ON THE IMPLICIT INTEGRATION OF DIFFERENTIAL-ALGEBRAIC EQUATIONS OF MULTIBODY DYNAMICS

by

Dan Negrut

A thesis submitted in partial fulfillment
of the requirements for the Doctor of
Philosophy degree in Mechanical Engineering
in the Graduate College of
The University of Iowa

July 1998

Thesis supervisor: Professor Edward J. Haug

Graduate College
The University of Iowa
Iowa City, Iowa

CERTIFICATE OF APPROVAL

PH.D. THESIS

This is to certify that the Ph.D. thesis of

Dan Negrut

has been approved by the Examining Committee
for the thesis requirement for the Doctor of
Philosophy degree in Mechanical Engineering at the July 1998 graduation.

Thesis committee: ______________________
Thesis supervisor

____
Member

____
Member

____
Member

____
Member

To my parents

ACKNOWLEDGMENTS

I would like to thank my advisor Professor Edward J. Haug for his continuous guidance, I learned many worthwhile things from him. I would also like to thank Professor Florian A. Potra, for his advice and help over the last 5 years and Professor Jeffrey S. Freeman for his patience and understanding during my early years at The University of Iowa.

My thanks and good thoughts go to Cristina, for always supporting and encouraging me, and to Feli, for providing inspiration and always cheering me up.

I am deeply grateful to my teachers and my friends.

ABSTRACT

The topic of the thesis is implicit integration of the differential-algebraic equations (DAE) of Multibody Dynamics. Methods used in the thesis for the solution of DAE are based on state-space reduction via generalized coordinate partitioning. In this approach, a subset of independent generalized coordinates , equal in number to the number of degrees of freedom of the mechanical system, is used to express the time evolution of the mechanical system. The second order state-space ordinary differential equations (SSODE) that describe the time variation of independent coordinates are numerically integrated using implicit formulas. Efficient means for acceleration and integration Jacobian computation are proposed and numerically implemented.

Methods proposed for numerical solution of the index 3 DAE of Multibody Dynamics are the State-Space Reduction Method, the Descriptor Form Method, and the First Order Reduction Method. Algorithms based on the State-Space Reduction and Descriptor Form Methods employ the extensively used family of Newmark multi-step formulas for implicit integration of the SSODE. More refined Runge-Kutta formulas are used in conjunction with both First Order Reduction and Descriptor Form Methods. Rosenbrock-Nystrom and SDIRK formulas of order 4 that are employed are L-stable methods with sound stability and accuracy properties. All integration formulas are provided with robust error control mechanisms based on integration step-size selection.

Several algorithms are developed, based on the proposed methods for numerical solution of index 3 DAE of Multibody Dynamics. These algorithms are shown to be robust and accurate. Typically, two orders of magnitude speed-up is achieved when these algorithms are compared to previously used, well established, explicit numerical integration algorithms for simulation of a stiff model of the High Mobility Multipurpose Wheeled Vehicle (HMMWV) of the US Army.

Computational methods developed in this thesis enable efficient dynamic analysis of systems containing bushings, stiff subsystem compliance elements, and high frequency subsystems that heretofore required tremendous amounts of CPU time, due to limitations of the previously employed numerical algorithms.

TABLE OF CONTENTS

Page

LIST OF TABLES viii

LIST OF FIGURES x

CHAPTER

1. INTRODUCTION 1
    - 1.1. Motivation 1
    - 1.2. Thesis Overview 2

2. REVIEW OF LITERATURE 5
    - 2.1. Review of Methods for the Solution of DAE 5
    - 2.2. Review of Methods for the Solution of ODE 19

3. THEORETICAL CONSIDERATIONS 31
    - 3.1. Generalized Coordinates Partitioning Algorithm 31
    - 3.2. State-Space Reduction Method 35
        - 3.2.1. Multi-step Methods 35
        - 3.2.2. Runge-Kutta Methods 44
    - 3.3. Descriptor Form Method 51
        - 3.3.1. Multi-step Methods 56
        - 3.3.2. Runge-Kutta Methods 56
    - 3.4. First Order Reduction Method 58
        - 3.4.1. Theoretical Considerations in First Order Reduction 59
        - 3.4.2. Computing Accelerations in the Cartesian Representation 68
        - 3.4.3. Computing Accelerations in Minimal Representation 84

4. NUMERICAL IMPLEMENTATIONS 111
    - 4.1. Trapezoidal-Based State-Space Implicit Integration 111
        - 4.1.1. General Considerations 111
        - 4.1.2. Algorithm Pseudo-code 112
    - 4.2. SDIRK Based Descriptor Form Implicit Integration 118
        - 4.2.1. General Considerations 118
        - 4.2.2. SDIRK4/16 123
        - 4.2.3. Algorithm Pseudo-code 127
    - 4.3. Trapezoidal-Based Descriptor Form Implicit Integration 131

4.3.1. Algorithm Pseudo-code 131
4.4. Rosenbrock-Based First Order Implicit Integration............ 133
4.4.1. General Consideration 133
4.4.2. Rosenbrock-Nystrom Methods............................... 135
4.4.3. Order Conditions For Rosenbrock-Nystrom Algorithm 142
4.4.4. Algorithm Pseudo-code 148
4.5. SDIRK-Based First Order Implicit Integration................... 153
4.5.1. SDIRK4/15 153
4.5.2. Algorithm Pseudo-code 159

5. NUMERICAL EXPERIMENTS... 165

5.1. Validation of First Order Reduction Method 165
5.2. Explicit versus Implicit Integration..................................... 174
5.3. Method Comparison... 182

6. CONCLUSIONS AND RECOMMENDATIONS................................... 192

APPENDIX A. PARALLEL COMPUTATION OF INTEGRATION JACOBIAN... 194

APPENDIX B. TANGENT-PLANE PARAMETRIZATION-BASED IMPLICIT INTEGRATION 200

REFERENCES 209

LIST OF TABLES

Table Page

1. Butcher's Tableau 44

2. Numerical Results: Solving For Accelerations and Lagrange Multipliers 82

3. Timing Results for Seven Body Mechanism 83

4. Timing Results for Chain of Pendulums 83

5. HMMWV14 Model - Component Bodies 105

6. Operation Count for Gaussian Elimination 107

7. Operation Count for *Alg-S* 108

8. Operation Count for *Alg-D* 109

9. JR Linear Solvers CPU Results 109

10. Timing Profiles – *Alg-S* and *Alg-D* 110

11. Pseudo-code for Trapezoidal-Based State-Space Method 113

12. Butcher's Tableau for SDIRK Formulas 124

13. SDIRK4/16 Formula for Descriptor Form Method 126

14. Pseudo-code for SDIRK4/16-Based Descriptor Form Method 128

15. Pseudo-code for Trapezoidal-Based Descriptor Form Method 132

16. Pseudo-code for Rosenbrock-Nystrom-Based First Order Reduction Method 149

17. Pseudo-code for SDIRK4/15-Based First Order Reduction Method 160

18. Parameters for the Double Pendulum 165

19. Initial Conditions for Double Pendulum 166

20. Position Error Analysis *ForSDIRK* 170

21. Position Error Analysis *ForRosen* 171

22. Velocity Error Analysis *ForSDIRK* 173

23. Velocity Error Analysis *ForRosen* 174

24. HMMWV14 Explicit Integration Simulation CPU Times 178

25. HMMWV14 Implicit Integration Simulation Results 179

26. Timing Results for *InflSDIRK* 183

27. Timing Results for *InflTrap* 183

28. Timing Results for *ForSDIRK* 183

29. Timing Results for *ForRosen* 184

30. *ForSDIRK* Analytical/Numerical Computation of Integration Jacobian 191

31. Pseudo-code for Parallel Computation of Integration Jacobian 197

LIST OF FIGURES

Figure Page

1. Seven Body Mechanism 69
2. Chain of Pendulums 76
3. Graph Representation: Seven-Body Mechanism 76
4. Reduced Matrix **B**: Seven-Body Mechanism 77
5. Two Joint Numbering Schemes: Chain of Pendulums 78
6. Reduced Matrix: Chain of Pendulums 78
7. A Pair of Connected Bodies in JR 86
8. A Tree Structure 90
9. HMMWV14 Body Model: Topology Graph 105
10. Spanning Tree – HMMWV14 106
11. Double Pendulum 166
12. Orientation Body 1 169
13. Angular Velocity Body 1 172
14. US Army HMMWV 175
15. 14 Body Model of HMMWV 175
16. Topology Graph for HMMWV14 176
17. Chassis Height HMMWV14 177
18. Explicit Integration Results for Tolerance 10^{-3} 180
19. Implicit Integration Results for Tolerance 10^{-3} 181
20. Timing Results for Different Tolerances 181

21. Algorithm Comparison for Different Tolerances 186

22. Algorithm Comparison for Different Simulation Lengths.......................... 187

23. Step-Size History for *ForSDIRK* and *SspTrap* 188

24. Step-Size History for *ForRosen* and *InflSDIRK* 189

25. Number of Iterations for *SspTrap* 191

CHAPTER 1
INTRODUCTION

1.1 Motivation

The motivation for this thesis lies in the importance of having effective mathematical and computational tools for virtual prototyping. In a world of increasing global competition and shrinking windows of opportunity, as design cycles are continuously compressed under the pressure of high quality standards and short time-to-market objectives, virtual prototyping becomes a powerful tool in the hands of designers. Once the privilege of few companies, steady growth in inexpensive computational power has established virtual prototyping as an essential link in the design cycle of virtually any successful company. In applications ranging from Aircraft and Automotive Industries, to Biomechanics and Mechatronics, cutting costs and reducing development cycles by eliminating hardware prototypes, along with quality improvement through design sensitivity and "what-if" studies are attributes that have made virtual prototyping an important segment of CAE/CAD/CAM integrated environments.

With these considerations in mind, at the beginning of 1996, the writer critically evaluated some of the mathematical methods of virtual prototyping in Multibody Dynamics. Research topics were identified that had the potential to qualitatively improve methods in use at that time. A first goal was to answer the challenge posed by dynamic analysis of richer and more complex dynamic systems, and then there was the objective of developing algorithms and methods that would enable effective numerical implementations of theoretical methods once deemed to be computationally intractable.

Computational stability and efficiency were the key issues, to be addressed using new algorithms for topology-based linear algebra and numerical integration of Differential-Algebraic Equations of Multibody Dynamics.

1.2 Thesis Overview

This Section provides an outline of the thesis. Brief remarks describe the content of each Chapter of the thesis.

Chapter 2 contains a review of the literature. Many different techniques for the solution of differential-algebraic equations (DAE) of Multibody Dynamics have been proposed over time. Some of the more important approaches are presented, pointing out their merits and deficiencies.

Chapter 3 contains the methods proposed for implicit numerical integration of DAE of multibody dynamics. The first method presented is the State Space Reduction Method, in which the DAE of Multibody Dynamics are reduced, via generalized coordinate partitioning (Wehage and Haug, 1982), to a set of ordinary differential equations (ODE).

The next method is the Descriptor Form Method, in which using an ODE numerical integration formula, the index one DAE problem obtained by associating the equations of motion and acceleration kinematic constraints equation, is directly integrated. Constraint error accumulation is prevented in this method by recovering dependent positions and velocities via position and velocity kinematic constraint equations.

The last method presented for the solution of stiff DAE of Multibody Dynamics is the First Order Reduction Method, which has the potential of using any standard implicit numerical code to numerically solve the resulting ODE problem. In this formulation, second order differential equations governing the time evolution of independent positions is reduced to a system of first order differential equations. Central to the First Order Reduction Method is the issue of generalized acceleration computation. Sections 3.4.2 and 3.4.3 are devoted to a comprehensive analysis of generalized acceleration computation in Cartesian and minimal coordinates representations, respectively.

Based on theoretical considerations of Chapter 3, several algorithms and codes were developed for the implicit integration of the DAE of Multibody Dynamics. These codes represent numerical implementation of the State Space, Descriptor Form, and First Order Reduction methods introduced in Sections 3.2, 3.3, and 3.4, respectively. The codes are presented in Chapter 4. The methods implemented are as follows:

(a). The State Space Method is implemented based on the Newmark family of implicit integration formulas. In particular, the Trapezoidal formula is used throughout the numerical experiments, due to its higher order compared to other Newmark formulas.

(b). The Descriptor Form Method is implemented based on two integration formulas
- (b1). Trapezoidal formula.
- (b2). A five stage, order 4, A-stable, stiffly-accurate singly diagonal implicit Runge-Kutta (SDIRK) method.

(c). The first order approach was implemented using two different integration formulas
- (c1). A four stage, order 4 Rosenbrock formula.
- (c2). An SDIRK five stage, order 4, A-stable, stiffly-accurate formula of Hairer and Wanner (1996).

When applicable, each numerical implementation is detailed in the following three aspects:

(1). Specifics of DAE-to-ODE reduction.

(2). ODE integration stage.

(3). Iteration stopping criteria, and step size control.

Based on the algorithms developed, numerical experiments are carried out using two test problems. Chapter 5 contains the results of these simulations. First, the numerical methods are validated in terms of asymptotic behavior. Then, the implicit integrators defined are compared in terms of efficiency with an explicit alternative for integration of stiff ODE of Multibody Dynamics. A third set of numerical experiments is aimed at comparing the implicit integrators among themselves.

Chapter 6 presents potential directions of future research, and concludes the thesis with final remarks concerning the topic of generalized coordinate-based state-space implicit integration of DAE of Multibody Dynamics. Appendix A presents more recent theoretical results regarding the potential of using multiprocessor architectures for speeding up the otherwise computationally intensive task of DAE implicit integration. Fast integration Jacobian evaluation is the focus of the analysis in this Section, which concludes with a strategy that can take advantage of parallel computer architectures. Finally, the theoretical framework derived for the coordinate partitioning approach to solving DAE of Multibody Dynamics is extended in Appendix B to the tangent-plane parametrization method. In this context, coordinate partitioning is in fact a particular case of the tangent-plane parametrization method, obtained by choosing a certain projection matrix.

CHAPTER 2
REVIEW OF LITERATURE

2.1 Review of Methods for the Solution of DAE

In the present work, only the case of rigid body mechanical system models is considered. However, the methods proposed in this work are suitable for dynamic analysis of models incorporating flexible components. The issue of generating derivative information specific to systems containing flexible components has not yet been addressed.

Throughout this document, $\mathbf{q} = [q_1, q_2, \ldots, q_n]^{\mathrm{T}}$ denotes the vector of generalized coordinates. The n generalized coordinates q_i define the state of the mechanical system at the position level; i.e., given a set of n values for q_i, the position of each element of the mechanical system model is uniquely determined. The generalized coordinates may be absolute (Cartesian) coordinates of body reference frames, relative coordinates between bodies, or a combination of both. Generalized velocities are defined as the first time derivative of the generalized coordinates. In what follows an over-dot signifies time derivative. Thus, generalized velocity is defined as

$$\dot{\mathbf{q}} = [\dot{q}_1, \dot{q}_2, \ldots, \dot{q}_n]^{\mathrm{T}} \tag{2.1}$$

The set of generalized positions and velocities define the state of the mechanical system model; i.e., once these quantities are available there is a unique configuration of the system at a given instant in time. Conversely, to each state of the mechanical system there corresponds unique generalized position $\mathbf{q}$ and velocity $\dot{\mathbf{q}}$.

Joints connecting bodies of a mechanical system model restrict the relative and/or absolute motion of components of the model. From a mathematical standpoint, these mobility constraints are accounted for by the requirement that a set of algebraic expressions must be satisfied throughout the simulation. In the most general case, the constraints may be equalities or inequalities involving generalized coordinates and their first time derivatives. In this thesis only the case of scleronomic and holonomic equality constraints will be considered; i. e., the constraint equations do not depend explicitly on time, and they do not contain any time derivatives of generalized coordinates. Inequality constraint equations are not treated in the thesis. Technically, this scenario can be addressed by considering event driven integration of DAE, when event location (discontinuity treatment) becomes the main issue of concern. More information about event driven integration can be found in the work of Winckler (1997).

Under the above assumptions, the position kinematic constraint equations assume the form

$$\boldsymbol{\Phi}(\mathbf{q}) \equiv [\Phi_1(\mathbf{q}), \Phi_2(\mathbf{q}), \ldots, \Phi_m(\mathbf{q})]^{\mathrm{T}} = \mathbf{0} \tag{2.2}$$

Differentiating the position kinematic constraint equation of Eq. (2.2) with respect to time yields the velocity kinematic constraint equation,

$$\boldsymbol{\Phi}_{\mathbf{q}}(\mathbf{q})\dot{\mathbf{q}} = \mathbf{0} \tag{2.3}$$

where subscripts denote partial differentiation; i.e., $\boldsymbol{\Phi}_{\mathbf{q}} = [(\partial \Phi_i)/(\partial q_j)]$. Finally, taking another time derivative of Eq. (2.3) yields the acceleration kinematic equation,

$$\boldsymbol{\Phi}_{\mathbf{q}}(\mathbf{q})\ddot{\mathbf{q}} = -(\boldsymbol{\Phi}_{\mathbf{q}}\dot{\mathbf{q}})_{\mathbf{q}}\dot{\mathbf{q}} \equiv \tau(\mathbf{q}, \dot{\mathbf{q}}) \tag{2.4}$$

Equations (2.1) through (2.4) characterize the admissible motion of the constrained mechanical system at position, velocity, and acceleration levels.

The state of the mechanical system will change in time under the effect of both applied and constraint forces. The Lagrange multiplier form of the equations of motion for the mechanical system model is (Haug , 1989)

$$\mathbf{M}(\mathbf{q})\ddot{\mathbf{q}} + \Phi_{\mathbf{q}}^{\mathrm{T}}(\mathbf{q})\lambda = \mathbf{Q}^{\mathrm{A}}(\mathbf{q}, \dot{\mathbf{q}}, t) \qquad (2.5)$$

where $\mathbf{M} \in \Re^{n \times n}$ is the mass matrix, which depends on the generalized coordinates $\mathbf{q}$; $\lambda \in \Re^{m}$ is the vector of Lagrange multipliers that account for the workless constraint forces; and $\mathbf{Q}^{\mathrm{A}} \in \Re^{n}$ is the vector of generalized applied forces that may depend on generalized coordinates, their time derivatives, and time.

Equations (2.2) through (2.5) are the so-called Newton-Euler constrained equations of motion. From a mathematical standpoint, they comprise a system of differential-algebraic equations (DAE). Mathematically, DAE are not ODE (Petzold , 1982). The task of obtaining a numerical solution of the DAE problem of Eqs. (2.2) through (2.5) is substantially more difficult and prone to intense numerical computation that the task of solving an ODE (Potra, 1994). This trend is more pronounced for higher index DAE, where the index (Brenan, Campbell, Petzold, 1989), is defined as the number of derivatives required to transform the DAE problem into an ODE problem.

To find the index of the DAE of Multibody Dynamics, note that acceleration kinematic constraint equation of Eq. (2.4) is obtained after taking two time derivatives of the position kinematic constraint equation of Eq. (2.2). The acceleration kinematic equations are associated with the equations of motion to obtain a linear system in generalized accelerations and Lagrange multipliers

$$\begin{bmatrix} \mathbf{M} & \Phi_{\mathbf{q}}^{\mathrm{T}} \\ \Phi_{\mathbf{q}} & 0 \end{bmatrix} \begin{bmatrix} \ddot{\mathbf{q}} \\ \lambda \end{bmatrix} = \begin{bmatrix} \mathbf{Q}^{\mathrm{A}} \\ \tau \end{bmatrix} \qquad (2.6)$$

The expression obtained for Lagrange multipliers, by formally solving Eq. (2.6), is a function of generalized positions and velocities, $\mathbf{q}$ and $\dot{\mathbf{q}}$, respectively. Taking a third derivative of Lagrange multipliers, and combining these derivatives with Eq. (2.5), results in a set of ODE. Consequently, the index of the DAE of Multibody Dynamics is three.

In practice this sequence of steps converting the DAE to ODE is never used to obtain the numerical solution of the original problem. Its only purpose is to determine the index of the DAE being analyzed. This information is useful, as generally the index is regarded as a measure of the difficulty that should be expected when numerically solving the DAE. In particular, the index 3 of the DAE of Multibody Dynamics is a rather high index when compared with DAE obtained from modeling problems arising in other areas of Physics and Engineering.

Dynamic analysis of mechanical systems concerns their time evolution under the action of applied forces. It is highly unlikely to obtain an analytical solution to the DAE of Multibody Dynamics, so approximate solutions are obtained by means of numerical methods. In this context, Eqs. (2.2) and (2.5) alone cease to characterize the time evolution of the system, since Eqs. (2.3) and (2.4) are not guaranteed to be satisfied. When this is the case, numerical solution at velocity and acceleration levels drifts away from analytical solution. Consequently, future position configurations of the system will be wrong, since the numerical integration stage embedded in the numerical algorithm uses corrupted derivative information.

Special numerical methods have been developed to deal with DAE, and the theory surrounding these methods builds around the index of the DAE. Thus, different numerical methods are proposed for different index DAE problems. In this Section, the focus is on methods for the solution of the index 3 DAE of Multibody Dynamics. Most of the methods for the solution of this class of DAE belong to one of the following categories:

(a). Stabilization Methods

(b). Projection Methods

(c). State Space Methods

An early stabilization-based numerical algorithm that allows for integration of DAE is the so called constraint stabilization technique (Baumgarte, 1972). The problem is reduced to index 1, as in Eq. (2.6), and is directly integrated. Since, after direct integration the constraints fail to be satisfied, at the next time step, the right-side of acceleration kinematic constraint equation is modified to take into account the constrain violation.

The form of the right side acceleration kinematic constraint equation is altered to

$$\bar{\tau} = \tau - 2\alpha\dot{\Phi} - \beta\Phi \tag{2.7}$$

where last two terms are the so called compensation terms. These two terms do not appear in the original form of acceleration kinematic constraint equation of Eq. (2.4), and they compensate for errors in satisfying constraint equations at position and velocity levels.

Ostermeyer (1990) discusses criteria for optimally choosing the positive scalars α and β. This process is problematic (Ascher et. al, 1995) and is yet to be resolved. Usually, $\alpha = \gamma$ and $\beta = \gamma^2$ where $\gamma > 0$. The manifold then becomes an attractor of the solution of the newly obtained system of ordinary differential equations of Eqs. (2.5) and (2.7). Ideally, the value of the scalar γ would be independent of both the method used to discretized the new set of ODE and the integration step-size. A simple example shows, however, that choosing an optimal γ depends on both these factors.

Consider for example the hypothetical case of a system with linear constraints at the position level,

$$\Phi(\mathbf{q}) = \mathbf{Gq} \tag{2.8}$$

Using the forward Euler integration formula, Baumgarte's technique results in

$$\begin{aligned}\dot{\mathbf{q}}_{n+1} &= \dot{\mathbf{q}}_n + h\mathbf{M}^{-1}\mathbf{Q}(\mathbf{q}_n, \dot{\mathbf{q}}_n) - h\mathbf{M}^{-1}\mathbf{G}^{\mathrm{T}}(\mathbf{G}\mathbf{M}^{-1}\mathbf{G}^{\mathrm{T}})^{-1} \\ &\cdot[\mathbf{G}\mathbf{M}^{-1}\mathbf{Q}(\mathbf{q}_n, \dot{\mathbf{q}}_n) + \alpha\mathbf{G}\mathbf{q}_n + \beta\mathbf{G}\dot{\mathbf{q}}_n] \\ \mathbf{q}_{n+1} &= \mathbf{q}_n + h\dot{\mathbf{q}}_n\end{aligned} \tag{2.9}$$

In order to see how constraint equations are satisfied at the new time step $n+1$, multiply the new positions and velocities by $\mathbf{G}$ to obtain

$$\begin{bmatrix}\hat{\mathbf{q}}_{n+1} \\ \hat{\dot{\mathbf{q}}}_{n+1}\end{bmatrix} = \begin{bmatrix}\mathbf{I} & h\mathbf{I} \\ -\alpha h\mathbf{I} & (1-\beta h)\mathbf{I}\end{bmatrix}\begin{bmatrix}\hat{\mathbf{q}}_n \\ \hat{\dot{\mathbf{q}}}_n\end{bmatrix} \tag{2.10}$$

where $\hat{\mathbf{q}} \equiv \mathbf{G}\mathbf{q}$ and $\hat{\dot{\mathbf{q}}} \equiv \mathbf{G}\dot{\mathbf{q}}$ are errors in constraints at the position and velocity levels, and $\mathbf{I}$ is the $m \times m$ identity matrix. Denoting by $\mathbf{B}$ the coefficient matrix in Eq. (2.10), ideally $\mathbf{B} \equiv \mathbf{0}$. Since this is not possible for the forward Euler method, α and β are optimally chosen; i. e., such that $\mathbf{B}^2 \equiv \mathbf{0}$. These optimal values are $\alpha = 1/h$ and $\beta = 1/h^2$. In other words, the scalar γ above depends on both the method used to discretize the ordinary differential equations and the integration step size, which is a drawback of the approach.

More sophisticated techniques should be considered. Asher et al. (1995) propose several algorithms that have as starting point the underlying ODE associated with the index 3 DAE of Multibody Dynamics. The underlying ODE is obtained by formally eliminating Lagrange multipliers from equations of motion of Eq. (2.5) using acceleration kinematic constraint equation to obtain

$$\lambda = (\Phi_{\mathbf{q}}\mathbf{M}^{-1}\Phi_{\mathbf{q}}^{\mathrm{T}})^{-1}[\Phi_{\mathbf{q}}\mathbf{M}^{-1}\mathbf{Q}^{\mathrm{A}} - \tau] \tag{2.11}$$

Lagrange multipliers are substituted back into Eq. (2.5) to obtain the underlying ODE associated to the DAE as

$$\mathbf{M}\ddot{\mathbf{q}} = \mathbf{Q}^{\mathrm{A}}(\mathbf{q}, \dot{\mathbf{q}}) - \Phi_{\mathbf{q}}^{\mathrm{T}}(\Phi_{\mathbf{q}}\mathbf{M}^{-1}\Phi_{\mathbf{q}}^{\mathrm{T}})^{-1}[\Phi_{\mathbf{q}}\mathbf{M}^{-1}\mathbf{Q}^{\mathrm{A}}(\mathbf{q}, \dot{\mathbf{q}}) - \tau(\mathbf{q}, \dot{\mathbf{q}})] \tag{2.12}$$

which theoretically, can be further transformed to a first order system of ODE

$$\dot{\mathbf{z}} = \hat{\mathbf{f}}(\mathbf{z}) \tag{2.13}$$

with $\mathbf{z} \equiv (\mathbf{q}^{\mathrm{T}}, \dot{\mathbf{q}}^{\mathrm{T}})^{\mathrm{T}}$. By directly integrating the set of second order ODE in Eq. (2.12), kinematic constraint equations at position and velocity levels will cease to be satisfied.

In the spirit of Baumgarte's method, the idea of Ascher et al. (1995) is to add a more general constraint stabilization term in right-side of Eq. (2.13), to obtain

$$\dot{\mathbf{z}} = \hat{\mathbf{f}}(\mathbf{z}) - \gamma \mathbf{F}(\mathbf{z})\mathbf{h}(\mathbf{z}) \tag{2.14}$$

where $\gamma > 0$ is a parameter, $\mathbf{F}(\mathbf{z})$ is a $2n \times 2m$ matrix, and

$$\mathbf{h}(\mathbf{z}) = \begin{bmatrix} \Phi(\mathbf{q}) \\ \Phi_{\mathbf{q}}\dot{\mathbf{q}} \end{bmatrix} \tag{2.15}$$

are the kinematic constraint equations at position and velocity levels. In a numerical implementation, the system of ODE of Eq. (2.14) is directly integrated by any explicit integration scheme to advance the simulation.

While in Baumgarte's method the parameters α and β were to be chosen, the approach proposed by Ascher needs to provide the matrix $\mathbf{F}(\mathbf{z})$ of Eq. (2.14). The authors suggested several forms for this matrix. Based on simulation results of two small-scale planar mechanical models, they concluded the approach is reliable for nonstiff and highly oscillatory problems.

The addition of a stabilization term is not justified from a physical, but only a mathematical standpoint; i. e., it is introduced to limit the drifting effect induced by direct integration of all generalized coordinates in the formulation. Depending on the value of the parameter γ and the expression of the compensation coefficient matrix $\mathbf{F}(\mathbf{z})$, the dynamics of the system would be altered if the formulation of Eq. (2.14) is directly integrated. Consequently, Ascher et al. slightly modify the proposed approach as follows: First, use an integrator of choice to directly integrate the underlying ODE reformulated as in Eq. (2.13). Then use Eq. (2.14) to stabilize the constraint equations. In this respect this approach is very similar to coordinate projection techniques presented later in this Section. Finally, the constraint stabilization algorithm proposed is as follows:

(1). Apply direct integration to the underlying ODE

$$\tilde{\mathbf{z}}_{n+1} = \mathbf{\Psi}_h^f(\mathbf{z}_n) \tag{2.16}$$

(2). Apply stabilization

$$\mathbf{z}_{n+1} = \tilde{\mathbf{z}}_{n+1} - \gamma h \mathbf{F}(\tilde{\mathbf{z}}_{n+1})\mathbf{h}(\tilde{\mathbf{z}}_{n+1}) \tag{2.17}$$

Ascher, Petzold, and Chin (1994), claim that, provided the integration formula used to obtain $\tilde{\mathbf{z}}_{n+1}$ in Eq. (2.16) is of order p, the method has global error $O(h^p)$ and constraints at the position and velocity levels are satisfied to $O(h^{p+1})$. More details about the choice of the compensation coefficient matrix $\mathbf{F}(\mathbf{z})$ and stability range for the parameter γ can be found in Ascher et al. (1995), Ascher, Petzold, and Chin (1994), and Ascher and Petzold (1993).

A second class of algorithms for numerical solution of the DAE of Multibody Dynamics is based on so called projection techniques (Eich et. al, 1990), in which all generalized coordinates are integrated at each time step. Additional multipliers are introduced to account for the requirement that the solution satisfy constraint equations at position, velocity, and for some formulations, acceleration level. Gear et al. (1985) reduce the DAE to an analytically equivalent index 2 problem, in which projection is only performed at the velocity level. An extra multiplier μ is introduced to insure that velocity kinematic constraint equation of Eq. (2.3) is also satisfied. The algorithm uses a backward differentiation formula (BDF) to discretize the following form of the equations of motion:

$$\begin{aligned}
&\dot{\mathbf{q}} = \mathbf{v} - \Phi_{\mathbf{q}}^{\mathrm{T}}(\mathbf{q})\mu \\
&\mathbf{M}(\mathbf{q})\dot{\mathbf{v}} = \mathbf{Q}^{\mathrm{A}} - \Phi_{\mathbf{q}}^{\mathrm{T}}(\mathbf{q})\lambda \\
&\Phi(\mathbf{q}) = \mathbf{0} \\
&\Phi_{\mathbf{q}}(\mathbf{q})\dot{\mathbf{q}} = \mathbf{0}
\end{aligned} \tag{2.18}$$

In a similar approach proposed, by Fuhrer and Leimkuhler (1991), the DAE is reduced to index 1 and an additional multiplier η is introduced, along with the requirement that the acceleration kinematic equation of Eq. (2.4) is satisfied.

In an analytical framework, all additional multipliers can be proved to be zero for the actual solution. However, under discretization, these multipliers assume nonzero values, due to truncation errors of the integration formula being used.

Starting from the previous index 1 formulation, Fuhrer and Leimkuhler (1991) propose an algorithm based on an index 1 formulation with no extra multipliers. All variables are integrated, and kinematic constraint equations at position, velocity, and acceleration levels are imposed. Under discretization, an over-determined set of $2 \cdot n + 3 \cdot m$ nonlinear equations in $2 \cdot n + m$ unknowns must be solved at each integration step. The discretization involves a backward differentiation formula (BDF). Because of truncation errors, the equations become inconsistent and can only be solved in a generalized sense. While the so-called ssf-solution obtained using a special oblique projection technique is, in the case of linear constraint equations, equivalent to that obtained by integrating the state-space form using the same discretization formula, this ceases to be the case in general (Potra, 1993). The method is robust, and it is comparable in terms of efficiency to the index 1 formulation with additional multipliers.

The methods proposed by Fuhrer and Leimkuhler (1991) and Gear et. al, (1985) belong to the class of so called derivative projection methods; i.e., expressions for derivatives are modified by additional multipliers that ensure constraint satisfaction. A second projection technique is based on the coordinate projection approach. The derivatives are no longer modified, and integration is carried out to obtain a solution of the index 1, or for some formulations index 2 DAE. Since all variables are integrated, they do not satisfy the constraint equations, so some form of coordinate projection technique is employed to bring the ODE solution to the constraint manifold. This is the approach followed by Lubich (1990), Shampine (1986), Eich (1993), and Brasey (1994).

From a physical standpoint, the projection stage is conventional, and typically the underlying ODE is integrated with very high accuracy to reduce the weight of the projection stage in the overall algorithm. The code MEXX (Lubich et. al, 1992) for integration of multibody systems is based on coordinate projection and uses relatively expensive but very accurate extrapolation methods for integration of the ODE.

When coordinate projection methods are applied, the index of the DAE of Multibody Dynamics is usually reduced to a lower order, usually two (Lubich (1990), Brasey (1994)). The most representative methods in this class are the half-explicit methods of Brasey (1994). The idea of half-explicit methods is introduced by starting with the simplest method; i. e., forward Euler. Starting from consistent initial values $(\mathbf{q}_0, \mathbf{v}_0)$, one step of the explicit Euler method is applied to the equations of motion of Eq. (2.5), yielding

$$\begin{aligned} &\mathbf{q}_1 - \mathbf{q}_0 = h\,\mathbf{v}_0 \\ &\mathbf{M}(\mathbf{q}_0)(\hat{\mathbf{v}}_1 - \mathbf{v}_0) = h\mathbf{Q}^{\mathrm{A}}(\mathbf{q}_0, \mathbf{v}_0) - h\Phi_{\mathbf{q}}^{\mathrm{T}}(\mathbf{q}_0)\lambda(\mathbf{q}_0, \mathbf{v}_0) \end{aligned} \tag{2.19}$$

After this step, the velocity is stabilized. There are several ways in this can be done. One possibility is to keep the value $\mathbf{q}_1$ fixed, and to project $\hat{\mathbf{v}}_1$ onto the manifold defined by the velocity kinematic constraint equation of Eq. (2.3), as suggested by Alishenas (1992) and Lubich (1991). The solution $\mathbf{v}_1$ is obtained by solving

$$\begin{aligned} &\mathbf{M}(\mathbf{q}_1)(\mathbf{v}_1 - \hat{\mathbf{v}}_1) = -\Phi_{\mathbf{q}}^{\mathrm{T}}(\mathbf{q}_1)\mu \\ &\Phi_{\mathbf{q}}\mathbf{v}_1 = \mathbf{0} \end{aligned} \tag{2.20}$$

The idea of half-explicit methods is to replace the argument $\mathbf{q}_1$ with $\mathbf{q}_0$ in the first line of Eq. (2.20). Adding the resulting equations to the second of Eqs. (2.19) eliminates $\hat{\mathbf{v}}_1$. Introducing a new variable λ_0 for $\lambda(\mathbf{q}_0, \mathbf{v}_0) + \mu/h$, the following algorithm is obtained:

$$\begin{aligned}
&\mathbf{q}_1 - \mathbf{q}_0 = h\,\mathbf{v}_0 \\
&\mathbf{M}(\mathbf{q}_0)(\mathbf{v}_1 - \mathbf{v}_0) = h\mathbf{Q}^{\mathrm{A}}(\mathbf{q}_0, \mathbf{v}_0) - h\Phi_{\mathbf{q}}^{\mathrm{T}}(\mathbf{q}_0)\lambda_0 \\
&\Phi_{\mathbf{q}}(\mathbf{q}_1)\mathbf{v}_1 = \mathbf{0}
\end{aligned} \tag{2.21}$$

The first relation defines $\mathbf{q}_1$, whereas the remaining equations represent a linear system for $\mathbf{v}_1$ and λ_0.

The advantage of the approach of Eq. (2.21) over the one of Eqs. (2.19) and (2.20) is that neither the value $\lambda(\mathbf{q}_0, \mathbf{v}_0)$ (which requires the solution of a linear system), nor the intermediate value $\hat{\mathbf{v}}_1$ must be calculated. Consequently, this algorithm does not require the acceleration kinematic constraint equation of Eq. (2.5). Application of the above idea to each stage of an explicit Runge-Kutta method with coefficients a_{ij} and b_i yields the algorithm

$$\begin{aligned}
\mathbf{P}_i &= \mathbf{q}_0 + h\sum_{j=1}^{i-1} a_{ij}\mathbf{V}_j \\
\mathbf{V}_i &= \mathbf{v}_0 + h\sum_{j=1}^{i-1} a_{ij}\mathbf{V}'_j \\
\mathbf{0} &= \Phi_{\mathbf{q}}(\mathbf{P}_i)\mathbf{V}_i
\end{aligned} \tag{2.22}$$

where $\mathbf{V}'_j$ is given by

$$\mathbf{M}(\mathbf{P}_j)\mathbf{V}'_j = \mathbf{Q}^{\mathrm{A}}(\mathbf{P}_j, \mathbf{V}_j) - \Phi_{\mathbf{q}}^{\mathrm{T}}(\mathbf{P}_j)\Lambda_j \tag{2.23}$$

and $\mathbf{q}_1 = \mathbf{P}_{s+1}$, $\mathbf{v}_1 = \mathbf{V}_{s+1}$. For simplicity, the notation $a_{s+1,i} = b_i$, $i = 1, \ldots, s$, has been used. Attractive features of the proposed method are that only linear systems of equations must be solved, and the acceleration kinematic equation does not appear in the formulation.

Another class of algorithms for the solution of DAE is based on the state-space reduction method. The DAE is reduced to an equivalent ODE, using a parametrization of the constraint manifold. The dimension of the equivalent second order state-space ODE (SSODE) is significantly reduced to $ndof = n - m$. This method has the potential of using well established theory and reliable numerical techniques for the solution of ODE.

Since constraint equations in multibody dynamics are generally nonlinear, a parametrization of the constraint manifold can only be determined locally. Computational overhead results each time the parametrization is changed. Nonlinearity also leads to computational effort in retrieving dependent generalized coordinates through the parametrization. This stage requires the solution of a system of nonlinear equations, for which Newton-like methods are generally used.

The choice of constraint parametrization differentiates among algorithms in this class. The most used parametrization is based on an independent subset of position coordinates of the mechanical system (Wehage and Haug, 1982). The partition of variables into dependent and independent sets is based on $\mathbf{LU}$ factorization of the constraint Jacobian matrix. This partition is maintained as long as the sub-Jacobian matrix (the derivative of the constraint functions with respect to dependent coordinates) is not ill conditioned. The method has been used extensively with large scale applications in multibody dynamics and has proved to be reliable and accurate. This approach is presented in detail in Section 3.1.

State-space methods for solution of the DAE of multibody dynamics have been subject to critique in two aspects. First, the choice of projection subspace is generally not global. Second, as Alishenas and Olafsson (1994), have pointed out, bad choices of the projection space result in SSODE that are demanding in terms of numerical treatment, mainly at the expense of overall efficiency of the algorithm.

These critiques are answered to some extent by considering the tangent-plane parametrization method proposed by Mani et al. (1985), where the parametrization variables are obtained as linear combinations of generalized coordinates. The parametrization variables z_i are components of the vector $\mathbf{z} = \mathbf{V}^{\mathrm{I}}\mathbf{q}$, where $\mathbf{q}$ is the vector of generalized coordinates, and $\mathbf{V}^{\mathrm{I}} \in \Re^{ndof \times m}$ contains the last $ndof$ rows of the matrix $\mathbf{V}$ in the singular value decomposition (SVD) $\mathbf{\Phi}_{\mathbf{q}} = \mathbf{U}^{\mathrm{T}}\mathbf{D}\mathbf{V}$ of the constraint Jacobian. The SVD decomposition can be replaced by a more efficient QR factorization, and maintain the rest of the approach.

The benefits of this reduction are anticipated to be twofold. The resulting SSODE is expected to be numerically better conditioned and allow for significantly larger integration step-sizes. Second, dependent variable recovery can take advantage of information generated during state-space reduction.

The state-space based reduction alternatives presented above are particular cases of a more general formulation proposed by Potra and Rheinboldt (1991). The main idea is that, under discretization, Eqs. (2.2) through (2.5) result in an over-determined nonlinear algebraic system that is inconsistent. Therefore, projection is first performed at the position and velocity levels,

$$\begin{aligned} \mathbf{A}_1^{\mathrm{T}}(\dot{\mathbf{q}} - \mathbf{v}) &= \mathbf{0} \\ \mathbf{A}_2^{\mathrm{T}}(\dot{\mathbf{v}} - \mathbf{a}) &= \mathbf{0} \end{aligned} \tag{2.24}$$

where $\mathbf{A}_1$ and $\mathbf{A}_2$ are $n \times ndof$ matrices, chosen such that the augmented matrices

$$\begin{bmatrix} \mathbf{A}_i^{\mathrm{T}} \\ \mathbf{\Phi}_{\mathbf{q}} \end{bmatrix} , \qquad i = 1,2$$

are nonsingular. Equation (2.24) is appended to Eqs. (2.2) through (2.5), and rewritten in the form

$$\begin{aligned}
&\Phi(\mathbf{q}) = \mathbf{0} \\
&\Phi_{\mathbf{q}}(\mathbf{q})\mathbf{v} = \mathbf{0} \\
&\Phi_{\mathbf{q}}(\mathbf{q})\mathbf{a} = \tau(\mathbf{q},\mathbf{v}) \\
&\mathbf{M}(\mathbf{q})\mathbf{a} + \Phi_{\mathbf{q}}^{\mathrm{T}}\lambda = \mathbf{Q}^{\mathrm{A}}(t,\mathbf{q},\mathbf{v})
\end{aligned}$$

to obtain an index 1 DAE that is discretized by an integration formula. The resulting well determined nonlinear algebraic system is solved at each time step to recover position $\mathbf{q}$, velocity $\mathbf{v}$, acceleration $\mathbf{a}$, and Lagrange multiplier λ. The methods of Wehage and Haug (1982), Mani et al. (1985), Haug and Yen (1992) can be obtained from the formulation presented above by choosing particular projection matrices $\mathbf{A}_1$ and $\mathbf{A}_2$.

The last state-space type method discussed, is the differentiable null space method of Liang and Lance (1987). It reduces the DAE to an equivalent SSODE by projecting the equations of motion onto the tangent hyperplane of the manifold. The projection is done before discretization, and Lagrange multipliers are eliminated from the problem. The algorithm requires a set of $ndof$ vectors that span the constraint manifold hyperplane, along with their first time derivative. The Gram-Schimdt factorization method (Atkinson, 1989) is used to obtain this information. The algorithm is efficient and robust, the resultant SSODE of dimension $ndof$ being well conditioned. The implementation of an implicit formula to integrate the resulting state-space ODE is difficult, because of the Gram-Schmidt process embedded in the algorithm.

There are several conceptually different methods to obtain numerical solutions of the DAE of Multibody Dynamics. They share common features; e. g., the existence of an integration formula, and face similar challenges; e. g., the solution of systems of nonlinear equations. However, they are significantly different, in terms of the sequence of steps taken for solving the DAE problem.

Because of the heterogeneous character of the algorithms and techniques used by different numerical DAE solvers, robustness and efficiency considerations limit the breadth of the proposed research to state-space methods for solving DAE. This choice is motivated by the better accuracy and reliability of the algorithms in this class, when used for dynamic analysis of large scale mechanical systems.

2.2 Review of Methods for the Solution of ODE

There are three important classes of methods for numerical integration of ODE; multi-step methods, Runge-Kutta methods, and extrapolation methods. They are briefly discussed below, along with error control mechanisms.

When compared to one step methods, multi-step methods usually require fewer function evaluations to meet accuracy requirements. They are potentially efficient and useful if derivative information is expensive to obtain, as is the case in simulation of Multibody Dynamics.

In the language of simulation of Multibody Dynamics, a function evaluation is equivalent to independent acceleration computation. Given independent positions and velocities, computing independent accelerations requires solution of a set non-linear equations to retrieve depended coordinates and a linear system to obtain dependent velocities. Finally, after evaluating the composite mass matrix and generalized force vector, independent accelerations are obtained by solving the linear system of Eq. (2.6).

The most widely used multi-step formulas belong to the Adams family. They are typically used for non-stiff ODE integration. Multi-step formulas are also effective when dense output is required. This feature regards the capability of a method to generate cheap numerical approximations of the solution and its derivative between grid points. It is important for practical questions such as graphical output, or event location.

Multi-step methods are fast and reliable over a large range of tolerances, due to the fact that they can vary both order and step size automatically. In terms of error control, current implementations of multi-step methods use an estimate of the local truncation error, via a scaled predictor-corrector difference (Eich et. al, 1995). Using past solution and derivative information, multi-step methods construct an interpolating polynomial that is used to produce the numerical solution at the new time step. The grid points for this polynomial can be chosen arbitrarily (Krogh, 1969), or equidistant (Gear, 1971). In the latter case, a time step change requested by the error control mechanism requires determination of the new off-grid values by interpolation. An order change is much simpler in this respect. It is done by adding more past values of the solution and/or derivative. Families of formulas such as Adams are very attractive from this standpoint. The opposite is true for Nordsieck-based multi-step formulas (Nordsieck, 1962); i.e., step-size change only requires rescaling of each entry of the Nordsieck vector, while an order change is more complex.

A drawback of multi-step formulas is the need for starting values. Starting multi-step formulas is usually done by using one step formulas to generate the required number of past values, but more recently, self-starting multi-step methods that start with order one and very small step lengths have gained acceptance. This drawback is a matter of concern with state-space methods, because a change of parametrization usually results in a restart of integration. Furthermore, the method is not recommended for integration of intermittent motion, when repeated integration restarts must be effectively dealt with.

There are several good codes that are based on multi-step methods. The most popular is DEABM (Shampine and Gordon, 1975), which belongs to the package DEPAC designed by Shampine and Watts (1979). The code implements the variable step size, divided difference representation of the Adams formulas, using the Predict-Estimate-Predict-Estimate (PECE) strategy. The implementation requires two functions evaluations for each successful step.

VODE implements the Adams method in Nordsieck form. It is due to Brown et. al, (1989). The code is recommended for problems with widely different active time scales. The nonlinear equation is solved (for non-stiff initial value problems) by fixed point iteration, and for many steps one iteration is sufficient. The order selection strategy is to maximize the step-size. The order is kept constant over long intervals, which is reasonable since a change of order for the Nordsieck representation is expensive.

Finally, for non-stiff ODE, LSODE is another implementation of the Adams family, based on the fixed step size Nordsieck representation. It behaves similarly to VODE.

Hairer, Nørsett, and Wanner (1993), carried out numerical experiments on a set of small and large problems to compare these multi-step codes. For the sake of completeness, they also included the code DOPRI853 (Dormand and Prince, 1980), based on a one step method (Runge-Kutta) that is discussed below. Their conclusion was that LSODE and DEABM require, for equal accuracy, usually less function evaluations, with DEABM being the best for high precision ($Tol \leq 10^{-6}$). This suggests that when compared to the error control mechanism in LSODE, the one in DEABM is better designed. When computing time was measured instead of function evaluations, the situation changed dramatically in favor of DOPRI853. This was observed for small problems in which derivative evaluation is cheap. When the derivative is expensive to evaluate, the discrepancy is not as large. This is explained by the overhead of the error control mechanism for multi-step methods, compared to the one in DOPRI853, and by the cost of derivative evaluation for the test problems considered.

Single step methods for solving ODE require only knowledge of the solution at the current point, in order to advance the solution to the next grid-point. The most used single-step methods are Runge-Kutta (RK) methods and extrapolation methods. The truncation error for RK methods can be controlled in a more straightforward manner than for multi-step methods. However, these methods require more function evaluations, and this is a matter of concern in multibody dynamics.

Error control for RK methods is based almost always on step size selection only. The error estimation is obtained using either extrapolation methods, or (in most cases) embedded formulas of different order. The latter approach computes two approximations of the solution, $\mathbf{y}_1$ and $\hat{\mathbf{y}}_1$, and an estimate of the local truncation error is obtained as $\mathbf{y}_1 - \hat{\mathbf{y}}_1$. Componenetwise, this error is required to be smaller than some limit value that is computed based on user prescribed absolute and relative tolerances that are ensured by the error control mechanism.

The question of which value to choose, $\mathbf{y}_1$ or $\hat{\mathbf{y}}_1$, to continue the integration is usually answered by taking the value obtained with the method of higher order. This strategy is called local extrapolation. The rationale is that, due to unknown stability properties of the differential system, local errors have little in common with global error. Therefore, the error estimate $\mathbf{y}_1 - \hat{\mathbf{y}}_1$ is used solely for step-size selection and afterwards discarded.

There are many codes based on Runge-Kutta methods. Below are presented several that are commonly used. These codes are characterized by good error control that leads to efficient implementations. The code RKF45 was written by Shampine and Watts (1979). It is based on a pair of embedded formulas of order 4 and 5, and it uses local extrapolation. Because of the low orders considered, except for precision, the code requires the largest number of functional calls, compared to DOPRI, DOPRI853, and DVERK, which are discussed below. When the comparison is made in terms of CPU time, RKF45 shows some improvement relative to the performance of DOPRI5, DOPRI853, and DVERK, suggesting small overhead.

The code DOPRI5 is based on a RK method of order 5, with an embedded error estimator of order 4, as proposed by Dormand and Prince. Since the coefficients of the method are carefully chosen, error constants are much more optimized than for RKF45. Thus, for comparable numerical effort, between half and one digit better numerical precision is obtained for DOPRI5, compared to RKF45. The code performs well for medium precision error tolerances (between 10^{-3} and 10^{-5}).

The code DOPRI853 was theoretically expected to perform well for high accuracy. However, it outperformed codes based on lower order formulas even for low accuracy. Whenever more than 3 or 4 exact digits are desired, this code is recommended. The method is of order 8, while the error estimator is based on a 5th order method with 3rd order correction.

The code DVERK is based on a 6th order method due to Verner (1978), and is included in the IMSL library. The error constants are less optimal, and this code surpasses the performance of DOPRI5 only for very stringent error tolerances.

Another class of one step methods is based on extrapolation techniques. These methods are less popular than multi-step and RK methods. They are usually used when very high accuracy (errors less than 10^{-12}) is sought. Extrapolation methods use asymptotic expansion of the global error to successively eliminate more and more terms of the truncation error associated to an integration formula by repeated extrapolation. The method generates a tableau of numerical results that form a sequence of embedded methods. This allows easy estimates of local error and strategies for variable order formulas (Hairer, Nørsett, and Wanner, 1993). The method can easily generate dense output, and is adequate for multi-rate integration of ODE.

The best known code based on the extrapolation method is the code ODEX of Hairer, Nørsett, and Wanner (1993). Numerical experiments show good performance of the code, especially for stringent accuracy where it outperformed DOPRI853.

Many of the explicit codes discussed so far do a very poor job when applied for the solution of stiff initial value problems. While practical causes for stiffness are intuitive, there is controversy about its correct mathematical definition. The eigenvalues of the Jacobian of the derivative function play a key role in deciding if an initial value problem is stiff or not, but other quantities such as the smoothness of the solution, the dimension of the problem, and the length of the integration interval are also important (Hairer and Wanner, 1996).

From a practical standpoint, engineering applications such as the dynamic analysis of a car, tractor semi-trailer, etc., result in stiff problems, due to bushings, dampers, stiff springs, and flexible components present in the models. This indicates that stiff integration formulas must be used quite frequently for efficient solution of these problems.

The best known one step RK methods for stiff ODE are the collocation methods, diagonally implicit RK (DIRK) methods, and Rosenbrock-type methods. All these methods are implicit.

The most common collocation methods are fully implicit RK methods, with intermediate steps that are usually the zeros of certain orthogonal polynomials. The number of stages is rather low, because of limitation imposed by the nonlinear system that must be solved at each time step. Thus, if the dimension of the ODE is n, an s stage RK method of this type requires the solution of a nonlinear system of dimension $n \cdot s$ at each time step. The best known method in this class is RADAU5 (Hairer and Wanner, 1996). It is based on Radau quadrature, the intermediate points $c_1,\dots,c_s$ of the s stage RK method being the zeros of the polynomial

$$\frac{\mathrm{d}^{s-1}}{\mathrm{d}x^{s-1}}\left[x^{s-1}(x-1)^s\right]$$

and the weights b_i are determined by the quadrature conditions

$$\sum_{i=1}^{s} b_i c_i^{q-1} = \frac{1}{q} \ , \qquad q = 1,\dots,s$$

There are also good methods based on Lobatto (Axelsson, 1972), and Gauss (Ehle, 1968) quadrature formulas.

When compared to s stage explicit RK methods, the order of the s stage fully implicit methods is large. Furthermore, the s stage methods based on Gaussian quadrature of order $2\cdot s$, $2\cdot s-1$ for Radau, and $2\cdot s-2$ for Lobatto, have excellent stability properties. Thus, the 3 stage Radau and Lobatto formulas of order 5 and 4, respectively, are A-stable, a property that for multi-step methods can be associated only with orders up to two (Dahlquist, 1963).

Diagonally implicit Runge-Kutta (DIRK) methods are defined as

$$\begin{aligned} \mathbf{k}_i &= h\mathbf{f}(x_0 + c_i h, \mathbf{y}_0 + \sum_{j=1}^{i} a_{ij}\mathbf{k}_j) \ , \qquad i = 1,\dots,s \\ \mathbf{y}_1 &= \mathbf{y}_0 + \sum_{i=1}^{s} b_i \mathbf{k}_i \end{aligned} \tag{2.25}$$

They do not require the costly solution of a system of nonlinear equations of dimension $n\cdot s$. Instead, these methods solve, at each stage, a system of nonlinear equations of dimensions n, for the stage variables $\mathbf{k}_i$.
The efficiency of these methods is further improved if all elements a_{ii} in Eq. (2.25) are identical. The resulting methods are called singly diagonal implicit Runge-Kutta (SDIRK). In this case, the Jacobian matrix used by the quasi-Newton algorithm is

identical at each stage, one factorization of this matrix allowing for recovery of all stage values $\mathbf{k}_i$. The better efficiency is obtained at the price of lower order methods, compared to the fully implicit approach. However, s stage SDIRK methods can result in order s methods that are L stable (Hairer and Wanner, 1996).

Finally, Rosenbrock-type formulas attempt to improve efficiency by applying only one Newton iteration for the solution $\mathbf{k}_i$ required by DIRK methods. When applied to an autonomous differential equation, a DIRK method is linearized to yield

$$\begin{aligned} \mathbf{k}_i &= h[\mathbf{f}(\mathbf{g}_i) + a_{ii}\mathbf{f}'(\mathbf{g}_i)\mathbf{k}_i] \\ \mathbf{g}_i &= \mathbf{y}_0 + \sum_{j=1}^{i-1} a_{ij}\mathbf{k}_j \end{aligned} \tag{2.26}$$

Important computational advantage is obtained by replacing the Jacobian $\mathbf{f}'(\mathbf{g}_i)$ by $\mathbf{J} = \mathbf{f}'(\mathbf{y}_0)$, so that the method requires its calculation and factorization only once. Additional linear combinations of terms $\mathbf{Jk}_j$ are introduced in Eq. (2.26) (Kaps and Rentrop, 1979) to obtain an s stage Rosenbrock method,

$$\begin{aligned} \mathbf{k}_i &= h\mathbf{f}(\mathbf{y}_0 + \sum_{j=1}^{i-1} a_{ij}\mathbf{k}_j) + h\mathbf{J}\sum_{j=1}^{\mathrm{i}} \gamma_{\mathrm{ij}}\mathbf{k}_j, \qquad i = 1,\ldots,s \\ \mathbf{y}_1 &= \mathbf{y}_0 + \sum_{i=1}^{s} b_i\mathbf{k}_i \end{aligned} \tag{2.27}$$

where a_{ij}, γ_{ij}, and b_i are formula defining coefficients and $\mathbf{J} = \mathbf{f}'(\mathbf{y}_0)$.

In terms of step-size control, with some modifications, strategies based on Richardson extrapolation or embedded methods are as for explicit RK methods. Integrators for stiff initial value problems replace the quantity $\hat{\mathbf{y}}_1 - \mathbf{y}_1$ previously used for step-size selection by $\mathbf{P}(\hat{\mathbf{y}}_1 - \mathbf{y}_1)$, where $\mathbf{P} = (\mathbf{I} - h\gamma\mathbf{J})^{-1}$ depends on the method being used through the coefficient γ. The new error estimate for stiff equations is more reliable, compared to the older one, which becomes unbounded for the typical Dahlquist test problem $y' = \lambda y$, $y(0) = 1$.

Newton-like methods are used to solve the discretized systems of nonlinear equations. Since the step-size appears in expressions for matrices that must be factored (such as in $\mathbf{P}$ above), a step-size change requires refactorization of these matrices. Therefore, the step-size is decreased, based on stability consideration, whenever necessary, but it is only increased when the estimated new step-size h_{new} is significantly larger than the current one. Hairer and Wanner (1996), in their step size control mechanism for RADAU5, do not change the step size as long as $c_1 h_{old} \le h_{new} \le c_2 h_{old}$, with $c_1 = 1.0$ and $c_2 = 1.2$. The values for these constants should be adjusted to take into account the dimension of the problem and the complexity of the LU factorization associated with a step-size change.

In terms of multi-step formulas, much as numerical integration considerations lead to the Adams family of formulas for the numerical solution of non-stiff ODE, numerical differentiation is used to obtain the family of BDF that, due to their stability properties, are well suited for numerical integration of stiff initial value problems. The idea is to determine an interpolation polynomial satisfying $\mathrm{P_k}(x_{n+1-j}) = y_{n+1-j}$, $j = 0,1,\ldots,k$, and require that at the new configuration x_{n+1},

$$\mathrm{P}'_\mathrm{k}(x_{n+1}) = f(x_{n+1}, y_{n+1}) \qquad (2.28)$$

The value $y_{n+1} = \mathrm{P_k}(x_{n+1})$ is taken as the numerical solution of the ODE at grid point x_{n+1}. The BDF are implicit formulas, since recovering y_{n+1} amounts to solving a system of non-linear algebraic equations resulting from Eq. (2.28).

Although the procedure outlined above produces formulas of any order, the high order formulas can not be used in practice. If the step-size is constant, formulas of order 7 and higher are computationally useless, because they are not stable (Shampine, 1994). Furthermore, BDF are not as accurate as Adams-Moulton formulas of the same order, and only relatively low order BDF are stable. Nevertheless, the stable BDF are so much more stable than the Adams formulas of the same order that they are essential for the solution of stiff problems.

If simple calculations are carried out, BDF can be brought to the standard form

$$y_{n+1} = \sum_{i=1}^{k} \gamma_i y_{n+1-i} + h\gamma_0 f(x_{n+1}, y_{n+1}) \qquad (2.29)$$

These formulas come in families of order 1, 2, 3, etc.. The lowest order is one (backward Euler), which is a one step method. Starting integration with this formula, which is L stable, and building up to an appropriate order is the usual strategy for BDF-based codes. The order increase is eventually followed by a step-size increase. This strategy is recent (Shampine and Zhang, 1990), while older implementations (Brankin et al., 1988) start the integration using RK methods with dense output capabilities.

Compared to explicit multi-step formulas, the error control mechanism for BDF is more conservative, in terms of step-size variation. A change of step-size occurs when the step fails on accuracy considerations, or convergence of the iterative process that retrieves y_{n+1} is unacceptable slow. On the other hand, the step-size is increased when the overhead associated to this process is worthwhile. There are two main sources of overhead. First, any step-size change requires refactoring certain matrices used in the iterative process. Second, a step-size change is equivalent to an integration restart, since past information to suit the new step-size must be generated.

Although it does not explicitly appear in Eq. (2.29), it is usually assumed that the grid points are equidistant; i.e., past values $y_n, \ldots, y_{n-k}$ are obtained with the same step-size h. There are implementations that allow arbitrary grid points, but such methods must recompute some of the coefficients γ_i of Eq. (2.29) at each time step.

Finally, for most methods, adjusting step-size is more difficult than changing order. Frequent time step changes corrupt the stability properties of BDF methods, especially if the step change occurs more often than once every k steps, where k is the number of past values used in Eq. (2.29).

Hairer and Wanner (1996) performed numerical experiments to compare the performance of several ODE codes based on one and multiple-step methods for integration of stiff initial value problems. For multi-step methods, the codes available are basically the same as for explicit integrators. Nevertheless, they use implicit formulas, and the user must decide if the problem should be deemed stiff or not. Among the codes based on multi-step methods, VODE, LSODE, and DEBDF were compared. All of them are variable order, variable step-size implementations. The code VODE was briefly discussed previously, and it is based on BDF methods on a non-uniform grid. The codes LSODE and DEBDF are very similar, and are based on Nordsieck representation of the uniform grid BDF methods. They are similar in performance, DEBDF being usually slightly faster than LSODE. When used to integrate small problems, the codes LSODE and DEBDF were faster than VODE. The difference vanished when larger problems were considered. When comparing the performance of multi-step and one step methods for small problems, the code RADAU5, a 3 stage, order 5 fully implicit RK method, outperformed the multi-step methods. This trend reverses when the codes are tested on large problems, because for RADAU5 the dimension of the nonlinear system to be solved at each step becomes rather large. However, RADAU5 proves to be a reliable code with consistent behavior over all test problems considered.

In terms of one step methods, RADAU5 and the code RODAS based on 5 stage, 4th order Rosenbrock method with embedded error control of order 3 (Hairer and Wanner, 1996), turned out to be the best. For small problems, RODAS appears to perform better than RADAU5. It should be mentioned that the latter code is also used to integrate low index DAE. One drawback of the codes based on Rosenbrock methods is the requirement to provide an accurate Jacobian. For complex problems, when an analytical representation of the Jacobian can not be provided, numerical techniques to generate this information are employed, and this corrupts performance.

Finally, there are codes based on implicit extrapolation formulas ; i. e., , SODEX and SEULEX (the stiff versions of ODEX and EULEX), that have not been described here. For very stringent error tolerances (usually less than 10^{-12}), these are the methods of choice. A presentation of these methods can be found in the work of Bader and Deuflhard (1983) and the book of Hairer and Wanner (1996).

CHAPTER 3
THEORETICAL CONSIDERATIONS

3.1 Generalized Coordinates Partitioning Algorithm

In Chapter 2 several methods for the solution of differential-algebraic equations (DAE) of Multibody Dynamics were proposed, each with its own merits and drawbacks. This thesis is not aimed at the otherwise worthy goal of comparing these methods in a consistent and unitary manner, but rather its objective is to extend the range of applicability of the state-space method using implicit numerical integrators. The focus is set primarily on coordinate partitioning based state-space reduction (Wehage and Haug, 1992). Theoretical considerations pertaining to tangent-plane parametrization-based state-space reduction are made in Appendix B, but this alternative has not been numerically investigated.

Central to the coordinate partitioning based DAE-to-ODE reduction is the notion of partitioning the vector of generalized coordinates $\mathbf{q}$ into dependent and independent vectors $\mathbf{u}$ and $\mathbf{v}$, respectively. Let $\mathbf{q}_0$ be a consistent configuration of the mechanical system; i. e., $\Phi(\mathbf{q}_0) = \mathbf{0}$. In this configuration, the constraint Jacobian $\Phi_{\mathbf{q}}$ is evaluated and numerically factored using the Gaussian elimination with full pivoting (Atkinson, 1989), yielding

$$\Phi_{\mathbf{q}}(\mathbf{q}_0) \xrightarrow{Gauss} [\Phi_{\mathbf{u}}^* | \Phi_{\mathbf{v}}^*] \tag{3.1}$$

where, provided the Jacobian is full row rank, the matrix $\Phi_{\mathbf{u}}^*$ is upper triangular and

$$\det(\Phi_{\mathbf{u}}^*) \neq 0 \tag{3.2}$$

Let $\sigma(i), i = 1, \ldots, n$ be the set of column permutations done during the Gaussian elimination algorithm. By formally rearranging the generalized coordinates in the vector $\mathbf{q}$ such that

$$\mathbf{q}_{new}(i) = \mathbf{q}(\sigma(i)), \qquad i = 1, \ldots, n \tag{3.3}$$

the first m components of $\mathbf{q}_{new}$ comprise the vector of dependent coordinates $\mathbf{u}$, while the remaining $ndof \equiv n - m$ components form the vector of independent coordinates $\mathbf{v}$.

It will be assumed henceforth that reordering has been done, so the permutation array is equal to the identity; i. e.,

$$\sigma(i) = i, \qquad i = 1, \ldots, n \tag{3.4}$$

Thus, no column permutation is necessary during the factorization process. This is not a limiting assumption, and it is introduced here to simplify the notation and enhance the clarity of the presentation. Under this assumption, the kinematic constrained equations at position, velocity, and acceleration levels, along with the Newton-Euler form of the equations of motion can be expressed in partitioned form as

$$\mathbf{M}^{\mathbf{vv}}(\mathbf{u},\mathbf{v})\ddot{\mathbf{v}} + \mathbf{M}^{\mathbf{vu}}(\mathbf{u},\mathbf{v})\ddot{\mathbf{u}} + \Phi_{\mathbf{v}}^{\mathbf{T}}(\mathbf{u},\mathbf{v})\lambda = \mathbf{Q}^{\mathbf{v}}(\mathbf{u},\mathbf{v},\dot{\mathbf{u}},\dot{\mathbf{v}}) \tag{3.5}$$

$$\mathbf{M}^{\mathbf{uv}}(\mathbf{u},\mathbf{v})\ddot{\mathbf{v}} + \mathbf{M}^{\mathbf{uu}}(\mathbf{u},\mathbf{v})\ddot{\mathbf{u}} + \Phi_{\mathbf{u}}^{\mathbf{T}}(\mathbf{u},\mathbf{v})\lambda = \mathbf{Q}^{\mathbf{u}}(\mathbf{u},\mathbf{v},\dot{\mathbf{u}},\dot{\mathbf{v}}) \tag{3.6}$$

$$\Phi(\mathbf{u},\mathbf{v}) = \mathbf{0} \tag{3.7}$$

$$\Phi_{\mathbf{u}}(\mathbf{u},\mathbf{v})\dot{\mathbf{u}} + \Phi_{\mathbf{v}}(\mathbf{u},\mathbf{v})\dot{\mathbf{v}} = \mathbf{0} \tag{3.8}$$

$$\Phi_{\mathbf{u}}(\mathbf{u},\mathbf{v})\ddot{\mathbf{u}} + \Phi_{\mathbf{v}}(\mathbf{u},\mathbf{v})\ddot{\mathbf{v}} = \tau(\mathbf{u},\mathbf{v},\dot{\mathbf{u}},\dot{\mathbf{v}}) \tag{3.9}$$

The condition of Eq. (3.2) implies that

$$\det(\Phi_{\mathbf{u}}) \neq 0 \tag{3.10}$$

over some time interval. The implicit function theorem (Corwin, Szczarba, 1982) guarantees that Eq. (3.7) can be locally solved in a neighborhood of the consistent configuration $\mathbf{q}_0$ for $\mathbf{u}$ as a function of $\mathbf{v}$. Thus,

$$\mathbf{u} = \mathbf{g}(\mathbf{v}) \tag{3.11}$$

where the function $\mathbf{g}(\mathbf{v})$ has as many continuous derivatives as does the constraint function $\Phi(\mathbf{q})$. In other words, at an admissible configuration $\mathbf{q}_0$, there exist neighborhoods $U_1(\mathbf{v}_0)$ and $U_2(\mathbf{u}_0)$, and a function $\mathbf{g}: U_1 \to U_2$, such that for any $\mathbf{v} \in U_1$, Eq. (3.7) is identically satisfied when $\mathbf{u}$ is as given by Eq. (3.11). Note that the dependency of $\mathbf{u}$ as a function of $\mathbf{v}$ in Eq. (3.11) it is not explicitly determined, but is a theoretical result that enables DAE-to-ODE reduction.

Since the coefficient matrix $\Phi_{\mathbf{u}}$ in Eq. (3.8) is nonsingular, $\dot{\mathbf{u}}$ can be expressed in terms of $\mathbf{v}$ and $\dot{\mathbf{v}}$, as

$$\dot{\mathbf{u}} = -\Phi_{\mathbf{u}}^{-1}\Phi_{\mathbf{v}}\dot{\mathbf{v}} \equiv \mathbf{H}\dot{\mathbf{v}} \tag{3.12}$$

The dependency of the quantities in Eq. (3.12) and the following equations on $\mathbf{v}$ and $\dot{\mathbf{v}}$ is suppressed for notational simplicity. Following the same argument, the dependent accelerations are expressed as function of independent positions, velocities, and accelerations using Eqs. (3.9), (3.11), and (3.12),

$$\ddot{\mathbf{u}} = \mathbf{H}\ddot{\mathbf{v}} + \Phi_{\mathbf{u}}^{-1}\tau \tag{3.13}$$

Finally, Lagrange multipliers are formally expressed as function of independent positions, velocities, and accelerations, by using Eq. (3.6) to obtain

$$\lambda = \Phi_{\mathbf{u}}^{-T}[\mathbf{Q}^{\mathbf{u}} - \mathbf{M}^{\mathbf{uv}}\ddot{\mathbf{v}} - \mathbf{M}^{\mathbf{uu}}(\mathbf{H}\ddot{\mathbf{v}} + \Phi_{\mathbf{u}}^{-1}\tau)] \tag{3.14}$$

Once $\mathbf{u}$, $\dot{\mathbf{u}}$, $\ddot{\mathbf{u}}$, and λ are formally expressed as functions of independent variables, the DAE is reduced to a second order ODE called the state-space ODE (SSODE), which is obtained by substituting the dependent variables in Eq. (3.5) to yield

$$\hat{\mathbf{M}}\ddot{\mathbf{v}} = \hat{\mathbf{Q}} \tag{3.15}$$

where

$$\hat{\mathbf{M}} = \mathbf{M}^{\mathbf{vv}} + \mathbf{M}^{\mathbf{vu}}\mathbf{H} + \mathbf{H}^{\mathrm{T}}(\mathbf{M}^{\mathbf{uv}} + \mathbf{M}^{\mathbf{uu}}\mathbf{H}) \tag{3.16}$$

$$\hat{\mathbf{Q}} = \mathbf{Q}^{\mathbf{v}} + \mathbf{H}^{\mathrm{T}}\mathbf{Q}^{\mathbf{u}} - (\mathbf{M}^{\mathbf{vv}} + \mathbf{H}^{\mathrm{T}}\mathbf{M}^{\mathbf{uu}})\Phi_{\mathbf{u}}^{-1}\tau \tag{3.17}$$

An argument based on the positive definiteness of the quadratic form associated with kinetic energy of any mechanical system is at the foundation of a result (Haug, 1989) that states that the coefficient matrix $\hat{\mathbf{M}}$ in Eq. (3.15) is positive definite. Therefore, the system in Eq. (3.15) has a unique solution at each time step, which is used to numerically advance the integration to the next time step.

From a practical standpoint, the independent accelerations are not computed by first evaluating the matrices $\hat{\mathbf{M}}$ and $\hat{\mathbf{Q}}$, and then solving the system in Eq. (3.15). This strategy for solving for independent accelerations would be inefficient, because of the costly matrix-matrix multiplications involved. Rather, an approach based on first solving the linear system of Eq. (2.6), and then extracting the independent accelerations $\ddot{\mathbf{v}}$ from the set of generalized accelerations $\ddot{\mathbf{q}}$, based on the permutation array σ, is more efficient. Details about this strategy are given in Sections 3.4.2 and 3.4.3.

In this context, it is worth pointing out that efficient state-space based explicit integration of the DAE of Multibody Dynamics requires in the first place a fast method to obtain the independent accelerations $\ddot{\mathbf{v}}$. Using an explicit integration formula, the independent positions $\mathbf{v}$, and velocities $\dot{\mathbf{v}}$ are obtained at the next time step by integrating the initial value problem (IVP) $\ddot{\mathbf{v}} = \mathbf{f}(t, \mathbf{v}, \dot{\mathbf{v}})$ (with $\mathbf{f} = \hat{\mathbf{M}}^{-1}\hat{\mathbf{Q}}$) for a given set of initial conditions $\mathbf{v}_0$ and $\dot{\mathbf{v}}_0$; then using Eqs. (3.7) and (3.8) $\mathbf{u}$ and $\dot{\mathbf{u}}$ at the new time step can be obtained. These two last stages amount to the solution of a set of non-linear equations and a set of linear equations, respectively. Thus, for explicit integration the linear algebra stage is central. It appears during each of the three steps of the process (computation of $\ddot{\mathbf{v}}$, recovery of $\mathbf{u}$, recovery of $\dot{\mathbf{u}}$) and the extent to which topology information of the mechanical system model is taken into account will determine the overall performance of the method.

The following Sections address the issue of implicit integration in the framework of the state-space method. Both multi-step and Runge-Kutta methods are considered. Theoretical considerations in these Sections are based on results of Haug, Negrut and Iancu (1997a), and Negrut, Haug, and Iancu (1997).

3.2 State-Space Reduction Method

In the state-space reduction framework, several alternatives for DAE-to-ODE reduction are available. Although not mentioned explicitly in the title, the method presented in this section uses the coordinate partitioning based reduction algorithm (Wehage and Haug, 1982). Multi-step and Runge-Kutta integration formulas are going to be used to integrate the resulting SSODE.

3.2.1 Multi-step Methods

3.2.1.1 General Considerations on Multi-step Methods

This section contains a brief introduction of the multi-step numerical integration methods. Characteristic properties relevant in the context of implicit integration of the DAE of Multibody Dynamics via state-space reduction are presented. The works of Gear (1971), Hairer, Nørsett, and Wanner (1993), and Hairer and Wanner (1996) should be consulted for a comprehensive treatment of the topic of multi-step numerical integration methods.

Consider the general implicit multi-step integration formula (Atkinson, 1989)

$$y_{n+1} = \sum_{j=0}^{p} a_j y_{n-j} + h \sum_{j=-1}^{p} b_j f(x_{n-j}, y_{n-j}) \tag{3.18}$$

where h is the integration step-size, $a_0, \ldots, a_p, b_{-1}, b_0, \ldots, b_p$ are constants, and $p \geq 0$. This integration formula is rewritten as

$$y_{n+1} = \tilde{y}_{n+1} + bh f(t_{n+1}, y_{n+1}) \tag{3.19}$$

where $b \equiv b_{-1}$, and

$$\tilde{y}_{n+1} = \sum_{j=0}^{p} a_j y_{n-j} + h \sum_{j=0}^{p} b_j f(x_{n-j}, y_{n-j})$$

includes all the terms that do not depend on y_{n+1}. Depending on the set of coefficients $a_0, \ldots, a_p, b_{-1}, b_0, \ldots, b_p$, the multi-step formula will have certain order and stability properties.

In what follows, multi-step methods are applied for numerical integration of the generic second order IVP

$$\ddot{\mathbf{v}} = \mathbf{f}(t, \mathbf{v}, \dot{\mathbf{v}}) \tag{3.20}$$

with $\mathbf{v}(t_0) = \mathbf{v}_0$ and $\dot{\mathbf{v}}(t_0) = \dot{\mathbf{v}}_0$. This development will be instrumental in presenting an approach for the multi-step state-space based integration of DAE of Multibody Dynamics in the next Section. Moreover, anticipating the significance of $\mathbf{v}$, $\dot{\mathbf{v}}$, and $\ddot{\mathbf{v}}$, these variables are called the independent positions, velocities, and accelerations.

Generally, two different formulas could be used, first to integrate independent accelerations to obtain independent velocities, and then to integrate independent velocities to obtain independent positions. Based on Eq. (3.19), independent velocities and positions can be expressed as

$$\dot{\mathbf{v}}_{n+1} = \tilde{\dot{\mathbf{v}}}_{n+1} + \gamma h \ddot{\mathbf{v}}_{n+1} \tag{3.21}$$

$$\mathbf{v}_{n+1} = \overline{\mathbf{v}}_{n+1} + \overline{\beta} h \dot{\mathbf{v}}_{n+1} \tag{3.22}$$

Notice that $\tilde{\dot{\mathbf{v}}}_{n+1}$ and $\overline{\mathbf{v}}_{n+1}$ contain only past information, and generally $\gamma \neq \overline{\beta}$. Using Eq. (3.21) to express independent velocities in Eq. (3.22) in terms of independent accelerations, the independent positions assume the form

$$\mathbf{v}_{n+1} = \tilde{\mathbf{v}}_{n+1} + \beta h^2 \ddot{\mathbf{v}}_{n+1} \tag{3.23}$$

where

$$\tilde{\mathbf{v}}_{n+1} \equiv \overline{\mathbf{v}}_{n+1} + \overline{\beta} h \tilde{\dot{\mathbf{v}}}_{n+1}$$

$$\beta \equiv \gamma \overline{\beta}$$

Discretizing the second order ODE in Eq. (3.20), and using Eqs. (3.21) and (3.23) yields

$$\ddot{\mathbf{v}}_{n+1} = \mathbf{f}\left(t, \mathbf{v}_{n+1}(\ddot{\mathbf{v}}_{n+1}), \dot{\mathbf{v}}(\ddot{\mathbf{v}}_{n+1})\right) \tag{3.24}$$

The dependence of independent positions and velocities on independent accelerations is via the integration formulas (3.23) and (3.21), respectively. Equation (3.24) is a system of non-linear algebraic equations that must be solved at each time step for the set of independent accelerations $\ddot{\mathbf{v}}_{n+1}$. Recasting this system into the form

$$\Psi(\ddot{\mathbf{v}}_{n+1}) \equiv \ddot{\mathbf{v}}_{n+1} - \mathbf{f}\left(t, \mathbf{v}_{n+1}(\ddot{\mathbf{v}}_{n+1}), \dot{\mathbf{v}}(\ddot{\mathbf{v}}_{n+1})\right) = \mathbf{0} \tag{3.25}$$

its solution is found by means of a quasi-Newton method. For this, Jacobian information must be provided to the non-linear solver. After applying the chain rule of differentiation, the Jacobian (henceforth this quantity is going to be called integration Jacobian to avoid confusion with the constraint Jacobian $\Phi_{\mathbf{q}}$) is obtained as

$$\boldsymbol{\Psi}_{\ddot{\mathbf{v}}} = \mathbf{I} - \mathbf{f}_{\dot{\mathbf{v}}} \cdot \dot{\mathbf{v}}_{\ddot{\mathbf{v}}} - \mathbf{f}_{\mathbf{v}} \cdot \mathbf{v}_{\ddot{\mathbf{v}}} \qquad (3.26)$$

In Eq. (3.26), subscripts $n+1$ have been dropped for convenience. Using Eqs. (3.21) and (3.23), the derivatives of independent positions and velocities with respect to independent accelerations are readily obtained as

$$\mathbf{v}_{\ddot{\mathbf{v}}} = \beta h^2 \mathbf{I} \qquad (3.27)$$

$$\dot{\mathbf{v}}_{\ddot{\mathbf{v}}} = \gamma h \mathbf{I} \qquad (3.28)$$

where $\mathbf{I}$ is the identity matrix of appropriate dimension. Finally, substituting the expressions of $\mathbf{v}_{\ddot{\mathbf{v}}}$ and $\dot{\mathbf{v}}_{\ddot{\mathbf{v}}}$ in the expression for the integration Jacobian yields

$$\boldsymbol{\Psi}_{\ddot{\mathbf{v}}} = \mathbf{I} - \gamma h \cdot \mathbf{f}_{\dot{\mathbf{v}}} - \beta h^2 \cdot \mathbf{f}_{\mathbf{v}} \qquad (3.29)$$

The derivatives $\mathbf{f}_{\dot{\mathbf{v}}}$ and $\mathbf{f}_{\mathbf{v}}$ are problem dependent, and they are to be provided to the non-linear solver. How to provide them for the particular case of SSODE of Multibody Dynamics is the topic of the next Section.

With these considerations, the algorithm proposed for the implicit integration of the second order IVP is as follows:

(1). Provide an initial estimate for the set of independent accelerations $\ddot{\mathbf{v}}$.

(2). Integrate to obtain independent positions and velocities, using the expressions in Eqs. (3.23) and (3.21), respectively.

(3). If stopping criteria are satisfied, then stop. Otherwise, using a quasi-Newton correction, correct the value of $\ddot{\mathbf{v}}$ and go to step (2).

The stopping criteria in step (3) are based on the norm of the error in satisfying the non-linear system $\boldsymbol{\Psi}(\ddot{\mathbf{v}}) = \mathbf{0}$, and on the size of the last correction applied to $\ddot{\mathbf{v}}$.
In a practical implementation, the integration Jacobian $\boldsymbol{\Psi}_{\ddot{\mathbf{v}}}$ is computed once at the end of each successful integration step, and kept constant during the iterative process at least for

the following integration step. This strategy is motivated by the CPU intensive process of generating $\mathbf{f}_{\dot{\mathbf{v}}}$ and $\mathbf{f}_{\mathbf{v}}$.

<u>3.2.1.2 Multi-step State-Space Based Implicit Integration</u>

The previous Section described how an implicit multi-step integration formula is used to integrate a generic set of second order ODE. The generic ODE is replaced here with the SSODE obtained via coordinate partitioning based DAE-to-ODE reduction. The challenge of multi-step state-space based implicit integration lies in providing derivative information, which stands as the backbone of integration Jacobian computation. The focus is thus set on computing what amounts to be the analogue of the quantities $\mathbf{f}_{\dot{\mathbf{v}}}$ and $\mathbf{f}_{\mathbf{v}}$ of the previous Section.

Using a multi-step implicit integration formula to directly discretize the SSODE of Eq. (3.15) is impractical, since solving the resultant set of non-linear equations requires Jacobian information whenever a Newton-like method is involved. For this implicit form ODE, generating the integration Jacobian would be a difficult task. Instead, using Eq. (3.5), from which Eq. (3.15) originates, leads to a consistent and tractable way to obtain the needed derivative information.

Equations (3.21) and (3.23) are used to discretize the independent equations of motion of Eq. (3.5). From a theoretical standpoint, calling these differential equations independent in the sense in which the generalized coordinates were is not rigorous. It is only to underline that these are the equations leading to the state-space ODE of Eq. (3.15) after substitution of all dependent variables. Thus, regarding Eq. (3.5) as a set of second order ODE in independent coordinates $\mathbf{v}$, and substituting $\mathbf{v}_{n+1}$ and $\dot{\mathbf{v}}_{n+1}$ from Eqs. (3.23) and (3.21), yields a set of non-linear equations in independent accelerations at time t_{n+1},

$$\psi(\ddot{\mathbf{v}}_{n+1}) \equiv \mathbf{M}^{\mathbf{vv}}_{n+1}\ddot{\mathbf{v}}_{n+1} + \mathbf{M}^{\mathbf{vu}}_{n+1}\ddot{\mathbf{u}}_{n+1} + (\Phi^{\mathrm{T}}_{\mathbf{v}})_{n+1}\lambda_{n+1} - \mathbf{Q}^{\mathbf{v}}_{n+1} = \mathbf{0} \tag{3.30}$$

The system of non-linear equations $\Psi(\ddot{\mathbf{v}}_{n+1}) = \mathbf{0}$, obtained after discretization of the independent equations of motion is numerically solved for $\ddot{\mathbf{v}}_{n+1}$ by employing a quasi-Newton method.

The process of obtaining the required integration Jacobian turns out to be an exercise in applying the chain rule of differentiation, at the level of kinematic constraint equations and dependent equations of motion.

Taking the derivative of Eq. (3.30) with respect to independent accelerations and using the chain rule of differentiation yields

$$\begin{aligned}\Psi_{\ddot{\mathbf{v}}} &= \mathbf{M}^{\mathbf{vv}} + (\mathbf{M}^{\mathbf{vv}}\ddot{\mathbf{v}})_{\mathbf{u}}\mathbf{u}_{\ddot{\mathbf{v}}} + (\mathbf{M}^{\mathbf{vv}}\ddot{\mathbf{v}})_{\mathbf{v}}\mathbf{v}_{\ddot{\mathbf{v}}} + \mathbf{M}^{\mathbf{vu}}\ddot{\mathbf{u}}_{\ddot{\mathbf{v}}} \\ &\quad + (\mathbf{M}^{\mathbf{vu}}\ddot{\mathbf{u}})_{\mathbf{u}}\mathbf{u}_{\ddot{\mathbf{v}}} + (\mathbf{M}^{\mathbf{vu}}\ddot{\mathbf{u}})_{\mathbf{v}}\mathbf{v}_{\ddot{\mathbf{v}}} + \Phi^{\mathrm{T}}_{\mathbf{v}}\lambda_{\ddot{\mathbf{v}}} + (\Phi^{\mathrm{T}}_{\mathbf{v}}\lambda)_{\mathbf{u}}\mathbf{u}_{\ddot{\mathbf{v}}} + (\Phi^{\mathrm{T}}_{\mathbf{v}}\lambda)_{\mathbf{v}}\mathbf{v}_{\ddot{\mathbf{v}}} \\ &\quad - \mathbf{Q}^{\mathbf{v}}_{\mathbf{u}}\mathbf{u}_{\ddot{\mathbf{v}}} - \mathbf{Q}^{\mathbf{v}}_{\mathbf{v}}\mathbf{v}_{\ddot{\mathbf{v}}} - \mathbf{Q}^{\mathbf{v}}_{\dot{\mathbf{u}}}\dot{\mathbf{u}}_{\ddot{\mathbf{v}}} - \mathbf{Q}^{\mathbf{v}}_{\dot{\mathbf{v}}}\dot{\mathbf{v}}_{\ddot{\mathbf{v}}}\end{aligned} \tag{3.31}$$

In Eq. (3.31), the subscript $n+1$ is suppressed for notational simplicity. In what follows the derivatives $\mathbf{u}_{\ddot{\mathbf{v}}}$, $\dot{\mathbf{u}}_{\ddot{\mathbf{v}}}$, $\ddot{\mathbf{u}}_{\ddot{\mathbf{v}}}$, and $\lambda_{\ddot{\mathbf{v}}}$ are evaluated.

In order to compute the derivative of dependent positions with respect to independent accelerations, the position kinematic constraint equation of Eq. (3.7) is differentiated with respect to $\ddot{\mathbf{v}}$, to obtain $\Phi_{\mathbf{u}}\mathbf{u}_{\ddot{\mathbf{v}}} = -\Phi_{\mathbf{v}}\mathbf{v}_{\ddot{\mathbf{v}}}$. Using Eq. (3.27), this reduces to

$$\mathbf{u}_{\ddot{\mathbf{v}}} = -\beta h^2 \Phi_{\mathbf{u}}^{-1}\Phi_{\mathbf{v}} = \beta h^2 \mathbf{H} \tag{3.32}$$

Taking the derivative of the velocity kinematic constraint equation of Eq. (3.8) with respect to $\ddot{\mathbf{v}}$ yields

$$\begin{aligned}\Phi_{\mathbf{u}}\dot{\mathbf{u}}_{\ddot{\mathbf{v}}} &= -[(\Phi_{\mathbf{u}}\dot{\mathbf{u}})_{\mathbf{u}}\mathbf{u}_{\ddot{\mathbf{v}}} + (\Phi_{\mathbf{u}}\dot{\mathbf{u}})_{\mathbf{v}}\mathbf{v}_{\ddot{\mathbf{v}}} + \Phi_{\mathbf{v}}\dot{\mathbf{v}}_{\ddot{\mathbf{v}}} + (\Phi_{\mathbf{v}}\dot{\mathbf{v}})_{\mathbf{u}}\mathbf{u}_{\ddot{\mathbf{v}}} + (\Phi_{\mathbf{v}}\dot{\mathbf{v}})_{\mathbf{v}}\mathbf{v}_{\ddot{\mathbf{v}}}] \\ &= -\beta h^2[(\Phi_{\mathbf{u}}\dot{\mathbf{u}})_{\mathbf{u}}\mathbf{H} + (\Phi_{\mathbf{u}}\dot{\mathbf{u}})_{\mathbf{v}} + (\Phi_{\mathbf{v}}\dot{\mathbf{v}})_{\mathbf{u}}\mathbf{H} + (\Phi_{\mathbf{v}}\dot{\mathbf{v}})_{\mathbf{v}}] - \gamma h \Phi_{\mathbf{v}}\end{aligned} \tag{3.33}$$

This equation can be further reduced to

$$\begin{aligned}\dot{\mathbf{u}}_{\ddot{\mathbf{v}}} &= -\beta h^2 \Phi_{\mathbf{u}}^{-1}[(\Phi_{\mathbf{q}}\dot{\mathbf{q}})_{\mathbf{u}}\mathbf{H} + (\Phi_{\mathbf{q}}\dot{\mathbf{q}})_{\mathbf{v}}] - \gamma h \Phi_{\mathbf{u}}^{-1}\Phi_{\mathbf{v}} \\ &\equiv \beta h^2 \mathbf{J} + \gamma h \mathbf{H}\end{aligned} \tag{3.34}$$

The derivative $\ddot{\mathbf{u}}_{\ddot{\mathbf{v}}}$ is obtained by differentiating the acceleration kinematic equation of Eq. (3.9) with respect to $\ddot{\mathbf{v}}$, to yield

$$\begin{aligned}\Phi_{\mathbf{u}}\ddot{\mathbf{u}}_{\ddot{\mathbf{v}}} &= \tau_{\mathbf{u}}\mathbf{u}_{\ddot{\mathbf{v}}} + \tau_{\dot{\mathbf{u}}}\dot{\mathbf{u}}_{\ddot{\mathbf{v}}} + \tau_{\dot{\mathbf{u}}}\dot{\mathbf{u}}_{\ddot{\mathbf{v}}} + \tau_{\dot{\mathbf{v}}}\dot{\mathbf{v}}_{\ddot{\mathbf{v}}} - [(\Phi_{\mathbf{q}}\ddot{\mathbf{q}})_{\mathbf{u}}\mathbf{u}_{\ddot{\mathbf{v}}} + (\Phi_{\mathbf{q}}\ddot{\mathbf{q}})_{\mathbf{v}}\mathbf{v}_{\ddot{\mathbf{v}}} + \Phi_{\mathbf{v}}] \\ &= \beta h^2[\tau_{\mathbf{u}}\mathbf{H} + \tau_{\mathbf{v}} + \tau_{\dot{\mathbf{u}}}\mathbf{J} - (\Phi_{\mathbf{q}}\ddot{\mathbf{q}})_{\mathbf{u}}\mathbf{H} - (\Phi_{\mathbf{q}}\ddot{\mathbf{q}})_{\mathbf{v}}] + \gamma h(\tau_{\dot{\mathbf{u}}}\mathbf{H} + \tau_{\dot{\mathbf{v}}}) - \Phi_{\mathbf{v}}\end{aligned} \tag{3.35}$$

Then

$$\begin{aligned}\ddot{\mathbf{u}}_{\ddot{\mathbf{v}}} &= \beta h^2 \Phi_{\mathbf{u}}^{-1}\left\{[\tau_{\mathbf{u}} - (\Phi_{\mathbf{q}}\ddot{\mathbf{q}})_{\mathbf{u}}]\mathbf{H} + \tau_{\mathbf{v}} + \tau_{\dot{\mathbf{u}}}\mathbf{J} - (\Phi_{\mathbf{q}}\ddot{\mathbf{q}})_{\mathbf{v}}\right\} \\ &\quad + \gamma h \Phi_{\mathbf{u}}^{-1}(\tau_{\dot{\mathbf{u}}}\mathbf{H} + \tau_{\dot{\mathbf{v}}}) + \mathbf{H} \equiv \beta h^2\mathbf{L} + \gamma h\mathbf{N} + \mathbf{H}\end{aligned} \tag{3.36}$$

Finally, differentiating the dependent equations of motion of Eq. (3.6) with respect to $\ddot{\mathbf{v}}$ yields

$$\begin{aligned}\Phi_{\mathbf{u}}^{\mathrm{T}}\lambda_{\ddot{\mathbf{v}}} &= \mathbf{Q}_{\mathbf{u}}^{\mathbf{u}}\mathbf{u}_{\ddot{\mathbf{v}}} + \mathbf{Q}_{\mathbf{v}}^{\mathbf{u}}\mathbf{v}_{\ddot{\mathbf{v}}} + \mathbf{Q}_{\dot{\mathbf{u}}}^{\mathbf{u}}\dot{\mathbf{u}}_{\ddot{\mathbf{v}}} + \mathbf{Q}_{\dot{\mathbf{v}}}^{\mathbf{u}}\dot{\mathbf{v}}_{\ddot{\mathbf{v}}} - [(\Phi_{\mathbf{u}}^{\mathrm{T}}\lambda)_{\mathbf{u}}\mathbf{u}_{\ddot{\mathbf{v}}} + (\Phi_{\mathbf{u}}^{\mathrm{T}}\lambda)_{\mathbf{v}}\mathbf{v}_{\ddot{\mathbf{v}}} + \mathbf{M}^{\mathbf{uv}} \\ &\quad + (\mathbf{M}^{\mathbf{uv}}\ddot{\mathbf{v}})_{\mathbf{u}}\mathbf{u}_{\ddot{\mathbf{v}}} + (\mathbf{M}^{\mathbf{uv}}\ddot{\mathbf{v}})_{\mathbf{v}}\mathbf{v}_{\ddot{\mathbf{v}}} + \mathbf{M}^{\mathbf{uu}}\ddot{\mathbf{u}}_{\ddot{\mathbf{v}}} + (\mathbf{M}^{\mathbf{uu}}\ddot{\mathbf{u}})_{\mathbf{u}}\mathbf{u}_{\ddot{\mathbf{v}}} + (\mathbf{M}^{\mathbf{uu}}\ddot{\mathbf{u}})_{\mathbf{v}}\mathbf{v}_{\ddot{\mathbf{v}}}] \\ &= \beta h^2[\mathbf{Q}_{\mathbf{u}}^{\mathbf{u}}\mathbf{H} + \mathbf{Q}_{\mathbf{v}}^{\mathbf{u}} + \mathbf{Q}_{\dot{\mathbf{u}}}^{\mathbf{u}}\mathbf{J} - (\Phi_{\mathbf{u}}^{\mathrm{T}}\lambda)_{\mathbf{u}}\mathbf{H} - (\Phi_{\mathbf{u}}^{\mathrm{T}}\lambda)_{\mathbf{v}} - (\mathbf{M}^{\mathbf{uv}}\ddot{\mathbf{v}})_{\mathbf{u}}\mathbf{H} - (\mathbf{M}^{\mathbf{uv}}\ddot{\mathbf{v}})_{\mathbf{v}} \\ &\quad - \mathbf{M}^{\mathbf{uu}}\mathbf{L} - (\mathbf{M}^{\mathbf{uu}}\ddot{\mathbf{u}})_{\mathbf{u}}\mathbf{H} - (\mathbf{M}^{\mathbf{uu}}\ddot{\mathbf{u}})_{\mathbf{v}}] + \gamma h(\mathbf{Q}_{\dot{\mathbf{u}}}^{\mathbf{u}}\mathbf{H} + \mathbf{Q}_{\dot{\mathbf{v}}}^{\mathbf{u}} - \mathbf{M}^{\mathbf{uu}}\mathbf{N}) - \mathbf{M}^{\mathbf{uv}} - \mathbf{M}^{\mathbf{uu}}\mathbf{H}\end{aligned}$$

and the derivative of Lagrange multipliers with respect to independent accelerations is

$$\begin{aligned}\lambda_{\ddot{\mathbf{v}}} &= \beta h^2 \Phi_{\mathbf{u}}^{-\mathrm{T}}\left\{[\mathbf{Q}_{\mathbf{u}}^{\mathbf{u}} - (\Phi_{\mathbf{u}}^{\mathrm{T}}\lambda)_{\mathbf{u}} - (\mathbf{M}^{\mathbf{uv}}\ddot{\mathbf{v}})_{\mathbf{u}} - (\mathbf{M}^{\mathbf{uu}}\ddot{\mathbf{u}})_{\mathbf{u}}]\mathbf{H} + \mathbf{Q}_{\mathbf{v}}^{\mathbf{u}} + \mathbf{Q}_{\dot{\mathbf{u}}}^{\mathbf{u}}\mathbf{J} - (\Phi_{\mathbf{u}}^{\mathrm{T}}\lambda)_{\mathbf{v}}\right. \\ &\quad \left. - (\mathbf{M}^{\mathbf{uv}}\ddot{\mathbf{v}})_{\mathbf{v}} - (\mathbf{M}^{\mathbf{uu}}\ddot{\mathbf{u}})_{\mathbf{v}} - \mathbf{M}^{\mathbf{uu}}\mathbf{L}\right\} + \gamma h \Phi_{\mathbf{u}}^{-\mathrm{T}}(\mathbf{Q}_{\dot{\mathbf{u}}}^{\mathbf{u}}\mathbf{H} + \mathbf{Q}_{\dot{\mathbf{v}}}^{\mathbf{u}} - \mathbf{M}^{\mathbf{uu}}\mathbf{N}) \\ &\quad - \Phi_{\mathbf{u}}^{-\mathrm{T}}(\mathbf{M}^{\mathbf{uv}} + \mathbf{M}^{\mathbf{uu}}\mathbf{H}) \equiv -\Phi_{\mathbf{u}}^{-\mathrm{T}}(\beta h^2\mathbf{R} + \gamma h\mathbf{S} + \mathbf{M}^{\mathbf{uv}} + \mathbf{M}^{\mathbf{uu}}\mathbf{H})\end{aligned} \tag{3.37}$$

In Eqs. (3.32) through (3.37), the following notations are used:

$$\mathbf{J} = -\Phi_{\mathbf{u}}^{-1}[(\Phi_{\mathbf{q}}\dot{\mathbf{q}})_{\mathbf{u}}\mathbf{H} + (\Phi_{\mathbf{q}}\dot{\mathbf{q}})_{\mathbf{v}}] \tag{3.38}$$

$$\mathbf{L} = -\Phi_{\mathbf{u}}^{-1}\left\{[\tau_{\mathbf{u}} - (\Phi_{\mathbf{q}}\dot{\mathbf{q}})_{\mathbf{u}}]\mathbf{H} + \tau_{\mathbf{v}} + \tau_{\dot{\mathbf{u}}}\mathbf{J} - (\Phi_{\mathbf{q}}\dot{\mathbf{q}})_{\mathbf{v}}\right\} \tag{3.39}$$

$$\mathbf{N} = \Phi_{\mathbf{u}}^{-1}(\tau_{\dot{\mathbf{u}}}\mathbf{H} + \tau_{\dot{\mathbf{v}}}) \tag{3.40}$$

$$\begin{aligned}\mathbf{R} = -\{&[\mathbf{Q}^{\mathbf{u}}_{\mathbf{u}} - (\Phi^{\mathrm{T}}_{\mathbf{u}}\lambda)_{\mathbf{u}} - (\mathbf{M}^{\mathbf{u}}\ddot{\mathbf{q}})_{\mathbf{u}}]\mathbf{H} + \mathbf{Q}^{\mathbf{u}}_{\mathbf{v}} + \mathbf{Q}^{\mathbf{u}}_{\dot{\mathbf{u}}}\mathbf{J} - (\Phi^{\mathrm{T}}_{\mathbf{u}}\lambda)_{\mathbf{v}} \\ &-(\mathbf{M}^{\mathbf{u}}\ddot{\mathbf{q}})_{\mathbf{v}} - \mathbf{M}^{\mathbf{uu}}\mathbf{L}\}\end{aligned} \tag{3.41}$$

$$\mathbf{S} = -(\mathbf{Q}^{\mathbf{u}}_{\dot{\mathbf{u}}}\mathbf{H} + \mathbf{Q}^{\mathbf{u}}_{\dot{\mathbf{v}}} - \mathbf{M}^{\mathbf{uu}}\mathbf{N}) \tag{3.42}$$

After substituting the derivative expressions into Eq. (3.31) and collecting terms according to powers of the step-size h, the integration Jacobian can be expressed as

$$\begin{aligned}\Psi_{\ddot{\mathbf{v}}} = \mathbf{M}^{\mathbf{vv}} + \mathbf{M}^{\mathbf{vu}}\mathbf{H} + \mathbf{H}^{\mathrm{T}}(\mathbf{M}^{\mathbf{uv}} + \mathbf{M}^{\mathbf{uu}}\mathbf{H}) + \gamma h(\mathbf{M}^{\mathbf{vu}}\mathbf{N} + \mathbf{H}^{\mathrm{T}}\mathbf{S} - \mathbf{Q}^{\mathbf{v}}_{\dot{\mathbf{v}}} - \mathbf{Q}^{\mathbf{v}}_{\dot{\mathbf{u}}}\mathbf{H}) \\ +\beta h^2[(\mathbf{M}^{\mathbf{v}}\ddot{\mathbf{q}})_{\mathbf{u}}\mathbf{H} + (\mathbf{M}^{\mathbf{v}}\ddot{\mathbf{q}})_{\mathbf{v}} + \mathbf{M}^{\mathbf{vu}}\mathbf{L} + \mathbf{H}^{\mathrm{T}}\mathbf{R} + (\Phi^{\mathrm{T}}_{\mathbf{v}}\lambda)_{\mathbf{u}}\mathbf{H} + (\Phi^{\mathrm{T}}_{\mathbf{v}}\lambda)_{\mathbf{v}} \\ -\mathbf{Q}^{\mathbf{v}}_{\mathbf{u}}\mathbf{H} - \mathbf{Q}^{\mathbf{v}}_{\mathbf{v}} - \mathbf{Q}^{\mathbf{v}}_{\dot{\mathbf{u}}}\mathbf{J}]\end{aligned} \tag{3.43}$$

Taking into account the definition of $\hat{\mathbf{M}}$, and introducing the notations

$$\hat{\mathbf{M}}_1 \equiv \mathbf{M}^{\mathbf{vu}}\mathbf{N} + \mathbf{H}^{\mathrm{T}}\mathbf{S} - \mathbf{Q}^{\mathbf{v}}_{\dot{\mathbf{v}}} - \mathbf{Q}^{\mathbf{v}}_{\dot{\mathbf{u}}}\mathbf{H} \tag{3.44}$$

$$\begin{aligned}\hat{\mathbf{M}}_2 \equiv &(\mathbf{M}^{\mathbf{v}}\ddot{\mathbf{q}})_{\mathbf{u}}\mathbf{H} + (\mathbf{M}^{\mathbf{v}}\ddot{\mathbf{q}})_{\mathbf{v}} + \mathbf{M}^{\mathbf{vu}}\mathbf{L} + \mathbf{H}^{\mathrm{T}}\mathbf{R} \\ &+(\Phi^{\mathrm{T}}_{\mathbf{v}}\lambda)_{\mathbf{u}}\mathbf{H} + (\Phi^{\mathrm{T}}_{\mathbf{v}}\lambda)_{\mathbf{v}} - \mathbf{Q}^{\mathbf{v}}_{\mathbf{u}}\mathbf{H} - \mathbf{Q}^{\mathbf{v}}_{\mathbf{v}} - \mathbf{Q}^{\mathbf{v}}_{\dot{\mathbf{u}}}\mathbf{J}\end{aligned} \tag{3.45}$$

the integration Jacobian assumes the form

$$\Psi_{\ddot{\mathbf{v}}} = \hat{\mathbf{M}} + \gamma h\hat{\mathbf{M}}_1 + \beta h^2\hat{\mathbf{M}}_2 \tag{3.46}$$

The integration Jacobian in Eq. (3.46) is nonsingular for small enough step-sizes h, since the matrix $\hat{\mathbf{M}}$ is positive definite. The solution $\ddot{\mathbf{v}}$ is found via an iterative process of the form

$$\ddot{\mathbf{v}}^{(k+1)} = \ddot{\mathbf{v}}^{(k)} - \left(\Psi_{\ddot{\mathbf{v}}}(\ddot{\mathbf{v}}^{(0)})\right)^{-1}\Psi(\ddot{\mathbf{v}}^{(k)}) \tag{3.47}$$

In a numerical implementation, the rather complicated form of the integration Jacobian needs to be coded once. In other words, software should be provided to compute $\hat{\mathbf{M}}$, $\hat{\mathbf{M}}_1$, and $\hat{\mathbf{M}}_2$. This is a matter of book-keeping and efficient level 2 and 3 Basic Linear Algebra Subroutine (BLAS) operations. This process should, however, be supported by a library of derivatives that contains for each typical joint and force element derivatives with respect to generalized positions and velocities. This is a one-time effort, and in terms of constraint derivatives the problem is addressed elegantly by in fact providing the derivative information for several basic constraint functions. These constraints are $\Phi^{d1}(\mathbf{a}_i, \mathbf{a}_j)$, $\Phi^{d2}(\mathbf{a}_i, \mathbf{d}_{ij})$, $\Phi^{s}(\mathrm{P_i}, \mathrm{P_j})$, $\Phi^{ss}(\mathrm{P_i}, \mathrm{P_j}, \mathrm{C})$, along with other several absolute constrain functions (Haug, 1989). These are the building blocks, that in a Cartesian representation of a mechanical system, are assembled to generate derivative information for complex joint elements.

Computing force derivatives is more difficult, because the family of forces that are readily expressible in an analytic closed form is rather small. Derivative information can be easily obtainable for Translational-Spring-Damper-Actuator (TSDA), and Rotational-Spring-Damper-Actuator (RSDA) elements, but it is virtually impossible to generate for the interaction force between a bulldozer’s blade pushing a pile of gravel, for example. In the latter case, the solution is to either neglect the contribution of these derivatives in the quasi-Newton algorithm, or to compute them numerically.

The issue of obtaining derivative information is not the focus of this thesis. A comprehensive treatment of this topic can be found in the work of Serban (1998). Here, it is assumed that derivative information is available and organized in a library that is called during the numerical integration stage to compute, among other things the integration Jacobian of Eq. (3.43). It remains to assemble and use this information, according to the particular integration method being considered.

3.2.2 Runge-Kutta Methods

3.2.2.1 General Considerations on Runge-Kutta Methods

This Section presents basic concepts underlying the family of Runge-Kutta numerical integrators. There is a deep theory behind this class of numerical integrators, and the presentation here is by no means complete. A very thorough presentation of this very active area of Numerical Analysis has been given by Hairer, Nørsett, and Wanner (1993), and Hairer and Wanner (1996).

A Runge-Kutta integration formula can be represented using the so called Butcher's tableau of Table 1.

Table 1. Butcher's Tableau

c_1	a_{11}	a_{12}	…	…	a_{1s}
c_2	a_{11}	a_{11}	…	…	a_{2s}
…	…	…	…	…	…
c_s	a_{s1}	a_{s2}	…	…	a_{ss}
	b_1	b_2	…	…	b_s

The Runge-Kutta formula defined in Table 1 is an s-stage formula, defined by the coefficients a_{ij}, b_i, and c_j. If this integration formula is applied to numerically integrate the IVP

$$y' = f(x, y) \tag{3.48}$$

with $y(x_0) = y_0$, after one integration step the solution is given by

$$y_1 = y_0 + h\sum_{i=1}^{s} b_i k_i \tag{3.49}$$

where, with h being the integration step-size,

$$k_i = f(x_0 + c_i h, y_0 + h\sum_{j=1}^{s} a_{ij} k_j) \quad i = 1, \ldots, s \tag{3.50}$$

If $a_{ij} = 0$ for $j \geq i$, the formula is called explicit. When some of the values a_{ij}, for $j \geq i$, are non-zero, the formula is called implicit. If all a_{ij} are non-zero, the formula is called fully implicit. A particular class of implicit formulas is the so called family of diagonally implicit formulas (DIRK), where $a_{ij} = 0$ for $i < j$; i. e., when all the supra-diagonal elements are zero. In the case of DIRK formulas, significant gain in efficiency can be obtained if all the diagonal entries in the Butcher tableau are identical, yielding the so called singly diagonal implicit Runge-Kutta (SDIRK) integrators. In the present work, formulas considered for numerical integration of the SSODE of Multibody Dynamics belong to the SDIRK family. However, the theoretical considerations that are made here in conjunction with this class of integrators are directly applicable to the class of fully implicit Runge-Kutta integrators. For the dynamic analysis of the class of problems considered so far, the performance of the SDIRK family has been very good. More challenging applications might recommend taking the extra step of implementing and using the more complex class of fully implicit Runge-Kutta formulas. In this context, the RADAU family of collocation methods (Hairer and Wanner 1996), is generally considered to be the most promising direction to follow.

The basic idea behind the Runge-Kutta family of integration formulas is to choose the coefficients defining the formula; i. e., a_{ij}, b_i, and c_j, of Table 1 such that the Taylor expansion of the solution of the IVP in Eq. (3.48) is identical to the Taylor expansion of the numerical solution provided by Eq. (3.49) and (3.50). Taylor expansions are done with respect to the variable h, and the number of matching terms in the Taylor expansion determines the order of the method.

Among the conditions usually imposed on the coefficients are (Hairer, Nørsett, and Wanner, 1993),

$$\sum_{j=1}^{i} a_{ij} = c_i \tag{3.51}$$

$$\sum_{i=1}^{s} b_i = 1 \tag{3.52}$$

As the upper summation limit in Eq. (3.51) suggests, the theoretical framework is that of SDIRK formulas ($a_{ii} = \gamma$, $i = 1,\ldots,s$). For this family of implicit integrators, Eq. (3.50) assumes the form

$$k_i = f(x_0 + c_i h, y_0 + h\sum_{j=1}^{i} a_{ij} k_j)$$

By introducing the stage variables

$$z_i \equiv y_0 + h\sum_{j=1}^{i} a_{ij} k_j \tag{3.53}$$

k_i can be expressed as

$$k_i = f(x_0 + c_i h, z_i)$$

In the context of Multibody Dynamics, the stage variables k_i and z_i can be regarded as stage accelerations and velocities, respectively; or stage velocities and positions, respectively. From a physical standpoint, the stage variables are not the numerical solution at intermediate integration grid points $x_0 + c_i h$, but they are merely intermediate values that are used to provide the numerical solution at the end of one macro-step of the Runge-Kutta formula.

Before applying an SDIRK formula for numerical integration of the SSODE it is instructive to apply the same formula to integrate a generic second order set of ODE. This will serve in Section 3.2.2.2 as the model to be followed when integrating the independent equations of motion of Eq. (3.5). Consider for this the IVP

$$\ddot{v} = f(t, v, \dot{v}) \tag{3.54}$$

with $v(t_0) = v_0$ and $\dot{v}(t_0) = \dot{v}_0$. The second order ODE is formally reduced to a first order ODE

$$\frac{dy}{dt} = g(t, y), \qquad y \equiv \begin{bmatrix} v \\ \dot{v} \end{bmatrix} \qquad g(t, y) \equiv \begin{bmatrix} \dot{v} \\ f(t, v, \dot{v}) \end{bmatrix} \tag{3.55}$$

Then,

$$k_i = g(t_0 + c_i h, y_0 + h\sum_{j=1}^{i} a_{ij} k_j) \tag{3.56}$$

and the solution at the new time step “1” is given by

$$y_1 = y_0 + h\sum_{i=1}^{s} b_i k_i$$

Introducing the stage variables $z_i \equiv [v^{(i)}\ \dot{v}^{(i)}]^{\mathrm{T}}$ as

$$z_i = y_0 + h\sum_{j=1}^{i} a_{ij} k_j \tag{3.57}$$

the stage variables k_i are

$$k_i = g(t_0 + c_i h, z_i) = \begin{bmatrix} \dot{v}^{(i)} \\ f(t_0 + c_i h, v^{(i)}, \dot{v}^{(i)}) \end{bmatrix} = \begin{bmatrix} \dot{v}^{(i)} \\ \ddot{v}^{(i)} \end{bmatrix} \tag{3.58}$$

Writing Eq. (3.57) explicitly yields

$$\begin{bmatrix} v^{(i)} \\ \dot{v}^{(i)} \end{bmatrix} = \begin{bmatrix} v_0 \\ \dot{v}_0 \end{bmatrix} + h\sum_{j=1}^{i} a_{ij} \begin{bmatrix} \dot{v}^{(j)} \\ \ddot{v}^{(j)} \end{bmatrix}$$

or equivalently,

$$v^{(i)} = v_0 + h\sum_{j=1}^{i} a_{ij}\dot{v}^{(j)} \tag{3.59}$$

$$\dot{v}^{(i)} = \dot{v}_0 + h\sum_{j=1}^{i} a_{ij}\ddot{v}^{(j)} \tag{3.60}$$

If the stage values for $\dot{v}^{(j)}$ from Eq. (3.60) are substituted back into Eq. (3.59), the stage value $v^{(i)}$ can be expressed exclusively in terms of $\ddot{v}^{(j)}$,

$$v^{(i)} = v_0 + hc_i\dot{v}_0 + h^2\sum_{j=1}^{i} \mu_{ij}\ddot{v}^{(j)} \tag{3.61}$$

with

$$\mu_{ij} \equiv \sum_{k=j}^{i} a_{ik}a_{kj} \tag{3.62}$$

Since the function $f(\bullet)$ in Eq. (3.54) is generally non-linear, a system of non-linear equations must be solved at each stage of the formula to retrieve $\ddot{v}^{(i)}$. The strategy to find the solution $\ddot{v}^{(i)}$ is based on a quasi-Newton approach, and therefore derivative information must be provided to the non-linear solver.

This process of retrieving $\ddot{v}^{(i)}$, thus k_i, is the cornerstone of the integration algorithm, since the solution at the new grid-point is immediately obtained as a linear combination of these quantities, as indicated in Eq. (3.49).

At each stage, $\ddot{v}^{(i)}$ is obtained as follows:

(1). Provide an initial estimate for $\ddot{v}^{(i)}$

(2). Based on Eqs. (3.61) and (3.59), obtain stage variables $v^{(i)}$ and $\dot{v}^{(i)}$

(3). If the stopping criteria are satisfied, then the iterative process is stopped, and $k_i = [\dot{v}^{(i)}\ \ddot{v}^{(i)}]^{\mathrm{T}}$. Otherwise, using a quasi-Newton correction, correct the value of $\ddot{v}^{(i)}$ and go to step (2).

The stopping criteria in step (3) are based on the size of the norm of the error in satisfying Eq. (3.54), as well as on the size of the last correction in $\ddot{v}^{(i)}$.

Central to the process of obtaining the stage value $\ddot{v}^{(i)}$ is the correction step. The non-linear system to be solved is

$$\ddot{v}^{(i)} - f(t_0 + c_i h, v^{(i)}(\ddot{v}^{(i)}), \dot{v}^{(i)}(\ddot{v}^{(i)})) = 0 \tag{3.63}$$

The quantities $v^{(i)}$ and $\dot{v}^{(i)}$ in Eq. (3.63) depend on $\ddot{v}^{(i)}$, through the integration formulas of Eqs. (3.61) and (3.59), respectively. Consequently, the Jacobian of the non-linear system is

$$J = 1 - f_v \cdot v^{(i)}{}_{\ddot{v}^{(i)}} - f_{\dot{v}} \cdot \dot{v}^{(i)}{}_{\ddot{v}^{(i)}} \tag{3.64}$$

The derivatives $v^{(i)}_{\ddot{v}^{(i)}}$ and $\dot{v}^{(i)}_{\ddot{v}^{(i)}}$ are readily available by taking the partial derivative of Eqs. (3.61) and (3.59) with respect to $\ddot{v}^{(i)}$. On the other hand, the derivatives f_v and $f_{\dot{v}}$ are problem dependent and must be provided.

Usually, in a quasi-Newton approach for the solution of this non-linear system, the quantities f_v and $f_{\dot{v}}$ are evaluated at the beginning of a macro-step, and kept constant during at least that macro-step, thus saving a substantial amount of CPU effort by circumventing the costly process of derivative evaluation. CPU savings associated with the SDIRK method, as noted at the beginning of this Section, are due to the expression assumed by the derivatives $v^{(i)}_{\ddot{v}^{(i)}}$ and $\dot{v}^{(i)}_{\ddot{v}^{(i)}}$. For all stages, these derivatives are identical. In particular, for an SDIRK method with $a_{11} = a_{22} = \ldots = a_{ss} = \gamma$,

$$v^{(i)}_{\ddot{v}^{(i)}} = \gamma^2 h^2 \tag{3.65}$$

$$\dot{v}^{(i)}_{\ddot{v}^{(i)}} = \gamma h \tag{3.66}$$

Therefore, for an SDIRK integration formula, the Jacobian used at each stage to solve for $\ddot{v}^{(i)}$ is the same and assumes the form

$$J = 1 - \gamma h \cdot f_{\dot{v}} - \gamma^2 h^2 \cdot f_v \tag{3.67}$$

The foregoing is the framework in which numerical integration of the SSODE is going to be carried out. Appropriate modifications of the formulas in this Section accommodate the situation in which, instead of scalar values, vector quantities are dealt with.

<u>3.2.2.2 Runge-Kutta State-Space Based Implicit Integration</u>

As in the case of multi-step methods, discretization using an SDIRK formula is applied to the independent equations of motion, rather than to the implicit form second order SSODE in Eq. (3.15). In the light of considerations made in Section 3.2.1.2, applying a different numerical ODE integrator to find the solution of the state space ODE is straight forward. Instead of using an implicit multi-step formula, the second order ODE is integrated using the methodology introduced in the previous Section.

In order to extend the presentation from implicit multi-step methods to SDIRK formulas, Eqs. (3.61) and (3.59) are reformulated in vector notation as

$$\mathbf{v}^{(i)} = \tilde{\mathbf{v}}^{(i)} + h^2\gamma^2\ddot{\mathbf{v}}^{(i)} \tag{3.68}$$

and

$$\dot{\mathbf{v}}^{(i)} = \tilde{\dot{\mathbf{v}}}^{(i)} + \gamma h\ddot{\mathbf{v}}^{(i)} \tag{3.69}$$

respectively. With μ_{ij} defined in Eq. (3.62), the following notations have been used:

$$\tilde{\mathbf{v}}^{(i)} = \mathbf{v}_0 + hc_i\dot{\mathbf{v}}_0 + \sum_{j=1}^{i-1}\mu_{ij}\ddot{\mathbf{v}}^{(j)} \tag{3.70}$$

$$\tilde{\dot{\mathbf{v}}}^{(i)} = \dot{\mathbf{v}}_0 + h\sum_{j=1}^{i-1}a_{ij}\ddot{\mathbf{v}}^{(j)} \tag{3.71}$$

Comparing Eqs. (3.68) and (3.69), to Eqs. (3.23) and (3.21), it results that each stage of one macro-step is identical to one step of a generic multi-step method, as introduced in the framework of Sections 3.2.1.1 and 3.2.1.2. Consequently, the integration Jacobian that is required during each stage of an SDIRK method is obtained as in Eq. (3.43), with the only difference that $\beta = \gamma^2$. This might be the case even for multi-step methods, provided the same integration formulas are used to integrate independent accelerations to obtain independent velocities, and then independent velocities to obtain independent positions. In other words, the integration Jacobian required for SDIRK based implicit integration of the state-space ODE is

$$\begin{aligned}\Psi_{\ddot{\mathbf{v}}} = \mathbf{M}^{\mathbf{vv}} + \mathbf{M}^{\mathbf{vu}}\mathbf{H} + \mathbf{H}^{\mathrm{T}}(\mathbf{M}^{\mathbf{uv}} + \mathbf{M}^{\mathbf{uu}}\mathbf{H}) + \gamma h(\mathbf{M}^{\mathbf{vu}}\mathbf{N} + \mathbf{H}^{\mathrm{T}}\mathbf{S} - \mathbf{Q}^{\mathbf{v}}_{\dot{\mathbf{v}}} - \mathbf{Q}^{\mathbf{v}}_{\dot{\mathbf{u}}}\mathbf{H}) \\ +\gamma^2 h^2[(\mathbf{M}^{\mathbf{v}}\ddot{\mathbf{q}})_{\mathbf{u}}\mathbf{H} + (\mathbf{M}^{\mathbf{v}}\ddot{\mathbf{q}})_{\mathbf{v}} + \mathbf{M}^{\mathbf{vu}}\mathbf{L} + \mathbf{H}^{\mathrm{T}}\mathbf{R} + (\Phi^{\mathrm{T}}_{\mathbf{v}}\lambda)_{\mathbf{u}}\mathbf{H} + (\Phi^{\mathrm{T}}_{\mathbf{v}}\lambda)_{\mathbf{v}} \\ -\mathbf{Q}^{\mathbf{v}}_{\mathbf{u}}\mathbf{H} - \mathbf{Q}^{\mathbf{v}}_{\mathbf{v}} - \mathbf{Q}^{\mathbf{v}}_{\dot{\mathbf{u}}}\mathbf{J}]\end{aligned} \tag{3.72}$$

or, using the matrix notation introduced in Section 3.2.1.2,

$$\Psi_{\ddot{\mathbf{v}}} = \hat{\mathbf{M}} + \gamma h\hat{\mathbf{M}}_1 + \gamma^2 h^2 \hat{\mathbf{M}}_2 \tag{3.73}$$

where γ is the diagonal element in Butcher's tableau for the SDIRK formula considered. Technically, these considerations complete the derivation of SDIRK based state-space implicit integration.

3.3 Descriptor Form Method

This Section introduces a new method for the implicit integration of the DAE of Multibody Dynamics. The results presented here are based on the work of Haug, Negrut, and Engstler (1998); Haug, Negrut, and Iancu (1997b); and Iancu, Haug, and Negrut (1997). In Section 3.3.1 is presented the case in which the method employs an implicit multi-step method to discretize the index 1 DAE of Eqs. (3.5), (3.6), and (3.9). In Section 3.3.2 is detailed the case when an SDIRK formula is used for this purpose. Although the discretization is done at the index 1 DAE level, the method is truly a state-

space method, since after each integration step, dependent variables at position and velocity levels are recovered using kinematic constraint equations.
Compared to the State-Space Method, this approach results in system of non-linear algebraic equations of significantly larger dimension. Thus, the discretization of the index 1 DAE results in a set of $n+m$ non-linear equations that is solved at each step/stage for generalized accelerations and Lagrange multipliers.

Since implicit multi-step methods and the family of SDIRK formulas were introduced in Sections 3.2.1.1 and 3.2.2.1, respectively, the focus here is primarily on integration Jacobian computation. Generic implicit integration formulas

$$\mathbf{v}_{n+1} = \tilde{\mathbf{v}}_{n+1} + \beta h^2 \ddot{\mathbf{v}}_{n+1} \tag{3.74}$$

$$\dot{\mathbf{v}}_{n+1} = \tilde{\dot{\mathbf{v}}}_{n+1} + \gamma h \ddot{\mathbf{v}}_{n+1} \tag{3.75}$$

are used to present the new method, where $\tilde{\mathbf{v}}_{n+1}$ and $\tilde{\dot{\mathbf{v}}}_{n+1}$ contain only past information. To simplify the presentation, it is assumed that a reordering of the vector of generalized coordinates has been done, such that the first m entries are dependent generalized coordinates, while the last $ndof$ entries are independent generalized coordinates.

Using the integration formulas of Eqs. (3.74) and (3.75), the equations of motion and the acceleration constraint equation are discretized to obtain

$$\mathbf{\Psi} \equiv \begin{bmatrix} \mathbf{M}(\mathbf{q}_{n+1})\ddot{\mathbf{q}}_{n+1} + \mathbf{\Phi}_{\mathbf{q}}^{\mathrm{T}}(\mathbf{q}_{n+1})\lambda_{n+1} - \mathbf{Q}^{\mathrm{A}}(\mathbf{q}_{n+1}, \dot{\mathbf{q}}_{n+1}) \\ \mathbf{\Phi}_{\mathbf{q}}(\mathbf{q}_{n+1})\ddot{\mathbf{q}}_{n+1} - \tau(\mathbf{q}_{n+1}, \dot{\mathbf{q}}_{n+1}) \end{bmatrix} = \mathbf{0} \tag{3.76}$$

This system is solved at each integration step of a multi-step method, or at each stage of an SDIRK method, for $\ddot{\mathbf{q}}_{n+1}$ and λ_{n+1}. Once these values are available, the integration formulas are used to obtain the independent positions and velocities. Dependent variables (positions and velocities) are recovered via the kinematic constraint equations at the position and velocity levels.

The integration proceeds as follows:

(1). Provide a starting value for $\ddot{\mathbf{q}}_{n+1}$ and λ_{n+1}

(2). Use Eqs. (3.74) and (3.75) to obtain independent positions and velocities

(3). If stopping criteria are met, stop. Otherwise, in a quasi-Newton framework, correct the values of the generalized accelerations and Lagrange multipliers, and go to Step (2).

For an SDIRK method, these three steps are repeated at each stage of the method. The stopping criteria in the last step should be designed based on the norm of the residual $\mathbf{\Psi}^{(k)}$ (evaluated at the end of iteration k), and the norm of the last correction in generalized acceleration and Lagrange multipliers.

The remainder of this Section focuses on how to compute the Jacobian of the discretization system of non-linear algebraic equations of Eq. (3.76). Subscripts will be suppressed to keep the presentation simple.

Derivatives of independent positions and velocities with respect to independent accelerations are easily obtained by differentiating Eqs. (3.74) and (3.75), respectively, yielding

$$\mathbf{v}_{\ddot{\mathbf{v}}} = \beta h^2 \mathbf{I} \tag{3.77}$$

$$\dot{\mathbf{v}}_{\ddot{\mathbf{v}}} = \gamma h \mathbf{I} \tag{3.78}$$

where $\mathbf{I}$ is the identity matrix of dimension $ndof$.

If a Boolean matrix is used to select the independent generalized coordinates, as in

$$\mathbf{v} = \mathbf{P}\mathbf{q} \tag{3.79}$$

the simplifying assumption introduced earlier concerning the ordering of variables in the vector of generalized coordinates $\mathbf{q}$, implies that $\mathbf{P} = [\mathbf{0}\ \mathbf{I}]$, with $\mathbf{P} \in \Re^{ndof \times m}$, and $\mathbf{I}$ being the identity matrix of dimension $ndof$. Recalling that $\mathbf{P}$ is constant, $\ddot{\mathbf{v}} = \mathbf{P}\ddot{\mathbf{q}}$, and

$$\ddot{\mathbf{v}}_{\ddot{\mathbf{q}}} = \mathbf{P} \tag{3.80}$$

Based on the results in Eqs. (3.77) and (3.32), the derivative of the generalized coordinate vector $\mathbf{q}$ with respect to independent accelerations $\ddot{\mathbf{v}}$ is

$$\mathbf{q}_{\ddot{\mathbf{v}}} = \beta h^2 \begin{bmatrix} \mathbf{H} \\ \mathbf{I} \end{bmatrix} \tag{3.81}$$

Using the chain rule of differentiation yields

$$\mathbf{q}_{\ddot{\mathbf{q}}} = \mathbf{q}_{\ddot{\mathbf{v}}} \cdot \ddot{\mathbf{v}}_{\ddot{\mathbf{q}}} = \beta h^2 \begin{bmatrix} \mathbf{HP} \\ \mathbf{P} \end{bmatrix} \equiv \beta h^2 \hat{\mathbf{H}} \tag{3.82}$$

In order to compute the derivative $\dot{\mathbf{q}}_{\ddot{\mathbf{v}}}$, first recall the definition of the matrix $\mathbf{J}$ of Eq. (3.38) and the expression for $\dot{\mathbf{u}}_{\ddot{\mathbf{v}}}$ of Eq. (3.34) in Section 3.2.1.2. Using these results, the desired derivative is

$$\dot{\mathbf{q}}_{\ddot{\mathbf{v}}} = \gamma h \begin{bmatrix} \mathbf{H} \\ \mathbf{I} \end{bmatrix} + \beta h^2 \begin{bmatrix} \mathbf{J} \\ \mathbf{0} \end{bmatrix} \tag{3.83}$$

Applying the chain rule of differentiation and using expression of $\dot{\mathbf{q}}_{\ddot{\mathbf{v}}}$ and $\ddot{\mathbf{v}}_{\ddot{\mathbf{q}}}$,

$$\dot{\mathbf{q}}_{\ddot{\mathbf{q}}} = \dot{\mathbf{q}}_{\ddot{\mathbf{v}}} \cdot \ddot{\mathbf{v}}_{\ddot{\mathbf{q}}} = \gamma h \begin{bmatrix} \mathbf{HP} \\ \mathbf{P} \end{bmatrix} + \beta h^2 \begin{bmatrix} \mathbf{JP} \\ \mathbf{0} \end{bmatrix} \equiv \gamma h \hat{\mathbf{H}} + \beta h^2 \hat{\mathbf{J}}$$

Finally, using the chain rule of differentiation and expressions for $\mathbf{q}_{\ddot{\mathbf{q}}}$ and $\dot{\mathbf{q}}_{\ddot{\mathbf{q}}}$, the integration Jacobian required by the quasi-Newton algorithm is

$$\Psi_{\ddot{\mathbf{q}}} = \begin{bmatrix} \mathbf{M} + \beta h^2 \left[(\mathbf{M}\ddot{\mathbf{q}})_{\mathbf{q}} + \left(\Phi_{\mathbf{q}}^T \lambda_i \right)_{\mathbf{q}} - \mathbf{Q}_{\mathbf{q}}^A \right] \hat{\mathbf{H}} - \mathbf{Q}_{\dot{\mathbf{q}}}^A \left[\gamma h \hat{\mathbf{H}} + \beta h^2 \hat{\mathbf{J}} \right] \\ \Phi_{\mathbf{q}} + \beta h^2 \left[(\Phi_{\mathbf{q}} \ddot{\mathbf{q}})_{\mathbf{q}} - \tau_{\mathbf{q}} \right] \hat{\mathbf{H}} - \tau_{\dot{\mathbf{q}}} \left[\gamma h \hat{\mathbf{H}} + \beta h^2 \hat{\mathbf{J}} \right] \end{bmatrix}$$

$$\Psi_{\lambda} = \begin{bmatrix} \Phi_{\mathbf{q}}^{T} \\ \mathbf{0} \end{bmatrix}$$

The iterative process is carried out as follows:

$$\begin{bmatrix} \mathbf{M} + \beta h^2 \left\{ \left[\left(\mathbf{M}\ddot{\mathbf{q}} \right)_{\mathbf{q}} + \left(\Phi_{\mathbf{q}}^{T}\lambda \right)_{\mathbf{q}} - \mathbf{Q}_{\mathbf{q}}^{A} \right] \hat{\mathbf{H}} - \mathbf{Q}_{\dot{\mathbf{q}}}^{A} \hat{\mathbf{J}} \right\} - h\gamma \mathbf{Q}_{\dot{\mathbf{q}}}^{A} \hat{\mathbf{H}} & \Phi_{\mathbf{q}}^{T} \\ \Phi_{\mathbf{q}} + \beta h^2 \left\{ \left[\left(\Phi_{\mathbf{q}}\ddot{\mathbf{q}} \right)_{\mathbf{q}} - \tau_{\mathbf{q}} \right] \hat{\mathbf{H}} - \tau_{\dot{\mathbf{q}}} \hat{\mathbf{J}} \right\} - h\gamma \tau_{\dot{\mathbf{q}}} \hat{\mathbf{H}} & \mathbf{0} \end{bmatrix}^{(j-1)} \begin{bmatrix} \Delta\ddot{\mathbf{q}} \\ \Delta\lambda \end{bmatrix}^{(j)} = -\Psi^{(j-1)}$$

$$\begin{bmatrix} \ddot{\mathbf{q}} \\ \lambda \end{bmatrix}^{(j)} = \begin{bmatrix} \ddot{\mathbf{q}} \\ \lambda \end{bmatrix}^{(j-1)} + \begin{bmatrix} \Delta\ddot{\mathbf{q}} \\ \Delta\lambda \end{bmatrix}^{(j)}$$

where j is the iteration counter. At each iteration, integration formulas determine $\mathbf{v}$ and $\dot{\mathbf{v}}$, and kinematic constraint equations at the position and velocity levels are solved for $\mathbf{u}$ and $\dot{\mathbf{u}}$. Iteration is continued until stopping criteria are met.

One attractive feature of this method is that, when compared to the previously introduced State-Space Method, computation of the integration Jacobian is by far less CPU intensive. This is primarily due to the lack of expensive matrix-matrix multiplications in integration Jacobian computation for the State-Space Method. Likewise, this method does not require the extra effort of computing $\ddot{\mathbf{u}}$ and λ at each iteration, as does the State-Space Reduction Method, since these quantities are now being solved for.

The major drawback of this method is the rather large dimension of the non-linear system that must be solved at each step/stage. For a High Mobility Multipurpose Wheeled Vehicle (HMMWV) model comprising 14 bodies (Serban, Negrut, and Haug, 1998) that serves as a test problem later in this document, the dimension of this non-linear system becomes 178. The same model, when analyzed using the State-Space Method, results in a non-linear system of dimension 18. To some extent, this dimensional problem for the Descriptor Form Method is alleviated if sparse solvers are used to carry out the iterative process. For the HMMWV model, the fill-in ratio of the integration Jacobian is in fact less than 15%.; i. e., over 85% of the entries are zero.

3.3.1 Multi-step Methods

In the case of a multi-step formula used in the framework of the Descriptor Form Method, generic formulas of Eqs. (3.74) and (3.75) are replaced by actual multi-step formulas.

As in Section 3.2.1.1, the actual multi-step integration formulas are used to obtain independent positions and velocities as

$$\mathbf{v}_{n+1} = \tilde{\mathbf{v}}_{n+1} + \beta h^2 \ddot{\mathbf{v}}_{n+1}$$

$$\dot{\mathbf{v}}_{n+1} = \tilde{\dot{\mathbf{v}}}_{n+1} + \gamma h \ddot{\mathbf{v}}_{n+1}$$

with $\tilde{\mathbf{v}}_{n+1}$ and $\tilde{\dot{\mathbf{v}}}_{n+1}$ containing formula-specific past information. Consequently, nothing is changed in the approach presented in the previous Section. The integration Jacobian assumes exactly the same form, with the coefficients γ and β provided by the multi-step integration formula being considered. With this, the 3 steps presented for the Descriptor Form Method are immediately applicable.

3.3.2 Runge-Kutta Methods

When an SDIRK type formula is used to express independent positions and velocities in terms of independent accelerations, this dependency was shown in Section 3.2.2.1 to assume the form

$$\mathbf{v}^{(i)} = \tilde{\mathbf{v}}^{(i)} + h^2 \gamma^2 \ddot{\mathbf{v}}^{(i)}$$

$$\dot{\mathbf{v}}^{(i)} = \tilde{\dot{\mathbf{v}}}^{(i)} + \gamma h \ddot{\mathbf{v}}^{(i)}$$

With μ_{ij} defined in Eq. (3.62),

$$\tilde{\mathbf{v}}^{(i)} = \mathbf{v}_0 + hc_i\dot{\mathbf{v}}_0 + \sum_{j=1}^{i-1}\mu_{ij}\ddot{\mathbf{v}}^{(j)}$$

$$\tilde{\dot{\mathbf{v}}}^{(i)} = \dot{\mathbf{v}}_0 + h\sum_{j=1}^{i-1}a_{ij}\ddot{\mathbf{v}}^{(j)}$$

where a_{ij} are the a coefficients in Butcher's tableau, and γ is the diagonal element of the formula. Superscript i refers to the stage of the formula.

The integration Jacobian is then easily obtained by replacing the coefficient β with γ^2. As noted before, an attractive feature of SDIRK formulas is that the integration Jacobian needs to be evaluated only once at the beginning of the macro-step. This Jacobian is then used during each stage to compute $\ddot{\mathbf{q}}^{(i)}$ and $\lambda^{(i)}$.

Once these quantities are available for each stage, independent positions and velocities are obtained at the new time step as

$$\mathbf{v}_1 = \mathbf{v}_0 + h\dot{\mathbf{v}}_0 + h^2\sum_{i=1}^{s}\varphi_i\ddot{\mathbf{v}}^{(i)}$$

$$\dot{\mathbf{v}}_1 = \dot{\mathbf{v}}_0 + h\sum_{i=1}^{s}b_i\ddot{\mathbf{v}}^{(i)}$$

where $\varphi_i = \sum_{j=i}^{s}b_j a_{ji}$. At the end of the macro-step, it remains to recover dependent positions and velocities, based on kinematic constraint equations at position and velocity levels.

3.4 First Order Reduction Method

The State Space and First Order Reduction Methods share the same DAE-to-ODE reduction strategy. The starting point for both methods is the independent equations of motion

$$\mathbf{M}^{\mathbf{vv}}(\mathbf{u},\mathbf{v})\ddot{\mathbf{v}} + \mathbf{M}^{\mathbf{vu}}(\mathbf{u},\mathbf{v})\ddot{\mathbf{u}} + \Phi_{\mathbf{v}}^{\mathbf{T}}(\mathbf{u},\mathbf{v})\lambda = \mathbf{Q}^{\mathbf{v}}(\mathbf{u},\mathbf{v},\dot{\mathbf{u}},\dot{\mathbf{v}})$$

In the State Space Reduction Method, this second order differential equations is discretized, and the resulting non-linear equations are solved for the independent accelerations $\ddot{\mathbf{v}}$.

The idea behind the First Order Reduction Method is to go one step further, and transform this set of second-order ordinary differential equations in an equivalent set of first order differential equations. If the dimension of the second order ODE dealt with in the State Space Method is equal to the number of degrees of freedom $ndof$ of the mechanism, the dimension of the resulting first order ODE in the First Order Reduction Method will be $2 \times ndof$.

The motivation for the extra step for first order reduction resides in the existence of very good software for the numerical solution of stiff first order IVP. The goal is to adapt some of these codes, to accommodate numerical integration of the first order ODE obtained from the SSODE of Multibody Dynamics after order reduction.

The implication of the additional order reduction step for the First Order Method is substantial. If a standard ODE integration code is applied, corrections during the iterative solution of the discretized non-linear equations, are done in independent positions and velocities. During each iteration, after recovering dependent positions and velocities, generalized accelerations and Lagrange multipliers must be computed. This sets the First Order Reduction Method on a different path than the one followed by the State Space Method presented in Section 3.2. The computation of generalized accelerations, as the solution of a set of linear equations, brings into the picture the challenge of efficient linear algebra. Sparsity and topology-based matrix manipulations are at the heart of two methods proposed in the Sections 3.4.2 and 3.4.3 for efficiently accommodating the linear algebra demands of the First Order Reduction Method.

3.4.1 Theoretical Considerations in First Order Reduction

When the First Order Reduction Method is compared to the State Space Reduction Method, there are no qualitative differences between the DAE-to-ODE reduction stages, and the new method requires only an extra two-to-one ODE order reduction. The derivative information for the method is the standard required by any well established code for numerical solution of first order systems of ordinary differential equations; how to compute derivative information is discussed in Section 3.4.1.2. Relevant aspects of the quasi-Newton algorithm for solution of the discretization non-linear algebraic equations are discussed in Section 3.4.1.3. Section 3.4.1.4 contains remarks on the First Order Reduction Method that will bridge in a unitary framework the issues of numerical ODE integration, and method characteristic linear algebra.

3.4.1.1 First Order Reduction of SSODE

By replacing all dependent variables in the independent equations of motion

$$\mathbf{M}^{\mathbf{vv}}(\mathbf{u},\mathbf{v})\ddot{\mathbf{v}} + \mathbf{M}^{\mathbf{vu}}(\mathbf{u},\mathbf{v})\ddot{\mathbf{u}} + \Phi_{\mathbf{v}}^{\mathbf{T}}(\mathbf{u},\mathbf{v})\lambda = \mathbf{Q}^{\mathbf{v}}(\mathbf{u},\mathbf{v},\dot{\mathbf{u}},\dot{\mathbf{v}})$$

the second order set of State-Space differential equations was shown in Section 3.1 to assume the form

$$\hat{\mathbf{M}}\ddot{\mathbf{v}} = \hat{\mathbf{Q}} \tag{3.84}$$

with $\hat{\mathbf{M}}$ a positive definite matrix. Expressions for $\hat{\mathbf{M}}$ and $\hat{\mathbf{Q}}$ are provided in Eqs. (3.16) and (3.17).

Since $\hat{\mathbf{M}}$ is positive definite, the implicit form of the second order differential equations in Eq. (3.84) can be expressed as a set of ordinary differential equations of the form

$$\ddot{\mathbf{v}} = \mathbf{f}(t, \mathbf{v}, \dot{\mathbf{v}}) \tag{3.85}$$

where $\mathbf{f} \equiv \hat{\mathbf{M}}^{-1}\hat{\mathbf{Q}}$. The second order ODE in Eq. (3.85) is further reduced to a first order system. Denoting $\mathbf{w} \equiv [\mathbf{v}^{\mathrm{T}}\ \dot{\mathbf{v}}^{\mathrm{T}}]^{\mathrm{T}}$, the first order ODE assumes the form

$$\dot{\mathbf{w}} = \mathbf{g}(t, \mathbf{w}) \tag{3.86}$$

with

$$\mathbf{g}(t, \mathbf{w}) = \begin{bmatrix} \dot{\mathbf{v}} \\ \mathbf{f}(t, \mathbf{v}, \dot{\mathbf{v}}) \end{bmatrix}$$

First order implicit numerical integration requires derivative information for the solution of the discretized non-linear algebraic equations. Consequently, the integration Jacobian $\mathbf{G} \equiv \mathbf{g}_{\mathbf{w}}$ needs to be provided. For the particular form of the first order ODE in Eq. (3.86), the integration Jacobian assumes the form

$$\mathbf{G} \equiv \begin{bmatrix} \mathbf{0} & \mathbf{I} \\ \mathbf{f}_{\mathbf{v}} & \mathbf{f}_{\dot{\mathbf{v}}} \end{bmatrix} \tag{3.87}$$

where $\mathbf{I}$ is the identity matrix of dimension $ndof$.

To compute $\mathbf{G}$, the derivatives of the right side of the differential equation of Eq. (3.85) must be provided. In other words, $\mathbf{J}_1 \equiv \ddot{\mathbf{v}}_{\mathbf{v}}$ and $\mathbf{J}_2 \equiv \ddot{\mathbf{v}}_{\dot{\mathbf{v}}}$ are required. The next Section details the process of computing $\mathbf{J}_1$ and $\mathbf{J}_2$.

3.4.1.2 Computing the Derivative Information

Taking the partial derivative of the independent set of equations of motion with respect to independent positions yields

$$\begin{aligned}&\mathbf{M}^{\mathbf{vv}}\mathbf{J}_{\mathbf{1}}+\left(\mathbf{M}^{\mathbf{vv}}\ddot{\mathbf{v}}\right)_{\mathbf{v}}+\left(\mathbf{M}^{\mathbf{vv}}\ddot{\mathbf{v}}\right)_{\mathbf{u}}\mathbf{u}_{\mathbf{v}}+\mathbf{M}^{\mathbf{vv}}\ddot{\mathbf{u}}_{\mathbf{v}}+\left(\mathbf{M}^{\mathbf{vu}}\ddot{\mathbf{u}}\right)_{\mathbf{v}}+\left(\mathbf{M}^{\mathbf{vu}}\ddot{\mathbf{u}}\right)_{\mathbf{u}}\mathbf{u}_{\mathbf{v}}\\&+\Phi_{\mathbf{v}}^{\mathrm{T}}\lambda_{\mathbf{v}}+\left(\Phi_{\mathbf{v}}^{\mathrm{T}}\lambda\right)_{\mathbf{v}}+\left(\Phi_{\mathbf{v}}^{\mathrm{T}}\lambda\right)_{\mathbf{u}}\mathbf{u}_{\mathbf{v}}=\mathbf{Q}_{\mathbf{v}}^{\mathbf{v}}+\mathbf{Q}_{\mathbf{u}}^{\mathbf{v}}\mathbf{u}_{\mathbf{v}}+\mathbf{Q}_{\dot{\mathbf{u}}}^{\mathbf{v}}\dot{\mathbf{u}}_{\mathbf{v}}\end{aligned} \tag{3.88}$$

In order to compute $\mathbf{J}_{\mathbf{1}}$, the quantities $\mathbf{u}_{\mathbf{v}}$, $\dot{\mathbf{u}}_{\mathbf{v}}$, $\ddot{\mathbf{u}}_{\mathbf{v}}$, and $\lambda_{\mathbf{v}}$ must be obtained. Taking the derivative of the position kinematic constraint equation with respect to independent positions and applying the chain rule of differentiation yields

$$\Phi_{\mathbf{u}}\mathbf{u}_{\mathbf{v}}+\Phi_{\mathbf{v}}=\mathbf{0}$$

Using the definition of the matrix $\mathbf{H}$ in Eq. (3.12),

$$\mathbf{u}_{\mathbf{v}}=\mathbf{H} \tag{3.89}$$

Taking the derivative of the velocity kinematic constraint equation in Eq. (3.8) with respect to independent positions yields

$$\left(\Phi_{\mathbf{u}}\dot{\mathbf{u}}\right)_{\mathbf{v}}+\left(\Phi_{\mathbf{u}}\dot{\mathbf{u}}\right)_{\mathbf{u}}\mathbf{u}_{\mathbf{v}}+\Phi_{\mathbf{u}}\dot{\mathbf{u}}_{\mathbf{v}}+\left(\Phi_{\mathbf{v}}\dot{\mathbf{v}}\right)_{\mathbf{v}}+\left(\Phi_{\mathbf{v}}\dot{\mathbf{v}}\right)_{\mathbf{u}}\mathbf{u}_{\mathbf{v}}=\mathbf{0}$$

Then,

$$\Phi_{\mathbf{u}}\dot{\mathbf{u}}_{\mathbf{v}}=-\left[\left(\Phi_{\mathbf{q}}\dot{\mathbf{q}}\right)_{\mathbf{u}}+\left(\Phi_{\mathbf{q}}\dot{\mathbf{q}}\right)_{\mathbf{u}}\mathbf{H}\right]$$

Using the definition of the matrix $\mathbf{J}$ in Eq. (3.38), the derivative of dependent velocities with respect to independent positions becomes

$$\dot{\mathbf{u}}_{\mathbf{v}}=\mathbf{J} \tag{3.90}$$

In order to compute the derivative of dependent accelerations with respect to independent positions, the acceleration kinematic constraint equation of Eq. (3.9) is differentiated to yield

$$\Phi_{\mathbf{v}}\mathbf{J}_{1}+\left(\Phi_{\mathbf{q}}\ddot{\mathbf{q}}\right)_{\mathbf{v}}+\left(\Phi_{\mathbf{q}}\ddot{\mathbf{q}}\right)_{\mathbf{u}}\mathbf{H}+\Phi_{\mathbf{u}}\ddot{\mathbf{u}}_{\mathbf{v}}=\tau_{\mathbf{v}}+\tau_{\mathbf{u}}\mathbf{H}+\tau_{\dot{\mathbf{u}}}\mathbf{J}$$

Using the definition of the matrix $\mathbf{L}$ in Eq. (3.39), the derivative $\ddot{\mathbf{u}}_{\mathbf{v}}$ is

$$\ddot{\mathbf{u}}_{\mathbf{v}} = \mathbf{H}\mathbf{J}_1 + \mathbf{L} \tag{3.91}$$

Finally, in order to compute the derivative of Lagrange multipliers with respect to independent positions, the dependent equations of motions are differentiated to obtain

$$\begin{aligned}&\mathbf{M}^{\mathbf{uv}}\mathbf{J}_1 + \left(\mathbf{M}^{\mathbf{uv}}\ddot{\mathbf{v}}\right)_{\mathbf{u}}\mathbf{H} + \left(\mathbf{M}^{\mathbf{uv}}\ddot{\mathbf{v}}\right)_{\mathbf{v}} + \mathbf{M}^{\mathbf{uu}}\left(\mathbf{H}\mathbf{J}_1 + \mathbf{L}\right) + \left(\mathbf{M}^{\mathbf{uu}}\ddot{\mathbf{u}}\right)_{\mathbf{v}}\\ &+\left(\mathbf{M}^{\mathbf{uu}}\ddot{\mathbf{u}}\right)_{\mathbf{u}}\mathbf{H} + \Phi_{\mathbf{u}}^{\mathrm{T}}\lambda_{\mathbf{v}} + \left(\Phi_{\mathbf{u}}^{\mathrm{T}}\lambda\right)_{\mathbf{v}} + \left(\Phi_{\mathbf{u}}^{\mathrm{T}}\lambda\right)_{\mathbf{u}}\mathbf{H} = \mathbf{0}\end{aligned}$$

Using the definition of the matrix $\mathbf{R}$ in Eq. (3.41), the derivative of Lagrange multipliers with respect to independent positions is obtained as

$$\lambda_{\mathbf{v}} = -\left(\Phi_{\mathbf{u}}^{\mathrm{T}}\right)^{-1}\left[\mathbf{R} + \left(\mathbf{M}^{\mathbf{uv}} + \mathbf{M}^{\mathbf{uu}}\mathbf{H}\right)\mathbf{J}_{\mathbf{1}}\right] \tag{3.92}$$

Once expressions for the required derivatives are available, results in Eqs. (3.89) through (3.92) are substituted into Eq. (3.88) to obtain the matrix $\mathbf{J}_1$ as the solution of the multiple right side linear equation

$$\begin{aligned}\hat{\mathbf{M}}\mathbf{J}_1 = \mathbf{Q}_{\mathbf{v}}^{\mathbf{v}} + \mathbf{Q}_{\mathbf{u}}^{\mathbf{v}}\mathbf{H} + \mathbf{Q}_{\dot{\mathbf{u}}}^{\mathbf{v}}\mathbf{J} - \Big[&\mathbf{M}^{\mathbf{vu}}\mathbf{L} + \mathbf{H}^{\mathbf{T}}\mathbf{R} + \left(\Phi_{\mathbf{v}}^{\mathrm{T}}\lambda\right)_{\mathbf{u}}\mathbf{H} + \left(\Phi_{\mathbf{v}}^{\mathrm{T}}\lambda\right)_{\mathbf{v}}\\ &\left(\mathbf{M}^{\mathbf{vv}}\ddot{\mathbf{v}} + \mathbf{M}^{\mathbf{vu}}\ddot{\mathbf{u}}\right)_{\mathbf{v}} + \left(\mathbf{M}^{\mathbf{vv}}\ddot{\mathbf{v}} + \mathbf{M}^{\mathbf{vu}}\ddot{\mathbf{u}}\right)_{\mathbf{u}}\mathbf{H}\Big]\end{aligned} \tag{3.93}$$

where $\hat{\mathbf{M}}$ is defined in Eq. (3.16).

The same steps are taken to compute the derivative $\mathbf{J}_2$ of independent accelerations with respect to independent velocities. Differentiating the independent equations of motion with respect to independent velocities, and using the chain rule of differentiation, yields

$$\mathbf{M}^{\mathbf{vv}}\mathbf{J}_2 + \mathbf{M}^{\mathbf{vu}}\ddot{\mathbf{u}}_{\dot{\mathbf{v}}} + \Phi_{\mathbf{v}}^{\mathrm{T}}\lambda_{\dot{\mathbf{v}}} = \mathbf{Q}_{\dot{\mathbf{u}}}^{\mathbf{v}}\dot{\mathbf{u}}_{\dot{\mathbf{v}}} + \mathbf{Q}_{\dot{\mathbf{v}}}^{\mathbf{v}} \tag{3.94}$$

The quantities $\dot{\mathbf{u}}_{\dot{\mathbf{v}}}$, $\ddot{\mathbf{u}}_{\dot{\mathbf{v}}}$, and $\lambda_{\dot{\mathbf{v}}}$ are evaluated below based on kinetic and kinematic information. Taking the derivative of the velocity kinematic constraint equation with respect to independent velocities yields

$$\mathbf{\Phi_u \dot{u}_{\dot{v}}} + \mathbf{\Phi_v} = \mathbf{0}$$

so

$$\mathbf{\dot{u}_{\dot{v}}} = \mathbf{H} \tag{3.95}$$

To compute $\mathbf{\ddot{u}_{\dot{v}}}$, the acceleration kinematic constraint equation is differentiated with respect to independent velocities to obtain

$$\mathbf{\Phi_u \ddot{u}_{\dot{v}}} + \mathbf{\Phi_v J}_2 = \mathbf{\tau_{\dot{u}} \dot{u}_{\dot{v}}} + \mathbf{\tau_{\dot{v}}}$$

Using the definition of the matrix $\mathbf{N}$ in Eq. (3.40), the derivative of dependent accelerations with respect to independent velocities assumes the form

$$\mathbf{\ddot{u}_{\dot{v}}} = \mathbf{N} + \mathbf{HJ}_2 \tag{3.96}$$

Finally, in order to compute the derivative of Lagrange multipliers with respect to independent velocities, the dependent equations of motions are differentiated with respect to independent velocities to yield

$$\mathbf{M^{uv}J}_2 + \mathbf{M^{uu}}\left(\mathbf{N} + \mathbf{HJ}_2\right) + \mathbf{\Phi_u^T \lambda_{\dot{v}}} = \mathbf{Q^u_{\dot{u}}H} + \mathbf{Q^u_{\dot{v}}}$$

The quantity $\mathbf{\lambda_{\dot{v}}}$ is then obtained as

$$\mathbf{\lambda_{\dot{v}}} = \left(\mathbf{\Phi_u^T}\right)^{-1}\left[\mathbf{Q^u_{\dot{u}}H} + \mathbf{Q^u_{\dot{v}}} - \mathbf{M^{uu}N} - \left(\mathbf{M^{uv}} + \mathbf{M^{uu}H}\right)\mathbf{J}_2\right] \tag{3.97}$$

By substituting expressions for $\mathbf{\dot{u}_{\dot{v}}}$, $\mathbf{\ddot{u}_{\dot{v}}}$, and $\mathbf{\lambda_{\dot{v}}}$, into Eq. (3.94), the matrix $\mathbf{J}_2$ is obtained as the solution of the multiple right side linear system

$$\mathbf{\hat{M}J}_2 = \mathbf{W} - \mathbf{H^T S} \tag{3.98}$$

where the matrix $\mathbf{S}$ is defined in Eq. (3.42), and the matrix $\mathbf{W}$ is defined as

$$\mathbf{W} = \mathbf{Q^v_{\dot{u}}H} + \mathbf{Q^v_{\dot{v}}} - \mathbf{M^{vu}N} \tag{3.99}$$

3.4.1.3 Iterative Process in the First Order Reduction Method

The integration Jacobian $\mathbf{G}$ is used to correct via an iterative process the configuration $\mathbf{w}$ at each time step. Corrections $\delta\mathbf{w}$ are the solution of a linear system of equations that generally assumes the form

$$(\alpha\mathbf{I}-\mathbf{G})\delta\mathbf{w}=\mathbf{err} \qquad (3.100)$$

where $\mathbf{err}$ is the residual in satisfying the integration formula for a multi-step formula, or the stage equation for a Runge-Kutta method. Likewise, $\alpha \equiv 1/\gamma h$ is a coefficient that depends on the integration step-size h and an integration-formula coefficient γ, while $\mathbf{I}$ is the identity matrix of appropriate dimension. For the reduced order SSODE, the dimension of this matrix is $2\times ndof$.

When solving the system of Eq. (3.100), advantage can be taken of the special structure of $\mathbf{G}$. As a consequence of the fact that the first order ODE is obtained after reducing the original second order SSODE, the vector $\delta\mathbf{w}\equiv[\delta\mathbf{p}^{\mathrm{T}}\;\delta\mathbf{v}^{\mathrm{T}}]^{\mathrm{T}}$ is the solution of the linear system

$$\begin{bmatrix}\alpha\mathbf{I} & -\mathbf{I}\\ -\mathbf{J}_1 & \alpha\mathbf{I}-\mathbf{J}_2\end{bmatrix}\begin{bmatrix}\delta\mathbf{p}\\ \delta\mathbf{v}\end{bmatrix}=\begin{bmatrix}\mathbf{b}_1\\ \mathbf{b}_2\end{bmatrix}$$

where $\mathbf{I}$ is the identity matrix of dimension $ndof$ and $\mathbf{err}\equiv[\mathbf{b}_1^{\mathrm{T}}\,\mathbf{b}_2^{\mathrm{T}}]^{\mathrm{T}}$. Analytically, the solution of this system can be obtained by solving two linear systems of dimension $ndof$ for the variations $\delta\mathbf{p}$ and $\delta\mathbf{v}$. The variation in independent positions is obtained as the solution of

$$\left(\alpha^2\mathbf{I}-\alpha\mathbf{J}_2-\mathbf{J}_1\right)\delta\mathbf{p}=\alpha\mathbf{b}_1+\mathbf{b}_2-\mathbf{J}_2\mathbf{b}_1 \qquad (3.101)$$

whereas the variation in independent velocities is obtained solving

$$\left(\alpha^2\mathbf{I}-\alpha\mathbf{J}_2-\mathbf{J}_1\right)\delta\mathbf{v}=\mathbf{b}_2+\mathbf{J}_1\mathbf{b}_1 \qquad (3.102)$$

As noted in the previous Section, $\mathbf{J}_1$ and $\mathbf{J}_2$ are the solution of two systems of linear equations with multiple-hand sides. CPU savings are obtained if, instead of solving these systems, Eqs. (3.101) and (3.102) are multiplied by the nonsingular matrix $\left(1/\alpha^2\right)\cdot\hat{\mathbf{M}}$. Using the notation

$$\Pi \equiv \hat{\mathbf{M}} - \frac{1}{\alpha}\hat{\mathbf{M}}\mathbf{J}_2 - \frac{1}{\alpha^2}\hat{\mathbf{M}}\mathbf{J}_1 \tag{3.103}$$

corrections in independent positions and velocities are solutions of

$$\Pi \cdot \delta\mathbf{p} = \frac{1}{\alpha^2}\hat{\mathbf{M}}\left(\alpha\mathbf{b}_1 + \mathbf{b}_2\right) - \frac{1}{\alpha^2}\hat{\mathbf{M}}\mathbf{J}_2\mathbf{b}_1 \tag{3.104}$$

$$\Pi \cdot \delta\mathbf{v} = \frac{1}{\alpha}\hat{\mathbf{M}}\mathbf{b}_2 + \frac{1}{\alpha^2}\hat{\mathbf{M}}\mathbf{J}_1\mathbf{b}_1 \tag{3.105}$$

Two things are worth noting here. First, there is no need to solve the systems in Eqs. (3.93) and (3.98). This is because now the matrices $\mathbf{J}_1$ and $\mathbf{J}_2$ do not appear isolated, but only in products of the form $\hat{\mathbf{M}}\mathbf{J}_1$ and $\hat{\mathbf{M}}\mathbf{J}_2$. Wherever these products appear, they can be replaced by the right-sides of the Eqs. (3.93) and (3.98). This leads to the second observation. After making these substitutions for $\hat{\mathbf{M}}\mathbf{J}_1$ and $\hat{\mathbf{M}}\mathbf{J}_2$ in the expression for Π, Π is identical to $\Psi_{\ddot{\mathbf{v}}}$ in Eq. (3.43). This is true provided the same integration formula is used for the State Space Method to integrate for both independent velocities and positions. In other words the following holds:

Proposition 1. Assume that the same integration formula is consistently used for the resulting ODE in the State Space and First Order Reduction Methods. Then, coefficient matrices of the linear systems that provide corrections in independent accelerations, and independent positions and velocities, for the State Space, and First Order Reduction Methods, respectively, are the same.

The proof of this result is obtained by direct substitution.

The benefit of the result stated in **Proposition 1** is that the rather complicated Jacobian computation for two different methods can be numerically managed using only one set of software routines. Furthermore, for the First Order Reduction Method, the proposed approach eliminates the need for explicit computation of the matrices $\mathbf{J}_1$ and $\mathbf{J}_2$.

3.4.1.4 Observations Regarding the First Order Reduction Method

Once the derivatives $\ddot{\mathbf{v}}_{\mathbf{v}}$ and $\ddot{\mathbf{v}}_{\dot{\mathbf{v}}}$ are evaluated, using Eqs. (3.93) and (3.98), the integration Jacobian $\mathbf{G}$ in Eq. (3.87) is available. Thus, practically any standard ODE implicit solver can be considered to determine the time evolution of the independent positions and velocities. It remains to provide a set of robust and efficient routines for dependent variable recovery (at the position and velocity levels), along with a fast algorithm for computation of accelerations and Lagrange multipliers.

Dependent position and velocity recovery is an important issue for efficiency and robustness. In this work, this issue is not addressed. A comprehensive analysis of this topic can be found in the work of Serban (1998). However, the issue of efficiently computing accelerations and Lagrange multipliers is addressed in detail, in what follows.

For the implicit integration of the DAE of Multibody Dynamics, as proposed in Section 3.4.1, each correction in independent positions and velocities requires computation of generalized accelerations $\ddot{\mathbf{q}}$. On the other hand, when explicit integration is considered, Eq. (2.6) must be solved for $\ddot{\mathbf{q}}$, since independent accelerations are used to advance the integration to the next time step. Depending on the type of analysis being considered, the Lagrange multipliers are also quantities of interest. These are reasons that motivated development of methods for efficient computation of generalized accelerations $\ddot{\mathbf{q}}$ and Lagrange multipliers λ.

In terms of the number of variables used to model a multibody system, there are two extreme approaches;

(1). The descriptor form, or the Cartesian representation, or the body representation, in which the mechanical systems are represented using for each body a set of coordinates specifying the position of a particular point on that body, along with a choice of parameters that specify the orientation of the body with respect to a global reference frame

(2). The minimal form, or the recursive formulation, or the joint representation, in which mechanical systems are represented in terms of a minimal set of generalized coordinates

Throughout this document, BR (for Body Representation) and JR (for Joint Representation) denote these two approaches. The Cartesian formulation (BR) is convenient for representing the state of a mechanical system, because kinetic and kinematic information is readily available for each body in the system. The major drawback of the Cartesian approach is that the dimension of the problem increases dramatically, compared to the alternative provided by the recursive formulation (JR).

Sections 3.4.2 and 3.4.3 focus on constructing methods that efficiently solve for generalized accelerations $\ddot{\mathbf{q}}$ and Lagrange multipliers λ in both formulations. These quantities are the solution of the linear system in Eq. (2.16). The coefficient matrix of this system is called in what follows the augmented matrix. The objective is to take advantage of both structure, and topology-induced sparsity when solving for $\ddot{\mathbf{q}}$ and λ. As mentioned by Andrezejewski and Schwerin (1995), no code seems to exploit both the structure of the augmented matrix, and the sparsity in a satisfying manner.

3.4.2 Computing Accelerations in the Cartesian Representation

In this Section, a method for computing accelerations and Lagrange multipliers in the Cartesian representation is presented, following the approach presented by Negrut, Serban, and Potra (1996). A slightly different approach is proposed by Serban, Negrut, Haug, and Potra (1997), where numerical results showing the good performance of the algorithm were provided.

The Newton-Euler constrained equations of motion for a multibody system assume the form

$$\begin{aligned} &\mathbf{M}(\mathbf{q})\ddot{\mathbf{q}} + \Phi_{\mathbf{q}}^{\mathrm{T}}(\mathbf{q})\lambda = \mathbf{Q}(\mathbf{q},\dot{\mathbf{q}}) \\ &\Phi(\mathbf{q}) = \mathbf{0} \end{aligned} \tag{3.106}$$

where $\mathbf{q}$, $\dot{\mathbf{q}}$, and $\ddot{\mathbf{q}}$ are vectors in $\Re^n$ that represent generalized position, velocity, and acceleration; $\mathbf{Q}(\mathbf{q},\dot{\mathbf{q}}) \in \Re^n$ is the vector of generalized forces; $\lambda \in \Re^m$ is the vector of Lagrange multipliers; and $\mathbf{M}(\mathbf{q})$ is the $n \times n$ mass matrix. Finally, $\Phi_{\mathbf{q}}$ is the $m \times n$ constraint Jacobian, $m < n$. The kinematic constraints are assumed to be independent, so the Jacobian matrix has full row rank.

Differentiating the position kinematic constraint equation twice with respect to time, and replacing position kinematic constraint equation with the newly obtained acceleration kinematic constraint equation reduces the index of the DAE of Eq. (3.106) from 3 to 1. The resulting index 1 DAE assumes the from

$$\begin{bmatrix} \mathbf{M} & \Phi_{\mathbf{q}}^{\mathrm{T}} \\ \Phi_{\mathbf{q}} & \mathbf{0} \end{bmatrix} \begin{bmatrix} \ddot{\mathbf{q}} \\ \lambda \end{bmatrix} = \begin{bmatrix} \mathbf{Q} \\ \tau \end{bmatrix} \tag{3.107}$$

where the right side of the acceleration kinematic equation is defined in Eq. (2.4). The coefficient matrix of the linear system in Eq. 3 is of dimension $(n+m)\times(n+m)$, and in what follows it is denoted by $\mathbf{A}$, and called the augmented matrix.

The dimension of the augmented matrix assumes large values for relatively simple mechanical systems. For the seven body mechanism in Figure 1 modeled as a planar mechanical, $n = 21$; i.e., 7 bodies with 3 coordinates, two positions and one orientation angle. The number of constraints m is 20. The mechanism has one degree of freedom, and the augmented matrix is of dimension 41. The situation is more drastic if the mechanism is modeled as a spatial system. In this case n is 42, while the number of constraints imposed is 41. The augmented matrix is thus of dimension 83. It can be seen that the dimension of the augmented is large, especially when spatial mechanical systems with few degrees of freedom are considered.

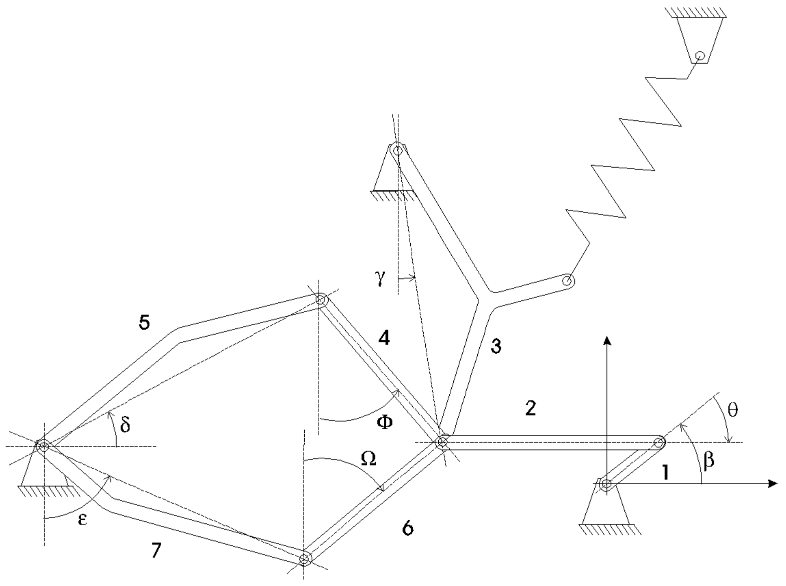


Figure 1. Seven Body Mechanism

The strategy proposed for solution of the linear system in Eq. (3.107) proceeds by formally expressing accelerations $\ddot{\mathbf{q}}$ in terms of Lagrange multipliers. Assuming for the moment that the mass matrix is nonsingular, accelerations are substituted back into acceleration kinematic equation to obtain

$$\left(\boldsymbol{\Phi}_{\mathbf{q}}\mathbf{M}^{-1}\boldsymbol{\Phi}_{\mathbf{q}}^{\mathrm{T}}\right)\lambda = \boldsymbol{\Phi}_{\mathbf{q}}\mathbf{M}^{-1}\mathbf{Q} - \tau \tag{3.108}$$

For notational convenience, the dependency of the Jacobian on positions, and of the generalized force on both positions and velocities, is suppressed.

The matrix $\mathbf{B} \equiv \boldsymbol{\Phi}_{\mathbf{q}}\mathbf{M}^{-1}\boldsymbol{\Phi}_{\mathbf{q}}^{\mathrm{T}}$ is referred as the reduced matrix. Diagonal blocks in this matrix have the form

$$\mathbf{B}_{jj} = (\boldsymbol{\Phi}_{\mathbf{q}_{i1}}^{j})\mathbf{M}_{i1}^{-1}(\boldsymbol{\Phi}_{\mathbf{q}_{i1}}^{j})^{\mathrm{T}} + (\boldsymbol{\Phi}_{\mathbf{q}_{i2}}^{j})\mathbf{M}_{i2}^{-1}(\boldsymbol{\Phi}_{\mathbf{q}2}^{j})^{\mathrm{T}} \tag{3.109}$$

if joint j connects bodies $i1$ and $i2$. Off-diagonal blocks assume the form

$$\mathbf{B}_{jk} = (\boldsymbol{\Phi}_{\mathbf{q}_i}^{j})\mathbf{M}_{i}^{-1}(\boldsymbol{\Phi}_{\mathbf{q}_i}^{k})^{\mathrm{T}}, \qquad j \neq k \tag{3.110}$$

if joints j and k are on the same body i. Otherwise, they are zero.

In Eqs. (3.109) and (3.110), $\boldsymbol{\Phi}_{\mathbf{q}_i}^{j}$ is the derivative of the constraint function corresponding to joint j with respect to the generalized coordinates $\mathbf{q}_i$ of body i, and $\mathbf{M}_i^{-1}$ is the inverse of the mass matrix of body i.

3.4.2.1 The Planar Case

For planar mechanisms, the mass matrix $\mathbf{M}$ is positive definite. Therefore, since the constraint Jacobian is assumed to have full row rank, the reduced matrix $\mathbf{B}$ is positive definite. Assuming that Lagrange multipliers are available, accelerations are easily computed using the equations of motion

$$\mathbf{M}\ddot{\mathbf{q}} = \mathbf{Q} - \boldsymbol{\Phi}_{\mathbf{q}}^{\mathrm{T}}\lambda \tag{3.111}$$

The mass matrix $\mathbf{M}$ is of dimension $(3nb \times 3nb)$, where nb is the number of bodies in the system, and has a diagonal block structure

$$\mathbf{M} = \begin{bmatrix} \mathbf{M}_1 & \mathbf{0} & \cdots & \mathbf{0} \\ \mathbf{0} & \mathbf{M}_2 & \dots & \mathbf{0} \\ \dots & \dots & \dots & \dots \\ \mathbf{0} & \mathbf{0} & \cdots & \mathbf{M}_{nb} \end{bmatrix} \tag{3.112}$$

For the planar case, given a certain joint numbering scheme, the following algorithm is used to compute $\ddot{\mathbf{q}}$ and λ:

Algorithm 1

(1). Assemble the reduced matrix $\mathbf{B}$, based on Eqs. (3.109) and (3.110)

(2). Solve the linear system of Eq. (3.108) for λ

(3). Recover the generalized accelerations, based on Eqs. (3.111) and (3.112)

Step 2 is discussed in Section 3.4.2.3. Step 3 requires the solution of nb systems of linear equations of dimension 3×3, to recover generalized accelerations. This process can be carried out in parallel. For rigid body simulation, the matrices $\mathbf{M}_i$ are diagonal, and they are constant over the entire simulation. Therefore, in an efficient implementation, matrix $\mathbf{M}$ is factored during the pre-processing stage of the simulation, computation of generalized accelerations requiring only a matrix-vector multiplication.

3.4.2.2 The Spatial Case

With notations used by Haug (1989), generalized accelerations and Lagrange multipliers are the solution of the linear system

$$\begin{bmatrix} \mathbf{M} & \mathbf{0} & \Phi_{\mathbf{r}}^{\mathrm{T}} & \mathbf{0} \\ \mathbf{0} & 4\mathbf{G}^{\mathrm{T}}\mathbf{J}'\mathbf{G} & \Phi_{\mathbf{p}}^{\mathrm{T}} & \left(\Phi_{\mathbf{p}}^{\mathbf{p}}\right)^{\mathrm{T}} \\ \Phi_{\mathbf{r}} & \Phi_{\mathbf{p}} & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \Phi_{\mathbf{p}}^{\mathbf{p}} & \mathbf{0} & \mathbf{0} \end{bmatrix} \begin{bmatrix} \ddot{\mathbf{r}} \\ \ddot{\mathbf{p}} \\ \lambda \\ \lambda^{\mathbf{p}} \end{bmatrix} = \begin{bmatrix} \mathbf{F}^{\mathrm{A}} \\ 2\mathbf{G}^{\mathrm{T}}\mathbf{n}'^{\mathrm{A}} + 8\dot{\mathbf{G}}^{\mathrm{T}}\mathbf{J}'\dot{\mathbf{G}}\mathbf{p} \\ \tau \\ \tau^{\mathbf{p}} \end{bmatrix} \tag{3.113}$$

where

$$
\begin{aligned}
\mathbf{r} &\equiv [\mathbf{r}_1^{\mathrm{T}},\ldots,\mathbf{r}_{nb}^{\mathrm{T}}]^{\mathrm{T}} \\
\mathbf{p} &\equiv [\mathbf{p}_1^{\mathrm{T}},\ldots,\mathbf{p}_{nb}^{\mathrm{T}}]^{\mathrm{T}} \\
\mathbf{F} &\equiv [\mathbf{F}_1^{\mathrm{T}},\ldots,\mathbf{F}_{nb}^{\mathrm{T}}]^{\mathrm{T}} \\
\mathbf{n}' &\equiv [\mathbf{n}_1'^{\mathrm{T}},\ldots,\mathbf{n}_{nb}'^{\mathrm{T}}]^{\mathrm{T}} \\
\mathbf{M} &\equiv \mathrm{diag}(\mathbf{M}_i) \qquad (3.114) \\
\mathbf{J}' &\equiv \mathrm{diag}(\mathbf{J}_i') \\
\mathbf{G} &\equiv \mathrm{diag}(\mathbf{G}_i) \\
\Phi^{\mathbf{p}} &\equiv [\mathbf{p}_1^{\mathrm{T}}\mathbf{p}_1 - 1,\ldots,\mathbf{p}_{nb}^{\mathrm{T}}\mathbf{p} - 1]^{\mathrm{T}} \\
\Phi_{\mathbf{p}}^{\mathbf{p}} &\equiv \mathrm{diag}(2\mathbf{p}_i^{\mathrm{T}})
\end{aligned}
$$

The quantity τ in Eq. (3.113) is the right side of the acceleration kinematic equation, as given in Eq. (2.4). The right side of the Euler parameter constraint equation is obtained easily as

$$\tau^{\mathbf{p}} = [-2\dot{\mathbf{p}}_1^{\mathrm{T}}\mathbf{p}_1,\ldots,-2\mathbf{p}_{nb}^{\mathrm{T}}\mathbf{p}]$$

In Eq. (3.113), Lagrange multipliers corresponding to position kinematic constraint equations are denoted by λ, while those corresponding to Euler parameter normalization constraint equations $\mathbf{p}_i^{\mathrm{T}}\mathbf{p}_i = 1$ are denoted by $\lambda^{\mathbf{p}}$.

The coefficient matrix in Eq. (3.113) can be brought to a form that resembles the two dimensional case by denoting

$$\overline{\mathbf{M}} \equiv \mathrm{diag}(\mathbf{M}, 4\mathbf{G}^{\mathrm{T}}\mathbf{J}'\mathbf{G})$$

$$\overline{\Phi} \equiv [\Phi^{\mathrm{T}}, \left(\Phi^{\mathbf{p}}\right)^{\mathrm{T}}]^{\mathrm{T}}$$

Then, the coefficient matrix would assume the form in Eq. (3.107). However, the idea used for the two dimensional case is not directly applicable here, since the newly defined composite mass matrix $\overline{\mathbf{M}}$ fails to be positive definite. In fact, by simply taking the vector $\mathbf{z} \equiv [\mathbf{0}^{\mathrm{T}}, \mathbf{p}^{\mathrm{T}}]^{\mathrm{T}} \neq \mathbf{0}$, $\overline{\mathbf{M}}\mathbf{z} = \mathbf{0}$. This is a direct consequence of the fact that in any consistent configuration of the bodies in the system, $\mathbf{Gp} = \mathbf{0}$ (Haug, 1989). The approach in which the reduced system is first solved for Lagrange multipliers, and then generalized accelerations are recovered fails, since it is based on positive definiteness of the mass matrix.

The solution proposed is as follows: *keep the algorithm and temporarily change the representation*. Thus, instead of following the Euler parameter representation of the constrained equations of motion for a mechanical system, for the moment consider the original Newton-Euler formulation in which acceleration is retrieved by solving the linear system (Haug, 1989)

$$\begin{bmatrix} \mathbf{M} & \mathbf{0} & \Phi_{\mathbf{r}}^{\mathrm{T}} \\ \mathbf{0} & \mathbf{J}' & \Phi_{\pi}^{\mathrm{T}} \\ \Phi_{\mathbf{r}} & \Phi_{\pi} & \mathbf{0} \end{bmatrix} \begin{bmatrix} \ddot{\mathbf{r}} \\ \dot{\omega}' \\ \lambda \end{bmatrix} = \begin{bmatrix} \mathbf{F}^{\mathrm{A}} \\ \mathbf{n}' - \tilde{\omega}'\mathbf{J}'\omega' \\ \tau \end{bmatrix} \tag{3.115}$$

At each integration time step, since the quantities $\mathbf{p}$ and $\dot{\mathbf{p}}$ are known, the matrix $\mathbf{G}$ can be constructed, and once the time derivative of the angular velocities $\dot{\omega}'$ is available, the Euler parameters accelerations are calculated using the relation (Haug, 1989)

$$\ddot{\mathbf{p}} = \frac{1}{2}\mathbf{G}^{\mathrm{T}}\dot{\omega}' - (\dot{\mathbf{p}}^{\mathrm{T}}\dot{\mathbf{p}})\mathbf{p} \tag{3.116}$$

Notice that the construction of the coefficient matrix and the right side of Eq. (3.115) is less CPU intensive when compared to the approach in Eq. (3.113). The reason for which the ω angular velocity-formulation of Eq. (3.115) is not commonly used is that the angular velocity is not integrable (Haug, 1989). The Euler parameter formulation avoids this difficulty. For the two formulations discussed above, there is a formulation that is not recommended on mathematical integration grounds, but is desirable based on linear algebra considerations, and conversely. The idea proposed by Negrut, Serban, and Potra (1996) is to consider each formulation when it is more beneficial: the ω -formulation for the linear algebra part, and the Euler parameter formulation for the numerical integration part. What makes this feasible is the ability to move back and forth between the two formulations at relatively no CPU penalty.

With this, the case of spatial mechanisms can be treated much as the planar case. With nb being the number of bodies of the mechanical system model, the following notation is introduced

$$\mathbf{x}_i \equiv [\ddot{\mathbf{r}}_i \; \dot{\omega}_i'^{\mathrm{T}}]^{\mathrm{T}} \qquad \mathbf{x} \equiv [\mathbf{x}_1^{\mathrm{T}}, \mathbf{x}_2^{\mathrm{T}}, \ldots, \mathbf{x}_{nb}^{\mathrm{T}}]^{\mathrm{T}} \tag{3.117}$$

$$\mathbf{Q}_i \equiv \begin{bmatrix} \mathbf{F}_i^{\mathrm{A}} \\ \mathbf{n}_i' - \tilde{\omega}_i' \mathbf{J}_i' \omega_i' \end{bmatrix} \qquad \mathbf{Q} \equiv [\mathbf{Q}_1^{\mathrm{T}}, \mathbf{Q}_2^{\mathrm{T}}, \ldots \mathbf{Q}_{nb}^{\mathrm{T}}]^{\mathrm{T}} \tag{3.118}$$

$$\overline{\mathbf{M}}_i \equiv \mathrm{diag}(\mathbf{M}_i, \mathbf{J}_i') \qquad \overline{\mathbf{M}} \equiv \mathrm{diag}(\overline{\mathbf{M}}_1, \overline{\mathbf{M}}_2, \ldots, \overline{\mathbf{M}}_{nb}) \tag{3.119}$$

The spatial case reduces to finding the solution of the system

$$\begin{bmatrix} \overline{\mathbf{M}} & \Phi_{\mathbf{q}}^{\mathrm{T}} \\ \Phi_{\mathbf{q}} & \mathbf{0} \end{bmatrix} \begin{bmatrix} \mathbf{x} \\ \lambda \end{bmatrix} = \begin{bmatrix} \mathbf{Q} \\ \tau \end{bmatrix} \tag{3.120}$$

The matrix $\overline{\mathbf{M}}$ is positive definite and the algorithm presented for the planar case can be employed to compute the unknowns $\mathbf{x}$ and λ. Thus, first proceed by solving for Lagrange multipliers the equivalent of Eq. (3.108), which for the spatial case with the notation in Eqs. (3.117) through (3.119) assumes the form

$$\left(\Phi_{\mathbf{q}} \overline{\mathbf{M}} \Phi_{\mathbf{q}}^{\mathrm{T}}\right) \lambda = \Phi_{\mathbf{q}} \overline{\mathbf{M}}^{-1} \mathbf{Q} - \tau \tag{3.121}$$

and then recover the $\ddot{\mathbf{r}}_i$ and $\dot{\omega}_i'$ by solving

$$\overline{\mathbf{M}}_i \mathbf{x}_i = \mathbf{Q}_i - \Phi_{\mathbf{q}_i}^{\mathrm{T}} \lambda \tag{3.122}$$

The following algorithm is proposed:

Algorithm 2

(1). Assemble the reduced matrix $\mathbf{B}$, based on Eqs. (3.109) and (3.110)

(2). With the notations introduced in Eqs. (3.117) through (3.119), solve the reduced linear system of Eq. (3.121)

(3). For each body in the mechanical system, solve the 6×6 system of Eq. (3.122) to recover $\ddot{\mathbf{r}}_i$ and $\dot{\omega}'_i$

(4). Compute $\ddot{\mathbf{p}}$ via Eq. (3.116)

3.4.2.3 Factoring the Reduced Matrix

In order to efficiently solve Eq. (3.108) for Lagrange multipliers, the topology of the mechanical system should be used to advantage. Two simple examples are presented to illustrate this point. The first example is Andrew's squeezing mechanism, or the Seven-Body Mechanism, which is a standard test problem. A description of this mechanism can be found in the work of Schiehlen (1990). The mechanism is presented in Figure 1. The second example is a 50-link chain consisting of simple pendulums shown in Figure 2. This problem is larger and admits a joint numbering scheme that results in a block diagonal matrix $\mathbf{B}$.

The Seven-Body Mechanism is a closed loop mechanism, whose graph is presented in Figure 3. The joints are represented as vertices, while the bodies are connecting edges. This representation is different from the usual one, in which bodies are vertices and joints are connecting edges of the graph. In the proposed representation, one joint connects exactly two bodies, and the same body can appear more than once as an edge. This is the case when a body is connected to two or more bodies by joints. Thus, when a body is attached through k joints with neighboring bodies it yields k edges in the graph. The ground is not represented as a body, so free vertices located at the end of the graph, vertices 1, 8, and 10 in Figure 3(a), connect the adjacent body to ground.

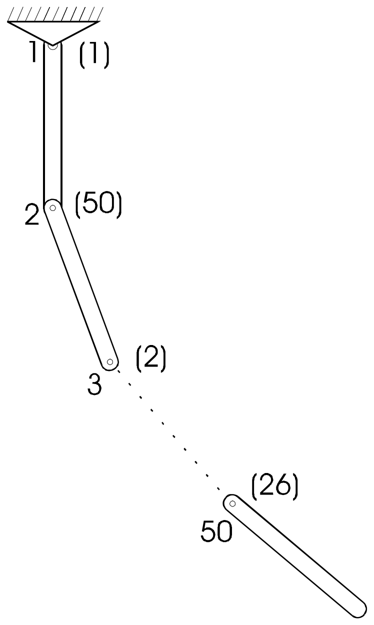
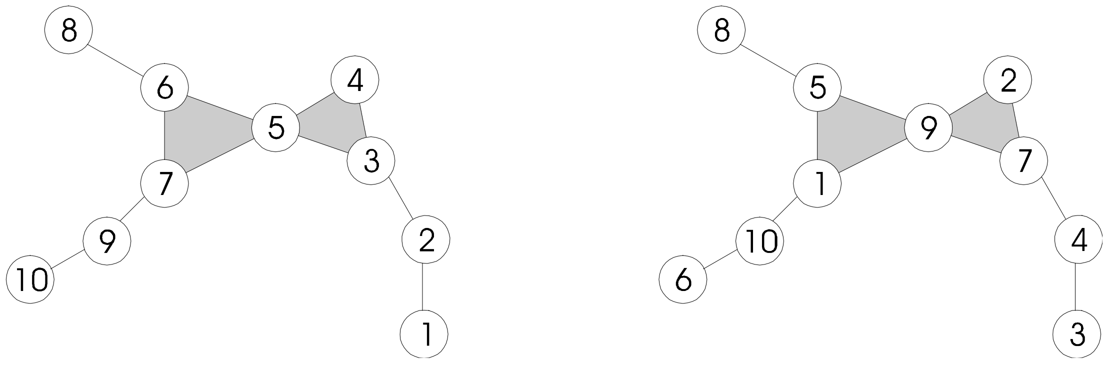


Figure 2. Chain of Pendulums



(a) (b)

Figure 3. Graph Representation: Seven-Body Mechanism

For the seven-body mechanism, two different joint numbering sequences, shown in Figure 3(a) and (b), yield the corresponding reduced matrices $\mathbf{B}_1$ and $\mathbf{B}_2$ in Figure 4 (a) and (b), respectively. Only non-zero elements are represented.

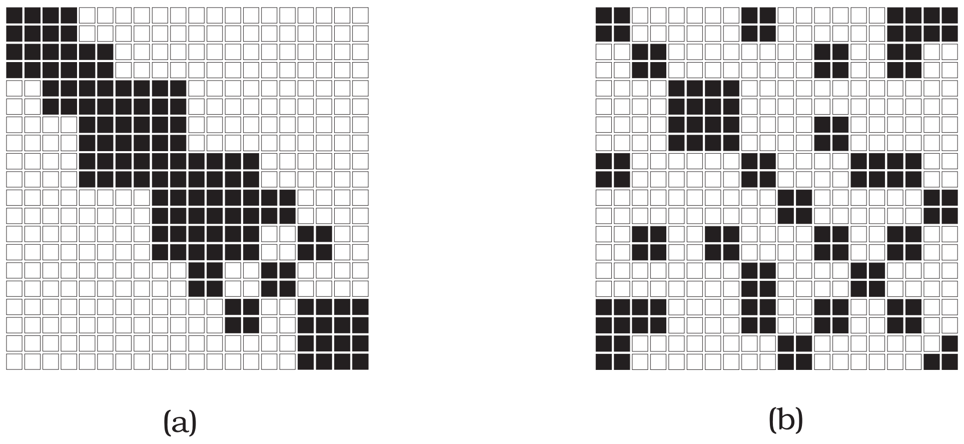


Figure 4. Reduced Matrix **B**: Seven-Body Mechanism

Figure 6 shows the sparsity pattern of the $\mathbf{B}$ matrix corresponding to two different joint-numbering schemes for the chain of 50 pendulums. The first matrix is obtained by numbering the joints in the natural order (1,2,3, ...), starting with the joint between body 1 and ground as shown in Figure 5(a). The second matrix corresponds to the numbering in which the first joint of the chain is 1, the second is 50, the third is 2, the next is 49, and so on, as shown in Figure 5, (b). This latter numbering scheme is unlikely to ever be used, and it is considered only to illustrate the impact of a bad joint numbering scheme on the sparsity pattern of the reduced matrix $\mathbf{B}$.

These results show that the bandwidth of the reduced matrix, and therefore the efficiency of a sparse solver, depend on the joint-numbering scheme. The choice of joint numbering is made at the stage of mechanical system modeling, and then used throughout the simulation. For improved performance, it is important to determine a strategy that automatically numbers the joints of the model, such that the amount of work in factoring the reduced mass matrix is minimized.

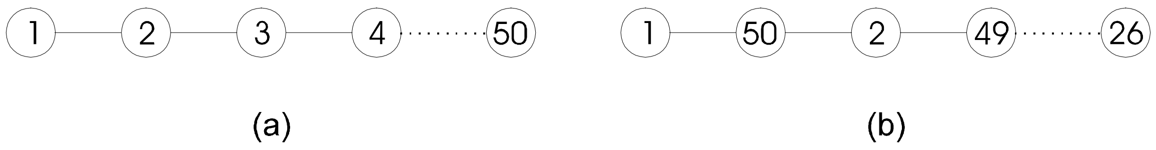


Figure 5. Two Joint Numbering Schemes: Chain of Pendulums

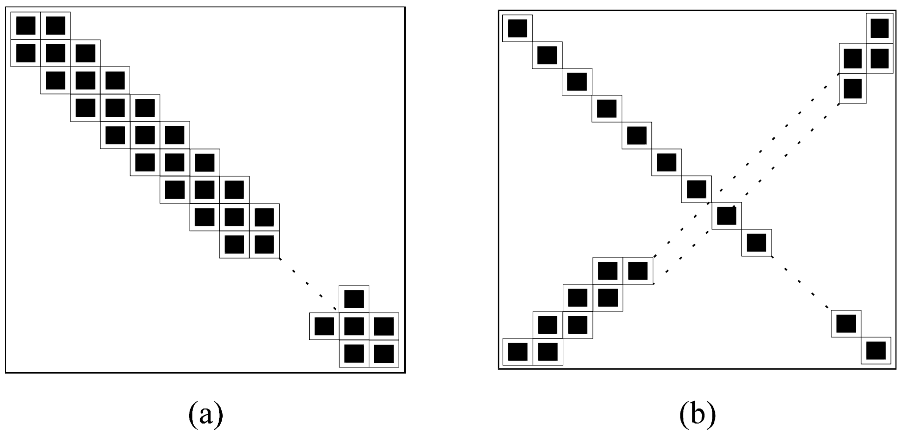


Figure 6. Reduced Matrix: Chain of Pendulums

Since the reduced matrix is positive definite, factorization is based on block Cholesky decomposition, taking advantage of the sparse-block structure of the reduced matrix. The block structure is given by Eqs. (3.109) and (3.110). The block width β_i of row i is defined in terms of blocks as

$$\beta_i = \max\{(i-j) : \overline{B}_{ij} \neq 0,\ j < i\} \tag{3.123}$$

The bandwidth β of the reduced matrix, in terms of blocks, is given by the maximum row width as

$$\beta = \max\{\beta_i : i = 1,2,\ldots,m\} \tag{3.124}$$

The envelope or profile of the matrix is given by

$$\mathrm{env}(\mathbf{B}) = \sum_{i=1}^{m} \beta_i \tag{3.125}$$

In the Cholesky factorization of $\mathbf{B}$, the computational work of an algorithm that makes use of an envelope storage scheme can be bounded from above by (Negrut, Serban, and Potra, 1996),

$$\mathrm{work}(\mathbf{B}) = \frac{1}{2}\sum_{i=2}^{m} \beta_i(\beta_i + 3) \tag{3.126}$$

This estimate is an upper bound on the actual work in a block oriented Cholesky factorization algorithm. An operation in Eq. (3.126) is considered to be either a block-block multiplication, or a block inversion. Equation (3.126) does not take into account the square root of the diagonal elements (for the scalar case), or the Cholesky decomposition of diagonal blocks (in the block form). If these operations were to be considered, the above formula becomes $\mathrm{work}(\mathbf{B}) = 1/2\sum_{i=2}^{m}(\beta_i^2 + 3\beta_i + 2)$

The values of the bandwidth, envelope, and the actual work performed in factorization of the reduced matrix depend on the choice of row and column ordering in matrix $\mathbf{B}$. In general the minima for these three quantities will not be obtained with the same ordering. Minimizing the bandwidth of a matrix is an NP-complete problem, and minimizing any of the other two quantities considered above is an intractable task. It is common practice to use a bandwidth and/or envelope reduction algorithm to reorder the matrix, prior to applying Cholesky factorization. Although this approach does not exactly minimize the amount of work in the Cholesky factorization, the results are satisfactory. Algorithms such as Gibbs-King, or Gibbs-Poole-Stockmeyer (Lewis, 1982) perform this operation. These algorithms employ a local-search in the adjacent graph of the matrix. Central to this discussion is the fact that this graph is identical to the graph representation of the mechanism, as proposed earlier in this Section. Thus, permutations of rows and columns of the reduced matrix via symmetric permutation of the form $\mathbf{P}^{\mathrm{T}}\mathbf{BP}$; i.e., renumbering the vertices in the adjacency graph of the matrix, is equivalent to renumbering the joints of the mechanical system. Therefore, the reordered index set given by any of the above algorithms is translated into a new numbering of the joints of the mechanical system.

In the discussion above, blocks of the reduced matrix were manipulated as if they were simple entries in a matrix. These blocks are inherited from the structure of the problem, and their dimension is dictated by the number of constraint equations associated with different joint types, as expressed by Eqs. (3.109) and (3.110). These blocks could be further broken down, going to entry-level. The bandwidth and/or envelope reduction algorithms would result in better-structured matrices, in terms of operations needed for Cholesky factorization. However, in this case the immediate relationship between the topology of a mechanism, as defined at the beginning of this Section, and the structure of the reduced matrix is lost. Ultimately, this is the difference between regarding a certain mechanical joint as a set of basic constraint equations that are manipulated together or, manipulating these basic constraint equations on an individual basis, disregarding the fact that some of them together describe a certain mechanical joint.

3.4.2.4 Numerical Results

The Seven-Body Mechanism and the chain of pendulums are considered to illustrate the benefit of using the proposed algorithm. Numerical results for other mechanical systems are presented in the paper of Serban, Negrut, Haug, and Potra (1997). The conclusion drawn there, along with the observed speed-up factors for the proposed algorithm, are qualitatively the same.

Based on the theoretical considerations of Sections 3.4.2.1 through 3.4.2.3, an algorithm to compute Lagrange multipliers and generalized accelerations is defined as follows:

Algorithm 3

(1). For an arbitrary joint numbering, create the reduced matrix $\mathbf{B}$, based on Eqs. (3.109) and (3.110).

(2). Use a joint numbering algorithm Gibbs-King (Lewis, 1982), Gibbs-Poole-Stockmeyer (Lewis, 1982), etc. to reduce the profile of matrix $\mathbf{B}$.

(3). Renumber the joints of the mechanical system model, as suggested by the profile reduction algorithm.

(4). Apply *Algorithm 1* for the planar case, or *Algorithm 2* for the spatial case, to recover first the Lagrange multipliers, and then on a per body basis, the generalized accelerations.

During simulation, steps (1) through (3) are done once at the pre-processing stage. The resulting joint numbering is then used throughout the simulation. On the other hand, step (4) is taken once at each integration step for the situation when explicit integration is used, or once for each iteration of the quasi-Newton algorithm, when implicit integration is used.

A set of numerical experiments, denoted below with *ExpA*, is carried out to compare two strategies for obtaining accelerations and Lagrange multipliers. The strategies are as follows

- *ExpA1*: Solve the augmented system of Eq. (3.107), using the routine *ma28* of the direct sparse solver Harwell, (Duff, 1980)
- *ExpA2*: Apply *Algorithm 3* above

The routine *ma28* employs a multi-frontal sparse Gaussian elimination, combined with a modified Markowitz strategy for local pivoting.

Table 2 contains CPU times in seconds required to compute accelerations and Lagrange multipliers 1000 times. All numerical experiments were performed on a HP9000 model J210 computer, with two PA 7200 processors.

Table 2. Numerical Results: Solving For Accelerations and Lagrange Multipliers

	Seven Body Mechanism	*Chain of 50 Pendulums*
ExpA1	8.888	1.2933
ExpA2	1.018	0.2313

The results suggest that a speed-up of a factor of 6 to 8 is obtained by using the proposed method, compared to the direct approach of solving the augmented system. The observed speed-up ratio is cut down to values of 3 to 4, when spatial mechanical system models are considered (Serban, Negrut, Haug, Potra, 1997).

A second set of numerical experiments was carried out to assess the impact of different joint numbering schemes on CPU time required to solve the reduced system for Lagrange multipliers. For both planar and spatial cases, the reduced matrix $\mathbf{B}$ is positive definite.

The proposed algorithm is compared to well established dense solvers from Lapack and Harwell. The sparse Cholesky solver developed is block oriented and takes advantage of the sparse skyline format used to store the reduced matrix $\mathbf{B}$. Two LAPACK routines, one based on the *dgetrf/dgetrs* pair and a second that takes advantage of the positive definiteness of the reduced matrix, (*dppsvx*) are considered. Finally, the sparse solver *MA28* of Harwell is used for comparison. The CPU results in seconds for 1000 solutions of the reduced system are presented in Tables 3 and 4. A bad joint numbering is considered first (case *ExpB1*), while a good joint numbering obtained after applying Gibbs-King (Lewis, 1982) envelope reduction algorithm is used in the second case (*ExpB2*).

Table 3. Timing Results for Seven Body Mechanism

	dgetrfr/dgetrs	*dppsvx*	*MA28*	*Sparse Cholesky*
ExpB1	0.79	0.47	0.88	0.71
ExpB2	0.79	0.47	0.88	0.23

Table 4. Timing Results for Chain of Pendulums

	dgetrfr/dgetrs	*dppsvx*	*MA28*	*Sparse Cholesky*
B1	53.87	27.45	4.14	54.47
B2	53.87	27.45	4.14	0.98

For the algorithm developed, the envelope reducing approach of reordering blocks in the matrix $\mathbf{B}$ influences the efficiency of Lagrange multiplier computation. While for the other methods a reordering process is irrelevant, for the sparse Cholesky algorithm it leads to a speed-up of more than 3 for a rather restrictive example of a closed loop mechanical system such as the Seven-Body Mechanism.

The proposed algorithm is especially attractive when open loop mechanisms such as the chain of 50 pendulums are considered. In this instance, since the reduced matrix is block diagonal, the proposed algorithm is extremely fast, when compared to the dense alternatives provided by Lapack. Compared with the sparse solver from the Harwell library, the speed-up obtained is about 4, and this was similar for both test problems. Surprisingly enough, the Harwell solver compares poorly with the dense solvers from Lapack for the Seven-Body Mechanism. Apparently, the dimension of the problem (20 by 20) is small and the fill-in index for this problem is too high for the sparse solver to have an impact. In this case however, a speed-up factor of 4 can be observed, comparing the proposed algorithm to all the other alternatives, dense or sparse.

3.4.3 Computing Accelerations in Minimal Representation

In this Section is presented an algorithm that takes advantage of the topology of the mechanical system modeled using a minimal set of generalized coordinates. It is shown that the generalized mass matrix, the so called composite inertia matrix, associated with any open or closed loop mechanism is positive definite. Based on this result, an algorithm that efficiently solves for accelerations is designed. Significant speed-ups are obtained, due both to the no fill-in factorization of the composite inertia matrix and to the degree of parallelism attainable with the new algorithm.

The discussion is organized as follows. In Section 3.4.3.1 the joint representation (JR) approach to modeling multibody systems is briefly outlined. The notation and relevant results of Tsai (1989) are referred to in this discussion. In Section 3.4.3.2, positive definiteness of the CIM for open and closed loop mechanisms is proved. The proposed technique of factoring the CIM is presented in Section 3.4.3.3. The potential for parallel implementation of the proposed factorization is outlined in Section 3.4.3.4. Results of numerical experiments aimed at showing the capabilities of the new approach conclude the discussion.

3.4.3.1 JR Modeling of Multibody Systems

3.4.3.1.1 Basic Concepts

The formalism behind JR modeling of multibody systems is complex. This Section provides a brief introduction to this topic, with the goal of defining quantities related to the issue of computing generalized accelerations. The interested reader is referred to the work of Tsai (1989) for a detailed description of the formalism.

The main concept in JR modeling is that body j is viewed as being located and oriented relative to its inboard body i. Figure 7 shows a pair of connected bodies with general relative rotation and translation. The inboard body i is located by the position vector $\mathbf{r}_i$ from the origin of the global xyz coordinate frame to the origin of the body $x_i'y_i'z_i'$ frame. The $x_i'y_i'z_i'$ frame is oriented by an orthogonal transformation matrix $\mathbf{A}_i$, which transforms a vector in the body i reference frame to the global reference frame. A joint $x_{ij}''y_{ij}''z_{ij}''$ reference frame, is defined and fixed on body i at the joint connection point O_{ij}'', which is located by the constant vector $\mathbf{s}_{ij}'$ from the origin of the $x_i'y_i'z_i'$ frame. A vector $\mathbf{d}_{ij}$ is defined from the origin of the joint $x_{ij}''y_{ij}''z_{ij}''$ frame, to the origin O_j' of the $x_j'y_j'z_j'$ frame on the outboard body. Reference frames for each successive body in the kinematic chain are defined in the same way as those for body i.

The so called body reference frame is the reference frame located at the joint that connects the considered body to its inboard body. This definition applies to all bodies except the base body, which has no inboard body. For the base body, a body reference frame is attached at the center of gravity. The transformation from the inboard $x_i'y_i'z_i'$ body reference frame to the outboard body reference frame, requires only the constant transformation to the $x_{ij}''y_{ij}''z_{ij}''$ joint reference frame, followed by a joint transformation to the $x_j'y_j'z_j'$ reference frame of body j. The reason for defining the body reference frame at the inboard joint is that every body, except for the base body, has one and only one inboard body in the kinematic chain, while it may have several outboard bodies.

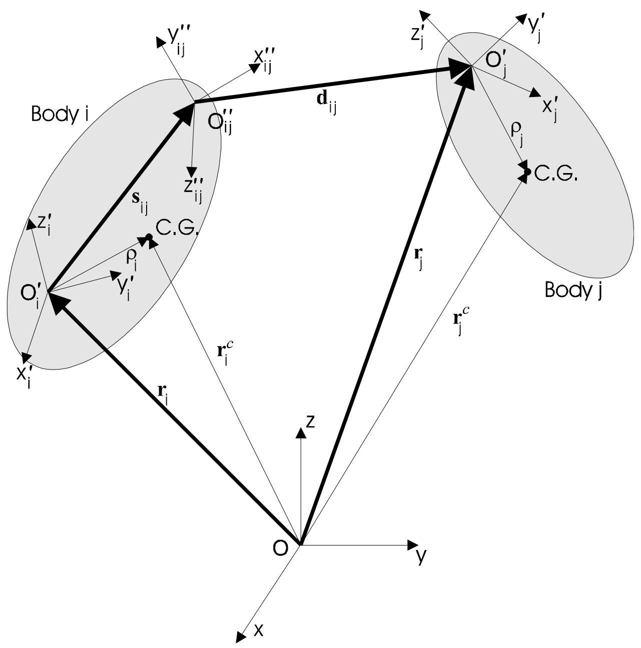


Figure 7. A Pair of Connected Bodies in JR

The origin of the body reference frame at the inboard joint is uniquely defined. Given the position of body i, the origin of the body j reference frame is located by the position vector $\mathbf{r}_j$

$$\mathbf{r}_j = \mathbf{r}_i + \mathbf{s}_{ij} + \mathbf{d}_{ij} \tag{3.127}$$

where $\mathbf{d}_{ij} = \mathbf{A}_i \mathbf{C}_{ij} \mathbf{d}''_{ij}(\mathbf{q}_j)$ is a vector from the joint reference frame origin O''_{ij} on body i to the body reference frame origin O'_j on body j, $\mathbf{q}_j$ is the vector of relative coordinates for the joint, and $\mathbf{C}_{ij}$ is the constant orthogonal transformation matrix between the O''_{ij} and O'_i frames on body i.

The angular velocity of body j can be expressed as

$$\omega_j = \omega_i + \omega_{ij} \tag{3.128}$$

where ω_i is the angular velocity of body i, ω_j is the angular velocity of body j, and ω_{ij} is the angular velocity of body j relative to body i. The vector ω_{ij} can be obtained from the relative coordinate velocity $\dot{\mathbf{q}}_j$ as

$$\omega_{ij} = \mathbf{H}(\mathbf{A}_i, \mathbf{q}_j) \cdot \dot{\mathbf{q}}_j \tag{3.129}$$

where $\mathbf{H}(\mathbf{A}_i, \mathbf{q}_j)$ is a transformation matrix that depends on the orientation of body i and the vector $\mathbf{q}_j$ of relative coordinates, which is defined for each type of joint.
The velocity of body j can be found by differentiating Eq. (3.127) with respect to time to yield

$$\dot{\mathbf{r}}_j = \dot{\mathbf{r}}_i + \dot{\mathbf{s}}_{ij} + \dot{\mathbf{d}}_{ij} \tag{3.130}$$

where $\mathbf{d}_{ij} = \frac{d}{dt}\left(\mathbf{A}_i \mathbf{C}_{ij} \mathbf{d}''_{ij}(\mathbf{q}_j)\right) = \tilde{\omega}_i \mathbf{d}_{ij} + \frac{\partial \mathbf{d}_{ij}}{\partial \mathbf{q}_j} \dot{\mathbf{q}}_j$. The tilde operator (as in $\tilde{\omega}_i$) signifies the skew symmetric vector product operator (Haug, 1989) applied to the vector ω_i.

After simple manipulations, Eqs. (3.128) and (3.130) can be combined in matrix form to yield

$$\begin{bmatrix} \dot{\mathbf{r}}_j + \tilde{\mathbf{r}}_j \omega_j \\ \omega_j \end{bmatrix} = \begin{bmatrix} \dot{\mathbf{r}}_i + \tilde{\mathbf{r}}_i \omega_i \\ \omega_i \end{bmatrix} + \begin{bmatrix} \dfrac{\partial \mathbf{d}_{ij}}{\partial \mathbf{q}_j} + \tilde{\mathbf{r}}_j \mathbf{H}_j \\ \mathbf{H}_j \end{bmatrix} \dot{\mathbf{q}}_j \tag{3.131}$$

Equation (3.131) can be expressed in the so called state-vector notation (Tsai, 1989) as

$$\hat{\mathbf{Y}}_j = \hat{\mathbf{Y}}_i + \mathbf{B}_j \dot{\mathbf{q}}_j \tag{3.132}$$

where the velocity state-vector of body i is defined as

$$\hat{\mathbf{Y}}_i = \begin{bmatrix} \dot{\mathbf{r}}_i + \tilde{\mathbf{r}}_i \omega_i \\ \omega_i \end{bmatrix} \tag{3.133}$$

and the velocity transformation matrix $\mathbf{B}_j$ between bodies i and j is defined as

$$\mathbf{B}_j = \begin{bmatrix} \dfrac{\partial \mathbf{d}_{ij}}{\partial \mathbf{q}_j} + \tilde{\mathbf{r}}_j \mathbf{H}_j \\ \mathbf{H}_j \end{bmatrix} \tag{3.134}$$

For the base body, this matrix is the identity matrix of appropriate dimension. Finally, the acceleration state-vector of body j is obtained by differentiating Eq. (3.132) with respect to time, to obtain

$$\dot{\hat{\mathbf{Y}}}_j = \dot{\hat{\mathbf{Y}}}_i + \mathbf{B}_j \ddot{\mathbf{q}}_j + \dot{\mathbf{B}}_j \dot{\mathbf{q}}_j \equiv \dot{\hat{\mathbf{Y}}}_i + \mathbf{B}_j \ddot{\mathbf{q}}_j + \mathbf{D}_j \tag{3.135}$$

Once the velocity state-vector $\hat{\mathbf{Y}}_j$ and the acceleration state-vector $\dot{\hat{\mathbf{Y}}}_j$ are available, the Cartesian velocity and acceleration, $\mathbf{Y}_j$ and $\dot{\mathbf{Y}}_j$, for body j can be recovered by using the transformations $\mathbf{Y}_j = \mathbf{T}_j \hat{\mathbf{Y}}_j$ and $\dot{\mathbf{Y}}_j = \mathbf{T}_j \dot{\hat{\mathbf{Y}}}_j - \mathbf{R}_j$, with $\mathbf{T}_j$ and $\mathbf{R}_j$ defined as

$$\mathbf{T}_j = \begin{bmatrix} \mathbf{I} & -\tilde{\mathbf{r}}_j \\ \mathbf{0} & \mathbf{I} \end{bmatrix} \qquad \mathbf{R}_j = -\dot{\mathbf{T}}_j \hat{\mathbf{Y}}_j = \begin{bmatrix} \dot{\tilde{\mathbf{r}}}_j \omega_j \\ \mathbf{0} \end{bmatrix} \tag{3.136}$$

In Section 3.4.3.1.2 is outlined the process of generating the equations of motion for an open-loop mechanical system model. In Section 3.4.3.1.3 is presented the case of closed-loop mechanical systems.

3.4.3.1.2 Equations of Motion for a Tree-Structure System

The variational form of the equations of motion can be written in matrix notation as

$$\delta\mathbf{Z}^{\mathrm{T}}\left(\mathbf{M}\dot{\mathbf{Y}}-\mathbf{Q}\right)=0 \qquad (3.137)$$

where the Cartesian acceleration, Cartesian virtual displacement $\delta\mathbf{Z}$, modified mass matrix $\mathbf{M}$, and generalized force are defined as

$$\mathbf{Y}=\begin{bmatrix}\ddot{\mathbf{r}}\\ \omega\end{bmatrix},\qquad \delta\mathbf{Z}=\begin{bmatrix}\delta\mathbf{r}\\ \delta\pi\end{bmatrix},\quad \mathbf{M}=\begin{bmatrix}m\mathbf{I} & -m\tilde{\rho}\\ m\tilde{\rho} & \mathbf{J}\end{bmatrix},\quad \mathbf{Q}=\begin{bmatrix}\mathbf{F}-m\tilde{\omega}\tilde{\omega}\rho\\ \mathbf{n}-\tilde{\omega}\mathbf{J}\omega\end{bmatrix} \quad (3.138)$$

In Eq. (3.138), ρ is the vector from the body reference frame location O' to the body center of gravity location. The corresponding state-vector form of Eq. (3.137) is

$$\delta\hat{\mathbf{Z}}^{\mathrm{T}}\left(\hat{\mathbf{M}}\dot{\hat{\mathbf{Y}}}-\hat{\mathbf{Q}}\right)=0 \qquad (3.139)$$

where, with the notations of Eq. (3.136),

$$\delta\hat{\mathbf{Z}}=\mathbf{T}^{-1}\delta\mathbf{Z} \qquad \hat{\mathbf{M}}=\mathbf{T}^{\mathrm{T}}\mathbf{MT} \qquad \mathbf{Q}=\mathbf{T}^{\mathrm{T}}\left(\mathbf{Q}+\mathbf{MR}\right) \qquad (3.140)$$

For the multibody system of Figure 8, the variational form of the equations of motion assume the form

$$\sum_{i=1}^{p}\delta\hat{\mathbf{Z}}_i^{\mathrm{T}}(\hat{\mathbf{M}}_i\dot{\hat{\mathbf{Y}}}_i-\hat{\mathbf{Q}}_i)+\sum_{i=p+1}^{m}\delta\hat{\mathbf{Z}}_i^{\mathrm{T}}(\hat{\mathbf{M}}_i\dot{\hat{\mathbf{Y}}}_i-\hat{\mathbf{Q}}_i)+\sum_{i=m+1}^{n}\delta\hat{\mathbf{Z}}_i^{\mathrm{T}}(\hat{\mathbf{M}}_i\dot{\hat{\mathbf{Y}}}_i-\hat{\mathbf{Q}}_i) \qquad (3.141)$$

$$\equiv \mathrm{EQ}(1)+\mathrm{EQ}(2)+\mathrm{EQ}(3)=0$$

The state variation $\delta\hat{\mathbf{Z}}_i$ of body i in chain 2 is expressed recursively, in terms of inboard joint relative coordinate variations $\delta\mathbf{q}_k$ and the state variation $\delta\mathbf{Z}_p$ of the junction body, as

$$\delta\hat{\mathbf{Z}}_i=\delta\hat{\mathbf{Z}}_p+\sum_{k=p+1}^{i-1}\mathbf{B}_k\delta\mathbf{q}_k$$

while the acceleration state $\dot{\hat{\mathbf{Y}}}_i$ of body i is recursively expressed as

$$\dot{\hat{\mathbf{Y}}}_i = \dot{\hat{\mathbf{Y}}}_p + \sum_{k=P+1}^{i} (\mathbf{B}_k \ddot{\mathbf{q}}_k + \mathbf{D}_k)$$

with $\mathbf{B}_k$ and $\mathbf{D}_k$ defined in Eqs. (3.134) and (3.135).

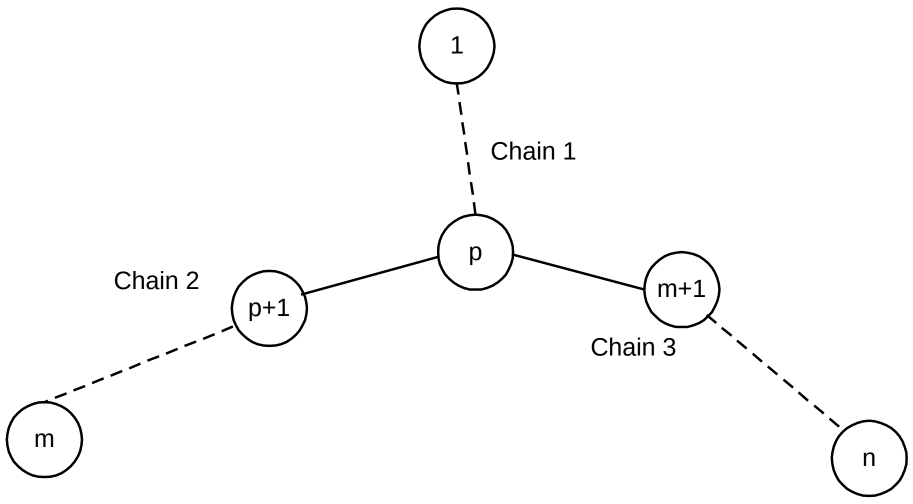


Figure 8. A Tree Structure

Concentrating on chain 2, direct manipulations bring Eq. (3.141) to the form

$$\mathrm{Eq}(1) + \mathrm{Eq}(3) + \delta\hat{\mathbf{Z}}_p^{\mathrm{T}} \left(\mathbf{K}_{p+1} \dot{\hat{\mathbf{Y}}}_p + \sum_{k=p+1}^{m} \mathbf{K}_k \mathbf{B}_k \ddot{\mathbf{q}}_k - (\mathbf{L}_{p+1} - \mathbf{K}_{p+1} \mathbf{D}_{p+1}) \right) \quad (3.142)$$

$$+\delta\mathbf{q}_i^{\mathrm{T}} \left(\mathbf{B}_i^{\mathrm{T}} \mathbf{K}_i \dot{\hat{\mathbf{Y}}}_p + \sum_{k=p+1}^{m} \mathbf{B}_i^{\mathrm{T}} \mathbf{K}_v \mathbf{B}_k \ddot{\mathbf{q}}_k - \mathbf{B}_i^{\mathrm{T}} (\mathbf{L}_i - \mathbf{K}_i \sum_{j=p+1}^{i} \mathbf{D}_j) \right) = 0$$

where the subscript v of $\mathbf{K}_v$ is i, if $i > k$, or k if $i < k$. The composite mass and force matrices $\mathbf{K}_i$ and $\mathbf{L}_i$ are recursively defined as

$$\mathbf{K}_i = \mathbf{K}_{i+1} + \hat{\mathbf{M}}_i \ , \qquad \mathbf{L}_i = \mathbf{L}_{i+1} - \mathbf{K}_{i+1} \mathbf{D}_{i+1} + \hat{\mathbf{Q}}_i \qquad (3.143)$$

The recursion starts from the last body in the chain; in this case the end body m, and proceeds inward along the chain toward the base body. For the end body, the composite mass and force matrices $\mathbf{K}_m$ and $\mathbf{L}_m$ are identical to state-space reduced mass matrix $\hat{\mathbf{M}}_m$ and force matrix $\hat{\mathbf{Q}}_m$, respectively.

For chains 3 and 1, the same steps as for chain 2 are followed, expressing the state variations $\delta\hat{\mathbf{Z}}_i$ of body i in terms of the inboard joint relative coordinate variations $\delta\mathbf{q}_k$ and state variation $\delta\mathbf{Z}_p$ for chain 3 and $\delta\hat{\mathbf{Z}}_1$ for chain 1. State-vector accelerations $\dot{\hat{\mathbf{Y}}}_i$ are generated recursively along the chain toward the base body. For junction body p, the composite mass and force matrices are defined as

$$\begin{aligned}\mathbf{K}_p &= \hat{\mathbf{M}}_p + \mathbf{K}_{p+1} + \mathbf{K}_{m+1} \\ \mathbf{L}_p &= \hat{\mathbf{Q}}_p + (\mathbf{L}_{p+1} - \mathbf{K}_{p+1}\mathbf{D}_{p+1}) + (\mathbf{L}_{m+1} - \mathbf{K}_{m+1}\mathbf{D}_{m+1})\end{aligned} \tag{3.144}$$

After direct manipulation, the variational equations of motion assume the form

$$\begin{aligned}0 &= \delta\mathbf{Z}_1^{\mathrm{T}}\left(\mathbf{K}_1\dot{\hat{\mathbf{Y}}}_1 - \mathbf{L}_1 + \sum_{k=2}^{n}\mathbf{K}_k\mathbf{B}_k\ddot{\mathbf{q}}_k\right) \qquad (3.145) \\ &+\sum_{i=2}^{n}\delta\mathbf{q}_i^{\mathrm{T}}\left(\mathbf{B}_i^{\mathrm{T}}\mathbf{K}_i\dot{\hat{\mathbf{Y}}}_1 + \sum_{k=2}^{n}\mathbf{B}_i^{\mathrm{T}}\mathbf{K}_v\mathbf{B}_k\ddot{\mathbf{q}}_k - \mathbf{B}_i^{\mathrm{T}}\left(\mathbf{L}_i - \mathbf{K}_i\sum_{j=2}^{i}\mathbf{D}_j\right)\right) \\ &+\sum_{i=p+1}^{m}\delta\mathbf{q}_i^{\mathrm{T}}\left(\mathbf{B}_i^{\mathrm{T}}\mathbf{K}_i\dot{\hat{\mathbf{Y}}}_1 + \sum_{k=2}^{p}\mathbf{B}_i^{\mathrm{T}}\mathbf{K}_v\mathbf{B}_k\ddot{\mathbf{q}}_k + \sum_{k=p+1}^{m}\mathbf{B}_i^{\mathrm{T}}\mathbf{K}_v\mathbf{B}_k\ddot{\mathbf{q}}_k - \mathbf{B}_i^{\mathrm{T}}\left(\mathbf{L}_i - \mathbf{K}_i\left(\sum_{j=2}^{P}\mathbf{D}_j + \sum_{j=p+1}^{i}\mathbf{D}_j\right)\right)\right) \\ &+\sum_{i=m+1}^{m}\delta\mathbf{q}_i^{\mathrm{T}}\left(\mathbf{B}_i^{\mathrm{T}}\mathbf{K}_i\dot{\hat{\mathbf{Y}}}_1 + \sum_{k=2}^{p}\mathbf{B}_i^{\mathrm{T}}\mathbf{K}_v\mathbf{B}_k\ddot{\mathbf{q}}_k + \sum_{k=m+1}^{n}\mathbf{B}_i^{\mathrm{T}}\mathbf{K}_v\mathbf{B}_k\ddot{\mathbf{q}}_k - \mathbf{B}_i^{\mathrm{T}}\left(\mathbf{L}_i - \mathbf{K}_i\left(\sum_{j=2}^{p}\mathbf{D}_j + \sum_{j=m+1}^{i}\mathbf{D}_j\right)\right)\right)\end{aligned}$$

Equation (3.145) is obtained by starting from Eq. (3.141) and recursively expressing state-vector virtual displacements and accelerations. Equating to zero expressions multiplying arbitrary virtual displacements in the variational form of the equation of motion produces the differential equations of motion of the open loop mechanism in Figure 8. For any tree structured mechanism, the equations of motion are obtained following the steps outlined above.

3.4.3.1.3 Equations of Motion for a Closed-Loop System

If a mechanism contains closed loops, a number of joints are cut in the process of obtaining a spanning tree. Constraints should therefore be imposed in order to preserve the behavior of the mechanism. Consequently, the virtual displacements in Eq. (3.145) are no longer arbitrary. They are related through the constraint equations. Thus, Eq. (3.145) expressed in matrix form as

$$\delta\mathbf{q}^{\mathrm{T}}(\overline{\mathbf{M}}\ddot{\mathbf{q}}-\overline{\mathbf{Q}})=0 \tag{3.146}$$

should hold for any virtual displacement satisfying $\Phi_{\mathbf{q}}\cdot\delta\mathbf{q}=\mathbf{0}$. It is assumed here that the collection of all cut joints is replaced by an equivalent set of constraint equations that assumes the form $\Phi(\mathbf{q})=\mathbf{0}$.

In this framework, using Lagrange multiplier theorem (Haug, 1989), and taking into account the constraint acceleration equations, the equations of motion for closed-loop mechanical systems assume the form

$$\begin{bmatrix}\overline{\mathbf{M}} & \Phi_{\mathbf{q}}^{\mathrm{T}}\\ \Phi_{\mathbf{q}} & \mathbf{0}\end{bmatrix}\begin{bmatrix}\ddot{\mathbf{q}}\\ \lambda\end{bmatrix}=\begin{bmatrix}\overline{\mathbf{Q}}\\ \tau\end{bmatrix} \tag{3.147}$$

3.4.3.2 Positive Definiteness of CIM

Two properties of the composite inertia matrix $\overline{\mathbf{M}}$ (CIM); the special structure (or sparsity pattern) and the form of the entries, are used to prove its positive definiteness. Both these properties are dictated by the topology of the system, as reflected by the equations of motion. In JR, a graph is associated with each mechanism by considering its bodies and joints as being vertices and connecting edges of the graph, respectively. In addition to the graph concepts introduced by Tsai (1989), a few others are defined in what follows.

Two concepts introduced in Section 3.4.3.1.1 induce a direction in the spanning tree that is obtained after cutting a required number of joints. These concepts are the inboard-outboard relationship between pairs of neighbor bodies, along with the fact that each body in the spanning tree has one and only one inboard body, whereas it can have an arbitrary number of outboard bodies. In the following, the *directed* spanning tree associated with the cut joint mechanism is referred as the *spanning tree*, unless otherwise stated.

Body j is a descendent of body i if there is a path in the spanning tree from body i to body j. *Family* i, denoted by $\mathbf{F}(i)$, is the set of all descendant of body i. If body i is added to $\mathbf{F}(i)$, then the new collection of bodies is denoted by $\mathbf{F}[i]$. The sequence of bodies on the unique path starting from body i and ending at body j is called a *subchain*, denoted $\mathbf{d}[i, j]$. If body i is left out of this sequence, the subchain is denoted by $\mathbf{d}(i, j]$. The subchains $\mathbf{d}[i, j)$ and $\mathbf{d}(i, j)$ are defined similarly. The subchain starting at the base body and ending at body i is denoted by $\mathbf{d}[i]$. Note that a subchain inherits its order from the spanning tree, while a family does not. By concatenating subchain $\mathbf{d}[i]$ to family $\mathbf{F}(i)$, the *subtree* $\mathbf{c}(i)$ is obtained, which will be ordered from the base body outward to body i. If b is the base body, the subtree $\mathbf{c}(b)$ represents the entire spanning tree and, for convenience, it is denoted simply by $\mathbf{S}$. Finally, body i is a *leaf* of the spanning tree if it has no outboard bodies. The results below are based on the work of Negrut, Serban, and Potra (1998).

Lemma 1. Let $\mathbf{x}_w$ be a matrix or vector quantity associated with body w and $\mathbf{y}_s$ be a matrix or vector quantity associated to body s, such that the multiplication $\mathbf{x}_w\mathbf{y}_s$ is well defined. Then,

$$\sum_{s\in\mathbf{F}(r)}\sum_{w\in\mathbf{F}[s]}\mathbf{x}_w\mathbf{y}_s=\sum_{w\in\mathbf{F}(r)}\sum_{s\in\mathbf{d}(r,w]}\mathbf{x}_w\mathbf{y}_s$$

where r is an arbitrary body of the spanning tree.

The right-side term is actually a reordering of the summation in the left-side. Based on **Lemma 1**, taking r to be a fictitious inboard body of the base body, the following result is obtained.

Corollary 1. With $\mathbf{y}_s^{\mathrm{T}}$ and $\mathbf{x}_w$ vectors of appropriate dimension associated with bodies s and w, respectively,

$$\sum_{s\in\mathbf{S}}\sum_{w\in\mathbf{F}[s]}\mathbf{y}_s^{\mathrm{T}}\mathbf{x}_w=\sum_{w\in\mathbf{S}}\sum_{s\in\mathbf{d}[w]}\mathbf{y}_s^{\mathrm{T}}\mathbf{x}_w$$

Generally, the unknowns related to body u occupy position i in the vector of unknowns. A permutation $\mathbf{p}$ is defined such that for each body it gives its position in the global vector of unknowns, $\mathbf{p}(u)=i$. If u and v are two bodies of the spanning tree, with $i=\mathbf{p}(u)$ and $j=\mathbf{p}(v)$, the block entry (i,j) of CIM of Eq. (3.147), is given as

$$\overline{\mathbf{M}}[i,j]=\begin{cases}\mathbf{B}_u^{\mathrm{T}}\mathbf{K}_u\mathbf{B}_v & \text{if } v\in\mathbf{d}[u]\\ \mathbf{B}_u^{\mathrm{T}}\mathbf{K}_v\mathbf{B}_v & \text{if } v\in\mathbf{F}(\mathrm{u})\\ \mathbf{0} & \text{if } v\notin\mathbf{c}(u)\end{cases}\qquad(3.148)$$

The matrix $\mathbf{B}$ is defined in Eq. (3.134), while the $\mathbf{K}$ matrix is defined in Eqs. (3.143) and (3.144). Rather than dealing with scalar entries, $\overline{\mathbf{M}}$ is defined in terms of blocks. The number of rows and columns of block $\overline{\mathbf{M}}[i,j]$ is equal to the dimension of $\mathbf{q}_i$ and $\mathbf{q}_j$, respectively. Generally, if N is the number of bodies in the system and n_i is the dimension of the generalized coordinate vector $\mathbf{q}_i$, $\mathbf{q}_i \in \Re^{n_i}$, then $\overline{\mathbf{M}} \in \Re^{n\times n}$, with $n = \sum_{i=1}^{N} n_i$.

Proposition 2. The composite inertia matrix $\overline{\mathbf{M}}$ defined in Eq. (3.148) is positive definite.

Defining $\mathbf{v} = [\mathbf{v}_1^{\mathrm{T}}, \mathbf{v}_2^{\mathrm{T}}, \ldots, \mathbf{v}_N^{\mathrm{T}}]^{\mathrm{T}}$, it is to be proved that $(1/2)\mathbf{v}^{\mathrm{T}}\overline{\mathbf{M}}\mathbf{v} \geq 0$ and equality is obtained only if $\mathbf{v} = \mathbf{0}$. Considering the vector $\mathbf{z} = \overline{\mathbf{M}}\mathbf{v}$ and denoting $\mathbf{y}_k \equiv \mathbf{B}_k \mathbf{v}_k$,

$$\begin{aligned} \mathbf{z}(r) &= \sum_{s\in\mathbf{d}[r]} \mathbf{B}_r^{\mathrm{T}}\mathbf{K}_r\mathbf{y}_s + \sum_{s\in\mathbf{F}(r)} \mathbf{B}_r^{\mathrm{T}}\mathbf{K}_s\mathbf{y}_s \\ &= \sum_{s\in\mathbf{d}[r]} \mathbf{B}_r^{\mathrm{T}} \sum_{w\in\mathbf{F}[r]} \hat{\mathbf{M}}_w\mathbf{y}_s + \sum_{s\in\mathbf{F}(r)} \mathbf{B}_r^{\mathrm{T}} \sum_{w\in\mathbf{F}[s]} \hat{\mathbf{M}}_w\mathbf{y}_s \end{aligned}$$

Using the result of **Lemma 1** and defining $\mathbf{u}_w \equiv \sum_{s\in\mathbf{d}[w]} \mathbf{y}_s$,

$$\begin{aligned} \mathbf{z}(r) &= \sum_{w\in\mathbf{F}[r]} \sum_{s\in\mathbf{d}[r]} \mathbf{B}_r^{\mathrm{T}}\hat{\mathbf{M}}_w\mathbf{y}_s + \sum_{w\in\mathbf{F}(r)} \sum_{s\in\mathbf{d}(r,w]} \mathbf{B}_r^{\mathrm{T}}\hat{\mathbf{M}}_w\mathbf{y}_s \\ &= \sum_{s\in\mathbf{d}[r]} \mathbf{B}_r^{\mathrm{T}}\hat{\mathbf{M}}_r\mathbf{y}_s + \sum_{w\in\mathbf{F}(r)} \sum_{s\in\mathbf{d}[w]} \mathbf{B}_r^{\mathrm{T}}\hat{\mathbf{M}}_w\mathbf{y}_s = \sum_{w\in\mathbf{F}[r]} \sum_{s\in\mathbf{d}[w]} \mathbf{B}_r^{\mathrm{T}}\hat{\mathbf{M}}_w\mathbf{y}_s \\ &= \sum_{w\in\mathbf{F}[r]} \mathbf{B}_r^{\mathrm{T}}\hat{\mathbf{M}}_w \sum_{s\in\mathbf{d}[w]} \mathbf{y}_s = \mathbf{B}_r^{\mathrm{T}} \sum_{w\in\mathbf{F}[r]} \hat{\mathbf{M}}_w\mathbf{u}_w \end{aligned}$$

Finally, using the result of **Corollary 1**,

$$\frac{1}{2}\mathbf{v}^{\mathrm{T}}\overline{\mathbf{M}}\mathbf{v} = \frac{1}{2}\sum_{r\in\mathbf{S}} \mathbf{y}_r^{\mathrm{T}} \sum_{w\in\mathbf{F}[r]} \hat{\mathbf{M}}_w\mathbf{u}_w = \frac{1}{2}\sum_{r\in\mathbf{S}} \sum_{w\in\mathbf{F}[r]} \mathbf{y}_r^{\mathrm{T}}\hat{\mathbf{M}}_w\mathbf{u}_w \tag{3.149}$$

$$= \frac{1}{2}\sum_{w\in\mathbf{S}} \sum_{r\in\mathbf{d}[w]} \mathbf{y}_r^{\mathrm{T}}\hat{\mathbf{M}}_w\mathbf{u}_w = \frac{1}{2}\sum_{w\in\mathbf{S}} \mathbf{u}_w^{\mathrm{T}}\hat{\mathbf{M}}_w\mathbf{u}_w \equiv \frac{1}{2}\sum_{w\in\mathbf{S}} \left\|\mathbf{u}_w\right\|_{\hat{\mathbf{M}}_w}^2$$

In Eq. (3.149), $\|\cdot\|_{\hat{\mathbf{M}}_w}$ defines a norm, since $\hat{\mathbf{M}}_w$ is positive definite. To see this, first note that the state-vector-reduced matrix assumes the form

$$\hat{\mathbf{M}}_w = \mathbf{T}^{\mathrm{T}}\mathbf{M}_w\mathbf{T} = \mathbf{T}^{\mathrm{T}}\mathbf{H}^{\mathrm{T}}\mathbf{M}_w^c\mathbf{H}\mathbf{T}$$

with $\mathbf{M}$ and $\mathbf{T}$ defined as in Eqs. (3.138) and (3.136), and $\mathbf{M}_c$ and $\mathbf{H}$ given by

$$\mathbf{M}^c = \begin{bmatrix} m\mathbf{I} & \mathbf{0} \\ \mathbf{0} & \mathbf{J}^c \end{bmatrix} \qquad \mathbf{H} = \begin{bmatrix} \mathbf{I} & \mathbf{0} \\ \rho & \mathbf{I} \end{bmatrix} \tag{3.150}$$

where $\mathbf{J}^c$ is the body inertia matrix with respect to a reference frame located at the centroid of the body and parallel to the body reference frame. Therefore, $\mathbf{J}^c$ is positive definite. Since matrices $\mathbf{H}$ and $\mathbf{T}$ are nonsingular, and $\mathbf{M}^c$ is positive definite, it results that $\hat{\mathbf{M}}_w$ is positive definite.

The last sum in Eq. (3.149) is zero only when $\mathbf{u}_w = \mathbf{0}$, for all $w \in \mathbf{S}$. If this is the case, then $\mathbf{y}_w = \mathbf{0}$, for all $w \in \mathbf{S}$. Equivalently, $\mathbf{B}_w \mathbf{v}_w = \mathbf{0}$, for all $w \in \mathbf{S}$. Since proper modeling in JR requires $\mathbf{B}_w$ to be of full column rank, it is concluded that $\mathbf{v}_w = \mathbf{0}$, for all $w \in \mathbf{S}$. This completes the proof. ■

Taking

$$\mathbf{v} = [\mathbf{v}_1^{\mathrm{T}}, \mathbf{v}_2^{\mathrm{T}}, \ldots, \mathbf{v}_N^{\mathrm{T}}]^{\mathrm{T}} \equiv [\dot{\mathbf{q}}_1^{\mathrm{T}}, \dot{\mathbf{q}}_2^{\mathrm{T}}, \ldots, \dot{\mathbf{q}}_N^{\mathrm{T}}]^{\mathrm{T}} \tag{3.151}$$

define the quadratic form $(1/2)\mathbf{v}^{\mathrm{T}}\overline{\mathbf{M}}\mathbf{v}$ as representing the kinetic energy of the tree-structured mechanism, in the state-vector space. Note that $\mathbf{u}_w$, as defined above, represents the state-vector velocity of body w; i.e., the vector previously denoted by $\hat{\mathbf{Y}}_w$. Hence, Eq. (3.149) implies that the kinetic energy of the system in the state-vector space is equal to the sum of the kinetic energy in the state-vector space of all bodies in the system. Based on Eq. (3.140) and the fact that $\mathbf{Y}_w = \mathbf{T}_w \hat{\mathbf{Y}}_w$, the following holds.

Corollary 2. The kinetic energy of each body in the system is the same as its kinetic energy in the state-vector space. Therefore the kinetic energy of the tree-structured mechanism has the same property.

3.4.3.3 Factoring the Augmented Matrix

In the most general case of a mechanism containing closed loops, the augmented matrix that must be factored is the coefficient matrix of the linear system of Eq. (3.147), denoted here by $\mathbf{A}$. In the case of a tree-structured mechanism, the augmented matrix is identical to the composite inertia matrix. Block Cholesky factorization of the matrix $\mathbf{A}$ will result in a $\mathbf{LL}^{\mathrm{T}}$ decomposition, which is detailed below.

Cholesky factorization is not directly applicable when closed loops are present in the mechanical system, since the augmented matrix is not positive definite, due to the presence of the constrain Jacobian. The augmented matrix will therefore be factored as follows:

$$\mathbf{A} \equiv \begin{bmatrix} \overline{\mathbf{M}} & \Phi_{\mathbf{q}}^{\mathrm{T}} \\ \Phi_{\mathbf{q}} & \mathbf{0} \end{bmatrix} = \begin{bmatrix} \mathbf{L} & \mathbf{0} \\ \mathbf{T}^{\mathrm{T}} & \mathbf{I}_m \end{bmatrix} \begin{bmatrix} \mathbf{I}_n & \mathbf{0} \\ \mathbf{0} & -\mathbf{T}^{\mathrm{T}}\mathbf{T} \end{bmatrix} \begin{bmatrix} \mathbf{L}^{\mathrm{T}} & \mathbf{T} \\ \mathbf{0} & \mathbf{I}_m \end{bmatrix} \tag{3.152}$$

where $\mathbf{L}$ is obtained from the Cholesky factorization of $\overline{\mathbf{M}} = \mathbf{LL}^{\mathrm{T}}$, and $\mathbf{T} \in \Re^{n\times m}$ is the solution of the matrix equation

$$\mathbf{LT} = \Phi_{\mathbf{q}}^{\mathrm{T}} \tag{3.153}$$

The matrix $\mathbf{T}^{\mathrm{T}}\mathbf{T}$ is positive definite since the Jacobian of the position kinematic constraint equations is assumed to have full row rank.

3.4.3.3.1 Factoring Composite Inertia Matrix

Contrary to general perception, sparsity in the CIM is not lost, and furthermore it is structured. Qualitatively, the sparsity is dictated by the topology of the mechanism, and in the case of closed loop mechanisms also by the cut joints. The major problem with direct algorithms for sparse systems is that, to a certain extent, sparsity is lost during factorization. The proposed algorithm preserves the sparsity pattern of CIM; i.e., no fill-in occurs during the Cholesky factorization. For this to happen, a renumbering of the bodies of the model may be necessary. In a sense, the strategy required to exploit sparsity in the JR formulation is complementary to the one introduced for the Cartesian representation. While in the latter case the joints were renumbered, in JR formulation the bodies are renumbered to yield a certain order of the unknowns. If the vector of unknowns is of the form $\ddot{\mathbf{q}} = [\ddot{\mathbf{q}}_1^{\mathrm{T}}, \ddot{\mathbf{q}}_2^{\mathrm{T}}, \ldots, \ddot{\mathbf{q}}_N^{\mathrm{T}}]^{\mathrm{T}}$, a block permutation defined by a block permutation matrix $\mathbf{P}$ is applied to $\ddot{\mathbf{q}}$, to obtain

$$\ddot{\mathbf{q}}_{new} = \mathbf{P}\ddot{\mathbf{q}} = [\ddot{\mathbf{q}}_{\mathbf{b}(1)}^{\mathrm{T}}, \ddot{\mathbf{q}}_{\mathbf{b}(2)}^{\mathrm{T}}, \ldots \ddot{\mathbf{q}}_{\mathbf{b}(N)}^{\mathrm{T}}]^{\mathrm{T}} \tag{3.154}$$

where $\mathbf{b}$ is a permutation such that, if body u is assigned via the permutation $\mathbf{P}$ position i in $\ddot{\mathbf{q}}_{new}$, then $\mathbf{b}(i) = u$. Note that $\mathbf{b}$ is the inverse of the permutation $\mathbf{p}$ introduced before defining the entries of the composite inertia matrix in Eq. (3.148).

Lemma 2. Let u and v be two bodies in the spanning tree associated with a mechanical system. No fill-in occurs during the classical block Cholesky factorization of CIM if, for $u \in \mathbf{F}(v)$, then $\mathbf{p}(u) < \mathbf{p}(v)$

To prove this result the symmetry and the block structure of the composite inertia matrix are used. Matrix $\overline{\mathbf{M}}$ is factored using a block-oriented Cholesky factorization and it is to be shown that blockwise, $\mathbf{L}[k,j] = \mathbf{0}$, if $\overline{\mathbf{M}}[k,j] = \mathbf{0}$. $\mathbf{L}[1,1]$ is determined from $\mathbf{L}[1,1]\mathbf{L}^{\mathrm{T}}[1,1] = \overline{\mathbf{M}}[1,1]$ and, because of the positive definiteness of $\overline{\mathbf{M}}$, it is not identically zero. The matrix $\mathbf{L}[k,1]$ is the solution of $\mathbf{L}[1,1]\mathbf{L}^{\mathrm{T}}[k,1] = \overline{\mathbf{M}}[k,1]$, $1 < k < N$, and clearly $\mathbf{L}[k,1] = \mathbf{0}$ when $\overline{\mathbf{M}}[k,1] = \mathbf{0}$.

Suppose that, up to column $j-1$, the conclusion of **Lemma 2** holds. The positive definiteness of $\overline{\mathbf{M}}$ implies that $\mathbf{L}[j,j]$, obtained from $\mathbf{L}[j,j]\mathbf{L}^{\mathrm{T}}[j,j]$ $= \overline{\mathbf{M}}[j,j] - \sum_{i=1}^{j-1}\mathbf{L}[j,i]\mathbf{L}^{\mathrm{T}}[j,i]$, is not identically zero. For $k > j$, $\mathbf{L}[k,j]$ is the solution of

$$\mathbf{L}[j,j]\mathbf{L}^{\mathrm{T}}[k,j] = \overline{\mathbf{M}}[k,j] - \sum_{i=1}^{j-1} \mathbf{L}[j,i]\mathbf{L}^{\mathrm{T}}[k,i] \tag{3.155}$$

Let $\overline{\mathbf{M}}[k,j] = \mathbf{0}$, and suppose there is an i, $i < j < k$, such that $\mathbf{L}[j,i]\mathbf{L}^{\mathrm{T}}[k,i] \neq \mathbf{0}$. Then $\overline{\mathbf{M}}[i,j] \neq \mathbf{0}$ and $\overline{\mathbf{M}}[i,k] \neq \mathbf{0}$. Let bodies u, v, and w be such that $\mathbf{p}(u) = i$, $\mathbf{p}(v) = j$, and $\mathbf{p}(w) = k$. Because of the way the precedence is defined in the vector of unknowns, this implies that $v \in \mathbf{d}[u]$ and $w \in \mathbf{d}[u]$. However, since $j < k$, $w \in \mathbf{d}[v]$. Therefore, $\overline{\mathbf{M}}[k,j] \neq \mathbf{0}$, which is a contradiction. Consequently, the right-side of Eq. (3.155) is zero. Then, $\mathbf{L}[j,i]\mathbf{L}^{\mathrm{T}}[k,i] = \mathbf{0}$ and $\mathbf{L}[k,j] = \mathbf{0}$. This completes the proof. ■

The reordering induced by the permutation array $\mathbf{p}$ is not unique, and developing one is straightforward. A good initial joint numbering based on the preceding lemma, ensures no fill-in factorization of the CIM for the entire simulation.

3.4.3.3.2 The Closed-loop Case

In the case of a closed-loop mechanical system, M joints connecting bodies in the system are cut to obtain a spanning tree of the mechanism. A set of constraint equations $\mathbf{\Phi}^{(1)} = \mathbf{\Phi}^{(2)} = \ldots = \mathbf{\Phi}^{(M)} = \mathbf{0}$, is imposed to account for the cut joints. In Section 3.4.3.3.2, the collection of constraint equations was denoted by $\mathbf{\Phi} = [\mathbf{\Phi}^{(1)^{\mathrm{T}}}, \mathbf{\Phi}^{(2)^{\mathrm{T}}}, \ldots, \mathbf{\Phi}^{(M)^{\mathrm{T}}}]^{\mathrm{T}}$, with $m = \sum_{i=1}^{M} m_i$ and $\mathbf{\Phi}^{(i)} \in \Re^{m_i}$. With a columnwise partition of $\mathbf{T}$ in Eq. (3.153), the component $\mathbf{T}^{(j)} \in \Re^{n \times m_j}$ is obtained solving $\mathbf{L}\mathbf{T}^{(j)} = \mathbf{\Phi}_{\mathbf{q}}^{(j)^{\mathrm{T}}}$, $1 \leq j \leq M$. With the precedence in the vector of unknowns induced by **Lemma 2**, the following result singles out the zeros of the matrix $\mathbf{T}$.

Lemma 3. Let constraint j account for the cut joint between bodies k and l, and let $i = \mathbf{p}(u)$, with $u \notin \mathbf{d}[k] \cup \mathbf{d}[l]$. Then $\mathbf{T}^{(j)}[i] = \mathbf{0}$.

For proof, let i be the smallest integer that satisfies the hypothesis of the lemma and yet violates the conclusion. Row i of $\mathbf{L}\mathbf{T}^{(j)} = \Phi_{\mathbf{q}}^{(j)\mathrm{T}}$ is given by

$$\sum_{v=1}^{i-1} \mathbf{L}[i,v]\mathbf{T}^{(j)}[v] + \mathbf{L}[i,i]\mathbf{T}^{(j)}[i] = \Phi_{\mathbf{q}_u}^{(j)}$$

Since $u \notin \mathbf{d}[k] \cup \mathbf{d}[l]$, $\Phi_{\mathbf{q}_u}^{(j)} = \mathbf{0}$ and, therefore, i cannot be 1. Suppose there exists v, $1 \leq v < i$, such that $\mathbf{L}[i,v]\mathbf{T}^{(j)}[v] \neq \mathbf{0}$. With the precedence induced by **Lemma 2** and with w defined by $v = \mathbf{b}(w)$, $w \in \mathbf{F}(u)$. Therefore, $u \in \mathbf{d}[w]$. Finally, since $\mathbf{T}^{(j)}[v] \neq \mathbf{0}$ and $v < i$, $w \in \mathbf{d}[k] \cup \mathbf{d}[l]$. However, $u \in \mathbf{d}[k] \cup \mathbf{d}[l]$, which is a contradiction. This completes the proof. ■

3.4.3.3.3 Algorithm for Factorization of Composite Inertia Matrix

The proposed algorithm is based on the decomposition in Eq. (3.152). The ordering of elements of the unknown vector induced by **Lemma 2** is assumed.

Algorithm 4

```
Step 0      Set $\mathbf{y}_n = \mathbf{0}$, $\mathbf{y}_n \in \Re^n$
Step 1      Factor $\overline{\mathbf{M}} = \mathbf{L}\,\mathbf{L}^{\mathrm{T}}$
Step 2      Solve $\mathbf{L}\mathbf{z}_n = \mathbf{Q}^{\mathrm{A}}$, $\mathbf{z}_n \in \Re^n$
            IF (closed-loop) then:
Step 3.1        Solve $\mathbf{L}\,\mathbf{T} = \Phi_{\mathbf{q}}^{\mathrm{T}}$, $\mathbf{T} \in \Re^{n\times m}$
Step 3.2        Set $\mathbf{z}_m = \mathbf{T}^{\mathrm{T}}\mathbf{z}_n - \tau$, $\mathbf{z}_m \in \Re^m$
Step 3.3        Compute $\mathbf{T}^{\mathrm{T}}\,\mathbf{T}$
Step 3.4        Solve $\mathbf{T}^{\mathrm{T}}\,\mathbf{T}\lambda = \mathbf{z}_m$, $\lambda \in \Re^m$
Step 3.5        Set $\mathbf{y}_n = \mathbf{T}\lambda$
```

```
                    ENDIF
Step 4              Solve $\mathbf{L}^{\mathrm{T}}\ddot{\mathbf{q}} = \mathbf{z}_n - \mathbf{y}_n$
```

During Step 1, CIM is factored using block Cholesky with no fill-in. Therefore, the sparsity pattern for **L** is known beforehand. Solving for $\mathbf{z}_n$ requires only forward substitution. Without taking sparsity into account in Step 2, row i of the linear system $\mathbf{L}\mathbf{z}_n = \mathbf{Q}^{\mathrm{A}}$ would be

$$\sum_{v=1}^{i-1} \mathbf{L}[i,v]\mathbf{z}_n[v] + \mathbf{L}[i,i]\mathbf{z}_n[i] = \mathbf{Q}^{\mathrm{A}}[i]$$

A reduced number of operations results if when solving for $\mathbf{z}_n[i]$, in the light of **Lemma 2**, row i is equivalently expressed as

$$\sum_{k \in \mathbf{F}(\mathbf{p}(i))} \mathbf{L}[i,\mathbf{b}(k)]\mathbf{z}_n[\mathbf{b}(k)] + \mathbf{L}[i,i]\mathbf{z}_n[i] = \mathbf{Q}^{\mathrm{A}}[i]$$

The same result holds when backward substitution is used to retrieve the accelerations $\ddot{\mathbf{q}}$ in Step 4.

Using the result of **Lemma 3**, further advantage can be taken of the problem structure when computing the matrix $\mathbf{T}$ during Step 3.1. Some entries of this matrix are known beforehand to be zero and need not be computed. The coefficient matrix in $\mathbf{T}^{\mathrm{T}}\,\mathbf{T}\lambda = \mathbf{z}_m$ is dense and positive definite, and it is factored via Cholesky decomposition.

In general, it is not possible to give an operation count for the proposed algorithm. The number of operations will depend on the topology of the particular mechanical system model being considered. Section 3.4.3.5 presents a comparison in terms of number of operations and CPU time of several alternatives to compute generalized accelerations and Lagrange multipliers when dealing with a vehicle model.

3.4.3.4 Taking Advantage of Parallelism

In this Section, it is assumed that the precedence in the vector of unknowns is as in **Lemma 2**. Let j be an integer such that $1 \le j \le N$, where N is the number of bodies in the mechanical system model. Define

$$\mathbf{D}(j) \equiv \{i \,|\, 1 \le i < j \text{ and } \mathbf{b}(\mathrm{i}) \in \mathbf{F}(\mathbf{b}(\mathrm{j}))\}$$

$$\mathbf{U}(j) \equiv \{i \,|\, 1 \le i \le N \text{ and } \mathbf{b}(\mathrm{i}) \in \mathbf{d}[\mathbf{b}(\mathrm{j})]\}$$

Lemma 4. During Cholesky factorization of the CIM, for any j, $1 \le j \le N$ and $k \in \mathbf{U}(j)$, $\mathbf{L}[k, j]$ can be computed, provided that for each $i \in \mathbf{D}(j)$, $\mathbf{L}[l,i]$ is available for $l \in \mathbf{U}(i)$.

For proof, let v be the body in the system for which $\mathbf{p}(v) = j$. It is first shown that the conclusion holds if j corresponds to a leaf v. Then it is shown that it holds for any j.

One stage of the block Cholesky algorithm is defined as follows:

$$\mathbf{L}[j,j]\mathbf{L}^{\mathrm{T}}[j,j] = \overline{\mathbf{M}}[j,j] - \sum_{i=1}^{j-1} \mathbf{L}[j,i]\mathbf{L}^{\mathrm{T}}[j,i] \tag{3.156}$$

$$\mathbf{L}[j,j]\mathbf{L}^{\mathrm{T}}[k,j] = \overline{\mathbf{M}}[k,j] - \sum_{i=1}^{j-1} \mathbf{L}[j,i]\mathbf{L}^{\mathrm{T}}[k,i] \tag{3.157}$$

Equation (3.156) is solved for $\mathbf{L}[j,j]$, and then $\mathbf{L}[k,j]$, $j < k \le N$, is computed from Eq. (3.157).

Let j in $\mathbf{p}(v) = j$ be such that v is a leaf. Then, $\mathbf{D}(j) = \{\varnothing\}$. For $1 \le i < j$, let u be such that $\mathbf{p}(u) = i$. Then, $u \notin \mathbf{F}(v)$, because otherwise $\mathbf{D}(j) \ne \{\varnothing\}$. Likewise, $u \notin \mathbf{d}[v]$, since $\mathbf{p}(u) < \mathbf{p}(v)$, and this would violate the precedence induced by **Lemma 2**. Therefore, $u \notin \mathbf{c}[v]$, and with the definition of $\overline{\mathbf{M}}[j,i]$ given in Eq. (3.148), along with the result of **Lemma 2**, $\mathbf{L}[j,j]$ is the solution of $\mathbf{L}[j,j]\mathbf{L}^{\mathrm{T}}[j,j] = \overline{\mathbf{M}}[j,j]$. Furthermore, since $\mathbf{L}[j,i] = \mathbf{0}$ for $1 \le i < j$, $\mathbf{L}[k,j]$ is the solution of $\mathbf{L}[j,j]\mathbf{L}^{\mathrm{T}}[k,j] = \overline{\mathbf{M}}[k,j]$,

$j < k \leq N$. Thus, for the case of j corresponding to a leaf of a tree, $\mathbf{L}[k, j]$ can be computed for $k \in \mathbf{U}(j)$.

For the last step of the proof, let j be such that $\mathbf{D}(j) \neq \{\varnothing\}$; i.e., v in $\mathbf{p}(v) = j$ is not a leaf of the tree. Assume that for each $i \in \mathbf{D}(j)$, $\mathbf{L}[l,i]$ is available for $l \in \mathbf{U}(i)$. It is to be shown that $\mathbf{L}[k, j]$ can be computed for $k \in \mathbf{U}(j)$. With i being the summation index in Eq. (3.156), let u be such that $\mathbf{p}(u) = i$. There are two alternatives, relative to the position of u in the spanning tree of the mechanism; $u \notin \mathbf{F}(v)$, or $u \in \mathbf{F}(v)$. In the first case, $u \notin \mathbf{d}[v]$. Otherwise, the precedence induced by **Lemma 2** is violated. Then, $u \notin \mathbf{c}[v]$ and again, with the definition of $\overline{\mathbf{M}}[j,i]$ along with the result of **Lemma 2**, $\mathbf{L}[j,i] = \mathbf{0}$. On the other hand, if $u \in \mathbf{F}(v)$, then $j \in \mathbf{U}(i)$. Therefore, $\mathbf{L}[j,i]$ is known. Consequently, one can evaluate the summation in Eq. (3.156) and obtain the value $\mathbf{L}[j, j]$.

Finally, to compute $\mathbf{L}[k, j]$ for $k \in \mathbf{U}(j)$, it is necessary to evaluate the summation in Eq. (3.157). It was shown above that $\mathbf{L}[j,i]$ is either identically zero (in this case $\mathbf{L}[k,i]$ need not be evaluated) or known (when $j \in \mathbf{U}(i)$). In the latter situation, the value of $\mathbf{L}[k,i]$ is needed. If $j \in \mathbf{U}(i)$, since $k \in \mathbf{U}(j)$, $k \in \mathbf{U}(i)$ as well. Consequently, $\mathbf{L}[k,i]$ is known and the summation can be evaluated. This completes the proof. ■

When the Cholesky factorization progresses through the sequence described by Eqs (3.156) and (3.157), the process moves columnwise. Each column is filled, starting from the diagonal element and proceeding down to the last row of the matrix. **Lemma 4** states that, based on a certain amount of information, some of the entries of column j, namely $\mathbf{L}[k, j]$ with $k \in \mathbf{U}(j)$, can be computed. The other entries are identically zero, since when $k \notin \mathbf{U}(j)$ and $j < k \leq N$, $\mathbf{L}[k, j] = \mathbf{0}$.

The results in this Section give a practical way in which the CIM can be factored; column j can be computed once the columns $i = \mathbf{p}(u)$ corresponding to all $u \in \mathbf{F}(v)$ have been computed. In other words, the factorization progresses independently from the tree-end bodies (leafs) toward the root of the tree. These observations yield the following corollary, based on the result of **Lemma 4**.

Corollary 3. Let u and v be two bodies in the spanning tree associated with a mechanical system, with $u \notin \mathbf{F}(v)$, and $v \notin \mathbf{F}(u)$. Then, the columns $k = \mathbf{p}(w)$ for $w \in \mathbf{F}(u)$ and $l = \mathbf{p}(t)$ for $t \in \mathbf{F}(v)$ of the matrix $\mathbf{L}$ in the Cholesky factorization of CIM can be computed in parallel.

Note that there is another stage in *Algorithm 4* that can be easily parallelized, since any forward or backward substitution involving the matrix $\mathbf{L}$ can be done in parallel.

3.4.3.5 Numerical Experiments

Numerical experiments are performed on a 14-body model of the Army's High Mobility Multipurpose Wheeled Vehicle (HMMWV) (Serban, Negrut, Haug, 1998). The topological graph of the mechanical system model is shown in Figure 9. The bodies of the model are represented as the vertices of the graph, while the joints are the connecting edges. Here, R stands for revolute joint, T for translational joint, S for spherical joint, and D for distance constraint. The bodies used to model the vehicle are listed in Table 5.

For this problem, the coefficient matrix in Eq. (3.147) has dimension 43. A number of 16 constraint equations account for the cut joints; the vector of generalized coordinates is of dimension 27. The vehicle model has 11 degrees of freedom. The constraints marked with an arrow in Figure 9 are cut to obtain the spanning tree in Figure 10(a).

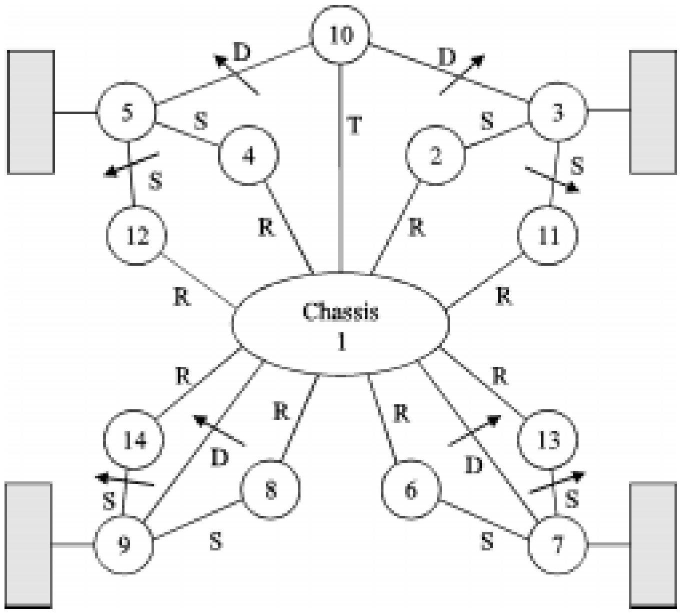


Figure 9. HMMWV14 Body Model: Topology Graph

Table 5. HMMWV14 Model - Component Bodies

Vertex	Body	Vertex	Body
1	Chassis	8	Left rear upper control arm
2	Right front upper control arm	9	Left rear wheel spindle
3	Right front wheel spindle	10	Rack
4	Left front upper control arm	11	Right front lower control arm
5	Left front wheel spindle	12	Left front lower control arm
6	Right rear upper control arm	13	Right rear lower control arm
7	Right rear wheel spindle	14	Left rear lower control arm

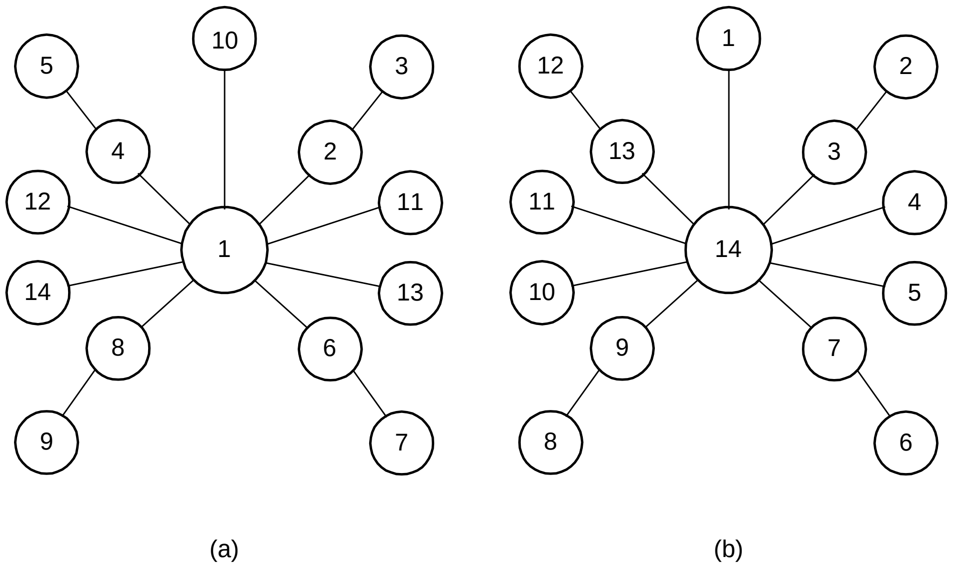


Figure 10. Spanning Tree – HMMWV14

Four methods for the solution of the augmented linear system in Eq. (3.147) are compared. The comparison is made in terms of CPU time and, for Gaussian elimination and the proposed algorithm, also in terms of number of operations.
The first method analyzed is based on Gaussian elimination, and denoted by *Gauss.* The method *Symmetric* is based on a $\mathbf{PAP}^{\mathrm{T}} = \mathbf{LDL}^{\mathrm{T}}$ decomposition of the symmetric augmented matrix $\mathbf{A}$, where $\mathbf{D}$ is a diagonal matrix with blocks of dimension 1×1 or 2×2. The method *Harwell* solves the augmented system by using linear algebra subroutines from the Harwell library. The fourth method considered is the one introduced here, denoted by *Alg-S*, the "S" standing for sparse.

In the NADS Vehicle Dynamics Software (1995), the strategy for solving the augmented system in Eq. (3.147) is based on Gaussian elimination. Table 6 contains the number of operations for Gaussian elimination for this model. In this table, **Fact** stands for factorization, **FS** for forward substitution, and **BS** for back substitution. The number of additions **A**, multiplications **M**, divisions **D,** and square roots **SQ** is counted at each stage of *Gauss*.

Table 6. Operation Count for Gaussian Elimination

Gauss	**Fact**	**FS**	**BS**	**Total**
A	25585	903	903	27391
M	25585	903	903	27391
D	903	0	43	946
SQ	0	0	0	0

Table 7 provides operation counts for algorithm *Alg-S,* following the steps of *Algorithm 4*. This implementation of the proposed algorithm uses topology information to take advantage of sparsity. In order to preserve the sparsity pattern of CIM, in the light of **Lemma 2**, the bodies of the system were renumbered as shown in Figure 10 (b).

The number of additions, multiplications and divisions for *Alg-S* is clearly smaller than for *Gauss*. As implemented for the test problem, *Alg-S* required the calculation of 43 square roots when performing the two Cholesky factorizations, while *Gauss* required none. Square root calculations can be eliminated using an $\mathbf{LDL}^{\mathrm{T}}$ approach. The benefit of this alternative has not yet been investigated though.

One advantage of *Alg-S* over the Gaussian elimination family of solvers is that no pivoting is involved. Gaussian elimination requires pivoting in order to ensure numerical stability. *Alg-S* avoids this by employing Cholesky factorizations twice (during Steps 1 and 3 of *Algorithm 4*), and this factorization method is known to be numerically stable (Golub and Van Loan, 1989).

Table 7. Operation Count for *Alg-S*

Alg-S	**1**	**2**	**3.1**	**3.2**	**3.3**	**3.4**	**3.5**	**4**	**Total**
A	975	165	708	172	834	680	145	192	3871
M	975	165	804	172	1135	680	162	165	4258
D	165	7	174	0	0	120	0	27	513
SQ	27	0	0	0	0	16	0	0	43

A disadvantage of the new algorithm is the sparsity-related overhead; i. e., data accessing pattern and number of vector touches (number of memory accesses). While for Gaussian elimination the fashion in which data is manipulated is intuitive, it is not straightforward for the proposed algorithm. It is difficult to asses to what extent this will affect the overall performance. It depends on the particular mechanical system being modeled and the way the algorithm is coded. An ideal implementation of *Alg-S* would be a no-loop, hard-coded, problem dependent version that is generated during the preprocessing stage of the simulation. This would require a program that based on topology information, writes the code for the specific mechanical system, implementing *Algorithm 4* at the pre-processing stage.

An attempt was made to evaluate sparsity-related overhead for the test problem considered. For this, the steps of *Algorithm 4* were to be followed, but sparsity-related book-keeping present in *Alg-S* is eliminated using dense-matrix operations. Basic Linear Algebra Subroutine (BLAS) 2 and 3 operations, along with dense Cholesky factorization, are at the core of a new algorithm denoted below by *Alg-D*, the D standing for dense. For this algorithm, there is no need to renumber the bodies of the mechanical system, or to keep track of topology information. The operation count for *Alg-D*, using the same HMMWV example, is provided in Table 8.

Table 8. Operation Count for *Alg-D*

Alg-D	**1**	**2**	**3.1**	**3.2**	**3.3**	**3.4**	**3.5**	**4**	**Total**
A	3276	351	5616	432	3536	2280	405	378	15914
M	3276	351	5616	432	3672	2280	432	351	16410
D	351	27	432	0	0	152	0	27	989
SQ	27	0	0	0	0	16	0	0	43

Table 9 lists the CPU times in microseconds for the methods discussed above. The times correspond to one solution of the augmented linear system. For comparison, timing results for *Harwell* and *Symmetric* were also included. All numerical experiments presented in this Section were performed on a HP9000 model J210 computer with two PA 7200 processors.

Table 9. JR Linear Solvers CPU Results

Gauss	*Symmetric*	*Harwell*	*Alg-S*	*Alg-D*
2336	2022	3532	1201	1179

For the test problem considered, *Alg-D* is the algorithm of choice. The problem considered is too small for *Alg-S* to show any benefit from taking into account the topology-induced sparsity. It should be noted however that the numerical implementation of *Alg-S* could be further improved.

In *Alg-D*, the positive definite matrices $\overline{\mathbf{M}}$ and $\mathbf{T}^{\mathrm{T}}\mathbf{T}$, were factored using the driver *dposv*. In *Gauss*, *Symmetric*, and *Harwell*, the routines used were *dgesv*, *dsysv*, and the triple *ma28ad/ma28bd/ma28cd***,** respectively. With the exception of the triple *ma28* taken from an older Harwell public domain library, the other routines were available in Lapack. The faster routine *ma47* of the more recent versions of Harwell was unavailable at the time of this study, and it should improve the performance of the algorithm *Harwell*.

Finally, Table 10 presents a detailed profile of the algorithms *Alg-S* and *Alg-D*, listing CPU times in microseconds for each step of *Algorithm 4* in the two implementations.

Table 10. Timing Profiles – *Alg-S* and *Alg-D*

	1	**2**	**3.1**	**3.2**	**3.3**	**3.4**	**3.5**	**4**	**Total**
Alg-S	300	59	246	50	276	159	57	54	1201
Alg-D	289	31	347	27	276	147	31	31	1179

For the test problem considered, there is no advantage in considering sparsity for forward/backward elimination. It is only when sparsity information is used for the coefficient and right side matrices in step 3.1 (when computing the $\mathbf{T}$ matrix) that *Alg-S* gains an edge over *Alg-D*.

CHAPTER 4
NUMERICAL IMPLEMENTATIONS

This Chapter contains details regarding numerical implementation of methods presented in Chapter 3 for the implicit numerical integration of the DAE of Multibody Dynamics. To define the methods in Chapter 3, generic integration formulas were used. In this Chapter, specific integration formulas are considered, and details are provided about how a particular integration formula is embedded in the overall architecture of an implicit DAE integration algorithm.

4.1 Trapezoidal-Based State-Space Implicit Integration

4.1.1 General Considerations

The trapezoidal integration formula is often used for implicit integration of mildly stiff initial value problems (IVP). It is a popular multi-step formula that uses information only from the prior time step. The order of the method is two, and the method is A stable (Atkinson, 1989). For the IVP $y' = f(x, y)$, $y(x_0) = y_0$, it assumes the form

$$y_1 = y_0 + \frac{1}{2}h(f(x_0, y_0) + f(x_1, y_1))$$

This formula is used to integrate independent accelerations to obtain independent velocities, and then to integrate independent velocities to obtain independent positions. Thus,

$$\mathbf{v}_1 = \tilde{\mathbf{v}}_1 + \frac{h^2}{4}\ddot{\mathbf{v}}_1 \tag{4.1}$$

$$\dot{\mathbf{v}}_1 = \tilde{\dot{\mathbf{v}}}_1 + \frac{h}{2}\ddot{\mathbf{v}}_1 \tag{4.2}$$

where

$$\tilde{\mathbf{v}}_1 \equiv \mathbf{v}_0 + h\dot{\mathbf{v}}_0 + \frac{h^2}{4}\ddot{\mathbf{v}}_0$$

$$\tilde{\dot{\mathbf{v}}}_1 = \mathbf{v}_0 + \frac{h}{2}\ddot{\mathbf{v}}_0$$

The generic formulas of Eqs. (3.23) and (3.21) of Section 3.2.1 are replaced with Eqs. (4.1) and (4.2). Therefore, the coefficients that appear in the integration Jacobian of Eq. (3.43), are $\gamma = 0.5$ and $\beta = 0.25$.

The trapezoidal formula is known to be a simple answer to the challenge of solving stiff IVP. Compared to other integration formulas, in particular to a member of the Rosenbrock family presented later, the order is rather low, and stability properties are not sound. A typical problem encountered when using the trapezoidal formula is the loss of accuracy for very stiff ODE. This is caused by its lack of L-stability, a notion that is defined when discussing Runge-Kutta formulas in conjunction with the Descriptor Form Method.

4.1.2 Algorithm Pseudo-code

The numerical implementation of the State Space Method presented in this document is based on the trapezoidal formula of the previous Section. The pseudo-code of the implementation is provided in Table 11.

Table 11. Pseudo-code for Trapezoidal-Based State-Space Method

```
1.  Initialize Simulation
2.  Set Integration Tolerance
3.  While (t<tend) do
4.          Setup Integration Step
5.          Get Integration Jacobian
6.          Factor Integration Jacobian
7.          Do while (.NOT. converged)
8.                  Integrate
9.                  Recover Dependent Positions
10.                 Recover Dependent Velocities
11.                 Evaluate Mass Matrix and Active Forces
12.                 Evaluate Dependent Accelerations
13.                 Evaluate Lagrange Multipliers
14.                 Evaluate Error Term
15.                 Correct Independent Accelerations
16.         End do
17.         Check Accuracy.  Select Step-size
18.         Check Partition
19. End do
```

Step 1 initializes the simulation. A consistent set of initial conditions is determined, simulation starting and ending times are defined, and an initial step-size is provided. User set integration tolerances are read during Step 2.

Step 3 starts the simulation loop. The system state is stored, to be used later to restart the integration upon a rejected step. An initial estimate for the independent accelerations is also provided. Currently, values of independent accelerations from the previous time step are used to start the iterative process. Several other operations are done during this step, as is noted below.

At Step 5, the integration Jacobian of Eq. (3.43) is computed, with $\gamma = 0.5$ and $\beta = 0.25$. The integration Jacobian is factored at Step 6. Since the dimension of the integration Jacobian is equal to the number of degrees of freedom of the model, this matrix is rather small and dense. In the case of a moderately large HMMWV military vehicle that is modeled using 14 bodies (Serban, Negrut, and Haug 1998), which is used in the next Chapter for validation purposes, the dimension of integration Jacobian is 18. Consequently, dense Lapack routines are used to factor this matrix.

Step 7 starts the loop that solves the discretized non-linear algebraic equations. For the State-Space Method, the system

$$\psi(\ddot{\mathbf{v}}_1) \equiv \mathbf{M}_1^{\mathbf{vv}}\ddot{\mathbf{v}}_1 + \mathbf{M}_1^{\mathbf{vu}}\ddot{\mathbf{u}}_1 + (\Phi_{\mathbf{v}}^{\mathrm{T}})_1\lambda_1 - \mathbf{Q}_1^{\mathbf{v}} = \mathbf{0} \tag{4.3}$$

which is obtained after discretizing the independent equations of motion, is solved for independent accelerations. Given new independent accelerations, independent velocities and positions are obtained using at Step 8 Eqs. (4.1) and (4.2). Since dependent accelerations are available, using the same integration formulas they are also integrated twice to provide a better starting point for dependent positions recovery during Step 9. With a given $\mathbf{v}$, dependent coordinates $\mathbf{u}$ are obtained via a quasi-Newton approach,

$$\mathbf{u}^{(k+1)} = \mathbf{u}^{(k)} - \Phi_{\mathbf{u}}^{-1}(\mathbf{q}_0)\,\Phi(\mathbf{u}^{(k)}, \mathbf{v})$$

where $\Phi(\mathbf{u}, \mathbf{v}) = \mathbf{0}$ represents position kinematic constraint equation of Eq. (3.7). Here $\mathbf{q}_0$ is the vector of generalized coordinates from the last time step, or the value computed during the previous iteration for $\ddot{\mathbf{v}}$; i. e., the value obtained as a result of the last call to position recovery.

At the end of Step 9, the sub-Jacobian $\Phi_{\mathbf{u}}$ is factored in a consistent configuration of the mechanism. Dependent velocities are cheaply obtained during Step 10, by solving

$$\Phi_{\mathbf{u}}\dot{\mathbf{u}} = -\Phi_{\mathbf{v}}\dot{\mathbf{v}}$$

During Steps 12 and 13, the generalized mass and external forces are evaluated in the most recent configuration; i. e., using the last generalized positions and velocities available after Step 10. The computation proceeds by solving the linear systems

$$\Phi_{\mathbf{u}}\ddot{\mathbf{u}} = -\Phi_{\mathbf{v}}\ddot{\mathbf{v}} + \tau$$

to obtain dependent accelerations, and

$$\Phi_{\mathbf{u}}^{\mathrm{T}}\lambda = \mathbf{Q}^{\mathbf{u}} - \mathbf{M}^{\mathbf{uv}}\ddot{\mathbf{v}} - \mathbf{M}^{\mathbf{uu}}\ddot{\mathbf{u}}$$

to obtain the Lagrange multipliers. Note that finding the solution of these systems is inexpensive, since a factorization of the sub-Jacobian $\Phi_{\mathbf{u}}$ is available.

During Step 14, the error in satisfying the non-linear system of Eq. (4.3) is evaluated. Based on the norm of the residual and the norm of the last correction in independent accelerations, the decision to stop or continue the iterative process is taken. If stopping criteria are not met, another iteration is taken, unless the number of iterations already exceeds a specific number. In the latter case, the step-size is decreased, the code returns at Step 4, and the iterative process is restarted. However, the costly integration Jacobian computation is skipped. The integration Jacobian was shown in Section 3.2.1.2 to assume the form

$$\Psi_{\ddot{\mathbf{v}}} = \hat{\mathbf{M}} + \gamma h\hat{\mathbf{M}}_1 + \beta h^2\hat{\mathbf{M}}_2 \tag{4.4}$$

When the time step is rejected, matrices $\hat{\mathbf{M}}$, $\hat{\mathbf{M}}_1$, and $\hat{\mathbf{M}}_2$ are reused and only the step-size is going to be changed. Therefore, time step rejection is cheap as far as integration Jacobian evaluation is concerned.

If stopping criteria are not meet and the number of iterations is less than the limit number, the independent accelerations at iteration k are corrected at Step 15 as

$$\ddot{\mathbf{v}}_1^{(k+1)} = \mathbf{v}_1^{(k)} - \mathbf{\Psi}_{\ddot{\mathbf{v}}}^{-1} \cdot \mathbf{\Psi}(\mathbf{v}_1^{(k)})$$

and the code proceeds to Step 8 for a new iteration.

Once the iterative process has converged, accuracy of the numerical solution is verified. The configuration of the mechanical system at the new time step is known, and it remains to determine whether accuracy of the numerical solution meets requirements imposed by the user via the integration tolerances set at Step 2. An embedded method is used to obtain a second approximate solution that is used for integration error estimation. The difference of the two approximate solutions is taken to be the local truncation error. The new step-size is obtained by imposing the condition that the norm of the local truncation error is smaller than a certain composite norm, based on user imposed integration tolerances. This approach is detailed in Section 4.2 in the framework of Runge-Kutta methods. The embedded formula used in conjunction with the trapezoidal method is the backward Euler formula.

If accuracy requirements are met, the step-size computed as a by-product of error analysis is used for the next time step. Otherwise, the time step is rejected and, after proceeding to Step 4, the new step size is used in conjunction with Eq. (4.4) to reevaluate the integration Jacobian.

The partitioning of generalized coordinates is checked at Step 18. A repartitioning is triggered by a large value of the condition number of the dependent sub-Jacobian $\mathbf{\Phi}_{\mathbf{u}}$. In this context, large means a condition number that exceeds by a factor of

$\alpha = 1.25$ the value of the first condition number associated with the current partition. The value $\alpha = 1.25$ was determined as a result of numerical experiments. For the case of a vehicle model with 14 bodies and 98 states, the initial partitioning of coordinates was valid for 40 seconds of simulation, and no repartitioning request was made. This mechanism did not cause unjustified repartitioning requests, and proved to be reliable. Increasing the factor α will reduce the number of repartitionings, and conversely. Note that computing the condition number of the dependent sub-Jacobian is inexpensive, since a factorization of this matrix is available after the last call to dependent position recovery.

There are two CPU intensive stages of the proposed algorithm. First is computation of the integration Jacobian, which employs many matrix-matrix multiplications, as shown in Section 3.2.1.2. The computation of matrices $\hat{\mathbf{M}}$, $\hat{\mathbf{M}}_1$, $\hat{\mathbf{M}}_2$, $\mathbf{R}$, and $\mathbf{S}$ are intensive CPU operations. Second is factorization of the dependent sub-Jacobian $\mathbf{\Phi_u}$, which in the current numerical implementation is done after each dependent position recovery (Step 9). This is an important matrix that appears often in the implementation. It is used to obtain dependent velocities (Step 10), recover dependent accelerations $\ddot{\mathbf{u}}$ (Step 12), evaluate Lagrange multipliers λ (Step 13), and compute the matrices $\mathbf{H}$, $\mathbf{J}$, $\mathbf{N}$, and $\mathbf{L}$ (Step 5). These latter matrices are obtained as solutions of multiple right-side systems of linear equations whose coefficient matrix is the dependent sub-Jacobian $\mathbf{\Phi_u}$. Finally, the factorization of $\mathbf{\Phi_u}$ is needed in Step 18, to evaluate the condition matrix required for checking the validity of the current partitioning.

Usually, the dimension of the dependent sub-Jacobian $\mathbf{\Phi_u}$ is rather large. For the HMMWV14 model mentioned above, the dimension of this matrix is 80. Typically, 3 iterations are required to solve the non-linear system $\Psi(\ddot{\mathbf{v}}) = \mathbf{0}$ to retrieve $\ddot{\mathbf{v}}$. Consequently, each integration successful step requires 3 factorizations of $\mathbf{\Phi_u}$ and one factorization of the integration Jacobian. As will be shown later, the Rosenbrock formula embedded in the framework of the First Order Reduction Method, has better stability properties (L-stable), has order 4, includes a more reliable step-size control mechanism, and requires about the same computational effort per time step as the trapezoidal formula when used in conjunction with the State-Space Method.

4.2 SDIRK Based Descriptor Form Implicit Integration

4.2.1 General Considerations

Accuracy and stability are the issues of concern when applying a numerical integration formula to determine the solution of an IVP. When dealing with stiff IVP, explicit integration formulas are compelled to very small step-sizes due to stability limitations.

For most mechanical engineering applications, precision in the range of 10^{-4} through 10^{-2} usually suffice. For non-stiff IVP, the step-size of an explicit mid- to high-order formula might assume large values, and still meet the relatively mild accuracy requirements of 10^{-4}-10^{-2}. Yet when applied to stiff IVP, explicit codes produce meaningless results, unless the step-size is reduced to ridiculously small values to keep the whole process away from uncontrollable behavior.

Implicit integration formulas are effective in avoiding this drawback of explicit methods, and the step-size is again limited by accuracy considerations, as was the case with the explicit formulas applied to non-stiff IVP.

However, the good stability properties of implicit formulas have a computational penalty, as these formulas are more complex, more difficult to implement, and on a per-step basis more CPU intensive. Overall though, CPU times for solving stiff IVP can be cut by orders of magnitude if implicit integration is used.

From a theoretical standpoint, implicit formulas introduce concepts such as A and L-stability, contractivity (B-convergence) and order reduction, existence of numerical solution, and convergence issues. There are also issues regarding the numerical implementation of these formulas, in which linear algebra plays an important role. Stopping criteria, how these are reflected in the global integration error, and starting values for solving the discretized system of algebraic equations are issues that increase the complexity of implicit methods.

General notions related to the class of implicit Runge-Kutta methods are next considered. These are the notions that are relevant to particular Runge-Kutta formulas that are used in conjunction with the Descriptor Form and First Order Reduction Methods. An excellent reference on the topic of implicit integration is the book of Hairer and Wanner (1996), which contains a detailed treatment of this problem.

Consider Dahlquist test equation

$$y' = \lambda y$$

with $y(x_0) = 1$. If the forward Euler formula is applied to integrate this IVP, after one integration step the solution is given by

$$y_1 = (1 + h\lambda) y_0$$

or,

$$y_1 = R(h\lambda) y_0$$

where

$$R(z) \equiv 1 + z$$

The function $R(z)$ is called the stability function, which is specific to the integration formula. By definition, setting $z = h\lambda$, the stability function is obtained as the solution of the Dahlquist test problem after one integration step. The set

$$S = \left\{ z \in \boldsymbol{C}\, ; |R(z)| \leq 1 \right\}$$

is called the stability domain of the method.

A method whose stability domain satisfies

$$S \supset \boldsymbol{C}^{-} = \left\{ z\, ; \operatorname{Re} z \leq 0 \right\}$$

is called A-stable. This is a desirable property of an integration formula, to successfully deal with the class of stiff IVP. However, as Alexander (1977) has remarked, A-stability is not the entire answer to the problem of stiff equations. Using A-stable one-step methods to solve large systems of stiff non-linear differential equations, Prothero and Robinson (1974) concluded that some A-stable methods give highly unstable solutions. The accuracy of solutions obtained when the equations are very stiff often appeared to be unrelated to the order of the method used.

The observations of Prothero and Robinson characterize what is called the process of order reduction of implicit Runge-Kutta methods when applied to very stiff differential equations. Methods that display good behavior, even for extremely stiff IVP have been designed. One attribute of these methods is their L-stability property. A method is L-stable, if it is A-stable and satisfies the condition

$$\lim_{z \to \infty} R(z) = 0$$

One class of methods that satisfy this condition (Hairer and Wanner, 1996) is the class of stiffly-accurate Runge-Kutta methods, for which

$$a_{sj} = b_j \qquad\qquad j = 1, \ldots, s$$

In other words, the last line of a elements and the stage value weights in Butcher's tableau coincide. In particular, this class of problems is extensively used for the solution of singularly perturbed problems and low index DAE via direct methods.

For step-size control, Runge-Kutta formulas use a lower order method that works in conjunction with the integration formula to provide a second approximate solution. This additional solution is generally used only to calculate an approximation of the local truncation error. Keeping this local error lower than user prescribed accuracy requirements is the basic idea of the step-size controller.

To compute the second approximation of the solution at the new grid point efficiently, information available during the process of obtaining the numerical solution should be used to advantage. Usually, stage values are used with weights $\hat{b}_i$ to provide a second approximation $\hat{y}_1$ of the solution

$$\hat{y}_1 = y_0 + \sum_{i=1}^{s} \hat{b}_i k_i \tag{4.5}$$

Following the presentation of Hairer, Nørsett, and Wanner (1993), an estimate of the error for the less precise result is $y_1 - \hat{y}_1$. Componentwise, this error is kept smaller than a composite error tolerance sc_i

$$|y_{1i} - \hat{y}_{1i}| \leq sc_i \qquad sc_i = Atol_i + \max(|y_{0i}|, |y_{1i}|) \cdot Rtol_i \tag{4.6}$$

where $Atol_i$ and $Rtol_i$ are user prescribed integration tolerances. As a measure of the error, the value

$$err = \sqrt{\frac{1}{n} \sum_{i=1}^{n} \left(\frac{y_{1i} - \hat{y}_{1i}}{sc_i} \right)^2} \tag{4.7}$$

is considered. Other norms might be chosen; one alternative used often being the max norm. Then, err in Eq. (4.7) is compared to 1, in order to find an optimal step-size.

From asymptotical error behavior, $err \approx C \cdot h^{q+1}$, and from $1 \approx C \cdot h_{opt}^{q+1}$ (where $q = \min(p, \hat{p})$, with p and $\hat{p}$ being the order of the formulas used), the optimal step size is obtained as

$$h_{opt} = h \cdot \left(\frac{1}{err} \right)^{\frac{1}{q+1}} \tag{4.8}$$

A safety factor fac usually multiplies h_{opt} such that the error is acceptable at the end of the next step with high probability. Further, h is not allowed to increase or decrease too fast. Thus, the value used for the new step-size is

$$h_{new} = h \cdot \min(\text{facmax}, \max(\text{facmin}, fac \cdot (1/err)^{1/(q+1)}))$$

and if at the end of the current step, $err \leq 1$, the step is accepted. The solution is then advanced with y_1 and a new step is tried, with h_{new} as step-size. Otherwise, the step is rejected and computations for the current step are repeated with the new step-size h_{new}. The maximal step-size increase facmax, usually chosen between 1.5 and 5, prevents the code from taking too large a step and contributes to its safety. When chosen too small, it may also unnecessarily increase the computational work. Finally, it is advisable to put $\text{facmax} = 1$ in steps after a step-rejection (Shampine and Watts, 1979).

The last important aspect, from a practical standpoint, is a dense output capability. This capability is important for many practical questions such as event location and treatment of discontinuities in differential equations, graphical output, etc. From a mathematical perspective, this issue is important if the number of output points is very large. Cutting the step-size, to meet output request and not accuracy or stability limitations, may reduce efficiency of the algorithm significantly.

These are considerations that motivate the construction of dense output formulas (Horn, 1983). The idea is to provide, in addition to the numerical result y_1, cheap numerical approximations to $y(x_0 + \theta h)$, for the entire integration interval $0 \le \theta \le 1$. Thus, using stage values k_i, a set of coefficients depending on the output point through θ, are provided to give an approximation of the solution at the intermediate point $x_0 + \theta h$. This approximate solution is then obtained as

$$u(\theta) = y_0 + h\sum_{i=1}^{s} b_i(\theta)k_i \tag{4.9}$$

where $b_i(\theta)$ are polynomials in θ. Based on order conditions, they are determined such that

$$u(\theta) - y(x_0 + \theta h) = \mathrm{O}(h^{p+1})$$

where $y(x)$ is the solution of the IVP and p is the order of the method.

A Runge-Kutta method provided with a formula such as Eq. (4.9) is called a continuous Runge-Kutta method. The two SDIRK methods presented in this document for implicit integration of SSODE are continuous methods. Further information regarding continuous methods can be found in the works of Hairer, Nørsett, and Wanner (1993) and Shampine (1986).

4.2.2 SDIRK4/16

Since there is no extra price for using a stiffly-accurate Runge-Kutta formula that can successfully handle even very stiff ODE, this class of integrators was embedded in the Descriptor Form Method. The Runge-Kutta method implemented was a 5 stage, order 4, stiffly accurate formula, with error control based on an order 3 embedded formula. With γ being the diagonal element of the SDIRK formula, define

$$p_1 = 1 - \gamma \qquad p_2 = 1/2 - 2\gamma + \gamma^2$$

$$p_3 = 1/3 - 2\gamma + 3\gamma^2 - \gamma^3 \qquad p_4 = 1/6 - 3/2\,\gamma + 3\gamma^2 - \gamma^3$$

$$p_5 = 1/4 - 2\gamma + 9/2\,\gamma^2 - 4\gamma^3 + \gamma^4 \qquad p_6 = 1/8 - 4/3\gamma + 4\gamma^2 - 4\gamma^3 + \gamma^4$$

$$p_7 = 1/12 - \gamma + 7/2\,\gamma^2 - 4\gamma^3 + \gamma^4 \qquad p_8 = 1/24 - 2/3\gamma + 3\gamma^2 - 4\gamma^3 + \gamma^4$$

The following simplified conditions assure the order four of the method (Hairer and Wanner (1996))

$$\begin{aligned}
&b_1 + b_2 + b_3 + b_4 = p_1\\
&b_2c_2' + b_3c_3' + b_4c_4' = p_2\\
&b_2c_2'^2 + b_3c_3'^2 + b_4c_4'^2 = p_3\\
&b_3a_{32}c_2' + b_4(a_{42}c_2' + a_{43}c_3') = p_4\\
&b_2c_2'^3 + b_3c_3'^3 + b_4c_4'^3 = p_5\\
&b_3c_3'a_{32}c_2' + b_4c_4'(a_{42}c_2' + a_{43}c_3') = p_6\\
&b_3a_{32}c_2'^2 + b_4(a_{42}c_2'^2 + a_{43}c_3'^2) = p_7\\
&b_3a_{43}a_{32}c_2' = p_8
\end{aligned} \tag{4.10}$$

The coefficients c_i, are defined as $c_i' = \sum_{j=1}^{i-1} a_{ij}$, $i = 1,2,3$. The associated Butcher's tableau is given in Table 12.

Table 12. Butcher's Tableau for SDIRK Formulas

c_1	γ	0	…	…	0
c_2	a_{21}	γ	…	…	0
…	…	…	…	…	…
c_s	a_{s1}	a_{s2}	…	…	γ
	b_1	b_2	…	…	b_s

The A-stability property of the method is determined by the choice of γ in this tableau (Hairer and Wanner, 1996). In order to obtain good stability properties, along with a small leading coefficient of the local truncation, Hairer and Wanner (1996) suggest values of γ in the range 0.25 and 0.29. For the Descriptor Form Method, the value chosen was $\gamma = 0.25$ (or $\gamma = 4/16$, the name of this formula). According to Butcher's tableau, if the number of stages is $s = 5$, there are 11 coefficients to be determined, namely the sub-diagonal elements a_{ij}. Once these coefficients are known, since the method is stiffly stable, the coefficients b_i are known. Finally, the coefficients c_i are obtained using the conditions

$$c_i = \sum_{i=1}^{i} a_{ij} \quad , \qquad i = 1,\dots,5$$

With the coefficient γ chosen, eight order conditions are available in Eq. (4.10) to compute 11 unknowns. The order of the method and its stability properties are satisfied by the conditions imposed. Therefore, the two extra degrees of freedom in the choice of coefficients are used to minimize the fifth-order error terms. This is expected to attain additional accuracy for the otherwise fourth order formula. This condition leads to the recommendation that $c_2' = 0.5$ and $c_3' = 0.3$ (Hairer and Wanner, 1996). With these two conditions, a non-linear system is solved for a_{ij}, and the resulting stiffly-accurate, L-stable, 5 stage, order 4, Runge-Kutta formula obtained is shown in Table 13.

Step size control is based on an order 3 embedded formula. To obtain the coefficients of the second formula, order conditions are revisited. They are reformulated in general form, to obtain a set of equations for the new weights $\hat{b}_i$. Since the same stage values are used, the a_{ij} and c_i coefficients remain the same. Only the weights that are to change to produce a new approximation of the solution. The order three conditions assume the form

$$\sum_{j} \hat{b}_j = 1$$

$$\begin{aligned} &\sum_{j,k} \hat{b}_j a_{jk} = \frac{1}{2} \\ &\sum_{j,k,l} \hat{b}_j a_{jk} a_{jl} = \frac{1}{3} \\ &\sum_{j,k,l} \hat{b}_j a_{jk} a_{kl} = \frac{1}{6} \end{aligned} \qquad (4.11)$$

Table 13. SDIRK4/16 Formula for Descriptor Form Method

1/4	1/4	0	0	0	0
3/4	1/2	1/4	0	0	0
11/20	17/50	-1/25	1/4	0	0
1/2	371/1360	-137/2720	15/544	1/4	0
1	25/24	-49/48	125/16	-85/12	1/4
$y_1 =$	25/24	-49/48	125/16	-85/12	1/4
$\hat{y}_1 =$	59/48	-17/96	225/32	-85/12	0

Using values of a_{ij} from Table 13, these equations lead to the following linear system for the weights $\hat{b}_i$:

$$\begin{bmatrix} 1 & 1 & 1 & 1 \\ 1/4 & 3/4 & 11/20 & 1/20 \\ 1/16 & 9/16 & 121/400 & 1/4 \\ 1/16 & 5/16 & 77/400 & 29/170 \end{bmatrix} \begin{bmatrix} \hat{b}_1 \\ \hat{b}_2 \\ \hat{b}_3 \\ \hat{b}_4 \end{bmatrix} = \begin{bmatrix} 1 \\ 1/2 \\ 1/3 \\ 1/6 \end{bmatrix}$$

The solution of this system is provided as the last row in Table 13.

Finally, the coefficients of the dense output formula are given by

$$\begin{aligned} b_1(\theta) &= \frac{11}{3}\theta - \frac{463}{72}\theta^2 + \frac{217}{36}\theta^3 - \frac{20}{9}\theta^4 \\ b_2(\theta) &= \frac{11}{2}\theta - \frac{385}{16}\theta^2 + \frac{661}{24}\theta^3 - 10\theta^4 \\ b_3(\theta) &= -\frac{125}{18}\theta + \frac{20125}{432}\theta^2 - \frac{8875}{216}\theta^3 + \frac{250}{27}\theta^4 \\ b_4(\theta) &= -\frac{85}{4}\theta^2 + \frac{85}{6}\theta^3 \\ b_5(\theta) &= -\frac{11}{9}\theta + \frac{557}{108}\theta^2 - \frac{359}{54}\theta^3 + \frac{80}{27}\theta^4 \end{aligned} \tag{4.12}$$

These coefficients are obtained by symbolically solving simplified order conditions; e. g., those provided in Eq. , with modified right-sides that are functions of θ. Details are provided by Hairer and Wanner (1996).

4.2.3 Algorithm Pseudo-code

This Section provides the pseudo-code of the algorithm developed based on SDIRK4/16 formula used in conjunction the Descriptor Form Method.

During Step 1, the initial configuration of the mechanical system is read in, along with initial and final times. An initial estimation for the integration step-size is provided. This step size is subsequently changed by the step-size controller, to accommodate accuracy constraints. Step 2 reads values of tolerances prescribed by the user; i. e., the values $Atol_i$ and $Rtol_i$ of Eq. (4.6).

Table 14. Pseudo-code for SDIRK4/16-Based Descriptor Form Method

```
1.  Initialize Simulation
2.  Set Integration Tolerance
3.  While (t < tend) do
4.          Setup Macro-step
5.          Get Integration Jacobian
6.          Sparse Factor Integration Jacobian
7.          Do stage 1 to 5
8.                  Setup Stage
9.                  Do while (.NOT. converged)
10.                         Integrate
11.                         Recover Positions and Velocities
12.                         Get Error Term
13.                         Correct Accelerations and Lagrange Multipliers
14.                 End do
15.         End do
16.         Check Accuracy.  Determine New Step-size
17.         Check Partition
18. End do
```

At Step 3 the simulation loop is started, and the code proceeds by setting the new macro-step. If the step is successful, the current configuration is saved. Otherwise, when rejected, the initial settings are used again to restart the integration, with step-size suggested by the step-size controller.

Step 5 is the pivotal point of the implementation. If the current time step has not been rejected, the integration Jacobian is computed according to considerations of Section 3.3, which is the CPU intensive part of the code. Otherwise, if the current call to the integration Jacobian computation comes after an unsuccessful time step, the integration Jacobian is known, since it assumes the form $\mathbf{M}_1 + \gamma h \mathbf{M}_2 + \beta h^2 \mathbf{M}_3$, and matrices $\mathbf{M}_1$, $\mathbf{M}_2$, and $\mathbf{M}_3$ are available from the first rejected attempt. Only the step-size h is modified, and re-computing the integration Jacobian is cheap.

Harwell sparse linear algebra routines are used for factorization of the integration Jacobian. Since the sparsity pattern of the integration Jacobian does not change during integration, the factorization process is cheap, once the routine *ma48ad* of Harwell has analyzed its structure and a factorization sequence has been determined. All subsequent calls to integration Jacobian factorization use the much faster *ma48bd* factorization routine. Care should be taken to make sure the defining call to *ma48ad* is done with the typical sparsity pattern of integration Jacobian, and sometimes this might not be induced by the first configuration of the mechanism.

At Step 7 is started to loop for the stage values k_i. Values iterated for are the generalized accelerations and Lagrange multipliers. At Step 8, starting values for these quantities are provided, and the iteration counter is reset to zero. If during a stage of the formula, this counter exceeds a limit value, the time step is deemed rejected, the integration step-size is decreased, and the code proceeds to Step 4.

The solution of the discretized non-linear algebraic equations is obtained during the loop that starts at Step 9 and ends at Step 14. As described in Section 3.3.2, based on the SDIRK formula of Table 13, accelerations are integrated to obtain generalized velocities, which are integrated to obtain generalized positions. After direct integration of all generalized accelerations and velocities, the generalized positions and velocities fail to satisfy the kinematic constraint equations at position and velocity levels. Dependent positions obtained after direct integration are considered only as starting estimates for recovering the consistent configuration of the mechanism, via solution of position kinematic constraint equations. The same observations apply for dependent velocities, which are computed using velocity kinematic constraint equations. This is the reason for which, although the discretization is done at the index 1 DAE level, the Descriptor Form is truly a state-space method.

At Step 13, the error term in satisfying the index 1 DAE is evaluated. At step 14, corrections in generalized accelerations and Lagrange multipliers are computed. Based on the norm of these corrections, the integration tolerance, and the norm of the error in the discretized index 1 DAE, an iteration stopping decision is made. If stopping criteria are met, the code proceeds to Step 16, once all five stages of the SDIRK4/16 formula have been completed. Otherwise, corrections in accelerations and Lagrange multipliers are made, and if the limit number of iterations has not been reached, another iteration is started.

During Step 16, the step size controller described in Section 4.2.1, based on an embedded formula provided in the last row of Table 13, analyzes the accuracy of the numerical solution. If accuracy of the approximate solution is satisfactory, the configuration at this grid point is accepted, and integration proceeds with a step-size that is obtained as a by-product of the accuracy check. Otherwise, with the newly computed step-size, the code proceeds to Step 4 to restart integration, this time on the premise of a rejected time step.

During Step 17, the partitioning of the vector of generalized coordinates is checked. If necessary, a new dependent/independent partitioning is determined. This process is detailed for the trapezoidal-based State-Space Method in Section 4.1.2.

Finally, Step 18 is the end of the simulation loop.

4.3 Trapezoidal-Based Descriptor Form Implicit Integration

The same trapezoidal integration formula used in the framework of the State-Space Method is used here in conjunction with the Descriptor Form Method. Characteristics of this integrator, such as stability properties, order, etc. were presented in Section 4.1.1. Pseudo-code for trapezoidal-based implicit integration of DAE via the Descriptor Form Method is presented below.

4.3.1 Algorithm Pseudo-code

A large part of the discussion of Section 4.2.2 regarding SDIRK4/16-based integration via the Descriptor Form Method applies here. Since the discretization is based on the trapezoidal rather than SDIRK4/16 formula, the numerical implementation is identical, with the exception that instead of having 5 stages, there is only one stage that advances the simulation. The pseudo-code of the algorithm is provided in Table 15.

During Stage 4, the system configuration at the beginning of the time step is saved. This information is used after a rejected time step for reinitializing the state of the system and restarting integration with a new step-size. Compared to the pseudo-code for

SDIRK4/16, the *Setup Stage* and the loop after the 5 stages do not appear. All other steps of the implementation are as discussed in Section 4.2.3., in conjunction with the SDIRK4/16 algorithm. Additional information about this algorithm is provided in the work of Haug, Negrut, and Engstler (1998).

Table 15. Pseudo-code for Trapezoidal-Based Descriptor Form Method

```
1.   Initialize Simulation
2.   Set Integration Tolerance
3.   While (t < tend) do
4.           Setup Step
5.           Get Integration Jacobian
6.           Sparse Factor Integration Jacobian
7.           Do while (.NOT. converged)
8.                   Integrate
9.                   Recover Positions and Velocities
10.                  Get Error Term
11.                  Correct Accelerations and Lagrange Multipliers
12.          End do
13.          Check Accuracy.  Determine New Step-size
14.          Check Partition
15.  End do
```

4.4 Rosenbrock-Based First Order Implicit Integration

4.4.1 General Considerations

The fundamental idea of the proposed algorithm is to reduce the second order SSODE to an equivalent first order ODE, and apply a Rosenbrock formula to integrate the ODE. The discussion starts with a brief description of Rosenbrock methods, followed by a more detailed presentation of how this class of methods is used to integrate the SSODE of Multibody Dynamics. Finally, a four stage, order 4, L-stable Rosenbrock method is derived (Sandu, et al., 1998).

For the differential equation

$$\mathbf{y}' = \mathbf{f}(\mathbf{y}) \tag{4.13}$$

an s stage diagonal implicit Runge-Kutta method defined by the set of coefficients (a_{ij}, b_i) assumes the form

$$\mathbf{k}_i = h\mathbf{f}(\mathbf{y}_0 + \sum_{j=1}^{s-1} a_{ij}\mathbf{k}_j + a_{ii}\mathbf{k}_i) \quad , \qquad i = 1, \ldots, s \tag{4.14}$$

$$\mathbf{y}_1 = \mathbf{y}_0 + \sum_{i=1}^{s} b_i \mathbf{k}_i \tag{4.15}$$

At stage i, the system of nonlinear equations of Eq. (4.14) is solved for $\mathbf{k}_i$. The solution is obtained as a linear combination of stage values $\mathbf{k}_i$ as indicated in Eq. (4.15).

For Rosenbrock methods, instead of solving a non-linear system at each stage, the term $\mathbf{f}(\cdot)$ in the right side of Eq. (4.14) is linearized in terms of $\mathbf{k}_i$. The stage value $\mathbf{k}_i$ is then obtained as the solution of a linear system of equations. The benefit of Rosenbrock methods lies in this approach to finding $\mathbf{k}_i$. It can be shown (Hairer and Wanner, 1996) that a Rosenbrock method can be interpreted as the application of *one*

Newton iteration to each stage in Eq. (4.14), with starting values $\mathbf{k}_i^{(0)} = \mathbf{0}$. This approach yields the following method:

$$\begin{aligned} \mathbf{k}_i &= h\mathbf{f}(\mathbf{g}_i) + h\mathbf{f}'(\mathbf{g}_i)a_{ii}\mathbf{k}_i \\ \mathbf{g}_i &\equiv \mathbf{y}_0 + \sum_{j=1}^{i-1} a_{ij}\mathbf{k}_j \end{aligned} \tag{4.16}$$

To gain even more computational advantage, the Jacobian term $\mathbf{f}'(\mathbf{g}_i)$ in Eq. (4.16) is replaced by $\mathbf{J} \equiv \mathbf{f}'(\mathbf{y}_0)$. To satisfy order conditions more easily, additional linear combinations of terms $\mathbf{J}\mathbf{k}_i$ are introduced into Eq. (4.16) (Nørsett and Wolfbrandt 1979, Kaps and Rentrop 1979), to arrive at the following class of methods.

Definition 4.1. An s-stage Rosenbrock method is given by the formulas

$$\begin{aligned} \mathbf{k}_i &= h\mathbf{f}(\mathbf{y}_0 + \sum_{j=1}^{i-1} a_{ij}\mathbf{k}_j) + h\mathbf{J}\sum_{j=1}^{\mathrm{i}} \mathrm{e}_{\mathrm{ij}}\mathbf{k}_j, \qquad i = 1, \ldots, s \\ \mathbf{y}_1 &= \mathbf{y}_0 + \sum_{i=1}^{s} b_i\mathbf{k}_i \end{aligned} \tag{4.17}$$

where a_{ij}, e_{ij}, and b_i are the defining coefficients and $\mathbf{J} = \mathbf{f}'(\mathbf{y}_0)$.

Each stage of this method consists of a system of linear equations, with unknowns $\mathbf{k}_i$ and coefficient matrix $\mathbf{I} - he_{ii}\mathbf{J}$. Of special interest are methods with $e_{11} = \cdots = e_{ss} = e$, for which only one LU decomposition is needed per step.

The case of non-autonomous ordinary differential equations

$$\mathbf{y}' = \mathbf{f}(t, \mathbf{y}) \tag{4.18}$$

is converted to autonomous form by appending to the original ODE the differential equation $t' = 1$, and regarding time as one of the variables. In this case, if the method of Eq. (4.17) is applied, the corresponding Rosenbrock method is obtained as

$$\begin{aligned}\mathbf{k}_i &= h\mathbf{f}(t_0 + a_i h, \mathbf{y}_0 + \sum_{j=1}^{i-1} a_{ij}\mathbf{k}_j) + e_i h^2 \mathbf{f}_t(t_0, \mathbf{y}_0) + h\mathbf{f}_{\mathbf{y}}(t_0, \mathbf{y}_0)\sum_{j=1}^{i} e_{ij}\mathbf{k}_j \\ \mathbf{y}_1 &= \mathbf{y}_0 + \sum_{i=1}^{s} b_i \mathbf{k}_i\end{aligned} \tag{4.19}$$

where the additional coefficients are given by

$$a_i = \sum_{j=1}^{i-1} a_{ij}, \qquad\qquad e_i = \sum_{j=1}^{i} e_{ij} \tag{4.20}$$

and the notation $\mathbf{f}_t \equiv \partial\mathbf{f}/\partial t$, and $\mathbf{f}_{\mathbf{y}} \equiv \partial\mathbf{f}/\partial\mathbf{y}$ has been used.

4.4.2 Rosenbrock-Nystrom Methods

Assume in what follows that a second order scalar ordinary differential equation is given in the form

$$y'' = f(t, y, y') \tag{4.21}$$

The assumption that the differential equation in Eq. (4.21) is scalar simplifies notation and makes the presentation more accessible. The implications of the case in which $\mathbf{y}$ is a vector will be pointed out when necessary. Technically, they amount to substituting certain matrix-matrix, and matrix-vector products with tensorial products.

The goal is to apply the formula of Eq. (4.19) to the differential equation in Eq. (4.21). This is the scenario when the SSODE obtained after DAE-to-ODE reduction is integrated numerically.

First, the second order ODE is reduced to the first order ODE

$$\begin{bmatrix} y \\ y' \end{bmatrix}' = \begin{bmatrix} y' \\ f(t, y, y') \end{bmatrix} \tag{4.22}$$

The Rosenbrock method is formally applied to integrate this ODE. If k_i and r_i are intermediate stage values at independent position and velocity levels, applying the method of Eq. (4.19) to the differential equation of Eq. (4.22) yields

$$\begin{bmatrix} k_i \\ r_i \end{bmatrix} = h\begin{bmatrix} y_0' + \sum_{j=1}^{i-1} a_{ij} r_i \\ f(t_0 + a_i h, y_0 + \sum_{j=1}^{i-1} a_{ij} k_j, y_0' + \sum_{j=1}^{i-1} a_{ij} r_j) \end{bmatrix} + e_i h^2 \begin{bmatrix} 0 \\ f_t(t_0, y_0, y_0') \end{bmatrix} \tag{4.23}$$

$$+h\begin{bmatrix} 0 & I \\ J_1 & J_2 \end{bmatrix} \sum_{j=1}^{i} e_{ij} \begin{bmatrix} k_j \\ r_j \end{bmatrix}$$

$$\begin{bmatrix} y_1 \\ y_1' \end{bmatrix} = \begin{bmatrix} y_0 \\ y_0' \end{bmatrix} + \sum_{i=1}^{s} b_i \begin{bmatrix} k_i \\ r_i \end{bmatrix} \tag{4.24}$$

where

$$J_1 \equiv \frac{\partial f}{\partial y}(t_0, y_0, y_0') \qquad , \qquad J_2 \equiv \frac{\partial f}{\partial y'}(t_0, y_0, y_0') \tag{4.25}$$

Although a_{ii}, $i = 1, \dots, s$, do not appear among the coefficients of the formula, they are introduced and formally set to zero, to simplify the summation indices. Likewise, the following notation is introduced

$$a_i = \sum_{j=1}^{i} a_{ij} \qquad , \qquad e_i = \sum_{j=1}^{i} e_{ij} \tag{4.26}$$

To obtain a numerical method able to directly integrate Eq. (4.21), the y-stage characteristic k_i unknowns are express in terms of the y'-stage unknowns r_i. Denoting $\beta_{ij} = a_{ij} + e_{ij}$, the first row of Eq. (4.23) becomes

$$k_i = hy_0' + h\sum_{j=1}^{i} \beta_{ij} r_j , \qquad i = 1, \dots, s \tag{4.27}$$

In what follows, using Eq. (4.27), the unknowns k_i are systematically eliminated from Eq. (4.23).

The term $\sum_{j=1}^{i-1} a_{ij} k_j$ appears as the second argument of $f(\cdot,\cdot,\cdot)$. Using Eq. (4.27), and interchanging the order of summation, this term is expressed as

$$\sum_{j=1}^{i-1} a_{ij} k_j = \sum_{j=1}^{i-1} a_{ij} \left(h y_0' + \sum_{m=1}^{j} \beta_{jm} r_m \right) = h a_i y_0' + \left(\sum_{j=1}^{i-1} \sum_{m=1}^{j} a_{ij} \beta_{jm} r_m \right)$$
$$= h a_i y_0' + \left(\sum_{j=1}^{i-1} a_{ij} \beta_{j1} r_1 + \sum_{j=2}^{i-1} a_{ij} \beta_{j2} r_2 + \ldots \sum_{j=i-1}^{i-1} a_{ij} \beta_{j,i-1} r_{i-1} \right) = h a_i y_0' + \sum_{j=1}^{i-1} \delta_{ij} r_m$$

where

$$\mu_{ij} = \sum_{m=j}^{i-1} a_{im} \beta_{mj} \tag{4.28}$$

Since this procedure of expressing k_i in terms of r_i will to appear several times during this discussion, a matrix representation that simplifies the process is introduced. If $\mathbf{k} \equiv [k_1, k_2, \ldots, k_s]^{\mathrm{T}}$, and $\mathbf{r} \equiv [r_1, r_2, \ldots, r_s]^{\mathrm{T}}$, based on Eq. (4.27),

$$\mathbf{k} = h y_0' \cdot \mathbf{1} + h \mathbf{B} \mathbf{r} \tag{4.29}$$

where the vector $\mathbf{1} \in \Re^s$ is defined as $\mathbf{1} \equiv [1,1,\ldots,1]^{\mathrm{T}}$, and

$$\mathbf{B} \equiv (\beta_{ij}), \qquad \mathbf{B} \in \Re^{s \times s} \tag{4.30}$$

For stage $i = 1, \ldots, s$, terms such as $\sum_{j=1}^{i-1} a_{ij} k_j$ can be simultaneously evaluated by using the matrix notation introduced. In this notation, using Eq. (4.29), the array $\mathbf{z} \in \Re^s$, $\mathbf{z} \equiv \left(\sum_{j=1}^{i-1} a_{ij} k_j \right)_i$, can be expressed as

$$\mathbf{z} = \mathbf{A}\mathbf{k} = \mathbf{A}(h y_0' \cdot \mathbf{1} + h \mathbf{B}\mathbf{r}) = h y_0' \begin{bmatrix} a_1 \\ \vdots \\ a_s \end{bmatrix} + h \mathbf{A}\mathbf{B}\mathbf{r} = h y_0' (a_i) + h \Delta \mathbf{r}$$

where, if multiplication is carried out to compute the matrix $\Delta \equiv \mathbf{AB}$, it can be seen that $\Delta = (\mu_{ij})$, with μ_{ij} given by Eq. (4.28).

The second row of Eq. (4.23) can now be expressed as

$$
\begin{aligned}
r_i = {} & hf\,(t_0 + a_i h, y_0 + h a_i y_0' + h\sum_{j=1}^{i-1} \delta_{ij} r_j, y_0' + \sum_{j=1}^{i-1} a_{ij} r_j) \\
& + e_i h^2 \frac{\partial f}{\partial t}(t_0, y_0, y_0') + hJ_1 \sum_{j=1}^{i} e_{ij} k_j + hJ_2 \sum_{j=1}^{i} e_{ij} r_j
\end{aligned}
\tag{4.31}
$$

In order to eliminate the k_j that multiply J_1 in Eq. (4.31), the matrix notation and the notation in Eq. (4.26) are used to yield

$$
\begin{aligned}
\left(\sum_{j=1}^{i} e_{ij} k_j \right)_i & = \mathbf{Ek} = \mathbf{E}(hy_0' \cdot \mathbf{1} + h\mathbf{Br}) = hy_0' \begin{bmatrix} e_1 & \cdots & e_s \end{bmatrix}^{\mathrm{T}} + h\mathbf{EBr} \\
& \equiv hy_0' (e_i)_s + h\mathbf{Q}
\end{aligned}
$$

where $\mathbf{Q} = (q_{ij}) \equiv \mathbf{EB}$. Stage i finally assumes the form

$$
\begin{aligned}
r_i = {} & hf\,(t_0 + a_i h, y_0 + h a_i y_0' + h\sum_{j=1}^{i-1} \delta_{ij} r_j, y_0' + \sum_{j=1}^{i-1} a_{ij} r_j) \\
& + e_i h^2 \left(\frac{\partial f}{\partial t}(t_0, y_0, y_0') + J_1 y_0' \right) + h^2 J_1 \sum_{j=1}^{i} q_{ij} r_j + hJ_2 \sum_{j=1}^{i} e_{ij} r_j
\end{aligned}
\tag{4.32}
$$

Since $q_{ii} = e_{ii}\beta_{ii} = e_{ii}(e_{ii} + a_{ii}) = e_{ii}^2 = e^2$, at stage i, $1 \le i \le s$, r_i is computed as the solution of the linear system

$$
\begin{aligned}
\mathbf{S} r_i = {} & hf\,(t_0 + a_i h, y_0 + h a_i y_0' + h\sum_{j=1}^{i-1} \delta_{ij} r_j, y_0' + \sum_{j=1}^{i-1} a_{ij} r_j) \\
& + e_i h^2 \left(\frac{\partial f}{\partial t}(t_0, y_0, y_0') + J_1 y_0' \right) + h^2 J_1 \sum_{j=1}^{i-1} q_{ij} r_j + hJ_2 \sum_{j=1}^{i-1} e_{ij} r_j
\end{aligned}
\tag{4.33}
$$

$$
\mathbf{S} \equiv I - heJ_2 - h^2 e^2 J_1 \tag{4.34}
$$

Due to the choice of coefficients $e_{11} = \ldots = e_{ss} = e$, only one matrix factorization needs to be computed per successful time step. At each stage, this factorization is used to compute r_i. Note that the right side of Eq. (4.33) depends on past information only. Once r_i. $i = 1, \ldots, s$, are available, the solution is obtained from Eq. (4.24) via Eq. (4.27).

In matrix notation, if $\mathbf{b}$ is the row vector $\mathbf{b} = [b_1, \ldots, b_s]$, the solution at the velocity level is given by

$$y_1' = y_0' + \mathbf{br} \tag{4.35}$$

Taking into account Eq. (4.29) and the fact that for Runge-Kutta methods $b_1 + \ldots + b_s = 1$, the solution at the position level is obtained as

$$y_1 = y_0 + hy_0' + h\mathbf{mr} \tag{4.36}$$

where the row vector $\mathbf{m}$ is defined as

$$\mathbf{m} = \mathbf{bB} \tag{4.37}$$

To summarize, the following linearly implicit method for solution of the second order ODE of Eq. (4.21) has been defined:

$$Y_i = y_0 + ha_i y_0' + h\sum_{j=1}^{i-1} \mu_{ij} r_j \tag{4.38}$$

$$Y_i' = y_0' + \sum_{j=1}^{i-1} a_{ij} r_j \tag{4.39}$$

$$\begin{aligned} \mathbf{S}r_i = hf(t_0 + a_i h, Y_i, Y_i') + e_i h^2 \left(\frac{\partial f}{\partial t}(t_0, y_0, y_0') + J_1 y_0' \right) \\ + h^2 J_1 \sum_{j=1}^{i-1} q_{ij} r_j + hJ_2 \sum_{j=1}^{i-1} e_{ij} r_j \end{aligned} \tag{4.40}$$

$$y_1 = y_0 + hy_0' + h\sum_{i=1}^{s} m_i r_i \tag{4.41}$$

$$y_1' = y_0' + h\sum_{i=1}^{s} b_i r_i \tag{4.42}$$

Following a remark of Hairer and Wanner (1996), when implementing the Rosenbrock method of Eqs. (4.38) through (4.42), one matrix-vector multiplication can be saved. For this, a new set of unknowns is introduced as

$$u_i \equiv \sum_{j=1}^{i} e_{ij} r_j \tag{4.43}$$

Using matrix notation, the original unknowns r_i are expressed in terms of $\mathbf{u} \equiv [u_1, \ldots, u_s]^{\mathrm{T}}$ as

$$\mathbf{r} = \mathbf{E}^{-1}\mathbf{u} \tag{4.44}$$

Denoting $\tilde{f} \equiv [f(t_0 + a_1 h, Y_1, Y_1'), \ldots f(t_0 + a_s h, Y_s, Y_s')]^{\mathrm{T}}$, Eq. (4.32) can be rewritten in matrix form as

$$e\mathbf{r} = he\tilde{f} + e^2h^2(f_t + J_1 y_0') + h^2 e J_1 \mathbf{Qr} + heJ_2\mathbf{Er} \tag{4.45}$$

Simple manipulations of Eq. (4.45) yield

$$\mathbf{Su} = he\tilde{f} + e^2h^2(f_t + J_1 y_0') + h^2 J_1 e(\mathbf{Qr} - e\mathbf{u}) + (\mathbf{E} - \mathrm{diag}(e))\mathbf{r} \tag{4.46}$$

Using the new set of unknowns $\mathbf{u}$ eliminates the multiplication in Eq. (4.46) with J_2. It remains to express terms containing the unknowns r_i in the right side of Eq. (4.46) in terms of $\mathbf{u}$. Thus,

$$(\mathbf{E} - \mathrm{diag(e)})\mathbf{r} = (\mathbf{E} - \mathrm{diag(e)})\mathbf{E}^{-1}\mathbf{u} = e(\mathrm{diag}(1/e) - \mathbf{E}^{-1})\mathbf{u} \tag{4.47}$$

and

$$\begin{aligned} \mathbf{Qr} - e\mathbf{u} &= \mathbf{QE}^{-1}\mathbf{u} - e\mathbf{u} = [\mathbf{E}(\mathbf{A} + \mathbf{E})\mathbf{E}^{-1} - \mathrm{diag}(e)]\mathbf{u} \\ &= [\mathbf{EAE}^{-1} + \mathbf{E} - \mathrm{diag}(e)]\mathbf{u} \end{aligned} \tag{4.48}$$

To take advantage of the matrix notation introduced, in what follows (e_{ij}) represents the matrix $\mathbf{E}$ used above, (a_{ij}) stands for the matrix whose elements are the coefficients a_{ij}, etc. With this, the algorithm for Rosenbrock-based integration of the differential equation of Eq. (4.21) is as follows:

Algorithm: Rosenbrock-Nystrom

Given the coefficients (a_{ij}), (e_{ij}), (b_i), $(\hat{b}_i)$ of an s stage Rosenbrock method, the associated Rosenbrock-Nystrom method is defined as

$$y_1 = y_0 + hy_0' + h\sum_{i=1}^{s} p_i u_i \quad , \qquad \hat{y}_1 = y_0 + hy_0' + h\sum_{i=1}^{s} \hat{p}_i u_i \tag{4.49}$$

$$y_1' = y_0' + \sum_{i=1}^{s} s_i u_i \quad , \qquad \hat{y}_1' = y_0' + \sum_{i=1}^{s} \hat{s}_i u_i \tag{4.50}$$

$$Y_i = y_0 + ha_i y_0' + h\sum_{j=1}^{i-1} t_{ij} u_j \tag{4.51}$$

$$Y_i' = y_0' + \sum_{j=1}^{i-1} w_{ij} u_j \tag{4.52}$$

$$\begin{aligned}\mathbf{S}u_i &= hef\,(t_0 + a_i h, Y_i, Y_i') + h^2 ee_i \left(\frac{\partial f}{\partial t}(t_0, y_0, y_0') + J_1 y_0' \right) \\ &\quad + e\sum_{j=1}^{i-1} c_{ij} u_j + h^2 e J_1 \sum_{j=1}^{i-1} d_{ij} u_j\end{aligned} \tag{4.53}$$

where

$$\begin{aligned}
(w_{ij}) &= (a_{ij})(e_{ij})^{-1} \\
(c_{ij}) &= \mathrm{diag}(1/\mathrm{e}) - (e_{ij})^{-1} \\
(d_{ij}) &= (e_{ij}) + (e_{ij})(a_{ij})(e_{ij})^{-1} \\
(t_{ij}) &= (a_{ij}) + (a_{ij})^2 (e_{ij})^{-1} \\
(s_i) &= (b_i)(e_{ij})^{-1} \\
(\hat{s}_i) &= (b_i)(e_{ij})^{-1} \\
(p_i) &= (b_i) + (b_i)(a_{ij})(e_{ij})^{-1} \\
(\hat{p}_i) &= (\hat{b}_i) + (\hat{b}_i)(a_{ij})(e_{ij})^{-1} \\
(e_i) &= (e_{ij}) \cdot \mathbf{1} \\
(a_i) &= (a_{ij}) \cdot \mathbf{1}
\end{aligned} \tag{4.54}$$

The matrices of coefficients (a_{ij}), (e_{ij}), and (b_i) define the Rosenbrock-Nystrom algorithm, including order and stability properties. All other coefficients of the algorithm are derived based on these coefficients.

Depending on the application that is numerically integrated, different orders and stability requirements should be considered. For the specific class of Multibody Dynamics problems, a low-to-medium accuracy method with good stability properties is desirable. Formulas with few function evaluations are favored, since function evaluations for mechanical system simulation amount to acceleration evaluations, which are expensive.

A method that is L stable allows for robust integration of very stiff problems. Consequently, bushing elements and flexible components used in modeling mechanical systems can be efficiently handled. An order 4 formula should reliably meet all accuracy requirements likely be considered for Multibody Dynamics simulation. With these considerations in mind, the defining matrices of coefficients (a_{ij}), (e_{ij}), and (b_i) are selected such that the resulting Rosenbrock-Nystrom method is of order 4 and is L stable. The number of stages of the method is chosen to be 4. This leaves some freedom in the choice of the embedded method for step-size control. Following an idea suggested by Hairer and Wanner (1996), the number of function evaluations is kept to 3; i.e., one function evaluation is saved. This makes the Rosenbrock-Nystrom method competitive with the trapezoidal method whenever the latter method requires 3 or more iterations for convergence. However, the proposed method is L stable while the trapezoidal method is not. Furthermore, the difference in order is 2: 4 for Rosenbrock-Nystrom, and 2 for trapezoidal.

4.4.3 Order Conditions For Rosenbrock-Nystrom Algorithm

The order conditions refer to conditions the defining coefficients (a_{ij}), (e_{ij}), and (b_i) should satisfy such that the method has a certain order. The discussion here is based on results presented by Hairer and Wanner (1996). In order to construct an order 4 Rosenbrock method, the order conditions are as follows:

$$b_1 + b_2 + b_3 + b_4 = 1 \tag{4.55}$$

$$b_2\beta_2' + b_3\beta_3' + b_4\beta_4' = 1/2 - e \tag{4.56}$$

$$b_2 a_2^2 + b_3 a_3^2 + b_4 a_4^2 = 1/3 \tag{4.57}$$

$$b_3\beta_{32}\beta_2' + b_4(\beta_{42}\beta_2' + \beta_{43}\beta_3') = 1/6 - e + e^2 \tag{4.58}$$

$$b_2 a_2^3 + b_3 a_3^3 + b_4 a_4^3 = 1/4 \tag{4.59}$$

$$b_3 a_3 a_{32}\beta_2' + b_4 a_4(a_{42}\beta_2' + a_{43}\beta_3') = \frac{1}{8} - \frac{e}{3} \tag{4.60}$$

$$b_3\beta_{32} a_2^2 + b_4(\beta_{42} a_2^2 + \beta_{43} a_3^2) = 1/12 - e/3 \tag{4.61}$$

$$b_4\beta_{43}\beta_{32}\beta_2' = 1/24 - \frac{e}{2} + \frac{3}{2}e^2 - e^3 \tag{4.62}$$

where the following abbreviations are used

$$a_i = \sum_{j=1}^{i-1} a_{ij} \qquad , \qquad \beta_i' = \sum_{j=1}^{i-1} \beta_{ij}$$

The step size control mechanism is based on an embedded formula (Kaps and Rentrop, 1979) of the form

$$\hat{y}_1 = y_0 + \sum_{j=1}^{s} \hat{b}_j k_j$$

which uses the stage values k_i from Eqs. (4.19). The embedded method should be of order 3; i. e., the order conditions of Eqs. (4.55) through (4.57) must be satisfied. These conditions lead to the system

$$\begin{pmatrix} 1 & 1 & 1 & 1 \\ 0 & \beta_2' & \beta_3' & \beta_4' \\ 0 & a_2^2 & a_3^2 & a_4^2 \\ 0 & 0 & \beta_{32}\beta_2' & \beta_{42}\beta_2' + \beta_{43}\beta_3' \end{pmatrix} \begin{pmatrix} \hat{b}_1 \\ \hat{b}_2 \\ \hat{b}_3 \\ \hat{b}_4 \end{pmatrix} = \begin{pmatrix} 1 \\ 1/2 - e \\ 1/3 \\ 1/6 - e + e^2 \end{pmatrix} \tag{4.63}$$

If the coefficient matrix in Eq. (4.63) is non-singular, uniqueness of the solution of this linear system implies $b_i = \hat{b}_i$, $i = 1,\dots,4$, and the order 3 embedded method can not be used for step size control. Consequently, beside the order conditions imposed on the defining coefficients a_{ij}, e_{ij}, and b_i, one additional condition should be considered for step-size control purposes, namely the coefficient matrix in Eq. (4.63) to be singular. This condition guarantees the existence of a third order embedded method when the system of non-linear equations (4.55) through (4.62) possesses a solution. The condition assumes the form

$$(\beta_2' a_4^2 - \beta_4' a_2^2)\beta_{32}\beta_2' = (\beta_2' a_3^2 - \beta_3' a_2^2)(\beta_{42}\beta_2' + \beta_{43}\beta_3') \tag{4.64}$$

The number of conditions that should be satisfied so far by the coefficients of the method is 9. The number of unknowns is 17; the diagonal element e, 6 coefficients e_{ij}, 6 coefficients a_{ij}, and 4 weights b_i. There are several degrees of freedom in the choice of coefficients, which are primarily used to construct a method with few function evaluations.

A first set of conditions is set in the form

$$\begin{aligned} a_{43} &= 0 \\ a_{42} &= a_{32} \\ a_{41} &= a_{31} \end{aligned} \tag{4.65}$$

These conditions assure that the argument of f in Eqs. (4.40) and (4.53) is the same for stages 3 and 4. Hence, the number of function evaluations is reduced by one. Further, free parameters can be determined such that several conditions of order five are satisfied. When Eq. (4.65) is satisfied, one of the nine order 5 conditions is satisfied, provided

$$a_3 = \frac{1/5 - a_2/4}{1/4 - a_2/3} \tag{4.66}$$

A second order 5 condition is satisfied by imposing the condition

$$b_4\beta_{43}a_3^2(a_3-a_2)=\frac{1}{20}-\frac{e}{4}-a_2\left(\frac{1}{12}-\frac{e}{3}\right) \qquad (4.67)$$

Next, two conditions are chosen as

$$\begin{aligned} b_3 &= 0 \\ a_2 &= 2e \end{aligned} \qquad (4.68)$$

to make the task of finding a solution more tractable. The last condition regards the choice of the diagonal element e. The value of this parameter determines the stability properties of the Rosenbrock method. Since interest is in constructing an L-stable method, the coefficient e should be taken as (Hairer and Wanner, 1996)

$$e = 0.57281606 \qquad (4.69)$$

Equations (4.55) through (4.62) and (4.64) through (4.69) comprise a system of 17 non-linear equations in 17 unknowns. The solution of this system is (Sandu, et al., 1998):

$$\begin{aligned}
e &= 0.57281606 \\
e_{21} &= -2.341993127112013949170520 \\
e_{31} &= -0.02733374654348983619650460 \\
e_{32} &= 0.2138116508366996898674720 \\
e_{41} &= -0.2590838377855102221126410 \\
e_{42} &= -0.1905958077323117516163580 \\
e_{43} &= -0.2280310359731338294777440 \\
\\
a_{21} &= 1.14563212 \\
a_{31} &= 0.5209207891306290293285160 \\
a_{32} &= 0.1342941868425048001492320 \\
a_{41} &= 0.5209207891306290293285160 \\
a_{42} &= 0.1342941868425048001492320 \\
a_{43} &= 0.0
\end{aligned} \qquad (4.70)$$

$$
\begin{aligned}
b_1 &= 0.32453470789173451347419\,6\\
b_2 &= 0.04908654478752330868463\,3\\
b_3 &= 0.0\\
b_4 &= 0.62637874732074217784117\,11
\end{aligned}
$$

These coefficients define the proposed 4-stage, L-stable, order 4 Rosenbrock-Nystrom method with 3 function evaluations per successful integration time-step. The weights of the embedded formula are computed as the solution of the linear system in Eq. (4.63),

$$
\begin{aligned}
\hat{b}_1 &= 0.52092078913062902932851\,6\\
\hat{b}_2 &= 0.14454971466536459958468\,1\\
\hat{b}_3 &= 0.12455968641470204977489\,7\\
\hat{b}_4 &= 0.20996980978930432131190\,6
\end{aligned}
\tag{4.71}
$$

In order to save one matrix-vector multiplication, the approach in Eqs. (4.49) through (4.53) is implemented. The coefficients in this formulation are obtained from the coefficients of Eqs. (4.70) and (4.71), based on Eq. (4.54). The following are the coefficients that are actually used in the numerical implementation of the proposed Rosenbrock-Nystrom method:

$$
\begin{aligned}
w_{21} &= 0.200000000000000000000000E0\\
w_{31} &= 0.186794814949823713234476E1\\
w_{32} &= 0.234445568517238850023220E0\\
w_{41} &= 0.186794814949823713234476E1\\
w_{42} &= 0.234445568517238850023220E0\\
w_{43} &= 0.0
\end{aligned}
\tag{4.72}
$$

$$
\begin{aligned}
c_{21} &= -0.713764994334997983036926E1\\
c_{31} &= 0.258092366650965771448805E1\\
c_{32} &= 0.651629887302032023387417E0\\
c_{41} &= -0.213711526650661911680637E1\\
c_{42} &= -0.321469531339951070769241E0\\
c_{43} &= -0.694966049282445225157329E0
\end{aligned}
\tag{4.73}
$$

$$\begin{aligned}
d_{21} &= -0.11963610071120139491705 2E1 \\
d_{31} &= 0.147028025440978071463387E1 \\
d_{32} &= 0.348105837679204490016704E0 \\
d_{41} &= 0.003765094355556165798974E0 \\
d_{42} &= -0.109762486758103255675398E0 \\
d_{43} &= -0.228031035973133829477744E0
\end{aligned} \tag{4.74}$$

$$\begin{aligned}
t_{21} &= 0.114563212E1 \\
t_{31} &= 0.789509162815638629626980E0 \\
t_{32} &= 0.134294186842504800149232E0 \\
t_{41} &= 0.789509162815638629626980E0 \\
t_{42} &= 0.134294186842504800149232E0 \\
t_{41} &= 0.0
\end{aligned} \tag{4.75}$$

$$\begin{aligned}
s_{1} &= 0.225556622860456524372884E1 \\
s_{2} &= 0.287055063194157607662630E0 \\
s_{3} &= 0.435311963379983213402707E0 \\
s_{4} &= 0.109350765640324780321482E1
\end{aligned} \tag{4.76}$$

$$\begin{aligned}
ev_{1} &= 0.187167068076981509470170E0 \\
ev_{2} &= 0.048373711126624809706137E0 \\
ev_{3} &= 0.071938617944591505140960E0 \\
ev_{4} &= 0.726950528467092658905657E0
\end{aligned} \tag{4.77}$$

$$\begin{aligned}
p_{1} &= 0.159275081940958534207490E1 \\
p_{2} &= 0.195938266310250609693329E0 \\
p_{3} &= 0.0 \\
p_{4} &= 0.626378747320742177841171E0
\end{aligned} \tag{4.78}$$

$$\begin{aligned}
ep_{1} &= 0.157846847561375869447 80E0 \\
ep_{2} &= -0.027040406278447759351824E0 \\
ep_{3} &= -0.124559686414702049774897E0 \\
ep_{4} &= 0.416408937531437856529265E0
\end{aligned} \tag{4.79}$$

$$\begin{aligned} e_1 &= 0.572816060E0 \\ e_2 &= -0.176917706711201394917052E1 \\ e_3 &= 0.759293964293209853670967E0 \\ e_4 &= -0.104894621490955803206743E0 \end{aligned} \tag{4.80}$$

$$\begin{aligned} a_1 &= 0.1145632120E1 \\ a_2 &= 0.655214975973133829477748E0 \\ a_3 &= 0.655214975973133829477748E0 \end{aligned} \tag{4.81}$$

In Eqs. (4.77) and (4.79), instead of providing the value of the weights $\hat{s}_i$ and $\hat{p}_i$ for the embedded method, the difference between the weights of the actual and embedded methods were computed. The coefficients ev_i and ep_i, $i = 1,\ldots,4$ are used to compute the composite error at the position and velocity levels, using the stage unknowns u_i.

4.4.4 Algorithm Pseudo-code

Pseudo-code for numerical implementation of the Rosenbrock-Nystrom method presented in the previous two Sections is provided in Table 16. Coding this algorithm requires implementation of Eqs. (4.49) through (4.53), using coefficients provided in the previous section.

The first two steps of the implementation are identical to those of the previously presented pseudo-codes. Step 4 saves the system configuration to be used upon a rejected time step. During the following two steps, the integration Jacobian is evaluated and factored. The quantity that is computed is in fact not the matrix $\mathbf{S}$ of Eq. (4.34), as the Rosenbrock formula suggests, but the matrix $\Pi \equiv (1/e^2) \cdot \hat{\mathbf{M}}\mathbf{S}$, with $\hat{\mathbf{M}}$ being the positive definite matrix of Eq. (3.16). Matrix Π is not obtained by first computing the matrices $\hat{\mathbf{M}}$ and $\mathbf{S}$, and then multiplying them. As shown in Section 3.4.1.3 (**Proposition 1**), the matrix Π is identical to the integration Jacobian $\Psi_{\ddot{\mathbf{v}}}$ of the State

Space Method. Compared to matrix $\mathbf{S}$, $\Psi_{\ddot{\mathbf{v}}}$ is easier to compute. Therefore, at each stage of the Rosenbrock formula, instead of solving systems of the form

$$\mathbf{S}\mathbf{x} = \mathbf{b}$$

the vector $\mathbf{x}$ is obtained as the solution of the equivalent system

$$\Pi\mathbf{x} = \hat{\mathbf{M}}\mathbf{b} \tag{4.82}$$

Table 16. Pseudo-code for Rosenbrock-Nystrom-Based First Order Reduction Method

```
1.   Initialize Simulation
2.   Set Integration Tolerance
3.   While (t < tend) do
4.        Set Macro-step
5.        Get Integration Jacobian
6.        Factor Integration Jacobian
7.        Get the Time Derivative
8.        Resolve Stage 1
9.        Resolve Stage 2
10.       Resolve Stage 3
11.       Resolve Stage 4
12.       Get Solution.  Check Accuracy.  Compute new Step-size
13.       Check Partition
14.  End do
```

The matrix Π is factored during Step 6. The dimension of this matrix is equal to the number of degrees of freedom of the model, and therefore is rather small. During

Step 7, the partial derivative $\partial f / \partial t$ is evaluated in the consistent configuration from the beginning of each macro-step. In the case of scleronomic mechanical systems, for which all the external applied forces are time independent, the quantity $\partial f / \partial t$ is zero. Otherwise, adequate means to provide this quantity should be embedded in the code.

The next four steps compute the stage variables $\mathbf{u}_i$, $i = 1, \ldots, 4$. The challenge is how to retrieve the right side of Eq. (4.53). During each of the four stages, some or all of the following steps are taken:

(a). Obtain a consistent configuration at position and velocity levels

(b). Compute generalized accelerations

(c). Evaluate right side of Eq. (4.53)

(d). Solve linear system to obtain the stage values $\mathbf{u}_i$

Step (a) is justified by the necessity to compute generalized accelerations required by step (c). Since the fundamental idea in DAE integration hinges upon state-space reduction, the Rosenbrock formula actually integrates independent variables only. Dependent variables must be recovered, since any call to acceleration computation requires consistent position and velocity information. Independent positions are computed based on Eq. (4.51), while independent velocities are computed at each stage, as indicated by Eq. (4.52).

After recovering dependent positions and velocities, based on kinematic constraint equations of Eqs. (3.7) and (3.8), topology based linear algebra algorithms introduced in Sections 3.4.2 and 3.4.3 are employed to compute accelerations during sub-step (c). During sub-step d), the available information is used to obtain the right side of Eq. (4.53). The actual implementation is based on the matrix Π. Therefore, one additional step is taken to multiply $\hat{\mathbf{M}}$ of Eq. (3.16) with the original right-side. As shown in Section 3.4.1.3, this approach adds one matrix-vector multiplication, but eliminates the need to explicitly solve for $\mathbf{J}_1$ and $\mathbf{J}_2$.

Due to the particular choice of coefficients defining the Rosenbrock formula and the way in which the code was implemented, each of the four stages has its own particularities. Thus,

(1). Stage 1 (Step 8 of the pseudo-code) coincides with the beginning of the current macro-step, or equivalently the end of the prior macro-step. Therefore, the system is in an assembled configuration, and sub-step a) is skipped. During this stage, the matrix Π is evaluated, and generalized accelerations are obtained cheaply as a by-product of this effort. To obtain Π, the matrix $\hat{\mathbf{M}}$ is computed, and the dependent sub-Jacobian $\Phi_{\mathbf{u}}$ is factored. The latter is needed to obtain the matrices $\mathbf{H}$, $\mathbf{J}$, $\mathbf{L}$, and $\mathbf{N}$ of Section 3.4.1.3. If these two quantities are available, $\ddot{\mathbf{v}}$ is first computed as the solution of the positive definite linear system of Eq. (3.15), and $\ddot{\mathbf{u}}$ is computed using acceleration kinematic equation of Eq. (3.9), since $\Phi_{\mathbf{u}}$ is already factored. Consequently, sub-stages (a) and (b) are effectively dealt with by stage 1. Obtaining the right-side during sub-step (c) is straightforward. Then, the matrix Π is factored, and the stage value $\mathbf{u}_1$ computed. Note that factorization of Π is used for all stages of the formula, since the diagonal elements in Butcher's tableau are identical.

(2). Stages 2 and 3 (Steps 9 and 10) simply follow the sub-stages (a) through (d) above.

(3). Stage 4 by-passes sub-stage (b), the function evaluation, due to the choice of formula defining coefficients in Eq. (4.65). Since no acceleration computation is required, there is no need to provide a consistent configuration at the position and velocity levels. Therefore sub-stage (a) is skipped. It remains to compute the right side of the linear equation and, with the coefficient matrix already factored, to do a forward/backward substitution to obtain $\mathbf{u}_4$.

Once all the stage values $\mathbf{u}_i$, $i = 1, \ldots, 4$ are available, the solution at the new grid point is computed during Step 12, based on Eqs. (4.49) and (4.50). Extra effort goes into recovering dependent positions and velocities to obtain a consistent configuration of the mechanism, which is used during the next time step, at stage 1, sub-stage (a).

Based on a second approximation of the solution at the new grid point given by the embedded formula, values $\hat{\mathbf{y}}_1$ and $\hat{\mathbf{y}}_1'$ of Eqs. (4.49) and (4.50), the accuracy of the solution is analyzed, and the step is accepted or rejected. As a result of error analysis, a new step-size is provided. This is used to recompute the current time step upon a rejected step, or to proceed to the next time step upon a successful step.

Finally, the last step of the algorithm checks the partitioning. Based on the condition number of the dependent sub-Jacobian $\Phi_{\mathbf{u}}$, which is available on exit from Step 12, and the value of the repartitioning coefficient α previously introduced, the partitioning can be rejected or preserved for the next time step.

The most important feature of any Rosenbrock formula is its ability to provide accuracy and stability without the penalty of having to solve discretized non-linear algebraic equations. No iterative process is embedded in the formula, and the delicate issue of stopping criteria is eliminated. It is recognized in the numerical analysis community that implementation of a Rosenbrock formula is significantly simpler than any other Runge-Kutta type formula.

The factor that has limited extensive use of the attractive class of Rosenbrock formulas is the requirement that the integration Jacobian should be exact. This is a limiting factor for many engineering applications that are likely to contain complex components for which computing exact Jacobian is an intractable task. In the context of the dynamic analysis of multibody systems, the limiting factor remains the task of providing force derivatives. In situations such as these, when no close/analytic expressions are available to compute required derivative information, the solution is either to use automatic differentiation tools such as ADIFOR (Bischof et al., 1994), or to use numerical means to obtain derivative information. In the latter case however, using a Rosenbrock type formula is no longer an option, and the only alternative is to use more robust and somehow more CPU-expensive methods. One of these methods is the SDIRK4/15 formula that is discussed in the next Section.

4.5 SDIRK-Based First Order Implicit Integration

4.5.1 SDIRK4/15

According to Hairer and Wanner (1996), the choice $\gamma = 4/15$ instead of $\gamma = 4/16$ as the diagonal element of a 5 stage, order 4 stiffly-accurate SDIRK method is numerically superior. This motivated their effort to code this formula, rather than the one proposed in Section 4.2 in conjunction with the Descriptor Form Method. The resulting code is much more refined, and special attention was paid to issues such as stopping criteria, mechanisms for early detection of convergence failure, estimation of iteration starting points, avoiding rounding errors, etc.

One positive feature of the First Order Reduction Method is its ability to link to any standard code for the numerical solution of stiff ODE. This feature is exploited here by embedding a public domain ODE code of Hairer and Wanner in the framework of the First Order Reduction Method. The objective is to keep as much as possible of the original layout that made the code of Hairer and Wanner robust and efficient. There is no major difference between the SDIRK formula of this Section, and the one defined in Section 4.2.2, so the notation of that Section is retained.

Simplified order conditions for SDIRK4/15 are provided in Eq. (4.10), and values for p_i, $i = 1,\ldots,8$, are obtained by setting $\gamma = 4/15$. The SDIRK4/15 formula is chosen to be stiffly-stable, and the particular choice of γ implies that it is A-stable (Hairer and Wanner, 1996). Consequently, it is L-stable. As in Section 4.2.2, in order to minimize fifth-order error terms, the same values $c'_2 = 0.5$ and $c'_3 = 0.3$ are considered for the two extra coefficients defining the integration formula. After solving the non-linear system in Eq. (4.10), the coefficients of the formula are obtained as

Stage 1	$a_{11} = 4/15$
Stage 2	$a_{21} = 1/2$
	$a_{22} = 4/15$
Stage 3	$a_{31} = 51069/144200$
	$a_{32} = -7809/144200$
	$a_{33} = 4/15$
Stage 4	$a_{41} = 12047244770625658/141474406359725325$
	$a_{42} = -3057890203562191/47158135453241775$
	$a_{43} = 2239631894905804/28294881271945065$
	$a_{44} = 4/15$
Stage 5	$a_{51} = 181513/86430$
	$a_{52} = -89074/116015$
	$a_{53} = 83636/34851$
	$a_{54} = -69863904375173/23297141763930$
	$a_{55} = 4/15$

Since the formula is designed to be stiffly stable, $b_i = a_{5i}$, $i = 1,\ldots,5$. The coefficients c_i are obtained as

$$c_i = \sum_{j=1}^{i} a_{ij} \ , \ \ i = 1,\ldots,5$$

It remains to provide an embedded formula for step-size control. Much like the case of SDIRK4/16, the original order conditions are imposed up to order 3 to construct a formula that uses the same values of a_{ij} and c_i. The order conditions for the embedded formula are given in Eq. (4.11). With the values of a_{ij} given above, the linear system that gives the weights $\hat{b}_i$ for the embedded order 3 formula is

$$\hat{b}_1 + \hat{b}_2 + \hat{b}_3 + \hat{b}_4 = 1$$

$$\frac{4}{15}\hat{b}_1 + \frac{23}{30}\hat{b}_2 + \frac{17}{30}\hat{b}_3 + \frac{707}{1931}\hat{b}_4 = \frac{1}{2}$$

$$\frac{16}{225}\hat{b}_1 + \frac{529}{900}\hat{b}_2 + \frac{289}{900}\hat{b}_3 + \frac{499849}{3728761}\hat{b}_4 = \frac{1}{3}$$

$$\frac{16}{225}\hat{b}_1 + \frac{76}{225}\hat{b}_2 + \frac{529591}{2595600}\hat{b}_3 + \frac{8356414509}{72360335966}\hat{b}_4 = \frac{1}{6}$$

The weights $\hat{b}_i$ obtained by solving this system are

$$\hat{b}_1 = \frac{33665407}{11668050}$$

$$\hat{b}_2 = \frac{-2284766}{15662025}$$

$$\hat{b}_3 = \frac{11244716}{4704885} \tag{4.83}$$

$$\hat{b}_4 = \frac{-96203066666797}{23297141763930}$$

$$\hat{b}_5 = 0$$

To reduce the influence of round-off errors during numerical integration of the IVP $y' = f(x, y)$, $y(x_0) = y_0$, Hairer and Wanner in their code preferred to work at each stage with the quantities

$$z_i = h\sum_{j=1}^{i} a_{ij} f(x_0 + c_j h, y_0 + z_j), \quad i = 1, \ldots, s \tag{4.84}$$

Whenever the solution $z_1, z_2, \ldots, z_s$ of the system in Eq. (4.84) is known, the stage values k_i are obtained as

$$k_i = f(x_0 + c_i h, y_0 + z_i)$$

This assumes s-extra function evaluations. The additional effort can be avoided if the matrix $\mathbf{A} = (a_{ij})$; i. e., the matrix of a_{ij} coefficients in Butcher's tableau, is non-singular. Since this is the case with any SDIRK formula, the system in Eq. (4.84) is rewritten as

$$\begin{pmatrix} z_1 \\ \vdots \\ z_s \end{pmatrix} = \mathbf{A} \begin{pmatrix} hf(x_0 + c_1 h, y_0 + z_1) \\ \vdots \\ hf(x_0 + c_s h, y_0 + z_s) \end{pmatrix} \tag{4.85}$$

The solution at the new grid point is obtained as

$$y_1 = y_0 + \sum_{i=1}^{s} d_i z_i$$

where,

$$(d_1, \ldots, d_s) = (b_1, \ldots, b_s) \cdot \mathbf{A}^{-1}$$

One attractive feature of the z-approach is that, since SDIRK4/15 is stiffly stable, the vector d is simply $(0,0,0,0,1)$. Another advantage of using the z-approach is the following. The quantities $z_1, z_2, \ldots, z_s$ are computed iteratively and therefore affected by small iteration errors. The evaluation of k_i; i. e., evaluations of the form $f(x_0 + c_i h, y_0 + z_i)$ would then, due to the large Lipschitz constant of f, amplify these errors. This could be disastrously inaccurate for a stiff problem (Shampine, 1980).

A consequence of the z-approach is that expressions for stage values k_i, $i = 1,\dots,s$ must be replaced by linear combinations of z_i, $i = 1,\dots,s$, as induced by Eq. (4.85). Accordingly, the step-size controller needs to be modified to accommodate the z-representation. In other words, as the vector b of weights is replaced by the d vector, the weights of the embedded method must be modified accordingly. The second approximation of the solution $\hat{y}_1$ is obtained as

$$\hat{y}_1 = y_0 + \sum_{i=1}^{s} \hat{d}_i z_i$$

where,

$$(\hat{d}_1,\dots,\hat{d}_s) = (\hat{b}_1,\dots,\hat{b}_s)\cdot \mathbf{A}^{-1}$$

Since the actual quantity of interest is the approximation $y_1 - \hat{y}_1$ of the local truncation error, this is obtained as

$$err = (b - \hat{b})\mathbf{A}^{-1} z \equiv \bar{d} z \tag{4.86}$$

where $\bar{d} = (b - \hat{b})\mathbf{A}^{-1}$. Based on the values of b, $\hat{b}$, and $\mathbf{A}$, the vector $\bar{d}$ is obtained as

$$\begin{aligned}
\bar{d}_1 &= \frac{-7752107607}{11393456128} \\
\bar{d}_2 &= \frac{17881415427}{11470078208} \\
\bar{d}_3 &= \frac{-2433277665}{179459416} \\
\bar{d}_4 &= \frac{96203066666797}{6212571137048} \\
\bar{d}_5 &= 1
\end{aligned} \tag{4.87}$$

The original code written by Hairer and Wanner also provided dense output formulas. Without going into details regarding the process of obtaining these coefficients, they are

$$\bar{d}_{11} = 24.74416644927758$$
$$\bar{d}_{12} = -4.325375951824688$$
$$\bar{d}_{13} = 41.39683763286316$$
$$\bar{d}_{14} = -61.04144619901784$$
$$\bar{d}_{15} = -3.391332232917013$$

$$\bar{d}_{21} = -51.98245719616925$$
$$\bar{d}_{22} = 10.52501981094525$$
$$\bar{d}_{23} = -154.2067922191855$$
$$\bar{d}_{24} = 214.3082125319825$$
$$\bar{d}_{25} = 14.71166018088679$$

$$\bar{d}_{31} = 33.14347947522142$$
$$\bar{d}_{32} = -19.72986789558523$$
$$\bar{d}_{33} = 230.4878502285804$$
$$\bar{d}_{34} = -287.6629744338197$$
$$\bar{d}_{35} = -18.99932366302254$$

$$\bar{d}_{41} = -5.905188728329743$$
$$\bar{d}_{42} = 13.53022403646467$$
$$\bar{d}_{43} = -117.6778956422581$$
$$\bar{d}_{44} = 134.3962081008550$$
$$\bar{d}_{45} = 8.678995715052762$$

The way in which these coefficients are computed was discussed in Section 4.2.2, when defining dense output for SDIRK4/16. These coefficients were scaled by $\mathbf{A}^{-1}$ to account for the z_i rather the k_i implementation of the formula. For $i = 1,\ldots,5$, with

$$\bar{b}_i(\theta) = \sum_{j=1}^{4} \bar{d}_{ji}\theta^j$$

the solution at an intermediate point $x_0 + \theta h$, $0 < \theta \le 1$, is given by

$$y(x_0 + \theta h) \approx y_0 + \sum_{i=1}^{5} \overline{b_i}(\theta) z_i$$

which is of order 3 for $0 < \theta < 1$, and updates to the fourth order approximation y_1 for $\theta = 1$.

4.5.2 Algorithm Pseudo-code

The SDIRK4/15 formula was embedded in the First Order Reduction Method to provide a robust and efficient algorithm. The numerical implementation is based on a public domain code developed by Hairer and Wanner for the integration of stiff IVP, which was adapted to support the proposed algorithm.

An obstacle to immediate conversion of the algorithm is the need to embed dependent variable recovery in the original code. Likewise, the SSODE obtained in Multibody Dynamics is of second order, and the original code was designed for first order systems with tailored linear algebra, iteration starting values, and stopping criteria. The pseudo-code of the resulting algorithm is presented in Table 17.

The first four steps of the implementation are the same as discussed in Section 4.1.2. At Step 5, the integration Jacobian is evaluated. The quantity that is computed is the matrix Π , which is used to carry out the iterative process (Steps 9 through 16). The next step factorizes Π . Dense Lapack routines are used for this operation, since the dimension of the matrix is low and sparsity is not a factor.

Table 17. Pseudo-code for SDIRK4/15-Based First Order Reduction Method

```
1.  Initialize Simulation
2.  Set Integration Tolerance
3.  While (t < tend) do
4.          Set Macro-Step
5.          Get Integration Jacobian
6.          Factor Integration Jacobian
7.          For stage from 1 to 5 do
8.                  Set up Stage
9.                  While (.NOT. converged) do
10.                         Evaluate Accelerations
11.                         Build RHS
12.                         Get Correction in Independent Positions
13.                         Get Correction in Independent Velocities
14.                         Analyze Convergence
15.                         Recover Dependent Positions
16.                         Recover Dependent Velocities
17.                 End do
18.         End do
19.         Check Accuracy.  Compute new Step-size.
20.         Check Partition
21. End do
```

Step 7 starts the loop of the stages of the formula. First, the stage is set up at Step 8. The code provides initial iteration estimates, based on information available from prior stages. When setting up the last stage, the starting point is provided by the embedded formula that uses information only from first four stages. The embedded formula is of order 3, so this should be a good approximation of the solution for stage 5. This idea works because SDIRK4/15 is stiffly-accurate, and the last stage initial prediction can be taken to be the solution provided by the embedded method.

The iterative process for recovering z_i starts at Step 9. The process adopted is described in Section 3.4.1.3. To see how the z_i-formulation relates to the iterative method described in that section, first note that in the case of the SSODE, following the notations of Section 3.4.1.1,

$$\mathbf{z}_i = h\sum_{j=1}^{i} a_{ij}\mathbf{g}(x_0 + c_j h, \mathbf{w}_0 + \mathbf{z}_j) \qquad (4.88)$$

In Section 3.4.1.1, the second order SSODE obtained after DAE reduction was considered to assume the form $\dot{\mathbf{w}} = \mathbf{g}(t, \mathbf{w})$, with $\mathbf{w} \equiv [\mathbf{v}^{\mathrm{T}}\ \dot{\mathbf{v}}^{\mathrm{T}}]^{\mathrm{T}}$. The stage values

$$\mathbf{z}_i \equiv \begin{bmatrix} \mathbf{v}^{(i)} \\ \dot{\mathbf{v}}^{(i)} \end{bmatrix}$$

are obtained as the solution of the non-linear system of Eq. (4.88). The quasi-Newton algorithm employed requires computation of the integration Jacobian, of Eq. (3.87). The corrections in $\mathbf{z}_i$; i. e., in $\mathbf{v}^{(i)}$ and $\dot{\mathbf{v}}^{(i)}$, are given in Eqs. (3.101) and (3.102). A computational advantage is obtained if, instead of explicitly computing the derivatives $\mathbf{J}_1$ and $\mathbf{J}_2$, the matrix Π is introduced, and appropriate alterations are made in the right side of the linear system. In light of Eq. (4.88), the coefficients $\mathbf{b}_1$ and $\mathbf{b}_2$ at stage i, iteration k, are

$$\mathbf{b}_1^{(i,k)} = h\sum_{j=1}^{i-1} a_{ij}(\dot{\mathbf{v}}_0 + \dot{\mathbf{v}}^{(j)}) + \gamma h(\dot{\mathbf{v}}_0 + \dot{\mathbf{v}}^{(i,k)})$$

$$\mathbf{b}_2^{(i,k)} = h\sum_{j=1}^{i-1} a_{ij}\ddot{\mathbf{v}}^{(j)} + \gamma h\ddot{\mathbf{v}}^{(i,k)}$$

where $\ddot{\mathbf{v}}^{(j)}$ represents independent accelerations at stage j, corresponding to the configuration $\mathbf{v}_0 + \mathbf{v}^{(j)}$ and $\dot{\mathbf{v}}_0 + \dot{\mathbf{v}}^{(j)}$, with $\mathbf{v}_0$ and $\dot{\mathbf{v}}_0$ being independent positions and velocities at the beginning of the macro-step. Thus, for $j = 1,\ldots,i-1$, since all past information is available, the first terms in the expressions for $\mathbf{b}_1^{(i,k)}$ and $\mathbf{b}_2^{(i,k)}$ can be immediately obtained. They are held constant during the iterative process. To evaluate the last part of the right side of these two expressions, $\gamma = 4/15$, h is the step-size, and the value $\dot{\mathbf{v}}^{(i,k)}$ is provided by the iterative process. Discussion in Sections 3.4.2 and 3.4.3 showed how to compute generalized accelerations $\ddot{\mathbf{v}}^{(i,k)}$ at a given system configuration.

The remaining steps of the pseudo-code manipulate data according to the considerations above. During stage i, $i = 1,\ldots,5$, Step 10 computes at each iteration k, the independent accelerations $\ddot{\mathbf{v}}^{(i,k)}$. The quantities $\mathbf{b}_1^{(i,k)}$ and $\mathbf{b}_2^{(i,k)}$ are computed in Step 11, while during Step 12 corrections in $\mathbf{v}^{(i)}$ and $\dot{\mathbf{v}}^{(i)}$ are made using the matrix Π, according to considerations presented in Section 3.4.1.3.

Step 13 contains a sophisticated mechanism that does the following:

(a). Convergence rate analysis

(b). Stops the iterative process based on stopping criteria

(c). Convergence forecast

These functions are closely related to estimation of iteration error. Since convergence of the quasi-Newton algorithm (steps 9 through 16) is linear, corrections at two consecutive iterations satisfy

$$|\delta \mathbf{z}^{(i,k+1)}| \leq \theta |\delta \mathbf{z}^{(i,k)}| \tag{4.89}$$

with $\theta < 1$. With $\mathbf{z}^{(i)}$ the exact solution of the non-linear system in Eq. (4.88), applying the triangle inequality yields

$$\mathbf{z}^{(i,k+1)} - \mathbf{z}^{(i)} = (\mathbf{z}^{(i,k+1)} - \mathbf{z}^{(i,k+2)}) + (\mathbf{z}^{(i,k+2)} - \mathbf{z}^{(i,k+3)}) + \ldots$$

and taking into account Eq. (4.89) yields

$$|\mathbf{z}^{(i,k+1)} - \mathbf{z}^{(i)}| \leq \frac{\theta}{1-\theta} |\delta \mathbf{z}^{(i,k)}| \tag{4.90}$$

An estimation of the convergence rate at iteration k is available as

$$\theta_k = |\delta \mathbf{z}^{(i,k)}| / |\delta \mathbf{z}^{(i,k-1)}|$$

It is clear that the iteration error should not be larger than the local discretization error, which because of the step-size control mechanism is kept close to Tol (the integration tolerance computed based on the accuracy requirements imposed by the user). Therefore taking into account Eq. (4.90), iteration is stopped when, with $\eta_k = \theta_k / (1 - \theta_k)$,

$$\eta_k |\delta \mathbf{z}^{(i,k)}| \leq \kappa \cdot Tol \tag{4.91}$$

The value $\mathbf{z}^{(i,k)}$ is then accepted as the approximation of $\mathbf{z}^{(i)}$.

This strategy can be applied after at least 2 iterations. In order to be able to apply this stopping criteria even after the first iteration, for $k = 0$ the quantity η_0 is defined as

$$\eta_0 = (\max(\eta_{old}, uround))^{0.8}$$

where η_{old} is the last η_k of the preceding step. It remains to make a good choice of the parameter κ in Eq. (4.91). After extensive numerical experiments with values between 10 and 10^{-4}, Hairer and Wanner (1996) recommended a value of κ in the range 10^{-1} and 10^{-2}.

It remains to address point (c) above, which attempts to avoid unjustified computational effort in an iterative process that appears at some point to be doomed to fail. Usually, $k_{\max} = 7$ to 10 iterations are allowed. However, the computation is stopped and the step rejected as soon as one of the following conditions holds

(c1). There is a k for which $\theta_k \geq 1$ (the iteration is expected to diverge)

(c2). For some k,

$$\frac{(\theta_k)^{k_{\max}-k}}{1-\theta_k} |\delta \mathbf{z}^{(i,k)}| > \kappa \cdot Tol$$

The left side of the last expression is an estimate of the iteration error to be expected after $k_{\max} - 1$ iterations. Whenever the step is rejected because of (c1) or (c2), integration is restarted with a smaller step-size, usually $h/2$.

If the iteration process has not converged, and if, based on observations obtained in Step 13, there is no apparent sign for future convergence failure, the value $\mathbf{z}^{(i,k)}$ is corrected to produce the next approximation $\mathbf{z}^{(i,k+1)}$. During Steps 15 and 16, dependent positions and velocities corresponding to newly computed independent positions and velocities are obtained. Both these steps in the actual code are based on iterative processes. Most importantly, they use the same matrix, namely the constraint sub-Jacobian $\Phi_{\mathbf{u}}$ that was factored in the configuration at the beginning of the macro-step.

Once all the stages have been completed, accuracy of the solution is verified. Since the integration formula used is stiffly-accurate, stage 5 provides the solution at the new time step. An approximation of the local error is computed, based on the coefficients $\overline{d}_i$ of Eq. (4.87). The step-size selection process used was detailed in Section 4.2.1. Step 20 checks the validity of the partitioning, as was described for trapezoidal method used in conjunction with the State-Space Reduction Method in Section 4.1.2. Finally, Step 21 is the end of the simulation loop.

CHAPTER 5
NUMERICAL EXPERIMENTS

5.1 Validation of First Order Reduction Method

This Section focuses on validation of the codes implemented in conjunction with the First Order Reduction Method of Section 4.4. Validation of the code based on the State-Space Reduction Method of Section 4.1 is presented by Haug, Negrut, and Iancu (1997a) and Negrut, Haug, and Iancu (1997). Codes based on the Descriptor From Method presented in Sections 4.2 and 4.3 are validated in the work of Haug, Negrut, and Engstler (1998).

The double pendulum shown in Figure 11 is the model used for validation purposes. It is a two-body, two-degree-of-freedom planar mechanical system. The mechanism is modeled using planar Cartesian coordinates; i. e., the x and y coordinates of the body center of the mass and the angle θ that defines the orientation of the local centroidal reference frames with respect to the global reference frame. A large amount of stiffness is induced by means of two rotational spring-damper-actuators (RSDA). The parameters of the model are provided in SI units in Table 18.

Table 18. Parameters for the Double Pendulum

L_1	m_1	k_1	C_1	L_2	m_2	k_2	c_2
1.0	3.0	400	15.0	1.5	0.3	3.0E5	5.0E4

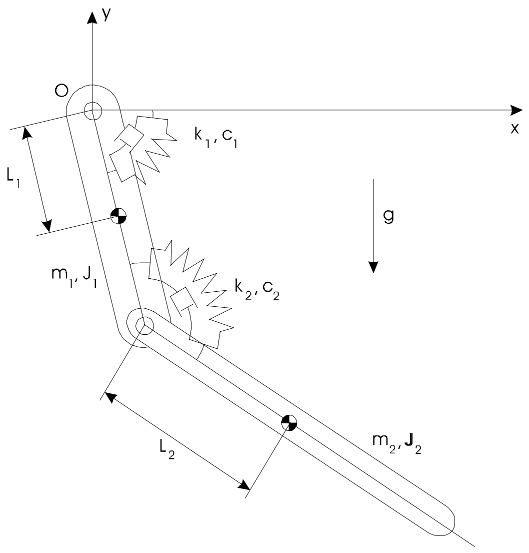


Figure 11. Double Pendulum

The double pendulum problem analyzed is on a scale starting from mildly stiff, and ending with extremely stiff, somewhere in between, with a dominant eigenvalue that has a small imaginary part, and real part of the order of -10^5.

Initial condition for the problem are given in Table 19. The first row contains position information and the second contains velocity information.

Table 19. Initial Conditions for Double Pendulum

Body 1			Body 2		
x-coordinate	*y-coordinate*	*θ-coordinate*	*x-coordinate*	*y-coordinate*	*θ-coordinate*
1.0	0.0	2π	3.4488887	-0.388228	$23\pi/12$
0.0	0.0	0.0	0.0	0.0	10.0

For validation purposes, several tolerances are imposed, and simulation results are analyzed to see if the imposed accuracy requirements are met. A reference solution is first generated by imposing a very tight tolerance. All simulations are compared to a reference simulation, to find the infinity norm of the error, the time at which it occurred, and average error per time step.

A second code that compares simulation results has been developed. This post-processing tool reads results of the current simulation and fits each time step (grid-point) between grid points of the reference simulation. The latter points are expected to be in much larger number, due to stringent accuracy with which this simulation is run. Spline cubic interpolation is used to generate a reference value at each grid point of the current simulation, based on reference simulation data. The reference value and the current value at each grid point are then compared. This comparison is done for all grid points of the current simulation, and the largest difference in solutions is defined as the infinity norm of the error. This quantity, along with the grid point at which it occurred, is reported by the post-processing tool.

The average error per time step is obtained by summing the square simulation errors at each grid point, and dividing the square root of this sum by the number of time steps taken during the simulation.

Suppose that n time steps are taken during the current simulation, and the variable used for error analysis is denoted by e. The grid points of the current simulation are denoted by $t_{init} = t_1 < t_2 < \ldots < t_n = t_{end}$, and results of the current simulation are obtained as e_i, for $1 \le i \le n$. If N is the number of time steps taken during reference simulation, it is expected that $N >> n$. Let $T_{init} = T_1 < \ldots < T_N = T_{end}$ be the simulation time steps, and E_j, for $1 \le j \le N$ be the corresponding reference values. For each i, $1 \le i \le n$, an integer $\mathbf{r}(i)$ is defined such that $T_{\mathbf{r}(i)} \le t_i \le T_{\mathbf{r}(i)+1}$. Based on reference data

$E_{\mathbf{r}(i)-1}$, $E_{\mathbf{r}(i)}$, $E_{\mathbf{r}(i)+1}$, and $E_{\mathbf{r}(i)+2}$, spline cubic interpolation is used to generate an interpolated value E_i^* at time t_i. If $\mathbf{r}(i)-1 \leq 0$, the first four reference points are considered for interpolation, while if $\mathbf{r}(i)+2 \geq N$, the last four reference points are considered for interpolation. The error at time step i is defined as

$$\Delta_i = | E_i^* - e_i | \tag{5.1}$$

The infinity norm of the simulation error is defined as

$$\Delta^{(k)} = \max_{1 \leq i \leq n} \Delta_i \tag{5.2}$$

where k denotes the tolerance set for the current simulation. The average error per time step is defined as

$$\overline{\Delta}^{(k)} = \frac{1}{n} \sqrt{\sum_{i=1}^{n} \Delta_i^2} \tag{5.3}$$

The code *ForRosen* is an implementation of the Rosenbrock-Nystrom formula discussed in Section 4.4. *ForSDIRK* is an implementation of SDIRK4/15 formula of Section 4.5.1. The codes are run with tolerances between 10^{-2} and 10^{-5}, and results are compared to the reference solution.

Figure 12 presents the time variation of orientation angle θ_1, for which error analysis is carried out. The length of the reference simulation; i. e., the length for which error analysis is carried out, is $T = 2$ seconds. Reference data are generated using the code *ForSDIRK*. With an integration tolerance set to 10^{-8}, the code takes 354,090 successful time steps to run 2 seconds of simulation.

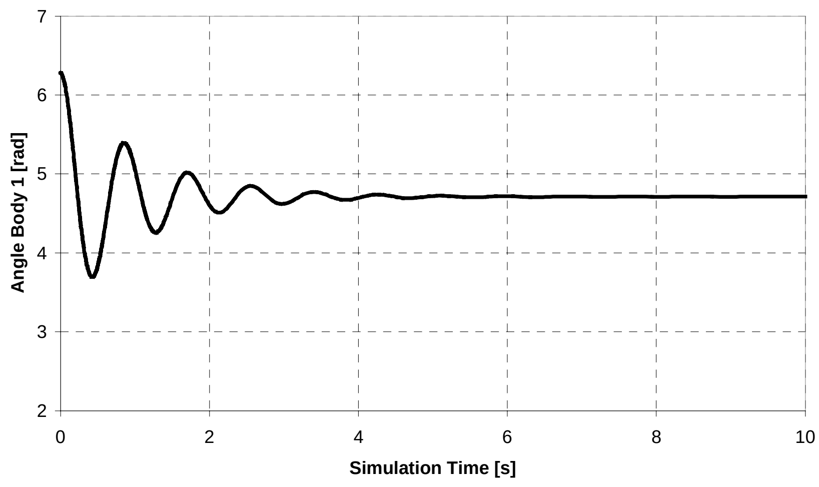


Figure 12. Orientation Body 1

Table 20 contains results of error analysis at the position level for the code *ForSDIRK*. The first column contains the value of the tolerance with which the simulation was run. Relative and absolute tolerances ($Atol_i$ and $Rtol_i$ of Eq. (4.6)) are set to 10^k, and they apply for both position and velocity. The second column contains the time t^* at which the largest error Δ_i of Eq. (5.2) occurred. The third column contains $\Delta^{(k)}$. Column four contains the relative error, defined as

$$\text{RelErr} = \frac{\Delta^{(k)}}{E^*} \times 100.0 \tag{5.4}$$

where E^* is the reference value evaluated via cubic spline interpolation, at time t^*. Equation (5.4) holds, provided E^* is not too close to zero. Finally, the last column contains the average error per time step, as defined in Eq. (5.3).

The most relevant information for method validation is $\Delta^{(k)}$. If $k = -3$; i. e., accuracy of 10^{-3} is demanded, $\Delta^{(-3)}$ should have this order of magnitude. It can be seen in Table 20 that this is the case for all tolerances, except for the case $k = -5$, when however the magnitude of the error is close to order 10^{-5}. One reason for which the value of k is not always reflected exactly in the value of $\Delta^{(k)}$ is that the relative tolerance comes into play. For this experiment, the relative tolerance is not zero. In light of Eq. (4.6), depending on the magnitude of the variable being analyzed, it loosens (for large magnitudes) or tightens the step-size control. Based on results shown in Figure 12, the relative tolerance is multiplied by a value that oscillates between 4.0 and 6.0. Therefore, the actual upper bound of accuracy imposed on solution (according to Eq. (4.6)) fluctuates and reaches values up to $7\cdot10^{-k}$.

Table 20. Position Error Analysis *ForSDIRK*

k	t	$\Delta^{(k)}$	RelErr[%]	$\overline{\Delta}^{(k)}$
-2	0.413557	0.032796853043	0.887234318934	0.0011473218118
-3	0.425395	0.003786644530	0.102567339140	0.0001111232537
-4	0.351082	0.000754569135	0.019573880291	0.0000196564539
-5	1.084304	0.000170608670	0.003625286505	0.0000042107989

The results shown in Figure 12 and Table 20 confirm the reliability of the step-size controller embedded in *ForSDIRK*, and indicate it as being slightly optimistic in step-size prediction. This is not the case with the code *ForRosen*, for which error analysis results are provided in Table 21. The step-size controller for this method is conservative, and this has a negative impact on the CPU performance of the method, especially for high accuracy requirements, as shown in Section 5.3. The step-size controller is for all situations almost one order of magnitude too stringent; i. e., this code is quite conservative.

Table 21. Position Error Analysis *ForRosen*

k	t	$\Delta^{(k)}$	RelErr[%]	$\overline{\Delta}^{(k)}$
-2	0.592127	0.005223114419	0.121266541360	0.0002292862384
-3	0.599954	0.000419882301	0.009640229496	0.0000138889609
-4	0.626135	0.000049169233	0.001087817377	0.0000011485216
-5	1.065146	0.000019027152	0.000397621626	0.0000002516347

The reason for which *ForRosen* is conservative can be explained using Dahlquist's test problem $y' = \lambda y$, $y(x_0) = y_0$. For large values of step-size h, as λ assumes large real negative values, $h\lambda \to -\infty$, and the local truncation obtained using the embedded method is approximately (Hairer and Wanner, 1996)

$$\hat{y}_1 - y_1 \approx \gamma h \lambda y_0 \tag{5.5}$$

where γ is a coefficient depending on the integration formula considered.

The truncation error in Eq. (5.5) is large in magnitude, and it increases as $h \cdot |\lambda|$ increases. Therefore, this mechanism for step-size control ceases to be effective when the problem is very stiff, or when due to loose tolerances, the step size becomes large. These considerations explain very accurately the behavior of *ForRosen*, and why the looser the tolerance, the more conservative the results.

Shampine suggests using the quantity $(1 - h\gamma\lambda)^{-1}(\hat{y}_1 - y_1)$ instead the $\hat{y}_1 - y_1$ for step-size control purposes. For the general IVP $\mathbf{y}' = f(t, \mathbf{y})$, $\mathbf{y}(x_0) = \mathbf{y}_0$, the quantity considered is

$$\mathbf{err} = (\mathbf{I} - h\gamma\mathbf{J})^{-1}(\hat{\mathbf{y}}_1 - \mathbf{y}_1) \tag{5.6}$$

where $\mathbf{J} = \partial f / \partial \mathbf{y}$ is evaluated at x_0, and $\mathbf{I}$ is the identity matrix of appropriate dimension. The factorization of the matrix $(1/h\gamma)\,\mathbf{I} - \mathbf{J}$ is available, since it is used to compute stage variables $\mathbf{k}_i$. Therefore, $\mathbf{err}$ is cheap to compute. Especially for very stiff systems, the idea of Shampine eliminates conservatism of the step-size controller. This approach, initially in the code of Hairer and Wanner and inherited by *ForSDIRK*, is currently not implemented in *ForRosen*.

The results in Tables 20 and 21 indicate that theoretical predictions are confirmed by numerical results, and they underline the importance of the step-size controller. A code with an optimistic step-size controller, may not attain the required accuracy, while using an integrator that is too pessimistic in terms of step-size control results in excessive computational effort.

Error analysis is also performed at the velocity level. The time variation of angular velocity $\dot{\theta}_1$ of body 1 is shown in Figure 13.

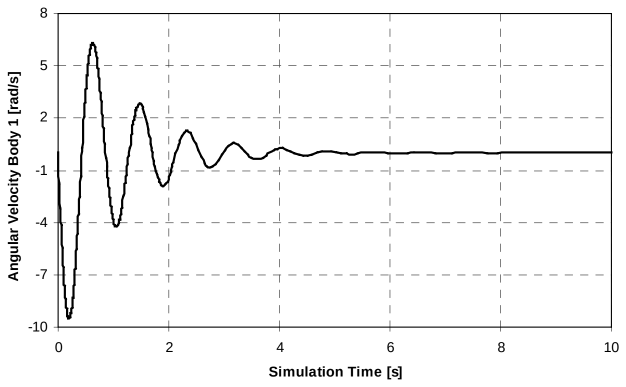


Figure 13. Angular Velocity Body 1

The angular velocity of body 1 fluctuates between –10 and 7 rad/s. The absolute and relative tolerances were set to 10^k, $k = -2,\ldots,-5$, for all numerical experiments. The order of magnitude for $\Delta^{(k)}$ is expected to be 10^{k+1}.

Results in Table 22 are obtained with *ForSDIRK*. They confirm theoretical predictions. The optimistic tendency of the step-size controller embedded in *ForSDIRK* can be seen in these results.

Table 22. Velocity Error Analysis *ForSDIRK*

k	t	$\Delta^{(k)}$	RelErr[%]	$\overline{\Delta}^{(k)}$
-2	0.578444	0.228908866004	3.840899692349	0.0083749741182
-3	0.291094	0.031312203070	0.427597266808	0.0009066499320
-4	0.551010	0.006406931668	0.118822262986	0.0001530648603
-5	0.136235	0.001485443259	0.017255571544	0.0000350200500

Results obtained at the velocity level with *ForRosen* are presented in Table 23 and they confirm the observations made earlier about this code. The step-size controller is slightly conservative. Instead of values of 10^{-k+1}, the accuracy of the results as indicated by $\Delta^{(k)}$ is one order of magnitude better, most of the time.

Results presented in this Section validate the codes and confirm reliability of the step-size controllers embedded. With one slightly on the optimistic side (*ForSDIRK*), and one on the conservative side (*ForRosen*), the step size controllers show that the idea of using an embedded formula for local error estimation is sound. The accuracy obtained with these codes is good. It remains to adapt the step-size controller for *ForRosen,* according to Eq. (5.6), to avoid unjustified CPU penalties.

Table 23. Velocity Error Analysis *ForRosen*

k	t	$\Delta^{(k)}$	RelErr[%]	$\overline{\Delta}^{(k)}$
-2	0.795548	0.040612124932	1.844343795194	0.0016645922425
-3	0.373114	0.003792150415	0.123400255245	0.0001151519879
-4	0.217757	0.000865201346	0.009222397857	0.0000134313788
-5	0.186183	0.000234309801	0.002467113637	0.0000023859574

5.2 Explicit versus Implicit Integration

This Section contains a comparison, in terms of CPU time, of two different numerical methods used for dynamic analysis of a High Mobility Multipurpose Wheeled Vehicle (HMMWV). A picture of the vehicle is provided in Figure 14. The topology of the mechanism is presented in Figure 15.

The vehicle is modeled using the same bodies as in Section 3.4.3.5, but the body numbering of that Section is changed. Since 14 bodies are used to model this vehicle, in what follows the model is referred as HMMWV14. More details about HMMWV specific parameters are provided by Serban, Negrut, and Haug (1998). Finally, all timing results in this and the following sections are obtained on a SGI Onyx machine with eight R10000 processors.

Figure 14. US Army HMMWV

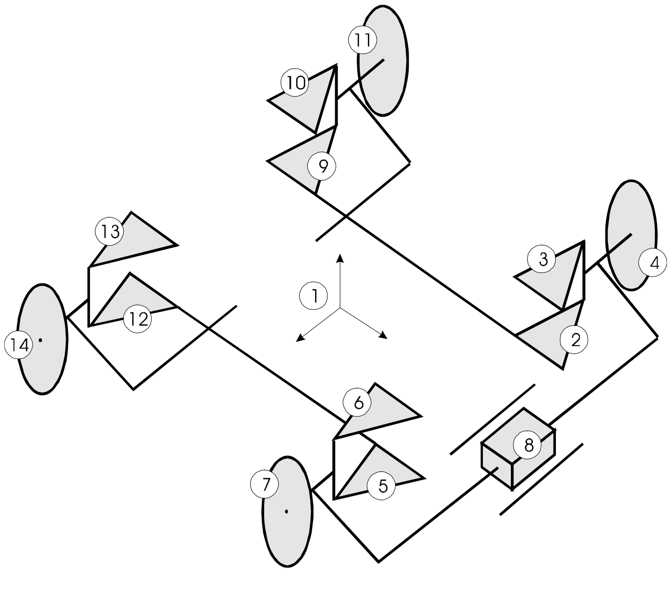


Figure 15. 14 Body Model of HMMWV

Figure 16(a) contains the original topology graph of HMMWV14. Since this model is not stiff, stiffness is added by replacing revolute joints between upper control arms and the chassis with spherical joints. Each joint replacement results in two additional degrees of freedom. For each spherical joint, two Translational-Spring-Damper-Actuators (TSDA,) acting in complementary directions model bushings that control the extra degrees of freedom. The stiffness coefficient of each TSDA is $2.0 \cdot 10^7$ N/m, while the damping coefficient is $2.0 \cdot 10^6$ Ns/m. Tires are modeled as vertical TSDA elements with stiffness coefficient 296325 N/m, and damping coefficient 3502 Ns/m.

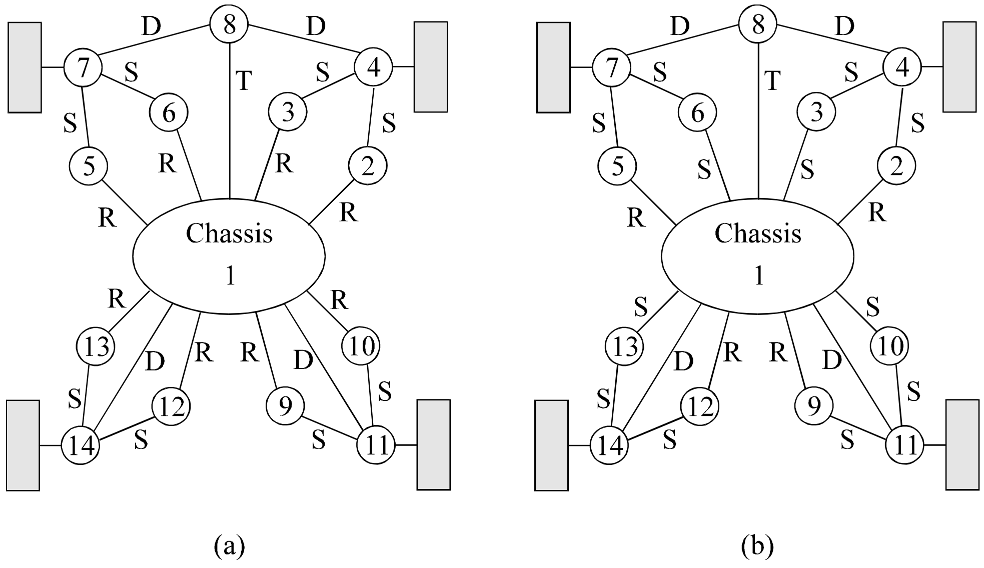


Figure 16. Topology Graph for HMMWV14

The topology of the new model is presented in Figure 16(b). The stiff TSDA's behave like bushing elements, and induce stiffness into the model. The dominant eigenvalue for this example has a small imaginary value, and the real part is of the order $-2.6 \cdot 10^5$.

The vehicle is driven straight at 10mph and hits a bump. The bump's shape is a half cylinder, of diameter 0.1 m. The number of degrees of freedom is 19, but the steering rack is locked, to make sure the vehicle drives straight. This reduces the number of degrees of freedom to 18.

The length of the simulation is from 1, to 4 seconds. Figure 17 shows the time variation of chassis height. The front wheels hit the bump at $T \approx 0.5$ s, and the rear wheels hit the bump at $T \approx 1.2$ s. The length of the simulation in this plot is 5 seconds. Toward the end of the simulation (approximately after 4 seconds), due to overdamping, the chassis height stabilizes at approximately $z_1 = 0.71$ m.

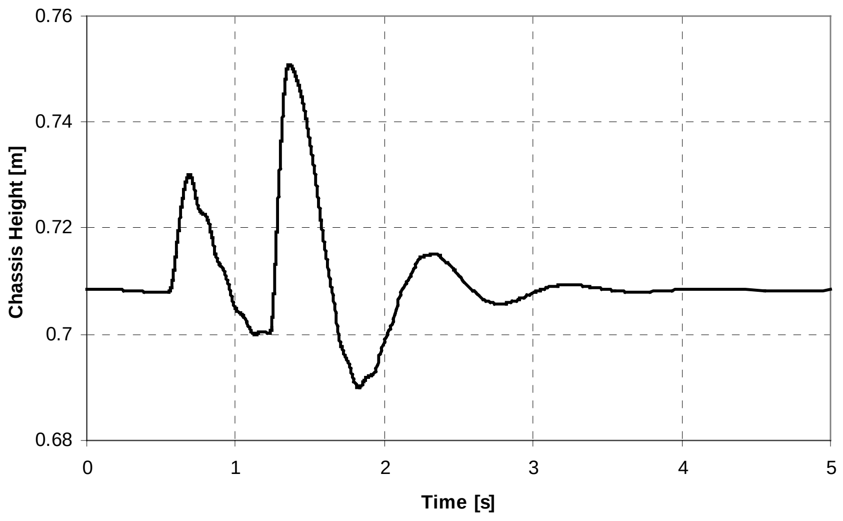


Figure 17. Chassis Height HMMWV14

The test problem is run with an explicit integrator based on the code DEABM of Shampine and Watts, and with an implicit code based on the State-Space Reduction Method of Section 3.2, used in conjunction with a trapezoidal formula. Among the implicit algorithms compared in next Section, the implicit integrator used here displays average performance.

A set of 4 tolerances is imposed, starting from a very loose value of 10^{-2} and ending with a rather conservative tolerance of 10^{-5}. Tolerances in the range 10^{-3} to 10^{-4} usually suffice in engineering applications.

Computer times required by the explicit integrator are listed in Table 24 in CPU seconds. The code of Shampine and Watts is a multi-step code that is adapted to allow efficient acceleration computation, based on the method proposed by Serban, Negrut, Haug, and Potra (1997).

Table 24. HMMWV14 Explicit Integration Simulation CPU Times

TOL	10^{-2}	10^{-3}	10^{-4}	10^{-5}
1 s	3618	3641	3667	3663
2 s	7276	7348	7287	7276
3 s	10865	11122	10949	10965
4 s	14480	14771	14630	14592

Results in Table 24 confirm observations made in Section 4.2.1 concerning the use of explicit integration formulas for stiff IVP. For the stiff test problem considered, the performance limiting factor is stability of the explicit code. For any tolerance in the range 10^{-2} through 10^{-5}, for a given simulation length, CPU times are almost identical. The average step-size is between 10^{-5} and 10^{-6} and it is not affected by accuracy requirements. The code is compelled to select very small step-sizes to assure stability of the integration process, and this is the criteria for step-size selection for a broad spectrum of tolerances. Only when extremely sever accuracy constraints are imposed on integration, is step-size limited by accuracy.

Results in Table 25 show CPU times for the implicit integrator, in seconds. A speed-up of almost two orders of magnitude are obtained with this code.

Table 25. HMMWV14 Implicit Integration Simulation Results

TOL	10^{-2}	10^{-3}	10^{-4}	10^{-5}
1 s	42	54	122	312
2 s	79	140	330	849
3 s	92	168	400	1053
4 s	98	179	424	1095

The plot in Figure 18 shows CPU times for up to 4 seconds of simulation time. As mentioned earlier, the vehicle clears the bump in less than 2 seconds. Since the system is overdamped, the bounce vanishes, and the vehicle runs straight on a smooth road. It is desired that the integrator senses that no event is taking place, and adjusts step-size to larger values, enhancing integration efficiency. As shown in Figure 18, the explicit code is not able to increase the step-size, and CPU time increases linearly with simulation time. This is a limitation of the numerical method, since physically, the evolution of the mechanical system does not pose the same transient challenges after the bump is negotiated.

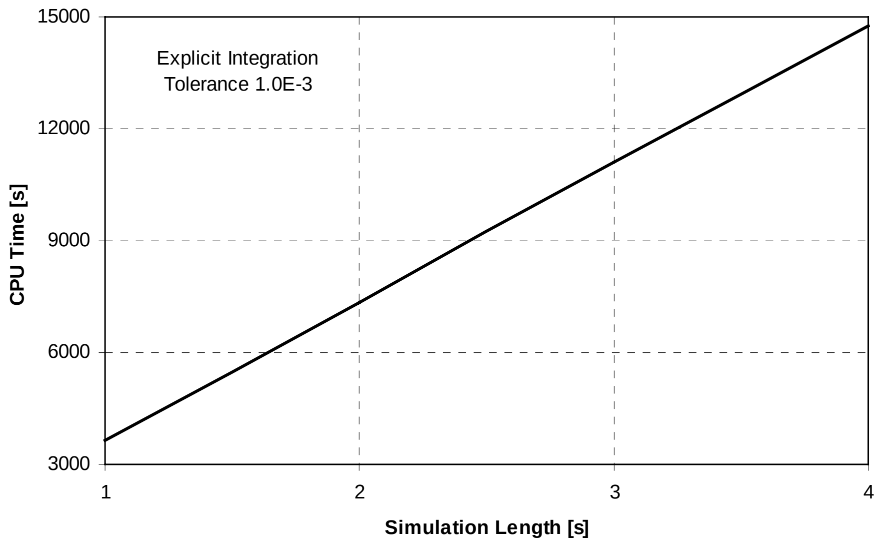


Figure 18. Explicit Integration Results for Tolerance 10^{-3}

For the same tolerance of 10^{-3}, Figure 19 shows timing results for the implicit integrator. The step-size is adjusted according to the mechanical system response. As the vehicle rides over the bump, simulation is CPU intensive, but as motion smoothes larger step-sizes are taken. This conclusion is better reflected in the integration step-size history results shown in next Section.

Figure 20 shows, on a logarithmic scale, timing results for 4 seconds of simulation. Absolute and relative tolerances are assigned values 10^{-2}, 10^{-3}, 10^{-4}, and 10^{-5}, at both position and velocity levels. The explicit integrator is insensitive to changes in integrator tolerance, while the implicit code displays a linear CPU time increase.

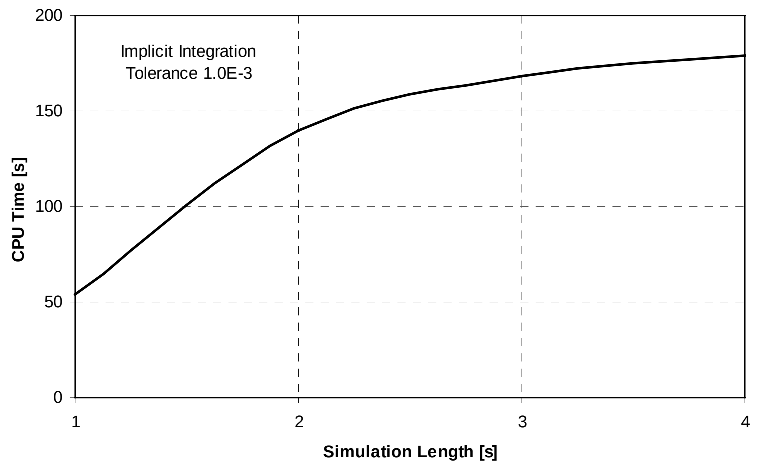


Figure 19. Implicit Integration Results for Tolerance 10^{-3}

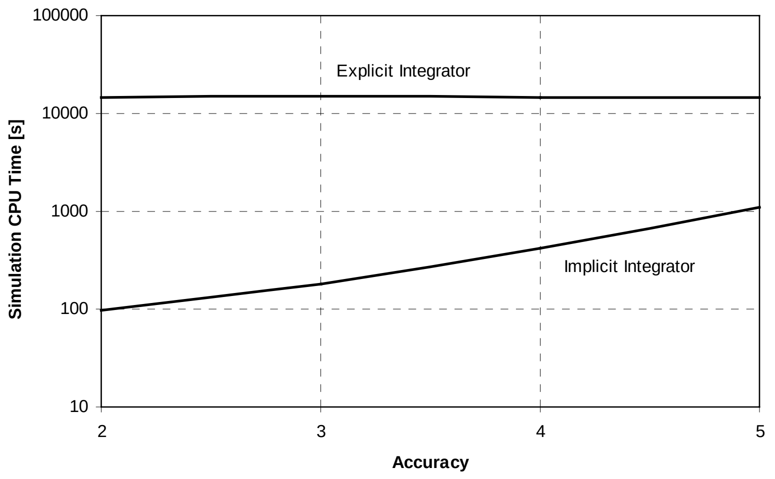


Figure 20. Timing Results for Different Tolerances

5.3 Method Comparison

This Section contains a comparison of algorithms developed for implicit integration of DAE of Multibody Dynamics. The algorithms compared are presented in Chapter 4, and they are as follows:

(1). *SspTrap*, the algorithm of Section 4.1, compared in the previous Section with the explicit code. It is based on the trapezoidal formula used in conjunction with the State-Space Reduction Method

(2). *InflSDIRK*, the algorithm of Section 4.2, based on the SDIRK4/16 formula used in conjunction with the Descriptor Form Method

(3). *InflTrap*, the algorithm of Section 4.3, based on the trapezoidal formula and the Descriptor Form Method

(4). *ForSDIRK*, the algorithm of Section 4.5, validated in Section 5.1, and based on the SDIRK4/15 formula used in conjunction with the First Order Reduction Method

(5). *ForRosen*, the algorithm of Section 4.4, validated in Section 5.1, and based on the Rosenbrock-Nystrom formula used in conjunction with the First Order Reduction Method

CPU results reported in seconds are obtained on an SGI Onyx machine with R10000 processors. The simulation is identical to that in the previous Section; i. e., the HMMWV14 model is driven at 10mph over a bump of cylindrical shape. Four tolerances are imposed, and simulations are from 1 to 4 seconds long. The same tolerance is imposed at both position and velocity levels.

The following four tables present timing results for *InflTrap*, *InflSDIRK*, *ForSDIRK*, and *ForRosen*. For *SspTrap*, timing results are provided in Table 25.

Table 26. Timing Results for *InflSDIRK*

TOL	10^{-2}	10^{-3}	10^{-4}	10^{-5}
1 s	33	52	90	170
2 s	69	124	218	433
3 s	81	150	248	493
4 s	84	155	256	500

Table 27. Timing Results for *InflTrap*

TOL	10^{-2}	10^{-3}	10^{-4}	10^{-5}
1 s	42	61	158	463
2 s	79	155	420	1198
3 s	92	189	524	1521
4 s	100	206	552	1568

Table 28. Timing Results for *ForSDIRK*

TOL	10^{-2}	10^{-3}	10^{-4}	10^{-5}
1 s	10.1	21	33	57
2 s	25.3	49	78	139
3 s	29.5	58	92	166
4 s	30	61	94	184

Table 29. Timing Results for *ForRosen*

TOL	10^{-2}	10^{-3}	10^{-4}	10^{-5}
1 s	5.6	13.2	40.7	172
2 s	12.6	32.6	95	405
3 s	13	36.3	105	422
4 s	13.3	37	106	428

Based on the results presented in these tables, in terms of methods used for numerical solution of DAE of Multibody Dynamics, the First Order Reduction Method is the most efficient, while the Descriptor Form Method is the slowest. The conclusion is that linear algebra is a key factor in deciding performance of a method. The First Order and State-Space Reduction Methods result in algebraic problems of lower dimension, compared to the Descriptor Form Method. In the latter method, iterations are carried out to simultaneously solve for Lagrange multipliers and generalized accelerations. At each iteration, this requires solution of a linear system of dimension $m+n$, where m is the number of position constraint equations, and n is the number of generalized coordinates used to model the mechanical system. For the HMMWV14 model, $n=98$, $m=80$, and the dimension of the discretized non-linear system of equations is 178×178. Although sparse linear routines of the Harwell library are used to solve this system, the penalty is still significant.

The fastest algorithm for mild tolerances is *ForRosen*, which is Rosenbrock-Nystrom formula embedded in the First Order Reduction Method. This algorithm is more than 250 times faster than the explicit code DEABM. The algorithm requires three function evaluations per successful time step. Practically, this number is reduced to two, since one acceleration evaluation is obtained as a by-product of integration Jacobian computation. The algorithm ceases to be efficient for stringent accuracy requirements. However, this is a problem that can be easily fixed by adjusting the step-size controller, according to Eq. (5.6).

For tight tolerances, *ForSDIRK* becomes the most efficient algorithm. It uses the step-size controller and convergence analysis tools inherited from a code of Hairer and Wanner. The step-size controller has a major impact on results of the simulation. Although *ForRosen* and *ForSDIRK* both use fourth order formulas for integration and third order formulas for step-size control, for a 1 second long simulation of the HMMWV14 with tolerances set to 10^{-5}, *ForRosen* requires 2211 successful time steps to carry out simulation, while *ForSDIRK* requires 308.

Figure 21 contains a comparison of the five algorithms. For one second of simulation, tolerance is successively set to 10^{-2}, 10^{-3}, 10^{-4}, and 10^{-5}. The same relative and absolute tolerances are used for positions and velocities. The results are plotted on a logarithmic scale.

For tolerances almost up to 10^{-4}, *ForRosen* is the best algorithm. For higher tolerances, *ForSDIRK* is the most efficient algorithm. The slopes of the two SDIRK based algorithms are the same, and smaller than the slope displayed by the trapezoidal based algorithms. This is explained by the lower order of the trapezoidal formula. The conclusion is that an algorithm based on a lower order formula is not recommended for high accuracy. The higher slopes for lower order formulas indicate a more rapid increase in computational effort as the tolerance becomes tighter.

Results in Figure 21 underline the importance of a good integration formula, embedded in the framework of a DAE method. The Descriptor Form Method is characterized by more intense linear algebra, when compared with the State-Space

Reduction Method. This is illustrated by the fact that the curve for *SspTrap* lies below the curve for *InflTrap*. However, when embedding the SDIRK4/16 formula in the Descriptor Form Method, the algorithm *InflSDIRK* performs better than *SspTrap*.

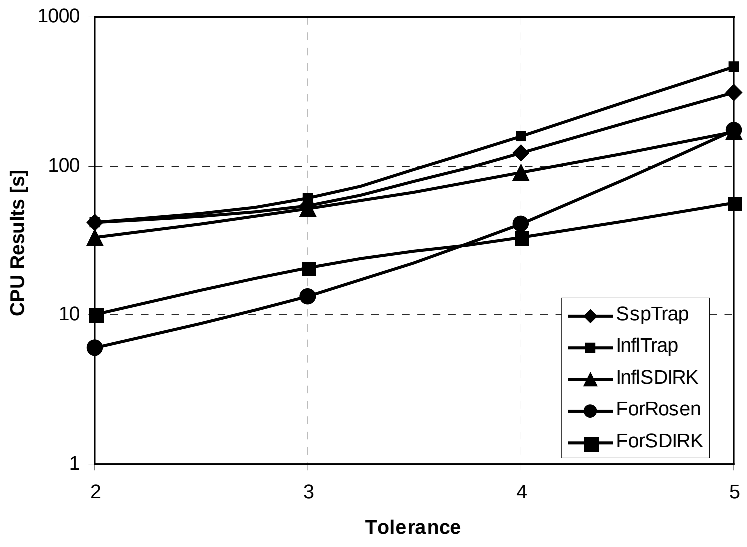


Figure 21. Algorithm Comparison for Different Tolerances

Results in Figure 22 are obtained by setting the integration tolerance to 10^{-3}, and running several simulations between 1 and 4 seconds long. At this tolerance, *ForRosen* is the most efficient algorithm, while the poorest is *InflTrap*.

A good step-size controller should sense that during the first 2 seconds of simulation; i. e., when the vehicle negotiates the bump, the mechanical system experiences extreme behavior. In order to maintain the imposed accuracy, step-size is quickly adjusted and smaller values are taken during this period. After the motion stabilizes, no significant external excitations perturb the evolution of the system, and it is expected that the step-size is increased to take advantage of the relatively smooth evolution of the system. This is exactly what the results in Figure 22 suggest. The curves are rather steep for short simulations, but become almost horizontal for longer simulations. This indicates that after simulation passes the 2 second barrier, an increase in simulation length does not significantly increase the total CPU time. The same conclusion is supported by results in Figure 23, where for *ForSDIRK* and *SspTrap*, simulation step-size is plotted for a 4 second long simulation. The tolerances (absolute and relative) are set to 10^{-3}.

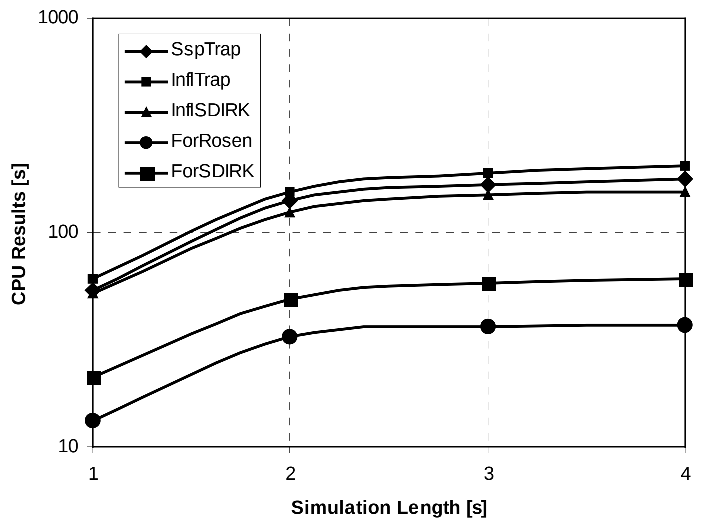


Figure 22. Algorithm Comparison for Different Simulation Lengths

The algorithm *ForSDIRK* is based on a higher order integration formula. Therefore, when compared to *SspTrap*, it is capable of taking larger step-size. The consequence is that the first algorithm takes fewer time steps, therefore fewer integration Jacobian computations and factorizations. One integration Jacobian evaluation accounts for 55% of the CPU time of a successful time step, so keeping the number of integration Jacobian evaluations low is important. For implicit integration of the DAE of Multibody Dynamics via the state-space formulation, one lesson that must be learned is that it is recommended to consider more sophisticated integration formulas. This will pay off in fewer calls to expensive integration Jacobian evaluations and factorizations, as well as function evaluations (acceleration computations).

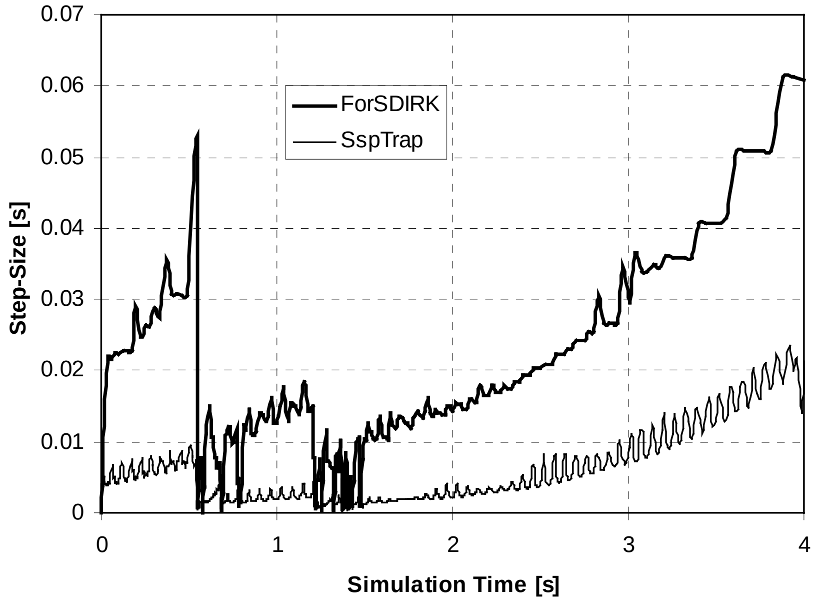


Figure 23. Step-Size History for *ForSDIRK* and *SspTrap*

The results in Figure 23 demonstrate that the step-size is rapidly cut to small values when the vehicle experiences extreme motion. This is the case when the wheels hit the bump ($T \approx 0.55$ s, $T \approx 1.2$ s). After that, the step-size increases, and the impact of larger step-sizes reflects in the plot of Figure 22.

Results presented in Figure 24 show the evolution of step-size over a longer simulation. The tolerance (absolute and relative) is set to 10^{-3}, and the length of the simulation is 30 seconds. Step-sizes for *InflSDIRK* and *ForRosen* very soon reach the limit value of 1.0 s, imposed because of safety considerations. The results in this plot clearly demonstrate the potential of the implicit algorithms developed, compared to an explicit formula based algorithm. In particular, for the explicit code DEABM of Section 5.1, since CPU average time for one second of simulation is slightly more than 1 hour, a simulation such as the one in Figure 24 would require more than one day of CPU time. For *ForRosen*, it takes 38 seconds to complete this run

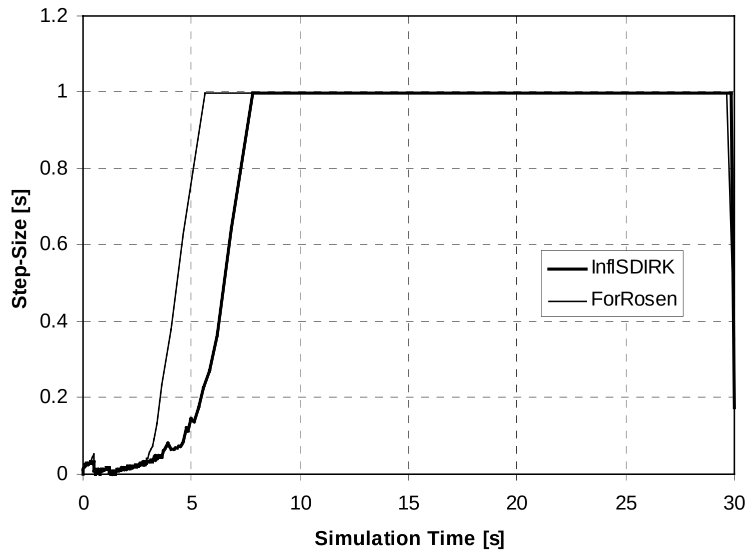


Figure 24. Step-Size History for *ForRosen* and *InflSDIRK*

The very good performance of *ForRosen* is due to its good order, good stability properties, and low number of function evaluations. The order 4 was mentioned earlier to allow for larger step-sizes, and thus fewer integration Jacobian evaluations. The good stability properties of this algorithm enable it to deal with very stiff mechanical systems. Finally, the number of function evaluations is 3, but it was mentioned that one acceleration evaluation comes at no price. Therefore, there are only two requests for acceleration computation for each successful time step.

Figure 25 displays the number of iterations necessary for *SspTrap* to retrieve, during each time step, the solution of the discretized system of non-linear equations. The length of simulation is 10 seconds with tolerances set to 10^{-3}. Compared to *ForRosen*, the number of iterations, and therefore the number of function evaluations required by *SspTrap*, especially during the critical part of simulation; i. e., the first 2 seconds, is large. One function evaluation for the State-Space Method is cheaper than one function evaluation for *ForRosen*, but not cheap enough to compensate for only two function evaluations required by the latter algorithm.

The sole reason that Rosenbrock methods are not used extensively is because it requires an exact integration Jacobian. This quantity is not always possible to compute, and automatic differentiation or numerical means must be considered to provide derivatives appearing in the integration Jacobian. Most frequently, obtaining force derivatives is the difficult part, since certain force elements are either too complex, or as in the case of multidisciplinary applications, are provided by other sub-models, with little or no adjacent information. The algorithm *ForSDIRK* possesses the numerical means to generate the entire integration Jacobian numerically. Table 30 lists CPU timing results in seconds required for simulations when the integration Jacobian is first computed analytically and then numerically.

Employing numerical means for integration Jacobian computation results in 4 to 5 times larger simulation CPU times. Still, it is much more advantageous to use this approach, rather than to resort to explicit integration for dynamic analysis of stiff mechanical systems.

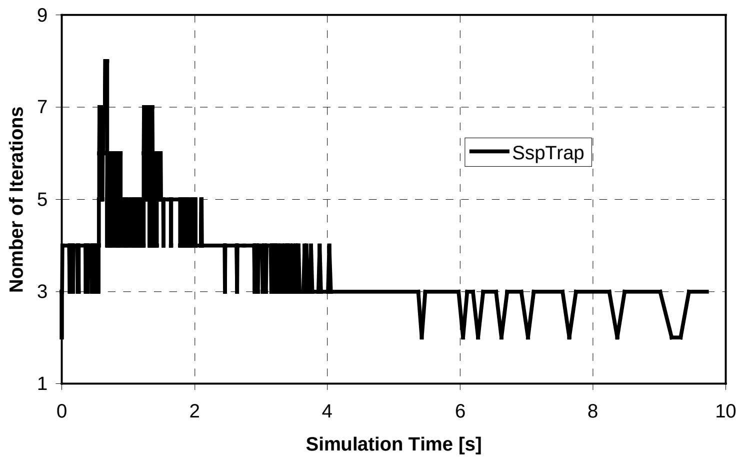


Figure 25. Number of Iterations for *SspTrap*

Table 30. *ForSDIRK* Analytical/Numerical Computation of Integration Jacobian

	1 second	2 seconds	3 seconds	4 seconds
Analytic	21	49	58	61
Numeric	104	229	269	278

CHAPTER 6
CONCLUSIONS AND RECOMMENDATIONS

Three new methods have been developed for implicit numerical integration of the DAE of Multibody Dynamics. Based on these methods, five algorithms were implemented. When used for dynamic analysis of stiff mechanical systems, these algorithms are two orders of magnitude faster than previously available integrators.

Two efficient methods for topology based linear algebra have been introduced. For Cartesian representation of mechanical systems, the proposed algorithms enable computation of accelerations and Lagrange multipliers 3 to 4 times faster than previous implementations. When using a joint formulation to represent mechanical systems, the computational time for computing accelerations is halved. The algorithms for fast acceleration computation can be used for both explicit and First Order Reduction-based implicit integration of the DAE of Multibody Dynamics.

Several issues remain to be investigated and/or numerically implemented, as follows:

(a). Implement methods for parallel computation of the integration Jacobian.

(b). Investigate and implement the tangent-plane parametrization-based DAE-to-ODE reduction method.

(c). Embed topology based linear algebra routines in numerical implementations of the First Order Reduction Method. Currently, advantage is not taken of the subroutines developed for fast acceleration computation.

(d). Improve stopping criteria for State-Space Reduction and Descriptor Form Methods.

(e). Adjust the step-size controller of the algorithm *ForRosen*, based on the Rosenbrock-Nystrom formula used in conjunction with the First Order Reduction Method, to eliminate its conservative estimate.

(f). Apply methods developed for numerical solution of systems that include flexible bodies and intermittent motion

Appendix A and B provide details regarding the method of parallel computation of integration Jacobian, and implicit integration of DAE of Multibody Dynamics via tangent-plane parametrization-based state-space reduction.

APPENDIX A
PARALLEL COMPUTATION OF INTEGRATION JACOBIAN

In order to simplify the presentation, in this Appendix it is assumed that the vector $\mathbf{q} \in \Re^n$ of generalized coordinates has been reordered such that the first m entries contain dependent coordinates, while the last $ndof$ entries contain independent coordinates. The focus is on computing quantities $\mathbf{J}_1 = \ddot{\mathbf{v}}_{\mathbf{v}}$ and $\mathbf{J}_2 = \ddot{\mathbf{v}}_{\dot{\mathbf{v}}}$, that were shown in Chapter 3 to be needed for implicit integration of the SSODE of Multibody Dynamics.

The matrices $\mathbf{J}_1$ and $\mathbf{J}_2$ are computed one column at a time by first computing the quantities $\ddot{\mathbf{q}}_{q_{m+1}}$ through $\ddot{\mathbf{q}}_{q_n}$, and then $\ddot{\mathbf{q}}_{\dot{q}_{m+1}}$ through $\ddot{\mathbf{q}}_{\dot{q}_n}$. The last $ndof$ entries of these vectors are the columns of the matrices $\mathbf{J}_1$, and $\mathbf{J}_2$, respectively.

For $k \in \{1,\ldots,ndof\}$, let $i = m+k$. Taking the derivative of the equations of motion with respect to independent coordinate q_i, yields

$$\left(\mathbf{M}\ddot{\mathbf{q}}\right)_{\mathbf{q}}\mathbf{q}_{q_i} + \mathbf{M}\ddot{\mathbf{q}}_{q_i} + \left(\Phi_{\mathbf{q}}^{\mathrm{T}}\lambda\right)_{\mathbf{q}}\mathbf{q}_{q_i} + \Phi_{\mathbf{q}}^{\mathrm{T}}\lambda_{q_i} = \mathbf{Q}_{\mathbf{q}}^{\mathrm{A}}\mathbf{q}_{q_i} + \mathbf{Q}_{\dot{\mathbf{q}}}^{\mathrm{A}}\dot{\mathbf{q}}_{q_i} \qquad \text{(A.1)}$$

The derivative of the vector of generalized coordinates with respect to independent coordinate q_i is obtained by differentiating the position kinematic constraint equation of Eq. (3.7), to obtain

$$\Phi_{\mathbf{u}}\mathbf{u}_{q_i} + \Phi_{\mathbf{v}} \cdot \mathbf{1}_k = \mathbf{0}$$

where, $\mathbf{u} = [q_1, q_2, \ldots, q_m]^{\mathrm{T}}$, and the vector $\mathbf{1}_k \in \Re^{ndof}$ has all its entries zero, except the k^{th} entry which is 1. With $\mathbf{H} = -\Phi_{\mathbf{u}}^{-1}\Phi_{\mathbf{v}}$, $\mathbf{H} \equiv [\mathbf{h}_1, \mathbf{h}_2, \ldots, \mathbf{h}_{ndof}] \in \Re^{m \times ndof}$, $\mathbf{u}_{q_i} = \mathbf{h}_k$ and

$$\mathbf{q}_{q_i} = \begin{bmatrix} \mathbf{h}_k \\ \mathbf{1}_k \end{bmatrix} \qquad \text{(A.2)}$$

The derivative $\dot{\mathbf{q}}_{q_i}$ is obtained by differentiating the velocity kinematic constraint equation of Eq. (3.8) with respect to q_i, to obtain

$$\left(\Phi_{\mathbf{q}}\dot{\mathbf{q}}\right)_{\mathbf{q}}\mathbf{q}_{q_i} + \Phi_{\mathbf{u}}\dot{\mathbf{u}}_{q_i} = \mathbf{0}$$

Therefore

$$\dot{\mathbf{u}}_{q_i} = -\Phi_{\mathbf{u}}^{-1}\left(\Phi_{\mathbf{q}}\dot{\mathbf{q}}\right)_{\mathbf{q}}\mathbf{q}_{q_i}$$

Finally,

$$\dot{\mathbf{q}}_{q_i} = \begin{bmatrix} \dot{\mathbf{u}}_{q_i} \\ 0 \\ \vdots \\ 0 \end{bmatrix} \tag{A.3}$$

Equation (A.1) is rewritten in the form

$$\mathbf{M}\ddot{\mathbf{q}}_{q_i} + \Phi_{\mathbf{q}}^{\mathrm{T}}\lambda_{q_i} = \mathbf{Q}_i^{\mathrm{P}} \tag{A.4}$$

where

$$\mathbf{Q}_i^{\mathrm{P}} = \mathbf{Q}_{\dot{\mathbf{q}}}^{\mathrm{A}}\dot{\mathbf{q}}_{q_i} - [\left(\mathbf{M}\ddot{\mathbf{q}}\right)_{\mathbf{q}} + \left(\Phi_{\mathbf{q}}^{\mathrm{T}}\lambda\right)_{\mathbf{q}} - \mathbf{Q}_{\mathbf{q}}^{\mathrm{A}}]\mathbf{q}_{q_i} \tag{A.5}$$

Differentiating the acceleration kinematic constraint equation of Eq. (3.9) with respect to q_i and rearranging terms yields

$$\Phi_{\mathbf{q}}\ddot{\mathbf{q}}_{q_i} = \tau_i^{\mathrm{P}} \tag{A.6}$$

where

$$\tau_i^{\mathrm{P}} = \tau_{\dot{\mathbf{q}}}\dot{\mathbf{q}}_{q_i} - [\left(\Phi_{\mathbf{q}}\ddot{\mathbf{q}}\right)_{\mathbf{q}} - \tau_{\mathbf{q}}]\mathbf{q}_{q_i} \tag{A.7}$$

Appending Eq. (A.6) to (A.4), a linear system of dimension $m+n$ is obtained that provides derivatives $\ddot{\mathbf{q}}_{q_i}$ and λ_{q_i},

$$\begin{bmatrix} \mathbf{M} & \Phi_{\mathbf{q}}^{\mathrm{T}} \\ \Phi_{\mathbf{q}} & \mathbf{0} \end{bmatrix}\begin{bmatrix} \ddot{\mathbf{q}}_{q_i} \\ \lambda_{q_i} \end{bmatrix} = \begin{bmatrix} \mathbf{Q}_i^{\mathrm{P}} \\ \tau_{\mathrm{i}}^{\mathrm{P}} \end{bmatrix} \tag{A.8}$$

The linear system of Eq. (A.8) is solved $ndof$ times for $k \in \{1,\ldots,ndof\}$, with $i = m + k$. The coefficient matrix of Eq. (A.8) is identical to the augmented matrix of Chapter 3, for which an efficient factorization sequence was presented in Section 3.4.2. After the factorization is available, it is used $ndof$ times to compute the columns of the matrix $\mathbf{J}_1$ as the last $ndof$ components of solution $\ddot{\mathbf{q}}_{q_i}$ of Eq. (A.8).

Computation of $\mathbf{J}_2$ follows the same path. For $k \in \{1,\ldots,ndof\}$, with $i = m + k$, equations of motion and acceleration kinematic constraint equation are successively differentiated with respect to $\dot{q}_i$. This yields

$$\begin{aligned} &\mathbf{M}\ddot{\mathbf{q}}_{\dot{q}_i} + \Phi_{\mathbf{q}}^{\mathrm{T}}\lambda_{\dot{q}_i} = \mathbf{Q}_{\dot{\mathbf{q}}}^{\mathrm{A}}\dot{\mathbf{q}}_{\dot{q}_i} \\ &\Phi_{\mathbf{q}}\ddot{\mathbf{q}} = \tau_{\dot{\mathbf{q}}}\dot{\mathbf{q}}_{\dot{q}_i} \end{aligned} \tag{A.9}$$

To compute $\dot{\mathbf{q}}_{\dot{q}_i}$, the velocity kinematic constraint equation is differentiated with respect to $\dot{q}_i$, yielding

$$\Phi_{\mathbf{u}}\dot{\mathbf{u}}_{\dot{q}_i} + \Phi_{\mathbf{v}}\mathbf{1}_k = \mathbf{0}$$

Therefore $\dot{\mathbf{u}}_{\dot{\mathbf{q}}_i} = \mathbf{h}_k$; i. e., $\dot{\mathbf{u}}_{\dot{\mathbf{q}}_i}$ is obtained as the k^{th} column of matrix $\mathbf{H}$. Finally,

$$\dot{\mathbf{q}}_{\dot{q}_i} = \begin{bmatrix} \mathbf{h}_k \\ \mathbf{1}_k \end{bmatrix} \tag{A.10}$$

Introducing the notation

$$\mathbf{Q}_i^{\mathrm{V}} = \mathbf{Q}_{\dot{\mathbf{q}}}^{\mathrm{A}}\,\dot{\mathbf{q}}_{\dot{q}_i} \quad , \qquad \tau_i^{\mathrm{V}} = \tau_{\dot{\mathbf{q}}}\dot{\mathbf{q}}_{\dot{q}_i} \tag{A.11}$$

$\ddot{\mathbf{q}}_{\dot{q}_i}$ is obtained by solving

$$\begin{bmatrix} \mathbf{M} & \Phi_{\mathbf{q}}^{\mathrm{T}} \\ \Phi_{\mathbf{q}} & \mathbf{0} \end{bmatrix} \begin{bmatrix} \ddot{\mathbf{q}}_{\dot{q}_i} \\ \lambda_{\dot{q}_i} \end{bmatrix} = \begin{bmatrix} \mathbf{Q}_i^{\mathrm{V}} \\ \tau_i^{\mathrm{V}} \end{bmatrix} \tag{A.12}$$

The last $ndof$ components of $\ddot{\mathbf{q}}_{\dot{q}_i}$ form the k^{th} column of $\mathbf{J}_2$. The coefficient matrix in Eq. (A.12) is identical to the one in Eq. (A.8). This matrix is factored once and then used to obtain both derivatives $\ddot{\mathbf{v}}_{\mathbf{v}}$ and $\ddot{\mathbf{v}}_{\dot{\mathbf{v}}}$. The pseudo-code for the proposed algorithm is outlined in Table 31.

Table 31. Pseudo-code for Parallel Computation of Integration Jacobian

```
1.   Evaluate and Factor Augmented Matrix
2.   Evaluate Basic Derivatives
3.   Compute $\mathbf{H}$ matrix
4.   For i from m+1 to n do
5.        Set $k = i - m$
6.        Compute $\mathbf{q}_{q_i}$
7.        Compute $\dot{\mathbf{q}}_{q_i}$
8.        Compute $\mathbf{Q}_i^{\mathrm{P}}$ and $\tau_i^{\mathrm{P}}$
9.        Solve System in Eq. (A.8) for $\ddot{\mathbf{q}}_{q_i}$ (column k of $\mathbf{J}_1$)
10.       Compute $\mathbf{Q}_i^{\mathrm{V}}$ and $\tau_i^{\mathrm{V}}$
11.       Solve System in Eq. (A.12) for $\ddot{\mathbf{q}}_{\dot{q}_i}$ (column k of $\mathbf{J}_2$)
12.  End do
```

Based on results of Section 3.4.2, Step 1 produces information required by the solution sequence that is employed to solve a linear system of the form

$$\begin{bmatrix} \mathbf{M} & \mathbf{\Phi}_{\mathbf{q}}^{\mathrm{T}} \\ \mathbf{\Phi}_{\mathbf{q}} & \mathbf{0} \end{bmatrix} \begin{bmatrix} \mathbf{x} \\ \mathbf{y} \end{bmatrix} = \begin{bmatrix} \mathbf{b}_1 \\ \mathbf{b}_2 \end{bmatrix} \qquad \text{(A.13)}$$

The first three sub-steps to be taken are as in *Algorithm 3* of Section 3.4.2.4, followed by factorization of the reduced matrix $\mathbf{B}$. The last sub-step; i. e., Step 4 , of *Algorithm 3*, retrieves the solution of the system of Eq. (A.8) or (A.12), and is carried out after the right side of the linear systems is available (Steps 9 and 11).

Step 2 of the pseudo-code evaluates the derivatives $(\mathbf{M}\ddot{\mathbf{q}})_{\mathbf{q}}$, $(\Phi_{\mathbf{q}}^{\mathrm{T}}\lambda)_{\mathbf{q}}$, $(\Phi_{\mathbf{q}}\dot{\mathbf{q}})_{\mathbf{q}}$, $(\Phi_{\mathbf{q}}\ddot{\mathbf{q}})_{\mathbf{q}}$, $\mathbf{Q}_{\mathbf{q}}^{\mathrm{A}}$, $\mathbf{Q}_{\dot{\mathbf{q}}}^{\mathrm{A}}$, $\tau_{\mathbf{q}}$, and $\tau_{\dot{\mathbf{q}}}$. During Step 3, the sub-Jacobian $\Phi_{\mathbf{u}}$ is factored, and the matrix $\mathbf{H} = -\Phi_{\mathbf{u}}^{-1}\Phi_{\mathbf{v}}$ is computed.

Step 4 starts the loop that computes the columns of matrices $\mathbf{J}_1$ and $\mathbf{J}_2$. The derivatives $\mathbf{q}_{q_i}$ and $\dot{\mathbf{q}}_{q_i}$ are computed based on Eqs. (A.2) and (A.3). The right sides of the linear systems of Eqs. (A.8) and (A.12) are obtained based on Eqs. (A.5) and (A.7), and Eq. (A.11), respectively. The solutions of these systems provide column k of $\mathbf{J}_1$ and $\mathbf{J}_2$. Each column is obtained as the last $ndof$ components of the vector $\mathbf{x} \in \Re^n$ of Eq. (A.13). The loop ends after all $ndof$ columns of $\mathbf{J}_1$ and $\mathbf{J}_2$ have been computed.

The advantage of using this approach versus the one described in Section 3.4.1.2, is twofold. First, for linear systems such as in Eq. (A.13), an efficient solution sequence is available. It is the same as that used to compute Lagrange multipliers and generalized accelerations. The algorithm for the Cartesian formulation is presented in Section 3.4.2.

The proposed approach has two levels of parallelism. Steps 1 through 3 can be done in parallel, since these activities are not related in any way. Step 3 can be further distributed to parallel processors to simultaneously compute the derivatives $(\mathbf{M}\ddot{\mathbf{q}})_{\mathbf{q}}$, $(\Phi_{\mathbf{q}}^{\mathrm{T}}\lambda)_{\mathbf{q}}$, $(\Phi_{\mathbf{q}}\dot{\mathbf{q}})_{\mathbf{q}}$, $(\Phi_{\mathbf{q}}\ddot{\mathbf{q}})_{\mathbf{q}}$, $\mathbf{Q}_{\mathbf{q}}^{\mathrm{A}}$, $\mathbf{Q}_{\dot{\mathbf{q}}}^{\mathrm{A}}$, $\tau_{\mathbf{q}}$, and $\tau_{\dot{\mathbf{q}}}$. Finally, Steps 5 through 11 are carried out $ndof$ times. If $ndof$ processors are available, this task can be distributed to compute the columns of $\mathbf{J}_1$ and $\mathbf{J}_2$ in parallel.

This proposed strategy has not been numerically implemented, and therefore no results are available to assess to what extend it is superior to the algorithm provided in Section 3.4.1.2. The algorithm is attractive, since usually the number $ndof$ of degrees of freedom of the model is small; and, with the fast solution sequence available for systems such in Eq. (A.13), obtaining each of the $ndof$ columns of $\mathbf{J}_1$ and $\mathbf{J}_2$ is fast. From an efficiency standpoint, the proposed approach closes the gap between implicit and explicit integration, since

(a). For explicit integration, the coefficient matrix of Eq. (A.13) must be factored once, and one solution is computed based on factorization.

(b). For implicit integration based on the proposed algorithm for computation of the integration Jacobian, the same coefficient matrix must be factored once, and $2 \times ndof$ solutions corresponding to $2 \times ndof$ different right sides are computed.

For both (a) and (b) above, the costly computation is factoring the coefficient matrix (or equivalently, in the framework of *Algorithm 3*, Steps 1 through 3). Generally, retrieving the solution of an $l \times l$ linear system when the factorization is available, reduces to a forward/backward sequence, which is an order l^2 operation. On the other hand, the factorization is an order l^3 operation. Thus, on a sequential machine, (b) requires additional $2 \times ndof - 1$ forward/backward elimination sequences. However, what is costly, when comparing to explicit integration, is Step 2 of the pseudo-code.
The final conclusion is that if (b) is to become competitive, compared to (a), multi-processor architecture must be considered. Then, the impact of Step 2 is reduced, and the extra effort related to the additional $2 \times ndof - 1$ forward/backward sequences disappears, provided enough processors are available. Under this scenario, a Rosenbrock formula with a small number of function evaluations, such as the one presented in Chapter 4, could be considered for applications ranging from mildly to extremely stiff problems. This remains to be investigated.

APPENDIX B
TANGENT-PLANE PARAMETRIZATION-BASED IMPLICIT INTEGRATION

In this thesis, numerical solution of the DAE of Multibody Dynamics is based on state-space reduction technique. In the framework of state-space methods, the index 3 DAE of Multibody Dynamics induce a differential equation on the constraint manifold, which is first projected on a subspace of the n-dimensional Euclidean space. This sub-space is parameterized by a set of independent variables, which are called parametrization variables. The resulting state-space ODE (SSODE) is integrated using a classical numerical integration formula. The one-to-one local chart from the manifold to the projection subspace is then used to determine the point on the manifold corresponding to the solution of the SSODE.

This framework was first proposed by Rheinboldt (1984), in an effort to formalize the theory of numerical solution of DAE, using the language of differential manifolds. More applied considerations following this path are due to Wehage and Haug (1982), Liang and Lance (1987), Potra and Rheinboldt (1990, 1991), and Yen (1993). What distinguishes these methods is the choice of manifold parameterization. In Chapter 2 it was mentioned that this thesis uses the generalized coordinate state-space reduction due to Wehage and Haug (1982), in which parameterization variables are a subset of generalized coordinates.

State-space methods for solution of the DAE of multibody dynamics have been subject to critique in two aspects. First, the choice of projection subspace is generally not global. Second, as Alishenas and Olafsson (1994) have pointed out, bad choices of the projection space result in SSODE that are demanding in terms of numerical treatment, mainly at the expense of overall efficiency of the algorithm.

The approach proposed in this Section uses the manifold tangent hyper-plane as the projection sub-space. Parametrization variables are obtained as linear combinations of generalized coordinates. The benefits of this reduction are anticipated to be twofold. The resulting SSODE is expected to be numerically better conditioned and allow for significantly larger integration step-sizes. Second, dependent variable recovery can take advantage of information generated during the process of state-space reduction.

Tangent-space parametrization requires QR decomposition of the constraint Jacobian, an operation that is twice as costly as Gaussian elimination used in the coordinate partitioning technique. One possibility to diminish this difference is to take into account the structure of the constraint Jacobian, as induced by connectivity between bodies of the model. Sparsity and structure can be preserved and efficiently exploited by using Givens rotation (Golub and Van Loan, 1989) for QR factorization of the constraint Jacobian. A future research objective is to analyze to what extent the better conditioning of the resulting SSODE compensates for the somewhat more expensive state-space reduction stage of the tangent-plane method.

As a result of QR factorization of the constraint Jacobian $\mathbf{\Phi_q} \in \Re^{m\times n}$, a unitary matrix $\mathbf{Q} \in \Re^{n\times n}$, and a matrix $\mathbf{R} \in \Re^{n\times m}$ are obtained such that

$$\mathbf{\Phi}_{\mathbf{q}}^{\mathrm{T}} = \mathbf{Q}\cdot\mathbf{R} \tag{B.1}$$

The matrix $\mathbf{Q}$ is partitioned in the form

$$\mathbf{Q} = [\mathbf{Q}_1\, \mathbf{Q}_2] \tag{B.2}$$

with $\mathbf{Q}_1 \in \Re^{n\times m}$ and $\mathbf{Q}_2 \in \Re^{n\times ndof}$. The matrix $\mathbf{R}$ assumes the form

$$\mathbf{R} = \begin{bmatrix} \mathbf{R}_1 \\ \mathbf{0} \end{bmatrix} \tag{B.3}$$

where $\mathbf{R}_1 \in \Re^{m\times m}$ is upper triangular with non-zero diagonal elements, provided the constraint Jacobian matrix has full row rank. The following identities are used in this section:

$$\mathbf{Q}_1^{\mathrm{T}}\mathbf{Q}_2 = \mathbf{0}_{m\times ndof} \quad , \qquad \mathbf{Q}_1^{\mathrm{T}}\mathbf{Q}_1 = \mathbf{I}_{m\times m} \quad , \qquad \mathbf{Q}_2^{\mathrm{T}}\mathbf{Q}_2 = \mathbf{I}_{ndof\times ndof} \tag{B.4}$$

To avoid the confusion that might be caused by using the same letter to denote both the vector of generalized forces and the matrix in the $\mathbf{QR}$ factorization, the equations of motion and kinematic constraint equations at the position, velocity, and acceleration levels are rewritten in the form

$$\mathbf{M}\ddot{\mathbf{q}} + \Phi_{\mathbf{q}}^{\mathrm{T}}\lambda = \mathbf{F}^{\mathrm{A}} \tag{B.5}$$

$$\Phi(\mathbf{q}) = \mathbf{0} \tag{B.6}$$

$$\Phi_{\mathbf{q}}\dot{\mathbf{q}} = \mathbf{0} \tag{B.7}$$

$$\Phi_{\mathbf{q}}\ddot{\mathbf{q}} = \tau \tag{B.8}$$

A new set of generalized coordinates $\mathbf{z}$ is defined as

$$\mathbf{z} = \mathbf{Q}^{\mathrm{T}}\mathbf{q} \tag{B.9}$$

Let $\mathbf{u} \in \Re^{m}$ contain the first m components of $\mathbf{z}$ and $\mathbf{v} \in \Re^{ndof}$ contain the last *ndof* components of $\mathbf{z}$. Since $\mathbf{Q}$ is unitary,

$$\mathbf{q} = \mathbf{Q}\mathbf{z} \tag{B.10}$$

Therefore,

$$\mathbf{q} = [\mathbf{Q}_1\ \mathbf{Q}_2]\begin{bmatrix}\mathbf{u}\\ \mathbf{v}\end{bmatrix} = \mathbf{Q}_1\mathbf{u} + \mathbf{Q}_2\mathbf{v} \tag{B.11}$$

Since the matrices $\mathbf{Q}_1$ and $\mathbf{Q}_2$ are constant,

$$\dot{\mathbf{q}} = \mathbf{Q}_1\dot{\mathbf{u}} + \mathbf{Q}_2\dot{\mathbf{v}} \quad , \qquad \ddot{\mathbf{q}} = \mathbf{Q}_1\ddot{\mathbf{u}} + \mathbf{Q}_2\ddot{\mathbf{v}} \tag{B.12}$$

The next objective is to express the equations of motion and constraint equations in Eqs. (B.5) through (B.8) in terms of the components $\mathbf{u}$ and $\mathbf{v}$ of the new variable $\mathbf{z}$. Multiplying the equations of motion on the left by $\mathbf{Q}^{\mathrm{T}}$ and using Eq. (B.12) yields

$$\mathbf{Q}^{\mathrm{T}}\mathbf{M}\mathbf{Q}\ddot{\mathbf{z}} + \mathbf{Q}^{\mathrm{T}}\Phi_{\mathbf{q}}^{\mathrm{T}}\lambda = \mathbf{Q}^{\mathrm{T}}\mathbf{F}^{\mathrm{A}} \tag{B.13}$$

Formulated in terms of the variables $\mathbf{u}$ and $\mathbf{v}$, Eq. (B.13) assumes the form

$$\mathbf{Q}_1^{\mathrm{T}}\mathbf{M}\mathbf{Q}_1\ddot{\mathbf{u}} + \mathbf{Q}_1^{\mathrm{T}}\mathbf{M}\mathbf{Q}_2\ddot{\mathbf{v}} + \mathbf{Q}_1^{\mathrm{T}}\Phi_{\mathbf{q}}^{\mathrm{T}}\lambda = \mathbf{Q}_1^{\mathrm{T}}\mathbf{F}^{\mathrm{A}} \tag{B.14}$$

$$\mathbf{Q}_2^{\mathrm{T}}\mathbf{M}\mathbf{Q}_1\ddot{\mathbf{u}} + \mathbf{Q}_2^{\mathrm{T}}\mathbf{M}\mathbf{Q}_2\ddot{\mathbf{v}} + \mathbf{Q}_2^{\mathrm{T}}\Phi_{\mathbf{q}}^{\mathrm{T}}\lambda = \mathbf{Q}_2^{\mathrm{T}}\mathbf{F}^{\mathrm{A}} \tag{B.15}$$

Equation (B.15) can be regarded as a set of second order SSODE in $\mathbf{v}$, since all other variables in this equation; i. e., $\mathbf{u}$, $\dot{\mathbf{u}}$, $\ddot{\mathbf{u}}$, and λ can be expressed in terms of $\mathbf{v}$ and its time derivative.

To express dependent variables in terms of $\mathbf{v}$ and $\dot{\mathbf{v}}$, note first that using the position kinematic constraint equation of Eq. (B.6) and Eq. (B.11)

$$\Phi(\mathbf{q}) = \Phi(\mathbf{u},\mathbf{v}) = \mathbf{0} \tag{B.16}$$

Since $\Phi_{\mathbf{u}} = \Phi_{\mathbf{q}} \cdot \mathbf{q}_{\mathbf{u}} = \Phi_{\mathbf{q}}\mathbf{Q}_1 = \mathbf{R}_1$, the matrix $\Phi_{\mathbf{u}}$ is non-singular, and the implicit function theorem (Corwin, Szczarba, 1982) guarantees that Eq. (B.16) can be locally solved in a neighborhood of the consistent configuration $\mathbf{q}$ for $\mathbf{u}$ as a function of $\mathbf{v}$.

Next, velocity kinematic constraint equation of Eq. (B.7), assumes the expression

$$\Phi_{\mathbf{q}}(\mathbf{Q}_1\dot{\mathbf{u}} + \mathbf{Q}_2\dot{\mathbf{v}}) = \mathbf{0}$$

and therefore

$$\dot{\mathbf{u}} = -\left(\Phi_{\mathbf{q}}\mathbf{Q}_1\right)^{-1}\mathbf{Q}_2\dot{\mathbf{v}} \tag{B.17}$$

Finally, using the acceleration kinematic constraint equation, $\ddot{\mathbf{u}}$ is obtained as the solution of the lower triangular linear system

$$\mathbf{R}_1^{\mathrm{T}}\ddot{\mathbf{u}} = \tau \tag{B.18}$$

With $\mathbf{u}$, $\dot{\mathbf{u}}$, and $\ddot{\mathbf{u}}$ expressed as functions of $\mathbf{v}$ and $\dot{\mathbf{v}}$, Eq. (B.14) is used to obtain λ in terms of $\mathbf{v}$ and its first and second time derivatives. The Lagrange multipliers λ is the solution of

$$\mathbf{R}_1\lambda = \mathbf{Q}_1^{\mathrm{T}}\mathbf{F}^{\mathrm{A}} - \left(\mathbf{Q}_1^{\mathrm{T}}\mathbf{M}\mathbf{Q}_1\mathbf{R}_1^{-\mathrm{T}}\tau + \mathbf{Q}_1^{\mathrm{T}}\mathbf{M}\mathbf{Q}_2\ddot{\mathbf{v}}\right) \tag{B.19}$$

The second order ODE in $\mathbf{v}$ is now readily available by substituting dependent variables into Eq. (B.15). Formally, this ODE can be brought to the form

$$\ddot{\mathbf{v}} = \mathbf{f}(t, \mathbf{v}, \dot{\mathbf{v}}) \tag{B.20}$$

and further reduced to a first order ODE, suitable for integration using any standard ODE code. Derivatives $\mathbf{J}_1 \equiv \ddot{\mathbf{v}}_{\mathbf{v}}$ and $\mathbf{J}_2 \equiv \ddot{\mathbf{v}}_{\dot{\mathbf{v}}}$ must be provided. The presentation here focuses on how to provide these quantities, in the framework of tangent-plane parametrization-based state-space reduction of the index 3 DAE of Multibody Dynamics.

Since $\ddot{\mathbf{v}}_{\dot{\mathbf{v}}}$ is obtained most easily, the matrix $\mathbf{J}_2$ is computed first. For this, Eq. (B.15) is differentiated with respect to $\dot{\mathbf{v}}$ to obtain

$$\mathbf{Q}_2^{\mathrm{T}}\mathbf{M}\mathbf{Q}_1\ddot{\mathbf{u}}_{\dot{\mathbf{v}}} + \mathbf{Q}_2^{\mathrm{T}}\mathbf{M}\mathbf{Q}_2\mathbf{J}_2 + \mathbf{Q}_2^{\mathrm{T}}\Phi_{\mathbf{q}}^{\mathrm{T}}\lambda_{\dot{\mathbf{v}}} = \mathbf{Q}_2^{\mathrm{T}}\left(\mathbf{F}_{\dot{\mathbf{u}}}^{\mathrm{A}}\dot{\mathbf{u}}_{\dot{\mathbf{v}}} + \mathbf{F}_{\dot{\mathbf{v}}}^{\mathrm{A}}\right) \tag{B.21}$$

To obtain $\mathbf{J}_2$, the quantities $\dot{\mathbf{u}}_{\dot{\mathbf{v}}}$, $\ddot{\mathbf{u}}_{\dot{\mathbf{v}}}$, and $\lambda_{\dot{\mathbf{v}}}$ must be available. Defining

$$\mathbf{H} = -\left(\Phi_{\mathbf{q}}\mathbf{Q}_1\right)^{-1}\left(\Phi_{\mathbf{q}}\mathbf{Q}_2\right) \tag{B.22}$$

and taking the derivative of the velocity kinematic constraint equation of Eq. (B.7) with respect to $\dot{\mathbf{v}}$ yields

$$\dot{\mathbf{u}}_{\dot{\mathbf{v}}} = \mathbf{H} \tag{B.23}$$

To obtain $\ddot{\mathbf{u}}_{\dot{\mathbf{v}}}$, the acceleration kinematic constraint equation of Eq. (B.8) is differentiated with respect to $\dot{\mathbf{v}}$. By introducing the notation

$$\mathbf{P} = \mathbf{Q}_1\mathbf{H} + \mathbf{Q}_2 \tag{B.24}$$

the derivative $\ddot{\mathbf{u}}_{\dot{\mathbf{v}}}$ assumes the form

$$\ddot{\mathbf{u}}_{\dot{\mathbf{v}}} = \mathbf{H}\mathbf{J}_2 + \left(\Phi_{\mathbf{q}}\mathbf{Q}_1\right)^{-1}\tau_{\dot{\mathbf{q}}}\mathbf{P} \tag{B.25}$$

To compute the derivative $\lambda_{\dot{\mathbf{v}}}$, Eq. (B.14) is differentiated with respect to $\dot{\mathbf{v}}$. This yields

$$\lambda_{\dot{\mathbf{v}}} = \left(\mathbf{Q}_1^{\mathrm{T}}\Phi_{\mathbf{q}}^{\mathrm{T}}\right)^{-1}\left[\mathbf{Q}_1^{\mathrm{T}}\mathbf{F}_{\dot{\mathbf{q}}}^{\mathrm{A}}\mathbf{P} - \left(\mathbf{Q}_1^{\mathrm{T}}\mathbf{M}\mathbf{Q}_1\ddot{\mathbf{u}}_{\dot{\mathbf{v}}} + \mathbf{Q}_1^{\mathrm{T}}\mathbf{M}\mathbf{Q}_2\mathbf{J}_2\right)\right] \tag{B.26}$$

Substituting Eqs. (B.23), (B.25), and (B.26) into Eq. (B.21), $\mathbf{J}_2$ is obtained as the solution of the multiple right side system

$$\left(\mathbf{P}^{\mathrm{T}}\mathbf{M}\mathbf{P}\right)\mathbf{J}_2 = \mathbf{P}^{\mathrm{T}}\left[\mathbf{F}_{\dot{\mathbf{q}}}^{\mathrm{A}} - \mathbf{M}\mathbf{Q}_1\left(\Phi_{\mathbf{q}}\mathbf{Q}_1\right)^{-1}\tau_{\dot{\mathbf{q}}}\right]\mathbf{P} \tag{B.27}$$

The matrix $\mathbf{J}_2$ can be computed if the coefficient matrix in Eq. (B.27) is non-singular. This matrix can be shown to be positive definite, provided the quadratic form $1/2\,\dot{\mathbf{q}}^{\mathrm{T}}\mathbf{M}\dot{\mathbf{q}}$ is always positive for any non-zero $\dot{\mathbf{q}}$ satisfying velocity kinematic constraint equation. This latter statement is not a restrictive assumption, since under these circumstances, the above quadratic form represents the kinetic energy of the mechanical system, which is positive for non-zero $\dot{\mathbf{q}}$. Let $\dot{\mathbf{v}} \in \Re^{ndof}$ be an arbitrary vector. Then

$$\dot{\mathbf{v}}^{\mathrm{T}}\mathbf{P}^{\mathrm{T}}\mathbf{M}\mathbf{P}\dot{\mathbf{v}} = \mathbf{v}^{\mathrm{T}}\left(\mathbf{H}^{\mathrm{T}}\mathbf{Q}_1^{\mathrm{T}} + \mathbf{Q}_2^{\mathrm{T}}\right)\mathbf{M}\left(\mathbf{Q}_1\mathbf{H} + \mathbf{Q}_2\right)\dot{\mathbf{v}}$$

Using Eqs. (B.12) and (B.22)

$$\dot{\mathbf{u}} = \mathbf{H}\dot{\mathbf{v}}$$

Therefore

$$\left(\mathbf{Q}_1\mathbf{H} + \mathbf{Q}_2\right)\dot{\mathbf{v}} = \mathbf{Q}_1\dot{\mathbf{u}} + \mathbf{Q}_2\dot{\mathbf{v}} = \dot{\mathbf{q}}$$

Then,

$$\dot{\mathbf{v}}^{\mathrm{T}}\mathbf{P}^{\mathrm{T}}\mathbf{M}\mathbf{P}\dot{\mathbf{v}} = \dot{\mathbf{q}}^{\mathrm{T}}\mathbf{M}\dot{\mathbf{q}} > 0$$

Consequently, the coefficient matrix in Eq. (B.27) is positive definite, and the computation of $\mathbf{J}_2$ is possible.

The computation of $\mathbf{J}_1 = \ddot{\mathbf{v}}_\mathbf{v}$ follows the same approach used for computation of $\mathbf{J}_2$. Taking the derivative of Eq. (B.15) with respect to $\mathbf{v}$ yields

$$\begin{aligned}&\left(\mathbf{Q}_2^\mathrm{T}\mathbf{M}\mathbf{Q}_1\ddot{\mathbf{u}}\right)_\mathbf{v} + \mathbf{Q}_2^\mathrm{T}\mathbf{M}\mathbf{Q}_1\ddot{\mathbf{u}}_\mathbf{v} + \left(\mathbf{Q}_2^\mathrm{T}\mathbf{M}\mathbf{Q}_2\right)_\mathbf{v} + \mathbf{Q}_2^\mathrm{T}\mathbf{M}\mathbf{Q}_2\mathbf{J}_1\\ &+\mathbf{Q}_2^\mathrm{T}\left(\Phi_\mathbf{q}^\mathrm{T}\lambda\right)_\mathbf{v} + \mathbf{Q}_2^\mathrm{T}\Phi_\mathbf{q}^\mathrm{T}\lambda_\mathbf{v} = \mathbf{Q}_2^\mathrm{T}\left(\mathbf{F}_\mathbf{q}^\mathrm{A}\mathbf{q}_\mathbf{v} + \mathbf{F}_{\dot{\mathbf{q}}}^\mathrm{A}\dot{\mathbf{q}}_\mathbf{v}\right)\end{aligned} \tag{B.28}$$

The quantities that must be computed are $\mathbf{u}_\mathbf{v}$, $\dot{\mathbf{u}}_\mathbf{v}$, $\ddot{\mathbf{u}}_\mathbf{v}$, and $\lambda_\mathbf{v}$. In order to compute $\mathbf{u}_\mathbf{v}$, position kinematic constraint equation of Eq. (B.6) is differentiated with respect to $\mathbf{v}$. With the definition of the matrix $\mathbf{H}$ in Eq. (B.22),

$$\mathbf{u}_\mathbf{v} = \mathbf{H} \tag{B.29}$$

and based on Eqs. (B.11) and (B.24),

$$\mathbf{q}_\mathbf{v} = \mathbf{Q}_1\mathbf{H} + \mathbf{Q}_2 = \mathbf{P} \tag{B.30}$$

The derivative $\dot{\mathbf{u}}_\mathbf{v}$ is obtained by differentiating the velocity kinematic constraint equation of Eq. (B.7) with respect to $\mathbf{v}$. Rearranging the terms and taking into account the definition of matrix $\mathbf{P}$,

$$\dot{\mathbf{u}}_\mathbf{v} = -\left(\Phi_\mathbf{q}\mathbf{Q}_1\right)^{-1}\left(\Phi_\mathbf{q}\dot{\mathbf{q}}\right)_\mathbf{q}\mathbf{P} \tag{B.31}$$

To compute $\ddot{\mathbf{u}}_\mathbf{v}$, the acceleration kinematic constraint equation of Eq. (B.8) is differentiated with respect to $\mathbf{v}$ to obtain

$$\left(\Phi_\mathbf{q}\ddot{\mathbf{q}}\right)_\mathbf{q}\mathbf{q}_\mathbf{v} + \Phi_\mathbf{q}\left(\mathbf{Q}_1\ddot{\mathbf{u}}_\mathbf{v} + \mathbf{Q}_2\mathbf{J}_1\right) = \tau_\mathbf{q}\mathbf{q}_\mathbf{v} + \tau_{\dot{\mathbf{q}}}\mathbf{Q}_1\dot{\mathbf{u}}_\mathbf{v}$$

and, after rearranging terms,

$$\ddot{\mathbf{u}}_\mathbf{v} = \mathbf{H}\mathbf{J}_1 + \left(\Phi_\mathbf{q}\mathbf{Q}_1\right)^{-1}\left[\left(\tau_\mathbf{q} - \Phi_\mathbf{q}\ddot{\mathbf{q}}\right)_\mathbf{q} - \tau_{\dot{\mathbf{q}}}\mathbf{Q}_1\left(\Phi_\mathbf{q}\mathbf{Q}_1\right)^{-1}\left(\Phi_\mathbf{q}\dot{\mathbf{q}}\right)_\mathbf{q}\right]\mathbf{P} \tag{B.32}$$

Finally, the derivative $\lambda_\mathbf{v}$ is obtained by differentiating Eq. (B.14) with respect to $\mathbf{v}$, to obtain

$$\begin{aligned}&\left(\mathbf{Q}_1^{\mathrm{T}}\mathbf{M}\mathbf{Q}_1\ddot{\mathbf{u}}\right)_{\mathbf{v}} + \mathbf{Q}_1^{\mathrm{T}}\mathbf{M}\mathbf{Q}_1\ddot{\mathbf{u}}_{\mathbf{v}} + \left(\mathbf{Q}_1^{\mathrm{T}}\mathbf{M}\mathbf{Q}_2\ddot{\mathbf{v}}\right)_{\mathbf{v}} + \mathbf{Q}_1^{\mathrm{T}}\mathbf{M}\mathbf{Q}_2\mathbf{J}_1\\ &+\mathbf{Q}_1^{\mathrm{T}}\left(\Phi_{\mathbf{q}}^{\mathrm{T}}\lambda\right)_{\mathbf{v}} + \mathbf{Q}_1^{\mathrm{T}}\Phi_{\mathbf{q}}^{\mathrm{T}}\lambda_{\mathbf{v}} = \mathbf{Q}_1^{\mathrm{T}}\left(\mathbf{F}_{\mathbf{q}}^{\mathrm{A}}\mathbf{P} + \mathbf{F}_{\dot{\mathbf{q}}}^{\mathrm{A}}\mathbf{Q}_1\dot{\mathbf{u}}_{\mathbf{v}}\right)\end{aligned}$$

and, after rearranging the terms,

$$\begin{aligned}\lambda_{\mathbf{v}} = -\left(\mathbf{Q}_1^{\mathrm{T}}\Phi_{\mathbf{q}}^{\mathrm{T}}\right)^{-1}\Big[&\mathbf{Q}_1^{\mathrm{T}}\left(\mathbf{M}\ddot{\mathbf{q}} + \Phi_{\mathbf{q}}^{\mathrm{T}}\lambda\right)_{\mathbf{v}} + \mathbf{Q}_1^{\mathrm{T}}\mathbf{M}\mathbf{Q}_1\ddot{\mathbf{u}}_{\mathbf{v}}\\ &+\mathbf{Q}_1^{\mathrm{T}}\mathbf{M}\mathbf{Q}_2\mathbf{J}_1 - \mathbf{Q}_1^{\mathrm{T}}\left(\mathbf{F}_{\mathbf{q}}^{\mathrm{A}}\mathbf{P} + \mathbf{F}_{\dot{\mathbf{q}}}^{\mathrm{A}}\mathbf{Q}_1\dot{\mathbf{u}}_{\mathbf{v}}\right)\Big]\end{aligned} \qquad \text{(B.33)}$$

Substituting the expression of the derivatives in Eqs. (B.29), (B.31), (B.32), and (B.33) into Eq. (B.28), after matrix manipulations, $\mathbf{J}_1$ is obtained as the matrix solution of the linear system

$$\begin{aligned}\left(\mathbf{P}^{\mathrm{T}}\mathbf{M}\mathbf{P}\right)\mathbf{J}_1 = \mathbf{P}^{\mathrm{T}}\Big\{&\mathbf{M}\mathbf{Z}\left[\left(\Phi_{\mathbf{q}}\ddot{\mathbf{q}}\right)_{\mathbf{q}} - \tau_{\mathbf{q}}\right] - \left[\left(\mathbf{M}\ddot{\mathbf{q}}\right)_{\mathbf{q}} + \left(\Phi_{\mathbf{q}}^{\mathrm{T}}\lambda\right)_{\mathbf{q}} - \mathbf{F}_{\mathbf{q}}^{\mathrm{A}}\right]\\ &+\left(\mathbf{M}\mathbf{Z}\tau_{\dot{\mathbf{q}}} - \mathbf{F}_{\dot{\mathbf{q}}}^{\mathrm{A}}\right)\mathbf{Z}\left(\Phi_{\mathbf{q}}\dot{\mathbf{q}}\right)_{\mathbf{q}}\Big\}\mathbf{P}\end{aligned} \qquad \text{(B.34)}$$

where

$$\mathbf{Z} = \mathbf{Q}_1\left(\Phi_{\mathbf{q}}\mathbf{Q}_1\right)^{-1} \qquad \text{(B.35)}$$

In a quasi-Newton implementation used in conjunction with a one-step method, the integration Jacobian is evaluated once at the beginning of a macro-step. It is then used for all stages. In this configuration, matrices $\mathbf{H}$, $\mathbf{P}$, and $\mathbf{Z}$ defined in Eqs. (B.22), (B.24), and (B.35), respectively, assume simpler expressions because of the identities provided in Eq. (B.4). Thus,

$$\mathbf{H} = \mathbf{0} \qquad \text{(B.36)}$$

$$\mathbf{P} = \mathbf{Q}_2 \qquad \text{(B.37)}$$

$$\mathbf{Z} = \mathbf{Q}_1\mathbf{R}_1^{-\mathrm{T}} \qquad \text{(B.38)}$$

When compared to the coordinate partitioning alternative, the tangent-plane parametrization-based state-space reduction results in a different SSODE, provided in implicit form in Eq. (B.15). Equations (B.27) and (B.34) provide derivative information that is shown in Section 3.4 to be sufficient for the First Order Reduction Method. Once the second order ODE, and derivative information are available, the First Order Reduction Method is directly applicable.

Although the theoretical framework for the tangent-plane parameterization algorithm was outlined several years ago (Mani, Haug, and Atkinson, 1985; Potra and Rheinboldt, 1991), there have been no systematic numerical experiments with large scale mechanical systems to confirm better performance in terms of integration step size and robustness. Implementing the tangent-plane parametrization reduction algorithm and carrying out a systematic comparison with the alternative provided by the coordinate partitioning algorithm remains one direction of future work.

REFERENCES

Alexander, R., “Diagonally Implicit Runge-Kutta Methods for Stiff ODE’s,” *SIAM J. Numer. Anal.*, vol. 14, pp. 1006-1021, 1977

Alishenas, T., “Zur Numerische Behandlung, Stabilisierung durch Projection und Modellierung mechanischer Systeme mit Nebenbedingungen und Invarianten,” Doctoral Thesis, The University of Stockholm, TRITA-NA-9202, 1992

Alishenas, T., Olafsson, O., “Modeling and Velocity Stabilization of Constrained Mechanical Systems,” *BIT,* Vol. 34, pp. 455-483, 1994

Andrezerjewski, T., Schwerin, R., "Exploiting Sparsity in the Integration of Multibody Systems in Descriptor Form," Preprint 95-24, Universität Heidelberg, 1995

Ascher, U., Chin, H., Petzold, L. R., Reich, S., ”Stabilization of Constrained Mechanical Systems with DAEs and Invariant manifolds,” *Mech.Struct.&Mach.*, vol. 23(2), pp. 125-157, 1995

Ascher, U., Petzold, L. R., “Stability of Computational Methods for Constrained Dynamics Systems,” *SIAM J. Sci., Stat. Comput*, vol. 14, pp. 95-120, 1993

Ascher, U., Petzold, L. R., Chin, H., “Stabilization of DAEs and invariant manifolds”, submitted to, *Numer. Math.* 1994

Atkinson, K. E., An Introduction to Numerical Analysis, New York: John Wiley & Sons, 2nd Edition, 1989

Axelsson, O., “A note on a class of strongly A-stable methods,” *BIT*, vol. 12, pp. 1-4, 1972

Bader, G., Deuflhard, P., “A semi-implicit mid-point rule for stiff systems of ordinary differential equations,” *Numer. Math.*, vol. 41, pp. 373-398, 1983

Baumgarte, J., “Stabilization of constraints and integrals of motion in dynamical systems,” *Comp. Meth. In Appl. Mech. and Eng.*, vol. 1, pp. 1-16, 1972

Bischof, C., Carle, A., Khademi, P., Mauer, A., “The ADIFOR2.0 System for the Automatic Differentiation of Fortran77 Programs,” Argonne Preprint ANL-MCS-P481-1194, 1994

Brankin, R. W., Gladwell, I., Shampine, L. F., “Starting BDF and Adams Codes at Optimal Order,” *J. Comp. Appl. Math.*, vol. 21, pp. 357-368, 1988

Brasey, V., “Half-explicit method for semi-explicit differential-algebraic equations of index 2,” Thesis No. 2664, Sect. Math., University of Geneva, 1994

Brenan, K. E., Campbell, S. L., Petzold, L. R., The Numerical Solution of Initial Value Problems in Ordinary Differential-Algebraic Equations. New York: North Holland Publishing Co., 1989

Brown, P. N., Byrne, G. D., Hindmarsch, A. C., “VODE: a variable coefficient ODE solver,” *SIAM J. Sci., Stat. Comput.*, vol. 10, pp. 1038-1051, 1989

Corwin, L. J., Szczarba, R. H., Multivariable Calculus, New York: Marcel Dekker, 1982

DADS Reference Manual, Revision 8.0, CADSI, Coralville, Iowa, 1995

Dahlquist, G., “A special stability problem for linear multistep methods,” *BIT*, vol. 3, pp. 27-43, 1963

Dormand, J. R., Prince, P. J., “A family of embedded Runge-Kutta formulae,” *J. Comp. Appl. Math.*, vol. 6, pp. 19-26, 1980

Duff, I. S., “Harwell MA28 - A set of FORTRAN subroutines for sparse unsymmetric linear equations,” Report AERE-R8730, 1980

Ehle, B. L., “High order A-stable methods for the numerical solution of systems of DEs,” *BIT*, vol. 8, pp. 276-278, 1968

Eich, E., “Convergence results for a coordinate projective method applied to mechanical systems with algebraic constraints,” *SIAM J. Numer. Anal.*, vol. 30, pp. 1467-1482, 1993

Eich, E., Fuhrer, C., Leihmkuhler, B. J., Reich, S., “Stabilization and Projection Methods for Multibody Dynamics,” Helsinki University of Technology, Institute of Mathematics, Research Report A281, 1990

Eich, E., Fuhrer, C., Yen, J., “On the Error Control for Multistep Methods Applied to ODEs with Invariants and DAEs in Multibody Dynamics,” *Mech. Struct.&Mach.*, vol.23(2), 1995

Fuhrer, C., Leihmkuhler, B. J., “Numerical Solution of Differential-Algebraic Equations for Constraint Mechanical Motion,” *Numerische Mathematik*, vol. 59, pp. 5-69, 1991

Gear, C. W., Numerical Initial Value Problems in Ordinary Differential Equations, Prentice Hall, 1971

Gear, C. W., Gupta, G. K., Leihmkuhler, B. J., ”Automatic Integration of the Euler-Lagrange Equations with Constraints,” *J. Comp. Appl. Math.*, vol.12&13, pp. 77-90, 1985

Golub, G. H., Van Loan, C. F., Matrix Computations, John Hopkins University Press, 1989

Hairer, E., Nørsett S. P., Wanner, G., Solving Ordinary Differential Equations I. Nonstiff Problems, Berlin Heidelberg New York: Springer-Verlag, 1993

Hairer, E., Wanner, G., Solving Ordinary Differential Equations II. Stiff and Differential-Algebraic Problems, Berlin Heidelberg New York: Springer-Verlag, 1996.

Haug, E. J., Computer-Aided Kinematics and Dynamics of Mechanical Systems. Boston, London, Sydney, Toronto: Allyn and Bacon, 1989

Haug, E. J., Negrut, D., Engstler, C., "Runge-Kutta Integration of the Equations of Multibody Dynamics in Descriptor Form," in preparation, 1998

Haug E. J., Negrut, D., Iancu, M., "A State-Space Based Implicit Integration Algorithm for Differential-Algebraic Equations of Multibody Dynamics," *Mech. Struct.&Mach.*, vol. 25(3), pp. 311-334, 1997(a)

Haug E. J., Negrut, D., Iancu, M., "Implicit Integration of the Equations of Multibody Dynamics," in Computational Methods in Mechanical Systems, J. Angeles and E. Zakhariev eds., NATO ASI Series: Springer-Verlag, vol. 161, pp. 242-267, 1997(b)

Haug, E. J., Yen, J., "Implicit Numerical Integration of Constrained Equations of Motion Via Generalized Coordinate Partitioning," *J. Mech. Design*, vol. 114, pp. 296-304, 1992

Horn, M. K., "Forth and fifth-order scaled Runge-Kutta algorithms for treating dense output," *SIAM J. Numer. Anal.*, vol. 20, pp. 558-568, 1983

Iancu, M., Haug, E. J., Negrut, D., "Implicit Numerical Integration of the Equations of Stiff Multibody Dynamics: Descriptor Form," in Proceeding of the NATO Advanced Study Institute in Computational Methods in Mechanisms, vol. II, pp. 91-99, J. Angeles and E. Zakhariev eds., Varna, Bulgaria, 1997

Kaps, P., Rentrop, P., "Generalized Runge-Kutta methods of order four with step-size control for stiff ordinary differential equations," *Numer. Math.*, vol. 33, pp. 55-68, 1979

Krogh, F. T., "A variable step variable order mutistep method for the numerical solution of ordinary differential equations," *Information Processing*, North-Holland, Amsterdam, vol. 68, pp. 194-199, 1969

Lapack Users' Guide, *SIAM*, Philadelphia, 1992

Lewis, G. L., "Implementation of the Gibbs-Poole-Stockmeyer and Gibbs-King algorithms," *ACM Trans. on Math. Soft.*, vol. 8, pp. 180-189, 1982

Liang, G. G., Lance, G. M., "A Differentiable Null Space Method for Constrained Dynamic Analysis," *ASME Journal of Mechanisms, Transmissions, and Automation in Design*, vol. 109, pp. 405-411, 1987

Lubich, C., "Extrapolation Methods for Constrained Multibody Systems," Technical Report A-6020, University of Innsbruck, Institute for Mathematics and Geometry, Innsbruck, 1990

Lubich, C., "Extrapolation integrators for constrained multibody systems," *Impact Comp. Sci. Eng.*, vol. 3, pp. 213-234, 1991

Lubich, C., Nowak, U., Pohle, U., Engstler, C., "MEXX-numerical software for the integration of constrained mechanical multibody systems," Preprint SC 92-12, Konrad-Zuse-Zentrum, Berlin, 1992

Mani, N. K., Haug, E. J., Atkinson, K. E., "Application of Singular Value Decomposition for the Analysis of Mechanical System Dynamics," *ASME Journal of Mechanisms, Transmissions, and Automation in Design*, vol. 107, pp. 82-87, 1985

NADS Vehicle Dynamics Software, vol. 2, Release 4, Center for Computer Aided Design, The University of Iowa, 1995

Negrut, D., Haug, E. J., Iancu, M., "Variable Step Implicit Numerical Integration of Stiff Multibody Systems," in Proceeding of the NATO Advanced Study Institute in Computational Methods in Mechanisms, vol. II, pp. 157-166, J. Angeles and E. Zakhariev eds., Varna, Bulgaria, 1997

Negrut, D., Serban, R., Potra, F. A., " A Topology Based Approach for Exploiting Sparsity in Multibody Dynamics," The University of Iowa, Dept. of Mathematics, Report No. 84, Dec. 1995

Negrut, D., Serban, R., Potra, F. A., "A Topology Based Approach for Exploiting Sparsity in Multibody Dynamics: Joint Formulation,", *Mech. Struct.&Mach*, vol. 25(2), pp. 221-241, 1997

Nordsieck, A., "On numerical integration of ordinary differential equations," *Math. Comp.*, vol. 16, pp. 22-49, 1962

Nørsett, S. P., Wolfbrandt, A., "Order Conditions for Rosenbrock Types Methods," *Numer. Math.*, vol. 38, pp. 193-208, 1979

Ostermeyer, G. P., "Baumgarte stabilization for differential algebraic equations". In NATO Advance Research Workshop in Real-Time Integration Methods for Mechanical System Simulation, E. Haug and R. Deyo, eds. Berlin Heidelberg New York: Springer-Verlag, 1990

Petzold, L. R., "Differential/Algebraic Equations are not ODE's", *SIAM J. Sci., Stat. Comput.*, vol. 3(3), pp. 367-384, 1982

Potra, F. A., "Implementation of linear multistep methods for solving constrained equations of motion," *SIAM. Numer. Anal.*, vol. 30(3), pp. 474-489, 1993

Potra, F. A., "Numerical Methods for the Differential-Algebraic Equations with Application to Real-Time Simulation of Mechanical Systems," *ZAMM*, vol. 74(3), pp. 177-187, 1994

Potra, F. A., Rheinboldt, W. C., “Differential-Geometric Techniques for Solving Differential Algebraic Equations,” In *Real-Time Integration of Mechanical System Simulation*, E. Haug and R. Deyo, eds., Springer-Verlag, Berlin, 1990

Potra, F. A., Rheinboldt, W. C., “On the numerical solution of Euler-Lagrange equations,” *Mech. Struct. & Mech.*, vol. 19(1), pp. 1-18, 1991

Prothero, A., Robinson, A., “On the stability and accuracy of one-step methods for solving stiff systems of ordinary differential equations,” *Math. of Comput.*, vol. 28, pp. 145-162, 1974

Rheinboldt, W. C., “Differential-Algebraic Systems as Differential Equations on Manifolds,” *Math. Comp.,* vol. 43, pp. 473-482, 1984

Sandu, A., Negrut, D., Haug, E. J., Potra, A., Sandu, C., “A Rosenbrock Method for State Space Based Integration of Differential Algebraic Equations of Multibody Dynamics,” in preparation, 1998

Schiehlen, W., Multibody Systems Handbook. Berlin, Heidelberg, New York: Springer-Verlag, 1990

Serban, R., “Dynamic and Sensitivity Analysis of Multibody Systems,” Ph.D. Thesis, The University of Iowa, 1998

Serban, R., Negrut, D., Haug, E. J., “HMMWV Multibody Models,” Technical Report R-211, Center for Computer-Aided Design, The University of Iowa, 1998

Serban, R., Negrut, D., Haug, E. J., Potra, F. A., "A Topology Based Approach for Exploiting Sparsity in Multibody Dynamics in Cartesian Formulation,” *Mech. Struct.&Mach*., vol. 25(3), pp. 379-396, 1997

Shampine, L. F., “Implementation of implicit formulas for the solution of ODEs,” *SIAM J. Sci. Stat. Comput.*, vol. 1, pp. 103-118, 1980

Shampine, L. F., “Conservation Laws and the Numerical Solution of ODEs,” *Comp. And Math. With Appls.*, vol. 12B, pp. 1287-1296, 1986

Shampine, L. ,F., Numerical Solution of Ordinary Differential Equations, Chapmann & Hall, New York, 1994

Shampine, L. F., Gordon, M. K., Computer Solution of Ordinary Differential Equations. The Initial Value Problem, Freeman and Company, San Francisco, 1975

Shampine, L. F., Watts, H. A., “The art of writing a Runge-Kutta code.II,” *Appl. Math. Comput.*, vol. 5, pp. 93-121, 1979

Shampine, L. F., Zhang, W., “Rate of Convergence of Multi-step Codes Started by Variation of Order and Step-size,” *SIAM J. Numer. Anal.*, vol. 27, pp. 1506-1518, 1990

Tsai, F. F., "Automated methods for high-speed simulation of multibody dynamic systems," Ph.D. Thesis, The University of Iowa, 1989

Verner, J. H., "Explicit Runge-Kutta methods with estimates of the local truncation error," *SIAM J. Numer. Anal.*, vol. 15, pp.772-790, 1978

Wehage, R. A., Haug, E. J., "Generalized Coordinate Partitioning for Dimension Reduction in Analysis of Constrained Dynamic Systems," *J. Mech. Design*, vol. 104, pp. 247-255, 1982

Winckler, M. J., "Semiautomatic Discontinuity Treatment in FORTRAN77-Coded ODE Models," in Proceedings of the 15th IMACS World Congress 1997 on Scientific Computation Modeling and Applied Mathematics, 1997

Yen, J., "Constrained Equations of Motion in Multibody Dynamics as ODEs on Manifolds," *SIAM J. Numer. Anal.*, vol. 30, pp. 553-568, 1993