

# Simulating Mechanical Systems in Simulink with SimMechanics

Giles D. Wood  
Dallas C. Kennedy

The MathWorks, 3 Apple Hill Drive, Natick, MA, USA  
www.mathworks.com  
91124v00

## Abstract

This paper systematically presents the mathematical and software developments needed for efficient simulation of mechanical systems in the Simulink<sup>®</sup> dynamic simulation environment. It should be of interest to specialists and nonspecialists alike.

*Copyright 2003* The MathWorks. No part of this article may be photocopied or reproduced in any form without prior written consent from The MathWorks, Inc.

## 1 Introduction

The equations of motion of mechanical systems have undergone historical development associated with such distinguished mathematicians as Newton, D'Alembert, Euler, Lagrange, Gauss, and Hamilton, among others. While all made significant contributions to the subject, often providing elegant abstract formulations of the underlying equations of motion, our interest here is on the computational aspects of mechanical simulation and the implementation of efficient solvers in an existing dynamic simulation package. Introducing mechanical simulation capability into an existing and well-established simulation environment such as Simulink makes for a number of interesting challenges. The equations governing the motion of mechanical systems are typically higher-index differential algebraic equations (DAEs). Simulink is currently designed to model systems governed by ordinary differential equations (ODEs) and a restricted class of index-1 DAEs [25, 26].

### 1.1 Modeling multibody systems

The reader is assumed to be familiar with the concept of a multibody system as an abstract collection of bodies whose relative motions are constrained by means of joints and other, perhaps

more complicated, constraints. The reader is also assumed to be familiar with the representation of a multibody system by means of an abstract graph. The bodies are placed in direct correspondence with the nodes of the graph, and the constraints (where pairs of bodies interact) are represented by means of edges. We will have cause to distinguish between two fundamental types of systems, multibody systems whose graphs are acyclic, often referred to as tree topology systems, and multibody systems that give rise to cyclic graphs with closed loops.

There are numerous derivations of the governing equations of motion of a multibody system [13, 16, 20, 23, 24]. Various protagonists adhere quite strongly to certain formulations in preference to others. But, in all cases, the equations of motion of a multibody system can be written in the following descriptor form:

$$\dot{q} = \tilde{H}v \quad (1)$$

$$M(q)\dot{v} = f(t, q, v) + \tilde{H}^T(q)G^T(q, t)\lambda \quad (2)$$

$$g(q, t) = 0 \quad . \quad (3)$$

Here  $q : \mathbb{R} \rightarrow \mathbb{R}^{n_q}$  is the vector of configuration variables used to define the configuration of the multibody system at any instant of time  $t$ . The mass matrix  $M(q) \in \mathbb{R}^{n_v \times n_v}$  is a symmetric positive-definite matrix.  $f : \mathbb{R} \times \mathbb{R}^{n_q} \times \mathbb{R}^{n_v} \rightarrow \mathbb{R}^{n_v}$  represents the contribution of centrifugal, Coriolis, and external forcing terms. The matrix  $H : \mathbb{R}^{n_q} \rightarrow \mathbb{R}^{n_v}$  is a purely kinematic relationship between the velocity variable  $v : \mathbb{R} \rightarrow \mathbb{R}^{n_v}$  and the derivative of the configuration variable  $q$ , expressed as  $v = H(q)\dot{q}$ . Often  $H$  is the identity mapping. In the case of spatial mechanisms, however, an over-parameterized representation of orientation is convenient, to avoid the usual difficulties with representation singularities. In this case, the velocity variable  $v$  is taken to be the angular velocity  $\omega$  of a body and  $q$  a set of four Euler parameters representing the orientation of the body.  $H$  then represents the mapping from the time derivative of the vector  $q$  to the angular velocity vector  $\omega$ . The matrix  $\tilde{H} : \mathbb{R}^{n_v} \rightarrow \mathbb{R}^{n_q}$  denotes the right inverse of  $H$ . Typically  $H$  has full row rank in which case  $\tilde{H} = H^T(HH^T)^{-1}$ .

The equation  $g : \mathbb{R}^{n_q} \times \mathbb{R} \rightarrow \mathbb{R}^{n_m}$  defines an  $n_q - n_m$  dimensional time-dependent sub-manifold of  $\mathbb{R}^{n_q}$  and it is assumed that the velocity constraint Jacobian  $G = \partial g / \partial q$  has full row rank (it is further assumed that  $g \in C^2$  is twice continuously differentiable). The appearance of the time variable  $t$  in the constraint equations allows for modeling of explicit motion drivers. Apart from requiring consistency among these drivers, it does not complicate the analysis significantly. Finally the variable  $\lambda$  represents the vector of Lagrange multipliers associated with the constraint forces.

The form of the equations of motion and the structure of their matrices depend on the type of coordinates that define the configuration of the system. Ideally it should be possible to choose a minimal set of independent coordinates, equal in number to the global degrees of freedom exhibited by the mechanical system. This eliminates the need for constraint equations. Unfortunately, for anything other than the most trivial systems, it is not possible to obtain a global chart parameterizing the manifold defined by the equation  $g(q, t) = 0$ . The problem of solving the system defined by equation 3 (an index-3 DAE) remains. Simulink, not at present designed

for DAE analysis, has difficulty solving such systems.

## 1.2 Relative versus absolute coordinate formulations

Another consideration is the need for formulating the equations in a way that is both transparent to analysis and that efficiently computes the acceleration vector  $\dot{v}$ . In this regard, the two most popular choices of coordinates are relative coordinates and absolute coordinates (or reference coordinates). In the case of absolute coordinates, each body in the mechanism is given six degrees of freedom by describing its position as a reference point on the body and its orientation with a set of four Euler parameters. In this approach, the mass matrix  $M(q)$  is particularly simple, being block diagonal. But every interaction between a pair of bodies must be represented by a set of constraint equations. (An example is two bodies connected by a joint.) Invariably this leads to a model that has a large number of configuration variables (the dimension of  $q$  can be quite large even for simple systems), and a correspondingly large number of constraint equations. Despite these disadvantages, absolute coordinate formulations are typically the method of choice for commercial packages, where software criteria such as uniformity and ease of constraint representation often dominate.

By contrast, relative coordinate approaches minimize the number of coordinates necessary for representing the configuration by implicitly parameterizing certain constraints (for example, joint interactions) between bodies. This re-parameterization is accomplished by restricting the relative motion between bodies to an allowable subspace. This typically results in far fewer variables in the configuration vector  $q$  and a corresponding reduction in the number of constraint equations<sup>1</sup>, as compared to the absolute coordinate formulation. While the dimension of  $q$  and the number of constraint equations is significantly reduced, a drawback with this approach is that the mass matrix  $M(q)$  now becomes dense and the constraint equations more complicated to express. The computational cost of constructing and inverting the mass matrix contributes significantly to the overall computational cost of the formulation, and so is an important aspect to consider.

The evolution of recursive computational techniques makes it possible to factorize the mass matrix and to invert it very efficiently, even when using a relative coordinate formulation [2, 7, 10, 11, 21, 22, 28]. For this reason, the SimMechanics solvers use a relative coordinate formulation together with recursive computational procedures. This is in keeping with the general approach to system simulation currently used in Simulink, which is based around ODEs. Clearly it is necessary that the formulation be amenable to integration by all of the existing numerical solvers in Simulink. The choice of relative coordinates allows an important class of mechanisms (for example, many robotic systems) to be simulated without constraint equations. Where constraints do arise, they will be few in number, making it much easier to solve them efficiently. There are systems where the number of constraints is proportional to the number of bodies, even when relative coordinates are used. Such systems are pathological and do not arise often in practice.

---

<sup>1</sup>In some cases the constraints can be completely eliminated, this is true for systems that have an acyclic graph, but systems represented by a cyclic graph still generate constraint equations.

The rest of this paper is organized as follows:

- Section 2 introduces notation and derives the recursive techniques needed to implement the equations of motion for simple serial chains.
- Section 3 addresses some of the interesting issues that arise when attempting to model constrained systems in current versions of Simulink. Section 3.1 examines the issue of efficient Lagrange multiplier computation. Section 3.2 discusses problems that arise with singularities. In Section 3.3 the problem of numerical drift is outlined with Section 3.4 covering approaches based on projection and Section 3.5 covering stabilization methods for real-time applications. Section 3.6 deals with redundant constraints and examines the problem of consistent motion drivers.
- Linearization of mechanical systems is examined in Section 4 followed by trimming in Section 5.
- Finally in Section 6 we briefly discuss the interesting topic of discrete topology changes due to joint lockup.

## 2 Unconstrained systems

In this section we restrict attention to the class of multibody systems represented by noncyclic graphs, otherwise known as branched trees, by virtue of the fact that the graphs have the form of a rooted tree. For pedagogical purposes we further restrict attention to the subset of simple-chain systems. The results are easily extended to the more general case of branched trees but the analysis is much clearer in the case of simple chains. Simple chains are multibody systems where each body is connected to a unique predecessor body and a unique successor body (with the exception of the first and last bodies in the chain) by means of a joint. A common example is robotic manipulators. In recent years, a number of techniques have appeared for solving the dynamics of these systems, ranging in computational complexity from  $O(n^3)$  (where  $n$  is the number of bodies in the system) – for example the composite rigid body method (CRBM) [30] – to  $O(n)$  articulated body methods (ABM) [10, 11, 21, 22].

Close connections exist between the problem of computing the dynamics of simple chains and branched trees and certain two-point boundary value problems that arise in discrete optimal control and estimation theory [21]. For simplicity, we present a derivation based on the work in [18], where the different approaches are derived by forming an augmented system of equations and performing block matrix elimination. This yields a uniform derivation of the  $O(n)$  ABM and the  $O(n^3)$  CRBM and should be amenable to readers who do not have a background in control theory.

## 2.1 Analysis of simple chain systems

Consider the  $n$ -link serial chain in Figure 1, where the links are numbered consecutively from the tip of the chain to the base, which also acts as the inertial reference frame. Each body is connected to two joints, an inboard joint (closer to the base), and an outboard joint (closer to the tip). An arbitrary joint, the  $k$ th joint, has an inboard side labeled  $O_k^+$  attached to body  $k+1$  and an outboard side labeled  $O_k$  attached to the  $k$ th body. On the  $k$ th joint, the vector from the outboard side of the joint  $O_k$  to the center of mass on the  $k$ th body is denoted by  $p(k)$ , and the vector from  $O_k$  to  $O_{k-1}$  is denoted by  $r(k, k-1)$ . This vector plays an important role in shifting forces and velocities between bodies. The spatial velocity  $V(k) \in \mathbb{R}^6$  at the outboard side of the  $k$ th joint is defined by  $V(k) = [\omega(k)^T, v(k)^T]^T$ , where  $\omega(k) \in \mathbb{R}^3$  is the angular velocity of the  $k$ th body and  $v(k) \in \mathbb{R}^3$  is the linear velocity of the  $k$ th body at the point  $O_k$ . Here we assume that both vectors are expressed in inertial coordinates. The spatial force  $f(k) \in \mathbb{R}^6$  applied by the  $k$ th joint to the  $k$ th body at the point  $O(k)$  is defined by  $f(k) = [T(k)^T, F(k)^T]^T$ , where  $T(k) \in \mathbb{R}^3$  is the applied torque and  $F(k) \in \mathbb{R}^3$  is the applied force, again expressed in inertial coordinates.

The spatial inertia matrix  $M(k)$  is defined as

$$M(k) = \begin{bmatrix} J(k) & m(k)\tilde{p}(k) \\ -m(k)\tilde{p}(k) & m(k)I_3 \end{bmatrix} \in \mathbb{R}^{6 \times 6},$$

where  $m(k)$  is the mass of the  $k$ th body, and  $J(k) \in \mathbb{R}^{3 \times 3}$  is the inertia tensor of the  $k$ th body about the center of mass, computed in inertial coordinates. Given a vector  $p \in \mathbb{R}^3$  we let  $\tilde{p} \in \mathbb{R}^{3 \times 3}$  denote the cross product matrix generated from the vector  $p$ . Thus  $\tilde{p}(k)$  denotes the cross product matrix generated from  $p(k)$ . Denote the time derivative of  $V(k)$  by  $\alpha(k)$ . The following kinematics equations describe the motion of the chain:

$$V(k) = \phi^T(k+1, k)V(k+1) + H^T(k)\dot{q}(k), \quad V(n+1) = 0, \quad k = n, n-1, \dots, 1 \quad (4)$$

$$\alpha(k) = \phi^T(k+1, k)\alpha(k+1) + H^T(k)\ddot{q}(k) + a(k), \quad \alpha(n+1) = 0, \quad k = n, n-1, \dots, 1 \quad (5)$$

Here  $q(k) \in \mathbb{R}^{n_{qk}}$  is the vector of configuration variables for the  $k$ th joint, the columns of  $H^T(k) \in \mathbb{R}^{6 \times n_{qk}}$  span the relative velocity space between the  $k$ th body and the  $(k+1)$ th body, and  $\phi^T(k+1, k)$  is the transpose of the matrix

$$\phi(k+1, k) = \begin{bmatrix} I_3 & \tilde{r}(k+1, k) \\ 0 & I_3 \end{bmatrix},$$

where the vector  $r(k+1, k)$  is the vector from the outboard side of the  $(k+1)$ th joint to the outboard side of the  $k$ th joint. The vector  $a(k) \in \mathbb{R}^6$  represents the Coriolis acceleration of the  $k$ th body and is given by

$$a(k) = \begin{bmatrix} \tilde{\omega}(k+1)\omega(k) \\ \tilde{\omega}(k+1)(v(k) - v(k+1)) \end{bmatrix}.$$

The role of various terms can be clarified with an example. Suppose the  $k$ th joint is a simple

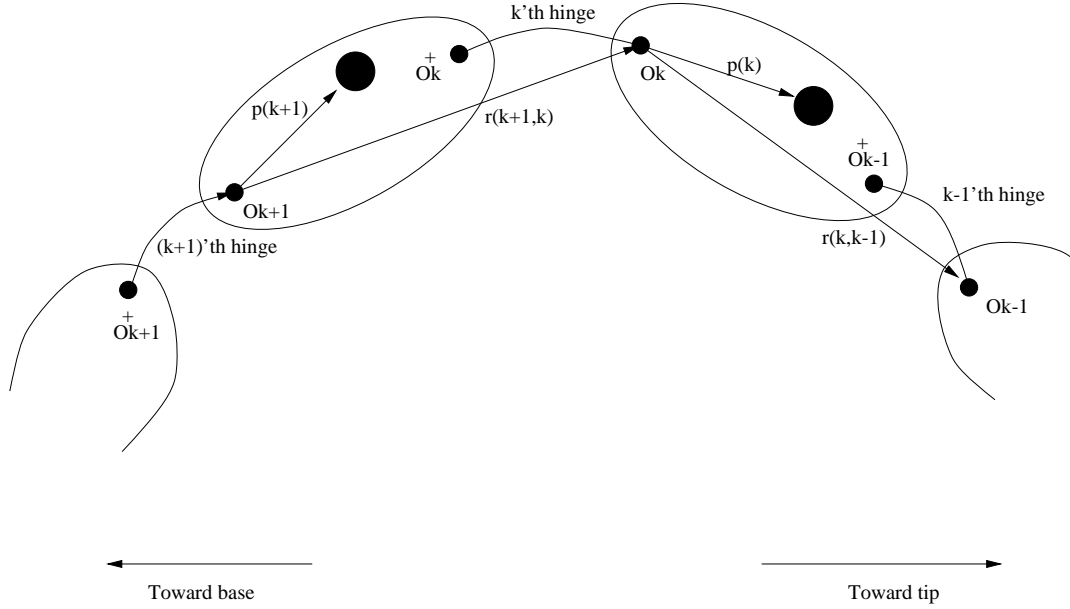


Figure 1: Bodies and joints in a serial chain.

revolute joint. This joint only allows rotation between the  $(k+1)$ th body and the  $k$ th body. In this case, the variable  $q(k)$  represents the rotation angle between the two bodies and the matrix  $H(k)$  is given by  $H(k) = [u_k^T, 0_{3 \times 1}^T]$ , with  $u_k \in \mathbb{R}^3$  being the unit vector along the instantaneous axis of the revolute joint (in inertial coordinates). The kinematic relationships for more complex joints can also be expressed in this form. The matrix  $\phi(k+1, k)$  shifts spatial forces (according to the usual rules for shifting forces between points on a rigid body) from the point  $O(k)$  to the point  $O(k+1)$  while the transpose (or adjoint) shifts spatial velocities from  $O(k+1)$  to  $O(k)$ . Given the joint velocity degrees of freedom (DoFs)  $\dot{q}(k)$  and acceleration DoFs  $\ddot{q}(k)$ , these equations provide a simple recursive procedure for determining the velocities and accelerations of the bodies constituting the chain.

The equations of motion, formulated about the reference points  $O(k)$ , are

$$\begin{aligned} f(k) &= \phi(k, k-1)f(k-1) + M(k)\alpha(k) + b(k) \quad , \quad f(0) = 0_{6 \times 1}, \quad k = 1, \dots, n \\ T(k) &= H(k)f(k) \quad . \end{aligned} \quad (6)$$

The vector  $b(k) \in \mathbb{R}^6$  is the gyroscopic force vector at  $O(k)$ :

$$b(k) = \begin{bmatrix} \tilde{\omega}(k)I(k)\omega(k) \\ m(k)\tilde{\omega}(k)\tilde{\omega}(k)p(k) \end{bmatrix} .$$

The second equation in 6 is obtained by performing a virtual work balance across the  $k$ th joint (a massless entity). With the externally applied generalized torque vector<sup>2</sup> for the joint given

<sup>2</sup>These could be torques or forces applied by motors around various joint axes.

by  $T(k)$ , we obtain

$$T^T(k)\delta q_k = f^T(k)H^T(k)\delta q_k \quad , \quad \forall \delta q_k \in \mathbb{R}^{n_{q_k}} \quad .$$

Since  $\delta q_k$  is arbitrary, this gives the desired relationship between the externally applied generalized torque  $T(k)$  and the reaction force  $f(k)$ . Readers familiar with optimal control theory will recognize equation 5 and equation 6 as being similar to those that arise in the optimal control equations of discrete dynamic systems [6]. After substituting  $f(k) = P(k)\alpha(k) + z(k)$ , sweep methods lead to a discrete Riccati equation for  $P(k)$  (which can be solved recursively) and then directly to the desired joint accelerations. The solution, when expressed in matrix form, leads to a square factorization of the mass matrix and its inverse.

## 2.2 Recursive solution

We can express these recursive relationships in matrix form as follows. Sum the velocity recursion and use the fact that  $\phi(i, i) = I_6$  and  $\phi(i, k)\phi(k, j) = \phi(i, j)$ . The second relationship follows from the fact that shifting a spatial force from  $O_j$  to  $O_k$  and then from  $O_k$  to  $O_i$  is equivalent to shifting the force directly from  $O_j$  to  $O_i$ . Then

$$V(k) = \sum_{i=k}^n \phi^T(i, k)H^T(i)\dot{q}_i \quad .$$

A natural definition of the matrix operators follows:

$$H^T = \text{diag}[H^T(1), \dots, H^T(n)] \quad .$$

The operator  $\phi$  is defined by:

$$\phi = \begin{bmatrix} I_{6 \times 6} & 0 & \dots & 0 \\ \phi(2, 1) & I_{6 \times 6} & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ \phi(n, 1) & \phi(n, 2) & \dots & I_{6 \times 6} \end{bmatrix} \quad .$$

In terms of the matrix  $\phi$ , the velocity and force recursions can be written as

$$\begin{aligned} V &= \phi^T H^T \dot{q} \\ \alpha &= \phi^T [H^T \ddot{q} + a] \\ f &= \phi [M\alpha + b] \\ T &= Hf \quad , \end{aligned}$$

where the spatial velocity vector  $V$  is defined as  $V^T = [V^T(1), V^T(2), \dots, V^T(n)] \in \mathbb{R}^{6n}$ , and similarly for the spatial acceleration vector  $\alpha \in \mathbb{R}^{6n}$ , the Coriolis acceleration vector  $a \in \mathbb{R}^{6n}$ , the gyroscopic force vector  $b \in \mathbb{R}^{6n}$ , and the spatial force vector  $f \in \mathbb{R}^{6n}$ . Finally the matrix  $M \in \mathbb{R}^{6n \times 6n}$  is defined to be  $M = \text{diag}[M(1), M(2), \dots, M(n)]$ . Substituting into the last equation, we obtain the equations of motion:

$$T = H\phi M\phi^T H^T \ddot{q} + H\phi [M\phi^T a + b] \quad . \quad (7)$$

Equation 7 is exactly what would result from a Lagrangian analysis of the system. We can see that the mass matrix  $\mathcal{M} = H\phi M\phi^T H^T$  and the Coriolis vector  $\mathcal{C} = H\phi[M\phi^T a + b]$  are both expressed in a factorized form. It is easy to implement equation 7 recursively. This form of the equations of motion is called the Newton-Euler factorization [22]. Equation 7 could be used to recursively compute  $T - H\phi[M\phi^T a + b]$ . One would update the positions and the velocities of all bodies in the chain using a base to tip recursion from knowledge of  $q$  and  $\dot{q}$ , setting  $\ddot{q} = 0$  in all the joints (an artificial zero acceleration computation), followed by a tip to base recursion to compute the spatial force vector  $f$  that is consistent with this motion. The desired term is then given by  $T - H\phi[M\phi^T a + b] = T - Hf$ , where  $f$  is the spatial force vector just computed. A similar strategy can be used to construct the mass matrix  $\mathcal{M}$ . The CRBM is essentially implemented in this fashion, but in order to compute the joint acceleration vector  $\ddot{q}$ , it is necessary to factorize the mass matrix  $\mathcal{M}$ , usually using Cholesky factorization, an  $O(n^3)$  operation. Since the equations of motion have been expressed using relative coordinates, the mass matrix  $\mathcal{M}$  is typically dense and there is little or no opportunity for exploiting any sparseness.

To derive the  $O(n)$  ABM, we follow [7] and first recast the kinematic and dynamic relationships into a block diagonal form. For instance in the case of a chain with  $n = 3$  bodies the equations can be written as  $\tilde{M}\tilde{x} = \tilde{b}$  where

$$\tilde{x} = \begin{bmatrix} -\alpha(1) \\ \ddot{q}(1) \\ f(1) \\ -\alpha(2) \\ \ddot{q}(2) \\ f(2) \\ -\alpha(3) \\ \ddot{q}(3) \\ f(3) \end{bmatrix}, \quad \tilde{b} = \begin{bmatrix} b(1) \\ T(1) \\ -a(1) \\ b(2) \\ T(2) \\ -a(2) \\ b(3) \\ T(3) \\ -a(3) \end{bmatrix},$$

$$\tilde{M} = \begin{bmatrix} M(1) & 0 & I_{6 \times 6} & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & H(1) & 0 & 0 & 0 & 0 & 0 & 0 \\ I_{6 \times 6} & H^T(1) & 0 & -\phi^T(2,1) & 0 & 0 & 0 & 0 & 0 \\ \hline 0 & 0 & -\phi(2,1) & M(2) & 0 & I_{6 \times 6} & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & H(2) & 0 & 0 & 0 \\ 0 & 0 & 0 & I_{6 \times 6} & H^T(2) & 0 & -\phi^T(3,2) & 0 & 0 \\ \hline 0 & 0 & 0 & 0 & 0 & -\phi(3,2) & M(3) & 0 & I_{6 \times 6} \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & H(3) \\ 0 & 0 & 0 & 0 & 0 & 0 & I_{6 \times 6} & H^T(3) & 0 \end{bmatrix}. \quad (8)$$

The elimination procedure then proceeds as follows. For  $k = 1, 2, \dots, n$ , eliminate the off-diagonal terms in the middle row of each block using the first row of the block and then the last row of the block. Then using the  $k$ th block block-rows, eliminate the term  $-\phi(k+1, k)$  coupling the  $k$ th block with the  $(k+1)$ th block. This elimination produces a decoupled  $(k+1)$ th block

where only  $M(k + 1)$  needs to be updated to

$$\hat{M}(k + 1) = M(k + 1) + \phi(k + 1, k)\hat{M}(k)\phi^T(k + 1, k) - \phi(k + 1, k)\hat{M}(k)H^T(k)(H(k)\hat{M}(k)H^T(k))^{-1}H(k)\hat{M}(k)\phi^T(k + 1, k), \quad \hat{M}(1) = M(1) \quad .$$

A corresponding update to the right hand side replaces  $T(k)$  with  $\hat{T}(k) = T(k) - H(k)\hat{b}(k) - H(k)\hat{M}(k)a(k)$ , where

$$\hat{b}(k+1) = b(k+1) + \phi(k+1, k)[\hat{b}(k) + \hat{M}(k)a(k)] + \phi(k+1, k)\hat{M}(k)H^T(k)[H(k)\hat{M}(k)H^T(k)]^{-1}\hat{T}(k) \quad .$$

Following this elimination procedure, the mass matrix has the zero structure given below (where a  $\times$  stands for a possibly nonzero term and a  $\otimes$  stands for a non-zero term).

$$\left[ \begin{array}{ccc|ccc|ccc} \times & 0 & \times & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & \otimes & 0 & \times & 0 & 0 & 0 & 0 & 0 \\ \otimes & \times & 0 & \times & 0 & 0 & 0 & 0 & 0 \\ \hline 0 & 0 & 0 & \times & 0 & \times & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & \otimes & 0 & \times & 0 & 0 \\ 0 & 0 & 0 & \otimes & \times & 0 & \times & 0 & 0 \\ \hline 0 & 0 & 0 & 0 & 0 & 0 & \times & 0 & \times \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & \otimes & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & \otimes & \times & 0 \end{array} \right] \quad .$$

This procedure yields a linear system of equations for  $\ddot{q}(k)$  and  $\alpha(k)$ . A back substitution gives the ABM, with the joint accelerations  $\ddot{q}(k)$  computed using the circled pivot blocks. Since all operations are done block-wise, the computation count is clearly  $O(n)$ . These recursive techniques exploit the structure and sparsity pattern of the matrices that arise in the multibody dynamics problem to efficiently simulate systems represented by acyclic graphs. Recursive techniques are easily extended to arbitrary tree topologies and provide the underpinning of an efficient computational approach to the general case where constraints arise.

### 3 Constrained systems

A general purpose solver must be capable of dealing with constrained mechanical systems where the underlying graph is cyclic. Systems that have a cyclic graph are reduced to an open spanning tree by the removal of a subset of the joints. The joints in this cut-set are replaced by a set of constraint equations to ensure that the spanning tree undergoes precisely the same motion as the original mechanism with the cyclic topology. The choice of joints in the cut-set is often not unique and has a direct effect on the number of generated constraint equations and the sparsity pattern of the resulting constraint Jacobian. The problem of making a suitable choice is related to the problems of ordering columns in sparse matrices and can be addressed with the techniques of graph theory [9]. One such choice is to choose the cut-set so as to maximize the number of independent loops, thereby transforming the constraint equations into smaller sets of independent equations which are easier to solve. SimMechanics automatically selects the

cut-set but does allow users to determine this set manually. Astute analysts should be able to intuitively select a cut-set that exploits the properties of the specific mechanism being analyzed, if necessary.

### 3.1 Efficient computation of the Lagrange multipliers

The equations of motion of the spanning tree can be expressed using the recursive relationships outlined in section 2. This step results in the descriptor form given by equation 1, equation 2, and equation 3. Since the descriptor form is an index-3 DAE, it is necessary to reduce the index by explicitly differentiating the constraints. If we differentiate the constraint  $g(q, t) = 0$  in equation 3 twice, we get

$$G\tilde{H}v + \frac{\partial g}{\partial t} = 0 \quad (9)$$

$$G\tilde{H}\dot{v} + \frac{\partial(G\tilde{H}v)}{\partial q}\tilde{H}v + 2\frac{\partial G}{\partial t}\tilde{H}v + \frac{\partial^2 g}{\partial t^2} = 0 \quad (10)$$

Substituting the equation of motion in equation 2 into equation 10 gives

$$M\dot{v} = f + \tilde{H}^T G^T \lambda \quad (11)$$

$$G\tilde{H}M^{-1}\tilde{H}^T G^T \lambda = f_\lambda \quad (12)$$

The vector  $f_\lambda$  is given by

$$f_\lambda = -G\tilde{H}M^{-1}f - \frac{\partial(G\tilde{H}v)}{\partial q}\tilde{H}v - 2\frac{\partial G}{\partial t}\tilde{H}v - \frac{\partial^2 g}{\partial t^2} \quad (13)$$

Equation 11 and equation 12 can be treated as an index-1 DAE [29], or we can solve for  $\lambda$  directly, and treat the system as an ODE on a manifold. In the index-1 DAE approach, it is possible to exploit the structure of the equations of motion, together with carefully designed multi-step integration schemes, to implement very efficient domain specific solvers for multibody systems [29]. Such an approach still suffers from numerical drift, however, and is not an option for SimMechanics, which must work seamlessly with the current suite of ODE solvers in Simulink.

For this reason, we are forced to consider the ODE approach. There are two problems with this approach: First, we need to compute the Lagrange multipliers efficiently. Second, because we have reduced the DAE to an ODE on a manifold, blindly integrating the resulting ODE numerically gives a solution that drifts away from the desired manifold given by  $g(q, t) = 0$  [5, 27]. The same is true of the index-1 DAE approach [29]. An alternative procedure is to eliminate the Lagrange multipliers from the analysis and to make use of coordinate partitioning to split the variables into independent and dependent sets [19, 32]. The existence of such a partitioning, at least locally, follows directly from the implicit function theorem and is equivalent to an explicit local parameterization of the manifold. Our principal objection to this approach is that, if the subset of independent variables is integrated, it results in a projection of the mass matrix that eliminates the structure that we exploit in the  $O(n)$  computations. In practical applications, it is necessary to carefully monitor and recompute the parameterization as the simulation proceeds

since the choice of independent and dependent variables is somewhat arbitrary and is unlikely to be globally applicable.

The approach taken in SimMechanics is a direct approach in some ways similar to the direct approach used to solve the index-1 DAEs arising in models that have algebraic loops. The Lagrange multipliers are computed using efficient recursive computations to evaluate the coefficient matrix

$$\Lambda \triangleq G\tilde{H}M^{-1}\tilde{H}^TG^T \quad (14)$$

and the vector  $f_\lambda$ . To demonstrate this, consider the problem of forming the coefficient matrix  $\Lambda$ . The velocity Jacobian can be factorized to give

$$G = A\phi^T H^T \quad , \quad (15)$$

where the configuration-dependent matrix  $A(q, t)$  is determined by the nature of the constraints. Using this factorization, it is possible to generate the coefficient matrix in  $O(n_c \times n)$  operations, where  $n_c$  is the number of independent velocity constraint equations. Similarly the vector  $f_\lambda$  can be computed very efficiently using the factorized form of the velocity Jacobian in equation 15. This allows  $f_\lambda$  to be computed as the sum of a free acceleration solution (where the constraint forces are set to zero) and the product of the matrix  $A$  and its time derivative with the spatial acceleration vector determined from knowledge of  $q$  and  $v$  (but not  $\dot{v}$ ). Again this computation can be implemented using the recursive techniques outlined in section 2.

Solving for the Lagrange multipliers,  $\lambda$  still requires the coefficient matrix to be factorized. This is an  $O(n_c^3)$  operation. Here we see the reasoning behind the use of a relative coordinate formulation. Since the number of constraint equations  $n_c$  is typically small, it is still possible to efficiently compute the Lagrange multipliers. We note here that since the matrix  $\Lambda$  is symmetric and positive-definite, it is possible to solve for the Lagrange multipliers using an iterative solution technique like conjugate gradient iteration, all implemented using the matrix factorizations already discussed. In the absence of roundoff effects, this approach could generate a solution in  $O(n_c \times n)$  operations. However, in the general case where the coefficient matrix might be ill-conditioned, and in the absence of a suitable preconditioner, we choose to use the direct approach.

### 3.2 Handling singularities

An important issue arises when implementing the solvers to determine the Lagrange multipliers. Mechanical systems can pass through singular configurations during their motion. Singular configurations exist when the number of independent constraint equations is reduced on a finite set of points in the configuration space. At such points, the coefficient matrix  $\Lambda$  becomes rank deficient, although it is still possible in principle to determine a vector of Lagrange multipliers (no longer unique) that satisfies the determining equation. Unfortunately, in the vicinity of these singular points the ill-conditioning of  $\Lambda$  makes the computation of  $\lambda$  extremely sensitive to roundoff errors in  $f_\lambda$  [31, 17]. The most popular approach to deal with singular configurations is the use of damped least squares. This approach minimizes the sum,

$$\|\Lambda\lambda - f_\lambda\|^2 + \sigma^2\|\lambda\|^2 \quad , \quad (16)$$

where  $\sigma$  is the damping factor. Here  $\sigma$  controls the tradeoff between keeping the residual small and keeping the norm of  $\lambda$  small in the vicinity of a singularity. Unfortunately, there is no simple or generic method for determining a suitable damping factor  $\sigma$ . So we use a slightly different approach based on a truncated QR decomposition of  $\Lambda$ . This is similar, in some respects, to the truncated singular value decomposition approach [17] but is computationally more efficient. The approach is based upon QR decomposition of  $\Lambda$  with full column pivoting [15]

$$\Lambda E = QR \quad , \quad (17)$$

where  $E$  is a permutation matrix determined by the pivot selection. The pivots are chosen so that the absolute values of the terms on the diagonal of  $R$  appear in numerically decreasing order:  $|R(1,1)| > |R(2,2)| > \dots |R(n_c, n_c)|$ . It is well known that this algorithm can be used to estimate the numerical rank of a matrix [15], and in our computations, we estimate the rank  $r$  according to

$$r = \max_k : R(k,k) > n_c \epsilon_M R(1,1) \quad , \quad (18)$$

where  $\epsilon_M$  is the machine epsilon (distance from one to the next floating point number)<sup>3</sup>. If  $r < n_c$  the decomposition can be written as

$$\Lambda E = \begin{bmatrix} \tilde{Q} & \tilde{Q}^\perp \end{bmatrix} \begin{bmatrix} R_{11} & R_{12} \\ 0 & 0 \end{bmatrix} \quad , \quad (19)$$

where  $\tilde{Q} \in \mathbb{R}^{n_c \times r}$ ,  $\tilde{Q}^\perp \in \mathbb{R}^{n_c \times (n_c - r)}$ ,  $R_{11} \in \mathbb{R}^{r \times r}$ , and  $R_{12} \in \mathbb{R}^{r \times (n_c - r)}$ . The solution  $\lambda$  is then

$$\lambda = E \begin{bmatrix} R_{11}^{-1} \tilde{Q}^T f_\lambda \\ 0_{(n_c - r) \times 1} \end{bmatrix} \quad , \quad (20)$$

which is the minimum norm solution (in the Euclidean metric) in the projected subspace. SimMechanics offers users the choice between a solver based on Cholesky decomposition of the coefficient matrix, which is numerically more efficient, and a solution based on the truncated QR decomposition, which is more robust in the presence of singularities.

### 3.3 Compensating for numerical drift

In the absence of roundoff, index reduction and integration of the resulting ODE would be perfectly adequate. Unfortunately, numerical drift of the solution from the invariant manifold  $g(q,t)$  requires alternative schemes. One such scheme, based on *coordinate partitioning*, has already been mentioned. Here the coordinates are split into independent and dependent sets. Finding a robust detection method for changing the parameterization is the most challenging aspect of the scheme. This scheme is not suitable for real-time HIL simulations, an important consideration for SimMechanics as it supports code generation for real-time deployment.

Other schemes are based on *stabilization* and *coordinate projection* [3, 4, 5]. Stabilization involves the addition of extra terms to the equation of motion. These terms vanish on the manifold

<sup>3</sup>The only truly reliable manner of determining the numerical rank of a matrix is the singular value decomposition [12], although the QR decomposition is known to work very well in practice.

$g(q, t) = 0$  but have the effect of making the solution asymptotically attractive to the manifold. If the solution does drift off the manifold, it is ultimately attracted back onto it, although there are no predefined bounds on the extent of the drift. The most popular stabilization technique is Baumgarte stabilization. Its simplicity has made it a popular choice in engineering applications. But Baumgarte stabilization requires parameterization, and there is no known generic procedure for choosing these parameters to make the stabilization robust (see [14] for attempts in this regard). Indeed the choice of suitable parameters depends on the discretization scheme used to integrate the equations of motion [5], a serious practical limitation.

### 3.4 Coordinate projection

Coordinate projection involves the numerical discretization of the ODE. At the end of the discretization step, the solution is projected onto the invariant manifold. In [27], various practical aspects of these projection schemes are studied in some detail. For example, the effect of coordinate projection on step size selection, event location, and order of convergence are all considered. It is shown how the codes in the present Simulink ODE suite, ode23, ode45, ode113 and ode15s, can all be adapted to allow for coordinate projection without compromising accuracy or efficiency [25]. A fundamental change to Simulink to enable mechanical simulation with SimMechanics has been the addition of a projection method to the ODE suite that can be called once the discretized solution has been updated following the acceptance of a successful step based on the error estimates. The scheme works for the one-step Runge-Kutta formulas ode23 and ode45, as well as for variable-step variable-order codes like ode113 [27]. Finally, the standard theory for convergence in BDF codes like ode15s is still applicable when projection is carried out in this way.

An important property of the Simulink solver suite is its ability to localize and detect events using discontinuity locking (to ensure that the integrators see a continuous vector field) and its ability to output solutions at any time in the interval of integration using continuous extension formulae [25]. Both of these features are affected by projection. Simulink allows users to refine the output from the solvers using highly efficient interpolation schemes, and these interpolated outputs typically satisfy the invariants to an accuracy comparable with the accuracy of the numerical solution. In SimMechanics, the interpolated values are further projected to ensure that the invariants are satisfied. It could be argued that projection of the interpolated values reduces efficiency. Our approach avoids surprising users who might notice that the interpolants do not satisfy the invariants. More critical is the use of projection in event location. The event location capabilities of the current Simulink solvers are also built around the continuous extension formula and the use of switching functions to determine the exact location of events in time. To ensure that events are located correctly, the outputs of the interpolants are projected before sampling the switching functions. This makes event location more expensive, but is necessary for robust event detection.

For example, the projection approach is appropriate for a one-step method used to compute an approximate solution at time  $t_{n+1}$  from a solution at  $t_n$ . The step size is  $h$ , and  $t_{n+1} = t_n + h$ .

The solution takes the form

$$\begin{bmatrix} q_{n+1}^* \\ v_{n+1}^* \end{bmatrix} = h\Phi(t_n, q_n, v_n) \quad . \quad (21)$$

If we consider the nonlinear position constraint  $g(q, t) = 0$ , the predicted position variables  $q_{n+1}^*$  are projected onto the closest point on the manifold<sup>4</sup>, denoted by  $q_{n+1}$ . Linearizing the constraints about the predicted value  $q_{n+1}^*$  gives

$$0 = g(t_{n+1}, q_{n+1}) = g(t_{n+1}, q_{n+1}^*) + G(t_{n+1}, q_{n+1}^*)(q_{n+1} - q_{n+1}^*) + O(\|q_{n+1} - q_{n+1}^*\|^2) \quad . \quad (22)$$

To leading order, the projected solution  $q_{n+1}$  is obtained from solving

$$G\delta = g \quad (23)$$

$$q_{n+1} = q_{n+1}^* - \delta \quad . \quad (24)$$

If the Euclidean norm is used,

$$\delta = G^T(GG^T)^{-1}g \quad . \quad (25)$$

In [29], the projection process is repeated, and so is referred to as sequential projection, until a suitable level of convergence is attained. This is much more efficient than forming the full set of Karush-Kuhn-Tucker (KKT) equations to determine the exact minimization on the manifold and is easily motivated by the fact that error control has made the predicted solution  $q_{n+1}^*$  close to the manifold already. In practical applications, it is advantageous to perform the projection in a weighted norm. The projection weights reflect the weights used in the error control of the ODE solver. Suitable convergence is somewhat ambiguous. In SimMechanics, we offer users the ability to terminate the projection process based on relative and absolute tolerances for  $\delta$ . We also allow users to carry on the projection process to completion. This implies that the iteration is continued until the new iterates fail to reduce the residual due to rounding error. This is similar to the current approach taken in solving algebraic loops using direct methods in Simulink. The latter approach could be slightly more expensive.

### 3.5 Stabilization for code generation

Numerical projection affects an important feature of SimMechanics, the generation of stand-alone code for faster simulation and deployment on rapid prototyping systems with Real-Time Workshop<sup>®</sup>. For systems that have cyclic graphs, code implementation requires projection to be carried out in real time, and neither of the projection schemes outlined thus far are deterministic, in the sense that the number of iterations needed cannot be predicted. Stabilization codes thus provide an appealing alternative to the projection approach for real-time applications. The generalizations of the Baumgarte technique in [5] are appealing since they can be implemented without the need for selecting problem-specific parameters. For notational convenience, suppressing explicit time dependence, we can write the reduced ODE representation

---

<sup>4</sup>How close depends on the metric and potentially on the weighting. Usually, if weights are used in ODE error estimation, the same weights should be used in the projection scheme.

(with the Lagrange multipliers eliminated) of equation 1, equation 2 and equation 3 as:

$$\dot{z} = \hat{f}(z) \quad (26)$$

$$0 = c(z) = \begin{bmatrix} g(q) \\ G(q)\tilde{H}(q)v \end{bmatrix} . \quad (27)$$

Here the variable  $z$  is defined as  $z^T = [q^T, v^T]^T$ . Reference [5] considers the family of stabilization methods given by

$$\dot{z} = \hat{f}(z) - \gamma F(z)c(z) \quad , \quad (28)$$

where  $\gamma > 0$  is a parameter, and

$$F = D(CD)^{-1} \quad \text{and} \quad C(z) = \frac{\partial c}{\partial z} \quad . \quad (29)$$

The matrix  $C(z)$  is given by

$$C(z) = \begin{bmatrix} G\tilde{H} & 0 \\ \frac{\partial(G\tilde{H}v)}{\partial q} & G\tilde{H} \end{bmatrix} \quad , \quad (30)$$

and  $D(z)$  is chosen such that  $CD$  is nonsingular; for instance

$$D = \begin{bmatrix} \tilde{H}^T G^T & 0 \\ 0 & \tilde{H}^T G^T \end{bmatrix} \quad , \quad (31)$$

which gives,

$$F = \begin{bmatrix} \tilde{H}^T G^T (G\tilde{H}\tilde{H}^T G^T)^{-1} & 0 \\ 0 & \tilde{H}^T G^T (G\tilde{H}\tilde{H}^T G^T)^{-1} \end{bmatrix} \begin{bmatrix} I & 0 \\ -\frac{\partial(G\tilde{H}v)}{\partial q} \tilde{H} G^T (G\tilde{H}\tilde{H}^T G^T)^{-1} & I \end{bmatrix} . \quad (32)$$

By discretizing the stabilization term independently of the differential equation, one can show that an optimal choice of  $\gamma$  is proportional to the inverse of the step size [5]. The stabilization scheme is implemented as follows. Start with  $(q_n, v_n)$  at time  $t_n$  and integrate with some ODE numerical method to advance the solution to  $(q_{n+1}^*, v_{n+1}^*)$  at  $t = t_{n+1}$ . Stabilize, by modifying the solution

$$\begin{bmatrix} q_{n+1} \\ v_{n+1} \end{bmatrix} = \begin{bmatrix} q_{n+1}^* \\ v_{n+1}^* \end{bmatrix} - F(q_{n+1}^*, v_{n+1}^*)c(q_{n+1}^*, v_{n+1}^*) \quad . \quad (33)$$

Since the term  $\partial(G\tilde{H}v)/\partial q$  is expensive to compute, a cheaper alternative is to use

$$F = \begin{bmatrix} M^{-1}\tilde{H}^T G^T (G\tilde{H}M^{-1}\tilde{H}^T G^T)^{-1} & 0 \\ 0 & M^{-1}\tilde{H}^T G^T (G\tilde{H}M^{-1}\tilde{H}^T G^T)^{-1} \end{bmatrix} \quad , \quad (34)$$

obtained by neglecting the  $\partial(G\tilde{H}v)/\partial q$  term and setting

$$D = \begin{bmatrix} M^{-1}\tilde{H}^T G^T & 0 \\ 0 & M^{-1}\tilde{H}^T G^T \end{bmatrix} \quad . \quad (35)$$

This is similar to projection. However, this projection is carried out in a metric defined by the positive-definite mass matrix  $M$ . This scheme is particularly efficient because the term  $\Lambda = G\tilde{H}M^{-1}\tilde{H}^TG^T$  has already been computed and factorized when the Lagrange multipliers were computed. The stabilization step is computationally almost free and ideal for real-time application. Finally, an even better choice is to use the cheaper variant of  $F$ , but to apply the scheme twice [3]. This adds negligible cost but significantly increases the accuracy. This is the scheme that users can choose when simulating systems in SimMechanics with stabilization and is the scheme used to generate code for constrained systems.

### 3.6 Redundant constraints

When using a mechanical simulation tool it is easy to generate redundant constraints. In this case the constraint Jacobian  $G$  fails to be surjective, and the redundant constraints need to be identified and removed. An additional consequence is that the constraint forces are no longer unique. This may not be important if the primary interest is the motion of the system, but it could be very important if the constraint forces themselves are of interest. Also SimMechanics allows users to move certain joints and constraints with known time trajectories (motion drivers). This is an important feature, often used in computed-torque control schemes and biomechanics applications. Here it is possible for the user to overspecify the motions or to impose inconsistent motions between the driven joints.

SimMechanics uses a relative coordinate formulation, and the motion drivers are treated differently according to the nature of the graph associated with the system. For systems that have an acyclic graph, the motions induced by motion drivers can never be inconsistent and they can be dealt with very efficiently using the recursive methods of section 2 without the need for any constraint equations. It is only systems that have a cyclic topology where inconsistencies can arise, and for these systems we need to be careful to distinguish between motion drivers applied to the joints in the spanning tree and motion drivers applied to joints in the cut-set. Drivers that are applied to joints in the spanning tree can still be dealt with without adding additional constraint equations using recursive methods. They must now be checked for consistency however. Drivers applied to joints in the cut-set do result in additional constraint equations, and if these are redundant, they can result in inconsistent motions. In this section we discuss how the redundant constraints are resolved and how potential inconsistencies are detected in SimMechanics. These issues can also arise during simulation when joints lock or unlock due to stiction behavior. We delay discussion of this topic until section 6.

The primary tool used in this analysis is the  $QR$  decomposition with full column pivoting [15]. We begin by reordering the rows and columns of  $G$

$$G(q, t) = \begin{bmatrix} G_{11}(q) & G_{12}(q) \\ G_{21}(q) & G_{22}(q) \end{bmatrix} \begin{bmatrix} q_1 \\ q_2 \end{bmatrix}, \quad (36)$$

where  $q^T = [q_1^T, q_2^T]$  has been partitioned into a set of unknown configuration variables  $q_1 : \mathbb{R} \rightarrow \mathbb{R}^{n_1}$  and a set of known, time-dependent variables,  $q_2 : \mathbb{R} \rightarrow \mathbb{R}^{n_2}$ ,  $n_1 + n_2 = n_q$ . The columns of  $G$  have been partitioned accordingly. Similarly the rows of  $G$  have been ordered such that the

rows associated with the homogeneous constraint equations appear first and the rows associated with motion drivers applied to joints in the cut-set appear below them. The velocity constraints<sup>5</sup> then take the form

$$\begin{bmatrix} G_{11}(q) & G_{12}(q) \\ G_{21}(q) & G_{22}(q) \end{bmatrix} \begin{bmatrix} q_1 \\ q_2 \end{bmatrix} = \begin{bmatrix} 0 \\ -\frac{\partial g_{21}}{\partial t} \end{bmatrix} . \quad (37)$$

Redundancy and consistency are detected before the simulation is started. The first task is to detect potential inconsistencies in the motion drivers applied to the joints in the spanning tree. To this end, we determine the effective rank of the following submatrix

$$QR = \begin{bmatrix} G_{11} \\ G_{21} \end{bmatrix} E , \quad (38)$$

where  $E$  is a permutation matrix determined by the pivoting strategy in the  $QR$  decomposition. Using the technique discussed in section 3.1, we determine the numerical rank of this matrix. To determine potential inconsistencies in the motion drivers, and to provide feedback to identify which drivers cause the difficulty, the columns of the submatrix  $[G_{12}^T \ G_{22}^T]^T$  are appended, one by one, and the  $QR$  decomposition recomputed. After  $k$  such stages, we obtain the following decomposition

$$QR = \begin{bmatrix} G_{11} & G_{12} \\ G_{21} & G_{22} \end{bmatrix} \begin{bmatrix} I & 0 \\ 0 & F_k \end{bmatrix} E . \quad (39)$$

Here  $F_k$  is the matrix that selects the first  $k$  columns of  $[G_{12}^T \ G_{22}^T]^T$ . If, in this process, the rank of the augmented matrix increases, we have identified (1) that a potential inconsistency exists, and (2) the driver causing the inconsistency. Increasing the rank of the augmented matrix, and the fact that the elements of  $q_2$  are arbitrary quantities determined by the user, implies that a solution to the velocity constraint equation may not exist. Of course it is possible for the user to define the motions of the drivers in such a way that they are consistent. It is impossible to determine whether the user will do this, and consequently an error is reported if inconsistencies are detected.

Having verified that the motion drivers in the spanning tree are consistent, we move to the task of eliminating redundancies among the homogeneous constraints. To achieve this we compute the  $QR$  decomposition of the submatrix

$$QR = G_{11}^T E . \quad (40)$$

This allows us to identify and eliminate the dependent rows. Finally we append the constraints generated by motion drivers applied to joints in the cut-set, one by one. After  $k$  stages, we obtain the following decomposition

$$QR = [ G_{11}^T \ G_{21}^T F_k^T ] E , \quad (41)$$

where  $F_k$  selects out the first  $k$  rows of  $G_{21}$ . If the rank of the augmented matrix increases, we have again identified a possible inconsistency between the motion drivers and the source of the

---

<sup>5</sup>For simplicity, we ignore nonholonomic constraints here, but they are easily included in the analysis.

inconsistency.

Typically mechanical systems have few explicit motion drivers, and this computation need only be performed once (except when discrete topology changes occur due to stiction). One difficulty with this approach arises when a simulation is started from a singular configuration. In this case, the analysis eliminates a constraint or constraints that are instantaneously redundant. Consequently the system likely violates the constraint as the simulation progresses. SimMechanics detects the constraint violation, but at present does not automatically detect whether the simulation is being started from such a singular configuration.

### 3.7 Consistent initial conditions

The reduction of a DAE to an ODE requires that we compute initial conditions that are consistent with the position and velocity constraints:

$$\begin{aligned} g(q, t) &= 0 \\ G(q, t)v + \frac{\partial g}{\partial t} &= 0 \quad . \end{aligned}$$

Suitable regularity assumptions ensure existence and uniqueness of solutions to the ODE but only a subset of these will satisfy the constraints. The problem of generating consistent initial conditions is different from, and more difficult than, the projection problem discussed in section 3.3. In the projection case, we typically have a good initial estimate of the solution  $(q, v)$  making the nonlinear position constraint much easier to solve. We do not have such an initial estimate to determine consistent initial conditions.

The definition of a mechanical model in SimMechanics requires that specific geometric relationships between bodies connected by joints be adhered to. For example, two bodies connected by a revolute joint must be attached to the joint at coincident points. This restriction implies that the position and velocity constraints are automatically satisfied for systems that have acyclic graphs. This is also true of the spanning tree of a system that has a cyclic graph. There are exceptions to this rule. SimMechanics implements a class of disassembled joints, which do not have to satisfy the geometric relationships, and SimMechanics allows users to impose finite initial joint displacements on the system.

Disassembled joints can only be used in cyclic systems and are automatically placed in the cut set. For these joints, the constraint equations must be satisfied. Even when the system is defined in an assembled configuration, an arbitrary finite displacement of a joint DoF will result in constraint violation. The projection solvers in SimMechanics are based on a modified Gauss-Newton solver with a *trust region* approach to improve robustness [8]. For the finite displacement problem at assembly, we use a homotopy (continuation) solver to achieve a better level of robustness. The problem is embedded in a one-parameter family of solutions using an artificial homotopy. The homotopy parameter  $\lambda$  is introduced to give the modified problem:

$$f(q, \lambda) = g(q, 0) - (1 - \lambda)g(q_0, 0) \quad . \quad (42)$$

Here  $q_0 \in \mathbb{R}^{n_q}$  is the initial guess. Homotopy techniques exploit the contractive properties of the manifold,

$$f(q, \lambda) = 0 \quad , \quad (43)$$

to obtain efficient numerical procedures that evolve the solution from  $q_0$  at  $\lambda = 0$  to the desired value at  $\lambda = 1$  [1]. The reader should refer to [1] for details, including the almost-everywhere convergence behavior of these solvers.

The velocity constraints are linear in the velocity variable and can be solved directly once the position constraints have been satisfied. All finite displacements introduced at the start of simulation are tested for consistency using the methods of section 3.6 before attempting assembly.

## 4 Linearization

A large class of dynamical systems can be satisfactorily controlled using the highly developed tools of linear control theory. To facilitate this it is necessary to obtain linear models of the system dynamics that accurately reflect the behavior of the system in the neighborhood of a specified nominal trajectory. For mechanical systems, we start with the reduced ODE form

$$\dot{q} = \tilde{H}(q)v \quad (44)$$

$$\dot{v} = \hat{F}(t, u, q, v) \quad , \quad (45)$$

where  $\hat{F} : \mathbb{R} \times \mathbb{R}^{n_u} \times \mathbb{R}^{n_q} \times \mathbb{R}^{n_v} \rightarrow \mathbb{R}^{n_v}$  is given by:

$$\hat{F} = M^{-1} \left[ f + \tilde{H}^T G^T (G \tilde{H} M^{-1} \tilde{H}^T G^T)^{-1} (-G \tilde{H} M^{-1} f - \frac{\partial(G \tilde{H} v)}{\partial q} \tilde{H} v - 2 \frac{\partial G}{\partial t} \tilde{H} v - \frac{\partial^2 g}{\partial t^2}) \right] \quad . \quad (46)$$

We have included the dependence of the dynamics on exogenous inputs (external forces and motor torques) through the vector  $u : \mathbb{R} \rightarrow \mathbb{R}^{n_u}$ . Linearization about a nominal trajectory  $(\bar{u}, \bar{q}, \bar{v})$  proceeds by introducing perturbations  $(\delta u, \delta q, \delta v)$  and expanding the equations about the nominal trajectory to get:

$$\dot{\bar{v}} + \delta \dot{v} = \hat{F}(t, \bar{u}, \bar{q}, \bar{v}) + \frac{\partial \hat{F}}{\partial u}(t, \bar{u}, \bar{q}, \bar{v}) \delta u + \frac{\partial \hat{F}}{\partial q}(t, \bar{u}, \bar{q}, \bar{v}) \delta q + \frac{\partial \hat{F}}{\partial v}(t, \bar{u}, \bar{q}, \bar{v}) \delta v + O(\|\delta u\|^2, \|\delta q\|^2, \|\delta v\|^2) \quad .$$

To first order, this gives a time-varying representation of the perturbative dynamics:

$$\delta \dot{v} = \frac{\partial \hat{F}}{\partial u}(t, \bar{u}, \bar{q}, \bar{v}) \delta u + \frac{\partial \hat{F}}{\partial q}(t, \bar{u}, \bar{q}, \bar{v}) \delta q + \frac{\partial \hat{F}}{\partial v}(t, \bar{u}, \bar{q}, \bar{v}) \delta v \quad .$$

Simulink can numerically linearize dynamic systems to generate a linear representation. Recently, Simulink was extended to incorporate the analytic derivatives, where these exist, either determined internally or provided by the user, of blocks or subsystems by using linear fractional transformations. This is important since, without this extension it would not be possible to generate linear models that accurately reflect the dynamics of cyclic systems. Very simply,

any perturbation (numerical or otherwise) introduced to determine the linear dynamics, must also satisfy the constraints to the same order of accuracy as the linearized dynamics. Since Simulink is essentially a general purpose solver, the perturbation is introduced to linearize the reduced differential equations of motion without knowledge of the constraint manifold. Since cyclic systems have constraints, only a subset of the perturbed motions satisfy equation 44 and equation 45 to first order and also satisfy the constraints to the same order.

Acyclic systems can be linearized in the standard manner, so we concentrate on the cyclic case. Here a linearization of the mechanical subsystem is provided to Simulink by SimMechanics. Simulink can then linearize a system containing other blocks and subsystems that are connected to the mechanical subsystem. Since the linearization of an interconnection can be obtained by the interconnection of the linearizations, Simulink can construct the linearized model for the whole system (mechanical and other) very efficiently. The task then falls on SimMechanics to register a linearization method with Simulink and to construct the desired linearization of the mechanical components. For cyclic systems this means linearizing the ODE on the constraint manifold. The perturbation must therefore satisfy

$$g(\bar{q}, t) + G(\bar{q}, t)\delta q = 0 \quad (47)$$

$$G(\bar{q}, t)\delta\dot{q} + G(\bar{q} + \delta q, t)\dot{\bar{q}} + \frac{\partial g}{\partial t} = 0 \quad (48)$$

To introduce perturbations that satisfy the constraints, the coordinates are partitioned into dependent and independent sets. The configuration variables are partitioned as  $q^T = [q_d^T, q_i^T]$  where  $q_d : \mathbb{R} \rightarrow \mathbb{R}^{n_c}$  is the set of dependent variables and  $q_i : \mathbb{R} \rightarrow \mathbb{R}^{n_q - n_c}$  is the set of independent variables. A similar partition is introduced for the velocity variables  $v = [v_d^T, v_i^T]$  where  $v_d : \mathbb{R} \rightarrow \mathbb{R}^{n_c}$  and  $v_i : \mathbb{R} \rightarrow \mathbb{R}^{n_v - n_c}$ . The partitioning scheme is implemented using a  $QR$  decomposition similar to that already discussed, except that we allow users to control the selection of variables to be placed in the independent set. The consistency and redundancy analysis is performed before the model is linearized and allows the choice of independent variables to be checked. If multiple linearizations are sought over a nominal trajectory, SimMechanics retains the initial choice of independent variables to prevent undesired discontinuities in the state matrices.

The linearization is still implemented numerically. The independent variables are perturbed, and the corresponding perturbations in the dependent variables determined to satisfy equation 47 and equation 48. The dynamics are then evaluated to generate a column in the linear matrices. The resulting linearization satisfies the constraint equations to first order. But it is not minimal because the linearized model and the original nonlinear model are assumed to have the same number of states (a standard assumption at present in Simulink and for ODE systems in general). For cyclic systems, the number of independent states  $n_i < n_q$  suffices to determine the linear dynamics. The non-minimal states are easily identified, as they are not reachable from the inputs or detectable from the outputs, and can be eliminated by the user.

Numerical linearization in SimMechanics allows users to specify a tolerance for the relative perturbation of the components in the state vector. SimMechanics also supports an adaptive

perturbation procedure. This procedure uses the initial perturbations defined by the user and attempts to obtain a good tradeoff between truncation error and rounding error. Simulink then uses the linearized model of the mechanical components to obtain the linearization for the system as a whole.

## 5 Trimming and equilibrium determination

Trimming of dynamic systems is an important part of control design. Usually control laws are designed to control systems about equilibrium configurations. However, trimming, in the more general sense, is an important part of many types of engineering analysis. Simulink allows general dynamic systems to be trimmed through the MATLAB<sup>®</sup> interface.

As with linearization, trimming acyclic systems poses no additional difficulties since the equations of motion are ODEs. Cyclic systems introduce difficulties because it is necessary to account for the constraints. The problem of trimming a cyclic system can be posed as follows. Find the states of the joints in the spanning tree to (1) satisfy the trim targets, and (2) satisfy the constraint equations. SimMechanics has a trimming mode where the position and velocity residuals are exported as outputs from the Simulink block diagram. In this mode, the user is able to set up very general trimming problems with the additional requirement that the position and velocity constraint residuals be driven to zero. This allows any mechanical system modeled in SimMechanics to be trimmed using the MATLAB interface.

An alternative approach to trimming cyclic systems is through inverse dynamics. Here unknown states in joints lying in the spanning tree can be driven with explicit motion drivers. The inputs to these motion drivers become the unknown variables in the trim problem. SimMechanics is able to compute the generalized forces that are consistent with the imposed motions. These generalized forces become the outputs of the trim problem. A trim condition can be found by setting to zero the nonlinear equations that map the inputs (to the motion drivers) to the outputs (excess generalized forces<sup>6</sup>). The advantage to using this approach is that the user no longer has to worry about the constraints, which are imposed internally. Inverse dynamics trimming does require SimMechanics to solve the kinematics of the cyclic system each time the function is evaluated. This can be less accurate and somewhat slower for certain types of systems. It is left to users to decide which approach is best for their applications.

## 6 Handling events

Mechanical systems can undergo discrete topological changes when joint DoFs lock up when subjected to static friction (stiction). SimMechanics allows users to model the topology changes that come about from stiction in joints using special actuators that can be connected to a subset of the joints. Discrete topology changes result in interesting numerical issues that must be addressed by further modifications to Simulink.

---

<sup>6</sup>By excess, we mean the generalized forces over and above any generalized forces that might be applied to the specific joint DoF through physical actuators.

Simulink already has the ability to detect events through discontinuity locking and switching functions. In Simulink, the continuous portions of the vector field are called modes and modes are only changed if and when a switching function becomes zero. An interesting feature of mechanical systems is the possibility of multiple mode changes at an instant of time. A joint locking can easily result in another joint surpassing a friction limit and consequently unlocking. In general, it is not possible to determine beforehand the mode transition for the mechanical system as a whole when a switching function crosses zero. This problem can be addressed using complementarity theory but would require a significant change to the current architecture of Simulink to implement. The approach used in SimMechanics is to introduce a mode iteration into Simulink. In this approach the predicted mode change is introduced and the system analyzed to determine whether the mode change is consistent with motion of, and forces applied to, the system. If the mode is not correct, an iteration is implemented to find the mode that is consistent with the motion and forces applied. In this way, SimMechanics is able to detect multiple events at a single time instant and to select the correct mode for the whole system before restarting the integration.

An additional difficulty with discrete topology changes occurs in cyclic systems. Here it is possible for joint locking, or unlocking, to cause inconsistencies between applied motion drivers or to introduce/eliminate redundancies among constraint equations. To ensure motion consistency and to determine the active constraint set, the redundancy analysis of section 3.6 must be repeated when an event is detected. To make the analysis more efficient, we bypass sequential addition of the motion drivers to determine the source of any inconsistencies. Instead, during simulation we simply add all of the motion drivers and error out if any inconsistencies are detected following a topology change.

Simulating mechanical systems that undergo discrete topology changes in real time is a difficult but important problem. Hardware in the loop (HIL) simulation of drive-train components is an important part of the rapid prototyping environment used in the design of automatic control systems for vehicles. These systems exhibit discrete topology changes when clutches are locked and unlocked. For HIL simulation, these topology changes must be modeled in real time. SimMechanics generates code for mechanical systems that exhibit this type of behavior with one exception. The mode iteration alluded to earlier is disabled in real-time simulation. The reasons are twofold: firstly, accurate event location is difficult to implement in real-time simulation; and secondly, the mode iteration process does not terminate in a deterministic number of floating point operations. For real-time systems, the mode iteration is instead carried out over multiple time-steps.

The present built-in event handling capabilities of Simulink further allow users to model a restricted class of contact and impact problems. The penalty method provides an approach for dealing with contact between bodies with simple geometries. Joint limits, for example, can be modeled using the hit-crossing block to detect impact between bodies with known geometrical shape. A force penetration law can be used to impose the unilateral contact constraint. This can be implemented using standard Simulink blocks and the stiff solver capabilities of codes like

ode15s.

## 7 Conclusions

In this paper we have addressed some of the interesting issues that arise when introducing mechanical simulation capability into an existing ODE-based simulation environment. We addressed the issue of computational efficiency by demonstrating how a relative coordinate formulation can be implemented using recursive computations to produce efficient  $O(n)$  algorithms. The problems of Lagrange multiplier determination and preventing numerical drift were also addressed. A number of strategies were outlined. Each has different properties which make it applicable in different situations.

Issues associated with redundant constraints and consistent motion drivers were discussed briefly. A strategy based on  $QR$  decomposition was put forward as an efficient and reliable way to eliminate redundant constraints and detect consistency problems. The important topic of linearization on a manifold and how this is currently achieved in Simulink was also examined. Finally, we mentioned some of the issues that arise when simulating mechanical systems that undergo discrete topology changes and the implications for real-time simulation.

## References

- [1] E. Allgower, K. Georg, *Numerical Continuation Methods: An Introduction*, Springer-Verlag, 1990
- [2] W. W. Armstrong, *Recursive Solution to the Equations of Motion of an  $n$ -link Manipulator*, Proceedings of the 5th World Congress on Theory of Machines and Mechanisms, Vol. 2, 1979, pp. 1343-1346
- [3] U. Ascher, H. Chin, L. Petzold, S. Reich, *Stabilization of Constrained Mechanical Systems with DAEs and Invariant Manifolds*, J. Mech. Struct. Machines, 23, 1993, pp. 135-158
- [4] U. Ascher, L. Petzold, *Stability of Computational Methods for Constrained Dynamics Systems*, SIAM J. SISI, 14, 1993, pp. 95-120
- [5] U. Ascher, H. Chin, S. Reich, *Stabilization of DAEs and invariant manifolds*, Numer. Math., 67, 1994, pp. 131-149
- [6] A. E. Bryson, Y. Ho, *Applied Optimal Control*, Blaisdell Publishing, 1969
- [7] B. Cloutier, D. Pai, U. M. Ascher, *The Formulation Stiffness of Forward Dynamics Algorithms and Implications for Robot Simulation*, Proceedings of the IEEE Conference on Robotics and Automation, 1995
- [8] J. Dennis, R. Schnabel, *Numerical Methods for Unconstrained Optimization and Nonlinear Equations*, SIAM, 1996

- [9] I. Duff, A. Erisman, J. Reid, *Direct Methods for Sparse Matrices*, Clarendon Press, Oxford, 1986
- [10] R. Featherstone, *The Calculation of Robot Dynamics Using Articulated-Body Inertias*, International Journal of Robotics Research, 2(1), 1983, pp. 13-30
- [11] R. Featherstone, *Robot Dynamics Algorithms*, Kluwer Academic Publishers, 1987
- [12] G. Golub, C. Van Loan, *Matrix Computations*, Johns Hopkins University Press, 1996
- [13] E. J. Haug, *Computer Aided Kinematics and Dynamics of Mechanical Systems, Volume 1, Basic Methods*, Allyn and Bacon, 1989
- [14] E. Haug, R. Deyo, *Real-Time Integration Methods for Mechanical System Simulation*, NATO ASI Series, Springer, 1991
- [15] N. Higham, *Accuracy and Stability of Numerical Algorithms*, SIAM, 1996
- [16] T. R. Kane, D. A. Levinson, *Multibody Dynamics*, Journal of Applied Mechanics, 50, 1983, pp. 1071-1078
- [17] A. A. Maciejewski, C. A. Klein, *Numerical Filtering for the Operation of Robotic Manipulators through Kinematically Singular Configurations*, Journal of Robotic Systems, 5(6), 1988, pp. 527-552
- [18] D. K. Pai, U. M. Ascher, Paul G. Kry, *Forward Dynamic Algorithms for Multibody Chains and Contact*, Proceedings of the IEEE Conference on Robotics and Automation, 2000
- [19] T. Park, E. Haug, *A Hybrid Constraint Stabilization Generalized Coordinate Partitioning Method for Machine Dynamic Simulation*, J. Mech. Trans. Auto. Des., 108 (2), 1986, pp. 211-216
- [20] R. E. Roberson, R. Schwertassek, *Dynamics of Multibody Systems*, Springer-Verlag, 1988
- [21] G. Rodriguez, *Kalman Filtering, Smoothing and Recursive Robot Arm Forward and Inverse Dynamics*, IEEE Journal of Robotics and Automation, 3(6), 1987, pp. 624-639
- [22] G. Rodriguez, A. Jain, K. Kreutz-Delgado, *A Spatial Operator Algebra for Manipulator Modeling and Control*, The International Journal of Robotics Research, 10(4), 1991, pp. 371-381
- [23] W. O. Schiehlen, *Multibody System Handbook*, Springer-Verlag, 1990
- [24] A. A. Shabana, *Dynamics of Multibody Systems*, Wiley, 1989
- [25] Lawrence F. Shampine, Mark W. Reichelt, *The MATLAB ODE Suite*, SIAM J. Sci. Comp., 18, 1997, pp. 1-22
- [26] Lawrence F. Shampine, Mark W. Reichelt, Jacek A. Kierzenka, *Solving index-1 DAEs in MATLAB and Simulink*, SIAM Review, 41, 1999, pp. 538-552

- 
- [27] L. F. Shampine, *Conservation Laws and the Numerical Solution of ODEs II*, Computers and Mathematics with Applications, 38, 1999, pp. 61-72
- [28] A. F. Vereshchagin, *Computer Simulation of the Dynamics of Complicated Mechanisms of Robot-Manipulators*, Engineering Cybernetics, 6, 1974, pp. 65-70
- [29] R. von Schwerin, *Multibody System Simulation*, Springer-Verlag, 1999
- [30] M. W. Walker, D. E. Orin, *Efficient Dynamic Computer Simulation of Robotic Mechanisms*, Journal of Dynamic Systems, Measurement, and Control, 104, 1982, pp. 205-211
- [31] C. W. Wampler, *Manipulator Inverse Kinematics Solutions Based on Vector Formulations and Damped Least Square Methods*, IEEE Transactions on Systems, Man, and Cybernetics, SMC 16, 1986, pp. 93-101
- [32] R. Wehage, E. Haug, *Generalized Coordinates Partitioning for Dimension Reduction in Analysis of Constrained Dynamic Systems*, J. Mech. Des., 104, 1982, pp. 247-255

This document was prepared with MiKTeX and Adobe Acrobat®.