f,t_0)^{-1}\delta \boldsymbol{x}(t_f)\right)\right\}\end{aligned} \tag{3.57}$$

where "$\otimes$" represents a dyadic product and the second-order state transition matrix is defined as

$$\Phi^{(2)}(t_f,t_0) = \frac{\partial^2 \boldsymbol{x}(t_f)}{\partial \boldsymbol{x}^2(t_0)} \tag{3.58}$$

Again, the first- through fourth-order state transition matrix differential equations are shown in Appendix C. We note once again 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 $\boldsymbol{f}(\boldsymbol{x}(t),t)$ 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.

3.3.2.3 Differential Inclusions

The method of differential inclusions 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.

Here we consider solving problems using differential inclusions of the following type

$$\dot{\boldsymbol{x}} = \boldsymbol{f}(\boldsymbol{x}(t), t) + \boldsymbol{u}(t) \tag{3.59}$$

$$J = \frac{1}{2}\int_{0}^{t_f} \left(\boldsymbol{x}^T Q \boldsymbol{x} + u^T R u \right) \; dt \tag{3.60}$$

where the final time is fixed.

The procedure entails discretizing the equations of motion. For the i^th equation of Eq. (3.59) we have

$$\frac{x_{k+1}^{(i)} - x_k^{(i)}}{\Delta^{(i)}} - f\left(\frac{x_k^{(i)} + x_{k+1}^{(i)}}{2} \right) = u_k^{(i)} \tag{3.61}$$

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. (3.60) is also rewritten as

$$J = \tfrac{1}{2}\sum_{i=1}^{m}\sum_{j=1}^{m}\left(\sum_{k=1}^{N^{(i)}}\left(q^{(i,j)}x_k^{(i)}x_k^{(j)} + r^{(i,j)}u_k^{(i)}u_k^{(j)}\right)\right) \tag{3.62}$$

where m is the number of states ($\boldsymbol{x} \in R^m$).

The expression on the left-hand side of Eq. (3.61) can be substituted into the performance index of Eq. (3.62) which eliminates the control from the formulation. Therefore, the unknowns to be solved for are the discretized state coordinates defined as $x_k^{(i)} = x^{(i)}(t_k) = x^{(i)}(k\Delta^{(i)})$. This can be accomplished by minimization of the nonlinear function that results from substituting the control into Eq. (3.62).

The capability of OCEA to automatically compute the partial derivatives of J makes this problem readily solvable by a code that applies to a large class of problems. 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 $\nabla J = \boldsymbol{0}$. 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. (3.61) and solve for the control. With the control and state time histories, we can then compute the initial co-states from the differential inclusions solution. These can be used as an improved starting guess for another solution method.

3.3.3 Midcourse Corrections

An issue that must be addressed in any trajectory design is performing midcourse corrections when the vehicle deviates from the reference path. In many historical applications, e.g. the Apollo Saturn V guidance laws, 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:

$$\delta \boldsymbol{x}(t^*) = \begin{bmatrix} \delta \boldsymbol{r}(t^*) \\ \delta \boldsymbol{v}(t^*) \end{bmatrix} = \Phi(t^*, t_f) \begin{bmatrix} \delta \boldsymbol{r}(t_f) \\ \delta \boldsymbol{v}(t_f) \end{bmatrix} \tag{3.63}$$

where

$$\Phi(t^*, t_f) = \left[\begin{array}{c|c} \dfrac{\partial \boldsymbol{r}(t^*)}{\partial \boldsymbol{r}(t_f)} & \dfrac{\partial \boldsymbol{r}(t^*)}{\partial \boldsymbol{v}(t_f)} \\ \hline \dfrac{\partial \boldsymbol{v}(t^*)}{\partial \boldsymbol{r}(t_f)} & \dfrac{\partial \boldsymbol{v}(t^*)}{\partial \boldsymbol{v}(t_f)} \end{array}\right]$$

The procedure is given by Battin in Reference 19. Suppose we wish to maintain the reference path, then we choose $\delta \boldsymbol{r}(t_f) = \boldsymbol{0}$ and solve Eq. (3.63) for $\delta \boldsymbol{v}(t^*)$, the instantaneous velocity correction to compensate for the position error, $\delta \boldsymbol{r}(t^*)$, at that time. The state transition matrix required in Eq. (3.63) can be computed by utilizing the group chain rule property of the first-order state transition matrices which hold in the general case

$$\begin{aligned} \Phi(t_3, t_1) &= \Phi(t_3, t_2)\Phi(t_2, t_1) \\ &= \Phi(t_3, t_2)\Phi(t_1, t_2)^{-1} \end{aligned} \tag{3.64}$$

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 $t_0 \le t \le t_f$ and then solve Eq. (3.63). This property of first-order state transition matrix was described in section 3.2.

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

$$\delta \boldsymbol{x}(t^*) = \Phi(t^*, t_f)\delta \boldsymbol{x}(t_f) + \tfrac{1}{2}\Phi^{(2)}(t^*, t_f) \otimes \delta \boldsymbol{x}(t_f) \otimes \delta \boldsymbol{x}(t_f) \tag{3.65}$$

where $\Phi^{(2)}(t^*, t_f)$ is a second-order state transition matrix and is given symbolically by

$$\Phi^{(2)}(t^*, t_f) = \left[\begin{array}{c|c} \dfrac{\partial^2 \boldsymbol{r}(t^*)}{\partial \boldsymbol{r}(t_f)^2} & \dfrac{\partial^2 \boldsymbol{r}(t^*)}{\partial \boldsymbol{v}(t_f)^2} \\ \hline \dfrac{\partial^2 \boldsymbol{v}(t^*)}{\partial \boldsymbol{r}(t_f)^2} & \dfrac{\partial^2 \boldsymbol{v}(t^*)}{\partial \boldsymbol{v}(t_f)^2} \end{array}\right]$$

Again, choosing $\delta \boldsymbol{r}(t_f) = \boldsymbol{0}$, we can solve Eq. (3.65) for $\delta \boldsymbol{v}(t_f)$

$$\begin{aligned} \delta \boldsymbol{v}(t_f) &= \left[\frac{\partial \boldsymbol{r}(t^*)}{\partial \boldsymbol{v}(t_f)}\right]^{-1} \delta \boldsymbol{r}(t^*) \\ &- \tfrac{1}{2}\left[\frac{\partial \boldsymbol{r}(t^*)}{\partial \boldsymbol{v}(t_f)}\right]^{-1} \left\{\left(\frac{\partial^2 \boldsymbol{r}(t^*)}{\partial \boldsymbol{v}(t_f)^2}\right) \otimes \left(\left[\frac{\partial \boldsymbol{r}(t^*)}{\partial \boldsymbol{v}(t_f)}\right]^{-1} \delta \boldsymbol{r}(t^*)\right) \otimes \left(\left[\frac{\partial \boldsymbol{r}(t^*)}{\partial \boldsymbol{v}(t_f)}\right]^{-1} \delta \boldsymbol{r}(t^*)\right)\right\} \end{aligned} \tag{3.66}$$

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

$$\delta \boldsymbol{v}(t^*) = \frac{\partial \boldsymbol{v}(t^*)}{\partial \boldsymbol{v}(t_f)}\delta \boldsymbol{v}(t_f) + \tfrac{1}{2}\frac{\partial^2 \boldsymbol{v}(t^*)}{\partial \boldsymbol{v}(t_f)^2} \otimes \delta \boldsymbol{v}(t_f) \otimes \delta \boldsymbol{v}(t_f) \tag{3.67}$$

Equation (3.67) represents the velocity change required to eliminate terminal errors from the reference path including up to second-order terms. Excluding the second-order terms in Eq. (3.66), the first term on the right hand side of Eq. (3.67) is identically the well known first-order result. We stop at second-order here only to compact notation, higher-order terms can easily be included if desired. The first- through fourth-order generalized midcourse corrections are given in Appendix D.

In the above developments it was shown that the first-order state transition matrix in Eq. (3.63) is readily computed from the onboard stored data. We noted in section 3.2 that this property also holds for the second- and higher-order state transition matrices as well by the state vector augmentation approach. Thus we can solve for the second-order state transition matrix in Eq. (3.65) using the forward stored history of first- and second-order state transition matrices (due to coupling). Otherwise, we would need to solve for the second-order state transition matrix of Eq. (3.65) by integrating them along with the first-order state transition matrices backward from time t_f in order to compute $\Phi^{(2)}(t^*, t_f)$. Higher-order guidance laws will increase the computational 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. More to the point for this application, we are less concerned with computational cost here than with convergence and computational storage requirements. These are usually pre-mission calculations which are computed a priori and stored onboard.

3.3.4 Automatic Generation of Co-state Equations

The OCEA automatic differentiation feature enables the automatic generation of co-state differential equations. It can be seen from Eq. (3.50) 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 considerable 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 elapsed time and programming savings, especially for systems containing many states and/or complicated equations of motion.

3.4 NUMERICAL EXAMPLES

In this section, numerical results are presented to demonstrate the methods presented in the previous sections. Examples are presented to demonstrate the Nonlinear Least Squares Algorithm, the GLSDC algorithm, and the methods for solving trajectory optimization problems.

3.4.1 Ballistic Projectile Identification Problem

As an example, consider the orientation of an aerodynamically and inertially symmetric projectile. Along the trajectory, measurements are taken of the pitch and yaw angles. The following algebraic model is assumed for the pitch and yaw angles, respectively

$$\begin{aligned}\theta(t,\boldsymbol{x}) = h_1(t,\boldsymbol{x}) &= k_1 e^{\lambda_1 t}\cos(\omega_1 t+\delta_1) + k_2 e^{\lambda_2 t}\cos(\omega_2 t+\delta_2)\\ &+ k_3 e^{\lambda_3 t}\cos(\omega_3 t+\delta_3) + k_4\end{aligned} \tag{3.68}$$

$$\begin{aligned}\psi(t,\boldsymbol{x}) = h_2(t,\boldsymbol{x}) &= k_1 e^{\lambda_1 t}\sin(\omega_1 t+\delta_1) + k_2 e^{\lambda_2 t}\sin(\omega_2 t+\delta_2)\\ &+ k_3 e^{\lambda_3 t}\sin(\omega_3 t+\delta_3) + k_5\end{aligned} \tag{3.69}$$

where $\boldsymbol{x} = (k_1, k_2, k_3, k_4, k_5, \lambda_1, \lambda_2, \lambda_3, \omega_1, \omega_2, \omega_3, \delta_1, \delta_2, \delta_3)$ are the 14 unknown constant model parameters. Therefore, first- through fourth-order sensitivities require 14, 14^2, 14^3, and 14^4 partial derivatives to be derived, programmed, computed and validated. OCEA automates all of this work.

The Nonlinear Least Squares algorithm described earlier can be coded once and for all. In order to solve the problem at hand, only the measurement model and starting guess must be specified. For this problem, the FORTRAN90 subroutine containing the measurement model given in Eqs. (3.68) and (3.69) is shown Appendix E. An important remark to be made about the measurement model subroutine given in Appendix E is that the analyst can invoke automatic differentiation by standard FORTRAN programming. A USE statement (USE EB_HANDLING) is included in order to invoke the automatic differentiation subroutines. The embedded variables (the model parameters) and embedded functions (the measurement model) are defined as embedded objects (Type EB) and coded using standard FORTRAN arithmetic operators. The output of this subroutine is a numerical evaluation of the measurement model, its gradient, and higher-order partials, evaluated at the current state and time.

The automatic differentiation capability makes higher-order computational methods readily available. Results for norms of the measurement residuals for first-and second-order solutions are shown in Table 3.1.

Table 3.1. Residual error for Ballistic Projectile Identification problem

Iteration	First-order	Second-order
0	0.46453e4	0.46453e4
1	0.70828e3	0.40676e3
2	0.15229e3	0.21104e2
3	0.12652e2	0.39243e-2
4	0.88894e-1	0.19141e-11
5	0.78899e-5	
6	0.19504e-11	

The results of Table 3.1 show that for an initial guess of more than 50% error, the first-order method converges in 6 iterations and as expected the second-order method shows rapid convergence in only 4 iterations. For this problem, only modest convergence improvement is achieved by going to second-order.

In order to benchmark the approach, we also solved this problem using Matlab's Quasi-Newton algorithm (**fminunc**). In comparison to the results of Table 3.1, we find that the Matlab Quasi-Newton algorithm requires 163 iterations with a resulting residual error of 1.01781e-6. Attempts were made to improve the residual error by specifying tolerance options in the solution; however, no improvement beyond the residual error noted above can be made by the Quasi-Newton algorithm for this problem. We do note that there is a subtle difference in these approaches in that Matlab utilizes finite difference approximation for computing the partial derivatives of the *cost function* whereas OCEA utilizes exact analytical partials of the *measurement model* in the solution. Certainly, finite difference approximation is accompanied by a loss of accuracy as compared to methods utilizing exact analytical partials as evidenced by this example.

3.4.2 Orbit Determination

In order to demonstrate the higher-order generalization of the GLSDC algorithm, we consider as an example the planar motion of projectile in a constant gravity field. We also consider a quadratic drag model of the form $\boldsymbol{f}_{drag} = -p|\boldsymbol{V}|\boldsymbol{V}$ where p is the drag constant, $\boldsymbol{V} = [\dot{x}_1 \quad \dot{x}_2]$ is the velocity vector, and $|\boldsymbol{V}|$ is the magnitude of velocity. The equations of motion in first-order form are thus given by

$$\dot{\boldsymbol{x}} = \begin{Bmatrix} \dot{x}_1 \\ \dot{x}_2 \\ \dot{x}_3 \\ \dot{x}_4 \\ \dot{p} \end{Bmatrix} = \begin{Bmatrix} x_3 \\ x_4 \\ -px_3|V| \\ -g - px_4|V| \\ 0 \end{Bmatrix} \tag{3.70}$$

Range and line of sight measurements are taken along the projectile's trajectory. The measurement model is given by

$$\boldsymbol{h} = \begin{Bmatrix} r \\ \theta \end{Bmatrix} = \begin{Bmatrix} \sqrt{x_1^2 + x_2^2} \\ \tan^{-1}(x_2 / x_1) \end{Bmatrix} \tag{3.71}$$

OCEA automatically computes the partial derivatives of Eq. (3.70) which are required for automatically deriving the state transition matrix differential equations given in Appendix C, and also computes the partial derivatives of Eq. (3.71) required for the sensitivity calculations shown in Appendix B. A generalized GLSDC algorithm has been created which has been coded once and for all since partial derivatives need not be hand coded for each dynamics model or measurement model. For a particular problem, the analyst can simply change these models, the required sensitivities are automatically computed, and the problem can then be readily solved.

The objective here is to estimate the initial state whose true value is given by $\boldsymbol{x}(t_0) = \left[-1000m \quad 2500m \quad 200m/s \quad 100m/s\right]$. It is assumed that the standard deviation of the range and the line of sight measurements are 10 meters and 0.01 rad, respectively. For these simulations, the projectile is observed for a total of 60 seconds at 5 second intervals. Results are shown for two cases: I) without drag and II) with drag. When drag is present, the equations of motion are nonlinear as seen in Eq. (3.70); however, without drag this is a linear system. It can be observed from Appendix C that for a linear system, second- and higher-order state transition matrices are zero for all time; however, first-order state transition matrices are not. Therefore, second- and higher-order sensitivities are not zero for a linear system since they are a function of the first-order state transition matrices as can be seen in Appendix B.

In each of the cases, we are primarily interested in evaluating rate of convergence and domain of convergence, or put another way, we want to know how fast the algorithms converge and from how poor of a guess they will converge.

For Case I, with zero drag, we remove the drag parameter p from Eq. (3.70) and we estimate only the initial position and velocity. Since the measurements are a function of only position (no velocity dependence), we expect to have a better guess for the initial position than the velocity. For this reason and practical issues dealing with the large number of possible guesses, we simulate the first- and second-order algorithms with an initial position guess in 10% error of the truth and varied initial velocity guess error. The results for the number of iterations required for convergence for the first- and second-order GLSDC algorithms are given in Table 3.2. The stopping criterion used for these simulations is 10 digit consistency of the measurement residual error.

Table 3.2. GLSDC convergence study for Case I

Initial guess	First-order Iteration Count	Second-order Iteration Count
$0.9\,\boldsymbol{x}_{true}$, $1.0\,\dot{\boldsymbol{x}}_{true}$	3	3
$0.9\,\boldsymbol{x}_{true}$, $0.9\,\dot{\boldsymbol{x}}_{true}$	3	3
$0.9\,\boldsymbol{x}_{true}$, $0.8\,\dot{\boldsymbol{x}}_{true}$	4	3
$0.9\,\boldsymbol{x}_{true}$, $0.7\,\dot{\boldsymbol{x}}_{true}$	4	3
$0.9\,\boldsymbol{x}_{true}$, $0.6\,\dot{\boldsymbol{x}}_{true}$	4	3
$0.9\,\boldsymbol{x}_{true}$, $0.5\,\dot{\boldsymbol{x}}_{true}$	4	3
$0.9\,\boldsymbol{x}_{true}$, $0.4\,\dot{\boldsymbol{x}}_{true}$	5	4
$0.9\,\boldsymbol{x}_{true}$, $0.3\,\dot{\boldsymbol{x}}_{true}$	5	4
$0.9\,\boldsymbol{x}_{true}$, $0.2\,\dot{\boldsymbol{x}}_{true}$	6	5
$0.9\,\boldsymbol{x}_{true}$, $0.1\,\dot{\boldsymbol{x}}_{true}$	7	6
$0.9\,\boldsymbol{x}_{true}$, $0.0\,\dot{\boldsymbol{x}}_{true}$	11	7

The results of Table 3.2 show that for a large practical range of poor guesses in the initial velocity the second-order algorithm converges in fewer iterations. The state convergence history for an initial velocity guess of $0.2\,\dot{\boldsymbol{x}}_{true}$ is given in Table 3.3 for the first-order algorithm and in Table 3.4 for the second-order algorithm.

Table 3.3. Case I first-order algorithm state history results

Iteration	X_0	Z_0	$\dot{X}_0$	$\dot{Z}_0$	**Cost**
0	-900	2250	40	20	
1	-2012.5	2587.4	127.8	-9.0	883.6
2	-2114.6	2528.8	272.2	68.4	703.0
3	-1067.8	2559.5	203.7	102.0	690.1
4	-995.0	2504.3	199.9	99.6	22.76
5	-996.0	2503.7	199.9	99.6	3.361
6	-996.0	2503.7	199.9	99.6	3.315

Table 3.4. Case I second-order algorithm state history results

Iteration	X_0	Z_0	$\dot{X}_0$	$\dot{Z}_0$	**Cost**
0	-900	2250	40	20	
1	-2813.4	2516.1	216.1	15.0	883.6
2	-1020.0	2601.1	186.5	96.3	624.8
3	-996.9	2503.4	199.9	99.6	147.5
4	-996.0	2503.7	199.9	99.6	3.317
5	-996.0	2503.7	199.9	99.6	3.315

For Case II, with drag, we look at estimating five states including initial position and velocity, and the drag parameter. The results of Table 3.5 show that the first-order algorithm converges faster than the second-order algorithm, although the advantage is only one iteration.

Table 3.5. GLSDC convergence study for Case II

Initial guess	First-order Iteration Count	Second-order Iteration Count
$0.9\,\boldsymbol{x}_{true}$, $1.0\,\dot{\boldsymbol{x}}_{true}$, p = 1.0*ptrue	4	3
$0.9\,\boldsymbol{x}_{true}$, $0.9\,\dot{\boldsymbol{x}}_{true}$, p = 0.90*ptrue	4	3
$0.9\,\boldsymbol{x}_{true}$, $0.8\,\dot{\boldsymbol{x}}_{true}$, p = 0.80*ptrue	4	4
$0.9\,\boldsymbol{x}_{true}$, $0.7\,\dot{\boldsymbol{x}}_{true}$, p = 0.70*ptrue	5	5
$0.9\,\boldsymbol{x}_{true}$, $0.6\,\dot{\boldsymbol{x}}_{true}$, p = 0.60*ptrue	5	5
$0.9\,\boldsymbol{x}_{true}$, $0.5\,\dot{\boldsymbol{x}}_{true}$, p = 0.50*ptrue	6	5

Tables 3.6 and 3.7 show results for one particular initial guess corresponding to the last entry in Table 3.5.

Generally, we expect more rapid convergence of higher-order methods when near the answer. These results confirm this result; however, significant improvement is not seen in this example. There appears to be a reduction in second-order sensitivity near the answer. For a problem in which convergence is slow near the answer using a first-order method, we would expect that the second-order method would provide more significant improvement. Second-order algorithms are well known to have some potential difficulties far from the solution where curvature changes may occur en-route to the solution. Whereas first-order algorithms are typically insensitive to the starting guess, second-order algorithms can have a diminished domain of convergence because some starting guesses outside the region of convergence have second-order sensitivity (or curvature) which has the wrong sign. Simply put, if the initial guess lies where the curvature is wrong, the predicted corrections to the state estimate in the second-order term are not necessarily in the correct direction. Convergence depends upon the ability to extrapolate from one state to another state with a reduction in the performance index, which does not happen when the curvature has the wrong sign. As is known in the optimization literature, higher-order methods are typically "wonderful" near the solution but are "more dangerous" than first-order methods far from the solution.

Table 3.6. Case II first-order state history results

Iteration	X_0	Z_0	$\dot{X}_0$	$\dot{Z}_0$	p	**Cost**
0	-900	2250	100	50	5.0e-4	
1	-1297.5	2496.4	211.8	70.6	8.3e-4	420.9
2	-1029.6	2507.3	216.1	97.1	9.8e-4	120.6
3	-992.4	2498.6	195.8	100.2	9.9e-4	22.30
4	-992.5	2499.4	196.6	99.8	9.9e-4	3.245
5	-992.5	2499.4	196.6	99.8	9.9e-4	3.102
6	-992.5	2499.4	196.6	99.8	9.9e-4	3.102

Table 3.7. Case II second-order state history results

Iteration	X_0	Z_0	$\dot{X}_0$	$\dot{Z}_0$	p	**Cost**
0	-900	2250	100	50	5.0e-4	
1	-992.0	2451.5	144.5	88.2	8.1e-4	420.9
2	-989.1	2500.4	183.1	96.4	9.7e-4	83.98
3	-992.6	2499.5	196.0	99.6	9.9e-4	12.30
4	-992.5	2499.4	196.6	99.8	9.9e-4	3.182
5	-992.5	2499.4	196.6	99.8	9.9e-4	3.102

3.4.3 Trajectory Optimization

In this section, we look at the optimization of TPBVPs. Two examples are presented – the solution of a Low-thrust Mars-Earth minimum time transfer by the method of differential corrections and the solution of another example using the method of differential inclusions.

3.4.3.1 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. (3.72-3.76). These equations are in the form of a fixed time problem with the final time as a free parameter.

$$J = \int_0^1 t_f d\tau \tag{3.72}$$

$$r' = t_f u \tag{3.73}$$

$$\theta' = t_f \frac{v}{r} \tag{3.74}$$

$$u' = t_f \left(\frac{v^2}{r} - \frac{\mu}{r^2} + \frac{2\varepsilon P}{I_{sp} g (m_0 + m' \tau)} \sin\alpha \right) \tag{3.75}$$

$$v' = t_f \left(\frac{-uv}{r} + \frac{2\varepsilon P}{I_{sp} g (m_0 + m' \tau)} \cos\alpha \right) \tag{3.76}$$

where the mass flow rate is given by

$$m' = -t_f \frac{2\varepsilon P}{\left(I_{sp} g\right)^2}$$

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

Equation (3.51) results in the following optimal control law

$$\alpha = \tan^{-1}\left(\frac{-\lambda_u}{-\lambda_v}\right) \tag{3.77}$$

The Hamiltonian for the system is defined as

$$H = t_f\left[1 + \lambda_r(u) + \lambda_\theta\left(\frac{v}{r}\right) + \lambda_u\left(\frac{v^2}{r} - \frac{\mu}{r^2} + \gamma\sin\alpha\right) + \lambda_v\left(\frac{-uv}{r} + \gamma\cos\alpha\right)\right] \tag{3.78}$$

The co-state differential equations are automatically generated using OCEA through Eq. (3.50) by differentiating the Hamiltonian given in Eq. (3.78). The system is comprised of 10 total equations including Eqs. (3.73-3.76), the four co-state differential equations, and Eqs. (3.79-3.80). Here, t_f is considered as a free parameter; therefore, we must satisfy an additional necessary condition as described in Eq. (3.55). Eq. (3.80) is the differential equation representation of this integral boundary condition given by Eq. (3.55).

$$t_f^{'} = 0 \tag{3.79}$$

$$z^{'} = \frac{\partial H}{\partial t_f} \tag{3.80}$$

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 which are consistent with conceived Mars sample return missions.

The desired initial and terminal boundary conditions are given in Table 3.8. For this problem we are using astronomical units for distance and non-dimensional time units where 1 AU = 1.4959965e11 m and 1 TU = 5.0226757e6 sec.

Table 3.8. Low-thrust Mars-Earth transfer initial and final conditions

	Initial Time ($\tau = 0$)	Final Time ($\tau = 1$)
$r(AU)$	1.524	1
θ	0	Free
$u(AU/TU)$	0	0
$v(AU/TU)$	0.81	1
z	0	0

We could eliminate λ_θ 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 system; 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 3.8. 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 in the position and velocity solutions.

For Case I, we choose the following good starting guess with optimal solution for the unknowns and iteration histories given in Tables 3.9 and 3.10.

Case I: Initial guess for unknowns: $\left\{\lambda_r, \lambda_u, \lambda_v, t_f\right\} = \left\{1.0, 1.0, 1.7, 4.4\right\}$

Table 3.9. Low-thrust Mars-Earth transfer first-order solution terminal errors for Case I

Iteration	$\delta r(\tau=1)$	$\delta u(\tau=1)$	$\delta v(\tau=1)$	$\delta z(\tau=1)$	$\varepsilon = \boldsymbol{\delta}^T \boldsymbol{\delta}$
0	0.124e-1	-0.412e-1	-0.455e-1	-0.850e0	0.727e0
1	0.552e-2	-0.454e-1	-0.232e-1	0.668e-2	0.268e-2
2	-0.102e-1	-0.102e-1	0.458e-2	0.155e-2	0.234e-3
3	0.551e-3	0.248e-3	-0.464e-3	0.303e-4	0.582e-6
4	0.243e-6	0.528e-6	-0.263e-6	0.518e-7	0.411e-12
5	-0.131e-11	-0.239e-12	0.113e-11	0.916e-13	0.306e-23
6	0.888e-15	0.891e-15	-0.222e-15	-0.393e-15	0.178e-29

First-order solution:

$$\left\{\lambda_r, \lambda_u, \lambda_v, t_f\right\} = \left\{6.822434207,\ 7.020330890,\ 10.852450714,\ 4.591579951\right\}$$

Table 3.10. Low-thrust Mars-Earth transfer first- and second-order solution terminal errors for Case I

Iteration	$\delta r(\tau=1)$	$\delta u(\tau=1)$	$\delta v(\tau=1)$	$\delta z(\tau=1)$	$\varepsilon = \boldsymbol{\delta}^T \boldsymbol{\delta}$
0	0.124e-1	-0.412e-1	-0.455e-1	-0.850e0	0.727e0
1	0.552e-2	-0.454e-1	-0.232e-1	0.668e-2	0.268e-2
2	**-0.239e-2**	**-0.177e-2**	**0.140e-2**	**0.239e-3**	**0.109e-4**
3	**0.182e-6**	**0.361e-7**	**-0.152e-6**	**0.560e-8**	**0.577e-13**
4	**0.388e-14**	**0.265e-14**	**-0.155e-14**	**-0.101e-14**	**0.256e-28**

First- and second-order solution:

$$\left\{\lambda_r, \lambda_u, \lambda_v, t_f\right\} = \left\{6.822434200,\ 7.020330887,\ 10.852450689,\ 4.591579951\right\}$$

Tables 3.9 and 3.10 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 3.10, bold indicates a second-order solution was used. Rapid convergence is found when utilizing second-order sensitivity calculations after two iterations as shown in Table 3.10. The time required for the solutions in Tables 3.9 and 3.10 require 24 second and 1 min 50 seconds, respectively. Both solutions agree to 8 digits. The time of flight is 267 days.

We also solved this problem using Matlab's constrained function minimization function (**fmincon**). The Matlab solution converges in a number of iterations equal to that of the first-order solution given in Table 3.9. In comparing the elapsed time for the solutions, we find that the OCEA first-order solution is minimally faster. With regard to accuracy, we find the Matlab solution is incapable of achieving better than 10^{-10} accuracy in the terminal errors. This result indicates that the solution by Matlab, which utilizes finite difference approximations, does not achieve the same accuracy as the OCEA approach which utilizes exact analytical partial derivative calculations.

For Case II, with a relatively poor starting guess the optimal solution for the unknowns and iteration histories is given in Tables 3.11 and 3.12.

Case II: Initial guess for unknowns: $\left\{\lambda_r, \lambda_u, \lambda_v, t_f\right\} = \left\{1.0, 1.0, 1.0, 4.0\right\}$

Table 3.11. Low-thrust Mars-Earth transfer first-order solution terminal errors for Case II

Iteration	$\delta r(\tau=1)$	$\delta u(\tau=1)$	$\delta v(\tau=1)$	$\delta z(\tau=1)$	$\varepsilon=\boldsymbol{\delta}^T\boldsymbol{\delta}$
0	-0.837e-1	0.180e0	0.106e0	-0.884e0	0.833e0
1	0.140e0	0.261e0	-0.237e-1	0.212e0	0.134e0
2	-0.588e-1	0.171e0	0.345e-1	0.219e0	0.825e-1
3	-0.377e0	-0.228e-1	0.251e0	0.852e-1	0.213e0
4	-0.111e0	0.579e-1	0.319e-1	0.840e-1	0.239e-1
5	-0.612e-1	-0.295e-1	0.414e-1	0.173e-2	0.634e-2
6	0.591e-2	0.533e-2	-0.548e-2	0.350e-3	0.935e-4
7	0.234e-4	0.153e-3	0.410e-5	0.449e-5	0.242e-7
8	-0.137e-6	-0.116e-6	0.895e-7	0.244e-8	0.405e-13
9	0.132e-12	0.121e-12	-0.837e-13	-0.490e-13	0.418e-25

First-order solution:

$$\left\{\lambda_r,\lambda_u,\lambda_v,t_f\right\}=\left\{6.822434207,\ 7.020330890,\ 10.852450714,\ 4.591579951\right\}$$

Table 3.12. Low-thrust Mars-Earth transfer first- and second-order solution terminal errors for Case II

Iteration	$\delta r(\tau=1)$	$\delta u(\tau=1)$	$\delta v(\tau=1)$	$\delta z(\tau=1)$	$\varepsilon=\boldsymbol{\delta}^T\boldsymbol{\delta}$
0	-0.837e-1	0.180e0	0.106e0	-0.884e0	0.833e0
1	0.140e0	0.261e0	-0.237e-1	0.212e0	0.134e0
2	-0.588e-1	0.171e0	0.345e-1	0.219e0	0.825e-1
3	-0.377e0	-0.228e-1	0.251e0	0.852e-1	0.213e0
4	-0.111e0	0.579e-1	0.319e-1	0.840e-1	0.239e-1
5	-0.612e-1	-0.295e-1	0.414e-1	0.173e-2	0.634e-2
6	0.591e-2	0.533e-2	-0.548e-2	0.350e-3	0.935e-4
7	**-0.623e-5**	**-0.442e-5**	**0.425e-5**	**-0.599e-6**	**0.768e-10**
8	**0.666e-14**	**0.221e-14**	**-0.488e-14**	**0.803e-15**	**0.737e-28**

First- and second-order solution:

$$\left\{\lambda_r,\lambda_u,\lambda_v,t_f\right\}=\left\{6.822434200,\ 7.020330887,\ 10.852450689,\ 4.591579951\right\}$$

Tables 3.11 and 3.12 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 3.12, although by only one iteration. The time required for the solutions in Tables 3.11 and 3.12 require 42 second and 1 min 41 seconds, respectively.

We provide details from a study of the domain of convergence of the first- and second-order algorithms. Here, we simply vary the initial guess by scaling the known optimal solution (a unit scaling corresponds to zero error). The results are given in Table 3.13. 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.

The results in Table 3.13 show the empirically determined 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 in this case. These results are consistent with those 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.

In summary, it should be kept in mind when examining these solutions that no partial derivatives were derived and coded by hand in producing the co-state differential equations and the state transition matrix differential equations. Obviously, the corresponding validation effort was also by-passed. Extending the method to second-order increased the computational burden; however, there is still no increase in analyst effort in going to higher-order since no hand derivation or additional coding, other than the general algorithm itself which is coded once and for all. The decrease in time for hand-derived 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 computational 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 one 4x4 partition is needed to solve by the method of differential corrections. Significant reductions in computational cost related to symmetry and sparsity have not been exploited in this work, and no advanced optimization techniques, e.g. incorporating line searching, have been attempted in these solutions. Therefore, they represent a worst case for computational overhead.

Table 3.13. Domain of convergence comparison for Low-thrust Mars-Earth transfer

$\alpha : \left(\boldsymbol{x}_{guess} = \alpha \boldsymbol{x}_{optimal} \right)$	**First-order iteration count**	**Second-order iteration count**
0.50	Does not converge	Does not converge
0.60	7	Does not converge
0.70	6	5
0.80	6	4
0.85	5	4
0.90	5	3
0.95	4	3
0.98	4	3
0.99	3	2
0.995	3	2
1.0	***1***	***1***
1.005	3	2
1.01	3	2
1.02	4	3
1.05	4	3
1.10	5	3
1.15	6	6
1.20	7	7
1.30	9	Does not converge
1.40	Does not converge	Does not converge
1.50	Does not converge	Does not converge

The results for each converged solution are equivalent to eight digits. Therefore, we show plots for the optimal motion only once. Figure 3.1 shows the Mars-Earth optimal trajectory. The velocity states and co-states are plotted in Figures 3.2 and 3.3 respectively. Figure 3.4 shows the optimal thrust angle control.

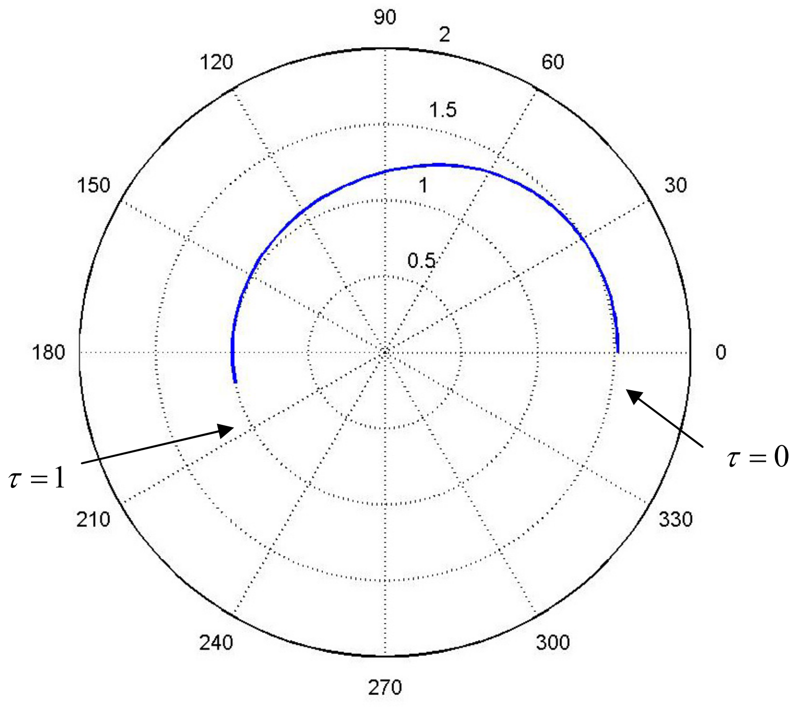

Figure 3.1. Mars-Earth optimal trajectory

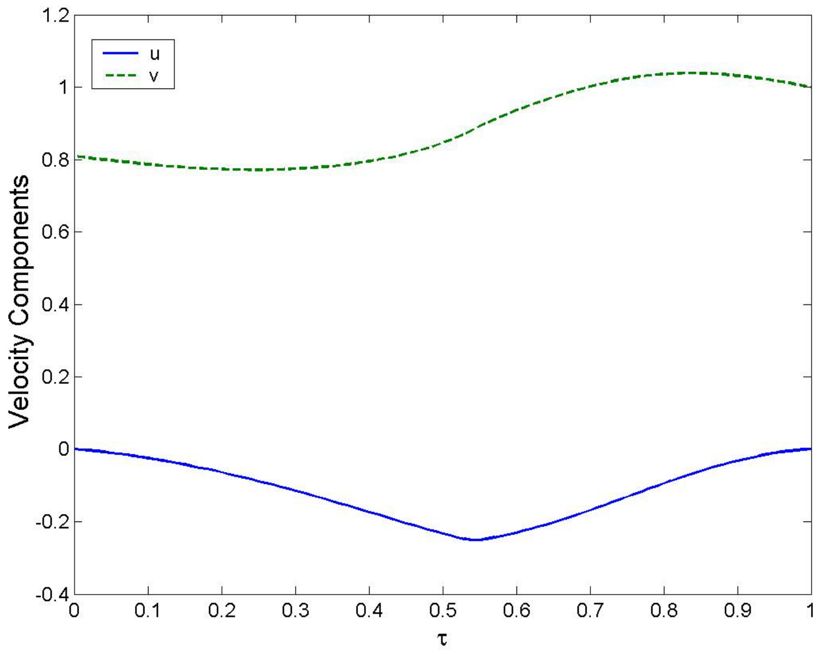

Figure 3.2. Velocity components for low-thrust Mars-Earth transfer

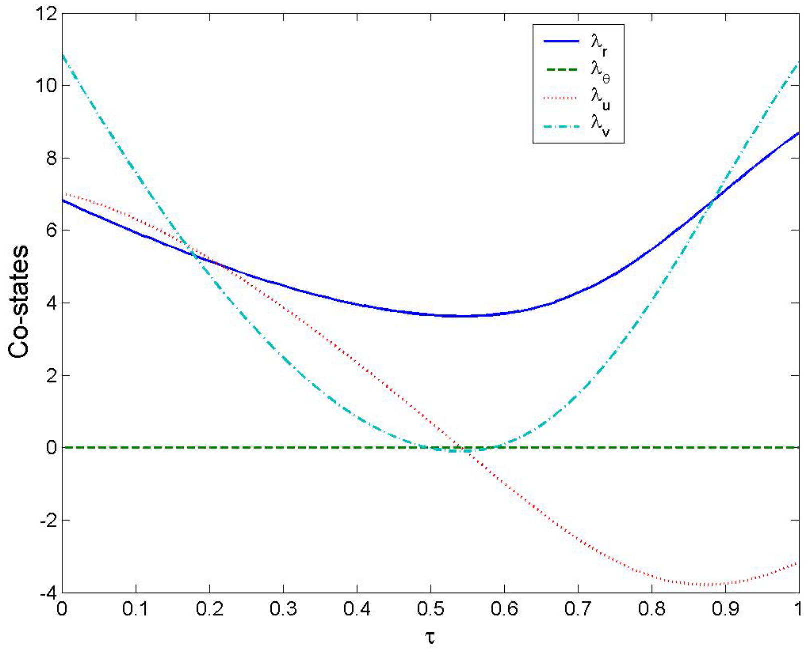

Figure 3.3. Co-states for low-thrust Mars-Earth transfer

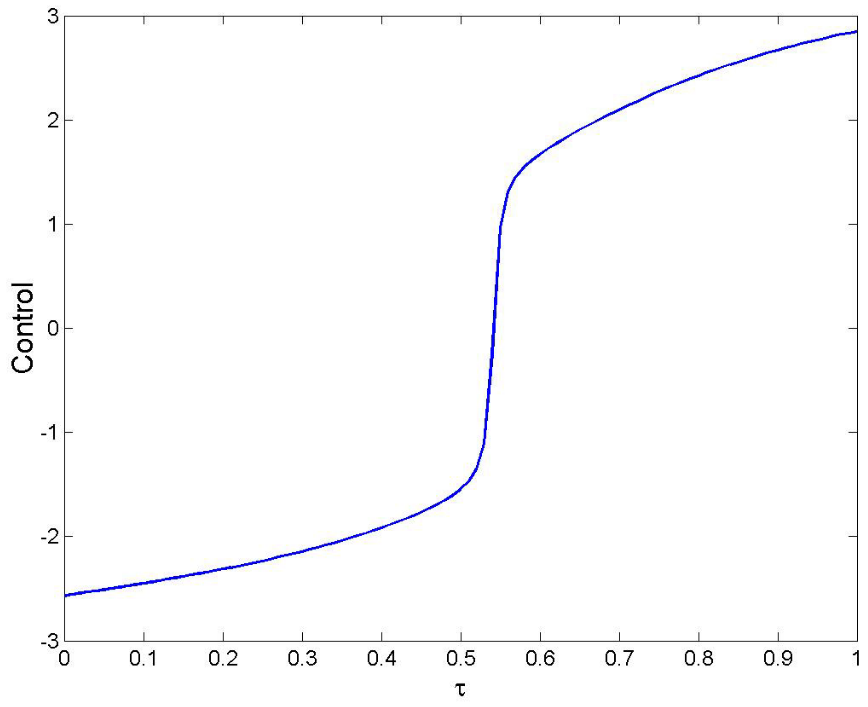

Figure 3.4. Optimal thrust angle for low-thrust Mars-Earth transfer

3.4.3.2 Differential Inclusions

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

$$\dot{x} = -x + \varepsilon x^3 + u \tag{3.81}$$

with performance index to be minimized given by

$$J = \frac{1}{2}\int_0^5 u^2 dt \tag{3.82}$$

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

$$\frac{x_{k+1}-x_k}{\Delta}+\frac{x_{k+1}+x_k}{2}+\varepsilon\left(\frac{x_{k+1}+x_k}{2}\right)^3=u_k \tag{3.83}$$

and

$$J=\frac{1}{2}\sum_{k=1}^{10}u_k^2 \tag{3.84}$$

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

Once the relation for the control given in Eq. (3.83) is substituted into Eq. (3.84) we have the cost as a function of the unknown states as

$$J = J(x_k); \qquad k = 2,3,...10 \tag{3.85}$$

Here, we consider using OCEA in an optimization approach in which we directly minimize the cost function. In summary, $\nabla J = \boldsymbol{0}$ defines a set of nonlinear equations ($\boldsymbol{g}(x_k) = \mathbf{0}$) which must be solved iteratively for the unknown states. Thus, the Hessian of *J*, $\nabla^2 J = \nabla \boldsymbol{g}$, provides the matrix of first-order sensitivities ($G = \nabla^2 J$) required for the solution. 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

$$\Delta \boldsymbol{x} = (G^T G)^{-1} G^T \boldsymbol{g}(\boldsymbol{x}) \tag{3.86}$$

We choose to solve this example using Matlab's Symbolic Toolbox in order to compare the effort in computing partial derivatives required for OCEA and Matlab solutions. 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. (3.85). 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. (3.85). 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 3.14 shows a comparison of solving the problem by first- and second-order methods. More rapid convergence is found for the second-order method, although only one iteration improvement is found.

Table 3.14. Differential inclusions residual errors

Iteration count	First-order error	Second-order error
1	1.4313e2	1.4313e02
2	2.1830e-1	1.8792e-2
3	2.9154e-7	4.0911e-12
4	2.6237e-18	8.6737e-30
5	8.3470e-30	

A comparison of the optimal solutions and that by differential inclusions is shown in Figure 3.5.

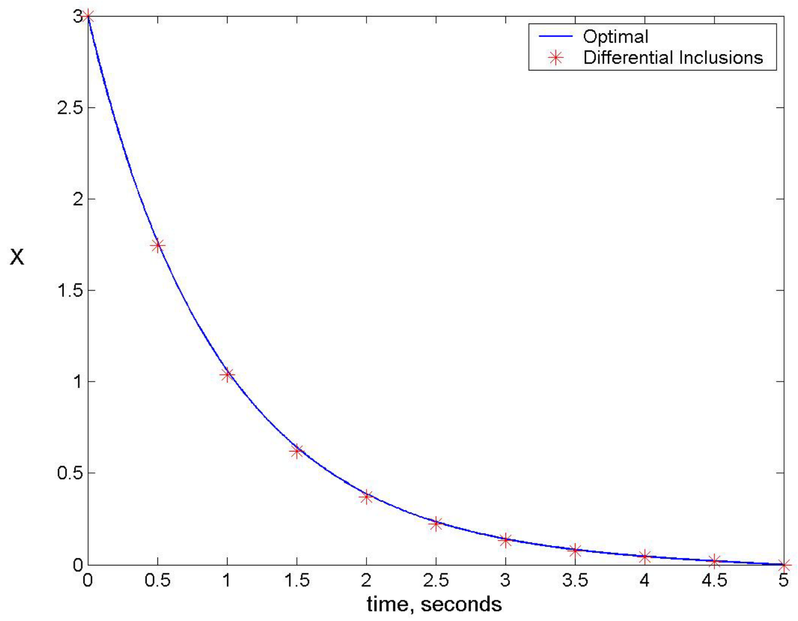

Figure 3.5. Optimal versus differential inclusions solution

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 **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 which would result in a more optimal solution; 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 solving a direct optimization problem.

3.5 SUMMARY

In this chapter, new methods for solving problems encountered in estimation, control and optimization of dynamical systems have been presented. Standard first-order algorithms were reviewed. Higher-order sensitivity calculations were presented for developing new algorithms. One particularly important development is the higher-order generalization of the state transition matrix. This development makes it possible to implement many of the new methods developed in this chapter including the higher-order GLSDC algorithms for orbit determination, the method of differential corrections for trajectory optimization, and higher-order algorithms for computing midcourse corrections. It is expected that the higher-order generalization of the state transition matrix will have an impact on nonlinear filtering applications as well.

The procedure for solving the numerical problems in this chapter show in general that OCEA significantly reduces the time required in going from the problem statement to a solution. In fact, many of these algorithms can be coded in a general manner once and for all. For example, all orders of the state transition matrix differential equations can be coded once and for all in a general code. In some cases, second-order methods provided only minimal improvement in rate of convergence. However, the added computational cost of the second-order methods is more than offset by the decrease in effort for analytical derivation and validation of partial derivatives which is bypassed by using OCEA. The results of this chapter indicate that OCEA has significant capability in solving problems in estimation and optimization; however, more work remains in optimizing the OCEA enabled algorithms.

CHAPTER IV

MODELING OF DYNAMICAL SYSTEMS[*]

In industry, research, and the classroom the study of dynamical systems by the derivation and integration of equations of motion is a rite of passage for computational scientists and engineers. For all but the simplest problems, however, this task can be laborious and error-prone. In fact, real-world applications requiring engineering-level fidelity models can take man-months to man-years of effort to develop and validate by hand. In this chapter, we consider a new direction in generating equations of motion for dynamical systems.

In the first section, we overview the current computational approaches for solving multibody dynamics problems. We discuss the different choices for equation of motion formulation in the second section. We go into greater detail in the third section in discussing the formulation chosen for in this work. Here we discuss generating equations of motion via Lagrange's Equations in a direct fashion. Additionally, a special form of Lagrange's Equation is presented when dealing with rigid bodies; and a general formulation is presented for flexible multibody systems. In the fourth section, an OCEA enabled approach for integrating the equations of motion is presented. Finally, several numerical examples are detailed in order to demonstrate the method.

[*] Portions of this chapter published in References 25 and 26.

4.1 OVERVIEW

The most common energy-based equation of motion generation method consists of Lagrange's method. There are two approaches for developing computer-based multibody dynamics models: (1) Symbolic methods that use computer symbol manipulation algorithms, and (2) Numerical codes that build the system kinematic and acceleration equations on-the-fly.

Symbolic manipulation engines such as Maple, Mathematica, or Mathcad are commercially available and theoretically capable of implementing Lagrange's method. These computer symbol manipulating systems are tasked for carrying out the required differentiations of the scalar Lagrangian function. The basic algorithm consists of three major steps: (1) The symbolic system Lagrangian function is built, (2) symbolic computation of the necessary derivatives are evaluated and the resulting equations are transformed to FORTRAN, and (3) The partial derivative models are assembled and introduced into Lagrange's equations. A serious limitation of this approach is that a model revision or engineering design change corrupts the assumed model for the system Lagrangian and forces extensive derivation and computer code revisions for all of the system partial derivatives. A further drawback of symbolic-based methods is that the resulting symbolic files can be huge and the resulting software is generally only machine-readable.

Purely numerically-based algorithms implement a generic solution algorithm for linking systems of rigid bodies that can undergo large and rapid relative motions. Examples of commercially available tools for the automated generation and integration of equations of motion include DADS, DISCOS, and ADAMS. Though powerful and broadly applicable, these tools use rigid algorithms and are generally not amenable for introducing approximations, unique coordinate choices, or testing advanced solution algorithms. It is very difficult to generate linearized equations of motion or partial derivative models for the purely numerically-based class of algorithms.

The focus of this chapter is a new method for the automatic generation and integration of equations of motion using operator overloading techniques for overcoming the limitations of both the symbolic and numerical multibody modeling and simulation tools[9]. The new method combines the best of the symbolic and numerical equation of motion generation methods, while retaining the simplicity and elegance of the Lagrangian method. An added benefit of the operator overloaded approach is that new approximation strategies and computational algorithms are easily introduced and evaluated.

4.2 EQUATIONS OF MOTION FORMULATION

For many years, a primary focus of dynamics has been the development of methods for generating equations of motion. The classical formulations include Newton/Euler methods, D'Alembert's equations, the Lagrangian energy approach, and Hamiltonian approaches[27-28]. More recent approaches have been developed in the past century including the Gibbs-Appell equations, the Boltzmann/Hammel equations, and Kane's equations.

Each of these methods has specific advantages and shortcomings when it comes to implementing them in multibody dynamics codes. A brief overview of some of these issues is given in Reference 29, and is summarized again here. Newton/Euler methods are simple in form; however, algebraic elimination of constraints poses a significant drawback. This shortcoming is alleviated by using Lagrange's equations, because the topology constraints are automatically eliminated from the problem formulation at the joint interconnection points used to mechanically link the bodies. Very lengthy expressions for the Lagrangian function typically result which must then be differentiated with respect to generalized coordinates and generalized velocities. The sheer volume of the number of terms required for modeling the equations of motion, their mathematical complexity, coding complexity, storage and computational requirements become a shortcoming with this method. On the other hand, we only need to consider velocity-level kinematics at most in the problem formulation. This becomes a significant advantage since it reduces the analytical effort and places more of the task in the automated computational process.

Existing multibody dynamics codes are widely varied in the choice of formulation of the equations of motion[30]. The most popular choices are Kane's equations, Lagrange's equation, and Newton/Euler methods. When confronted with solving a typical classroom type dynamics problem, the analyst is faced with the first task of choosing which of these formulations is best suited for the analysis. In fact, for the most of the problems that can be solved by hand, the available methods are roughly equivalent when assessed from the standpoint of analyst/programmer effort (conservation of misery, although each method has a passionate school of proponents!). For simple problems, the choice of equation of motion formulation method is rendered essentially, a matter of personal taste and experience.

However, the generation of equations of motion for complicated systems cannot be done by hand, and in fact many real-world applications can require man-weeks to man-months or beyond for completing. The applications force many issues to be addressed, including handling a fairly wide number of bodies, many fixed and time-varying constraints, complicated geometries, various coordinate choices, and body flexibility. The solution approaches via multibody dynamics codes are also fairly wide ranging. The choices depend on dominant engineering modeling goals for the simulation. For example, the ADAMS code is well suited for mechanisms studies; DISCOS handles light-weight low-frequency flexible bodies typical of satellites; DADS is geared to handling heavy high-frequency applications containing many closed-loop topologies typical of stiff multibody systems, automotive, construction, and agricultural applications.

One major result of this work is a demonstration of the use of the automatic differentiation capability of OCEA to generate equations of motion. The most obvious dynamic formulation to utilize with automatic differentiation is Lagrange’s equations. As previously mentioned, one drawback to this approach is forming and evaluating a potentially lengthy Lagrangian partial derivative expression. Modified forms of Lagrange’s formulation are presented in this work which take advantage of simplified special structure for certain classes of systems.

The end goal of this work is *to model and simulate the behavior of linked mechanical systems*. Special emphasis is given to the formulation of the equations of motion. The remainder of the chapter is summarized as follows: (1) automatic generation of equations of motion via Lagrange's Method, (2) calculation of the system mass matrix, (3) Solutions for open-loop chains of rigid body systems, (4) a generalization of the kinetic energy expression for multiple flexible bodies, and (5) solutions for open- and closed-loop chains of flexible body systems.

In the course of the preparation of this chapter, it was discovered that some work had been published regarding the use of automatic differentiation for deriving equations of motion in the same time frame as this work[31]. In Reference 31 the automatic differentiation program ADIFOR 2.0 was used to produce FORTRAN 77 subroutines for the computation of the first-order derivatives of the constraint equations. Since ADIFOR 2.0 only computes first-order partials, the automatic differentiation package AUTODERIVE was additionally used to produce the code for the second-order derivative terms. Here we make the point that the OCEA approach requires no code generation since all partials can be compute within the OCEA-FORTRAN environment. Furthermore, first- through fourth-order partials are readily computed, thus OCEA solves the problem by itself. A comparison of the accuracy of this method versus the new OCEA enabled method will be presented later in this chapter.

4.3 FORMULATION VIA LAGRANGE'S EQUATION

4.3.1 General Formulation

In the Lagrangian formulation, partial derivatives of energy functions are utilized to produce the equations of motion. An obvious advantage of the Lagrangian formulation over, for example, Newton/Euler methods is that only velocity level kinematic expressions need to be developed in order to specify the energy functions, specifically kinetic energy. In this section, the framework for solving a class of problems in which formulation of kinetic and potential energy functions are readily formed is presented. The main result here is that automatic differentiation tools are perfectly suited for directly implementing the Lagrangian formulation in solving this important class of engineering problems.

Equation (4.1) presents the most general form of Lagrange's equations, including generalized forces and constraint forces.

$$\frac{d}{dt}\left(\frac{\partial L}{\partial \dot{\boldsymbol{q}}}\right)-\frac{\partial L}{\partial \boldsymbol{q}}=\boldsymbol{Q}+C^{T}\boldsymbol{\lambda} \tag{4.1}$$

subject to

$$C\dot{\boldsymbol{q}}=\boldsymbol{b}$$

where the Lagrangian is defined as $L=T-V$, $\boldsymbol{Q}$ are the generalized forces, C is the constraint matrix, and $\boldsymbol{\lambda}$ is the Lagrange multiplier vector. In the most general form, kinetic energy (T) is written as a function of the generalized coordinates ($\boldsymbol{q}$) and the generalized velocities ($\dot{\boldsymbol{q}}$), and the potential energy (V) is a function of the generalized coordinates.

The implied derivatives with respect to the generalized coordinates and generalized velocities in these equations can be readily computed by automatic differentiation by simply specifying the Lagrangian function. Organizing these equations in a manner in which they can be integrated; however, requires explicitly solving for the acceleration terms, which are buried in the first term of Eq. (4.1). In order to proceed down this path, we can rewrite Eq. (4.1) in the following form since the potential energy has no dependence on the generalized velocities.

$$\frac{d}{dt}\left(\frac{\partial T}{\partial \dot{\boldsymbol{q}}}\right)-\frac{\partial L}{\partial \boldsymbol{q}}=\boldsymbol{Q}+C^T\boldsymbol{\lambda} \tag{4.2}$$

For the time being, we focus our attention on the first term in Eq. (4.2), and the most common case of a natural system, $T=T(\dot{\boldsymbol{q}})=\frac{1}{2}\dot{\boldsymbol{q}}^T M(\boldsymbol{q})\dot{\boldsymbol{q}}$. The first term in Eq. (4.2) can be rewritten as

$$\begin{aligned}\frac{d}{dt}\left(\frac{\partial T}{\partial \dot{q}_i}\right)&=\frac{\partial^2 T}{\partial \dot{q}_i\partial \dot{q}_j}\ddot{q}_j+\frac{\partial^2 T}{\partial \dot{q}_i\partial q_j}\dot{q}_j\\ &=m_{ij}\ddot{q}_j+\dot{m}_{ij}\dot{q}_j\end{aligned} \tag{4.3}$$

where

$$M \leftrightarrow m_{ij}=\frac{\partial^2 T}{\partial \dot{q}_i\partial \dot{q}_j} \tag{4.4}$$

$$\dot{M} \leftrightarrow \dot{m}_{ij}=\frac{\partial^2 T}{\partial \dot{q}_i\partial q_j} \tag{4.5}$$

As is shown in Eq. (4.3), we can compute both the mass matrix and its time derivative by second order differentiation of the kinetic expression. It is well known that the mass matrix can be formulated in this fashion. However, it is not obviously apparent that the time derivative of the mass matrix can be computed in this fashion as well since the time derivative of the mass matrix can also be computed by third order differentiation, that is, by once differentiating the mass matrix with respect to time. A proof of this fact is given in Appendix F.

By utilizing automatic differentiation, the constraint matrix can also be formed automatically. Let's consider a holonomic constraint of the following form.

$$\phi(\boldsymbol{q},t)=\boldsymbol{0} \tag{4.6}$$

The Pfaffian form of this constraint is developed by time differentiating Eq. (4.6).

$$\begin{aligned}\dot{\phi}(\boldsymbol{q},t)&=\frac{\partial\phi}{\partial\boldsymbol{q}}\dot{\boldsymbol{q}}+\frac{\partial\phi}{\partial t}\\&=C\dot{\boldsymbol{q}}+\frac{\partial\phi}{\partial t}\end{aligned} \tag{4.7}$$

Therefore, the constraint matrix is simply computed by differentiating the holonomic constraint with respect to the generalized coordinates, $C=\frac{\partial\phi}{\partial\boldsymbol{q}}$.

With Eqs. (4.4), (4.5), and (4.7), we arrive at the following form for Lagrange's equations,

$$M\ddot{\boldsymbol{q}}+\dot{M}\dot{\boldsymbol{q}}-\frac{\partial L}{\partial\boldsymbol{q}}=\boldsymbol{Q}+C^T\boldsymbol{\lambda} \tag{4.8}$$

which are solved for the accelerations as follows

$$\ddot{\boldsymbol{q}}=M^{-1}\left(-\dot{M}\dot{\boldsymbol{q}}+\frac{\partial L}{\partial\boldsymbol{q}}+\boldsymbol{Q}+C^T\boldsymbol{\lambda}\right) \tag{4.9}$$

As has been shown above, generating the equations given in Eq. (4.9) can be accomplished by simply specifying the Lagrangian function, the constraint relation, and the generalized forces. OCEA accomplishes all required derivative operations leading to the right hand side of Eq. (4.9). The background second partials of T, for example, can be assessed to obtain M from Eq. (4.4).

4.3.2 Specialized Formulation for Rigid Bodies

The need to explicitly form the Lagrangian function in the above formulations can be alleviated by looking at a modified form[20] of Eq. (4.1).

$$M(\boldsymbol{q})\ddot{\boldsymbol{q}} + \boldsymbol{G}(\boldsymbol{q},\dot{\boldsymbol{q}}) + \frac{\partial V(\boldsymbol{q})}{\partial \boldsymbol{q}} = \boldsymbol{Q} + C^T \boldsymbol{\lambda} \tag{4.10}$$

The Coriolis forces (mass matrix time derivatives) are accounted for in the function $\boldsymbol{G}(\boldsymbol{q},\dot{\boldsymbol{q}})$.

$$\boldsymbol{G}(\boldsymbol{q},\dot{\boldsymbol{q}}) = \begin{bmatrix} \dot{\boldsymbol{q}}^T H^{(1)} \dot{\boldsymbol{q}} & \dots & \dot{\boldsymbol{q}}^T H^{(n)} \dot{\boldsymbol{q}} \end{bmatrix} \tag{4.11}$$

where the elements of the Christoffel operator $H^{(i)} = H^{(i)}(\boldsymbol{q})$ are generated by

$$h_{jk}^{(i)} = \frac{1}{2}\left(\frac{\partial m_{ij}}{\partial q_k} + \frac{\partial m_{ik}}{\partial q_j} - \frac{\partial m_{jk}}{\partial q_i} \right) \tag{4.12}$$

It is apparent from Eq. (4.10) that accelerations can be computed by identifying the mass matrix, computing $\boldsymbol{G}(\boldsymbol{q},\dot{\boldsymbol{q}})$ from spatial derivatives of the mass matrix elements, and differentiation of the potential energy function. These accelerations are can be obviously written as

$$\ddot{\boldsymbol{q}} = M^{-1}\left(-\boldsymbol{G} - \frac{\partial V}{\partial \boldsymbol{q}} + \boldsymbol{Q} + C^T \boldsymbol{\lambda}\right) \tag{4.13}$$

Now we consider an approach for generating the mass matrix and its derivatives using Eq. (4.12) for planar rigid body chain systems.

Rigid body chain topologies are encountered in many robotics and industrial machinery applications. The assumption here is that the size of the overall motions of the bodies is much greater than their deformations. When this is the case, rigid body dynamical formulations provide suitable models.

The Modified Lagrangian Formulation presented above is well suited for automating the process of generating equations of motion for rigid body systems. It is shown here that the mass matrix need not be computed by twice differentiating the kinetic energy function. The advantage gained in the developments of this section is that only first-order derivatives of the position vectors locating the mass centers of the bodies are required for computing the mass matrix. In addition, second-order derivatives of these position vectors are needed to generate the mass matrix derivatives.

Let's consider a system of rigid bodies translating and rotating in planar (2D) motion. The kinetic energy for this system of bodies can be written as

$$T(\boldsymbol{v}, \boldsymbol{\omega}) = \frac{1}{2}\sum_{i=1}^{n}\left(m_i v_i^T v_i + I_i \omega_i^T \omega_i\right) \tag{4.14}$$

where m_i and I_i are the mass and mass moment of inertia about the center of mass for the ith body, $\boldsymbol{v}_i$ is the velocity of the center of mass, $\boldsymbol{\omega}_i$ is the angular velocity of the ith body, and n is the total number of bodies.

We can rewrite Eq. (4.14) in terms of the generalized coordinates and generalized velocities by introducing the following transformations.

$$\boldsymbol{v}_i = A_i(\boldsymbol{q})\dot{\boldsymbol{q}} \tag{4.15}$$

$$\boldsymbol{\omega}_i = B_i(\boldsymbol{q})\dot{\boldsymbol{q}} \tag{4.16}$$

In terms of the generalized coordinates and generalized velocities, the kinetic energy expression can be rewritten using Eqs. (4.15) and (4.16).

$$\begin{aligned} T(\boldsymbol{q},\dot{\boldsymbol{q}}) &= \frac{1}{2}\sum_{i=1}^{n}\left(m_i\dot{q}_i^T A_i^T A_i\dot{q}_i + I_i\dot{q}_i^T B_i^T B_i\dot{q}_i\right) \\ &= \frac{1}{2}\sum_{i=1}^{n}\left(\dot{q}_i^T\left[m_iA_i^TA_i + I_iB_i^TB_i\right]\dot{q}_i\right) \\ &= \frac{1}{2}\sum_{i=1}^{n}\left(\dot{q}_i^T M(\boldsymbol{q})\dot{q}_i\right) \end{aligned} \tag{4.17}$$

The mass matrix is immediately identified as given in Eq. (4.18).

$$M(\boldsymbol{q}) = \sum_{i=1}^{n} m_iA_i^TA_i + I_iB_i^TB_i \tag{4.18}$$

The procedure for computing the mass matrix now depends on the masses and inertias (lengths) of the bodies, and the two transformation matrices for each body.

Upon closer inspection of Eqs. (4.15) and (4.16), we can write the following

$$\boldsymbol{v}_i = \frac{\partial \boldsymbol{r}_i}{\partial \boldsymbol{q}}\dot{\boldsymbol{q}} = \frac{\partial \dot{\boldsymbol{r}}_i}{\partial \dot{\boldsymbol{q}}}\ddot{\boldsymbol{q}} \tag{4.19}$$

$$\boldsymbol{\omega}_i = \frac{\partial \theta_i}{\partial \boldsymbol{q}}\dot{\boldsymbol{q}} = \frac{\partial \dot{\theta}_i}{\partial \dot{\boldsymbol{q}}}\ddot{\boldsymbol{q}} \tag{4.20}$$

It is immediately obvious from comparing Eq. (4.19) with Eq. (4.15) and Eq. (4.20) with Eq. (4.16) that

$$A_i = \frac{\partial \boldsymbol{r}_i}{\partial \boldsymbol{q}} = \frac{\partial \dot{\boldsymbol{r}}_i}{\partial \dot{\boldsymbol{q}}} \tag{4.21}$$

and

$$B_i = \frac{\partial \theta_i}{\partial \boldsymbol{q}} = \frac{\partial \dot{\theta}_i}{\partial \dot{\boldsymbol{q}}} \tag{4.22}$$

Therefore the mass matrix can be computed by taking first partials of the position vectors locating the mass center of the bodies, thus simplifying the formation of the transformation matrices.

Up to this point, we have not discussed the choice for generalized coordinates which is an important matter for an automated process. Implicit in the above developments is the assumption that the chosen generalized coordinates are the angular rotations of the bodies. Fundamental choices for the generalized coordinates include (1) the mass center locations (x, y) and angular rotations (θ) and (2) the angular rotations only. For a planar open chain system (with $n > 1$), for choice (1) we have a system that is over-parameterized while with choice (2) we obtain a minimal coordinate unconstrained system is automatically produced. Within choice (2) there are also two choices of either absolute angles (all angles measured with respect to same frame) or relative angles (successive bodies angular rotation measured with respect to the orientation of the preceding body). For this formulation, we choose the minimal set of angular coordinates described by absolute measurement with respect to an inertial frame. A further description of the choice of coordinates for the multibody formulation can be found in References 32, 33, and 34.

The next step involves computing $\boldsymbol{G}(\boldsymbol{q},\dot{\boldsymbol{q}})$ for Eq. (4.13). Because this involves computing first order partials of the elements of the mass matrix, we can differentiate Eq. (4.18) with respect to the generalized coordinates, keeping in mind that the transformation matrices are functions of the generalized coordinates.

$$\frac{\partial M(\boldsymbol{q})}{\partial \boldsymbol{q}} = \sum_{i=1}^{n} m_i \left(\frac{\partial A_i^T}{\partial \boldsymbol{q}} A_i + A_i^T \frac{\partial A_i}{\partial \boldsymbol{q}} \right) + I_i \left(\frac{\partial B_i^T}{\partial \boldsymbol{q}} B_i + B_i^T \frac{\partial B_i}{\partial \boldsymbol{q}} \right) \tag{4.23}$$

These partials are substituted into Eq. (4.12) and we can then compute $\boldsymbol{G}(\boldsymbol{q},\dot{\boldsymbol{q}})$ by Eq. (4.11). In addition, conservative forces derived from the potential energy function are simply computed by differentiating the potential energy function with respect to the generalized coordinates.

A key point in these developments is that they can be generalized for n-body open-chain and closed-chain topologies. In forming the accelerations in Eq. (4.13), a generalized code allows the analyst to merely specify the number of bodies in the topology, their physical properties (mass and length), and the initial conditions. The method relies upon recursively forming the center of mass position vectors for each body in order to compute the transformation matrices and their spatial derivatives.

4.3.3 Formulation for Flexible Body Systems

When the rigid body assumption is used to model a dynamical system, there is no need to consider spatial integrals over the body for computing kinetic and potential energy expressions (i.e. the Lagrangian). However, when flexibility is considered we encounter a Lagrangian expression which is computed as an integral over the volume of the body, and, of course, we require additional coordinates to define deformations. For the case of slender beams, we can simplify this to an integral over the length of the body. Furthermore, what is often done to simplify the formulation of equations of motion for flexible dynamical systems is the introduction of approximations for the flexible motion coordinates that aid in producing a Lagrangian with no explicit dependence on the spatial coordinates. Examples of approximation techniques include the well known Finite Element Method[35,20] and Method of Assumed Modes[36]. In essence, these techniques make it possible to produce a Lagrangian for the flexible dynamical system of the same form as that of a rigid body dynamical system. Thus, once the spatial discretization approximations are utilized, we can proceed to generate equations of motion for a flexible dynamical system just as we do for a rigid body dynamical system by directly implementing Lagrange's Equations of the form of Eq. (4.1).

In this section, we demonstrate the use of OCEA in generating equations of motion for systems comprised of flexible elements. Toward this end, we develop recursive expressions for kinetic and potential energy functions for a series of linked flexible beams. Since we are not considering rapid angular motions of the beams, we model the beams using Euler-Bernoulli assumptions.

We now consider generating equations of motion for a chain of linked flexible bodies. The aim of this section is to generalize the formulation for multiple flexible bodies. It is assumed that the first link is pinned without translation, and successive links are joined with pins as shown in Figure 4.1. Up to this point, we have not discussed the choice for generalized coordinates which is an important matter for automating the process. Make note in Figure 4.1 that we choose absolute angular coordinates, which are measured with respect to a common frame, in this case the horizontal. Note the x_i axis connects the tips of the flexible members; thus the elastic deformation of each domain vanishes at the ends of that domain.

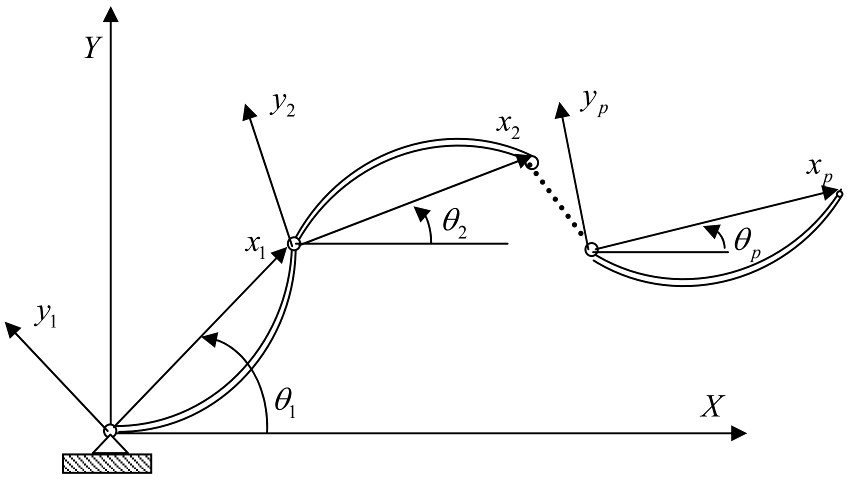

Figure 4.1. Geometry of multiple flexible link configuration

The main development of this section is a recursion for the kinetic energy of the (p+1)th link of the form:

$$T_{p+1} = \frac{1}{2}\int_0^{L_{p+1}} \rho_{p+1}(x_{p+1})\; \dot{\boldsymbol{r}}_{p+1}(x_{p+1},t)\cdot\dot{\boldsymbol{r}}_{p+1}(x_{p+1},t)\; dx_{p+1} \tag{4.24}$$

where $\rho_{p+1}(x_{p+1})$ is the mass density distribution and $\dot{\boldsymbol{r}}_{p+1}(x_{p+1},t)$ is the velocity expression for the (p+1)th link.

We begin the development of the velocity expression by looking at the first beam in the chain. The position and velocity of any point along the first beam are written as follows

$$\boldsymbol{r}_1(x_1,t) = x_1\hat{\boldsymbol{i}}_1 + v_1(x_1,t)\hat{\boldsymbol{j}}_1 \tag{4.25}$$

$$\dot{\boldsymbol{r}}_1(x_1,t) = -\dot{\theta}_1 v_1(x_1,t)\hat{\boldsymbol{i}}_1 + \left(\dot{v}_1(x_1,t) + x_1\dot{\theta}_1\right)\hat{\boldsymbol{j}}_1 \tag{4.26}$$

where $v_1(x_1,t)$ is the transverse beam deformation, $\dot{\theta}_1$ is angular velocity, and x_1 is the coordinate measurement along the beam frame of reference.

The velocity of any point along the second beam in the chain can be written as

$$\dot{\boldsymbol{r}}_2(x_2,t) = \dot{\boldsymbol{r}}_1(L_1,t) \ - \ \dot{\theta}_2 v_2(x_2,t)\hat{\boldsymbol{i}}_2 + \left(\dot{v}_2(x_2,t) + x_2\dot{\theta}_2\right)\hat{\boldsymbol{j}}_2 \tag{4.27}$$

and, in general, the velocity expression for the (p+1)th link can be written as

$$\begin{aligned}\dot{\boldsymbol{r}}_{p+1}(x_{p+1},t) = \dot{\boldsymbol{r}}_p(L_p,t) \ &- \ \dot{\theta}_{p+1} v_{p+1}(x_{p+1},t)\hat{\boldsymbol{i}}_{p+1} \\ &+ \left(\dot{v}_{p+1}(x_{p+1},t) + x_{p+1}\dot{\theta}_{p+1}\right)\hat{\boldsymbol{j}}_{p+1}\end{aligned} \tag{4.28}$$

where the velocity of the tip of the pth link, $\dot{\boldsymbol{r}}_p(L_p,t)$, is written as

$$\dot{\boldsymbol{r}}_p(L_p,t) = L_1\dot{\theta}_1\hat{\boldsymbol{j}}_1 + L_2\dot{\theta}_2\hat{\boldsymbol{j}}_2 + \ \ldots \ + L_p\dot{\theta}_p\hat{\boldsymbol{j}}_p = \sum_{i=1}^{p} L_i\dot{\theta}_i\hat{\boldsymbol{j}}_i \tag{4.29}$$

Here, we have enforced the zero tip deformation constraint by choosing admissible functions with zero deflection at the endpoints (i.e. $\phi_{p+1,i} = \sin(\frac{i\pi x_{p+1}}{L_{p+1}})$)[37]. In this way, the beams satisfy pinned-pinned boundary conditions. Additionally, the expression for beam tip velocity in Eq. (4.29) has been greatly simplified since $v_{p+1}(L_{p+1},t) = \dot{v}_{p+1}(L_{p+1},t) = 0$. Equation (4.28) can be written as

$$\begin{aligned}\dot{\boldsymbol{r}}_{p+1}(x_{p+1},t) = \sum_{i=1}^{p} L_i\dot{\theta}_i\hat{\boldsymbol{j}}_i \; &- \; \dot{\theta}_{p+1}v_{p+1}(x_{p+1},t)\hat{\boldsymbol{i}}_{p+1} \\ &+\left(\dot{v}_{p+1}(x_{p+1},t)+x_{p+1}\dot{\theta}_{p+1}\right)\hat{\boldsymbol{j}}_{p+1}\end{aligned} \tag{4.30}$$

Now, we can rewrite the kinetic energy expression of Eq. (4.24) as

$$\begin{aligned}T_{p+1} &= \frac{1}{2}\int_0^{L_{p+1}} \rho_{p+1}\, \dot{\boldsymbol{r}}_{p+1}(x_{p+1},t)\cdot\dot{\boldsymbol{r}}_{p+1}(x_{p+1},t)\, dx_{p+1} \\ &= \frac{1}{2}\int_0^{L_{p+1}} \rho_{p+1} \left\{ \begin{array}{l} \left(\sum_{i=1}^{p} L_i\dot{\theta}_i\hat{\boldsymbol{j}}_i\right)\cdot\left(\sum_{j=1}^{p} L_j\dot{\theta}_j\hat{\boldsymbol{j}}_j\right) \\ +2\left(\sum_{i=1}^{p} L_i\dot{\theta}_i\hat{\boldsymbol{j}}_i\right)\cdot\left\{-\dot{\theta}_{p+1}v_{p+1}\hat{\boldsymbol{i}}_{p+1}+\left(\dot{v}_{p+1}+x_{p+1}\dot{\theta}_{p+1}\right)\hat{\boldsymbol{j}}_{p+1}\right\} \\ +\dot{\theta}_{p+1}^2 v_{p+1}^2+\dot{v}_{p+1}^2+2x_{p+1}\dot{\theta}_{p+1}\dot{v}_{p+1}+x_{p+1}^2\dot{\theta}_{p+1}^2 \end{array} \right\} dx_{p+1}\end{aligned} \tag{4.31}$$

We can proceed further in simplifying (4.31) by carrying out the remaining dot products. If we consider absolute angular coordinates which are measured from a common reference frame, then we can write the following expressions that relate the two frames attached to any two links, here frames i and j $(i < j)$.

$$\begin{pmatrix}\hat{\boldsymbol{i}}_i \\ \hat{\boldsymbol{j}}_i\end{pmatrix} = \begin{bmatrix}\cos(\theta_j-\theta_i) & -\sin(\theta_j-\theta_i) \\ \sin(\theta_j-\theta_i) & \cos(\theta_j-\theta_i)\end{bmatrix}\begin{pmatrix}\hat{\boldsymbol{i}}_j \\ \hat{\boldsymbol{j}}_j\end{pmatrix} \tag{4.32}$$

At this point, we can introduce approximations in order to facilitate automatic generation of ODE's in terms of the time dependent variables by the assumed modes method. In Eq. (4.33) we introduce an expression for $v_{p+1}(x_{p+1},t)$:

$$v_{p+1}(x_{p+1},t) = q_{p+1,i}(t)\phi_{p+1,i}(x_{p+1}) = \boldsymbol{q}_{p+1}^{T}(t)\boldsymbol{\phi}_{p+1}(x_{p+1}) \tag{4.33}$$

We note here that the first index before the comma denotes the body (*p+1*), and the indices after the comma indicate the index of the element of the array (*i*) in the typical mathematical notation.

Now with the relation given by Eq. (4.32) and the approximation given by Eq. (4.33), we can write Eq. (4.31) as

$$\begin{aligned} T_{p+1} &= T_{P+1}(\boldsymbol{q},\dot{\boldsymbol{q}},\boldsymbol{\theta},\dot{\boldsymbol{\theta}}) \\ &= \tfrac{1}{2} m_{p+1} \sum_{i=1}^{p}\sum_{j=1}^{p} L_i\dot{\theta}_i L_j\dot{\theta}_j \cos\left(\theta_j-\theta_i\right) - \dot{\theta}_{p+1}\boldsymbol{q}_{p+1}^T\boldsymbol{b}_{p+1}\sum_{i=1}^{p} L_i\dot{\theta}_i \sin\left(\theta_{p+1}-\theta_i\right) \\ &+ \dot{\boldsymbol{q}}_{p+1}^T\boldsymbol{b}_{p+1}\sum_{i=1}^{p} L_i\dot{\theta}_i\cos\left(\theta_{p+1}-\theta_i\right) + \tfrac{1}{2}\dot{\theta}_{p+1}m_{p+1}L_{p+1}\sum_{i=1}^{p} L_i\dot{\theta}_i\cos\left(\theta_{p+1}-\theta_i\right) \\ &+ \tfrac{1}{2}\dot{\theta}_{p+1}^2\boldsymbol{q}_{p+1}^T M_{p+1}\boldsymbol{q}_{p+1} + \tfrac{1}{2}\dot{\boldsymbol{q}}_{p+1}^T M_{p+1}\dot{\boldsymbol{q}}_{p+1} + \dot{\theta}_{p+1}\dot{\boldsymbol{q}}_{p+1}^T\boldsymbol{a}_{p+1} + \tfrac{1}{6} m_{p+1}L_{p+1}^2\dot{\theta}_{p+1}^2 \end{aligned} \tag{4.34}$$

where $\boldsymbol{q}$, $\dot{\boldsymbol{q}}$, $\boldsymbol{\theta}$, and $\dot{\boldsymbol{\theta}}$ are the vectors containing all of the time dependent quantities for the flexible coordinates and the angular coordinates. The elemental mass matrix, M_{p+1}, and the vectors, $\boldsymbol{a}_{p+1}$ and $\boldsymbol{b}_{p+1}$ are given in Eqs. (4.35-4.37), respectively.

$$M_{p+1,ij} = \int_0^{L_{p+1}} \rho_{p+1}\phi_{p+1,i}\phi_{p+1,j}\,dx_{p+1} = \frac{m_{p+1}}{2}\delta_{ij} \tag{4.35}$$

$$a_{p+1,i} = \int_0^{L_{p+1}} \rho_{p+1}x_{p+1}\phi_{p+1,i}\,dx_{p+1} = -\frac{m_{p+1}L_{p+1}}{i\pi}\cos(i\pi) \tag{4.36}$$

$$b_{p+1,i} = \int_0^{L} \rho_{p+1}\phi_{p+1,i}\,dx_{p+1} = \frac{m_{p+1}L_{p+1}}{i\pi}\left(1-\cos(i\pi)\right) \tag{4.37}$$

Equation (4.34) is used to produce the kinetic energy for the second beam and so on for $p \geq 1$. The potential energy due to bending is given as

$$\begin{aligned} V_{p+1}(\boldsymbol{q}_{p+1}) &= \tfrac{1}{2} q_{p+1,i}q_{p+1,j}\int_0^{L} EI_{p+1}(x_{p+1})\ \phi''_{p+1,i}\phi''_{p+1,j}\ dx_{p+1} \\ &= \tfrac{1}{2}\boldsymbol{q}_{p+1}^T K_{p+1}\boldsymbol{q}_{p+1} \end{aligned} \tag{4.38}$$

where K_{p+1} is an elemental stiffness matrix.

The kinetic and potential energy of the first link are to be specified individually as given in Eqs. (4.39) and (4.40), respectively.

$$T_1 = \tfrac{1}{2}\dot{\theta}_1^2 \boldsymbol{q}_1^T M_1 \boldsymbol{q}_1 + \tfrac{1}{2}\dot{\boldsymbol{q}}_1^T M_1 \dot{\boldsymbol{q}}_1 + \dot{\theta}_1 \dot{\boldsymbol{q}}_1^T \boldsymbol{a}_1 + \tfrac{1}{6} m_1 L_1^2 \dot{\theta}_1^2 \tag{4.39}$$

$$V_1 = \tfrac{1}{2}\boldsymbol{q}_1^T K_1 \boldsymbol{q}_1 \tag{4.40}$$

With Eqs. (4.34) and (4.38-4.40) we can form the system level Lagrangian function explicitly in terms of the time dependent coordinates ($\boldsymbol{q}, \dot{\boldsymbol{q}}, \boldsymbol{\theta}, \dot{\boldsymbol{\theta}}$) for an arbitrary number of links and implement Lagrange's Equations of the standard form given in Eq. (4.1) in order to produce the equations of motion.

4.4 NUMERICAL INTEGRATION OF EQUATIONS OF MOTION

Along with the many choices for equation of motion formulation are also many choices for numerically integrating differential equations also exist[38]. The standard method for integrating nonlinear differential equations, such as those in Eq. (4.41), is the fourth-order Runge-Kutta algorithm.

$$\dot{\boldsymbol{x}} = \boldsymbol{f}(\boldsymbol{x}(t), t) \tag{4.41}$$

The essence of this method begins with first formulating first-order equations from the second-order equations of motion. These first-order equations provide a means to exactly compute the first-order derivatives of the states to be integrated (position and velocity). These first-order derivatives are then utilized to approximate the derivatives of position and velocity up through fourth-order. One form of the well known fourth-order Runge-Kutta algorithm is given by:

$$\boldsymbol{x}_{k+1} = \boldsymbol{x}_k + \frac{h}{6}\left[\boldsymbol{k}_0 + 2\boldsymbol{k}_1 + 2\boldsymbol{k}_2 + \boldsymbol{k}_3\right] \tag{4.42}$$

where

$$\begin{aligned} \boldsymbol{k}_0 &= \boldsymbol{f}(\boldsymbol{x}_k, t_k) \\ \boldsymbol{k}_1 &= \boldsymbol{f}(\boldsymbol{x}_k + {}^{\boldsymbol{k}_0}\!/_2, t_k + {}^{h}\!/_2) \\ \boldsymbol{k}_2 &= \boldsymbol{f}(\boldsymbol{x}_k + {}^{\boldsymbol{k}_1}\!/_2, t_k + {}^{h}\!/_2) \\ \boldsymbol{k}_3 &= \boldsymbol{f}(\boldsymbol{x}_k + \boldsymbol{k}_2, t_k + h) \end{aligned} \tag{4.43}$$

With the capability of OCEA to exactly compute up through fourth-order derivatives, it is possible to construct an integrator in which time derivatives of the state can be computed exactly through fifth-order. This type of integration, sometimes referred to as Taylor integration (or analytical continuation) for obvious reasons, is shown here.

$$\begin{aligned} \boldsymbol{x}(t+\Delta t) = \boldsymbol{x}(t) &+ \boldsymbol{f}(\boldsymbol{x}(t),t)\Delta t + \frac{1}{2!}\dot{\boldsymbol{f}}(\boldsymbol{x}(t),t)\Delta t^2 \\ &+ \frac{1}{3!}\ddot{\boldsymbol{f}}(\boldsymbol{x}(t),t)\Delta t^3 + \frac{1}{4!}\dddot{\boldsymbol{f}}(\boldsymbol{x}(t),t)\Delta t^4 \\ &+ \frac{1}{5!}\ddddot{\boldsymbol{f}}(\boldsymbol{x}(t),t)\Delta t^5 \end{aligned} \tag{4.44}$$

The first through fourth-order time derivatives of the function $\boldsymbol{f}(\boldsymbol{x}(t),t)$ are given in Appendix G.

Additionally, it should be noted that a fourth-order method can be constructed with fifth-order exact step size control because the fifth-order term can be computed exactly.

This capability is highly advantageous in that the integrator can be optimally tuned to the correct step size at each instant of time for a user specified accuracy!

In the examples in the next section, the integration scheme proposed in this section was not used for the simulations. It should be noted that when equations of motion are automatically generated, two orders of differentiations are consumed in this process. Thus we would require four additional orders of differentiation to implement this approach together with the equation of motion generation.

4.5 NUMERICAL EXAMPLES

4.5.1 Spring Pendulum

Here, we present an example in which the equations of motion are automatically generated and integrated by explicitly forming the Lagrangian function. The spring pendulum, as shown in Figure 4.2, is a simple two degree of freedom example which can be readily solved by hand. However, we present it here in order to demonstrate the method, which can be applied generally to solving additional problems.

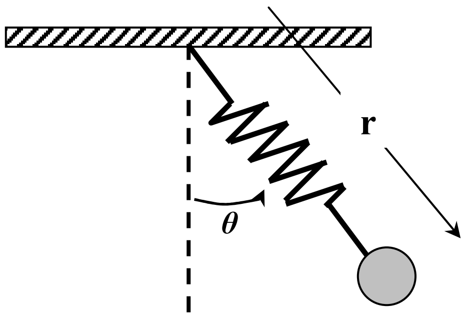

Figure 4.2. Spring pendulum

As was mentioned previously, in order to solve this type of problem we need to simply specify the Lagrangian function, the constraint relations (if they exist), and the generalized forces, and as well, the system physical parameters and initial conditions. The Lagrangian function is $L = T - V$, where

$$T = \tfrac{1}{2} m(\dot{r}^2 + r^2\dot{\theta}^2) \tag{4.45}$$

and

$$\begin{aligned} V &= V_{spring} + V_{grav} \\ &= \tfrac{1}{2} k(r - r_0)^2 + mg(r_0 - r\cos\theta) \end{aligned} \tag{4.46}$$

The values for m, k, and r_0 are 1 kg, 75 N/m, and 0.5 m, respectively. The initial conditions are chosen to be $\{r \quad \theta \quad \dot{r} \quad \dot{\theta}\} = \{0.55 \quad 0 \quad 0.01 \quad 2.5\}$. For this problem, no constraints need be specified and no additional forces need to be accounted for. However, it would be a simple matter to include force law expressions for damping elements or drag, or to modify the spring force model to account for effects such as nonlinearity. In addition to the integration parameters, these are the only parameters to be specified in order to compute the solution.

The subroutine containing the above specified problem data for the equations of motion for the spring pendulum is given in Appendix H. Here, it should be noted that the subroutine begins by including the EB_Handling routine (USE EB_HANDLING) which specifies that this subroutine contains data objects which are to be differentiated. Embedded variables and embedded functions are declared as TYPE(EB) which is an OCEA defined variable type. The specification of the kinetic and potential energy functions are highlighted in order to show that embedded functions are typed in a standard user-friendly manner. The structure of OCEA is such that by invoking EB_Handling, derivatives are automatically computed in the background without user intervention. Extraction of partial derivative information is also highlighted to show the ease with which this information can be accessed. Here, the analyst simply needs to define the dimension of the partial derivatives to be extracted. By setting this variable equal to the embedded function variable, for example, HES_L = L, we can readily extract the Hessian of the Lagrangian in order to get access to the mass matrix and its time derivative.

The overhead associated with deriving, coding, and validating the equations of motion has been avoided in computing the solutions for position and velocity which are shown in Figures 4.3 and 4.4.

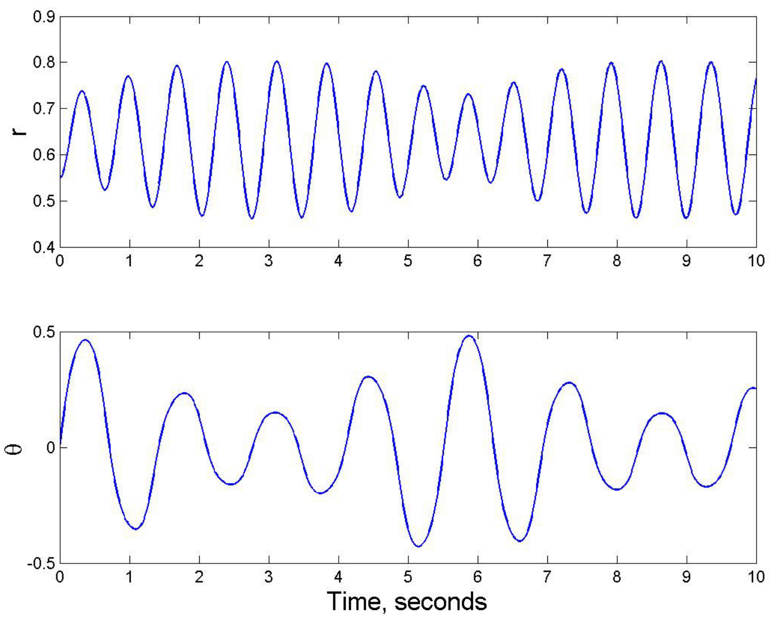

Figure 4.3. Spring pendulum position solution

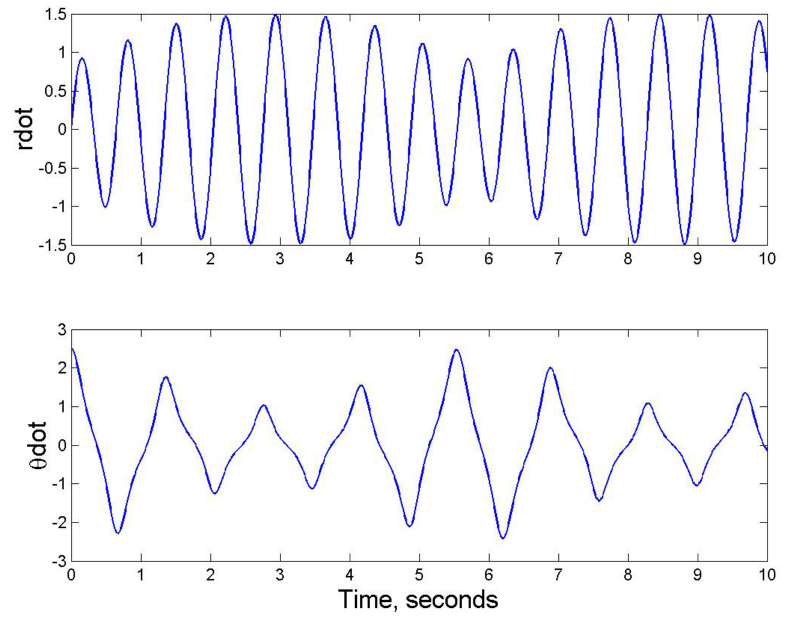

Figure 4.4. Spring pendulum velocity solution

4.5.2 Rigid Body Systems

4.5.2.1 Open Link Chain

In this section, the generalized n-body code based on the Modified Lagrangian Formulation is utilized to produce the solution for an open-chain topology of rigid bodies as shown in Figure 4.5. The mass matrix and its time derivatives are computed (See Eqs. 4.18 and 4.23) by specifying the position of the mass centers of the bodies, which are computed recursively. That is, the position of the mass center of body (i+1) can be computed from the mass center of the i^{th} body.

The example application is the deployment of an inflated beam aerospace structure. Inflated structures have been proposed as a means of creating large space structures on-orbit. Their primary advantages are low launch weight and low stowage volume. One proposed stowage method is folding the structure like an accordion. In this example, the

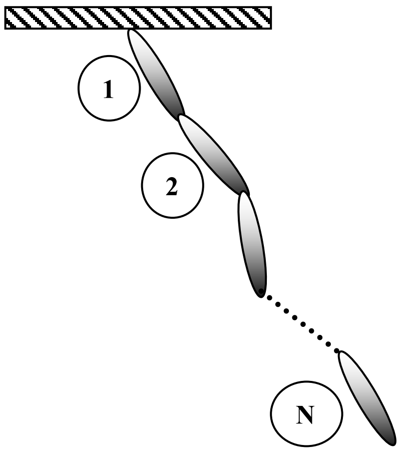

Figure 4.5. N-body open-chain geometry

inflated structure is modeled as a series of ten rigid links with an initial stowed accordion configuration. To simulate pressurization at the base of the first body, an initial angular velocity of -0.05 rad/sec is given to the body one. All additional initial angular velocities are chosen to be zero. Each of the ten bodies has mass of 0.01 kg and length of 0.5 m.

In order to simulate laboratory conditions, the effect of gravity is included in the potential energy description. Generalized forces due to damping at the hinge points is included in the simulation to account for any dampers, such as velcro, used to control the deployment. These generalized forces are of the form

$$Q_i = c_i(\dot{q}_{i-1} - \dot{q}_i) \tag{4.47}$$

where c_i is a positive constant. Computation of these generalized forces merely relies upon specifying the damping coefficients.

The deployment dynamics are plotted in Figure 4.6. At time zero, the structure is released and slowly begins to unfold. The force due to joint damping aids in controlling the deployment in the initial couple of seconds. The structure is nearly fully deployed at 5 seconds and comes to rest in the fully deployed configuration after about 10 seconds.

Model changes can be easily accommodated by this method. The advantage of having the ability to change the number of bodies in the model to quickly produce a new solution for the motion cannot be overstated. In addition, changes in models for generalized forces and key parameters such as mass, length, and damping can be easily accomplished.

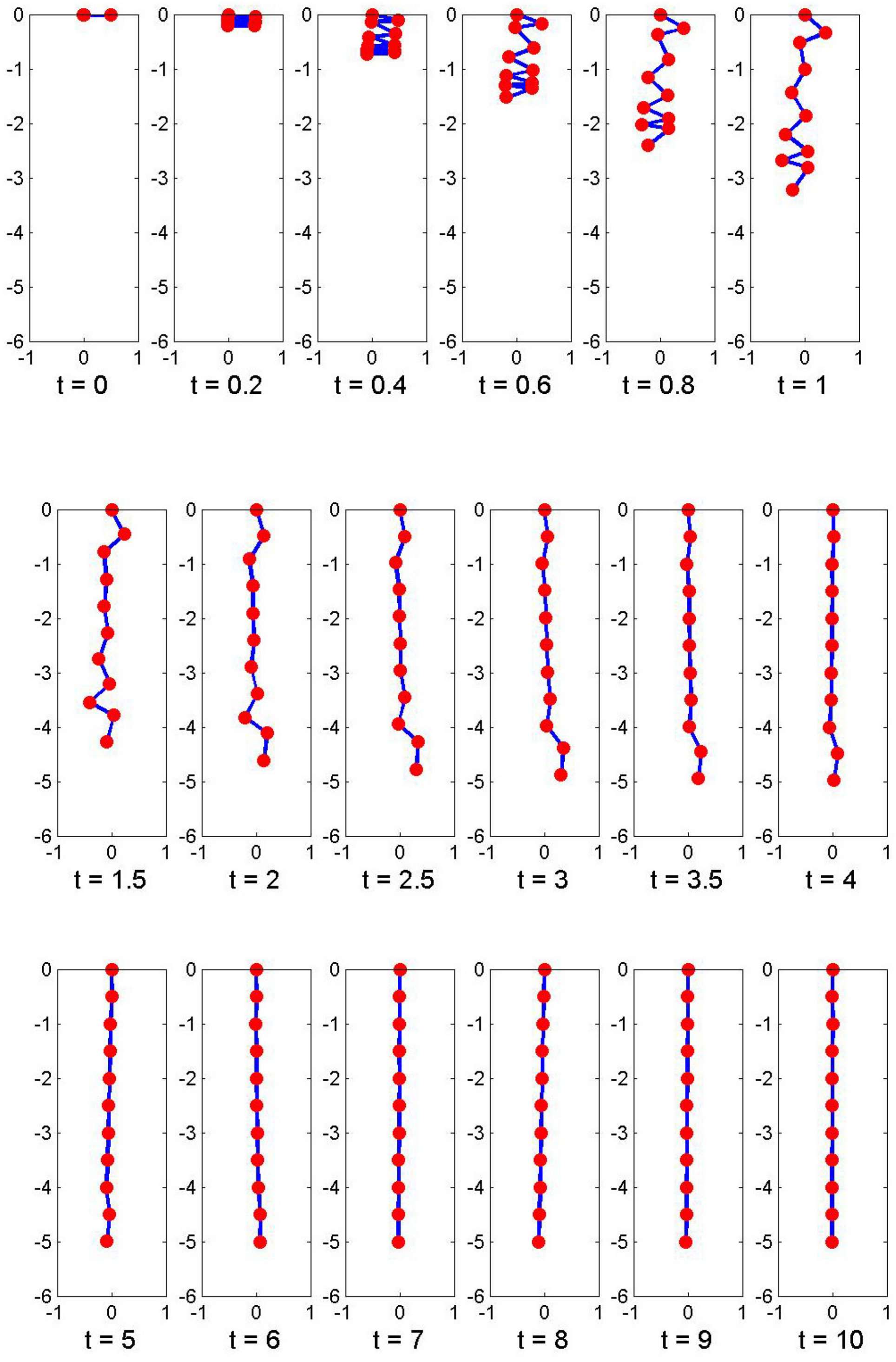

Figure 4.6. Deployment dynamics for ten link model

4.5.2.2 Closed Link Chain

We now consider the closed-chain system shown in Figure 4.7. Here we consider a five link example choosing to describe the system by the five absolute angles shown in the figure. The generalized n-body code, which was used to solve the previous deployment example, is again used here. As in the case of the previous example, all angles are measured with respect to a fixed reference, in this case the horizontal, and are chosen such that the positions of the mass centers can easily be computed recursively. Additionally, two holonomic constraints must be specified for this system. These constraints are used to automatically compute the constraint matrix as shown in Eq. (4.7).

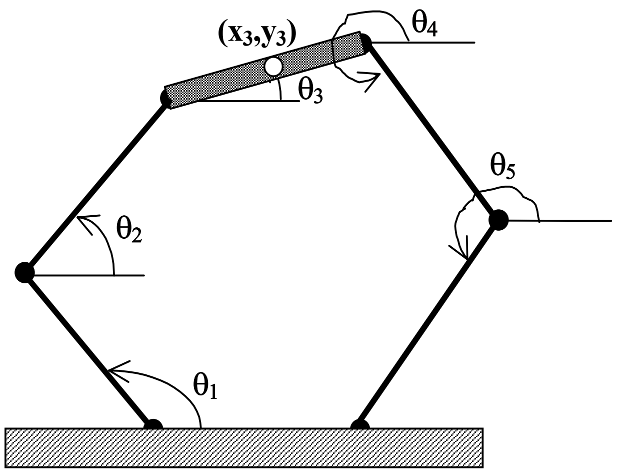

Figure 4.7. Five link closed-chain topology

Previous studies on this system include an analytical effort to generate a model for the system shown in Figure 4.7 for the purpose of studies in adaptive control[39-40]. The effort in generating the equations of motion in these papers should be compared to the present effort.

The 5 link system is solved for 10 seconds of the motion from rest with each link having mass of 2 kg and length of 1 m. The torques acting on the system are those due to gravity, a sinusoidal driving control specified on the first link, and damping at the joints. The Lagrange multipliers are solved using the so called Range-Space method (See Appendix I) in this example. The angular motion of the payload (link 3) is shown in Figure 4.8.

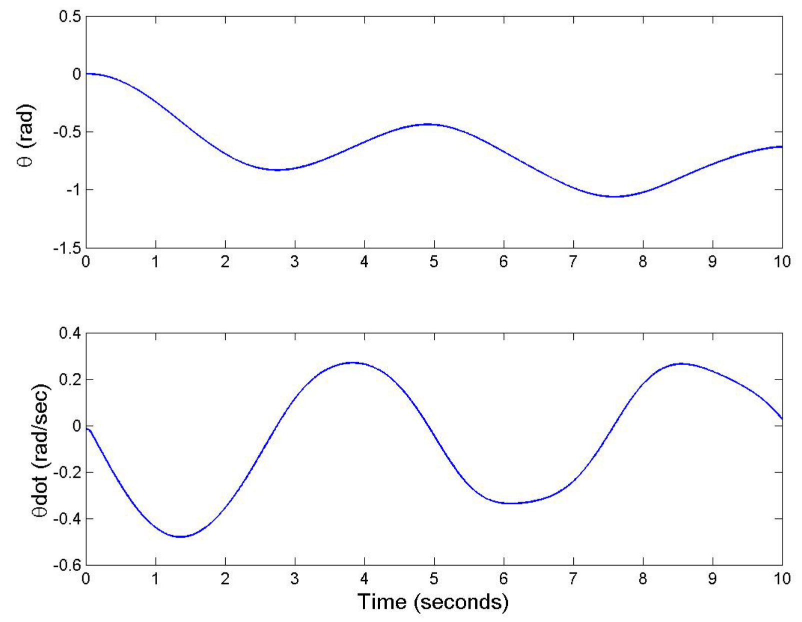

Figure 4.8. Payload motion for five link closed-chain system

Certainly, we can present any number of results. The main point here is an illustration of the rapid model generation capability of the generalized n-body code.

4.5.3 Flexible Body Systems

4.5.3.1 Open Link Chain

In this section, we present simulated results for multiple flexible beams in an open-chain topology. Here, we define the kinetic and potential energy for the first link with Eqs. (4.39) and (4.40), respectively. For the second link and so on (p=1 and so on) we define the kinetic and potential energy by Eqs. (4.34) and (4.38).

The system is comprised of three beams, each with mass of 12 kg, length of 10 m, and stiffness (EI) of 14e3 Nm^2. The beams are initially oriented with angles $\{\theta_1,\theta_2,\theta_3\} = \left\{\frac{3\pi}{2},\frac{3\pi}{2},\frac{3\pi}{2}\right\}$ as shown in Figure 4.1. All initial deflections are zero with the exception of the midpoint deflection of the third beam, which is $q_{3,1} = 0.01\,\text{m}$. All initial velocities are zero with the exception of the angular velocity of the third beam ($\dot{\theta}_3 = 0.5\frac{rad}{\sec}$).

With the ability to quickly generate models and solutions for the motion, a considerable number of analyses are readily available. Here, we show results for the rigid body and flexible contributions to the kinetic energy of the individual links as shown in Figures (4.9-4.11). Here we see that for this quite flexible system, the magnitudes of the kinetic energy due to flexibility (dotted lines) are of significant amplitude. The integration step size was chosen to satisfy constancy of total energy, which in this case was found (for the case of zero damping, of course) to be constant to 9 significant digits.

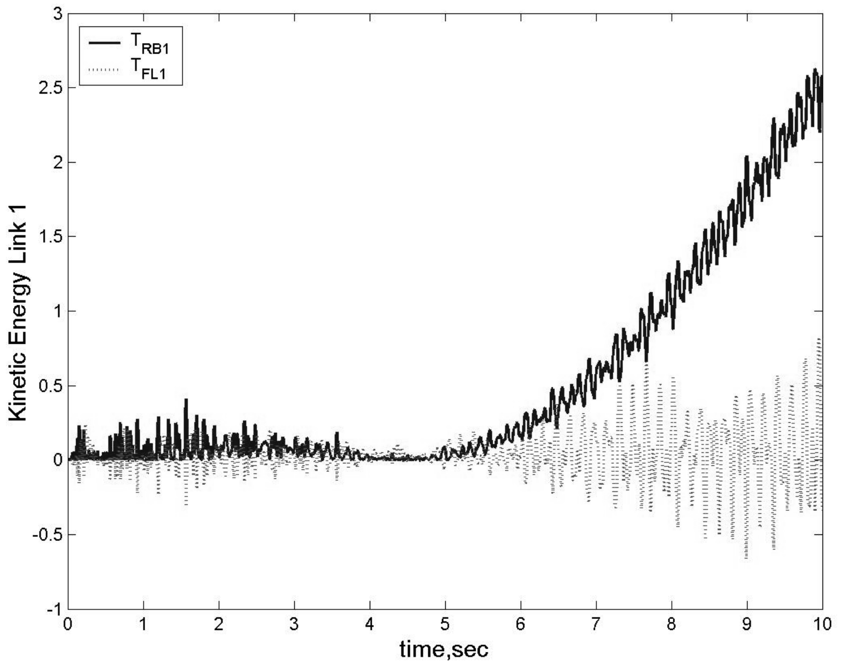

Figure 4.9. Kinetic energy for link one

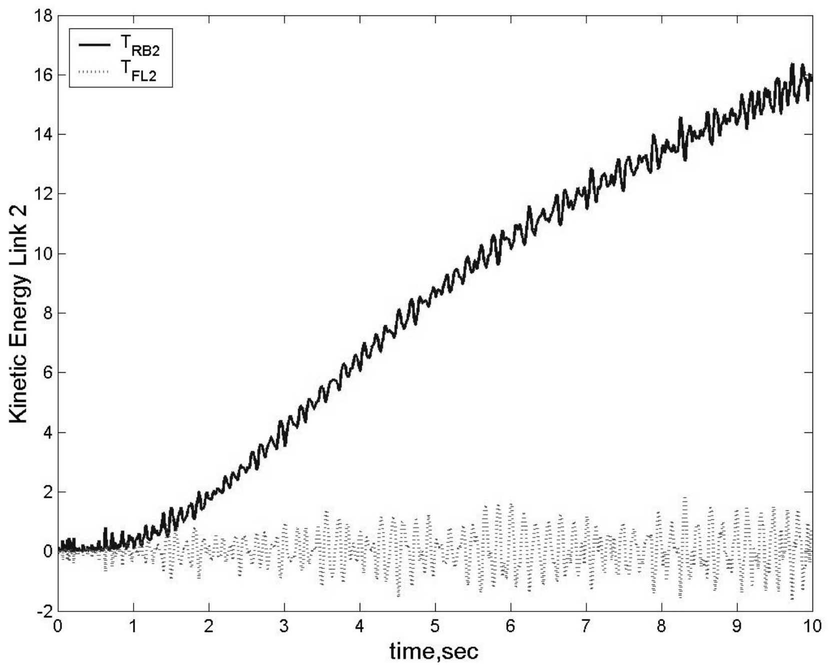

Figure 4.10. Kinetic energy for link two

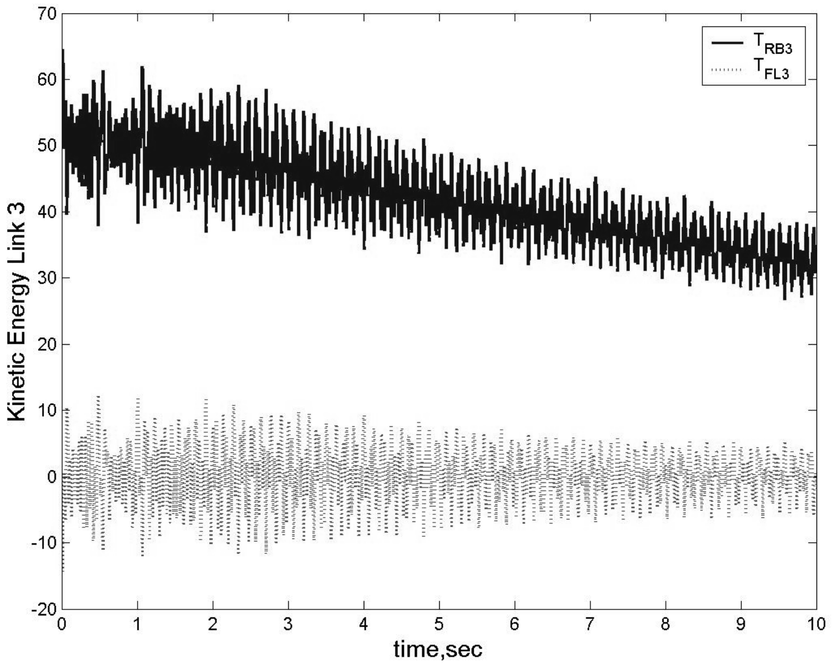

Figure 4.11. Kinetic energy for link three

4.5.3.2 Closed Link Chain

In this section, we present simulated results for multiple flexible beams in a closed-chain constrained topology. Here, we define the kinetic and potential energy for the links as we did before; however, in this case, we must simply compute the constraint forces in order to satisfy the geometric constraints, which are of the holonomic form:

$$\boldsymbol{\phi} = \begin{pmatrix} \sum_{i=1}^{n} L_i \cos(\theta_i) - D \\ \sum_{i=1}^{n} L_i \sin(\theta_i) \end{pmatrix} = \boldsymbol{0} \tag{4.48}$$

where n = 5 links and $D = 10\,\text{m}$. Constraint forces are computed by solving for the Lagrange multipliers using the Range-Space Method (Appendix I). Additionally, we include potential due to gravity from rest with initial angles for the links of $\{\theta_1,\theta_2,\theta_3,\theta_4,\theta_5\} = \left\{\frac{5\pi}{4},\frac{7\pi}{4},0,\frac{\pi}{4},\frac{3\pi}{4}\right\}$. Initially, there is no flexible energy in the system. Damping is included at all joints with the exception of the base joint at $\{X,Y\} = \{10,0\}$. The properties of the links is identical to the previous example with the exception that EI = 14e4 Nm. Again, conservation of total energy for the undamped case was used to determine the integration step size and to help validate the model.

Figure 4.12 shows a few snap shots of the motion history for the 10 second simulation.

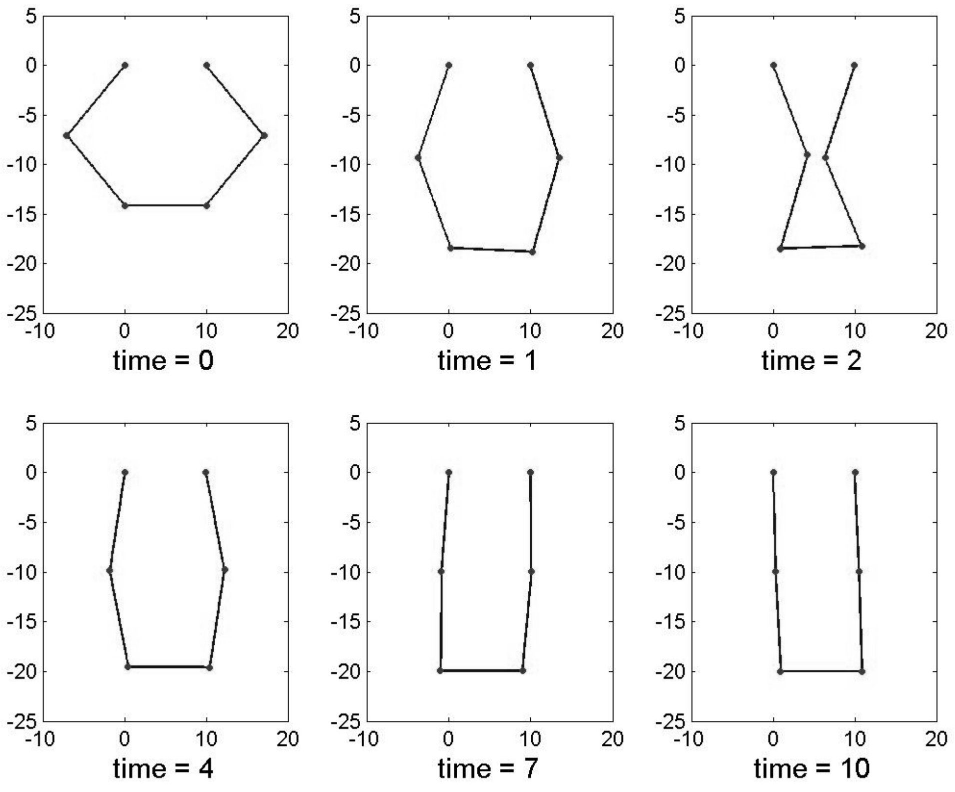

Figure 4.12: Motion for 5 link flexible closed-chain system

4.5.3.3 Planar Truss

Here we consider a four flexible link system containing flexible elements. We now consider an example which includes general translational motion. In forming the system Lagrangian, we must include a number of terms arising from translation in the kinetic energy expression. These additional terms are given in Appendix J.

We now focus on the simulation of this system as shown in Figure 4.13. The lines crossing the diagonal represent linear spring elements. The four links are pinned, and the vector $\boldsymbol{r}_A = \begin{bmatrix} x_A \\ y_A \end{bmatrix}$ locates the basepoint of the first link.

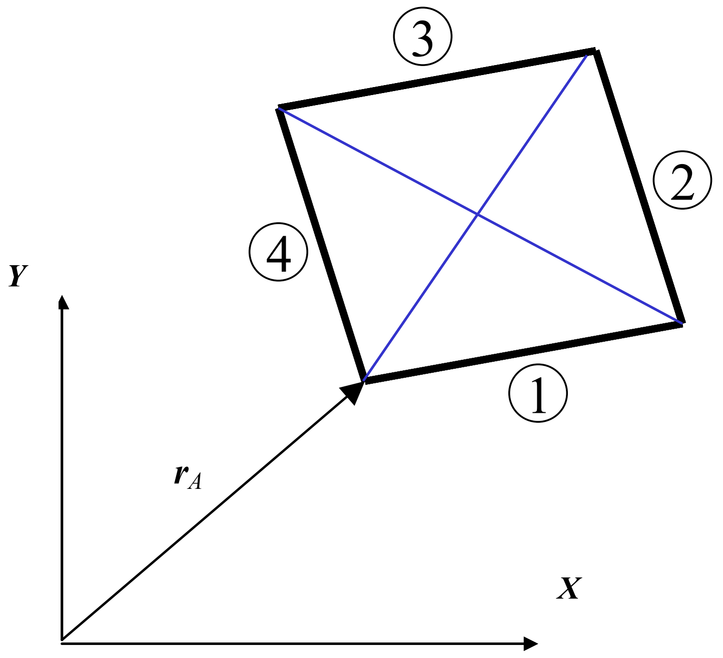

Figure 4.13. Planar truss geometry

We conveniently choose masses of the links to be 1 kg and lengths to be 1 m. The bending stiffness is 14e4 Nm for this simulation. The spring constant is 15 N/m for each spring. The constraints are given by

$$\boldsymbol{\phi} = \begin{pmatrix} \sum_{i=1}^{4} L_i \cos(\theta_i) \\ \sum_{i=1}^{4} L_i \sin(\theta_i) \end{pmatrix} = \boldsymbol{0} \tag{4.49}$$

Again, the Range Space method is used to solve for the multipliers. We assume that the links are initially undeformed with the following nonzero initial conditions:

$$\{x_A,\ y_A,\ \theta_1,\ \theta_2,\ \theta_3,\ \theta_4\} = \left\{1.0,\ 1.0,\ 0.1,\ \frac{\pi}{2}-0.1,\ \pi+0.1,\ \frac{3\pi}{2}-0.1\right\}$$

$$\{\dot{x}_A,\ \dot{y}_A,\ \dot{\theta}_1,\ \dot{\theta}_2,\ \dot{\theta}_3,\ \dot{\theta}_4\} = \{0.5,\ 0.5,\ 0.1,\ 0.1,\ 0.1,\ 0.1\}$$

The substructure rigid body and flexible components of energy are given in Figures 4.14 and 4.15, respectively.

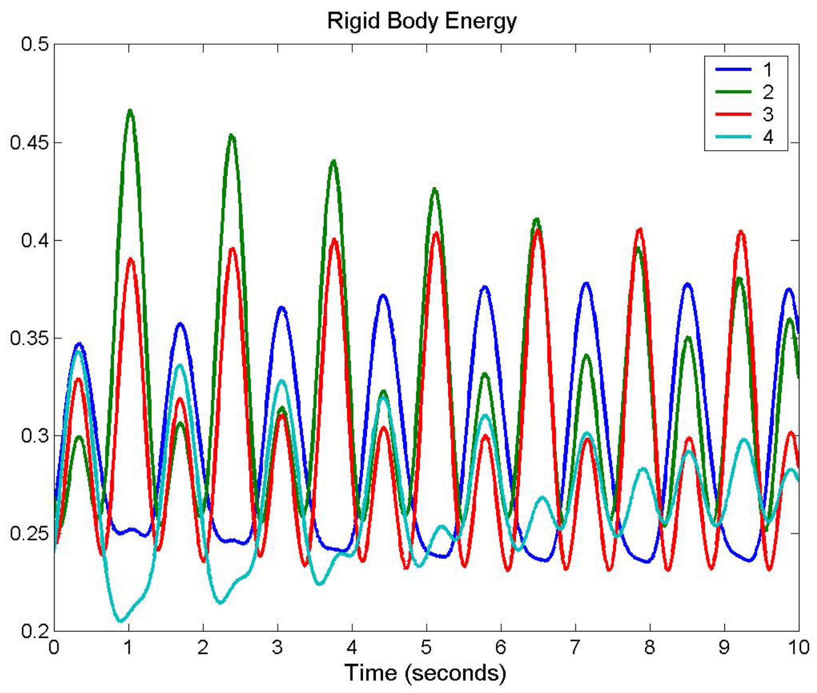

Figure 4.14. Planar truss rigid body energy

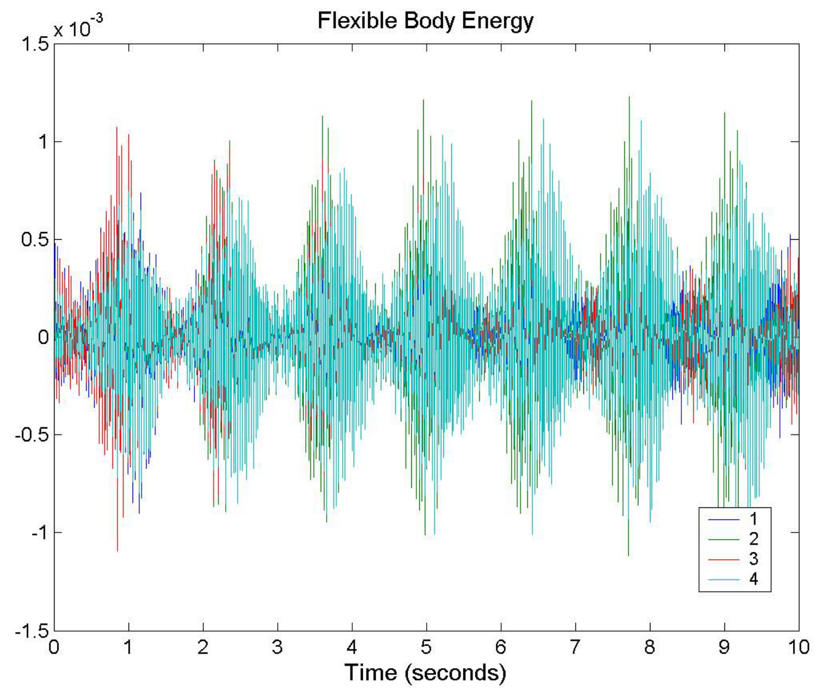

Figure 4.15. Planar truss flexible energy

It should be apparent from Figures 4.14 and 4.15 that from comparing the magnitudes of the rigid body and flexible components of energy that this system is appropriately modeled as a flexible system.

We now address the issue of the accuracy of the position and velocity level constraints for this system. In Figure 4.16 we show the error in the position level constraints. In Figure 4.17 we show the error in the velocity level constraint. As the figures show, these errors are quite small. Errors of size 10^{-5} and 10^{-2} were reported in Reference 31 for the position and velocity constraints. With the OCEA approach we find significant improvement: 9 orders of magnitude improvement in the position level constraint satisfaction and 12 orders of magnitude improvement in the satisfaction of the velocity level constraint. We do note that the system studied in Reference 31 is a three link closed chain system with only one link modeled with flexibility, and is not the same as that studied here. However, we draw comparison of the ability of each approach to produce the expected result of zero error in satisfying the position and velocity level constraints.

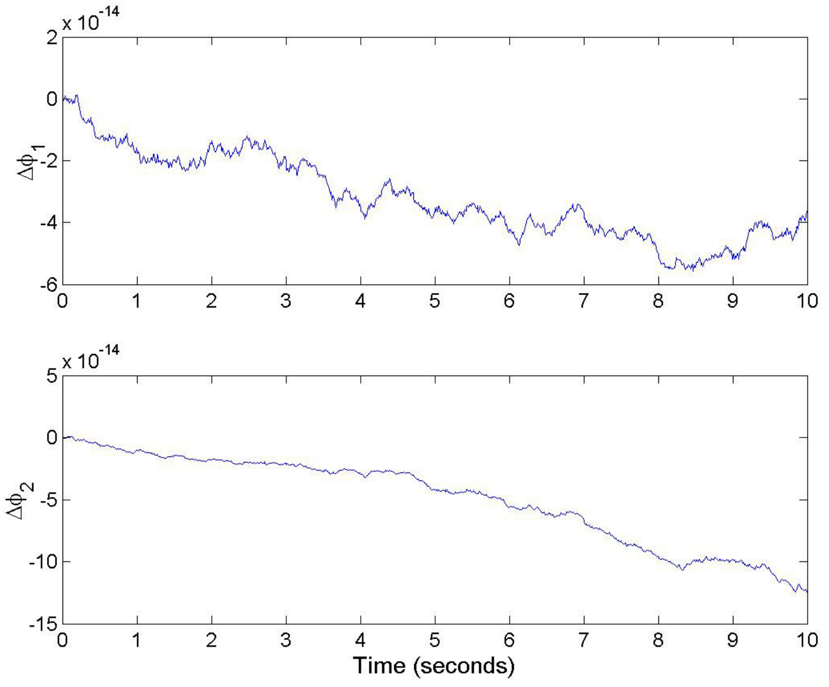

Figure 4.16. Errors in position level constraint for planar truss

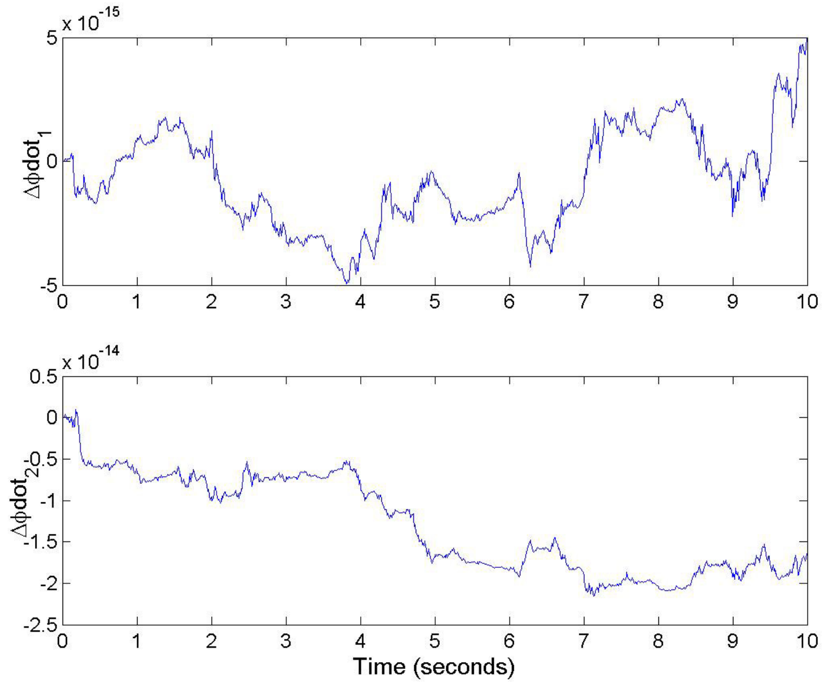

Figure 4.17. Errors in velocity level constraint for planar truss

4.6 COMPARISON WITH HARD-CODING EQUATIONS OF MOTION

We now focus on the accuracy of solutions for the system response from automatically generated and integrated equations of motion as compared to hand-derived, hard-coded equations of motion. Due to the complexity involved in hand-deriving equations of motion for many bodies, we consider a system comprised of only two flexible links. This system is commonly referred to as a double flexible pendulum. Additionally, we choose to model the system using five assumed modes per link. Starting at a point in the derivation with the kinetic and potential energy expressions, which were developed in this chapter, in hand, we find that the time required to develop the equations of motion by hand is about one hour. The process of coding and verifying the accuracy of the equations of motion adds an additional hour to the process. This exercise demonstrates the efficiency of the method developed in this chapter even for a system containing only two bodies since the number of bodies in the model is merely an input to the code.

In Figure 4.18 we show the errors in the angular and angular rate coordinates. Figures 4.19-4.20 show the errors in the flexible coordinates and their time derivatives for the first link. Figures 4.21-4.22 show the errors in the flexible coordinates and their time derivatives for the second link. Each of the figures show that the errors between the solution using automatic differentiation and that from the hard-coded solution agree to high accuracy on the order of the machine error. Errors of size 10^{-7} were reported in Reference 31 for the angular and flexible coordinates and their time derivatives. With the OCEA approach we see 10 orders of magnitude improvement over the results of Reference 31, as regards validating the two models were in fact numerically consistent.

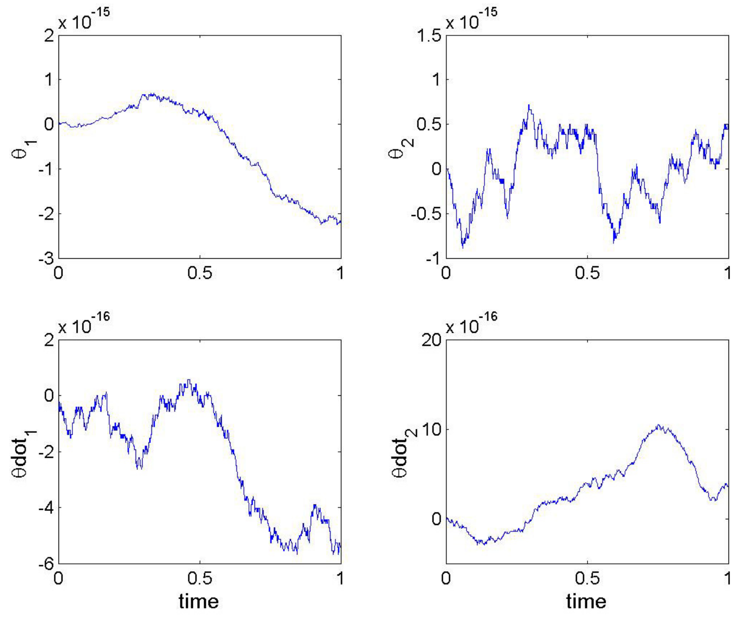

Figure 4.18. Errors in angular and angular rate coordinates

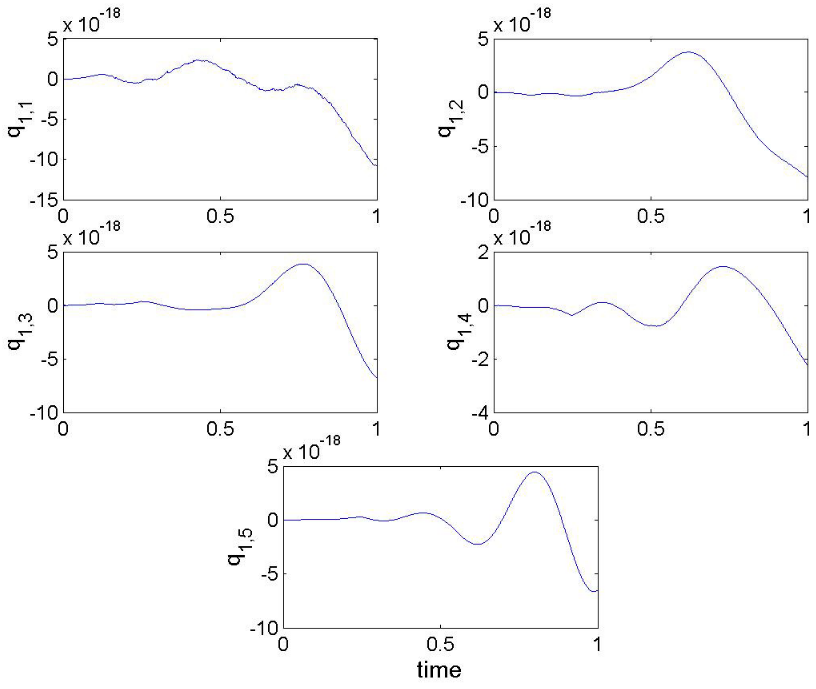

Figure 4.19. Errors in flexible coordinates for link one

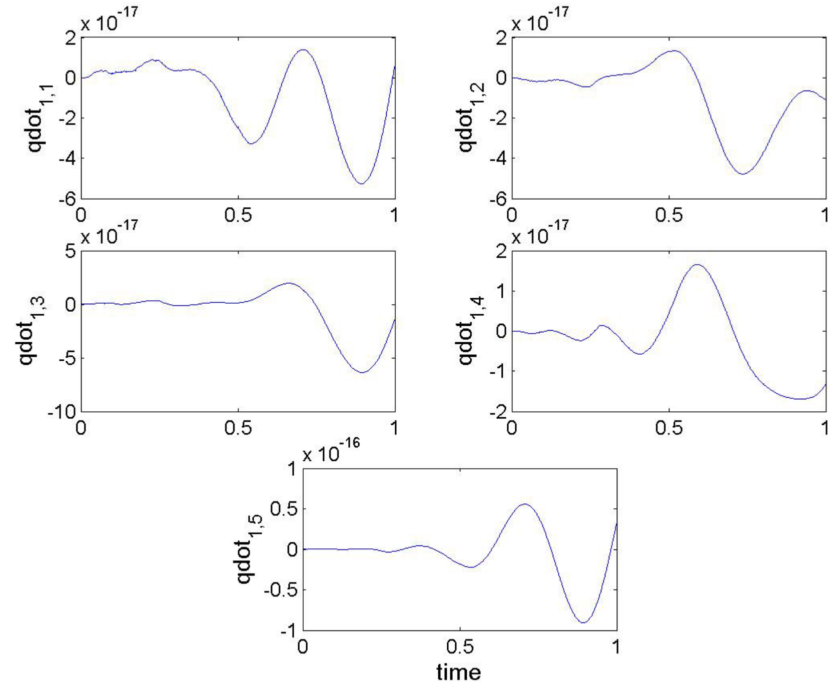

Figure 4.20. Errors in time derivatives of flexible coordinates for link one

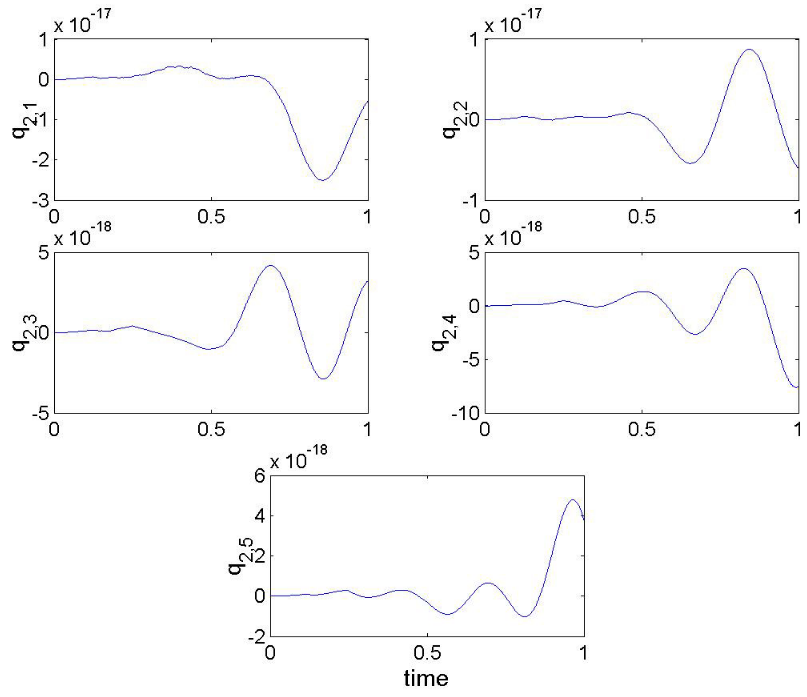

Figure 4.21. Errors in flexible coordinates for link two

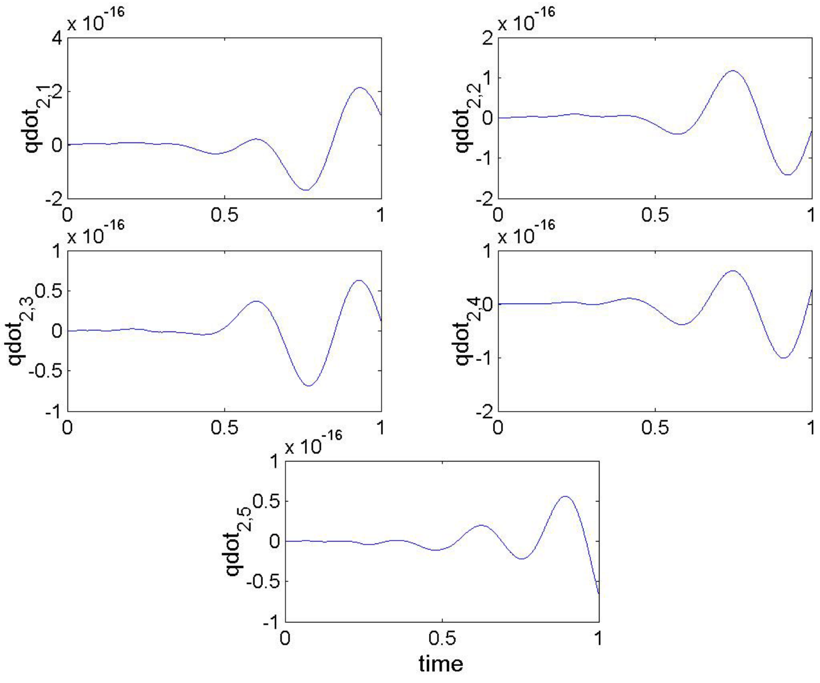

Figure 4.22. Errors in time derivatives of flexible coordinates for link two

4.7 SUMMARY

In this chapter, a new method for modeling and simulating dynamical systems was presented. The key development is a demonstration of the capability of automatic differentiation in automatically generating and integrating the equations of motion by simply specifying the system Lagrangian and the system constraints. No further work need be done by the analyst in order to produce the system response. Several examples were presented for rigid and flexible linked mechanical systems in open- and closed-chain topologies. It is remarkable that a single approach can produce such a variety of dynamical models, including simulations, with minimal coding.

The results presented in this chapter show that the OCEA approach for modeling dynamical systems is superior to other AD approaches for deriving equations of motion vis-à-vis user derivation and coding effort. With respect to coding effort, we find that with OCEA no code needs to be generated offline. This is not the case when using other programs such as ADIFOR or AUTODERIVE. All required first- and second-order partial derivatives are computed within the OCEA-FORTRAN environment. Furthermore, the accuracy of the OCEA solutions regarding the satisfaction of position and velocity level constraints, or when evaluating the errors between the OCEA solution and that from hand-derived equations of motion is profoundly better. While the results are indeed promising, it is not claimed that these initial developments are the final word on this subject. For example, only one dimensional elastic bodies (rods and beams) are considered. Therefore, further extensions of the well-established approach are required.

CHAPTER V

VALIDATION OF SOLUTION ACCURACY FOR DYNAMICAL SYSTEMS

5.1 INTRODUCTION

We now turn to a different subject – namely validation of model accuracy for a class of distributed parameter systems. What is often done to simplify the formulation of equations of motion for flexible dynamical systems is to introduce spatial approximations to represent the flexible body motion as an affine combination of shape functions. Examples of such approximation techniques include the well known Finite Element Method and Method of Assumed Modes (Ritz Method). However, the question that must be asked is: How accurately does a given approximate model describe the actual system?

The validation of approximate models for flexible body systems has typically been accomplished by comparing the approximate solution from a general code, specialized for a very simple system, with an exact analytical solution. This is known as the method of exact solutions. Since exact solutions typically do not exist for practical engineering level models, other methods must be employed to determine if the approximate model is valid or under what conditions it is valid. Two such methods are the method of manufactured solutions[41-44] and the method of nearby problems[45-48]. At first glance, these two methods appear to be quite similar; however, the "benchmark solution" is determined in a different manner for each method. With the method of manufactured solutions, we choose an analytical "benchmark solution" a priori. With the method of nearby problems, we first solve the problem for a fine discretization resolution and then proceed to determine a "benchmark solution" (in fact, a manufactured solution) by curve fitting the solution variable. For each method we desire to use this "benchmark solution" to compute analytical source terms by inverse dynamics. When the analytical source terms are added to the approximate equations of motion, a new set of equations of motion is formed which have as an exact solution the "benchmark solution". Thus, with the method of manufactured solutions, we produce the analytical source terms from a "benchmark solution" which may or may not have physical meaning, although it does satisfy the new set of equations of motion which include these analytical source terms. On the other hand, with the method of nearby problems, the "benchmark solution" does have physical meaning since it is a solution of the approximate dynamical equations of motion. An "exact" solution has been manufactured for the system of differential equations with no exact analytical solution. Therefore, we proceed as we do for the method of exact solutions. Solutions of the benchmark problem are compared to the manufactured solution in order to evaluate the solution accuracy. These methods are discussed in greater detail in the next section.

The analytical source terms must be computed using the exact dynamical equations of motion for both methods discussed above. The majority of the early literature on the method of manufactured solutions[41-44] and the method of nearby problems[48] deals with solving problems in computational fluid dynamics where the exact dynamical equations of motion (Navier-Stokes Equations) are well established for most problems. When dealing with the study of dynamical systems, we encounter a different situation since the equations of motion change significantly from problem to problem depending on; for example, the number of bodies in the system. Developing exact dynamical relationships for systems comprised of rigid bodies is rather straightforward; however, the problem is rather complex when the system is comprised of elastic elements. Thus, one necessary development which is presented in this chapter is the derivation of the exact governing equations of motion and boundary conditions for systems comprised of elastic elements. Given previous generalizations of Hamilton's Principle for discrete-parameter systems[20,47], these exact partial differential equations can be computed by differentiation as opposed to integration by parts, which was the only method until new developments were made in the past decade. In this chapter, it is demonstrated that equations of motion for discrete-parameter systems – systems containing discrete and flexible coordinates - can be generated by using automatic differentiation. The intention is to develop an advanced method for validating multibody dynamics codes. Previous work on validation has been done by using equations of motion which are developed by hand or using symbolic manipulation programs. The potential impact of this work is to have the capability to validate solutions from multibody dynamics codes for not only the most simple case example, but for more complicated systems by automating the process of generating exact dynamical representations.

OCEA is a promising tool for generation of equations of motion for discrete-parameter systems, as they are referred to in the literature. While such capability offers the possibility of new solution approaches, an alternative use of this capability, for flexible body systems, lies in validation of solutions for these systems. Typically, the solution process for hybrid discrete coordinate flexible body systems require both time and spatial discretization of the equations of motion. A choice for the integration time step and the number of elements are usually based on experience. One particular approach involves increasing the number of time steps and elements until the solution no longer changes by further discretizing the system. While this approach renders an adequate method for systems with a few flexible elements, it does not offer a practical means for properly choosing a discretization resolution for systems with many flexible elements. Furthermore, this type of approach does not guarantee the accuracy of the solution. It merely shows that no further changes are found in the solution. The methods for validating solution accuracy described in this work results in rigourous quantification of the error in the solution since the exact PDE/ODE relationships are used to evaluate the accuracy of the approximate solution. An approach is envisioned whereby the exact partial differential equation models are used to form an acceleration error at any point in space and time. These errors can be fed back to tune the discretization process and validate the solution.

It should be apparent that one method for validating the accuracy of solutions is to simply evaluate the exact dynamical representation using the approximate solution. The errors in satisfying the exact PDE/ODEs and boundary conditions, for various discretization resolution, can be considered as one simple method to quantify solution error. We address this issue later in the chapter. We discuss the method of nearby problems and we propose some alternatives to the traditional steps taken with this method.

Additionally, we investigate some of the more fundamental issues when approximating the dynamics of a flexible dynamical system by examining a simple system. Since the method of manufactured solutions and the method of nearby problems require computing higher-order space/time derivatives of the flexible coordinates (deflection variables), we study the accuracy of the solution for these higher-order space/time derivatives and the impact of their accuracy on computing analytical source terms for the purpose of validating solution accuracy.

5.2 METHODS FOR VALIDATING SOLUTION ACCURACY

There are several paths we can take to validate a solution derived from an assumed modes or FEM approach. We list some of these methods here:

1) The method of exact solutions
2) Evaluation of the error in satisfying the exact dynamical relationship
3) The method of manufactured solutions
4) The method of nearby problems

For the method of exact solutions, we compare the approximate solution from a general code, specialized for a very simple system, with the exact analytical solution. Of course this is preferred method; however, an exact solution is not available for practical engineering level models. We can however, with a significant amount of work by some methods, produce exact dynamical relationships including ordinary differential equations

(ODEs)/integro-ODEs, partial differential equations (PDEs), and boundary conditions. Although we typically do not have an exact analytical solution for these equations, we can consider method #2 as a means of evaluating the accuracy of a solution. Given a solution from an approximate model, we can then evaluate the system ODE/PDEs and boundary conditions to ascertain the accuracy of that solution with respect to satisfying the exact governing equations of motion and boundary conditions. We address this issue specifically for a clamped-free beam later in this chapter. We now focus on several methods (namely methods #3 and #4) that utilize the exact dynamical relationship to "manufacture" an "exact" solution.

As was previously mentioned, with the method of manufactured solutions we choose an analytical "benchmark solution" a priori and use the "benchmark solution" to compute analytical source terms. When added to the approximate equations of motion, a new set of equations of motion are formed by adding the analytical source terms which have as an exact solution the "benchmark solution". Thus, with the method of manufactured solutions, we produce the analytical source terms from a "benchmark solution" which may or may not have physical meaning, although it does satisfy the new set of equations of motion which include these analytical source terms. To illustrate the idea of the method of manufactured solutions, we consider the following example for validating the accuracy for an ODE.

Suppose we wish to evaluate the accuracy of solving the following differential equation for different time step sizes.

$$\dot{x} = x^2 \sin x \tag{5.1}$$

Now, without solving this equation, we choose for no particular reason the following benchmark solution, $x_b(t)$.

$$x_b(t) = t\cos t \tag{5.2}$$

where the time derivative of the benchmark solution is given by

$$\dot{x}_b(t) = \cos t - t\sin t \tag{5.3}$$

Now, we compute the analytical source term by inverse dynamics

$$\begin{aligned} e(t) &= \dot{x}_b - x_b^2 \sin x_b \\ &= \cos t - t\sin t - (t\cos t)^2 \sin(t\cos t) \end{aligned} \tag{5.4}$$

We add the analytical source term to Eq. (5.1) to produce the benchmark problem

$$\dot{x} = x^2 \sin x + e(t) \tag{5.5}$$

Equation (5.5) has as an exact solution the benchmark solution given by Eq. (5.2). Thus, the benchmark problem can be solved for various time step sizes in producing a solution for the motion. The accuracy of the solution is obtained by comparison with the benchmark solution. Of course, this is only a simple illustration. The more complicated problem when dealing with flexible dynamical systems is presented later in the chapter.

On the other hand, with the method of nearby problems, the "benchmark solution" does have physical meaning since it is a solution of the approximate dynamical equations of motion. We first solve the problem for some choice of resolution for the time step size (and spatial discretization for a flexible dynamical system). Subsequently, we begin the process of determining an exact benchmark solution. Firstly, we approximate this solution by fitting a curve through the solution. For a distributed-parameter, multibody system we fit curves through the discrete coordinates $\boldsymbol{q}(t)$ and the flexible coordinates $\boldsymbol{v}(x,t)$ to produce the nearby solution given by $\tilde{\boldsymbol{q}}(t)$ and $\tilde{\boldsymbol{v}}(x,t)$. Thus we have an analytical (symbolic) relationship for the solution variables as a function of space and time in which we can readily compute space/time derivatives since we know the chosen mathematical model for these coordinates. Now, we can evaluate the exact dynamical relationships at this nearby solution and its space/time derivatives to produce the analytical source terms. These analytical source terms are then used to compute fictitious forcing terms, which when included in the approximate dynamical equations, represent the forces needed in order for the nearby solution to be an exact solution for the new set of equations (the benchmark problem). Subsequently, we can discretize the system as we choose for various numbers of assumed modes or number of finite elements, and solve the benchmark problem. Then, we can evaluate the accuracy of solutions by comparing the solution from the benchmark problem with the nearby solution.

The entire procedure for the method of nearby solutions is quite tedious. Careful attention must be given to handling the creation of an exact benchmark solution. This process firstly requires choosing a set of basis functions which are capable of accurately approximating the solution variables. Typically, this is done using high-order polynomials[46,47]. Secondly, we require an exact dynamical relationship for the system. Typically, this is accomplished by symbolic manipulation [46,47]. Thirdly, careful attention must be given to computing the analytical source terms. Symbolic manipulation must be utilized once again in order to create a mapping for the errors in the dynamical relationships (analytical source terms). These analytical source terms are then used to compute the fictitious forcing functions. For example, the error in a PDE governing the flexible dynamics, given by $e(x,t)$, is required to compute the fictitious forcing functions (generalized forces) as follows

$$\hat{f}_i(t) = \int_0^L e(x,t)\phi_i(x)dx \tag{5.6}$$

The computation of these generalized forces, and any other generalized forces associated with the errors in the ODEs or the boundary conditions is thus the key result of the method of nearby problems, as demonstrated in parallel with Eq. (5.5), since these generalized forces produce the benchmark problem with an "exact" solution. These are the only terms we directly require in evaluating the solution accuracy of our approximate dynamical model.

We now consider alternatives for executing this process known as the method of nearby problems. One alternative is the method by which the exact dynamical equations are developed. We propose here that we develop these equations using automatic differentiation as opposed to symbolic differentiation. The method by which this can be accomplished is given in the next section. We now focus on the qualitative ideas associated with this choice. Firstly, we note that by using automatic differentiation, we are limited to numerically evaluating the exact dynamical equations. Thus, we cannot produce analytical source terms (or ultimately, generalized forces) in an analytical/symbolic fashion. For example, when evaluating the error in the exact PDE we process a numerical map of the error over space and time, which results from evaluating the solution variables (deflection, and its time and spatial derivatives) *numerically* at the approximate solution for each time and spatial value. Considering this issue, we consider as one option approximating the error in the exact equations $e^*(x,t)$ as $\tilde{e}^*(x,t)$ (curve fit in space and time), then proceed to compute the fictitious forcing functions *analytically* as follows

$$\hat{f}_i(t) = \int_0^L \tilde{e}^*(x,t)\phi_i(x)dx \tag{5.7}$$

We now consider an additional option for computing the fictitious forcing functions using OCEA. As opposed to approximating the error functions, we can simply proceed directly to computing the forcing functions by numerically integrating the error functions and their product with the appropriate basis functions over space. Thus, we are forming a fictitious forcing function value for each point in time. For, example

$$\hat{f}_i(t) = \int_0^L e^*(x,t)\phi_i(x)dx \tag{5.8}$$

where $e^*(x,t)$ is computed numerically at each point in space and time as

$$e^*(x,t) = \frac{d}{dt}\left(\frac{\partial \hat{L}}{\partial \dot{v}}\right) - \frac{\partial \hat{L}}{\partial v} + \frac{\partial}{\partial x}\left(\frac{\partial \hat{L}}{\partial v'}\right) - \frac{\partial^2}{\partial x^2}\left(\frac{\partial \hat{L}}{\partial v''}\right) - \hat{\boldsymbol{f}}^T \tag{5.9}$$

The numerical values for the fictitious forcing functions given in Eq. (5.9) can be approximated by an analytical function, if desired. Otherwise, a simple lookup table could be formed in order to implement these fictitious forcing functions in the benchmark problem.

Thus far, we have considered global approximation methods. On the other hand, we can consider a different manner in which we create the nearby solution at the deflection level. Instead of using high-order polynomials we can utilize the basis functions which we have chosen to characterize the solution. These basis functions are the assumed mode shape functions and the FEM shape functions. Thus, the approximation process requires only approximating a function in time which modulates the assumed mode shape functions. This would be the most straightforward method by which we tend in the direction of a local fitting approach. Otherwise, we can chose to compute the nearby solution by using locally defined low-order functions. However, here careful attention must be given to the issue of continuity of the locally defined functions.

5.3 AUTOMATIC GENERATION OF EXACT DYNAMICAL MODELS

In this section, we discuss generating exact PDE/ODE dynamical models and the corresponding boundary conditions using OCEA. As was mentioned earlier, the primary challenge involved in the validation effort for flexible dynamical systems is the generation of the exact dynamical representation.

Here we consider as the path the hybrid coordinate generalizations of Hamilton's Principle for generating these dynamical relationships[20,47]. In general, we can write the system Lagrangian to be composed of discrete terms (L_D), spatial terms ($\hat{L}_i$), and boundary terms (L_B). Considering a general system composed of multiple elastic domains, we write the Lagrangian as follows

$$L = L_D + \sum_{i=1}^{n}\left[\int_0^{l_i} \hat{L}_i dx_i\right] + L_B \tag{5.10}$$

The ordinary differential equations of motion are derived from

$$\frac{d}{dt}\left(\frac{\partial L}{\partial \dot{\boldsymbol{q}}}\right) - \frac{\partial L}{\partial \boldsymbol{q}} = \boldsymbol{Q}^T \tag{5.11}$$

whereas the partial differential equations of motion are derived from

$$\frac{d}{dt}\left(\frac{\partial \hat{L}_i}{\partial \dot{v}_i}\right)-\frac{\partial \hat{L}_i}{\partial v_i}+\frac{\partial}{\partial x_i}\left(\frac{\partial \hat{L}_i}{\partial v_i^{'}}\right)-\frac{\partial^2}{\partial x_i^2}\left(\frac{\partial \hat{L}_i}{\partial v_i^{''}}\right)=\hat{f}_i^T \tag{5.12}$$

and the boundary conditions are generated from

$$\begin{aligned}&\left\{\frac{\partial \hat{L}_i}{\partial v_i^{'}}-\frac{\partial}{\partial x_i}\left(\frac{\partial \hat{L}_i}{\partial v_i^{''}}\right)\right\}\delta v_i\Bigg|_0^{l_i}+\left\{\frac{\partial \mathcal{L}_B}{\partial v_i(l_i)}-\frac{d}{dt}\left(\frac{\partial \mathcal{L}_B}{\partial \dot{v}_i(l_i)}\right)\right\}\delta v_i(l_i)+\hat{\boldsymbol{f}}_i^{1T}\delta v_i(l_i)=0\\&\frac{\partial \hat{L}_i}{\partial v_i^{''}}\delta v_i^{'}\Bigg|_0^{l_i}+\left\{\frac{\partial \mathcal{L}_B}{\partial v_i^{'}(l_i)}-\frac{d}{dt}\left(\frac{\partial \mathcal{L}_B}{\partial \dot{v}_i^{'}(l_i)}\right)\right\}\delta v_i^{'}(l_i)+\hat{\boldsymbol{f}}_i^{2T}\delta v_i^{'}(l_i)=0\end{aligned} \tag{5.13}$$

where $\mathcal{L}_B$ is defined as

$$\mathcal{L}_B=\sum_{i=1}^{n}\left[\int_0^{l_i}\hat{L}_i dx_i\right]+L_B \tag{5.14}$$

Whereas we would derive the governing equations of motion and boundary conditions starting with Hamilton's Principle via integration by parts; here we see that these equations can be derived by differentiation.

We now discuss in detail how to automatically generate these equations by using OCEA. First we consider automatically generating the governing ODE. Considering the first term in Eq. (5.11) we write

$$\frac{d}{dt}\left(\frac{\partial L}{\partial \dot{\boldsymbol{q}}}\right)=\frac{d}{dt}\left(\frac{\partial L_D}{\partial \dot{\boldsymbol{q}}}+\sum_{i=1}^{n}\left[\int_0^{l_i}\frac{\partial \hat{L}_i}{\partial \dot{\boldsymbol{q}}}dx_i\right]+\frac{\partial L_B}{\partial \dot{\boldsymbol{q}}}\right) \tag{5.15}$$

We can rewrite the first term on the RHS of Eq. (5.15) as

$$\frac{d}{dt}\left(\frac{\partial L_D}{\partial \dot{\boldsymbol{q}}}\right)=\frac{\partial^2 L_D}{\partial \dot{\boldsymbol{q}}\partial \boldsymbol{X}}\frac{d\boldsymbol{X}}{dt} \tag{5.16}$$

where $\boldsymbol{X}$ is a vector containing all of the coordinates in which the system Lagrangian consists

$$\boldsymbol{X} = \left[\boldsymbol{x}, \boldsymbol{q}(t), \dot{\boldsymbol{q}}(t), \boldsymbol{w}(x,t), \dot{\boldsymbol{w}}(x,t), \boldsymbol{w}'(x,t), \boldsymbol{w}''(x,t), \boldsymbol{w}(l), \dot{\boldsymbol{w}}(l), \boldsymbol{w}'(l), \dot{\boldsymbol{w}}'(l) \right]$$

We can rewrite the second term on the RHS of Eq. (5.15) as

$$\frac{d}{dt}\left(\sum_{i=1}^{n} \left[\int_0^{l_i} \frac{\partial \hat{L}_i}{\partial \dot{\boldsymbol{q}}} dx_i \right] \right) = \sum_{i=1}^{n} \left[\int_0^{l_i} \frac{\partial^2 \hat{L}_i}{\partial \dot{\boldsymbol{q}} \partial \boldsymbol{X}} \frac{d\boldsymbol{X}}{dt} dx_i \right] \tag{5.17}$$

We can rewrite the third term on the RHS of Eq. (5.15) as

$$\frac{d}{dt}\left(\frac{\partial L_B}{\partial \dot{\boldsymbol{q}}} \right) = \frac{\partial^2 L_B}{\partial \dot{\boldsymbol{q}} \partial \boldsymbol{X}} \frac{d\boldsymbol{X}}{dt} \tag{5.18}$$

We write the second term in Eq. (5.11) as

$$\frac{\partial L}{\partial \boldsymbol{q}} = \frac{\partial L_D}{\partial \boldsymbol{q}} + \sum_{i=1}^{n} \left[\int_0^{l_i} \frac{\partial \hat{L}_i}{\partial \dot{\boldsymbol{q}}} dx_i \right] + \frac{\partial L_B}{\partial \boldsymbol{q}} \tag{5.19}$$

In summary, we can compute the ODEs based on Eq. (5.11) by

$$\frac{\partial^2 L_D}{\partial \dot{\boldsymbol{q}} \partial \boldsymbol{X}} \frac{d\boldsymbol{X}}{dt} + \sum_{i=1}^{n} \left[\int_0^{l_i} \left(\frac{\partial^2 \hat{L}_i}{\partial \dot{\boldsymbol{q}} \partial \boldsymbol{X}} \frac{d\boldsymbol{X}}{dt} - \frac{\partial \hat{L}_i}{\partial \boldsymbol{q}} \right) dx_i \right] + \frac{\partial^2 L_B}{\partial \dot{\boldsymbol{q}} \partial \boldsymbol{X}} \frac{d\boldsymbol{X}}{dt} - \frac{\partial L_D}{\partial \boldsymbol{q}} - \frac{\partial L_B}{\partial \boldsymbol{q}} = \boldsymbol{Q}^T \tag{5.20}$$

For obvious reasons, we must specify the individual terms that form the system Lagrangian separately in order to accomplish the spatial integrals.

We now turn our attention to generating the PDEs. Considering the first term in Eq. (5.12) we write

$$\frac{d}{dt}\left(\frac{\partial \hat{L}_i}{\partial \dot{w}_i} \right) = \frac{\partial^2 \hat{L}_i}{\partial \dot{w}_i \partial \boldsymbol{X}} \frac{d\boldsymbol{X}}{dt} \tag{5.21}$$

The second term in Eq. (5.12) can be computed straightforwardly. The third and fourth terms must be computed offline since the partials of $\hat{L}$ with respect to w' and w'' are implicit functions of the spatial variable x and cannot be accomplished by explicit differentiation means such as automatic differentiation. On the other hand, we can define $\hat{L}$ in such a way that it is an explicit function of x with no explicit dependence on to w' and w'' if we utilize the assumed form for these expressions based on the chosen modeling approach (FEM or Assumed Modes). In this way, the third and fourth terms in Eq. (5.12) can be computed using OCEA.

Regarding the boundary conditions in Eq. (5.13), we see that we need to compute partials with respect to the Lagrangian density functions ($\hat{L}_i$) and $\mathcal{L}_B$. We've discussed computing partials of the Lagrangian density functions in the previous paragraph. We focus on four terms in the two boundary conditions given in Eq. (5.13), namely

$$\frac{\partial \mathcal{L}_B}{\partial v_i(l_i)},\ \frac{\partial \mathcal{L}_B}{\partial v_i'(l_i)},\ \frac{d}{dt}\left(\frac{\partial \mathcal{L}_B}{\partial \dot{v}_i(l_i)}\right),\ \text{and}\ \frac{d}{dt}\left(\frac{\partial \mathcal{L}_B}{\partial \dot{v}_i'(l_i)}\right)$$

The first two terms are straightforwardly computed as

$$\frac{\partial \mathcal{L}_B}{\partial v_i(l_i)} = \sum_{i=1}^{n}\left[\int_0^{l_i} \frac{\partial \hat{L}_i}{\partial v_i(l_i)} dx_i\right] + \frac{\partial L_B}{\partial v_i(l_i)} \tag{5.22}$$

$$\frac{\partial \mathcal{L}_B}{\partial v_i'(l_i)} = \sum_{i=1}^{n}\left[\int_0^{l_i} \frac{\partial \hat{L}_i}{\partial v_i'(l_i)} dx_i\right] + \frac{\partial L_B}{\partial v_i'(l_i)} \tag{5.23}$$

The second and third terms are computed as

$$\frac{d}{dt}\left(\frac{\partial \mathcal{L}_B}{\partial \dot{v}_i(l_i)}\right) = \sum_{i=1}^{n}\left[\int_0^{l_i} \frac{\partial^2 \hat{L}_i}{\partial \dot{v}_i(l_i)\partial \mathbf{X}} \frac{d\mathbf{X}}{dt} dx_i\right] + \frac{\partial^2 L_B}{\partial \dot{v}_i(l_i)\partial \mathbf{X}} \frac{d\mathbf{X}}{dt} \tag{5.24}$$

$$\frac{d}{dt}\left(\frac{\partial \mathcal{L}_B}{\partial \dot{v}_i'(l_i)}\right)=\sum_{i=1}^{n}\left[\int_0^{l_i}\frac{\partial^2 \hat{L}_i}{\partial \dot{v}_i'(l_i)\partial \boldsymbol{X}}\frac{d\boldsymbol{X}}{dt}dx_i\right]+\frac{\partial^2 L_B}{\partial \dot{v}_i'(l_i)\partial \boldsymbol{X}}\frac{d\boldsymbol{X}}{dt} \tag{5.25}$$

Thus the complete dynamical representation including ODEs, PDEs, and boundary conditions can be generated, for a class of dynamical systems, in the fashion described in this section using OCEA. The errors in satisfying these exact governing equations of motion and boundary conditions can be evaluated numerically to produce the analytical source terms as described in the previous section. We now present some numerical examples to demonstrate the accuracy of this method in generating the exact dynamical representation.

5.4 MULTIBODY SYSTEM EXAMPLES

In this section we investigate the accuracy of the method presented in the previous section for automatically deriving the exact dynamical relationships for a distributed-parameter system. We present one numerical example, and generalize the formulation to the multibody case

5.4.1 Double Pendulum

We consider as an example a two link pendulum comprised of flexible elements. The discrete part of the Lagrangian is given by

$$L_D=\tfrac{1}{6}m_1L_1^2\dot{\theta}_1^2+\tfrac{1}{6}m_2L_2^2\dot{\theta}_2^2 \tag{5.26}$$

The boundary part of the Lagrangian is given by

$$L_B=\tfrac{1}{2}m_2L_1^2\dot{\theta}_1^2+\tfrac{1}{2}m_2L_1L_2\dot{\theta}_1\dot{\theta}_2\cos(\theta_2-\theta_1) \tag{5.27}$$

and the Lagrangian density functions for the first and second links are given by

$$\hat{L}_1 = \tfrac{1}{2}\rho_1\left\{\dot{\theta}_1^2 v_1^2 + \dot{v}_1^2 + 2x_1\dot{v}_1\dot{\theta}_1\right\} - \tfrac{1}{2}EI_1\left(v_1''\right)^2 \tag{5.28}$$

$$\hat{L}_2 = \tfrac{1}{2}\rho_2\left\{\begin{matrix}\dot{\theta}_2^2 v_2^2 + \dot{v}_2^2 + 2x_2\dot{v}_2\dot{\theta}_2 - 2L_1\dot{\theta}_1\dot{\theta}_2 v_2\sin(\theta_2-\theta_1) \\ +2L_1\dot{\theta}_1\dot{v}_2\cos(\theta_2-\theta_1)\end{matrix}\right\} - \tfrac{1}{2}EI_2\left(v_2''\right)^2 \tag{5.29}$$

Thus, we can form the system Lagrangian of Eq. (5.10) by Eqs. (5.26-5.29). The ODEs and PDEs are derived by hand using Eq. (5.11) and (5.12), respectively. The resulting exact ODEs derived by hand are given in Eqs. (5.30-5.31).

$$\begin{gathered}\left(\tfrac{1}{3}m_1L_1^2 + m_2L_1^2\right)\ddot{\theta}_1 + \tfrac{1}{2}m_2L_1L_2\ddot{\theta}_2\cos(\theta_2-\theta_1) - \tfrac{1}{2}m_2L_1L_2\dot{\theta}_2^2\sin(\theta_2-\theta_1) \\ +\int_0^{L_1}\rho_1\left\{v_1^2\ddot{\theta}_1 + 2v_1\dot{v}_1\dot{\theta}_1 + x_1\ddot{v}_1\right\}dx_1 \\ +\int_0^{L_2}\rho_2\left\{\left(-L_1\ddot{\theta}_2 v_2 - 2L_1\dot{\theta}_2\dot{v}_2\right)\sin(\theta_2-\theta_1) + \left(-L_1v_2\dot{\theta}_2^2 + L_1\ddot{v}_2\right)\cos(\theta_2-\theta_1)\right\}dx_2 \\ = 0\end{gathered} \tag{5.30}$$

$$\begin{gathered}\tfrac{1}{3}m_2L_2^2\ddot{\theta}_2 + \tfrac{1}{2}m_2L_1L_2\ddot{\theta}_1\cos(\theta_2-\theta_1) + \tfrac{1}{2}m_2L_1L_2\dot{\theta}_1^2\sin(\theta_2-\theta_1) \\ +\int_0^{L_2}\rho_2\left\{v_2^2\ddot{\theta}_2 + 2v_2\dot{v}_2\dot{\theta}_2 + x_2\ddot{v}_2 - L_1\ddot{\theta}_1v_2\sin(\theta_2-\theta_1) + L_1v_2\dot{\theta}_1^2\cos(\theta_2-\theta_1)\right\}dx_2 \\ = 0\end{gathered} \tag{5.31}$$

The resulting exact PDEs derived by hand are given in Eqs. (5.32-5.33).

$$\rho_1\left(\ddot{v}_1 + x_1\ddot{\theta}_1\right) - \rho_1v_1\dot{\theta}_1^2 + EI_1v_1'''' = 0 \tag{5.32}$$

$$\rho_2\left(\ddot{v}_2 + x_2\ddot{\theta}_2 + L_1\ddot{\theta}_1\cos(\theta_2-\theta_1)\right) - \rho_2v_2\dot{\theta}_2^2 + \rho_2L_1\dot{\theta}_1^2\sin(\theta_2-\theta_1) + EI_2v_2'''' = 0 \tag{5.33}$$

We code the functions given in Eqs. (5.26-5.29) in an OCEA-FORTRAN subroutine in order to compute the exact ODE/PDEs for this system. We evaluate these OCEA-generated equations using a solution for the motion of the double pendulum by methods described in Chapter IV. We also hand code and subsequently evaluate the ODE/PDEs given by Eqs. (5.30-5.33). Figure 5.1 shows the numerical difference between the hand-derived ODEs and the OCEA-derived ODEs for the double flexible pendulum. We note that a Gaussian Quadrature formula was used to compute the spatial integrals in Eqs. (5.30) and (5.31). Figure 5.2 shows the numerical difference between the hand-derived PDEs and the OCEA-derived PDEs. In each plot, the upper portion shows the discrepancy for link one and the lower plot shows the discrepancy for link two.

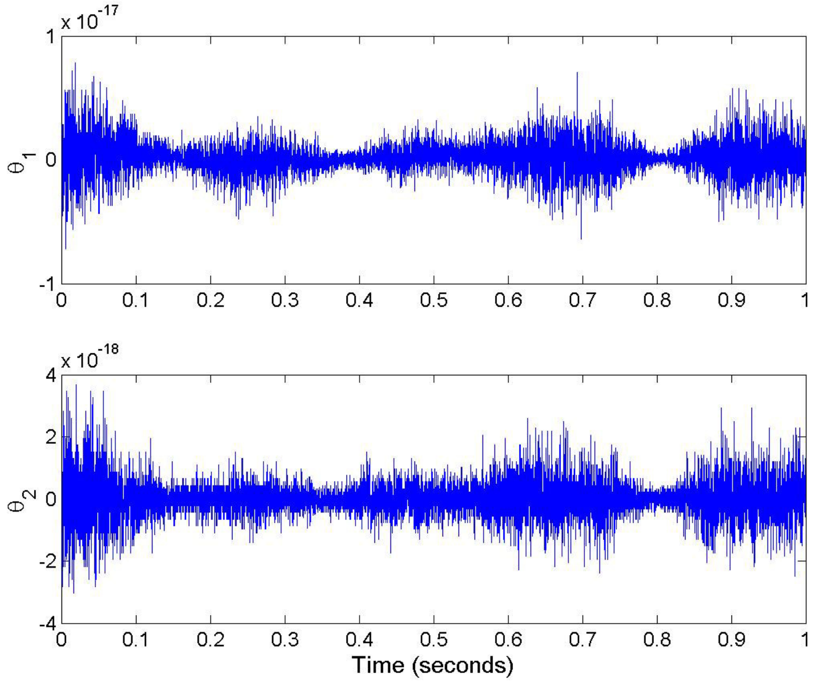

Figure 5.1. Comparison of hard-coded and OCEA-derived ODEs

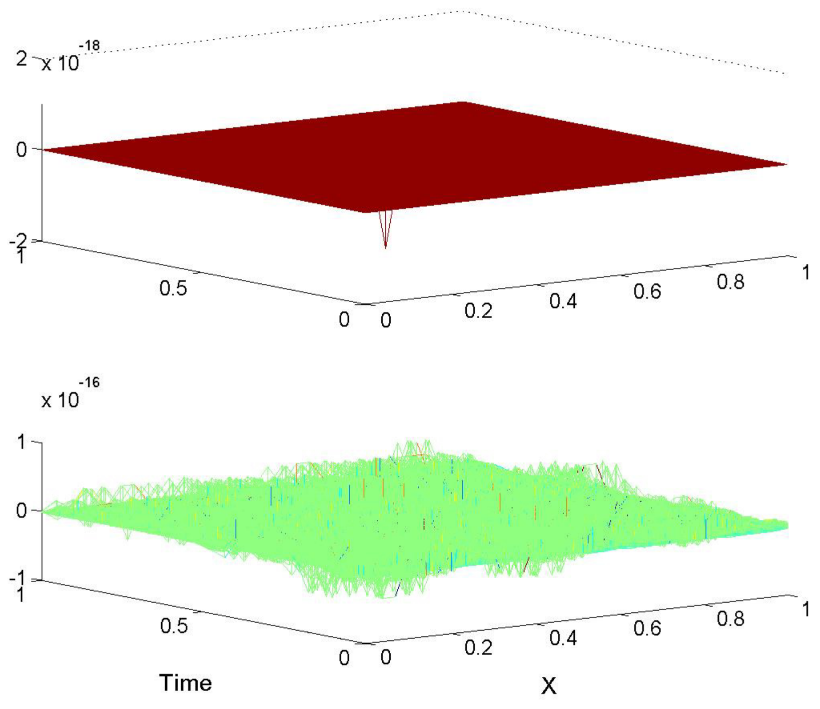

Figure 5.2. Comparison of hard-coded and OCEA-derived PDEs

As can be seen in Figures 5.1 and 5.2, the numerical difference between the hand-derived ODE/PDEs and the OCEA-derived ODE/PDEs is on the order of machine error. This demonstrates that the method introduced in the previous section to automatically generate the exact dynamical representation is very accurate and is suitable for use in computing analytical source terms for the purpose of validating solution accuracy. In the next section, we generalize the approach in order to generate exact dynamical representations for multibody systems.

5.4.2 General Multibody System

We now consider generalizing the formulation of the system Lagrangian for a general multibody system containing an arbitrary number of bodies, in an open- or closed-chain configuration, including general planar translational motion. We provide this generalization to fully demonstrate the capability of the method described in the previous section for generating the exact dynamical representation for complicated dynamical systems. The planar truss, as described in Chapter IV, falls into this category.

We now present the exact energy expressions for a general multibody system. Here we show results which follow the general formulation for flexible multibody systems described in Chapter IV. Here, though, we do not introduce approximation for the flexible coordinates in presenting the Lagrangian expression.

The kinetic energy of the first link is given by

$$\begin{aligned} T_1 &= \tfrac{1}{2} m_1 \left(\dot{x}_A^2 + \dot{y}_A^2 \right) + \tfrac{1}{6} m_1 L_1^2 \dot{\theta}_1^2 - \tfrac{1}{2} m_1 L_1 \dot{x}_A \dot{\theta}_1 \sin\theta_1 + \tfrac{1}{2} m_1 L_1 \dot{y}_A \dot{\theta}_1 \cos\theta_1 \\ &\quad + \int_0^{L_1} \tfrac{1}{2} \rho_1 \left\{ \begin{matrix} \dot{\theta}_1^2 v_1^2 + \dot{v}_1^2 + 2 x_1 \dot{v}_1 \dot{\theta}_1 \\ -2\dot{x}_A \dot{\theta}_1 v_1 \cos\theta_1 - 2\dot{x}_A \dot{v}_1 \sin\theta_1 \\ -2\dot{y}_A \dot{\theta}_1 v_1 \sin\theta_1 + 2\dot{y}_A \dot{v}_1 \cos\theta_1 \end{matrix} \right\} dx_1 \end{aligned} \tag{5.34}$$

and the potential energy for the first link is given by

$$V_1 = \int_0^{L_1} \tfrac{1}{2} EI_1 \left(v_1'' \right)^2 dx_1 \tag{5.35}$$

Thus, the discrete part of the Lagrangian for link one is given by

$$L_D^{(1)} = \tfrac{1}{2} m_1 \left(\dot{x}_A^2 + \dot{y}_A^2 \right) + \tfrac{1}{6} m_1 L_1^2 \dot{\theta}_1^2 \tag{5.36}$$

and the boundary part of the Lagrangian for link one is given by

$$L_B^{(1)} = -\tfrac{1}{2} m_1 L_1 \dot{x}_A \dot{\theta}_1 \sin\theta_1 + \tfrac{1}{2} m_1 L_1 \dot{y}_A \dot{\theta}_1 \cos\theta_1 \tag{5.37}$$

and, of course, the Lagrangian density function for the first link is given by

$$\hat{L}_1 = \tfrac{1}{2}\rho_1 \begin{Bmatrix} \dot{\theta}_1^2 v_1^2 + \dot{v}_1^2 + 2x_1\dot{v}_1\dot{\theta}_1 \\ -2\dot{x}_A\dot{\theta}_1 v_1 \cos\theta_1 - 2\dot{x}_A\dot{v}_1 \sin\theta_1 \\ -2\dot{y}_A\dot{\theta}_1 v_1 \sin\theta_1 + 2\dot{y}_A\dot{v}_1 \cos\theta_1 \end{Bmatrix} \tag{5.38}$$

We now look at a generalization for link (p+1), where p>1. The kinetic energy for link (p+1) is given by

$$\begin{aligned} T_{p+1} &= \tfrac{1}{2}m_{p+1}\left(\dot{x}_A^2 + \dot{y}_A^2\right) + \tfrac{1}{6}m_{p+1}L_{p+1}^2\dot{\theta}_{p+1}^2 + \tfrac{1}{2}m_{p+1}\sum_{i=1}^{p}\sum_{j=1}^{p} L_i\dot{\theta}_i L_j\dot{\theta}_j\cos(\theta_j - \theta_i) \\ &-m_{p+1}\dot{x}_A\sum_{i=1}^{p} L_i\dot{\theta}_i \sin\theta_i - \tfrac{1}{2}m_{p+1}L_{p+1}\dot{x}_A\dot{\theta}_{p+1}\sin\theta_{p+1} \\ &+m_{p+1}\dot{y}_A\sum_{i=1}^{p} L_i\dot{\theta}_i \cos\theta_i + \tfrac{1}{2}m_{p+1}L_{p+1}\dot{y}_A\dot{\theta}_{p+1}\cos\theta_{p+1} \\ &+\tfrac{1}{2}m_{p+1}L_{p+1}\dot{\theta}_{p+1}\sum_{i=1}^{p} L_i\dot{\theta}_i\cos(\theta_{p+1} - \theta_i) \\ &+\int_0^{L_{p+1}} \tfrac{1}{2}\rho_{p+1} \begin{Bmatrix} \dot{\theta}_{p+1}^2 v_{p+1}^2 + \dot{v}_{p+1}^2 + 2x_{p+1}\dot{v}_{p+1}\dot{\theta}_{p+1} \\ -2\dot{x}_A\dot{\theta}_{p+1}v_{p+1}\cos\theta_{p+1} - 2\dot{x}_A\dot{v}_{p+1}\sin\theta_{p+1} \\ -2\dot{y}_A\dot{\theta}_{p+1}v_{p+1}\sin\theta_{p+1} + 2\dot{y}_A\dot{v}_{p+1}\cos\theta_{p+1} \\ -2\dot{\theta}_{p+1}v_{p+1}\sum_{i=1}^{p} L_i\dot{\theta}_i\sin(\theta_{p+1} - \theta_i) \\ +2\dot{v}_{p+1}\sum_{i=1}^{p} L_i\dot{\theta}_i\cos(\theta_{p+1} - \theta_i) \end{Bmatrix} dx_{p+1} \end{aligned} \tag{5.39}$$

and the potential energy for link (p+1) is given by

$$V_{p+1} = \int_0^{L_{p+1}} \tfrac{1}{2}EI_{p+1}\left(v_{p+1}''\right)^2 dx_{p+1} \tag{5.40}$$

Thus we can write the discrete, boundary and Lagrangian density portions of the Lagrangian for the second link and so on as

$$L_D^{(p+1)} = \tfrac{1}{2} m_{p+1} \left(\dot{x}_A^2 + \dot{y}_A^2 \right) + \tfrac{1}{6} m_{p+1} L_{p+1}^2 \dot{\theta}_{p+1}^2 \tag{5.41}$$

$$\begin{aligned} L_B^{(p+1)} = {} & \tfrac{1}{2} m_{p+1} \sum_{i=1}^{p} \sum_{j=1}^{p} L_i \dot{\theta}_i L_j \dot{\theta}_j \cos(\theta_j - \theta_i) \\ & - m_{p+1} \dot{x}_A \sum_{i=1}^{p} L_i \dot{\theta}_i \sin\theta_i - \tfrac{1}{2} m_{p+1} L_{p+1} \dot{x}_A \dot{\theta}_{p+1} \sin\theta_{p+1} \\ & + m_{p+1} \dot{y}_A \sum_{i=1}^{p} L_i \dot{\theta}_i \cos\theta_i + \tfrac{1}{2} m_{p+1} L_{p+1} \dot{y}_A \dot{\theta}_{p+1} \cos\theta_{p+1} \\ & + \tfrac{1}{2} m_{p+1} L_{p+1} \dot{\theta}_{p+1} \sum_{i=1}^{p} L_i \dot{\theta}_i \cos(\theta_{p+1} - \theta_i) \end{aligned} \tag{5.42}$$

$$\hat{L}_{p+1} = \tfrac{1}{2} \rho_{p+1} \left\{ \begin{aligned} & \theta_{p+1}^2 v_{p+1}^2 + \dot{v}_{p+1}^2 + 2 x_{p+1} \dot{v}_{p+1} \dot{\theta}_{p+1} \\ & -2 \dot{x}_A \dot{\theta}_{p+1} v_{p+1} \cos\theta_{p+1} - 2 \dot{x}_A \dot{v}_{p+1} \sin\theta_{p+1} \\ & -2 \dot{y}_A \dot{\theta}_{p+1} v_{p+1} \sin\theta_{p+1} + 2 \dot{y}_A \dot{v}_{p+1} \cos\theta_{p+1} \\ & -2 \dot{\theta}_{p+1} v_{p+1} \sum_{i=1}^{p} L_i \dot{\theta}_i \sin(\theta_{p+1} - \theta_i) \\ & +2 \dot{v}_{p+1} \sum_{i=1}^{p} L_i \dot{\theta}_i \cos(\theta_{p+1} - \theta_i) \end{aligned} \right\} \tag{5.43}$$

Given the expressions in Eqs. (5.36-5.38) and Eqs. (5.41-5.43), we can define the exact system Lagrangian as given in Eq. (5.10) and proceed to generate the exact dynamical representation for this system using the approach described in section 5.3. We note here that in forming the system Lagrangian, the discrete and boundary portions as described by Eq. (5.10) are formed by summing the individual components for each link.

5.5 ACCURACY OF SOLUTION AND SPACE/TIME DERIVATIVES

5.5.1 Standard Approaches: Assumed Modes and FEM

We now make some observations about the accuracy of space/time derivatives computed from solutions based on the method of assumed modes or FEM. We note here

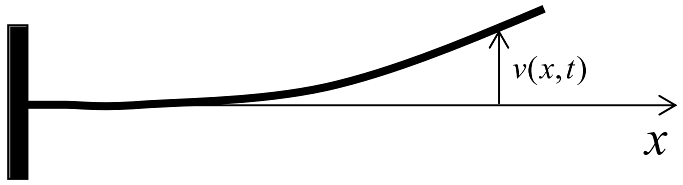

Figure 5.3. Clamped-free beam

that the primary objective of this section is to make observations on the impact of modeling assumptions, particularly the chosen spatial resolution of the model, on the error in satisfying the exact governing equations of motion. Again, we bring about the point that these errors are the analytical source terms used to create a benchmark problem for validating solution accuracy. Toward this end we consider, as an example, a clamped-free Euler-Bernoulli beam, as shown in Figure 5.3, whose exact dynamical representation is given by Eq. (5.44).

$$\rho\ddot{v}(x,t) + EI\frac{\partial^4 v(x,t)}{\partial x^4} = 0 \tag{5.44}$$

where $v(x,t)$ is the transverse beam deformation, ρ is the beam mass per unit length and EI is the beam bending stiffness constant.

Although an exact analytical solution exists for this problem, we look at solving this problem by approximate methods - the assumed modes method and the Finite Element method. First we consider solving this problem by the assumed modes method.

Equation (5.45) gives the form of the assumed modes model for the clamped-free beam.

$$M_A \ddot{\eta}_A + K_A \eta_A = 0 \tag{5.45}$$

where M_A is the mass matrix, K_A is the stiffness matrix, and η_A is a vector of time dependent flexible coordinates defined by

$$v(x,t) = \phi^T(x)\eta_A(t) \tag{5.46}$$

where $\phi(x)$ is a vector of assumed mode shape functions given by

$$\phi_i(x) = 1 - \cos(\tfrac{i\pi x}{L}) + \tfrac{1}{2}(-1)^{i+1}\left(\tfrac{i\pi x}{L}\right)^2 \tag{5.47}$$

Of course, this is only one of many choices for these mode shape functions. This particular choice is motivated by previous studies[20].

Thus we solve Eq. (5.45) for the time dependent coordinates, which we call $\overline{\eta}_A(t)$, in order to compute the solution for $\overline{v}(x,t) = \phi^T(x)\overline{\eta}_A(t)$ at all points along the beam and for all time.

One method to evaluate the accuracy of this solution is to compare with the exact analytical solution for the clamped-free beam which is given by

$$v_{true}(x,t) = \sum_{i=1}^{N} A_i y_i(x)\cos(2\pi f_i t + \alpha_i) \tag{5.48}$$

where $y_i(x)$ is the exact eigenfunction associated with the ith mode and is given by

$$y_i(x) = \cosh\left(\lambda_i x/L\right) - \cos\left(\lambda_i x/L\right) - \sigma_i\left(\sinh\left(\lambda_i x/L\right) - \sin\left(\lambda_i x/L\right)\right) \qquad (5.49)$$

The parameters λ_i and σ_i are constants associated with the ith mode. A_i is an amplitude constant and α_i a phase angle. They are determined from the initial conditions. f_i is the natural freqency in hertz associated with the ith mode.

Thus the error in the solution is given by

$$\delta v(x,t) = v_{true}(x,t) - \bar{v}(x,t) \qquad (5.50)$$

for some chosen time and spatial discretization.

Additionally, we can compute the higher-order space/time derivatives of Eq. (5.46) and (5.48) and compute the error in each in the way as we do in Eq. (5.50) (e.g. $\delta v'(x,t) = v'_{true}(x,t) - \bar{v}'(x,t)$). These include the time derivatives $\dot{\bar{v}}(x,t)$ and $\ddot{\bar{v}}(x,t)$, and the spatial derivatives $\bar{v}'(x,t)$, $\bar{v}''(x,t)$, $\bar{v}'''(x,t)$, and $\bar{v}''''(x,t)$.

Alternatively, we can develop a model for the clamped-free beam based on the Finite Element method, which will have a form identical to Eq. (5.45). However, the mass and stiffness matrices are formed in a different manner, primarily due to the chosen form of the mode shape functions. We proceed without reproducing these developments since they can be found in many books[35,20]. We do make note that the standard cubic shape functions are utilized in these solutions. The procedure involves producing the Finite Element solution and then evaluating the error in a similar manner to that given in Eq. (5.50).

Figures 5.4 and 5.5 each show three plots including the analytical solution for the clamped-free beam, the approximate solution, and the error in the deflection solution for the assumed modes solution and FEM solution, respectively. Figures 5.6-5.9 show the analytical solution, approximate solution, and corresponding error in the solutions for the higher-order time derivatives, for the assumed modes solution and FEM solution, respectively. Figures 5.10-5.17 show the results for the higher-order spatial derivatives, for the assumed modes solution and FEM solution. In these solutions, all parameters are set to unity including ρ, E, I, and L. The motion is solved for 4 seconds. The initial conditions are set with an initial deflection corresponding to the first mode with a tip deflection equaling 2% of the beam length from rest. Eight assumed modes are chosen to model the system. Four elements are chosen in order for the FEM model and assumed modes models to have an equal system order.

Now we consider the next test of the solution. That is, we substitute the solution into the exact equation of motion in order to produce an acceleration level error term as given in Eq. (5.51).

$$\bar{e}(x,t) = \rho\ddot{\bar{v}}(x,t) + EI\frac{\partial^4\bar{v}(x,t)}{\partial x^4} \tag{5.51}$$

Obviously, the exact solution has error which is identically zero for all space and time in Eq. (5.51). However, this is not the case for the solutions of approximate models. These acceleration level errors are shown in Figures 5.18 and 5.19 for the assumed modes solution and FEM solution respectively.

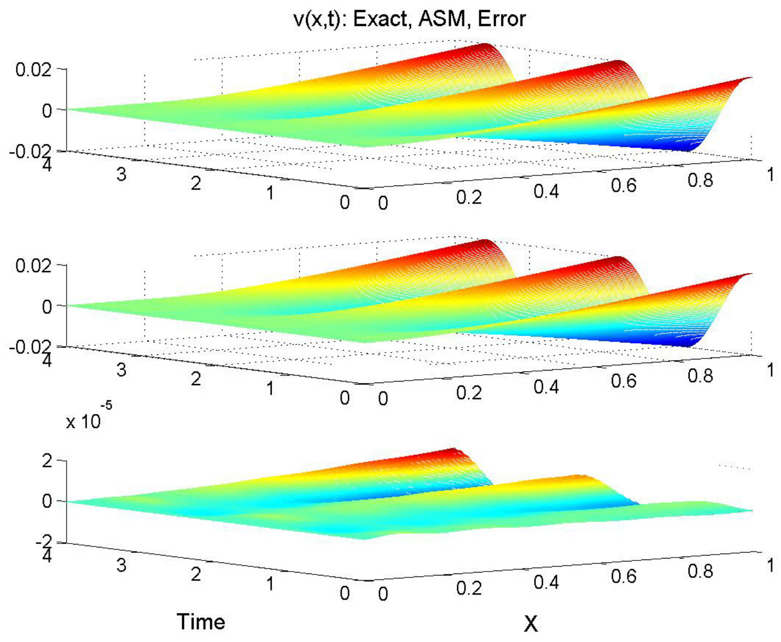

Figure 5.4. Assumed modes deflection solution

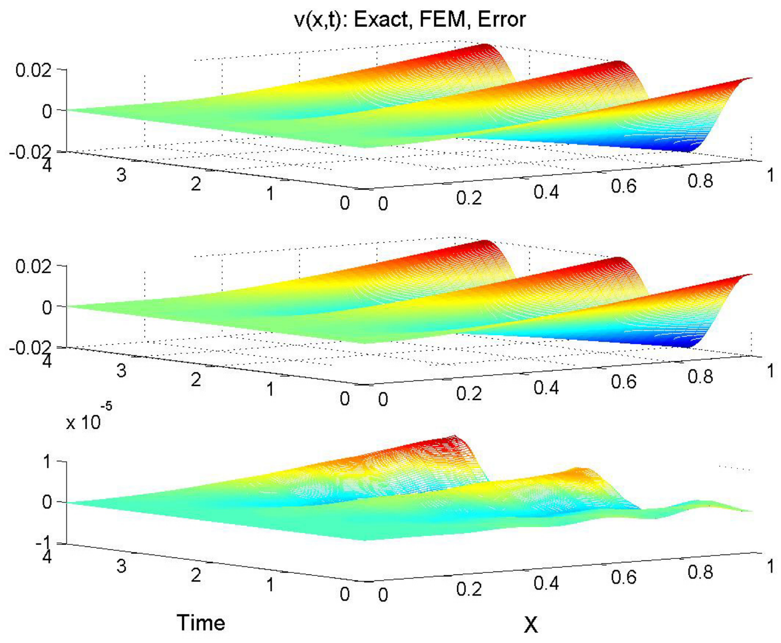

Figure 5.5. FEM deflection solution

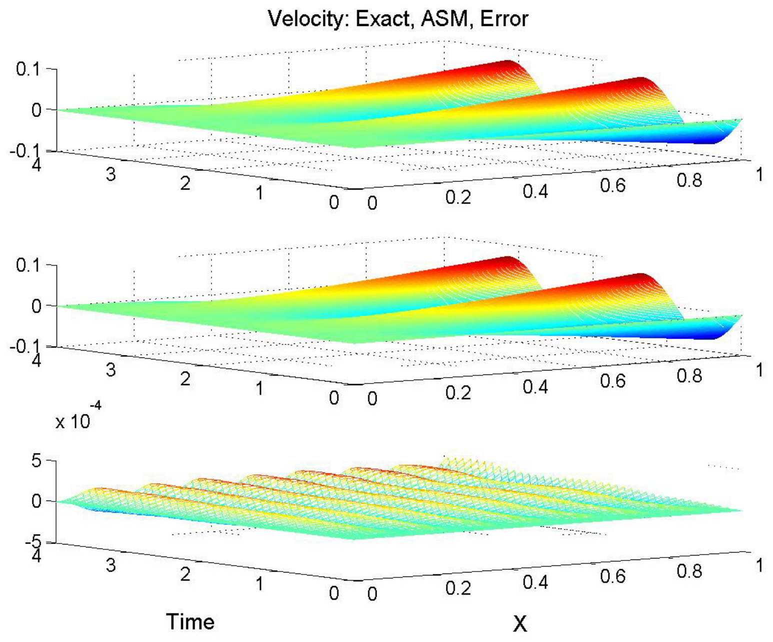

Figure 5.6. Assumed modes velocity solution

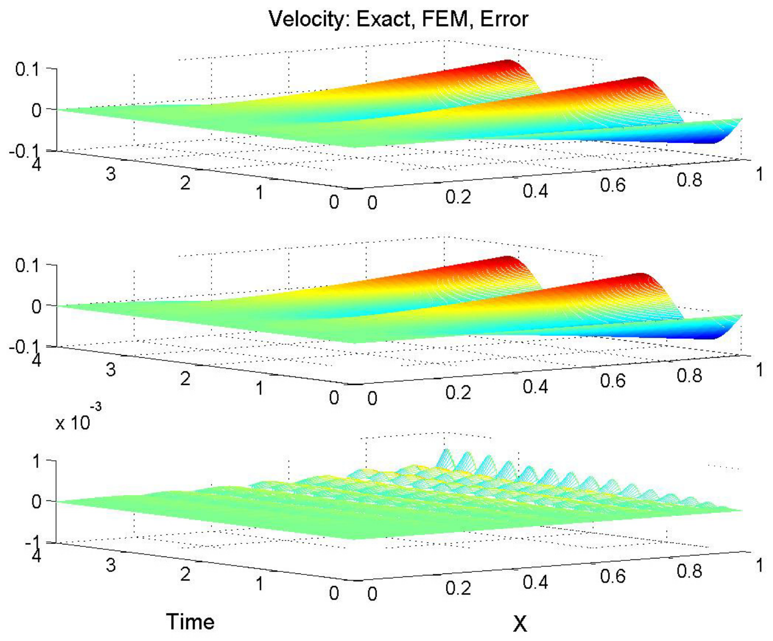

Figure 5.7. FEM velocity solution

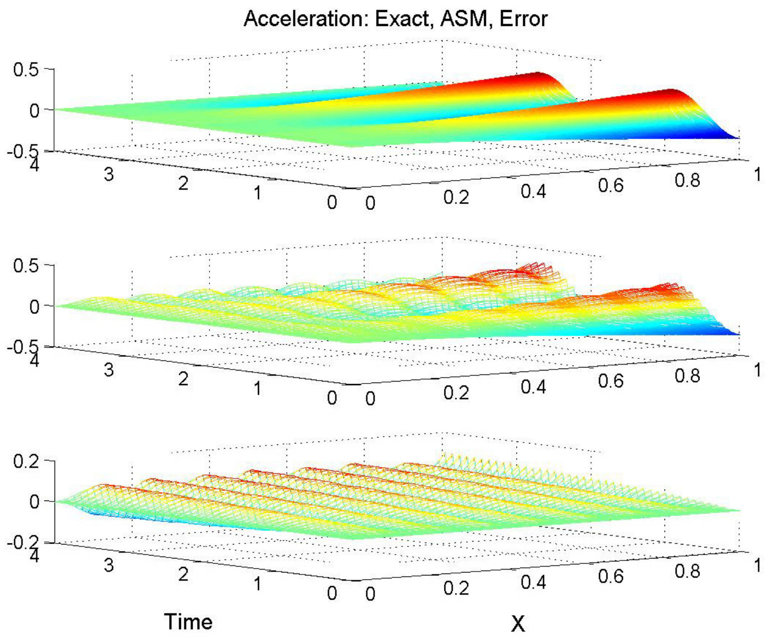

Figure 5.8. Assumed modes acceleration solution

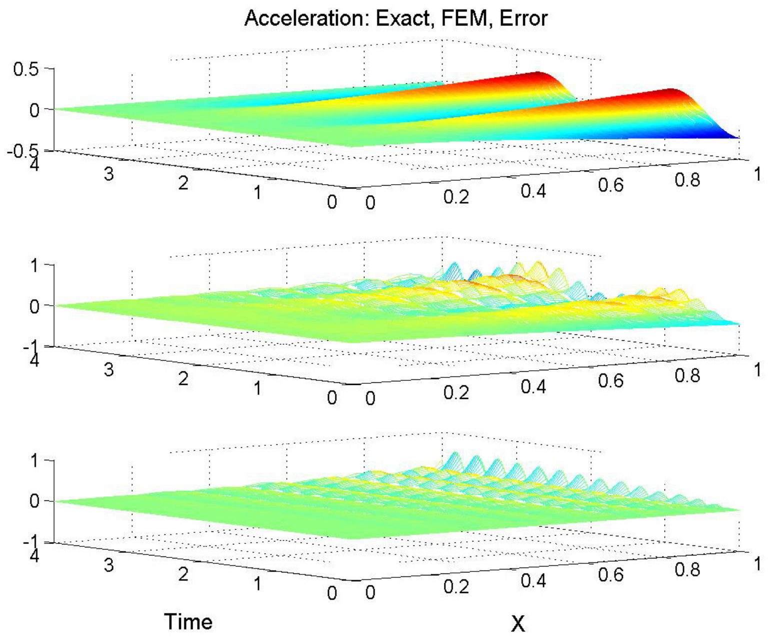

Figure 5.9. FEM acceleration solution

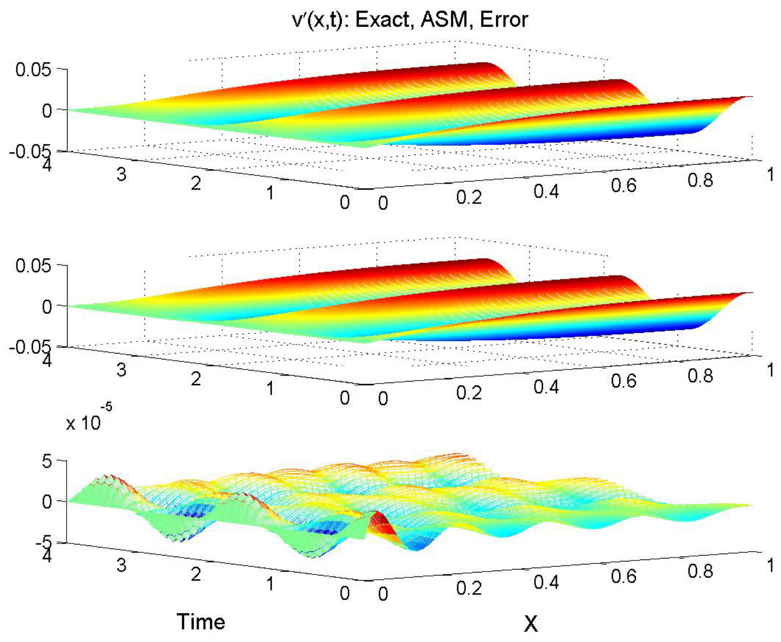

Figure 5.10. Assumed modes slope solution

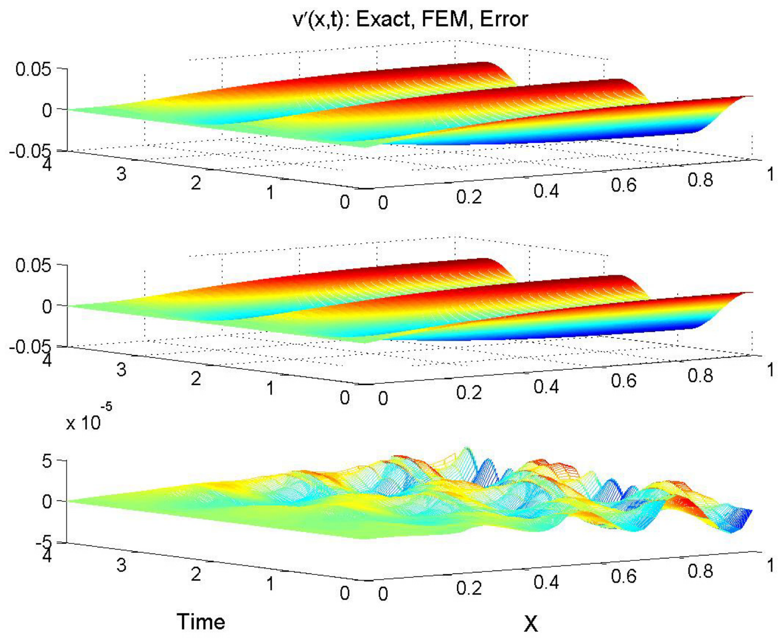

Figure 5.11. FEM slope solution

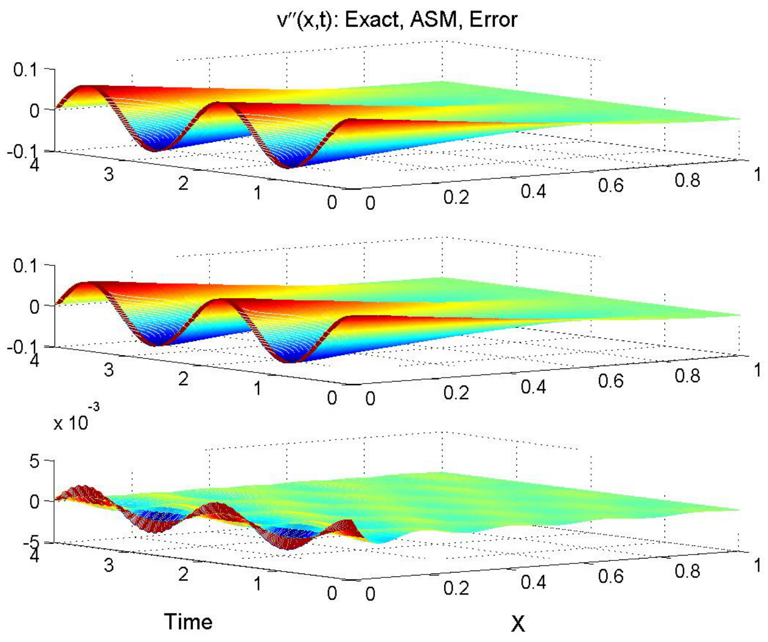

Figure 5.12. Assumed modes curvature solution

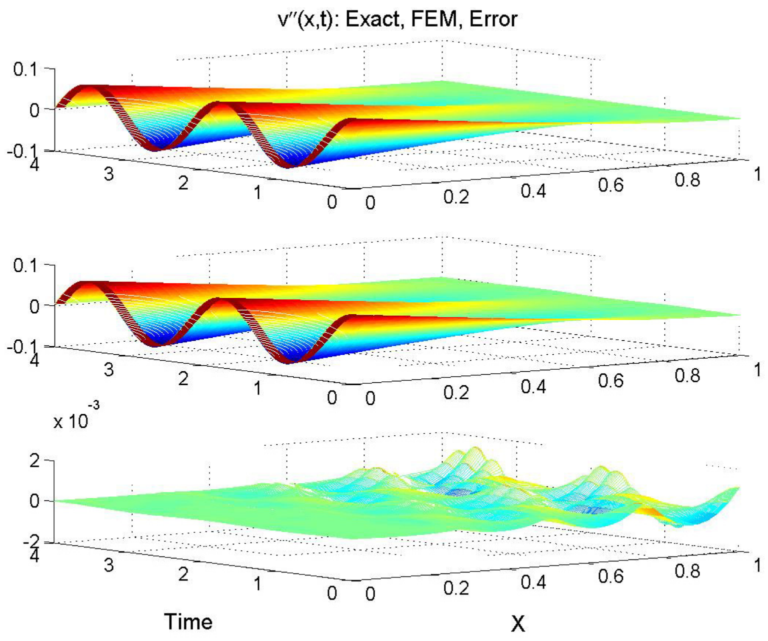

Figure 5.13. FEM curvature solution

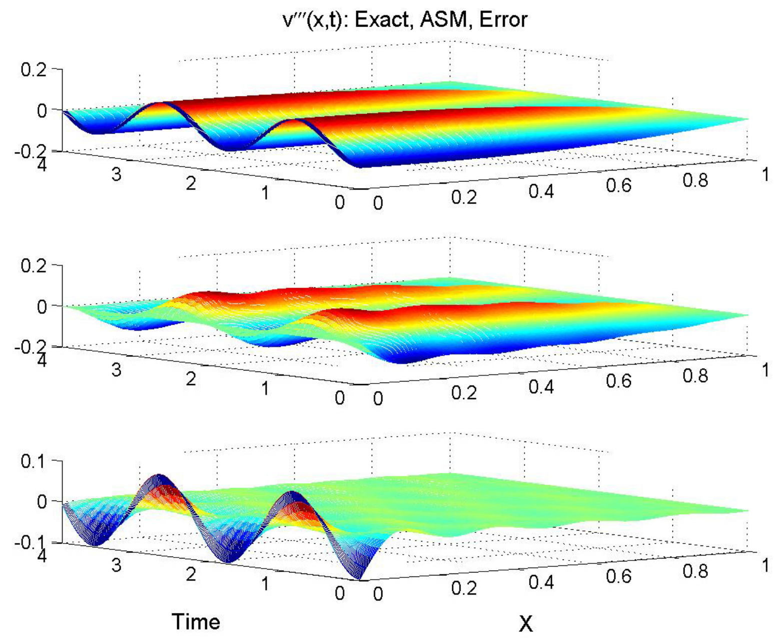

Figure 5.14. Assumed modes $v'''(x,t)$ solution

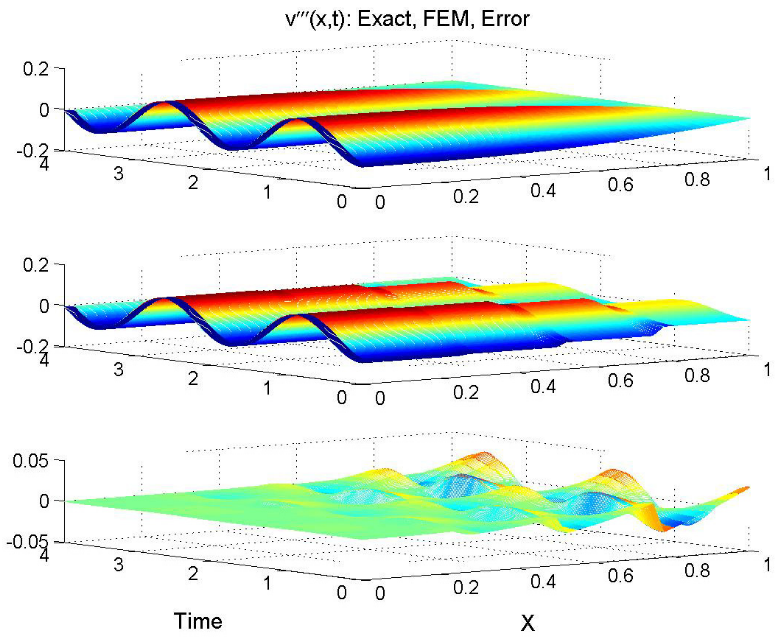

Figure 5.15. FEM $v'''(x,t)$ solution

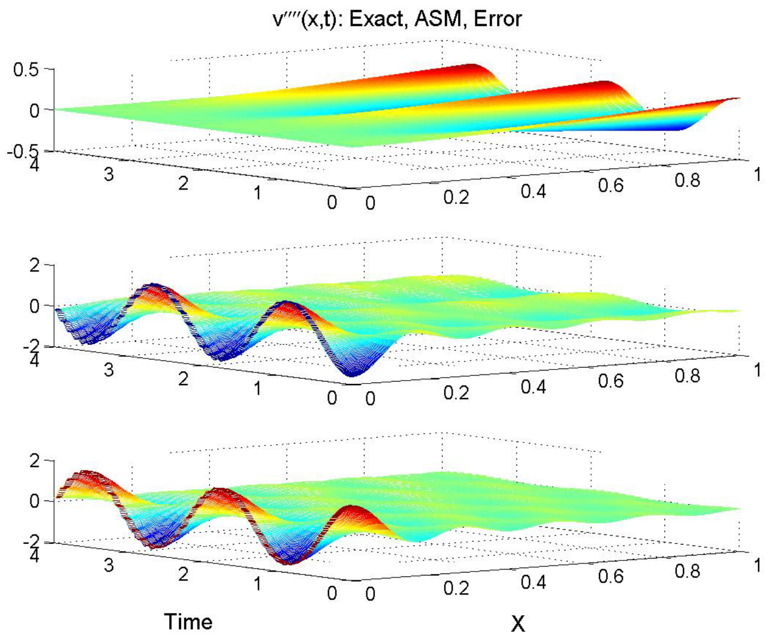

Figure 5.16. Assumed modes $v''''(x,t)$ solution

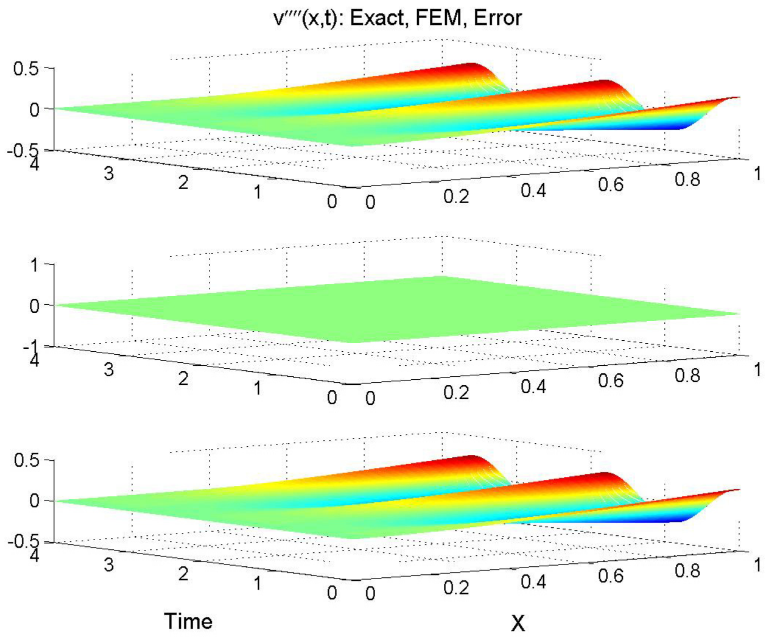

Figure 5.17. FEM $v''''(x,t)$ solution

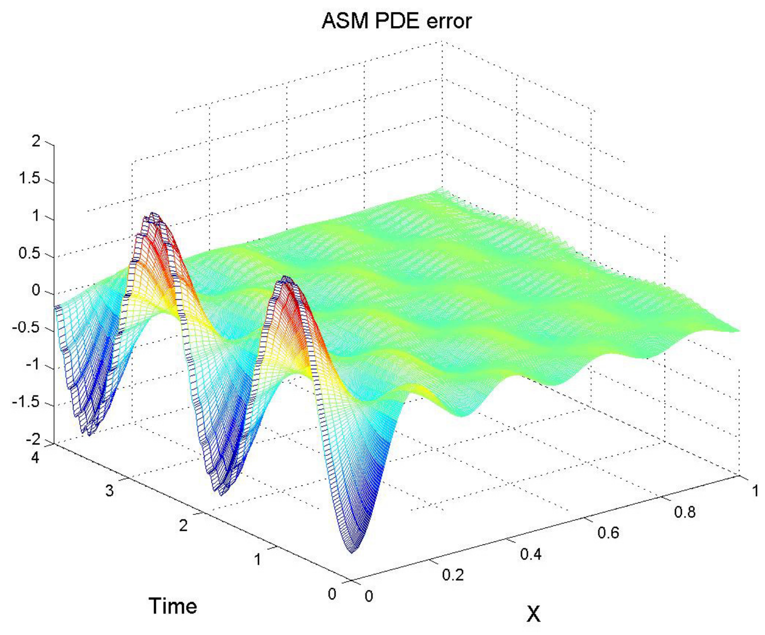

Figure 5.18. Error in exact PDE for assumed modes solution

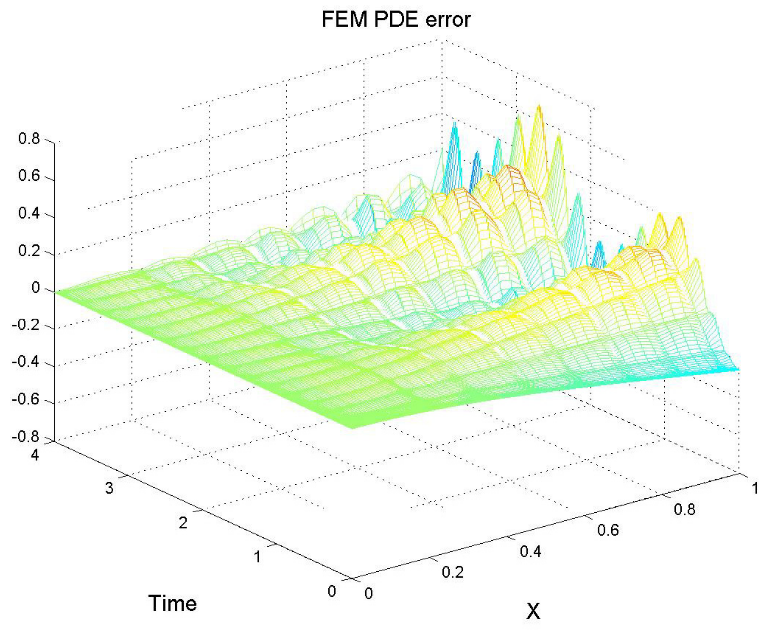

Figure 5.19. Error in exact PDE for FEM solution

We now make qualitative remarks regarding the solutions from the assumed modes method and the Finite Element Method by considering the solutions for equal system order (N=8) as seen in Figures 5.4-5.19 which compare the assumed modes solution and the FEM solution. We see that the error in the solutions for the transverse deflection is small and nearly identical for each solution. Generally, the solutions for the lowest order derivatives are best. Once we reach second-order derivatives with respect to time and third-order derivatives with respect to the spatial variable the results tend to become less accurate. In fact, the FEM solution is incapable of producing a nonzero fourth-order spatial derivative since the shape functions are cubic polynomials. Furthermore, the assumed modes method produces a solution which is poor near the clamped end; however, the solution is better near the free end. We find an opposite trend in the case of the FEM solution.

Clearly, when we are concerned with solutions for the time and spatial derivatives of the transverse deflection, neither the assumed modes method nor the FEM using cubic shape functions provides an acceptable result for all solution variables.

We make the point that, for complicated dynamical systems in which no analytical solutions exist, evaluating the error of the solution by Eq. (5.50) is not possible. However, evaluating the error in the solution by Eq. (5.51) is possible as long as an exact dynamical representation of the system can be produced. We addressed the issue of generating the exact dynamical representation previously in this chapter. The principle point we make in this section is that although the method of assumed modes and the

FEM are generally accepted to provide good solutions for the problem at hand, they do not satisfy the exact governing equations.

We now quantify the RMS errors for the assumed modes and FEM solutions for each variable for comparison. We adopt the following measure to compute the RMS error.

$$RMS = \int_0^{t_f} \int_0^L \overline{e}(x,t) dx dt \tag{5.52}$$

Table 5.1 shows the RMS errors for each variable for both solutions for an equal system order of eight. These results correspond to the figures previously presented. In Table 5.2, we present the RMS errors, for a system order of N=12. In both tables, the more accurate result is bolded.

Table 5.1. RMS errors of assumed modes and FEM solutions for equal system order (N=8)

Variable	ASM RMS	FEM RMS
$\delta v(x,t)$	3.9391e-4	**2.4448e-4**
$\delta \dot{v}(x,t)$	**7.8210e-3**	8.2523e-3
$\delta \ddot{v}(x,t)$	**4.3221**	7.6366
$\delta v'(x,t)$	1.3736e-3	**1.0980e-3**
$\delta v''(x,t)$	4.4623e-2	**2.7247e-2**
$\delta v'''(x,t)$	1.9413	**7.9135e-1**
$\delta v''''(x,t)$	4.6966e1	**1.0822e1**
$\overline{e}(x,t)$	4.6771e1	**1.3245e1**

Table 5.2. RMS errors of assumed modes and FEM solutions for equal system order (N=12)

Variable	ASM RMS	FEM RMS
$\delta v(x,t)$	1.2365e-4	**5.9574e-5**
$\delta \dot{v}(x,t)$	9.4236e-3	**9.3979e-3**
$\delta \ddot{v}(x,t)$	**1.2432e1**	2.0194e1
$\delta v'(x,t)$	5.3948e-4	**4.3413e-4**
$\delta v''(x,t)$	2.8571e-2	**1.6516e-2**
$\delta v'''(x,t)$	1.7401	**6.5761e-1**
$\delta v''''(x,t)$	5.9795e1	**1.3135e1**
$\overline{e}(x,t)$	5.8524e1	**2.4090e1**

Although not a focus of this chapter, we note a comparison of the accuracy of these two modeling approaches for computing the deflection, its space/time derivatives, and the resulting error in satisfying the exact governing PDE. These results show that the FEM method is generally more accurate with the exception of the accuracy of the acceleration solution. A most important point to be made is that although the deflection solution is quite accurate, there is a significant loss in accuracy as the results are viewed in the direction of increasing order of space/time differentiation. Of course, the highest order space and time derivatives (least accurate solutions) are those which show up in the governing PDE. Thus we make the point that these methods, which are geared toward accurate deflection solutions, do not accurately satisfy the exact governing equation of motion.

In Tables 5.3 and 5.4, we rearrange the results from Tables 5.1 and 5.2 in order to study the effect of system order on solution accuracy for each method.

Table 5.3. Effect of system order on assumed modes solution errors

Variable	ASM RMS (N=8)	ASM RMS (N=12)
$\delta v(x,t)$	3.9391e-4	**1.2365e-4**
$\delta \dot{v}(x,t)$	**7.8210e-3**	9.4236e-3
$\delta \ddot{v}(x,t)$	**4.3221**	1.2432e1
$\delta v^{'}(x,t)$	1.3736e-3	**5.3948e-4**
$\delta v^{''}(x,t)$	4.4623e-2	**2.8571e-2**
$\delta v^{'''}(x,t)$	1.9413	**1.7401**
$\delta v^{''''}(x,t)$	**4.6966e1**	5.9795e1
$\bar{e}(x,t)$	**4.6771e1**	5.8524e1

Table 5.4. Effect of system order on FEM solution errors

Variable	FEM RMS (N=8)	FEM RMS (N=12)
$\delta v(x,t)$	2.4448e-4	**5.9574e-5**
$\delta \dot{v}(x,t)$	**8.2523e-3**	9.3979e-3
$\delta \ddot{v}(x,t)$	**7.6366**	2.0194e1
$\delta v^{'}(x,t)$	1.0980e-3	**4.3413e-4**
$\delta v^{''}(x,t)$	2.7247e-2	**1.6516e-2**
$\delta v^{'''}(x,t)$	7.9135e-1	**6.5761e-1**
$\delta v^{''''}(x,t)$	**1.0822e1**	1.3135e1
$\bar{e}(x,t)$	**1.3245e1**	2.4090e1

The results of Tables 5.1 and 5.2 detail the effect of modeling approach on solution accuracy, whereas Tables 5.3 and 5.4 show the effect of increasing the system order on solution accuracy for each method. Interestingly, for each method we see an improvement in accuracy for the deflection solution and the first three spatial derivatives. We see the opposite trend for the velocity, acceleration, and fourth-order spatial derivative. Furthermore, we find less accuracy in satisfying the exact governing

PDE. We conclude by noting that regardless of system order, neither approach provides an accurate result in satisfying the exact PDE. We arrive at an interesting paradox in that the accuracy in satisfying the exact PDE follows an opposite trend to that of the deflection solution for increasing system order!

In the next section we study an alternative choice for shape or interpolation functions for the FEM beam element in an attempt to not only improve the accuracy of satisfying the exact PDE, but also to shed more light on the divergent trends in error in the deflection solution and error in satisfying the exact PDE.

5.5.2 Quintic Shape Functions

A limitation of cubic shape (interpolation) functions for the beam element is the fact that fourth-order spatial derivatives are zero, thus these shape functions are "not admissible"[35]. Cubic elements arise from an element containing four boundary coordinates (two at each boundary or node corresponding to deflection and slope). If we consider adding additional coordinates to the element boundary, then we can develop an element which requires a higher than cubic order shape function. In order to develop a quintic beam element, we add a single curvature coordinate to each boundary such that a total of six coordinates are required on the boundary of each element. Thus, the deflection over a single element is given by the following

$$v(x,t) = \sum_{i=1}^{6} \phi_i(x)\eta_i(t) \qquad (5.53)$$

where

$$\phi_i(x) = c_{1i} + c_{2i}x + c_{3i}x^2 + c_{4i}x^3 + c_{5i}x^4 + c_{6i}x^5$$

The boundary conditions for a single element are given by

$$\begin{aligned} &v(0,t) = \eta_1(t); \quad v'(0,t) = \eta_2(t); \quad v''(0,t) = \eta_3(t) \\ &v(h,t) = \eta_4(t); \quad v'(h,t) = \eta_5(t); \quad v''(h,t) = \eta_6(t) \end{aligned} \tag{5.54}$$

Thus the following conditions on the shape functions must hold

$$\begin{aligned} &\phi_1(0) = 1; && \phi_2(0) = \phi_3(0) = \phi_4(0) = \phi_5(0) = \phi_6(0) = 0 \\ &\phi_2'(0) = 1; && \phi_1'(0) = \phi_3'(0) = \phi_4'(0) = \phi_5'(0) = \phi_6'(0) = 0 \\ &\phi_3''(0) = 1; && \phi_1''(0) = \phi_2''(0) = \phi_4''(0) = \phi_5''(0) = \phi_6''(0) = 0 \\ &\phi_4(h) = 1; && \phi_1(h) = \phi_2(h) = \phi_3(h) = \phi_5(h) = \phi_6(h) = 0 \\ &\phi_5'(h) = 1; && \phi_1'(h) = \phi_2'(h) = \phi_3'(h) = \phi_4'(h) = \phi_6'(h) = 0 \\ &\phi_6''(h) = 1; && \phi_1''(h) = \phi_2''(h) = \phi_3''(h) = \phi_4''(h) = \phi_5''(h) = 0 \end{aligned} \tag{5.55}$$

Using the 36 conditions given in Eq. (5.55) we can compute the 36 unknown coefficients of the 6 quintic shape functions. The resulting quintic shape functions are given by Eq. (5.56).

$$\begin{aligned} \phi_1(x) &= 1 - 10\left(\tfrac{1}{h}\right)^3 x^3 + 15\left(\tfrac{1}{h}\right)^4 x^4 - 6\left(\tfrac{1}{h}\right)^5 x^5 \\ \phi_2(x) &= x - 6\left(\tfrac{1}{h}\right)^2 x^3 + 8\left(\tfrac{1}{h}\right)^3 x^4 - 3\left(\tfrac{1}{h}\right)^4 x^5 \\ \phi_3(x) &= \tfrac{1}{2}x^2 - \tfrac{3}{2}\left(\tfrac{1}{h}\right)x^3 + \tfrac{3}{2}\left(\tfrac{1}{h}\right)^2 x^4 - \tfrac{1}{2}\left(\tfrac{1}{h}\right)^3 x^5 \\ \phi_4(x) &= 10\left(\tfrac{1}{h}\right)^3 x^3 - 15\left(\tfrac{1}{h}\right)^4 x^4 + 6\left(\tfrac{1}{h}\right)^5 x^5 \\ \phi_5(x) &= -4\left(\tfrac{1}{h}\right)^2 x^3 + 7\left(\tfrac{1}{h}\right)^3 x^4 - 3\left(\tfrac{1}{h}\right)^4 x^5 \\ \phi_6(x) &= \tfrac{1}{2}\left(\tfrac{1}{h}\right)x^3 - \left(\tfrac{1}{h}\right)^2 x^4 + \tfrac{1}{2}\left(\tfrac{1}{h}\right)^3 x^5 \end{aligned} \tag{5.56}$$

We can utilize these quintic shape functions to compute the 6x6 elemental mass and stiffness matrices for the kth element as shown in Eqs. (5.57) and (5.58).

$$M_{ij}^k = \int_0^h \overset{k}{\rho}\, \phi_i(x^k)\phi_j(x^k)dx^k \tag{5.57}$$

$$K_{ij}^{k} = \int_{0}^{h} EI^{k} \phi_{i}''(x^{k}) \phi_{j}''(x^{k}) dx^{k} \tag{5.58}$$

A distributed load over the kth beam element will result in the following generalized forces.

$$\hat{f}_{i}^{k} = \int_{0}^{h} p^{k}(x^{k}, t) \phi_{i}(x^{k}) dx^{k} \tag{5.59}$$

One method for evaluating the accuracy of an approximation technique is to compare the natural frequencies from the approximate solution with the exact analytical solution. Table 5.5 shows a comparison of the natural frequencies for the standard cubic and quintic shape functions with the exact analytical solution for the first eight modes. The relative error is computed as $(\omega - \omega_{true}) / \omega_{true}$. The system order equals 24 for each case, i.e. we use 12 cubic elements and 8 quintic elements.

Table 5.5. Natural frequencies comparison for cubic and quintic beam elements

Exact Solution	Cubic		Quintic	
ω	n=12 (N=24)	Relative Error	n=8 (N=24)	Relative Error
3.5160	3.5160	0.0000	3.5160	0.0000
22.0345	22.0348	0.0000	22.0345	0.0000
61.6972	61.7049	0.0001	61.6972	0.0000
120.9019	120.9586	0.0005	120.9020	0.0000
199.8595	200.1100	0.0013	199.8600	0.0000
298.5555	299.3686	0.0027	298.5596	0.0000
416.9908	419.1355	0.0051	417.0120	0.0001
555.1652	560.0272	0.0088	555.2072	0.0001

As can be seen from Table 5.5, the higher-order quintic shape functions show an improvement in computing natural frequencies. It is well known (and was shown in the previous section) that the accuracy of the FEM typically improves by increasing the number of elements (n), or put it another way by increasing the system order (N). The results here show that by utilizing higher-order shape functions the effect is more dramatic. In fact, we can reduce the system order and compute solutions with higher accuracy than those derived from cubic shape functions as shown in Table 5.6.

Table 5.6. Natural frequencies comparison for reduced system order using quintic beam elements

Exact Solution	**Cubic**		**Quintic**	
ω	**n=12 (N=24)**	**Relative Error**	**n=5 (N=15)**	**Relative Error**
3.5160	3.5160	0.0000	3.5160	0.0000
22.0345	22.0348	0.0000	22.0345	0.0000
61.6972	61.7049	0.0001	61.6973	0.0000
120.9019	120.9586	0.0005	120.9036	0.0000
199.8595	200.1100	0.0013	199.8699	0.0001
298.5555	299.3686	0.0027	298.9066	0.0012
416.9908	419.1355	0.0051	418.1411	0.0028
555.1652	560.0272	0.0088	558.5964	0.0062

Furthermore, we present a set of results for the RMS errors for the deflection, space/time derivatives and error in satisfying the governing PDE for a FEM approach using quintic elements as we did for the cubic element in the previous section. Table 5.7 shows a comparison of the RMS errors for cubic versus quintic beam elements for each solution variable. The system order is equal for each model, i.e. 6 cubic elements versus 4 quintic elements to represent the clamped-free beam.

Table 5.7. Comparison of FEM RMS solution errors

Variable	**Cubic FEM RMS (n=6/N=12)**	**Quintic FEM RMS (n=4/N=12)**
$\delta v(x,t)$	5.9574e-5	**1.1573e-7**
$\delta \dot{v}(x,t)$	9.3979e-3	**1.6224e-4**
$\delta \ddot{v}(x,t)$	2.0194e1	**2.3349e-1**
$\delta v^{'}(x,t)$	4.3413e-4	**4.2982e-6**
$\delta v^{''}(x,t)$	1.6516e-2	**1.9170e-4**
$\delta v^{'''}(x,t)$	6.5761e-1	**9.5304e-3**
$\delta v^{''''}(x,t)$	1.3135e1	**2.8523e-1**
$\overline{e}(x,t)$	2.4090e1	**3.8064e-1**

Table 5.7 shows an improvement in accuracy for each solution variable, including a more accurate satisfaction of the exact PDE. The plots for the exact solution, FEM solution, and error are given for each variable listed in Table 5.7 are given in Figures 5.20-5.27. As can be see the errors are significantly reduced, especially for the solutions for transverse deflection and the low-order derivatives. Significant improvement is found for the acceleration and fourth-order spatial derivative solutions as well, which very importantly with regard to this study, improves the satisfaction of the exact PDE governing equation of motion. Thus, when we desire that the solution locally satisfy the exact governing equations of motion, we must increase the order of the locally defined shape functions. Quintic shape functions represent a step in that direction; however, they do not completely solve the issue of simultaneously providing a good deflection level solution while accurately satisfying the exact PDE. Of course, we could add even more boundary coordinates on the elements in order to extent this idea to higher-order.

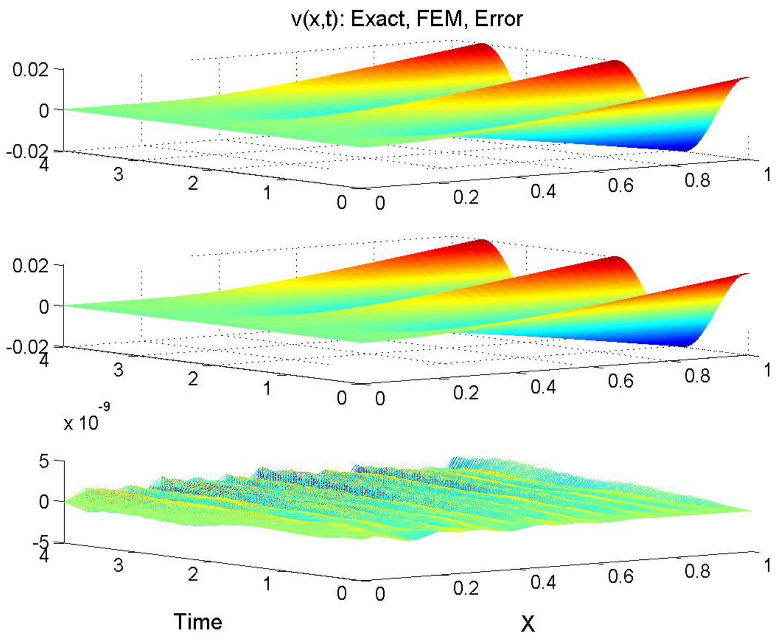

Figure 5.20. Quintic FEM deflection solution

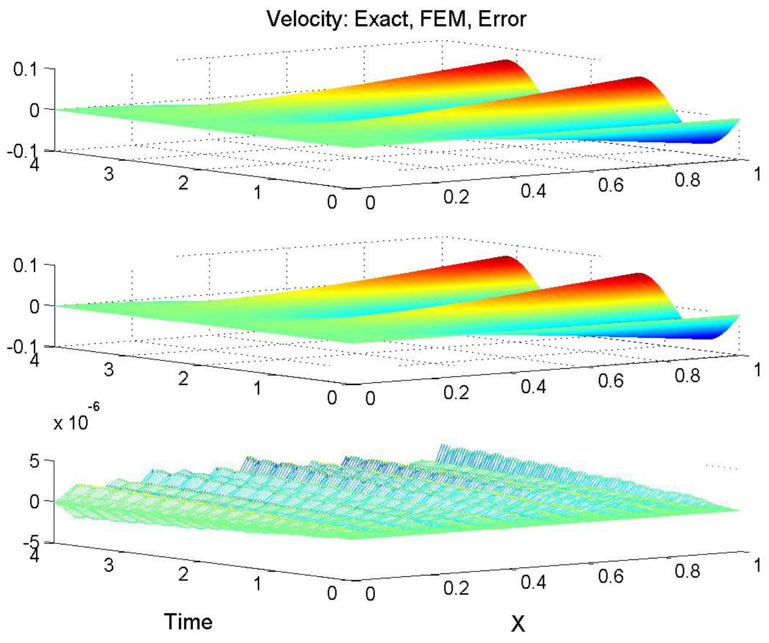

Figure 5.21. Quintic FEM velocity solution

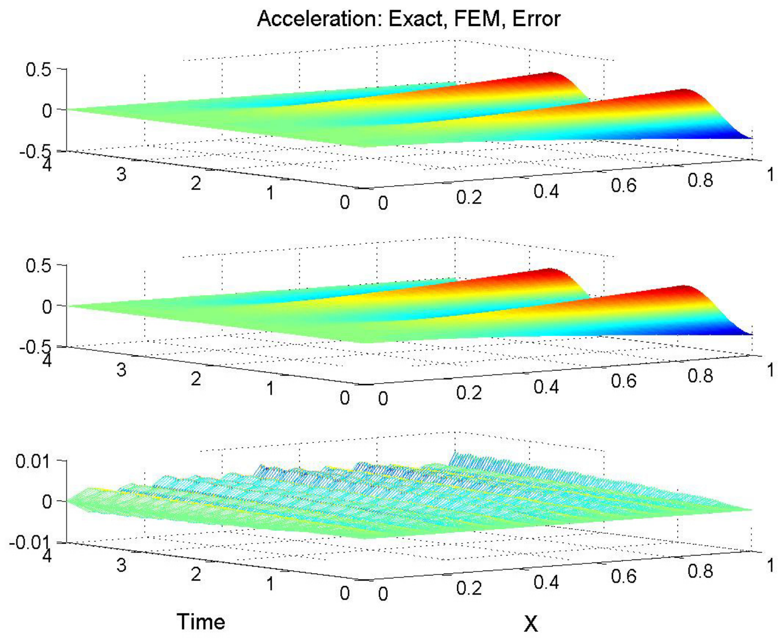

Figure 5.22. Quintic FEM acceleration solution

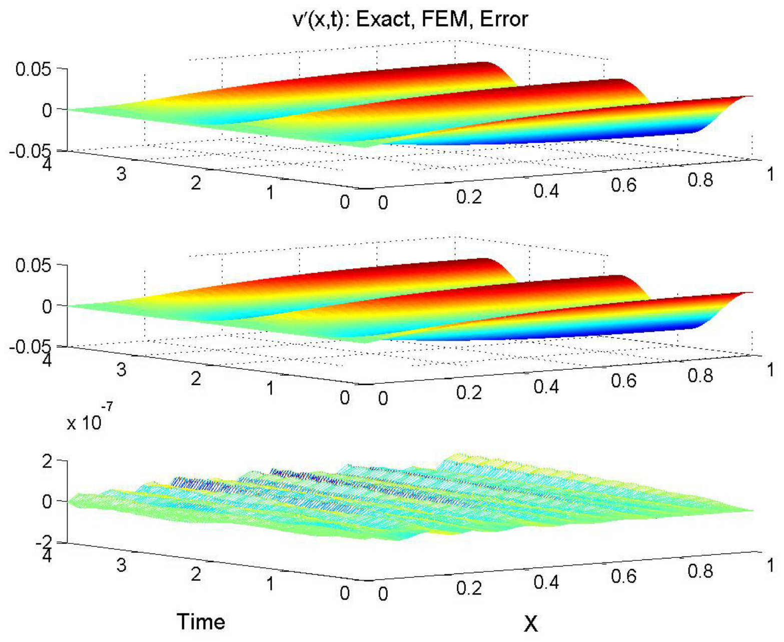

Figure 5.23. Quintic FEM slope solution

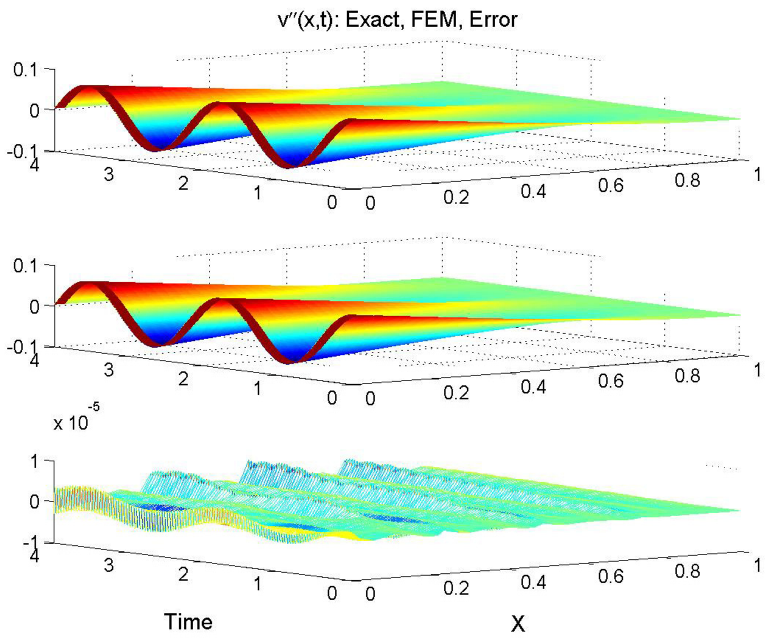

Figure 5.24. Quintic FEM curvature solution

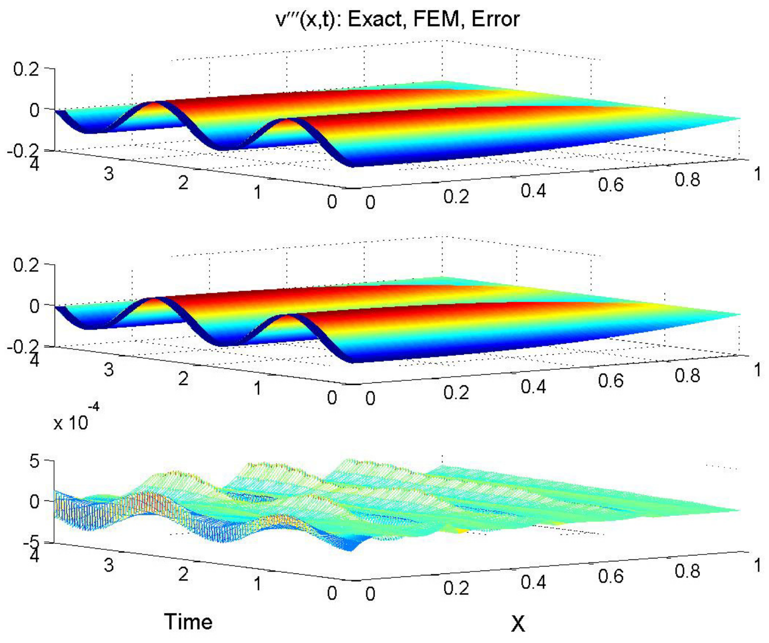

Figure 5.25. Quintic FEM $v'''(x,t)$ solution

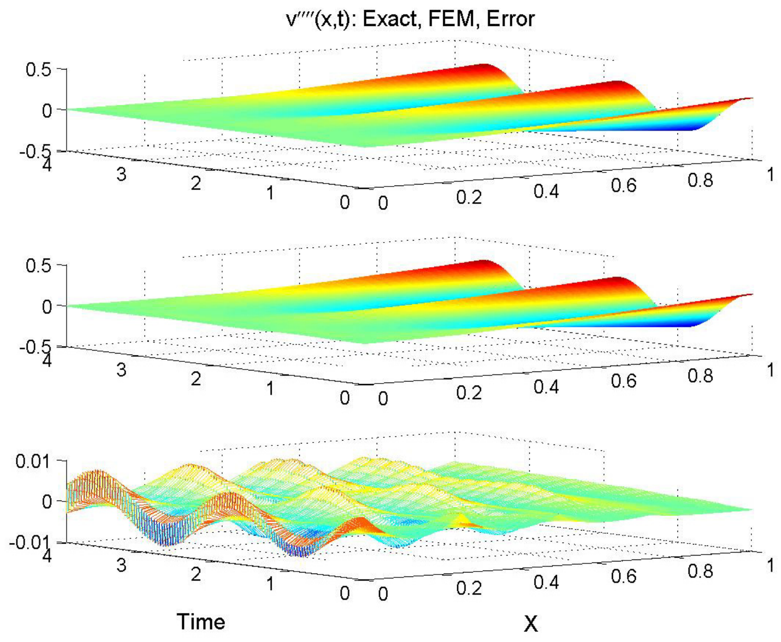

Figure 5.26. Quintic FEM $v''''(x,t)$ solution

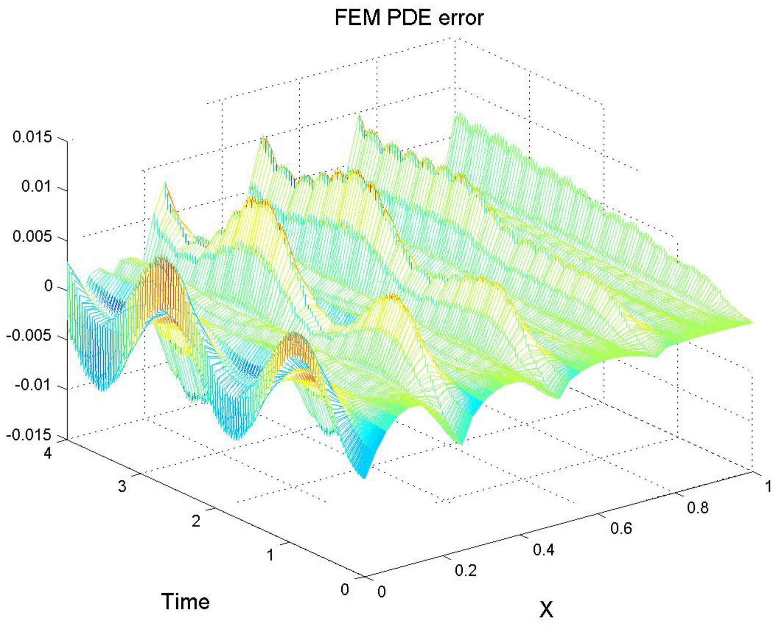

Figure 5.27. Error in exact PDE for quintic FEM solution

5.6 SUMMARY

In this chapter, we presented some ideas used in the validation of solution accuracy of dynamical systems. We reviewed several approaches including the method of exact solutions, the method of manufactured solutions, and the method of nearby problems. The method of manufactured solutions and the method of nearby problems, which are popular methods for rigorously validating solution accuracy for complicated dynamical systems, depend upon computing an acceleration level error, which is a departure from the exact dynamical equations. This acceleration level is used to compute fictitious forcing functions which provide a means of creating a benchmark problem which has an "exact" solution. This is the most important part of the analysis and must be performed with careful, expert attention. Therefore, in this chapter we investigated two issues related to this problem. Firstly, we presented a method for generating the exact ODE/PDEs and corresponding boundary conditions. Secondly, we looked at a simple clamped-free cantilever beam to illustrate in detail the individual sources of solution error that contribute the errors in the exact ODE/PDEs and boundary conditions that result from standard modeling approaches –an assumed modes approach (Ritz Method) or a FEM approach.

The method described in this chapter to automatically generate the exact ODE/PDEs and corresponding boundary conditions for a general multibody distributed parameter system is a significant development. As is the case when dealing with dynamical systems, the process of validating the accuracy of solutions is complicated by the fact that the dynamical model changes greatly from one system to another. Therefore, generating the exact dynamical representation is an issue that must be addressed. The method presented here provides a means to easily generate an exact dynamical representation for a class of distributed parameter systems. The examples show that this method is quite accurate and is suitable for validating solution accuracy.

The results for the clamped-free beam example demonstrated that, first and foremost, although the solution for the transverse deflection and first-order space/time derivatives are computed quite accurately the higher-order space/time derivatives are not. In fact, if we increase the system order of the approximate model by either choosing a larger number of assumed modes or a larger number of elements, we find an opposite trend for the solutions for the acceleration and fourth-order spatial derivatives. The principle point we make is that although the method of assumed modes and the FEM are generally accepted as providing accurate solutions for deflection, they do not satisfy the exact governing equations. This is an important issue to keep in mind when validating solution accuracy.

A limitation of the beam finite element which utilizes cubic polynomial interpolation functions is that fourth-order spatial derivatives are identically zero, thus not being admissible in the sense that the FEM solution does not satisfy the exact governing PDE. Therefore, we studied the use of quintic polynomial interpolation functions which provide nonzero fourth-order spatial derivatives. We found that the accuracy of the solution for the transverse deflection and *all* of its space/time derivatives are improved with the quintic interpolation functions over the standard cubic interpolation functions. Furthermore, we find that the computed natural frequencies associated with the quintic interpolation functions are more accurate than those from cubic interpolation functions even when the system order is reduced. Considering the improved accuracy in all quantities studied and the ability to reduce system order and maintain accuracy, we find no reason to choose cubic shape functions over quintic shape functions. However, quintic shape functions do not improve, in a significant way, the error in satisfying the exact governing PDE.

The apparent theoretical disconnect where, as we have shown by example, accurate solutions for deflection do not result in accurate satisfaction of the exact dynamical governing equations of motion is an issue that should be kept in mind when validating solution accuracy, and poses an interesting research question.

CHAPTER VI

SUMMARY AND FUTURE WORK

This dissertation presented new results in estimation, modeling, and validation of dynamical systems using automatic differentiation. In the second chapter, an overview of computerized differentiation was presented. Computerized differentiation by symbolic methods were compared with automatic differentiation. The basic difference in these approaches is that automatic differentiation by OCEA is a highly optimized approach, which frees the analyst from onscreen type programming and porting of subroutines which is typical of the symbolic approach. For simple problems, especially those which require viewing the differentiated expressions, symbolic differentiation is the preferred method. When developing a high-level code automatic differentiation by OCEA is preferred. The types of applications developed in this dissertation are of this type.

In Chapter III, we presented some new results in estimation, control and optimization of dynamical systems. Standard first-order algorithms were reviewed. Higher-order sensitivity calculations were presented for developing new algorithms. One particularly important development is the higher-order generalization of the state transition matrix. This development makes it possible to implement many of the new methods developed in this chapter including the GLSDC algorithm for orbit determination, the method of differential corrections for trajectory optimization, and higher-order laws for computing midcourse corrections. Furthermore, the properties of the first- and higher-order state transition matrices were discussed. We showed that the well-known and valuable group chain-rule property of the first-order state transition matrix also holds for the second- and higher-order state transition matrices as well. It is expected that the higher-order generalization of the state transition matrix will have an impact on filtering applications as well.

Future research can dramatically reduce the number of higher-order partial derivatives required, by exploiting the symmetry and sparsity of the gradient tensor operators. For example, with 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 were attempted. Future development of these algorithms should include the incorporation of advanced optimization features such as constrained optimization capability and line searching. Line searching for the second-and higher-order methods should be viewed as a future research problem. The effect of higher-order algorithms for computing midcourse corrections should be studied with respect to initial condition errors (inaccurate execution of midcourse correction) and poorly modeled dynamics, e.g. inaccurate force models.

It is well known that the first-order state transition matrix provides a means of mapping uncertainties throughout time. This concept is usually referred to as propagation of uncertainty. Propagation of uncertainty is, in fact, a key feature of filtering algorithms such as the Kalman filter. The other key feature of filtering algorithms is the state propagation method. Many approaches have been investigated to propagate the state and covariance in a linear fashion. Currently, much attention has been given to nonlinear filtering methods which attempt to propagate the state or uncertainty[49] in a way that retains some of the nonlinear information in the problem. We expect that the higher-order generalizations of the state transition matrix will have an impact in these areas.

In Chapter IV, a new method for modeling dynamical systems was presented. The key development is a demonstration of the capability of automatic differentiation in automatically generating and integrating the equations of motion by simply specifying the system Lagrangian and the system constraints. No further work need be done by the analyst in order to produce the system response. Several examples were presented for rigid and flexible linked mechanical systems in open- and closed-chain topologies. It is remarkable that a single approach can produce such a variety of dynamical models, including simulations, with minimal coding.

The results of Chapter IV show that the OCEA approach is superior to other AD approaches for deriving equations of motion. With respect to coding effort, we find that with OCEA no code needs to be generated offline. This is not the case when using other programs such as ADIFOR or AUTODERIVE. All required first- and second-order partial derivatives are computed within the OCEA-FORTRAN environment. Furthermore, the accuracy of the OCEA solutions when evaluating the satisfaction of position and velocity level constraints, or when evaluating the errors between the

OCEA-enabled solution and that from hand-derived equations of motion is profoundly better.

Future work in modeling of dynamical systems should include extending the method for general 3D motion, and developing a framework in which arbitrary selections for joint interconnections can be made. In the interest of improved performance, measures should be considered for optimizing the process of deriving the equations of motion. Furthermore, approaches for solving constrained systems which minimize the problem dimension should be considered, especially for highly constrained systems containing many bodies.

In Chapter V, we presented some ideas used in the validation of solution accuracy for dynamical systems. We reviewed several approaches including the method of exact solutions, the method of manufactured solutions, and the method of nearby problems. The latter two methods rely upon producing an exact dynamical relationship for the system whose response is being validated. Therefore, a method was introduced and demonstrated by which the exact PDE/ODEs and boundary conditions for a distributed-parameter system can be automatically generated using automatic differentiation.

The method of manufactured solutions and the method of nearby problems upon computing an acceleration level error, which is a departure from the exact dynamical equations. This acceleration level is used to compute analytical source terms by inverse dynamics, which are used to create a benchmark problem which has an “exact” analytical solution. Computing these analytical source terms is the most important part of the analysis and must be performed with careful, expert attention. Therefore, we described some alternative ideas for creating the benchmark problem when using OCEA to generate the exact dynamical representation. Additionally, we looked at a simple clamped-free cantilever beam to illuminate some of the more fundamental aspects of solving problems for flexible systems that result from taking either and assumed modes approach or a FEM approach.

The results for the clamped-free beam example demonstrated that, first and foremost, although the solution for the transverse deflection and first-order space/time derivatives are computed quite accurately, the higher-order space/time derivatives are not for an assumed modes or FEM approach. We make the point that these methods, which are generally accepted to be “good”, do not satisfy the exact governing equation. This is an issue that an analyst conducting a validation study on any dynamical system must keep in mind since “good” solutions can result in large errors in satisfying the exact governing equations of motion.

A limitation of the beam finite element which utilizes cubic polynomial interpolation functions is that fourth-order spatial derivatives are identically zero, thus not being admissible in the sense that the FEM solution does not satisfy the exact governing PDE. Therefore, we studied the use of quintic polynomial interpolation functions which provide nonzero fourth-order spatial derivatives. We found that the accuracy of the solution for the transverse deflection and all of its space/time derivatives are improved with the quintic interpolation functions over the standard cubic interpolation functions. Furthermore, we find that the computed natural frequencies associated with the quintic interpolation functions are more accurate. Considering the improved accuracy in all quantities studied and the ability to reduce system order and maintain accuracy, we find no reason to choose cubic shape functions over quintic shape functions. However, an increase in the order of the locally defined interpolation functions does not completely resolve the issue of the disparity in accuracy between the deflection solution and satisfaction of the exact governing equations of motion. This issue should be considered as a future research question.

The results of this dissertation have provided insights regarding the use of the automatic differentiation tool OCEA in solving a variety of problems in estimation, modeling and validation of dynamical systems. The usefulness of OCEA is profound since there are many applications in science and engineering which require computing partial derivatives. A few of these applications have been studied in this dissertation. With regard to ease of use, it should be noted that this work, which covers three distinct areas of the study of dynamics and control, was completed over the course of 14 months beginning with an average knowledge of FORTRAN programming.

REFERENCES

1. Hunt, B.R., Lipsman, R.L., and Rosenberg, J.M., *A Guide to MATLAB: for Beginners and Experienced Users*, Cambridge University Press, New York, 2001.

2. Richards, D., *Advanced Mathematical Methods With Maple*, Cambridge University Press, Cambridge, 2001.

3. Larsen, R.W., *Introduction to Mathcad 11*, Prentice Hall, Englewood Cliffs, New Jersey, 2004.

4. Macsyma Users Guide, Macsyma, Inc., Boston, 1998.

5. Griewank, A., "On Automatic Differentiation" in *Mathematical Programming: Recent Developments and Applications*, edited by M. Iri and K. Tanabe, Kluwer Academic Publishers, Amsterdam, 1989, pp. 83-108.

6. Bischof, C., A. Carle, G. Corliss, A. Griewank, and P. Hovland, "ADIFOR: Generating Derivative Codes from Fortran Programs," *Scientific Programming*, Vol. 1, 1992, pp. 1-29.

7. Bischof, C., A. Carle, P. Khademi, A. Mauer, and P. Hovland, ADIFOR 2.0 User's Guide", Technical Report ANL/MCS-TM-192, Mathematics and Computer Science Division, Argonne National Laboratory, Argonne, IL, 1995.

8. Eberhard, P. and C. Bischof, Automatic Differentiation of Numerical Integration Algorithms, Technical Report ANL/MCS-P621-1196, Mathematics and Computer Science Division, Argonne National Laboratory, Argonne, IL, 1996.

9. Durrbaum, A., Klier, W., and Hahn, H., "Comparison of Automatic and Symbolic Differentiation in Mathematical Modeling and Computer Simulation of Rigid Body Systems", *Multibody System Dynamics*, Vol. 7, 2002, pp. 331-355.

10. Turner, J.D., "Quaternion-Based Partial Derivative and State Transition Matrix Calculations for Design Optimization," *40th AIAA Aerospace Sciences Meeting And Exhibit*, Reno, Nevada, January 14-17 2002.

11. Turner, J.D., "Object Oriented Coordinate Embedding Algorithm for Automatically Generating the Jacobian and Hessian Partials of Nonlinear Vector Functions," Invention Disclosure, University Of Iowa, Iowa City, IA, May 2002.

12. Turner, J.D., "The Application of Clifford Algebras for Computing the Sensitivity Partial Derivatives of Linked Mechanical Systems, *USNCTAM14: Fourteenth U.S.*

National Congress of Theoretical and Applied Mechanics, Blacksburg, Virginia, June 23-28, 2002 (invited paper).

13. Turner, J.D., "Automated Generation of High-Order Partial Derivative Models," *AIAA Journal*, Vol. 51, No. 8, August 2003, pp. 1590-1598.

14. Turner, J.D., "Generalized Gradient Search and Newton's Methods for Multilinear Algebra Root-Solving and Optimization Applications," *John L. Junkins Astrodynamics Symposium*, George Bush Conference Center, College Station, Texas, May 23-24, 2003, AAS 03-261 (invited paper).

15. Turner, J.D., "Generalized Gradient Search and Newton's Methods for Multilinear Algebra Root-Solving and Optimization Applications," *Journal Of The Astronautical Sciences*, (to appear in a special issue commemorating the John L. Junkins Astrodynamics Symposium).

16. Griffith, D.T., Turner, J.D., and Junkins, J.L., "An Embedded Function Tool for Modeling and Simulating Estimation Problems in Aerospace Engineering," *AAS/AIAA Spaceflight Mechanics Meeting*, Maui, HI, USA, February 8-12, 2004, Paper AAS 04-148.

17. Griffith, D.T., Turner, J.D., Vadali, S.R., and Junkins, J.L., "Higher Order Sensitivities for Solving Nonlinear Two-Point Boundary Value Problems," *AIAA/AAS Astrodynamics Specialist Conference and Exhibit*, Providence, RI, August 16-19, 2004, AIAA-2004-5404.

18. Crassidis, J.L., and Junkins, J.L., *Optimal Estimation of Dynamic Systems*, CRC Press, Boca Raton, FL, 2004.

19. Battin, Richard H., *An Introduction to the Mathematics and Methods of Astrodynamics*, AIAA Education Series, Reston, VA, 1987.

20. Junkins, J.L. and Kim, Y., *Introduction to Dynamics and Control of Flexible Structures*, AIAA Education Series, Washington, DC, 1993.

21. Moler, C. and van Loan, C., "Nineteen Dubious Ways to Compute the Exponential of a Matrix," *SIAM Review*, Vol. 20, No. 4, 1978, pp. 801-836.

22. Bryson, A.E., and Ho, Y., *Applied Optimal Control*, Taylor and Francis, Levittown, PA, 1975.

23. Junkins, J.L., and Turner, J.D., *Optimal Spacecraft Rotational Maneuvers: Studies in Astronautics*, Vol. 3, Elsevier Science Publishers, Amsterdam, The Netherlands, 1986.

24. Seywald, H., "Trajectory Optimization Based on Differential Inclusions*," Journal of Guidance, Control, and Dynamics*, Vol. 17, No. 3, May-June 1994, pp. 480-487.
25. Griffith, D.T., Sinclair, A.J., Turner, J.D., Hurtado, J.E., and Junkins, J.L., "Automatic Generation and Integration of Equations of Motion by Operator-Overloading Techniques," *AAS/AIAA Spaceflight Mechanics Meeting*, Maui, HI, February 8-12, 2004, Paper AAS 04-242.

26. Griffith, D.T., Junkins, J.L., and Turner, J.D., "Automatic Generation and Integration of Equations of Motion for Linked Mechanical Systems," *6th International Conference on Dynamics and Control of Systems and Structures in Space*, July 18-22, 2004, Riomaggiore, Cinque Terre, Liguria, Italy.

27. Baruh, H., *Analytical Dynamics*, McGraw Hill, New York, 1998.

28. Schaub, H., and Junkins, J.L., *Analytical Mechanics of Space Systems*, AIAA, Reston, VA, 2003.

29. Banerjee, A. K., "Contributions of Multibody Dynamics to Space Flight: A Brief Review", *Journal of Guidance, Control, and Dynamics*, Vol. 26, No. 3, May-June 2003, pp. 385-394.

30. Schiehlen, W. (editor), *Multibody Systems Handbook*, Springer-Verlag, New York, 1990.

31. Lee, M.G., "Application of Automatic Differentiation in Numerical Solution of a Flexible Mechanism," *Proceedings of the International Conference on Computational Methods in Science and Engineering*, Kastoria, Greece, September 12-16, 2003, pp. 350-359.

32. Shabana, A. A., *Computational Dynamics*, Second Edition, John Wiley and Sons, New York, 2001.

33. Garcia de Jalon, J. and Bayo, E*., Kinematic and Dynamic Simulation of Multibody Systems: The Real-Time Challenge*, Springer-Verlag, New York, 1994.

34. Huston, Ronald L., *Multibody Dynamics*, Butterworth-Heinemann, New York, 1990.

35. Reddy, J.N., *An Introduction to the Finite Element Method*, McGraw-Hill, New York, 1993.

36. Thomson, W.T., and Dahleh, M.D., *Theory of Vibration with Applications*, Prentice Hall, New Jersey, 1998.

37. Baruh, H. and Radisavljevic, V., "Modeling of Flexible Mechanisms by Constrained Coordinates," *Journal of the Chinese Society of Mechanical Engineers*, Vol. 21, No. 1, 2000, pp. 1-14.
38. Schwerin, R., *MultiBody System SIMulation: Numerical Methods, Algorithms, and Software*, Springer-Verlag, Berlin, 1994.

39. Junkins, J.L., Akella, M.R., and Kurdilla, A.J., "Adaptive realization of desired constraint stabilization dynamics in the control of multibody systems*," Proceedings of the Royal Society of London Series A: Mathematical, Physical and Engineering Sciences*, Vol. 259, 2001, pp. 2231-2249.

40. Sanyal, A.K., Verma A., and Junkins, J.L., "Adaptation and Cooperation in Control of Multiple Robot Manipulators," *Richard H. Battin Astrodynamics Conference*, College Station, Texas, March 20-21, 2000, Paper No. AAS 00-263.

41. Roache, P.J., *Verification and Validation in Computational Science and Engineering*, Hermosa Publishers, New Mexico, First Edition, 1998.

42. Roache, P.J., "Code Verification by the Method of Manufactured Solutions," *ASME Journal of Fluids Engineering*, Vol. 124, March 2002, pp. 4-10.

43. Salari, K., and Knupp, P., "Code Verification by the Method of Manufactured Solutions," SAND 2000-1444, Sandia National Laboraties, Albuquerque, NM, June 2000.

44. Roy, C.J., Smith, T.M., and Ober, C.C., "Verification of a Compressible CFD Code using the Method of Manufactured Solutions," *32nd AIAA Fluid Dynamics Conference and Exhibit*, St. Louis, Missouri, June 24-26, 2002, AIAA Paper 2002-3110.

45. Lee, S., and Junkins, J.L., "Construction of Benchmark Problems for Solution of Ordinary Differential Equations," *Journal of Shock and Vibration*, Vol. 1, No. 5, 1994, pp. 403-414.

46. Junkins, J.L., and Lee. S., "Validation of Finite-Dimensional Approximate Solutions for Dynamics of Distributed-Parameter Systems," *Journal of Guidance, Control, and Dynamics*, Vol. 18, No. 1, 1995, pp. 87-95.

47. Lee, Sangchul, "Formulation and Validation of Mathematical Models for Hybrid Coordinate Dynamical Systems," Ph.D. dissertation, Texas A&M University, Department of Aerospace Engineering, 1994.

48. Roy, C.J., and Hopkins, M.M., "Discretization Error Estimates Using Exact Solutions to Nearby Problems," *41st Aerospace Sciences Meeting and Exhibit*, January 6-9, 2003, Reno, Nevada, AIAA Paper 2003-629 (invited paper).

49. Crassidis, J.L., and Markley, F.L., "Unscented Filtering for Spacecraft Attitude Estimation," *Journal of Guidance, Control, and Dynamics*, Vol. 26, No. 4, 2003, pp. 536-542.

APPENDIX A[*]

ADDITIONAL NOTES ON OCEA

The following text provides some additional information about OCEA. This information can also be found in Reference 14 pages 12-16 and is reprinted here with permission:

OCEA Algorithm

OCEA methodology is a *transformational process* that changes functional and partial derivative data into new forms during calculations. A single **OCEA** function evaluation generates exact numerical values for the function as well as hidden values for the Jacobian, and higher-order partial derivatives. All of the mathematical library functions require generalized composite function transformations for linking current calculations with all previous sensitivity calculations. Module functions support a mixed object/data type computational environment, where integer, real, double precision, complex, and OCEA data types can co-exist. The partial derivatives are extracted by utility routines as a *post-processing* step. Fortran 90 (**F90**) and Macsyma 2.4 **OCEA** prototype codes have been developed.

[*] Reprinted with permission from "Generalized Gradient Search And Newton's Methods For Multilinear Algebra Root-Solving And Optimization Applications" by James D. Turner, *The John L. Junkins Astrodynamics Symposium*, George Bush Conference Center, College Station, Texas, May 23-24, 2003, AAS 03-261. 2003 by James D. Turner.

The development of an **OCEA toolbox** for engineering and scientific applications is addressing the following seven software issues: Defining how *independent variables* are transformed to **OCEA** form; Developing *derived data types* for vectors, tensors, and embedded variables; Defining *interface operators* for supporting generalized operations; Using *Module functions* to hide **OCEA** computational resources; Defining **OCEA**-enhanced *library routines* that encode chain rule models; Providing utility routines to access the **OCEA** partial derivative calculations; and Developing application suites of software for solving broad classes of mathematical programming problems. Second-order OCEA models are presented for discussing each of these issues.

Data Structures. Each scalar variable is modeled as a compound data object consisting of a concatenation of the original scalar variable and its first and higher order partial derivatives. Compound data objects are created using D*erived data types*. The partial derivative models are not visible to the user during calculations, and can be thought of as hidden *artificial dimensions* for the transformed scalar variables. A significant benefit of employing hidden artificial dimensions is that the structure and computational sequencing of standard algorithms is not impacted.

For example, in the **OCEA** algorithm, the 1x1 scalar g has the following transformed *data* structure[1-3]*:*

$$\overbrace{1x\left(1+m+m^2\right)}^{g} := \begin{bmatrix} \overbrace{1x1}^{g} & \overbrace{mx1}^{\nabla g} & \overbrace{mxm}^{\nabla^2 g} \\ & & \end{bmatrix}$$

where the transformed version of g has dimension $1x(1+m+m^2)$. The new object consists of a concatenation of the variable and two orders of partial derivatives. Generalizations for higher dimensional versions of **OCEA** are obvious. Future enhancements will embed additional computational information for handling data sub-structuring information, sparse structure data, symmetry, and parallel computations.

Operator Overloading. **OCEA**-based **AD** capabilities require four mathematical elements: 1) Generalized intrinsic *binary operators* { +, -, *, **, / }, 2) Generalized *unary functions* { cos(x), sin(x), tan(x), log(x),….}, 3) Encoded multi-level chain rule implementations for all new operators and functions, and 4) Generalized composite function operators. *Derived data types* and *interface operators*[1-3] manage the **OCEA** binary operators and unary functions. Expressing these requirements in **F90**, leads to *50+* Module-hidden routines for redefining the intrinsic and mathematical library functions. Operator overloading facilities manage the definitions for interface operators that allow the compiler to recognize 1) The mathematical operators or functions, and 2) The argument list data types (including user-defined data types) for automatically building links to the hidden routines at compile time.

Four factors impact the efficiency of OCEA partial derivative calculations: 1) partial derivative order, 2) exploitation of sparse structure, 3) exploitation of symmetry, and 4) level of optimization for the OCEA generalized intrinsic and mathematical functions. These topics remain active areas of continuing research. A full exploitation of all of these factors is anticipated to impact the performance of OCEA-based tools by 10-100 fold.

Initializing OCEA Independent Variables. Independent variables are identified for each application and transformed to **OCEA** form. For example, given the following set of independent variables $x_1, x_2, \cdots, x_n$, the **OCEA** form of x$_i$ is given by

$$\underbrace{x_i}_{1+\left(1+n+n^2\right)} := \left[\underbrace{x_i}_{1x1}, \quad \underbrace{\begin{bmatrix} \delta_{1i} \\ \vdots \\ \delta_{ni} \end{bmatrix}}_{nx1}, \quad \underbrace{\begin{bmatrix} 0 & \cdots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \cdots & 0 \end{bmatrix}}_{nxn} \right]$$

where δ_{ij} denotes the standard kronecker delta function and i = 1,…,n. The vector part represents the Jacobian and the matrix part represents the Hessian for x$_i$. The non-vanishing part of the Jacobian is the essential element for enabling the numerical partial derivative calculations. During calculations, the partial derivatives operations are accumulated, and a general **OCEA** variable is defined by:

$$v := \left[v, \quad \begin{bmatrix} x \\ \vdots \\ y \end{bmatrix}, \quad \begin{bmatrix} \alpha & \cdots & \beta \\ \vdots & \ddots & \vdots \\ \beta & \cdots & \gamma \end{bmatrix} \right]$$

where x, y, α, β, γ denote general filling elements.

Accessing Data Stored In The Artificial Dimensions. Structure constructor variables[1-3] (**SCV**) are used to access the artificial dimensions of an **OCEA** compound data object. Assuming that a scalar is defined as $f := \lfloor f, \ \nabla f, \ \nabla^2 f \rfloor$ the individual objects are extracted as

$$f = f\%E, \nabla f = f\%V, \nabla^2 f = f\%T$$

where *%E* denotes the scalar **SCV** , *%V* denotes the vector **SCV** , and *%T* denotes the tensor **SCV** . At a finer level of detail, the individual components are extracted by defining

$$f = f\%E,\ (\nabla f)_i = f\%V\%VPART(i),\ (\nabla f)_{ij} = f\%T\%TPART(i,j)$$

where $(*)_i$ denotes the i^{th} component, $(*)_{ij}$ denotes the i-j^{th} component, *VPART(i)* denotes a vector **SCV,** and *TPART(i,j)* denotes a tensor **SCV** . The computational advantage of this approach is that high-level variable assignments are made for updating vector, matrix, and tensor computations in the intrinsic and mathematical library functions.

Intrinsic Operators and Functions. Module-based operator-overloading methodologies are used to redefine the computers operational rules for processing numerical calculations. Advanced partial derivative capabilities are enabled because multiple levels of the *chain rule* are encoded in each operator and function. Two fourth-order **OCEA** variables are used to define the math models, as follows

$$\begin{aligned} a &:= \left[a,\quad \nabla a,\quad \nabla^2 a,\quad \nabla^3 a,\quad \nabla^4 a\right] \\ b &:= \left[b,\quad \nabla b,\quad \nabla^2 b,\quad \nabla^3 b,\quad \nabla^4 b\right] \end{aligned}$$

Generalizations for the intrinsic mathematical operators and functions are presented for addition, subtraction, multiplication, division, and composite functions.

1. <u>Addition</u>: Adding two variables yields

$$a+b := \left[a+b,\quad \nabla a+\nabla b,\quad \nabla^2 a+\nabla^2 b,\quad \nabla^3 a+\nabla^3 b,\quad \nabla^4 a+\nabla^4 b\right]$$

2. <u>Subtraction</u>: Subtracting two variables yields

$$a - b := \left[a - b, \quad \nabla a - \nabla b, \quad \nabla^2 a - \nabla^2 b, \quad \nabla^3 a - \nabla^3 b, \quad \nabla^4 a - \nabla^4 b \right]$$

3. <u>*Product Rule*</u>**:** Multiplying two variables yields

$$a * b := \lfloor a * b \quad \partial_i (a*b) \quad \partial_j \partial_i (a*b) \quad \partial_k \partial_j \partial_i (a*b) \quad \partial_r \partial_k \partial_j \partial_i (a*b) \rfloor$$

where

$$\begin{aligned}
\partial_i (a*b) &= ab_{,i} + ba_{,i} \\
\partial_j \partial_i (a*b) &= a_{,j} b_{,i} + b_{,j} a_{,i} + ab_{,i,j} + ba_{,i,j} \\
\partial_k \partial_j \partial_i (a*b) &= a_{,j,k} b_{,i} + b_{,j,k} a_{,i} + a_{,k} b_{,i,j} + b_{,k} a_{,i,j} + \\
&\quad a_{,j} b_{,i,k} + b_{,j} a_{,i,k} + ab_{,i,j,k} + ba_{,i,j,k} \\
\partial_r \partial_k \partial_j \partial_i (a*b) &= a_{,j,k,r} b_{,i} + b_{,j,k,r} a_{,i} + a_{,k,r} b_{,i,j} + b_{,k,r} a_{,i,j} + \\
&\quad a_{,j,r} b_{,i,k} + b_{,j,r} a_{,i,k} + a_{,r} b_{,i,j,k} + b_{,r} a_{,i,j,k} + \\
&\quad a_{,j,k} b_{,i,r} + b_{,j,k} a_{,i,r} + a_{,k} b_{,i,j,r} + b_{,k} a_{,i,j,r} + \\
&\quad a_{,j} b_{,i,k,r} + b_{,j} a_{,i,k,r} + ab_{,i,j,k,r} + ba_{,i,j,k,r}
\end{aligned}$$

∂_i denotes the partial derivative w.r.t. the ith variable, i,j,k,r = 1,…,m , and m denotes the number of independent variables. The use of index notation allows the order of the operations to be preserved for the implied tensor contraction operations.

4. *Composite Function Rule:* The composite function transformation evaluates $b = b(a)$, where $b \in \{\sin, \cos, \tan, \exp, \ln, \cosh, \sinh, \tanh, a\sin, etc.\}$. For example, if b = ln(a), then the primed b quantities are defined by

$$[b', \quad b'', \quad b''', \quad b''''] = \left[\frac{1}{a}, \quad \frac{-2}{a^2}, \quad \frac{6}{a^3}, \quad \frac{-24}{a^4} \right].$$

The structure of the a-tensors, however, is independent of the library function being processed and will be exploited in advanced software implementations. The index form of the transformation is given by $b := \lfloor b \quad \partial_i b \quad \partial_j \partial_i b \quad \partial_k \partial_j \partial_i b \quad \partial_r \partial_k \partial_j \partial_i b \rfloor$, where

$$
\begin{aligned}
\partial_i b &= b' a_{,i} \\
\partial_j \partial_i b &= b'' a_{,i} a_{,j} + b' a_{,i,j} \\
\partial_k \partial_j \partial_i b &= b''' a_{,i} a_{,j} a_{,k} + b'' \left(a_{,i,k} a_{,j} + a_{,i} a_{,j,k} + a_{,i,j} a_{,k} \right) + b' a_{,i,j,k} \\
\partial_r \partial_k \partial_j \partial_i b &= b'''' a_{,i} a_{,j} a_{,k} a_{,r} + b''' \left(a_{,i,k} a_{,j} + a_{,i} a_{,j,k} + a_{,i,j} a_{,k} \right) a_{,r} + \\
& b''' \left(a_{,i,r} a_{,j} a_{,k} + a_{,i} a_{,j,r} a_{,k} + a_{,i} a_{,j} a_{,k,r} \right) + \\
& b'' \left(a_{,i,k,r} a_{,j} + a_{,i,r} a_{,j,k} + a_{,i,j,r} a_{,k} + a_{,i,k} a_{,j,r} + a_{,i} a_{,j,k,r} + a_{,i,j} a_{,k,r} \right) + \\
& b'' a_{,i,j,k} a_{,r} + b' a_{,i,j,k,r}
\end{aligned}
$$

$$
\left[b', \quad b'', \quad b''', \quad b'''' \right] = \left[\frac{db}{da}, \quad \frac{d^2 b}{da^2}, \quad \frac{d^3 b}{da^3}, \quad \frac{d^4 b}{da^4} \right]
$$

5. *Division Rule:* A two-step strategy is presented for developing the division rule. The goal is to replace the operation *b/a* with *b*h*, where $h = a^{-1}$. Numerical experiments have demonstrated that the two-stage approach is *~30%* faster than using a direct OCEA division operator. The first step uses the composite function transformation to generate the reciprocal *h*-variable, where

$$
\left[h', \quad h'', \quad h''', \quad h'''' \right] = \left[\frac{-2}{a^2}, \quad \frac{6}{a^3}, \quad \frac{-24}{a^4}, \quad \frac{120}{a^5} \right]
$$

The second step forms the product *b*h* using the product operator, which completes the definition of the division rule.

References

1. L. R. Nyhoff, *Introduction to FORTRAN 90 for Engineers and Scientists*, Prentice Hall, ISBN 0135052157, 1996.

2. W. H. Press, and W.T. Vetterling, *Numerical Recipes in FORTRAN 90*, Vol.2, Cambridge, ISBN 0521574390, 1996.

3. L. P. Meissner, *FORTRAN 90*, PWS Publishers, ISBN 0534933726, 95.

APPENDIX B

GENERALIZED FIRST- THROUGH FOURTH-ORDER SENSITIVITY CALCULATIONS

First order:

$$h_{i,j} = h_{i,s}\Phi_{sj} \tag{B.1}$$

$$h_{i,j} = \frac{\partial \boldsymbol{h}_i(t)}{\partial \boldsymbol{x}_j(t_0)} \tag{B.2}$$

Second order:

$$h_{i,jk} = h_{i,s}\Phi^{(2)}_{sjk} + h_{i,st}\Phi_{tk}\Phi_{sj} \tag{B.3}$$

$$h_{i,jk} = \frac{\partial^2 \boldsymbol{h}_i(t)}{\partial \boldsymbol{x}_j(t_0)\partial \boldsymbol{x}_k(t_0)} \tag{B.4}$$

Third order:

$$\begin{aligned} h_{i,jkl} = {} & h_{i,s}\Phi^{(3)}_{sjkl} + h_{i,su}\Phi_{ul}\Phi^{(2)}_{sjk} + h_{i,st}\Phi_{tk}\Phi^{(2)}_{sjl} \\ & + h_{i,st}\Phi^{(2)}_{tkl}\Phi_{sj} + h_{i,stu}\Phi_{ul}\Phi_{tk}\Phi_{sj} \end{aligned} \tag{B.5}$$

$$h_{i,jkl} = \frac{\partial^3 \boldsymbol{h}_i(t)}{\partial \boldsymbol{x}_j(t_0)\partial \boldsymbol{x}_k(t_0)\partial \boldsymbol{x}_l(t_0)} \tag{B.6}$$

Fourth order:

$$\begin{aligned} h_{i,jklm} = {} & h_{i,s}\Phi^{(4)}_{sjklm} + h_{i,sv}\Phi_{vm}\Phi^{(3)}_{sjkl} \\ & + h_{i,su}\Phi_{ul}\Phi^{(3)}_{sjkm} + h_{i,su}\Phi^{(2)}_{ulm}\Phi^{(2)}_{sjk} + h_{i,suv}\Phi_{vm}\Phi_{ul}\Phi^{(2)}_{sjk} \\ & + h_{i,st}\Phi_{tk}\Phi^{(3)}_{sjlm} + h_{i,st}\Phi^{(2)}_{tkm}\Phi^{(2)}_{sjl} + h_{i,stv}\Phi_{vm}\Phi_{tk}\Phi^{(2)}_{sjl} \\ & + h_{i,st}\Phi^{(2)}_{tkl}\Phi^{(2)}_{sjm} + h_{i,st}\Phi^{(3)}_{tklm}\Phi_{sj} + h_{i,stv}\Phi_{vm}\Phi^{(2)}_{tkl}\Phi_{sj} \\ & + h_{i,stu}\Phi_{ul}\Phi_{tk}\Phi^{(2)}_{sjm} + h_{i,stu}\Phi_{ul}\Phi^{(2)}_{tkm}\Phi_{sj} + h_{i,stu}\Phi^{(2)}_{ulm}\Phi_{tk}\Phi_{sj} \\ & + h_{i,stuv}\Phi_{vm}\Phi_{ul}\Phi_{tk}\Phi_{sj} \end{aligned} \tag{B.7}$$

$$h_{i,jklm} = \frac{\partial^4 \boldsymbol{h}_i(t)}{\partial \boldsymbol{x}_j(t_0)\partial \boldsymbol{x}_k(t_0)\partial \boldsymbol{x}_l(t_0)\partial \boldsymbol{x}_m(t_0)} \tag{B.8}$$

$i = 1, 2, ..., n_m$

$j, k, l, m, s, t, u, v = 1, 2, ..., n_s$

n_m = number of measurements

n_s = number of states

APPENDIX C

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:

$$\dot{\Phi}_{ij} = f_{i,s}\Phi_{sj} \tag{C.1}$$

$$\Phi_{ij} = \Phi_{ij}(t,t_o) = \frac{\partial \boldsymbol{x}_i(t)}{\partial \boldsymbol{x}_j(t_o)} \tag{C.2}$$

Second order:

$$\dot{\Phi}^{(2)}_{ijk} = f_{i,s}\Phi^{(2)}_{sjk} + f_{i,st}\Phi_{tk}\Phi_{sj} \tag{C.3}$$

$$\Phi^{(2)}_{ijk} = \Phi^{(2)}_{ijk}\left(t,t_o\right) = \frac{\partial^2 \boldsymbol{x}_i(t)}{\partial \boldsymbol{x}_j(t_o)\partial \boldsymbol{x}_k(t_o)} \tag{C.4}$$

Third order:

$$\begin{aligned}\dot{\Phi}^{(3)}_{ijkl} &= f_{i,s}\Phi^{(3)}_{sjkl} + f_{i,su}\Phi_{ul}\Phi^{(2)}_{sjk} + f_{i,st}\Phi_{tk}\Phi^{(2)}_{sjl} \\ &+ f_{i,st}\Phi^{(2)}_{tkl}\Phi_{sj} + f_{i,stu}\Phi_{ul}\Phi_{tk}\Phi_{sj}\end{aligned} \tag{C.5}$$

$$\Phi^{(3)}_{ijkl} = \Phi^{(3)}_{ijkl}\left(t,t_o\right) = \frac{\partial^3 \boldsymbol{x}_i(t)}{\partial \boldsymbol{x}_j(t_o)\partial \boldsymbol{x}_k(t_o)\partial \boldsymbol{x}_l(t_o)} \tag{C.6}$$

Fourth order:

$$\begin{aligned}\dot{\Phi}^{(4)}_{ijklm} &= f_{i,s}\Phi^{(4)}_{sjklm} + f_{i,sv}\Phi_{vm}\Phi^{(3)}_{sjkl} \\ &\quad + f_{i,su}\Phi_{ul}\Phi^{(3)}_{sjkm} + f_{i,su}\Phi^{(2)}_{ulm}\Phi^{(2)}_{sjk} + f_{i,suv}\Phi_{vm}\Phi_{ul}\Phi^{(2)}_{sjk} \\ &\quad + f_{i,st}\Phi_{tk}\Phi^{(3)}_{sjlm} + f_{i,st}\Phi^{(2)}_{tkm}\Phi^{(2)}_{sjl} + f_{i,stv}\Phi_{vm}\Phi_{tk}\Phi^{(2)}_{sjl} \\ &\quad + f_{i,st}\Phi^{(2)}_{tkl}\Phi^{(2)}_{sjm} + f_{i,st}\Phi^{(3)}_{tklm}\Phi_{sj} + f_{i,stv}\Phi_{vm}\Phi^{(2)}_{tkl}\Phi_{sj} \\ &\quad + f_{i,stu}\Phi_{ul}\Phi_{tk}\Phi^{(2)}_{sjm} + f_{i,stu}\Phi_{ul}\Phi^{(2)}_{tkm}\Phi_{sj} + f_{i,stu}\Phi^{(2)}_{ulm}\Phi_{tk}\Phi_{sj} \\ &\quad + f_{i,stuv}\Phi_{vm}\Phi_{ul}\Phi_{tk}\Phi_{sj}\end{aligned} \qquad \text{(C.7)}$$

$$\Phi^{(4)}_{ijklm} = \Phi^{(4)}_{ijklm}\left(t,t_o\right) = \frac{\partial^4 \boldsymbol{x}_i(t)}{\partial \boldsymbol{x}_j(t_o)\partial \boldsymbol{x}_k(t_o)\partial \boldsymbol{x}_l(t_o)\partial \boldsymbol{x}_m(t_o)} \qquad \text{(C.8)}$$

APPENDIX D

ALGORITHMS FOR MIDCOURSE CORRECTIONS

The first- through fourth-order algorithms for computing midcourse corrections are given here.

First order:

$$\delta \boldsymbol{v}(t^*) = \left[\frac{\partial \boldsymbol{v}(t^*)}{\partial \boldsymbol{v}(t_f)}\right] \delta \boldsymbol{v}(t_f) \tag{D.1}$$

where

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

Second order:

$$\delta \boldsymbol{v}(t^*) = \left[\frac{\partial \boldsymbol{v}(t^*)}{\partial \boldsymbol{v}(t_f)}\right] \delta \boldsymbol{v}(t_f) + \tfrac{1}{2}\left(\frac{\partial^2 \boldsymbol{v}(t^*)}{\partial \boldsymbol{v}(t_f)^2}\right) \otimes \delta \boldsymbol{v}(t_f) \otimes \delta \boldsymbol{v}(t_f) \tag{D.3}$$

where

$$\begin{aligned} \delta \boldsymbol{v}(t_f) &= \left[\frac{\partial \boldsymbol{r}(t^*)}{\partial \boldsymbol{v}(t_f)}\right]^{-1} \delta \boldsymbol{r}(t^*) \\ &-\tfrac{1}{2}\left[\frac{\partial \boldsymbol{r}(t^*)}{\partial \boldsymbol{v}(t_f)}\right]^{-1} \left\{\left(\frac{\partial^2 \boldsymbol{r}(t^*)}{\partial \boldsymbol{v}(t_f)^2}\right) \otimes \left(\left[\frac{\partial \boldsymbol{r}(t^*)}{\partial \boldsymbol{v}(t_f)}\right]^{-1} \delta \boldsymbol{r}(t^*)\right) \otimes \left(\left[\frac{\partial \boldsymbol{r}(t^*)}{\partial \boldsymbol{v}(t_f)}\right]^{-1} \delta \boldsymbol{r}(t^*)\right)\right\} \end{aligned} \tag{D.4}$$

The third- and fourth-order expressions can be computed from Eqs. (D.5) and (D.6), respectively.

$$\delta \boldsymbol{v}(t^*) = \left[\frac{\partial \boldsymbol{v}(t^*)}{\partial \boldsymbol{v}(t_f)}\right]\delta \boldsymbol{v}(t_f) + \tfrac{1}{2}\left(\frac{\partial^2 \boldsymbol{v}(t^*)}{\partial \boldsymbol{v}(t_f)^2}\right)\otimes \delta \boldsymbol{v}(t_f)\otimes \delta \boldsymbol{v}(t_f) + \tfrac{1}{6}\left(\frac{\partial^3 \boldsymbol{v}(t^*)}{\partial \boldsymbol{v}(t_f)^3}\right)\otimes \delta \boldsymbol{v}(t_f)\otimes \delta \boldsymbol{v}(t_f)\otimes \delta \boldsymbol{v}(t_f) \tag{D.5}$$

$$\delta \boldsymbol{v}(t^*) = \left[\frac{\partial \boldsymbol{v}(t^*)}{\partial \boldsymbol{v}(t_f)}\right]\delta \boldsymbol{v}(t_f) + \tfrac{1}{2}\left(\frac{\partial^2 \boldsymbol{v}(t^*)}{\partial \boldsymbol{v}(t_f)^2}\right)\otimes \delta \boldsymbol{v}(t_f)\otimes \delta \boldsymbol{v}(t_f) + \tfrac{1}{6}\left(\frac{\partial^3 \boldsymbol{v}(t^*)}{\partial \boldsymbol{v}(t_f)^3}\right)\otimes \delta \boldsymbol{v}(t_f)\otimes \delta \boldsymbol{v}(t_f)\otimes \delta \boldsymbol{v}(t_f) + \tfrac{1}{24}\left(\frac{\partial^4 \boldsymbol{v}(t^*)}{\partial \boldsymbol{v}(t_f)^4}\right)\otimes \delta \boldsymbol{v}(t_f)\otimes \delta \boldsymbol{v}(t_f)\otimes \delta \boldsymbol{v}(t_f)\otimes \delta \boldsymbol{v}(t_f) \tag{D.6}$$

where $\delta \boldsymbol{v}(t_f)$ is computed by Eq. (D.7) for the third-order expression in Eq. (D.5) and is computed by Eq. (D.8) for the fourth-order expression in Eq. (D.6).

$$\delta \boldsymbol{v}(t_f) = -\left.\frac{dx}{ds}\right|_{s=1} + \frac{1}{2!}\left.\frac{d^2x}{ds^2}\right|_{s=1} - \frac{1}{3!}\left.\frac{d^3x}{ds^3}\right|_{s=1} \tag{D.7}$$

$$\delta \boldsymbol{v}(t_f) = -\left.\frac{dx}{ds}\right|_{s=1} + \frac{1}{2!}\left.\frac{d^2x}{ds^2}\right|_{s=1} - \frac{1}{3!}\left.\frac{d^3x}{ds^3}\right|_{s=1} + \frac{1}{4!}\left.\frac{d^4x}{ds^4}\right|_{s=1} \tag{D.8}$$

where

$$\left.\frac{dx}{ds}\right|_{g} = -\left[\frac{\partial \boldsymbol{r}(t^*)}{\partial \boldsymbol{v}(t_f)}\right]^{-1}\delta \boldsymbol{r}(t^*) \tag{D.9}$$

$$\left.\frac{d^2x}{ds^2}\right|_{s=1} = -\left[\frac{\partial \boldsymbol{r}(t^*)}{\partial \boldsymbol{v}(t_f)}\right]^{-1}\left\{\left(\frac{\partial^2 \boldsymbol{r}(t^*)}{\partial \boldsymbol{v}(t_f)^2}\right)\otimes \left.\frac{dx}{ds}\right|_{s=1}\otimes \left.\frac{dx}{ds}\right|_{s=1}\right\} \tag{D.10}$$

$$\left.\frac{d^3x}{ds^3}\right|_{s=1} = -\left[\frac{\partial \boldsymbol{r}(t^*)}{\partial \boldsymbol{v}(t_f)}\right]^{-1} \left\{ \begin{array}{l} \left(\frac{\partial^3 \boldsymbol{r}(t^*)}{\partial \boldsymbol{v}(t_f)^3}\right) \otimes \left.\frac{dx}{ds}\right|_{s=1} \otimes \left.\frac{dx}{ds}\right|_{s=1} \otimes \left.\frac{dx}{ds}\right|_{s=1} \\ +2\left(\frac{\partial^2 \boldsymbol{r}(t^*)}{\partial \boldsymbol{v}(t_f)^2}\right) \otimes \left.\frac{d^2x}{ds^2}\right|_{s=1} \otimes \left.\frac{dx}{ds}\right|_{s=1} \\ +\left(\frac{\partial^2 \boldsymbol{r}(t^*)}{\partial \boldsymbol{v}(t_f)^2}\right) \otimes \left.\frac{dx}{ds}\right|_{s=1} \otimes \left.\frac{d^2x}{ds^2}\right|_{s=1} \end{array} \right\} \qquad \text{(D.11)}$$

$$\left.\frac{d^4x}{ds^4}\right|_{s=1} = -\left[\frac{\partial \boldsymbol{r}(t^*)}{\partial \boldsymbol{v}(t_f)}\right]^{-1} \left\{ \begin{array}{l} \left(\frac{\partial^4 \boldsymbol{r}(t^*)}{\partial \boldsymbol{v}(t_f)^4}\right) \otimes \left.\frac{dx}{ds}\right|_{s=1} \otimes \left.\frac{dx}{ds}\right|_{s=1} \otimes \left.\frac{dx}{ds}\right|_{s=1} \otimes \left.\frac{dx}{ds}\right|_{s=1} + \\ 3\left(\frac{\partial^3 \boldsymbol{r}(t^*)}{\partial \boldsymbol{v}(t_f)^3}\right) \otimes \left.\frac{d^2x}{ds^2}\right|_{s=1} \otimes \left.\frac{dx}{ds}\right|_{s=1} \otimes \left.\frac{dx}{ds}\right|_{s=1} + \\ 2\left(\frac{\partial^3 \boldsymbol{r}(t^*)}{\partial \boldsymbol{v}(t_f)^3}\right) \otimes \left.\frac{dx}{ds}\right|_{s=1} \otimes \left.\frac{d^2x}{ds^2}\right|_{s=1} \otimes \left.\frac{dx}{ds}\right|_{s=1} + \\ \left(\frac{\partial^3 \boldsymbol{r}(t^*)}{\partial \boldsymbol{v}(t_f)^3}\right) \otimes \left.\frac{dx}{ds}\right|_{s=1} \otimes \left.\frac{dx}{ds}\right|_{s=1} \otimes \left.\frac{d^2x}{ds^2}\right|_{s=1} \\ +3\left(\frac{\partial^2 \boldsymbol{r}(t^*)}{\partial \boldsymbol{v}(t_f)^2}\right) \otimes \left.\frac{d^3x}{ds^3}\right|_{s=1} \otimes \left.\frac{dx}{ds}\right|_{s=1} + \\ 3\left(\frac{\partial^2 \boldsymbol{r}(t^*)}{\partial \boldsymbol{v}(t_f)^2}\right) \otimes \left.\frac{d^2x}{ds^2}\right|_{s=1} \otimes \left.\frac{d^2x}{ds^2}\right|_{s=1} \\ +\left(\frac{\partial^2 \boldsymbol{r}(t^*)}{\partial \boldsymbol{v}(t_f)^2}\right) \otimes \left.\frac{dx}{ds}\right|_{s=1} \otimes \left.\frac{d^3x}{ds^3}\right|_{s=1} \end{array} \right\} \qquad \text{(D.12)}$$

APPENDIX E

FORTRAN90 NONLINEAR LEAST SQUARES MEASUREMENT MODEL

```
SUBROUTINE NONLINEAR_FX( T, EB_VAR, EB_FCTN )
!       THIS PROGRAM EVALUATES A VECTOR FUNCTION USING EMBEDDED
!        PROCESSING.
!       THE USER INPUTS A VECTOR OF OCEA-INITIALIZED INDEPENDENT
VARIABLES
!       AND EVALUATES A VECTOR FUNCTION.
!
! INPUT:
!EB_VAR:      NVx1 VECTOR OF OCEA-INITIALIZED INDEPENDENT VARIABLES
! OUTPUT:
!EB_FCTN:     NFx1 VECTOR OF OCEA-EVALUATED NONLINEAR FUNCTIONS
!       = [ F, DEL(F), DEL^2(F) ] = [function, gradient, hessian]


        USE EB_HANDLING
        IMPLICIT NONE
! ARGUMENT LIST VARIABLES
        REAL(DP)::T
        TYPE(EB),     DIMENSION(NV), INTENT(IN   ):: EB_VAR
        TYPE(EB),     DIMENSION(NF), INTENT(INOUT):: EB_FCTN
! DEFINE LOCAL + EMBEDDED VARIABLES
        REAL(DP),     DIMENSION(NF):: FX, DELX
        REAL(DP),     DIMENSION(NF,NV):: JAC
        REAL(DP),     DIMENSION(NF,NV,NV):: HES
        REAL(DP),     DIMENSION(NV,NV):: A
        TYPE(EB)::    K1, K2, K3, K4, K5, LAM1, LAM2, LAM3
        TYPE(EB)::    OMEG1, OMEG2, OMEG3, DEL1, DEL2, DEL3

! ASSIGN LOCAL VARIABLES
        K1=EB_VAR(1);K2=EB_VAR(2);K3=EB_VAR(3);K4=EB_VAR(4);K5=EB_VAR(5)
        LAM1=EB_VAR(6);LAM2=EB_VAR(7);LAM3=EB_VAR(8)
        OMEG1=EB_VAR(9);OMEG2=EB_VAR(10);OMEG3=EB_VAR(11)
        DEL1=EB_VAR(12);DEL2=EB_VAR(13);DEL3=EB_VAR(14)

! COMPUTE NONLINEAR FUNCTION USING EMBEDDED ALGEBRA

        EB_FCTN(1) = K1*EXP(LAM1*T)*COS(OMEG1*T+DEL1) + K2*EXP(LAM2*T)*&
        COS(OMEG2*T+DEL2) + K3*EXP(LAM3*T)*COS(OMEG3*T+DEL3) + K4

        EB_FCTN(2) = K1*EXP(LAM1*T)*SIN(OMEG1*T+DEL1) + K2*EXP(LAM2*T)*&
        SIN(OMEG2*T+DEL2) + K3*EXP(LAM3*T)*SIN(OMEG3*T+DEL3) + K5

END SUBROUTINE NONLINEAR_FX
```

APPENDIX F

MASS MATRIX TIME DERIVATIVE PROOF

Here, the following identity is proved, which allows for computation of the mass matrix time derivative by second order differentiation.

$$\dot{m}_{ij} = \frac{\partial^2 T}{\partial \dot{q}_i \partial q_j} = \frac{\partial^3 T}{\partial \dot{q}_i \partial \dot{q}_j \partial q_k} \dot{q}_k \tag{F.1}$$

We know that

$$\frac{\partial T}{\partial \dot{q}_i} = m_{ij} \dot{q}_j \tag{F.2}$$

where

$$m_{ij} = \frac{\partial^2 T}{\partial \dot{q}_i \partial \dot{q}_j}$$

therefore from (F.2) we have

$$\begin{aligned} \frac{d}{dt}\left(\frac{\partial T}{\partial \dot{q}_i}\right) &= m_{ij}\ddot{q}_j + \dot{m}_{ij}\dot{q}_j \\ &= m_{ij}\ddot{q}_j + \frac{\partial m_{ij}}{\partial q_k}\dot{q}_k \dot{q}_j \\ &= m_{ij}\ddot{q}_j + \frac{\partial^3 T}{\partial \dot{q}_i \partial \dot{q}_j \partial q_k}\dot{q}_k \dot{q}_j \end{aligned} \tag{F.3}$$

However, from a purely mathematical point of view considering $T = T(\boldsymbol{q}, \dot{\boldsymbol{q}})$ we have

$$\begin{aligned} \frac{d}{dt}\left(\frac{\partial T}{\partial \dot{q}_i}\right) &= \frac{\partial^2 T}{\partial \dot{q}_i \partial \dot{q}_j}\ddot{q}_j + \frac{\partial^2 T}{\partial \dot{q}_i \partial q_k}\dot{q}_k \\ &= m_{ij}\ddot{q}_j + \frac{\partial^2 T}{\partial \dot{q}_i \partial q_k}\dot{q}_k \end{aligned} \tag{F.4}$$

By equating (F.3) and (F.4) we have (F.5) where the expression in parentheses must be zero thus showing (F.1) is true.

$$\frac{\partial^2 T}{\partial \dot{q}_i \partial q_k}\dot{q}_k = \frac{\partial^3 T}{\partial \dot{q}_i \partial \dot{q}_j \partial q_k}\dot{q}_k \dot{q}_j \Rightarrow \left(\frac{\partial^2 T}{\partial \dot{q}_i \partial q_k} - \frac{\partial^3 T}{\partial \dot{q}_i \partial \dot{q}_j \partial q_k}\dot{q}_j \right)\dot{q}_k = 0 \qquad \text{(F.5)}$$

APPENDIX G

TIME DERIVATIVES OF EQUATIONS OF MOTION

Equations of motion:

$$\dot{x}_i = f_i(\boldsymbol{x}(t), t) \tag{G.1}$$

First order:

$$\dot{f}_i = f_{i,s}\dot{x}_s \tag{G.2}$$

Second order:

$$\ddot{f}_i = f_{i,st}\dot{x}_t\dot{x}_s + f_{i,s}\ddot{x}_s \tag{G.3}$$

Third order:

$$\dddot{f}_i = f_{i,stu}\dot{x}_u\dot{x}_t\dot{x}_s + f_{i,st}\ddot{x}_t\dot{x}_s + 2f_{i,st}\dot{x}_t\ddot{x}_s + f_{i,s}\dddot{x}_s \tag{G.4}$$

Fourth order:

$$\begin{aligned}\ddddot{f}_i = {} & f_{i,stuv}\dot{x}_v\dot{x}_u\dot{x}_t\dot{x}_s + f_{i,stu}\ddot{x}_u\dot{x}_t\dot{x}_s + f_{i,stu}\dot{x}_u\ddot{x}_t\dot{x}_s + f_{i,stu}\dot{x}_u\dot{x}_t\ddot{x}_s \\ & + f_{i,stu}\dot{x}_u\ddot{x}_t\dot{x}_s + f_{i,st}\dddot{x}_t\dot{x}_s + f_{i,st}\ddot{x}_t\ddot{x}_s \\ & + 2(f_{i,stu}\dot{x}_u\dot{x}_t\ddot{x}_s + f_{i,st}\ddot{x}_t\ddot{x}_s + f_{i,st}\dot{x}_t\dddot{x}_s) \\ & + f_{i,st}\dot{x}_t\dddot{x}_s + f_{i,s}\ddddot{x}_s\end{aligned} \tag{G.5}$$

Note: All partials taken with respect to x. Overdot indicates time derivative.

APPENDIX H

FORTRAN 90 SUBROUTINE FOR SPRING PENDULUM EQUATIONS OF MOTION

```
SUBROUTINE SPRING_PEND_EQNS( PASS, TIME, X0, DXDT, FLAG )
      USE EB_HANDLING
      IMPLICIT NONE
!.....ARGUMENT LIST VARIABLES
      **********
      REAL(DP),           INTENT(IN)::   TIME
      TYPE(EB),DIMENSION(NV),INTENT(IN   ):: X0
      TYPE(EB),DIMENSION(NV), INTENT(INOUT):: DXDT

!.....LOCAL VARIABLES
      TYPE(EB)::L, T, V        ! LAGRANGIAN, KINETIC, POTENTIAL
      REAL(DP):: M, K          ! MASS AND STIFFNESS VALUES
      REAL(DP), DIMENSION(NV):: JAC_L
      REAL(DP), DIMENSION(NV,NV):: HES_L
      REAL(DP), DIMENSION(NV/2,NV/2):: MASS, MASS_INVERSE, MASSDOT
      REAL(DP), DIMENSION(NV/2)::  JAC_L_Q, QDOTDOT, QDOT
      TYPE(EB):: R0! UNSTRETCHED SPRING LENGTH
      REAL(DP):: GRAV

      M  = 1.0D0;  K  = 75.0D0 ! MASS AND STIFFNESS
      ***********
      T = 0.5D0*M*(X0(3)**2 + X0(1)**2*X0(4)**2)                 ! DEFINE KE
      V = 0.5D0*K*(X0(1)-R0)**2 + M*GRAV*(R0-X0(1)*COS(X0(2)))  ! DEFINE PE
      L = T – V ! DEFINE LAGRANGIAN FUNCTION

      JAC_L = L       ! EXTRACT JACOBIAN OF LAGRANGIAN
      JAC_L_Q = JAC_L(1:NV/2)    ! EXTRACT PARTIALS W.R.T. GEN. COORDS.

      HES_L = L       ! EXTRACT SECOND ORDER  PARTIALS OF LAGRANGIAN

      MASS      = HES_L(NV/2+1:NV,NV/2+1:NV) ! COMPUTE MASS MATRIX
      MASSDOT = HES_L(NV/2+1:NV,1:NV/2)            !COMPUTE MDOT
      **********

      QDOTDOT = MATMUL(MASS_INVERSE,(JAC_L_Q - (MATMUL(MASSDOT,QDOT))))

      DXDT(1)%E = X0(3)%E                     ! RDOT
      DXDT(2)%E = X0(4)%E                     ! THETADOT
      DXDT(3)%E = QDOTDOT(1)                  ! RDOTDOT
      DXDT(4)%E = QDOTDOT(2)                  ! THETADOTDOT

END SUBROUTINE SPRING_PEND_EQNS
```

Note: $X0(1) = r;\quad X0(2) = \theta;\quad X0(3) = \dot{r};\quad X0(4) = \dot{\theta}$

APPENDIX I

RANGE SPACE FORMULATION

Here, we derive the solution for the Lagrange multiplier. As is shown, the Range Space method, as it is frequently called, goes about the constraint "elimination" process by explicitly solving for the multipliers and ultimately the constraint force.

Consider a holonomic constraint of the form (assuming no explicit time dependence for simplicity):

$$\phi(\boldsymbol{q}) = \boldsymbol{0} \tag{I.1}$$

where the Pfaffian form of this constraint is developed by time differentiating Eq. (I.1).

$$\begin{aligned} \dot{\phi}(\boldsymbol{q}) &= \frac{\partial \phi}{\partial \boldsymbol{q}} \dot{\boldsymbol{q}} = 0 \\ &= C\dot{\boldsymbol{q}} = 0 \end{aligned} \tag{I.2}$$

we now time differentiate Eq. (I.2) to obtain

$$\ddot{\phi}(\boldsymbol{q}) = C\ddot{\boldsymbol{q}} + \dot{C}\dot{\boldsymbol{q}} = 0 \tag{I.3}$$

We now look at equations of motion of the following form:

$$\ddot{\boldsymbol{q}} = M^{-1}\left(-\dot{M}\dot{\boldsymbol{q}} + \frac{\partial L}{\partial \boldsymbol{q}} + \boldsymbol{Q} + C^T \boldsymbol{\lambda} \right) \tag{I.4}$$

and multiply Eq. (I.4) through by C and then equate with Eq. (I.3) to obtain:

$$\begin{aligned} C\ddot{\boldsymbol{q}} &= CM^{-1}\left(-\dot{M}\dot{\boldsymbol{q}} + \frac{\partial L}{\partial \boldsymbol{q}} + \boldsymbol{Q} + C^T \boldsymbol{\lambda} \right) \\ &= -\dot{C}\dot{\boldsymbol{q}} \end{aligned} \tag{I.5}$$

After equating the terms on the right hand side of Eq. (I.5), we can solve for the multipliers:

$$\boldsymbol{\lambda} = \left(CM^{-1}C^{T} \right)^{-1} \left\{ -CM^{-1} \left(Q + \frac{\partial L}{\partial \boldsymbol{q}} - \dot{M}\dot{\boldsymbol{q}} \right) - \dot{C}\dot{\boldsymbol{q}} \right\} \tag{I.6}$$

The constraint force is computed by $C^{T}\boldsymbol{\lambda}$ at each time step in order to integrate Eq. (I.4).

APPENDIX J

TRANSLATIONAL TERMS IN KINETIC ENERGY EXPRESSIONS FOR FLEXIBLE MULTIBODY SYSTEMS

We add the following term to kinetic energy expression in Eq. (4.39) in order to generalize the approach for translational motion. For the first link of the assumed modes model for a flexible link system we add

$$\begin{aligned}T_1^{translation} &= \tfrac{1}{2} m_1 \left(\dot{x}_A^2 + \dot{y}_A^2 \right) - \tfrac{1}{2} m_1 L_1 \dot{x}_A \dot{\theta}_1 \sin\theta_1 + \tfrac{1}{2} m_1 L_1 \dot{y}_A \dot{\theta}_1 \cos\theta_1 \\ &\quad - \dot{x}_A \dot{\theta}_1 \cos\theta_1 \boldsymbol{q}_1^T \boldsymbol{b}_1 - \dot{x}_A \sin\theta_1 \dot{\boldsymbol{q}}_1^T \boldsymbol{b}_1 \\ &\quad - \dot{y}_A \dot{\theta}_1 \sin\theta_1 \boldsymbol{q}_1^T \boldsymbol{b}_1 + \dot{y}_A \cos\theta_1 \dot{\boldsymbol{q}}_1^T \boldsymbol{b}_1\end{aligned} \tag{J.1}$$

For successive links, we add the following term:

$$\begin{aligned}T_{p+1}^{translation} &= \tfrac{1}{2} m_{p+1} \left(\dot{x}_A^2 + \dot{y}_A^2 \right) - \tfrac{1}{2} m_{p+1} L_{p+1} \dot{x}_A \dot{\theta}_{p+1} \sin\theta_{p+1} + \tfrac{1}{2} m_{p+1} L_{p+1} \dot{y}_A \dot{\theta}_{p+1} \cos\theta_{p+1} \\ &\quad - m_{p+1} \dot{x}_A \sum_{i=1}^{p} L_i \dot{\theta}_i \sin\theta_i + m_{p+1} \dot{y}_A \sum_{i=1}^{p} L_i \dot{\theta}_i \cos\theta_i \\ &\quad - \dot{x}_A \dot{\theta}_{p+1} \cos\theta_{p+1} \boldsymbol{q}_{p+1}^T \boldsymbol{b}_{p+1} - \dot{x}_A \sin\theta_{p+1} \dot{\boldsymbol{q}}_{p+1}^T \boldsymbol{b}_{p+1} \\ &\quad - \dot{y}_A \dot{\theta}_{p+1} \sin\theta_{p+1} \boldsymbol{q}_{p+1}^T \boldsymbol{b}_{p+1} + \dot{y}_A \cos\theta_{p+1} \dot{\boldsymbol{q}}_{p+1}^T \boldsymbol{b}_{p+1}\end{aligned} \tag{J.2}$$

VITA

Daniel Todd Griffith was born on September 8, 1974 in Hazard, Kentucky to the proud parents Dan and Barbara Griffith. He is oldest child of four. Todd attended high school at Breathitt County High School in Jackson, Kentucky and graduated with valedictorian honors in 1993. He completed undergraduate degrees in Physics and Mechanical Engineering from Morehead State University in Morehead, Kentucky and from the University of Kentucky in Lexington, Kentucky with honors in 1998. He promptly began graduate studies in Mechanical Engineering at the University of Kentucky where he completed his Master's work in May 2000. In the spirit of the great frontiersmen from years past, Todd then decided to go west to do his doctoral work in Aerospace Engineering at Texas A&M University in College Station, Texas with one of the nation's most respected "gunslingers", Dr. John Junkins. There he learned to operate many weapons in the arsenal of dynamics and controls en route to graduating in December 2004.

In the summer of 2003, Todd met his true love Loraine Marie Fano while on a pilgrimage trip to Rome, Florence, and Assisi, Italy. Todd and Loraine will begin their life together as husband and wife the day after he graduates with his doctoral degree, December 18, 2004. Todd and Loraine will reside together at a location which is to be determined. They can be reached in the meantime at the address of his parents: 297 Hurricane Branch Road; Jackson, KY 41339.