8-2001

Simulation of Engineering Systems Described by High-Index DAE and Discontinuous ODE Using Single Step Methods

Marc Compere
University of Texas at Austin, comperem@erau.edu

The dissertation committee for Marc Damon Compere

Certifies that this is the approved version of the following dissertation:

Simulation of Engineering Systems Described by High-Index DAE and Discontinuous ODE Using Single Step Methods

Committee:

Raul G. Longoria, Supervisor

Robert H. Bishop

Benito Fernández-Rodriguez

Glenn Y. Masada

S. V. Sreenivasan

Simulation of Engineering Systems Described by High-Index DAE and Discontinuous ODE Using Single Step Methods

by

Marc Damon Compere, B.S., M.S.

Dissertation

Presented to the Faculty of the Graduate School of

The University of Texas at Austin

in Partial Fulfillment

of the Requirements

for the Degree of

Doctor of Philosophy

The University of Texas at Austin

August 2001

To the God of the Christian bible and my parents Mark and Jackie Compere.

Acknowledgments

I would like to acknowledge the people whose encouragement and help made the past 5 years a great experience. Many thanks to Dr. Raul Longoria, who was an excellent advisor and manager throughout the process and was one who gave timely, insightful advice. Many thanks also go to Dr. Benito Fernandez whose encouragement early in my degree and insights into sliding mode control got me started down the road to completion. I extend thanks to Dr. David Thompson for patiently teaching me numerous aspects of Linux, Unix, C, and C++ during my time at UT. I would like to thank both Larry Lucas at the Army's AMCOM, Huntsville Alabama and Dr. Bill Prescott at LMS-International, Corallville Iowa, for discussions on challenging practical problems that motivated much of this dissertations development. Also, many thanks go to my parents who gave unwavering and complete support throughout my academic and personal pursuits. Lastly, I would like to acknowledge the sources of funding for this research, both from the National Science Foundation (NSF grant numbers # EEC-9706083 and #EEC-9815637) and the U.S. Army.

Marc Damon Compere

The University of Texas at Austin

August 2001

Simulation of Engineering Systems Described by High-Index DAE and Discontinuous ODE Using Single Step Methods

Publication No. ____________________

Marc Damon Compere, Ph.D.
The University of Texas at Austin, 2001

Supervisor: Raul G. Longoria

This dissertation presents numerical methods for solving two classes of ordinary diferential equations (ODE) based on single-step integration methods. The first class of equations addressed describes the mechanical dynamics of constrained multibody systems. These equations are ordinary differential equations (ODE) subject to algebraic constraints. Accordinly they are called differential-algebraic equations (DAE).

Specific contributions made in this area include an explicit transformation between the Hessenberg index-3 form for constrained mechanical systems to a canonical state-space form used in the nonlinear control communities. A hybrid solution method was developed that incorporates both sliding-mode control (SMC) from the controls literature and post-stabilization from the DAE related literature. The process of developing the hybrid method produced insights into both areas in a way that allowed both areas to benefit from the other's strengths. First, the hybrid method produced an accurate and efficient method for simulating sliding-mode control systems. A technique called post-stabilization provides a more efficient method for simulating SMC systems than conventional methods using the discontinuous control term. Second, use of SMC mathematical framework allows the hybrid method to handle arbitrary, or inconsistent initial conditions.

The second class of equations addressed here are discontinuous ODE. Specific contributions made in solving DODE include further classification of discontinnuities into parametric or structural discontinuities as well as unilateral or bilateral events. Consistent event location and discontinuity sticking from Park and Barton[56] originally addressed bilateral events only and were implemented in a single-step environment and then extended to address unilateral events as well. An effective detection scheme was developed using low-order interpolants for detecting most events in the correct order. For rare cases when the detection scheme fails, a try-catch model was implemented to deal with two possible failure scenarios. The detection and location methods successfully handled all events in the correct order for the benchmark problems solved. Lastly, a region of concurrency was developed that can provide large efficiency gains for some systems containing multiple closely spaced events.

Keywords:

multi-body systems, holonomic, non-holonomic, high-index differential-algebraic equations, inconsistent initial conditions, sliding-mode control, simulation, post-stabilization, discontinuous ODE, single-step methods, Runge-Kutta, detect-locate-restart, consistent event location, unilateral and bilateral discontinuities

Contents

List of Tables

List of Figures

Chapter 1

Introduction

Simulation has grown to become an indispensible tool in the sciences, engineering, and operations research. As defined by Bennett[28], simulation is a technique or set of techniques used to examine the dynamical behavior of abstract models. This can include anything from predicting where a rover will touch down on the surface of Mars, to estimating micron-scale thicknesses using obscure sensor information, to performing virtual surgery on a patient prior to the actual procedure. Although simulation was brought to the public's awareness by the space race in the late 1960's, widespread *use* of simulation is still limited by factors such as availability, low awareness of it's capabilities and high cost-to-benefit ratios. Even within technical communities where simulation awareness and availabilty are high, the costs are still significant, requiring some combination of modeling, numerical methods, problem solving and troubleshooting expertise.

This dissertation research is aimed at reducing the total cost in the simulation process by addressing one facet of the overall simulation process, namely solution methods for solving two classes of ordinary differential equations (ODE). The first solution method addresses differential-algebraic equations (DAE)[3, 8] that model constrained multibody dynamic systems. The second focuses on the general problem of solving discontinuous ODE (DODE)[28, 30]. Examples of both DAE and DODE are found in models of mechanical, electrical, hydraulic, thermal, and magnetic systems. The focus in solving DAE is clearly on constrained mechanical systems however, the solution method is intended to address a wider class of engineering systems. Besides constrained mechanical systems, it is useful for lumped-parameter models of constrained electrical, hydraulic, and chemical systems that can be cast into the standard 2^{nd}-order form found in both [3] and (2.1)-(2.4). In addition to lumped-parameter modeling, to the extent that certain classes of PDE can be reduced to ODE, the equations describing distributed-parameter systems are also candidates for solution with either of these methods. So long as the equations are in the correct form, the usefulness of these solution methods is limited neither by the formulation used to generate the equations nor the system(s) they describe.

1.1 Solution methods overview

The solution methods developed are based on single-step numerical integration techniques, namely Runge-Kutta [7, 8] methods. Runge-Kutta integrators can take relatively large steps compared to multi-step integrators and produce fully high-order state estimates. They can also accommodate arbitrary stepsize changes from step to step whereas multistep integrators initially require a special startup procedure to achieve full order and cannot accept arbitrary stepsize changes without cost.

The first class of equations addressed result from some multibody dynamics formulations used to model constrained mechanical systems. Within the total simulation process (Fig. 1.1), equation formulation deals primarily with how to *generate* equations. This is a completely separate task from finding solution(s) to, or solving those equations. Some formulations favor efficiency over generality and go to great lengths to produce a minimal set of ODE. Other formulations favor generality and scope of systems able to be addressed. These typically result in a combined set of unconstrained ODE along with algebraic constraints, or differential-algebraic (DAE) equations. The ODE often represent unconstrained dynamics and the algebraic equations represent physical constraints such as pin, prismatic, rolling, or bracket-type joints. These differential-algebraic (DAE) equations are significantly more challenging to solve than ODE and require specicial consideration to ensure the algebraic constraints are satisfied at each timestep. Imposing constraints on an un-

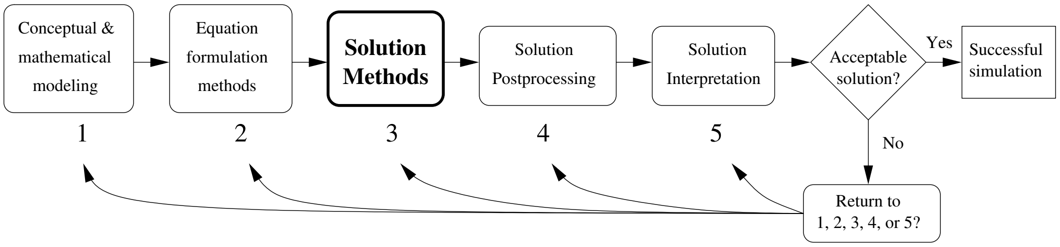


Figure 1.1: The general equation-based simulation process.

constrained system will change the permitted motion and theoretically the equations of motion can typically be reduced to a single set of ODE. However, because of a system's topology or the desire to maintain a generalized modeling environment, analytic reduction of all dynamics and constraints into a single set of ODE is often impractical and sometimes impossible. As a result, there are several techniques for both generating and solving constrained dynamics equations that implement some combination of analytic and numerical techniques to solve the total set of DAE. The DAE solution method presented here relies on analytic techniques borrowed from the computational mathematics based literature combined with the framework and additional features borrowed from the sligind-mode control literature. An explicit transformation is presented between constrained multibody dynamics equations in the mathematics-based form and the canonical control-theoretic form. Both areas of specialization are shown to benefit from the connection between the DAE literature and the sliding mode control literature.

The second class of equations addressed here are discontinuous ODE. In modeling physical systems, discontinuities typically arise from simplified or macroscopic models of otherwise complicated interactions. Discontinuous events produce sharp, nonsmooth changes in state trajectories. The underlying theory on which most modern ODE (or DAE) solvers are based assumes continuity and smoothness. As a result, when discontinuous changes occur the methods perform poorly and the results can be less accurate or less efficient than intended. Often in physical system models, discontinuous events are predictable functions of state or time. In this situation, or others where some function is available to help locate events, this information can be used to improve solver performance. The second half of this dissertation incorporates this information into an ODE solver so that discontinuous events may be traversed in a way that is more efficient and accurate than a standard smooth solver. To improve efficiency, this method considers each right-hand-side (RHS) function evaluation as computationally expensive and attempts to economize on the total number of RHS evaluations. Currently, other DODE solvers also attempt to solve discontinuous ODE with the smallest possible number of RHS evaluations. To improve accuracy, this method is equipped with event location capabilities that can locate events down to machine tolerance. This tolerance is independent of the integration tolerance so high-accuracy event location is available although not required. Both the event location and integration tolerances can be adjusted as necessary to provide a tradeoff between cost and efficiency. In comparison, some multi-step DODE solvers exploit the internally generated interpolants from the underlying ODE (or DAE) solver. This not only means the method is dependent upon a particular integrator, but also that integration could be adversely affected by the event location tolerance.

1.2 Motivation

A generalized multibody dynamics software environment is used here as a motivational example for improvements in the equation-based simulation process. Modern multibody dynamics codes are capable of formulating equations for large-scale systems (e.g. hundreds of rigid bodies) and, in theory, are limited only by computational resources. However, in practice, they are frequently limited by solution times of hours or days rather than computer resources. In these cases it is apparent that the dominant simulation bottleneck lies in the computer solution of ODE or DAE. Generally, some significant factors that influence this bottleneck include programming language and associated memory management, the underlying hardware, and the solution method(s). Several identifiable areas of improvement specifically for multibody dynamics codes include equation formulation, topology exploitation, utilization of multiple processors, linear equation solver and sparse matrix techniques, and inefficiencies in solution methods for stiff discontinuous differential equations. This research presents algorithms designed to improve efficiency and accuracy associated with solving systems of DAE and (separately) discontinuous ODE. The equations addressed and solution methods developed are intended for use within the framework of a large-scale multibody dynamics environment.

1.3 Contributions

As previously stated, the high cost-to-benefit ratio is still the limiting factor in simulation's use and the aim of this dissertation is to reduce some of the costs. Before addressing the numerical methods in detail, some background is presented to identify the costs of simulation to provide a framework in which to pinpoint the contributions made by this research. Some benefits of simulation are first presented as motivation for continued research in the area of simulation methods. Next, the particular subclass of models addressed by this research is presented along with the general simulation process for equation-based simulation. Lastly, the general simulation process is further expanded into some well-known techniques that more clearly locate the research contributions of this dissertation in the scope of related simulation practice.

1.3.1 Benefits of simulation

Bennet[28] argues that one of the great strengths of simulation is it's use for examining systems that do not necessarily exist - a kind of "virtual prototyping." Although simulation is primarily a means for analysis, its ability to analyze virtual systems makes is an ideal tool for the design process. Some examples of how simulation is beneficial to the design process are:

- In product design, simulation allows the designer to test options or evaluate "what if" scenarios.
- In control system design, simulation allows the control engineer to evaluate system performance with various controllers, plant variations, or external disturbances.
- In safety system design, simulation allows the designer to test safety systems virtually without requiring the real emergency (e.g. a nuclear power plant safety system.)

Other examples that illustrate how simulation is beneficial for systems analysis:

- Simulation aids analysis for predicting or recreating failures.
- Simulation can be used to estimate or reconstruct quantities not directly or easily measureable (e.g. heat flux or internal stress.)
- Simulation can be used to predict a system's future behavior (e.g. the weather in meterology.)
- Simulation can help evaluate the impact of system performance from an upgrade or retrofit.

1.3.2 The general simulation process

The subclass of models addressed by this research are equation-based, deterministic, continuous-variable, dynamic models. This is in contrast to block oriented (i.e. those designed for analog computers), stochastic, discrete-variable, or static models. The general equation-based simulation process is shown in Fig. 1.1.

This research aims to reduce the costs associated with the "solution methods" box in Fig. 1.1 by improving both accuracy and efficiency when solving DAE and DODE. It also addresses the **"solution interpretation"** box in Fig. 1.1 by presenting an energy-based metric for assessing solution quality. This has proven to be an invaluable aid in identifying and troubleshooting problems during both equation and solution method development. An energy metric can help answer the question "Is this an acceptable solution?" After arriving at a solution, asking the questions, "Is this solution reasonable?", or "Is there a way to tell how close this solution is to the right answer?" is good simulation practice but is often difficult to answer. Although somewhat arbitrary, an energy metric can help answer these questions by providing a means by which to assess a solution's quality. Again, this arbitrary "quality" may be in reference to an expected theoretical result or an anticipated energy variation based on integration tolerance. Certainly, easy-to-see or "large" variations such as unexpected spikes in an energy function is useful for troubleshooting during a simulation's development.

As mentioned above, the focus of this research is on these two solution methods and is somewhat independent of how, or from what system, the equations were developed. For example, if a finite-element package generates DAE or DODE in the form presented in chapters 2 or 5, then these solution methods are just as applicable as if the equations were developed from a Bond-Graph model of an electro-mechanical-hydraulic system or a rigid multi-body dynamics system. Fig. 1.1 helps clarify the functional components of equation-based simulation and highlights the distinction between equation formulation and equation soltuion methods. Fig. 1.2 shows the boxes from Fig. 1.1 expanded with an incomplete list of techniques in multibody dynamics simulation to further delineate the areas in which this dissertation makes contributions.

1.4 Dissertation organization

Chapter 2 introduces the multibody dynamics DAE problem along with a literature review, including a review of some control theoretic approaches to solving these DAE. Chapter 3 reviews some introductory concepts in sliding

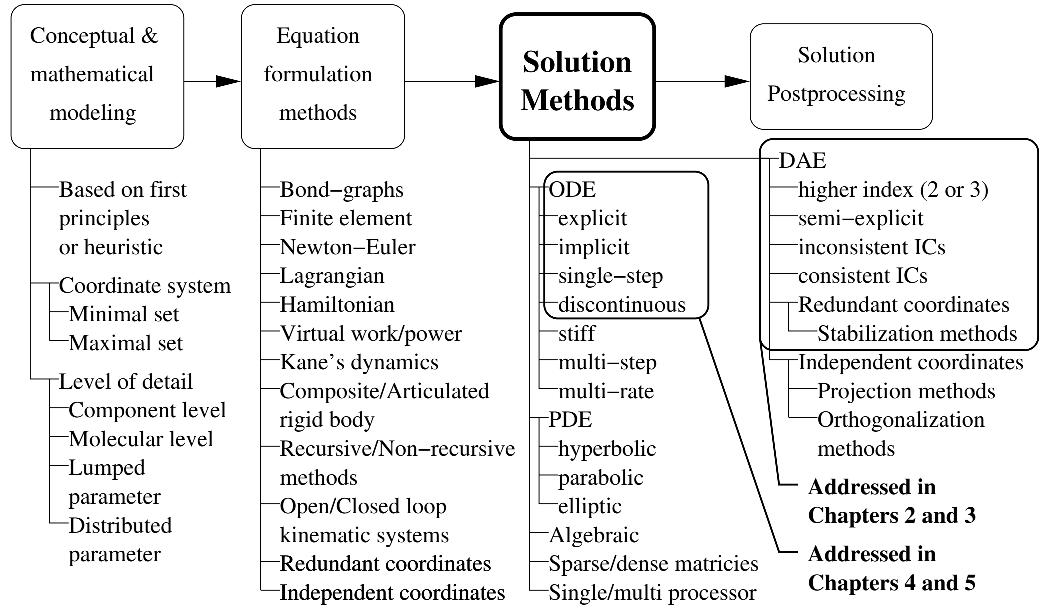


Figure 1.2: Some well known techniques in the multibody simulation process.

mode control theory and then presents the multibody dynamics DAE problem as a control problem. It also presents the hybrid solution method using techniques from computational mathematics and sliding mode control theory. Benchmark problems from the literature are used to demonstrate the new method's effectiveness as well as demonstrate the potential shortcomings of post-stabilization which is a DAE-based solution technique. Chapter 4 introduces the discontinuous ODE problem along with a review of the literature in that area. Chapter 5 presents some new characterizations of DODE, some extensions of previous work, and a new method based on single-step integration methods. Benchmark problems are presented with enough information to compare efficiency and accuracy against results from the literature. Chapter 6 presents conclusions from both solution methods, and Chapter 7 draws the two somewhat unrelated solution methods together to form a vision for future work on a discontinuous high-index DAE solver.

Chapter 2

Background on Control Theory for Multibody Dynamics Simulation

2.1 Introduction

Many large scale multibody dynamics software packages have equation formulations that lead to higher index DAE. Since DAE are composed of some differential equations and some algebraic equations, a natural approach to solving them might be to differentiate the algebraic constraints until an explicit set of ODE were reached. Assuming the DAE in question are solvable with this approach (see [3] and [8] for restrictions) the *index* is the number of differentiations required to convert the DAE into an explicit set of ODE. Higher-index DAE are those with index ≥ 2 and contain some "hidden", or implied constraints. Solution methods for these equations have recently received much attention in the literature. Several well established methods for solving DAE are constraint stabilization methods, variational methods, and state-space methods. For a complete review, see [1, 2, 3, 4, 5] and the references therein.

There are two basic approaches to equation formulation of multibody dynamic systems (MBS) with both holonomic and nonholonomic constraints. The first approach analytically eliminates the constraint forces and generates a minimal set of ODE. However, this contributes to the numerical instability of direct integration [2, 3, 6].

The second approach explicitly leaves all constraint forces in the equations and attempts to find forces such that the constraints are satisfied. This approach results in semi-explicit, higher index differential algebraic equations (DAE) [2, 8, 9]. Yun and Sarkar [10] present a thorough literature review on solving these types of systems and develop a unified state-space treatment for systems with both holonomic and nonholonomic constraints. Their solution approach explicitly solves for the forces that drive the constraints to satisfaction. This is analogous to the control problem where inputs to a system are found that regulate the output (i.e. the constraints) to zero.

Using a control theory framework to solve DAE systems has been reported in the literature for some time. Baumgarte's constraint stabilization method [4] can be viewed as application of a classical PD controller to the constraint dynamics. Besides the difficulty in choosing his PD controller coefficients (α,β) a common negative aspect of all stabilization methods whose action relies on constraint violation is the presence of constraint dynamics. By allowing constraint dynamics, the original problem is changed into a new problem with more degrees of freedom and may yield a significantly different solution. In particular, use of linear control theory ensures the presence and perpetuation of constraint dynamics. Admittedly, the effects of these dynamics may be reduced by forcing them to have time-constants much smaller than the original DAE, however this may, in turn, add unwanted numerical stiffness to the problem. More recently, McClamroch developed a theoretical framework for feedback control of smooth systems described by nonlinear DAE and proves that a smooth solution exists [11, 12]. However, he is careful to point out that showing the equivalence of DAE to controlled ODE is not the same as actually *developing* the control inputs required to maintain the constraints at zero. Chiou and Wu [13] use input-output feedback linearization to transform the nonlinear DAE into a set of linear equations. However, their constraint violation stabilization technique introduces fictitious constraint dynamics. Assuming a consistent set of initial conditions, or ICs, the introduction of constraint violation dynamics can be avoided through sliding-mode control (SMC). In addition, SMC's accommodation for reaching-phase dynamics eliminates the requirement for consistent ICs (Fig. 3.2). Gordon, Liu, and Asada present a similar development using SMC to produce a state-space realization of high-index DAE [14]. They make connections between control theory and DAE solution methods and solve the underlying ODE with singular-perturbation methods along with both classical discontinuous SMC inputs and SMC with a boundary layer. The singular-perturbation approach converges to the exact solution with a residual error. However, the amount of error can be significant and a lower bound on the possible error achievable through this method is unclear. The method presented here combines similar theoretical development using SMC as a formalism for defining invariant manifolds composed of linear combinations of the constraints and their derivatives. However, the difference lies in the numerical solution of the underlying ODE. While one of their solution methods carefully considers the computational costs involved and aims to lower that cost through use of a switching input, as long as a SMC input is used to drive s to zero, the lower bound on constraint violation is limited to that of the integrator, $O(h^p)$. The solution method presented here can provide constraint satisfaction to $O(h^{2(p+1)})$ using post-stabilization which, for even a modest integrator order of $p = 4$ and $h = 0.01$, can easily solve the constraints to machine tolerance (i.e. as good or better than $O(10^{-16}.)$

Zhao and Utkin present a paper that develops a step-by-step algorithm for simulating SMC systems [15] using a Newton-Raphson technique. Their method performs well for single-input systems and eliminates the chattering caused by discretization of continuous SMC theory (i.e. numerical integration.) However for systems with m inputs and constraints, it requires m integration steps to create a rank-m $\frac{ds}{du}$ gradient matrix. For systems with many constraints this method quickly becomes prohibitively costly.

This chapter presents a combination of SMC and other stabilization methods to solve smooth constrained mechanical system dynamics. The underlying problem to be solved in simulating mechanical dynamic systems is to find the numerical solution to an initial value problem (IVP)[3, 7, 8]. For the class of equations addressed in these twp chapters, the challenge is to find some inputs to a dynamic system such that the constraints will be satisfied. The system motion is discretized along the independent (time) axis and the goal of computational dynamics is to find a discretization that is both sufficiently accurate and reasonably efficient. The use of linear control theory for solving DAE systems can potentially provide the accuracy but at the sacrifice of efficiency. For example, stiff springs and dampers may be used to satisfy constraints, however this introduces numerical stiffness[8] which causes other problems. Relaxing the stiffness and damping produces constraints that are not guaranteed to be fully satisfied. In fact, the linear control theory itself ensures the presence of constraint dynamics with it's own rise-time, overshoot, etc. The presence of constraint dynamics and the inability to guarantee nonlinear constraint satisfaction with linear control theory presents a need for a more sophistocated approach. SMC is applied because of its ability to address holonomic and nonholonomic systems simultaneously, as well as being able to address multiple-input/multiple-output (MIMO) nonlinear systems without resorting to approximations or other simplifying assumptions [16]. A switching surface is chosen as a function of the constraints and a smooth control input, u_{eq}, is presented that defines the surface as an invariant manifold [17, 18]. A hybrid stabilization method is depicted in Fig. 2.1 that eliminates chattering commonly found in simulation of discontinuous SMC systems [15, 19].

For consistent ICs, $u = u_{eq}$ and post-stabilization is used throughout the interval $[t_{initial} \;\; t_{final}]$. However, for inconsistent ICs, $u_i = u_{eq,i} + u_{robust,i}$ where $u_{robust,i} = -\eta_i \cdot sgn(s_i)$ until s_i reaches zero (i.e. $|s_i| \leq O(h^p)$). Afterward,

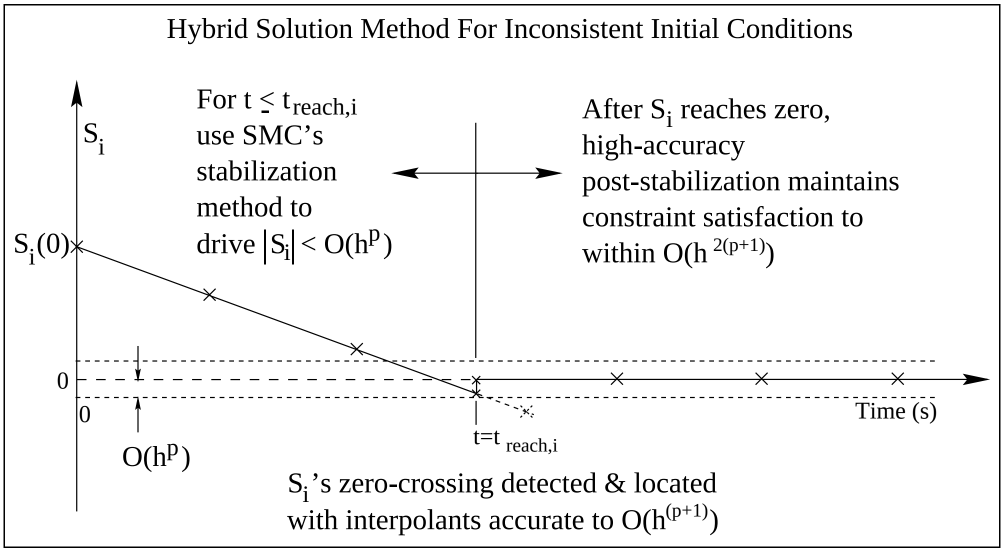

Hybrid Solution Method For Inconsistent Initial Conditions
S_i
For t ≤ t_reach,i use SMC's stabilization method to drive |S_i| < O(h^p)
After S_i reaches zero, high-accuracy post-stabilization maintains constraint satisfaction to within O(h^2(p+1))
S_i(0)
0
0
O(h^p)
t=t reach,i
Time (s)
S_i's zero-crossing detected & located with interpolants accurate to O(h^(p+1))

Figure 2.1: Time intervals of stabilization methods

$u_i = u_{eq,i}$ and post-stabilization is used for that surface over $[t_{reach,i} \quad t_{final}]$. Each surface's reaching time is located accurately with Hermite-Birkhoff interpolants [20] signaling the transition between stabilization methods. The theory behind post-stabilization suggests a new way of using SMC boundary layer dynamics. This idea is developed into an effective acceleration-level stabilization method that used over the same time intervals as post-stabilization.

2.2 Multibody dynamics DAE

We seek the solution of semi-explicit, index-3, DAE given by,

$$\dot{q} = v \tag{2.1}$$

$$M(q)\dot{v} = D + J^T\lambda \tag{2.2}$$

$$0 = \Phi_n(q, v) \tag{2.3}$$

$$0 = \Phi_h(q), \tag{2.4}$$

The system is described by N position coordinates, $q = [q_1, q_2, ..., q_N]^T$, and N velocity coordinates, $v = [v_1, v_2, ..., v_N]^T$. $M(q)$ is an invertible, position-dependent mass matrix $\in \Re^N$. D contains position and velocity dependent terms along with any system inputs, τ, and is given by,

$$D = E(q)\tau - V(q, v) - G(q). \tag{2.5}$$

The functions $V(q, v)$ and $G(q)$ are both $\in \Re^N$, and represent velocity and position, or gravity dependent terms, respectively. The external input τ is $\in \Re^p$ representing actuator torques (or forces), with $E(q) \in \Re^{p \times N}$ mapping the actuator space into the (q,v) coordinate space. Φ_n represents m_n independent, first-order, non-integrable, nonholonomic constraints,

$$\Phi_n \equiv [\phi_{n,1}(q, v), \phi_{n,2}(q, v), ..., \phi_{n,m_n}(q, v)]^T = 0, \tag{2.6}$$

and Φ_h represents m_h independent holonomic constraints,

$$\Phi_h \equiv [\phi_{h,1}(q), \phi_{h,2}(q), ..., \phi_{h,m_h}(q)]^T = 0. \tag{2.7}$$

For simplicity, (2.6) and (2.7) are both scleronomic, or time-independent. This assumption is only for notational convenience and not a restriction of the theory developed in the remainder of this chapter [18].

In equation (2.2), J is the combined velocity and position gradients of the velocity- and position-level constraints, namely,

$$J^T = \left[\begin{array}{cc} J_v^T & J_q^T \end{array} \right]. \tag{2.8}$$

In this expression, J_v is the nonholonomic constraint Jacobian, $J_v = \frac{\partial \Phi_n(q,v)}{\partial v} \in \Re^{m_n \times N}$ and J_q is the holonomic constraint Jacobian, $J_q = \frac{\partial \Phi_h(q)}{\partial q} \in \Re^{m_h \times N}$. All constraints are assumed independent, so J is rank-m where $m = m_n + m_h$. Finally, $\lambda^T = [\ \lambda_n^T \mid \lambda_h^T\ ]$ are the Lagrange multipliers associated with the nonholonomic and holonomic constraints, respectively.

The number of ICs of this system is $n = 2N$ which is the number required by the underlying ODE. In addition, the requirement for consistent initialization of the original DAE is not necessary, in contrast to other DAE methods [1, 21, 22].

This chapter opened by presenting two standard methods for formulating constrained multibody dynamics equations. The first results in a minimal set of ODE while the second results in a larger set of DAE. Narrowing the discussion to DAE solution methods, some previous work in solving DAE using techniques from control theory were presented along with their respective strengths and weaknesses. Sliding mode control was presented as a viable candidate from control theory and, again, previous related work was critically reviewed. A hybrid solution method was introduced that combined the mathematical framework and guarantees of SMC with some techniques from the computational mathematics community. Finally, the standard computational mathematics form for constrained multibody dynamics DAE was presented in preparation for an explicit transformation to a standard equation form from the controls community.

Chapter 3

An MBS DAE Solver Using MIMO SMC

This chapter starts by presenting one standard equation form from nonlinear control theory. An explicit transformation is then developed between this form and the standard mathematics based form presented in the last chapter. A typical sliding mode control design procedure is presented along with further explanation of SMC's benefits. Portions of the SMC design procedure are combined with some techniques borrowed from (and inspiried by) the DAE related literature to create the hybrid solution method. Details of the hybrid method are then presented including the SMC switching surfaces and equivalent control, post-stabilization, and an acceleration-level stabilization. Finally, the hybrid method is applied to two example problems with numerical reults.

3.1 Formulation of MBS DAE as a control problem

An affine MIMO nonlinear control system is given by [18],

$$\dot{x} = f(x) + B(x)u(x) \tag{3.1}$$

$$y = h(x) \tag{3.2}$$

$$y_d \equiv some\ desired\ function. \tag{3.3}$$

The vectors $\dot{x}$ and $f(x)$ are both $\in \Re^{n\times 1}$, $B(x)$ is $\in \Re^{n\times m}$, and $u(x)$ and $h(x)$ are $\in \Re^{m\times 1}$. Equations (3.1)-(3.3) are in *companion* form or *control canonical* form [23]. To take advantage of the knowledge base and well developed theory found in the controls field, this section will show how equations (2.1)-(2.4) can be rewritten into an equivalent problem in the form of (3.1)-(3.3).

Assuming n states, each (x_i,x_{i+1}) corresponds to each of the N Cartesian coordinate pairs, (q_j, v_j). It is clear that $n = 2N$ and the $n \times 1$ simulation state vector is,

$$x = [q_1, v_1, q_2, v_2, \ldots, q_N, v_N]^T. \tag{3.4}$$

Introduce two constant rank(N) matrices, e and k, that are both $n \times N$ with structures,

$$e \equiv \begin{bmatrix} 0 & 0 & 0 & 0 & \dots & 0 & 0 \\ 1 & 0 & 0 & 0 & \dots & 0 & 0 \\ 0 & 0 & 0 & 0 & \dots & 0 & 0 \\ 0 & 1 & 0 & 0 & \dots & 0 & 0 \\ 0 & 0 & 0 & 0 & \dots & 0 & 0 \\ 0 & 0 & 1 & 0 & \dots & 0 & 0 \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & 0 & 0 & \dots & 0 & 0 \\ 0 & 0 & 0 & 0 & \dots & 1 & 0 \\ 0 & 0 & 0 & 0 & \dots & 0 & 0 \\ 0 & 0 & 0 & 0 & \dots & 0 & 1 \end{bmatrix}, \qquad k \equiv \begin{bmatrix} 1 & 0 & 0 & 0 & \dots & 0 & 0 \\ 0 & 0 & 0 & 0 & \dots & 0 & 0 \\ 0 & 1 & 0 & 0 & \dots & 0 & 0 \\ 0 & 0 & 0 & 0 & \dots & 0 & 0 \\ 0 & 0 & 1 & 0 & \dots & 0 & 0 \\ 0 & 0 & 0 & 0 & \dots & 0 & 0 \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & 0 & 0 & \dots & 1 & 0 \\ 0 & 0 & 0 & 0 & \dots & 0 & 0 \\ 0 & 0 & 0 & 0 & \dots & 0 & 1 \\ 0 & 0 & 0 & 0 & \dots & 0 & 0 \end{bmatrix}. \quad (3.5)$$

The relation between the $N \times 1$ position and velocity vectors,

$$q = [q_1, q_2, \dots, q_N]^T, \quad (3.6)$$

$$v = [v_1, v_2, \dots, v_N]^T, \quad (3.7)$$

and the $n \times 1$ state vector, x, is,

$$x = ev + kq. \quad (3.8)$$

Similarly,

$$\dot{x} = e\dot{v} + k\dot{q}. \quad (3.9)$$

Both matrices e and k have the property that their transpose multiplied by themselves is equal to an $N \times N$ identity matrix, $e^T e = k^T k \equiv I_{N\times N}$, and the first transposed and multiplied by the other yields an $N \times N$ zero matrix, $e^T k = k^T e \equiv O_{N\times N}$. Given the above relations, premultiplying equation (3.9) first by e^T then by k^T gives,

$$\dot{v} = e^T \dot{x}, \quad \text{and} \quad \dot{q} = k^T \dot{x}. \tag{3.10}$$

To express equations (2.1) and (2.2) in control canonical form, $\dot{x} = f + Bu$, both are written in terms of $\dot{x}$ then added together to produce $2N$ first-order ODE. First, noting from (3.10) and (2.1) that $k^T \dot{x}$ and v are equal, this relation is premultiplied by k yielding:

$$kk^T \dot{x} = kv. \tag{3.11}$$

Next, equating $\dot{v}$ from (3.10) and (2.2) and premultiplying by e yeilds:

$$ee^T \dot{x} = eM^{-1}D + eM^{-1}J^T\lambda. \tag{3.12}$$

Adding (3.11) and (3.12) produces

$$(ee^T + kk^T)\dot{x} = (eM^{-1}D + kv) + eM^{-1}J^T\lambda \tag{3.13}$$

By observing $(ee^T + kk^T) = I_{n\times n}$, (3.13) reduces to

$$\dot{x} = f + Bu \tag{3.14}$$

where

$$f = (eM^{-1}D + kv) \qquad B = eM^{-1}J^T \qquad u = \lambda$$

Note that the leading matrix B causes the inputs to enter the plant only at the acceleration level. Because accelerations are proportional to external forces, use of acceleration-level inputs only is viewed as a physically realistic approach to constraint satisfaction.

Now, expressing the constraints (2.3)-(2.4) as control system outputs, $y = h(x)$, or,

$$y^T \equiv [\ \phi_n^T \mid \phi_h^T\ ], \tag{3.15}$$

with the desired trajectories all zero,

$$y_d^T \equiv [\ 0, \ldots, 0 \mid 0, \ldots, 0\ ]. \tag{3.16}$$

This section has shown how constrained MBS DAE (2.1)-(2.4) can be expressed in the standard control canonical form (3.14)-(3.16).

Although the connection between control system equations and constrained MBS DAE has been identified [11, 12, 14], this explicit transformation between the two has not been previously presented in the literature.

3.2 Introduce sliding mode control

Sliding mode control (SMC) has it's origins in the Russian literature from the late 1950's and early 1960's. SMC is an attractive control strategy for physical systems that exhibit nonlinear, time-varying, and uncertain characteristics. Not only does it provide performance guarantees for this class of systems, but it is especially attractive here because of the mathematical framework in which it is set. Sliding mode controller design is a two stage process of defining surfaces that represent the desired system behavior and then computing some input that will bring about, and maintain the desired behavior despite uncertainty and disturbances.

3.2.1 Order reduction

The first stage in the design process transforms the general n^{th}-order tracking problem into a 1^{st}-order stabilization problem. This transformation results from the assertion that it is easier to control 1^{st}-order differential equations, whether they be nonlinear or uncertain, than it is to control general n^{th}-order differential equations. Transforming the high-order tracking problem into a 1^{st}-order stabilization problem essentially extracts the relevant error dynamics from the overall system motion and, for MIMO systems, decouples them into individual partial components. At this point conventional control methods may be applied component-wise to maintain each individual surface at zero. For example a PID or optimal control law may be applied to drive all s_i to zero. The reduced-order system to control may be expressed as the total system motion projected onto subspace s,

$$\dot{s} = Gf + GBu \tag{3.17}$$

where $G = \partial s/\partial x$.

The *sliding mode* is a system motion designed by the control engineer and achieved through the control input. Once reached, however, the sliding mode is independent of the input and described by the surfaces themselves. The sliding mode occurs in the state-space at the intersection of m surfaces, $s_i = 0$, and once there, the order of the motion equations is m less than the order of the original system. This is precisely the motion, or solution, sought when solving constrained MBS DAE. Although all m surfaces are at s_i=0, the other $(n - m)$ coordinates are free to move as the dynamics dictate. Since m constraints are successfully imposed on the original n-dimensional system, the constrained system motion is $(n-m)$-dimensional. So, the sliding surfaces not only represent a place but also a dynamics [23].

3.2.2 Design the surfaces

Designing sliding surfaces for the general nonlinear control system in (3.1)-(3.3) may be chosen with intuition and knowledge of the physical system being controlled. More generally, surface design is aided with a definition similar to that given in Slotine,

$$s_i(x,t) = \left(\frac{d}{dt} + \sigma_i\right)^{r_i - 1} \cdot err_i \qquad i = 1 \ldots m. \tag{3.18}$$

The tracking errors are represented by $err_i = y_{d,i} - y_i$ and r_i is the relative order of the i^{th} output. The output's relative order is essentially the number of time-derivatives required of the output for the input to show up. For example, $r = 2$ for a displacement-level output in a typical 2^{nd}-order robotic system since $d^{[2]}(y)/dt^{[2]} = f(\ddot{q})$ and $\ddot{q} = f(q, \dot{q}, u)$. Note that the exponent in (3.18) is $(r_i - 1)$ which defines the surfaces one derivative away from revealing the control input. This is essential for providing convergence and invariance guarantees since it allows the control input to directly influence $\dot{s}$. If $\dot{s}$ can be specified arbitrarily, then s can also be controlled arbitrarily. As in Fig. 3.1, if $s \neq 0$, u can be used to control $\dot{s}$ such that s is driven to zero.

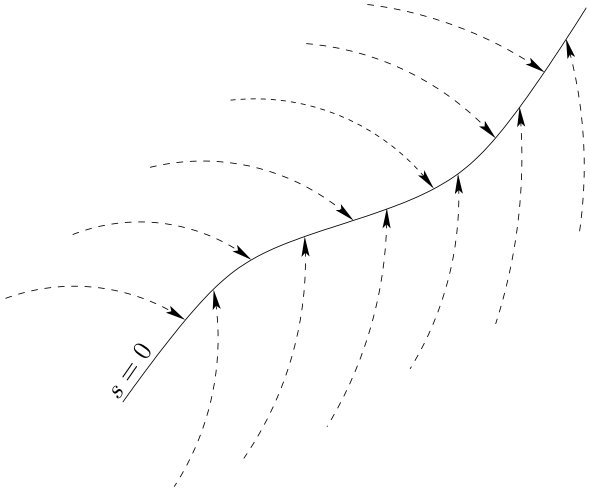


Figure 3.1: The desired dynamics occur at the surface $s = 0$.

3.2.3 Design the control

The second stage in the design process is to choose some discontinuous control input that guarantees the system will achieve and remain in the sliding mode. One way to do this is to solve for u_{eq} and η in the total control input, $u = u_{eq} + \eta \cdot sgn(s)$. Neglecting any uncertainties, the equivalent control term, u_{eq} will maintain $\dot{s} = 0$ and hold s constant. The final step in identifying u is to compute η such that the surfaces are both attractive and invariant. Surface attractiveness assures the sliding mode is achieved within finite time (Fig. 3.2). Finite-time reaching phase is a distinguishing feature of SMC whereas other control strategies might provide asymptotic convergence. Surface invariance ensures all s_i remain at zero once in the sliding mode despite disturbances and/or uncertainty. Invariance is achieved by choosing the "robustness" term η that satisfies the sliding equation:

$$\frac{1}{2}\frac{d}{dt}s_i^2 \leq -\eta_i|s_i| \tag{3.19}$$

The sliding equation can be interpreted as a way to ensure s_i^2 remains a Lyupanov-like function of the closed loop system and that the squared distance from $s_i = 0$ decreases until the sliding mode is reached and maintained. Fig. 3.2 shows the two motion phases achieved by proper choice of η.

3.2.4 Assemble the hybrid method

The hybrid method presented here fully utilizes SMC's first design stage of creating the sliding surfaces, but only partially utilizes the second. In the second design stage, typically an input is derived that ensures the desired behavior is achieved, then the same input is also used to maintain the desired behavior once the sliding mode has been reached. In the hybrid method,

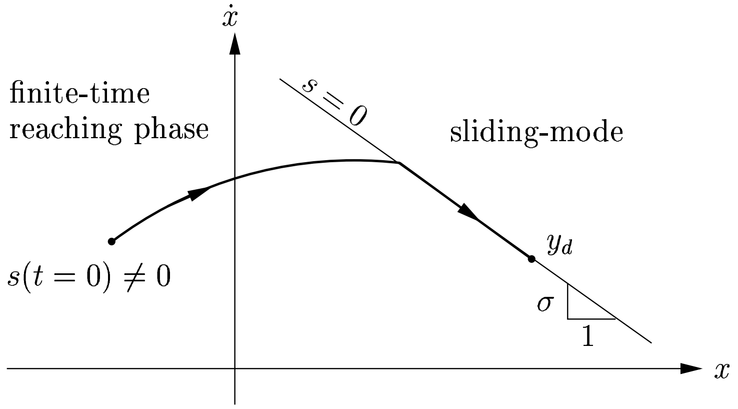


Figure 3.2: Finite-time reaching-phase dynamics for a typical second-order system's displacement output, $y = f(x)$, $r = 2$.

the desired dynamics are achieved in the standard way but afterward, the discontinuous portion of the control input is dropped in favor of a numerical procedure from the DAE-based literature. The discontinuous term is the mechanism that provides SMC's exceptional disturbance rejection and insensitivity to uncertainty. However, in a simulation environment the discontinuous term causes difficulty in numerical integration. It is well known that discontinuities in ODE or DAE cause both inefficiency and inacuracy when solved with most (smooth) integrators. SMC's discontinuous term is perhaps the most disruptive type of discontinuity possible because $\eta \cdot sign(s)$ can change sign at every function evaluation computed by the integrator. The result is often excessive simulation run times once the system achieves the sliding mode.

The hybrid method utilizes η to ensure surface attractiveness, but once each surface is reached (i.e. once each $s_i = 0$), η_i is set to zero and post-stabilization is used to guarantee surface invariance. In DAE terminology, the discontinuous control input $\eta \cdot sgn(s)$ is the SMC equivalent of a stabilization method. For simulation however, post-stabilization from the DAE literature is a much higher accuracy method for enforcing surface invariance than SMC's native stabilization term, $\eta \cdot sgn(s)$.

Lastly, choosing η to guarantee surface attractiveness is similar to DAE some solution methods for finding a consistent set of initial conditions, although it is not a direct analogy. The hybrid method can begin numerical integration with inconsistent initial conditions (i.e. $s_i \neq 0$) and proper choice of η_i will drive the constraints to satisfaction (i.e. $s_i(t_{reach}) = 0$). As previously mentioned, this is guaranteed to occur within a finite time, namely each s_i will reach zero at

$$t_{reach} \leq \frac{|s_i(t=0)|}{\eta_i}. \tag{3.20}$$

Typical DAE solution methods require a pre-processing step to find consistent initial conditions to then pass on to the DAE solver.

3.3 Switching surfaces and equivalent control

Nonholonomic systems have output relative degree $r = 1$, and holonomic constraints have output relative degree $r = 2$ [11]. Therefore,

$$S \equiv \begin{bmatrix} s_n \\ s_h \end{bmatrix} = \begin{bmatrix} \Phi_n \\ J_q \cdot v + \sigma \Phi_h \end{bmatrix}, \tag{3.21}$$

where σ is a measure of the "speed" of the holonomic surfaces s_h and is an $m_h \times m_h$ diagonal matrix,

$$\sigma \equiv \begin{bmatrix} \sigma_1 & 0 & \dots & 0 \\ 0 & \sigma_2 & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \dots & \sigma_{m_h} \end{bmatrix}. \tag{3.22}$$

For control affine systems like $\dot{x} = f + Bu$, Utkin, et al. [18] show,

$$\dot{S} = G\dot{x}, \tag{3.23}$$

where $G = \frac{\partial S}{\partial x}$ and,

$$u_{eq} = -(GB)^{-1}Gf. \tag{3.24}$$

This is the control input that defines S as an invariant manifold and is equivalent to an unstabilized index reduction as discussed in [3, 8]. For the surfaces in (3.21),

$$G = Ak^T + Je^T, \tag{3.25}$$

where J is from (2.8),

$$A \equiv \begin{bmatrix} J_{qn} \\ (J_{qq}v) + \sigma J_q \end{bmatrix}, \quad \text{and} \quad J_{qn} \equiv \left[\frac{\partial \Phi_n(q,v)}{\partial q}\right]. \tag{3.26}$$

3.4 Constraint stabilization

3.4.1 Post-Stabilization

To guarantee the constraint surfaces are attractive, we now consider a stabilization method called post-stabilization [1, 24]. Essentially, this method subtracts the constraint component orthogonal to the constraint manifold from the state vector. The methods presented by Ascher and Petzold [3] and Chin [24] perform a two step post-stabilization shown to provide constraint satisfaction accuracy of $O(h^{2(p+1)})$. A slight variation on this method is to post-stabilize on positions first using the constraints, then to post-stabilize at the velocity level using the *surfaces*, S. Notice in equation (3.21), with $\Phi_h \approx 0$, s_h is approximately equal to the implied velocity-level constraint, $J_q v$.

The procedure now proposed is outlined as follows. Assuming each integration step provides $\bar{x}$ at t_{k+1}, first post-stabilize on positions using Φ then post-stabilize on velocities using S, i.e.,

$$\bar{x}^q = \bar{x} - kF(\bar{x})\Phi(\bar{x}), \tag{3.27}$$

$$\hat{x} = \bar{x}^q - eF(\bar{x})S(\bar{x}^q). \tag{3.28}$$

Then perform the second post-stabilization step consecutively on position and velocity,

$$\hat{x}^q = \hat{x} - kF(\hat{x})\Phi(\hat{x}), \tag{3.29}$$

$$x = \hat{x}^q - eF(\hat{x})S(\hat{x}^q). \tag{3.30}$$

In the steps above, $F = J^T(JJ^T)^{-1}$, where J is the constraint Jacobian in equation (2.8) and $\Phi^T = [\ \Phi_n^T \mid \Phi_h^T\ ]$.

This post-stabilization procedure performs exceptionally well in satisfying the constraints and surfaces. Indeed, it satisfies the constraints and surfaces so well that extra care must be taken to ensure physically realistic corrections are made from $\bar{x}$ to x. Fig. 3.3, shows an example integration step when $\dot{S} \neq 0$ during intermediate stage values on $[t_k \quad t_{k+1}]$. This results in a state vector whose corresponding surface lies outside the integrator uncertainty band of width $O(h^{p-1})$.

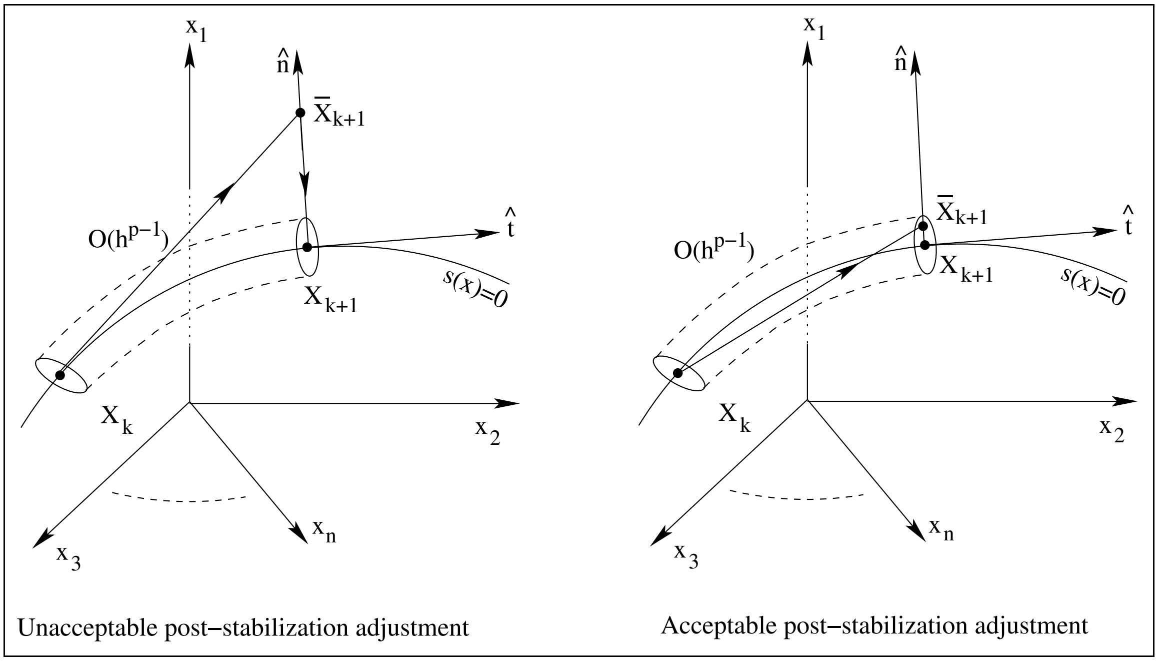

x1
n̂
X̄k+1
O(hp−1)
t̂
Xk+1
s(x)=0
Xk
x2
x3
xn
Unacceptable post−stabilization adjustment
x1
n̂
X̄k+1
O(hp−1)
t̂
Xk+1
s(x)=0
Xk
x2
x3
xn
Acceptable post−stabilization adjustment

Figure 3.3: Example of how DAE post-stabilization can exceed physically realistic adjustment in x.

Post-stabilization directly adjusts the state vector from $\bar{x}$ to x. From a physics-based standpoint, this is an instantaneous change in the total system energy, from $E(\bar{x})$ to some new $E(x)$. If this energy change, $\Delta E = E(\bar{x}) - E(x)$, is large, the result is a physically incorrect adjustment. This paper presents an upper bound on post-stabilization adjustment as anything less than the uncertainty in the numerical integration. Specifically, it should be smaller than the local truncation error which is $O(h^p)$, so $\|\bar{x} - x\|_\infty \leq O(h^{p-1})$. In the results presented in section 3.5, p was 3, 5, or 8 corresponding to the truncation error of the Runge-Kutta method used. The definition of $O(h^{p-1})$ involves a multiplicative constant[3], C, which is assumed equal to one when determining if an adjustment is acceptable or not.

3.4.2 Acceleration-level stabilization

There are at least two options to prevent incorrect post-stabilization adjustments. First, $u = u_{eq}$ may be recomputed at each integrator stage. Second, by studying boundary layer dynamics as in Utkin [17] or Utkin, et al. [18], an *acceleration*-level adjustment may be made that is very similar to post-stabilization. The motion in a boundary layer about $S = 0$ is given by $\dot{x} = f + Bu_{eq} + B(GB)^{-1}\dot{S}$. Upon evaluation of $\dot{\bar{x}} = f + Bu_{eq}$, the term $\dot{\bar{S}} = G\dot{\bar{x}}$ will be non-zero. Substituting this into,

$$\dot{x} = \dot{\bar{x}} - B(GB)^{-1}\dot{\bar{S}},$$

results in $\dot{x}$ that causes $\dot{S} = 0$. The term $-B(GB)^{-1}\dot{\bar{S}}$ subtracts the component of $\dot{\bar{S}}$ orthogonal to the switching surface similar to post-stabilization. Noticing that $\frac{d\dot{S}}{du} = GB$, this adjustment is equivalent to one Newton-Raphson iteration on u similar to [15]. No limitations are placed on the magnitude of this correction term because instantaneous acceleration-level changes are considered adjustments toward the *correct* dynamics. Without this adjustment, the trajectories would be incorrect as clearly indicated in Fig. 3.8 by the experiment performed in section 3.5.3. From a practical standpoint, $B = B(x)$ and $G = G(x)$ must be updated at each integrator stage in order to generate $\dot{S} = 0$ close to machine tolerance.

In the same way that post-stabilization can be interpreted as removing the component of x normal to the constraint surface (Fig. 3.3) acceleration-level stabilization removes the componenent of $\dot{x}$ normal to the switching surface. This is simply another way of forcing $\dot{x}$ to be tangential to the surface $S = 0$ as described in [18]. Since u_{eq} is defined as the input that causes $\dot{S} = 0$, when performed at each integration stage, acceleration-level stabilization is identical to recomputing u_{eq} during each RHS-function evaluation. Computing u_{eq} and performing acceleration-level stabilization both require $(GB)^{-1}$ and thus require roughly the same computational cost.

3.5 Example 1: two equivalent pendulum formulations

Consider a baseline example for a single degree-of-freedom compound pendulum with a rod of mass, $m = 36$(kg), and length, $l = 1$(m) (Fig. 3.4).

The system is first described and solved with a single ODE, then by a set of index-3 DAE.

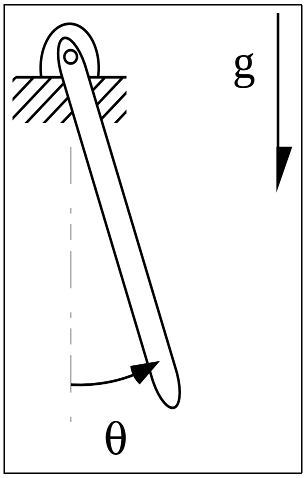


Figure 3.4: Compound pendulum described by an ODE

3.5.1 ODE method

The ODE describing the pendulum motion is $\ddot{\theta} = -3g\sin(\theta)/(2l)$. This ODE was solved with 3rd, 5th, and 8th order explicit, Runge-Kutta solvers with fixed step size. Initial conditions were $\theta(t = 0) = 20\pi/180$ (rad) and $\dot{\theta}(t = 0) = 1.0$(rad/s). The three solutions provided nearly identical results to that shown in Fig. 3.5.

An energy-based measure of how different integrator orders affect the ODE solution is shown below. Table (3.1) shows results with several integration orders and stepsizes. There is no energy dissipation in the system, thus any $\Delta Energy$ is a result of truncation and roundoff error. $\Delta Energy$ is given as $\Delta E = max(E) - min(E)$. The total energy in this system is

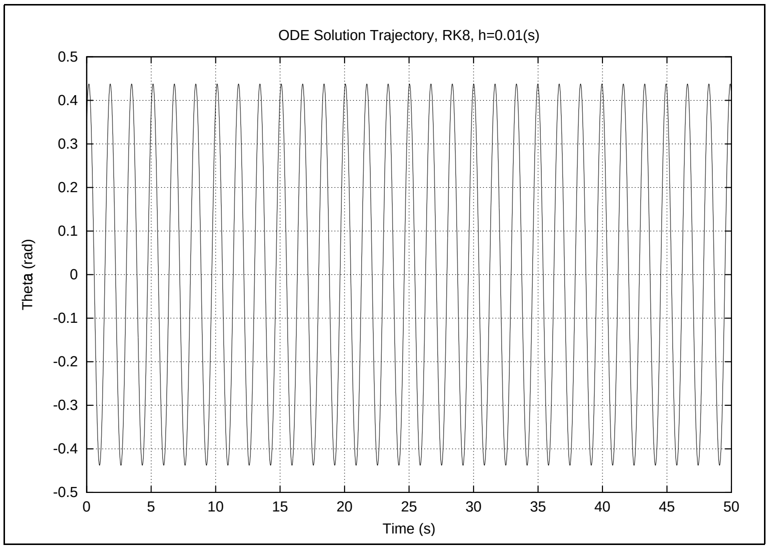


Figure 3.5: This trace is representative of all three solutions. Note, $\omega_n = \sqrt{3g/2l}$ for small angle motion.

Stepsize	ΔE^a for RK-3	ΔE for RK-5	ΔE for RK-8
h=0.05	1.8×10^{-1}	1.4×10^{-4}	5.3×10^{-11}
h=0.01	1.4×10^{-3}	4.5×10^{-8}	5.7×10^{-14}
h=0.005	1.8×10^{-4}	1.4×10^{-9}	5.7×10^{-14}
h=0.001	1.4×10^{-6}	5.1×10^{-13}	1.4×10^{-13}

[a] $\Delta E = max(E) - min(E)$ and represents the maximum variation in system energy.

Table 3.1: Variation in energy as functions of stepsize and integrator order for the ODE pendulum problem formulation.

$E(t = 0) = 159.93$(J).

3.5.2 DAE method

To demonstrate the effectiveness of the proposed solution method, the single-link pendulum is now modeled as a set of DAEs similar to the example by Yun and Sarkar [10]. First assume the link is unconstrained and able to move freely in a plane.

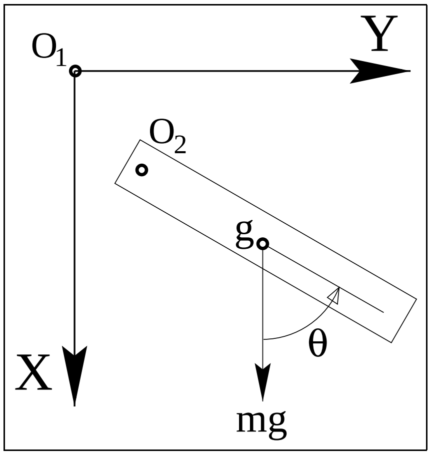


Figure 3.6: Planar rod

Next, assume no external input torques, $\tau = 0$, and a state vector of $z = [x_g \; v_{xg} \; y_g \; v_{yg} \; \theta \; \omega]^T$, the unconstrained equations of motion are,

$$\sum \vec{F}_x = m\dot{v}_{xg} = mg,$$

$$\sum \vec{F}_y = m\dot{v}_{yg} = 0,$$

$$\sum \vec{M}_g = I_g\dot{\omega} = 0.$$

where $I_g = \frac{1}{12}ml^2$ (Fig. 3.6). These are rearranged to define,

$$D = \begin{bmatrix} mg \\ 0 \\ 0 \end{bmatrix} \qquad M = \begin{bmatrix} m & 0 & 0 \\ 0 & m & 0 \\ 0 & 0 & I_g \end{bmatrix} \tag{3.31}$$

Now, add the constraints by specifying that position O_1 and O_2 must coincide. The two holonomic constraints are,

$$\Phi_h = \begin{bmatrix} x_g - \frac{l}{2}cos(\theta) \\ y_g - \frac{l}{2}sin(\theta) \end{bmatrix}. \tag{3.32}$$

The proposed reformulation was made using (3.14) - (3.16), then the resulting equations solved using S from (3.21), u_{eq} from (3.24), post-stabilization, and acceleration-level stabilization. This simulation used the same three integration methods and step-sizes as the ODE solution, along with $\sigma = 10$ s^{-1}, and $g = 9.81$ m/s^2. Similar to the ODE scenario, the three solutions were nearly identical to each other and can be seen in Fig. 3.7. Both S and Φ are maintained near machine tolerance (i.e. $O(10^{-16})$).

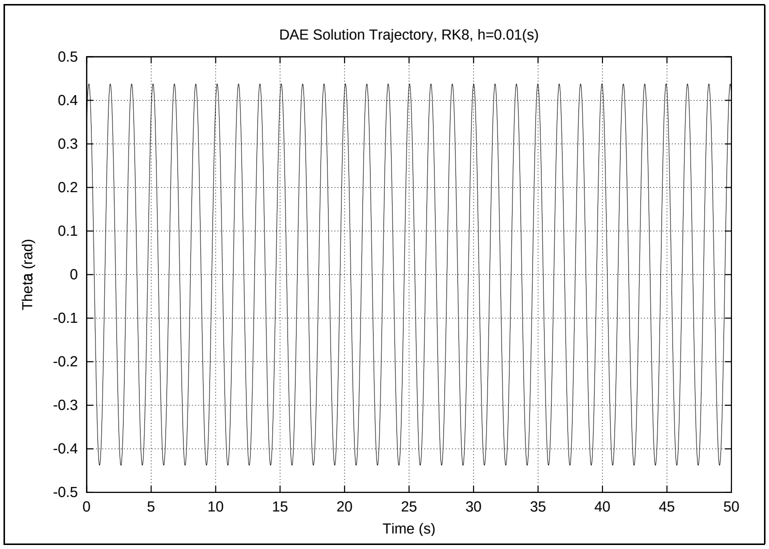


Figure 3.7: This trace is representative of all three solutions.

The change in energy for the DAE solution method is shown in Table (3.2) for several integration orders and stepsizes. Simulations used to create the table were solved to $t_{final} = 5$(s). With no energy losses in system, solution quality can be asessed by how well the solution preserves the total system energy.

Stepsize	ΔE for RK-3	ΔE for RK-5	ΔE for RK-8
h=0.05	2.0×10^{-1}	2.8×10^{-4}	4.4×10^{-8}
h=0.01	1.8×10^{-3}	8.9×10^{-8}	1.7×10^{-13}
h=0.005	2.3×10^{-4}	2.8×10^{-9}	8.5×10^{-14}
h=0.001	1.8×10^{-6}	9.7×10^{-13}	1.4×10^{-13}

Table 3.2: Variation in energy as functions of stepsize and integrator order for the DAE pendulum problem formulation.

Again, $\Delta Energy$ is a result of truncation and roundoff error.

3.5.3 Comparisons and results

By comparing Figs. 3.5 and 3.7, the ODE and DAE solution trajectories are indistinguishable. For example, the RK8 solutions were within $O(10^{-12})$ of each other. With such close agreement in state trajectories, it naturally follows that the $\Delta Energy$ for both solution methods are nearly the same. Tables 3.1 and 3.2 indicate nearly identical ΔE orders of magnitude with the exception of the larger stepsizes for RK-8. Based on these two findings, it is clear that the DAE solution method generates correct forces and moments to the unconstrained system such that the constraints, or control system outputs, are maintained at zero. In addition to the states satisfying S and Φ to machine tolerance, they also satisfy an energy invariant possessed by the physical system, yet unknown to the numerical method.

To highlight the significance of this finding, another DAE system solution trajectory is shown in Fig. 3.8. The system was integrated using the same RK8 method, step size, u_{eq}, and post-stabilization techniques. The acceleration-level stabilization was not used, however, and so $\dot{S}$ was kept small, but not at machine tolerance. The post-stabilization method was still effective enough to maintain the constraints near machine tolerance despite significant surface drift of $O(10^{-9})$. Extremely small Φ and small S might lead one to think this is a correct solution, however, comparing Fig. (3.8) with Figs. (3.7) and (3.5) reveals that it is an incorrect solution. This is also verified by noticing $\Delta Energy$ for Fig. 3.8 is 13.8(J), compared to 4.8E-13(J) for Fig. 3.7. Fig. (3.8) is an example of how application of post-stabilization can cause a physically incorrect change in x while still satisfying the constraints. It seems that post-stabilization does an exceptional job of satisfying the algebraic equations in DAE. If applied incorrectly however, it can actually produce incorrect results by "steering" the differential equation trajectories through overadjustment of the state vector.

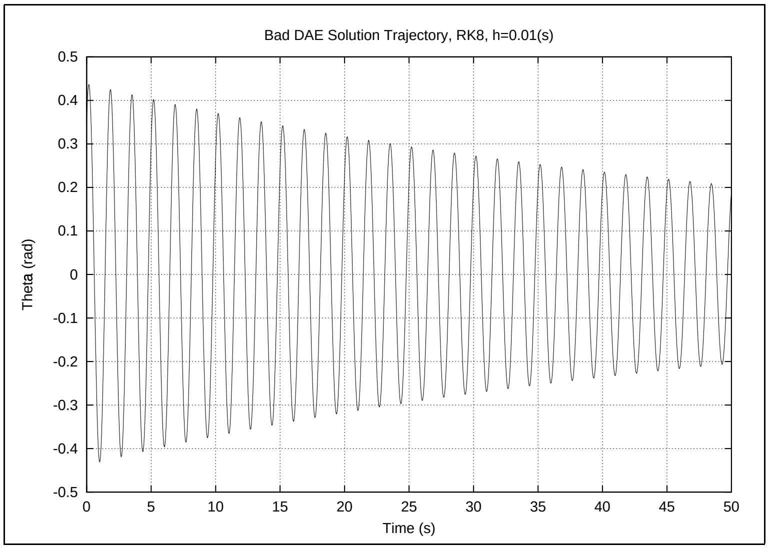


Figure 3.8: The states satisfy the constraints and switching surfaces very well but have physically incorrect trajectories.

3.6 Example 2: two rolling spheres

3.6.1 Description and features

A second example is taken from Yun and Sarkar [10] and demonstrates:

1. the method's ability to accommodate both holonomic and nonholonomic constraints, and

2. SMC's reaching-phase dynamics which drive the constraints to satisfaction given inconsistent initial conditions.

The system consists of a small sphere of radius r constrained to roll without slipping along the outer surface of a larger sphere of radius R. The larger sphere's center is fixed to the inertial reference frame origin as shown in Fig. 3.9.

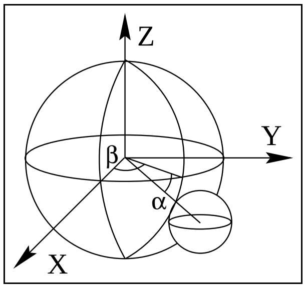


Figure 3.9: Two rolling spheres

The center of the small sphere is located in the inertial frame, XYZ, with spherical coordinates (α,β,ρ) and its orientation is represented with body-fixed Z-X-Z Euler angles, (ϕ,θ,ψ).

3.6.2 Equations of motion

Both the dynamics and constraint equations are developed more fully in [10] but are presented in final form here. In the absence of gravity,

$$M(q)\ddot{q} + V(q, v) = 0 \tag{3.33}$$

where

$$M = \begin{bmatrix} m & 0 & 0 & 0 & 0 & 0 \\ 0 & m\rho^2 & 0 & 0 & 0 & 0 \\ 0 & 0 & m\rho^2 \cos^2 a & 0 & 0 & 0 \\ 0 & 0 & 0 & I & 0 & I\cos\theta \\ 0 & 0 & 0 & 0 & I & 0 \\ 0 & 0 & 0 & I\cos\theta & 0 & I \end{bmatrix}$$

$$V = \begin{bmatrix} -m\rho(\dot{\alpha}^2 + \dot{\beta}^2 \cos^2\alpha) \\ m\rho(2\dot{\rho}\dot{\alpha} + \rho\dot{\beta}^2 \cos\alpha \sin\alpha) \\ 2m\rho(\dot{\rho}\dot{\beta}\cos^2\alpha - \rho\dot{\beta}\dot{\alpha}\cos\alpha\sin\alpha) \\ -I\dot{\theta}\dot{\psi}\sin\theta \\ I\dot{\phi}\dot{\psi}\sin\theta \\ -I\dot{\phi}\dot{\theta}\sin\theta \end{bmatrix}$$

The single holonomic constraint indicates the distance between the center of the two spheres remains constant:

$$\Phi_h = \rho - (R + r)$$

The two nonholonomic constraints are equalities between tangential velocities at the interface that prevent slipping during rolling,

$$\Phi_n = \begin{bmatrix} (R+r)\dot{\alpha} - r(\dot{\phi}\sin\theta\cos\psi - \dot{\theta}\sin\psi) \\ (R+r)\dot{\beta} - r(\dot{\phi}\sin\theta\sin\psi + \dot{\theta}\cos\psi) \end{bmatrix}.$$

The resulting Jacobians required to compute u_{eq} from (3.24) are

$$J_q = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} \qquad J_{qq}v = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} \tag{3.34}$$

$$J_v = \begin{bmatrix} 0 & (R+r) & 0 & -r\sin\theta\cos\psi & r\sin\psi & 0 \\ 0 & 0 & (R+r) & -r\sin\theta\sin\psi & -r\cos\psi & 0 \end{bmatrix}$$

$$J_{qn} = \begin{bmatrix} 0 & 0 & 0 & 0 & -r\dot{\phi}\cos\theta\cos\psi & (r\dot{\phi}\sin\theta\sin\psi + r\dot{\theta}\cos\psi) \\ 0 & 0 & 0 & 0 & -r\dot{\phi}\cos\theta\sin\psi & (-r\dot{\phi}\sin\theta\cos\psi + r\dot{\theta}\sin\psi) \end{bmatrix}$$

Similar to example 1, total system energy is used as a metric with which to verify a physically realistic numerical solution. There is no potential energy storage in this system so the kinetic energy function represents the total system energy. Again, in final form from [10],

$$T = \frac{1}{2}m(\dot{\rho}^2 + \rho^2\dot{\alpha}^2 + \rho^2\dot{\beta}^2cos^2\alpha) + \frac{1}{5}mr^2(\dot{\phi}^2 + \dot{\theta}^2 + \dot{\psi}^2 + 2\dot{\phi}\dot{\psi}cos\theta). \quad (3.35)$$

3.6.3 Results

Results for the two-sphere problem were generated using the same fixed-step 3^{rd}-, 5^{th}-, and 8^{th}-order Runge-Kutta solver. Parameters required for the simulations were $R = 0.55$(m), $r = 0.05$(m) and, although the equations contain the sphere mass, the solution is independent of m, however it was taken as 1(kg) to compute the total energy. Consistent initial conditions on displacements were $\alpha_o = 0$(rad), $\beta_o = \frac{\pi}{2}$(rad),$\rho_o = 0.6$(m), and $\phi_o = 0$(rad), $\theta_o = \frac{\pi}{2}$(rad),$\psi_o = \frac{\pi}{2}$(rad). Consistent ICs on velocities were $\dot{\alpha}_o = -\frac{45}{8}\frac{\pi}{180}$(rad), $\dot{\beta}_o = \frac{45}{8}\frac{\pi}{180}$(rad), $\dot{\rho}_o = 0$(m), and $\dot{\phi}_o = \frac{(R+r)}{r}\dot{\beta}_o$(rad), $\dot{\theta}_o = -\frac{(R+r)}{r}\dot{\alpha}_o$(rad), $\dot{\psi}_o = 0$(rad).

Simulation results are shown for consistent ICs in Fig. 3.10 and inconsistent ICs in Fig. 3.11. The lower-right plot shows the total system energy remained constant throughout $[t_{initial} \;\; t_{final}]$.

Like the previous example, this system posesses an energy invariant useful for asessing solution quality. Table (3.3) shows variations in energy using several integration orders and stepsizes. The results shown are variations from the nominal system energy, $E(t = 0) = 4.858 \times 10^{-3}$(J). This means that the result for RK-3 at $h = 0.01$(s) preserves the energy invariant to within ≈0.0001%.

Inconsistent ICs were created by specifying $\rho_o = 0.7$(m) and augment-

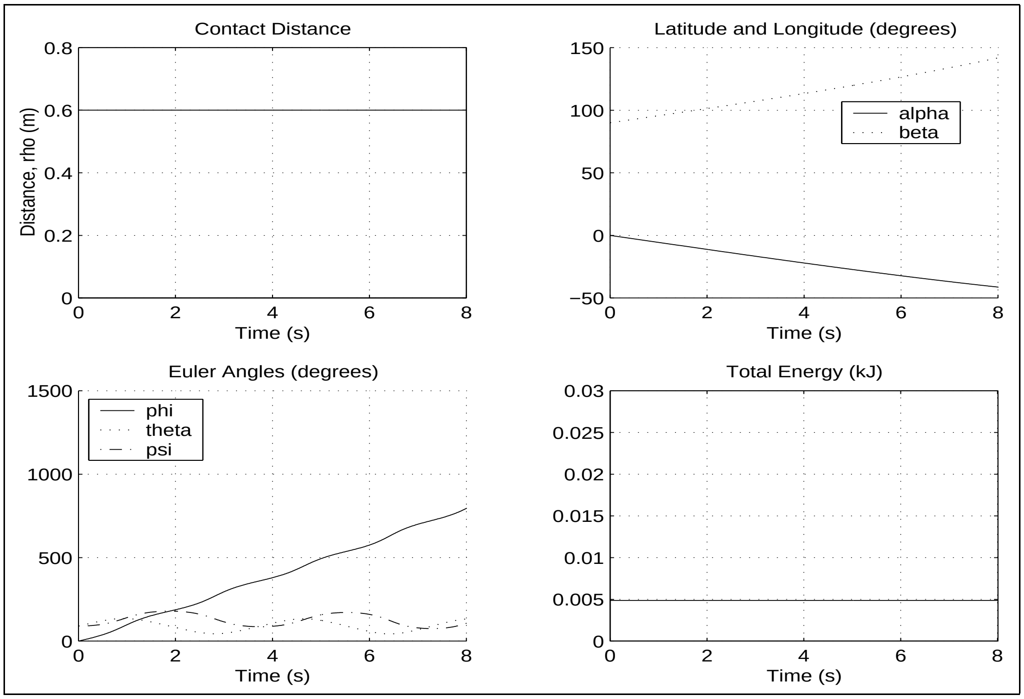


Figure 3.10: Given consistent ICs, post-stabilization is used over $[t_{initial} \;\; t_{final}]$

ing the consistent IC's with $\dot{\phi}_o = 1.2\dot{\phi}_o$ and $\dot{\theta}_o = 1.5\dot{\theta}_o$. All solutions computed with inconsistent IC's were generated using a fixed-step 5th-order RK integrator using $h = 0.01$(s).

To more clearly reveal the reaching phase surface dynamics, Fig. 3.12 shows all three $s_i(t)$ when using:

1. SMC's stabilization method until $|s_i| \leq O(h^p)$ at $t_{reach,i}$, then
2. acceleration-level and post-stabilization for $t > t_{reach,i}$.

The lower-right plot in Fig. 3.11 shows the total system energy remained constant after $t_{reach,3}$. It is worth noting that, although inconsistent ICs can be accommodated with this methd, the larger the inconsistency, the larger the

Stepsize	ΔE for RK-3	ΔE for RK-5	ΔE for RK-8
h=0.05	6.0×10^{-7}	3.3×10^{-10}	2.3×10^{-14}
h=0.01	4.6×10^{-9}	9.2×10^{-14}	2.4×10^{-17}
h=0.005	5.7×10^{-10}	2.8×10^{-15}	2.3×10^{-17}
h=0.001	4.6×10^{-12}	6.2×10^{-17}	6.6×10^{-17}

Table 3.3: Variation in energy as functions of stepsize and integrator order for the rolling-spheres problem.

energy change is after driving them to satisfaction. One factor that affects how much energy is added by the controller during the reaching phase is the magnitude of η_i in $u_{robust,i}$.

3.7 Conclusions

The index-3 DAE resulting from constrained mechanical systems have been reformulated as an equivalent control problem. An explicit transformation from DAE literature notation to control canonical state space form was presented. The sliding-mode control framework was chosen primarily because of it's ability to address both nonholonomic and holonomic systems in a unified framework. Additionally, SMC theory can address nonlinear time-varying systems without approximations or simplifying assumptions [16]. Since control theory is designed with large output errors in mind, application of a variable structure control (VSC) to the constrained MBS problem eliminates the requirement for consistent ICs. The flexibility of SMC as a VSC methodology easily accommodates the decision to apply a traditional SMC control law (i.e. discontinuous) during the reaching phase, then switch to the smooth acceleration-level and post-stabilization methods once the surface has been

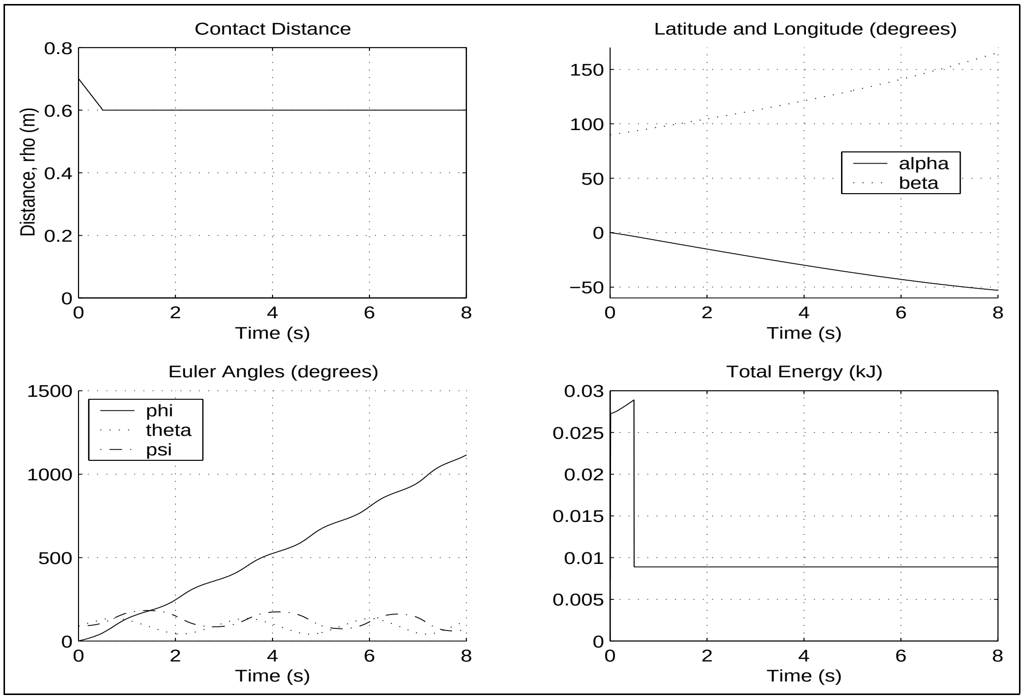


Figure 3.11: Given inconsistent ICs, SMC's $u_{robust,i}$ drives the constraints to satisfaction

reached. With this hybrid approach, constraint violations at $t = t_{start}$ are guaranteed to be eliminated within finite time [23] and then remain zero for the duration of the simulation.

After reformulating the MBS index-2 or index-3 DAE problem into $\dot{x} = f + Bu$, SMC theory is used to define switching surfaces for both nonholonomic and holonomic constraints. Additionally it is used to find the smooth input, u_{eq}, that defines the switching surfaces as invariant sets. This is the control-theoretic equivalent to an unstabilized index reduction found in DAE solution literature [3]. For consistent ICs, or after a surface has reached zero, instead of using $u_{robust} = -\eta \cdot sgn(S)$ in u as is typical for the SMC design procedure, a post-stabilization method found in the DAE solution literature was used to

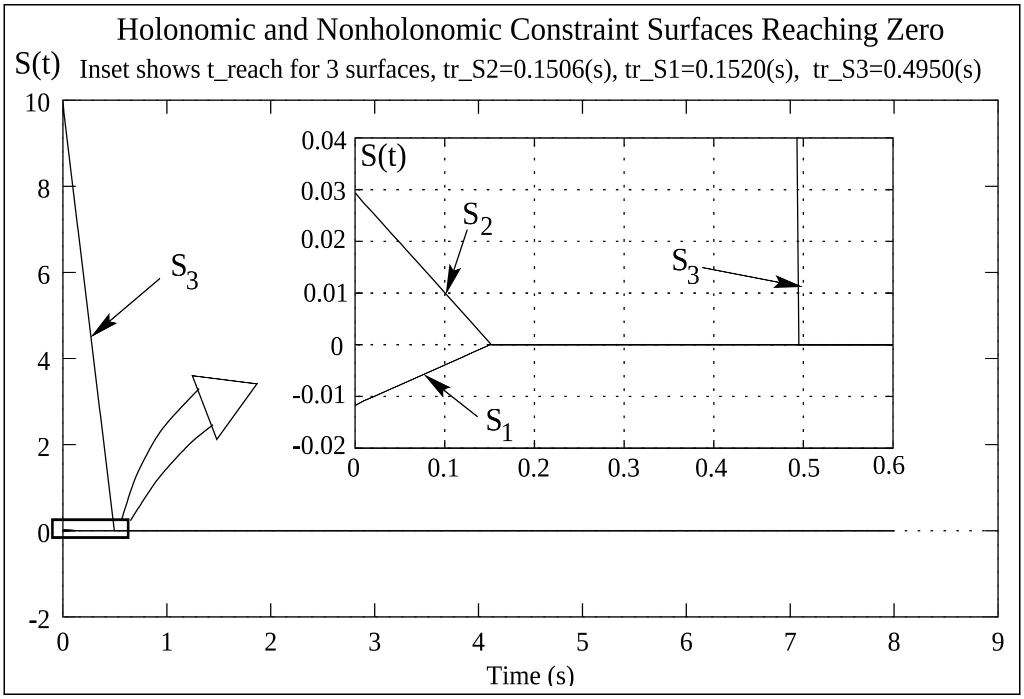


Figure 3.12: SMC Forces Constraints to Satisfaction Given Inconsistent ICs

guarantee the switching surfaces remained attractive [1, 3, 24].

DAE stabilization methods are chosen over SMC's stabilization method because u_{robust} works *through* the integration process which has an uncertainty of $O(h^p)$. For this reason it is unreasonable to expect SMC to provide constraint satisfaction better than $O(h^p)$ while still maintaining reasonably large step sizes. This uncertainty is the primary source of chatter in simulation of SMC systems. The DAE stabilization methods operate after each time step which elimintates discretization chatter. The uncertainty with these methods have been estimated at $O(h^{2(p+1)})$[1] which is a level of accuracy frequently better than machine tolerance.

An acceleration-level stabilization method is presented and used in conjunction with post-stabilization. The acceleration-level stabilization was developed from the study of SMC boundary layer dynamics along with insights from post-stabilization [17, 18].

The combination of SMC's switching surfaces and equivalent control, DAE post-stabilization methods, and an acceleration-level stabilization was used to generate trajectories of two example systems by numerical integration. For example one, comparison of the DAE solution with the equivalent ODE solution for a simple pendulum provides initial verification that the proposed method produces correct state trajectories. As a second metric used for verification, the total system energy was shown to have similar error accumulations in both the ODE and DAE formulations.

The energy metric was also used to explain how post-stabilization can be incorrectly used to successfully satisfy the constraints but still fail to generate correct state trajectories. An upper bound on the magnitude of post-stabilization adjustment at any given time step was presented as the integration truncation error, $O(h^{p-1})$.

Finally, example two was chosen because of its increased complexity in dynamics as well as constraints. The hybrid numerical method performed well on this example, driving both holonomic and nonholonomic constraints to satisfaction, then keeping them satisfied to machine tolerance.

Chapter 4

Background on Solving Discontinuous ODE

4.1 Introduction

The next two chapters present a discontinuity handling procedure for solving discontinuous ordinary differential equations (DODE) using single-step methods. The algorithm uses a detect-locate-restart approach to traverse discontinuous events while avoiding the drawbacks typically found when solving DODE with a smooth ODE solver. It combines efficiency and accuracy in both the detection and location phases with the simplicity of a single-step integrator in the restarting phase. The advantages single-step integrators have over multi-step methods are that each step is fully high-order and each stepsize can be easily adjusted to closely match the local error criterion. In addition it provides guarantees on locating events either slightly before or after the actual event by a user-specified amount. This feature helps overall integrator efficiency and ensures a uniform environment in which to make discontinuous changes. Lastly, this algorithm makes use of a region of concurrency to further improve efficiency for some problems by allowing multiple events to occur at the same time.

In physical system modeling, DODE result from simplified models that approximate abrupt or intermittent effects. This encompasses a wide variety of events such as Coulomb friction, mechanical system contact, electrical or hydraulic gate changes, or other power-path topological changes such as transmission gear shifting, step changes in amplifier gains, or other locking effects that model "hard stops." The common challenge to solving equations with these effects are the discontinuous nature of their solutions. Inefficiencies and inaccuracies at discontinuous events are the two primary drawbacks when solving DODE with most variable step (smooth) ODE solvers. One way to reduce these effects is to reduce the integration tolerance(s). For small sets of DODE, the accompanying inefficiency may go unnoticed, even for very small tolerances. For larger systems of equations, this method for improving accuracy can make solution times prohibitively lengthy. Another way to reduce these drawbacks is to explicitly make use of the extra information often available with DODE, namely the time and/or state conditions describing circumstances of a discontinuous event. There is an increased level of effort associated with organizing and using the extra information but, as will be shown, the performance improvement for some systems justifies the cost.

Traversing discontinuities cause smooth integrators to attempt and reject many steps and yield a final average stepsize smaller than necessary for the requested tolerance. Explicitly handling discontinuities is rewarded most when solving systems with many events and when each RHS function evaluation is computationally expensive. Accordingly, the method presented here is designed for such systems and attempts to economize on the total number of RHS evaluations. It achieves this by partitioning the independent axis into intervals of smooth equations that can be effectively solved by a smooth integrator. Some features this numerical method incorporates are:

1. the ability to efficiently handle many concurrent discontinuous events.

2. the ability to use any general purpose single-step ODE solver.

3. the ability to locate discontinuous events to within machine tolerance if necessary, but still be able to operate with relaxed tolerances for low-accuracy requirements.

4. to eliminate discontinuity sticking for both bi- and unilateral events, a phenomenon identified by Park and Barton [56] as a significant source of inefficiency.

The first feature is intended to eliminate some drawbacks found in problems containing many closely spaced events. Typically a discontinuty handling procedure aims to detect and locate all events in sequence. In situations with many events very closely spaced together, locating each individual event can inadvertantly result in a performance penalty rather than gain. For these cases, a region of concurrency (ROC) is presented as a way of retaining the benefits while at the same time reducing the penalties.

The second feature is intended to leverage advancements made in smooth ODE solution methods. Other DODE/DDAE solvers designed around a specific smooth solver (e.g. [30, 41, 56]) gain the benefits of fully exploiting that solver but have the drawback of being tied to one specific method. By remaining general and modular, this algorithm can be improved with future advancements in the detection, location, or smooth integration areas. In addition, a modular single-step environment lends itself to development of a stiff discontinuous ODE solver through use of a stiff ODE integrator. The same applies to development of a stiff discontinuous high-index DAE solver [34, 52, 53].

The third and fourth features are included to provide comparable features to those found in Park and Barton's BDF-based discontinuous DAE solver [56]. Their paper made significant advancements in solving discontinuous differential equations. This work further characterizes discontinuous events into unilateral and bilateral types and extends one of their features, namely consistent event location, to both discontinuity types.

The next two chapters are organized into six sections. Section 4.2 is a literature review outlining the contributions of some seminal papers along with several associated difficulties related to the DODE problem. Sections 5.1 and 5.2 present the formal problem definition and the new proposed algorithm. Section 5.3 presents several benchmark problems solved with the new algorithm and finally, section 5.4 presents conclusions based on the benchmark problem results.

4.2 Literature review

A review of the literature between 1978 and 1996 reveals similarities in solving discontinuous ODE and DDAE with common difficulties, approaches, and performance measures. The common difficulty is finding accurate solutions efficiently using integration methods originally designed to solve smooth equations, whether they be stiff or nonstiff [3, 8, 52]. Assuming modern smooth solvers are the most effective means for obtaining solutions along smooth intervals, the common approach is to partition the independent axis into subintervals between discontinuous events. If the event times are explicitly known beforehand, efficient use of this information is demonstrated in [39, 27, 58]. However, if event times are unknown beforehand, other methods attempt to detect then locate events with varying degrees of success and accuracy.

Although some papers take an approach designed to minimize user-input by using information local to the integrator exclusively [32, 44, 41], most methods address the detection and location phases with the use of discontinuity functions, $g(t, x)$. These are auxiliary functions whose zeros indicate the presence of a discontinuous change in the equations. By detecting and locating roots of $g(t, x)$ during integration, the benefits of modern smooth ODE solvers can be leveraged to generate discontinuous solutions while at the same time avoiding their accuracy and efficiency penalties [31, 41, 39]. With discontinuity functions available, a variety of methods for root detection and location have been presented and implemented. A good review of these efforts can be found in [56].

In the most recent paper surveyed, Park and Barton[56] identified and defined the problem of *discontinuity sticking* and also presented a solution, *consistent event location.* Discontinuity sticking occurs when the same event is incorrectly detected and located more than once. They mentioned that this is not an uncommon occurrence and can happen when solving either discontinuous DAE or ODE. Figure (4.1) illustrates this phenomenon across two integration steps. The first step from t_k to t_{k+1} detects the discontinuity, then successfully locates it such that $|g(x_{k+1}, t_{k+1})| \leq \epsilon_d$. Although a time is found that satisfies the root finding procedure, the event is not yet triggered because the located time is prior to the actual event time. Since this event is a *bilateral* discontinuity (discussed further in section 5.1.3) it *requires* $g(t, x)$ to cross the zero axis to occur. The step from t_{k+1} to t_{k+2} will re-detect and re-locate the same event which, for the same reasons, may or may not trigger the event.

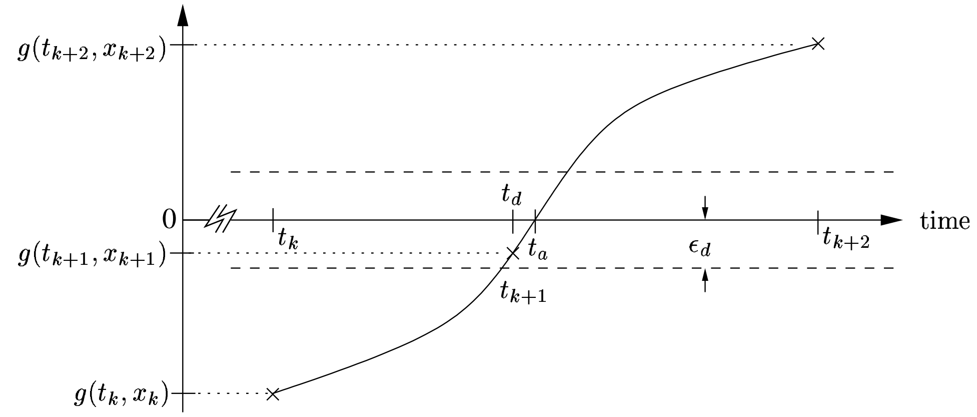


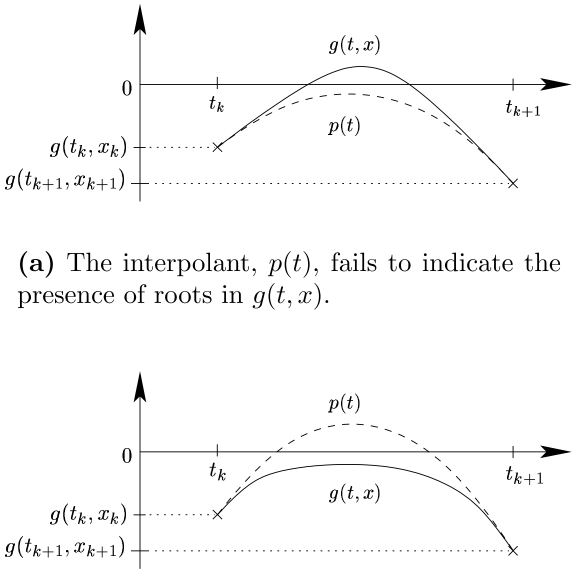
Figure 4.1: For bilateral events, discontinuity sticking occurs when the located time, t_d, is not past the actual event time, t_a.

Park and Barton's strategy for eliminating discontinuity sticking is called *consistent event location*, referred to here as CEL$^+$. CEL$^+$ guarantees the located times are actually *past* (i.e. to the right of) the actual event times. This not only increases integrator efficiency by eliminating discontinuity sticking, but it also provides a uniform environment in which to make discontinuous model changes. Birta et.al. [29] recognized and implemented a technique similar to CEL$^+$ for ODE by adding an offset to the discontinuity function during root finding, however they neither located events to machine tolerance nor attempted to provide guarantees on event location.

Efficiency gains using CEL$^+$ may be measured in two ways. First with respect to smooth solvers and second with respect to other discontinuous solvers that do not implement CEL$^+$. By explicitly handling discontinuous events with CEL$^+$, a DODE solver can traverse an event in a single step in contrast to the multiple failed steps smooth solvers require, and even though a DODE solver may explicitly handle discontinuities, if it does not use CEL$^+$, it still allows for the possibility of discontinuity sticking. Explicitly detecting and locating discontinuous events with CEL$^+$ eliminates all failed or repeated attempts to traverse a bilateral discontinuity.

Park and Barton's discontinuous DAE solver is based on the BDF method, a multistep ODE solver. They exploited the BDF method's internally generated polynomial interpolants for event detection and location by appending the discontinuity functions onto the state vector. Like Carver[30], Park and Barton use the integrators's stepsize control to ensure root detection and location accuracy. Using these interpolants, they were able to eliminate discontinuity sticking and achieve event location to within machine tolerance.

Although most modern single-step ODE solvers contain an interpolant for dense output, the interpolant itself contains errors on the order of the states or greater. Thus, the dense output interpolants are not suitable for use in the same way as the BDF interpolants. Attempting to use interpolants that are less accurate than the states in event detection allows for the possibility that root crossings exist but are missed due to the discrepancy between interpolant and discontinuity function, $g(t, x)$ (Fig. 4.2(a)). Likewise, the same discrepancy can cause lower order interpolants to falsely indicate the presence of roots when $g(t, x)$ contains none (Fig. 4.2(b)). Eliminating both of these problems while using lower-order interpolants proved to be the primary difficulty in developing a single-step DODE solver with guaranteed event detection



(a) The interpolant, $p(t)$, fails to indicate the presence of roots in $g(t, x)$.

(b) The interpolant, $p(t)$, falsely indicates the presence of roots in $g(t, x)$.

Figure 4.2: Two difficult event detection scenarios.

and location.

One way to increase interpolant accuracy while maintaining the single step nature of the algorithm would be to use the information generated in the interior of a step. Although Horn [48] and others [59, 40] have shown the popular Fehlberg RK-4(5) pair contains a 4^{th}-order estimate of the states along $[t_k \;\; t_{k+1}]$, at $O(h^4)$ this is still significantly less accurate than the state estimates at $O(h^5)$. Others have gone to great lengths to construct and/or use higher-order interpolants with Runge-Kutta methods [55, 61, 64, 47, 59, 60], but in general there is no RK method that provides polynomial interpolants of equal or higher order than the state estimates with no cost in the number of function evaluations.

Ideally, one would use an interpolant of higher order than the methods underlying ODE state estimate [40, 59]. This would ensure the dominant error associated with an interpolant was due to the states error of $O(h^p)$ and not the interpolant's error of $O(h^{p+1})$ or smaller. Enright et.al. [40] develop such an interpolant and show that one extra function evaluation produces an interpolant of $O(h^p)$ and two extra function evaluations can produce an interpolant of $O(h^{p+1})$. In [41], they develop a method that only constructs the high-order interpolant for steps with a suspected discontinuity. This saves the cost of computing high-order interpolants at each step, however they acknowledge that their method for detecting events is the major limitation of their approach. The difficulty they faced was choosing when to create the high-order interpolant. The extra information would improve event detection and location, however the extra function evaluations are expensive and underutilized when many integration steps are taken with $g(t, x)$ far from the zero axis. Their integrator coupled with some of the detection ideas presented here hold potential as another effective and efficient DODE solver.

Use of interpolants introduces their own set of challenges. High order polynomial interpolants in power-form are notoriously ill-conditioned [57] and, especially for orders higher than 5 or 6, are dominated by roundoff errors. Nonweiler [54] outlines a recursive method for creating interpolating polynomials in Newton's reverse-form that, upon construction, seem to be less prone to roundoff error than inverting an ill-condifioned matrix for creating power-form interpolants. These interpolants could be useful in the root location phase, however the efficient detection phase in Park and Barton uses an interval method requiring polynomials in power form. Converting the reverse-Newton form interpolants into a power form suitable for use with the root exclusion test in [51] introduces significant roundoff error. A workable solution might be found in some combination of (reverse) Newton form and power form interpolants for the detection and location phases, however a more straightforward and less expensive approach was sought for the final algorithm.

Aside from roundoff error, even though an $O(h^{p+1})$ interpolant for x could theoretically provide an unlimited number of points from which to construct an arbitrarily high order interpolant for $g(t, x)$, an overall error estimate of the final interpolant is complicated by the fact that each $g(t, x)$ propagates some generally unknown error as a result of the $O(h^p)$ errors in x. To summarize, roundoff error in high-order interpolant construction, uncertainty in x due to integration and interpolation errors, along with the unknown way in which a general function $g(t, x)$ is affected by roundoff errors all contribute to the uncertainty involved in using RK-based interpolants for high-accuracy root detection and location. Shampine et.al. [60] and Enright et.al.[41] address many of these issues and both present well thought out algorithms although neither presents a complete set of numerical problmes and results to use for a thorough comparison. The solution presented here uses 3^{rd}- and 5^{th}-order Hermite interpolants to detect root crossings and estimate their locations, however Newton's method and numerical integration is used in the final root location procedure.

The following is an incomplete list of currently available solvers that have root location features. The FORTRAN codes `sdasrt.f`, `ddasrt.f`, `sderoot.f`, or `lsodar.f` from `netlib.org`[52] include a "g-stop" capability, Matlab[1] [50] has "event handling", ACSL[2] [25] implements a "schedule" function, and Easy5[3] [37, 62] uses "switched states." The solvers at `netlib.org` are freely available in source form and the rest are proprietary commercial codes. These solvers are widely available and are general enough to solve a wide range of systems. A full comparison of these softwares is beyond the scope of this work. However, based on the root detection algorithms, the primary way in which [52], [50] and [25] can be improved is in *guaranteed* detection and location of *all* discontinuous events.

They detect events through sign changes in the discontinuity functions at t_k and t_{k+1} only. This means events can be missed entirely if the trajectory

[1] MATLAB is a trademark of The MathWorks, Inc.

[2] ACSL is a trademarks of The AEgis Technologies Group, Inc.

[3] EASY5 is a registered trademark of The Boeing Company.

crosses zero in the middle of a time step and crosses again before t_{k+1} as shown in Fig. 4.3. This is likely in ratchet-like mechanisms, gear-pairs, or other systems that have sinusoidal or notched discontinuity function profiles. There were numerous interpolants similar to Fig. 4.3 in the marble jar problem presented in Section 5.

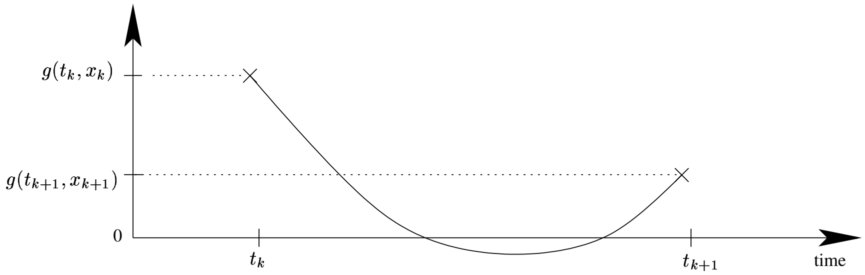


Figure 4.3: Some detection algorithms fail when $g(t_k, x_k)$ and $g(t_{k+1}, x_{k+1})$ both have the same sign.

The "switched-states" implementation in EASY5x seems to be quite effective in solving multi-domain systems with discontinuities [37, 62]. EASY5x is a comprehensive engineering analysis software that models, formulates, solves, and post-processes multidomain dynamic systems. They use a detect-locate-restart approach and discontinuity functions to locate state-based events. Their binary search routine apprantly does not integrate during event location and likely uses some type of interpolant. They claim improvements in simulation speeds of up to three orders of magnitude.

The software environment DAEPACK[36] is a comprehensive equation solving environment that, among other things, implements much of what is presented in Park and Barton's paper [56]. The work developed here is, in some part, an implementation of a subset of DAEPACK's functionality within a single-step, Runge-Kutta based environement. The discontinuity handling procedures in DAEPACK are, presumably, based on DASSL, a multi-step BDF integrator.

Finally, two common performance measures used throughout the literature are the total number of integration steps taken and the number of right-hand-side function evalutions. It is worth mentioning that direct comparison between single- and multi-step methods using the total number of steps is flawed because single-step integrators appear to take much fewer steps but their cost-per-step is greater. Single-step integrators are able to take comparatively large steps and produce fully high-order state estimates. They can also accommodate arbitrary stepsize changes from step to step. In contrast, multi-step integrators generally require fewer RHS function evaluations per step but take more steps. Multi-step methods also require a boot-strapping procedure at startup or a backstepping procedure after stepsize changes.

In an attempt to find a common performance metric for comparing single- and multi-step integrators both [32] and [59] suggest using a stepsize-per-cost measure. This quantity is difficult to achieve because some integrators can take variable-order steps and different DODE solution methods may evaluate $\dot{x}$ for various reasons. So, the performance metric of choice for comparing DODE solvers is the total number of RHS function evaluations because it is integrator independent. Other possible performance metrics are the total number of FLOPS (floating-point operations) or the total CPU-time required to solve a benchmark problem. These two metrics have the advantage that they include all overhead associated with integration, discontinuity function evaluations, and other computations, however their drawback is being specific to underlying libraries, CPU's, and software implementation environments.

Chapter 5

A Discontinuous ODE Solver Using Single-Step Methods

5.1 Preliminaries

This section presents the general discontinuous ODE problem and the detect-locate-restart solution method. It also introduces terminology and ideas used in the DODE solution method presented in section 5.2.

5.1.1 Define the problem

We seek the solution of nonsmooth, generally nonlinear ordinary differential equations,

$$\dot{x} = f(t, x, L), \qquad x(t_o) = x_o \tag{5.1}$$

assuming availability of discontinuity functions and their rates,

$$g = g(t, x) \qquad \frac{dg}{dt} = \dot{g}(t, x, \dot{x}) \tag{5.2}$$

The states and derivatives x and $\dot{x}$ are $\in \Re^{N_{states}}$, t is the scalar independent variable, and $g(t, x)$ is $N_g \times 1$. The discontinuity functions are designed such that their zeros coincide with discontinuous events in (5.1). $L = L(t, x)$ is an $N_g \times 1$ array representing the locked or unlocked status of each discontinuous element in $\dot{x}$. Each element in L is a function of it's corresponding element in $g(t, x)$,

$$L_i = \begin{cases} 0, & g_i(t_k, x_k) < 0 \\ 1, & g_i(t_k, x_k) \geq 0. \end{cases} \tag{5.3}$$

The L-array, borrowed from [56] and [40], is the mechanism used to implement discontinuity locking. Discontinuity locking is discussed in the next section and, essentially, is a technique for avoiding the inefficiencies that arise when smooth ODE solvers step across discontinuous events.

In physical system modeling, many events are functions of state only which means, in the absence of an analytical solution, the event times are unknown explicitly. Thus, the problem of efficiently and accurately solving DODE becomes that of detecting and locating all events, making appropriate discontinuous changes, then restarting integration after the event. This is referred to as the detect-locate-restart procedure. In essence, this procedure aims to partition the independent axis into subintervals of smooth ODE which can then be solved by modern smooth solvers. This approach leverages smooth ODE solver technology while avoiding the inaccuracies and inefficiencies smooth solvers exhibit at discontinuities.

5.1.2 Discontinuity locking

Lacking information about impending discontinuous events, a standard variable-step solver steps forward assuming the built-in smoothness assumptions are valid. If a discontinuity occurrs within a step, the net result is a loss of accuracy and efficiency because the solver "hunts" before and after the event attemping to satisfy the integration error criterion [31, 39, 60]. The hunting phenomenon happens because the discontinuity destroys the smooth relationship between the local error estimate and the next stepsize. To avoid losses in accuracy and efficiency, discontinuity locking is used to facilitate detection of impending discontinuous events without actually triggering the events.

This requires a special differential equation form which allows the algorithm to "lock" the system configuration over an entire time-step [40]. If a discontinuous event is triggered during the integration step, the step is rejected and the the discontinuity time is accurately located. Fig. 5.1 shows how locking the system configuration provides a smooth trajectory over which to locate a root in $g(t, x)$ before making discontinuous changes. Upon location, appropriate changes are made and integration is resumed from the located time. This is the mechanism that allows smooth ODE solvers to be used for solving nonsmooth differential equations.

The way equations in (5.1) are written depends not only on the solution method algorithm [41], but also what types of discontinuities they describe.

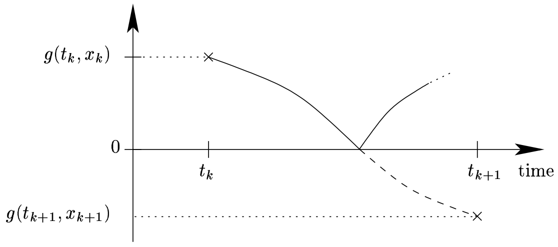


Figure 5.1: Discontinuity locking prevents discontinuous changes during event location.

Systems that undergo *equation structure* discontinuities can be written with the discontinuous terms as a function of L. This ensures discontinuity locking is implemented because the algorithm holds L constant over each interval $[t_k \;\; t_{k+1}]$. Examples of this category of discontinuous phenomenon include mechanical contact, backlash, dry or Coulombic friction, stiction, saturation, or various system configurations containing valves, gates, transistors, or diodes. Systems that experience *parametric* discontinuities and whose equation structure remains the same do not necessarily need to use L to implement discontinuity locking. For these systems, discontinuity locking is ensured by way of consistent event location through box 6e, Fig. 5.6. Examples of parametric discontinuous phenomenon include any capacitive, inductive, or resistive elements whose coefficients change discontinuously. For example, step changes in spring stiffnesses, material properties, or valve orifice coefficients all can fall within this category. When a discontinuity is located, the parameter changes can be made and integration resumed without explicit dependence upon L.

Neither of the commercially available ODE/DAE solvers from Matlab $R12$ [50] or ACSL $v11.1$ [25] explicitly implement discontinuity locking, although both solving environments could allow a creative programmer to implement a custom version of discontinuity locking. *ACSL's* solver does implement CEL$^+$ which is discussed further in section 5.1.4. The freely available solver(s) from `www.netlib.org` [52] do not explicitly implement discontinuity locking however, they are available in source format and may be modified as necessary.

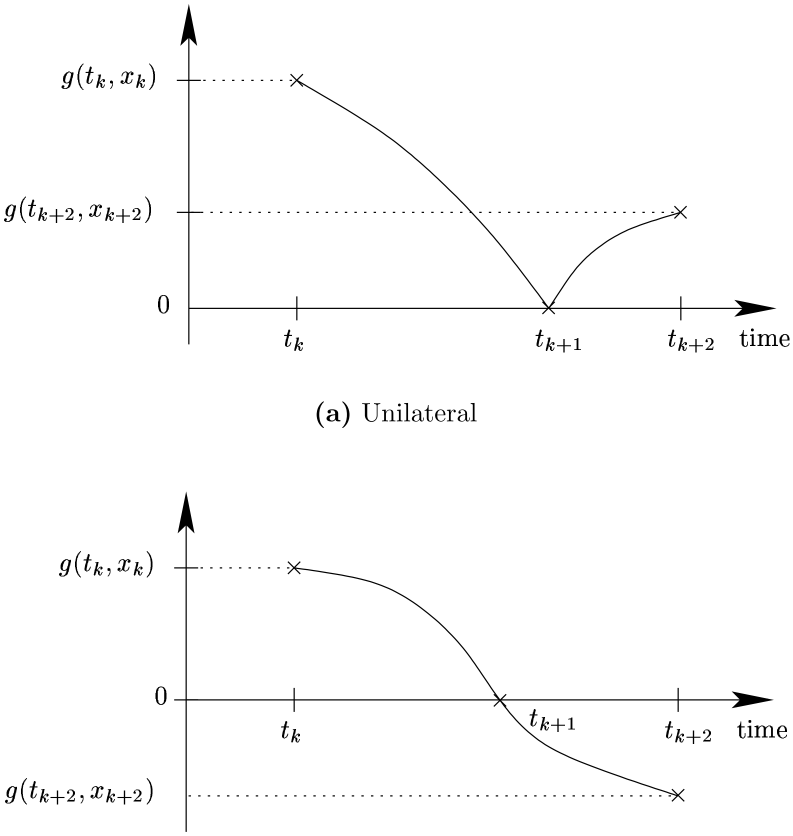
5.1.3 Unilateral and bilateral events

A further distinction can be drawn between unilateral and bilateral events. *Unilateral* discontinuities are those whose discontinuity function trajectories are guaranteed to stay on one side of the zero-axis (Fig. 5.2(a)). Some systems containing unilateral discontinuities include coefficient of restitution contact models or electrical systems with idealized gate, transistor or diode models. Specific examples include the bouncing ball problem in section 5, hydraulic valve models in [45, 58], and example 3 in both Carver [29] and [30].

Bilateral discontinuities are those whose trajectories fully cross the zero axis (Fig. 5.2(b)). Examples of bilateral discontinuities are Coulombic friction models [49], spring-damper contact models (e.g. Hertzian or otherwise) [46], or variable structure control systems [34].

5.1.4 Consistent event location with CEL$^+$

The details of guaranteeing consistent event location, referred to here as CEL$^+$, are presented in Park and Barton [56] but are restated here as an



(a) Unilateral

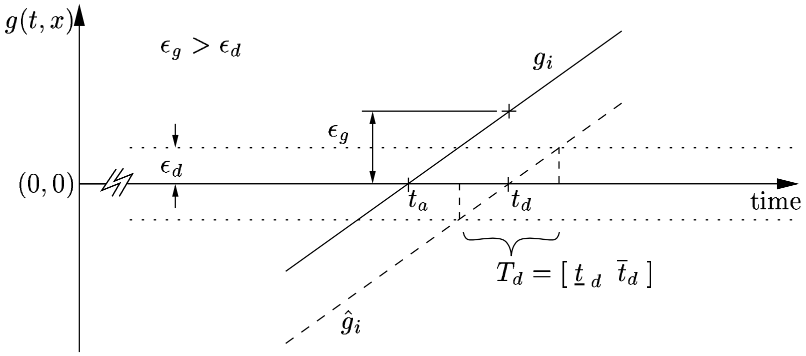
(b) Bilateral

Figure 5.2: Two types of discontinuous events.

introduction to its variant, CEL$^-$. CEL$^+$ guarantees the located event time, t_d, will be *after* the actual event, t_a, to within machine tolerance. This ensures bilateral events are triggered during root location which, in turn, eliminates discontinuity sticking (Fig. 4.1). CEL$^+$ is implemented by finding the root of an auxiliary function slightly offset from g_i,

$$\hat{g}_i = g_i - \epsilon_g \cdot sgn(\dot{g}_i). \tag{5.4}$$

As long as the offset between $\hat{g}_i$ and g_i is larger than the root finding procedure uncertainty then for bilateral events, g_i is guaranteed to lie "on the other side" of zero [1]. Fig. 5.3 shows that g_i is certain to cross zero when t_d is located past the actual event time, t_a. Ensuring $\epsilon_g > \epsilon_d$ allows the root finding procedure can locate t_d anywhere within the interval of uncertainty, T_d, and still provide CEL^+. This is useful not only in solving discontinuous DAE, as in [56], but also solving discontinuous ODE because, in the absence of the guarantee, the same event may cause inefficiency or inaccuracy by getting detected and relocated during the following integration step.



$g(t,x)$
$\epsilon_g > \epsilon_d$
g_i
ϵ_g
ϵ_d
$(0,0)$
t_a
t_d
time
$T_d = [\, \underline{t}_d \;\; \overline{t}_d \,]$
$\hat{g}_i$

Figure 5.3: Locating roots of $\hat{g}_i$ from (5.4) implements CEL^+ which, in turn, guarantees g_i crosses the zero axis and eliminates discontinuity sticking for bilateral events.

5.1.5 Consistent event location with CEL^-

The previous section showed how CEL^+ eliminates discontinuity sticking for bilateral events. However CEL^+ cannot be used for unilateral events

[1] All solutions were found using $|\epsilon_g| = \epsilon_d + \frac{\epsilon_m}{2}$ where ϵ_m is the machine tolerance. For the 32-bit machine used $\epsilon_m \approx 2.22 \times 10^{-16}$.

because their definitions are mutually exclusive. CEL$^+$ guarantees g_i crosses the zero axis while the definition of a unilateral event is one whose final trajectory does not cross the zero axis. This motivates development of CEL$^-$, a slight variation of consistent event location that guarantees g_i will approach but not cross the zero axis. An example of discontinuity sticking for unilateral events is shown in Fig. 5.4. Similar to Fig. 4.1, although the root location tolerance is satisfied, the step from t_{k+1} to t_{k+2} will incorrectly re-detect and re-locate the same event.

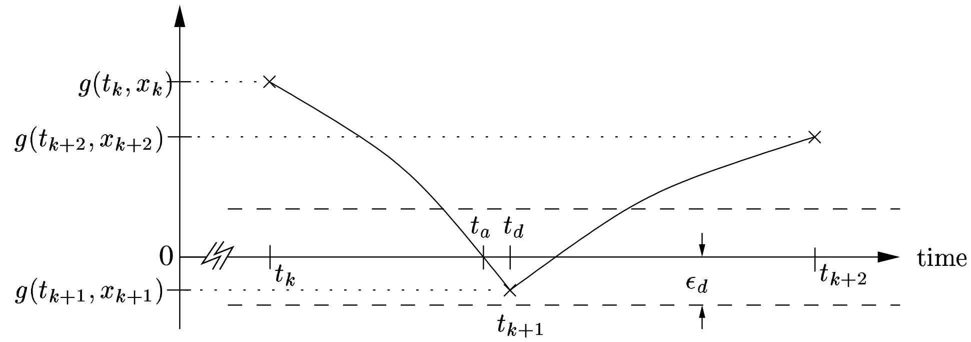


Figure 5.4: For unilateral events, discontinuity sticking occurs when the located time, t_d, is past the actual event time, t_a.

CEL$^-$ is realized by changing the offset direction in (5.4),

$$\hat{g}_i = g_i + \epsilon_g \cdot sgn(\dot{g}_i). \tag{5.5}$$

This guarantees the event time, t_d, is located *before* the actual zero crossing time, t_a, preventing g from crossing the zero axis. By using (5.5), the same root finding procedure that guaranteed CEL$^+$ will now guarantee CEL$^-$ (Fig. 5.5). The ability for a DODE solver to generate a unilateral discontinuity function

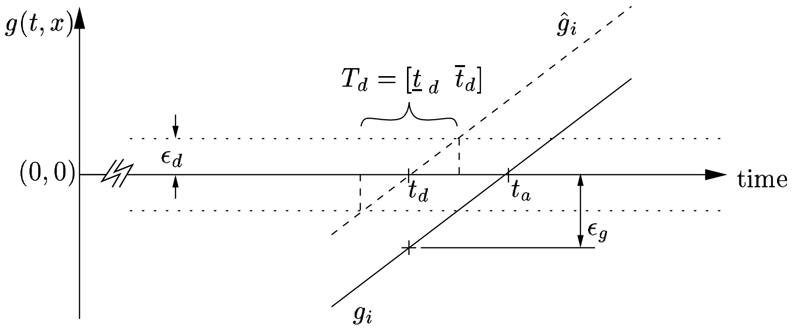


Figure 5.5: Locating roots of $\hat{g}_i$ from (5.5) implements CEL^{-} which, in turn, guarantees g_i *does not* cross the zero axis and eliminates discontinuity sticking for unilateral events.

trajectory has been recognized and implemented by Birta, et. al.[29] in their example problem 3. Other systems that benefit from (or require) CEL^{-} are mechanical systems that use coefficient of restitution contact models, electrical systems containing transistors or diodes, systems with singularities, or any other system where g_i needs to approach some value but not cross it.

5.2 The single-step DODE algorithm

This section outlines the new proposed method for detecting and locating events using information along $[t_k \;\; t_{k+1}]$ only. Special attention is given to both the detection and location phases for overcoming the discrepancies between $g(t,x)$ and it's interpolants (Fig. 4.2).

5.2.1 The event detection phase

The flowchart presented in Fig. 5.6 represents a general single-step integration algorithm with the added capability of detecting and locating all discontinuous events, making discontinuous changes, and restarting integration. The detection and location phases are represented in Fig. 5.6 with blocks 5 and 6a, respectively. All discontinuous changes are made in blocks 6c and 6e, then integration is continued with blocks 7, 8, 2, 3, and 4. After each successful integration step, the detection phase checks for the presence of roots in 3^{rd}-order Hermite interpolants. Interpolants are constructed using the dis-

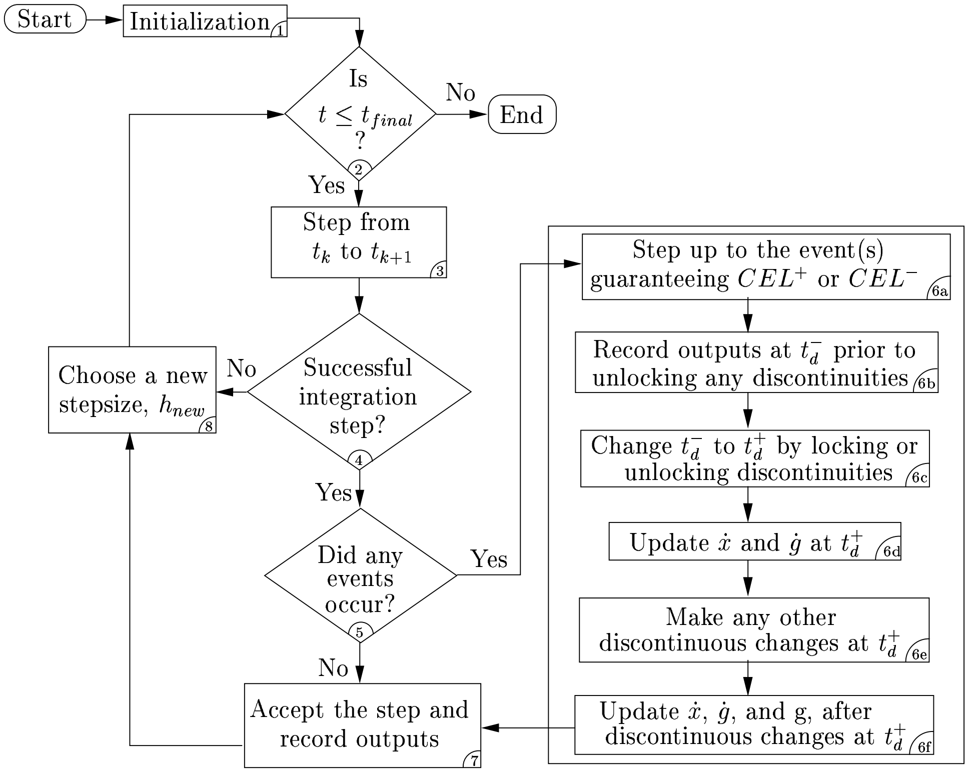

Start
Initialization
1
Is $t \leq t_{final}$?
2
No
End
Yes
Step from t_k to t_{k+1}
3
Successful integration step?
4
No
Choose a new stepsize, h_{new}
8
Yes
Did any events occur?
5
Yes
No
Accept the step and record outputs
7
Step up to the event(s) guaranteeing CEL^+ or CEL^-
6a
Record outputs at t_d^- prior to unlocking any discontinuities
6b
Change t_d^- to t_d^+ by locking or unlocking discontinuities
6c
Update $\dot{x}$ and $\dot{g}$ at t_d^+
6d
Make any other discontinuous changes at t_d^+
6e
Update $\dot{x}$, $\dot{g}$, and g, after discontinuous changes at t_d^+
6f

Figure 5.6: The overall DODE algorithm

continuity function values at both endpoints, $g(t_k)$ and $g(t_{k+1})$, and the rates, $\dot{g}(t_k)$ and $\dot{g}(t_{k+1})$. Similar to [56], a root exclusion test using interval arithmetic [51] is used to eliminate all interpolants guaranteed to contain no roots within $[t_k \;\; t_{k+1}]$. Because of the nature of the exclusion test, this leaves a subset of interpolants that may or may not contain a zero. If this subset is not empty, the roots of all those interpolants are located. Interpolants containing one or more real roots within $[t_k \;\; t_{k+1}]$ are placed in the N_r-set. In addition, error-prone discontinuity functions are included in the N_r-set regardless of whether they contain a root or not. Error-prone interpolants are those whose slopes have different signs at t_k and t_{k+1} and whose curvature points toward the zero-axis. This handles the discrepancy shown in Fig. 4.2(a) by ensuring all g_i are included in the N_r-set that potentially contain a root but whose interpolants do not indicate a root. Discontinuity functions that exhibit higher-order trajectories (i.e. more than one inflection point) may go undetected, however they are unlikely to occur during a single integration step.

Next the solution is advanced to the first estimated event time with numerical integration using a stepsize computed by,

$$h_{d,est} = t_{d,est} - t_k. \tag{5.6}$$

Any single-step integrator that can provide x and $\dot{x}$ at $t+h$ can be used for integration. It is worth noting that several Runge-Kutta pairs were successfully used with the algorithm [2]. A complete analysis of solution accuracy and

[2] The different RK methods include a 2(3) pair[33], Fehlberg's and Dormand-Prince's 45 pairs[3], Fehlberg's 7(8) pair[42], and Hairer and Wanner's implicit 4^{th}-order stiff solver,

efficiency among the different integrators is beyond the scope of this chapter, however the success and functionality implemented during development is evidence that the algorithm is indeed modular and integrator independent.

The stepsize determined by eq.(5.6) is the distance between the current integration time, t_k, and the smallest estimated root from the N_r-set as determined by 3^{rd}-order Hermite interpolants. The smallest root, $t_{d,est}$, is found from a sorted array of 3^{rd}-order roots. If an interpolant like Fig. 4.2(a) contains no roots, the array entry is the the inflection point of $\dot{g}_i$.

The step to $t_{d,est}$ is likely to result in an $x(t_{d,est})$ that does not locate the first event to within the requested discontinuity tolerance, ϵ_d. This is the root location tolerance and, for Newton-Raphson, may be almost as small as the machine tolerance. Fig. 5.7 shows a possible scenario after integration to $t_{d,est}$. At this point the first event can be determined with increased certainty with the extra information available at $t_{d,est}$. All discontinuity functions in the N_r-

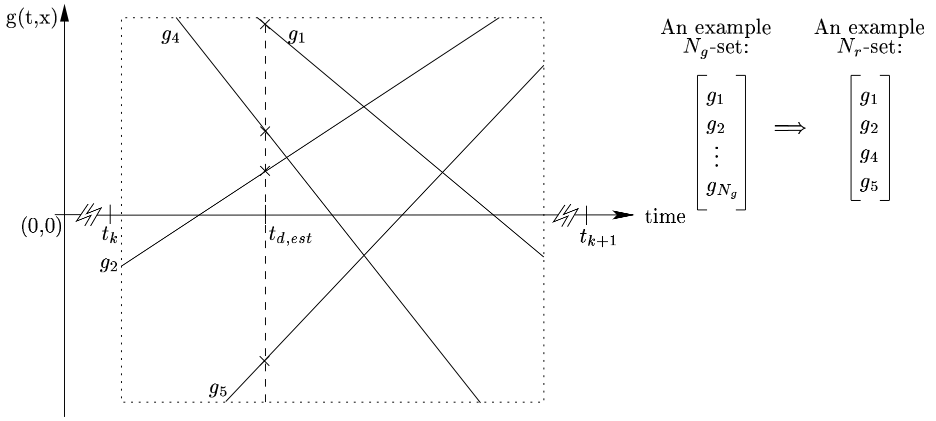


Figure 5.7: An example closeup showing the scenario at $t_{d,est}$.

SDIRK4[8].

set are re-evalulated for root existence and location with either two 3^{rd}-order or one 5^{th}-order Hermite interpolant. Use of two 3^{rd}-order interpolants avoids the Runge effect found in some higher-order interpolants. Nonweiler[54] states that the Runge effect is the unusual deviation from the dataset commonly found when creating high-order interpolant polynomials, especially those with precariously spaced data points.

5.2.2 The root location phase and try-catch model

Since all event detection thus far has used interpolants containing errors as large, or larger than, the state estimates there are no guarantees that any of the indicated roots actually exist in $g(t, x)$. This is the scenario depicted in Fig. 4.2(b). In addition, the interpolant roots have significant uncertainty which makes choosing the first, or leftmost event only an estimate. In summary, there are no guarantees on the existence or location of the “first” event in the N_r-set. However, for the large majority of events, the interpolants provide a good basis for detection and location of roots in $g(t, x)$ and only a small number of exceptions remain to be handled.

Borrowing terminology from programming languages, the following pseudocode in table 5.1 outlines the try-catch model for exception handling. In this case, anything other than locating the first root of the first g_i that crosses zero along $[t_k \;\; t_{k+1}]$ is considered an exception, or manageable error.

The detection phase uses interpolants to estimate zero crossings as well as event order. Then Newton-Raphson, which requires integration for each function evaluation, is used to locate the first estimated event. After the

```
set $i$ to the discontinuity function estimated to cross first
while the $N_r$-set is not empty,
    - locate the root of $g_i$ using Newton-Raphson and numerical integration
      if a root was successfully located,
          - determine if this was the first event or if all prior events lie within
            the ROC
          if this root was the first event, (case I)
                - exit while-loop gracefully (this is the most common case)
          else, (case II, some other event(s) occured prior to the root
            just found)
                - estimate which $g_i$ crossed first
                - prepare to locate this new first estimated event
          end
      else, (case III, $N_{iterations} > N_{itmax} \rightarrow$ no root found)
          - remove this $g_i$ from the $N_r$-set (interpolant discrepancy;
            nonexistent root)
          - prepare for root location on the next $g_i$ in the $N_r$-set
      end
end
```

Table 5.1: Try-catch pseudo-code

Newton-Raphson procedure exits, a post-analysis determines whether or not a root was found and if this was indeed the “first event”. It is this post-analysis, or “catch” phase of the try-catch model that provides a guarantee on locating all events in the correct order. Also the rarity of the exceptions allow the method to remain efficient even though some Newton-Raphson attempts are seemingly wasted. The try-catch model allows less stringent (i.e. more efficient) detection methods to construct the N_r-set at each timestep. This entire procedure is contained in functional block (6a) of Fig. 5.6.

The try-catch procedure outlines a way to handle three possible cases. The first and most common case occurs when the detection phase successfully estimates the first zero crossing in $[t_k \quad t_{k+1}]$. The second case occurs when the detection phase incorrectly estimates the first event. This situation results from the discrepancy between interpolant roots and discontinuity function roots. In these situations the smallest interpolant root is not the first root in $g(t, x)$. The third case handles the scenario depicted in Fig. 4.2(b) where an interpolant indicates the presence of a root in $g(t, x)$ when there actually is none.

5.2.3 Incorporating a region of concurrency

The means by which CEL$^+$ improves efficiency is first, by preventing discontinuity sticking which results in fewer rejected integration steps which in turn produces shorter solution times. However, for certain systems the benefits can be outweighed by the drawbacks. Park and Barton's algorithm guarantees location of each individual event but, for systems with many closely spaced events this effecvitvely guarantees very small step sizes. Small step sizes are one of the very things a discontinuity handling procedure aims to avoid. In some cases, the resulting stepsizes are so small, a standard smooth ODE solver might traverse the discontinuities with comparable or larger stepsizes. Or, even if stepsizes were roughly the same size, a smooth ODE solver not burdened with the computational overhead associated with discontinuity processing might produce a reasonably accurate solution in shorter time.

The region of concurrency (ROC) is a modification to CEL$^+$ that retains the benefits while at the same time allows the engineer to make a tradeoff between locating individual event times and allowing multiple closely spaced events to occur "at the same time." For systems whose bilateral discontinuous events may be allowed to occur “at the same time” without an intolerable loss of accuracy, the ROC provides the ability for many closely spaced events to be triggered during a single integration step. Using the ROC with systems that contain nearly concurrent events, only one integration step is necessary to trigger N events at N closely spaced times. In contrast, CEL^+ is more costly requiring N attempted integration steps at N distinct times even if the events are separated by 1×10^{-10} or 1×10^{-14}. It should be noted that the ROC is useful for bilateral discontinuities only. The definition on which unilateral events is based is mutually exclusive with the premise on which the ROC is based. This underlying premise states that if CEL^+ guarantees event location “slightly” after t_a, then if t_d can be located “slightly farther” past t_a without causing integration step failure, this may allow multiple events to occur during the same timestep.

The process of detecting and locating discontinuous events may be viewed as a sequential reduction of the N_g-set down to the (frequently empty) N_r-set (Fig. 5.8). Implementing the ROC is a two-phase procedure for reducing the N_r-set to the N_p-set, then the N_c-set. Phase one estimates how many events might occur once the root finding procedure locates an event time. These calculations occur between blocks 5 and 6a of Fig. 5.6 and are relatively inexpensive because they assume all g_i behave linearly in the neighborhood of $t_{d,est}$ and require no RHS function evaluations.

Without the ROC, the root finding procedure locates the root of the *first* estimated event after stepping up to $t_{d,est}$. With the ROC, the N_p-set is extracted from the N_r-set by including all estimated event times that lie within

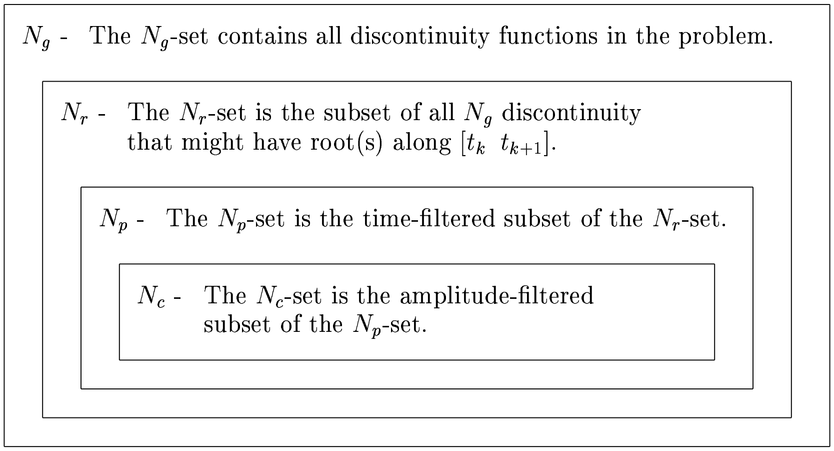

N_g - The N_g-set contains all discontinuity functions in the problem.
N_r - The N_r-set is the subset of all N_g discontinuity that might have root(s) along $[t_k \;\; t_{k+1}]$.
N_p - The N_p-set is the time-filtered subset of the N_r-set.
N_c - The N_c-set is the amplitude-filtered subset of the N_p-set.

Figure 5.8: Sets used within the DODE algorithm

ϵ_c of the first event. The N_p-set is a time-filtered subset of the N_r-set. Next, the N_p-set is distilled to an estimate of what the N_c-set will be by including all elements of the N_p-set whose linear estimates of $|\hat{g}_i| \leq \epsilon_{ROC}$. This is essentially an amplitude-filtered subset of the N_p-set (Fig. 5.9). The end goal for reducing the N_r-set to the estimated N_c-set is to choose the discontinuity function with the greatest estimated event time while still preserving all consistent event location guarantees. This is implemented by determining which $\hat{g}_i$ should be passed to the root finding procedure for root location. For the example shown in Fig. 5.9 Newton-Raphson would be used to locate the root of $\hat{g}_4$.

The second phase in constructing the ROC is an appraisal of how many events actually occurred after the root finding procedure has located the final event time, t_d. This is done by evaluating $g(t_k, x_k)$ and $g(t_{t_d}, x_{t_d})$ with the definition of L in (5.3). These calculations occur in block 6c of Fig. 5.6.

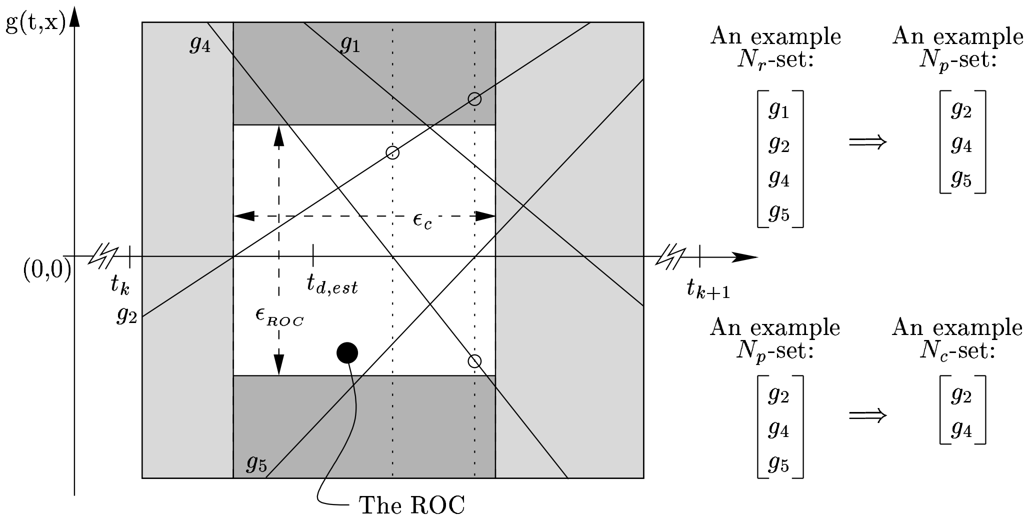

g(t,x)
g_4
g_1
ϵ_c
(0,0)
t_k
$t_{d,est}$
t_{k+1}
g_2
ϵ_{ROC}
g_5
The ROC
An example N_r-set:
$\begin{bmatrix} g_1 \\ g_2 \\ g_4 \\ g_5 \end{bmatrix} \Longrightarrow \begin{bmatrix} g_2 \\ g_4 \\ g_5 \end{bmatrix}$
An example N_p-set:
An example N_p-set:
$\begin{bmatrix} g_2 \\ g_4 \\ g_5 \end{bmatrix} \Longrightarrow \begin{bmatrix} g_2 \\ g_4 \end{bmatrix}$
An example N_c-set:

Figure 5.9: Potentially concurrent events (i.e. the N_p-set) lie within ϵ_c of the first event. The rightmost event estimated to cause all leftward events to lie within the ROC (i.e. the $N_c - set$ estimate) is chosen for root location.

5.3 Benchmark problems

The problems chosen to demonstrate the algorithm not only highlight the method's specific features but also provide a means of comparison with other solvers. Problems 1, 2, and 3 from both Birta et.al. (BOK) and Carver are presented as benchmark problems solved elsewhere in the literature [29, 30, 56]. The 100-bouncing-balls problem is shown as a simple example of what constitutes "nearly concurrent" events. It also demonstrates the potential for improvement provided by the region of concurrency for systems with many concurrent events. The marble jar problem is designed to provide a challenging set of DODE similar to what might be found in an industrial multibody dynamics problem. This problem proved to be the most difficult problem to solve and revealed several shortcomings of previous solver versions that performed exceptionally well on the first six benchmark problems. This previous version's detection and location methods were not robust enough to handle the numerous degenerate cases presented by the marble jar problem.

All problems are solved with the Dormand-Prince RK-5(4) pair, $h_{max} = 0.1(s)$, and the following (smooth) stepsize choice:

$$h_{new} = min\left[h_{max}, \quad 0.8 \cdot h \cdot \left(\frac{\tau}{\delta}\right)^{\frac{1}{6}}\right] \tag{5.7}$$

where δ is the estimated locate truncation error, τ is the allowable error, and tol is the requested integration tolerance,

$$\delta = \|x_5 - x_4\|_\infty, \quad \tau = tol \cdot max(\|x\|_\infty, 1.0). \tag{5.8}$$

Unless otherwise stated, the stepsize chosen immediately following an event is $min(h_0, h_{smooth})$. This chooses the smaller of either the initial stepsize, $h_0 = (t_{final} - t_{initial}) \cdot 1 \times 10^{-5}$, or the stepsize specified by the smooth integrator. Table (5.2) outlines the benchmark problems and their characteristics. Individual tables within the following subsections show the algorithm's performance on each problem.

5.3.1 The BOK problems

Problems 1 and 3 in Birta et.al.[29] have exact analytic solutions which provide the "correct" answer useful for evaluating a numerical solution. Although they provide an explicit solution for problem 2, it is suspected that these values were arrived at via numerical integration and thus are affected by integrator error. Regardless, they present the problems and provide means for comparison through N_{rhs} and the final state values. The results for this algorithm's performance are shown in table(5.3) for integration tolerance $\epsilon_x = 1 \times 10^{-5}$ and root location tolerance $\epsilon_d = 1 \times 10^{-10}$. For comparison purposes, in the following tables, N_{rej} is an accumulation of unused integration steps not only from oversized (smooth) stepsizes. In addition, it includes rejected steps to $t+h$ once the detection process initiates a step to $t_{d,est}$. It also includes rejected steps to $t_{d,est}$ after the event detection process verifies an empty N_r-set.

One advantage the current method has over that presented in Birta et.al. is the ability to locate events down to machine tolerance. Incorporating a Newton-Raphson method during the location phase is not a new idea as evidenced by their explicit solution for problem 2. However, the incorporation of CEL^{+} and CEL^{-} made high accuracy root location viable although not a requirement through the user-specifiable root location tolerance, ϵ_d. For the BOK problems, the penalty in N_{rhs} incurred by Newton-Raphson was offset by the improvements provided by the Dormand-Prince RK-5(4) pair.

The results presented in table 5.3 reflect improved solution efficiency for problems #1 and #2 and reduced efficiency for #2.

5.3.2 The Carver problems

Carver solved 4 example problems using a method based on Gear's stiff solver published by Hindmarsh (see [30]). This is a multi-step predictor corrector with Nordsieck stepsize control. The accuracy with which event times are located is not stated explicitly, however from his eqs.(3) and (4), it is presumed equal to the integration tolerance, $\epsilon_x = 1 \times 10^{-5}$. Results for the present algorithm's performance on his first three problems are shown in Fig. 5.4 with both integration tolerance and root location tolerance at 1×10^{-5}.

Carver's results for problem 1 are ambiguous however, for problems 2 and 3, the results presented here are less efficient. He presents $N_{rhs} = 387$ and 419 for problems 2 and 3, respectively. His greater efficiency is primarily attributed to the availability of inexpensive and sufficiently high-order interpolants provided by the multi-step integrator. These interpolants not only guarantee event detection, but also allow root location with no extra function evaluations.

The success found by Carver is impressive and only offset by the lack of features like consistent event location (either CEL^{+} or CEL^{-}) and the fact that his method does not have an integrator independent root location mechanism. This makes locating events to within machine tolerance difficult if at all possible. Because the discontinuity functions are appended to the integration state vector, specifying high accuracy in event location would likely become the limiting factor in stepsize choice. For larger systems, appending $\dot{g}$ to $\dot{x}$

and integrating ($N_{states} + N_g$) ODE's would significantly increase the cost of each integration step. This may, at some point, become prohibitively costly irrespective of event location tolerance.

5.3.3 100 bouncing balls

This problem is intended as a simple demonstration for the region of concurrency. Since the ROC is based on linear estimates it is intended to improve performace on systems that contain several events (very) closely spaced along the time axis. For example, events originally intended to occur "at the same time" are not always triggered during the same step because of roundoff, integration, or root location errors.

The system is an independent group of bouncing balls that undergo spring-damper contact individually with the ground only. There are 100 balls falling from initial heights, each offset by $\Delta y = 1 \times 10^{-9}$ from the previous resulting in groups of 100 events separated by $\approx 2.26 \times 10^{-10}(s)$.

By using the ROC, several events are allowed to occur within the same timestep. Table (5.5) shows performance improvements between a smooth ODE solver, the new algorithm without the ROC, and the new algorithm with the ROC. Although the problem may be solved with unilateral or bilateral events, the ROC only provides improvements for bilateral events.

Integration and event location tolerance were maintained at $\epsilon_x = 1 \times 10^{-3}$ and $\epsilon_d = 1 \times 10^{-10}$, and the ROC tolerances were $\epsilon_c = 1 \times 10^{-7}$, $\epsilon_{ROC} = 1 \times 10^{-6}$. It should be apparent that the ROC provides significant reduction in the number of RHS-evaluations compared to both the smooth solver and the discontinuous solver without the ROC.

5.3.4 The marble-jar

This system describes tangentially frictionless spring-damper contact between N_{balls} marbles and a jar-like container modeled with the intersection of a plane and cylinder.

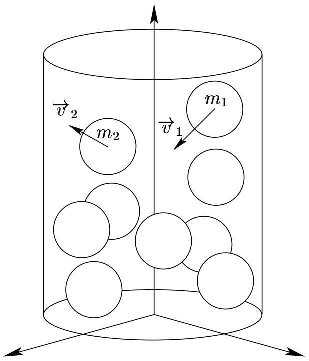


Figure 5.10: Simple DODE system modeling glass marbles being poured into a glass jar.

The system was designed to generate DODE for rigorously testing the solver's detection and location capabilities. The number of integrated states is $2N_{balls}$ and the discontinuity functions are distances between each marble and all other marbles as well as the plane and cylinder. With $N_{walls} = 2$ representing the cylinder and plane, the total number of discontinuity functions increases with the number of marbles with

$$N_g = N_{balls}^2 + N_{balls}N_{walls} - \sum_{i=1}^{N_{balls}} i \tag{5.9}$$

This problem serves to underscore the importance of verifying a *correct* solution in addition to improving solution efficiency. For no losses, the system's total energy should remain constant and is used as a scalar metric for assessing solution quality. With no losses, the total system energy is

$$E_k = \sum_{i=1}^{N_{balls}} \left(m_i g h_i + \frac{1}{2} m_i v_i^2 \right) + \sum_{j=1}^{N_g} \frac{1}{2} k_j \delta_j^2. \tag{5.10}$$

Fig. 5.11 and table 5.6 show improvements in both accuracy and efficiency of the new algorithm versus a smooth ODE solver.

The trajectories in Fig. 5.11 were made with $N_{balls} = 10$, $\epsilon_d = 1\times10^{-10}$ and requested integration tolerances of 1×10^{-3} and 1×10^{-6}. Although the four cases shown all exhibit some change in energy, the ΔE in the new algorithm can be attributed primarily to the underlying integration method. The smooth solver's ΔE results from a combination of the underlying integration method accompanied by the results of discontinuity "hunting".

Figs.5.12(a) and 5.12(b) shows the reduction in N_{rhs} and cpu-time provided by the new algorithm over a smooth solver for N_{balls}=2,4,8, and 16. From these plots, it is apparent that the new algorithm reduced the total number of RHS function evaluations by a factor of ≈4.6 for $N_{balls} = 2$ and ≈2.8 for $N_{balls} = 16$. (note: $4.6 \approx 2^{(15.8-13.6)}$)

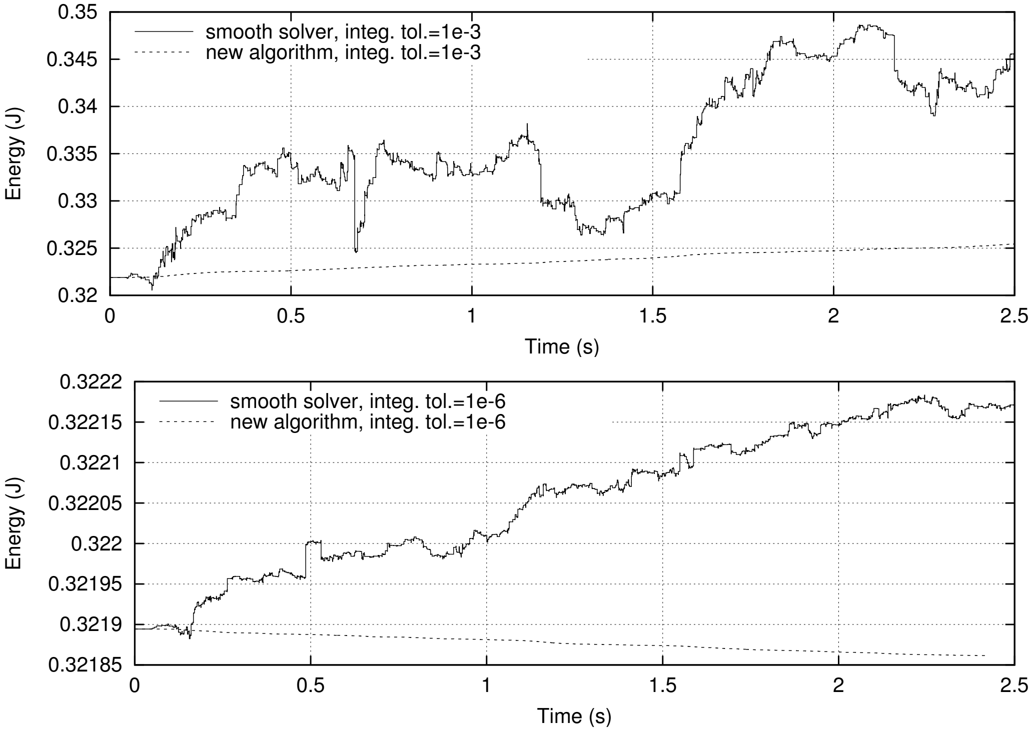


Figure 5.11: This algorithm improves solution quality compared to a smooth RK-5(4) ode solver.

5.4 Conclusions

Solving discontinuous ODE is a challenging task complete with issues in equation formulation, event detection, location, and solution verification. Given the ability to arbitrarily manipulate the state vector at each event, this automated numerical method is subject to greater risk of failure but also carries with it the potential for significant gains in both accuracy and efficiency. These two chapters presented a DODE solution method based on a generic single-step integrator. Ideas previously presented in the literature such as discontinuity locking, discontinuity sticking, and consistent event location were restated to lay the groundwork for further development. Assuming the use of discontinuity locking through a status vector, L, events were characterized as either structural or parametric in their discontinuous changes. With the help of Fig. 5.8, section 5.1.2 explained that parametrically discontinuous systems do not necessarily require L in their formulation, while structurally discontinuous system equations are conveniently formulated with the use of L.

Discontinuous events were also characterized as either unilateral or bilateral. The underlying causes of discontinuity sticking were extended to both types of events. Two forms of consistent event location, CEL^+ and CEL^- were presented as solutions to eliminate discontinuity sticking in bi- and unilateral systems.

Detecting and locating all events in the correct order while maintaining reasonable efficiency is particularly difficult using low accuracy interpolants. The primary difficulty in *detecting* all events is shown in Fig. 4.2(a). This is effectively dealt with by populating the N_r-set with all interpolants that contain roots or satisfy a test identifying it as error-prone. The event *location* problem, separate from the detection problem, contains primarily two difficulties, both of which are effectively handled using the try-catch model. The try-catch model uses Newton-Raphson and post-analysis to manage two types of exceptional cases during the root location process. The first is the problem of nonexistent roots and the second is incorrectly sequenced roots as predicted by the interpolants. The first problem is illustrated in Fig. 4.2(b) and both exceptional cases are handled by the pseudocode in section 5.2.2.

A region of concurency is presented as an efficient means for allowing multiple closely spaced events to occur during a single integration step. The ROC is only useful for bilateral events because the premise on which it is based excludes it from use with unilateral events.

Finally the example problems demonstrate the feasibility of the single-step method using low-order interpolants. The first six problems represent a benchmark set of DODE previously published by Carver and Birta et.al. The results of the new method on these problems revealed a mixture of increased and reduced efficiency, as measured by N_{RHS}. This is primarily attributed to the multi-step integrator's provision of interpolants with guaranteed accuracy. This decrease in performance is, however, offset by extra features. These include consistent event location for both uni- and bilateral events, the flexibility of integrator independence, the ability to locate roots to within machine tolerance independent of integrator tolerance, and a framework that does not increase the number of integrated states with the number of discontinuity functions. The 100 bouncing ball problem not only demonstrated the degradation in performance from multiple closely spaced events but also the improvements provided by the ROC. The marble jar problem provided a challenging set of DODE on which to test the method's detection and location capabilities. In the absence of an analytic solution, an energy metric was used to verify solution accuracy, and for the *smooth* solver at $\epsilon_x = 1 \times 10^{-3}$, indicated a failure that otherwise may have gone unnoticed. The method generated solutions that reduced the number of RHS function evaluations over comparable solutions from a smooth ODE solver by factors ranging from 1.4-4.2 for $N_{balls} = 10$ and 4.6-2.8 for $N_{balls} = 2 - 16$. It also produced more accurate results with roughly an order of magnitude smaller variations in total system energy for undamped scenarios.

Problem Name	N_{states}[a]	N_g[b]	g-type[c]	Feature
Carver #1	1	1	bi	requires an appropriate h_{max} to avoid missing events; $f = 2$
Carver #2	1	2	bi	parametric discontinuities (see section 5.1.2)
Carver #3	4	4	bi	challenging IC's; required i_1 and i_2 to be small but ≥ 0; equation structure discontinuities
BOK #1	3	1	bi	likely to cause schemes to fail that rely upon sign changes in g; parametric discontinuities
BOK #2	2	1	bi	second-order damped dynamics; event times become increasingly close as t increases
BOK #3	3	2	uni	requires CEL^{-} to maintain $g(t, x) \geq 0$; event times become increasingly close as t increases
bouncing balls	200	100	uni/bi	ball-ground contact only; spring-damper contact model $\mapsto$ bilateral events; coefficient of restitution contact model $\mapsto$ unilateral events
marble-jar, N_{balls}=2,4,8,10,16	4,8,16, 20,32	5,14,44, 65,152	bi	simulates pouring marbles into a glass jar; spring-damper contact models

[a] N_{states}: number of integrated states in (5.1)

[b] N_g: number of discontinuity functions in (5.2)

[c] g-type: discontinuity type, either bilateral or unilateral

Table 5.2: Discontinuous problem profiles

Problem (A=)	x_{final}[a]	N_{ev}[b]	N_{rhs}[c]	N_{acc}[d]	N_{rej}[e]	N_{it}[f]	N_{ctch}[g]	N_{fl}[h]
BOK #1, 0.35	0.8554069	3	307	42	3	5	0	0
BOK #1, 0.40	0.8000413	3	307	41	3	6	0	0
BOK #1, 0.41	0.7819808	1	237	35	2	2	0	0
BOK #1, 0.45	0.7432341	1	231	35	1	2	0	0
BOK #2	0.9427525	16	1041	123	16	29	0	0
BOK #3	0.1653847	20	725	58	20	36	0	0

[a] x_{final}: relevant state value at t_{final} for comparison with BOK results

[b] N_{ev}: total number of events located

[c] N_{rhs}: total number of right-hand-side function evaluations

[d] N_{acc}: total number of steps accepted

[e] N_{rej}: total number of steps rejected

[f] N_{it}: total number of Newton-Raphson iterations

[g] N_{ctch}: summation of case II occurrences sucessfully handled by the try-catch model

[h] N_{fl}: summation of case III occurrences sucessfully handled by the try-catch model

Table 5.3: Solver performance on the BOK problems.

Problem	N_{ev}	N_{rhs}	N_{acc}	N_{rej}	N_{it}	N_{catch}	N_{fl}
Carver #1,[a]	3	229	30	3	4	0	0
Carver #2	7	447	53	10	9	0	0
Carver #3[b], no ROC	22	873	88	27	23	0	0
Carver #3, ROC[c]	16	741	79	21	18	0	0

[a] $f = 2, t \in [0 \;\; 0.9]$, and after each event, $h_{restart} = 1 \times 10^{-6}$

[b] This problem required integration of $d^2 i_1/dt^2$ and $d^2 i_2/dt^2$ to fit the form $g = g(t, x)$

[c] $\epsilon_c = 1 \times 10^{-6}(s), \epsilon_{ROC} = 0.01$

Table 5.4: Solver performance on the Carver problems.

Solution method on 100 bouncing balls problem	N_{ev}	N_{rhs}	N_{acc}	N_{rej}	N_{it}	N_{catch}	N_{fl}
smooth ODE solver[a]	-	4963	401	426	-	-	-
new method, no ROC	1200	31423	1239	1200	2398	0	0
new method, with ROC	18	685	57	18	33	0	0

[a] based on the Dormand-Prince RK-5(4) pair

Table 5.5: Performance results for the 100 bouncing ball problem.

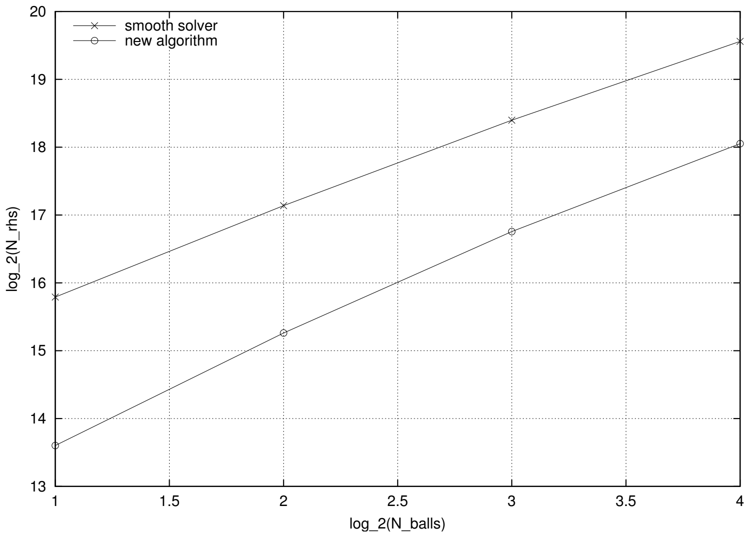
Solver	ϵ_x	ΔE[a]	N_{ev}	N_{rhs}	N_{acc}	N_{rej}	N_{it}	N_{catch}	N_{fl}
smooth[b], undamped	1e-3	2.81e-2	-	414025	36326	32679	-	-	-
new meth., undamped[c]	1e-3	3.55e-3	2272	98385	7653	3334	4653	7	0
smooth, undamped	1e-6	3.01e-4	-	439147	40324	32868	-	-	-
new meth., undamped	1e-6	3.34e-5	2088	132283	15189	2237	3925	0	0
smooth, damped	1e-3	failed	-	-	-	-	-	-	-
new meth., damped	1e-3	0.2937	13404	471259	32225	17876	23974	2	3
smooth, damped	1e-6	0.2939	-	624367	68810	35252	-	-	-
new meth., damped	1e-6	0.2939	5663	438329	50733	10593	9841	0	0

[a] $\Delta E = max(E) - min(E)$ and represents the total energy variation.

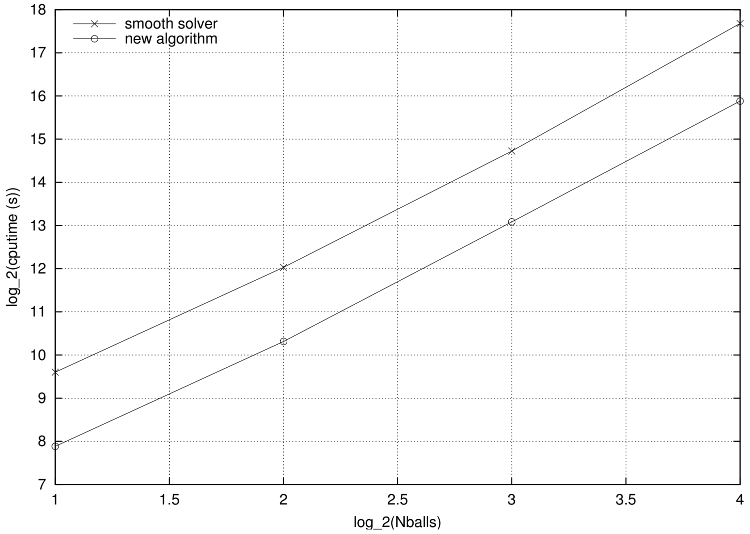
[b] smooth ODE solver based on Dormand-Prince RK-5(4) pair.

[c] All new method results are for $\epsilon_d = 1\times10^{-10}$, no ROC, and $h_{max} = 0.1$(s).

Table 5.6: Comparison of discontinuous and smooth solver efficiency on marble-jar problem.



(a) $log_2(N_{rhs})$ for N_{balls}=2,4,8, and 16.



(b) log_2(Cpu-times) for N_{balls}=2,4,8, and 16.

Figure 5.12: Parameter variation of N_{balls} in the marble jar problem.

Chapter 6

Conclusions and future work

6.1 Conclusions

Two solution methods were developed for solving a class of semi-explicit, high-index DAE and discontinuous ODE. Both methods demonstrated improved solution efficiency or accuracy (or both) over existing methods. In general, the costs associated with solving DAE and DODE were reduced enough to warrant further work. The development environment in which the numerical experiments were performed was in no way optimal. Significant efficiency improvements may be found by reimplementing both methods in a higher performance computational environment.

A hybrid DAE solution method was developed in chapters 2 and 3 that draws on strengths from computational mathematics-based DAE and sliding mode control theory. Fig. 6.1 is a graphical depiction of the major issues contributed by both SMC control theory and DAE solution methods as well as the contributions reflected back upon the two original bodies of literature. An explicit transformation was made between the mathematics form of con-

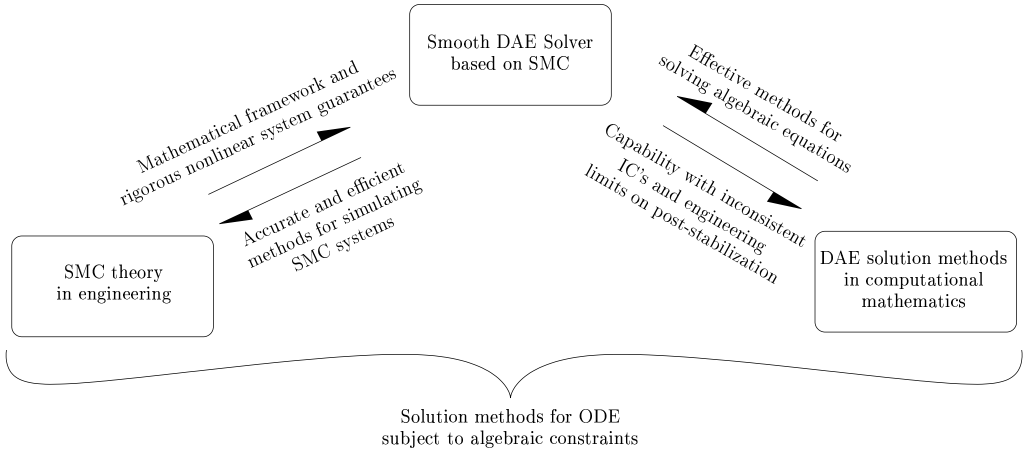


Figure 6.1: Major contributions transferred both to and from the original bodies of literature.

strained multibody dynamics DAE to an equivalent control problem in canonical state-space form in the control literature. SMC's ability to directly handle time-varying, nonlinear, holonomic and nonholonomic systems under a unified framework made it ideal as a control strategy for solving MBS DAE. In addition, SMC's reaching-phase dynamics naturally accommodates inconsistent initial conditions which otherwise is a known difficulty when solving DAE. Integration error was used as an upper bound for limiting post-stabilization adjustment in an attempt to produce physically realistic soltions. Finally, an acceleration-level stabilization method was identified from study of SMC's boundary layer dynamics and used during each RHS function evaluation. It was shown to be both necessary for correct solutions and analugous to computing all constraint Jacobians at each integrator stage value.

A DODE solution method was presented in Chapters 5 and 4 using a generic single-step integrator. Previous techniques such as discontinuity locking, interpolant creation, and consistent event location were implemented in the single-step environment. Fig. 6.2 is a graphical depiction of the major issues contributed by current multi-step methods in the literature and this single-step DODE solver. A distinction was drawn between how discontinuous equations are written. They can be parametric or structural in their formulation. This was followed by further classification of discontinuities into uni- and bi-lateral events along with their related mechanisms for discontinuity sticking. The process of (re)implementing past DODE technologies in a single-step environment led to further developments such as CEL$^-$, the try-catch method, and the region of concurrency. Unlike multistep methods that

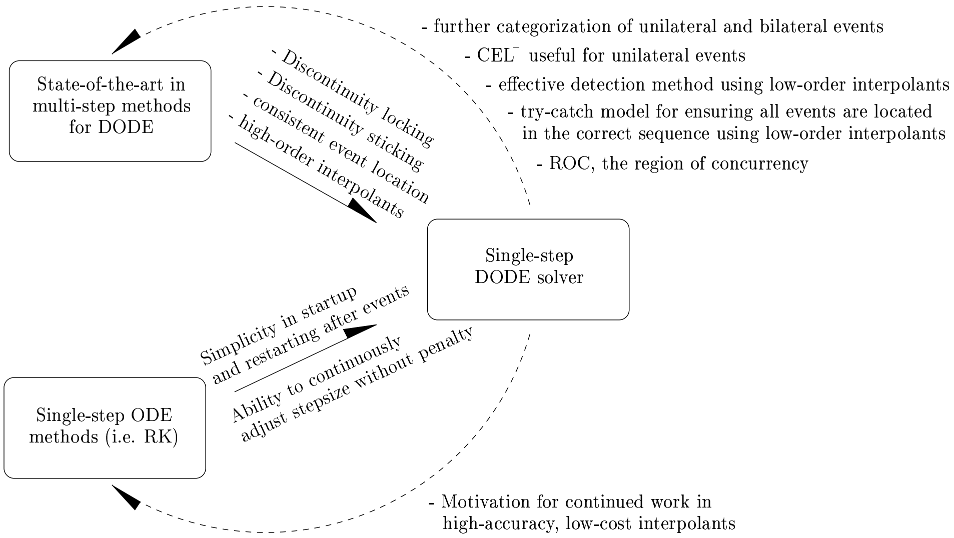

State-of-the-art in multi-step methods for DODE
- Discontinuity locking
- Discontinuity sticking
- consistent event location
- high-order interpolants
- further categorization of unilateral and bilateral events
- CEL$^-$ useful for unilateral events
- effective detection method using low-order interpolants
- try-catch model for ensuring all events are located in the correct sequence using low-order interpolants
- ROC, the region of concurrency
Single-step DODE solver
Simplicity in startup and restarting after events
Ability to continuously adjust stepsize without penalty
Single-step ODE methods (i.e. RK)
- Motivation for continued work in high-accuracy, low-cost interpolants

Figure 6.2: Major contributions contributed from previous work and this single-step DODE solver.

provide high-order interpolants, low-order interpolants were shown largely successful for event detection in the single-step envirnment. In the situations where low-order interpolants failed, a try-catch method of post-analysis was used to catch two specific failure cases during event location. Six benchmark problems were solved that revealed a mix of improved and decreased performance. The performance decrease is attributed primarily to the provision of high-order interpolants provided by multistep integrators.

6.2 Future work

The original intent in developing a single-step DAE solver along with a single-step, integrator-independent DODE solver was ultimately to combine the two, along with a stiff integrator, into a single solver (Fig. 6.3). This solver would represent the state-of-the-art in single-step methods for solving stiff discontinuous, high-index DAE. This class of equations is one of the most challenging classes to solve in physical system simulation. Currently the only other solving environment that directly addresses this class of equations and also implements discontinuity locking and consistent event location is DAEPACK[36]. DAEPACK, among other things, is a software implementation of Park and Barton's paper [56].

Future work in DAE solution methods include extension of SMC's capabilities to a broader, more general class of DAE. Also, post-stabilization apparently exhibits some growth in it's adjustment magnitude for long-time simulations. Investigating the reasons for and attenuating this growth is an

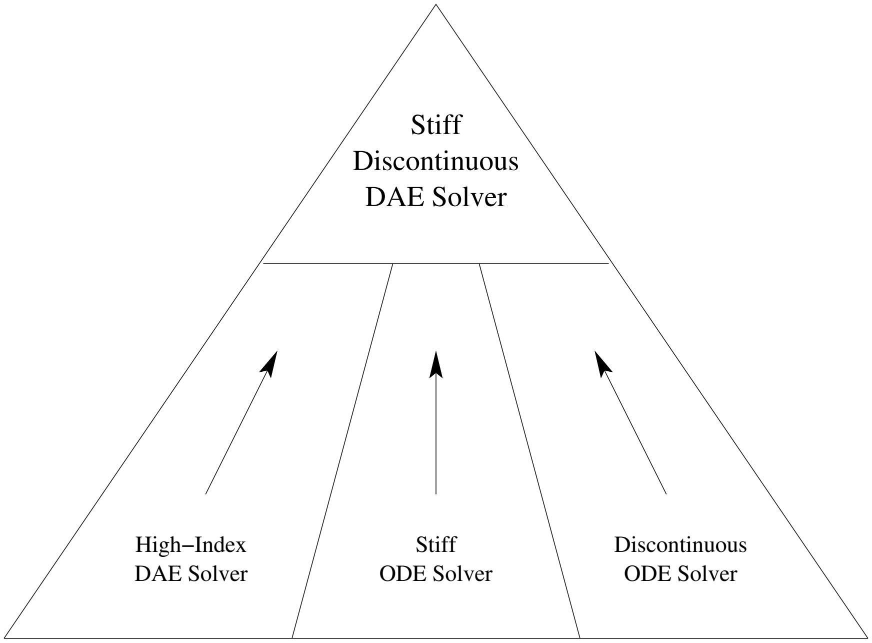


Figure 6.3: The original intent of creating single-step DAE and DODE solvers.

area for future work.

Other future work involves improving the detection process and, possibly providing detection guarantees using some combination of logic and the high-order interpolants developed by Enright et.al. in [40] and [41]. Also, implementing the region of concurrency and/or CEL^{-} in a multi-step environment could not only provide increased efficiency for a multi-step DODE solver, but would also provide the functionality for solving systems with unilateral discontinuities.

Lastly, some equation solving environments that might benefit from consistent event location (either CEL^{+} or CEL^{-}) and discontinity locking include Matlab, ACSL, the solvers at `netlib.org`, DAEPACK, and the dynamic systems modeling environments ADAMS[1] [26], DADS[2] [35], MSC.visualNastran 4D[3] [63] and EASY5[38].

[1] ADAMS is a registered trademarks of Mechanical Dynamics, Inc.

[2] DADS is a registered trademarks of LMS International

[3] MSC.visualNastran 4D is a registered trademark of The MSC.Software Corporation

Bibliography

[1] U.M. Ascher, H. Chin, L.R. Petzold, S. Reich, 1995, "Stabilization of Constrained Mechanical Systems with DAEs and Invariant Manifolds", *Mech. Struct, and Mach.*, v.23, no.2, pp.135-157

[2] U.M. Ascher, L.R. Petzold, 1993, "Stability of Computational Methods for Constrained Dynamics Systems", *SIAM J. Scientific Computing*, v.14, no.1, pp.95-120

[3] U.M. Ascher, L.R. Petzold, 1998, *Computer Methods for Ordinary Differential Equations and Differential-Algebraic Equations*, Society for Industrial and Applied Mathematics (SIAM), Philadelphia

[4] J. Baumgarte, 1972, "Stabilization of Constraints and Integrals of Motion in Dynamical Systems", *Computer Methods in Applied Mechanics and Engr.*, v.1, pp.1-16

[5] E.J. Haug, D. Negrut, M. Iancu, 1997, "A State-Space-Based Implicit Integration Algorithm for Differential-Algebraic Equations of Multibody Dynamics", *Mechanics of Structures and Machines*, v.25, no.3, pp.311-334

[6] V. Stejskal, M. Valasek, 1996, *Kinematics and Dynamics of Machinery*, Marcel Dekker Inc., New York

[7] E. Hairer, S. N⊘rsett, G. Wanner, 1993, *Solving Ordinary Differential Equations I, Nonstiff Problems*, 2nd Revised Ed., Springer, Berlin

[8] E. Hairer, G. Wanner, 1996, *Solving Ordinary Differential Equations II, Stiff and Differential-Algebraic Problems*, 2nd Revised Ed., Springer, Berlin

[9] J. Rismantab-Sany, A.A. Shabana, 1988, "Impulsive Motion of Non-Holonomic Deformable Multibody Systems, Part II: Impact Analysis", *J. Sound and Vibration*, v.127, no.2, pp.205-219

[10] X. Yun, N. Sarkar, 1998, "Unified Formulation of Robotic Systems with Holonomic and Nonholonomic Constraints", *IEEE Trans. Rob. Automation*, v.14, no.4, pp.640-650

[11] N.H. McClamroch, 1990, "Feedback Stabilization of Control Systems Described by a Class of Nonlinear Differential-Algebraic Equations", *Systems and Control Letters*, v.15, pp.53-60

[12] N.H. McClamroch, 1990, "On Control Systems Described by a Class of Nonlinear Differential-Algebraic Equations: State Realizations and Local Control", *Proc. of 1990 American Control Conference*, v.2, pp.1701-1706

[13] J.C. Chiou, S.D. Wu, 1998, “Constraint Violation Stabilization Using Input-Output Feedback Linearization in Multibody Dynamic Analysis”, *J. Guidance, Control, and Dynamics*, v.21, no.2, pp.222-228

[14] B.W. Gordon, S. Liu, H.H. Asada, 1999, ”State Space Modeling of Differential-Algebraic Systems Using Singularly Perturbed Sliding Manifolds,” DSC-Vol.67, *Proceedings of the ASME Dynamic Systems and Control Division-1999*, pp.537-544

[15] F. Zhao, V. Utkin, 1996, “Adaptive Simulation and Control of Variable-structure Control Systems in Sliding Regimes”, *Automatica*, v.32 no.7, pp.1037-1042

[16] R.A. DeCarlo, S. Drakunov, 1998, “A Unified Lyapunov Setting for Continuous and Discrete Time Sliding Mode Control”, *Proc. of the ASME, DSC-Vol. 64*, pp.547-554

[17] V.I. Utkin, 1992, *Sliding Modes in Control Optimization*, Springer-Verlag, Berlin

[18] V.I. Utkin, J.Guldner, J. Shi, 1999, *Sliding Mode Control in Electromechanical Systems*, Taylor and Francis, London

[19] G. Bartolini, A Ferrara, V.I. Utkin, 1995, “Adaptive Sliding Mode Control in Discrete-time Systems,” *Automatica*, v.31, no.5, pp.769-773

[20] W.H. Enright, K.R. Jackson, S.P. N⊘rsett, 1986, "Interpolants for Runge-Kutta Formulas", *ACM Transactions on Mathematical Software*, v.12, no.3, pp.193-218

[21] B.J. Leimkuhler, 1998, "Approximation Methods for the Consistent Initialization of Differential-Algebraic Equations", Ph.D. dissertation, Dept. Comp. Sci., Univ. Illinois Urbana

[22] C.C. Pantelides, 1998, "The Consistent Initialization of Differential-Algebraic Systems", *SIAM J. Scientific and Statistical Computing*, v.9, no.2, pp.213-231

[23] J. Slotine, W. Li, 1991, *Applied Nonlinear Control*, Prentice Hall, New Jersey

[24] H. Chin, 1995, "Stabilization Methods for Simulations of Constrained Multibody Dynamics," Ph.D. thesis, Institute of Applied Mathematics, Univ. of British Columbia

[25] ACSL Reference Manual 11.1, MGA Software, Concord Massachusetts, 1995

[26] ADAMS, Mechanical Dynamics, Inc., Ann Arbor, Michigan, `http://www.adams.com/`

[27] P.I. Barton, C.C. Pantelides, 1994, "Modeling of Combined Discrete/Continuous Processes", AIChE Journal, v.40, no.6, pp.966-979

[28] B.S. Bennett, 1995, *Simulation Fundamentals*, Prentice Hall International Series in systems and Control Engineering Practice, Prentice Hall International, London

[29] L.G. Birta, T.I. Oren, D.L. Kettenis, 1985, "A Robust Procedure for Discontinuity Handling in Continuous System Simulation," Trans. Soc. Computer Simulation, v.2, no.3, Sept., pp.189-205

[30] M.B. Carver, 1978, "Efficient Integration Over Discontinuities in Ordinary Differential Equations", Mathematics and Computers in Simulation, v.20, n.3, Sept., pp.190-196

[31] M.B. Carver, S.R. MacEwen, 1978, "Numerical Analysis of a System Described by Implicitly-Defined Ordinary Differential Equations Containing Numerous Discontinuities", Applied Mathematical Modelling, v.2, n.4, pp.280-286

[32] J.R. Cash, A.H. Karp, 1990, "A Variable Order Runge-Kutta Method for Initial Value Problems with Rapidly Varying Right-Hand Sides," ACM Transactions on Mathematical Software, v.16, no.3, pp.201-222

[33] Chapra and Canale, 1985, *Numerical Methods for Engineers, 2nd Ed.*, McGraw-Hill, New York

[34] M.D. Compere, R.G. Longoria, 2000, "Combined DAE and Sliding Mode Control Methods for Simulation of Constrained Mechanical Systems," Special Issue on Variable Structure Systems, ASME Journal of Dynamic Systems, Measurement, and Control, December 2000

[35] DADS, LMS International, Leuven, Belgium, http://www.lmsintl.com/

[36] DAEPACK, P. I. Barton, MIT, Cambridge, Massachusetts, http://yoric.mit.edu/daepack/daepack.html

[37] G.S. Duleba, C.W. Ginsburg, and J.E. Harrison, 2001, "Hydraulic system modeling, steady-state analysis, simulation and control system analysis using a lumped mass approach," The Boeing Company, Seattle, Washington, http://www.boeing.com/assocproducts/easy5/technical/files/hydraulic_paper.pdf

[38] EASY5, The Boeing Company, Seattle, Wachington, http://www.boeing.com

[39] D. Ellison, 1981, "Efficient Automatic Integration of Ordinary Differential Equations with Discontinuities", Mathematics and Computers in Simulation, v.XXIII, pp.12-20

[40] W.H. Enright, K.R. Jackson, S.P. N⊘rsett, P.G. Thomsen, 1986, "Interpolants for Runge-Kutta Formulas," ACM Transactions on Mathematical Software, v.12, no.3, p.193-218

[41] W.H. Enright, K.R. Jackson, S.P. N⊘rsett, P.G. Thomsen, 1988, "Effective Solution of Discontinuous IVP's Using a Runge-Kutta Formula Pair with Interpolants," Applied Mathematics and Computation, v.27, p.313-335

[42] E. Fehlberg, 1968, “Classical Fifth-, Sixth-, Seventh-, and Eighth-Order Runge-Kutta Formulas with Stepsize Control,” NASA Technical Report, NASA TR R-287

[43] C.W.Gear, 1984, “Efficient Step Size Control for Output and Discontinuities,” Trans. Soc. Computer Simulation, v.1, no.1, pp.27-31

[44] C.W. Gear, O. Osterby, 1984, “Solving Ordinary Differential Equations with Discontinuities”, ACM Trans. on Mathematical Software, v.10, no.1, pp.23-44

[45] H.J. Halin, 1979, “Integration across discontinuities in ordinary differential equations using power series,” Simulation, v.32, no.2, pp.33-35

[46] J.L. Hay, A.W.J. Griffin, 1979, “Simulation of Discontinuous Dynamical Systems”, Simulation of Systems ’79, pp.79-87

[47] D.J. Higham, 1991, “Highly Continuous Runge-Kutta Interpolants,” ACM Transactions on Mathematical Software, v.17, no. 3, pp.368-386

[48] M.K. Horn, 1983, “Fourth- and Fifth-Order, Scaled Runge-Kutta Algorithms for Treating Dense Output,” SIAM Journal on Numerical Analysis, v.20, no.3, pp.558-568

[49] C. Innocenti, 1994, “Managing Discontinuity in Dynamics”, ASME, Petroleum Division, Structural Dynamics and Vibration, Proc. of the Energy-Sources Technology Conf., pp.37-42

[50] The MathWorks, Inc., Natick, Massachusetts, http://www.mathworks.com/

[51] A. Neumaier, Interval methods for systems of equations, Encyclopedia of Mathematics and Its Applications, Cambridge University Press, Cambridge, 1990

[52] Software repository at http://www.netlib.org, specifically the ode and odepack libraries for solving stiff ODE/DAE at http://www.netlib.org/ode, http://www.netlib.org/odepack

[53] Software repository at http://www.netlib.org, specifically the manpak library for solving DAE at http://www.netlib.org/contin/manpak/

[54] T.R.F. Nonweiler, Computational Mathematics, An Introduction to Numerical Approximation, Ellis Horwood Limited, Chichester, 1984

[55] S.N. Papakostas, C.H. Tsitouras, 1997, "Highly Continuous Interpolants for One-Step ODE Solvers and Their Application to Runge-Kutta Methods," SIAM J. Numer. Anal. V.34, no.1, pp.22-47

[56] T. Park, P.I. Barton, 1996, "State Event Location in Differential-Algebraic Models", ACM Transactions on Modeling and Simulation, v.6, no.2, April, pp.137-165

[57] W. Press, B.P. Flannery, S.A. Teukolsky, W.T. Vetterling, Numerical Recipes in C, The Art of Scientific Computing, Cambridge University Press, Cambridge, 1986

[58] A.J. Preston, M. Berzins, 1991, "Algorithms for the Location of Discontinuities in Dynamic Simulation Problems", Computers in Chemical Engineering, v.15, no.10, pp.701-713

[59] L. Shampine, 1985, "Interpolation for Runge-Kutta Methods," SIAM Journal on Numerical Analysis, v.22, no.5, pp. 1014-1027

[60] L.F. Shampine, I. Gladwell, R.W. Brankin, 1991, "Reliable Solution of Special Event Location Problems for ODEs", ACM Trans. on Mathematical Software, v.17, no.1, pp.11-25

[61] P.W. Sharp, J.H. Verner, 1998, "Generation of High-Order Interpolants for Explicit Runge-Kutta Pairs," ACM Trans. on Mathematical Software, v.24, no.1, pp.13-29

[62] B.R. Ummel, 1986, "Simplified Modeling of Discontinuous Phenomena using EASY5 Switch States", Proc. Summer Computer Simulation Conf., ACM, Reno, Nevada

[63] MSC.Software Corporation, Santa Ana, CA, `http://www.mscsoftware.com/`

[64] J.H. Verner, 1993, "Differentiable Interpolants For High-Order Runge-Kutta Methods," SIAM Journal on Numerical Analysis, v.30, no.5, pp.1446-1466

Vita

Marc Damon Compere was born on October 9, 1970 in New Braunfels, Texas. He is the son of Mark and Jackie Compere and entered Texas A&M University in September 1989. He earned a B.S. Degree in Mechanical Engineering from Texas A&M University in May 1994 and received a Master of Science degree from Texas A&M University in summer 1996. His academic pursuits are focused on modeling, simulation, and control of multi-domain dynamic systems. He can be reached via email at `CompereM@asme.org`.

Permanent Address: 3607 Greystone Dr. #610
Austin, TX 78731

This dissertation was typeset with LaTeX 2$_\varepsilon$[4] by Marc Damon Compere.

[4] LaTeX 2$_\varepsilon$ is an extension of LaTeX. LaTeX is a collection of macros for TeX. TeX is a trademark of the American Mathematical Society.