Symbolic Lagrangian Multibody Dynamics

Brian Guenter*
Microsoft Research

Sung-Hee Lee†
Honda Research Institute

Abstract

Symbolic Lagrangian formulations of the equations of motion of tree structured constrained mechanical systems have the potential to be both more efficient and more numerically robust than formulations which use nonlinear kinematic constraint equations. We derive a simple recursive factorization of the Lagrangian equations of motion which, along with our extended implementation of the D* symbolic differentiation algorithm, yields empirically measured $O(n)$ inverse dynamics and $O(n^3)$ forward dynamics. Complex kinematic constraints such as point on surface, point on curve, and surface on surface are easily handled by the new algorithm. Numerous examples demonstrate the wide range of mechanical systems the algorithm can be applied to.

CR Categories: I.3.7 [COMPUTER GRAPHICS]: Three-Dimensional Graphics and Realism—Animation

1 Introduction

Lagrangian mechanics is a reformulation of classical mechanics which is particularly useful for modeling constrained mechanical systems. For tree structured systems, with holonomic constraints that can be expressed as differentiable parametric functions of the generalized coordinates[1], it is always possible to find a set of symbolic differential equations for the Lagrangian equations of motion. By contrast, non-Lagrangian formulations require differential equations *and* nonlinear algebraic equations (abbreviated DAE) [Ascher and Petzold 1998] to enforce kinematic constraints. Solving DAE systems is considerably more difficult than solving differential equations alone.

However, the difficulty of computing the complex derivatives in the Euler-Lagrange equations[2] have prevented wide acceptance of the approach [Herbert Goldstein 2001]. To overcome this difficulty, researchers have manually derived efficient differentiation methods of the Euler-Lagrange equations (see [Featherstone and Orin 2000] for an overview), but most algorithms assume a small set of simple constraint types such as revolute or prismatic joints and thus, not surprisingly, most of the current dynamics packages support only simple types of constraints.[3]Notably, an improved version of the Articulated Body Algorithm [Featherstone 1987] supports general scleronomic constraints, but the complexity of the algorithm makes it difficult to understand and implement [Baraff 1996]. If arbitrary parametric kinematic constraints are allowed then computing the derivatives is still problematic.

Computer graphics researchers have used symbolic or automatic differentiation to compute derivatives of complex functions [Kass 1992; Cohen 1992; Popović et al. 2000; Grinspun et al. 2003]. In this paper, we create symbolic equations of motion by using symbolic differentiation. Specifically, we derive a simple recursive factorization of the Lagrangian equations of motion which, along with our extended implementation of the D* symbolic differentiation algorithm [Guenter 2007], yields empirically measured $O(n)$ inverse dynamics and $O(n^3)$ forward dynamics. (**Note to reviewers**: D* symbolic differentiation software will be available to public in late 2009.)

In mechanics, symbolic or automatic differentiation have been employed mostly to augment multibody dynamics such as computing the derivatives of constraint equations or the mass matrix [Shi and McPhee 2000; Olsson et al. 2005; Samin and Fisette 2003]. Not only is symbolic differentiation used for dynamics simulation, but it is also used for linearizing the equations of motion and optimizing design parameters. However, for the core simulation of the dynamics, they employed existing, manually derived dynamics algorithms that do not require symbolic differentiation.

In contrast, we derive symbolic equations of motion by using symbolic differentiation. [Villard and Arnaldi 1996] applied symbolic differentiation to compute the equations for the constraint forces, which are numerically calculated using either the penalty method or the Lagrange mutltiplier tech-

*email:bguenter@microsoft.com
†email:slee@honda-ri.com

[1]This may sound restrictive but a wide variety of mechanisms can be modeled this way, as we will show in Section 5.

[2]For tree structured systems naïve direct approaches for evaluating these derivatives can have computational complexity as high as $O(n^4)$, where n is the number of degrees of freedom of the system.

[3]SD/FAST [Hollars et al. 1994], a multibody dynamics package based on Lagrangian formulation, supports complex, user-defined constraints only by using the Lagrange multipler technique, which is not related to our symbolic Lagrangian approach.

nique. By contrast, our algorithm generates minimal coordinate motion equations that eliminate all constraint forces. [Turner 2003] applied their automatic differentiation algorithm to Lagrange's equation to generate motion equations for very simple mechanisms, but they did not generalize their method to handle a broad scope of mechanisms.

One advantage of using symbolic differentiation for dynamics simulation is that one can easily incorporate complex constraints. In general, one needs to compute derivatives of the constraint equations up to the second order. Complex constraints have very complex derivatives, which are difficult to derive. By employing symbolic differentiation, one need only provide the constraint equations; the derivatives are computed by the symbolic differentation. Section 5 demonstrates that some constraints whose derivatives are very complex to derive manually can be easily simulated by our method. Some of our example mechanical systems are similar to those presented in [Kry and Pai 2003] and [Lee and Terzopoulos 2008] but the dynamics algorithms to create the examples are very different in that the former manually derived derivatives of the general surface constraints and the latter employed easily differentiable constraint functions, both under the framework of existing dynamics algorithms.

1.1 Contributions

To address the problem of applying symbolic Lagrangian mechanics to tree structured systems we present in this paper a simple and efficient factorization of the symbolic Lagrangian equations of motion which directly leads to an $O(n)$ algorithm for inverse dynamics. For the forward dynamics problem we have extended our implementation of the D*symbolic differentiation algorithm [Guenter 2007] to give an $O(n^3)$ algorithm. The new algorithm has several attractive properties:

- Because the symbolic Lagrangian equations always maintain kinematic constraints it is possible, in some cases, to use very large time steps. These large time steps would cause convergence problems for nonlinear constraint solvers.

- Mechanisms incorporating complex constraints which are challenging to simulate using existing techniques, such as point on surface, point on curve, surface on surface, are easily and robustly simulated.

The remainder of this paper is organized as follows: after deriving the Lagrangian equations of motion from the Newton-Euler dynamics equation (Section 2), we present a simple recursive factorization of the Lagrangian equation (Section 3). Then we show inverse and forward dynamics methods (Section 4) followed by numerous examples (Section 5). Finally, we report the performance of our methods (Section 6) and conclude the paper (Section 7).

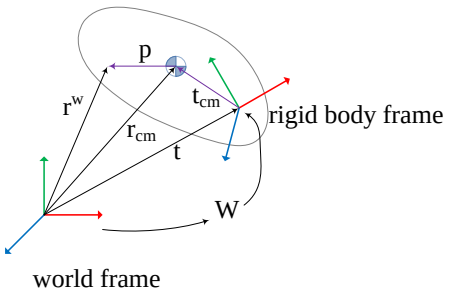


Figure 1: $\mathbf{r}_{cm}$ *and* $\mathbf{t}_{cm}$ *is the center of mass with respect to the world and body frame, respectively.* $\mathbf{r}^w$ *and* $\mathbf{p}$ *refer to a point of a rigid body with respect to the world and body frame, respectively.*

2 Lagrangian Equations of Motion

We start by deriving the Langrangian equations of motion of a multibody system. We assume a tree structured multibody system in which n generalized coordinates, $\mathbf{q} = (q_0, \ldots, q_{n-1})$, completely specify the configuration of every rigid body in the system. Each local transformation for rigid body i, $\mathbf{A}_i$, with respect to its parent is determined by some number of generalized coordinates $\mathbf{q}_i$, and its transformation $\mathbf{W}_i$ in world coordinates is expressed as the product of local transformations of its ancestors, or in recursive form, as:

$$\mathbf{W}_i(\mathbf{q}) = \begin{bmatrix} \mathbf{R}(\mathbf{q}_i) & \mathbf{t}(\mathbf{q}_i) \\ \mathbf{0} & 1 \end{bmatrix} = \mathbf{W}_{p_i}(\mathbf{q})\mathbf{A}_i(\mathbf{q}_i) \qquad (1)$$

where p_i denotes i's parent.

If the inertia matrix is defined in the center of mass coordinate frame then the Newton-Euler equations of motion for a system of n rigid bodies are:

$$\sum_{i=1}^{n} m_i \ddot{\mathbf{r}}_{cm_i} = \sum_{i=1}^{n} \mathbf{f}_i \qquad (2)$$

$$\sum_{i=1}^{n} \dot{\mathbf{h}}_{cm_i} = \sum_{i=1}^{n} \tau_i \qquad (3)$$

where m_i is the mass of a rigid body i and $\mathbf{r}_{cm_i} = \mathbf{W}_i \mathbf{t}_{cm_i}$ is its center of mass (Fig. 1). $\mathbf{h}_{cm_i} = \mathbf{I}_i \omega_i$ is the angular momentum about the center of mass where $\mathbf{I}_i$ is the rotational inertia and $\omega_i = \widetilde{(\mathbf{R}_i^T \dot{\mathbf{R}}_i)}$.[4] The force $\mathbf{f}_i$ and torque τ_i include external and internal forces and torques, respectively; i.e., $\mathbf{f}_i = \mathbf{f}_{ext_i} + \mathbf{f}_{int_i}$ and $\tau_i = \tau_{ext_i} + \tau_{int_i}$. By projecting the Newtonian motion equations onto the kinematic constraint manifold we obtain the Lagrangian equations of motion, which completely eliminate constraint forces ; i.e., we project (2) onto $\frac{\partial \mathbf{r}_{cm_i}}{\partial q_j}$. Constraint forces have zero projection on this vector so they all

[4] The ~ operator converts between vector and a skew-symmetric matrix representation; i.e., an angular velocity ω relates to the time derivative of the rotation matrix $\mathbf{R}$ as follows:

$$\tilde{\omega} = \begin{bmatrix} 0 & -\omega_z & \omega_y \\ \omega_z & 0 & -\omega_x \\ -\omega_y & \omega_x & 0 \end{bmatrix}, \widetilde{\mathbf{R}^T \dot{\mathbf{R}}} = \omega.$$

vanish per d'Alembert's principle of virtual work:

$$\begin{aligned}\sum_{i=1}^{n}\left(m_i\ddot{\mathbf{r}}_{cm_i}-\mathbf{f}_{ext_i}\right)^T\frac{\partial\mathbf{r}_{cm_i}}{\partial q_j} &= \sum_{i=1}^{n}\mathbf{f}_{int_i}^T\frac{\partial\mathbf{r}_{cm_i}}{\partial q_j}\\ \sum_{i=1}^{n}\mathbf{a}_i^T\frac{\partial\mathbf{r}_{cm_i}}{\partial q_j} &= Q_{\mathbf{f}_j}\end{aligned}\tag{4}$$

where $\mathbf{a}_i = m_i\ddot{\mathbf{r}}_{cm_i}-\mathbf{f}_{ext_i}$ and $Q_{\mathbf{f}_j}$ is the generalized force, due to forces, associated with generalized coordinate q_j. Likewise, we project the torques onto the constraint manifold:

$$\begin{aligned}\sum_{i=1}^{n}\left(\dot{\mathbf{h}}_{cm_i}-\tau_{ext_i}\right)^T\omega_{q_j} &= \sum_{i=1}^{n}\tau_{int_i}^T\omega_{q_j}\\ \sum_{i=1}^{n}\mathbf{b}_i^T\omega_{q_j} &= Q_{\tau_j}\end{aligned}\tag{5}$$

where $\omega_{q_j} = \widetilde{\left(\mathbf{R}_i^T\frac{\partial\mathbf{R}_i}{\partial q_j}\right)}$, $\mathbf{b}_i = \dot{\mathbf{h}}_{cm_i}-\tau_{ext_i}$, and Q_{τ_j} is the generalized force due to torques. We get a single set of differential equations by adding the two generalized forces.

$$\sum_{i=1}^{n}\left(\mathbf{a}_i^T\frac{\partial\mathbf{r}_{cm_i}}{\partial q_j}+\mathbf{b}_i^T\omega_{q_j}\right) = Q_j\tag{6}$$

where $Q_j = Q_{\mathbf{f}_j}+Q_{\tau_j}$. We have derived Lagrangain equations of motion (Eq. (6)) from Newton-Euler motion equations (Eqs. (2) and (3)) for a multibody system. However, this is not yet a computationally efficient form of the equations; the inverse dynamics equations generated by the D* algorithm using (6) is $O(n^2)$, n times more than today's manually optimized $O(n)$ inverse dynamics algorithms. In Section 3, we develop a novel factorization that yields an $O(n)$ inverse dynamics algorithms.

3 Factoring the Equations of Motion

The factorization of the motion equations is most easily approached by considering the force and torque equations separately. Substituting $\mathbf{r}_{cm_i} = \mathbf{W}_i\mathbf{t}_{cm_i}$ (Fig. 1) into (4) gives us this set of equations for $Q_{\mathbf{f}_j}$.

$$\sum_{i=1}^{n}\mathbf{a}_i^T\frac{\partial\mathbf{W}_i}{\partial q_j}\mathbf{t}_{cm_i} = Q_{\mathbf{f}_j}.$$

For $n=3$ we get this system of equations:

$$\begin{aligned}\mathbf{a}_0^T\frac{\partial\mathbf{W}_0}{\partial q_0}\mathbf{t}_{cm_0}+\mathbf{a}_1^T\frac{\partial\mathbf{W}_1}{\partial q_0}\mathbf{t}_{cm_1}+\mathbf{a}_2^T\frac{\partial\mathbf{W}_2}{\partial q_0}\mathbf{t}_{cm_2} &= Q_{\mathbf{f}_0}\\ \mathbf{a}_0^T\frac{\partial\mathbf{W}_0}{\partial q_1}\mathbf{t}_{cm_0}+\mathbf{a}_1^T\frac{\partial\mathbf{W}_1}{\partial q_1}\mathbf{t}_{cm_1}+\mathbf{a}_2^T\frac{\partial\mathbf{W}_2}{\partial q_1}\mathbf{t}_{cm_2} &= Q_{\mathbf{f}_1}\\ \mathbf{a}_0^T\frac{\partial\mathbf{W}_0}{\partial q_2}\mathbf{t}_{cm_0}+\mathbf{a}_1^T\frac{\partial\mathbf{W}_1}{\partial q_2}\mathbf{t}_{cm_1}+\mathbf{a}_2^T\frac{\partial\mathbf{W}_2}{\partial q_2}\mathbf{t}_{cm_2} &= Q_{\mathbf{f}_2}\end{aligned}$$

To simplify these equations start with two observations. First, because[5]

$$\frac{\partial\mathbf{A}_i}{\partial q_j} = 0 \qquad i\neq j\tag{7}$$

we can rewrite $\frac{\partial\mathbf{W}_i}{\partial q_j}$ as

$$\frac{\partial\mathbf{W}_i}{\partial q_j} = \mathbf{W}_{j-1}\frac{\partial\mathbf{A}_j}{\partial q_j}\mathbf{A}_{j+1}\ldots\mathbf{A}_i\tag{8}$$

Second

$$\frac{\partial\mathbf{W}_j}{\partial q_k} = 0 \text{ for } k\geq j$$

where the expression $k\geq j$ means that the rigid body k is further toward the leaves of the tree than the rigid body j. Apply these two observations to our equations of motion to get an upper triangular set of equations

$$\begin{aligned}\mathbf{a}_0^T\frac{\partial\mathbf{A}_0}{\partial q_0}\mathbf{t}_{cm_0}+\mathbf{a}_1^T\frac{\partial\mathbf{A}_0}{\partial q_0}\mathbf{A}_1\mathbf{t}_{cm_1}+\mathbf{a}_2^T\frac{\partial\mathbf{A}_0}{\partial q_0}\mathbf{A}_1\mathbf{A}_2\mathbf{t}_{cm_2} &= Q_{\mathbf{f}_0}\\ \mathbf{a}_1^T\mathbf{W}_0\frac{\partial\mathbf{A}_1}{\partial q_1}\mathbf{t}_{cm_1}+\mathbf{a}_2^T\mathbf{W}_0\frac{\partial\mathbf{A}_1}{\partial q_1}\mathbf{A}_2\mathbf{t}_{cm_2} &= Q_{\mathbf{f}_1}\\ \mathbf{a}_2^T\mathbf{W}_1\frac{\partial\mathbf{A}_2}{\partial q_2}\mathbf{t}_{cm_2} &= Q_{\mathbf{f}_2}\end{aligned}$$

Notice that $\frac{\partial\mathbf{A}_i}{\partial q_i}$ is included in every term in the LHS and we can factor out theses common terms by moving the $\mathbf{a}_i^T$ terms from the left to the right hand side. The products $\mathbf{a}_i^T\mathbf{W}_{i-1}\frac{\partial\mathbf{A}_i}{\partial q_j}\mathbf{t}_{cm_i}$ are of the form

$$\mathbf{x}^T\mathbf{y} \quad \text{where } \mathbf{x}^T = \mathbf{a}_i^T \text{ and } \mathbf{y} = \mathbf{W}_{i-1}\frac{\partial\mathbf{A}_i}{\partial q_j}\mathbf{t}_{cm_i}.$$

Using the identity

$$tr\left(\mathbf{y}\mathbf{x}^T\right) = \mathbf{x}^T\mathbf{y},\tag{9}$$

our three equations of motion become

$$\begin{aligned}tr\left(\frac{\partial\mathbf{A}_0}{\partial q_0}\mathbf{t}_{cm_0}\mathbf{a}_0^T+\frac{\partial\mathbf{A}_0}{\partial q_0}\mathbf{A}_1\mathbf{t}_{cm_1}\mathbf{a}_1^T+\frac{\partial\mathbf{A}_0}{\partial q_0}\mathbf{A}_1\mathbf{A}_2\mathbf{t}_{cm_2}\mathbf{a}_2^T\right) &= Q_{\mathbf{f}_0}\\ tr\left(\mathbf{W}_0\frac{\partial\mathbf{A}_1}{\partial q_1}\mathbf{t}_{cm_1}\mathbf{a}_1^T+\mathbf{W}_0\frac{\partial\mathbf{A}_1}{\partial q_1}\mathbf{A}_2\mathbf{t}_{cm_2}\mathbf{a}_2^T\right) &= Q_{\mathbf{f}_1}\\ tr\left(\mathbf{W}_1\frac{\partial\mathbf{A}_2}{\partial q_2}\mathbf{t}_{cm_2}\mathbf{a}_2^T\right) &= Q_{\mathbf{f}_2}\end{aligned}$$

Factoring out common terms, we get

$$\begin{aligned}tr\left(\frac{\partial\mathbf{A}_0}{\partial q_0}\left(\mathbf{t}_{cm_0}\mathbf{a}_0^T+\mathbf{A}_1\left(\mathbf{t}_{cm_1}\mathbf{a}_1^T+\mathbf{A}_2\mathbf{t}_{cm_2}\mathbf{a}_2^T\right)\right)\right) &= Q_{\mathbf{f}_0}\\ tr\left(\mathbf{W}_0\frac{\partial\mathbf{A}_1}{\partial q_1}\left(\mathbf{t}_{cm_1}\mathbf{a}_1^T+\mathbf{A}_2\mathbf{t}_{cm_2}\mathbf{a}_2^T\right)\right) &= Q_{\mathbf{f}_1}\\ tr\left(\mathbf{W}_1\frac{\partial\mathbf{A}_2}{\partial q_2}\left(\mathbf{t}_{cm_2}\mathbf{a}_2^T\right)\right) &= Q_{\mathbf{f}_2}\end{aligned}$$

[5] We are ignoring a technical detail that would unnecessarily complicate the equations at this point: the matrix $\mathbf{A}_j$ may be a function of up to six generalized coordinates q_k so the indices of the matrix $\mathbf{A}_j$ and the generalized coordinate may not be the same. This does not materially change the form of the equations or the factorization.

Because of common subexpression elimination in **D*** it takes $O(n)$ time to compute the equations in this form.

Let us turn now to the torque equations (5):

$$Q_{\tau_j} = \sum_{i=1}^{n} \widetilde{\mathbf{R}_i^T \frac{\partial \mathbf{R}_i}{\partial q_j}}^T \mathbf{b}_i.$$

We will show how we can move the term $\partial \mathbf{R}_i/\partial q_j$ in front for effecient factorization. To this end, we introduce the following identity

$$\begin{aligned} tr\left(\tilde{\mathbf{a}}\tilde{\mathbf{b}}\right) &= tr\left\{ \begin{bmatrix} 0 & -a_z & a_y \\ a_z & 0 & -a_x \\ -a_y & a_x & 0 \end{bmatrix} \begin{bmatrix} 0 & -b_z & b_y \\ b_z & 0 & -b_x \\ -b_y & b_x & 0 \end{bmatrix} \right\} \\ &= -2\mathbf{a}^T\mathbf{b}. \end{aligned}$$

Using this relation, we can rewrite the $\omega_{q_j}^T \mathbf{b}_i$ terms

$$\begin{aligned} \omega_{q_j}^T \mathbf{b}_i &= -.5tr\left(\tilde{\omega}_{q_j}\tilde{\mathbf{b}}_i\right) \\ &= tr\left(-\frac{1}{2}\mathbf{R}_i^T\frac{\partial \mathbf{R}_i}{\partial q_j}\tilde{\mathbf{b}}_i\right) \\ &= tr\left(-\frac{1}{2}\frac{\partial \mathbf{R}_i}{\partial q_j}\tilde{\mathbf{b}}_i\mathbf{R}_i^T\right) \end{aligned} \quad (10)$$

where we used the relation $tr(\mathbf{AB}) = tr(\mathbf{BA})$ for (10). Now we can rewrite the torque equations with $\frac{\partial \mathbf{R}_i}{\partial q_j}$ on the left-hand side where everything will factor nicely:

$$Q_{\tau_j} = \sum_{i=1}^{n} \omega_{q_j}^T \mathbf{b}_i \quad (11)$$

$$= tr\left\{ \sum_{i=1}^{n} \frac{\partial \mathbf{R}_i}{\partial q_j}\left(-\frac{1}{2}\tilde{\mathbf{b}}_i\mathbf{R}_i^T\right)\right\} \quad (12)$$

Define two new matrices,

$$\mathbf{U}_i = \begin{bmatrix} \mathbf{R}_i & 0 \\ \mathbf{0} & 0 \end{bmatrix}, \mathbf{B}_i = \begin{bmatrix} \tilde{\mathbf{b}}_i & 0 \\ \mathbf{0} & 0 \end{bmatrix}$$

to rewrite the torque equations using the full transformation $\mathbf{W}_i$ instead of $\mathbf{R}_i$:

$$\frac{\partial \mathbf{R}_i}{\partial q_j}\left(-\frac{1}{2}\tilde{\mathbf{b}}_i\mathbf{R}_i^T\right) = \frac{\partial \mathbf{W}_i}{\partial q_j}\left(-\frac{1}{2}\mathbf{B}_i\mathbf{U}_i^T\right).$$

Using the same 3 degree of freedom system that we used for the force equations we get the following three torque equations

$$\begin{aligned} -\frac{1}{2}tr\left\{\frac{\partial \mathbf{W}_0}{\partial q_0}\left(\mathbf{B}_0\mathbf{U}_0^T\right) + \frac{\partial \mathbf{W}_1}{\partial q_0}\left(\mathbf{B}_1\mathbf{U}_1^T\right) + \frac{\partial \mathbf{W}_2}{\partial q_0}\left(\mathbf{B}_2\mathbf{U}_2^T\right)\right\} &= Q_{\tau_0} \\ -\frac{1}{2}tr\left\{\frac{\partial \mathbf{W}_0}{\partial q_1}\left(\mathbf{B}_0\mathbf{U}_0^T\right) + \frac{\partial \mathbf{W}_1}{\partial q_1}\left(\mathbf{B}_1\mathbf{U}_1^T\right) + \frac{\partial \mathbf{W}_2}{\partial q_1}\left(\mathbf{B}_2\mathbf{U}_2^T\right)\right\} &= Q_{\tau_1} \\ -\frac{1}{2}tr\left\{\frac{\partial \mathbf{W}_0}{\partial q_2}\left(\mathbf{B}_0\mathbf{U}_0^T\right) + \frac{\partial \mathbf{W}_1}{\partial q_2}\left(\mathbf{B}_1\mathbf{U}_1^T\right) + \frac{\partial \mathbf{W}_2}{\partial q_2}\left(\mathbf{B}_2\mathbf{U}_2^T\right)\right\} &= Q_{\tau_2} \end{aligned}$$

As in the force relation, we transform the equations into upper triangular form using (7) and (8), and factor out common terms

$$\begin{aligned} -\frac{1}{2}tr\left\{\frac{\partial \mathbf{A}_0}{\partial q_0}\left(\left(\mathbf{B}_0\mathbf{U}_0^T\right) + \mathbf{A}_1\left(\mathbf{B}_1\mathbf{U}_1^T + \mathbf{A}_2\left(\mathbf{B}_2\mathbf{U}_2^T\right)\right)\right)\right\} &= Q_{\tau_0} \\ -\frac{1}{2}tr\left\{\mathbf{W}_0\frac{\partial \mathbf{A}_1}{\partial q_1}\left(\mathbf{B}_1\mathbf{U}_1^T + \mathbf{A}_2\left(\mathbf{B}_2\mathbf{U}_2^T\right)\right)\right\} &= Q_{\tau_1} \\ -\frac{1}{2}tr\left\{\mathbf{W}_1\frac{\partial \mathbf{A}_2}{\partial q_2}\left(\mathbf{B}_2\mathbf{U}_2^T\right)\right\} &= Q_{\tau_2} \end{aligned}$$

Algorithm 1 coefficient algorithm

```
nd: node to find coefficient of
lc: true if left child graph of this contains nd
rc: true if right child graph of this contains nd
coeff(function nd)
if this is leaf return this
else
    foreach (node c in this.children){c.coeff(nd)}
    if (this.type not one of {+,-,*,/}) ERROR
    if (this is * && lc && rc) ERROR
    if (this is / && rc) ERROR
    if (!(this is + || this is -)) return this
    else if (lc && !rc){return this.leftchild}
    else if (rc && !lc){return this.rightchild}
    else return this
```

Because they share common factors we can combine the generalized force and torque equations to get an equation for each of the generalized forces Q_0, Q_1, Q_2 illustrated in Fig. 2. Finally, we get the following efficient form of the Lagrangain equations of motion:

$$tr\{\mathbf{W}_{j-1}\frac{\partial \mathbf{A}_j}{\partial q_j}\phi_j\} = Q_j \quad (13)$$

$$\phi_j = \mathbf{t}_{cm_j}\mathbf{a}_j^T - \frac{1}{2}\mathbf{B}_j\mathbf{U}_j^T + \mathbf{A}_{j+1}\phi_{j+1}$$

4 Forward and Inverse Dynamics

For the *inverse dynamics* problem, we can use the motion specification, $\mathbf{q}(t)$, and (13) to compute the generalized forces. Q_j is a function of $\mathbf{q}(t)$, $\dot{\mathbf{q}}(t)$, $\ddot{\mathbf{q}}(t)$, all of which are computed from the motion specification. Computing the generalized forces takes $O(n)$ time.

The general dynamics formula of (13) is not in the proper form to directly solve the *forward dynamics* problem. However, the equations of motion have the following structure:

$$\mathbf{M}(\mathbf{q})\ddot{\mathbf{q}} + \mathbf{c}(\mathbf{q}, \dot{\mathbf{q}}) = \mathbf{Q}. \quad (14)$$

We can see that M_{ij}, the (i, j) entry of $\mathbf{M}$ is the coefficient of $\ddot{q}_j$ of the i-th equation. Also, if we set $\ddot{\mathbf{q}} = \mathbf{0}$, then the LHS of (13) becomes $\mathbf{c}(\mathbf{q}, \dot{\mathbf{q}})$.

We derive a symbolic expression for $\mathbf{M}$ and $\mathbf{c}$ using **D*** . For this, we have added two new functions, `coefficient` and `substitute`, to our implementation of **D***. These make it trivial to reorganize these equations into an easily solved form.

Given a **D*** function, `fi`, which computes $f_i(\mathbf{q}, \dot{\mathbf{q}}, \ddot{\mathbf{q}})$ and a **D*** variable, `qiddt`, which corresponds to $\ddot{q}_i$, the function `fi.coefficient(qiddt)`returns the symbolic expression which represents the coefficient of $\ddot{q}_i$. Since $\mathbf{M}$ is a symmetric matrix, we can create a symbolic expression of $\mathbf{M}$ by calling this function for $\ddot{q}_j$ in each f_i for $i \geq j$. The $c_i(\mathbf{q}, \dot{\mathbf{q}})$ are computed by calling

$$
\begin{aligned}
tr\left\{\frac{\partial \mathbf{A}_0}{\partial q_0}\left(\mathbf{t}_{cm_0}\mathbf{a}_0^T - \frac{1}{2}\mathbf{B}_0\mathbf{U}_0^T + \mathbf{A}_1\left(\mathbf{t}_{cm_1}\mathbf{a}_1^T - \frac{1}{2}\mathbf{B}_1\mathbf{U}_1^T + \mathbf{A}_2\left(\mathbf{t}_{cm_2}\mathbf{a}_2^T - \frac{1}{2}\mathbf{B}_2\mathbf{U}_2^T\right)\right)\right)\right\} &= Q_0 \\
tr\left\{\mathbf{W}_0\frac{\partial \mathbf{A}_1}{\partial q_1}\left(\mathbf{t}_{cm_1}\mathbf{a}_1^T - \frac{1}{2}\mathbf{B}_1\mathbf{U}_1^T + \mathbf{A}_2\left(\mathbf{t}_{cm_2}\mathbf{a}_2^T - \frac{1}{2}\mathbf{B}_2\mathbf{U}_2^T\right)\right)\right\} &= Q_1 \\
tr\left\{\mathbf{W}_1\frac{\partial \mathbf{A}_2}{\partial q_2}\left(\mathbf{t}_{cm_2}\mathbf{a}_2^T - \frac{1}{2}\mathbf{B}_2\mathbf{U}_2^T\right)\right\} &= Q_2
\end{aligned}
$$

Figure 2: *Combined force and torque equations*

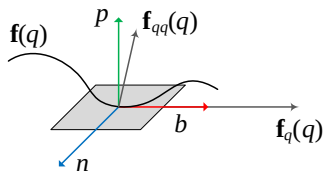


Figure 3: *Traveling coordinate frame defined by the tangent,* $\mathbf{f}_q(q)$*, and second derivative,* $\mathbf{f}_{qq}(q)$ *of the curve* $\mathbf{f}(q)$

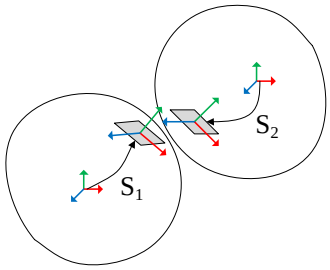


Figure 4: *Surface-surface coordinate frame constraints*

```
fi.substitute(new[]{q0ddt,...qnddt},
new[]{0,...,0})
```

This will substitute zero for all the $\ddot{q}_i$ leaving just $c_i(\mathbf{q},\dot{\mathbf{q}})$. Forward dynamics is performed by solving

$$\mathbf{M}\ddot{\mathbf{q}} = \mathbf{Q} - \mathbf{c}$$

using standard linear algebra algorithms. The Cholesky decomposition method is a good choice in this case since $\mathbf{M}$ is positive definite.

5 Examples of Complex Constraints

Systems are built by connecting rigid bodies with motion constraints, represented as a transformation matrix which is a function of up to 6 generalized coordinates. To create and simulate a mechanism the user has only to define these transformation matrices of rigid bodies with respect to their parents as functions of generalized coordinates. This is in contrast to conventional, numerical dynamics packages in which one must provide the derivatives of the transformations up to the second order. This can be very complicated as will be seen in some of our examples.[6]

The Frenet frame along a curve could be used, for example, to constrain beads to slide along a wire, or to make a roller coaster or train roll along a track. If the curve $\mathbf{f}(q)$ has a continuous, nonzero second derivative everywhere in a region, then it is easy to define a coordinate frame that travels along the curve

$$\mathbf{C}(q) = \begin{bmatrix} \mathbf{b} & \mathbf{p} & \mathbf{n} & \mathbf{f} \\ 0 & 0 & 0 & 1 \end{bmatrix},$$

where $\mathbf{b} = \frac{\mathbf{f}_q}{\|\mathbf{f}_q\|}$, $\mathbf{n} = \frac{\mathbf{b}\times\mathbf{f}_{qq}}{\|\mathbf{b}\times\mathbf{f}_{qq}\|}$, and $\mathbf{p} = \mathbf{n}\times\mathbf{b}$.

Assume we want to constrain an object to move along a space curve, $\mathbf{f}(q_p) : \mathbb{R}^1 \to \mathbb{R}^3$, so that the x axis in object model space always points in the direction of the tangent, $\mathbf{f}_{q_p}$, to the curve, and the object can rotate by q_r about the axis of the tangent vector. The combined constraint will use two transformation matrices: the curve orientation and translation constraint matrix, and a rotation about the x axis

$$\mathbf{A}_{path} = \mathbf{C}(q_p)\mathbf{R}_{rotX}(q_r).$$

An example of several systems constrained this way is shown in Fig. 5(a).

Moving up one dimension from curves we can define a surface frame whose origin is on the surface with two axes tangent to the surface. For the parametric surface

$$\mathbf{f}(q_1,q_2) = [f_x(q_1,q_2), f_y(q_1,q_2), f_z(q_1,q_2)]^T$$

we use the two tangent vectors of unit lengths

$$\mathbf{p}_0 = \frac{\partial \mathbf{f}}{\partial q_1}/\|\frac{\partial \mathbf{f}}{\partial q_1}\| \qquad \mathbf{p}_1 = \frac{\partial \mathbf{f}}{\partial q_2}/\|\frac{\partial \mathbf{f}}{\partial q_2}\|$$

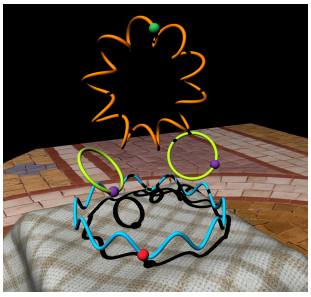
and the unit normal vector $\mathbf{n} = \mathbf{p}_0 \times \mathbf{p}_1$ to create a surface frame

$$\mathbf{S} = \begin{bmatrix} \mathbf{p}_0 & \mathbf{n}\times\mathbf{p}_0 & \mathbf{n} & \mathbf{f} \\ 0 & 0 & 0 & 1 \end{bmatrix}.$$

We can force the surface frames of two surfaces, shown in Fig. 4, to align by using this transformation

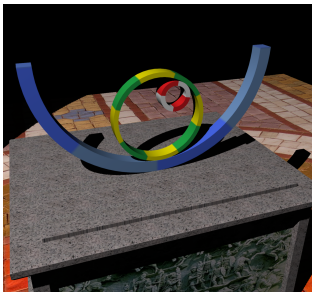
$$\mathbf{S}_{1,2} = \mathbf{S}_1\mathbf{S}_2^{-1} \tag{15}$$

[6]Note that some of the example constraints have singularity problems, but this is a separate issue from the main subject of this paper.



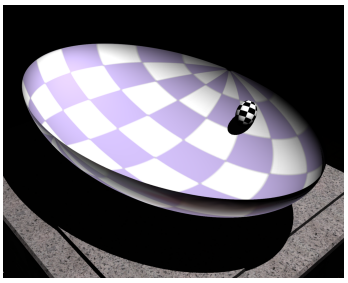
(a) *Path constraint*

(b) *Surface surface constraint*

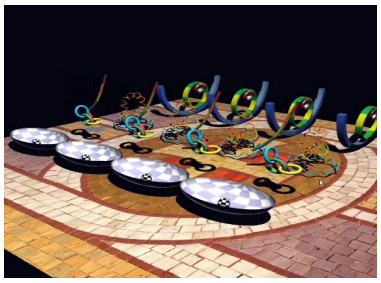


(c) *Rolling constraint*

(d) *Tong-like mechanism*



(e) *Ellipsoidal ball rolling over ellipsoidal surface*



(f) *Multiple systems*

Figure 5: *Example mechanisms*

where $\mathbf{S}_1$ and $\mathbf{S}_2$ define the two surface frames. The matrix $\mathbf{S}_{1,2}$ is a function of four generalized coordinates q_1, q_2, q_3, q_4. We can make a chain of torii, shown in Fig. 5(b), by applying (15) to the sequence of torus objects.

It is also possible to simulate certain types of rolling contact. In Fig. 5(c) the wheels are constrained to roll without sliding. Assume the wheels lie in the xy plane and rotate about the z axis. The relative transformation of the bigger wheel (colored yellow and green) can be parameterized by a single generalized coordinate q; the position of the body frame is $[\cos q, \sin q, 0]^T$, and, due to the non-slip constraint, the orientation is also defined as a function of q,

$$\mathbf{R} = \mathbf{R}_{rotZ}(\frac{r_1}{r_2}q)$$

where r_1 is the radius of the half circle and r_2 is the radius of the wheel. The relative transformation of the smallest wheel can be defined similarly.

Fig. 5(d) shows an example in which the generalized coordinates can be defined for a closed loop mechanism. A total of 32 rigid bodies is constrained such that each joint angle of the link can be defined as a function of a single generalized coordinate, i.e., $q_i = q_i(q)$ for all i. A "fish" object is connected to a tong-like chain via a universal joint. In this case, our algorithm creates an ODE with only three independent variables. In contrast, non-symbolic, conventional Lagrangian dynamics algorithms such as the Articulated-Body Algorithm [Featherstone 1987] cannot achieve this level of reduction for the closed loop system because they endow independent coordinates to each mobile rigid body followed by enforcing the loop constraints using Lagrange multipliers. For this example, they will obtain a 34 dimensional ODE coupled with 31 holonomic closed loop constraints. Newtonian dynamics algorithms will generate a 33×6 dimensional ODE for this type of mechanism.

Fig. 5(e) shows an ellipsoidal ball rolling over an ellipsoidal surface with the rolling constraint enforced using the Lagrange multipliers. To establish a rolling constraint between two surfaces let $(\mathbf{R}_1, \mathbf{t}_1)$ and $(\mathbf{R}_2, \mathbf{t}_2)$ denote the orientation and position of the surface frames of the two surfaces. When the two surfaces do not slip, the relative velocity of the two surface frames must be same, i.e., $\mathbf{R}_1^T \dot{\mathbf{t}}_1 = \mathbf{R}_2^T \dot{\mathbf{t}}_2$ which yields this constraint equation:

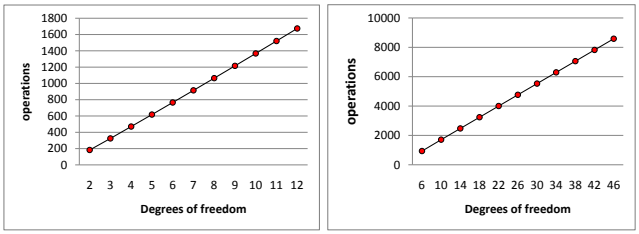
$$\mathbf{P}\dot{\mathbf{q}} = \mathbf{0},\ \mathbf{P} = \left[\mathbf{R}_1^{\mathrm{T}} \frac{\mathbf{dt_1}}{\mathbf{dq}} \quad -\mathbf{R}_2^{\mathrm{T}} \frac{\mathbf{dt_2}}{\mathbf{dq}}\right]. \tag{16}$$

Note that one must derive $\dot{\mathbf{P}}$, the time derivative of $\mathbf{P}$, in order to use Lagrange multipliers. **D*** computes $\dot{\mathbf{P}}$ as well as the derivatives in (16) so one need not worry about manually computing them.

6 Results

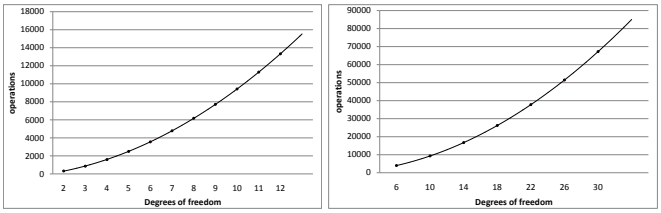
Because the symbolic Lagrangian algorithm performs algebraic simplification and common subexpression elimination the operations count is not just a function of the number of degrees of freedom in the system, as it is for previously published forward and inverse dynamics algorithms. We measured the time complexity of the new algorithm on two systems and present those results here.

Results for the inverse dynamics problem are shown in Fig. 6. Fig. 6(a) shows the total number of operations as a function of the number of degrees of freedom for a simple linear



(a) (b)

Figure 6: *Inverse dynamics, arithmetic operations versus number of generalized coordinates: (a) linear revolute chain (not illustrated). (b) torus chain (Fig. 5(b)). Growth is well approximated by a best fit line.*



(a) (b)

Figure 7: *Forward dynamics, arithmetic operations versus number of generalized coordinates: (a) linear revolute chain (not illustrated). (b) torus chain (Fig. 5(b)). Growth is well approximated by a best fit quadratic curve.*

chain of revolute joints. Fig. 6(b) shows the total number of operations for a linear chain of torii, of the type shown in Fig. 5(b). Both systems are well approximated by a best fit line.

For the forward dynamics problem Fig. 7 shows the measured number of operations in the equations of motion for the same systems of Fig. 6. The operation count is increasing quadratically in both cases. Because eventually the $O(n^3)$ matrix solution time will dominate the overall complexity of forward dynamics will be $O(n^3)$.

Existing forward and inverse dynamics algorithms are usually limited to a very restricted set of joint types, often just revolute or translational joints – other joint types require significant effort on the part of the end-user to implement. Both of these facts make direct comparison of our algorithm to existing algorithms problematic; the capabilities of the two sets of algorithms are very different so it is hard to make an apples to apples comparison.

We created a nonbranching system consisting of six 1 DOF revolute joints, connected in a linear chain, and used this as a comparison test case. Roughly best and worst case results for the symbolic Lagrangian algorithm are compared in Table 1 to the most efficient hand derived algorithms [Balafoutis and Patel 1991] that requires manual differentiation of the constraints. Our algorithm is competitive even in this restricted class, and, of course, it can easily be applied to a much wider range of joint types.

algorithm	±	*
Symb. best	328	365
Balafoutis	386	450
Symb. worst	509	971

(a) *Inverse dynamics*

algorithm	±	*
Symb. best	818	943
Balafoutis	790	956
Symb. worst	1875	2389

(b) *Forward dynamics*

Table 1: *Operations count comparison for a 6 degree of freedom chain of revolute joints: symbolic Lagrangian vs. Balafoutis.*

In practice the new algorithm generates efficient dynamics code. For example, Fig. 5(b), which has the largest number of generalized coordinates (14) of the examples shown in Fig. 5, requires only 0.02 miliseconds to evaluate the motion equation. Solving the matrix motion equation using Cholesky decomposition takes an additional 1.3×10^{-3} miliseconds on a 2.66Hz Intel Core 2 Quad CPU. Our code is not multithreaded so only a single processor is used for the motion equation computation. Comparing the compute time of the motion equation ($O(n^2)$) and that of the matrix solution ($O(n^3)$) suggests that the two procedures will take the same amount of time when the DOFs of the system are about 200. For models with $\ll 200$ DOFs, the algorithm is dominated by the evaluation of the $O(n^2)$ motion equation procedure.

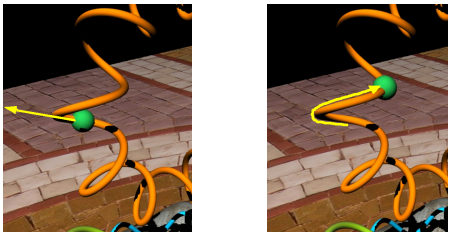
7 Discussion and Conclusion

Because the Lagrangian formulation always maintains kinematic constraints, it is possible to take very large time steps when precise modeling of the true dynamics is unnecessary. In computer games and animation for film or television it is rarely necessary to model dynamics to high precision, since human viewers don't notice small dynamic errors. By comparison, kinematic constraint errors are much more noticeable, since they result in parts of the mechanism failing to connect properly.

Fig. 8 shows two frames, (a), (b), from a dynamic simulation of a bead constrained to move along a wire: (a) shows a step along the tangent to the constraint manifold, as would be done when using nonlinear constraint equations, and (b) shows the step taken by our symbolic Lagriangian formulation. The tangent step moves the bead far off the constraint manifold; convergence of a nonlinear constraint solver to the correct point on the constraint manifold is highly unlikely. To guarantee convergence a much smaller time step would be necessary to correctly maintain the kinematic constraints.

We have taken advantage of this robustness property of symbolic Lagrangian dynamics in all the animations, included in the supplemental materials for this paper, which have a timestep of one frame time (either $\frac{1}{60}$th or $\frac{1}{30}$th second).

We do not deal with simulating contact and collision in this paper since it is a separate problem that is resolved after computing the dynamics of a system. Any collision and contact



(a) (b)

Figure 8: *(a) Yellow arrow shows step along tangent to constraint manifold. (b) Step taken by our algorithm.*

handling techniques for minimal coordinate approaches can be used with the new algorithm. For example, [Kokkevis and Metaxas 1998] can be used for simulating collision response, enforcing unilateral and bilateral constraints as well as joint limits, and computing contact or friction forces when the system is in contact with environment.

The most significant limitation of the new algorithm is that it can only be applied to systems with parameterizable kinematics. In general, closed loop systems do not have a closed form parametric description, although simple closed loop systems, such as 4 bar linkages, do. In some cases even open loop systems do not have closed form parametric kinematics. However, in practice parameterizability is not generally a limitation since the universe of mechanisms which can be described this way is very large.

Another issue is handling topology change. If we can predefine a set of sub-systems that a system can break into, then we can compute the motion equations of the sub-systems offline and the switch between the equations in real time. However, if one wants to simulate completely general topological changes, such as breaking a system into every single rigid bodies, a maximal coordinate approach will probably be a better choice.

The great strength of the new algorithm is that complex joints can be easily specified and robustly and rapidly simulated. For example, a common weakness of human figure animations is that the knee joint is modeled as a hinge when it is more accurately modeled as a four bar linkage. This gives natural human knees a significantly larger range of motion than artificial knee joints, which are simple hinges. Similarly, the human scapula is a complex joint which can be modeled as a surface-surface joint type. Incorporating these complex joints in a human body simulation would be straightforward with our algorithm but considerably more problematic using existing algorithms. In addition, the robustness, efficiency, and simplicity of the algorithm make it an attractive choice for real-time applications, such as computer simulations or games, where numerical simulation failure is not an option.

References

ASCHER, U. M., AND PETZOLD, L. R. 1998. *Computer Methods for Ordinary Differential Equations and Differential-Algebraic Equations*. SIAM.

BALAFOUTIS, C. A., AND PATEL, R. V. 1991. *Dynamic Analysis of Robot Manipulators: A Cartesion Tensor Approach*. Kluwer Academic Publishers.

BARAFF, D. 1996. Linear-time dynamics using lagrange multipliers. In *Proceedings of SIGGRAPH 96*, Computer Graphics Proceedings, Annual Conference Series, 137–146.

COHEN, M. F. 1992. Interactive spacetime control for animation. *Computer Graphics (Proceedings of SIGGRAPH 92* (July), 293–302.

FEATHERSTONE, R., AND ORIN, D. 2000. Robot dynamics: equations and algorithms. *IEEE Int'l Conf. on Robotics and Automation (IROS) 2000*, 826–834.

FEATHERSTONE, R. 1987. *Robot Dynamics Algorithms*. Kluwer Adademic Publishers, Boston.

GRINSPUN, E., HIRANI, A. N., DESBRUN, M., AND SCHRĆDER, P. 2003. Discrete shells. *2003 ACM SIGGRAPH / Eurographics Symposium on Computer Animation* (aug), 62–67.

GUENTER, B. 2007. Efficient symbolic differentiation for graphics applications. *ACM Transactions on Graphics 26*, 3 (July), 108:1–108:12.

HERBERT GOLDSTEIN, CHARLES P. POOLE, J. L. S. 2001. *Classical Mechanics*.

HOLLARS, M. G., ROSENTHAL, D. E., AND SHERMAN, M. A. 1994. *SD/FAST User's Manual*.

KASS, M. 1992. Condor: Constraint-based dataflow. *Computer Graphics (Proceedings of SIGGRAPH 92)* (July), 321–330.

KOKKEVIS, E., AND METAXAS, D. 1998. Efficient dynamic constraints for animating articulated figures. *Multibody System Dynamics*, 89–114.

KRY, P. G., AND PAI, D. K. 2003. Continuous contact simulation for smooth surfaces. *ACM Transactions on Graphics 22*, 1 (Jan.), 106–129.

LEE, S.-H., AND TERZOPOULOS, D. 2008. Spline joints for multibody dynamics. *ACM Transactions on Graphics 27*, 3 (aug), 22:1–22:8.

OLSSON, H., TUMMESCHEIT, H., AND ELMQVIST, H. 2005. Using automatic differentiation for partial derivatives of functions in modelica. *Proceedings of the 4th International Modelica Conference* (Mar.), 105–112.

POPOVIĆ, J., SEITZ, S. M., ERDMANN, M., POPOVIĆ, Z., AND WITKIN, A. P. 2000. Interactive manipulation of rigid body simulations. *Proceedings of ACM SIGGRAPH 2000* (July), 209:1–209:9.

SAMIN, J.-C., AND FISETTE, P. 2003. *Symbolic Modeling of Multibody Systems*. Springer.

SHI, P., AND MCPHEE, J. 2000. Dynamics of flexible multibody systems using virtual work and linear graph theory. *Multibody System Dynamics 4*, 4, 355–381.

TURNER, J. D. 2003. Automated generation of high-order partial derivative models. *AIAA Journal 41*, 8, 1590–1598.

VILLARD, D., AND ARNALDI, B. 1996. Symbolic differentiation library for simulation of multibody rigid systems. *Mathematics and Computers in Simulation 42*, 4-6, 659–673.