INTEGRATED FORMULATION-SOLUTION-DESIGN SCHEME FOR NONLINEAR MULTIDISCIPLINARY SYSTEMS USING THE MIXEDMODELS PLATFORM

by

SHILPA ARUN VAZE

B.S., University of Pune, 1996
M.S., Kansas State University, 2002

AN ABSTRACT OF A DISSERTATION

submitted in partial fulfillment of the requirements for the degree

DOCTOR OF PHILOSOPHY

Department of Electrical and Computer Engineering
College of Engineering

KANSAS STATE UNIVERSITY
Manhattan, Kansas

2007

Abstract

Most state-of-the-art systems are multidisciplinary in nature and encompass a wide range of components from domains such as electronics, mechanics, hydraulics, etc. Design considerations and design parameters of the system can come from any or a combination of these domains. The traditional optimization approach for multidisciplinary systems utilizes sequential optimization, wherein each subsystem is optimized in isolation in a predetermined order, assuming that the designs of the other subsystems remain fixed. This often leads to system designs that are suboptimal. In recent years emphasis has been placed on development of an integrated scheme for analysis and design of multidisciplinary systems. An important aspect is the software architecture required to support such a scheme.

This dissertation presents MIXEDMODELS (Multidisciplinary Integrated eXtensible Engine for Driving Metamodeling, Optimization and DEsign of Large-scale Systems) - a unified analysis and design tool for multidisciplinary systems that is based on a procedural, symbolic-numeric architecture. This architecture offers great modeling flexibility at the component level, allowing any engineer to add components in his/her domain of expertise to the platform in a modular fashion. The symbolic engine in the MIXEDMODELS platform synthesizes the system governing equations as a unified set of nonlinear differential-algebraic equations (DAEs). These equations are differentiated with respect to design variables to obtain an additional set of DAEs that describe the sensitivity coefficients of the system state variables. This combined set of DAEs is solved numerically to obtain the solution for the state variables and the state sensitivity coefficients of the system. Finally, knowing the system performance functions, their design sensitivity coefficients can be calculated by using the values of the state variables and state sensitivity coefficients obtained from the DAEs. For ease in error control and software implementation, sensitivity analysis formulation described in this work uses direct differentiation approach as opposed to the adjoint variable approach.

The MIXEDMODELS capabilities are demonstrated through several numerical examples and the results indicate that the MIXEDMODELS formulation and architecture is effective in terms of accuracy, modeling convenience, computational efficiency, and the ability to simulate the behavior of a general class of multidisciplinary systems.

INTEGRATED FORMULATION-SOLUTION-DESIGN SCHEME FOR NONLINEAR MULTIDISCIPLINARY SYSTEMS USING THE MIXEDMODELS PLATFORM

by

SHILPA ARUN VAZE

B.S., University of Pune, 1996
M.S., Kansas State University, 2002

A DISSERTATION

submitted in partial fulfillment of the requirements for the degree

DOCTOR OF PHILOSOPHY

Department of Electrical and Computer Engineering
College of Engineering

KANSAS STATE UNIVERSITY
Manhattan, Kansas

2007

Approved by:

Co-Major Professor
Prof. James E. DeVault
(Electrical and Computer Engineering Department)

Co-Major Professor
Dr. Prakash Krishnaswami
(Mechanical and Nuclear Engineering Department)

Copyright

Abstract

Most state-of-the-art systems are multidisciplinary in nature and encompass a wide range of components from domains such as electronics, mechanics, hydraulics, etc. Design considerations and design parameters of the system can come from any or a combination of these domains. The traditional optimization approach for multidisciplinary systems utilizes sequential optimization, wherein each subsystem is optimized in isolation in a predetermined order, assuming that the designs of the other subsystems remain fixed. This often leads to system designs that are suboptimal. In recent years emphasis has been placed on development of an integrated scheme for analysis and design of multidisciplinary systems (MDSs). An important aspect is the software architecture required to support such a scheme.

This dissertation presents MIXEDMODELS (Multidisciplinary Integrated eXtensible Engine for Driving Metamodeling, Optimization and DEsign of Large-scale Systems) - a unified analysis and design tool for MDSs that is based on a procedural, symbolic-numeric architecture. This architecture offers great modeling flexibility at the component level, allowing any engineer to add components in his/her domain of expertise to the platform in a modular fashion. The symbolic engine in the MIXEDMODELS platform synthesizes the system governing equations as a unified set of nonlinear differential-algebraic equations (DAEs). These equations are differentiated with respect to design variables to obtain an additional set of DAEs that describe the sensitivity coefficients of the system state variables. This combined set of DAEs is solved numerically to obtain the solution for the state variables and the state sensitivity coefficients of the system. Finally, knowing the system performance functions, their design sensitivity coefficients can be calculated by using the values of the state variables and state sensitivity coefficients obtained from the DAEs. For ease in error control and software implementation, sensitivity analysis formulation described in this work uses direct differentiation approach as opposed to the adjoint variable approach.

The MIXEDMODELS capabilities are demonstrated through several numerical examples and the results indicate that the MIXEDMODELS formulation and architecture is effective in terms of accuracy, modeling convenience, computational efficiency, and the ability to simulate the behavior of a general class of multidisciplinary systems.

Table of Contents

List of Figures

List of Tables

Acknowledgement

My time in Manhattan, KS (the recent 9 years of my life) has been a truly exceptional, untraditional and quite an amazing journey – one that I had never dreamt of. These nine years certainly added beautiful colors to my wealth of experiences and I am grateful to all who have made this journey so pleasant and enjoyable.

My dissertation would not have been possible without my co-major advisor Dr. Prakash Krishnaswami. Like every other Ph.D. student, I had my moments where it would seem completely impossible to get through, and it is because of Dr. Prakash that I pulled through every time. I truly appreciate his immense patience and tolerance during all these years, especially in reviewing all the technical papers I wrote and presented during my Ph.D. He has this unique ability of identifying strengths and weaknesses to bring out the best in every student, and he worked with me patiently bringing me up to speed in my understanding of research, programming techniques and technical writing. He walked with me throughout my academic career at K-State. His policy - "work hard and play hard" and his passion about research, teaching and in general towards life helped me learn a lot from him. He also encouraged me to participate in a variety of non-academic activities in Manhattan, KS, which in turn made my research career enjoyable. He is one of a kind, and a rare to find advisor. I am glad I ran into him at K-State and I am truly grateful to him for agreeing to be my advisor for my Ph.D.

Prof. James E. DeVault, also my co-major advisor, is simply an awesome teacher and a great advisor. I am deeply thankful to him for his confidence and faith in me during these five years of my Ph.D. I truly appreciate his patience and kindness towards me, especially during the time when I was going through a rough patch in my life. His organizational skills and passion to keep everything neat and tidy are admirable. That certainly helped me in getting my research organized, and also to work more efficiently. His office was like a mini-library to me, and he always let me borrow any book that I needed. Sometimes he would observe me teaching in a

lecture hall or a lab and his comments and suggestions always encouraged me and inspired me to consistently improve my teaching skills.

I would like to thank Dr. Steve Warren and Dr. John Devore for serving on my supervisory committee. I would also like to thank them both for their guidance and the valuable discussions we had during my doctoral program. I truly appreciate their time and patience in carefully reading through my dissertation and providing suggestions to improve the quality of writing. My special thanks to Dr. Michael O'Shea for serving as an outside chairperson. I would also like to express my sincere gratitude to Dr. Anil Pahwa for agreeing to serve on my committee on a very short time request and at a very crucial time in my Ph.D. I must also thank Dr. Carol Shanklin and Ms. Angie Pfizenmaier for their co-operation and guidance in completing the paperwork needed for the graduate school and working with me during some unavoidable circumstances.

I would like to acknowledge IEEE, ASME, ECCOMAS and WCCM for publishing my work and inspiring me to continue working on my research.

I am grateful to Prof. DeVault, Dr. Krishnaswami, Dr. Pahwa, Dr. Devore, Dr. Stanton, Dr. Soldan, Ms. Muguira and Dr. Gruenbacher for providing continued financial support through teaching and research assistantships during my years in Electrical Engineering Department at K-State. I enjoyed discussing my research with Dr. Lewis, Dr. Stanton, Dr. Morcos, and Dr. White, and I am thankful to them for their time and interest in my research, and also for lending me their books. I would like to take this opportunity to thank all the faculty members from my department and the K-State staff, especially Ms. Tammie Hartwick, Ms. Jayna Wieters, Ms. Sharon Hartwich, Mr. Joe Beck, Mr. Steve Booth, Mr. Sam Hays, Ms. Jacqueline Tweed and Mrs. Asha Muthukrishnan for helping out every time I needed anything, may it be a book from the library, a key to some lab or a tiny resistor from the electronics shop.

My friends have always been an important part of my life. My special thanks to my very close friend Satish Motipalli without whom it would have been difficult to go through these five years of my Ph.D. Satish is one of those few friends I can count on

for any help at any time. We have had several things in common during our Ph.D. program at K-State, and I am glad that he was always around to talk or sometimes merely to provide a patient ear. I would also like to thank Dr. Kelkar, Muktadidi, Samir, Mrinal, Indira, Dr. Das, Leenadi, Sunitha, Shama, Mandar, Jeet, Seemanti and Pradeep for all the encouragement and good times that we shared together. Dr. Kelkar was my co-major advisor for my master's program in Mechanical Engineering at K-State and has been a constant source of inspiration to me. Indira's and Mukta didi's thoughtfulness, kindness, and help, especially during the last few critical months of my Ph.D., made it much easier for me to manage my time and priorities properly. Many thanks to Leenadi for reviewing my dissertation on my last minute request and yet taking the time to go through each word providing a detailed analysis of my writing. Many thanks to Xinye, Dapeng and Praveen, my officemates, who gave me company working late hours in the department and helped in keeping our office atmosphere conducive and motivating. In the end I must say that this acknowledgement is incomplete without the mention of my very special friends, Sankaran and Divya, I have been very fortunate to have such friends who cherish our friendship despite my eccentricities and weirdness, and who have always been there for me in my good and bad times.

Finally, it would have been impossible without my family – the Vazes, the Joshis, the Krishnaswamis and the Shah-Gokharu family. Thank you all for everything - your love, support, encouragement, patience, and most importantly for being there for me always - this dissertation is dedicated to you all.

To all of you, thank you very much for this once-in-a-lifetime experience.

Dedication

I dedicate this dissertation to my family, without whom I could not have accomplished this. It is a product of Dr. Prakash's vision and a collaborative effort from the Vazes, the Joshis, the Krishnaswamis and the Gokharu-Shah family. I am very fortunate to have been blessed by these wonderful people. To me they represent an unlimited source of love, support, encouragement and wisdom.

My parents, Arun Vaze and Mrudul Vaze, always supported me through my Ph.D. and gave me the freedom to make life decisions. I truly appreciate their patience and tolerance to my erratic behavior during these years. Their confidence and faith in me meant a lot to me, and it gave me strength to keep myself motivated through my academic career. My sister, Anuradha Joshi, has always been a great source of inspiration to me, her energy and attitude of making the best of everything are truly amazing. She and her husband, Yogesh, provide a different perspective towards life, and I have always benefited from that.

My father, Dr. Prakash and Seema Gokharu are the three people whose love, care and words of wisdom have managed to keep me sane through all these years. They believed in me and taught me to believe in myself. They taught me to be there for others, and take responsibility for my actions. Dr. Prakash was always there for me during my numerous mishaps and wrong decisions, academic as well as personal, and discussions with him on several issues were always useful in my professional as well as personal development. On a lighter note, I would not have done some of the craziest things, like skydiving, if Seema had not jumped out of that plane with me. Many such things that Seema and I did together bring a lot of joy in my life. Seema gives me the courage and strength to face any kind of situation, and her ability to be positive, be there for others, let go of others' eccentricities and appreciate people for their goodness keeps me motivated and inspires me to be a better person. Dr. Prakash's "worst case analysis" tool always works, and then in the end a "never-imagined situation" seems pretty normal to deal with.

I enjoyed badminton and racquet-ball games and the long walks with Dr. Prakash. Those were a lot of fun. Most of the research ideas originated and developed during these walks and games; that is probably why at times I hit some good smashes in badminton.

Sujatha Prakash, an individual full of energy, love and compassion, a unique personality, without whom I would not have survived some storms. Her ability of multitasking, and being really good at whatever she does is just amazing. Jyoti Shah, a wonderful person and my partner in badminton is a lot of fun to be with. He always encouraged me and inspired me to perform well in my academic career. With Sujatha, Dr. Prakash, Seema and Jyoti being around, I always had a home to crash into - any day, any time. And there always used to be home-cooked delicious food, thanks to the two terrific cooks of Manhattan - Sujatha and Jyoti.

And finally, Dr. Prakash's parents (Amma-Appa) – Padma Krishnaswami, and C.S. Krishnaswami and the kids - Soumil, Ketaki, Abhishek, Siddharth and Azad - full of energy and fun. They brought a lot of joy in my life and helped me relax when the work pressure used to be just too much to handle.

This dissertation is dedicated to you all, thank you for being there, and making this journey so pleasant, bright and colourful.

CHAPTER 1 - INTRODUCTION

1.1 Multidisciplinary Systems – What are These and Why Are They Important?

During the late 1960s, the term "*mechatronic systems*" was coined by the Yaskawa Electric Company where applications were mainly limited to servo technology [1.8.1]. At that time, servo systems were mainly used to improve performance of simple electromechanical systems such as automatic doors, vending machines, simple vehicle speed controls, etc. The introduction of computing technology and microprocessors brought about a revolutionary change in the automobile industry, robotic engineering, aviation and space research, nuclear engineering, the chemical and process industry, and many other engineering disciplines. Extensive research in the fields of optics and VLSI design fostered multidisciplinary branches such as 'opto-mechatronics' and 'micro-mechatronics'. Furthermore, advancement in sensor technology and communication networks opened up additional engineering applications. As a result, the systems that were originally purely mechanical or pneumatic with limited capabilities were expanded in functionality and performance by the addition of electrical/electronic components, sensors, actuators and control components; thus the system dynamics could no longer be described in a single domain. This development in technology lead to a separate class of systems called Multidisciplinary Systems (MDSs). With this new advancement in technology, it became important to develop modeling and simulation techniques for multidisciplinary applications.

Another branch of engineering which focuses on this research is multibody dynamics [1.8.2]. In recent years emphasis has also been placed on development of an integrated approach for analysis and design of multidisciplinary systems. Traditionally, optimization approaches for multidisciplinary systems utilize sequential optimization, wherein each subsystem – electrical, mechanical, hydraulic, etc. – is optimized in isolation in a predetermined order, assuming that the designs of the other systems remain fixed. This often leads to system designs that are

suboptimal [1.8.3-1.8.5]. This is mainly due to the extensive interaction between domains such as electrical-electronic, mechanical, hydraulic, pneumatic etc. that the sequential design approach ignores by decoupling the subsystems. For example, in many multidisciplinary systems, design variables of an electrical subsystem may strongly affect the performance of the mechanical subsystem; hence the sensitivities of performance functions from the mechanical domain need to be calculated with respect to the design variables in the electrical domain in order to optimize the design of the entire system. This kind of interaction between the subsystems cannot be captured in a sequential design approach. Further, it has been shown in [1.8.3-1.8.5] that integrated optimization techniques for multidisciplinary systems result in better designs in terms of robustness and optimality.

Due to the nature of multidisciplinary systems, multi-sensor fusion is also in demand and is gaining popularity in academia and research industry. With the same argument that justifies the need for concurrent optimization over sequential optimization, for optimal design it is important that sensor fusion should also be modeled as an integral part of the system. Currently sensor fusion is applied to problems in an ad-hoc way, and the research focus is being shifted towards developing methods to model sensor-fusion strategies as an integrated part of the system [1.8.6]. Current formulations do not easily allow the designer to explore other sensor fusion approaches to enhance the system's performance, nor do they provide flexibility to add features like fault detection and isolation.

The purpose of this research, therefore, is to develop a unified analysis and design approach that would allow modeling, simulation and design of multidisciplinary systems with the capability of modeling sensor fusion strategies as an integrated part of the system. Along with mathematical formulation to model multidisciplinary systems, this work also presents a procedural, symbolic-numeric software architecture that supports the mathematical formulation. The architecture offers great modeling flexibility at the component level to accommodate component models at different levels of complexity, and facilitate extension to new problem domains.

1.1.1 Original Primary Research Problem Statement

Our original primary research objective was stated as the "Development of Concurrent Optimization Techniques to Enhance Performance of Multidisciplinary Systems with Sensor Fusion". However to solve the primary research problem, the main requirement was the availability of a unified platform for modeling, simulation and design of multidisciplinary systems. The following section provides a discussion on the current state of the art in modeling and simulation techniques for multidisciplinary systems, which leads us to redefine the primary research problem as given in section 1.5 below.

1.2 Literature Review - Modeling and Simulation of Multidisciplinary Systems

Dynamic modeling and simulation of multidisciplinary systems has been an open research topic for more than three decades. There are several analysis and design tools currently available on the market [1.8.7-1.8.9]. Generally, these approaches can be partitioned into three basic categories.

- Approaches based on integration at the formulation level
- Approached based on integration at the solution tools level
- Approaches based on integration at the equation level

1.2.1 Approaches Based on Integration at the Formulation Level

One of the approaches existing today for modeling MDS is to apply a single physical/mathematical formulation such as Bond Graph Theory [1.8.7-1.8.11], Linear Graph Theory [1.8.7, 1.8.8], Lagrangian Formulation [1.8.12-1.8.14], Multibody Constrained Formulation [1.8.6] and State Space Formulation [1.8.15] etc. to derive governing equations of an MDS. The following paragraphs will briefly discuss each of these formulations to investigate their suitability and applicability in the modeling and design of MDSs.

1.2.1.1 Bond Graph Theory

Bond graph theory is based on energy flow in the system and considers that the energy exchange between a system and its environment occurs through the connecting ports. Energy stores and energy dissipaters are connected by line segments called bonds. The bonds represent power flow in the system, and the elements are connected through 0- and 1- junctions that respectively represent Kirchhoff's current and voltage laws [1.8.9]. For mechanical systems, a 1-junction represents velocity and a 0-junction represents force elements. Similarly for hydraulic or pneumatic domains, a 1-junction and a 0-junction represent quantities in those domains. Initial system equations are formed using inputs, state variables and co-energy variables which are then reduced to a state space formulation. The system of equations is then solved using numerical methods.

Several packages to simulate bond graph models have been developed over the years, such as the Bond Graph Simulation Program (BGSP) and Hybrid Bond Graph Simulator (HyBrSim). Generally, bond graph models apply to linear systems, but a few languages such as CAMP-G and SIDOPS+ support bond graph modeling of nonlinear systems with continuous as well as discrete elements [1.8.9]. Bond graph theory has a generic structure to model a component in any domain, which makes it easy to model multidisciplinary systems. However, the bond graph approach seems to be weak in modeling nonlinear multidisciplinary systems and in handling kinematic constraints in mechanical systems [1.8.8]. In addition to these drawbacks bond graphs do not provide an intuitive representation of the actual system. This makes the system formulation cumbersome.

1.2.1.2 Linear Graph Theory

Similar to bond graphs, linear graph theory also represents energy flow through the system. The graph consists of edges that represent energy flow in a component, and the component terminals correspond to nodes of the graph. Energy flow is expressed through terminal variables called "through" and "across" variables [1.8.7, 1.8.8]. For example, in mechanical system modeling, forces and torques are identified as "through" variables, and displacements and velocities are identified as "across"

variables. The graph is split into chords and branches, and chord and branch transformations are used to obtain mechanical equilibrium and kinematic loop closure equations, respectively. Branch transformation uses the constrained multibody method. Together, the transformations result in the required differential algebraic equations to describe the mechanical part of the system. Similarly, to formulate the governing equations for electrical systems, the graph theory approach uses currents and voltages in the system. In this case, the chord transformations correspond to Kirchhoff's current law and branch transformations correspond to Kirchhoff's voltage law. Linear graph modeling is a fairly straightforward approach for system modeling and, unlike bond graph models, linear graph models reflect the system topology directly [1.8.9].

Both these approaches use Kirchhoff's current and voltage laws to generate the governing equations of the electrical system and techniques like Lagrange formulation, multibody constraint formulation, and state space modeling for the mechanical counterpart. These techniques are summarized in the following paragraphs and can also be used independently to model multidisciplinary systems.

1.2.1.3 Lagrangian Formulation

The Lagrangian approach is also based on system energy analysis. Dynamic systems are modeled in terms of variables that can be expressed as kinetic energy (T), dissipation energy (D) and potential energy (Π). Lagrange's formulation deals with scalar quantities and is easy to use when the number of elements in a system is low. The Lagrangian formulation suffers from the drawback that its complexity grows exponentially with an increase in the number of components [1.8.6]. System equations need to be derived individually for each element, and the differentiations become extremely cumbersome [1.8.14].

1.2.1.4 Multibody Constrained Formulation

The multibody constrained formulation, which can be considered as a special case of the Lagrangian formulation, overcomes the difficulties in the Lagrangian formulation and can be efficiently used to derive the governing equations of multidisciplinary systems. The basic idea here is to use the maximum set of generalized coordinates

for each body. Considering a general multibody system with N bodies and m constraints, for each body there will be as many as *6* degrees of freedom. That is to say, the system of N bodies will have $6N$ generalized coordinates. Accordingly, the system will have $6N$ differential equations and ($N \times m$) algebraic constraint equations. Let the constraint equations be given by

$$\phi_j^i(q,t) = 0 \qquad (i = 1,2,\ldots,N,\ j = 1,2,\ldots,m) \tag{1.2.1}$$

Then the Lagrangian equation to describe the system will lead to $6N$ differential equations given by Equation (1.2.2).

$$M\ddot{q} - \phi_q^T \lambda = Q \tag{1.2.2}$$

where M is the ($n \times n$) mass matrix, ϕ is the constraint vector, λ is the Lagrange multiplier vector and Q is the generalized force vector. Both the equations together constitute a set of differential algebraic equations in terms of $\ddot{q}$ and λ, which can be solved using standard numerical methods. Electrical and control elements have to be fitted into the above model generally by hand deriving suitable equations.

1.2.1.5 State Space Representation

State-space representation is a popular time-domain technique for mathematical modeling of electrical and mechanical systems. This section briefly describes the general formulation as extracted from [1.8.15]. Euler-Lagrange approaches, Newton's methods and Kirchhoff's laws can be used to derive the differential equations to describe a system. The n first-order differential equations can then be represented as a first order vector-matrix differential equation. The general form of the state-space representation is

$$\begin{aligned} \dot{\mathbf{x}}(t) &= f(\mathbf{x},\mathbf{u},t) \\ \mathbf{y}(t) &= g(\mathbf{x},\mathbf{u},t) \end{aligned} \tag{1.2.3}$$

Here, Equation (1.2.3) represents the state equation and the output equation in the given order. These equations, when linearized about an operating point, can be represented in the following form:

$$\dot{\mathbf{x}}(t) = \mathbf{A}(t)\mathbf{x}(t) + \mathbf{B}(t)\mathbf{u}(t)$$
$$\mathbf{y}(t) = \mathbf{C}(t)\mathbf{x}(t) + \mathbf{D}(t)\mathbf{u}(t) \quad (1.2.4)$$

where $\mathbf{x}$(t) = states, $\mathbf{y}$ = system outputs, $\mathbf{u}$ = system inputs

$\mathbf{A}(t)$ = state matrix, $\mathbf{B}(t)$= input matrix,

$\mathbf{C}(t)$ = output matrix, and $\mathbf{D}(t)$ = direct transmission matrix.

Bond graph theory uses the state space approach to get the governing equations of the system whereas linear graph theory uses Lagrangian formulation along with Kirchhoff's laws to set up the system equations.

1.2.2 Drawbacks of Integration at the Formulation Level

While these methods have been successfully used for multi-domain modeling, there are some disadvantages that limit their range of applicability.

- Not all formulations are best suited to model components from different domains.
- It is not clear how convenient it is to extend these approaches to model new application areas.
- Qualitative changes in system behavior, which are common in electromechanical systems, cannot be handled well by the current approaches. Also sensor fusion strategies often make decisions that may exhibit nonsmooth behavior. Current approaches do not provide support for direct modeling of sensor fusion strategies.
- Current approaches do not provide built-in support for sensitivity analysis.

Nevertheless, researchers have been successful in applying this approach in certain domains. NEWOPT/AIMS [1.8.16] is a software package that includes algorithms for simulation, sensitivity analysis and optimization of a class of mechatronic systems based on a linear state space model. Sensitivities are provided by the semi-analytical adjoint variable method, automatic differentiation or finite differences. Similarly, a linear state space formulation for design sensitivity and optimization of controlled mechanical systems is presented in [1.8.4 – 1.8.6].

1.2.3 Approaches Based on Integration at the Solution Tools Level

Another strategy is to develop different software packages for each relevant problem domain and then integrate these software packages as needed to model an MDS. These domain-specific packages then communicate with each other to simulate the system behavior. MATLAB/Simulink [1.8.17], AMESim [1.8.18], and Simplorer [1.8.19] are some of the existing tools that fall under this category. While this approach provides a working solution in many cases, it is often cumbersome and does not yield an explicit mathematical formulation of the system governing equation which can be used in analytical design sensitivity analysis.

1.2.3.1 MATLAB/Simulink

MATLAB/Simulink [1.8.17] is a popular block-diagram-driven modeling and simulation package. It uses a procedural modeling approach based on signal flow. Simulink was mainly designed for control system applications, and due to the increasing demand for multidisciplinary applications. The Mathworks later introduced several toolboxes such as simMechanics, simPower and simHydraulics to the Simulink architecture. These tool boxes communicate with each other under the Simulink platform to simulate MDSs. The Mathworks' latest version of MATLAB – Release 14 introduces "Link for ModelSim". "Link for ModelSim" is a co-simulation interface that integrates MATLAB and Simulink into the hardware design for field programmable gate array (FPGA) and application specific integrated circuit (ASIC) development. It supports a bidirectional link between MATLAB, Simulink and Model Technology's HDL simulator, ModelSim. This development supports digital system design using VHDL through MATLAB.

1.2.3.2 Simplorer

Simplorer, launched by Ansoft Corporation, was introduced as a multi-domain simulation package in 2001. Simplorer is an integration of external simulators and it provides a common platform for information exchange between the simulators Simulink, MATLAB, Maxwell, and SPICE. The models are developed using C++, VHDL-AMS, Finite Element Analysis package, MATLAB, RMxprt, PEmag, SPICE and SML, and it uses MathCad and MATLAB for mathematical manipulations.

Simplorer was originally developed for transportation and power applications. Later, it was extended to support several other interfaces to suit the requirements of multi-domain system modeling. One of the major motivations for this development was easy access to models provided by several suppliers on the Internet [1.8.19]. This is one of the reasons why all the toolboxes are not well equipped to handle complex systems. For example, in the SPICE library of Simplorer, the built-in static semiconductor models provide basic electrical behavior without representing dynamic semiconductor effects.

1.2.3.3 AMESim

AMESim (Advanced Modeling Environment for Simulation) is another simulation package which provides a modeling and simulation platform to integrate multidisciplinary systems into a single environment [1.8.18]. AMESim also supports various libraries for electrical, hydraulic, and mechanical components. However, it was mainly designed for hydraulic and mechanical systems and hence the mechanical side of AMESim is very sophisticated. On the other hand, the electrical part is developed only to a limited extent to serve mechanical needs, which limits its use in electronic and electrical circuit modeling.

1.2.3.4 Dynast

Dynast is a software package for modeling, simulation and analysis of multidisciplinary nonlinear dynamic systems. It is capable of solving nonlinear and non-stationary DAEs performing analysis in time as well as in frequency domains. It uses an energy based approach where each multipole models the total energetic interaction between a component and the rest of a dynamic system assuming that the interactions take place just in a limited number of component energy entrances like electrical terminals, pipe inlets, mechanical contacts etc. [1.8.20]. Dynast automatically formulates the equations describing the interactions between the components. While DYNAST has been successfully used in several industrial and research problems like electrical and magnetic circuits, mechanical, thermodynamic, fluid-power, and several others, it does not support parametric optimization very

well. Moreover, it is a commercial package, with its free student edition limited to solution of only 16 simultaneous equations.

1.2.3.5 DYMOLA

"Dymola, Dynamic Modeling Laboratory", as mentioned on [1.8.21] "is an environment for modeling and simulation of integrated and complex systems. The multi-engineering capabilities of Dymola allow the user to model and simulate any physical component that can be described by ordinary differential equations and algebraic equations. It is based on Modelica, which is an object-oriented language for physical modeling developed by the Modelica Association". Component models need to be developed using Modelica, and the governing equations as a set of differential and algebraic equations are then solved numerically using Dymola.

1.2.4 Drawbacks of Integration at the Solution Tools Level

While this approach provides a workable solution in some cases, it is often unwieldy and does not provide an integrated modeling, simulation and design platform for multidisciplinary systems. Limitations of this approach can be summarized as follows.

- These approaches do not provide an integrated formulation for modeling, simulation and design under one platform.
- The existing simulation packages are mainly developed for commercial use and the source code is not open to public. Thus, even though they work well as problem solving tools, they do not provide any flexibility for further research.
- All of these packages were designed for a purpose other than multidisciplinary system modeling. They were later modified to suit the current modeling and simulation demands. The downside of this strategy is that each simulation package is powerful in a particular domain but has limited expertise in other domains.
- This approach does not yield an explicit mathematical formulation of the system governing equation which can be used in analytical design sensitivity analysis.

1.2.5 Approaches Based on Integration at the Equation Level

The third approach that is becoming popular can be categorized as a declarative or equation based approach which emphasizes integration across domains at the level of the system governing equations. In the mid-1990s, two specification languages, Modelica [1.8.22, 1.8.23] and VHDL-AMS [1.8.24-1.8.26] were proposed for multi-domain system modeling. The equation-level integration approach, also called the component approach, is the central idea behind the Modelica object-oriented specification language for multidisciplinary systems. VHDL-AMS, openModelica and MathModelica are few other integrated approaches that fall under this category. The following paragraphs will discuss each of these approaches briefly.

1.2.5.1 VHDL-AMS

VHDL-AMS is an Analog and Mixed-Signal extension to VHDL languages [1.8.24-1.8.26]. It is a hardware description language based on a *procedural* paradigm. It was developed for the description and simulation of analog, digital and mixed-signal systems and is oriented more towards electrical/electronic domains. Several researchers have successfully used VHDL-AMS for multidisciplinary system modeling [1.8.26], however its simulation support and generalized libraries are not well developed [1.8.24]. Further, at this point it does not well support extension to parametric studies of multidisciplinary systems very well.

1.2.5.2 Modelica and its Environments – openModelica & MathModelica

Modelica is an equation-based, *object-oriented* language for modeling large, complex, heterogeneous physical systems [1.8.22, 1.8.23]. It has been designed by the developers of Dymola, Omola, SIDOPS+, Smile, ObjectMath and other modeling practitioners in different domains. This is the reason why the language caters to their specific needs. For other users who use Simulink, CAMP-G, Adams etc., it is cumbersome to convert Modelica equations to a compatible format for these packages. This is one of the main reasons why adoption of the Modelica tools is not well encouraged within the companies where the engineers develop their models using Simulink, Adams or other packages which are incompatible with Modelica. As a solution to this problem, interfaces such as Simelika (Simulink and Modelica

interface) [1.8.27] are being developed. Similar interfaces are being developed for VHDL-AMS as well [1.8.26].

Modelica is supported by a limited number of computational environments such as OpenModelica [1.8.28], Dymola [1.8.22, 1.8.29], and MathModelica [1.8.29]. OpenModelica is an effort which effectively integrates mathematical formulation and software architecture using Modelica. It is developed for modeling and simulation of complex multi-domain systems. Its goal is to create a complete Modelica modeling, compilation and simulation environment. The tool generates explicit mathematical equations for multidisciplinary systems [1.8.28], but the compilation process is complex and includes extensive procedures to generate the system equations. Also, it demands the user to have a deep knowledge of Modelica semantics, and the compiler program is complex, with some 100,000 lines of code [1.8.30]. Furthermore, Modelica generates a large number of equations, which need to be reduced extensively by its simulation environments to reduce computational effort.

MathModelica from MathCore is also an integrated interactive development tool [1.8.29] that provides a Modelica simulation environment which is closely integrated into Mathematica and Microsoft Visio. This environment has a graphical editor which is an extension of the Microsoft Visio diagramming tool, and symbolic manipulation is provided via Mathematica. Dymola's symbolic and simulation engine is used for the symbolic transformations and simulations. MathModelica is a commercial package, so its use for further research is limited. Further, it does not provide built-in support for design sensitivity analysis and parametric optimization.

It is important to note that both Modelica and VHDL-AMS are specification languages *only* and thus require an external computational environment to actually simulate system response.

Based on the idea of integration at the equation level, researchers have also used MATLAB's symbolic toolbox (based on Maple) to generate system equations to model multidisciplinary systems. An interesting application of this approach in the automotive field is presented in [1.8.31], where the authors use this strategy to globally model, simulate, and optimize complex industrial mechatronic systems

using MATLAB-Simulink and a Finite Element Method. In both these approaches, sensitivities are calculated using finite differences. However, the authors conclude from their studies that the optimization process could be greatly improved by adopting an integrated modeling approach that supports semi-analytical design sensitivity analysis.

1.3 The Current State-of-the-Art in the Development of Software Architectures for MDSs

One important aspect in the development of solution methods in this area which has not received much attention is the software architecture required to support the mathematical formulation. A suitable architecture for this field must provide a high degree of modularity, flexibility and extensibility. This is particularly important because of the diversity of components in these systems. These systems encompass a wide range of components from various domains, such as rigid bodies and joints from the multibody dynamics domain; circuit elements and semiconductor devices from the electrical domain; motors and tachometers from the electro-mechanical domain; valves and pumps from the hydraulics domain, etc. Each of these components can be best described by using the formulation that is most natural to its domain. For example, Lagrangian or Newtonian mechanics is a very good choice for modeling of multibody systems. Similarly, nodal analysis, linear graph theory are effective methods for modeling electrical systems. Therefore, for accurate modeling of components belonging to different domains, it is important that the software architecture be independent of the modeling approach and flexible enough to accommodate component models at different levels of complexity. Furthermore, the software architecture should provide a plug-and-play facility for domain experts to contribute component models independently, minimizing the need for them to understand other domains or components in detail. This will permit the development of a versatile and complete simulation tool without the need for one person to master all of the relevant domain knowledge. Support for symbolic computing is also desirable for extending the work to parametric studies, design sensitivity analysis, metamodeling, optimization, and robust design.

Modelica and its environments emphasize the development of a suitable software architecture through object-oriented design for convenience and flexibility in modeling. However, while object-oriented code provides modularity, it makes the architecture quite complex. Also, the current software architecture of openModelica and mathModelica is not directly suited for sensitivity analysis and parametric optimization.

1.4 What Is Needed Today?

What is needed today is an integrated formulation-solution scheme with a suitable software architecture to support analysis and design of multidisciplinary systems under a single platform, satisfying the following criteria.

At the mathematical formulation level, we need

- an analysis tool that can analyze all aspects of a multidisciplinary system,
- explicit generation of system governing equations for easy extension to analytical sensitivity analysis formulation, and
- robust and numerically viable formulations

The software architecture in support of the mathematical formulation should satisfy the following requirements:

- modularity in design,
- flexibility in modeling and choice of design variables,
- reusability of models and code,
- extensibility to new problem domains and parametric studies,
- maintainability of code,
- solver support,
- a domain independent architecture.

The primary research problem therefore had to be changed to devise such an integrated scheme.

1.5 Research Problem Presented in this Work

Recall from the previous discussion that the original primary research problem was to develop concurrent optimization techniques to enhance performance of multidisciplinary systems with sensor fusion. After a thorough literature review it was evident that the first step in devising an integrated optimization scheme for multidisciplinary systems is to develop a unified modeling, simulation and sensitivity analysis platform since it doesn't exist. Therefore the primary research problem was redefined and given the title "Integrated Formulation-Solution-Design Scheme for Nonlinear Multidisciplinary Systems in the MIXEDMODELS Platform".

1.6 MIXEDMODELS

MIXEDMODELS, **M**ultidisciplinary **I**ntegrated e**X**tensible **E**ngine for **D**riving **M**etamodeling, **O**ptimization and **DE**sign of **L**arge-scale **S**ystems, is a unified analysis and design tool for MDS that is based on a procedural, symbolic-numeric architecture instead of the object-oriented architecture proposed in the Modelica specification language.

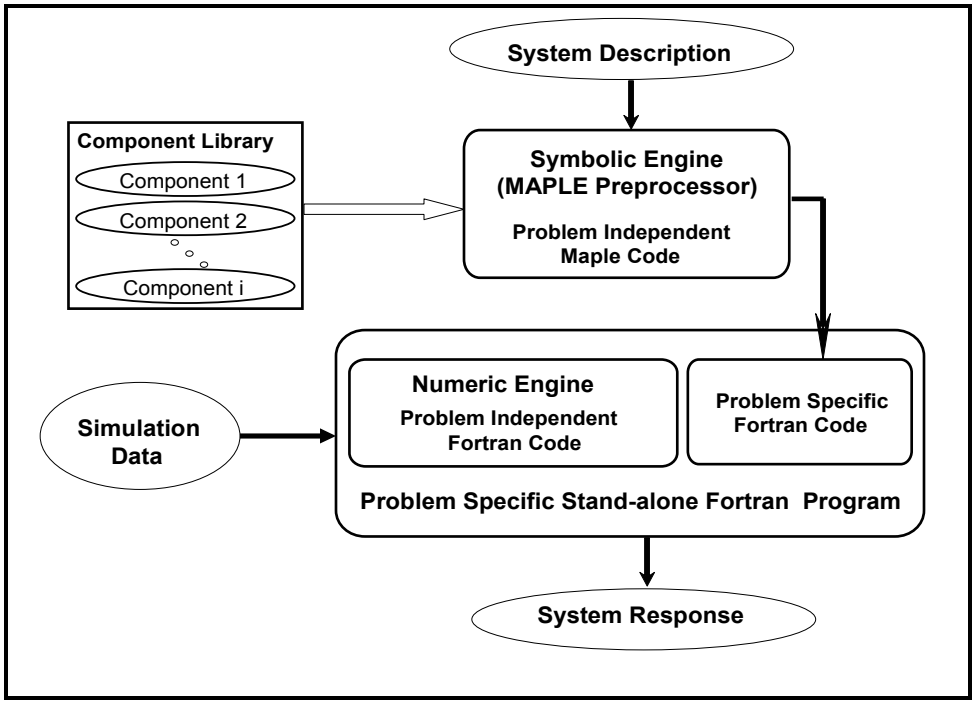


Figure 1.6.1: MIXEDMODELS Architecture

The method is strictly acausal, local/global approach that offers great modeling flexibility at the component level and facilitates extension to new problem domains. The architecture is as shown in Figure (1.6.1). It consists of three modules: the component library, the symbolic engine and numeric engine. This architecture allows any engineer to add components in his/her domain of expertise to the platform in a modular fashion. The symbolic engine in the MIXEDMODELS platform synthesizes the system governing equations as a unified set of nonlinear differential-algebraic equations (DAEs). These equations can then be differentiated with respect to design variables to obtain an additional set of DAEs that represent the sensitivity coefficients of the system state variables with respect to the system's design variables. This combined set of DAEs can be solved numerically to obtain the solution for the state variables and state sensitivity coefficients of the system. Finally, knowing the system performance functions, one can calculate the design sensitivity coefficients of these performance functions by using the values of the state variables and state sensitivity coefficients obtained from the DAEs.

1.6.1 Key Features of MIXEDMODELS

The key features of MIXEDMODELS follow:

- Component-based, procedural, symbolic-numeric architecture
- *Acausal* modeling approach
- Symbolic engine developed using Maple, leading to a simple, easily maintainable code
- Numeric engine developed using Fortran for linear & nonlinear system solution
- Implementation of a symbolic reduction routine and sparse matrix solvers to improve computational efficiency
- Implementation of a scaling algorithm to improve numerical accuracy
- Explicit analysis and sensitivity analysis formulation extended to support concurrent optimization techniques

- Open architecture providing plug-n-play facility for domain experts to contribute components independently
- Easy extension to new components and new problem domains
- Robust, efficient and computationally viable architecture that supports various numerical solvers

1.7 Dissertation Outline

This dissertation is organized into nine chapters and three appendices. It is based on the "alternative thesis format" which includes manuscripts as thesis chapters, in the same format as they are published in conference proceedings. These chapters are largely based on the original publications in conference proceedings; however, to provide completeness and improve continuity and readability from the reader's perspective, some details have been added to the original manuscripts which could not be included due to the restriction on the number of pages.

The first chapter presented a detailed introduction and literature review of the existing techniques related to the modeling and simulation of multidisciplinary systems. Chapters 2 through 6 include the manuscripts in the following order.

- Chapter 2: Component Based Modeling of Electromechanical Systems Published in the Proceedings of ASME - IDETC/CIE 2005.
- Chapter 3: Design Sensitivity Analysis of Multidisciplinary Multibody Systems Published in the Proceedings of Multibody Dynamics 2007 Thematic Conference.
- Chapter 4: Design Sensitivity Analysis of Nonlinear Multidisciplinary Multibody Systems in the MIXEDMODELS platform, Published in the Proceedings of ASME – IDETC/CIE 2007.
- Chapter 5: Symbolic-Numeric Computing in Software Development for Modeling and Simulation of Multidisciplinary Systems, Published in the Proceedings of Multibody Dynamics 2007 Thematic Conference.

- Chapter 6: Metamodeling of Mechatronic Systems in the MIXEDMODELS Platform, Published in the Proceedings of ASME – IDETC/CIE 2007.

Chapter 7 presents a discussion on several numerical solvers and methods used by the numeric engine of MIXEDMODELS. Each manuscript included here presents a different development stage of MIXEDMODELS along with supporting examples demonstrating the significance and effectiveness of each stage. Chapter 8 presents a numerical example which summarizes the concepts developed in all five of these manuscripts. Finally, chapter 9 presents a concluding discussion which gives an overview of what has been accomplished so far in the development of MIXEDMODELS. It also provides an extended discussion of the "work in progress" and discusses future work. For convenience and completeness from the reader's perspective, each of the chapters is self contained and includes a reference list for that chapter.

Appendix A provides a detailed derivation of MIXEDMODELS formulation and Appendix B provides the MAPLE code for the symbolic engine of MIXEDMODELS.

1.8 REFERENCES

1.8.1 Harashima F., Tomizuka, M., Fukuda, T., '*Mechatronics* – "*What is it, Why and How?*" An Editorial', IEEE/ASME Transactions on Mechatronics, Vol. 1, No.1, pp. 1-2, March 1996.

1.8.2 Arnold, M. and Heckmann, A., '*From Multibody Dynamics to Multidisciplinary Applications*', Multibody Dynamics: Computational Methods and Applications, pp. 273-294, 2007.

1.8.3 Carrigan, J., Kelkar, A. and Krishnaswami, P., '*Minimum Sensitivity Design of Controlled Multibody Systems*', ASME Multibody Systems, Nonlinear Dynamics, and Control Conference, 2005.

1.8.4 Kelkar, A., Krishnaswami, P., '*Multidisciplinary Optimization of Multibody Systems*', Proceedings of the ECCOMAS Conference on Multibody Systems, June 2005.

1.8.5 Carrigan, J., '*Integrated Design and Sensitivity Based Design and Optimization of Nonlinear Controlled Multibody Mechanisms*', Thesis, M.S. Iowa State Univ., 2003.

1.8.6 Luo, R.C., '*Sensor Technologies and Microsensor Issues for Mechatronic Systems*', IEEE/ASME Transactions on Mechatronics, Volume 1, Issue 1, pp. 39-49, 1996.

1.8.7 Scherrer, M. and McPhee, J., '*Dynamic Modelling of Electromechanical Multibody Systems*', Multibody System Dynamics, 9, pp. 87-115, 2003.

1.8.8 Sass, L., McPhee, J., Schmitke, C., Fisette, P. and Grenier, D., '*A Comparison of Different Methods for Modelling Electromechanical Multibody Systems*', Multibody System Dynamics, 12: pp. 209-250, 2004.

1.8.9 Sinha, R., Paredis, C.J.J., Liang, V-C., Khosla, P.K., '*Modeling and Simulation Methods for Design of Engineering Systems*', Journal of Computing and Information Science in Engg., Volume 1, Issue 1, pp. 84-91, March 2001.

1.8.10 Granda, J.J., '*The Role of Bond Graph Modeling and Simulation in Mechatronic Systems. An Integrated Software Tool: Camp-G, MATLAB-Simulink.*' Proceedings of Mechatronics Conference, Atlanta, GA, pp. 1271- 1295, 2000.

1.8.11 Taehyun, S., '*Introduction to Physical System modeling Using Bond Graphs*', Vetronics Inst. 2nd Annual Workshop Series, pp. 430-434, Dec. 2002.

1.8.12 Greenwood, D.T., '*Principles of Dynamics, Second Edition*', Prentice Hall, 1988.

1.8.13 Giaurgiutiu, V., Lyshevski, S.E., '*Micromechatronics, Modeling, Analysis, and Design with MATLAB*', CRC Press, 2003.

1.8.14 Huston, R.L., '*Multibody Dynamics*', Butterworth-Heinemann, 1990.

1.8.15 Ogata, K., '*Modern Control Engineering*', Third Edition, Prentice Hall, 1997.

1.8.16 Dignath, F., Breuninger, C., Eberhard, P., Kübler, L., '*Optimization of Mechatronic Systems Using the Software Package NEWOPT/AIMS*', Multibody System Dynamics, pp. 85-100, 2005.

1.8.17 Ram, O., Szymkat, M., Uhl, T., Betemps, M., Pjetursson, A. and Rod, J., '*Mechatronic Blockset for Simulink – Concept and Implementation*', Proceedings of the 1996 IEEE Intl. Symposium on Computer-Aided Control System Design, pp. 530-535, 1996.

1.8.18 http://www.amesim.com/, IMAGINE Software Inc., acquired on Jan 29, 2005.

1.8.19 Knorr, U.I., '*Electromechanical System Design*', Ansoft Corporation, 2002.

1.8.20 Mann, H., '*A versatile Modeling and Simulation Tool for Mechatronics Control System Development*', proceedings of the 1996 IEEE International Symposium on Computer-Aided Control System Design, pp. 524-529, Sept. 1996.

1.8.21 http://www.dynasim.com/simulations.htm, acquired on July 08, 2007.

1.8.22 Fritzson, P., Bunus, P., '*Modelica – A General Object-Oriented Language for Continuous and Discrete-Event System Modeling and Simulation*', Proceedings of the 35th Annual Simulation Symposium, pp. 365-380, IEEE, 2002.

1.8.23 Elmqvist, H., Mattsson, S.E., and Otter, M., '*Object-Oriented and Hybrid Modeling in Modelica*', Journal Européen des systèmes automatisés, 35, 1/2001, pp. 1 à X.

1.8.24 Smith, D.W., '*Analog/Mixed-signal Modeling*', Synopsis Incorporation, CANDE: AMS Modeling and Synthesis, 2003.

1.8.25 Pecheux, F., Allard, B., Lallement, C., Vachoux, A., Morel, H., '*Modeling and Simulation of Multi-Discipline Systems Using Bond Graphs and VHDL-AMS*',

International Conference on Bond Graph Modeling and Simulation, pp. 23-27, 2005.

1.8.26 Frey, P., Nellayappan, K., Shanmugasundaram, V., Mayiladuthurai, R.S., Chandrashekhar, C.L., Carter, H.W., '*SEAMS*: *Simulation Environment for VHDL-AMS*', Proceedings of the 1998 Winter Simulation Conference, pp. 539-546, 1998.

1.8.27 Dempsey, M., '*Automatic Translation of Simulink Models into Modelica using Simelica and the AdvancedBlocks Library*', The Modelica Association, Modelica, 2003.

1.8.28 Fritzson, P., Aronsson, P., Pop, A., Lundvall, H., Nystrom, K., Saldamli, L., Broman D., Sandholm, A. '*OpenModelica – A Free Open-Source Environment for System Modeling, Simulation, and Teaching*', Proceedings of IEEE Conference on Computer Aided Control Systems Design, pp. 1588-1595, 2006.

1.8.29 Fritzson, P., Gunnarsson, J., Jirstrand, M., '*MathModelica – An Extensible Modeling and Simulation Environment with Integrated Graphics and Literate Programming*', 2nd International Modelica Conference, Proceedings, pp. 41-54, Mar. 2002.

1.8.30 Larsson, J., Fritzson, P., '*A Modelica-based Format for Flexible Modelica Code Generation and Causal Model Transformations*', The Modelica Association, pp. 467-475, 2006.

1.8.31 Duysinx, P., Bruls, O., Collard, J-F., Fisette, P., Lauwerys, C., Swevers, J., '*Optimization of Mechatronic Systems: Application to a Modern Car Equipped with a Semi-active Suspension*', 6th World Congress of Structural and Multidisciplinary Optimization, 2005.

CHAPTER 2 - COMPONENT-BASED MODELING OF ELECTROMECHANICAL SYSTEMS

Manuscript Publication:

Proceedings of IDETC/CIE 2005:

ASME 2005 International Design Engineering Technical Conferences & Computers and Information in Engineering Conference

September 24-28, 2005, Long Beach, California, USA

Authors:

Shilpa A. Vaze[†], Prakash Krishnaswami[*], James E. DeVault[†].

Authors' Affiliations:

[†] Department of Electrical and Computer Engineering, Kansas State University, Manhattan, KS 66506.

[*] Department of Mechanical and Nuclear Engineering, Kansas State University, Manhattan, KS 66506.

NOTE: This chapter is largely based on the original publication, however; changes have been made to the manuscript to maintain continuity and consistency in reading.

2.1 Abstract

Simulation methods for electromechanical systems should accommodate their interdisciplinary nature and the fact that these systems often display qualitative changes in system behavior during operation, such as saturation effects and changes in kinematic structure. Current approaches are either based on deriving the system equations by applying a single formulation to all problem domains, or they are based on trying to integrate different software packages/modules to solve the interdisciplinary problem. In this paper, we present a component-based approach which allows the governing equations of each component to be defined in terms of its natural variables. The different component equations are then brought together to form a single system of differential-algebraic equations (DAEs), which can be numerically solved to obtain the system response. The fact that we have an explicit, unified form of the system governing equations means that this formulation can be easily extended to design sensitivity analysis and optimization of electromechanical systems (EMSs). The formulation includes monitor functions which can be used to detect when a qualitative system change has occurred, and to switch to a new set of governing equations to reflect this change. A single step integrator is used to make it easier to switch to a new system behavior, since this will always require a restart of the integrator. There is considerable flexibility in how the components can be defined, and connections between components are themselves modeled as special types of components. Examples of components from the mechanical and electrical side are presented, and two numerical examples are solved to illustrate the efficacy of the proposed method. One example is a link that is driven by a DC motor through a gearbox. The results of this example were verified against Simulink, and good agreement was observed. The second example is a motor driven slider-crank mechanism. The method can be extended to include components from any domain, such as hydraulics, thermal, controls, etc., as long as the governing equations can be written as DAEs.

2.2 Background

2.2.1 Modeling and Simulation of EMSs

Dynamic modeling and simulation of electromechanical systems has been an open topic of research for more than two decades [2.6.1]. An EMS can be modeled as a set of ordinary differential equations, algebraic equations or differential and algebraic equations (DAE). Several approaches such as differential geometric frame approach [2.6.2], bond graph modeling ([2.6.3], [2.6.4]), linear graph theory [2.6.2], Lagrangian formulation [2.6.5], and multibody constrained formulation ([2.6.6], [2.6.7]) have been developed to derive governing equations of mechatronic systems. Newtonian modeling can be used to derive the system governing equations for mechanical systems; however, its use for electromechanical system modeling is limited due to its inability to model electromagnetic systems [2.6.8]. Over the last few years, several simulation packages such as Simulink and P-Spice have been developed, but only a few of them can be used for modeling multidisciplinary systems [2.6.9]. Current modeling approaches for electromechanical systems can be categorized into types:

- Modeling approaches based on integration at the formulation level
- Modeling approaches based on integration at the solution level

2.2.1.1 Integration at the Formulation Level

At the formulation level, the idea is to develop the governing equations using one physical/mathematical formulation and solve the system of equations using numerical tools. Bond graph theory, linear graph theory and Lagrangian formulation are among the well-known approaches that fall under this category. These methods are based on energy flow in the system and derive system equations based on energy exchange between the system and its environment.

In the **bond graph approach,** system equations are formed using state space formulation. The system of equations is then solved using numerical methods. Bond graph theory has a generic structure to model a component in any domain which makes it possible to model electromechanical systems. However, this approach is not

as strong in modeling nonlinear multidisciplinary systems and in handling kinematic constraints in mechanical systems [2.6.10]. Bond graphs also do not provide an intuitive representation of the actual system.

Linear graph modeling, on the other hand, is a fairly straightforward approach for system modeling. Unlike bond graph models, linear graph models reflect the system topology directly [2.6.9]. Linear graph theory uses Lagrangian formulation along with Kirchhoff's laws to set up EMS governing equations.

Lagrange's formulation deals with scalar quantities and is easy to use when the number of elements in a system is low. It has been used to derive electromechanical system equations in some applications [2.6.6]. For mechanical systems, Lagrange's energy equations are derived using Newton's laws of motion, whereas for electrical systems they are derived using Kirchhoff's current and voltage laws as presented in [2.6.8].

State-space representation [2.6.11] is another popular time domain technique for mathematical modeling of electrical and mechanical systems. The drawback of state space modeling is its unsuitability for modeling nonlinear systems. Further, not all relevant domains and devices can be modeled naturally in the state space form.

2.2.1.1.1 Drawbacks of Integration at the Formulation Level

- The formulations are powerful in one domain but have limited modeling capabilities in other domains.
- It is not clear how convenient it is to extend these approaches to model new application areas.
- Qualitative changes in system behavior, which are common in electromechanical systems, cannot be handled well by the current approaches. Also sensor fusion strategies often make decisions that may exhibit non-smooth behavior. Current approaches do not provide support for direct modeling of sensor fusion strategies.

2.2.1.2 Integration at the Solution Tools Level

In this type of approach, different software modules are developed to model components belonging to different domains. These modules then communicate with each other to simulate the behavior of the electromechanical system. The Mathworks **Simulink** tool is a popular block-diagram type modeling and simulation package. Simulink was mainly designed for control system applications. To meet the current modeling requirements in electromechanical system modeling, two toolboxes, namely simMechanics and simPower, were added in the Simulink architecture. **AMESim** (Advanced Modeling Environment for Simulation) is another simulation package which provides a modeling and simulation platform to integrate multidisciplinary systems into a single environment [2.6.12]. It also supports various libraries for electrical, hydraulic, and mechanical components. However, it was mainly designed for hydraulic and mechanical systems, and hence the mechanical side of AMESim is sophisticated. On the other hand, the electrical part is developed only to a limited extent to serve mechanical needs, which limits its use in electrical circuit modeling. **Simplorer**, launched by Ansoft Corporation [2.6.13], was introduced as a multi-domain simulation package in 2001. Simplorer is an integration of external simulators, and it provides a common platform for information exchange between the simulators Simulink, MATLAB, Maxwell, and SPICE. Simplorer was originally developed for transportation and power applications. Later, it was extended to support several other interfaces to suit the requirements of multi-domain system modeling.

2.2.1.2.1 Drawbacks of Integration at the Solution Tools Level

- While these tools support problem solving in this area, they do not provide an integrated, easily extensible formulation.
- The existing simulation packages are mainly developed for commercial use and do not provide flexibility for further development from a research point of view.
- All these packages are powerful in a particular domain but have limited expertise in other domains.

- The current methodologies do not have an explicit mathematical model to support other functions such as sensitivity analysis, optimization, etc.

2.3 The Proposed Approach

We propose a component-based modeling approach where an electromechanical system is considered to be a collection of interacting components. This technique allows each component to be modeled using the formulation that is most natural for its domain. Each component is described by a set of variables of interest and governing equations which are allowed to be nonlinear differential-algebraic equations (DAEs). The component equations are then combined to form a system of equations that is solved simultaneously using single step numerical methods.

The concept of monitor functions is a key feature of this approach. In practice, it is common for electromechanical systems to undergo qualitative changes in their behavior while they are in operation. When such a qualitative change occurs, it usually represents a change in the structure of the system governing equations. When a system undergoes a change in behavior, monitor functions detect the change and allow the system to switch to alternate models which capture the new dynamics of the system. Current modeling approaches do not handle this type of behavior well.

This proposed formulation gives an explicit set of mathematical equations which will allow us to extend the formulation to support parametric optimization, sensitivity analysis, reliability studies, etc. The approach is based on the assumption that the system behavior is piecewise smooth in time. This assumption along with the component-based structure will allow extension to discrete and hybrid system modeling.

2.3.1 Mathematical Formulation

2.3.1.1 System Formulation

An electromechanical system can be completely described by a vector of time-invariant system parameters, $\mathbf{P}$; a vector of system variables, $\mathbf{X}$, which can also occur in first derivative form $\dot{\mathbf{X}}$ in the DAEs; a vector of algebraic system variables, $\mathbf{Y}$, that

occur algebraically in the DAEs; and a set of governing equations (DAEs) **f = 0**, i.e., the system governing equations are written in the following form.

$$\mathbf{f}(\mathbf{P}, \mathbf{X}, \dot{\mathbf{X}}, \mathbf{Y}, t) = 0 \tag{2.3.1}$$

Most electromechanical components can be described by governing equations written in a matrix form as:

$$\tilde{\mathbf{A}}\left(\mathbf{P}, \mathbf{X}, t\right)\begin{bmatrix}\dot{\mathbf{X}} \\ \mathbf{Y}\end{bmatrix} = \tilde{\mathbf{b}}\left(\mathbf{P}, \mathbf{X}, t\right) \tag{2.3.2}$$

Although this form is not as general as Equation (2.3.1), it is easier to work with and encompasses most systems of interest. Many electromechanical systems are governed by higher order differential equations. An n^{th} order differential equation must be reduced to n first-order differential equations by defining additional variables. The first derivatives of these additional variables can always be obtained directly from the **X** vector by assignment. We refer to these additional variables as fixed variables, and the state vector can be partitioned as

$$\mathbf{X} = \begin{bmatrix}\mathbf{X}_{free} \\ \mathbf{X}_{fixed}\end{bmatrix} \tag{2.3.3}$$

Computation of the fixed variables as a part of the matrix equation unnecessarily increases the system size and becomes computationally more expensive. These variables are therefore removed from the linear system of equations and handled separately by direct assignment equations which are expressed by the following form:

$$\dot{\mathbf{X}}_{fixed} = \mathbf{X}' \tag{2.3.4}$$

where $\mathbf{X}'$ is a subvector of **X.** The matrix form can then be rewritten as

$$\mathbf{A}\left(\mathbf{P}, \mathbf{X}, t\right)\begin{bmatrix}\dot{\mathbf{X}}_{free} \\ \mathbf{Y}\end{bmatrix} = \mathbf{b}\left(\mathbf{P}, \mathbf{X}, t\right) \tag{2.3.5}$$

Equation (2.3.4) and Equation (2.3.5) constitute the complete set of system equations which must be solved numerically to obtain the system response. Note that the

coefficient matrix and right hand side in Equation (2.3.5) differs from that in Equation (2.3.2) because the columns of the matrix and the rows of the right hand side corresponding to the fixed variables have been eliminated, reducing system size.

2.3.1.2 Mathematical Description of Components

A **component** is a part of an electromechanical system and is characterized by the following:

- A vector of time-invariant component parameters, $\mathbf{p}^i$, where $\mathbf{p}^i$ is a subvector of $\mathbf{P}$.
- A vector of transient component variables, $\mathbf{x}^i$, $\mathbf{x}^i$ being a subvector of $\mathbf{X}$, which can occur in the governing DAEs in first derivative form $\dot{\mathbf{x}}^i$
- A vector of component algebraic variables, $\mathbf{y}^i$, $\mathbf{y}^i$ being a subvector of $\mathbf{Y}$, which occur algebraically in the DAEs
- A set of component governing equations expressed in matrix form as

$$\mathbf{A}^c\left(\mathbf{P},\mathbf{X},t\right)\begin{bmatrix}\dot{\mathbf{X}}\\ \mathbf{Y}\end{bmatrix}_{free}=\mathbf{b}^c\left(\mathbf{P},\mathbf{X},t\right) \tag{2.3.6}$$

- Modification matrices $\mathbf{A}^m$ and $\mathbf{b}^m$ which allow this component to modify governing equations of other components. These modification matrices can be zero if the component does not modify the governing equations of any other component.

When a particular component is being processed, its $\mathbf{A}^c$ and $\mathbf{A}^m$ matrices are added to the current system matrix, $\mathbf{A}$, creating additional rows and/or columns as needed. Similarly, the $\mathbf{b}^c$ vector of the component is added to the current system $\mathbf{b}$ vector. Fixed variables for a component should be set using the direct assignment equation given by Equation (2.3.4). We distinguish between two types of components:

A **local component** is a component which does not modify the governing equations of any other component. The behavior of a local component can be specified in terms of time-invariant parameters and component variables defined locally for that

component, but these may be modified by other components in the system. From this definition it follows that the modification matrices $\mathbf{A}^m$ and $\mathbf{b}^m$ for a local component are zero. Each local component introduces a new set of transient variables ($\mathbf{x}^i$, $\dot{\mathbf{x}}^i$ and $\mathbf{y}^i$) and a new set of governing equations given by Equation (2.3.6) to the system of equations given by Equations (2.3.4) and (2.3.5). DC motors and rigid bodies are two examples of local components.

A **nonlocal component** is a component whose parameter list involves indices of at least one other component. It may introduce new variables to the system or may just be defined in terms of the variables of the components whose indices occur in that component's parameter list. Furthermore it may introduce new governing equations expressed in the form of Equation (2.3.6). Note that the form of Equation (2.3.6) does not allow governing equations of the form

$$\mathbf{f}(\mathbf{P}, \mathbf{X}, \dot{\mathbf{X}}_{fixed}, t) = 0 \tag{2.37}$$

The process of generating system DAEs in standard form may involve differentiation of some equations. This process, if not handled properly, may cause the coefficient matrix **A** to be singular [2.6.14]. The form (2.3.7) is forbidden to avoid this type of singularity.

A non-local component can also modify the equations of the local components whose indices occur in that component's parameter list. Local components need to be defined prior to the definition of non-local components, as the latter are defined in terms of variables of the former. Modifications caused by non-local components can be incorporated in the system coefficient matrices using the correction matrices $\mathbf{A}^{m_i}$ and $\mathbf{b}^{m_i}$. These modifications can be expressed using the following equation.

$$\begin{aligned} \mathbf{A}\left(\mathbf{P}, \mathbf{X}, t\right) &= \mathbf{A}\left(\mathbf{P}, \mathbf{X}, t\right) + \mathbf{A}^{m_i}\left(\mathbf{P}, \mathbf{X}, t\right) \\ \mathbf{b}\left(\mathbf{P}, \mathbf{X}, t\right) &= \mathbf{b}\left(\mathbf{P}, \mathbf{X}, t\right) + \mathbf{b}^{m_i}\left(\mathbf{P}, \mathbf{X}, t\right) \end{aligned} \tag{2.3.8}$$

Revolute joints, mechanical couplings, and electrical connections are a few examples of non-local components.

2.3.1.3 Component Examples

2.3.1.3.1 Local Component Examples

Every component will be described locally in terms of its own set of parameters, variables and governing equations.

a. DC Motor

This is an example of a DC motor component which is assumed to be component *i* in the system. The parameter list for this component is as described below.

Component Parameters:

$$\left[J_m, R_a, L_a, V_a, K_b, K_T, B_v\right]^T \equiv \mathbf{p}^i \tag{2.3.9}$$

where, J_m: motor inertia; R_a: armature resistance; L_a: armature inductance; V_a: armature voltage; K_b: back-emf constant; K_T: torque constant; B_v: viscous friction coefficient.

Component Variables:

DC motor component is described locally in terms of a set of four variables: θ_m: motor angular rotation; ω_m: motor angular velocity; I_a: armature current and T_D: motor driving torque. θ_m represents the *fixe*d variable, where as ω_m and I_a are the *free* variables. T_D represents the algebraic variable of the DC motor component.

$$\begin{aligned} &\left[\theta_m, \omega_m, I_a, T_D, \dot{\theta}_m, \dot{\omega}_m, \dot{I}_a\right]^T \\ &\text{Here, } \left[\theta_m, \omega_m, I_a\right]^T \equiv \mathbf{x}^i \qquad \left[T_D\right]^T \equiv \mathbf{y}^i \\ &\qquad \left[\dot{\theta}_m, \dot{\omega}_m, \dot{I}_a\right]^T \equiv \dot{\mathbf{x}}^i \qquad \left[\dot{\theta}_m\right]^T \equiv \dot{\mathbf{x}}^i_{fixed} \\ &\qquad \left[\dot{\omega}_m, \dot{I}_a\right]^T \equiv \dot{\mathbf{x}}^i_{free} \end{aligned} \tag{2.3.10}$$

Component Governing Equations:

DC motor component contributes three equations to the system. The first equation is a direct assignment equation which is generated during the process of converting the second order motor equation to the first order equation. Second equation describes the relation between the motor speed, generated emf and the torque, and the last

equation describes the voltage balance in a DC motor. These equations are summarized below.

$$\begin{aligned} \dot{\theta}_m &= \omega_m \\ \dot{\omega}_m J_m + T_D &= -\omega_m B_v + I_a K_T \\ \dot{I}_a L_a &= V_a - \omega_m K_b - I_a R_a \end{aligned} \tag{2.3.11}$$

b. Rigid Body

Figure (2.3.1) shows a rigid body component which is assumed to be component *i* in the system. Rigid body parameters, variables and governing equations are as listed below.

Component Parameters:

The rigid body component has mass, moment of inertia and the net forces as its time-invariant parameters. The time-invariant vector $\mathbf{p}^i$ for this component is as given below.

$$\left[m_i, J_i, F_x, F_y, M_i\right]^T \equiv \mathbf{p}^i \tag{2.3.12}$$

where m_i: mass of body *i*; J_i: moment of inertia of body *i*; F_x: net constant force in the *x* direction; F_y: net constant force in the *y* direction; M_i: net constant force in moments.

Component Variables:

$$\left[x_i, y_i, \theta_i, \dot{x}_i, \dot{y}_i, \dot{\theta}_i\right]^T \tag{2.3.13}$$

where x_i: *x*-coordinate of the center of mass (CM) of body *i;* y_i: *y*-coordinate of the center of mass of body *i*; θ_i: angular coordinate of body *i.* All coordinates are specified with respect to the global frame of reference.

Component Governing Equations:

Governing equations of a rigid body are described by Euler's laws. This component contributes three equations to the system. The first two equations describe how the forces control translational motion of the body in the *x* and *y* directions respectively.

The third equation, on the other hand, describes how the moment of forces control the change in angular momentum of the body. The equations of motion for the rigid body component are described below.

$$\begin{aligned} m_i \ddot{x}_i - F_x &= 0 \\ m_i \ddot{y}_i - F_y &= 0 \\ J_i \ddot{\theta}_i - M_i &= 0 \end{aligned} \tag{2.3.14}$$

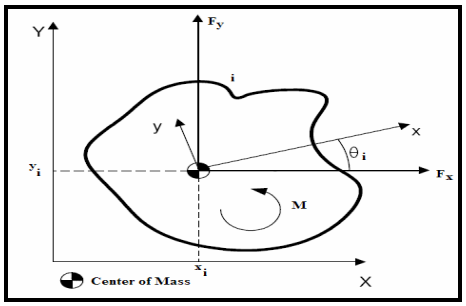


Figure 2.3.1: Rigid Body Component Model (Adapted from [7])

2.3.1.3.2 Nonlocal Component Examples

a. Revolute Joint

Figure (2.3.2) shows a revolute joint component which shows the parameters of the component. Revolute joint component is a nonlocal component which not only introduces new variables and equations, but also modifies the equations of the two rigid body components that it connects.

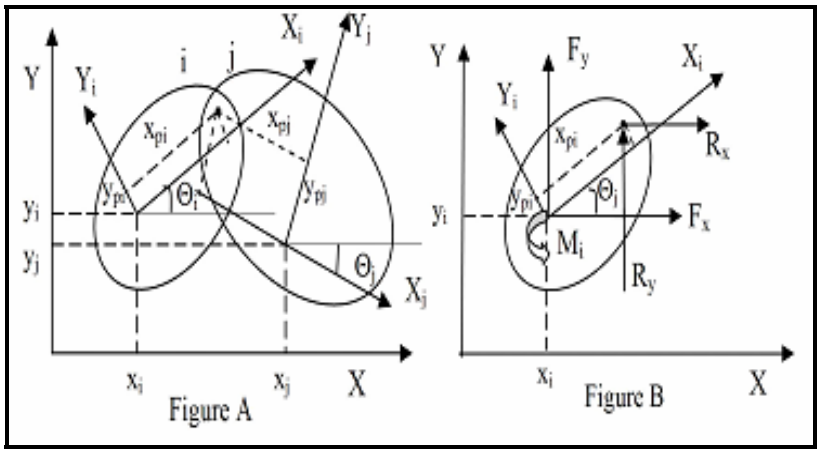


Figure 2.3.2: Revolute Joint Component Model

Component Parameters:

The parameter list of the revolute joint component consists of the indices *i* and *j*, of the two bodies it connects and the coordinates of the revolute joint with respect to the local frame of body *i* and *j*.

$$\left[i, j, x_{p_i}, y_{p_i}, x_{p_j}, y_{p_j}\right]^T \equiv \mathbf{p}^i \tag{2.3.15}$$

Component Variables:

This component introduces two algebraic variables to the system as given by Equation (2.3.16).

$$\left[R_x, R_y\right]^T \equiv \mathbf{y}^i \tag{2.3.16}$$

Here R_x and R_y are the reaction forces at the joint.

Component Governing Equations:

The governing equations of this component are the geometric constraint equations in *x* and *y* directions. These equations are described in terms of the local variables of the revolute joint component and rigid body components *i* and *j*.

$$\begin{aligned} &\ddot{x}_i + x_{p_i}\left(-\ddot{\theta}_i \mathrm{s}\theta_i - \dot{\theta}_i^2 \mathrm{c}\theta_i\right) - y_{p_i}\left(\ddot{\theta}_i \mathrm{c}\theta_i - \dot{\theta}_i^2 \mathrm{s}\theta_i\right) \\ &-\ddot{x}_j + x_{p_j}\left(-\ddot{\theta}_j \mathrm{s}\theta_j - \dot{\theta}_j^2 \mathrm{c}\theta_j\right) - y_{p_j}\left(\ddot{\theta}_j \mathrm{c}\theta_j - \dot{\theta}_j^2 \mathrm{s}\theta_j\right) = 0 \end{aligned} \tag{2.3.17}$$

$$\begin{aligned} &\ddot{y}_i + x_{p_i}\left(\ddot{\theta}_i \mathrm{c}\theta_i - \dot{\theta}_i^2 \mathrm{s}\theta_i\right) + y_{p_i}\left(-\ddot{\theta}_i \mathrm{s}\theta_i - \dot{\theta}_i^2 \mathrm{c}\theta_i\right) \\ &-\ddot{y}_j + x_{p_j}\left(\ddot{\theta}_j \mathrm{c}\theta_j - \dot{\theta}_j^2 \mathrm{s}\theta_j\right) + y_{p_j}\left(-\ddot{\theta}_j \mathrm{s}\theta_j - \dot{\theta}_j^2 \mathrm{c}\theta_j\right) = 0 \end{aligned} \tag{2.3.18}$$

Equations (2.3.17) and (2.3.18) are obtained by two differentiations of the geometric constraint equations of a revolute joint.

2.3.1.3.3 Modifications in Governing Equations of Rigid Bodies Due to Revolute Joint Connection

A revolute joint exerts reaction forces R_x and R_y on both the rigid bodies at the joint. These joint forces need to be incorporated in the individual governing equations of

bodies *i* and *j*. This effect is expressed in the individual governing equations of body *i*, and in the form of modification matrices $\mathbf{A}^{m_i}$ and $\mathbf{b}^{m_i}$.

Let (x_{pi}, y_{pi}) and (x_{pj}, y_{pj}) be the coordinates of the revolute joint with respect to the local frame of body *i* and *j*, respectively. Let (J_i, J_j), (m_i ,m_j) and (θ_i θ_j) be the moment of inertia, mass and angular coordinate of body *i* and *j* respectively. Let (x_i, y_i) and (x_j, y_j) be the coordinates of center of mass of bodies *i* and *j*, respectively relative to the global frame of reference. Then the modifications are given as follows.

Modification in Equations of Body i

$$\begin{aligned} \sum F_x &= m_i \ddot{x}_i - R_x = 0 \\ \sum F_y &= m_i \ddot{y}_i - R_y = 0 \\ \sum M &= J_i \ddot{\theta}_i + R_x (x_{p_i} \mathrm{s}\theta_i + y_{p_i} \mathrm{c}\theta) \\ &\quad - R_y (x_{p_i} \mathrm{c}\theta_i - y_{p_i} \mathrm{s}\theta_i) = 0 \end{aligned} \tag{2.3.19}$$

Modification in Equations of Body j

$$\begin{aligned} \sum F_x &= m_j \ddot{x}_j + R_x = 0 \\ \sum F_y &= m_j \ddot{y}_j + R_y = 0 \\ \sum M &= J_j \ddot{\theta}_j - R_x (x_{p_j} \mathrm{s}\theta_j + y_{p_j} \mathrm{c}\theta_j) \\ &\quad + R_y (x_{p_j} \mathrm{c}\theta_j - y_{p_j} \mathrm{s}\theta_j) = 0 \end{aligned} \tag{2.3.20}$$

2.3.1.4 Monitor Functions

The concept of monitor functions is an important aspect of this proposed approach. While in operation an EMS can undergo qualitative changes in its behavior. Such an event changes the system dynamics, which is reflected in the governing equations of the system. One example of this is current saturation in a DC motor, where the voltage-current relationship changes when we enter or leave current saturation. Similarly, when the leg of a walking machine touches down, new kinematic constraints must be added to the system of DAEs. These equations must be removed later when the leg is raised. In some cases, such as impact phenomena, the qualitative change may also be accompanied by discontinuous changes in variables. Current modeling approaches do not handle this type of behavior well. Facilities are available for finding the zeros of functions, such as the zero-crossing feature in Simulink, but these are not directly coupled to the required changes in the system governing equations.

We propose to resolve this difficulty by allowing monitor functions and alternate sets of component equations in the model. The monitor functions will be defined so that when such a function goes through a zero, it indicates the occurrence of an associated qualitative change in the component's behavior. The system then switches to an alternate set of governing equations which describe the new dynamics. The use of a single step integrator gives us the flexibility that is needed to make this change in the governing equations.

Every function, whether it is a monitor function or a governing equation, is associated with a flag which indicates whether this function is "on" or "off". Only functions that are "on" are considered to be active during the analysis, and the others are ignored. The active monitor functions are checked at every time step to see if any of them have gone through a zero. If this has happened within the last integration time step, then the associated logic of the relevant monitor function(s) is executed to reset the "on" and "off" flags of all the governing equations and monitor functions. The integration is then restarted from this point. If the last timestep was large, interpolation can be used to find the exact time when the change has occurred, and integration can be restarted from this point. Monitor functions thus enable the system to respond to qualitative changes in its behavior by turning the appropriate functions on or off.

This concept can be illustrated by studying the behavior of a DC motor when it enters current saturation. Two monitor functions track the behavior of the DC motor when it enters or leaves current saturation. These monitor functions are given in the following equations:

$$\left(I_a - I_{sat}\right) \geq 0 \tag{2.3.21}$$

$$\left(\dot{I}_a\right) > 0 \tag{2.3.22}$$

where I_a is the armature current and I_{sat} is the saturation current limit. When the motor starts from rest, the armature current starts building up in the circuit. The monitor function given by Equation (2.3.21) has a nonzero flag, and hence it is active till the system reaches saturation. In this situation, the monitor function given by

(2.3.22) has its flag set to zero and is therefore it is inactive. For this phase, the DC motor behavior is described by Equation (2.3.11). When the armature current in the motor hits its saturation limit, monitor function (2.3.21) trips and resets its flag to 0 and sets the flag of monitor function (2.3.22) to 1. At this point the system switches to an alternate set of governing equations which describe the behavior of DC motor under current saturation. These alternate equations are given by

$$\begin{aligned} &\dot{\omega}_m J_m + T_D = -\omega_m B_v + I_{sat} K_T \\ &\dot{I}_a = 0 \end{aligned} \tag{2.3.23}$$

The new monitor function checks the derivative of armature current at every time step to see if the motor is out of saturation. When the motor comes out of saturation, the monitor function (2.3.22) trips, resets its flag, and reactivates the monitor function in (2.3.21). The present set of governing equations is also deactivated, and the original governing equations are reactivated. The integration is restarted from this point. Simulation results depicting the behavior of a DC motor with and without saturation are presented in the next section.

2.3.1.5 Solution Method

The system of equations given by Equation (2.3.5) is solved for the system variables by simultaneously solving direct assignment equations given by Equation (2.3.4). Equation (2.3.5) is solved using a linear system solver, and sparse matrix routines could be used for this purpose. The free differential variables are then integrated numerically using single step algorithms, such as Runge-Kutta, which return the values of the free variables at each time step. We assume that the system behavior is piecewise smooth in time to allow extension to digital devices, so single step ODE solvers are better suited for our approach.

2.4 Examples

2.4.1 DC Motor Driving a Link

Figure (2.4.1) represents a DC motor driving a single link. The system is composed of five components – the DC motor, the gear box, the link, the connection between

the motor and the gearbox, and the connection between gearbox and the link. The system parameter vector, **P,** for this system has the following contributions:

Motor Parameters

J_m = Motor Inertia = 0.0044 oz-in-s^2

L_a = Motor Winding Inductance = 0.0048 mH

R_a = Motor Armature Resistance = 2.4 Ω

V_a = Motor Armature Voltage = 1 V

K_b = Back EMF Constant = 0.0401 v/rad/s

K_T = Torque Constant = 5.6 oz-in/amp

B_v = Viscous Friction = 0.0137 oz-in/rad/s

Gearbox Parameters

R = Gear Ratio = 12.5

Link Parameters

J_L = Link Inertia = 41.077 oz-in-s^2

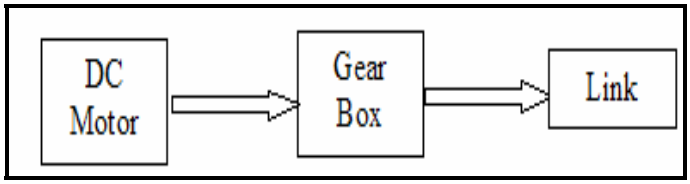


Figure 2.4.1: DC Motor Driving a Link

2.4.1.1 Case 1: No Saturation Effect

2.4.1.1.1 Variable Definition

$$\mathbf{X} = \begin{bmatrix} x_1: \text{motor velocity } (\dot{\theta}_m) \\ x_2: \text{armature current } (I_a) \\ x_3: \text{link velocity } (\dot{\theta}_l) \\ x_4: \text{motor angular rotation } (\theta_m) \\ x_5: \text{link angular rotation } (\theta_l) \end{bmatrix}; \dot{\mathbf{X}} = \begin{bmatrix} \dot{\mathbf{X}}_{free} \\ \dot{\mathbf{X}}_{fixed} \end{bmatrix} \tag{2.4.1}$$

$$\dot{\mathbf{X}} = \begin{bmatrix} \left\{ \begin{array}{l} \dot{x}_1 : \text{motor acceleration } (\ddot{\theta}_m) \\ \dot{x}_2 : \text{derivative of armature current } (\dot{I}_a) \\ \dot{x}_3 : \text{link acceleration } (\ddot{\theta}_l) \end{array} \right\} \\ \left\{ \begin{array}{l} \dot{x}_4 : \text{motor angular velocity}(\dot{\theta}_m) \\ \dot{x}_5 : \text{link angular velocity } (\dot{\theta}_l) \end{array} \right\} \end{bmatrix} \tag{2.4.2}$$

$$\mathbf{Y} = \begin{bmatrix} y_1 : \text{acceleration input to the gearbox } (\ddot{\theta}_{G\_in}) \\ y_2 : \text{acceleration output of the gearbox } (\ddot{\theta}_{Gout}) \\ y_3 : \text{driving torque of the motor } (T_D) \\ y_4 : \text{torque input to the gearbox } (T_{Gin}) \\ y_5 : \text{torque output of the gearbox}(T_{Gout}) \\ y_6 : \text{link torque}(T_L) \end{bmatrix}$$

2.4.1.1.2 Component Governing Equations

a. DC Motor

$$\dot{x}_4 = x_1 \tag{2.4.3}$$

$$\begin{array}{ll} \dot{x}_1 J_m + y_3 = -x_1 B_v + x_2 K_T & \text{(while } \textit{not} \text{ in saturation)} \\ \dot{x}_1 J_m + y_3 = -x_1 B_v + I_{sat} K_T & \text{(while in saturation)} \end{array} \tag{2.4.4}$$

$$\begin{array}{ll} \dot{x}_2 L_a = V_a - x_1 K_b - x_2 R_a & \text{(while } \textit{not} \text{ in saturation)} \\ \dot{x}_2 = 0 & \text{(while in saturation)} \end{array} \tag{2.4.5}$$

b. Gear Box

$$y_2 - (1/R) * y_1 = 0 \tag{2.4.6}$$

$$y_5 - (R) * y_4 = 0 \tag{2.4.7}$$

c. Link

$$\dot{x}_5 = x_3 \tag{2.4.8}$$

$$\dot{x}_3 J_L - y_6 = 0 \tag{2.4.9}$$

d. Connection between Motor and Gearbox

$$y_1 - \dot{x}_1 = 0 \tag{2.4.10}$$

$$y_4 - y_3 = 0 \tag{2.4.11}$$

e. Connection between Gearbox and Link

$$y_2 - \dot{x}_3 = 0 \tag{2.4.12}$$

$$y_6 - y_5 = 0 \tag{2.4.13}$$

2.4.1.1.3 Matrix Form of System Equations

$$\begin{bmatrix} J_m & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & L_a & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & -\frac{1}{R} & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & R & 1 & 0 \\ 0 & 0 & J_L & 0 & 0 & 0 & 0 & 0 & -1 \\ -1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & -1 & 1 & 0 & 0 \\ 0 & 0 & -1 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & -1 & 1 \end{bmatrix} \begin{bmatrix} \dot{x}_1 \\ \dot{x}_2 \\ \dot{x}_3 \\ y_1 \\ y_2 \\ y_3 \\ y_4 \\ y_5 \\ y_6 \end{bmatrix} = \begin{bmatrix} -x_1 B_v + x_2 K_T \\ V_a - x_1 K_b - x_2 R_a \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix} \tag{2.4.14}$$

Equations (2.4.3) and (2.4.8) represent the fixed equations and are solved for fixed differential variables by direct assignments. The system of linear equations combining all of the component equations can be formed as given in Equation (2.4.14) which is solved for the free differential variables and the algebraic variables. The MATLAB solver 'ode45', which is based on a Runge-Kutta algorithm, returns the values of the free variables at each time step. We simulated this system using our approach and compared our results with Simulink. Component model simulation results are shown in Figure (2.4.2), and Figure (2.4.3) shows the Simulink results. Both figures show that the two simulations matched well for this example.

2.4.1.2 Case 2: Saturation Effect

The concept of monitor functions is illustrated below by modeling current saturation for the DC motor used in example 1. The current saturation limit is set to 0.25 A. Given the available armature current, if the system demands more torque than the motor can supply, the appropriate monitor function kicks in at this moment, and the system switches to the constrained model of the DC motor and limits the current. Figure (2.4.4) shows the simulation results for the saturation effect on the DC motor.

At the beginning when a positive armature voltage is applied to the motor, the motor starts from rest at zero speed, and at that instant the armature current shoots up.

When the current hits the saturation limit, the monitor function (2.3.21) goes through a zero, and the motor model switches to an alternate model which is described by governing Equations (2.4.3) and (2.3.23). The state vector at this time step becomes the initial condition vector for the new system of equations. The system uses this new motor model until the current drops below the saturation limit. After the current drops below I_{sat}, Equation (2.3.22) goes through a zero, and the system switches back to the original motor model. Notice that the motor reaches saturation within a very small time. To capture the behavior change during this time, the integration step size is controlled accordingly. This effect is seen in the armature current plot where the current stays flat at 0.25 A until it starts dropping below I_{sat}. The results were compared with Simulink results and they were in good agreement.

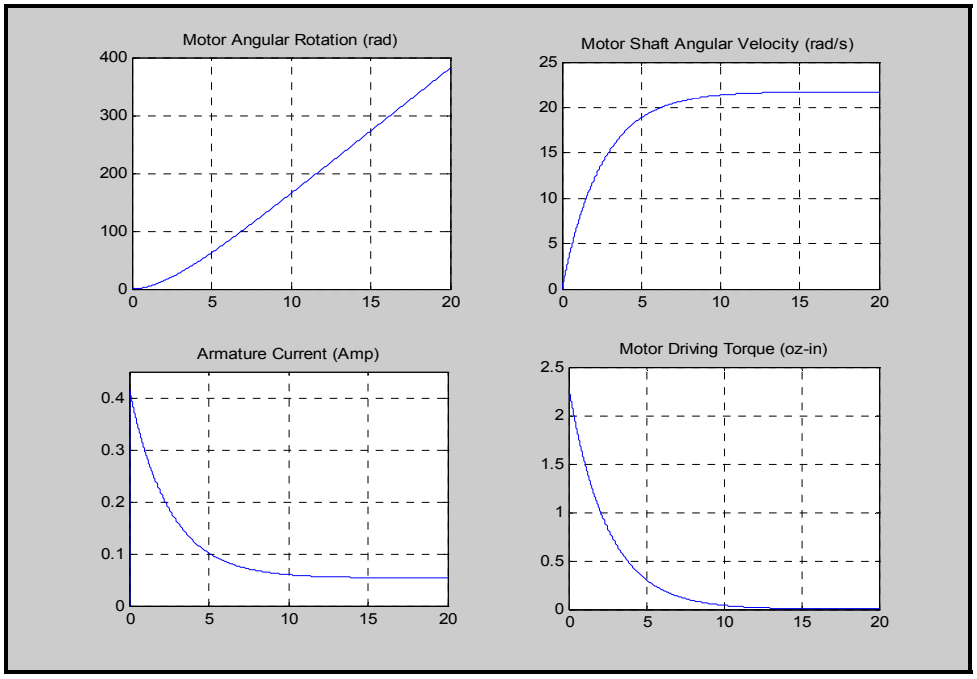


Figure 2.4.2: DC Motor with Link - Component Model

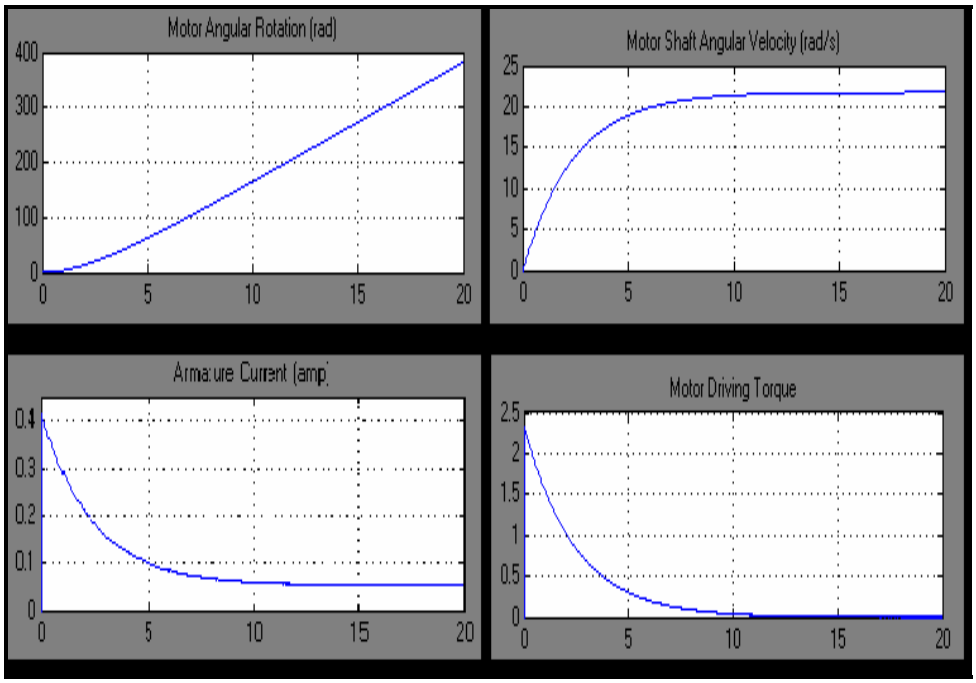


Figure 2.4.3: DC Motor with Link – Simulink

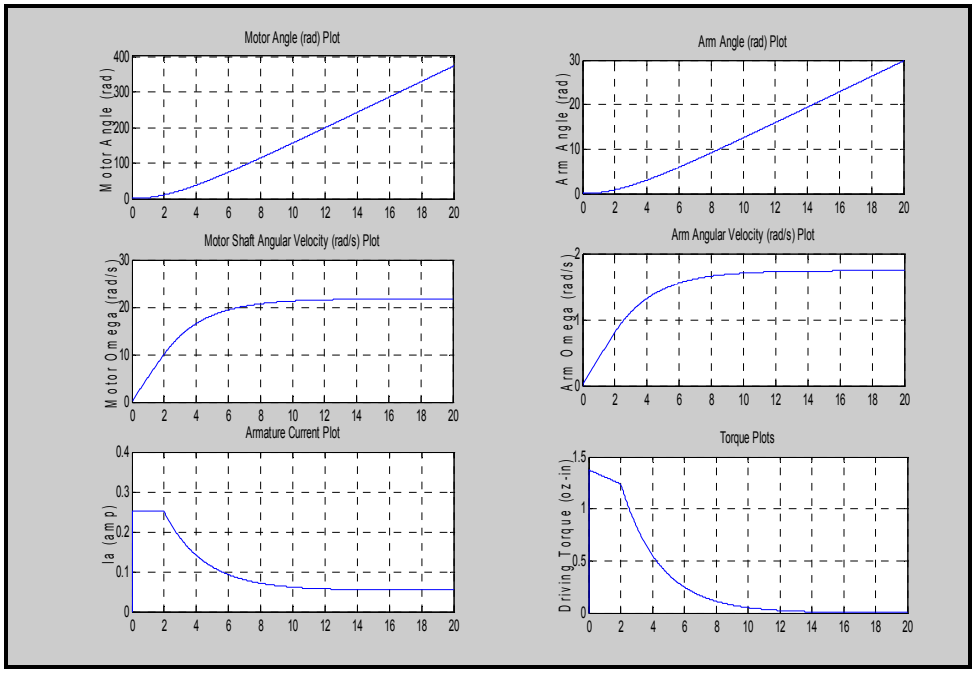


Figure 2.4.4: DC Motor with Link – Current Saturation

2.4.2 Slider-Crank Mechanism

For a nonlinear example, we simulated a motor driven slider crank mechanism which includes both rigid bodies and revolute joints. The system components are a DC

motor, a gear box, two links, two revolute joints, a slider block, a slider constraint and the connections between the components.

2.4.2.1 System Specifications

This slider crank example uses the same motor and gearbox as described in example (2.4.1) with a second link and a slider block added to the system.

System specifications are as listed below.

Specifications of the First Link

L_1 = length of link 1 = 3.0 in

m_1 = mass of link 1 = 0.0041 oz-s^2/in

J_1 = inertia of link 1 = 0.0373 oz-in-s^2

Specifications of the Second Link

L_2 = length of link 2 = 5.0 in

m_2 = mass of link 2 = 0.0041 oz-s^2/in

J_2 = inertia of link 2 = 0.0086 oz-in-s^2

Specifications of the Slider

m_s = slider mass = 0.6211 oz-s^2/in

J_s = slider inertia = 1.0 oz-in-s^2

2.4.2.2 Modeling of a Nonlinear EMS

The system consists of a DC motor (as the crank), a first link, a second link, two revolute joints and a slider constraint. The slider-crank mechanism is shown in Figure (2.4.5). This problem is described by a second order system of differential-algebraic equations which is transformed to a single order system of dimension 28.

2.4.2.3 Types of Components

The slider-crank example uses two types of links. In simulation, we modeled the links using different link models. The DC motor and the first link are modeled the same way as in the first example. The first link is modeled as a fixed-axis body, and

the second link and slider are modeled as rigid bodies. We also make use of two revolute joint components as well as two constraint components to ensure pure translation of the slider.

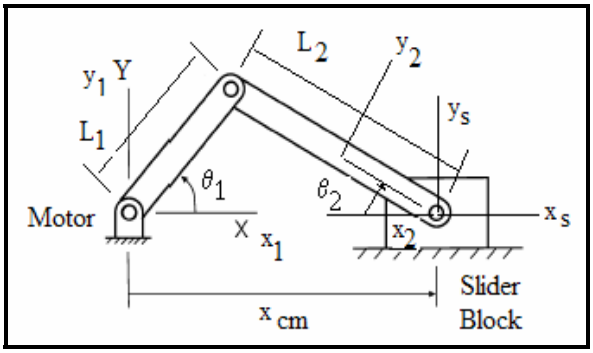


Figure 2.4.5: Slider - Crank Mechanism

2.4.2.4 Modifications in Governing Equations Due to Non-Local Components

The two links are connected through one revolute joint, and the second revolute joint connects the second link to the slider block. The revolute joint connecting the two links modifies the governing equations of both the links, as explained in section 2.

The second revolute joint modifies the governing equations of the second link and the slider. The constraint components modify the slider equations.

2.4.2.5 Case 1: No Saturation Effect

Simulation results of the slider-crank mechanism using the component-based approach are as given in Figure (2.4.6). For this simulation, it is assumed that the system has no friction losses.

Initial Conditions

$\theta_1 = (\pi/2)^c$

$x_{cm2} = 2.0$ in, $y_{cm2} = 1.5$ in (CM of link 2)

$\theta_2 = (-\tan^{-1}(3/4))^c$

$x_{cms} = 4.0$ in, $y_{cms} = 0.0$ in (CM of slider)

The slider is constrained to move only in the *x*-direction, and the simulation shows that it moves between 2″and 8″ as expected.

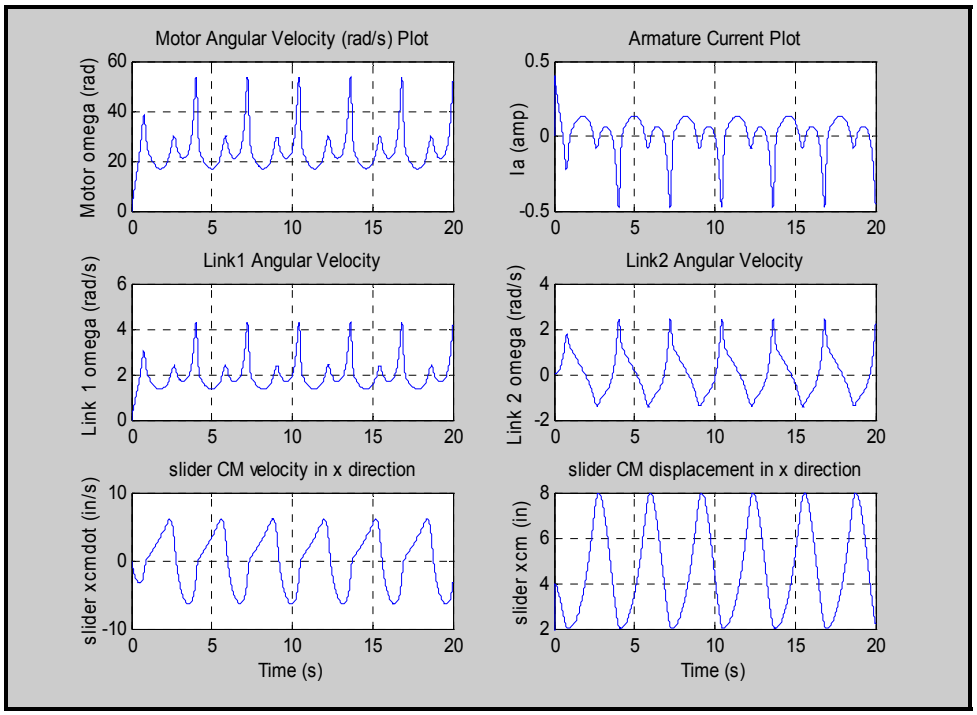


Figure 2.4.6: Slider-Crank Simulation - No Current Saturation

2.4.2.6 Case 2: Saturation Effect

Simulation results of the slider-crank mechanism using the component-based approach with saturation model are as given in Figure (2.4.7).

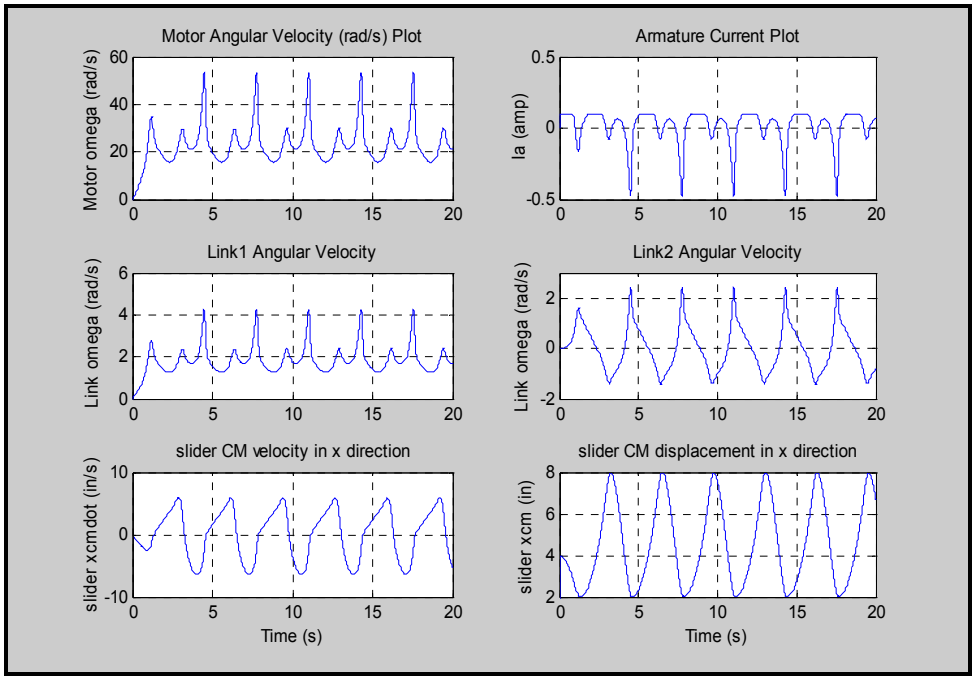


Figure 2.4.7: Slider-Crank Simulation – With Current Saturation

For this simulation the current saturation limit was set to 0.15 A which can be seen in the armature current plot of the figure. Simulation results for the slider-crank mechanism were verified based on analytical "spot checks" and they were in very good agreement. Note that in Figure (2.4.7) the motor saturation slows down the system.

2.5 Concluding Remarks

This paper presented a component-based mathematical approach to model electromechanical systems. In this formulation, an electromechanical system is viewed as a collection of interacting components. Each component is characterized by its own set of parameters and component variables which can appear in the component governing equations in differential or algebraic form. In addition to contributing its own equations, a component can also modify the equations of other components. These component equations are then combined to form a system of DAEs in a simplified form that applies to most electromechanical components. This system of equations is solved using numerical algorithms such as Gaussian elimination, and single step methods based on Runge-Kutta algorithm. The monitor functions approach proposed here is suitable on modeling the qualitative changes in system behavior that occur frequently in electromechanical systems.

The proposed approach was demonstrated using two electromechanical systems. In the first example, a DC motor driving a single link was modeled and the simulation was performed using MATLAB. Our approach was compared with Simulink, and they were in excellent agreement. For a nonlinear case, a slider-crank mechanism was simulated using our approach, and selected results were hand checked. The use of monitor functions was demonstrated by modeling current saturation for the DC motor.

The current formulation may suffer from joint drift if applied to large systems over long time periods. The most popular method to account for this drift is Baumgarte's stabilization technique [2.6.16] which can be incorporated into this formulation. Computational effort is also a concern. Connection components may be used for

variable elimination instead of equation generation, which will help to reduce the number of equations. Use of sparse matrix methods will also improve efficiency.

2.6 REFERENCES

2.6.1 Harashima F., Tomizuka, M., Fukuda, T., '*Mechatronics – "What is it, Why and How?" An Editorial*', IEEE/ASME Transactions on Mechatronics, Vol. 1, No.1, pp. 1-2, March 1996.

2.6.2 Scherrer, M. and McPhee, J., '*Dynamic Modelling of Electromechanical Multibody Systems*', Multibody System Dynamics, 9, pp. 87-115, 2003.

2.6.3 Granda, J.J., '*The Role of Bond Graph Modeling and Simulation in Mechatronic Systems. An Integrated Software Tool: Camp-G, MATLAB-Simulink.*' Proceedings of Mechatronics Conference, Atlanta, GA, pp. 1271-1295, 2000.

2.6.4 Taehyun, S., '*Introduction to Physical System Modeling Using Bond Graphs*', Vetronics Inst. 2nd Annual Workshop Series, pp. 430-434, Dec. 2002.

2.6.5 Greenwood, D.T., '*Principles of Dynamics, Second Edition*', Prentice Hall, 1988.

2.6.6 Carrigan, J., '*Integrated Design and Sensitivity Based Design and Optimization of Nonlinear Controlled Multibody Mechanisms*', Thesis, M.S. Iowa State University, 2003.

2.6.7 Huston, R.L., '*Multibody Dynamics*', Butterworth-Heinemann, 1990.

2.6.8 Giaurgiutiu, V., Lyshevski, S.E., '*Micromechatronics, Modeling, Analysis, and Design with MATLAB*', CRC Press, 2003.

2.6.9 Sinha, R., Paredis, C.J.J., Liang, V-C., Khosla, P.K., '*Modeling and Simulation Methods for Design of Engineering Systems*', Journal of Computing and Information Science in Engg., Volume 1, Issue 1, pp. 84-91, March 2001.

2.6.10 Sass, L., McPhee, J., Schmitke, C., Fisette, P. and Grenier, D., '*A Comparison of Different Methods for Modelling Electromechanical Multibody Systems*', Multibody System Dynamics, 12, pp. 209-250, 2004.

2.6.11 Ogata, K., '*Modern Control Engineering*', Third Edition, Prentice Hall, 1997.

2.6.12 http://www.amesim.com/, IMAGINE Software Inc., acquired on Jan 29, 2005.

2.6.13 Knorr, U.I., '*Electromechanical System Design*', Ansoft Corporation, 2002.

2.6.14 Schulz, S., '*Four Lectures on Differential-Algebraic Equations*', Humboldt University at zu Berlin, June 2003.

2.6.15 Wood, G.D., Kennedy D.C., '*Simulating Mechanical Systems in Simulink with SimMechanics*', www.mathworks.com, 2003.

CHAPTER 3 - DESIGN SENSITIVITY ANALYSIS OF MULTIDISCIPLINARY MULTIBODY SYSTEMS

Manuscript Publication:

Proceedings of Multibody Dynamics 2007 Thematic Conference

June 25-28, 2007, Milano, Italy

Authors:

Prakash Krishnaswami[*], Shilpa A. Vaze[†], and James E. DeVault[†].

Authors' Affiliations:

[*] Department of Mechanical and Nuclear Engineering, Kansas State University, Manhattan, KS 66506.

[†] Department of Electrical and Computer Engineering, Kansas State University, Manhattan, KS 66506.

NOTE: This chapter is largely based on the original publication, however; changes have been made to the manuscript to maintain continuity and consistency in reading.

3.1 Abstract

Design sensitivity information is useful in the design of multidisciplinary multibody systems. It can be used for gradient-based parametric optimization as well as optimal tolerancing. The robustness of the system design can also be improved by using design sensitivity information to perform minimum sensitivity or minimum variability design. Finally, design sensitivity information can be used to generate high fidelity system metamodels that can be useful in the design process, particularly in early stage design.

This paper presents an analytical design sensitivity analysis formulation for dynamic multidisciplinary multibody systems. The first step in devising a design sensitivity analysis formulation is to put together a viable scheme for the dynamics analysis of the class of systems under study. Here, a component-based approach is used for modeling the system. In this approach, the system is viewed as a collection of interconnected components from different domains. Each type of component has its own set of governing equations, which can be derived by applying the mathematical and/or physical principles that are best suited to the domain of that component. These component equations are combined to generate a system of differential-algebraic equations (DAEs). Based on these DAEs, a set of equations in the state design sensitivity coefficients is analytically derived using direct differentiation. These equations are a set of DAEs which can be solved simultaneously with the system governing equations. This integrated scheme of dynamic analysis and sensitivity analysis of multidisciplinary systems is successfully implemented in a symbolic-numeric computational platform called MIXEDMODELS (Multidisciplinary Integrated eXtensible Engine Driving Metamodeling, Optimization and DEsign of Large-scale Systems).

The symbolic-numeric implementation strategy of MIXEDMODELS gives great flexibility in the choice of design variables and in imposing relationships between parameters. The results obtained were checked by perturbation analysis, and the method was shown to be accurate and computationally viable. Representative

examples are presented to demonstrate the capabilities and performance of the proposed design sensitivity analysis formulation.

3.2 Introduction

In recent years, emphasis has been placed on the development of an integrated approach for analysis and design of multidisciplinary multibody systems (MDMSs). Design sensitivity information can be used in several ways in the design of an MDMS. It is a prerequisite for gradient-based optimization of the parametric design of these systems. The robustness of the system design can also be improved by using design sensitivity information to perform minimum sensitivity or minimum variability design. One can also use design sensitivity information as a basis for optimal allocation of manufacturing tolerances. Finally, design sensitivity information can be used to generate high fidelity system metamodels that can be useful in the design process, particularly in early stage design.

The traditional optimization approach for multidisciplinary systems (MDS) utilizes sequential optimization, wherein each subsystem – electrical, mechanical, hydraulic, etc. – is optimized in isolation in a predetermined order, assuming that the designs of the other subsystems remain fixed. This often leads to system designs that are suboptimal [3.7.1-3.7.3]. This is mainly due to the extensive interaction between different domains such as electrical, mechanical, hydraulic, pneumatic etc, that the sequential design approach ignores by decoupling the subsystems. For example, in many multidisciplinary systems, the design variables of the electrical subsystems may strongly affect the performance of the mechanical subsystem; hence the sensitivities of performance functions from the mechanical domain need to be calculated with respect to the design variables in the electrical domain in order to optimize the design of the entire system. This kind of interaction between the subsystems cannot be captured in a sequential design approach. Further, it has been shown in [3.7.1-3.7.3] that integrated optimization techniques for multidisciplinary systems result in better designs in terms of robustness and optimality.

The first step towards devising an integrated optimization scheme for multidisciplinary systems is to develop a unified modeling, simulation and sensitivity

analysis platform. Some analysis tools for MDSs are currently available in the market. However, there are certain disadvantages that limit their range of applicability. One of the existing approaches is to apply a single physical/mathematical formulation such as Lagrangian, Bond Graph Theory, Linear Graph Theory, State Space etc. to derive governing equations of a multidisciplinary system [3.7.4, 3.7.5]. However, these formulations are generally well-suited to particular application domain(s) but are not easily extensible to accommodate new domains. Nevertheless, researchers have been successful in applying this approach in certain domains. NEWOPT/AIMS [3.7.6] is a software package that includes algorithms for simulation, sensitivity analysis and optimization of a class of mechatronic systems based on a linear state space model. Sensitivities are provided by the semi-analytical adjoint variable method, automatic differentiation or finite differences. Similarly, a linear state space formulation for design sensitivity and optimization of controlled mechanical systems is presented in [3.7.1-3.7.3].

Another strategy is to develop different software packages for each relevant problem domain and then integrate these software packages as needed to model an MDS. These domain-specific packages then communicate with each other to simulate the system behavior. MATLAB/Simulink [3.7.7], Simplorer [3.7.8], and AMESim [3.7.9], are some of the existing tools that fall under this category. While this approach provides a working solution in many cases, it is often cumbersome and does not yield an explicit mathematical formulation of the system governing equation which can be used in analytical design sensitivity analysis.

The third approach that is becoming popular can be categorized as a declarative or equation-based approach which emphasizes integration across domains at the level of the system governing equations. An interesting application of this approach in the automotive field is presented in [3.7.10], where the authors use this strategy to globally model, simulate, and optimize complex industrial mechatronic systems using MATLAB-Simulink and a Finite Element Method. In both these approaches, sensitivities are calculated using finite differences. However, the authors conclude from their studies that the optimization process can be greatly improved by adopting an integrated modeling approach that could support semi-analytical design sensitivity

analysis. The equation level integration approach, also called the component approach, is also the central idea behind the Modelica object-oriented specification language for multidisciplinary systems [3.7.11, 3.7.12]. It has been shown in [3.7.13, 3.7.14] that this approach can be used to develop an effective analysis and solution tool for MDS modeling using a mixed symbolic-numeric software architecture.

In our previous work [3.7.13, 3.7.14] we presented a linear formulation scheme which can be applied more generally to include disciplines such as mechanical, electrical/electronic, hydraulic, pneumatic etc. Expanding on the work presented in [3.7.13, 3.7.14], this paper presents a general linear formulation for analysis and analytical design sensitivity analysis of multidisciplinary systems that has been successfully implemented in the MIXEDMODELS (Multidisciplinary Integrated eXtensible Engine for Driving Metamodeling, Optimization and DEsign of Large-scale Systems) platform. MIXEDMODELS is a unified analysis and design tool for multidisciplinary systems that utilizes a component-based formulation [3.7.15]. It is a flexible, extensible, and compact platform that is based on a procedural, symbolic-numeric software architecture. The sensitivity analysis formulation discussed in this paper uses the direct differentiation approach, which was preferred due to its simplicity of implementation and ease of error control.

3.3 Dynamic Analysis of Multidisciplinary Multibody Systems in the MIXEDMODELS Platform

The mathematical formulation of MIXEDMODELS as presented in [3.7.13] uses a strictly acausal, local/global approach where a multidisciplinary system is considered to be a collection of interconnected components.

A multidisciplinary system can be completely described by a vector of time-invariant system parameters, **P**, which may depend on the design variables; a vector of system variables, **X**, which can also occur in first derivative form, $\dot{\mathbf{X}}$ in the governing DAEs; a vector of algebraic system variables, **Y,** that occur algebraically in the governing DAEs; and a set of governing equations (DAEs) written in the following form:

$$\mathbf{f}(\mathbf{P}(\mathbf{d}), \mathbf{X}(\mathbf{d},t), \dot{\mathbf{X}}(\mathbf{d},t), \mathbf{Y}(\mathbf{d},t), \mathbf{d}, t) = \mathbf{0} \tag{3.3.1}$$

where **d** is the vector of design variables. Equation (3.3.1) can always be converted to an equivalent matrix form, possibly by differentiation of Equation (3.3.1) with respect to time. This matrix form can be written as

$$\hat{\mathbf{A}}\left(\mathbf{P}, \mathbf{X}, \mathbf{d}, t\right)\begin{bmatrix}\dot{\mathbf{X}} \\ \mathbf{Y}\end{bmatrix} = \hat{\mathbf{b}}\left(\mathbf{P}, \mathbf{X}, \mathbf{d}, t\right) \tag{3.3.2}$$

Most multidisciplinary systems are described by higher order differential equations. An n^{th} order differential equation must be reduced to n first order differential equations by defining $(n - 1)$ additional variables. These new variables introduced to the system represent the lower order derivatives, referred to as "lower-order variables", and are part of the **X** vector. Thus, the first derivatives of these additional variables can always be obtained directly from other elements in the **X** vector by assignment. The **X** vector can be partitioned as

$$\mathbf{X} = \begin{bmatrix}\mathbf{X_H} \\ \mathbf{X_L}\end{bmatrix} \tag{3.3.3}$$

where $\mathbf{X_H}$ represents the subvector of **X** such that the derivatives of the elements of $\mathbf{X_H}$ are not contained in **X**. Similarly, $\mathbf{X_L}$ represents the subvector of **X** containing all the elements of **X** such that their derivative is also an element of **X**, i.e., the lower-order variables. Thus, the lower-order variables can be calculated by a set of direct assignments of the form

$$\dot{\mathbf{X}}_\mathbf{L} = \mathbf{X}' \tag{3.3.4}$$

where $\mathbf{X}'$ is a sub-vector of **X**. Separating Equation (3.3.4) from the system of equations, the rest of the system governing equations can be written as

$$\mathbf{A}\left(\mathbf{P}, \mathbf{X}, \mathbf{d}, t\right)\begin{bmatrix}\dot{\mathbf{X}}\mathbf{H} \\ \mathbf{Y}\end{bmatrix} = \mathbf{b}\left(\mathbf{P}, \mathbf{X}, \mathbf{d}, t\right) \tag{3.3.5}$$

In order to support the above formulation at the system level, we require the following at the component level:

- A vector of time-invariant component parameters, $\mathbf{p}^i$; ∋ $\mathbf{p}^i \in \mathbf{P}$, where **P** forms the system parameter vector **P**.
- A vector of transient component differential variables, $\mathbf{x}^i$ ∋ $\mathbf{x}^i \in \mathbf{X}$, where **X** is the system differential variable vector. The $\mathbf{x}^i$ occur in the governing DAEs in first derivative form $\dot{\mathbf{x}}^i$.
- A vector of component algebraic variables, $\mathbf{y}^i$, which occur algebraically in the DAEs ∋ $\mathbf{y}^i \in \mathbf{Y}$, where **Y** forms the system algebraic variable vector **Y**.
- A set of component governing equations which can be expressed by

$$\mathbf{A}^c\left(\mathbf{P},\mathbf{X},\mathbf{d},t\right)\begin{bmatrix}\dot{\mathbf{X}}_{\mathbf{H}}\\ \mathbf{Y}\end{bmatrix}=\mathbf{b}^c\left(\mathbf{P},\mathbf{X},\mathbf{d},t\right) \tag{3.3.6}$$

- Direct assignments for lower order variables at component level can be written as

$$\dot{\mathbf{X}}^c_{\mathbf{L}} = \mathbf{X}^{c'} \tag{3.3.7}$$

- Modification matrices $\mathbf{A}^m$ and $\mathbf{b}^m$ which allow a component to modify governing equations of other components. Modification matrices can be zero if a component does not modify the governing equations of any other component.

A component can introduce new variables to the system or it can be described in terms of variables of other components. Similarly, it may contribute new equations to the system and/or modify equations contributed by other components. Finally, the contribution of a particular component *i* to the system governing equations can be written in the form

$$\begin{aligned}\mathbf{A}\left(\mathbf{P},\mathbf{X},\mathbf{d},t\right)&=\mathbf{A}\left(\mathbf{P},\mathbf{X},\mathbf{d},t\right)+\mathbf{A}_i^m\left(\mathbf{P},\mathbf{X},\mathbf{d},t\right)+\mathbf{A}_i^c\left(\mathbf{P},\mathbf{X},\mathbf{d},t\right)\\ \mathbf{b}\left(\mathbf{P},\mathbf{X},\mathbf{d},t\right)&=\mathbf{b}\left(\mathbf{P},\mathbf{X},\mathbf{d},t\right)+\mathbf{b}_i^m\left(\mathbf{P},\mathbf{X},\mathbf{d},t\right)+\mathbf{b}_i^c\left(\mathbf{P},\mathbf{X},\mathbf{d},t\right)\end{aligned} \tag{3.3.8}$$

where

A (**P**,**X**,**d**,*t*) is the global matrix in the governing equations of Equation (3.3.2),

$\mathbf{A}^c_i$ (**P**,**X**,**d**,*t*) is the contribution of component *i* to the global matrix via new equations contributed by this component,

$\mathbf{A}^m_i$ ($\mathbf{P}$,$\mathbf{X}$,$\mathbf{d}$,t) is the contribution of component i to the global matrix via modifications caused by this component to equations contributed by other components,

$\mathbf{b}^c_i$ ($\mathbf{P}$,$\mathbf{X}$,$\mathbf{d}$,t) is the contribution of component i to the global RHS vector via new equations contributed by this component,

$\mathbf{b}^m_i$ ($\mathbf{P}$,$\mathbf{X}$,$\mathbf{d}$,t) is the contribution of component i to the global RHS vector via modifications caused by this component to equations contributed by other components,

$\mathbf{P}$ is a vector of time-independent parameters that describe each component in the system,

$\mathbf{X}$ is the vector of system differentiable variables, and

$\mathbf{d}$ is the vector of design variables.

Thus, a component model is completely described by specifying the entries in the $\mathbf{A}^c$ and $\mathbf{A}^m$ matrices and in the $\mathbf{b}^c$ and $\mathbf{b}^m$ vectors contributed by that component. Once we have the component contributions, they can be combined to obtain the system governing equations as a set of differential-algebraic equations (DAEs) given by Equations (3.3.4) and (3.3.5).

The proposed formulation offers great modeling flexibility at the component level and accommodates component models of different levels of complexity. Extension of this formulation to new problem domains is easy and straightforward. The proposed formulation scheme requires a software architecture that automatically assembles contributions of individual components in a system of equations and numerically solves it to generate the system response. The following section presents a design sensitivity analysis technique that is based on the above formulation and has been implemented in the symbolic-numeric software architecture of the MIXEDMODELS platform [15].

3.4 Design Sensitivity Analysis Formulation in the MIXEDMODELS Platform

Based on the above formulation for the system analysis, we can now derive a formulation for design sensitivity analysis. In general, for nonlinear transient dynamic systems, there are two possible approaches for analytical design sensitivity analysis: the direct differentiation approach and the adjoint variable approach. The adjoint variable approach has the potential to reduce the computational effort, but it requires integrating the adjoint equations backwards in time. This can cause serious difficulties in error control and significant complexities in software implementation. Therefore, we choose the direct differentiation approach for our development here.

It is assumed that the performance functions of interest in the system are of the form

$$g_i\left(\dot{\mathbf{X}}(\mathbf{d},t),\mathbf{X}(\mathbf{d},t),\mathbf{Y}(\mathbf{d},t),\mathbf{d},t\right), \quad i=1,...,n_g \tag{3.4.1}$$

where n_g is the number of performance functions.

From this, we deduce that the sensitivity vector corresponding to a particular performance function g_i is given by

$$\left(g_i\right)_{\mathbf{d}} = \left(g_i\right)_{\mathbf{d}|\exp} + \left(g_i\right)_{\dot{\mathbf{X}}} \dot{\mathbf{X}}_{\mathbf{d}} + \left(g_i\right)_{\mathbf{X}} \mathbf{X}_{\mathbf{d}} + \left(g_i\right)_{\mathbf{Y}} \mathbf{Y}_{\mathbf{d}} \tag{3.4.2}$$

where a vector subscript denotes partial differentiation with respect to the subscript and $(g_i)_{\mathbf{d}|\exp}$ represents the explicit partial derivative of g_i with respect to $\mathbf{d}$.

In the above expression for the sensitivity vector of g_i, the derivatives of g_i can be directly obtained from the given form of the performance functions by differentiation in the MIXEDMODELS symbolic engine. However, the state sensitivity coefficients $\dot{\mathbf{X}}_{\mathbf{d}}$, $\mathbf{X}_{\mathbf{d}}$, and $\mathbf{Y}_{\mathbf{d}}$ need to be evaluated numerically by solving a suitable set of equations. Such a set of equations can be obtained by differentiating the system governing equations given by Equations (3.3.4) and (3.3.5) with respect to the design vector $\mathbf{d}$. Based on the $\mathbf{X}$-partition given by Equation (3.3.3), we can similarly partition $\mathbf{X}_{\mathbf{d}}$:

$$\mathbf{X}_{\mathbf{d}} = \begin{bmatrix} \mathbf{X}\mathbf{H}_{\mathbf{d}} \\ \mathbf{X}\mathbf{L}_{\mathbf{d}} \end{bmatrix} \tag{3.4.3}$$

Then, the state sensitivities $\dot{\mathbf{X}}_{\mathbf{L_d}}$ can be easily evaluated from the direct assignments given by

$$\dot{\mathbf{X}}_{\mathbf{L_d}} = \mathbf{X}'_{\mathbf{d}} \tag{3.4.4}$$

where $\mathbf{X'_d}$ is a subvector of $\mathbf{X_d}$. Further, a set of DAEs in the sensitivities $\dot{\mathbf{X}}_{\mathbf{H_d}}$ and $\mathbf{Y_d}$ can be obtained by differentiating Equation (3.3.5) as follows:

$$\begin{aligned}\mathbf{A}(\mathbf{P},\mathbf{X},\mathbf{d},t)\begin{bmatrix}\dot{\mathbf{X}}_{\mathbf{H_d}}\\ \mathbf{Y_d}\end{bmatrix} \;=\; & \left(\mathbf{b}(\mathbf{P},\mathbf{X},\mathbf{d},t)\right)_{\mathbf{d}|\exp} + \left(\mathbf{b}(\mathbf{P},\mathbf{X},\mathbf{d},t)\right)_{\mathbf{X}}\mathbf{X_d} \\ & -\left(\mathbf{A}(\mathbf{P},\mathbf{X},\mathbf{d},t)\begin{bmatrix}\tilde{\dot{\mathbf{X}}}_{\mathbf{H}}\\ \tilde{\mathbf{Y}}\end{bmatrix}\right)_{\mathbf{d}|\exp} - \left(\mathbf{A}(\mathbf{P},\mathbf{X},\mathbf{d},t)\begin{bmatrix}\tilde{\dot{\mathbf{X}}}_{\mathbf{H}}\\ \tilde{\mathbf{Y}}\end{bmatrix}\right)_{\mathbf{X}}\mathbf{X_d}\end{aligned} \tag{3.4.5}$$

where the ~ over a quantity implies that it is held constant for the partial differentiation indicated. The explicit partial derivatives with respect to **d** are necessary because the time-independent parameter vector **P** may depend on the design vector **d**. The above equation represents another system of DAEs, which must be solved numerically to obtain the state sensitivity information. Unlike the adjoint variable method, however, this system of DAEs is to be solved forward in time, not backward in time. Thus, we can solve the system governing equations and state sensitivity equations simultaneously, which allows effective error control during the numerical solution process. Furthermore, it should be noted that the coefficient matrix for the system governing equations is the same as that for the state sensitivity equations, making the solution process quite efficient.

The ODEs in these equations can be solved by any suitable ODE solver. The current implementation of MIXEDMODELS uses a stiff integrator DLSODES based on a sparse matrix version of Gear's algorithm and a linear sparse matrix solver Y12MAF. The dependent variables seen by the ODE solver are now the system variables **X** as well as their sensitivities $\mathbf{X_d}$.

3.4.1 Evaluating Initial Conditions

In order to start the integration from the given initial time, initial conditions must be provided on all the differential variables that we wish to find. Specifically, initial conditions must be given not only on **X,** but also on $\mathbf{X_d}$. In specifying initial conditions, care must be taken to ensure that the specified initial conditions are physically realizable and consistent with all system constraints, such as loop closure conditions on a mechanical system. Thus, it may be difficult for the user to manually supply a consistent set of initial conditions on **X** for a large system; if the user is further called upon to provide initial conditions on $\mathbf{X_d}$, the task is even more burdensome. To ensure consistency of initial conditions, we assume that the user provides a set of consistency equations that must be satisfied by the initial conditions. These equations are assumed to be of the form

$$\mathbf{h}(\mathbf{P}, \mathbf{X}(\mathbf{d}, t), \mathbf{d}, t) = \mathbf{0} \tag{3.4.6}$$

where **h** is a vector of nonlinear algebraic equations of dimension (N+L) that can be solved by Newton-Raphson iteration to obtain a consistent initial conditions vector, **X**. Here, N is the number of higher-order differential variables and L is the number of lower-order differential variables. All that the user needs to provide is a set of initial guesses for **X**, which are then corrected by the iteration. In order to compute initial conditions on sensitivities, we differentiate Equation (3.4.6) with respect to the design variable vector **d**. The initial condition vector $\mathbf{X_d}$ for sensitivities can then be obtained by solving the resulting Equation (3.4.7) as a linear system of algebraic equations.

$$\mathbf{h_X X_d} = -\mathbf{h_{d|exp}} \tag{3.4.7}$$

Since this is a system of linear equations, the user does not have to provide initial guesses for $\mathbf{X_d}$ values – the correct initial conditions are calculated directly from Equation (3.4.7). Once again, we can take advantage of the fact that the coefficient matrix in Equation (3.4.7) is the same as the Jacobian matrix used in the Newton-Raphson iteration for solving Equation (3.4.6).

3.5 Numerical example

3.5.1 Slider-Crank Mechanism

As an example, we simulate and compute design sensitivities for a slider-crank mechanism driven by a DC power supply Figure (3.5.1). The system has 16 components: one DC Motor, three Rigid Bodies (Crank, Connecting Rod, Slider), two Revolute Joints, one DC Voltage Source, one Electric Nodes, one Analog Ground Component, one Slider Constraint, one Fixed-Axis Body Constraint, two Mechanical Connections (torque and acceleration connection between the motor and crank), and two Performance Functions.

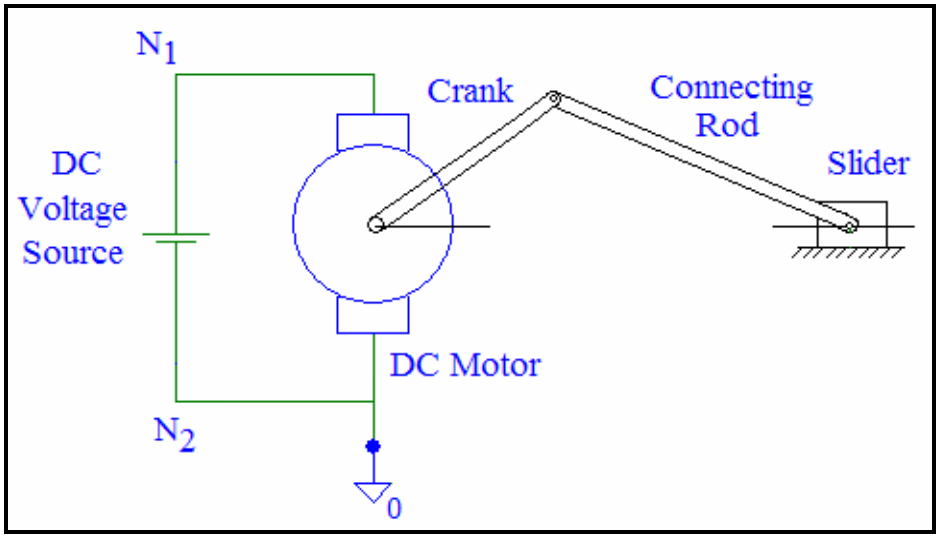


Figure 3.5.1: Slider-Crank Mechanism Driven by a DC Voltage Source

The crank is modeled as a fixed-axis body by adding constraint components. The entire mechanism is assumed to be at rest with a crank angle of 45 degrees when the DC power is turned on; thus all the currents and voltages in the system are zero at the initial time. The specifications of the system are given in Figure (3.5.2).

3.5.2 System Specifications

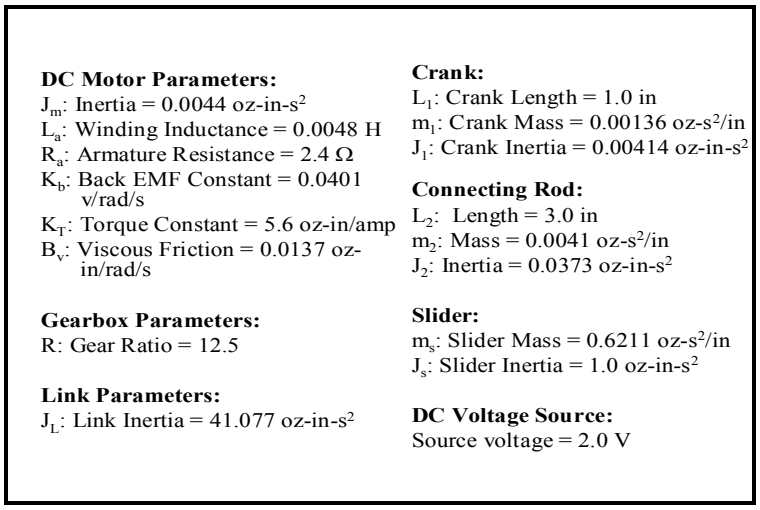


Figure 3.5.2: Slider-Crank Mechanism Specifications

3.5.3 Design Variables

To verify the validity of our approach, we chose the viscous friction coefficient of the motor and the moment of inertia of the connecting rod as the two design variables in the system. The system sensitivities are calculated with respect to each design variable using the method in the previous section. The perturbation check on the sensitivity analysis is done individually for each variable, i.e., we only introduce perturbation in one variable at a time. The nominal and perturbed values of the design variables are as given below.

$\mathbf{d}$ = [viscous friction coefficient, moment of inertia of the connecting rod]

$\mathbf{d}_{\mathbf{original}}$ = [0.0137, 0.0373], $\mathbf{d}_{\mathbf{perturbed}}$ = [0.0.0147, 0.0383]

3.5.4 Performance Functions

For this example we use two types of performance functions: a grid point function which is evaluated at every time instant of interest (also called a grid point) and an integral function which is an integrated value over the entire simulation interval. Instantaneous motor power is chosen to be the grid point performance function imposed at every 0.1 s, and the total energy consumption of the motor is treated as an integral performance function as given below:

$$g_1 = V_c I_a \tag{3.5.1}$$

$$g_2 = \int V_c I_a \, dt \tag{3.5.2}$$

3.5.5 Perturbation Analysis

Design sensitivity calculations are verified by perturbation analysis using finite differences. Let d_1 be the design variable of interest, let Δd_1 denote the perturbation to d_1, and let the response variable of interest be z_1. Before adding a perturbation to d_1, we calculate the response z_1 over the simulation interval. With all other design variables unchanged, we introduce a perturbation Δd_1 to d_1 such that

$$d_1 \rightarrow d_1 + \Delta d_1 \tag{3.5.3}$$

The system is then simulated to get the perturbed response z'_1. Then the actual change in the two responses can be calculated at any time t as

$$\Delta z_1(\text{actual}) = z_1 - z_1' \tag{3.5.4}$$

At any time t, the corresponding change in the response variable z_1 can also be estimated using the sensitivity information as follows:

$$\Delta z_1(\text{predicted}) = \frac{\partial z_1}{\partial d_1} \Delta d_1 \tag{3.5.5}$$

We can now compare Δz_1(actual) with Δz_1(predicted) to assess the accuracy of the sensitivity coefficient in Equation (3.5.5). For this check, we can also plot Δz_1(actual) and Δz_1(predicted) as functions of time. A similar check can be done on any performance function.

3.5.6 Simulation Results and Discussion

The example described earlier was simulated using the MIXEDMODELS platform and the results are summarized here. Figure (3.5.3) shows the response of the slider-crank at the reference design. The left column shows the plots of motor angular velocity, armature current and slider velocity as functions of time. In the second and third graphs, the right hand column shows slider acceleration and slider

displacement. The plot in the upper right hand corner is a detailed view of the slider acceleration before the spike in acceleration, showing details that are invisible due to the scaling in the figure below it.

3.5.6.1 Results with the Coefficient of Viscous Friction of the Motor as Design Variable

Figure (3.5.4) and Figure (3.5.5) present the perturbation analysis check with slider acceleration and armature current as the response variables, and the coefficient of viscous friction of the motor as the design variable. Both plots show that the predicted change obtained using the sensitivity analysis is in close agreement with the actual change found by finite difference analysis – in fact, in all cases the two plots are on top of each other. In both figures, the second plot captures the details before the spike. It can be seen that the results are in a very good agreement even during the spike in acceleration. In both figures, the second plot is a detailed view of the region before the first acceleration spike.

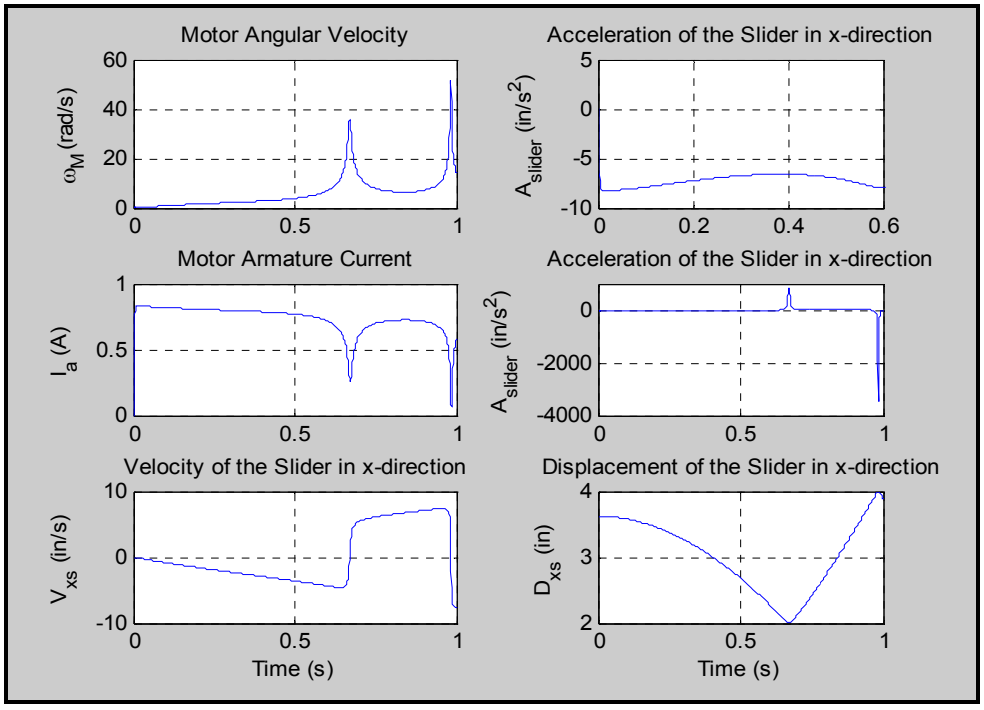


Figure 3.5.3: Slider-Crank Simulation Response

For the integral performance function for the total energy, the actual change found by finite differences was 3.087 e^{-4}J, whereas the predicted change was 3.05942 e^{-4}J, giving an absolute error between the predicted and actual change of 2.758 e^{-7} J.

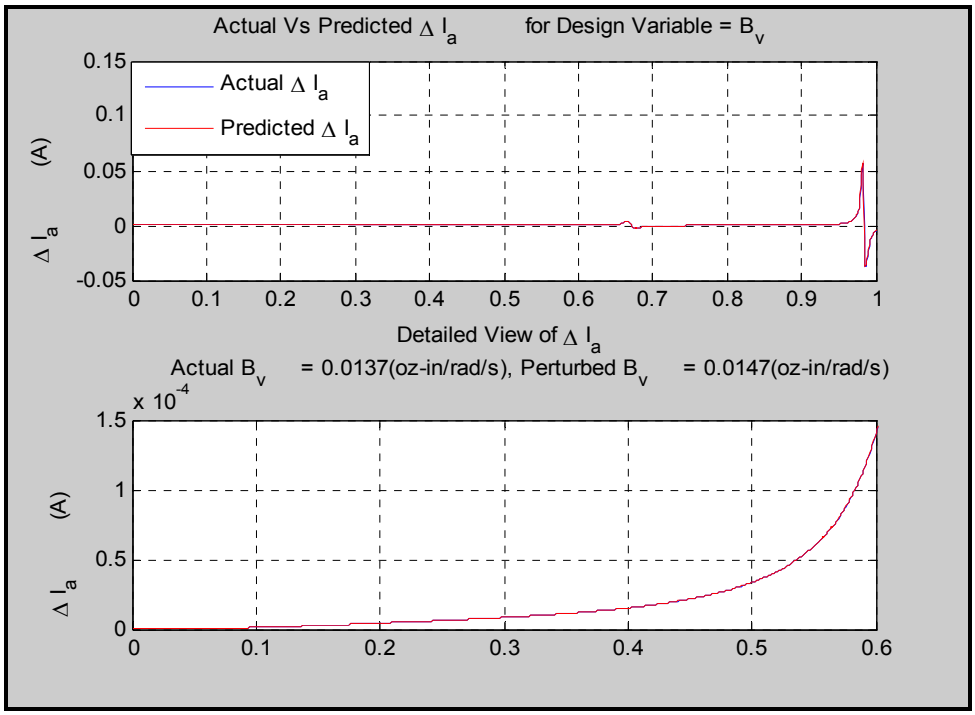


Figure 3.5.4: Actual Versus Predicted Change in Armature Current (Design Variable – Viscous Friction)

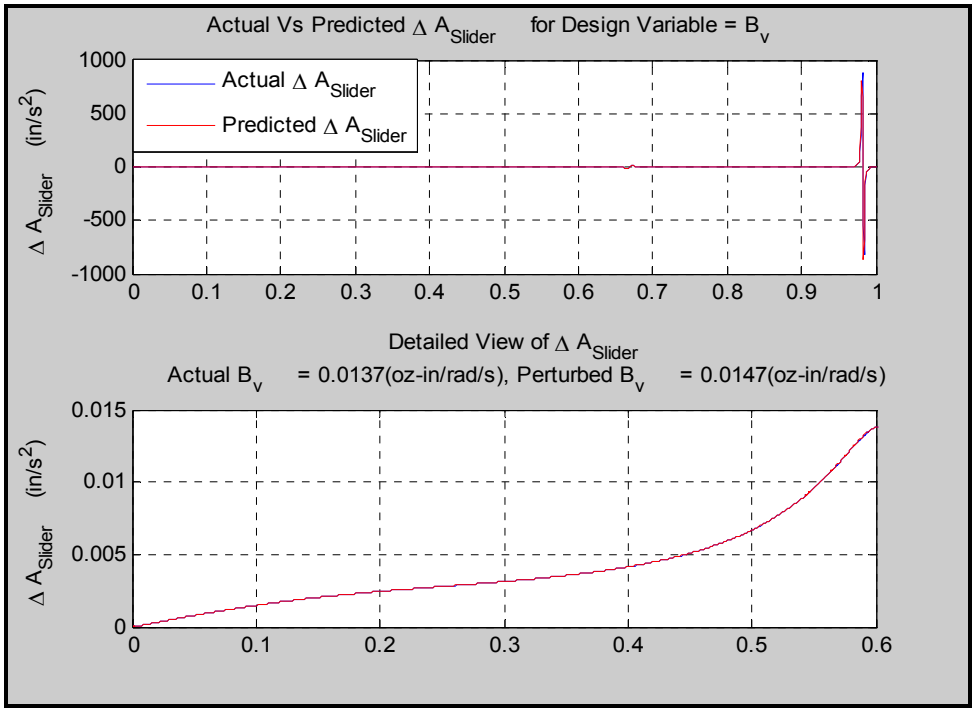


Figure 3.5.5: Actual Versus Predicted Change in Slider Acceleration (Design Variable – Viscous Friction)

This is an error of less than 1%. For the performance function on instantaneous power, the finite difference test results are shown in the Table (3.5.1).

From the table we see that the predicted and actual changes in instantaneous power for the reported times are in pretty close agreement. Maximum absolute error reported during this analysis was $3.1534e^{-4}$ W, which is an error of about 3.5%.

Time (s)	Actual Change in Instantaneous Power (W)	Predicted Change in Instantaneous Power (W)	Relative Difference (%)
0.1	2.653230E-06	2.6533900E-06	6.03117657E-03
0.2	8.826900E-06	8.8278349E-06	1.05901758E-02
0.3	1.688762E-05	1.6890262E-05	1.56439172E-02
0.4	3.048311E-05	3.0490789E-05	2.51850890E-02
0.5	6.683475E-05	6.6867008E-05	4.82422858E-02
0.6	2.812237E-04	2.8160780E-04	1.36404391E-01
0.7	-7.334711E-04	-7.2998800E-04	-4.7714174E-01
0.8	1.647975E-04	1.6518987E-04	2.37511745E-01
0.9	6.833143E-04	6.8661214E-04	4.80311690E-01
1	-9.296271E-03	-8.9809260E-03	-3.5112796E+00

Table 3.5.1: Actual Versus Predicted Change in Power

(Design Variable – Viscous Friction)

3.5.6.2 Results Obtained with Connecting Rod Moment of Inertia as the Design Variable

Figure (3.5.6) and Figure (3.5.7) show the perturbation analysis results with armature current and slider acceleration as the response variables, and the moment of inertia of the connecting rod as the design variable. Both plots show good agreement between the predicted change and actual change. It can also be inferred from these figures that the motor armature current is more sensitive to the friction coefficient than to connecting rod inertia. Once again, the second plot in each figure is a detailed view of the region before the first acceleration spike.

Comparing the actual and predicted change in instantaneous power given in Table (3.5.2), we can see that there is a good match in the results. The sudden rise in the predicted and actual change in the value of this performance function near t=0.7s and t=1.0s in both the tables is due to the proximity of the slider to its dead center. The maximum absolute error reported for this case at the corresponding grid points is approximately 1.58%. At other grid points, when the slider is not close to either dead

center, the error between the predicted and actual change is less than 1% for both design variables.

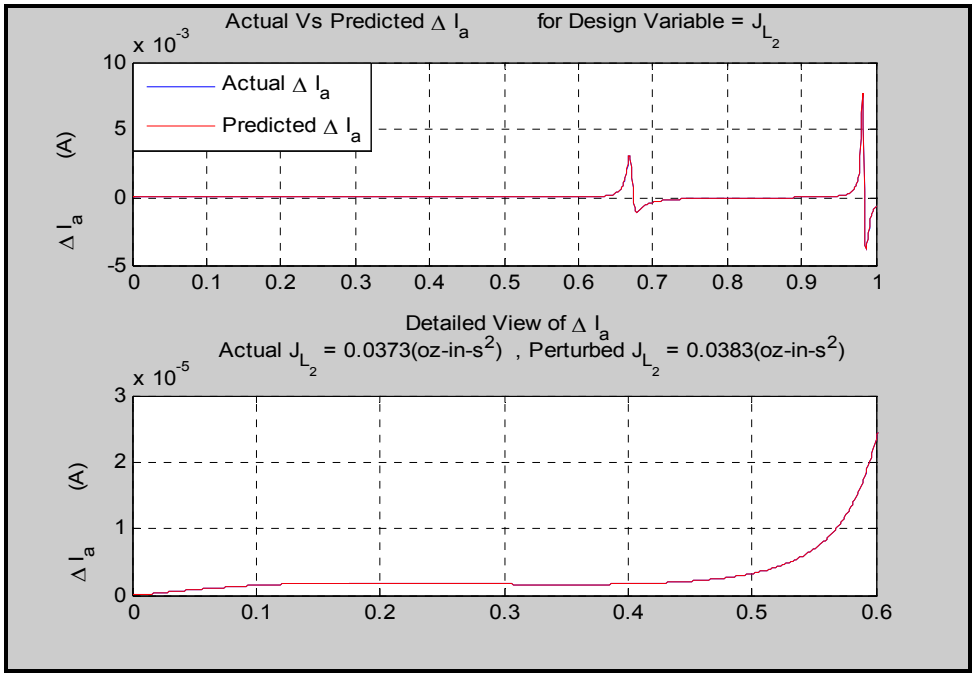


Figure 3.5.6: Actual Versus Predicted Change in Armature Current (Design Variable – Moment of Inertia)

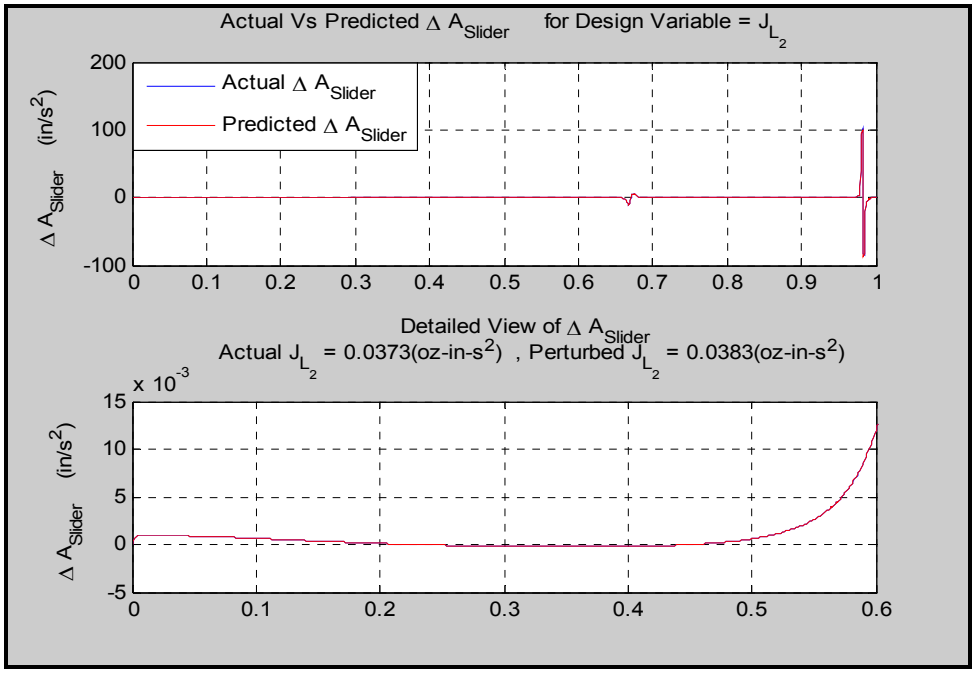


Figure 3.5.7: Actual Versus Predicted Change in Slider Acceleration (Design Variable – Moment of Inertia)

For the integral performance function, the total energy for the perturbed case is found to be 1.4665 J. The predicted change in the performance function based on the calculated sensitivities, and the actual change obtained by reanalysis at the perturbed design, are $4.1015e^{-5}$ J and $4.0897e^{-5}$ J, respectively. This represents an error of $1.188e^{-7}$ J, which is less than 1%. It was observed from the results that both performance functions show a higher sensitivity to the coefficient of friction as compared to the moment of inertia of the connecting rod.

Time (S)	Actual Change in Instantaneous Power (W)	Predicted Change in Instantaneous Power (W)	Relative Difference (%)
0.1	2.957720E-06	2.9579909E-06	9.15952418E-03
0.2	3.730820E-06	3.7309621E-06	3.80969493E-03
0.3	3.337290E-06	3.3373106E-06	6.17414149E-04
0.4	3.395060E-06	3.3951373E-06	2.27697768E-03
0.5	6.513020E-06	6.5137579E-06	1.13288759E-02
0.6	4.694290E-05	4.6963003E-05	4.28059482E-02
0.7	-7.133776E-04	-7.1333116E-04	-6.50866180E-03
0.8	-8.360055E-05	-8.3712255E-05	1.33438690E-01
0.9	1.305574E-05	1.2852473E-05	-1.58153746E+00
1	-1.291279E-03	-1.2847674E-03	-5.06808830E-01

Table 3.5.2: Actual Versus Predicted Change in Power

(Design Variable – Moment of Inertia)

3.6 Conclusion

This paper presented the MIXEDMODELS formulation for dynamic analysis and sensitivity analysis of multidisciplinary systems. The design sensitivity formulation uses the direct differentiation approach and is implemented using a symbolic-numeric implementation in the MIXEDMODELS platform. The architecture gives great flexibility in the choice of design variables and in imposing relationships between parameters. The results were checked by perturbation analysis, and the method was shown to be accurate and computationally viable. A representative example of a slider-crank mechanism powered by a DC voltage source is presented to demonstrate the capabilities and performance of the proposed design sensitivity analysis formulation. The results obtained indicate that the method is accurate, computationally viable and applicable to a broad class of multidisciplinary systems.

3.7 REFERENCES

3.7.1 Carrigan, J., Kelkar, A. and Krishnaswami, P., '*Minimum Sensitivity Design of Controlled Multibody Systems*', ASME Multibody Systems, Nonlinear Dynamics, and Control Conference, 2005.

3.7.2 Kelkar, A., Krishnaswami, P., '*Multidisciplinary Optimization of Multibody Systems*', Proceedings of the ECCOMAS Conference on Multibody Systems, June 2005.

3.7.3 Carrigan, J., '*Integrated Design and Sensitivity Based Design and Optimization of Nonlinear Controlled Multibody Mechanisms*', Thesis, M.S. Iowa State Univ., 2003.

3.7.4 Sinha, R., Paredis, C.J.J., Liang, V-C., Khosla, P.K., '*Modeling and Simulation Methods for Design of Engineering Systems*', Journal of Computing and Information Science in Engineering, Volume 1, Issue 1, pp. 84-91, 2001.

3.7.5 Sass, L., McPhee, J., Schmitke, C., Fisette, P. and Grenier, D., '*A Comparison of Different Methods for Modelling Electromechanical Multibody Systems*', Multibody System Dynamics, 12: pp. 209-250, 2004.

3.7.6 Dignath, F., Breuninger, C., Eberhard, P., Kübler, L., '*Optimization of Mechatronic Systems Using the Software Package NEWOPT/AIMS*', Multibody System Dynamics, pp. 85-100, 2005.

3.7.7 Ram, O., Szymkat, M., Uhl, T., Betemps, M., Pjetursson, A. and Rod, J., '*Mechatronic Blockset for Simulink – Concept and Implementation*', Proceedings of the 1996 IEEE Intl. Symposium on Computer-Aided Control System Design, pp. 530-535, 1996.

3.7.8 Knorr, U.I., '*Electromechanical System Design*', Ansoft Corporation, 2002.

3.7.9 http://www.amesim.com/, IMAGINE Software Inc., acquired on Jan 29, 2005.

3.7.10 Duysinx, P., Bruls, O., Collard, J-F., Fisette, P., Lauwerys, C., Swevers, J., '*Optimization of Mechatronic Systems: Application to a Modern Car Equipped with a Semi-active Suspension*', 6th World Congress of Structural and Multidisciplinary Optimization, 2005.

3.7.11 Fritzson, P., Aronsson, P., Pop, A., Lundvall, H., Nystrom, K., Saldamli, L., Broman D., Sandholm, A. '*OpenModelica – A Free Open-Source Environment for System Modeling, Simulation, and Teaching*', Proceedings of IEEE Conference on Computer Aided Control Systems Design, pp. 1588-1595, 2006.

3.7.12 Claeys, F.H.A., Fritzson, P., Vanrolleghem, P.A., '*Using Modelica Models for Complex Virtual Experimentation with the Tornado Kernel*', The Modelica Association, pp. 193-202, 2006.

3.7.13 Vaze S., Krishnaswami, P., DeVault, J., '*Component Based Modeling of Electromechanical Systems*', Proceedings of IDETC/CIE, Vol. 2, 2005.

3.7.14 Vaze S., DeVault, J., Krishnaswami, P., '*Modeling of Hybrid Electromechanical Systems using a Component-based Approach*', IEEE International Conference on Mechatronics and Automation, ICMA 2005, pp. 204-209, 2005.

3.7.15 Vaze, S., Krishnaswami, P., DeVault, J., '*Symbolic-Numeric Computing in Software Development for Modeling and Simulation of Multidisciplinary Multibody Systems*', Proceedings of Multibody Dynamics, An Eccomas Thematic Conference, June 2007.

CHAPTER 4 - DESIGN SENSITIVITY ANALYSIS OF NONLINEAR MULTIDISCIPLINARY MULTIBODY SYSTEMS IN THE MIXEDMODELS PLATFORM

Manuscript Publication:

Proceedings of IDETC/CIE 2007:

ASME 2007 International Design Engineering Technical Conferences & Computers and Information in Engineering Conference

September 4-7, 2007, Las Vegas, Nevada, USA

Authors:

Shilpa A. Vaze[†], Prakash Krishnaswami[*], James E. DeVault[†].

Authors' Affiliations:

[†] Department of Electrical and Computer Engineering, Kansas State University, Manhattan, KS – 66506.

[*] Department of Mechanical and Nucler Engineering, Kansas State University, Manhattan, KS – 66506.

NOTE: This chapter is largely based on the original publication, however; changes have been made to the manuscript to maintain continuity and consistency in reading.

4.1 Abstract

Most state-of-the-art multibody systems are multidisciplinary and encompass a wide range of components from various domains such as electrical, mechanical, hydraulic, pneumatic, etc. The design considerations and design parameters of system can come from any of these domains or from a combination of these domains. In order to perform analytical design sensitivity analysis on a multidisciplinary system (MDS), we first need a uniform modeling approach for this class of systems to obtain a unified mathematical model of the system. Based on this model, we can derive a unified formulation for design sensitivity analysis.

In this paper, we present a modeling and design sensitivity formulation for MDS that has been successfully implemented in the MIXEDMODELS (Multidisciplinary Integrated eXtensible Engine for Driving Metamodeling, Optimization and DEsign of Large-scale Systems) platform. MIXEDMODELS is a unified analysis and design tool for MDS that is based on a procedural, symbolic-numeric architecture. This architecture allows any engineer to add components in his/her domain of expertise to the platform in a modular fashion. The symbolic engine in the MIXEDMODELS platform synthesizes the system governing equations as a unified set of nonlinear differential-algebraic equations (DAEs). These equations can be differentiated with respect to design variables to obtain an additional set of DAEs in the sensitivity coefficients of the system state variables with respect to the system's design variables. This combined set of DAEs can be solved numerically to obtain the solution for the state variables and state sensitivity coefficients of the system. Finally, knowing the system performance functions, we can calculate the design sensitivity coefficients of these performance functions by using the values of the state variables and state sensitivity coefficients obtained from the DAEs.

In this work we use the direct differentiation approach for sensitivity analysis, as opposed to the adjoint variable approach, for ease in error control and software implementation. The capabilities and performance of the proposed design sensitivity analysis formulation are demonstrated through a numerical example consisting of an AC rectified DC power supply driving a slider crank mechanism. In this case the

performance functions and design variables come from both the electrical and mechanical domains. The results obtained were verified by perturbation analysis, and the method was shown to be accurate and computationally viable.

4.2 Introduction

4.2.1 Current State of the Art

In recent years emphasis has been placed on the development of an integrated approach for analysis and design of multidisciplinary multibody systems. The traditional optimization approach for multidisciplinary systems (MDS) utilizes sequential optimization, wherein each subsystem – electrical, mechanical, hydraulic, etc. – is optimized in isolation in a predetermined order, assuming that the designs of the other subsystems remain fixed. This often leads to system designs that are suboptimal [4.7.1-4.7.3]. This is mainly due to the extensive interaction between different domains such as electrical, mechanical, hydraulic, pneumatic etc, that the sequential design approach ignores by decoupling the subsystems. For example, in many multidisciplinary systems, the design variables of the electrical subsystems may strongly affect the performance of the mechanical subsystem; hence the sensitivities of the performance functions from the mechanical domain need to be calculated with respect to the design variables in the electrical domain in order to optimize the design of the entire system. This kind of interaction between the subsystems cannot be captured in a sequential design approach. Further, it has been shown in [4.7.1-4.7.3] that integrated optimization techniques for multidisciplinary systems result in better designs in terms of robustness and optimality.

The first step towards devising an integrated optimization scheme for multidisciplinary systems is to develop a unified modeling, simulation and sensitivity analysis platform. Some analysis tools for MDS are currently available in the market. However, there are certain traits that limit their range of applicability. One existing approaches applies a single physical/mathematical formulation such as a Lagrangian approach, Bond Graph Theory, Linear Graph Theory, State Space etc. to derive governing equations of a multidisciplinary system [4.7.4-4.7.5]. However, these

formulations are generally well-suited to particular application domain(s), are not easily extensible to accommodate new domains. Nevertheless, researchers have been successful in applying this approach in certain domains. NEWOPT/AIMS [4.7.6] is a software package that includes algorithms for simulation, sensitivity analysis and optimization of a class of mechatronic systems based on a linear state-space model. Sensitivities are provided by the semi-analytical adjoint variable method, automatic differentiation or finite differences. Similarly, a linear state space formulation for design sensitivity and optimization of controlled mechanical systems is presented in [4.7.1 – 4.7.3].

Another strategy is to develop different software packages for each relevant problem domain and then integrate these software packages as needed to model an MDS. These domain-specific packages communicate with each other to simulate the system behavior. MATLAB/Simulink [4.7.7], Simplorer [4.7.8], and AMESim [4.7.9], are some of the existing tools that fall under this category. While this approach provides a working solution in many cases, it is often cumbersome and does not yield an explicit mathematical formulation of the system governing equation which can be used in analytical design sensitivity analysis.

The third approach that is becoming popular can be categorized as a declarative or equation based approach which emphasizes integration across domains at the level of system governing equations. An interesting application of this approach in the automotive field is presented in [4.7.10], where the authors use this strategy to globally model, simulate, and optimize complex industrial mechatronic systems using MATLAB-Simulink and a Finite Element Method. In both these approaches, sensitivities are calculated using finite differences. However, the authors conclude from their studies that the optimization process can be greatly improved by adopting an integrated modeling approach that could support semi-analytical design sensitivity analysis. The equation level integration approach, also called the component approach, is also the central idea behind the Modelica object-oriented specification language for multidisciplinary systems [4.7.11, 4.7.12]. It has been shown in [4.7.13, 4.7.14] that this approach can be used to develop an effective analysis and solution tool for MDS modeling using a mixed symbolic-numeric software architecture.

In our previous work [4.7.13], we presented a formulation which can be applied more generally to include disciplines such as mechanical, electrical, hydraulic, pneumatic etc. A design sensitivity analysis scheme based on this formulation is discussed in [4.7.14]. However, this formulation shares a drawback with linear state-space methods when it comes to dealing with a general MDS whose component governing equations may include a large number of nonlinear algebraic or differential equations. For example, this is often the case when electronic, hydraulic, and pneumatic components are involved. The linear state space formulation and the formulation in [4.7.13, 4.7.14] require that these equations be converted to a pre-specified form by a process of differentiation. This has the undesirable effect of artificially increasing the number of differential equations in the system, and generally making the numerical solution harder; it is even possible that a non-stiff system may appear numerically stiff in the differentiated form. Thus, there is a need for a formulation for MDS that does not require this.

This paper presents a general nonlinear formulation for analysis and analytical design sensitivity analysis of multidisciplinary systems that is successfully implemented in the MIXEDMODELS (Multidisciplinary Integrated eXtensible Engine for Driving Metamodeling, Optimization and DEsign of Large-scale Systems) platform. MIXEDMODELS is a unified analysis and design tool for multidisciplinary systems that utilizes a component-based formulation [4.7.15]. It is a flexible, extensible, and compact platform that is based on a procedural, symbolic-numeric software architecture. The sensitivity analysis formulation discussed in this paper uses the direct differentiation approach, which was preferred due to its simplicity of implementation and ease of error control.

4.3 The MIXEDMODELS Formulation

This section presents the mathematical formulation developed for system analysis and design sensitivity analysis in the MIXEDMODELS platform. The goal is to develop a robust and numerically viable formulation that (a) allows us to analyze all aspects of a general class of multidisciplinary systems, and (b) provides built-in support for parametric studies such as design sensitivity analysis, optimization etc. In

section (4.3.1) we present the formulation for system analysis. Based on this we then develop an analytical design sensitivity analysis formulation which is presented in section (4.3.2).

4.3.1 Analysis Formulation

The modeling approach adopted in the MIXEDMODELS platform considers a multidisciplinary system to be a collection of interacting components, which can be completely described by a vector of time-invariant system parameters, **P**; a vector of system variables, **X**, which can also occur in the first derivative form $\dot{\mathbf{X}}$ in the DAEs; a vector of algebraic variables, **Y,** that can occur algebraically in the set of DAEs; and a set of governing DAEs of the following form:

$$\mathbf{F}(\mathbf{P}, \mathbf{X}(\mathbf{d}, t), \dot{\mathbf{X}}(\mathbf{d}, t), \mathbf{Y}(\mathbf{d}, t), \mathbf{d}, t) = \mathbf{0} \tag{4.3.1}$$

where **d** is a vector of design variables.

Many multidisciplinary systems are governed by higher-order differential equations. An n^{th}-order differential equation must be reduced to n first-order differential equations by defining additional variables. The (n-1) new variables that are introduced to represent the lower order derivatives are referred to as "lower-order variables" and are also part of the **X** vector. Thus, the first derivatives of these additional variables can always be obtained directly from the **X** vector by assignment. The **X** vector can then be partitioned as

$$\mathbf{X} = \begin{bmatrix} \mathbf{X_H} \\ \mathbf{X_L} \end{bmatrix} \tag{4.3.2}$$

where $\mathbf{X_H}$ represents the subvector of **X** such that the derivatives of the elements of $\mathbf{X_H}$ are not contained in **X**. Similarly, $\mathbf{X_L}$ represents the subvector of **X** containing all the elements of **X** such that their derivative is also an element of **X**, i.e. the lower order variables. Thus, the lower order variables can be calculated by a set of direct assignments of the form

$$\dot{\mathbf{X}}_{\mathbf{L}} = \mathbf{X}' \tag{4.3.3}$$

where $\mathbf{X}'$ is a subvector of $\mathbf{X}$. Separating Equation (4.3.3) from the system of equations given by Equation (4.3.1), the rest of the system governing equations can be written as

$$\mathbf{f}(\mathbf{P}, \mathbf{X}(\mathbf{d}, t), \dot{\mathbf{X}}_{\mathbf{H}}(\mathbf{d}, t), \mathbf{Y}(\mathbf{d}, t), \mathbf{d}, t) = \mathbf{0} \tag{4.3.4}$$

Equations (4.3.3) and (4.3.4) describe the complete system. As previously stated, an MDS can be viewed as a collection of components, where each component adds its contribution to the system governing equations to form the complete system of equations. A component can introduce new system variables, or it can be described in terms of system variables contributed by other components. It may add new governing equations to the system and/or modify system equations contributed by other components. In order to support the above formulation at the system level, we require some information to be provided at the component level whenever a new component class is defined. Specifically, if a component is component *i* in the MDS, we require the following for that component:

- a vector of time-invariant component parameters, $\mathbf{p}^i$,
- a vector of transient component differential variables, $\mathbf{x}^i$ (these $\mathbf{x}^i$ occur in the governing DAEs in first derivative form $\dot{\mathbf{x}}^i$),
- a vector of component algebraic variables, $\mathbf{y}^i$, which occur algebraically in the DAEs,

In addition, the component model must provide the following,

- a set of component governing equations which can be expressed as

$$\mathbf{f}_i^c(\mathbf{P}, \mathbf{X}(\mathbf{d}, t), \dot{\mathbf{X}}_{\mathbf{H}}(\mathbf{d}, t), \mathbf{Y}(\mathbf{d}, t), t) = \mathbf{0} \tag{4.3.5}$$

- a set of direct assignments given by

$$\dot{\mathbf{X}}_{\mathbf{L}_i}^{c} = \mathbf{X}^{c'} \tag{4.3.6}$$

- and a set of component modification equations of the form

$$\mathbf{f}_i^m(\mathbf{P}, \mathbf{X}(\mathbf{d},t), \dot{\mathbf{X}}_\mathbf{H}(\mathbf{d},t), \mathbf{Y}(\mathbf{d},t), t) = \mathbf{0} \tag{4.3.7}$$

These equations describe the modifications that this component makes in the equations of other components. The contributions of all the components in the system are summed to obtain the system governing equations

$$\mathbf{f}(\mathbf{P}, \mathbf{X}, \dot{\mathbf{X}}_\mathbf{H}, \mathbf{Y}, t) = \sum_i \left[\mathbf{f}_i^c(\mathbf{P}, \mathbf{X}, \dot{\mathbf{X}}_\mathbf{H}, \mathbf{Y}, t) + \mathbf{f}_i^m(\mathbf{P}, \mathbf{X}, \dot{\mathbf{X}}_\mathbf{H}, \mathbf{Y}, t) \right] \tag{4.3.8}$$

Similarly, direct assignment contributions from each component are added to get Equation (4.3.3). Thus, we get the complete set of system DAEs given by

$$\begin{aligned} \mathbf{f}(\mathbf{P}, \mathbf{X}(\mathbf{d},t), \dot{\mathbf{X}}_\mathbf{H}(\mathbf{d},t), \mathbf{Y}(\mathbf{d},t), \mathbf{d}, t) &= \mathbf{0} \\ \dot{\mathbf{X}}_\mathbf{L}(\mathbf{d},t) &= \mathbf{X}'(\mathbf{d},t) \end{aligned} \tag{4.3.9}$$

The symbolic engine in the MIXEDMODELS platform, which is written in Maple, performs the task of forming the system equations in explicit symbolic form from the component equations. These explicit expressions can then be output as C or FORTRAN code for numerical solution.

The ordinary differential equations (ODEs) in Equation (4.3.9) can be solved to obtain the differential variable vector **X** using any suitable ODE solver (the work reported in this paper uses the DLSODES solver [4.7.16]). To be able to do so, at any time *t*, given the current estimate of the dependent variable vector **X** in the ODEs, we need a way to calculate the derivatives $\dot{\mathbf{X}} = \left[\dot{\mathbf{X}}_\mathbf{H} \quad \dot{\mathbf{X}}_\mathbf{L} \right]^\mathrm{T}$. $\dot{\mathbf{X}}_\mathbf{L}$ are calculated directly by using Equation (4.3.3). $\dot{\mathbf{X}}_\mathbf{H}$ and **Y** are calculated using Newton-Raphson iteration as summarized below.

Let us define a vector **q** as

$$\mathbf{q} = \begin{bmatrix} \dot{\mathbf{X}}_\mathbf{H} \\ \mathbf{Y} \end{bmatrix}_{(N+M)\times 1} \tag{4.3.10}$$

where *N* denotes the number of higher order differential variables and *M* denotes the number of algebraic variables. Given the initial conditions, **X**, and initial guesses on $\dot{\mathbf{X}}_\mathbf{H}$ and **Y**, we can iteratively calculate

$$\mathbf{q}^{k+1} = \mathbf{q}^{k} - \mathrm{J}^{-1}(\mathbf{q}^{k})\mathbf{f}(\mathbf{P}, \mathbf{X}, \mathbf{q}^{k}, t) \tag{4.3.11}$$

where J is the Jacobian matrix given by

$$\mathrm{J}(\mathbf{P}, \mathbf{X}, \mathbf{d}, t) = \begin{bmatrix} \dfrac{\partial \mathbf{f}}{\partial \dot{\mathbf{X}}_{\mathbf{H}}} & \dfrac{\partial \mathbf{f}}{\partial \mathbf{Y}} \end{bmatrix} \tag{4.3.12}$$

Let $\Delta\mathbf{q}$ denote the Newton differences such that

$$\Delta\mathbf{q} = \mathbf{q}^{k+1} - \mathbf{q}^{k} = \begin{bmatrix} \Delta\dot{\mathbf{X}}_{\mathbf{H}} \\ \Delta\mathbf{Y} \end{bmatrix} \tag{4.3.13}$$

The Newton differences can be calculated by solving the linear system

$$\mathrm{J}(\mathbf{q}^{k}) \begin{bmatrix} \Delta\dot{\mathbf{X}}_{\mathbf{H}} \\ \Delta\mathbf{Y} \end{bmatrix} = -\mathbf{f}(\mathbf{q}^{k}) \tag{4.3.14}$$

The improved estimate $\mathbf{q}^{k+1}$ is then obtained from

$$\mathbf{q}^{k+1} = \mathbf{q}^{k} + \begin{bmatrix} \Delta\dot{\mathbf{X}}_{\mathbf{H}} \\ \Delta\mathbf{Y} \end{bmatrix} \tag{4.3.15}$$

This is equivalent to the iteration implied in Equation (4.3.11) but is preferred because we do not need to invert the Jacobian matrix. The Jacobian matrix is also formed symbolically by the MIXEDMODELS symbolic platform. Once the Newton-Raphson iteration has converged, we will have not only the derivatives needed by the ODE solver, but also the values of the algebraic system variables **Y**, since both of these are contained in the **q** vector.

4.3.2 Sensitivity Analysis Formulation

Based on the nonlinear analysis formulation described in the previous section, we can now derive a formulation for sensitivity analysis. There are two popular approaches for analytical sensitivity design analysis: the direct differentiation approach and the adjoint variable approach. The adjoint variable approach has the potential to reduce the computational burden, however it requires integrating the adjoint equations backward in time. This can cause difficulties in error control and

complexities in software implementation. Therefore, we choose the direct differentiation approach to develop the sensitivity formulation presented in this paper.

It is assumed that the performance functions of interest in the system are of the form

$$\mathrm{g}_i(\mathbf{P}, \mathbf{X}(\mathbf{d},t), \dot{\mathbf{X}}(\mathbf{d},t), \mathbf{Y}(\mathbf{d},t), \mathbf{d}, t), \quad i = 1 \cdots n_g \tag{4.3.16}$$

where n_g denotes the number of performance functions. For a particular performance function, g_i, we can then derive the sensitivity vector as given by,

$$(\mathrm{g}_i)_{\mathbf{d}} = (\mathrm{g}_i)_{\mathbf{d}|\exp} + (\mathrm{g}_i)_{\dot{\mathbf{X}}} \dot{\mathbf{X}}_{\mathbf{d}} + (\mathrm{g}_i)_{\mathbf{X}} \mathbf{X}_{\mathbf{d}} + (\mathrm{g}_i)_{\mathbf{Y}} \mathbf{Y}_{\mathbf{d}} \tag{4.3.17}$$

where a vector subscript denotes partial differentiation with respect to the subscript and $(\mathrm{g}_i)_{\mathbf{d}|\exp}$ represents the explicit derivative of g_i with respect to **d**. The derivatives of g_i in Equation (4.3.17) are directly obtained by differentiation in the MIXEDMODELS symbolic engine. However, the state sensitivities $\dot{\mathbf{X}}_{\mathbf{d}}$, $\mathbf{X}_{\mathbf{d}}$ and $\mathbf{Y}_{\mathbf{d}}$ need to be calculated numerically. These can be obtained by differentiating the system of governing equations with respect to the design vector. Based on the partition of the **X** vector given by Equation (4.3.2), we can partition the vector $\mathbf{X}_{\mathbf{d}}$ in a similar way:

$$\mathbf{X}_{\mathbf{d}} = \begin{bmatrix} \mathbf{X}_{\mathbf{H}_{\mathbf{d}}} \\ \mathbf{X}_{\mathbf{L}_{\mathbf{d}}} \end{bmatrix} \tag{4.3.18}$$

Then the state sensitivities $\mathbf{X}_{\mathbf{L}_{\mathbf{d}}}$ can be easily calculated from the direct assignment equations given by

$$\dot{\mathbf{X}}_{\mathbf{L}_{\mathbf{d}}} = \mathbf{X}'_{\mathbf{d}} \tag{4.3.19}$$

where, $\mathbf{X'_d}$ is a subvector of $\mathbf{X_d}$.

Let

$$\mathbf{q_d} = \begin{bmatrix} \dot{\mathbf{X}}_{\mathbf{H_d}} \\ \mathbf{Y_d} \end{bmatrix}_{(N+M)\times(N_d)} \tag{4.3.20}$$

where, N_d denotes the number of design variables. Now we need to calculate $\mathbf{q_d}$. By differentiating Equation (4.3.4) with respect to the design variable vector $\mathbf{d}$ we get

$$\mathbf{f_d} = \mathbf{f}_{\dot{\mathbf{X}}_\mathbf{H}} \dot{\mathbf{X}}_{\mathbf{H_d}} + \mathbf{f_X X_d} + \mathbf{f_Y Y_d} + \mathbf{f_{d|exp}} = \mathbf{0} \tag{4.3.21}$$

$$\therefore \quad \mathbf{f}_{\dot{\mathbf{X}}_\mathbf{H}} \dot{\mathbf{X}}_{\mathbf{H_d}} + \mathbf{f_Y Y_d} = -\mathbf{f_{d|exp}} - \mathbf{f_X X_d} \tag{4.3.22}$$

$$\therefore \quad \begin{bmatrix} \mathbf{f}_{\dot{\mathbf{X}}_\mathbf{H}} & \mathbf{f_Y} \end{bmatrix} \begin{bmatrix} \dot{\mathbf{X}}_{\mathbf{H_d}} \\ \mathbf{Y_d} \end{bmatrix} = -\mathbf{f_{d|exp}} - \mathbf{f_X X_d} \tag{4.3.23}$$

Equations (4.3.23) and (4.3.19) together represent a system of DAEs that can be solved numerically to obtain the state sensitivities in a manner analogous to the solution of the system governing DAEs of Equation (4.3.9). We further note that unlike Equation (4.3.9) (which is nonlinear in the derivatives of the differential variables as well as the algebraic variables) Equations (4.3.19) and (4.3.23) are linear in the corresponding sensitivities. Thus, they can be solved directly without iteration. Further, we note that

$$\begin{bmatrix} \mathbf{f}_{\dot{\mathbf{X}}_\mathbf{H}} & \mathbf{f_Y} \end{bmatrix} = \begin{bmatrix} \dfrac{\partial \mathbf{f}}{\partial \dot{\mathbf{X}}_\mathbf{H}} & \dfrac{\partial \mathbf{f}}{\partial \mathbf{Y}} \end{bmatrix} = \mathbf{J} \tag{4.3.24}$$

which is the same as the Jacobian in Equation (4.3.12). Thus, the coefficient matrix for Equation (4.3.23) is already available, which makes the solution of this equation convenient.

The system of DAEs of Equations in (4.3.19) and (4.3.23) is solved numerically concurrently with the DAEs of Equation (4.3.9). The ODEs in these equations can be solved by any suitable ODE solver. The dependent variables seen by the ODE solver are now the system differential variables $\mathbf{X}$ as well as their sensitivities $\mathbf{X_d}$. When the ODE solver calls for derivative evaluation at a particular time t with the current estimate for this set of dependent variables (i.e., $\mathbf{X}$ and $\mathbf{X_d}$), we do the following;

- First, perform the Newton-Raphson iteration of Equations (4.3.14) and (4.3.15) to calculate the derivatives of the higher-order differential state variables $\dot{\mathbf{X}}_{\mathbf{H}}$, and the algebraic state variables **Y**.
- Next, use the direct assignments in Equation (4.3.9) to set the values of the lower order differential state variables, $\dot{\mathbf{X}}_{\mathbf{L}}$.
- Once we have the values of all the state variables, we can solve Equation (4.3.23) to obtain the derivatives of the sensitivities of the higher-order differential variables $\dot{\mathbf{X}}_{\mathbf{H_d}}$ and the sensitivities of the algebraic state variables $\mathbf{Y_d}$.
- Finally, the derivatives of the sensitivities of the lower-order differential state variables are set through direct assignment from Equation (4.3.19).

4.3.3 Calculating Initial Conditions

In order to start the integration from the given initial time, initial conditions must be provided for all of the differential variables that we wish to determine. Specifically, initial conditions must be given not only for **X,** but also for $\mathbf{X_d}$. In specifying initial conditions, care must be taken to ensure that the specified initial conditions are physically realizable and consistent with all system constraints, such as loop closure conditions on a mechanical system. Thus, it may be difficult for the user to manually supply a consistent set of initial conditions for **X** for a large system; if the user is further called upon to provide initial conditions for $\mathbf{X_d}$, the task is even more burdensome. To ensure consistency of initial conditions, we assume that the user provides a set of consistency equations that must be satisfied by the initial conditions. These equations are assumed to be of the form

$$\mathbf{h}(\mathbf{P}, \mathbf{X}(\mathbf{d}, t), \mathbf{d}, t)|_{t=0} = \mathbf{0} \tag{4.3.25}$$

where **h** is a vector of nonlinear algebraic equations of dimension $(N+L)$ that can be solved by Newton-Raphson iteration to obtain a consistent initial conditions vector, **X**. Recall that N is the number of higher-order differential variables and L is the number of lower-order differential variables. All that the user needs to provide then is the set of initial guesses for **X**, which are then corrected by the iteration. In order

to compute initial conditions on sensitivities, we differentiate Equation (4.3.25) with respect to the design variable vector. The initial condition vector, $\mathbf{X_d}$, for sensitivities can then be obtained by solving the resulting Equation (4.3.26) as a linear system of algebraic equations.

$$\mathbf{h_X X_d} = -\mathbf{h_{d|exp}} \tag{4.3.26}$$

Since this is a system of linear equations, the user does not have to provide initial guesses for the $\mathbf{X_d}$ – the correct initial conditions are calculated directly from Equation (4.3.26). Once again, we can take advantage of the fact that the coefficient matrix in Equation (4.3.26) is the same as the Jacobian in the Newton-Raphson iteration for Equation (4.3.25).

4.4 Numerical Example

As an example we simulate and compute design sensitivities for a slider-crank mechanism driven by an AC-rectified DC power supply (Figure 4.4.1).

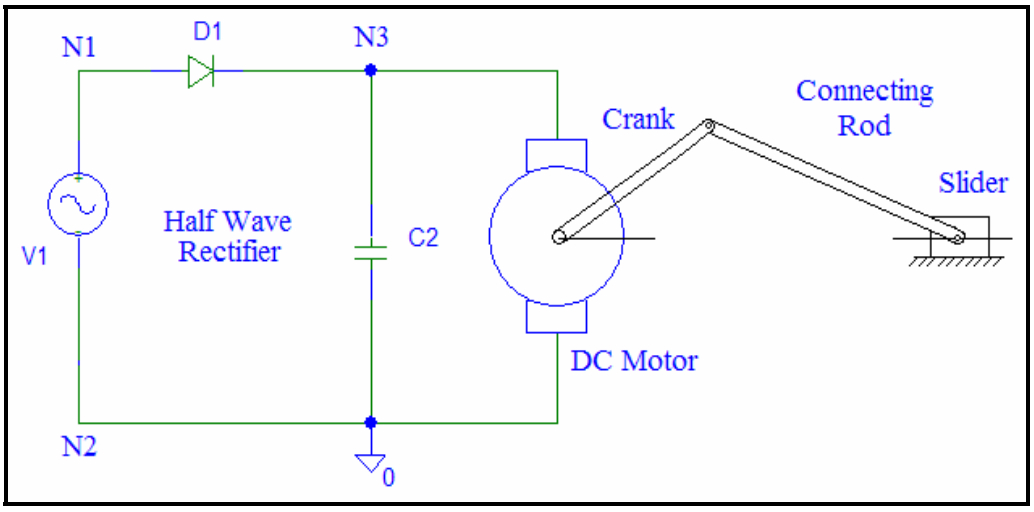


Figure 4.4.1: Slider Crank Mechanism Powered by a Half-Wave Rectifier

The system has 19 components, viz., one DC Motor, three Rigid Bodies (Crank, Connecting Rod, Slider), two Revolute Joints, one Bipolar Junction Diode, one Capacitor, one Sinusoidal Voltage Source, three Electric Nodes, one Analog Ground Component, one Slider Constraint, one Fixed Axis Body Constraint, two Mechanical Connections (torque and acceleration connection between the motor and crank), and

two Performance Functions. The crank is modeled as a fixed-axis body by adding constraint components. The diode and the capacitor circuit along with the sinusoidal voltage source form the half-wave rectifier. The slider-crank mechanism acts as the load to the electrical system. The entire system is assumed to be at rest with a crank angle of 45 degrees when the AC power is turned on; thus all the currents and voltages in the system are zero at the initial time.

4.4.1 System Specifications

4.4.1.1 DC Motor

J_m = Motor Inertia = 0.0044 oz-in-s^2

R_a = Motor Armature Resistance = 2.4 Ω

L_a = Motor Winding Inductance = 0.0048 mH

K_b = Back EMF Constant = 0.0401 v/rad/s

K_T = Torque Constant = 5.6 oz-in/amp

B_v = Viscous Friction = 0.0137 oz-in/rad/s

4.4.1.2 Crank

L_1 = length of link 1 = 1.0 in

m_1 = mass of link 1 = 0.00136 oz-s^2/in

J_1 = inertia of link 1 = 0.00414 oz-in-s^2

4.4.1.3 Connecting Rod

L_2 = length of link 2 = 3.0 in

m_2 = mass of link 2 = 0.0041 oz-s^2/in

J_2 = inertia of link 2 = 0.0373 oz-in-s^2

4.4.1.4 Slider

m_s = slider mass = 0.6211 oz-s^2/in

J_s = slider inertia = 1.0 oz-in-s^2

4.4.1.5 Bipolar Junction Diode (PSpice D1N4148)

η = emission coefficient = 1.836

V_t = thermal voltage = 0.026 V

I_s = saturation current = 2.682 nA

B_R = reverse breakdown voltage = 100 V

d = grading coefficient = 0.3333

τ_t = minority carrier lifetime = 11.54 ns

C_{j0} = zero bias depletion capacitance = 4 pF

ϕ_0 = built in potential = 0.5 V

R_b = junction ohmic resistance = 0.5664 Ω

4.4.2 Design Variables

To verify the validity of our approach we chose two design variables for the system. These two design variables are the slider mass and the source voltage. The system sensitivities are calculated with respect to each design variable using the method in the previous section. The perturbation check on the sensitivity analysis is done individually for each variable, i.e., we only introduce perturbation in one variable at a time. The nominal and perturbed values of the design variables are as given below.

d = [slider mass, source voltage]

$\mathbf{d}_{\mathbf{original}}$ = [0.6211, 5.0] , $\mathbf{d}_{\mathbf{perturbed}}$ = [0.6221, 5.01]

4.4.3 Performance Functions

For this example we use two types of performance functions: a "grid type" and an "integral type". Instantaneous power is chosen to be the grid type performance function imposed at every 0.1 s, and total energy consumption of the system is treated as an integral performance function as given below:

$$g_1 = V_c I_a \tag{4.4.1}$$

$$g_2 = \int V_c I_a \, dt \tag{4.4.2}$$

4.4.4 Perturbation Analysis

Design sensitivity calculations are verified by perturbation analysis using finite differences. Let d_1 be the design variable of interest, let Δd_1 denote the perturbation in d_1 and let the response variable of interest be z_1. Before adding perturbation to d_1, calculate response z_1. With all other design variables unchanged, introduce perturbation Δd_1 in d_1 such that

$$d_1 \rightarrow d_1 + \Delta d_1 \tag{4.4.3}$$

Simulate the system to get the perturbed response z'_1.

Then, the actual change in the two responses can be calculate at any time t as

$$\Delta z_1(\text{actual}) = z_1 - z_1' \tag{4.4.4}$$

At any time t, the corresponding change in the response variable z_1 can also be estimated using the sensitivity information as follows:

$$\Delta z_1(\text{predicted}) = \frac{\partial z_1}{\partial d_1} \Delta d_1 \tag{4.4.5}$$

We can now compare Δz_1(actual) with Δz_1(predicted) to assess the accuracy of the sensitivity coefficient in Equation (4.4.5). For this check, we can also plot Δz_1(actual) and Δz_1(predicted) as functions of time. A similar check can be done on any performance function.

4.5 Simulation Results and Discussion

The example described earlier was run using the MIXEDMODELS platform, and the results obtained are summarized here. Figure (4.5.1) shows the response plots obtained for the selected system variables. In left column, we have plots of motor angular velocity, armature current and slider velocity. In the right hand column we have slider acceleration and slider displacement. The plot in the upper right hand corner is a detailed view of the slider acceleration before the spike in acceleration,

showing details that are not visible due to the scaling in the figure below it. The wiggles seen in the current and acceleration plots are due to the transient response of the half-wave rectifier. This implies fatigue loading on the mechanism.

Figures (4.5.2) and (4.5.3) present the perturbation analysis check with slider acceleration and armature current as response variables, and slider mass as the design variable. Both plots show that the predicted change is in close agreement with the actual change found by finite differences – in fact, in all cases the two plots are on top of each other. In both figures, the second plot captures the details before the spike. It can be seen that the results are in a very good agreement even during the spike in acceleration.

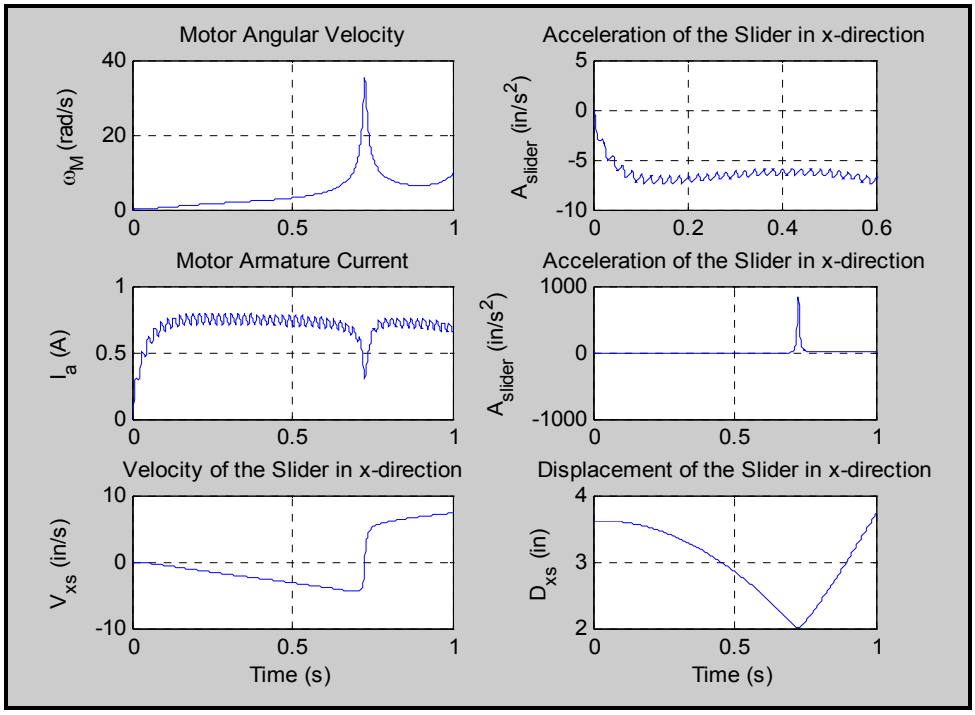


Figure 4.5.1: Slider Crank Simulation Results

For the integral performance function for the total energy, the actual change found by finite differences was 7.2603 e^{-5}J, whereas the predicted change was 7.3399 e^{-5}J, giving an absolute error between the predicted and actual change of 7.96 e^{-7} J. This is an error of less than 1%. For the performance function on instantaneous power, the finite difference test results are shown in the Table (4.5.1). From the table we see that the predicted and actual changes in instantaneous power for the reported times

are in close agreement. Maximum absolute error reported during this analysis was $5.22e^{-5}$ W which is an error of about 1.72%.

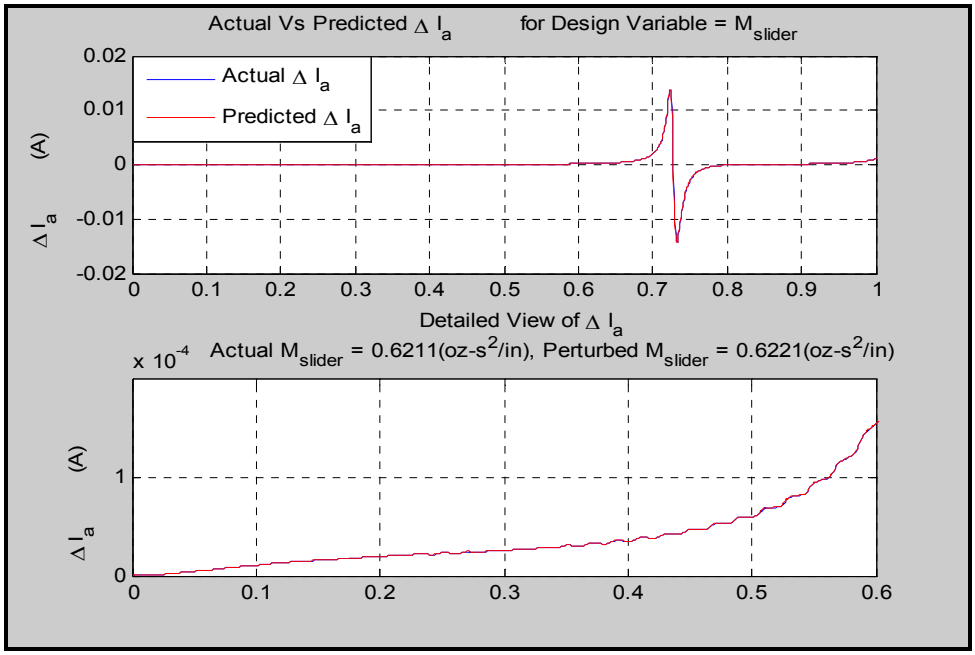


Figure 4.5.2: Actual Versus Predicted Change in Armature Current (Design Variable: Slider Mass)

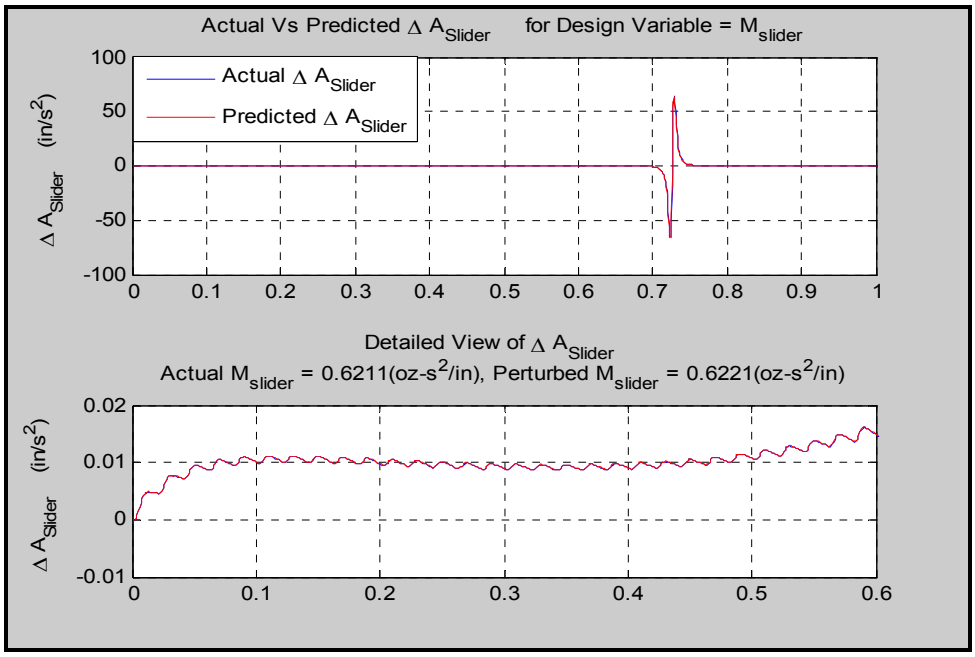


Figure 4.5.3: Actual Versus Predicted Change in Slider Acceleration (Design Variable: Slider Mass)

Figures (4.5.4) and (4.5.5) show the perturbation analysis check with armature current and slider acceleration as the response variables and source voltage as the design variable. Both the plots show a good agreement between the predicted change and actual change. It can also be inferred from Figures (4.5.2) and (4.5.4) that motor armature current is more sensitive to the source voltage than it is to slider mass.

Comparing the actual and predicted change in instantaneous power given in Table (4.5.2), we can see that there is a good match in the results.

Time (s)	Actual Change in Instantaneous Power (W)	Predicted Change in Instantaneous Power (W)	Relative Difference (%)
0.1	1.1391E-05	1.1409E-05	1.57770181E-01
0.2	1.4675E-05	1.4686E-05	7.49012665E-02
0.3	1.4834E-05	1.4854E-05	1.34643866E-01
0.4	2.1486E-05	2.1531E-05	2.09000975E-01
0.5	4.5633E-05	4.5773E-05	3.05857164E-01
0.6	1.5552E-04	1.5628E-04	4.86306629E-01
0.7	2.9802E-03	3.0324E-03	1.72140878E+00
0.8	5.0849E-04	5.1560E-04	1.37897595E+00
0.9	3.3350E-04	3.3409E-04	1.76599119E-01
1	1.5731E-03	1.5999E-03	1.67510469E+00

Table 4.5.1: Actual Versus Predicted Change in Power

(Design Variable: Slider Mass)

Note that the sudden rise in the predicted value and the actual change in the value of this performance function around t=0.7s is because around that time the slider is close to its dead center. The maximum absolute error reported for this case is also at the same grid point, and is approximately about 13%.

Also note that at other grid points, when the slider is not close to any dead center, the error between the predicted and actual change is less than 1% for both the design variables. For the integral performance function, the total energy for the perturbed case is reported as 1.3272 J.

The predicted change in the performance and the actual change are reported as 0.0067234 J and 0.0067299 J, respectively. The absolute error reported in the predicted change and the actual change by finite differencing is approximately equal to $6.5E^{-6}$ J yielding less than 1% error.

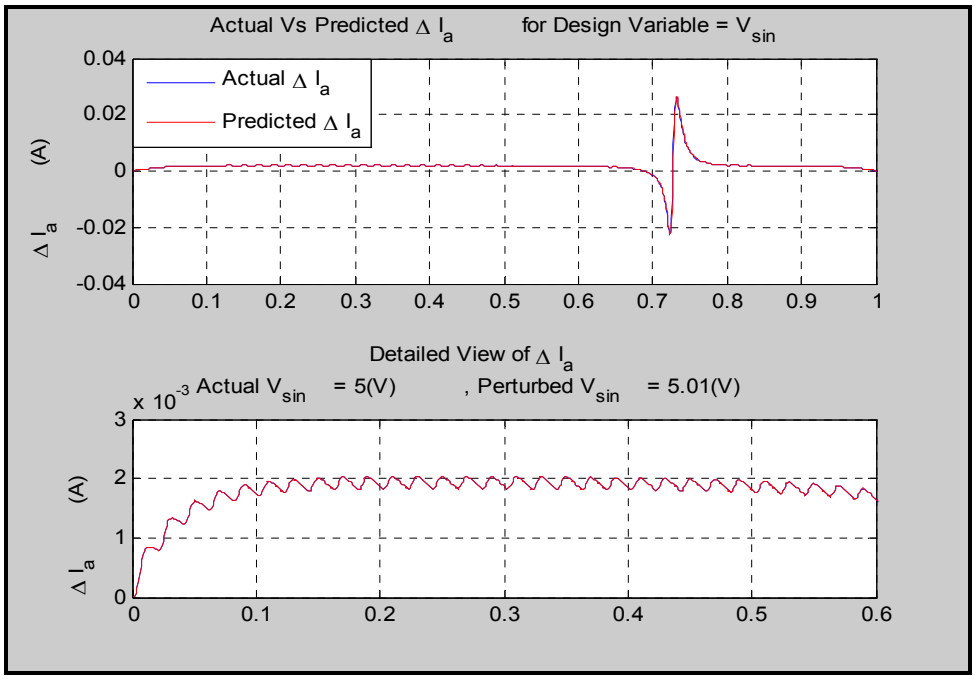


Figure 4.5.4: Actual Versus Predicted Change in Armature Current (Design Variable: Source Voltage)

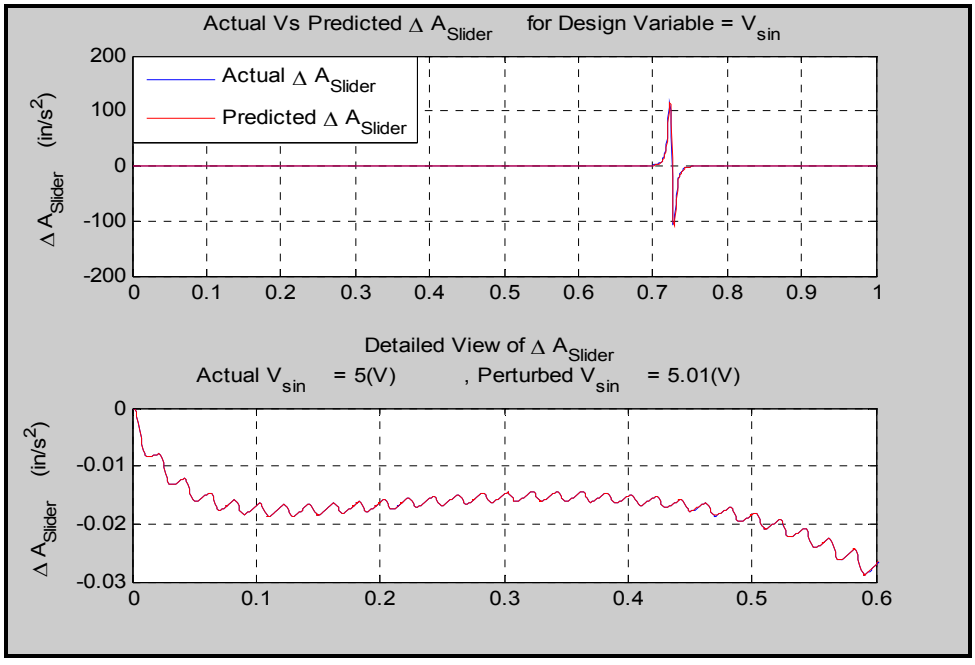


Figure 4.5.5: Actual Versus Predicted Change in Slider Acceleration (Design Variable: Source Voltage)

It was observed from the results that both the performance functions show a higher sensitivity to the source voltage as compared to the slider mass. The results with

respect to "source voltage" as a design variable were also very satisfactory and thus indicate that the proposed approach is very accurate for sensitivity calculations.

Time (s)	Actual Change in Instantaneous Power (W)	Predicted Change in Instantaneous Power (W)	Relative Difference (%)
0.1	5.6475E-03	5.6400E-03	-1.3297872E-01
0.2	6.5844E-03	6.5758E-03	-1.3078256E-01
0.3	6.6714E-03	6.6627E-03	-1.3057769E-01
0.4	6.6642E-03	6.6555E-03	-1.3071895E-01
0.5	6.6120E-03	6.6037E-03	-1.2568711E-01
0.6	6.3873E-03	6.3807E-03	-1.0343692E-01
0.7	1.0076E-03	1.1584E-03	1.3017955E+01
0.8	6.0781E-03	6.0903E-03	2.0031853E-01
0.9	6.1500E-03	6.1431E-03	-1.1232114E-01
1	3.5699E-03	3.6535E-03	2.2882167E+00

Table 4.5.2: Actual Versus Predicted Change in Power

(Design Variable: Source Voltage)

4.6 Conclusion

In this paper we presented a general nonlinear formulation for analysis and analytical design sensitivity analysis of general multidisciplinary systems. This formulation has been successfully implemented in the MIXEDMODELS platform, which allows users to seamlessly plug in components from different application domains. Thus, this formulation can be applied to a very general class of multidisciplinary systems, including domains such as electrical, mechanical, hydraulic, pneumatic, controls, etc. The sensitivity analysis formulation developed here uses the direct differentiation approach. The state sensitivity equations form a system of DAEs which can be concurrently solved along with the system governing equations to obtain the system variables and the state sensitivities.

The symbolic architecture as implemented in MIXEDMODELS is computationally viable and efficient even for large-scale systems and does not lead to expression swell. This is because as the system size grows, the number of equations increases; however, the symbolic complexity of the expressions does not grow with system size. Also, in a component-based approach, it is generally true that the connection

components add more equations and redundant variables to the system which further increases the system size. The symbolic engine includes a symbolic reduction facility which can be used to substantially reduce the size of the matrix in Equation (4.3.12). This makes the solution process numerically efficient even for large scale systems. The final set of equations thus obtained is similar in form and complexity to the corresponding equations that are solved by commercial packages such as such as Dymola and Adams. These packages have demonstrated that such equations can be solved effectively using numerical solvers.

The proposed approach was successfully demonstrated using a slider-crank mechanism driven by a DC motor powered by AC-rectified-DC supply. Design sensitivity calculations were validated using finite differences. The results obtained indicate that the method is accurate, computationally viable and applicable to a very broad class of multidisciplinary systems.

4.7 REFERENCES

4.7.1 Carrigan, J., Kelkar, A. and Krishnaswami, P., '*Minimum Sensitivity Design of Controlled Multibody Systems*', ASME Multibody Systems, Nonlinear Dynamics, and Control Conference, 2005.

4.7.2 Kelkar, A., Krishnaswami, P., '*Multidisciplinary Optimization of Multibody Systems*', Proceedings of the ECCOMAS Conference on Multibody Systems, June 2005.

4.7.3 Carrigan J., '*Integrated Design and Sensitivity Based Design and Optimization of Nonlinear Controlled Multibody Mechanisms*', Thesis, M.S. Iowa State Univ., 2003.

4.7.4 Sinha, R., Paredis, C.J.J., Liang, V-C., Khosla, P.K., '*Modeling and Simulation Methods for Design of Engineering Systems*', Journal of Computing and Information Science in Engineering, Volume 1, Issue 1, pp. 84-91, 2001.

4.7.5 Sass, L., McPhee, J., Schmitke, C., Fisette, P. and Grenier, D., '*A Comparison of Different Methods for Modelling Electromechanical Multibody Systems*', Multibody System Dynamics, 12, pp. 209-250, 2004.

4.7.6 Dignath, F., Breuninger, C., Eberhard, P., Kübler, L., '*Optimization of Mechatronic Systems Using the Software Package NEWOPT/AIMS*', Multibody System Dynamics, pp. 85-100, 2005.

4.7.7 Ram, O., Szymkat, M., Uhl, T., Betemps, M., Pjetursson, A. and Rod, J., '*Mechatronic Blockset for Simulink – Concept and Implementation*', Proceedings of the 1996 IEEE Intl. Symposium on Computer-Aided Control System Design, pp. 530-535, 1996.

4.7.8 Knorr, U.I., '*Electromechanical System Design*', Ansoft Corporation, 2002.

4.7.9 http://www.amesim.com/, IMAGINE Software Inc., acquired on Jan 29, 2005.

4.7.10 Duysinx, P., Bruls, O., Collard, J-F., Fisette, P., Lauwerys, C., Swevers, J., '*Optimization of Mechatronic Systems: Application to a Modern Car Equipped with a Semi-active Suspension*', 6th World Congress of Structural and Multidisciplinary Optimization, 2005.

4.7.11 Fritzson, P., Aronsson, P., Pop, A., Lundvall, H., Nystrom, K., Saldamli, L., Broman D., Sandholm, A. '*OpenModelica – A Free Open-Source Environment for System Modeling, Simulation, and Teaching*', Proceedings of IEEE Conference on Computer Aided Control Systems Design, pp. 1588-1595, 2006.

4.7.12 Claeys, F.H.A., Fritzson, P., Vanrolleghem, P.A., '*Using Modelica Models for Complex Virtual Experimentation with the Tornado Kernel*', The Modelica Association, pp. 193-202, 2006.

4.7.13 Vaze S., DeVault, J., Krishnaswami, P., '*Modeling of Hybrid Electromechanical Systems using a Component-based Approach*', IEEE International Conference on Mechatronics and Automation, ICMA 2005, pp. 204-209, 2005

4.7.14 Krishnaswami, P., Vaze, S., DeVault, J., '*Design Sensitivity Analysis of Multidisciplinary Multibody Systems*', Proceedings of Multibody Dynamics, An Eccomas Thematic Conference, June 2007.

4.7.15 Vaze, S., Krishnaswami, P., DeVault, J., '*Symbolic-Numeric Computing in Software Development for Modeling and Simulation of Multidisciplinary Multibody Systems*', Proceedings of Multibody Dynamics, An Eccomas Thematic Conference, June 2007.

4.7.16 http://www.netlib.org/, Netlib Repository, acquired on May 15, 2005.

CHAPTER 5 - SYMBOLIC-NUMERIC COMPUTING IN SOFTWARE DEVELOPMENT FOR MODELING AND SIMULATION OF MULTIDISCIPLINARY SYSTEMS

Manuscript Publication:

Proceedings of Multibody Dynamics 2007 Thematic Conference

June 25-28, 2007, Milano, Italy

Authors:

Shilpa A. Vaze[†], Prakash Krishnaswami[*], and James E. DeVault[†].

Authors' Affiliations:

[†] Department of Electrical and Computer Engineering, Kansas State University, Manhattan, KS – 66506.

[*] Department of Mechanical and Nuclear Engineering, Kansas State University, Manhattan, KS – 66506.

Keywords:

MIXEDMODELS, Multidisciplinary Systems, Symbolic-Numeric Computing

NOTE: This chapter is largely based on the original publication, however; changes have been made to the manuscript to maintain continuity and consistency in reading.

5.1 Abstract

The development of mathematical formulations and the associated software architecture for modeling and simulation of multidisciplinary systems are best done concurrently. Due to the diversity of the components in multidisciplinary systems, the software architecture for this field must provide a high degree of modularity, flexibility, maintainability and extensibility. For accurate modeling of components belonging to different domains, it is important that the software architecture be independent of the particular modeling approach used for any component and flexible enough to accommodate component models at different levels of complexity. Furthermore, the architecture should allow domain experts to contribute component models independently, without having to understand other domains in detail.

In this paper, we present a procedural, symbolic-numeric architecture for multidisciplinary systems. This architecture offers great modeling flexibility at the component level and facilitates extension to new problem domains. The contribution of all the system components are combined symbolically to obtain the system governing equations as a set of differential-algebraic equations (DAEs) which are solved numerically to obtain the system response. Through this architecture we achieve flexibility and convenience in modeling, along with efficiency in computation. The independence of the component models also provides extensibility. The implementation generates explicit equations in symbolic form which will allow convenient extension of this implementation for design sensitivity calculations.

To demonstrate the efficacy of the proposed symbolic-numeric formulation and software architecture, we present two examples that include a detailed description of a component model, and how it fits in the complete system of DAEs. The results are validated against Simulink and PSpice. The results indicate that the proposed approach is effective in terms of accuracy, modeling convenience, computational efficiency and the ability to simulate the behavior of multidisciplinary multibody systems.

5.2 Introduction

Over the years mathematical formulations such as the differential geometric approach [1], bond graph modeling [2, 3], linear graph theory [1], Lagrangian formulation [1, 4] and multibody constrained formulation [5, 6] have become popular for modeling of multidisciplinary systems (MDSs). Several simulation packages such as AMESIM [7], the Mechatronic Blockset for Simulink [8, 9], and Simplorer [10] have also been developed for mathematical modeling and simulation of MDS. While the current approaches provide a working solution, they are generally biased towards a particular domain and do not support extension to other domains very well. One important aspect in the development of solution methods in this area which has not received much attention is the software architecture that is required to support the mathematical formulation. A suitable architecture for this field must provide a high degree of modularity, flexibility and extensibility. This is particularly important because of the diversity of components in these systems. These systems encompass a wide range of components from various domains, such as rigid bodies and joints from the multibody dynamics domain; circuit elements and semiconductor devices from the electrical domain; motors and tachometers from the electro-mechanical domain; valves and pumps from the hydraulics domain, etc. Each of these components can be best described by using the formulation that is most natural to its domain. For example, Lagrangian or Newtonian mechanics is a good choice for modeling of multibody systems. Similarly, nodal analysis is an effective method for modeling electrical systems. Therefore, for accurate modeling of components belonging to different domains, it is important that the software architecture be independent of the modeling approach and flexible enough to accommodate component models at different levels of complexity. Furthermore, the software architecture should provide a plug-and-play facility for domain experts to contribute component models independently, minimizing the need for them to understand other domains or components in detail. This will permit the development of a versatile and complete simulation tool without the necessity of having one person master all the relevant domain knowledge. Support for symbolic computing is

also desirable for extending the work to parametric studies, design sensitivity analysis, metamodeling, optimization, and robust design.

In the mid 1990s, two specification languages, Modelica [11, 12] and VHDL-AMS [13, 14] were proposed for multi-domain system modeling. Modelica is based on an *object-oriented* paradigm and was developed for modeling of large, complex, heterogeneous physical systems. It is supported by a limited number of computational environments such as OpenModelica [12], Dymola [15, 16], and MathModelica [16]. OpenModelica is an effort which effectively integrates the mathematical formulation and the software architecture using Modelica, and it is developed for modeling and simulation of complex multi-domain systems. It generates explicit mathematical equations for the MDS [12], but the compilation process is complex and includes extensive procedures to generate the system equations. Also, it demands the user to have a deep knowledge of Modelica semantics, and the compiler program is complex, with some 100000 lines of code [17]. Further, its support for parametric studies is limited. VHDL-AMS, on the other hand, is based on a *procedural* approach and focuses on modeling of mixed-signal systems in electrical domains. It is important to note that both Modelica and VHDL-AMS are specification languages *only* and thus require an external computational environment to actually simulate system response.

It follows that the software architecture is a crucial element in the development of simulation and design tools for MDS. In this paper we present MIXEDMODELS (Multidisciplinary Integrated eXtensible Engine Driving Metamodeling, Optimization and DEsign of Large-scale Systems), which is a complete specification-formulation-solution approach for multidisciplinary systems based on a procedural, symbolic-numeric architecture. The proposed method is a strictly acausal, local/global approach that offers great modeling flexibility at the component level and facilitates extension to new problem domains. The formulation uses symbolic computing to formulate the system governing equations and numeric computing to solve these equations. Through this architecture we achieve flexibility and convenience in modeling, along with high efficiency in computation. The independence of the component models also provides extensibility. The

implementation generates explicit equations in symbolic form which will allow convenient extension of this implementation for design sensitivity calculations.

Section 2 presents an overview of the mathematical formulation in MIXEDMODELS which is discussed in [18, 19]. Section 3 presents the software architecture of MIXEDMODELS. Section 4 presents two examples, including a detailed description of (a) how the symbolic engine forms the system governing equations by automatically processing each component, and (b) how the numeric engine solves these equations.

5.3 Mathematical Formulation of MIXEDMODELS

The mathematical formulation of MIXEDMODELS uses a strictly acausal, local/global approach wherein a multidisciplinary system is considered to be a collection of interacting components that belong to different domains. A multidisciplinary system can be completely described by a vector of time-invariant system parameters, **P**; a vector of system variables, **X**, which can also occur in first derivative form $\dot{\mathbf{X}}$ in the governing DAEs; a vector of algebraic system variables, **Y**, that occur algebraically in the governing DAEs; and a set of governing equations (DAEs) written in the following form,

$$\mathbf{f}(\mathbf{P}, \mathbf{X}, \dot{\mathbf{X}}, \mathbf{Y}, t) = \mathbf{0} \tag{5.3.1}$$

Many multidisciplinary systems are described by higher-order differential equations. An n^{th}-order differential equation must be reduced to n first-order differential equations by defining additional $(n - 1)$ variables. These new variables introduced to the system represent the lower order derivatives are referred to as "lower-order variables", and are also part of the **X** vector. Thus, the first derivatives of these additional variables can always be obtained directly from the **X** vector by assignment. The **X** vector can then be partitioned as

$$\mathbf{X} = \begin{bmatrix} \mathbf{X_H} \\ \mathbf{X_L} \end{bmatrix} \tag{5.3.2}$$

where $\mathbf{X_H}$ represents the subvector of **X** such that the derivatives of the elements of $\mathbf{X_H}$ are not contained in **X**. Similarly, $\mathbf{X_L}$ represents the subvector of **X** containing all

the elements of **X** such that their derivative is also an element of **X**, i.e., the lower-order variables. Thus, the lower-order variables can be calculated by a set of direct assignments of the form:

$$\dot{\mathbf{X}}_{\mathbf{L}} = \mathbf{X}' \tag{5.3.3}$$

where $\mathbf{X}'$ is a sub-vector of **X**. Separating Equation (5.3.3) from the system of equations, the rest of the system governing equations in Equation (5.3.1) can be always be written as:

$$\mathbf{A}\left(\mathbf{P},\mathbf{X},t\right)\begin{bmatrix}\dot{\mathbf{X}}\mathbf{H}\\ \mathbf{Y}\end{bmatrix} = \mathbf{b}\left(\mathbf{P},\mathbf{X},t\right) \tag{5.3.4}$$

It should be noted that Equation (5.3.1) may have to be differentiated with time to obtain an equivalent set of equations in the form of Equation (5.3.4).

In order to support the above formulation at the system level, we require the following at the component level.

- A vector of time-invariant component parameters $\mathbf{p}^i$ ∋ $\mathbf{p}^i \in \mathbf{P}$, where **P** is the system parameter vector **P**.
- A vector of transient component differential variables $\mathbf{x}^i$ ∋ $\mathbf{x}^i \in \mathbf{X}$, where **X** is the system differential variable vector. $\mathbf{x}^i$ occurs in the governing DAEs in first derivative form $\dot{\mathbf{x}}^i$.
- A vector of component algebraic variables $\mathbf{y}^i$ which occur algebraically in the DAEs ∋ $\mathbf{y}^i \in \mathbf{Y}$, where **Y** is the system algebraic variable vector.
- A set of component-governing equations which can be expressed as

$$\mathbf{A}^c\left(\mathbf{P},\mathbf{X},t\right)\begin{bmatrix}\dot{\mathbf{X}}_{\mathbf{H}}\\ \mathbf{Y}\end{bmatrix} = \mathbf{b}^c\left(\mathbf{P},\mathbf{X},t\right) \tag{5.5.5}$$

- Direct assignments for lower-order variables at the component level which can be written as

$$\dot{\mathbf{X}}^c_{\mathbf{L}} = \mathbf{X}^{c'} \tag{5.5.6}$$

Modification matrices $\mathbf{A}^m$ and $\mathbf{b}^m$ which allow a component to modify governing equations of other components. Modification matrices can be zero if a component does not modify the governing equations of any other component.

A component can introduce new variables to the system or it can be described in terms of variables of other components. Similarly, it may contribute new equations to the system and/or modify equations contributed by other components. Finally, the contribution of a particular component *i* to the system governing equations can be written in the form

$$\begin{aligned}\mathbf{A}\left(\mathbf{P},\mathbf{X},t\right) &= \mathbf{A}\left(\mathbf{P},\mathbf{X},t\right) + \mathbf{A}_i^m\left(\mathbf{P},\mathbf{X},t\right) + \mathbf{A}_i^c\left(\mathbf{P},\mathbf{X},t\right) \\ \mathbf{b}\left(\mathbf{P},\mathbf{X},t\right) &= \mathbf{b}\left(\mathbf{P},\mathbf{X},t\right) + \mathbf{b}_i^m\left(\mathbf{P},\mathbf{X},t\right) + \mathbf{b}_i^c\left(\mathbf{P},\mathbf{X},t\right)\end{aligned} \tag{5.3.7}$$

where

$\mathbf{A}$ ($\mathbf{P}$,$\mathbf{X}$,t) is the global matrix in the governing equations

$\mathbf{A}^c_{\mathrm{i}}$ ($\mathbf{P}$,$\mathbf{X}$,t) is the contribution of component *i* to the global matrix via new equations contributed by this component

$\mathbf{A}^m_i$ ($\mathbf{P}$,$\mathbf{X}$,t) is the contribution of component *i* to the global matrix via modifications caused by this component to equations contributed by other components

$\mathbf{b}^c_i$ ($\mathbf{P}$,$\mathbf{X}$,t) is the contribution of component *i* to the global RHS vector via new equations contributed by this component

$\mathbf{b}^m_{\mathrm{i}}$ ($\mathbf{P}$,$\mathbf{X}$,t) is the contribution of component *i* to the global RHS vector via modifications caused by this component to equations contributed by other components $\mathbf{P}$ is a vector of time-independent parameters that describe each component in the system

$\mathbf{X}$ is the vector of system differentiable variables.

Thus, a component model is completely described by specifying the entries in the $\mathbf{A}^c$ and $\mathbf{A}^m$ matrices and in the $\mathbf{b}^c$ and $\mathbf{b}^m$ vectors contributed by that component. Once we have the component contributions, they can be combined to obtain the system governing equations as a set of DAEs given by Equations (5.3.3) and (5.3.4).

The proposed formulation offers great modeling flexibility at the component level and accommodates component models of different levels of complexity. Extension of this formulation to new problem domains is easy and straightforward. The proposed formulation scheme requires a software architecture that automatically assembles contributions of individual component into a system of equations, and numerically solves it to generate the system response. The following section presents a symbolic-numeric software architecture developed in the MIXEDMODELS platform to support the above formulation.

5.4 Software Architecture of MIXEDMODELS

The software architecture proposed in this section is based on the procedural paradigm as opposed to the object-oriented architecture proposed in the Modelica specification language. Figure (5.4.1) shows a structural view of the proposed symbolic-numeric architecture; solid arrows denote input-output flow and the block arrow indicates internal files that get executed at runtime.

The description of the system to be simulated is organized in a set of two files. The system specification file specifies the number of components, i.e., the size of the system of interest. The component data file contains descriptions of each component in terms of its string-valued, real-valued and integer-valued parameter arrays. A detailed description of the data file and component files for an example system is provided in Section 5.5. The overall process of symbolic formulation of the system governing equations and their numerical solution within the above architecture can be summarized as follows: The governing equations (i.e., the component models) for each component type are coded in Maple, and the interpretation and combination of the individual component contributions to the system equations is done by the Maple symbolic pre-processor. These equations are then written out explicitly as problem-dependent Fortran code. This problem-dependent code is then compiled and linked with problem independent Fortran solvers and driver code to obtain a complete, stand-alone Fortran program that is specific to this problem. The execution of this program results in the numerical computation of the system response.

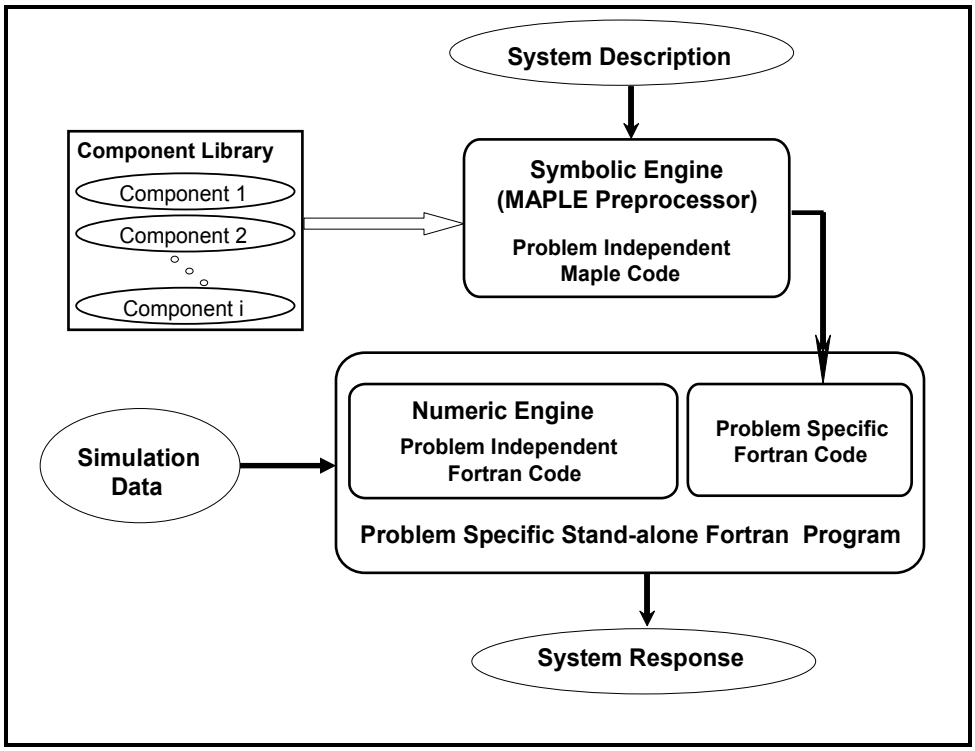


Figure 5.4.1: System Architecture – Structural View

With this general idea of the overall process in mind, we now present a detailed discussion of the architecture, including a functional overview of individual modules. The symbolic-numeric architecture in MIXEDMODELS incorporates three basic modules: (1) a component library written in Maple, (2) a symbolic pre-processing engine written in Maple, and (3) a numeric engine, written in Fortran, that includes a set of problem independent numerical solvers.

5.4.1 Component Library

The component library is a collection of component models. Every component is an independent entity characterized by a set of time invariant parameters, a set of differential and/or algebraic variables, a set of component governing equations, and the component's contributions to the equations of other components. Each component has two files associated with it. The "*header*" file specifies information about the number of differential and/or algebraic variables and the number of equations each component contributes to the system. The "*model*" file describes the behavior of a component by specifying, in symbolic form, the component's

contribution to the system matrix A, the right hand side vector b, and the direct assignment vector $\dot{\mathbf{X}}_L$. Both these files are named using a specific naming convention, which facilitates automatic access to each component within the component library whenever the symbolic processor needs to process a particular component. To ensure modularity, the component's governing equations are written in terms of its local variables, and the symbolic engine automatically maps these local variables to system global variables at runtime. This mapping is done by maintaining arrays that track the offsets between the local and global indices. The component models are written in terms of these offset arrays, and the actual offset values are supplied to the component model code by the preprocessor at runtime. A detailed description of the local-to-global mapping scheme and the system matrix generation scheme is given in Section 5.5.

5.4.2 Symbolic Engine

The symbolic engine is a problem-independent, Maple code which serves as an interface between the component library and the numeric engine. It takes as its input a component data file and a system specification file written as Maple statements and performs the following two functions.

- At runtime it evaluates the offsets required to map components' local variables and equations to the global system variables and system equations.
- Using these offsets, the symbolic engine populates the system matrix **A**, the right hand side vector **b** and the direct assignment vector $\dot{\mathbf{X}}_L$ and explicitly writes them out as problem-specific Fortran code.

The symbolic engine uses a *two-pass* procedure to formulate the system equations. During both passes, the component files for the system at hand are automatically executed from the component library in the order in which they appear in the data file. In the first pass, the symbolic engine scans through the component data file and identifies the name of the component files to be executed at runtime. During this pass the symbolic engine executes only the *header* files for all the components that appear in the component data file supplied by the user. The purpose of this pass is to

calculate the total number of differential and/or algebraic variables, and the total number of system governing equations; this information is sufficient to determine the array space to be allocated for the system matrix and to calculate the offset arrays for local-to-global mapping. Offsets calculated during this internal file execution are maintained by the symbolic engine and are automatically applied to local equations and variables while populating the system matrix from the component models during the second pass.

During the second pass of the procedure, the symbolic engine executes only the *model* files of the components that appear in the user supplied data file. The component models are written in terms of the offsets, which are now known because they are evaluated during the first pass. In the second pass these known offsets automatically get applied and the local information is directly entered in the appropriate rows and columns of the system matrices and vectors. After processing the entire component data file, each individual component model written in terms of its component matrices and modification matrices is mapped to the system matrix **A**, the right hand side vector **b**, and the direct assignment vector $\dot{\mathbf{X}}_{\mathbf{L}}$. Thus, at the end of the second pass, the system matrices and vectors are fully populated.

5.4.3 Numeric Engine

The numeric engine is a problem-independent Fortran driver which incorporates numerical solvers such as an ODE solver and a linear matrix solver. The interface between the numeric engine and the symbolic engine can be briefly explained as follows. The numeric engine calls the problem-specific Fortran program which is generated by the symbolic engine and evaluates the system matrix **A**, the right hand side vector **b**, and the direct assignment vector $\dot{\mathbf{X}}_{\mathbf{L}}$ as given by Equation (5.3.3). The linear matrix solver then solves the system given by Equation (5.3.4) to obtain $\left[\dot{\mathbf{X}}_{\mathbf{H}} \;\; \mathbf{Y}\right]^{\mathrm{T}}$. Then, the ODE solver integrates $\left[\dot{\mathbf{X}}_{\mathbf{H}} \;\; \dot{\mathbf{X}}_{\mathbf{L}}\right]^{\mathrm{T}}$ to obtain the differential variables. In short, the problem-independent Fortran code is compiled and linked with the problem-specific Fortran code to get a stand-alone Fortran program which is then executed to obtain the system response.

The user needs to provide information about simulation parameters, initial conditions and error tolerances for the numerical solvers. The current implementation uses a stiff integrator DLSODES [5.7.20] based on a sparse matrix version of Gear's algorithm and a linear sparse matrix solver Y12MAF [5.7.20] which solves the system using Gaussian elimination. The fact that the formulation generates explicit mathematical equations makes this scheme flexible for the user to choose between different numerical solvers or even to write new solvers. It also allows extension of this architecture for parametric studies such as sensitivity analysis and optimization of multidisciplinary systems.

5.5 Examples

5.5.1 DC Motor Driving a Single Link through a Gearbox

Figure (5.5.1) shows an example system, viz., a DC motor driving a fixed-axis link through a gearbox, taken from [5.7.18].

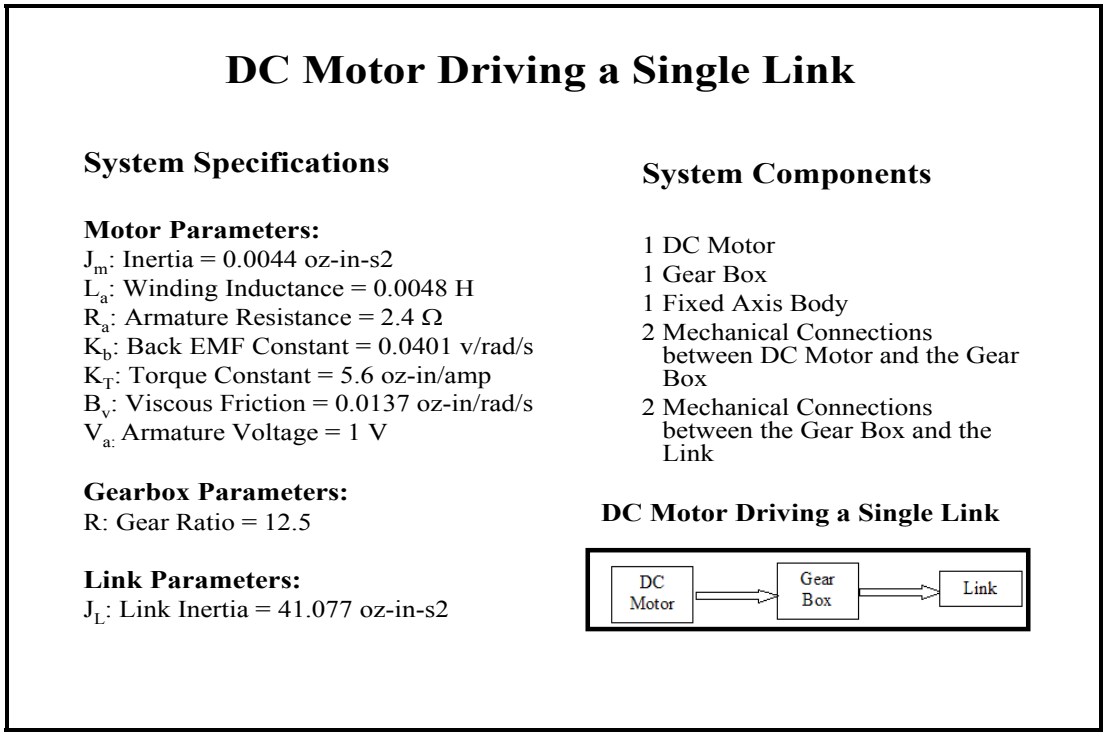


Figure 5.5.1: DC Motor Driving a Fixed-Axis Link – System Specification

For simplicity and clarity in explanation, we will drive the DC motor at a constant armature voltage (V_a) of 1 V, and include (V_a) in the parameter list of the DC motor

component. The system thus formed has the following components - DC motor, gear box, link, connections between the motor and the gearbox, and connections between the gearbox and the link.

Before deriving the system equations let us consider the DC motor component as an example of a component. The DC motor component is described locally in terms of its own component variables, governing equations and time invariant parameters as seen in Figure (5.5.2).

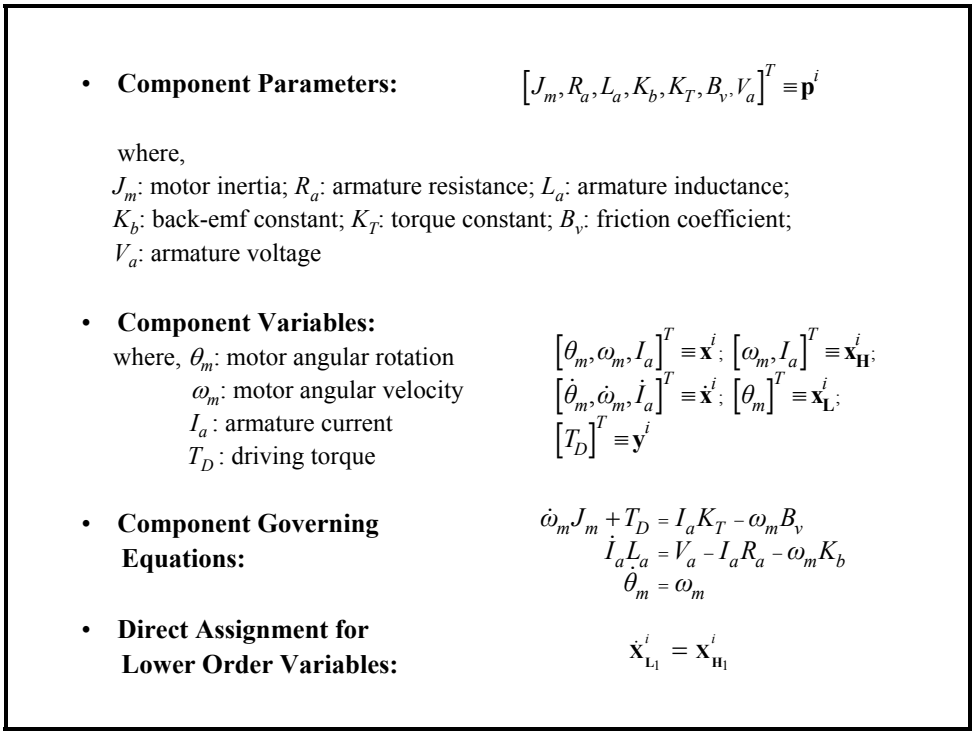


Figure 5.5.2: DC Motor – An Electromechanical Component

The Maple code for the DC motor *model* file is compact and is written directly in terms of the offsets required to map the local variables to the global variables as given below.

Here, *i* represents the component index and *PR* represents the real-valued parameter array, $\mathbf{p}^i$, for the DC motor component. *offA*, *offXH, offXL* and *offY* represent offsets for local-to-global mapping of the governing equations, higher-order differential variables, lower-order differential variables, and algebraic variables respectively; the values of these offsets are set by the Maple symbolic engine before the *model* file is called.

A[offA[i],offXH[i]] := PR[i,1]:

A[offA[i],offY[i]] := 1:

A[offA[i]+1,offXH[i]+1] := PR[i,3]:

b[offA[i]] := -**XH**[offXH[i]]*PR[i,6] + **XH**[offXH[i]+1]*PR[i,5]:

b[offA[i]+1] := PR[i,7] - **XH**[offXH[i]]*PR[i,4] - **XH**[offXH[i]+1]*PR[i,2]:

Returning now to the complete example system, all of the other system components of the system will be described locally in terms of their own matrix/RHS contributions, differential and/or algebraic variables, and time invariant parameters. The equations from all the components of the system are shown below:

5.5.1.1 DC Motor

$$\begin{aligned} \dot{x}_{H_1} J_m + y_1 &= -x_{H_1} B_v + x_{H_2} K_T \\ \dot{x}_{H_2} L_a &= V_a - x_{H_1} K_b - x_{H_2} R_a \\ \dot{x}_{L_1} &= x_{H_1} \end{aligned} \tag{5.5.1}$$

5.5.1.2 Gear Box

$$\begin{aligned} y_3 - (1/R)*y_2 &= 0 \\ y_5 - (R)*y_4 &= 0 \end{aligned} \tag{5.5.2}$$

5.5.1.3 Fixed Axis Link

$$\begin{aligned} \dot{x}_{L_2} &= x_{H_3} \\ \dot{x}_{H_3} J_L - y_6 &= 0 \end{aligned} \tag{5.5.3}$$

5.5.1.4 Connections between Motor and Gearbox - C_{MG}

$$\begin{aligned} y_2 - \dot{x}_{H_1} &= 0 \\ y_4 - y_1 &= 0 \end{aligned} \tag{5.5.4}$$

5.5.1.5 Connections between Gearbox and Link - C_{GL}

$$\begin{aligned} y_3 - \dot{x}_{H_3} &= 0 \\ y_6 - y_5 &= 0 \end{aligned} \tag{5.5.5}$$

The component data file for this example specifies string-valued, real-valued and integer-valued parameter arrays for each component. The file is written as Maple statements and is as shown in Figure (5.5.3).

```
PS[1, 1] := "DCMotorVer3":
PS[1, 2] := "DCM1":
PS[1, 3] := "NE":
PS[1, 4] := "Ngnd":
PR[1, 1] := 0.0044:
PR[1, 2] := 2.4:
PR[1, 3] := 0.0048:
PR[1, 4] := 0.0401:
PR[1, 5] := 5.6:
PR[1, 6] := 0.0137:

PS[2,1] := "GearBox":
PS[2,2] := "GB1":
PR[2,1] := 12.5:

PS[3, 1] := "FixedAxisBody":
PS[3, 2] := "FAB1":
PR[3, 1] := 41.0767:

PS[4,1] := "ConnectionConstraint":
PS[4,2] := "CC_MG11":
PS[4,3] := "DCM1":
PS[4,4] := "GB1":
PI[4,1] := 1:
PI[4,2] := 0:
PI[4,3] := 0:
PI[4,4] := 1:

PS[5,1] := "ConnectionConstraint":
PS[5,2] := "CC_MG12":
PS[5,3] := "DCM1":
PS[5,4] := "GB1":
PI[5,1] := 0:
PI[5,2] := 1:
PI[5,3] := 0:
PI[5,4] := 3:

PS[6,1] := "ConnectionConstraint":
PS[ 6,2] := "CC_GL11":
PS[6,3] := "GB1":
PS[6,4] := "FAB1":
PI[6,1] := 0:
PI[6,2] := 2:
PI[6,3] := 1:
PI[6,4] := 0:

PS[7,1] := "ConnectionConstraint":
PS[7,2] := "CC_GL12":
PS[7,3] := "GB1":
PS[7,4] := "FAB1":
PI[7,1] := 0:
PI[7,2] := 4:
PI[7,3] := 0:
PI[7,4] := 1:
```

Figure 5.5.3: Component Data File for DC Motor Driving a Fixed-Axis Link

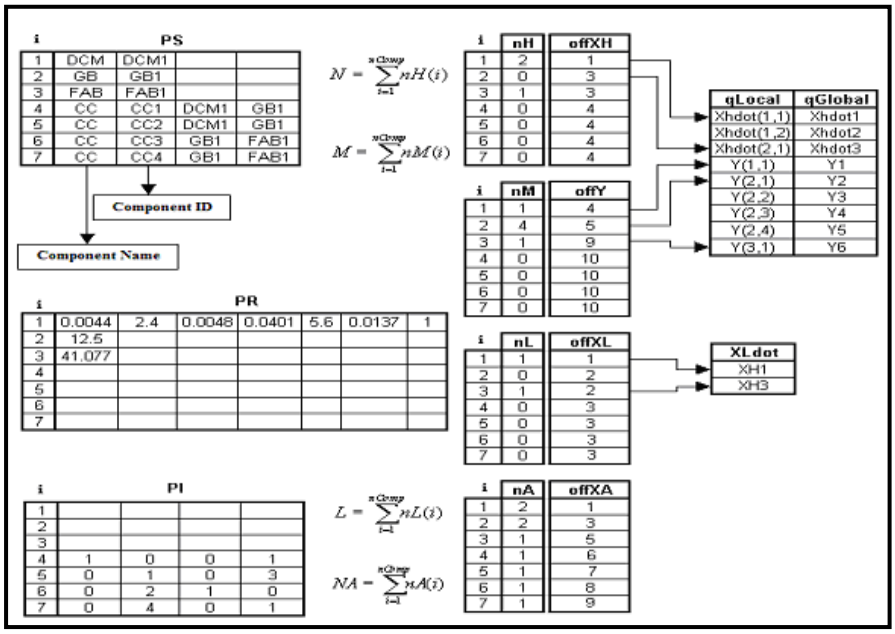


Figure 5.5.4: Memory Map for a DC Motor Driving a Fixed-Axis Link

During the first pass, the symbolic engine scans this data file and executes *header* files of all the components in the list. The symbolic engine then populates the parameter arrays, $\mathbf{p}^i$, of the components and calculates the offsets *offA*, *offXH, offXL* and *offY* required to map the local component variables to global system variables. The memory map at the end of the first pass is shown in Figure (5.5.4). During the first pass the symbolic engine also allocates space for the system matrices and vectors which get fully populated during the second pass. Figure (5.5.5) shows the local-to-global mapping, the system matrix, **A**, and the right hand side vector, **b**, for the above example.

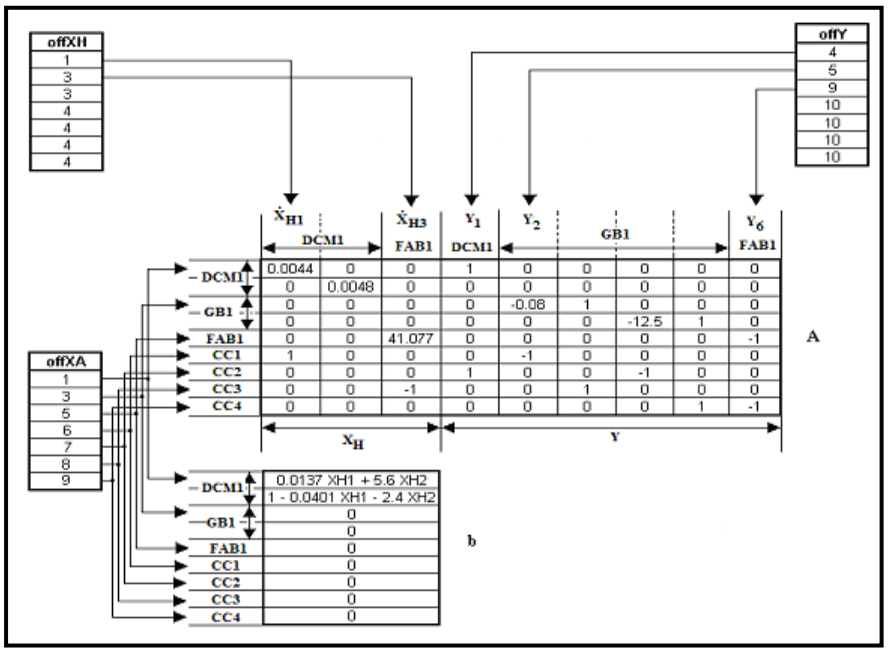


Figure 5.5.5: System Matrices with Local-to-Global Mapping

Using this information, the symbolic processor generates problem-specific symbolic expressions as depicted in Figure (5.5.6), which it then converts to problem-specific Fortran code as shown by Figure (5.5.7). Figure (5.5.6) lists all the system variables and equations; here $\mathbf{X_H}$ is the vector of higher-order differential variables, $\mathbf{X_L}$ is the vector of lower-order differential variables and **Y** is the vector of algebraic variables.

The problem-independent Fortran code, which includes the ODE solver DLSODES [5.7.20] and the linear equation solver Y12MAF [5.7.20], is compiled and linked

with the problem-specific Fortran code generated by the symbolic preprocessor. The resulting stand-alone Fortran program is then executed to obtain the system response.

$$\begin{bmatrix} 0.0044 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0.0048 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & -0.08000000000 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & -12.5 & 1 & 0 \\ 0 & 0 & 41.0767 & 0 & 0 & 0 & 0 & 0 & -1 \\ 1 & 0 & 0 & 0 & -1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & -1 & 0 & 0 \\ 0 & 0 & -1 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & -1 \end{bmatrix} \begin{bmatrix} XHD_1 \\ XHD_2 \\ XHD_3 \\ Y_1 \\ Y_2 \\ Y_3 \\ Y_4 \\ Y_5 \\ Y_6 \end{bmatrix} = \begin{bmatrix} -0.0137\,XH_1 + 5.6\,XH_2 \\ 1 - 0.0401\,XH_1 - 2.4\,XH_2 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix}$$

$$XLD_1 = XH_1$$
$$XLD_2 = XH_3$$

Higher Order Differential Variables

XH_1 = Motor Angular Velocity
XH_2 = Motor Armature Current
XH_3 = Link Angular Velocity

Lower Order Differential Variables

XL_1 = Motor Angular Rotation
XL_2 = Link Angular Rotation

Algebraic Variables

Y_1 = Motor Driving Torque
Y_2 = Acceleration Input to Gearbox
Y_3 = Acceleration Output of Gearbox
Y_4 = Torque Input to Gearbox
Y_5 = Torque Output of Gearbox
Y_6 = Load Torque

Figure 5.5.6: Problem-Specific Maple Expressions

(Generated by the Symbolic Engine)

```
A(1,1) = 0.44D-2
A(1,4) = 1
A(2,2) = 0.48D-2
A(3,5) = -0.8000000000D-1
A(3,6) = 1
A(4,7) = -0.125D2
A(4,8) = 1
A(5,3) = 0.410767D2
A(5,9) = -1
A(6,1) = 1
A(6,5) = -1
A(7,4) = 1
A(7,7) = -1
A(8,3) = -1
A(8,6) = 1
A(9,8) = 1
A(9,9) = -1

b(1) = -0.137D-1 * XH(1) + 0.56D1 * XH(2)
b(2) = 0.1D1 - 0.401D-1 * XH(1) - 0.24D1 * XH(2)

XLDot(1) = XH(1)
XLDot(2) = XH(3)
```

Figure 5.5.7: Problem-Specific Fortran Code Fragments for the DC Motor Driving a Fixed-Axis Link

Figure (5.5.8) shows the system responses for angular rotation, angular velocity, armature current and driving torque of the DC motor. The results were validated against Simulink and the diagram shows that there was very good agreement between the two results.

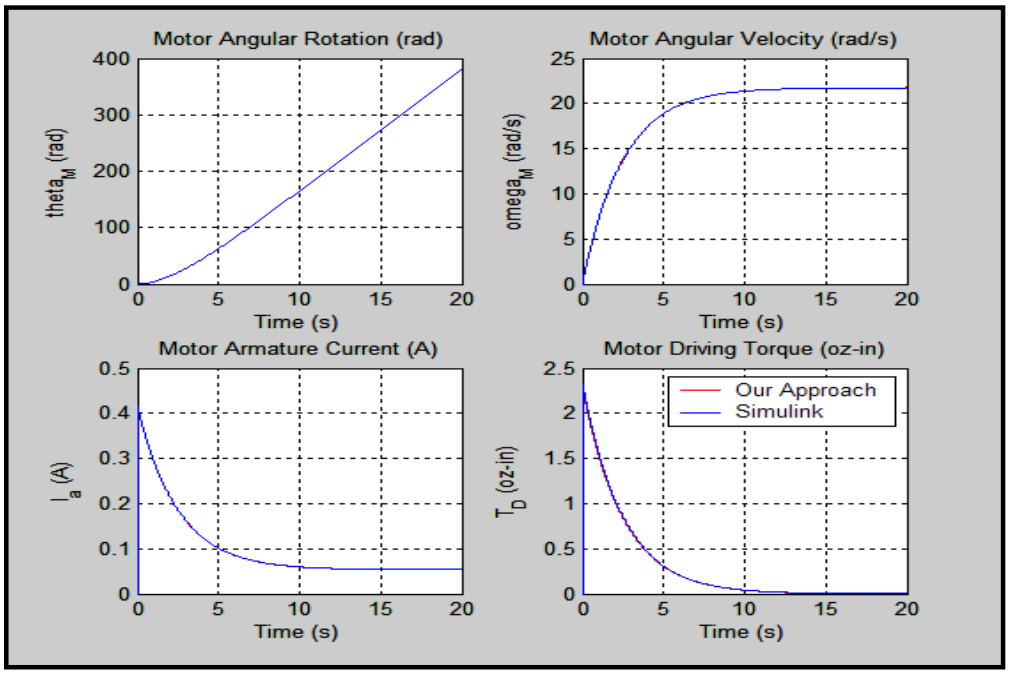


Figure 5.5.8: Simulation Results for DC Motor Driving a Fixed-Axis Link

5.5.2 DC Motor, Powered by a Full-Bridge Rectifier, Driving a Fixed-Axis Link

In this example, we replace the constant voltage source in Example (5.5.1) with a full-bridge rectifier as shown in Figure (5.5.9). The rest of the system remains the same. The simulation results in Figure (5.5.10) were validated against P-Spice and Simulink and were observed to be in close agreement. The p-n junction diode used here has the same parameters as the PSpice diode D1N4148 and as listed below.

5.5.2.1 Bipolar Junction Diode (PSpice D1N4148)

η = emission coefficient = 1.836

V_t = thermal voltage = 0.026 V

I_s = saturation current = 2.682 nA

B_R = reverse breakdown voltage = 100 V

d = grading coefficient = 0.3333

τ_t = minority carrier lifetime = 11.54 ns

C_{j0} = zero bias depletion capacitance = 4 pF

ϕ_0 = built in potential = 0.5 V

R_b = junction ohmic resistance = 0.5664 Ω

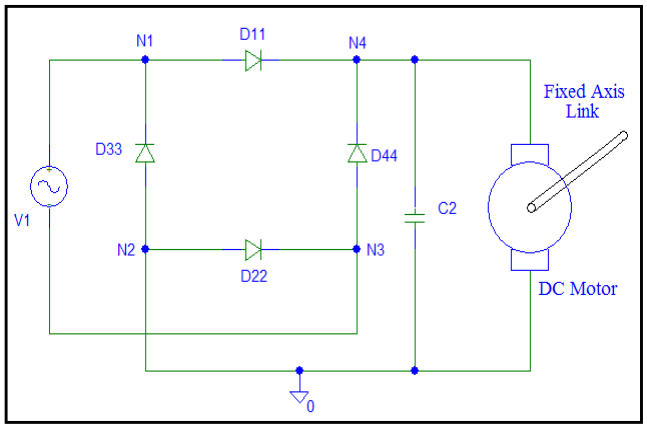


Figure 5.5.9: DC Motor, Powered by a Full-Bridge Rectifier Driving a Fixed-Axis Link

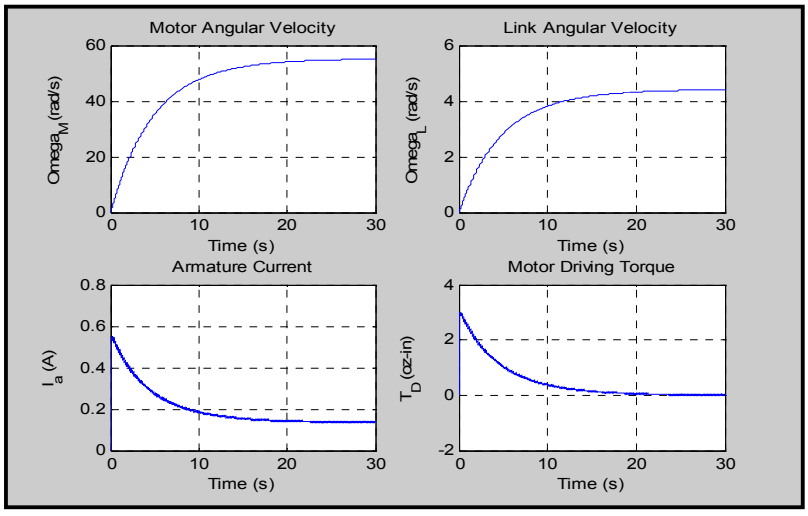


Figure 5.5.10: Simulation Results for DC Motor, Powered by a Full-Bridge Rectifier Driving a Fixed-Axis Link

5.6 Conclusion

This paper presented a mathematical formulation and a corresponding procedural, symbolic-numeric software architecture for the modeling and simulation of multidisciplinary systems. This architecture has been successfully implemented in the MIXEDMODELS platform. This symbolic-numeric architecture has several advantages that make it very attractive. Since each component model is in a separate file that is independent of other components, we have a high degree of modularity. By using symbolic computing to formulate the governing equations and numeric computing to solve these equations, we achieve great flexibility and convenience in modeling, along with efficiency in computation. The independence of the component models also provides easy extensibility. Extensibility is further enhanced by a file-naming convention that permits components to be added without the need for any modifications in any existing code; the new components are automatically included if the description of the system at hand refers to these components. It is also seen from the examples that the code is very compact and is hence easily maintainable. This is in sharp contrast to object-oriented approaches such as openModelica that generally lead to lengthy code. Finally, the fact that explicit equations are generated in symbolic form by the preprocessor will allow convenient extension of this implementation for design sensitivity calculations.

To demonstrate the efficacy of the proposed symbolic-numeric formulation and software architecture, we presented two examples including a detailed description of a component model and how it fits in the complete system of DAEs. The first example is a DC motor driving a single link, where the motor is driven by a constant voltage DC power supply. In the second example the same system is simulated with the DC motor driven by a full-bridge rectified DC power supply. The system is simulated using numerical solvers in Fortran, and the time responses are validated against Simulink and PSpice. The results indicate that the proposed approach and software architecture is effective in terms of accuracy, modeling convenience, computational efficiency and the ability to simulate the behavior of multidisciplinary multibody systems.

5.7 REFERENCES

5.7.1 Scherrer, M. and McPhee, J., '*Dynamic Modelling of Electromechanical Multibody Systems*', Multibody System Dynamics, 9, pp. 87-115, 2003.

5.7.2 Granda, J.J., '*The Role of Bond Graph Modeling and Simulation in Mechatronic Systems. An Integrated Software Tool: Camp-G, MATLAB-Simulink*' Proceedings of Mechatronics Conference, Atlanta, Georgia, pp. 1271-1295, 2000.

5.7.3 Taehyun, S., '*Introduction to Physical System modeling Using Bond Graphs*', Vetronics Institute 2nd Annual Workshop Series, pp. 430-434, Dec. 2000.

5.7.4 Greenwood, D.T., '*Principles of Dynamics, Second Edition*', Prentice Hall, 1988.

5.7.5 Carrigan, J., '*Integrated Design and Sensitivity Based Design and Optimization of Nonlinear Controlled Multibody Mechanisms*', Thesis, M.S. Iowa State Univ., 2003.

5.7.6 Huston, R.L., '*Multibody Dynamics*', Butterworth-Heinemann, 1990.

5.7.7 http://www.amesim.com/, IMAGINE Software Inc., acquired on Jan 29, 2005.

5.7.8 Ram, O., Szymkat, M., Uhl, T., Betemps, M., Pjetursson, A. and Rod, J., '*Mechatronic Blockset for Simulink – Concept and Implementation*', Proceedings of the 1996 IEEE Intl. Symposium on Computer-Aided Control System Design, pp. 530-535, 1996.

5.7.9 http://www.mathworks.com/, acquired on Jan 29, 2005.

5.7.10 Knorr, U.I., '*Electromechanical System Design*', Ansoft Corporation, 2002.

5.7.11 Fritzson, P., '*Principles of Object-Oriented Modeling and Simulation with Modelica 2.1*', IEEE Press, A John Wiley and Sons, Inc., Publication, 2004.

5.7.12 Fritzson, P., Aronsson, P., Pop, A., Lundvall, H., Nystrom, K., Saldamli, L., Broman D., Sandholm, A. '*OpenModelica – A Free Open-Source Environment*

for System Modeling, Simulation, and Teaching', Proceedings of IEEE Conference on Computer Aided Control Systems Design, pp. 1588-1595, 2006.

5.7.13 Design Automation Standards Committee of the IEEE Computer Society, '*IEEE Standard VHDL Analog and Mixed-Signal Extensions*', IEEE-SA Standards Board, Mar. 1999.

5.7.14 Frey, P., Nellayappan, K., Shanmugasundaram, V., Mayiladuthurai, R.S., Chandrashekhar, C.L., Carter, H.W., '*SEAMS: Simulation Environment for VHDL-AMS*', Proceedings of the 1998 Winter Simulation Conference, pp. 539-546, 1998.

5.7.15 Fritzson, P., Bunus, P., '*Modelica – A General Object-Oriented Language for Continuous and Discrete-Event System Modeling and Simulation*', Proceedings of the 35th Annual Simulation Symposium, pp. 365-380, IEEE 2002.

5.7.16 Fritzson, P., Gunnarsson, J., Jirstrand, M., '*MathModelica – An Extensible Modeling and Simulation Environment with Integrated Graphics and Literate Programming*', 2nd International Modelica Conference, Proceedings, pp. 41-54, Mar. 2002.

5.7.17 Larsson, J., Fritzson, P., '*A Modelica-based Format for Flexible Modelica Code Generation and Causal Model Transformations*', The Modelica Association, pp. 467-475, 2006.

5.7.18 Vaze S., Krishnaswami, P., DeVault, J., '*Component Based Modeling of Electromechanical Systems*', Proceedings of IDETC/CIE, Vol. 2, 2005.

5.7.19 Vaze S., DeVault, J., Krishnaswami, P., '*Modeling of Hybrid Electromechanical Systems using a Component-based Approach*', IEEE International Conference on Mechatronics and Automation, ICMA 2005, pp. 204-209, 2005.

5.7.20 http://www.netlib.org/, Netlib Repository, acquired on May 15, 2005.

CHAPTER 6 - METAMODELING OF MECHATRONIC SYSTEMS IN THE MIXEDMODELS PLATFORM

Manuscript Publication:

Proceedings of IDETC/CIE 2007:

ASME 2007 International Design Engineering Technical Conferences & Computers and Information in Engineering Conference

September 4-7, 2007, Las Vegas, Nevada, USA

Authors:

Shilpa A. Vaze[†], Prakash Krishnaswami[*], and James E. DeVault[†].

Authors' Affiliations:

[†] Department of Electrical and Computer Engineering, Kansas State University, Manhattan, KS – 66506.

[*] Department of Mechanical and Nuclear Engineering, Kansas State University, Manhattan, KS – 66506.

Keywords:

Metamodeling, Mechatronic Systems, MIXEDMODELS

NOTE: This chapter is largely based on the original publication, however; changes have been made to the manuscript to maintain continuity and consistency in reading.

6.1 Abstract

The parametric design of mechatronic systems requires several detailed analyses of the system, thereby slowing down the design process significantly. In the recent past, there has been a lot of interest in using lower-fidelity, but higher-efficiency metamodels (also called surrogate models) instead of the actual detailed models to guide parametric design, particularly in the early stages of parametric design. One common approach to forming metamodels is to run the detailed model to obtain the system response at selected points in design space and fit a response surface to the results which becomes the metamodel. Since this method uses only zero-order information at each design point, a large number of points are required to form a reasonably accurate metamodel. For example, in a single design variable problem, a two-point response surface can only be linear, whereas we can generate a cubic response surface if we also had derivative information at the two points.

In this paper, we present a metamodeling approach for mechatronic systems that computes and utilizes first-order derivative information at each point in the design space at which a detailed analysis is performed. The first-order derivative information that is computed is the set of design sensitivity coefficients of the system state variables and performance functions. A unified modeling approach for the mechanical, electrical, and electronic aspects of the system is first developed. This approach generates a single set of governing equations for the entire system in the form of a system of differential-algebraic equations (DAEs). Based on these DAEs, a set of equations in the state design sensitivity coefficients is analytically derived using a direct differentiation approach. This set of equations also turns out to be a set of DAEs which can be solved simultaneously in parallel with the system governing equations. We have successfully implemented this methodology for design sensitivity analysis of multidisciplinary systems in a computational platform called MIXEDMODELS (Multidisciplinary Integrated eXtensible Engine for Driving Metamodeling, Optimization, and DEsign of Large-scale Systems). Once we know the state design sensitivity coefficients, we can compute the design sensitivity coefficients of any system performance function. After we have obtained the necessary design sensitivity information, we can devise several schemes for

generating a metamodel for the system based on the sensitivity information. Some examples of metamodels obtained using this approach are presented for selected mechatronic systems, along with the relevant accuracy measures.

6.2 Introduction

6.2.1 Current State-of-the-Art

As mechatronic systems become more and more interdisciplinary, it has become necessary to develop better methods for analysis and design of these systems. It has been clearly shown that traditional sequential and modular approaches to the design of multidisciplinary systems often lead to system designs that are suboptimal [6.6.1-6.6.3]. This is mainly due to the extensive interaction between different domains such as electrical-electronic, mechanical, hydraulic, pneumatic etc, that the sequential design approach ignores by decoupling the subsystems. For example, in many mechatronic systems, the variables of the electrical subsystems may affect the performance of the mechanical subsystem; hence, sensitivities of performance functions from the mechanical domain need to be calculated with respect to the design variables in the electrical domain in order to optimize the design of the entire system. Quantitative examples of such interactions can be found in [6.6.4-6.6.8]. The interaction between subsystems cannot be captured in a sequential design approach. Thus, there is now a lot of interest in the development of integrated design approaches that treat all aspects of the mechatronic system concurrently [6.6.4-6.6.9].

One of the difficulties in implementing integrated design is that it requires integrated analysis of the system. This can be computationally very expensive, and may slow down the design process. A popular approach to handling this difficulty is the use of metamodels [6.6.10] (also called surrogate models). Metamodels are approximate models of the system that may not have the fidelity of a detailed system model, but are computationally efficient and accurate enough to drive design [6.6.11-6.6.14]. This is particularly true at the initial stages of parametric design, where several design changes are often made in a short time. It is also possible to have a hierarchy

of metamodels such that a very efficient low-fidelity metamodel can be used in early stage design, while metamodels with lower efficiency and higher fidelity can be used in later design iterations [6.6.15]. Of course, the detailed system model is still required for final tuning and proving of the design.

One of the most common approaches to metamodeling is to fit a response surface based on knowing the values of the system response at certain known points in design space. This is a simple and effective approach, and has been used successfully in several applications [6.6.16, 6.6.17], but it may require several points in order to obtain a useably accurate metamodel. However, if design sensitivity information is also available at the known points in design space, then it becomes possible to fit construct more accurate metamodels using the same number of known design points. This is the approach explored in this paper.

In order to perform sensitivity-based metamodeling, it is first necessary to devise a suitable formulation for analysis and design sensitivity analysis that can be applied to a wide class of mechatronic systems. One approach for modeling is to apply a single physical/mathematical formulation [6.6.18, 6.6.19] such as Lagrangian, Bond Graph Theory, Linear Graph Theory, State Space etc. to derive governing equations of a multidisciplinary system. However these formulations are generally well-suited in particular application domain(s) and are not easily extensible to accommodate new domains. Nevertheless, researchers have been very successful in applying this approach in certain domains. NEWOPT/AIMS [6.6.6] is a software package that includes algorithms for simulation, sensitivity analysis and optimization of a class of mechatronic systems based on a linear state-space model.

Another strategy is to develop different software packages for each relevant problem domain and then integrate these software packages as needed to model an MDS. These domain-specific packages then communicate with each other to simulate the system behavior. MATLAB/Simulink [6.6.20], Simplorer [6.6.21], and AMESim [6.6.22], are some of the existing tools that fall under this category. While this approach provides a workable solution in many cases, its main drawback is that it does not yield an explicit mathematical formulation of the system governing

equation which can be used efficiently in analytical design sensitivity analysis and optimization.

The third approach that is becoming popular can be categorized as a declarative or equation-based approach which emphasizes integration across domains at the level of system governing equations. An interesting application of this approach in the automotive field is presented in [6.6.4], where the authors use this strategy to globally model, simulate and optimize complex industrial mechatronic systems using MATLAB-Simulink and a Finite Element Method. The equation-level integration approach, also called the component approach, is also the central idea behind the Modelica object-oriented specification language for multidisciplinary systems [6.6.23, 6.6.24]. It has been shown in [6.6.25-6.6.27] that this approach can be used to develop an effective analysis and sensitivity analysis tool for MDS modeling using mixed symbolic-numeric software architecture. However, this formulation shares a drawback with the linear state space methods when it comes to dealing systems which may include components whose governing equations include a large number of nonlinear algebraic or differential equations. This is often the case when electronic, hydraulic and pneumatic components are involved. The linear state space formulation and the formulation in [6.6.25, 6.6.27] require that these equations be converted to a prespecified form by a process of differentiation. This has the undesirable effect of artificially increasing the number of differential equations in the system, and generally making the numerical solution much harder; it is even possible that a non-stiff system may appear numerically stiff in the differentiated form.

It follows that in order to investigate sensitivity-based metamodeling of mechatronic systems; we first need to develop a viable computational formulation for calculating the required design sensitivity information for a general mechatronic system. We take a broad view of mechatronic systems to include the possibility that the system may include subsystems and components from several application domains, such as hydraulics, pneumatics, etc. In order to achieve the desired breadth and flexibility in modeling the systems of interest, we propose the modeling approach that has been successfully implemented in the MIXEDMODELS (Multidisciplinary Integrated eXtensible Engine for Driving Metamodeling, Optimization and DEsign of Large-

scale Systems) platform. MIXEDMODELS is a unified analysis and design tool for multidisciplinary systems that utilizes a component-based formulation [6.6.26, 6.6.27]. It is a very flexible, extensible, and compact platform that is based on a procedural, symbolic-numeric software architecture [6.6.27]. We also discuss the extension of the basic MIXEDMODELS formulation to include design sensitivity analysis.

6.3 The MIXEDMODELS Formulation

The goal of the MIXEDMODELS formulation is to develop a numerically robust and viable approach that allows us to analyze all the aspects of a general class of multidisciplinary systems, and also provides built-in support for parametric studies such as design sensitivity analysis, optimization, and metamodeling.

6.3.1 Analysis Formulation

The modeling approach presented in the MIXEDMODELS platform considers a multidisciplinary system to be a collection of interacting components, and can be completely described by a vector of time-invariant system parameters **P**; a vector of system variables **X**, which can also occur in the first derivative form $\dot{\mathbf{X}}$ in the DAEs; a vector of algebraic variables **Y** that can occur algebraically in the set of DAEs; and a set of governing (DAEs) of the following form:

$$\mathbf{F}(\mathbf{P}, \mathbf{X}(\mathbf{d}, t), \dot{\mathbf{X}}(\mathbf{d}, t), \mathbf{Y}(\mathbf{d}, t), \mathbf{d}, t) = 0 \tag{6.3.1}$$

where, **d** is a vector of design variables.

Many multidisciplinary systems are governed by higher-order differential equations. An n^{th}-order differential equation must be reduced to n first-order differential equations by defining additional variables. The (n-1) new variables that are introduced to represent the lower-order derivatives are referred to as "lower-order variables", and are also part of the **X** vector. Thus, the first derivatives of these additional variables can always be obtained directly from the **X** vector by assignment. The **X** vector can then be partitioned as

$$\mathbf{X} = \begin{bmatrix} \mathbf{X}_{\mathbf{H}} \\ \mathbf{X}_{\mathbf{L}} \end{bmatrix} \tag{6.3.2}$$

where, $\mathbf{X_H}$ represents the subvector of $\mathbf{X}$ such that the derivatives of the element of $\mathbf{X_H}$ are not contained in $\mathbf{X}$. Similarly, $\mathbf{X_L}$ represents the subvector of $\mathbf{X}$ containing all the elements of $\mathbf{X}$ such that their derivative is also in element of $\mathbf{X}$, i.e. the lower order variables. Thus, the lower order variables can be calculated by a set of direct assignments of the form:

$$\dot{\mathbf{X}}_{\mathbf{L}} = \mathbf{X}' \tag{6.3.3}$$

where, $\mathbf{X}'$ is a subvector of $\mathbf{X}$. Separating Equation (6.3.3) from the system of equations given by Equation (6.3.1), the rest of the system governing equations can be written as:

$$\mathbf{f}(\mathbf{P}, \mathbf{X}(\mathbf{d},t), \dot{\mathbf{X}}_{\mathbf{H}}(\mathbf{d},t), \mathbf{Y}(\mathbf{d},t), \mathbf{d}, t) = 0 \tag{6.3.4}$$

Equation (6.3.3) along with Equation (6.3.4) describes the complete system of equations.

In the component-based modeling view, each component adds its contribution to the system governing equations to form the complete system of equations. A component can introduce new system variables, or it can be described in terms of system variables contributed by other components. It may add new governing equations to the system and/or modify system equations contributed by other components. In order to support the above formulation at the system level, we require some information to be provided at the component level whenever a new component class is defined. Specifically, if a component is component *i* in the system, we require the following for that component:

- A vector of time-invariant component parameters $\mathbf{p}^i$.
- A vector of transient component differential variables $\mathbf{x}^i$. These $\mathbf{x}^i$ occur in the governing DAEs in first derivative form $\dot{\mathbf{x}}^i$.

- A vector of component algebraic variables $\mathbf{y}^i$ which occur algebraically in the DAEs**.**
- In addition, the component model must provide the following:
- A set of component governing equations which can be written in the following nonlinear form:

$$\mathbf{f}_i^c(\mathbf{P}, \mathbf{X}(\mathbf{d},t), \dot{\mathbf{X}}_{\mathbf{H}}(\mathbf{d},t), \mathbf{Y}(\mathbf{d},t), t) = 0 \tag{6.3.5}$$

- A set of direct assignments given by

$$\dot{\mathbf{X}}_{\mathbf{L}_i}^c = \mathbf{X}^{c'} \tag{6.3.6}$$

- A set of component modification equations of the form

$$\mathbf{f}_i^m(\mathbf{P}, \mathbf{X}(\mathbf{d},t), \dot{\mathbf{X}}_{\mathbf{H}}(\mathbf{d},t), \mathbf{Y}(\mathbf{d},t), t) = 0 \tag{6.3.7}$$

These equations describe the modifications that this component makes in the equations of other components.

The contributions of all the components in the system are summed to obtain the system governing equations

$$\mathbf{f}(\mathbf{P}, \mathbf{X}, \dot{\mathbf{X}}_{\mathbf{H}}, \mathbf{Y}, t) = \sum_i \left[\mathbf{f}_i^c(\mathbf{P}, \mathbf{X}, \dot{\mathbf{X}}_{\mathbf{H}}, \mathbf{Y}, t) + \mathbf{f}_i^m(\mathbf{P}, \mathbf{X}, \dot{\mathbf{X}}_{\mathbf{H}}, \mathbf{Y}, t) \right] \tag{6.3.8}$$

Similarly, direct assignment contributions from each component are added to get the set of system direct assignment equations in Equation (6.3.3). Thus, we get the complete set of system DAEs given by,

$$\begin{aligned} \mathbf{f}(\mathbf{P}, \mathbf{X}(\mathbf{d},t), \dot{\mathbf{X}}_{\mathbf{H}}(\mathbf{d},t), \mathbf{Y}(\mathbf{d},t), \mathbf{d}, t) &= 0 \\ \dot{\mathbf{X}}_{\mathbf{L}}(\mathbf{d},t) &= \mathbf{X}'(\mathbf{d},t) \end{aligned} \tag{6.3.9}$$

The ODEs in Equation (6.3.9) can be solved to obtain the differential variable vector **X** using any suitable ODE solver (the work reported in this paper uses the DLSODES solver). To be able to do so, at any time *t*, given the current estimate of

the dependent variable vector $\mathbf{X}$ in the ODEs, we need a way to calculate the derivatives $\dot{\mathbf{X}} = \begin{bmatrix} \dot{\mathbf{X}}_{\mathbf{H}} & \dot{\mathbf{X}}_{\mathbf{L}} \end{bmatrix}^{\mathrm{T}}$. $\dot{\mathbf{X}}_{\mathbf{L}}$ are calculated directly by using Equation (6.3.3). $\dot{\mathbf{X}}_{\mathbf{H}}$ and $\mathbf{Y}$ are calculated using Newton-Raphson iteration as summarized below.

Let us define a vector $\mathbf{q}$ as

$$\mathbf{q} = \begin{bmatrix} \dot{\mathbf{X}}_{\mathbf{H}} \\ \mathbf{Y} \end{bmatrix}_{(N+M)\times 1} \tag{6.3.10}$$

where, N denotes the number of higher order differential variables, M denotes the number of algebraic variables.

Given the initial conditions $\mathbf{X}$, and initial guesses on $\dot{\mathbf{X}}_{\mathbf{H}}$ and $\mathbf{Y}$, we can iteratively calculate

$$\mathbf{q}^{k+1} = \mathbf{q}^{k} - \mathrm{J}^{-1}(\mathbf{q}^{k})\mathbf{f}(\mathbf{P}, \mathbf{X}, \mathbf{q}^{k}, t) \tag{6.3.11}$$

where, J is the Jacobian matrix given by,

$$\mathrm{J}(\mathbf{P}, \mathbf{X}, \mathbf{d}, t) = \begin{bmatrix} \dfrac{\partial \mathbf{f}}{\partial \dot{\mathbf{X}}_{\mathbf{H}}} & \dfrac{\partial \mathbf{f}}{\partial \mathbf{Y}} \end{bmatrix} \tag{6.3.12}$$

Let $\Delta\mathbf{q}$ denote the Newton differences such that

$$\Delta\mathbf{q} = \mathbf{q}^{k+1} - \mathbf{q}^{k} = \begin{bmatrix} \Delta\dot{\mathbf{X}}_{\mathbf{H}} \\ \Delta\mathbf{Y} \end{bmatrix} \tag{6.3.13}$$

Newton differences can be calculated by solving the linear system

$$\mathrm{J}(\mathbf{q}^{k}) \begin{bmatrix} \Delta\dot{\mathbf{X}}_{\mathbf{H}} \\ \Delta\mathbf{Y} \end{bmatrix} = -\mathbf{f}(\mathbf{q}^{k}) \tag{6.3.14}$$

The improved estimate $\mathbf{q}^{k+1}$ is then obtained from

$$\mathbf{q}^{k+1} = \mathbf{q}^{k} + \begin{bmatrix} \Delta\dot{\mathbf{X}}_{\mathbf{H}} \\ \Delta\mathbf{Y} \end{bmatrix} \tag{6.3.15}$$

This is equivalent to the iteration implied in Equation (6.3.11), but is preferred because we do not need to invert the Jacobian matrix.

6.3.2 Sensitivity Analysis Formulation

Based on the nonlinear analysis formulation described in the previous section, we now derive a formulation for sensitivity analysis based on a direct differentiation approach.

It is assumed that the performance functions of interest in the system are of the form

$$\mathrm{g}_i(\mathbf{P}, \mathbf{X}(\mathbf{d},t), \dot{\mathbf{X}}(\mathbf{d},t), \mathbf{Y}(\mathbf{d},t), \mathbf{d}, t), \quad i = 1 \cdots n_g \tag{6.3.16}$$

where n_g denotes the number of performance functions. For a particular performance function g_i we can then derive the sensitivity vector as given by,

$$(\mathrm{g}_i)_{\mathbf{d}} = (\mathrm{g}_i)_{\mathbf{d}|\exp} + (\mathrm{g}_i)_{\dot{\mathbf{X}}} \dot{\mathbf{X}}_{\mathbf{d}} + (\mathrm{g}_i)_{\mathbf{X}} \mathbf{X}_{\mathbf{d}} + (\mathrm{g}_i)_{\mathbf{Y}} \mathbf{Y}_{\mathbf{d}} \tag{6.3.17}$$

where a vector subscript denotes partial differentiation with respect to the subscript and $(\mathbf{g_i})\mathbf{d}|\mathbf{exp}$ represents the explicit derivative of $\mathbf{g_i}$ with respect to $\mathbf{d}$. The derivatives of $\mathbf{g_i}$ in Equation (6.3.17) are directly obtained by differentiation in the MIXEDMODELS symbolic engine. However, the state sensitivities $\dot{\mathbf{X}}_{\mathbf{d}}$, $\mathbf{X_d}$ and $\mathbf{Y_d}$ need to be calculated numerically. These can be obtained by differentiating the system of governing equations with respect to the design vector. Based on the partition of the $\mathbf{X}$ vector given by Equation (6.3.2), we can partition the vector $\mathbf{X_d}$ in a similar way

$$\mathbf{X}_{\mathbf{d}} = \begin{bmatrix} \mathbf{X}_{\mathbf{H}_{\mathbf{d}}} \\ \mathbf{X}_{\mathbf{L}_{\mathbf{d}}} \end{bmatrix} \tag{6.3.18}$$

Then the state sensitivities $\mathbf{X}_{\mathbf{L_d}}$ can be easily calculated from the direct assignment equations given by

$$\dot{\mathbf{X}}_{\mathbf{L_d}} = \mathbf{X}'_{\mathbf{d}} \tag{6.3.19}$$

where, $\mathbf{X'_d}$ is a subvector of $\mathbf{X_d}$.

Let us define

$$\mathbf{q_d} = \begin{bmatrix} \dot{\mathbf{X}}_{\mathbf{H_d}} \\ \mathbf{Y_d} \end{bmatrix}_{(N+M)\times(N_d)} \tag{6.3.20}$$

where, N_d denotes the number of design variables. Now we need to calculate $\mathbf{q_d}$. By differentiating Equation (6.3.4) with respect to the design variable vector $\mathbf{d}$ we get

$$\mathbf{f_d} = \mathbf{f}_{\dot{\mathbf{X}}_\mathbf{H}} \dot{\mathbf{X}}_{\mathbf{H_d}} + \mathbf{f_X X_d} + \mathbf{f_Y Y_d} + \mathbf{f_{d|exp}} = 0 \tag{6.3.21}$$

$$\therefore \quad \mathbf{f}_{\dot{\mathbf{X}}_\mathbf{H}} \dot{\mathbf{X}}_{\mathbf{H_d}} + \mathbf{f_Y Y_d} = -\mathbf{f_{d|exp}} - \mathbf{f_X X_d} \tag{6.3.22}$$

$$\therefore \quad \begin{bmatrix} \mathbf{f}_{\dot{\mathbf{X}}_\mathbf{H}} & \mathbf{f_Y} \end{bmatrix} \begin{bmatrix} \dot{\mathbf{X}}_{\mathbf{H_d}} \\ \mathbf{Y_d} \end{bmatrix} = -\mathbf{f_{d|exp}} - \mathbf{f_X X_d} \tag{6.3.23}$$

Equations (6.3.23) and (6.3.19) together represent a system of DAEs that can be solved numerically to obtain the state sensitivities in a manner analogous to the solution of the system governing DAEs of Equation (6.3.9). We further note that unlike Equation (6.3.9), which is nonlinear in the derivatives of the differential variables as well as the algebraic variables, Equations (6.3.19) and (6.3.23) are linear in the corresponding sensitivities. Thus, they can be solved directly without iteration. Further, we note that

$$\begin{bmatrix} \mathbf{f}_{\dot{\mathbf{X}}_\mathbf{H}} & \mathbf{f_Y} \end{bmatrix} = \begin{bmatrix} \dfrac{\partial \mathbf{f}}{\partial \dot{\mathbf{X}}_\mathbf{H}} & \dfrac{\partial \mathbf{f}}{\partial \mathbf{Y}} \end{bmatrix} = \mathrm{J} \tag{6.3.24}$$

which is the same as the Jacobian matrix in Equation (6.3.12). Thus, the coefficient matrix for Equation (6.3.23) is already available, which makes the solution of this equation very convenient.

The system of DAEs of Equations (6.3.19) and (6.3.23) are solved numerically concurrently with the DAEs of Equation (6.3.9). The ODEs in these equations can be solved by any suitable ODE solver. The dependent variables seen by the ODE solver are now the system differential variables $\mathbf{X}$ as well as their sensitivities $\mathbf{X_d}$. When the

ODE solver calls for derivative evaluation at a particular time t with the current estimate for this set of dependent variables (i.e., $\mathbf{X}$ and $\mathbf{X_d}$), we do the following:

- First, we perform the Newton-Raphson iteration of Equations (6.3.14) and (6.3.15) to calculate the derivatives of the higher order differential state variables $\dot{\mathbf{X}}_\mathbf{H}$, and the algebraic state variables $\mathbf{Y}$.
- Next, we use the direct assignment equations in Equation (6.3.9) to set the values of the lower order differential state variables, $\dot{\mathbf{X}}_\mathbf{L}$.
- Once we have the values of all the state variables, we can solve Equation (6.3.23) to obtain the derivatives of the sensitivities of the higher order differential variables and the sensitivities of the algebraic state variables $\mathbf{Y_d}$.
- Finally, the derivatives of the sensitivities of the lower order differential state variables are set through direct assignment from Equation (6.3.19).

6.3.3 Calculating Initial Conditions

In order to start the integration from the given initial time, initial conditions must be provided on all the differential variables that we wish to solve for. Specifically, initial conditions must be given not only on $\mathbf{X}$, but also on $\mathbf{X_d}$. In specifying initial conditions, care must be taken to ensure that the specified initial conditions are physically realizable and consistent with all system constraints, such as loop closure conditions on a mechanical system. To ensure consistency of initial conditions, we assume that the user provides a set of consistency equations that must be satisfied by the initial conditions. These equations are assumed to be of the form

$$\mathbf{h}(\mathbf{P}, \mathbf{X}(\mathbf{d}, t), \mathbf{d}, t)\,|_{t=0} = 0 \tag{6.3.25}$$

where, $\mathbf{h}$ is a vector of nonlinear algebraic equations of dimension (N+L) that can be solved by Newton-Raphson iteration to obtain a consistent initial conditions vector $\mathbf{X}$. Recall that N is the number of higher-order differential variables and L is the number of lower-order differential variables. The initial condition vector $\mathbf{X_d}$ for sensitivities can then be obtained by solving the resulting Equation (6.3.26) as a linear system of algebraic equations.

$$\mathbf{h}_{\mathbf{X}}.\mathbf{X}_{\mathbf{d}} = -\mathbf{h}_{\mathbf{d}|\exp} \tag{6.3.26}$$

Since this is a system of linear equations, the user does not even have to provide initial guesses on the $\mathbf{X_d}$ – the correct initial conditions are calculated directly from Equation (6.3.26).

6.4 Sensitivity Based Modeling

Metamodeling has been used successfully to expedite design and optimization in several areas, particularly in the area of structural design and optimization. A popular technique that is used in the structural domain is response surface methodology, wherein a response surface is fitted to known points in design space, and this fitted surface is used to predict performance at other points in design space. If sensitivity information is not used, the system response must be determined at a large number of points in order to obtain a reasonably accurate response surface, which could involve a significant computational effort. The situation is more complex for mechatronic systems because they generally exhibit transient behavior. In this case, if we wish to use classical response surface methodology, we need to fit a response surface at every time point of interest, since the system response for the same design is different at different instants in time. While this is not impossible, it is a much more demanding task that fitting a response surface to a static or steady-state system.

Accordingly, we attempt here to develop a metamodeling approach that takes into account some typical characteristics of mechatronic design. First, we note that the number of design variables in a mechatronic system is usually not very large as compared to a large-scale structural problem. Secondly, the range of the design variables is usually known, and often is not very large. In cases where the design variables have a large range, we can subdivide the range into subranges of reasonable size, with an accompanying increase in the number of points in design space at which we need to perform exact analysis. The third factor that we can take advantage of is that the availability of design sensitivity information enables us to make predictions over a wider range than just using system response information. In

particular, if we know the system response **z** at a given design **d**, then the response at a perturbed design (**d**+Δ**d**) is given quite simply by

$$\mathbf{z}(\mathbf{d}+\Delta\mathbf{d}) \approx \mathbf{z}(\mathbf{d}) + \mathbf{z}_{\mathbf{d}}\Delta\mathbf{d} \qquad 6.4.1)$$

where $\mathbf{z}_{\mathbf{d}}$ is the sensitivity matrix of **z** with respect to **d**, which can be obtained using the technique described in Section 2.

Accordingly, we evaluate two very quick and efficient methods for metamodeling of mechatronic systems:

Method 1: Using a selected number of reference designs, obtain a set of predicted responses at the new design point using Equation (6.4.1). By plotting these responses, we can usually select one that will serve as our model approximation. Excessive discrepancies between the predictions is an indication that additional reference designs are needed.

Method 2: This is similar to Method 1, except that we obtain a single predicted response by forming a weighted sum from the individual predictions. In the example presented below, the prediction based on a particular reference point is given a weight that is inversely proportional to the distance of the reference point from the new design point being considered. The weights are normalized to add up to unity.

Computationally, both these methods are extremely lean and simple to implement, which are the desired characteristics of a metamodel. Further, Method 1 in particular is capable of giving an indication of the quality of the metamodel, which is very important to the designer.

As an example we consider a slider-crank mechanism with AC-Rectified DC power supply (Figure 6.4.1).

The system has 17 components, viz., one DC Motor, three Rigid Bodies (Crank, Connecting Rod, Slider), two Revolute Joints, one Bipolar Junction Diode, one Capacitor, one Sinusoidal Voltage Source, three Electric Nodes, one Analog Ground Component, one Slider Constraint, one Fixed Axis Body Constraint, and two Mechanical Connections (torque and acceleration connection between the motor and crank).

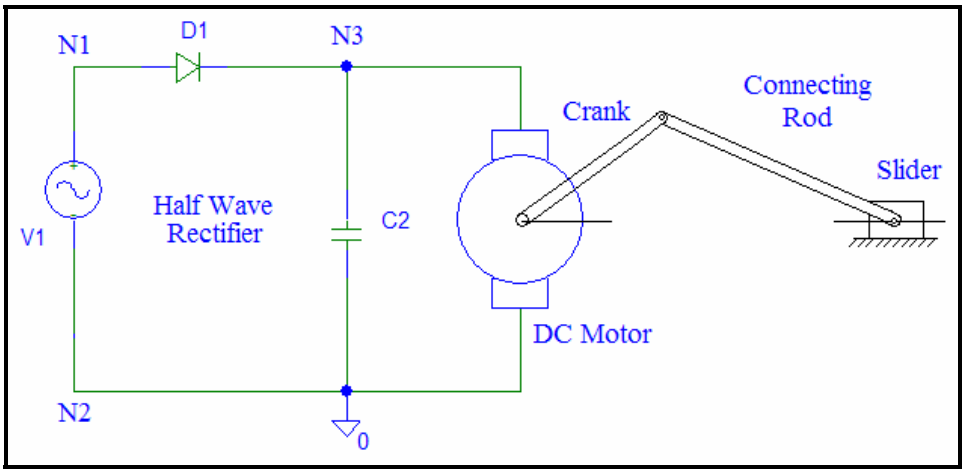


Figure 6.4.1: Slider Crank Mechanism with Half-Wave Rectifier

The entire system is at rest with a crank angle of 45 deg. when the AC power is turned on; thus, all currents and voltages in the system are zero at the initial time.

6.4.1 System Specifications

6.4.1.1 DC Motor

J_m = Motor Inertia = 0.0044 oz-in-s^2

R_a = Motor Armature Resistance = 2.4 Ω

L_a = Motor Winding Inductance = 0.0048 mH

K_b = Back EMF Constant = 0.0401 v/rad/s

K_T = Motor Constant = 5.6 oz-in/amp

B_v = Viscous Friction = 0.0137 oz-in/rad/s

6.4.1.2 Crank

L_1 = length of link 1 = 1.0 in

m_1 = mass of link 1 = 0.00136 oz-s^2/in

J_1 = inertia of link 1 = 0.00414 oz-in-s^2

6.4.1.3 Connecting Rod

L_2 = length of link 2 = 3.0 in

m_2 = mass of link 2 = 0.0041 oz-s^2/in

J_2 = inertia of link 2 = 0.0373 oz-in-s^2

6.4.1.4 Slider

m_s = slider mass = design variable d_1

J_s = slider inertia = 1.0 oz-in-s^2

6.4.1.5 Bipolar Junction Diode (PSpice D1N4148)

η = emission coefficient = 1.836

V_t = thermal voltage = 0.026 V

I_s = saturation current = 2.682 nA

B_R = reverse breakdown voltage = 100 V

d = grading coefficient = 0.3333

τ_t = minority carrier lifetime = 11.54 ns

C_{j0} = zero bias depletion capacitance = 4 pF

ϕ_0 = built in potential = 0.5 V

R_b = junction ohmic resistance = 0.5664 Ω

6.4.2 Design Variables

We chose two design variables for the system, viz., the slider mass and the source voltage, i.e.,

d = [slider mass, source voltage] = [d_1 d_2]

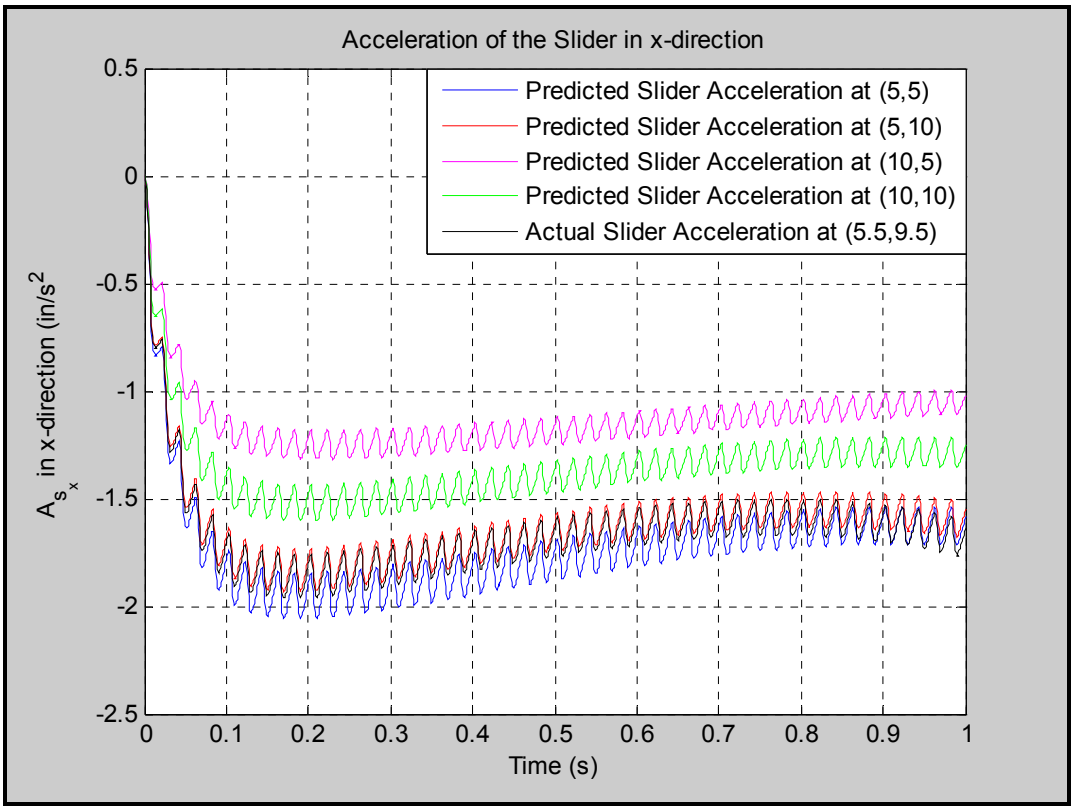
We set up metamodels of this system using both methods described earlier in this section. Four reference designs that were used to set up the metamodel, viz., [5 5], [5 10], [10 5] and [10 10]. Based on the system response and sensitivity information computed at these designs, we attempted to predict the system response at four new designs, given by [5.5 9.5], [8 8], [7.5 7.5] [6 8.5]. The slider acceleration was chosen as the response variable of interest. The predicted response and actual

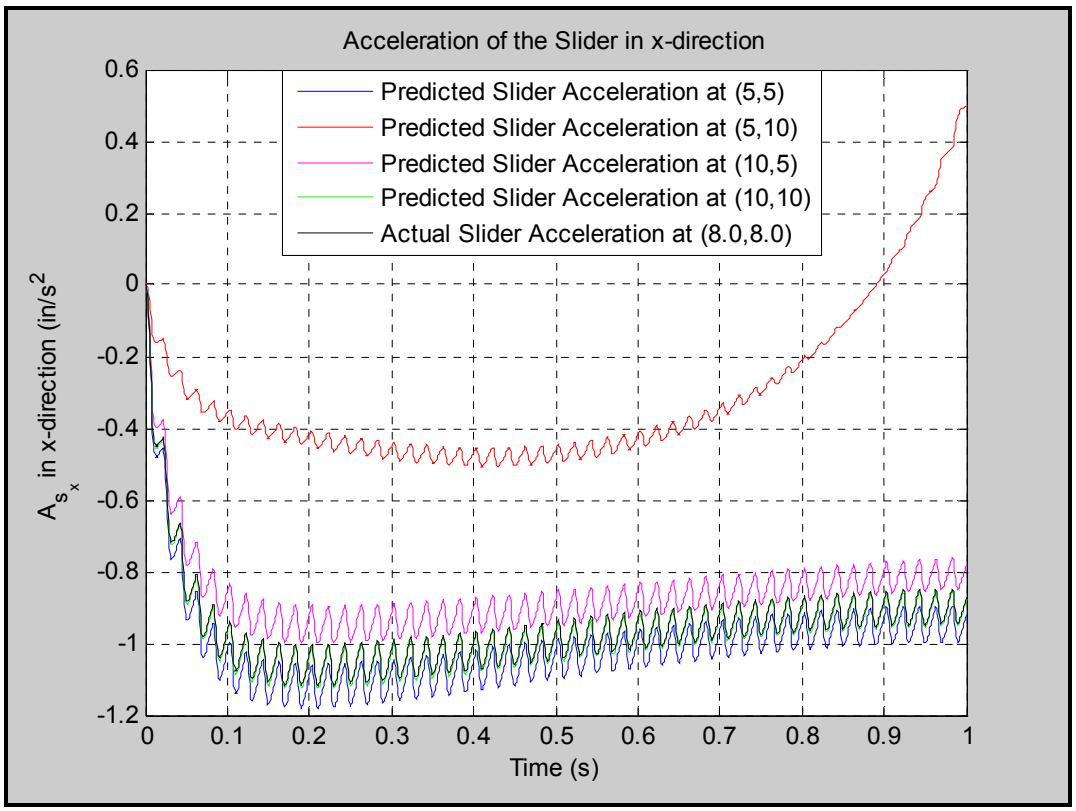
response obtained by full reanalysis at these designs are summarized in the Figures (6.4.2) and (6.4.3).

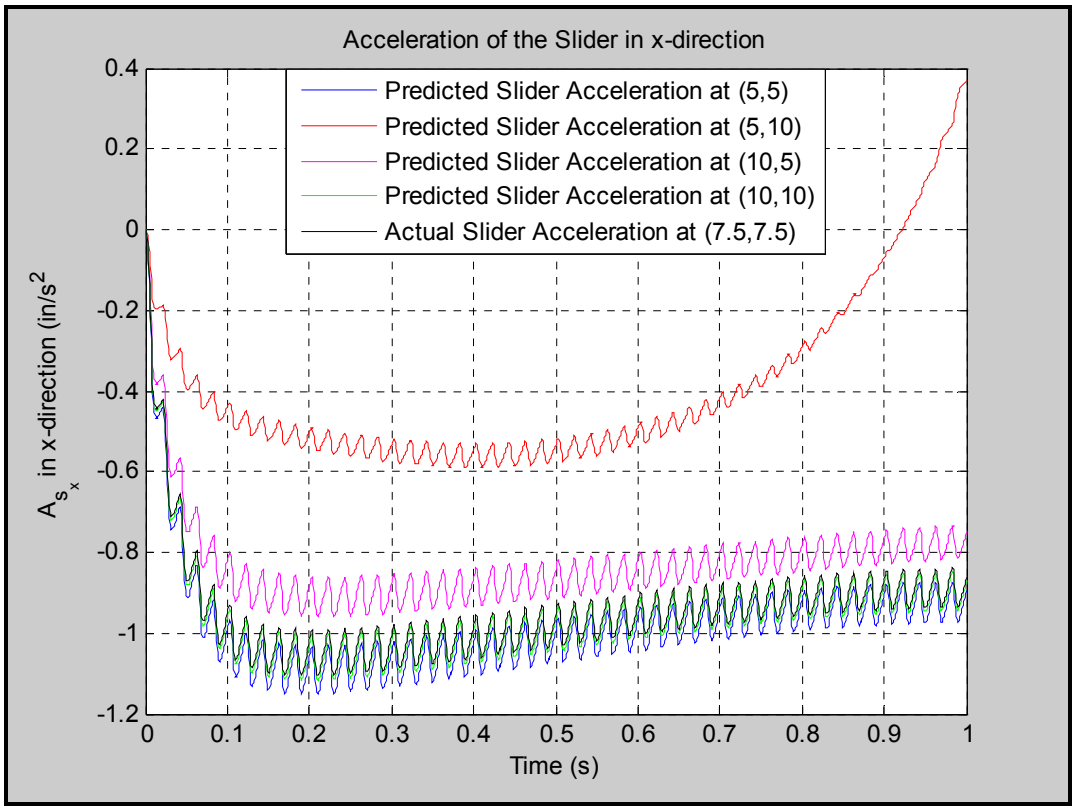
6.5 Discussion of Results

The results indicate that the proposed metamodeling approaches are capable of producing fairly good approximations of the system response at the new designs, but not all reference points produce good predictions. The advantage of Method 1 is that it is easy for the user to assess the quality of the predictions and choose a good one, while discarding outliers. On the other hand, the weighted sum approach generally produces better approximations, but cannot give an indication of whether there is a problem with the prediction or not (in practice, the actual solution will not be available). It appears that a combination of the two methods may be a good option in practice. We must do all the computations of Method 1 in order to use Method 2, and the extra calculations for Method 2 are minimal, so there is little computational overhead. Method 1 can then give an indication of the reliability of the prediction, and the Method 2 prediction can be used as the approximate system response. If the predictions are not accurate enough, a finer grid of reference designs may have to be used. The weighting scheme can also be improved. Interestingly, the nearest reference point to a new design is not necessarily the best one to use to make the prediction.

At this point, it does appear that sensitivity-based metamodeling for mechatronic systems is feasible, but clearly several improvements are needed. Adaptive selection of reference designs, improved weighting, and higher-order sensitivity analysis are possible improvements. Improved surface fitting techniques may also prove worthwhile, although this will increase the amount of computation required.


Acceleration of the Slider in x-direction
Predicted Slider Acceleration at (5,5)
Predicted Slider Acceleration at (5,10)
Predicted Slider Acceleration at (10,5)
Predicted Slider Acceleration at (10,10)
Actual Slider Acceleration at (5.5,9.5)
A_{s_x} in x-direction (in/s^2)
Time (s)
0.5
0
-0.5
-1
-1.5
-2
-2.5
0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0.9
1


Acceleration of the Slider in x-direction
Predicted Slider Acceleration at (5,5)
Predicted Slider Acceleration at (5,10)
Predicted Slider Acceleration at (10,5)
Predicted Slider Acceleration at (10,10)
Actual Slider Acceleration at (8.0,8.0)
A_{s_x} in x-direction (in/s^2)
Time (s)
0.6
0.4
0.2
0
-0.2
-0.4
-0.6
-0.8
-1
-1.2
0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0.9
1



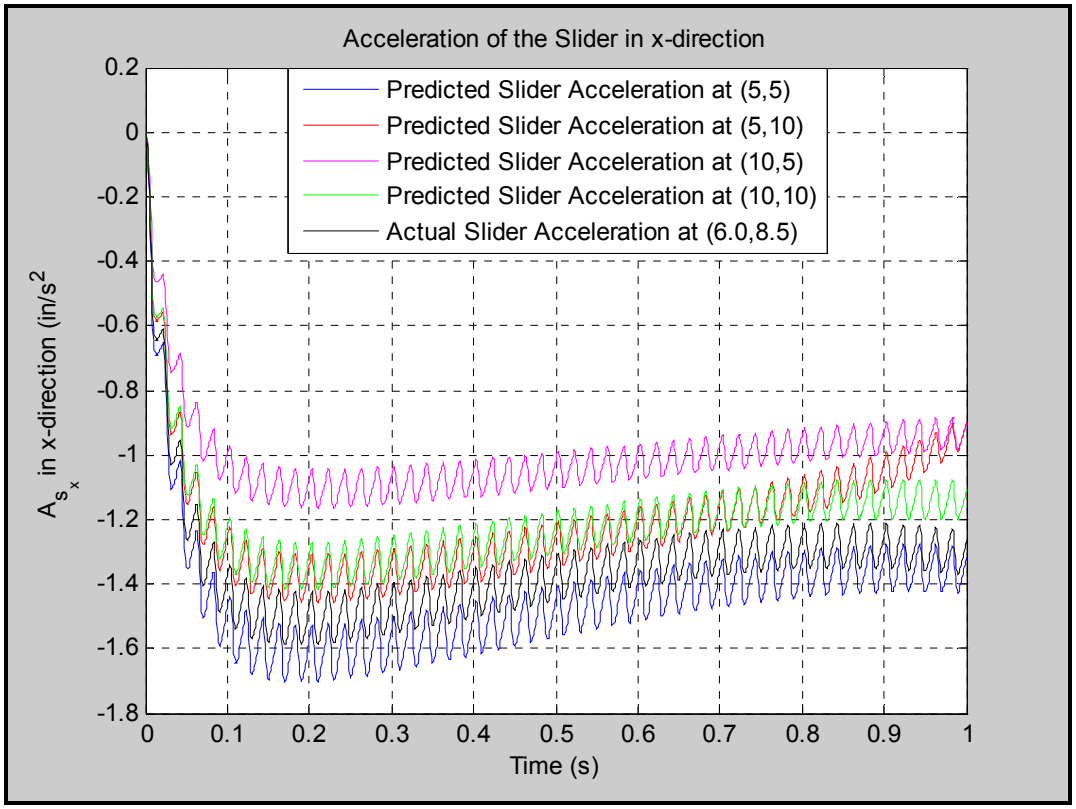
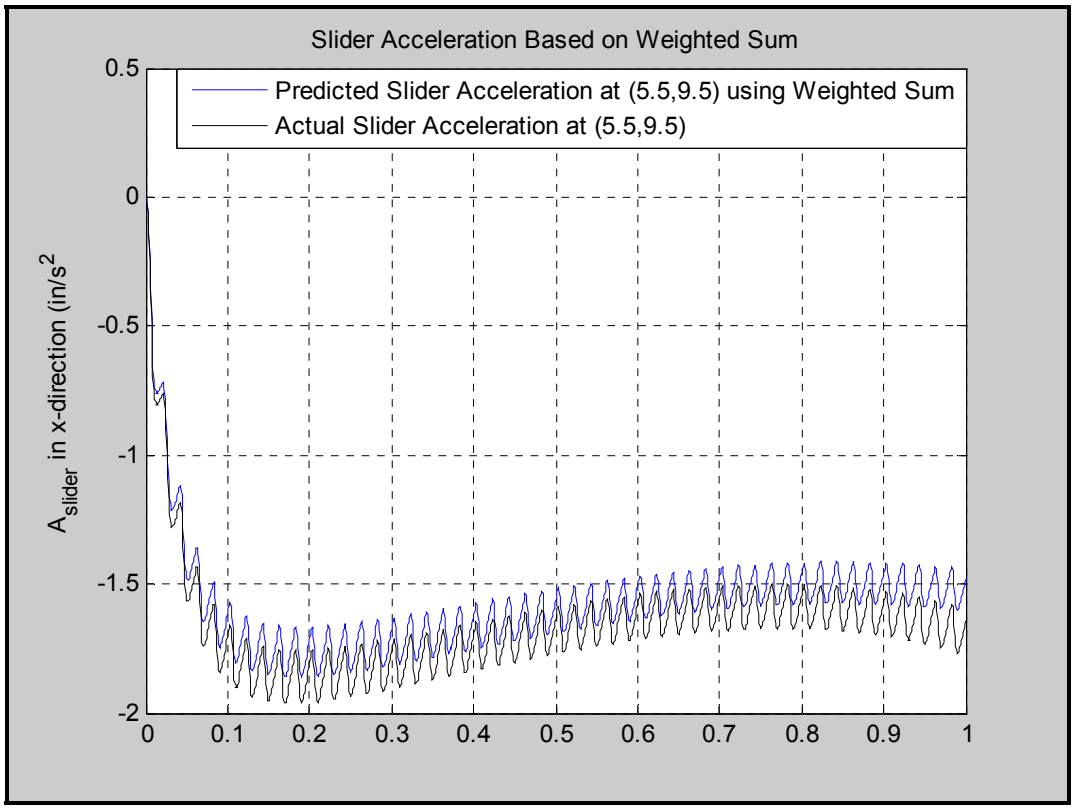
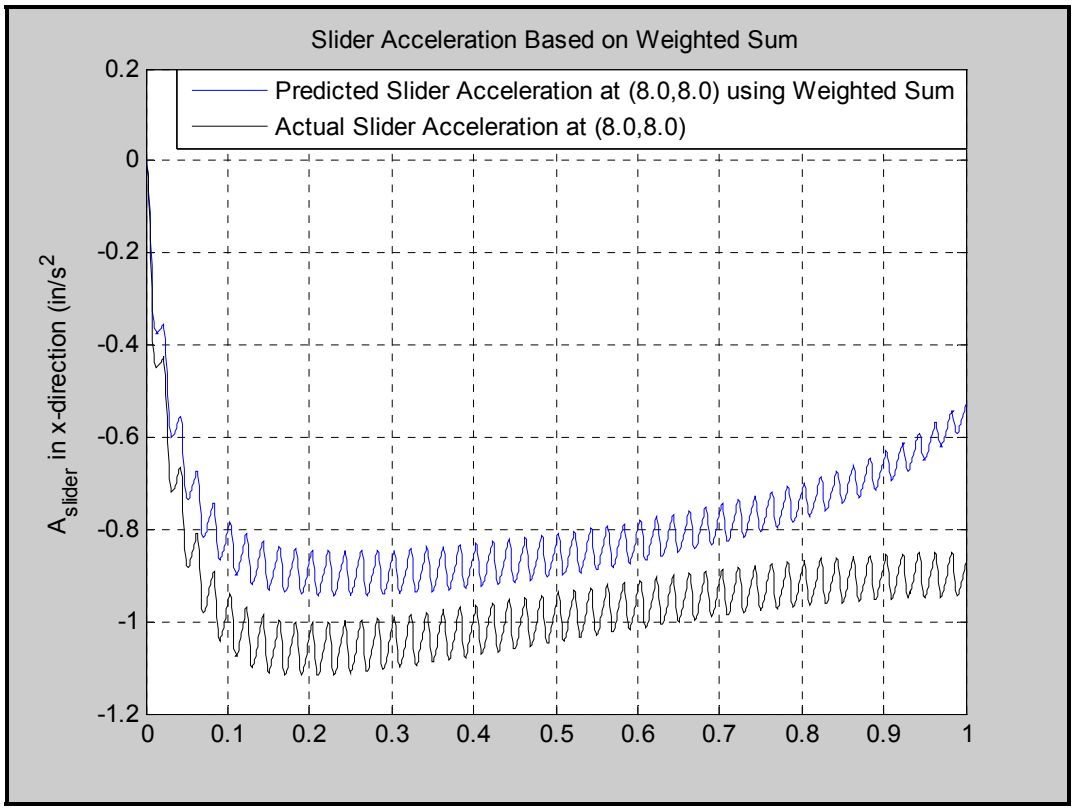


Figure 6.4.2: Predictions Using Method 1


Slider Acceleration Based on Weighted Sum
Predicted Slider Acceleration at (5.5,9.5) using Weighted Sum
Actual Slider Acceleration at (5.5,9.5)
A_slider in x-direction (in/s^2
0.5
0
-0.5
-1
-1.5
-2
0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0.9
1


Slider Acceleration Based on Weighted Sum
Predicted Slider Acceleration at (8.0,8.0) using Weighted Sum
Actual Slider Acceleration at (8.0,8.0)
A_slider in x-direction (in/s^2
0.2
0
-0.2
-0.4
-0.6
-0.8
-1
-1.2
0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0.9
1

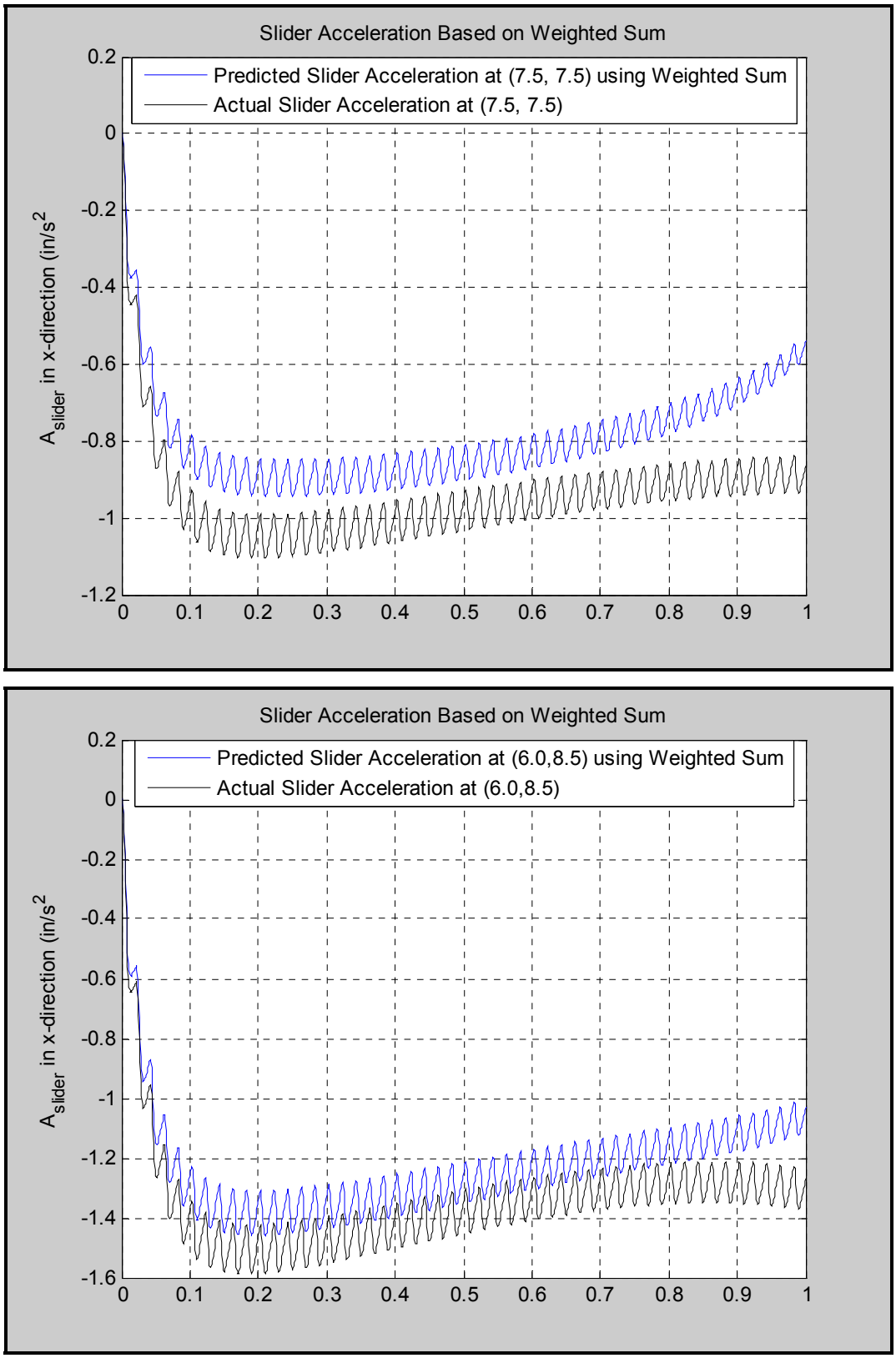


Figure 6.4.3: Predictions Using Method 2

6.6 REFERENCES

6.6.1 Carrigan, J., Kelkar, A. and Krishnaswami, P., '*Minimum Sensitivity Design of Controlled Multibody Systems*', ASME Multibody Systems, Nonlinear Dynamics, and Control Conference, 2005.

6.6.2 Kelkar, A., Krishnaswami, P., Carrigan, J., '*Multidisciplinary Optimization of Multibody Systems*', Proceedings of the ECCOMAS Conference on Multibody Systems, June 2005.

6.6.3 Carrigan, J., 'Integrated Design and Sensitivity Based Design and Optimization of Nonlinear Controlled Multibody Mechanisms', Thesis, M.S. Iowa State Univ., 2003.

6.6.4 Duysix, P., Brüls, O., Collard, J-F., Fisette, P., Lauwerys, C., Swevers, J., '*Optimization of Mechatronic Systems: Application to a Modern Car Equipped with a Semi-active Suspension*', 6th World Congress of Structural and Multidisciplinary Optimization, Brazil, 2005.

6.6.5 Schönning, A., Nayfeh, J., Zarda, R, '*An Integrated Optimization Environment for Industrial Large Scaled Systems*', Research in Engineering Design, pp. 86-95, 2005.

6.6.6 Dignath, F., Breuninger, C., Eberhard, P., Kübler, L., '*Optimization of Mechatronic Systems Using the Software Package NEWOPT/AIMS*', Multibody System Dynamics, pp. 85-100, 2005.

6.6.7 Barton, P.I., Lee, C.K., '*Modeling*, *Simulation*, *Sensitivity Analysis*, *and Optimization of Hybrid Systems*', ACM Transactions on Modeling and Computer Simulation, pp. 256-289, s2002.

6.6.8 Groothuis, M.A., Broenink, J.F., '*Multi-view Methodology for the Design of Embedded Mechatronic Control Systems*', Proceedings of the 2006 IEEE Conference on Computer Aided Control Systems Design, pp. 416-421, 2006.

6.6.9 Thramboulidis, K., '*Model-Integrated Mechatronics-Toward a New Paradigm in the Development of Manufacturing Systems*', IEEE Transactions on Industrial Informatics, Vol. 1, pp. 54-61, 2005.

6.6.10 Martin, J.D., Simpson, T.W., '*Use of Adaptive Metamodeling for Design Optimization*', AIAA/ISSMO Symposium on Multidisciplinary Analysis and Optimization, 2002.

6.6.11 Jin, R., Chen, W., Simpson, T.W., '*Comparative Studies of Metamodeling Techniques under Multiple Modeling Criteria*', Journal of American Institute of Aeronautics and Astronautics, 2000.

6.6.12 Cappelleri, D.J., Frecker, M. I., Simpson, T.W., Snyder, A., '*A Metamodel-Based Approach for Optimal Design of a PZT Biomorph Actuator for Minimally Invasive Surgery*', ASME Journal of Mechanical Design, pp. 354-357, 2002.

6.6.13 Lee, H.J., Crary, S.B., Affour, B., Bernstein, D., Gianchandani, Y.B., Woodcock, D.M., Maher, M.A., '*Generation of a Metamodel for a Micromachined Accelerometer using T-SpiceTM and the Iz-Optimality Option of I-OPTTM*', Technical Proceeding of the 2000 International Conference on Modeling and Simulation of Microsystems, 2000.

6.6.14 Wu, B., Prucka, R.G., Filipi, Z.S., Kramer, D.M., Ohl, G.L., '*Cam-phasing Optimization Using Artificial Neural Networks as Surrogate Models-Fuel Consumption and NOx Emissions*', SAE World Congress, 2006.

6.6.15 Karakasis, M.K., Koubogiannis, D.G., Giannakoglou, K.C., '*Hierarchical Distributed Metamodel-Assisted Evolutionary Algorithms in Shape Optimization*', International Journal for Numerical Methods in Fluids, pp. 455-469, 2007.

6.6.16 Seller, R.S., Batill, S.M., Renaud, J.E., '*Response Surface Based Concurrent Subspace Optimization for Multidisciplinary System Design*', Journal of American Institute of Aeronautics and Astronautics, 1996.

6.6.17 Wang, L., Lowther, D.A., '*Selection of Approximation Models for Electromagnetic Device Optimization*', IEEE Transactions on Magnetics, Vol. 42, 2006, pp. 1227-1230.

6.6.18 Sinha, R., Paredis, C.J.J., Liang, V-C., Khosla, P.K., '*Modeling and Simulation Methods for Design of Engineering Systems*', Journal of Computing and Information Science in Engineering, Volume 1, Issue 1, pp. 84-91, 2001.

6.6.19 Sass, L., McPhee, J., Schmitke, C., Fisette, P. and Grenier, D., '*A Comparison of Different Methods for Modelling Electromechanical Multibody Systems*', Multibody System Dynamics, 12: 209-250, 2004.

6.6.20 Ram, O., Szymkat, M., Uhl, T., Betemps, M., Pjetursson, A. and Rod, J., '*Mechatronic Blockset for Simulink – Concept and Implementation*', Proceedings of the 1996 IEEE Intl. Symposium on Computer-Aided Control System Design, pp. 530-535, 1996.

6.6.21 Knorr, U.I., '*Electromechanical System Design*', Ansoft Corporation, 2002.

6.6.22 http://www.amesim.com/, IMAGINE Software Inc., acquired on Jan 29, 2005

6.6.23 Fritzson, P., Aronsson, P., Pop, A., Lundvall, H., Nystrom, K., Saldamli, L., Broman D., Sandholm, A. '*OpenModelica – A Free Open-Source Environment for System Modeling, Simulation, and Teaching*', Proceedings of IEEE Conference on Computer Aided Control Systems Design, pp. 1588-1595, 2006.

6.6.24 Claeys, F.H.A., Fritzson, P., Vanrolleghem, P.A., '*Using Modelica Models for Complex Virtual Experimentation with the Tornado Kernel*', The Modelica Association, pp. 193-202, 2006.

6.6.25 Vaze, S., Krishnaswami, P., DeVault, J., '*Modeling of Hybrid Electromechanical Systems Using a Component-Based Approach*', Proceedings of the IEEE International Conference on Mechatronics & Automation, pp. 204-209, July 2005.

6.6.26 Krishnaswami, P., Vaze, S., DeVault, J., '*Design Sensitivity Analysis of Multidisciplinary Multibody Systems*', Proceedings of Multibody Dynamics, An Eccomas Thematic Conference, June 2007.

6.6.27 Vaze, S., Krishnaswami, P., DeVault, J., '*Symbolic-Numeric Computing in Software Development for Modeling and Simulation of Multidisciplinary Multibody Systems*', Proceedings of Multibody Dynamics, An Eccomas Thematic Conference, June 2007.

CHAPTER 7 - DISCUSSION OF NUMERICAL METHODS

Mathematical modeling and simulation have become significant elements of research and design in academia and industry. With significant advancement in computational speed, accuracy and availability, modeling and simulation capabilities continue to improve rapidly. Multidisciplinary systems, as stated before, can be described by sets of differential and algebraic equations. Not many DAEs have analytic solutions, and even if they exists, usually they are it is difficult to obtain and sometimes impractical [7.5.1]. Moreover, due to the nature of multidisciplinary systems, simulation techniques for these methods need to cater to specific numerical issues. Numerical simulation is thus a significant ingredient of the MIXEDMODELS recipe and is an open-ended section that needs further research to improve the capabilities of the numeric engine to make the algorithm numerically more robust.

To place the following discussion in context, recall that the symbolic engine of MIXEDMODELS uses a component-based approach for modeling of MDSs. In this approach, a system is viewed as a collection of interconnected components from different domains. Each type of component has its own set of governing equations, which can be derived by applying the mathematical and/or physical principles that are best suited to the domain. These component equations are combined to form a system of DAEs. The algebraic equations can either be linear or nonlinear. Based on these DAEs, a set of equations in the state design sensitivity coefficients is analytically derived using direct differentiation. These equations are a set of DAEs which can be solved simultaneously with the system governing equations to obtain the solution for the state variables and state sensitivity coefficients of the system. Finally, knowing the system performance functions, we can calculate the design sensitivity coefficients of these performance functions by using the values of the state variables and state sensitivity coefficients obtained from the DAEs.

DAEs are thus important and widely used techniques in mathematical modeling, and the simulation techniques to solve these DAEs should be able to handle the numerical challenges posed by the nature of multidisciplinary systems.

7.1 Numerical Challenges in the Simulation of Multidisciplinary Systems

- Multidisciplinary systems consist of components from different domains and different subsystems such as electronic, pneumatic, etc. which may have solutions with greatly varying time scales. For example, electronic systems are much faster than mechanical, pneumatic or hydraulic systems. This also makes the system of equations stiff and therefore harder to solve.
- Differential equations may add stiffness to the system and require special numerical methods.
- Nonlinear systems may start off non-stiff and become stiff, or vice versa, which means that they may have different stiff and non-stiff intervals. Such systems become even more complex to solve, and they need tighter error and step size control.
- In a component-based approach it is generally true that the connection components add more equations and redundant variables to the system, which increases the system size.
- The component-based architecture generally leads to a system in which the system matrix has a sparse structure.
- The numerical methods should be capable of solving DAEs, ODEs and pure AEs. Furthermore, systems of algebraic equations can be linear as well as nonlinear.
- A combination of different disciplines may lead to a large numerical range of component parameters, requiring scaling of the system matrices.

7.2 Numerical Capabilities of MIXEDMODELS

The numeric engine of MIXEDMODELS provides a small collection of numerical integrators and linear and nonlinear solvers to evaluate the system response. All of these solvers and integrators are written in FORTRAN and are provided by NETLIB repository. To improve performance based on speed and accuracy MIXEDMODELS also implements a symbolic reduction routine and a scaling routine. The following paragraphs discuss the capabilities of the numeric engine of MIXEDMODELS.

7.2.1 Numerical Integrators

The numeric engine of MIXEDMODELS supports four canned numerical integrators including DVERK, RKF45, LSODE and DLSODES, all taken from NETLIB numerical solver open-source repository [7.5.2]. All theses solvers are written in FORTRAN.

7.2.1.1 DVERK

"DVERK", as stated in its documentation in [7.5.2], "is a single-step Runge-Kutta subroutine based on Verner's fifth and sixth-order pair of formulas for finding approximations to the solution of a system of first- order ordinary differential equations with initial conditions. It attempts to keep the global error proportional to a tolerance specified by the user". This subroutine is efficient for solving non-stiff systems. However, if the function evaluations are costly, this method is not efficient [7.5.2].

7.2.1.2 RKF45

RKF45 is a Felberg fourth-fifth-order single-step Runge-Kutta method which was developed by Watts and Shampine at Sandia National Laboratories. RKF45 is primarily designed to solve non-stiff and mildly stiff differential equations when the derivative calculations are inexpensive. It is not very useful when the user is demanding high accuracy.

7.2.1.3 LSODE

LSODE is a double-precision version of an ordinary differential equation solver developed at Livermore National Labs. LSODE solves the initial value problem for

stiff or non-stiff systems of first order ODEs. It is a multi-step method that is based on the Gear and Gearb packages. The method varies the step size and the order of the methods to efficiently meet the error tolerance.

7.2.1.4 DLSODES

DLSODES is a double-precision version which is an extension of LSODE package, created for solving ordinary differential equations with general sparse Jacobian matrix. It is also a multi-step method with variable step size and order and solves the initial value problem for stiff or non-stiff systems of first order ODEs.

7.2.2 Linear and Nonlinear Solvers

Chapter 2 presented a mathematical formulation that can be applied in a general way to include disciplines such as mechanical, electrical/electronic, hydraulic, pneumatic, control systems etc. Chapter 3 presented a design sensitivity scheme based on this formulation. This formulation works well when the system is linear. However, it shares a drawback with linear state-space methods when it comes to dealing with a general multidisciplinary system whose component-governing equations may include a large number of nonlinear algebraic or differential equations. This is often the case when a wide range of component types are involved. The linear state space formulation and the formulation presented in chapters 2 and 3 require that these equations be converted to a pre-specified form by a process of differentiation. This has the undesirable effect of artificially increasing the number of differential equations in the system, and generally making the numerical solution more difficult. We occasionally have found that a non-stiff system may appear numerically stiff in the differentiated form. This formulation may also suffer from joint drift if applied to large systems over long time periods. The most popular method to account for this drift is Baumgarte's stabilization technique, which can be incorporated into this formulation. For example, this technique has been used to develop a mathematical model for a "stabilized revolute joint" to avoid constraint drift.

To avoid these numerical difficulties caused by the differentiation, a general nonlinear formulation for analysis and analytical design sensitivity analysis of multidisciplinary systems was developed and was successfully implemented in the

MIXEDMODELS platform. This form does not require any additional differentiation of the equations and therefore does not require the equations to be generated in a predefined form.

The symbolic-numeric architecture of MIXEDMODELS supports both the linear as well as nonlinear formulation to describe a multidisciplinary system. The numeric engine of MIXEDMODELS supports a sparse linear solver Y12MAF [7.5.2] to solve a linear system. It provides a Newton-Raphson nonlinear iterative solver to solve the nonlinear system of equations. The architecture also provides the user flexibility to choose between linear and nonlinear iterative solvers. The symbolic engine internally checks if the system of equations is linear or nonlinear. If the user chooses to use a linear solver when the system turns out to be nonlinear, the numeric engine writes out a message to the screen displaying that "the system is nonlinear and would be solved using Newton-Raphson nonlinear iterative solver". The algorithm for nonlinear Newton-Raphson iterative solver is described in detail in chapter 4. However, to maintain continuity in reading it will be briefly discussed in this section.

7.2.2.1 Newton-Raphson Nonlinear Solver

The Newton-Raphson algorithm, sometimes referred to as Newton's algorithm, is one of the very popular methods for solving nonlinear equations, mainly for its rapid convergence. It is an iterative method which solves equations of the form

$$\mathbf{f}(\mathbf{q}) = \mathbf{0} \tag{7.2.1}$$

The method starts with an initial guess and for every iteration the estimate is improved for more accuracy. Newton-Raphson solver is known to give quadratic convergence provided that the initial guess is sufficiently accurate [7.5.3].

Recall that with MIXEDMODELS analysis formulation we get the complete set of system DAEs given by

$$\begin{aligned} \mathbf{f}(\mathbf{P}, \mathbf{X}(\mathbf{d},t), \dot{\mathbf{X}}_{\mathbf{H}}(\mathbf{d},t), \mathbf{Y}(\mathbf{d},t), \mathbf{d}, t) &= \mathbf{0} \\ \dot{\mathbf{X}}_{\mathbf{L}}(\mathbf{d},t) &= \mathbf{X}'(\mathbf{d},t) \end{aligned} \tag{7.2.2}$$

The ODEs in Equation (7.2.2) can be solved to obtain the differential variable vector **X** using any suitable ODE solver. To be able to do so, at any time *t*, given the current estimate of the dependent variable vector **X** in the ODEs, we need a way to calculate the derivatives $\dot{\mathbf{X}} = \begin{bmatrix} \dot{\mathbf{X}}_{\mathbf{H}} & \dot{\mathbf{X}}_{\mathbf{L}} \end{bmatrix}^{\mathrm{T}}$. $\dot{\mathbf{X}}_{\mathbf{L}}$ are calculated by using direct assignments. $\dot{\mathbf{X}}_{\mathbf{H}}$ and **Y** are calculated using Newton-Raphson iteration as summarized below.

Let us define a vector **q** as

$$\mathbf{q} = \begin{bmatrix} \dot{\mathbf{X}}_{\mathbf{H}} \\ \mathbf{Y} \end{bmatrix}_{(N+M)\times 1} \tag{7.2.3}$$

where *N* denotes the number of higher-order differential variables and *M* denotes the number of algebraic variables. Given the initial conditions, **X**, and initial guesses on $\dot{\mathbf{X}}_{\mathbf{H}}$ and **Y**, we can iteratively calculate

$$\mathbf{q}^{k+1} = \mathbf{q}^{k} - \mathrm{J}^{-1}(\mathbf{q}^{k})\mathbf{f}(\mathbf{P}, \mathbf{X}, \mathbf{q}^{k}, t) \tag{7.2.4}$$

where, J is the Jacobian matrix given by

$$\mathrm{J}(\mathbf{P}, \mathbf{X}, \mathbf{d}, t) = \begin{bmatrix} \dfrac{\partial \mathbf{f}}{\partial \dot{\mathbf{X}}_{\mathbf{H}}} & \dfrac{\partial \mathbf{f}}{\partial \mathbf{Y}} \end{bmatrix} \tag{7.2.5}$$

Let $\Delta\mathbf{q}$ denote the Newton differences such that

$$\Delta\mathbf{q} = \mathbf{q}^{k+1} - \mathbf{q}^{k} = \begin{bmatrix} \Delta\dot{\mathbf{X}}_{\mathbf{H}} \\ \Delta\mathbf{Y} \end{bmatrix} \tag{7.2.6}$$

The Newton differences can be calculated by solving the linear system

$$\mathrm{J}(\mathbf{q}^{k}) \begin{bmatrix} \Delta\dot{\mathbf{X}}_{\mathbf{H}} \\ \Delta\mathbf{Y} \end{bmatrix} = -\mathbf{f}(\mathbf{q}^{k}) \tag{7.2.7}$$

The improved estimate $\mathbf{q}^{k+1}$ is then obtained from

$$\mathbf{q}^{k+1} = \mathbf{q}^{k} + \begin{bmatrix} \Delta\dot{\mathbf{X}}_{\mathbf{H}} \\ \Delta\mathbf{Y} \end{bmatrix} \tag{7.2.8}$$

This is equivalent to the iteration implied in Equation (7.2.4) but is preferred because we do not need to invert the Jacobian matrix. The Jacobian matrix is also formed symbolically by the symbolic engine of MIXEDMODELS. Once the Newton-Raphson iteration has converged, we will have not only the derivatives needed by the ODE solver, but also the values of the algebraic system variables, **Y**, since both these are contained in the **q** vector.

7.2.2.1.1 Convergence of Newton-Raphson Algorithm

This algorithm starts with an initial guess and keeps refining the solution for more accuracy on each iteration. This method, however, does not terminate naturally until a convergence criterion is forced upon it. In MIXEDMODELS, to avoid possibility of false convergence of the Newton-Raphson algorithm, the numeric engine uses both the *residue* criterion as well as the *update* criterion.

Residue Criterion

The residue criterion ensures the convergence of function evaluations and checks if the infinity norm of the function is within the specified tolerance ε_f.

$$\left\| \mathbf{f}(\mathbf{q}^{k+1}) \right\|_{\infty} \leq \varepsilon_f \tag{7.2.9}$$

To avoid false convergence it is also necessary that the solution at iteration (k+1) is as close as possible to the solution at iteration k, which is monitored by the *update* criterion.

Update Criterion

Update criterion ensures that there exists an ε_q such that

$$\left\| (\mathbf{q}^{k+1} - \mathbf{q}^{k}) \right\| \leq \varepsilon_q \tag{7.2.10}$$

The algorithm calculates absolute as well as relative error for every iteration.

7.2.2.1.2 Calculating Consistent Initial Conditions

In this approach Newton-Raphson algorithm is not only used to solve for the derivatives and algebraic variables, but also to obtain a consistent set of initial

conditions. In order to start the integration from the given initial time, initial conditions must be provided on all the differential variables that need to be calculated. Specifically, initial conditions must be given not only on **X,** but also on the sensitivities $\mathbf{X_d}$. In specifying initial conditions, care must be taken to ensure that the specified initial conditions are physically realizable and consistent with all system constraints [7.5.4]. While it may be very difficult for the user to manually supply a consistent set of initial conditions on **X** for a large system; it may get further overwhelming if the user is called upon to provide initial conditions on $\mathbf{X_d}$. To ensure consistency of initial conditions, it is assumed that the user provides a set of consistency equations that must be satisfied by the initial conditions. These equations are assumed to be of the form

$$\mathbf{h}(\mathbf{P}, \mathbf{X}(\mathbf{d}, t), \mathbf{d}, t)\,|_{t=0} = \mathbf{0} \tag{7.2.11}$$

where **h** is a vector of nonlinear algebraic equations of dimension (N+L) that can be solved by Newton-Raphson iteration to obtain a consistent initial conditions vector **X**. Here N is the number of higher-order differential variables and L is the number of lower-order differential variables. All that the user needs to provide then is a set of initial guesses for **X**, which is then corrected by the iteration. In order to compute initial conditions on sensitivities, Equation (7.2.11) is differentiated with respect to the design variable vector **d**. The initial condition vector $\mathbf{X_d}$ for sensitivities can then be obtained by solving the resulting Equation (7.2.12) as a linear system of algebraic equations.

$$\mathbf{h_X X_d} = -\mathbf{h_{d|exp}} \tag{7.2.12}$$

Since this is a system of linear equations, the user does not have to provide initial guesses on the $\mathbf{X_d}$ – the correct initial conditions are calculated directly from Equation (7.2.12).

7.2.2.2 Linear Sparse Matrix Solver

Along with the attempt to provide robust numerical solvers, computational effort is also a concern. The component-based formulation of MIXEDMODELS gives rise to a system matrix which is sparse, and hence use of sparse methods will improve

efficiency. Y12MAF, a linear solver provided by NETLIB's repository [7.5.2], has been successfully used for a variety of different applications. It solves sparse systems of linear algebraic equations by Gaussian elimination. The subroutine is designed to efficiently solve problems which contain only one system with a single right hand side. However, it can be easily modified to solve a system with multiple right hand sides. Y12MAF is a very efficient solver, however; it is limited by the number of equations it can solve. For further enhancement of numerical capabilities of MIXEDMODELS, the next step would be to replace Y12MAF with a better sparse matrix solver such as UVSS [7.5.5].

7.2.3 Symbolic Variable Reduction

Complex systems generally consist of a large number of components. Any such component model formulated using MIXEDMODELS is self-contained and is thus described only in terms of its local variables. Each component is connected to the rest of the system using "connections" which are also modeled as separate components. These connection components can be of various types. A simple connection component may be merely establishing equivalence between the variables of the two connecting components, with no additional variables of its own. On the other hand a complex connector such as a revolute joint not only relates the two connecting components in terms of their own variables, but also introduces its own variables. Most of the multidisciplinary systems contain numerous simple connection components which introduce many new equations to the system while keeping the number of variables unchanged. This is one major downside of using the component-based modeling approach, since it leads to very large systems which could be computationally expensive to solve.

To address this issue we use the connection components to eliminate system variables from the coefficient matrix. Using the two-pass procedure, the system coefficient matrix, **A**, and the right hand side vector, **b**, are generated as described in Chapter 5. The coefficient matrix, **A**, is then scanned column wise for each row to find a row with exactly two or one non-zero numeric entries. If such a row is found, then the variable with the lower coefficient between the two (in the case of a row

with two nonzero-entries) is expressed in terms of the variable with a higher coefficient, and the former is eliminated from the matrix. In case of the single nonzero numeric entry, that variable is removed from the matrix. Therefore, during each such elimination, this process generates a mathematical expression relating the eliminated variable to the kept variable and the corresponding right hand side entry. The row and column corresponding to the selected elimination are removed from the matrix, and the entire matrix is readjusted to accommodate these changes in the remaining matrix entries. The process is repeated until no further elimination is possible. The reduced system is then solved for the variables which do not get eliminated, and using these values and the mathematical expressions generated during elimination, the eliminated variables are restored in the reverse order of their elimination.

Symbolic reduction is very effective in reducing the computational burden, and it is observed that with this facility simulation times are reduced almost to 50%. A few simple systems such as a full-wave diode bridge rectifier and a non-inverting op-amp were simulated using symbolic reduction, and the results were validated against P-Spice. The results were in good agreement, and MIXEDMODELS was observed to be much faster than P-Spice.

7.2.4 Matrix Scaling

In many cases the accuracy of a linear system can be improved by rescaling the system or by iteratively improving the initial computed solution [7.5.3, 7.5.6]. Scaling affects the conditioning of the system and the selection of pivots in Gaussian elimination, which in turn affect the accuracy of the solution. Thus row and column scaling of a linear system can potentially improve numerical stability or accuracy.

MIXEDMODELS uses a scaling algorithm, developed by Daniel Ruiz [7.5.7], which equilibrates both rows and column norms in a matrix. It is an iterative procedure which asymptotically scales the infinity norm of both rows and columns to 1. The detailed theory behind this algorithm can be found in [7.5.7].

7.3 Software Used for Validation Purpose

In this work, three popular software packages MATLAB, Simulink and PSpice were used for validating system responses generated by MIXEDMODELS. PSpice was used to test purely electrical circuits, while MATLAB and Simulink were used to test simple systems integrating electromechanical, mechanical and control system disciplines. The results obtained are promising and show that MIXEDMODELS is accurate, computationally viable and applicable to a broad class of multidisciplinary systems. The final set of equations obtained is similar in form and complexity to the corresponding equations that are solved by commercial packages such as Dymola and Adams. These packages have demonstrated that such equations can be solved effectively using numerical solvers.

7.4 Plotting Engine

MATLAB is used as the plotting engine for MIXEDMODELS, simply because of its convenience and ease with which it manipulates and handles large data sets. The numeric engine writes the output to MATLAB-compatible files which are read and processed in MATLAB as required for plotting. The numeric engine makes all of the variables available once it solves the system. To avoid handling large output data sets, the user can choose the variables he/she is interested in plotting. In case of sensitivity analysis, the sensitivities of the response variables to all the design variables are written to the output file. The user has the flexibility to plot responses of the sensitivities of his/her interests.

7.5 REFERENCES

7.5.1 Sekar, S., '*Analysis of Linear and Nonlinear Stiff Problems using the RK-Butcher Algorithm*', Mathematical Problems in Engineering, Volume 2006.

7.5.2 http://www.netlib.org/ acquired in June 2005.

7.5.3 Heath, M. T.., '*Scientific Computing – An Introductory Survey*', Second Edition, 2002.

7.5.4 Barton, P.I., Lee, C.K., '*Modeling*, *Simulation*, *Sensitivity Analysis*, *and Optimization of Hybrid Systems*', ACM Transactions on Modeling and Computer Simulation, pp. 256-289, 2002.

7.5.5 Arnold, S. M., Bednarcyk, B.A., Aboudi, J., '*Thermo-Elastic Analysis of Internally Cooled Structures Using a Higher Order Theory*', NASA, Glenn Research Center Reports, 2001.

7.5.6 Skeel, R.D., '*Scaling for Numerical Stability in Gaussian Elimination*', Journal of the Association of Computing Machinery, Volume 26, Number 3, pp. 494-526, 1979.

7.5.7 Ruiz, D., '*A Scaling Algorithm to Equilibrate Both Rows and Columns Norms in Matrices*', Reports from Rutherford Appleton Laboratory, Computational Science and Engineering Department, 2001.

CHAPTER 8 - NUMERICAL EXAMPLE

So far in this dissertation several numerical examples have been presented which have successfully demonstrated the capabilities of MIXEDMODELS through its simple and efficient formulation-architecture scheme. Component-based modeling approach combined with the symbolic-numeric architecture allows the user to create component models at any level of abstraction and also to add new components "on the fly". The mathematical formulation of MIXEDMODELS is unique in the sense that it generates explicit symbolic equations giving the advantage to further extend the formulation to perform analytical sensitivity analysis. It is also seen from the examples that this approach can simulate a broad class of multidisciplinary systems and the method is computationally viable, efficient and robust.

The purpose of this chapter is to present a numerical example which integrates more number of disciplines and hence more number of components. It describes a multidisciplinary system which integrates components from four different domains viz. Electrical, Mechanical, Electromechanical and Control Systems. This system is generated by modifying the system discussed in Chapters 2-4.

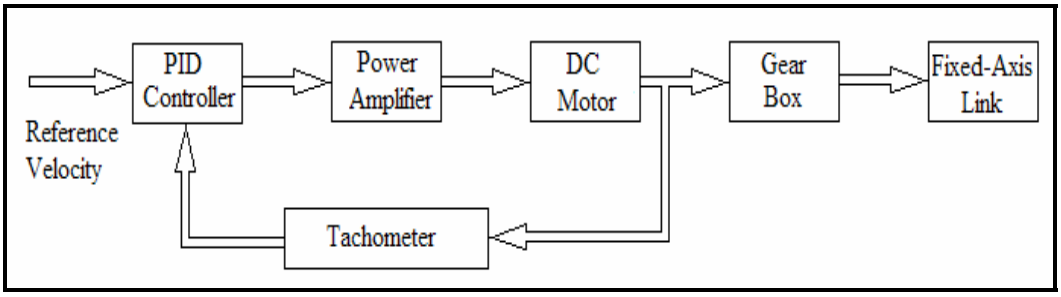


Figure 8.1.1a: Test System – Component Diagram

Component diagram of the system is depicted in Figure (8.1.1a). The DC motor is connected to the fixed-axis link through a gear box with a gear ratio of 12.5. The control objective is to maintain the angular velocity of the link constant at 5 rad/s. To achieve this objective the system uses tachometer feedback and a PID controller. Angular velocity of the DC motor is used as a feedback signal to the PID controller. A more intuitive picture of the final system is as shown in Figure (8.1.1b).

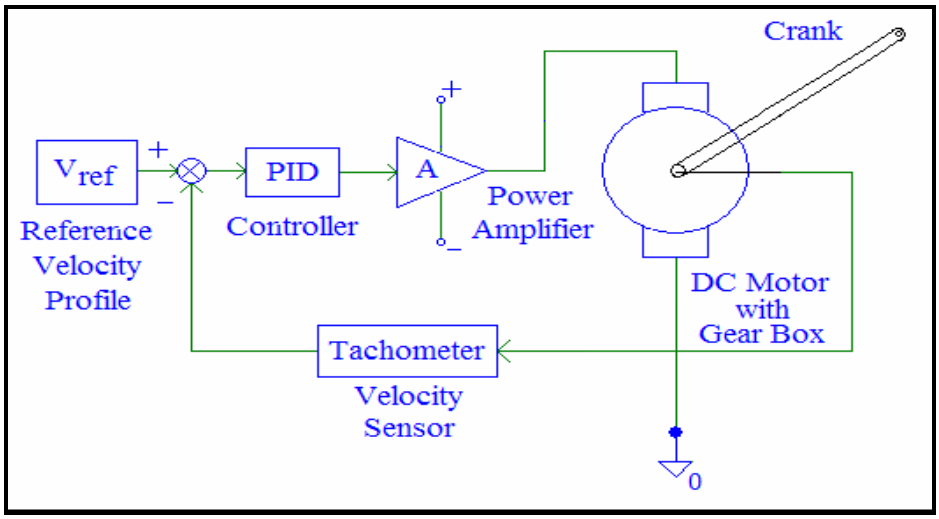


Figure 8.1.1b: Test System – Intuitive Diagram

The DC motor is powered by a power amplifier that consists of a macro-model of an operation amplifier (generally based on uA741 specifications - it being the most popular one) followed by a BJT-class B push-pull output stage. Figure (8.1.1c) shows a magnified view of the power amplifier.

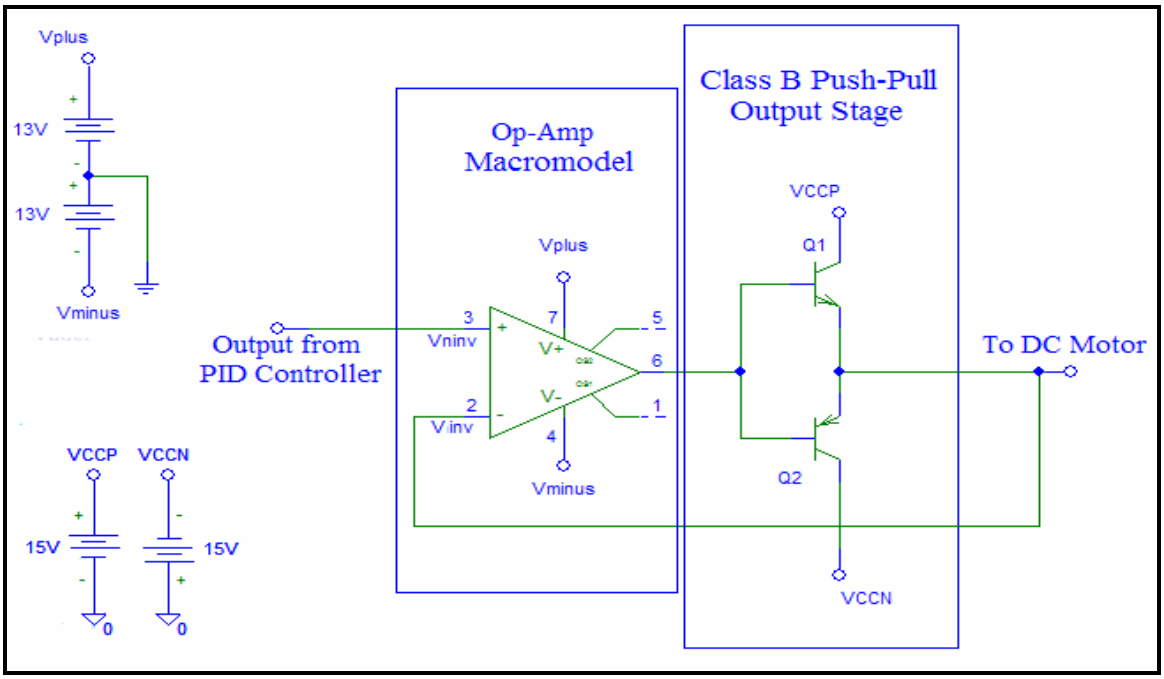


Figure 8.1.1c: Magnified View of Power Amplifier

The details of the op-amp macromodel can be seen in Figure (8.1.1d). The system was simulated in the MIXEDMODELS platform, and the results were verified against Simulink and PSpice during each development stage.

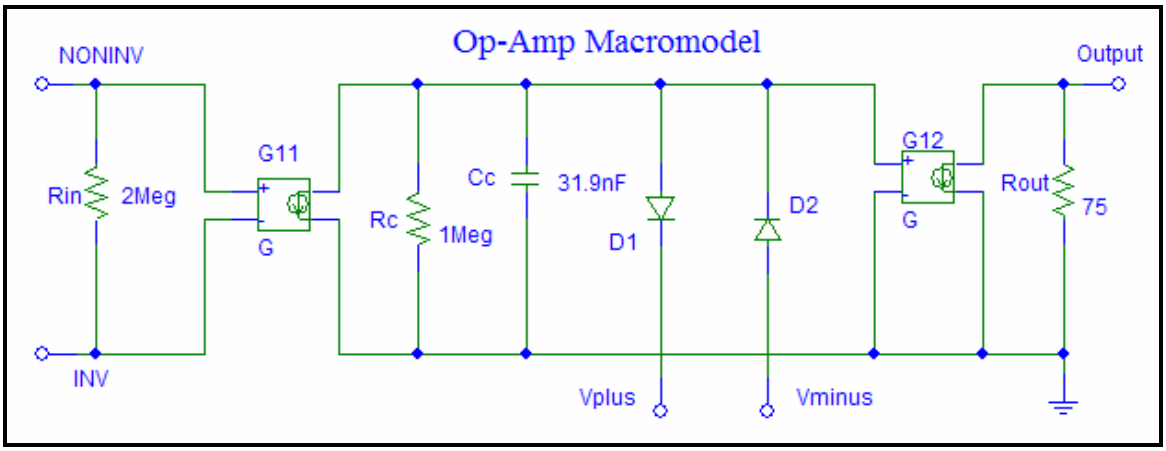


Figure 8.1.1d: Detailed View of Op-Amp Macromodel

The sensitivity analysis problem is developed for two design variables, R_{out} – the output resistance of the op-amp, and B_v – the coefficient of viscous friction of the DC motor. Sensitivities of the motor armature current and the angular velocity of the fixed-axis link are studied based on their responses to the design variables.

8.1 Simulation Set-up and Discussion

Simulations were performed in three different stages. The first stage was simulated without the power amplifier and the response was validated against Simulink. Figure (8.1.2) shows the Simulink diagram for this system.

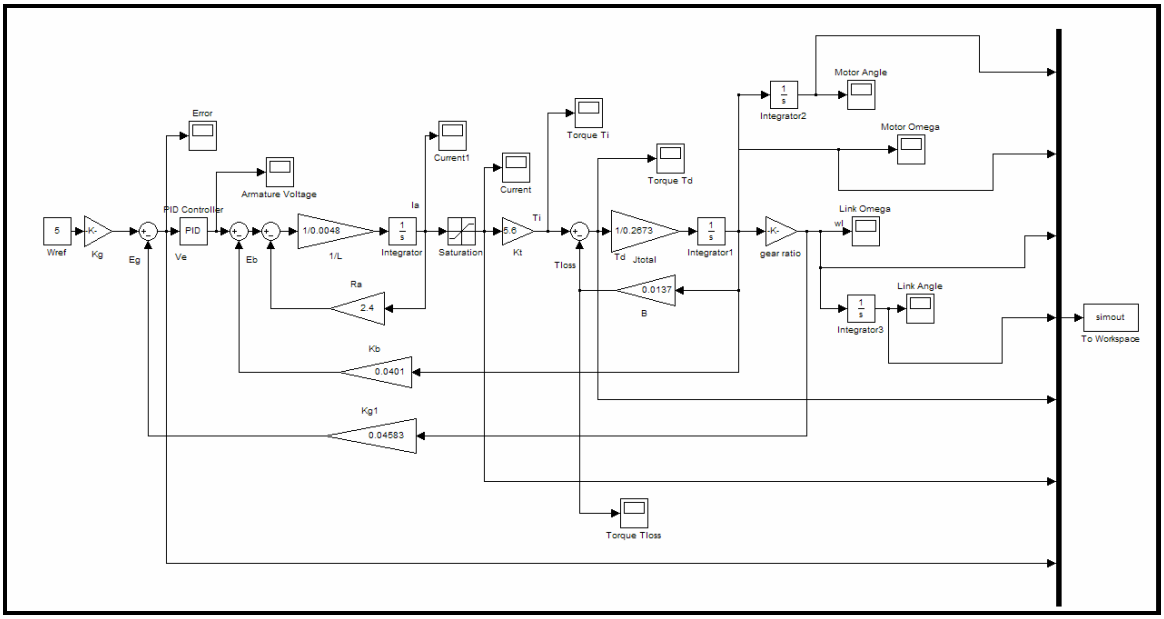


Figure 8.1.2: Simulink Model of the Test System

The goal of this simulation was to maintain angular velocity of the link constant at 5 rad/s. For simplicity there was no emphasis given on other criteria such as settling time, rise time etc. Note that the main objective is to be able to integrate different disciplines, and not to design the best controller. Figure (8.1.3) shows the system response where the Simulink results are superimposed on that obtained from MIXEDMODELS. It could be seen from the figure that the results are in very good agreement.

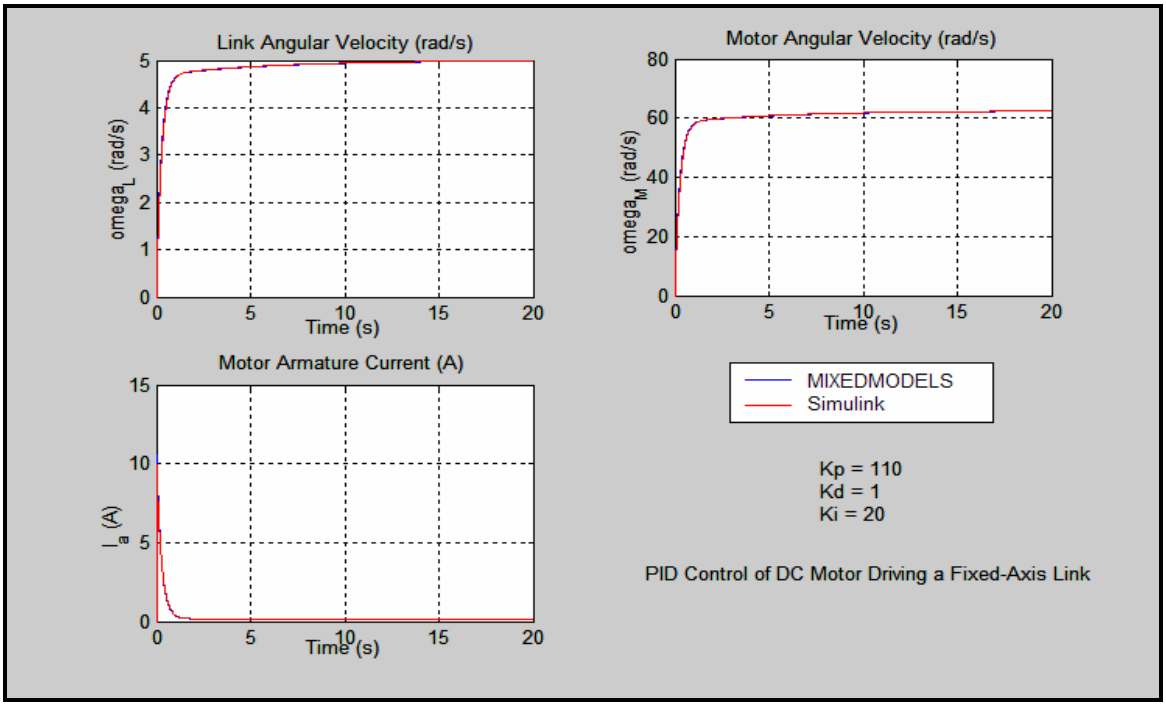


Figure 8.1.3: Time Response of the First Stage

After validating the system response against Simulink, op-amp macro-model was added to the system in a voltage-follower configuration. Op-amp is modeled to give a frequency response of 1MHz, and the design adds one more pole to the system making it a third order system. The PID controller was tuned accordingly to maintain the link velocity constant at 5 rad/s. The system was simulated using MIXEDMODELS and the response is as given by Figure (8.1.4).

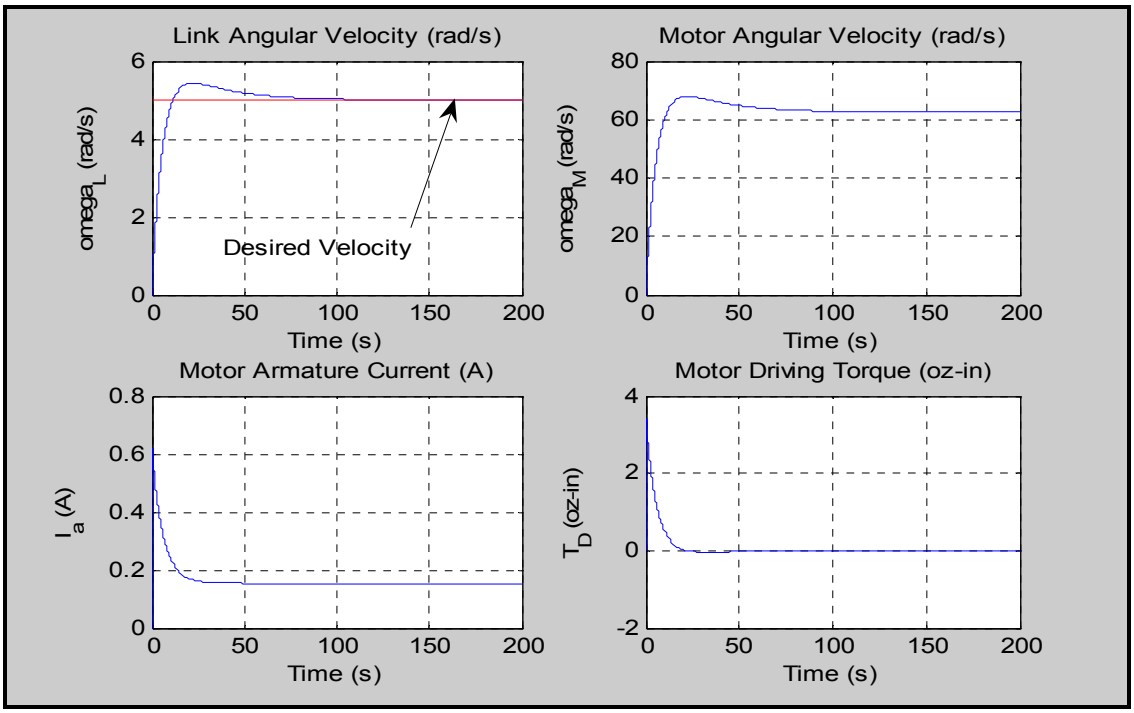


Figure 8.1.4: Time Response of the Second Stage

8.2 Sensitivity Analysis - Design Variables

For this example we performed sensitivity analysis for two design variables, R_{out} – the output resistance of the op-amp, and B_v – the coefficient of viscous friction of the DC motor. The system sensitivities are calculated with respect to each design variable using the method discussed in the previous section. In this discussion, sensitivities of motor armature current and angular velocity of link to the design variables are calculated and verified using perturbation analysis. The nominal and perturbed values of the design variables are as given below.

$\mathbf{d} = [R_{out}, B_v]$

$\mathbf{d}_{\mathbf{original}} = [75.0, 0.0137]$, $\mathbf{d}_{\mathbf{perturbed}} = [80.0, 0.0147]$

Figures (8.2.1) and (8.2.2) present the perturbation analysis results with link angular velocity and armature current as response variables, and output resistance of the op-amp as the design variable. Both plots show that the predicted change is in close agreement with the actual change found by finite differencing.

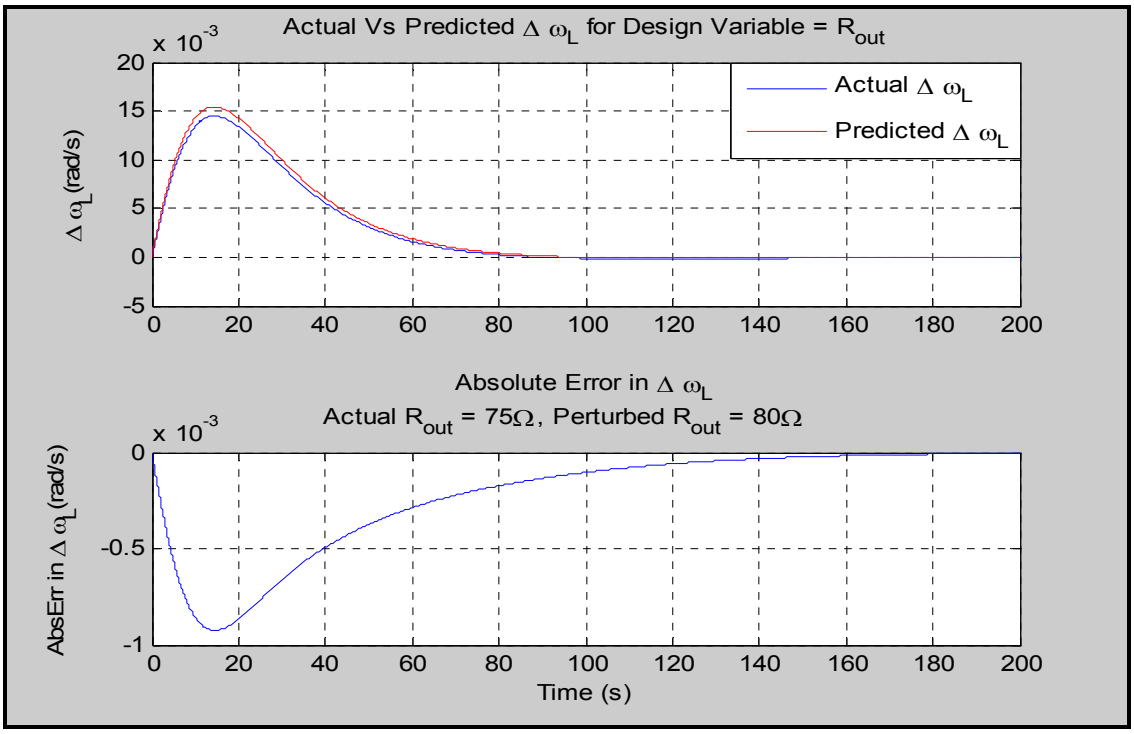


Figure 8.2.1: Actual Versus Predicted Change in Link Angular Velocity (Design Variable: Output Resistance of Op-Amp)

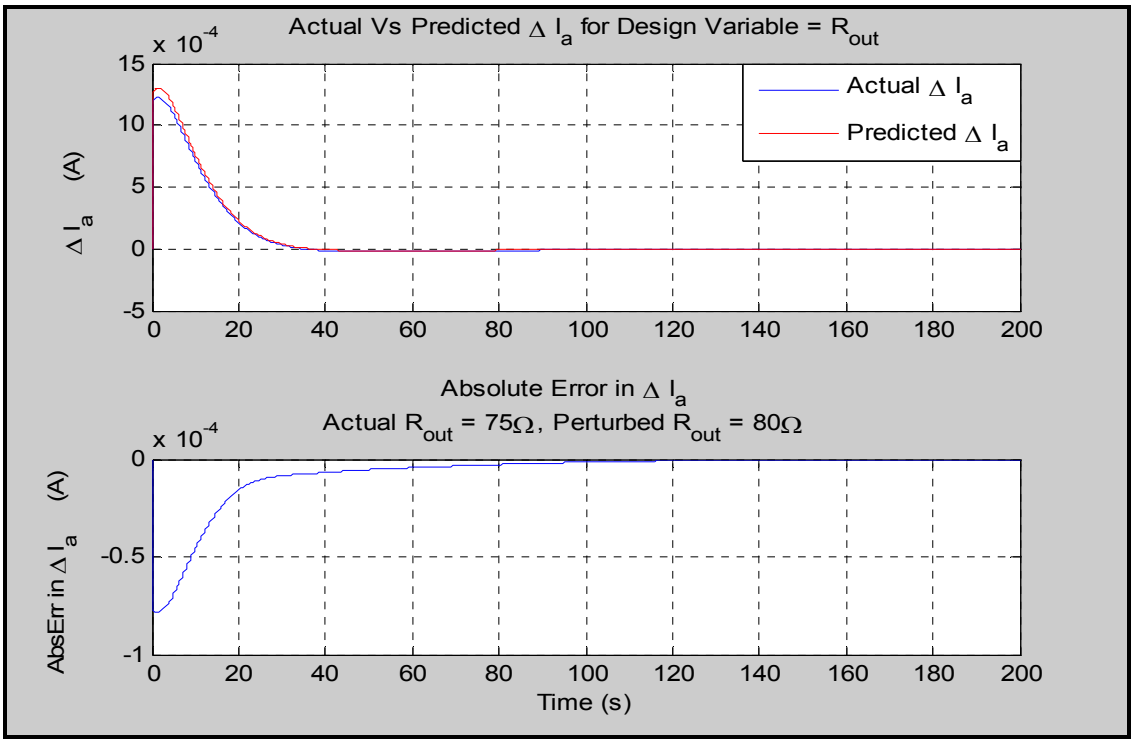


Figure 8.2.2: Actual Versus Predicted Change in Armature Current (Design Variable: Output Resistance of Op-Amp)

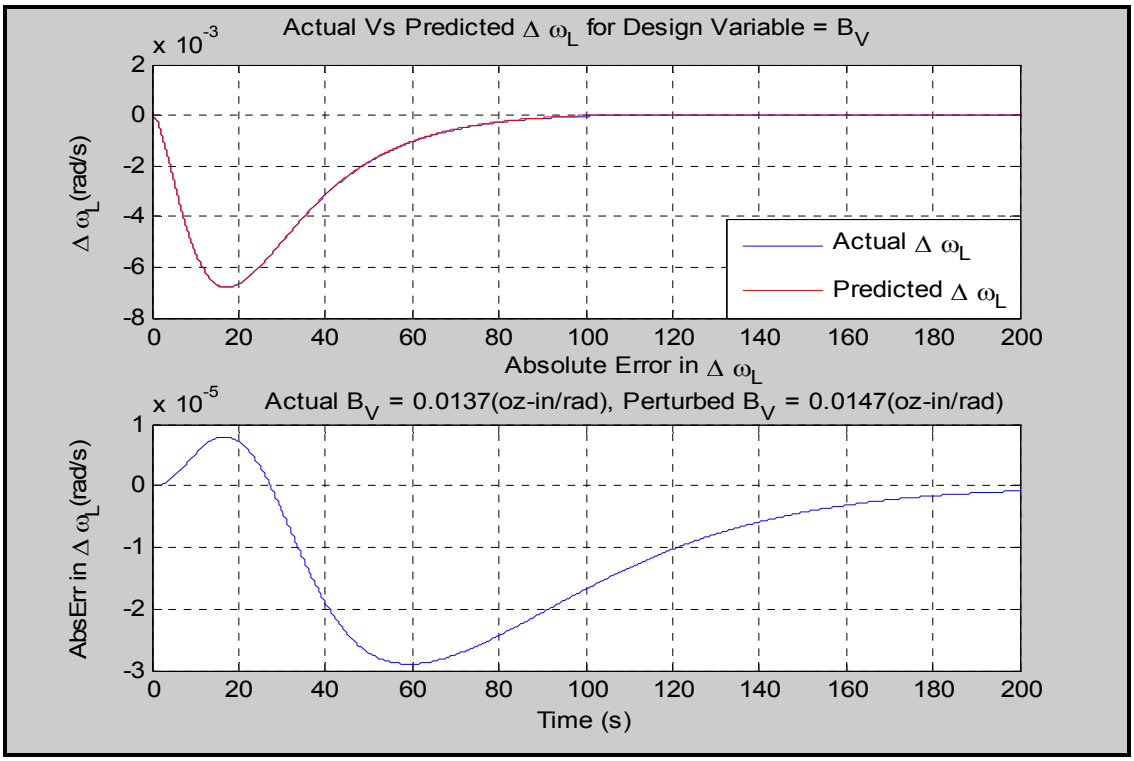


Figure 8.2.3: Actual Versus Predicted Change in Link Angular Velocity (Design Variable: Coefficient of Viscous Friction of DC Motor)

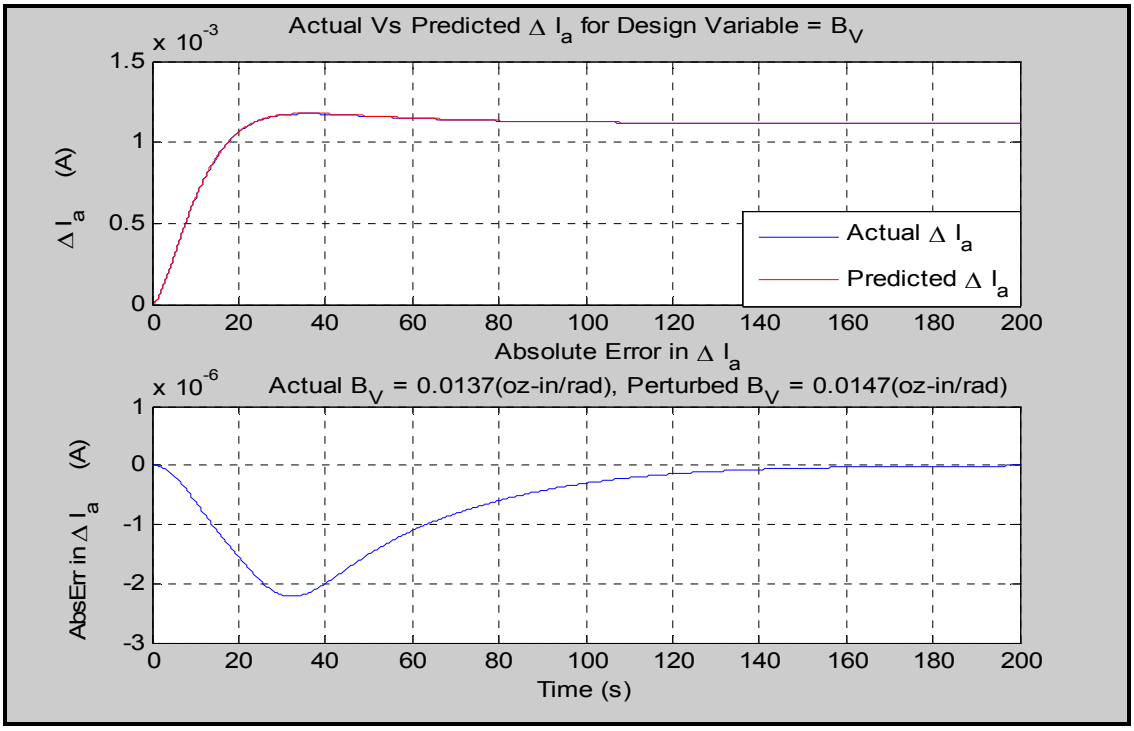


Figure 8.2.4: Actual Versus Predicted Change in Armature Current (Design Variable: Coefficient of Viscous Friction of DC Motor)

Similarly Figures (8.2.3) and (8.2.4) present the perturbation analysis results for the same two response variables for coefficient of viscous friction of the motor as the design variable. It can be seen from the plots that the results were in a good agreement with the actual calculations by finite differencing and thus verify that the method is accurate.

In the third stage, class B push-pull output stage is added to the amplifier and the time response is plotted which exactly matches the response as given in Figure (8.1.4). Sensitivity analysis is not performed on the third stage as it is still in the testing phase and more investigation is needed for the final simulations.

Transistor models need further testing and investigation and would be considered as work to be pursued in future. With this example we have successfully demonstrated that MIXEDMODELS is a simple, intuitive and straightforward approach and an efficient analysis and design platform for a broad spectrum of multidisciplinary systems.

CHAPTER 9 - CONCLUDING DISCUSSION

The final chapter of this dissertation gives an overview of what has been accomplished so far in the development of MIXEDMODELS. It also provides an extended discussion of the "work in progress" and in the end discusses the work that needs to be done in future.

9.1 MIXEDMODELS – Formulation and Architecture

This dissertation presented MIXEDMODELS – **M**ultidisciplinary **I**ntegrated e**X**tensible **E**ngine **D**riving **M**etamodeling, **O**ptimization and **DE**sign of **L**arge-scale **S**ystems, a platform which allows modeling, simulation and design of multidisciplinary systems, all in one scheme. MIXEDMODELS is a unified analysis and design tool based on procedural paradigm and symbolic-numeric architecture for multidisciplinary systems. The presented method is strictly acausal, local/global approach that offers great modeling flexibility at the component level, and facilitates extension to new problem domains. It allows users to seamlessly plug in components in his/her domain of expertise to the platform in a modular fashion. A component can be modeled using a mathematical formulation that most naturally describes its behavior in its domain. The user also has complete freedom to model a component to any level of abstraction. Thus, this formulation can be applied to a very general class of multidisciplinary systems including domains such as electrical, electronic, mechanical, hydraulic, pneumatic, controls, etc.

The system architecture of MIXEDMODELS can be accurately described by Figure (9.1.1). It incorporates three basic modules - a component library written in Maple; a symbolic preprocessing engine written in Maple; and a numeric engine, written in Fortran which also includes a set of problem independent numerical solvers. The symbolic-numeric architecture has several advantages that make it very attractive. Since each component model is in a separate file that is independent of other components, we have a high degree of modularity. By using symbolic computing to

formulate the governing equations and numeric computing to solve these equations, we achieve great flexibility and convenience in modeling along with efficiency in computation. The independence of the component models also provides easy extensibility. Extensibility is further enhanced by subsystem architecture and a file-naming convention that permits components or subsystems to be added without the need for any modifications in any existing code; and they are automatically included if the description of the system at hand refers to these components or subsystems. It is also seen from the examples that the code is very compact and is hence easily maintainable. This is in sharp contrast to object-oriented approaches such as openModelica that generally lead to lengthy and complex code.

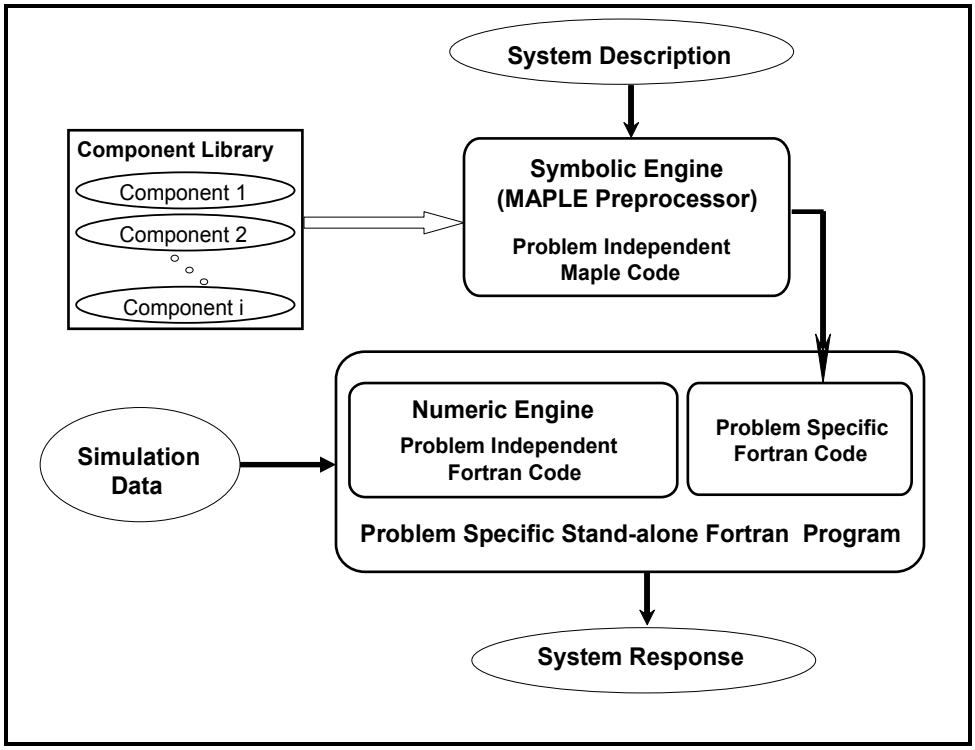


Figure 9.1.1: System Architecture – Structural View

The mathematical formulation developed in this work is a general nonlinear formulation for analysis and analytical design sensitivity of general multidisciplinary systems and has been successfully implemented in the MIXEDMODELS platform. The contributions of all the system components are combined symbolically by the symbolic engine which synthesizes the system governing equations as a unified set of nonlinear differential-algebraic equations (DAEs). The implementation generates

explicit equations in symbolic form which is conveniently used to extend the formulation for design sensitivity calculations. Note that MIXEDMODELS also supports a linear formulation, and a compatible sensitivity analysis formulation for the dynamic analysis of linear systems.

To obtain the sensitivity information, the system equations can be differentiated with respect to design variables to obtain an additional set of DAEs in the sensitivity coefficients. There are two popular approaches for analytical sensitivity design analysis, the direct differentiation approach and the adjoint-variable approach. The adjoint-variable approach has the potential to reduce the computational overhead, however; it requires integrating the adjoint equations backwards in time. This can cause serious difficulties in error control and significant complexities in software implementation. Therefore in this approach we chose the direct differentiation approach to develop the sensitivity formulation.

After differentiation the combined set of DAEs are numerically solved to obtain the solution for the state variable and state-sensitivity coefficients of the system. Knowing the system performance functions, we can calculate the design-sensitivity coefficients of these performance functions by using the values of the state variables and state-sensitivity coefficients obtained from the DAEs.

In order to start the integration from the given initial time, initial conditions must be provided for all the differential variables that the user wishes to solve for. Specifically, initial conditions must be given not only on system differential variables $\mathbf{X}$, but also on the sensitivities $\mathbf{X_d}$. In specifying initial conditions, care must be taken to ensure that the specified initial conditions are physically realizable and consistent with all system constraints. While it may be very difficult for the user to manually supply a consistent set of initial conditions on $\mathbf{X}$ for a large system; it might become overwhelming if the user were called upon to provide initial conditions on $\mathbf{X_d}$ as well. To ensure consistency of initial conditions MIXEDMODELS formulation provides a simple scheme where the user needs to provide a set of consistency equations that must be satisfied by the initial conditions.

The user can then supply initial guesses on $\mathbf{X}$ and $\mathbf{X_d}$, and Newton-Raphson algorithm can be used to obtain the exact initial conditions.

The numeric engine of MIXEDMODELS supports two single-step numerical integrators DVERK and RKF45 (based on Runge-Kutta methods) and two multi-step integrators LSODE and DLSODES (based on Gear's algorithm). All these are widely available ODE solvers that are written in FORTRAN and are provided by NETLIB repository. DLSODES is a sparse version of LSODES with variable step size and order and solves the initial value problem for stiff or non-stiff systems of first order ODEs. The numeric engine also supports a nonlinear solver based on the Newton-Raphson algorithm and a linear sparse matrix solver Y12MAF, also obtained from NETLIB repository. In addition to these solvers, to improve numerical performance based on speed and accuracy, MIXEDMODELS also implements a symbolic reduction routine and a scaling routine. The symbolic reduction routine is written in Maple and the scaling routine [9.7.1] is written in FORTRAN. The scaling algorithm performs row and column scaling of the linear system thereby improving the numerical stability and accuracy of the system. From the simulation results with the reduction routine it is observed that the simulation times are reduced to almost 50% when compared with the original runs.

To demonstrate the efficacy of MIXEDMODELS, the proposed approach was successfully demonstrated on several numerical examples as listed in Table (9.1.1). Design sensitivity calculations are validated using finite differencing while dynamic analysis calculations are validated using PSpice, MATLAB or Simulink. The results indicate that the proposed approach and software architecture is very effective in terms of accuracy, modeling convenience, computational efficiency and the ability to simulate the behavior of a general class of multidisciplinary systems. The symbolic architecture as implemented in MIXEDMODELS is computationally viable and efficient even for large-scale systems and does not lead to expression swell. This is because as the system size grows, the number of equations to be solved for increases; however, the symbolic complexity of the expressions does not grow with system size.

- **Electrical System –** A second order RLC circuit (*m)
- **Linear Electromechanical System –** DC motor driving a single link, powered with a constant dc voltage source, with and without saturation (*s)
- **Nonlinear Electromechanical System –** A slider-crank mechanism with a constant dc voltage source, with and without motor current saturation (*s)
- **Nonlinear Electronic System –** Diode half-wave rectifier and a full-bridge rectifier (*p)
- **Nonlinear Electromechanical System –** DC motor driving a single link with a half-wave rectifier and a full-bridge rectifier, with sensitivity analysis (*s, *p, *fd)
- **Linear Electromechanical System Integrated with Controls –** Velocity tracking control of a DC motor driving a single link using a tachometer feedback, with and without current saturation, (*s)
- **Hybrid Electromechanical System Integrated with Control Engineering –**Velocity tracking control of a DC motor driving a single link using a digital encoder feedback, with and without current saturation, (*s)
- **Op-amp Circuits** – A macro-model in inverting and non-inverting configuration, (*p)
- **Nonlinear BJT Circuits –** Common emitter configuration with *npn* and *pnp* transistor models based on PSpice BJT components, (*p)
- **Linear Op-amp Circuits –** Power amplifier using a macro-model followed by a class B push-pull BJT stage, (*p)
- **Nonlinear Electromechanical System –** Slider-crank mechanism with a half-wave and full-bridge rectified power supply with sensitivity analysis, (*s, *p, & *fd)
- **Nonlinear Electromechanical System Integrated with Control Engineering –** Velocity tracking control of a DC motor powered by a power amplifier with a class B push-pull output stage, driving a single link using a tachometer feedback, with sensitivity analysis, (*s & *fd)

Table 9.1.1: System Examples Simulated in the MIXEDMODELS Platform

Time Responses Validated Against

***m – MATLAB, *s – Simulink, *p – PSpice, *fd – Finite Differencing Technique**

Also, in a component-based approach it is generally true that the connection components add more equations and redundant variables to the system which further increases the system size. The symbolic engine includes a symbolic reduction facility which can be used to substantially reduce the size of the Jacobian matrix. This makes the solution process numerically much efficient even for large scale systems.

However, there is much room for enhancing numerical stability and robustness. One of the steps in this direction is exploring multirate integrators, which will allow us to use different step sizes for different components of a system of ordinary differential equations. This may give us better control on tuning the error tolerances and step sizes to improve efficiency. This may be significant in case of systems that exhibit stiff behavior in some parts and non-stiff in other. Choosing the right step sizes to begin with might be a difficult task, and to be able to intuitively change step sizes as required at runtime might be even harder. But certainly this is an approach which would be explored next to improve the performance of the numerical engine.

9.2 Metamodeling using Sensitivity Information

It can be seen from the study carried out in this work that extension of sensitivity analysis to generate metamodels for mechatronic systems is a very promising idea. The parametric design of mechatronic systems requires several detailed analyses of the system, thereby slowing down the design process significantly. In the recent past, there has been a lot of interest in using lower fidelity, but higher-efficiency metamodels (also called surrogate models) instead of the actual detailed models to guide parametric design, particularly in the early stages of parametric design. One common approach to forming metamodels is to run the detailed model to obtain the system response at selected points in design space and fit a response surface to the results which becomes the metamodel to predict performance at other points in design space. In this work, we present a metamodeling approach for mechatronic systems that computes and utilizes first-order derivative information at each point in the design space at which a detailed analysis is performed. The first-order derivative information that is computed is the set of design sensitivity coefficients of the system state variables and performance functions. Using the analysis and sensitivity analysis

formulation of MIXEDMODELS the necessary sensitivity information is obtained first and then several schemes for generating a metamodel for the system are devised. Two very quick and efficient methods for metamodeling of mechatronic systems that were developed in this work are as follows. The first method uses a selected number of reference designs to obtain a set of predicted responses at the new design point and the best one will then serve as our model approximation. In the second method a single predicted response is obtained by forming a weighted sum from the individual predictions acquired from the first method. Computationally, both these methods are extremely lean and simple to implement, which are the desired characteristics of a metamodel. Further, the first method in particular is capable of giving an indication of the quality of the metamodel, which is very important to the designer.

To demonstrate the approach, a numerical example considering a slider-crank mechanism powered by a half-wave rectifier was considered. Sensitivity analysis was performed on this system for two design variables, slider mass and source voltage. Slider acceleration was chosen to be response variable. It can be seen from the results that the proposed metamodeling approaches are capable of producing fairly good approximations of the system response at the new designs, but not all reference points produce good predictions. The advantage of making individual predictions at reference design points is that it is easy for the user to assess the quality of the predictions and choose a good one, while discarding outliers. On the other hand, the weighted sum approach generally produces better approximations, but cannot give an indication of whether there is a problem with the prediction or not (in practice, the actual solution will not be available). It appears that a combination of the two methods may be a good option in practice. It should be noted that in order to use the second method, we need to perform all the calculations of the first method. However, the extra calculations for the second method are minimal, so there is little computational overhead. The first method can then give an indication of the reliability of the prediction, and the prediction obtained using second method can be used as the approximate system response. If the predictions are not accurate enough, a finer grid of reference designs may have to be used. The weighting scheme can

also be improved. Interestingly, the nearest reference point to a new design is not necessarily the best one to use to make the prediction.

At this point, it does appear that sensitivity-based metamodeling for mechatronic systems is feasible, but clearly several improvements are needed. Adaptive selection of reference designs, improved weighting, and higher order sensitivity analysis are possible improvements. Improved surface fitting techniques may also prove worthwhile, although this will increase the amount of computation required.

9.3 Extension of MIXEDMODELS Formulation to Discrete Device Modeling – Concept of Monitor Functions

The concept of monitor functions is a very important aspect of this proposed approach. A multidisciplinary system while in operation can undergo qualitative changes in its behavior. Such an event changes the system dynamics which is reflected in the governing equations of the system. One example of this is current saturation in a DC motor, where the voltage-current relationship changes when we enter or leave current saturation. Current modeling approaches do not handle this type of behavior very well. Facilities are available for finding the zeros of functions, such as the zero-crossing feature in Simulink, but these are not directly coupled to the required changes in the system governing equations.

In this approach this difficulty is resolved by allowing monitor functions and alternate sets of component equations in the model. The monitor functions are defined so that when such a function goes through a zero, it indicates the occurrence of an associated qualitative change in the component's behavior. The system then switches to an alternate set of governing equations which describe the new dynamics.

Every function, whether it is a monitor function or a governing equation, is associated with a flag which indicates whether this function is "on" or "off". Only functions that are "on" are considered to be active during the analysis, and the others are ignored. The active monitor functions are checked at every time step to see if any of them have gone through a zero. If this has happened within the last integration time step, then the associated logic of the monitor function(s) concerned is executed

to reset the "on" and "off" flags of all the governing equations and monitor functions. The integration is then restarted from this point. If the last time step was large, interpolation can be used to find the exact time when the change has occurred and integration can be restarted from this point. Monitor functions thus enable the system to respond to qualitative changes in its behavior by turning the appropriate functions on or off.

To demonstrate the concept of monitor function a few systems were simulated including a hybrid system (Figure (9.3.1)) that uses encoder feedback to control velocity of a DC motor driving a fixed-axis link. The results are published in the proceedings of the IEEE International Conference on Mechatronics and Automation in 2005 [9.7.2]. The following paragraphs briefly discuss the implementation of this system and also present the simulation results.

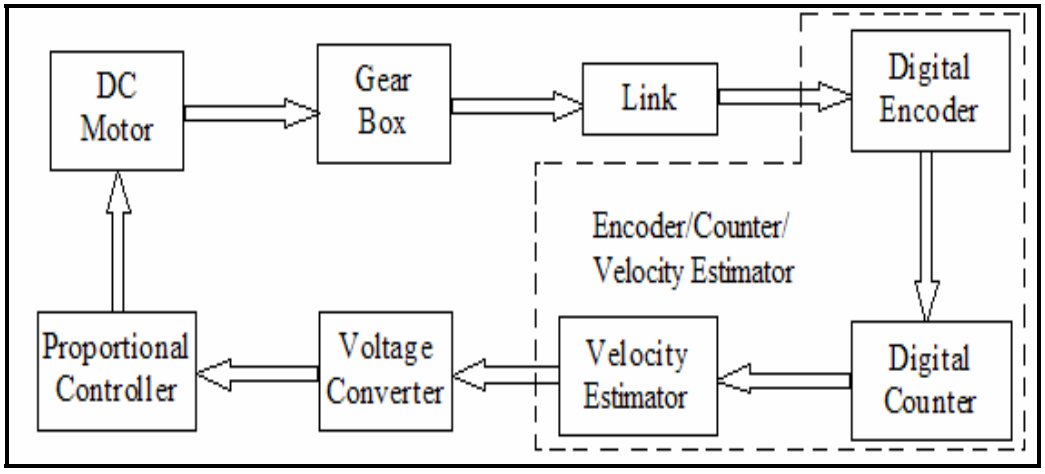


Figure 9.3.1: Velocity Control using Encoder Feedback

The encoder, which senses link position, is a discrete device with a resolution of 10 bits and is used to estimate velocity. Proportional control is used to maintain the link velocity at 5 rad/s. The system is simulated using component approach with the discrete device sampled at 0.1s and 0.01s and the integration step size is set as 0.001s. First the system is simulated with no limit on armature current. In the second simulation motor armature current is limited to 0.25 A. When the system enters current saturation, it changes its system dynamics and the DC motor switches to a different set of equations which are valid only in the saturation region. Again when the system comes out of saturation, it switches back to the original motor model. The

simulation results are shown in Figure (9.3.2). Both these simulation results were compared with the results for a continuous system with a tachometer feedback, which was simulated using Simulink. In both cases, the results for the link velocity were in good agreement. However, the hybrid system shows oscillations in the current and torque plots. This is due to the phase shift and truncation error in the velocity estimation using the encoder. The steady-state error and oscillatory response get worse when the discrete component is sampled at a higher frequency. The motor armature circuit has a low-pass filtering effect, but the inductance being very low, this filtering effect is quite weak. The filtering effect is seen more prominently if the armature inductance is increased by a factor of 100. Alternatively, such a system might require an external low-pass filter.

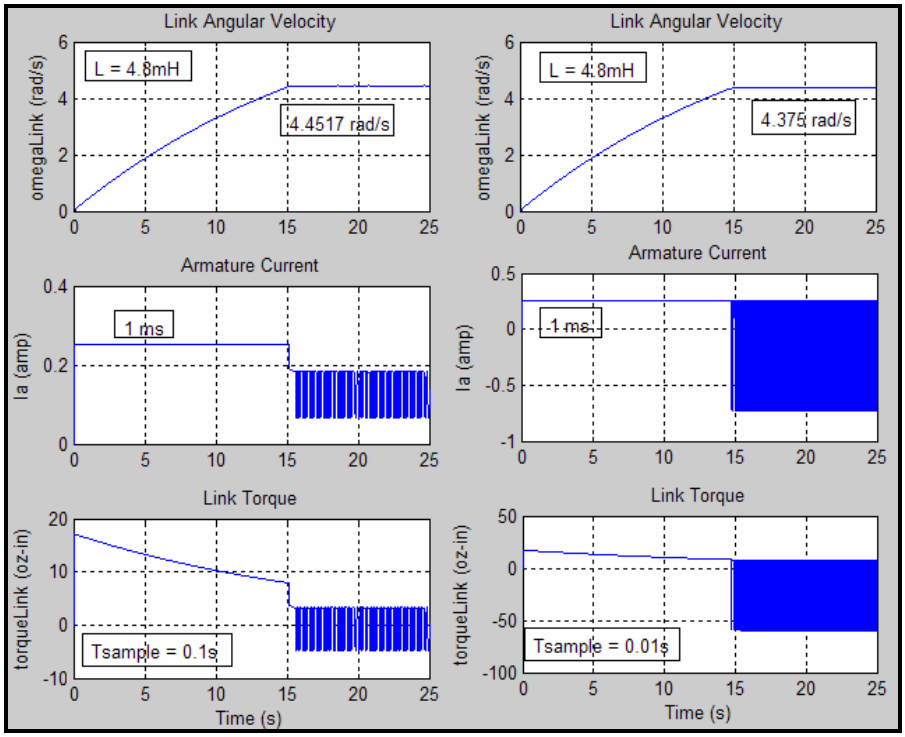


Figure 9.3.2: Hybrid System Simulation

It should be noted that a continuous model of this system would give the correct velocity response, but would not show the torque oscillations, which are critical for designing the shaft. This points out the importance of being able to simulate hybrid electromechanical systems accurately so that such effects can be captured.

Currently, in this formulation, monitor functions are implemented on a case-by-case basis, which means that for a particular system the user needs to set up the alternate model equations in a subroutine called a "userFunction", which needs to be written in FORTRAN. The numeric engine at runtime calls this subroutine and switches to the alternate models whenever the system undergoes a change in behavior. We believe this idea of monitor functions can be used to model discrete devices and also sensor fusion techniques, where the components have different functional states. It is not desirable to require the user to provide a separate subroutine for each system and so in future the plan is to generalize the concept of monitor functions.

One of the ideas in progress is to include alternate models in the description of the component itself, where all the possible alternate models for a component will be written in one single "*.model*" in Maple. Suppose that a component is described by a governing equation of the following type.

$$F_1 = a_{11}X_1 + a_{12}Y_1 \tag{9.3.1}$$

Let the alternate models be defined by,

$$F_2 = a_{11}X_2 + a_{15}Y_1 \tag{9.3.2}$$

$$F_3 = a_{11}X_3 + a_{12}Y_5 \tag{9.3.3}$$

Here X_i and Y_i are system variables and a_{ij} are system parameters. The idea being proposed here is to combine all these three equations symbolically in one equation and choose the right model at runtime by setting or resetting appropriate flags. For example, in this case the model equation can be written as,

$$F = mF_1*F_1 + mF_2*F_2 + mF_3*F_3 \tag{9.3.4}$$

Depending on the active state of a component at current time, the flags mF_i can choose either a value of "0" or "1" so that only one model is active at a time. When the system under study undergoes a qualitative change, the component model will be switched to the appropriate model for that state by resetting its original model flag to "0" and setting the alternate model flag to "1". We believe that this concept may come handy in the digital world.

This design of monitor functions is still in the development stage and would be tested and implemented as a part of future work.

9.4 Extension of the Formulation to Include Subsystems

A subsystem represents a system within another system, with each system being a collection of several other components. In electrical, electronic or control engineering domains it is very common to find an occurrence of a component which itself is a collection of components. Embedded system architecture which is gaining much popularity and attention is also a classic example where subsystem architecture would be very convenient. For example, an op-amp component itself is a collection of several other components such as transistors, mosfets, diodes, resistors, capacitors, electric nodes etc. and thus qualifies to be a subsystem. This can be further explained by another example, viz., a circuit such as an "instrumentation amplifier" includes four op-amps in addition to other simple components. If an op-amp component consists of 50 simple components then it is not practical to model an "instrumentation amplifier" with 200+ components, instead it is more convenient for the user to consider an op-amp as a subsystem and have 4 only op-amp components letting the system architecture take care of its constituent components. Another discipline which displays the significance of subsystem modeling is control engineering where embedded multi-loop control of a system is commonly used, especially in aircraft control.

To provide convenience and simplicity in modeling, MIXEDMODELS formulation is extended to include subsystems. A subsystem is considered to be a component and so its file structure is similar to that of any component's. All subsystems belong to the component library of MIXEDMODELS. Recall that for an expert to contribute a component to the library he/she needs to provide two files, viz., a *model* file and a *header* file, to completely describe a component. In case of a subsystem the user needs to provide three files, viz., a *subsystem* file with the file extension "*.sub*", a *header* file, with the file extension "*.hdr*" and a *setup* file that sets up all arrays for a subsystem, and has the extension "*.setup*". All these files are written in Maple.

Input data for subsystems is organized in three data files provided by the user. Recall also that the specification file named "*specFile.ems*" specifies the system size, i.e. the number of components. Now it also needs to specify the number of subsystems at the top-most level of hierarchy. The total number of components specified in the component data file excludes those components which belong to subsystems. The neat part of this structure is that in the component data file there is no restriction on the order in which the components and subsystems should be arranged. The second input file named "*subSys.ems*" lists the types of subsystems that occur at the top-most level. The third file is the component data file which provides parameter lists for components as well as subsystems. In case of subsystems, the top-level subsystem should have in its documentation information about all other subsystems in its subsequent levels. In other words every parameter array should contain parameters of all the components in a subsystem, including a subsystem within that subsystem.

The symbolic engine of MIXEDMODELS contains a set-up file named "*SubSysSetArrays.txt*" that is responsible for the following.

- It defines arrays and assignments for parentIDs, i.e., subsystem IDs for the *parent* nodes. *parentID* array is a two-dimensional array with the row index corresponding to the number of top-level subsystem and the column index corresponding to the i^{th} top-level subsystem including the top-level subsystem itself.
- The user enters string valued parameters, real valued parameters and integer valued parameters for all the components within a subsystem in a single dimensional PSS, PRS and PIS arrays respectively. This file splits up these arrays in order to establish one-to-one mapping between a component and its parameters.
- Using the information in the file "*specFile.ems*" and "*specSubSys.ems*" this file also calculates the total number of components in the system. This includes the components within subsystems and the components at the topmost level, provided by the user in the component data file

- It invokes the two-pass procedure to populate subsystem data.

Once the total number of components is known, the symbolic engine is ready to scan the component data file. The subsystems are traversed in a depth-first order and the information is populated in appropriate arrays using a two-pass procedure similar to the component structure. For example, consider the following arbitrary tree structure of subsystems, which has two top-most levels of subsystems, viz., *subSys1* and *subSys14*. These two would be considered as the parent nodes while their subsequent subsystems would be considered as leaf nodes. This tree structure is as shown in Figure (9.4.1). The subsystem numbers in the figure indicates the direction of traversal.

Prior to invoking the two-pass procedure, using the information provided in the specification files, the set-up file sets up required arrays for the subsystems. It then invokes the two-pass procedure. In the first pass it scans the *specSubSys* file and executes header files for all the top-level subsystems in the order in which they appear in the specification file. It then calculates the total number of components in the systems and proceeds to the second pass where it executes all the "*.sub*" files again in the same order in which the subsystems appear in the file. Once the symbolic engine has the knowledge of the components of each subsystem, a similar two-pass procedure for the components is executed and system matrices are populated as explained in chapter 5. In any subsystem architecture it is possible that the user may name a component with an ID which is already used by a subsystem in the component library, for example, an electric node can be named by the user as N_1 and the subsystem op-amp also might have an electrical node component with the same ID. However the user should not have to worry about the component IDs inside a subsystem. For this purpose the software architecture internally generates a unique ID for every component in the system. The unique ID is generated in such a way that one can trace any component even in the last leaf of the subsystem tree.

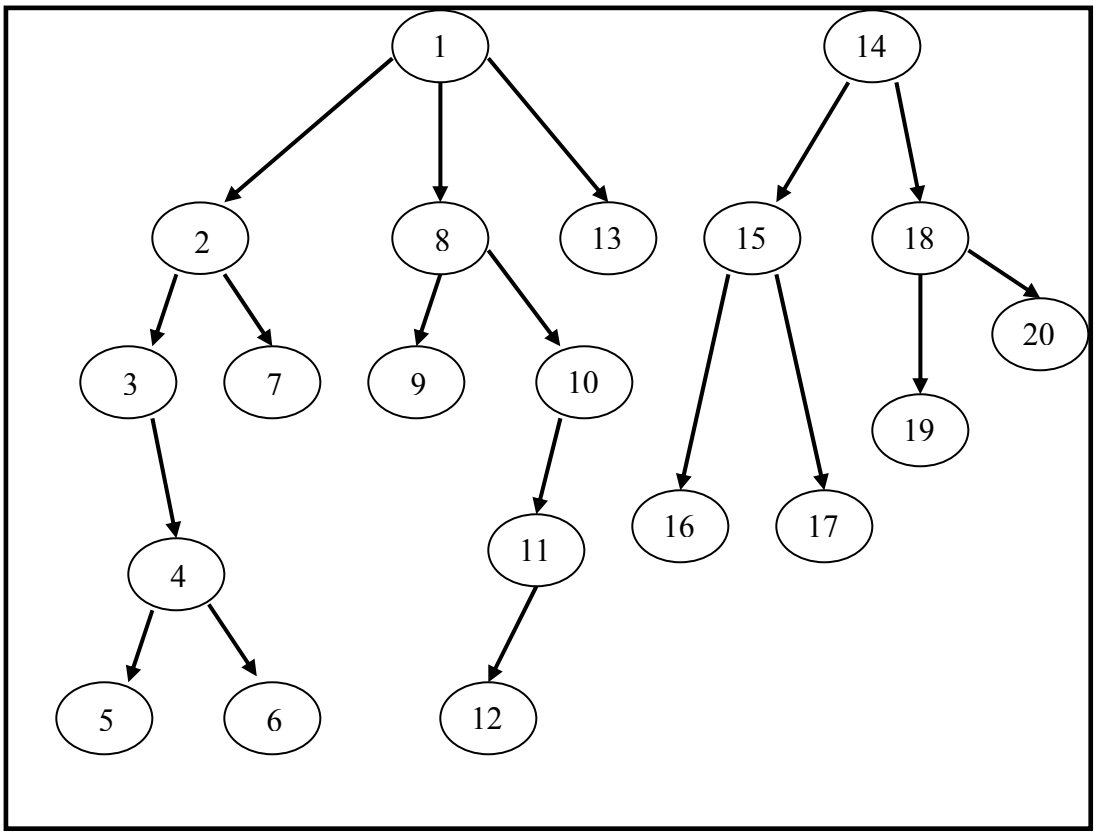


Figure 9.4.1: Subsystem Example Tree Structure

To provide suitable software architecture for the mathematical formulation there is a significant amount of bookkeeping and ordering involved, and yet the code is very compact and hence easily maintainable. The presented subsystem architecture has been successfully implemented in the MIXEDMODELS platform and it also has been tested on simple systems such as a subsystem of three parallel resistors and a subsystem of one resistor in series with another subsystem of two parallel resistors.

9.5 Existing Component Library

Currently MIXEDMODELS component library supports a limited number of electrical, mechanical, electronic, control and electromechanical components. Table (9.5.1) lists all the existing components.

The current component library is sufficient to demonstrate the capabilities of MIXEDMODELS, however; it needs to be extended not only by contributing new components from the existing domains but from other domains such as hydraulic, pneumatic etc. Subsystems also should be developed in various domains.

- **Electrical Components**
  - *Passive Components*: Resistor, Capacitor, Inductor
  - *Semiconductor Devices*: pn junction diode, BJT (npn, pnp)
  - *Connection Components*: Electric Node
  - *Macromodels*: op-amp
  - *Electrical Sources*: Independent Sources - V_{DC}, V_{AC}, V_{sin}, V_{gnd}, I_{DC}

    Dependent Sources - VCVS, CCVS, VCCS, CCCS, polyCCCS
- **Mechanical Components**
  - *Bodies*: Rigid Body, Gear Box, Fixed-Axis Body
  - *Joints*: Revolute Joint, Stabilized Revolute Joint
  - *Connection Components*: Mechanical Connection, Slider Constraint, Fixed-Axis Body Constraint
- **Control System Components**
  - PID Controller, Gain Block
- **Actuators**
  - DC Motor
- **Sensors**
  - Tachometer

Signals and Systems

- Summer, Signal Tap

Table 9.5.1: Existing Component Library

9.6 Recommendations for Future Work

Now that the mathematical formulation for analysis and sensitivity analysis of MDS along with compatible software architecture is in place, it opens up several research areas to extend the capabilities of MIXEDMODELS. The following is a list of several suggested areas for advance research:

- Generalization of the concept of monitor functions
- Discrete device modeling
- Modeling of control algorithms such as LQR control, Kalman Filter etc.
- Concurrent optimization techniques for multidisciplinary systems
- Extension of component and subsystem libraries
- Exploration of new techniques to enhance numerical stability, robustness and efficiency
- Modeling of sensor fusion strategies
- Exploration of the possibility of creating an open-source for MIXEDMODELS
- Creation of a graphical front end for MIXEDMODELS

In this dissertation we have successfully developed MIXEDMODELS - a modeling, simulation and design platform for multidisciplinary systems, which has the potential to be developed into a self-contained simulation package for research and commercial applications.

9.7 REFERENCES

9.7.1 Ruiz, D., '*A Scaling Algorithm to Equilibrate Both Rows and Columns Norms in Matrices*', Reports from Rutherford Appleton Laboratory, Computational Science and Engineering Department, 2001.

9.7.2 Vaze S., DeVault, J., Krishnaswami, P., '*Modeling of Hybrid Electromechanical Systems using a Component-based Approach*', IEEE International Conference on Mechatronics and Automation, ICMA 2005, 204-209, 2005.

APPENDIX A: DETAILED DERIVATION OF MIXEDMODELS FORMULATION

The development of an integrated optimization scheme for multidisciplinary systems, which was the primary focus of this research, requires a unified modeling, simulation and sensitivity analysis platform. This dissertation presents MIXEDMODELS, a unified analysis and design tool for multidisciplinary systems that is based on a procedural, symbolic-numeric architecture. MIXEDMODELS (Multidisciplinary Integrated eXtensible Engine for Driving Metamodeling, Optimization and DEsign of Large-scale Systems) involves two major parts – the mathematical formulation and the software architecture to support the formulation. The formulation needs to be general enough such that it captures all the aspects of an MDS and is also able to generate explicit set of system governing equations allowing easy extension to parametric studies such as sensitivity analysis and optimization. This appendix discusses the details involved in the development of analysis and sensitivity analysis formulation of MIXEDMODELS. MIXEDMODELS implements a linear first-order form to represent the system governing equations, which requires that the equations be converted to a predefined form by the process of differentiation. This form was found to have some limitations and therefore a general nonlinear formulation was developed for analysis of MDSs, the details of which are discussed in this Appendix. This appendix also provides a detailed discussion of the sensitivity formulation which was developed based on the nonlinear analysis formulation.

A.1 Example of an Multidisciplinary System

Consider the system shown in Figure (A.1.1). It represents a system which integrates multiple disciplines such as, in this case, electrical, mechanical and electromechanical. In this system, a DC motor, powered by a half-wave rectifier drives a fixed-axis link. One way to look at such a system is a collection of interacting components, where the components can come from any physical domain.

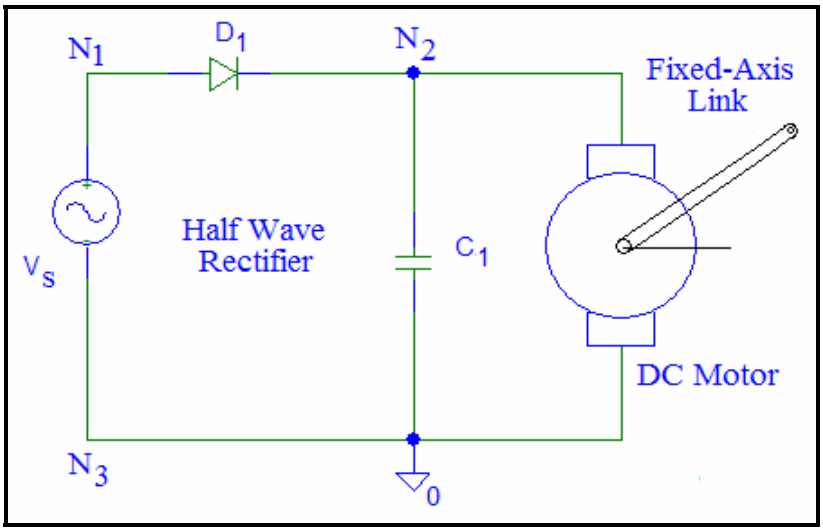


Figure A.1.1: DC Motor, Powered by a Half-Wave Rectifier

Driving a Fixed-Axis Link

In this example, bipolar junction diode D_1, capacitor C_1, sinusoidal voltage source V_s and the ground component belong to the electrical domain. Fixed-axis link component belongs to the mechanical domain; where as, the DC motor component belongs to the electromechanical domain. Given such a system, to simulate its system response, there should be a way to derive its mathematical model that provides a unique solution. This essentially means that one should be able to accurately model all the components of a system and the interactions between these components. One of the possible approaches to model interactions between components is by introducing new components that can accurately capture these interactions. For example, in Figure (A.1.1), an electric node component, N_2, can be introduced to capture interaction between the diode component and the capacitor component. The interaction between these components can be described by Kirchhoff's current law (KCL) which states that the algebraic sum of the currents at node N_2 should be zero. Thus KCL at node N_2 would be the governing equation of the node component N_2. Therefore, assuming that the interaction between any two components, irrespective of their domain, can be modeled by introducing a suitable component, if we can find a general way to describe any component, then it would be possible to model any multidisciplinary system in a unified way.

For the system shown in Figure (A.1.1), let us introduce new components that describe interactions between the interconnecting components. Then the complete list of components for this system is as given in Table (A.1.1). For indexing purposes a unique number is assigned to each component.

Component Index	Component Name
1	V_s – Sinusoidal Voltage Source
2	N_1 – Electric Node
3	N_2 – Electric Node
4	N_3 – Electric Node
5	V_{gnd} – Analog Ground
6	D_1
7	C_1
8	DC Motor
9	Fixed-Axis Link
10	$CCML_\omega$ - Mechanical Velocity Connection between DC Motor and Link
11	$CCML_\tau$ - Mechanical Torque Connection between DC Motor and Link

Table A.1.1: System Components

A.2 Component Description in MIXEDMODELS

A.2.1 Component Parameters

It was observed that any component is characterized by a set of time-independent and time-varying quantities which affect the system response. The quantities that describe the static behaviour of a component are the time-independent quantities, and in MIXEDMODELS we name them as *parameters*. For example, let us consider a capacitor component, as shown in Figure (A.2.1).

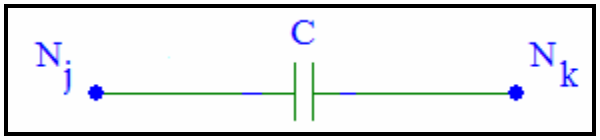


Figure A.2.1: Capacitor Model

To describe the static response of this capacitor component the only two things one needs to know are its value, and how it connects in the system. One of the possible ways to describe the parameter array for the capacitor component can be as follows.

$$\left[C_{ID}, N_j, N_k, C_{value} \right]^T \equiv \mathbf{p}^i \tag{A.2.1}$$

Here, i, represents the component index in the system, $\mathbf{p}^i$, is the parameter vector for component i, C_{ID}, is the component ID, N_j and N_k are the electric nodes which describe how the capacitor component connects to the rest of the system and C_{value} refers to the numerical value of the capacitance. Then for capacitor C_1, which is the 7^{th} component of the system shown in Figure (A.1.1), the parameter vector would be written as,

$$\left[C_1, N_2, N_3, 0.00045 \right]^T \equiv \mathbf{p}^7 \tag{A.2.2}$$

Similarly, the static behavior of all other components can be described in terms of their own parameters.

A.2.2 Component Variables

To describe the transient behavior of a component, we need some state variables such that when their values at time t are known, their values at some other time t' can be uniquely predicted. These time-varying quantities of a component describe the transient behavior, and thus these quantities are termed component *variables*. For example, in the system shown in Figure (A.1.1) the voltage potential of electric nodes vary at every time instant, and so the "node" component can have a "voltage" variable associated with it. Another example would be the same capacitor component discussed earlier. The variables that completely describe its transient behavior are V_c, the voltage across the capacitor, and I_c, the current through the capacitor. Similarly, every component will have its own set of variables that eventually contributes to the transient response of the system. We anticipate that some variables may only appear in the governing equations algebraically, whereas some other variables may appear in the governing equations in terms of their derivatives. We can divide the state variables into two categories based on this distinction: if a state variable appears in

the governing equations algebraically (i.e., without its derivatives), then the variable is called an *algebraic variable*; if a state variable's derivatives appear in the governing equations, then the variable is called a *differential variable*.

A.2.3 Component Governing Equations

Once there exists a general way to describe the static and transient behavior of a component, the obvious next step would be to formalize a way to represent governing equations of a component. Referring to Figure (A.1.1), for the capacitor component connected between nodes N_2 and N_3, let us analyze its governing equations. Let V_2 and V_3 be the variables for nodes N_2 and N_3, respectively, where, V_2, is the voltage at node N_2 and V_3 is the voltage at node N_3. Let V_c be the voltage across the capacitor, and let I_c be the current through the capacitor. Then the governing equations of the capacitor component, C_1, can be written as follows:

$$\begin{aligned} V_2 - V_3 &= V_c \\ C_{value}\dot{V}_c - I_c &= 0 \end{aligned} \tag{A.2.3}$$

Next, let us analyze the governing equation of the node component N_2. As described earlier, its governing equation is the KCL at that node. From Figure (A.1.1) it can be seen that this node connects three components with each other, the diode, the capacitor and the DC motor, which means that its equation is going to be written in terms of current variables of these three components, and not in terms of its own variables. If I_d, is the current through the diode and if I_a, is the armature current of the DC motor, then the governing equation of this component can be given by Equation (A.2.4).

$$I_d - I_c - I_a = 0 \tag{A.2.4}$$

From Equations (A.2.3) and (A.2.4) following observations can be made.

- The governing equations of the capacitor component are expressed in terms of its own parameters and variables.
- The governing equations of the capacitor component also include variables of other components - in this case, the electric nodes.

- The governing equation of the node component does not include its own variable, but is completely written in terms of variables of other components.
- One of the equations of the capacitor component is a first-order differential equation and the other is an algebraic equation. Further it can be said that Equation (A.2.3) represents a system of differential-algebraic equations (DAEs). On the other hand, the node component is described by a purely algebraic equation.
- One of the variables of the capacitor component, V_c, appears in differential form, where as the other variable, I_c, appears purely in algebraic form.

From the above observations, it is clear that to model a component of a multidisciplinary system, the mathematical formulation should be able to handle first-order ODEs and algebraic equations expressed using two types of variables, algebraic and differential. Further, it should accommodate for the case where a component introduces its own variables and equations, as well as the case where a component's behavior can be expressed in terms of variables of other components. If each component in the system adheres to the above conditions, then a mathematical formulation can be derived where the equations of all the components can be combined to form a system of governing equations.

This discussion raises a question – "Can we assume that the behavior of the entire system can always be described by a set of DAEs such that, i) only first-order derivatives of the state variables appear in the governing equations; and ii) the algebraic variables and derivatives of the differential variables appear linearly in the governing equations"? We define the form of a system of DAEs that satisfies the two conditions above as the linear first order form. It should be noted that in the linear first-order form, the differential variables themselves are allowed to appear nonlinearly in the governing equations – it is only required that their derivatives appear linearly. Thus, a system of DAEs that is in linear first-order form is not necessarily a set of linear DAEs, and may contain differential equations that are nonlinear ODEs.

On further investigation, it can be seen that before we can answer the above question in the affirmative, the following two issues need to be addressed:

- Components whose behavior is described using higher-order differential equations
- Components whose behavior is described by nonlinear algebraic equations

Multidisciplinary systems often consist of components which are governed by higher-order differential equations. As an example, consider a DC motor component connected between two nodes N_j and N_k, which is described by the following governing equations.

$$\begin{aligned} \ddot{\theta}_m J_m + \dot{\theta}_m B_v &= -T_D + I_a K_T \\ \dot{\theta}_m K_b - \left(V_j - V_k\right) &= -\dot{I}_a L_a - I_a R_a \end{aligned} \tag{A.2.5}$$

The first equation is a second-order differential equation which describes the relationship between torque and speed of the DC motor, and the second equation is a first-order differential equation which describes the voltage balance, that is the relation between armature voltage and back e.m.f. of the DC motor. Note that, the armature voltage is given by the voltage difference between the two nodes of the DC motor. There are components which are described by even higher order differential equations. This issue can be handled by converting an n^{th} order differential equation to n first-order differential equations by introducing (n-1) new variables, and hence (n-1) new equations. The DC motor component therefore can be described as follows.

Let ω_m be the new variable introduced which represents the angular velocity of the DC motor, which is given by the first derivative of the angular rotation, θ_m, of the DC motor. The component governing equations can now be written as,

$$\begin{aligned} \dot{\theta}_m &= \omega_m \\ \dot{\omega}_m J_m + T_D &= -\omega_m B_v + I_a K_T \\ \dot{I}_a L_a - \left(V_j - V_k\right) &= -\omega_m K_b - I_a R_a \end{aligned} \tag{A.2.6}$$

It can be seen from Equation (A.2.6) that the DC motor component has three differential variables, θ_m, ω_m, and I_a, the armature current in the DC motor, and one algebraic variable, T_D, the driving torque of the DC motor. The differential variables now appear in the governing equations in the first-order form. It is clear that we can use this technique to convert any set of governing DAEs to first-order form. Thus to standardise the representation, a mathematical formulation which only allows DAEs of first-order is sufficient.

The second issue which we had to consider is the fact that the behavior of some systems may be described using nonlinear algebraic equations. For example, consider the diode component, D_1, connected between two electric nodes N_j and N_k. A simple diode model can be built as given in Figure (A.2.2). Let V_j be the voltage at the node N_j and V_k be the voltage at the node N_k. Let I_d and V_d be the two variables of the diode component, where, I_d, is the current in the diode, and V_d, is the diode voltage.

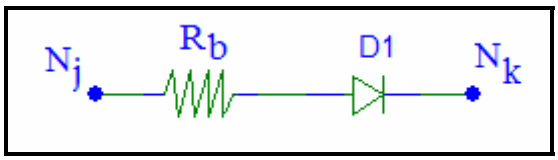


Figure A.2.2: Diode Model

Governing equations of the diode for the model given in Figure (A.2.2) are given by Equations (A.2.7) and (A.2.8).

$$I_d - I_s\left(e^{\frac{V_d}{\eta V_T}} - 1\right) = 0 \qquad \text{(A.2.7)}$$

$$\left(V_j - V_k\right) - R_b.I_d - V_d = 0 \qquad \text{(A.2.8)}$$

Here, R_b refers to the junction ohmic resistance, I_s refers to the saturation current, V_T refers to the thermal voltage and η refers to emission coefficient. All these are diode parameters. Notice that Equation (A.2.7) is nonlinear in terms of its algebraic variables I_d and V_d. However, by differentiating Equation (A.2.7) with respect to time, we obtain Equation (A.2.9), which is linear in the first time-derivative of I_d and V_d.

$$\frac{d}{dt}\left(I_d - I_s(e^{\frac{V_d}{\eta V_T}} - 1) \right) = 0 \qquad (a)$$

$$\therefore \dot{I}_d - \dot{V}_d(I_s.e^{\frac{V_d}{\eta V_T}}) = 0 \qquad (b)$$

Rearranging Equation (A.2.7) we get

(A.2.9)

$$(I_s.e^{\frac{V_d}{\eta V_T}}) = I_d + I_s \qquad (c)$$

$\therefore$ Equation (b) can be rewritten as

$$\dot{I}_d - (I_s + I_d).\dot{V}_d = 0 \qquad (d)$$

Using Equation (A.2.9d) the governing equations for the diode component can be rewritten as,

$$\begin{aligned} \dot{I}_d - (I_s + I_d).\dot{V}_d &= 0 \\ \left(V_j - V_k\right) &= R_b.I_d + V_d \end{aligned} \qquad (A.2.10)$$

After differentiating the nonlinear diode equation it can be seen that Equation (A.2.10) now represents a system of DAEs, where the first equation is linear in the differential variables, where as, the second equation is a linear algebraic equation.

Thus, higher-order differential equations and nonlinear algebraic equations can be successfully converted to fit the form of linear first-order form of a system of DAEs. As an example, the complete set of DAEs for the system given in Figure (A.1.1) is given by Equation (A.2.11). The system can be completely described in terms of eight differential and eight algebraic variables given below.

Differential Variables

- V_c – voltage across the capacitor,
- θ_m, ω_m, I_a – angular rotation, angular velocity and armature current of the DC motor,
- θ_L, ω_L – angular rotation and angular velocity of the fixed-axis link,
- V_d, I_d – voltage and current.

Algebraic Variables

- I_v – source current,
- V_1 – voltage at node N_1,
- V_2 – voltage at node N_2,
- V_3 – voltage at node N_3,
- I_{gnd} – ground current
- I_c – current through the capacitor
- T_D – driving torque of the DC motor
- T_L – load torque

Set of DAEs

$$\begin{aligned}
V_1 - V_3 &= V_s \sin\left(2\pi f t\right) \\
I_v &= I_d \\
-I_c &= -I_d + I_a \\
I_v - I_c + I_{gnd} &= I_a \\
V_3 &= 0 \\
\dot{I}_d - \left(I_s + I_d\right).\dot{V}_d &= 0 \\
V_1 - V_2 &= R_b.I_d + V_d \\
V_2 - V_3 &= V_c \\
C_{value}.\dot{V}_c - I_c &= 0 \\
\dot{\theta}_m &= \omega_m \\
J_m.\dot{\omega}_m + T_D &= -B_v.\omega_m + K_T.I_a \\
L_a.\dot{I}_a - V_2 + V_3 &= -K_b.\omega_m - R_a.I_a \\
\dot{\theta}_L &= \omega_L \\
J_L.\dot{\omega}_L - T_L &= 0 \\
\dot{\omega}_m - \dot{\omega}_L &= 0 \\
T_D - T_L &= 0
\end{aligned} \tag{A.2.11}$$

At this point, it would be appropriate to develop a general representation for multidisciplinary systems.

A.3 System Description in MIXEDMODELS

From Equation (A.2.11) it can be seen that the governing equations of all the components can be combined together to generate a system of equations that can be represented conveniently in a matrix form, as shown below by Equation (A.3.1).

$$\tilde{\mathbf{A}}\left(\mathbf{P},\mathbf{X},\mathbf{d},t\right)\begin{bmatrix}\dot{\mathbf{X}}\\ \mathbf{Y}\end{bmatrix}=\tilde{\mathbf{b}}\left(\mathbf{P},\mathbf{X},\mathbf{d},t\right) \tag{A.3.1}$$

Here, **X**, represents the vector of differential variables, **Y**, represents the vector of algebraic variables, **d**, represents the vector of design variables and, **P**, represents the vector of system parameters. $\tilde{\mathbf{A}}$ is the system coefficient matrix, and $\tilde{\mathbf{b}}$ is the right hand vector of the system. The matrix form given in Equation (A.3.1) encompasses most multidisciplinary systems and is easy to work with. Equation (A.2.11) can be represented in the matrix form as follows:

$$\begin{bmatrix}
0 & 1 & 0 & -1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -1 & 0 & 0 & 0 & 0 & 0 & 0 \\
1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & -1 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & -\left(\frac{I_s}{I_d}+\right) & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 1 & -1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 1 & -1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & -1 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & J_m & 0 & 0 & 1 & 0 & 0 & 0 \\
0 & 0 & -1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & L_a & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & J_L & -1 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & -1 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & -1
\end{bmatrix}
\begin{bmatrix} I_v \\ V_1 \\ V_2 \\ V_3 \\ I_{gnd} \\ \dot{V}_d \\ \dot{I}_d \\ \dot{V}_c \\ I_c \\ \dot{\theta}_m \\ \dot{\omega}_m \\ \dot{I}_a \\ T_D \\ \dot{\theta}_L \\ \dot{\omega}_L \\ T_L \end{bmatrix}
=
\begin{bmatrix} V_s.\sin(2\pi ft) \\ I_d \\ -I_d+I_a \\ I_a \\ 0 \\ 0 \\ R_b.I_d+V_d \\ V_c \\ 0 \\ \omega_m \\ -B_v.\omega_m+K_T.I_a \\ -K_b.\omega_m-R_a.I_a \\ \omega_L \\ 0 \\ 0 \\ 0 \end{bmatrix} \tag{A.3.2}$$

With this form, all we need is to provide is the matrix $\tilde{\mathbf{A}}$ and the right hand side $\tilde{\mathbf{b}}$ and then if we know the state at any time, *t*, then the system solution at some time, *t'*, can be obtained by integrating the differential variables over the interval from *t* to *t'*, using a suitable ODE solver to obtain the values of differential variables; whenever

the ODE solver requires the derivative values, we can solve the linear system above for the derivatives and algebraic variables using a suitable linear matrix solver.

Before we get to the solution process, however, the immediate question that needs to be answered is – "*How do we automatically generate the complete system of equations using components*"?

Recall that every component is defined in isolation in terms its parameters, variables and governing DAEs. From the matrix form presented in Equation (A.3.2) it can be seen that one of the possible ways to describe a component would be in terms of its contributions to the $\tilde{\mathbf{A}}$ matrix and the $\tilde{\mathbf{b}}$ vector. One should be able to represent the component information in a way such that it can be automatically applied to any kind of system to form the $\tilde{\mathbf{A}}$ matrix and the $\tilde{\mathbf{b}}$ vector for that system. In short, there should be a general scheme to formulate system matrices for any multidisciplinary system, and to support the global system structure there needs to be a compatible formulation strategy at the component level. The need for a general scheme to automatically formulate the system DAEs also calls for a symbolic-numeric architecture, where the symbolic code will formulate the system equations and the numeric code will solve the system to generate the system response.

Based on the discussion on component equations and the matrix in Equation (A.3.2) following observations can be made which will help in proceeding with deriving a general formulation scheme.

- The process of converting higher-order ODEs to first-order ODEs introduces new differential variables and the first derivatives of these new variables can be calculated directly by assignments. These variables therefore can be removed from the matrix structure and can be calculated separately.
- In order to dimension the system matrices, we need to know the total number of components in a system.
- To populate system matrices, the number and type of variables associated with each component and number of differential and algebraic equations that each component contributes to the system should be known.

- There should be a certain structure which will define how variables and equations will be arranged in the matrix form. This will not only help in automating the process of populating the matrices directly from the component data, but will also simplify the solution process.
- To provide reusability and modularity, each component should be maintained separately in independent files in a library of components and the controlling program should be able to execute the component files of interest on the fly. This also means that the components will be defined locally, i.e. in terms of local variables and indices. Thus, there also needs to be a scheme that maps the local component information to the global system information.

A.4 Mathematical Formulation of MIXEDMODELS

From the observation that the lower-order variables can be removed from the matrix structure, the vector of differential variables, $\mathbf{X}$ can be partitioned as

$$\mathbf{X} = \begin{bmatrix} \mathbf{X_H} \\ \mathbf{X_L} \end{bmatrix} \tag{A.4.1}$$

where $\mathbf{X_H}$ represents the subvector of $\mathbf{X}$ such that the derivatives of the elements of $\mathbf{X_H}$ are not contained in $\mathbf{X}$. Similarly, $\mathbf{X_L}$ represents the subvector of $\mathbf{X}$ containing all the elements of $\mathbf{X}$ such that their derivative is also an element of $\mathbf{X}$, i.e., the lower-order variables. Thus, the lower-order variables can be calculated by a set of direct assignments of the form

$$\dot{\mathbf{X}}_\mathbf{L} = \mathbf{X}' \tag{A.4.2}$$

where $\mathbf{X}'$ is a sub-vector of $\mathbf{X}$. Separating Equation (A.4.2) from the system of equations given by Equation (A.3.1), the rest of the system governing equations can be written as

$$\mathbf{A}\left(\mathbf{P}, \mathbf{X}, \mathbf{d}, t\right)\begin{bmatrix} \dot{\mathbf{X}}_\mathbf{H} \\ \mathbf{Y} \end{bmatrix} = \mathbf{b}\left(\mathbf{P}, \mathbf{X}, \mathbf{d}, t\right) \tag{A.4.3}$$

In order to support the above formulation at the system level, we require the following at the component level:

- A vector of time-invariant component parameters, $\mathbf{p}^i$; ∋ $\mathbf{p}^i \in \mathbf{P}$, where **P** forms the system parameter vector **P**.
- A vector of transient component differential variables, $\mathbf{x}^i$ ∋ $\mathbf{x}^i \in \mathbf{X}$, where **X** is the system differential variable vector. The $\mathbf{x}^i$ occur in the governing DAEs in first derivative form $\dot{\mathbf{x}}^i$.
- A vector of component algebraic variables, $\mathbf{y}^i$, which occur algebraically in the DAEs ∋ $\mathbf{y}^i \in \mathbf{Y}$, where **Y** forms the system algebraic variable vector **Y**.
- A set of component governing equations which can be expressed by

$$\mathbf{A}^c\left(\mathbf{P},\mathbf{X},\mathbf{d},t\right)\begin{bmatrix}\dot{\mathbf{X}}_{\mathbf{H}}\\ \mathbf{Y}\end{bmatrix}=\mathbf{b}^c\left(\mathbf{P},\mathbf{X},\mathbf{d},t\right) \tag{A.4.4}$$

- Direct assignments for lower order variables at component level can be written as

$$\dot{\mathbf{X}}_{\mathbf{L}}^c=\mathbf{X}^{c'} \tag{A.4.5}$$

- Modification matrices $\mathbf{A}^m$ and $\mathbf{b}^m$ which allow a component to modify governing equations of other components. Modification matrices can be zero if a component does not modify the governing equations of any other component.

Further, it should be recalled that a component can introduce new variables to the system or it can be described in terms of variables of other components. Similarly, it may contribute new equations to the system and/or modify equations contributed by other components. Finally, assuming that the **A** matrix and the **b** vector are initialized to zero, the contribution of a particular component *i* to the system governing equations can be written in the form,

$$\begin{aligned}\mathbf{A}\left(\mathbf{P},\mathbf{X},\mathbf{d},t\right)&=\mathbf{A}\left(\mathbf{P},\mathbf{X},\mathbf{d},t\right)+\mathbf{A}_i^m\left(\mathbf{P},\mathbf{X},\mathbf{d},t\right)+\mathbf{A}_i^c\left(\mathbf{P},\mathbf{X},\mathbf{d},t\right)\\ \mathbf{b}\left(\mathbf{P},\mathbf{X},\mathbf{d},t\right)&=\mathbf{b}\left(\mathbf{P},\mathbf{X},\mathbf{d},t\right)+\mathbf{b}_i^m\left(\mathbf{P},\mathbf{X},\mathbf{d},t\right)+\mathbf{b}_i^c\left(\mathbf{P},\mathbf{X},\mathbf{d},t\right)\end{aligned} \tag{A.4.6}$$

where

A (**P**,**X**,**d**,*t*) is the global matrix in the governing equations of Equation (A.4.3),

$\mathbf{A}^{c}_{i}$ (**P**,**X**,**d**,*t*) is the contribution of component *i* to the global matrix via new equations contributed by this component,

$\mathbf{A}^{m}_{i}$ (**P**,**X**,**d**,*t*) is the contribution of component *i* to the global matrix via modifications caused by this component to equations contributed by other components,

$\mathbf{b}^{c}_{i}$ (**P**,**X**,**d**,*t*) is the contribution of component *i* to the global RHS vector via new equations contributed by this component,

$\mathbf{b}^{m}_{i}$ (**P**,**X**,**d**,*t*) is the contribution of component *i* to the global RHS vector via modifications caused by this component to equations contributed by other components,

P is a vector of time-independent parameters that describe each component in the system,

X is the vector of system differentiable variables, and

d is the vector of design variables.

$\mathbf{A}^{c}$ and $\mathbf{A}^{m}$ matrices and $\mathbf{b}^{c}$ and $\mathbf{b}^{m}$ vectors are dimensioned to be of full size as **A** and **b**, and are initialized to zero. Thus, a component model is completely described by specifying the entries in the $\mathbf{A}^{c}$ and $\mathbf{A}^{m}$ matrices and in the $\mathbf{b}^{c}$ and $\mathbf{b}^{m}$ vectors contributed by that component. For convenience in automation of the process, the variables and equations in the matrix equation are ordered in a certain way. Matrix rows correspond to component equations, and the equations are entered in the matrix following the order in which a component occurs in the data file. Matrix columns correspond to the system variables, where the differential variables appear prior to the algebraic variables. A detailed discussion on the architectural aspects can be found in Chapter 5.

For the system shown in Figure (A.1.1) the system of equations can be rewritten as follows. The direct assignment equations are as given in Equation (A.4.7).

$$\begin{aligned} \dot{\theta}_m &= \omega_m \\ \dot{\theta}_L &= \omega_L \end{aligned} \tag{A.4.7}$$

$$\begin{bmatrix} 0 & 1 & 0 & -1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -1 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & -1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & -\begin{pmatrix} I_s + \\ I_d \end{pmatrix} & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & -1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & -1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & C_{value} & -1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & J_m & 0 & 1 & 0 & 0 \\ 0 & 0 & -1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & L_a & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & J_L & -1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & -1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & -1 \end{bmatrix} \begin{bmatrix} I_v \\ V_1 \\ V_2 \\ V_3 \\ I_{gnd} \\ V_d \\ \dot{I}_d \\ V_c \\ \dot{I}_c \\ \dot{\omega}_m \\ \dot{I}_a \\ T_D \\ \dot{\omega}_L \\ T_L \end{bmatrix} = \begin{bmatrix} V_s.\sin(2\pi ft) \\ I_d \\ -I_d + I_a \\ I_a \\ 0 \\ 0 \\ R_b.I_d + V_d \\ V_c \\ 0 \\ -B_v.\omega_m + K_T.I_a \\ -K_b.\omega_m - R_a.I_a \\ 0 \\ 0 \\ 0 \end{bmatrix} \qquad \text{(A.4.7)}$$

Note that Equation (A.4.8) does not have variables ordered according to Equation (A.4.3). To re-arrange the matrix equation in the final order let us look at the data file for this system as given by Figure (A.4.1). As discussed in Chapter (5), for convenience in modeling and automatically populating matrices from the component data, the component parameters are divided into three arrays – string-valued parameter array, integer-valued parameter array and real-valued parameter array. First five components of the system including a sinusoidal voltage source, three node components and one ground component are all described by equations that are purely algebraic. The sinusoidal voltage source is connected between two node components N_j and N_k, and has a current variable as its algebraic variable. The equation of such a sinusoidal voltage source is give by,

$$V_j - V_k = V_{offset} + V_{amplitude}.e^{\left(\theta.(t-t_d)\right)}.\sin\left(2.\pi.f.\left(t-t_d\right)+\left(\varphi/360\right)\right) \qquad \text{(A.4.9)}$$

Here, f, is the frequency in Hz, ϕ, is the phase in degrees, θ is the damping factor per second, t_d, is the time delay in seconds, V_{offset}, is the dc offset voltage and $V_{amplitude}$, is the amplitude of the sinusoidal voltage source.

```
PS[1, 1] := "VSineSource":      PR[6,1] := 1.836:          # eta      PS[9, 1] := "FixedAxisBody":
PS[1, 2] := "Vsine1":           PR[6,2] := 0.026:          # Vt       PS[9, 2] := "FAB1":
PS[1, 3] := "N1":               PR[6,3] := 2.682e-9:       # Is       PR[9, 1] := 41.0767:
PS[1, 4] := "N3":               PR[6,4] := 100:            # BR
PR[1, 1] := 5:                  PR[6,5] := 4.5:            # m        PS[10,1] := "ConnectionConstraint":
PR[1, 2] := 0:                  PR[6,6] := 1/3:            # Md       PS[10,2] := "CCMLw":
PR[1, 3] := 50:                 PR[6,7] := 11.54e-9:       # Tt       PS[10,3] := "DCM1":
PR[1, 4] := 0:                  PR[6,8] := 4e-12:          # Cj0      PS[10,4] := "FAB1":
PR[1, 5] := 0:                  PR[6,9] := 0.5:            # phi0     PI[10,1] := 1:
PR[1, 6] := 0:                  PR[6,10] := 0.5664:        # Rb       PI[10,2] := 0:
                                                                       PI[10,3] := 1:
PS[2, 1] := "ENode":            PS[7, 1] := "Capacitor":               PI[10,4] := 0:
PS[2, 2] := "N1":               PS[7, 2] := "C1":
                                PS[7, 3] := "N2":                      PS[11,1] := "ConnectionConstraint":
PS[3, 1] := "ENode":            PS[7, 4] := "N3":                      PS[11,2] := "CCMLt":
PS[3, 2] := "N2":               PR[7, 1] := 4.1666666666667e-2:        PS[11,3] := "DCM1":
                                                                       PS[11,4] := "FAB1":
PS[4, 1] := "ENode":            PS[8, 1] := "DCMotorVer3":             PI[11,1] := 0:
PS[4, 2] := "N3":               PS[8, 2] := "DCM1":                    PI[11,2] := 1:
                                PS[8, 3] := "N2":                      PI[11,3] := 0:
PS[5, 1] := "Vgnd":             PS[8, 4] := "N3":                      PI[11,4] := 1:
PS[5, 2] := "Vgnd":             PR[8, 1] := 0.0044:
PS[5, 3] := "N3":               PR[8, 2] := 2.4:
PR[5,1] := 0:                   PR[8, 3] := 0.0048:
                                PR[8, 4] := 0.0401:
PS[6,1] := "pnDiodeSimple":     PR[8, 5] := 5.6:
PS[6,2] := "D1":                PR[8, 6] := 0.0137:
PS[6,3] := "N1":
PS[6,4] := "N2":
```

Figure A.4.1: Data File for the System in Figure (A.1.1)

The matrix equation for these five components can be written as follows.

$$\begin{bmatrix} 0&0&0&0&0&0&0&1&0&-1&0&0&0&0 \\ 0&0&0&0&0&0&1&0&0&0&0&0&0&0 \\ 0&0&0&0&0&0&0&0&0&0&0&0&0&0 \\ 0&0&0&0&0&0&1&0&0&0&-1&0&0&0 \\ 0&0&0&0&0&0&0&0&0&1&0&0&0&0 \\ 0&0&0&0&0&0&0&0&0&0&0&0&0&0 \\ 0&0&0&0&0&0&0&0&0&0&0&0&0&0 \\ 0&0&0&0&0&0&0&0&0&0&0&0&0&0 \\ 0&0&0&0&0&0&0&0&0&0&0&0&0&0 \\ 0&0&0&0&0&0&0&0&0&0&0&0&0&0 \\ 0&0&0&0&0&0&0&0&0&0&0&0&0&0 \\ 0&0&0&0&0&0&0&0&0&0&0&0&0&0 \\ 0&0&0&0&0&0&0&0&0&0&0&0&0&0 \\ 0&0&0&0&0&0&0&0&0&0&0&0&0&0 \end{bmatrix} \begin{bmatrix} \dot{V}_d \\ \dot{I}_d \\ \dot{V}_c \\ \dot{\omega}_m \\ \dot{I}_a \\ \dot{\omega}_L \\ \dot{I}_v \\ \dot{V}_1 \\ \dot{V}_2 \\ \dot{V}_3 \\ \dot{I}_{gnd} \\ \dot{I}_c \\ \dot{T}_D \\ \dot{T}_L \end{bmatrix} = \begin{bmatrix} V_s.\sin(2\pi ft) \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix} \qquad \text{(A.4.10)}$$

In the above equation, the variables and equations are properly ordered, with differential variables preceding algebraic variables. Note that the matrix and the right hand side are of full system dimensions, and represent the system, **A** matrix and **b** vector at this stage of the derivation of the governing equations. The next component that appears in the data file is the diode component which not only contributes new equations to the system, but also modifies equations of the node components N_1 and N_2. Referring to the diode equations in Equation (A.2.11), the component matrices, $\mathbf{A}_c$ and $\mathbf{b}_c$, and the modification matrices, $\mathbf{A}_m$ and $\mathbf{b}_m$, for the diode component can be given as follows.

$$\mathbf{A}_c = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ -\left(I_s + I_d\right) & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & -1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}, \quad \mathbf{b}_c = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ R_b I_d + V_d \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix} \tag{A.4.11}$$

$$\mathbf{A}_m = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}, \quad \mathbf{b}_m = \begin{bmatrix} 0 \\ I_d \\ -I_d \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix} \tag{A.4.12}$$

Notice that the $\mathbf{A}_m$ matrix here is a null matrix. The diode component is the 6^{th} component in the data file. Component files are executed at runtime in the same

order in which the components appear in the data file. On execution of this component, Equation (A.4.10) will be automatically updated using Equation (A.4.6) to give the following.

$$\begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & -1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & -1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ -\begin{pmatrix} I_s + \\ I_d \end{pmatrix} & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & -1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} \dot{V}_d \\ \dot{I}_d \\ \dot{V}_c \\ \dot{\omega}_m \\ \dot{I}_a \\ \dot{\omega}_L \\ I_v \\ V_1 \\ V_2 \\ V_3 \\ I_{gnd} \\ I_c \\ T_D \\ T_L \end{bmatrix} = \begin{bmatrix} V_s.\sin(2\pi f t) \\ I_d \\ -I_d \\ 0 \\ 0 \\ 0 \\ R_b.I_d + V_d \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix} \tag{A.4.13}$$

Proceeding with the data file the rest of the component files are executed in the same manner, and after adding contributions from all these components the system matrices will be finally given by Equation (A.4.14). The direct assignment equations remain unchanged, and are given by Equation (A.4.8).

$$\begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & -1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & -1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ -\begin{pmatrix} I_s + \\ I_d \end{pmatrix} & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & -1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & -1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & C_{value} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -1 & 0 & 0 \\ 0 & 0 & 0 & J_m & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & -1 & 1 & L_a & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & J_L & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -1 \\ 0 & 0 & 0 & 1 & 0 & -1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & -1 \end{bmatrix} \begin{bmatrix} \dot{V}_d \\ \dot{I}_d \\ \dot{V}_c \\ \dot{\omega}_m \\ \dot{I}_a \\ \dot{\omega}_L \\ I_v \\ V_1 \\ V_2 \\ V_3 \\ I_{gnd} \\ I_c \\ T_D \\ T_L \end{bmatrix} = \begin{bmatrix} V_s.\sin(2\pi f t) \\ I_d \\ -I_d + I_a \\ I_a \\ 0 \\ 0 \\ R_b.I_d + V_d \\ V_c \\ 0 \\ -B_v.\omega_m + K_T.I_a \\ -K_b.\omega_m - R_a.I_a \\ 0 \\ 0 \\ 0 \end{bmatrix} \tag{A.4.14}$$

Once again, let us consider the differential and algebraic variables of this system and represent them in the form of a vector of differential variables, **X**, and a vector of algebraic variables, **Y**, as follows.

$$\mathbf{X} = \begin{bmatrix} V_d & I_d & V_c & \omega_m & I_a & \omega_L & \theta_m & \theta_L \end{bmatrix}^{\mathrm{T}}$$

$$\text{where, } \mathbf{X_H} = \begin{bmatrix} V_d & I_d & V_c & \omega_m & I_a & \omega_L \end{bmatrix}^{\mathrm{T}},\ \mathbf{X_L} = \begin{bmatrix} \theta_m & \theta_L \end{bmatrix}^{\mathrm{T}} \tag{A.4.15}$$

$$\mathbf{Y} = \begin{bmatrix} I_v & V_1 & V_2 & V_3 & I_{gnd} & I_c & T_D & T_L \end{bmatrix}^{\mathrm{T}}$$

Rewriting Equations (A.4.14) and (A.4.8) in X and Y we get the complete system of DAEs as follows.

$$\begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & -1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & -1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ -\begin{pmatrix} I_s + \\ \dot{X}_{H_2} \end{pmatrix} & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & -1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & -1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & C_{value} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -1 & 0 & 0 \\ 0 & 0 & 0 & J_m & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & -1 & 1 & L_a & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & J_L & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -1 \\ 0 & 0 & 0 & 1 & 0 & -1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & -1 \end{bmatrix} \begin{bmatrix} \dot{X}_{H_1} \\ \dot{X}_{H_2} \\ \dot{X}_{H_3} \\ \dot{X}_{H_4} \\ \dot{X}_{H_5} \\ \dot{X}_{H_6} \\ Y_1 \\ Y_2 \\ Y_3 \\ Y_4 \\ Y_5 \\ Y_6 \\ Y_7 \\ Y_8 \end{bmatrix} = \begin{bmatrix} V_s.\sin(2\pi ft) \\ X_{H_2} \\ -X_{H_2} + X_{H_5} \\ X_{H_5} \\ 0 \\ 0 \\ R_b.X_{H_2} + X_{H_1} \\ X_{H_3} \\ 0 \\ -B_v.X_{H_4} + K_T.X_{H_5} \\ -K_b.X_{H_4} - R_a.X_{H_5} \\ 0 \\ 0 \\ 0 \end{bmatrix} \tag{A.4.16}$$

$$\begin{aligned} \dot{X}_{L_1} &= X_{H_4} \\ \dot{X}_{L_2} &= X_{H_6} \end{aligned} \tag{A.4.17}$$

From the discussion presented so far, it can be seen that, the matrix structure for each component is sparse. If the contribution of each component to the system matrices needs to be expressed in terms of full-size matrices, then it will require massive memory storage and execution will slow down as well. For this reason, this formulation is implemented differently, where each component is defined in isolation in terms of its local variables and equations, and offsets are maintained which map this local information globally to the system matrices. Thus, only the non-zero elements of $\mathbf{A}_m$, $\mathbf{A}_c$, $\mathbf{b}_m$ and $\mathbf{b}_c$ are specified in the component models, and only these nonzero values are maintained in the program at runtime. Information for each component is stored in two separate files. One of the files, called the header file,

maintains information about the number and type of variables and number of equations a component contributes to the system. The second file contains information about the governing equations of the components, expressed in terms of offsets and local variables. The controlling program maintains the offsets, so the component specification code does not have to do any management of offsets. The controlling program uses a two-pass procedure (as explained in Chapter 5), wherein the first pass only executes the header files of all the components in the order in which they appear, and calculated the offsets required for local-to-global mapping. This process is explained in detail with the help of an example in Chapter 5.

Once the system of equations is formed, the next step is to solve the system to generate the system response.

A.5 Solution Process of MIXEDMODELS

The solution process involves calculating the algebraic and differential variables for any time *t*, and integrating the differential equations to generate the system response over the simulation time period. From Equation (A.4.2) one can directly obtain the values for derivatives of the lower-order variables $\dot{\mathbf{X}}_\mathbf{L}$. The linear matrix solver then solves the system given by Equation (A.4.3) to obtain $\left[\dot{\mathbf{X}}_\mathbf{H} \;\; \mathbf{Y}\right]^T$. Then, the ODE solver integrates $\left[\dot{\mathbf{X}}_\mathbf{H} \;\; \dot{\mathbf{X}}_\mathbf{L}\right]^T$ to obtain the differential variables. The current implementation uses a stiff integrator DLSODES based on a sparse matrix version of Gear's algorithm and a linear sparse matrix solver Y12MAF. The fact that the formulation generates explicit mathematical equations makes this scheme flexible for the user to choose between different numerical solvers or even to write new solvers. It also allows extension of this architecture for parametric studies such as sensitivity analysis which will be discussed in detail in the following sections. The user needs to provide information about simulation parameters, initial conditions and error tolerances for the numerical solvers.

A.6 Calculating Initial Conditions

In order to start the integration from the given initial time, initial conditions must be provided for all of the differential variables, **X**, that we wish to determine. In

specifying initial conditions, care must be taken to ensure that the specified initial conditions are physically realizable and consistent with all system constraints, such as loop closure conditions on a mechanical system. It may be difficult for the user to manually supply a consistent set of initial conditions for **X** for a large system. To ensure consistency of initial conditions, we assume that the user provides a set of consistency equations that must be satisfied by the initial conditions. These equations are assumed to be of the form

$$\mathbf{h}(\mathbf{P}, \mathbf{X}(\mathbf{d}, t), \mathbf{d}, t)|_{t=0} = \mathbf{0} \tag{A.6.1}$$

where **h** is a vector of nonlinear algebraic equations of dimension $(N+L)$ that can be solved by Newton-Raphson iteration to obtain a consistent initial conditions vector, **X**. Here, N is the number of higher-order differential variables and L is the number of lower-order differential variables, while **d** is the vector of design variables for the system. All that the user needs to provide then is the set of initial guesses for **X**, which are then corrected by iteration to satisfy Equation (A.4.15).

Sections (A.1) through (A.6) presented the development process for the mathematical formulation for analysis of multidisciplinary systems in the MIXEDMODELS platform. To test the working of the formulation-solution scheme presented here, several systems were modeled using this approach and the time responses for these systems were verified using MATLAB, Simulink and PSpice. In general it worked well but also raised some issues which led us to believe that the mathematical formulation based on the linear first-order form assumption on the DAEs needs to be improved.

A.7 Nonlinear Formulation for Analysis of Multidisciplinary Systems

The formulation based on linear first-order form works well when the system is linear. However, it shares a drawback with linear state-space methods when it comes to dealing with a general multidisciplinary system whose component-governing equations may include a large number of nonlinear algebraic or differential equations. This is often the case when a wide range of component types are involved. The linear state space formulation and the formulation presented in Section (A.4)

require that these equations be converted to a pre-specified form by a process of differentiation. This has the undesirable effect of artificially increasing the number of differential equations in the system, and generally making the numerical solution more difficult. We occasionally have found that a non-stiff system may appear numerically stiff in the differentiated form. This formulation may also suffer from constraint drift if applied to large systems over long time periods, i.e., even though the differentiated nonlinear algebraic equations are satisfied, after some time the original nonlinear algebraic equation may not be satisfied, due to numerical errors in the simulation. The most popular method to account for this drift is Baumgarte's stabilization technique, which can be incorporated into this formulation. For example, this technique has been used to develop a mathematical model for a "stabilized revolute joint" to avoid constraint drift. Still, this is a serious difficulty in converting nonlinear equations to the linear first order form.

To avoid these numerical difficulties caused by the differentiation, a general nonlinear formulation for analysis and analytical design sensitivity analysis of multidisciplinary systems needs to be developed which will be discussed in the subsequent paragraphs.

The main difference in this formulation is in the representation of Equation (A.4.3). Recall that the modeling approach adopted in the MIXEDMODELS platform considers a multidisciplinary system to be a collection of interacting components, which can be completely described by a vector of time-invariant system parameters, **P**; a vector of system variables, **X**, which can also occur in the first derivative form $\dot{\mathbf{X}}$ in the DAEs; a vector of algebraic variables, **Y,** that can occur algebraically in the set of DAEs; and a set of governing DAEs. The DAEs can be represented by the linear first-order form given by Equation (A.4.3), however, as mentioned in the previous section this formulation suffers from drawbacks due to differentiation that is involved in converting the nonlinear equations to fit to the linear form. Therefore it is preferable to describe the system using DAEs using a general nonlinear representation given by Equation (A.7.1).

$$\mathbf{F}(\mathbf{P}, \mathbf{X}(\mathbf{d}, t), \dot{\mathbf{X}}(\mathbf{d}, t), \mathbf{Y}(\mathbf{d}, t), \mathbf{d}, t) = \mathbf{0} \qquad \text{(A.7.1)}$$

where **d** is a vector of design variables.

This form does not require any additional differentiation of nonlinear equations and therefore does not require the equations to be generated in linear first-order form. The vector of differential variables, **X**, is partitioned as before and is given by Equation (A.4.1). Also note that the evaluations of first derivatives of the lower-order variables using direct assignment equations also remain unchanged and is therefore given by Equation (A.4.2), which is presented here again in Equation (A.7.2) to maintain continuity in reading.

$$\dot{\mathbf{X}}_{\mathbf{L}} = \mathbf{X}' \tag{A.7.2}$$

Separating Equation (A.7.2) from the system of equations given by Equation (A.7.1), the rest of the system governing equations can be written as

$$\mathbf{f}(\mathbf{P}, \mathbf{X}(\mathbf{d}, t), \dot{\mathbf{X}}_{\mathbf{H}}(\mathbf{d}, t), \mathbf{Y}(\mathbf{d}, t), \mathbf{d}, t) = \mathbf{0} \tag{A.7.3}$$

Equations (A.7.2) and (A.7.3) describe the complete system of DAEs. Note that, based on the above form of DAEs, appropriate changes should be made in component models. Therefore in order to support the above formulation at the system level, we require that a component, *i*, in an MDS should be described using the following:

- a vector of time-invariant component parameters, $\mathbf{p}^i$,
- a vector of transient component differential variables, $\mathbf{x}^i$ (these $\mathbf{x}^i$ occur in the governing DAEs in first derivative form $\dot{\mathbf{x}}^i$),
- a vector of component algebraic variables, $\mathbf{y}^i$, which occur algebraically in the DAEs,

In addition, the component model must provide the following,

- a set of component governing equations which can be expressed as

$$\mathbf{f}_i^c(\mathbf{P}, \mathbf{X}(\mathbf{d}, t), \dot{\mathbf{X}}_{\mathbf{H}}(\mathbf{d}, t), \mathbf{Y}(\mathbf{d}, t), t) = \mathbf{0} \tag{A.7.4}$$

- a set of direct assignments given by

$$\dot{\mathbf{X}}_{\mathbf{L}_i}^{c} = \mathbf{X}^{c'} \tag{A.7.5}$$

- and a set of component modification equations of the form

$$\mathbf{f}_i^m(\mathbf{P}, \mathbf{X}(\mathbf{d},t), \dot{\mathbf{X}}_{\mathbf{H}}(\mathbf{d},t), \mathbf{Y}(\mathbf{d},t), t) = \mathbf{0} \tag{A.7.6}$$

These equations describe the modifications that this component makes in the equations of other components. The contributions of all the components in the system are summed to obtain the system governing equations

$$\mathbf{f}(\mathbf{P}, \mathbf{X}, \dot{\mathbf{X}}_{\mathbf{H}}, \mathbf{Y}, t) = \sum_i \left[\mathbf{f}_i^c(\mathbf{P}, \mathbf{X}, \dot{\mathbf{X}}_{\mathbf{H}}, \mathbf{Y}, t) + \mathbf{f}_i^m(\mathbf{P}, \mathbf{X}, \dot{\mathbf{X}}_{\mathbf{H}}, \mathbf{Y}, t) \right] \tag{A.7.7}$$

Note that the vectors $\mathbf{f}_i^m$ and $\mathbf{f}_i^c$ are of full dimension of (N+M x 1) and are initialized to zero. Direct assignment contributions from each component are added to get Equation (A.7.2). Thus, we get the complete set of system DAEs given by

$$\begin{aligned} \mathbf{f}(\mathbf{P}, \mathbf{X}(\mathbf{d},t), \dot{\mathbf{X}}_{\mathbf{H}}(\mathbf{d},t), \mathbf{Y}(\mathbf{d},t), \mathbf{d}, t) &= \mathbf{0} \\ \dot{\mathbf{X}}_{\mathbf{L}}(\mathbf{d},t) &= \mathbf{X}'(\mathbf{d},t) \end{aligned} \tag{A.7.8}$$

Considering the same system given in Figure (A.1.1) the **f** vector for the first five components can be written as follows.

$$\mathbf{f} = \begin{bmatrix} V_1 - V_3 - V_s \sin(2\pi f t) \\ I_v \\ 0 \\ I_v + I_{gnd} \\ V_3 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix} = \begin{bmatrix} Y_2 - Y_4 - V_s \sin(2\pi f t) \\ Y_1 \\ 0 \\ Y_1 + Y_5 \\ Y_4 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix} \tag{A.7.9}$$

Notice that the node, N_2, connects to three components, the diode, the capacitor and the DC motor, which are 6th, 7th and 8th components respectively in the data file. An

electric node by itself does not contribute to its equation, but, the contributions from the components it connects to construct its governing equations. Thus, for the node component, N_2, the diode, the capacitor and the DC motor generate its governing equation, or the KCL at that node. Since these components are not executed yet by the programming controller, the entry at this time for this component in the **f** vector of Equation (A.7.9) is zero.

Proceeding with the data file let us consider the diode component in the system shown in Figure (A.1.1). The diode model for the diode component connected between the nodes N_1 and N_2 in Figure (A.1.1) is described using Equation (A.7.10).

$$\begin{aligned} I_d - I_s(e^{\frac{V_d}{\eta V_T}} - 1) &= 0 \\ (V_1 - V_2) - R_b.I_d - V_d &= 0 \end{aligned} \tag{A.7.10}$$

The diode is the 6th component of the system, which contributes two new variables and two new equations to the system. It also modifies the equations of the two node components it connects to. Also notice that since the diode equations are no longer differentiated with respect to t, the diode component therefore does not contribute any differential variables, but has two algebraic variables. Using this information $\mathbf{f}_6^m$ and $\mathbf{f}_6^c$ vectors for the diode component would be as given below.

$$\mathbf{f}_6^c = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ I_d - I_s(e^{\frac{V_d}{\eta V_T}} - 1) \\ (V_1 - V_2) - R_b.I_d - V_d \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ Y_7 - I_s(e^{\frac{Y_6}{\eta V_T}} - 1) \\ (Y_2 - Y_3) - R_b.Y_7 - Y_6 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix} \tag{A.7.11}$$

$$\mathbf{f}_6^m = \begin{bmatrix} 0 \\ -I_d \\ I_d \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix} = \begin{bmatrix} 0 \\ -Y_7 \\ Y_7 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix} \tag{A.7.12}$$

Using the system variables defined in Equation (A.7.13), and using Equations (A.7.7), (A.7.11) and (A.7.12) the **f** vector can be updated to give Equation (A.7.14). Notice that the modification by the diode component in the governing equation of the node N_2 can now be seen in its equation. Similarly, other components' contribution can be added to the system and the entire system of nonlinear DAEs can be generated as given by Equation (A.7.15).

$$\begin{aligned} \mathbf{X} &= \begin{bmatrix} V_c & \omega_m & I_a & \omega_L & \theta_m & \theta_L \end{bmatrix}^{\mathrm{T}} \\ \text{where, } \mathbf{X_H} &= \begin{bmatrix} V_c & \omega_m & I_a & \omega_L \end{bmatrix}^{\mathrm{T}}, \ \mathbf{X_L} = \begin{bmatrix} \theta_m & \theta_L \end{bmatrix}^{\mathrm{T}} \\ \mathbf{Y} &= \begin{bmatrix} I_v & V_1 & V_2 & V_3 & I_{gnd} & V_d & I_d & I_c & T_D & T_L \end{bmatrix}^{\mathrm{T}} \end{aligned} \tag{A.7.13}$$

$$\mathbf{f} = \begin{bmatrix} V_1 - V_3 - V_s \sin(2\pi f t) \\ I_v - I_d \\ I_d \\ I_v - I_{gnd} \\ V_3 \\ I_d - I_s(e^{\frac{V_d}{\eta V_T}} - 1) \\ (V_1 - V_2) - R_b.I_d - V_d \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix} = \begin{bmatrix} Y_2 - Y_4 - V_s \sin(2\pi f t) \\ Y_1 - Y_7 \\ Y_7 \\ Y_1 - Y_5 \\ Y_4 \\ Y_7 - I_s(e^{\frac{Y_6}{\eta V_T}} - 1) \\ (Y_2 - Y_3) - R_b.Y_7 - Y_6 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix} \tag{A.7.14}$$

Similarly, other components' contributions can be added to the system and the entire system of nonlinear DAEs can be generated as given by Equation (A.7.15).

$$\mathbf{f}=\begin{bmatrix} V_1-V_3-V_s\sin(2\pi ft) \\ I_v-I_d \\ I_d-I_c-I_a \\ I_v-I_c-I_a+I_{gnd} \\ V_3 \\ I_d-I_s(e^{\frac{V_d}{\eta V_T}}-1) \\ (V_1-V_2)-R_b.I_d-V_d \\ (V_2-V_3)-V_c \\ C_{value}.\dot{V}_c-I_c \\ J_m.\dot{\omega}_m+B_v.\omega_m-K_T.I_a+T_D \\ L_a.\dot{I}_a-(V_2-V_3)+K_b.\omega_m+R_a.I_a \\ J_L.\dot{\omega}_L-T_L \\ \dot{\omega}_m-\dot{\omega}_L \\ T_D-T_L \end{bmatrix}=\begin{bmatrix} Y_2-Y_4-V_s\sin(2\pi ft) \\ Y_1-Y_7 \\ Y_7-Y_8-X_{H_3} \\ Y_1-Y_8-X_{H_3}+Y_5 \\ Y_4 \\ Y_7-I_s(e^{\frac{Y_6}{\eta V_T}}-1) \\ (Y_2-Y_3)-R_b.Y_7-Y_6 \\ Y_3-Y_4-X_{H_1} \\ C_{value}.\dot{X}_{H_1}-Y_8 \\ J_m.\dot{X}_{H_2}+B_v.X_{H_2}-K_T.X_{H_3}+Y_9 \\ L_a.\dot{X}_{H_3}-(Y_3-Y_4)+K_b.X_{H_2}+R_a.X_{H_3} \\ J_L.\dot{X}_{H_4}-Y_{10} \\ \dot{X}_{H_2}-\dot{X}_{H_4} \\ Y_9-Y_{10} \end{bmatrix} \quad \text{(A.7.15)}$$

The symbolic engine in the MIXEDMODELS platform, which is written in Maple, performs the task of forming the system equations in explicit symbolic form from the component equations. These explicit expressions can then be output as C or FORTRAN code for numerical solution. The solution process for the nonlinear formulation would be explained in detail in the subsequent paragraphs. The ordinary differential equations (ODEs) in Equation (A.7.8) can be solved to obtain the differential variable vector **X** using any suitable ODE solver. To be able to do so, at any time *t*, given the current estimate of the dependent variable vector **X** in the ODEs, we need a way to calculate the derivatives $\dot{\mathbf{X}}=\begin{bmatrix}\dot{\mathbf{X}}_\mathbf{H} & \dot{\mathbf{X}}_\mathbf{L}\end{bmatrix}^{\mathrm{T}}$. $\dot{\mathbf{X}}_\mathbf{L}$ are calculated directly by using Equation (A.7.2). $\dot{\mathbf{X}}_\mathbf{H}$ and **Y** are calculated using Newton-Raphson iteration as summarized below.

Let us define a vector **q** as

$$\mathbf{q}=\begin{bmatrix}\dot{\mathbf{X}}_\mathbf{H} \\ \mathbf{Y}\end{bmatrix}_{(N+M)\times 1} \quad \text{(A.7.16)}$$

where N denotes the number of higher order differential variables and M denotes the number of algebraic variables. Given the initial conditions, $\mathbf{X}$, and initial guesses on $\dot{\mathbf{X}}_{\mathbf{H}}$ and $\mathbf{Y}$, we can iteratively calculate

$$\mathbf{q}^{k+1} = \mathbf{q}^{k} - \mathrm{J}^{-1}(\mathbf{q}^{k})\mathbf{f}(\mathbf{P}, \mathbf{X}, \mathbf{q}^{k}, t) \tag{A.7.17}$$

where J is the Jacobian matrix given by

$$\mathrm{J}(\mathbf{P}, \mathbf{X}, \mathbf{d}, t) = \begin{bmatrix} \dfrac{\partial \mathbf{f}}{\partial \dot{\mathbf{X}}_{\mathbf{H}}} & \dfrac{\partial \mathbf{f}}{\partial \mathbf{Y}} \end{bmatrix} \tag{A.7.18}$$

Let $\Delta\mathbf{q}$ denote the Newton differences such that

$$\Delta\mathbf{q} = \mathbf{q}^{k+1} - \mathbf{q}^{k} = \begin{bmatrix} \Delta\dot{\mathbf{X}}_{\mathbf{H}} \\ \Delta\mathbf{Y} \end{bmatrix} \tag{A.7.19}$$

The Newton differences can be calculated by solving the linear system

$$\mathrm{J}(\mathbf{q}^{k}) \begin{bmatrix} \Delta\dot{\mathbf{X}}_{\mathbf{H}} \\ \Delta\mathbf{Y} \end{bmatrix} = -\mathbf{f}(\mathbf{q}^{k}) \tag{A.7.20}$$

The improved estimate $\mathbf{q}^{k+1}$ is then obtained from

$$\mathbf{q}^{k+1} = \mathbf{q}^{k} + \begin{bmatrix} \Delta\dot{\mathbf{X}}_{\mathbf{H}} \\ \Delta\mathbf{Y} \end{bmatrix} \tag{A.7.21}$$

This is equivalent to the iteration implied in Equation (A.7.17) but is preferred because we do not need to invert the Jacobian matrix. The Jacobian matrix is also formed symbolically by the MIXEDMODELS symbolic platform. Once the Newton-Raphson iteration has converged, we will have not only the derivatives needed by the ODE solver, but also the values of the algebraic system variables $\mathbf{Y}$, since both of these are contained in the $\mathbf{q}$ vector.

$$\mathbf{J}=\begin{bmatrix}
0 & 0 & 0 & 0 & 0 & 1 & 0 & -1 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & -1 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & -1 & 0 & 0 \\
0 & 0 & 0 & 0 & -1 & 0 & 0 & 0 & -1 & 0 & 0 & 1 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & \left(-\frac{I_s}{\eta V_T}.e^{\frac{Y_6}{\eta V_T}}\right) & 1 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 1 & -1 & 0 & 0 & -1 & -R_b & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 1 & -1 & 0 & 0 & 0 & 0 & 0 & 0 \\
C_{value} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -1 & 0 & 0 \\
0 & J_m & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\
0 & 0 & L_a & 0 & 0 & 0 & -1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & J_L & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -1 \\
0 & 1 & 0 & -1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & -1
\end{bmatrix} \tag{A.7.22}$$

$$-\mathbf{f}=-\begin{bmatrix}
Y_2-Y_4-V_s\sin(2\pi ft) \\
Y_1-Y_7 \\
Y_7-Y_8-X_{H_3} \\
Y_1-Y_8-X_{H_3}+Y_5 \\
Y_4 \\
Y_7-I_s(e^{\frac{Y_6}{\eta V_T}}-1) \\
Y_2-Y_3-R_b.Y_7-Y_6 \\
Y_3-Y_4-X_{H_1} \\
C_{value}.\dot{X}_{H_1}-Y_8 \\
J_m.\dot{X}_{H_2}+B_v.X_{H_2}-K_T.X_{H_3}+Y_9 \\
L_a.\dot{X}_{H_3}-Y_3+Y_4+K_b.X_{H_2}+R_a.X_{H_3} \\
J_L.\dot{X}_{H_4}-Y_{10} \\
\dot{X}_{H_2}-\dot{X}_{H_4} \\
Y_9-Y_{10}
\end{bmatrix} \tag{A.7.23}$$

For the system shown in Figure (A.1.1), the Jacobian matrix, **J**, and the right hand side vector, (**-f**), are given by Equations (A.7.22) and (A.7.23). In this formulation also, a consistent set of initial conditions is calculated as explained in the Section (A.6). This formulation was tested on several systems and the results obtained were satisfactory. The same formulation can be used for linear systems also, because in case of linear systems, the Newton-Raphson algorithm will converge in one iteration. However, for very large systems, to avoid the extra calculations, the linear first-order form can be used. It may be difficult for the user to identify if the system is linear or nonlinear, and therefore to avoid this issue, the symbolic engine of MIXEDMODELS automatically identifies if the system is linear or nonlinear using a

very simple check. For this verification, the Jacobian matrix can be used. Notice that for a linear system the Hessian matrix, **H**, would be a null matrix. Hessian matrix is calculated by differentiating the Jacobian matrix with respect to the vector of unknowns, in this case, $\mathbf{q}^k$. Therefore, if the Hessian, **H**, for a system is nonzero, the controlling program automatically selects nonlinear formulation for analysis.

Once we have a general mathematical formulation that generates explicit equations, then it can be further extended for parametric studies such as sensitivity analysis, optimization etc.

A.8 Sensitivity Analysis of Multidisciplinary Systems

Based on the mathematical formulations for the system analysis, we can now derive a formulation for design sensitivity analysis. Design sensitivity formulation can be derived for the linear first-order form as well as the nonlinear form. Both these sensitivity formulations have been successfully implemented in the MIXEDMODELS platform. This appendix will present sensitivity formulation for the nonlinear form alone as it can be generally applied to any MDS. For the details about sensitivity analysis formulation for the linear first-order form, the reader is referred to Chapter 3.

Design sensitivity information is useful in the design of multidisciplinary multibody systems. It can be used for gradient-based parametric optimization as well as optimal tolerancing. The robustness of the system design can also be improved by using design sensitivity information to perform minimum sensitivity or minimum variability design. Finally, design sensitivity information can be used to generate high fidelity system metamodels that can be useful in the design process, particularly in early stage design.

In this appendix we present an analytical design sensitivity analysis formulation for dynamic multidisciplinary multibody systems. There are two popular approaches for analytical sensitivity design analysis: the direct differentiation approach and the adjoint variable approach. The adjoint variable approach has the potential to reduce the computational burden, however, it requires integrating the adjoint equations backward in time. This can cause difficulties in error control and complexities in

software implementation. Therefore, we choose the direct differentiation approach to develop the sensitivity formulation.

It is assumed that the performance functions of interest in the system are of the form

$$\mathrm{g}_i(\mathbf{P}, \mathbf{X}(\mathbf{d},t), \dot{\mathbf{X}}(\mathbf{d},t), \mathbf{Y}(\mathbf{d},t), \mathbf{d}, t), \quad i = 1 \cdots n_g \tag{A.8.1}$$

where n_g denotes the number of performance functions. For a particular performance function, g_i, we can then derive the sensitivity vector by differentiating Equation (A.8.1) with respect to the design variable vector as given below,

$$\begin{aligned} &\frac{d}{d\mathbf{d}}\left(\mathrm{g}_i(\mathbf{P}, \mathbf{X}(\mathbf{d},t), \dot{\mathbf{X}}(\mathbf{d},t), \mathbf{Y}(\mathbf{d},t), \mathbf{d}, t)\right) \\ &= \frac{d}{d\mathbf{d}}\left(\mathrm{g}_i\right) + \frac{\partial\left(\mathrm{g}_i\right)}{\partial\dot{\mathbf{X}}} \cdot \frac{\partial\left(\dot{\mathbf{X}}\right)}{\partial\mathbf{d}} + \frac{\partial\left(\mathrm{g}_i\right)}{\partial\mathbf{X}} \cdot \frac{\partial\left(\mathbf{X}\right)}{\partial\mathbf{d}} + \frac{\partial\left(\mathrm{g}_i\right)}{\partial\mathbf{Y}} \cdot \frac{\partial\left(\mathbf{Y}\right)}{\partial\mathbf{d}} \end{aligned} \tag{A.8.2}$$

Equation (A.8.2) can be written as,

$$\left(\mathrm{g}_i\right)_{\mathbf{d}} = \left(\mathrm{g}_i\right)_{\mathbf{d}|\exp} + \left(\mathrm{g}_i\right)_{\dot{\mathbf{X}}} \dot{\mathbf{X}}_{\mathbf{d}} + \left(\mathrm{g}_i\right)_{\mathbf{X}} \mathbf{X}_{\mathbf{d}} + \left(\mathrm{g}_i\right)_{\mathbf{Y}} \mathbf{Y}_{\mathbf{d}} \tag{A.8.3}$$

where a vector subscript denotes partial differentiation with respect to the subscript and $(\mathrm{g}_i)_{\mathbf{d}|\exp}$ represents the explicit derivative of g_i with respect to $\mathbf{d}$. If N_d is the number of design variables, then the dimension of each term in Equation (A.8.3) can be given by Equation (A.8.3.a) as follows.

$$\underset{n_g \times N_d}{\left(\mathrm{g}_i\right)_{\mathbf{d}}} = \underset{n_g \times N_d}{\left(\mathrm{g}_i\right)_{\mathbf{d}|\exp}} + \underset{n_g \times (N+L)}{\left(\mathrm{g}_i\right)_{\dot{\mathbf{X}}}} \cdot \underset{(N+L) \times N_d}{\dot{\mathbf{X}}_{\mathbf{d}}} + \underset{n_g \times (N+L)}{\left(\mathrm{g}_i\right)_{\mathbf{X}}} \cdot \underset{(N+L) \times N_d}{\mathbf{X}_{\mathbf{d}}} + \underset{n_g \times M}{\left(\mathrm{g}_i\right)_{\mathbf{Y}}} \cdot \underset{M \times N_d}{\mathbf{Y}_{\mathbf{d}}} \tag{A.8.3a}$$

From this equation it can be seen that we need to calculate the derivatives of g_i with respect to d, and the sensitivities $\dot{\mathbf{X}}_{\mathbf{d}}$, $\mathbf{X}_{\mathbf{d}}$ and $\mathbf{Y}_{\mathbf{d}}$. The derivatives of g_i in Equation (A.8.3) are directly obtained by differentiation in the MIXEDMODELS symbolic engine. However, the state sensitivities $\dot{\mathbf{X}}_{\mathbf{d}}$, $\mathbf{X}_{\mathbf{d}}$ and $\mathbf{Y}_{\mathbf{d}}$ need to be calculated numerically. These can be obtained by differentiating the system of governing equations with respect to the design vector. Note that based on the partition of the $\mathbf{X}$ vector given by Equation (A.4.1), we can partition the vector $\mathbf{X}_{\mathbf{d}}$ in a similar way:

$$\mathbf{X_d} = \begin{bmatrix} \mathbf{X_{H_d}} \\ \mathbf{X_{L_d}} \end{bmatrix}_{(N+L)\times N_d} \tag{A.8.4}$$

Then the state sensitivities $\mathbf{x_{L_d}}$ can be easily calculated from the direct assignment equations given by

$$\dot{\mathbf{X}}_{\mathbf{L_d}} = \mathbf{X'_d} \tag{A.8.5}$$

where, $\mathbf{X'_d}$ is a subvector of $\mathbf{X_d}$.

Let

$$\mathbf{q_d} = \begin{bmatrix} \dot{\mathbf{X}}_{\mathbf{H_d}} \\ \mathbf{Y_d} \end{bmatrix}_{(N+M)\times(N_d)} \tag{A.8.6}$$

where, N_d denotes the number of design variables. Now we need to calculate $\mathbf{q_d}$. By differentiating Equation (A.7.1) with respect to the design variable vector **d** we can solve for the sensitivities as follows:

Differentiating Equation (A.7.1) with respect to the design variable vector **d** we get,

$$\frac{d}{d\mathbf{d}}\left(\mathbf{f}(\mathbf{P},\mathbf{X}(\mathbf{d},t),\dot{\mathbf{X}}_{\mathbf{H}}(\mathbf{d},t),\mathbf{Y}(\mathbf{d},t),\mathbf{d},t)\right)=0 \qquad (a)$$

$$\therefore \quad \frac{d\mathbf{f}}{d\mathbf{d}}+\frac{\partial(\mathbf{f})}{\partial\dot{\mathbf{X}}_{\mathbf{H}}}.\frac{\partial(\dot{\mathbf{X}}_{\mathbf{H}})}{\partial\mathbf{d}}+\frac{\partial(\mathbf{f})}{\partial\mathbf{X}}.\frac{\partial(\mathbf{X})}{\partial\mathbf{d}}+\frac{\partial(\mathbf{f})}{\partial\mathbf{Y}}.\frac{\partial(\mathbf{Y})}{\partial\mathbf{d}}=0 \quad (b) \tag{A.8.7}$$

$$\Rightarrow \quad \mathbf{f_d}=\mathbf{f_{d|exp}}+\mathbf{f_{\dot{X}_H}}\dot{\mathbf{X}}_{\mathbf{H_d}}+\mathbf{f_X}\mathbf{X_d}+\mathbf{f_Y}\mathbf{Y_d}=\mathbf{0} \qquad (c)$$

The dimensions of each term in Equation (A.8.7c) are given in Equation (A.8.7d).

$$\underset{(N+M)\times N_d}{(\mathbf{f})_{\mathbf{d}}} = \underset{(N+M)\times N_d}{(\mathbf{f})_{\mathbf{d|exp}}} + \underset{(N+M)\times N}{(\mathbf{f})_{\mathbf{X_H}}}.\underset{N\times N_d}{\dot{\mathbf{X}}_{\mathbf{H_d}}} + \underset{(N+M)\times(N+L)}{(\mathbf{f})_{\mathbf{X}}}.\underset{(N+L)\times N_d}{\mathbf{X_d}} + \underset{(N+M)\times M}{(\mathbf{f})_{\mathbf{Y}}}.\underset{M\times N_d}{\mathbf{Y_d}} \tag{A.8.7d}$$

Rearranging Equation (A.8.7c) we get,

$$\therefore \quad \mathbf{f_{\dot{X}_H}}\dot{\mathbf{X}}_{\mathbf{H_d}}+\mathbf{f_Y}\mathbf{Y_d}=-\mathbf{f_{d|exp}}-\mathbf{f_X}\mathbf{X_d} \tag{A.8.8}$$

$$\therefore \qquad \begin{bmatrix} \mathbf{f}_{\dot{\mathbf{X}}_H} & \mathbf{f}_{\mathbf{Y}} \end{bmatrix} \begin{bmatrix} \dot{\mathbf{X}}_{H_d} \\ \mathbf{Y}_d \end{bmatrix} = -\mathbf{f}_{\mathbf{d}|\exp} - \mathbf{f}_{\mathbf{X}} \mathbf{X}_{\mathbf{d}} \tag{A.8.9}$$

Equations (A.8.9) and (A.8.5) together represent a system of DAEs that can be solved numerically to obtain the state sensitivities in a manner analogous to the solution of the system governing DAEs of Equation (A.7.1). We further note that unlike Equation (A.7.1) (which is nonlinear in the derivatives of the differential variables as well as the algebraic variables) Equations (A.8.5) and (A.8.9) are linear in the corresponding sensitivities. Thus, they can be solved directly without iteration. Further, we note that

$$\begin{bmatrix} \mathbf{f}_{\dot{\mathbf{X}}_H} & \mathbf{f}_{\mathbf{Y}} \end{bmatrix} = \begin{bmatrix} \dfrac{\partial \mathbf{f}}{\partial \dot{\mathbf{X}}_H} & \dfrac{\partial \mathbf{f}}{\partial \mathbf{Y}} \end{bmatrix} = \mathbf{J} \tag{A.8.10}$$

which is the same as the Jacobian in Equation (A.7.18). Thus, the coefficient matrix for Equation (A.8.9) is already available, which makes the solution of this equation convenient.

A.8.1 Calculating Initial Conditions

Recall from Section (A.6) that in order to start the integration from the given initial time, initial conditions must be provided for all of the differential variables that we wish to determine. With sensitivity analysis specifically, initial conditions must be given not only for $\mathbf{X}$, but also for $\mathbf{X_d}$. From Equation (A.6.1), recall that to ensure consistency of initial conditions, we assume that the user provides a set of consistency equations that must be satisfied by the initial conditions. To place the following discussion in context let us rewrite Equation (A.6.1).

$$\mathbf{h}(\mathbf{P}, \mathbf{X}(\mathbf{d}, t), \mathbf{d}, t)|_{t=0} = \mathbf{0} \tag{A.8.11}$$

where $\mathbf{h}$ is a vector of nonlinear algebraic equations of dimension $(N+L)$ that can be solved by Newton-Raphson iteration to obtain a consistent initial conditions vector, $\mathbf{X}$. Recall that N is the number of higher-order differential variables and L is the number of lower-order differential variables. All that the user needs to provide then is the set of initial guesses for $\mathbf{X}$, which are then corrected by the iteration. Once we

have a consistent set of **X**, Equation (A.7.8) can be solved to obtain $\dot{\mathbf{X}}$ and **Y** at initial time *t*. In order to compute initial conditions on sensitivities, we differentiate Equation (A.8.11) with respect to the design variable vector **d** as given by Equation (A.8.12).

$$\frac{d}{d\mathbf{d}}\left(\mathbf{h}(\mathbf{P},\mathbf{X}(\mathbf{d},t),\mathbf{d},t)|_{t=0}\right)=\frac{d\mathbf{h}}{d\mathbf{d}}+\frac{\partial(\mathbf{h})}{\partial\mathbf{X}}.\frac{\partial(\mathbf{X})}{\partial\mathbf{d}}=0 \qquad (a)$$

$$\mathbf{h_d}=\mathbf{h_{d|exp}}+\mathbf{h_X}\mathbf{X_d}=\mathbf{0} \qquad (b) \qquad \text{(A.8.12)}$$

Rearranging Equation (A.8.12b), the initial condition vector, $\mathbf{X_d}$, for sensitivities can then be obtained by solving the resulting Equation (A.8.13) as a linear system of algebraic equations.

$$\mathbf{h_X}.\mathbf{X_d}=-\mathbf{h_{d|exp}} \qquad \text{(A.8.13)}$$

Since this is a system of linear equations, the user does not have to provide initial guesses for the $\mathbf{X_d}$ – the correct initial conditions are calculated directly from Equation (A.8.13). Once we have the values for $\mathbf{X_d}$, then the values of $\dot{\mathbf{X}}_\mathbf{d}$ and $\mathbf{Y_d}$ at the initial time can be obtained by solving the system of equations given by Equation (A.8.9). Once again, we can take advantage of the fact that the coefficient matrix in Equation (A.8.13) is the same as the Jacobian in the Newton-Raphson iteration for Equation (A.8.10).

As an example, let us consider the *source voltage* and the *inertia of the DC motor* as the two design variables.

$$\mathbf{d} = [d_1\ d_2]^T = [\text{source voltage, motor inertia}]^T = [V_s, J_m]^T \qquad \text{(A.8.14)}$$

The system sensitivities are calculated with respect to each design variable. For sensitivity analysis, two types of performance functions are popular: a grid type" and an "integral type". As an example, let us choose *instantaneous power of the DC motor* as the grid type performance function which can be imposed at a certain time interval, for example, every 0.1 s.

$$g_1 = V_c I_a = X_{H_1}.X_{H_3} \qquad \text{(A.8.15)}$$

For the system shown in Figure (A.1.1), the number of higher-order differential variables, N, is 4, the number of lower-order differential variables, L, is 2, and the number of algebraic variables, M, is 10. For two design variables, $N_d = 2$.

From Equation (A.8.15), the sensitivities for g_1, can be calculated using Equation (A.8.3). Similarly Equations (A.8.5) and (A.8.9) can be symbolically generated and the state sensitivities can then be numerically computed. The solution process for the sensitivity formulation is briefly described in Section (A.8.2).

$$\left(g_1\right)_{\mathbf{d}} = \left(g_1\right)_{\mathbf{d}|\exp} + \left(g_1\right)_{\dot{\mathbf{X}}} \dot{\mathbf{X}}_{\mathbf{d}} + \left(g_1\right)_{\mathbf{X}} \mathbf{X}_{\mathbf{d}} + \left(g_1\right)_{\mathbf{Y}} \mathbf{Y}_{\mathbf{d}}$$

$$\underset{1\times 2}{\left[\frac{d}{d_1}\left(g_1\right) \quad \frac{d}{d_2}\left(g_1\right)\right]} = \underset{1\times 6}{\left[X_{H_3} \quad 0 \quad X_{H_1} \quad 0 \quad 0 \quad 0\right]} \cdot \underset{6\times 2}{\begin{bmatrix} \frac{\partial X_{H_1}}{\partial d_1} & \frac{\partial X_{H_1}}{\partial d_2} \\ \frac{\partial X_{H_2}}{\partial d_1} & \frac{\partial X_{H_2}}{\partial d_2} \\ \frac{\partial X_{H_3}}{\partial d_1} & \frac{\partial X_{H_3}}{\partial d_2} \\ \frac{\partial X_{H_4}}{\partial d_1} & \frac{\partial X_{H_4}}{\partial d_2} \\ \frac{\partial X_{L_1}}{\partial d_1} & \frac{\partial X_{L_1}}{\partial d_2} \\ \frac{\partial X_{L_2}}{\partial d_1} & \frac{\partial X_{L_2}}{\partial d_2} \end{bmatrix}}$$

$$= \underset{1\times 2}{\left[X_{H_3} \cdot \left(\frac{\partial X_{H_1}}{\partial d_1} + \frac{\partial X_{H_1}}{\partial d_2}\right) \quad X_{H_1} \cdot \left(\frac{\partial X_{H_3}}{\partial d_1} + \frac{\partial X_{H_3}}{\partial d_2}\right)\right]} \tag{A.8.16}$$

A.8.2 Solution Process for the Sensitivity Analysis Formulation

The system of DAEs of Equations in (A.8.5) and (A.8.9) is solved numerically concurrently with the DAEs of Equation (A.7.1). The ODEs in these equations can be solved by any suitable ODE solver. The dependent variables seen by the ODE solver are now the system differential variables $\mathbf{X}$ as well as their sensitivities $\mathbf{X_d}$. When the ODE solver calls for derivative evaluation at a particular time t with the current estimate for this set of dependent variables (i.e., $\mathbf{X}$ and $\mathbf{X_d}$), we do the following;

- First, perform the Newton-Raphson iteration of Equations (A.7.20) and (A.7.21) to calculate the derivatives of the higher-order differential state variables $\dot{\mathbf{X}}_{\mathbf{H}}$, and the algebraic state variables **Y**.
- Next, use the direct assignments in Equation (A.7.2) to set the values of the lower order differential state variables, $\dot{\mathbf{X}}_{\mathbf{L}}$.
- Once we have the values of all the state variables, we can solve Equation (A.8.9) to obtain the derivatives of the sensitivities of the higher-order differential variables $\dot{\mathbf{X}}_{\mathbf{H_d}}$ and the sensitivities of the algebraic state variables $\mathbf{Y_d}$.
- Finally, the derivatives of the sensitivities of the lower-order differential state variables are set through direct assignment from Equation (A.8.5).

Several systems were simulated and this formulation was verified using perturbation analysis, the details of which can be found in Chapters 4 and 6.

This appendix presented a detailed derivation of the linear first-order formulation and a general nonlinear formulation for the analysis of general multidisciplinary systems. Both the formulations generate symbolic expressions for the system governing equations allowing easy extension to sensitivity analysis. An analytical sensitivity analysis formulation was developed based on the general nonlinear formulation and this integrated analysis and design scheme was successfully implemented in the MIXEDMODELS platform. By using symbolic computing to formulate the governing equations and numeric computing to solve these equations, this formulation offers flexibility and convenience in modeling, along with efficiency in computation.

APPENDIX B: MAPLE CODE FOR THE SYMBOLIC ENGINE OF MIXEDMODELS

B.1 Driver Program in MAPLE for the Symbolic Engine of MIXEDMODELS

```
# To run the program in Maple, type the following command:
# read "C:/Research/workingDirectory/driverFiles/TestSystemSymbolicEngine.txt";
# Set the LinearFlag = 1 to set the default to be 'a linear system'
linFlag := 1:
# Number of Elements Stored  as History - Set the default value for nHist to 0
nHist := 1:
# Set the default values for alpha and beta for constraint stabilization
alpha := 150:
beta := 10:
# Set the default value for varElim
varElim := "no":
# Set the default value of mf for DLSODES
# For stiff integrator with internally calculated Jacobian, mf = 222
mf := 222:
# Read System Specification File
read "C:/Research/workingDirectory/dataSimFiles/TestSystemSpecFile.ems";
nComp := nCompMain:
# Allocate space in memory to store parameter and offset arrays
# Allocate space for parameter arrays
PS := array(1..nComp, 1..10):
```

```
PR := array(1..nComp, 1..15):
PI := array(1..nComp, 1..15):
PE := array(1..nComp, 1..10):
# Declare a vector of design variables
d := array(1..Nd):

# Allocate space for offset arrays
offXH := array(1..nComp):
offXL := array(1..nComp):
offY := array(1..nComp):
offYalg := array(1..nComp):
# For non-linear formulation A represents J, the Jacobian a component model
# can have only matrix A, b entrie or only function entries or both
offA := array(1..nComp):
offS := array(1..nComp):
offFunc := array(1..nComp):
# Allocate space for variable count arrays
nH := array(1..nComp):
nL := array(1..nComp):
nM := array(1..nComp):
nA := array(1..nComp):
nS := array(1..nComp):
# array to store number of nonlinear functions per component
nf := array(1..nComp):
```

```
# Initialize all arrays to zero
for i from 1 by 1 to nComp do
    nH[i] := 0:
    nL[i] := 0:
    nM[i] := 0:
    nA[i] := 0:
    nS[i] := 0:
    nf[i] := 0:
end do:
# Initialize variables and the first elements of offset arrays
N := 0:
L := 0:
M := 0:
NA :=0:
nSC := 0:
# NFunc =  total number of functions given by the user for a system
NFunc := 0:
offXH[1] := 1:
offXL[1] := 1:
offY[1] := 1:
offA[1] := 1:
offS[1] := 1:
offFunc[1] := 1:
offYalg[1] := 1:
```

```
# Read the Data File
#read "C:/Research/workingDirectory/dataSimFiles/TestSystem.txt";
# Specify the file path to access component files
filePath := "C:/Research/workingDirectory/ComponentFiles/":
for i from 1 by 1 to nComp do
   # component type
   check := i;
   # Read the header file to retrive information about variables and matrix equations
   fileHDR := cat(PS[i,1], ".hdr"):
   fileName := cat(filePath,fileHDR):
   read fileName;
   # Calculate the total number of variables and equations of the system
   N := N + nH[i]:
   L := L + nL[i]:
   M := M + nM[i]:
   NA := NA + nA[i] + nf[i]:
   nSC := nSC + nS[i]:
   NFunc := NFunc + nf[i]:
   # Calculate and update offsets for variables and equations
    if (i<>nComp) then
       offXH[i+1] := offXH[i] + nH[i]:
       offXL[i+1] := offXL[i] + nL[i]:
       offY[i+1] := offY[i] + nM[i]:
       offYalg[i+1] := offYalg[i] + nM[i]:
```

```
        offA[i+1] := offA[i] + nA[i] + nf[i]:
        offS[i+1] := offS[i] + nS[i]:
        offFunc[i+1] := offFunc[i] + nf[i]:
    end if;
end do;

# Dimension SC array and initialize it
if (nSC = 0) then
  nSC := 1:
end if;
SC := array(1..nSC):
for i from 1 by 1 to nSC do
    SC[i] := 0:
end do:

# Shift the Algebraic Variable vector array by N
# such that columns 1..N of A correspond to XHDot
# and columns N+1..N+M correspond to Y
for i from 1 by 1 to nComp do
     offY[i] := offY[i] + N:
end do:

# Dimension settings and space allocation
# allocate space for matrix A and vectors b
```

```
# and other work arrays and variable arrays
A := array(1..(N+M),1..(N+M)):
b := array(1..(N+M)):
ASave := array(1..(N+M),1..(N+M)):
bSave := array(1..(N+M)):
bLin := array(1..(N+M)):
bLinSave := array(1..(N+M)):
XHDot:= array(1..N):
XLDot:= array(1..L):
XH := array(1..N):
XL := array(1..L):
q := array(1..N+M):
f := array(1..NFunc):
Aq := array(1..N+M):
Aqn := array(1..N+M):

# Initialize A and b to 0
for i from 1 by 1 to (N+M) do
    b[i] := 0:
    bLin[i] := 0:
    bLinSave[i] := 0:
    Aq[i] := 0:
    Aqn[i] := 0:
end do:
```

```
for i from 1 by 1 to (N+M) do
    for j from 1 by 1 to (N+M) do
        A[i,j] := 0:
    end do:
end do:
# Set up the variable vector q = [XHDot Y]
for i from 1 by 1 to N do
    q[i] := XHDot[i]:
end do:
for i from 1 by 1 to M do
    q[N+i] := Y[i]:
end do:

# The following piece of code populates matrices A, b and XHDot
# based on the contribution each component has to make to the system
# It also generates maple statements to calculate constraints for each component

# Second Pass
# Generate A, b and XLDot symbolically
for i from 1 by 1 to nComp do
   fileMDL := cat(PS[i,1], ".model"):
   fileName := cat(filePath, fileMDL):
   read fileName;
  check := i;
```

```
end do;

# Nonlinear form
# save b for verification
ASave := A:
bSave := b:
evalm(ASave);
check_bSave_before_NL_calc;
evalm(bSave);
evalm(q);

# Evaluate Aq-b to convert A and b entries to functions
# Store all the function entries to b
for i from 1 by 1 to N+M do
    for j from 1 by 1 to N+M do
        Aq[i] := Aq[i] + A[i,j]*q[j]:
    end do:
    b[i] := Aq[i] - b[i]:
end do:

# Fill in the 'b' entries corresponding to function entries in the b vector
# Rearrange matrix A by filling in the Jacobian entries for the non-linear functions
# For each Jacobian entry check if the equation is linear or non-linear
# Reset linFlag to 0 on the first non-linear entry found and exit the 'linCheck' loop
```

```
evalm(nf);
evalm(A);
evalm(b);
linFlagdisp := linFlag;
linsave := 10;
for i from 1 by 1 to nComp do
    for j from 1 by 1 to nf[i] do
        b[offA[i] + nA[i] + j - 1] := f[offFunc[i] + j -1]:
        for k from 1 by 1 to N+M do
            A[offA[i] + nA[i] + j - 1, k] := diff(f[offFunc[i] + j - 1], q[k]):
            for k1 from 1 by 1 to N+M do
                if (linFlag <> 0) then
                  linCheck := diff(A[offA[i] + nA[i] + j - 1, k], q[k1]):
                  if (linCheck <> 0) then
                     compIndex := i:
                     varIndex := k1:
                     linFlag := 0:
                  end if:
                end if:
            end do;
        end do;
    end do;
end do;
evalm(A);
```

```
evalm(b):


#linFlagDisp := linFlag;
#compdisp:=compIndex;
#vardisp:=varIndex;
#evalm(SC);
nSC;


# if (linFlag = 1), generate the right hand side for Linear System as well and
# In this case write out both the linear and nonlinear right hand sides to Fortran
# Evaluate Aq-f to evaluate the right hand side for the linear system
# f is stored in b => bLin = Aq-b


if (linFlag = 1) then
   for i from 1 by 1 to N+M do
      for j from 1 by 1 to N+M do
           Aqn[i] := Aqn[i] + A[i,j]*q[j]:
      end do:
      bLin[i] := Aqn[i] - b[i]:
   end do:
end if:
bLinSave := bLin:
evalm(bLin);
evalm(bLinSave);
```

```
evalm(b);

# Maple Code for Sensitivity Analysis
# This code generates matrices bx and bd
if (Nd <> 0) then
   read "C:/Research/workingDirectory/driverFiles/SenctvtCalc.txt";
end if;

# Reduction Routine Selection Enable
if (varElim = "yes") then
   read "C:/Research/workingDirectory/driverFiles/variableReductionFacility.txt";
else
   nqElim := 0:
end if;

# Changes made to include Sparse Matrix Solver Routine Y12MAF
# Calculate the number of non-zero elements in matrix A
# Set up arrays SNR and RNR for Sparse Matrix Solver Y12MAF
nzcnt := 0:
for i from 1 to (N+M-nqElim) do
    for j from 1 to (N+M-nqElim) do
       if (A[i,j] <> 0) then
          nzcnt := nzcnt + 1:t
       end if:
```

```
    end do:
end do:
nsms := 3*nzcnt + 1:
As := array(1..nsms):
SNR := array(1..nsms):
RNR := array(1..nsms):

# Initialize arrays to 0.0
for i from 1 to nsms do
    SNR[i] := 0:
    RNR[i] := 0:
    As[i] := 0.0:
end do:

k := 1:
for i from 1 to (N+M-nqElim) do
    for j from 1 to (N+M-nqElim) do
        if (A[i,j]<>0) then
#  store non-zero elements of the matrix in nzA
            As[k] := A[i,j]:
# Set SNR -> column number of non-zero elements in A
            SNR[k] := j:
# Set RNR -> row number of non-zero elements in A
            RNR[k] := i:
```

```
            k := k + 1:
        end if:
    end do:
end do:

# Calculate the number of output files that need to be opened
# to store XHL, XHLD and Y from Fotran
# Each file can successfully display 4 columns
qXHL := iquo((N+L)*(Nd+1), 4, rXHL):
qY := iquo(M*(Nd+1), 4, rY):
qSC := iquo(nSC, 4, rSC):

if (rXHL = 0) then
  nXHLout := qXHL;
else
 nXHLout := qXHL + 1;
end if;

if (rY = 0) then
  mYout := qY;
else
 mYout := qY + 1;
end if;
```

```
if (rSC = 0) then
   nSCout := qSC;
else
  nSCout := qSC + 1;
end if;

nSCout;

# Fortran mainFile output
read
"C:/Research/workingDirectory/driverFiles/genProblemSpecificFortranCode.txt";
```

B.2 MAPLE Code for Sensitivity Analysis Calculations – "SenctvtCalc.txt"

```
#read "C:/Research/workingDirectory/driverFiles/SenctvtCalc.txt";
XHL := array(1..(N+L)*(Nd+1)):
XHLDot := array(1..(N+L)*(Nd+1)):
Y := array(1..M*(Nd+1)):
bx := array(1..N+M, 1..N+L):
bd := array(1..N+M, 1..Nd):
br := array(1..N+M, 1..Nd):

# Initialize bx and bd to 0
if (Nd <> 0) then
  for i from 1 by 1 to N+M do
      for j from 1 by 1 to N+L do
          bx[i,j] := 0.0:
      end do:
      for j from 1 by 1 to Nd do
          bd[i,j] := 0.0:
#           br[i,j] := 0.0:
      end do:
 end do:

# Calculate partial of f with respect to XHL => bx
  for i from 1 by 1 to N+M do
      for j from 1 by 1 to N do
```

```
            bx[i,j] := diff(b[i], XH[j]):
        end do:
        for j from 1 by 1 to L do
            bx[i,N+j] := diff(b[i], XL[j]):
        end do:
    end do:

# Calculate explicit partial of f with respect to d => bd
    for i from 1 by 1 to N+M do
        for j from 1 by 1 to Nd do
            bd[i,j] := diff(b[i], d[j]):
        end do:
    end do:
end if;
```

B.3 MAPLE Code for Symbolic Reduction – "variableReductionFacility.txt"

```
# Variable Reduction Code
with(LinearAlgebra):

# Generate an array q to store XHDot and Y variables
unassign('q'):
q := array(1..N+M):

# update qUpdate vector as and when a variable gets eliminated from the vector q
qUpdate := Matrix(1,N+M):

# Generate an array qElim to store eliminated variables
qElim := array(1..N+M):

# Generate an array qAssign to store respective assignments to qElim
qAssign := array(1..N+M):

#Assign 'qUpdate' to 'q'
for i from 1 by 1 to N+M do
    qUpdate[1,i] := q[i]:
end do;
evalm(q);
evalm(qUpdate);
```

```
# Matrix operations such as 'deleteRow', 'deleteColumn' do not work on arrays
# Therefore matrices bM and AM are defined with AM = A and bM = b
bb := array(1..N+M):
bM := Matrix(N+M,1):
for i from 1 by 1 to (N+M) do
    bM[i,1] := bb[i]:
end do:

AM := Matrix(N+M):
for i from 1 by 1 to (N+M) do
    for j from 1 by 1 to (N+M) do
        AM[i,j] := A[i,j]:
   end do:
end do:

# The following piece of code evaluates the indices of the row and column to be eliminated
matSize := N+M:
testSize := matSize:
qInd := 1:
while (testSize > 1) do

# Outer loop to identify the row to be eliminated
# The two loops - row loop and the column loop scan through each row to find a row
# with either only one nonzero column entry or only two nonzero column entries
```

```
# Once such a row is detected, variable elimination routine is applied to that row
# and the process repeats


# 'for' loop on rows
  for i from 1 by 1 to matSize do
        elimRow := 0:
        elimCol := 0:
        chCol := 0:
        cntr := 0:
        first := 0:
        second := 0:
        # inner loop to identify the column to be eliminated
        # 'for' loop on columns
        for j from 1 by 1 to matSize do
              if (AM[i,j] <> 0) then
                 cntr := cntr + 1:
# forced exit for loop on columns if third nonzero element is found
                 if (cntr > 2) then
                   break;
                 end if:
                 if (cntr = 1) then
                   first := j:
                 end if:
                 second := j:
```

```
            end if:
        end do;
# end of for loop on columns
    if (cntr <= 2) then
      if (type (AM[i,first], numeric)) then
        if (type (AM[i,second], numeric)) then
          # forced exit for loop on rows if 1-or-2 element row is found
          break;
        end if;
      end if:
    end if;

end do;
# end of for loop on rows
# forced exit on while loop if NO 1-or-2 element row is found
  if ((cntr > 2) or (i > matSize)) then
    break;
  end if:

    printf ("%d\n", i);
    elimRow := i:
    if (cntr = 2) then
      if (abs(AM[i, first]) > abs(AM[i, second])) then
            elimCol := second;
```

```
            chCol := first;
        else
            elimCol := first;
            chCol := second;
        end if;
    elif (cntr = 1) then
            elimCol := first:
    end if;

# end of row for loop

    printf("elimRowIndex = %d\n", elimRow);
    printf("elimColIndex = %d\n", elimCol);
    printf("retainColIndex = %d\n", chCol);

    for i from 1 by 1 to matSize do
        if (i <> elimRow) then
            b1 := bM[elimRow,1]/AM[elimRow, elimCol];
            bM[i,1] := bM[i,1] - AM[i, elimCol]*b1:
        end if;
    end do;

    printf("counter = %d\n", cntr);
    if (cntr = 2) then
```

```
  for i from 1 by 1 to matSize do
    if (i <> elimRow) then
      a1 := AM[elimRow, chCol]/AM[elimRow, elimCol];
      AM[i, chCol] := AM[i, chCol] - AM[i, elimCol]*a1:
    end if:
  end do:
 end if:

# store the eliminated variable in qElim array
# store the corresponding assignment in qAssign array

 if (cntr <= 2) then
   qElim[qInd] := qUpdate[1,elimCol]:
 if (cntr = 2) then
   qAssign[qInd] := b1 - a1*qUpdate[1,chCol]:
 else
   if (cntr = 1) then
     qAssign[qInd] := b1:
   end if:
  end if:
  qInd := qInd + 1:
 end if:

 bM[elimRow,1] := 9999:
```

```
 for k from 1 by 1 to matSize do
      AM[elimRow, k] := 9999:
      AM[k, elimCol] := 9999:
 end do:

 AM := DeleteRow(AM, elimRow):
 AM := DeleteColumn(AM, elimCol):
 bM := DeleteRow(bM, elimRow):
 qUpdate := DeleteColumn(qUpdate, elimCol):
 evalm(AM);
 evalm(bM);
 evalm(qUpdate);
 matSize := matSize - 1:
 testSize := matSize:

end do;
# end of while loop

# The commands to convert maple output to fortran are as follows
# with(CodeGeneration):
#Fortran (<variable name>, output = outFile)
# outFile has the file name with complete path
# The issue is that this command needs variables of type 'array', it doesn't work with
# matrices. Therefore we need to define arrays again
```

```
A := array(1..matSize,1..matSize):
RHS := array(1..matSize):
for i from 1 by 1 to matSize do
    for j from 1 by 1 to matSize do
        A[i,j] := AM[i,j]:
    end do:
end do:

for i from 1 by 1 to matSize do
    RHS[i] := bM[i,1]:
end do:

nqElim := qInd-1:
qElimFinal := array(1..nqElim):
qAssignFinal := array(1..nqElim):
for i from 1 by 1 to (qInd-1) do
    qElimFinal[i] := qElim[i]:
    qAssignFinal[i] := qAssign[i]:
end do:

evalm(qElimFinal);
evalm(qAssignFinal);
evalm(qUpdate);
evalm(q);
```

```
NElim := 0:
MElim := 0:
Iq := array(1..N+M):
for i from 1 by 1 to (N+M) do
    Iq[i] := 1:
end do:

IqElim := array(1..nqElim):
for i from 1 by 1 to nqElim do
    IqElim[i] := 0:
end do:

for i from 1 by 1 to (nqElim) do
    for j from 1 by 1 to (N+M) do
        if (qElimFinal[i] = q[j]) then
            Iq[j] := 0:
            IqElim[i] := j:
            break;
        end if:
    end do:
end do:
eval(Iq);
eval(IqElim);
```

```
for i from 1 by 1 to nqElim do
    q[IqElim[i]] := qAssignFinal[i]:
end do:
```

B.4 MAPLE Code to Generate Problem Specific FORTRAN Code – "genProblemSpecificFortranCode.txt"

```
with(CodeGeneration):
outFile := "C:/Research/workingDirectory/driverFiles/TestSystemFortranCode.for";


# Generate the subroutine that sets up arrays
fd := fopen(outFile, WRITE);
fprintf(fd, "      SUBROUTINE setArrays\n\n");
fclose(fd);
fd := fopen(outFile, APPEND);
fprintf(fd, "      INTEGER N, M, L, nHist, nSC, Nd\n");
fprintf(fd, "      INTEGER nzcnt, nsms\n");
fprintf(fd, "      INTEGER lrw, liw, mf\n");
fprintf(fd, "      PARAMETER N = %d \n", N);
fprintf(fd, "      PARAMETER M = %d \n", M);
fprintf(fd, "      PARAMETER L = %d \n", L);
fprintf(fd, "      PARAMETER Nd = %d \n", Nd);
fprintf(fd, "      PARAMETER nHist = %d \n", nHist);
fprintf(fd, "      PARAMETER nqElim = %d \n", nqElim);
fprintf(fd, "      PARAMETER nSC = %d \n", nSC);
fprintf(fd, "      PARAMETER nSCout = %d \n", nSCout);
fprintf(fd, "      PARAMETER nXHLout = %d \n", nXHLout);
fprintf(fd, "      PARAMETER mYout = %d \n", mYout);
fprintf(fd, "      PARAMETER mf = %d \n", mf);
```

```
# for sparse matrix solver Y12Maf
fprintf(fd, "      PARAMETER nzcnt = %d \n", nzcnt);
fprintf(fd, "      PARAMETER nsms = %d \n", nsms);
#-----------------------------------------------------------------
# Calculations for LSODE and DLSODES
mf10lrw := 20 + 16*(N+L)*(Nd+1):
mf10liw := 20:
mf22lrw := 22 + 9*(N+L)*(Nd+1) + ((N+L)*(Nd+1))**2:
mf22liw := 20 + (N+L)*(Nd+1):
nnz := 50 + ((N+L)*(Nd+1))**2;
lenrat := 2:
mf121or222lrw := 20 + (2 +(1/lenrat))*nnz + (11+(9/lenrat))*(N+L)*(Nd+1);
mf121or222liw := 30:
maxlrw := max(mf10lrw, mf22lrw, mf121or222lrw);
maxliw := max(mf10liw, mf22liw, mf121or222liw):
fprintf(fd, "      PARAMETER lrw = %d\n", maxlrw);
fprintf(fd, "      PARAMETER liw = %d\n\n", maxliw);
#-----------------------------------------------------------------
if (((N+L)<>0) and (M <> 0)) then
   fprintf(fd, "      REAL*8 C((N+L)*(Nd+1)+30), W((N+L)*(Nd+1),9) \n");
   fprintf(fd, "                  REAL*8 delXHDot(N), delY(M), saveXHLDot
         ((N+L)*(Nd+1))\n");
   fprintf(fd, "                  REAL*8 Y(M*(Nd+1)), XHL((N+L)*(Nd+1)),
         XHLdot((N+L)*(Nd+1))\n");
   fprintf(fd, "      REAL*8 qPast((N+L+M)*(Nd+1),nHist), SC(nSC) \n");
```

```
   fprintf(fd, "      REAL*8 br((N+M),Nd+1), bx((N+M),(N+L)), bd((N+M), Nd+1),
         q(N+M)\n");
else
   fprintf(fd, "     REAL*8 C((N+L)*(Nd+1)+30), W((N+L)*(Nd+1)+1,9)\n");
   fprintf(fd, "                    REAL*8 delXHDot(N+1), delY(M+1),
         saveXHLDot((N+L)*(Nd+1)+1)\n");
   fprintf(fd, "                REAL*8 Y(M*(Nd+1)+1), XHL((N+L)*(Nd+1)+1),
         XHLdot((N+L)*(Nd+1)+1)\n");
   fprintf(fd, "     REAL*8 qast((N+L+M)*(Nd+1), nHist), SC(nSC)\n");
   fprintf(fd, "          REAL*8 br((N+M),Nd+1), bx((N+M),(N+L+1)), bd((N+M),
         Nd+1), q(N+M)\n");
end if;
fprintf(fd, "           REAL*8 work(3+(N+L)*(Nd+1)*6), rwork(lrw), RHS(N+M-
       nqElim)\n");
# Sparse Matrix Solver
fprintf(fd, "     REAL*8 As(nsms), Pivot(N+M-nqElim), RData(10)\n");
fprintf(fd, "c     Variable Declaration for scaleMatrix routine\n");
fprintf(fd, "     REAL*8 Asprv(nzcnt), Asnxt(nzcnt), rScale(nzcnt), D1A(nzcnt),\n");
fprintf(fd, "      +          D1prv(N+M-nqElim), D1nxt(N+M-nqElim), D2prv(N+M-
       nqElim),\n");
fprintf(fd, "      +          D2nxt(N+M-nqElim), DR(N+M-nqElim), DRinv(N+M-
       nqElim),\n");
fprintf(fd, "    +      DC(N+M-nqElim), DCinv(N+M-nqElim)\n\n");
fprintf(fd, "           INTEGER iwork(liw), SNR(nsms), RNR(nsms), HA(N+M-
       nqElim,11)\n");
if (((N+L) <> 0) and (M <> 0)) then
```

```
    fprintf(fd,     "                    INTEGER     XHLD2Plot((N+L)*(Nd+1)),
            XHL2Plot((N+L)*(Nd+1)), \n");
    fprintf(fd, "    +       Y2Plot(M*(Nd+1))\n");
else
    fprintf(fd,     "                    INTEGER     XHLD2Plot((N+L)*(Nd+1)+1),
            XHL2Plot((N+L)*(Nd+1)+1), \n");
    fprintf(fd, "    +       Y2Plot(M*(Nd+1)+1)\n");
end if;
fprintf(fd, "     INTEGER Iq(N+M), IData(10)\n\n");

fprintf(fd, "       COMMON/SIZES/NN, MM, LL, nnHist, nnSC, llrw, lliw, mmf,
        nnqElim,\n");
fprintf(fd, "    +          NNd\n");
fprintf(fd, "     COMMON/ALGVAR/Y\n");
fprintf(fd, "     COMMON/NEWTONDIFF/delXHDot, delY\n");
fprintf(fd, "     COMMON/SAVEXHLDOT/saveXHLDot\n");
fprintf(fd, "     COMMON/FILENUM/nnXHLout, mmYout, nnSCout\n");
fprintf(fd, "     COMMON/SIZESMS/nnzcnt, nnsms\n");
fprintf(fd, "     COMMON/REDUCTION/Iq, q, RHS\n");
fprintf(fd, "     COMMON/DESIGN/br, bx, bd, d\n");
fprintf(fd, "     COMMON/USERDATA/IData, RData\n");
fprintf(fd, "     COMMON/SCALE/Asprv, Asnxt, D1prv, D1nxt, D2prv, D2nxt,\n");
fprintf(fd, "    +          DR, DRinv, DC, DCinv, rScale, D1A\n\n");

fprintf(fd, "     NN = N\n");
```

```
fprintf(fd, "      MM = M\n");
fprintf(fd, "      LL = L\n");
fprintf(fd, "      NNd = Nd\n");
fprintf(fd, "      llrw = lrw\n");
fprintf(fd, "      lliw = liw\n");
fprintf(fd, "      mmf = mf\n");
fprintf(fd, "      nnHist = nHist\n");
fprintf(fd, "      nnqElim = nqElim\n");
fprintf(fd, "      nnSC = nSC\n");
fprintf(fd, "      nnXHLout = nXHLout\n");
fprintf(fd, "      mmYout = mYout\n");
fprintf(fd, "      nnSCout = nSCout\n");
fprintf(fd, "      nnzcnt = nzcnt\n");
fprintf(fd, "      nnsms = nsms\n\n");

fprintf(fd, "      DO 10 I = 1, N+M-nqElim\n");
fprintf(fd, "           D1prv(I) = 1.0D0\n");
fprintf(fd, "           D1nxt(I) = 1.0D0\n");
fprintf(fd, "           D2prv(I) = 1.0D0\n");
fprintf(fd, "           D2nxt(I) = 1.0D0\n");
fprintf(fd, "   10 CONTINUE\n\n");

fprintf(fd, "      CALL mainDriver (C, W, XHL, XHLdot, Y, qPast, SC, work, rwork,
        \n");
```

```
fprintf(fd,"     +iwork, saveXHLDot, delXHLDot, delY, d,\n");
fprintf(fd,"     +XHLD2Plot, XHL2Plot, Y2Plot)\n\n");


fprintf(fd, "      RETURN\n");
fprintf(fd, "      END\n\n\n\n\n\n");
fclose(fd);
# ------------------------ END setArrays ---------------------------------------


# ------------------------- BEGIN getMat ----------------------------------------
# Generate the subroutine to generate matrices
fd := fopen(outFile, APPEND);
fprintf(fd, "      SUBROUTINE getMat(t, XH, XL, As, b, XHDot, XLDot, Y,\n");
fprintf(fd,"     +       SNR, RNR, XHL, XHLDot, d)\n\n");
if ((N<>0) and (L<>0) and (M <> 0)) then
   fprintf(fd, "      REAL*8 t, XH(N), XL(L), XHDot(N), XLDot(L), Y(M)\n");
   fprintf(fd, "      REAL*8 XHL((N+L)*(Nd+1)), XHLDot((N+L)*(Nd+1))\n");
else
   fprintf(fd, "          REAL*8 t, XH(N+1), XL(L+1), XHDot(N+1), XLDot(L+1),
          Y(M+1)\n");
   fprintf(fd,      "                                REAL*8      XHL((N+L)*(Nd+1)+1),
         XHLDot((N+L)*(Nd+1)+1)\n");
end if;
fprintf(fd, "      REAL*8 As(nsms), b(N+M), d(Nd+1), RData(10)\n\n");
fprintf(fd, "      INTEGER SNR(nsms), RNR(nsms), linFlag, Lflag, IData(10)\n\n");
```

```
fprintf(fd, "          COMMON/SIZES/N, M, L, nHist, nSC, lrw, liw, mf, nqElim,
        Nd\n");
fprintf(fd, "      COMMON/SIZESMS/nzcnt, nsms\n");
fprintf(fd, "      COMMON/USERDATA/IData, RData\n");
fprintf(fd, "      COMMON/FLAGS/linFlag, Lflag\n\n");

fprintf(fd, "      DO 10 I = 1, N+M\n");
fprintf(fd, "           b(I) = 0.0\n");
fprintf(fd, "   10 CONTINUE\n\n");

fclose(fd);

Fortran(As, output = outFile);
fd := fopen(outFile, APPEND);
fprintf(fd, "       \n\n");
fclose(fd);

Fortran(SNR, output = outFile);
fd := fopen(outFile, APPEND);
fprintf(fd, "       \n\n");
fclose(fd);

Fortran(RNR, output = outFile);
fd := fopen(outFile, APPEND);
```

```
fprintf(fd, "       \n\n");
fclose(fd);

#  Write out RHS to Fortran
fd := fopen(outFile, APPEND);
fprintf(fd, "c     Use the Following RHS for the Solution of Non-Linear System\n");
fprintf(fd, "      IF (linFlag.EQ.0.OR.Lflag.EQ.0) THEN\n\n");
fclose(fd);
Fortran(b, output = outFile);

fd := fopen(outFile, APPEND);
fprintf(fd, "       \n");
fprintf(fd, "      END IF\n\n");
fclose(fd);

if (linFlag = 1) then
  for i from 1 by 1 to (N+M) do
     b[i] := bLin[i]:
  end do;
  fd := fopen(outFile, APPEND);
  fprintf(fd, "      IF (linFlag.EQ.1.AND.Lflag.EQ.1) THEN\n");
  fprintf(fd,"c     Use the Following RHS for the Solution of Linear System\n");
#   fprintf(fd, "          print*, 'The System of Equations is Linear'\n");
#   fprintf(fd, "          print*, 'Newton-Raphson Iteration is Not Used'\n\n");
```

```
  fclose(fd);
#   Fortran(b, resultname = "bLin", output = outFile);
  Fortran(b, output = outFile);
  fd := fopen(outFile, APPEND);
  fprintf(fd, "       \n\n");
  fprintf(fd, "      END IF\n\n");
  fclose(fd);
end if;

# Write out direct assignments to fortran
if (L <> 0) then
   Fortran(XLDot, output = outFile);
end if;

fd := fopen(outFile, APPEND);
fprintf(fd, "       \n\n");

# Generate XHLDot for Sensitivity Analysis
if (L <> 0) then
 for i from 1 by 1 to L do
    for j from 1 by 1 to N do
          if (XLDot[i] = XH[j]) then
             for k from 1 by 1 to Nd do
```

```
            fprintf(fd,"          XHLDot(%d)  =  XHL(%d)\n",  (i+N)+(N+L)*k,
                    j+(N+L)*k);
          end do:
        end if:
    end do:
    for j from 1 by 1 to L do
        if (XLDot[i] = XL[j]) then
          for k from 1 by 1 to Nd do
            fprintf(fd,"          XHLDot(%d)  =  XHL(%d)\n",  (i+N)+(N+L)*k,
                    (j+N)+(N+L)*k);
          end do:
        end if:
    end do:
  end do:
end if;

fprintf(fd, "\n     RETURN\n\n");
fprintf(fd, "\n     END\n\n\n\n\n\n");
fclose(fd);
# ------------------------------ END getMat -------------------------------------

# --------------------------BEGIN getXd-----------------------------------------
fd := fopen(outFile, APPEND);
fprintf(fd, "     SUBROUTINE getXd(neq, t, XHL, XHLDot)\n\n");
if (((N+L) <> 0)and(M > 0)) then
```

```
   fprintf(fd, "          REAL*8 t, XHL(%d),  b(%d), Y(%d), XHLDot(%d),
         qPast(%d,%d)\n", (N+L)*(Nd+1), N+M, M*(Nd+1), (N+L)*(Nd+1),
         (N+L+M)*(Nd+1), nHist);

   fprintf(fd, "          REAL*8 saveXHLDot(%d), delY(%d), delXHDot(%d)\n",
         (N+L)*(Nd+1), M, N);

   fprintf(fd, "      REAL*8 br(%d,%d), bx(%d,%d), bd(%d,%d), d(%d), q(%d)\n",
         (N+M), Nd+1, (N+M), (N+L), (N+M), Nd+1, Nd+1, N+M);

else

   fprintf(fd, "          REAL*8 t, XHL(%d),  b(%d), Y(%d), XHLDot(%d),
         qPast(%d,%d)\n", (N+L)*(Nd+1)+1, N+M, M*(Nd+1)+1,
         (N+L)*(Nd+1)+1, (N+L+M)*(Nd+1), nHist);

   fprintf(fd, "          REAL*8 saveXHLDot(%d), delY(%d), delXHDot(%d)\n",
        (N+L)*(Nd+1)+1, M+1, N+1);

   fprintf(fd, "      REAL*8 br(%d,%d), bx(%d,%d), bd(%d,%d), d(%d), q(%d)\n",
         (N+M), Nd+1, (N+M), (N+L)+1, (N+M), Nd+1, Nd+1, N+M);

end if;

fprintf(fd, "     REAL*8 RData(10)\n");

fprintf(fd, "          REAL*8 As(%d), Pivot(%d), RHS(%d)\n", nsms, N+M-nqElim,
       N+M-nqElim);

fprintf(fd, "       REAL*8 Asprv(%d), Asnxt(%d), rScale(%d), D1A(%d)\n", nzcnt,
       nzcnt, nzcnt, nzcnt);

fprintf(fd, "     REAL*8 D1prv(%d), D1nxt(%d), D2prv(%d), D2nxt(%d)\n", N+M-
       nqElim, N+M-nqElim, N+M-nqElim, N+M-nqElim);

fprintf(fd, "          REAL*8 DR(%d), DRinv(%d), DC(%d), DCinv(%d)\n\n",N+M-
       nqElim, N+M-nqElim, N+M-nqElim, N+M-nqElim);

fprintf(fd, "          INTEGER IPVT(%d), SNR(%d), RNR(%d), HA(%d,11), Iq(%d),
       linFlag\n", N+M-nqElim, nsms, nsms, N+M-nqElim, N+M);
```

```
fprintf(fd, "      INTEGER IData(10)\n\n");
fprintf(fd, "          COMMON/SIZES/N, M, L, nHist, nSC, lrw, liw, mf, nqElim,
        Nd\n");
fprintf(fd, "      COMMON/SAVEXHLDOT/saveXHLDot\n");
fprintf(fd, "      COMMON/NEWTONDIFF/delXHDot, delY\n");
fprintf(fd, "      COMMON/ALGVAR/Y\n");
fprintf(fd, "      COMMON/SIZESMS/nzcnt, nsms\n");
fprintf(fd, "      COMMON/FLAGS/linFlag, Lflag\n");
fprintf(fd, "      COMMON/USERDATA/IData, RData\n");
fprintf(fd, "      COMMON/REDUCTION/Iq, q, RHS\n");
fprintf(fd, "      COMMON/DESIGN/br, bx, bd, d\n");
fprintf(fd, "      COMMON/SCALE/Asprv, Asnxt, D1prv, D1nxt, D2prv, D2nxt,\n");
fprintf(fd, "     +           DR, DRinv, DC, DCinv, rScale, D1A\n\n");

fprintf(fd, "      linFlag = %d\n\n", linFlag);
fclose(fd);

if (nqElim<>0) then
   Fortran(Iq, output = outFile);
end if;

fd := fopen(outFile, APPEND);
fprintf(fd, "      \n\n");
if ((N+L <> 0)) then
```

```
   fprintf(fd, "      DO 10 I = 1, (N+L)*(Nd+1)\n");
   fprintf(fd, "            XHLDot(I) = saveXHLDot(I)\n");
   fprintf(fd, "   10 CONTINUE\n\n");
end if;

fprintf(fd, "      CALL calcXdNL(t, As, b, XHL, XHLDot, Y, IPVT,\n");
fprintf(fd,"     +    delXHDot, delY, SNR, RNR, HA, Pivot, br, bx, bd, d, \n");
fprintf(fd,"     +    linFlag, Lflag, Iq, q, RHS, Asprv, Asnxt, D1prv, \n");
fprintf(fd,"          +       D1nxt, D2prv, D2nxt, DR, DRinv, DC, DCinv, rScale,
        D1A)\n\n");

if ((N+L <> 0)) then
   fprintf(fd, "      DO 20 I = 1, (N+L)*(Nd+1)\n");
   fprintf(fd, "            saveXHLDot(I) = XHLDot(I)\n");
   fprintf(fd, "   20 CONTINUE\n\n");
end if;

fprintf(fd, "\n      RETURN\n\n");
fprintf(fd, "\n      END\n\n\n\n\n\n");
fclose(fd);

# -------------------------- END getXd ----------------------------------------

# -------------------------- BEGIN calcConstraints ----------------------------
```

```
# Generate Fortran Subroutine calcConstraints from MAPLE

# This subroutine calculates constaints on the side

fd := fopen(outFile, APPEND);

fprintf(fd, "      SUBROUTINE calcConstraints(t, XH, XL, Y, XHDot, XLDot,
      SC)\n\n");

if (((N) <> 0) and (L <> 0) and (M <> 0)) then

   fprintf(fd, "      REAL*8 t, XH(N), XL(L), Y(M), SC(nSC), XHDot(N),
         XLDot(L)\n");

else

   fprintf(fd, "      REAL*8 t, XH(N+1), XL(L+1), Y(M+1), SC(nSC),
         XHDot(N+1),XLDot(L+1)\n");

end if;

fprintf(fd, "      REAL*8 RData(10)\n\n");

fprintf(fd, "      INTEGER IData(10)\n\n");

fprintf(fd, "      COMMON/SIZES/N, M, L, nHist, nSC, lrw, liw, mf, nqElim,
      Nd\n");

fprintf(fd, "      COMMON/USERDATA/IData, RData\n\n");

fclose(fd);

Fortran(SC, output = outFile);

fd := fopen(outFile, APPEND);

# Subroutine calConstraints ends here

fprintf(fd, "\n      RETURN\n");

fprintf(fd, "\n      END\n\n\n\n\n\n");

fclose(fd);

# -------------------------- END calcConstraints --------------------------------
```

```
# -------------------------- BEGIN restoreVariables --------------------------------

# Generate Fortran Subroutine restoreVariables from MAPLE
# This subroutine restores the original variable array q in the same order
# as it was formed after going through the 2-pass scan
# bb is the full right hand side vector

fd := fopen(outFile, APPEND);
fprintf(fd, "      SUBROUTINE restoreVariablesNL(t, XH, XL, q, bb, d)\n\n");
if ((N <> 0) and (L <> 0) and (M <> 0)) then
   fprintf(fd, "            REAL*8 t, XH(N), XL(L), q(N+M), bb(N+M-nqElim),
         d(Nd+1)\n");
else
   fprintf(fd, "          REAL*8 t, XH(N+1), XL(L+1), q(N+M), bb(N+M-nqElim),
         d(Nd+1)\n");
end if;
fprintf(fd, "      REAL*8 RData(10)\n\n");
fprintf(fd, "      INTEGER IData(10)\n\n");

fprintf(fd, "         COMMON/SIZES/N, M, L, nHist, nSC, lrw, liw, mf, nqElim,
       Nd\n");
fprintf(fd, "      COMMON/USERDATA/IData, RData\n\n");
fprintf(fd, "      IF (nqElim.eq.0) RETURN\n\n\n\n\n");

for i from nqElim by -1 to 1 do
```

```
    fclose(fd);
    Fortran(q[IqElim[i]], resultname = "temp", output = outFile);
    fd := fopen(outFile, APPEND);
    fprintf(fd, "      q(%d) = temp\n\n", (IqElim[i]));
end do:

fprintf(fd, "\n      RETURN\n");
fprintf(fd, "\n      END\n\n\n\n\n\n");
fclose(fd);

# -------------------------- END restoreVariables --------------------------------

# -------------------------------- BEGIN getBxBd --------------------------------
# Generate the subroutine getBxBd to write br, bx and bd to Fortran
fd := fopen(outFile, APPEND);
fprintf(fd, "      SUBROUTINE getBxBd(t, XH, XL, XHDot, Y, d, bx, bd)\n\n");
if ((N <> 0) and (L <> 0) and (M <> 0)) then
   fprintf(fd, "      REAL*8 d(Nd+1), bx((N+M),(N+L)), bd((N+M),Nd+1)\n");
   fprintf(fd, "      REAL*8 t, XH(N), XL(L), XHdot(N), Y(M)\n");
else
   fprintf(fd, "      REAL*8 d(Nd+1), bx((N+M),(N+L+1)), bd((N+M),Nd+1)\n");
   fprintf(fd, "      REAL*8 t, XH(N+1), XL(L+1), XHdot(N+1), Y(M+1)\n");
end if;
fprintf(fd, "      REAL*8 RData(10)\n\n");
```

```
fprintf(fd, "      INTEGER IData(10)\n\n");
fprintf(fd, "          COMMON/SIZES/N, M, L, nHist, nSC, lrw, liw, mf, nqElim,
        Nd\n");
fprintf(fd, "      COMMON/USERDATA/IData, RData\n\n");
fprintf(fd, "      IF (Nd.eq.0) RETURN\n\n\n\n\n");
fclose(fd);

if (Nd <> 0) then
  Fortran(bx, output = outFile);
  Fortran(bd, output = outFile);
end if:

fd := fopen(outFile, APPEND);
fprintf(fd, "\n      RETURN\n");
fprintf(fd, "\n      END\n\n\n\n\n\n");
fclose(fd);
# ---------------------------------- END getBxBd --------------------------------

# ---------------------------------- BEGIN getRHS --------------------------------
# Write subroutine getRHS to generate RHS from the full b vector
fd := fopen(outFile, APPEND);
fprintf(fd, "      SUBROUTINE getRHS(t, XH, XL, d, bb, RHS)\n\n");
if ((N <> 0) and (L<>0)) then
   fprintf(fd, "      REAL*8 t, XH(N), XL(L)\n");
```

```
else
   fprintf(fd, "      REAL*8 t, XH(N+1), XL(L+1)\n");
end if;

fprintf(fd, "      REAL*8 d(Nd+1), bb(N+M-nqElim), RHS(N+M-nqElim)\n");
fprintf(fd, "      REAL*8 RData(10)\n\n");
fprintf(fd, "      INTEGER IData(10)\n\n");
fprintf(fd, "           COMMON/SIZES/N, M, L, nHist, nSC, lrw, liw, mf, nqElim,
         Nd\n");
fprintf(fd, "      COMMON/USERDATA/IData, RData\n\n");
fclose(fd);

if (nqElim <> 0) then
   Fortran(RHS, output = outFile);
else
   fd := fopen(outFile, APPEND);
   fprintf(fd, "      DO 10 I = 1, (N+M-nqElim)\n");
   fprintf(fd, "           RHS(I) = bb(I)\n");
   fprintf(fd, "   10 CONTINUE\n\n");
   fclose(fd);
end if;

fd := fopen(outFile, APPEND);
fprintf(fd, "\n      RETURN\n");
```

```
fprintf(fd, "\n      END\n\n\n\n\n\n");

fclose(fd);


# -------------------------------- END getRHS --------------------------------
```