# Implicit Runge-Kutta Integration of the Equations of Multibody Dynamics in Descriptor Form

E. J. Haug

Department of Mechanical Engineering

The University of Iowa

D. Negrut

Mechanical Dynamics, Inc.

C. Engstler

Tubingen University

**Abstract.** Implicit Runge-Kutta integration algorithms based on generalized coordinate partitioning are presented for numerical solution of the differential-algebraic equations of motion of multibody dynamics. Second order integration formulas are derived from well known first order Runge-Kutta integrators, defining independent generalized coordinates and their first time derivative as functions of independent accelerations. The latter are determined as the solution of discretized equations of motion that are obtained by inflating underlying state space, second order ordinary differential equations of motion in independent coordinates to descriptor form. Dependent variables in the formulation, including Lagrange multipliers, are determined using kinematic and kinetic equations of multibody dynamics. The proposed method is tested with a large-scale mechanical system that exhibits stiff behavior. Results show that the algorithm is robust and has the capability to integrate the differential-algebraic equations of motion for stiff multibody dynamic systems.

## 1. Introduction

In this paper, $\mathbf{q} = [q_1, q_2, \ldots, q_k]^T$ denotes the vector of generalized coordinates that define the state of a multibody system (Haug, 1989). For rigid bodies, the generalized coordinates are Cartesian position coordinates and orientation Euler parameters of body centroidal reference frames. Joints connecting the bodies of a mechanical system restrict their relative motion and impose constraints on the generalized coordinates. Kinematic constraints are expressed as algebraic expressions involving generalized coordinates; i.e., expressions of the form

$$\boldsymbol{\Phi}(\mathbf{q}) \equiv [\Phi_1(\mathbf{q}), \Phi_2(\mathbf{q}), \ldots, \Phi_m(\mathbf{q})]^{\mathrm{T}} = \mathbf{0} \qquad (1)$$

Differentiating Eq. (1) with respect to time yields the kinematic velocity equation,

$$\boldsymbol{\Phi}_{\mathbf{q}}(\mathbf{q})\dot{\mathbf{q}} = \mathbf{0} \qquad (2)$$

where subscript denotes partial differentiation; i.e., $\Phi_{\mathbf{q}} = \left[\frac{\partial \Phi_i}{\partial q_j}\right]$, and an over dot denotes differentiation with respect to time. Differentiating Eq. (2) with respect to time yields the kinematic acceleration equation,

$$\boldsymbol{\Phi}_{\mathbf{q}}(\mathbf{q})\ddot{\mathbf{q}} = -(\boldsymbol{\Phi}_{\mathbf{q}}\dot{\mathbf{q}})_{\mathbf{q}}\dot{\mathbf{q}} \equiv \tau(\mathbf{q}, \dot{\mathbf{q}}) \qquad (3)$$

Equations (1) through (3) characterize the admissible motion of the mechanical system.

The mechanical system configuration changes in time under the effect of applied forces. The Lagrange multiplier form of the constrained equations of motion for the mechanical system is (Haug, 1989)

$$\mathbf{M}(\mathbf{q})\ddot{\mathbf{q}} + \boldsymbol{\Phi}_{\mathbf{q}}^{\mathrm{T}}(\mathbf{q})\lambda = \mathbf{Q}^{\mathrm{A}}(\mathbf{q}, \dot{\mathbf{q}}, \mathrm{t}) \qquad (4)$$

where $\mathbf{M}(\mathbf{q})$ is the system mass matrix, $\boldsymbol{\lambda}$ is the vector of Lagrange multipliers that account for workless constraint forces, and $\mathbf{Q}^{A}(\mathbf{q}, \dot{\mathbf{q}}, t)$ is the vector of generalized applied forces.

Equations (1) through (4) comprise a system of differential-algebraic equations (DAE). It is known (Petzold, 1982) that DAE are not ordinary differential equations (ODE). While analytically satisfying Eqs. (1) and (4) assures that Eqs. (2) and (3) are also satisfied, when the problem is solved numerically, this ceases to be the case. In general, the task of obtaining a numerical solution of the DAE of Eqs. (1) through (4) is substantially more difficult and prone to intense numerical computation than one of solving ODE. For a review of the literature on numerical integration methods for solution of the DAE of multibody dynamics, the reader is referred to a paper by Haug, Negrut, and Iancu (1997).

## 2. Differential-Algebraic Equations of Multibody Dynamics

Equations (3) and (4) may be written in matrix form as

$$\begin{bmatrix} \mathbf{M}(\mathbf{q}) & \boldsymbol{\Phi}_{\mathbf{q}}^{T} \\ \boldsymbol{\Phi}_{\mathbf{q}} & \mathbf{0} \end{bmatrix} \begin{bmatrix} \ddot{\mathbf{q}} \\ \boldsymbol{\lambda} \end{bmatrix} = \begin{bmatrix} \mathbf{Q}^{A}(\mathbf{q}, \dot{\mathbf{q}}, t) \\ \boldsymbol{\tau}(\mathbf{q}, \dot{\mathbf{q}}) \end{bmatrix} \tag{5}$$

which is called the descriptor form of the equations of motion. Equations (1), (2), and (5) must be satisfied by the numerical solution to be constructed. This system of differential-algebraic equations may be treated by reducing it to a set of state-space ordinary differential equations. This is most easily done by selecting an independent subset of the generalized coordinates **q** and reducing the equations of motion to differential equations in the independent coordinates.

In order to determine a partitioning of the generalized coordinates **q** into dependent and independent coordinate vectors **u** and **v**, respectively, a set of consistent generalized coordinates $\mathbf{q}^0$; i.e., satisfying Eq. (1), is first determined. In this configuration, the constraint Jacobian matrix is evaluated and numerically factored, using the Gauss-Jordan algorithm (Atkinson, 1989),

$$\Phi_{\mathbf{q}}(\mathbf{q}_0) \rightarrow \left[\Phi_{\mathbf{u}}(\mathbf{q}_0) \middle| \ \Phi_{\mathbf{v}}(\mathbf{q}_0)\right] \tag{6}$$

The order of appearance of generalized coordinates associated with columns of the resulting matrix yields a nonsingular sub-Jacobian with respect to **u**; i.e.,

$$\det\left(\Phi_{\mathbf{u}}(\mathbf{q}_0)\right) \neq 0 \tag{7}$$

This can always be done if the constraint equations are independent (Haug, 1989).

Having partitioned the generalized coordinates, Eqs. (1) through (4) can be rewritten in the associated partitioned form (Haug, 1989),

$$\mathbf{M}^{\mathbf{vv}}(\mathbf{u},\mathbf{v})\ddot{\mathbf{v}} + \mathbf{M}^{\mathbf{vu}}(\mathbf{u},\mathbf{v})\ddot{\mathbf{u}} + \Phi_{\mathbf{v}}^{\mathbf{T}}(\mathbf{u},\mathbf{v})\lambda = \mathbf{Q}^{\mathbf{v}}(\mathbf{u},\mathbf{v},\dot{\mathbf{u}},\dot{\mathbf{v}}) \tag{8}$$

$$\mathbf{M}^{\mathbf{uv}}(\mathbf{u},\mathbf{v})\ddot{\mathbf{v}} + \mathbf{M}^{\mathbf{uu}}(\mathbf{u},\mathbf{v})\ddot{\mathbf{u}} + \Phi_{\mathbf{u}}^{\mathbf{T}}(\mathbf{u},\mathbf{v})\lambda = \mathbf{Q}^{\mathbf{u}}(\mathbf{u},\mathbf{v},\dot{\mathbf{u}},\dot{\mathbf{v}}) \tag{9}$$

$$\Phi(\mathbf{u},\mathbf{v}) = \mathbf{0} \tag{10}$$

$$\Phi_{\mathbf{u}}(\mathbf{u},\mathbf{v})\dot{\mathbf{u}} + \Phi_{\mathbf{v}}(\mathbf{u},\mathbf{v})\dot{\mathbf{v}} = \mathbf{0} \tag{11}$$

$$\Phi_{\mathbf{u}}(\mathbf{u},\mathbf{v})\ddot{\mathbf{u}} + \Phi_{\mathbf{v}}(\mathbf{u},\mathbf{v})\ddot{\mathbf{v}} = \tau(\mathbf{u},\mathbf{v},\dot{\mathbf{u}},\dot{\mathbf{v}}) \tag{12}$$

The condition of Eq. (7) and the implicit function theorem (Corwin and Szczarba, 1982) guarantee that Eq. (10) can be solved for **u** as a function of **v**,

$$\mathbf{u} = \mathbf{g}(\mathbf{v}) \tag{13}$$

where the function $\mathbf{g}(\mathbf{q})$ has as many continuous derivatives as does the constraint function $\Phi(\mathbf{q})$. Thus, at an admissible configuration $\mathbf{q}_0$, there exist neighborhoods $\mathbf{U}_1$ of $\mathbf{v}^0$ and $\mathbf{U}_2$ of $\mathbf{u}^0$, and a function $\mathbf{g}: \mathbf{U}_1 \rightarrow \mathbf{U}_2$ such that for any $\mathbf{v} \in \mathbf{U}_1$, Eq. (10) is identically satisfied when **u** is given by Eq. (13). The analytical form of the function

$\mathbf{g}(\mathbf{v})$ is not known, but $\mathbf{g}(\mathbf{v}^*)$ can be evaluated by fixing $\mathbf{v} = \mathbf{v}^*$ in Eq. (10) and iteratively solving for $\mathbf{u}^* = \mathbf{g}(\mathbf{v}^*)$.

Using the partitioning of generalized coordinates induced by Eq. (6), the system of DAE in Eqs. (8), (9), and (12) is reduced to a state-space ODE, through a succession of steps that use information provided by Eqs. (10) and (11). Since the coefficient matrix of $\dot{\mathbf{u}}$ in Eq. (11) is nonsingular, $\dot{\mathbf{u}}$ can be determined as a function of **v** and $\dot{\mathbf{v}}$, where Eq. (13) is used to eliminate explicit dependence on **u**. Next, Eq. (12) uniquely determines $\ddot{\mathbf{u}}$ as a function of **v**, $\dot{\mathbf{v}}$, and $\ddot{\mathbf{v}}$, where results from Eqs. (11) and (13) are substituted. Since the coefficient matrix of $\lambda$ in Eq. (9) is nonsingular, $\lambda$ can be determined uniquely as a function of **v**, $\dot{\mathbf{v}}$, and $\ddot{\mathbf{v}}$, using previously derived results. Finally, each of the preceding results is substituted into Eq. (8) to obtain an underlying state-space ODE in only the independent generalized coordinates **v** (Haug, 1989),

$$\hat{\mathbf{M}}(\mathbf{q})\ddot{\mathbf{v}} = \hat{\mathbf{Q}}(\mathbf{q}, \dot{\mathbf{q}}, \mathrm{t}) \tag{14}$$

where

$$\begin{aligned} \hat{\mathbf{M}} &= \mathbf{M}^{\mathbf{vv}} - \mathbf{M}^{\mathbf{vu}}\Phi_{\mathbf{u}}^{-1}\Phi_{\mathbf{v}} - \Phi_{\mathbf{v}}^{\mathrm{T}}\left(\Phi_{\mathbf{u}}^{-1}\right)^{\mathrm{T}}\left[\mathbf{M}^{\mathbf{uv}} - \mathbf{M}^{\mathbf{uu}}\Phi_{\mathbf{u}}^{-1}\Phi_{\mathbf{v}}\right] \\ \hat{\mathbf{Q}} &= \mathbf{Q}^{\mathbf{v}} - \mathbf{M}^{\mathbf{vu}}\Phi_{\mathbf{u}}^{-1}\tau - \Phi_{\mathbf{v}}^{\mathrm{T}}\left(\Phi_{\mathbf{u}}^{-1}\right)^{\mathrm{T}}\left[\mathbf{Q}^{\mathbf{u}} - \mathbf{M}^{\mathbf{uu}}\Phi_{\mathbf{u}}^{-1}\tau\right] \end{aligned} \tag{15}$$

## 3. Implicit Runge-Kutta Integration Formulas

Implicit Runge-Kutta numerical integration methods (Hairer, Nørsett, and Wanner, 1993) have been well developed for the solution of first ordinary differential equations of the form

$$\dot{\mathrm{y}} = f(\mathrm{t}, \mathrm{y}) \tag{16}$$

A broad range of Runge-Kutta integrators for this problem can be written in the form

$$k_i = f\left(t_n + c_i h, y_n + h\sum_{j=1}^{i} a_{ij} k_j\right), \quad i = 1,\ldots,s \tag{17}$$

$$y_{n+1} = y_n + h\sum_{i=1}^{s} b_i k_i \tag{18}$$

where $t_n$ is the current time step; $y_n$ is the approximate solution at $t_n$; $a_{ij}$, $b_i$, and $c_i$ are constants; $s$ is the number of stages in integrating from $t_n$ to $t_{n+1}$; $k_i$ are stage variables; and $h$ is the step-size.

Note that in all cases treated here, $\mathrm{a_{ij}} = 0$ for $i < j$; i.e., only diagonally implicit Runge-Kutta (DIRK) methods (Hairer, Nørsett, and Wanner, 1993) with $\mathrm{a_{ii}} \neq 0$ are considered. According to Eq. (17), in successive stages, $k_i$ appears on both sides of the equation. Since $f$ assumes a nonlinear form, an iterative method for solving for the stage variable $k_i$ is required.

In order to make Runge-Kutta methods suitable for integration of the second order differential-algebraic equations of multibody dynamics, the second argument on the right of Eq. (17) is interpreted as an approximate solution at time $t_i = t_n + c_i h$; i.e.,

$$z_i = y_n + h\sum_{j=1}^{i} a_{ij} \dot{z}_j \tag{19}$$

Substituting Eq. (19) into Eq. (17) and regarding the left side as $\dot{z}_i$, the equation can be solved for $\dot{z}_i$. Once all stages in Eq. (17) are solved for the associated $\dot{z}_i$, the results are substituted into Eq. (18) to obtain

$$y_{n+1} = y_n + h\sum_{i=1}^{s} b_i \dot{z}_i \tag{20}$$

Applying Eq. (19) to integrate acceleration yields

$$\dot{z}_i = \dot{y}_n + h\sum_{j=1}^{i} a_{ij} \ddot{z}_j \tag{21}$$

Extending the integration formula of Eq. (20) to second order,

$$\dot{y}_{n+1} = \dot{y}_n + h\sum_{i=1}^{s} b_i \ddot{z}_i \tag{22}$$

In order to enable solution of second order differential equations using Eqs. (20) and (22), with $\ddot{z}_i$ as the solution variable in the discretized equations of motion, it is helpful to substitute from Eq. (21) into Eq. (20), to obtain

$$\begin{aligned} y_{n+1} &= y_n + h\sum_{i=1}^{s} b_i \left( \dot{y}_n + h\sum_{j=1}^{i} a_{ij}\ddot{z}_j \right) \\ &= y_n + h\left( \sum_{i=1}^{s} b_i \right)\dot{y}_n + h^2\sum_{j=1}^{s}\left( \sum_{i=1}^{s} b_i a_{ij} \right)\ddot{z}_j \\ &= y_n + h\dot{y}_n + h^2\sum_{j=1}^{s} \bar{b}_j \ddot{z}_j \end{aligned} \tag{23}$$

where it is recalled that $a_{ij} = 0$ for $j > i$ and $\sum_{i=1}^{s} b_i = 1$, and defining

$$\bar{b}_j = \sum_{i=1}^{s} b_i a_{ij} \tag{24}$$

Likewise, Eq. (21) may be substituted into Eq. (19) to obtain

$$\begin{aligned} z_i &= y_n + h\sum_{j=1}^{i} a_{ij}\left( \dot{y}_n + h\sum_{\ell=1}^{j} a_{j\ell}\ddot{z}_\ell \right) \quad , \quad j = 1,\dots,i \quad \ell = 1,\dots,j \\ &= y_n + h\left( \sum_{j=1}^{i} a_{ij} \right)\dot{y}_n + h^2\sum_{\ell=1}^{s}\left( \sum_{j=1}^{s} a_{ij}a_{j\ell} \right)\ddot{z}_\ell \quad , \quad \ell < j < i \\ &= y_n + hc_i\dot{y}_n + h^2\sum_{\ell=1}^{i} \bar{a}_{i\ell}\ddot{z}_\ell \end{aligned} \tag{25}$$

where the upper limits of the double summation in Eq. (25) are changed from $l$ and $j$ to $s$, since $l \le j \le i$ and $a_{jl} = 0$ for $l > j$, and the notation

$$\bar{a}_{il} = \sum_{j=1}^{s} a_{ij} a_{jl} \tag{26}$$

is used. Note that if $l > i$, each term in the sum is zero and $a_{il} = 0$.

The approach taken in this paper to integrating the differential-algebraic equations of multibody dynamics is to insert Eqs. (21) and (25) for independent coordinates into the state-space ordinary differential equation of Eq. (14). This approach is commonly used with Newmark methods in structural dynamics (Hughes, 1987) and is applicable for multibody dynamics (Haug, Iancu, and Negrut, 1997). The resulting equations are equivalent to making the same substitution into the descriptor form of Eq. (5), using Eqs. (10) and (11) to determine dependent coordinates and their first time derivatives. The resulting discretized equations of motion involve both independent and dependent accelerations and Lagrange multipliers as solution variables. They are solved numerically and the Runge-Kutta algorithm of Eqs. (25), (21), (20), and (23) is used, just as in the conventional first order implementation of Runge-Kutta methods.

## 4. Implicit Runge-Kutta Integration of the Equations of Multibody Dynamics

In order to apply implicit Runge-Kutta methods for integrating the equations of multibody dynamics, it is instructive to first apply them to the underlying state-space ordinary differential equation of Eq. (14). Substituting Eqs. (25) and (21) into Eq. (14), at stage $s$ of the Runge-Kutta method, yields

$$\hat{\mathbf{M}}\left(\mathbf{v}_n + hc_i\dot{\mathbf{v}}_n + h^2\sum_{\ell=1}^{i}\bar{a}_{i\ell}\ddot{\mathbf{z}}_\ell\right)\ddot{\mathbf{z}}_i = \hat{\mathbf{Q}}\left(\mathbf{v}_n + hc_i\dot{\mathbf{v}}_n + h^2\sum_{\ell=1}^{i}\bar{a}_{i\ell}\ddot{\mathbf{z}}_\ell,\, \dot{\mathbf{v}}_n + h\sum_{j=1}^{i}a_{ij}\ddot{\mathbf{z}}_j,\, t_n + c_i h\right),$$

$$\mathrm{i} = 1,\ldots,\mathrm{s} \tag{27}$$

Since $\ddot{\mathbf{z}}_i$ appears in all arguments of this nonlinear equation, iterative solution methods are required. If the equation is satisfied, then dependent coordinates and their first derivatives can be determined from the kinematic constraint equations and the numerical integration processed can be continued.

Since the functions arising in Eq. (27), with $\hat{\mathbf{M}}(\cdot)$ and $\hat{\mathbf{Q}}(\cdot)$ given in Eq. (15), are highly nonlinear and complex, it is not a simple matter to computate the Jacobian matrix of the discretized equations of Eq. (27). Alternatively, the integration formulas may be substituted into the inflated descriptor form of Eq. (5), to obtain an equivalent system of equations. In order to make this substitution, however, all generalized coordinates and their first time derivatives must be written in terms of the solution variables, in this case the accelerations.

It is assumed that the generalized coordinates have been partitioned and reordered so that $\mathbf{q} = \left[\mathbf{u}^{\mathrm{T}}, \mathbf{v}^{\mathrm{T}}\right]^{\mathrm{T}}$; i.e.,

$$\mathbf{v} = \mathbf{P}\mathbf{q} \equiv \left[\mathbf{0}, \mathbf{I}\right]\mathbf{q} \tag{28}$$

where $\mathbf{P}$ is a boolean matrix containing only zeros and unit values. Since $\mathbf{P}$ is a constant matrix, $\dot{\mathbf{v}} = \mathbf{P}\dot{\mathbf{q}}$ and $\ddot{\mathbf{v}} = \mathbf{P}\ddot{\mathbf{q}}$. Likewise, with $\mathbf{w}$ as the full vector of stage generalized coordinates and $\mathbf{z}$ as independent stage coordinates,

$$\mathbf{z} = \mathbf{P}\mathbf{w} \tag{29}$$

Using this notation and recalling that the boolean matrix is constant, Eqs. (25) and (21) may be written in the form

$$\mathbf{z}_i = \mathbf{P}\mathbf{q}_n + hc_i\mathbf{P}\dot{\mathbf{q}}_n + h^2\sum_{\ell=1}^{i}\bar{a}_{i\ell}\mathbf{P}\ddot{\mathbf{w}}_\ell \tag{30}$$

$$\dot{\mathbf{z}}_i = \mathbf{P}\dot{\mathbf{q}}_n + h\sum_{j=1}^{i}a_{ij}\mathbf{P}\ddot{\mathbf{w}}_j \tag{31}$$

Substituting independent stage coordinates $\mathbf{z}_i$ from Eq. (30) into Eq. (13) yields dependent stage coordinates $\mathbf{x}_i$ as

$$\mathbf{x}_i = \mathbf{g}(\mathbf{z}_i) = \mathbf{g}\left( \mathbf{P}\mathbf{q}_{\mathrm{n}} + hc_i\mathbf{P}\dot{\mathbf{q}}_{\mathrm{n}} + h^2 \sum_{\ell=1}^{i} \overline{a}_{i\ell}\mathbf{P}\ddot{\mathbf{w}}_{\ell} \right) \tag{32}$$

Even though the function $\mathbf{g}(\cdot)$ is not known explicitly, Eq. (32) shows clearly that dependent stage coordinates are functions of stage accelerations, through the Runge-Kutta integration formulas. From Eq. (11), dependent stage velocities may be written as functions of independent stage velocities and, through use of Eq. (31), as functions of stage accelerations,

$$\dot{\mathbf{x}}_i = -\mathbf{\Phi}_{\mathbf{u}}^{-1}\mathbf{\Phi}_{\mathbf{v}}\dot{\mathbf{z}}_i \equiv \mathbf{H}\dot{\mathbf{z}}_i = \mathbf{H}\mathbf{P}\dot{\mathbf{q}}_n + h\sum_{\ell=1}^{i} a_{ij}\mathbf{H}\mathbf{P}\ddot{\mathbf{w}}_j \tag{33}$$

where $\mathbf{H}$ is computed as the solution of the multiple right side system of linear equations

$$\mathbf{\Phi}_{\mathbf{u}}\mathbf{H} = -\mathbf{\Phi}_{\mathbf{v}} \tag{34}$$

Regarding all stage generalized coordinates and their first time derivatives to be functions of accelerations, via Eqs. (30) through (33), the equations of motion can be written in the form

$$\mathbf{M}(\mathbf{w}_i(\ddot{\mathbf{w}}_i))\ddot{\mathbf{w}}_i + \mathbf{\Phi}_{\mathbf{q}}^{T}(\mathbf{w}_i(\ddot{\mathbf{w}}_i))\lambda_i = \mathbf{Q}^{\mathrm{A}}(\mathbf{w}_i(\ddot{\mathbf{w}}_i), \dot{\mathbf{w}}_i(\ddot{\mathbf{w}}_i)) \tag{35}$$

Similarly, the kinematic acceleration equations of Eq. (3) may be written as

$$\mathbf{\Phi}_{\mathbf{q}}(\mathbf{w}_i(\ddot{\mathbf{w}}_i))\ddot{\mathbf{w}}_i = \tau(\mathbf{w}_i(\ddot{\mathbf{w}}_i), \dot{\mathbf{w}}_i(\ddot{\mathbf{w}}_i)) \tag{36}$$

Equations (35) and (36) emphasize the dependence of coefficients in the descriptor form of the equations of motion on the unknown stage accelerations. In order to iteratively solve Eqs. (35) and (36), all derivatives with respect to the unknown accelerations must be determined.

From Eq. (30),

$$\mathbf{z}_{i_{\ddot{\mathbf{w}}_i}} = h^2 \bar{a}_{ii} \mathbf{P} \tag{37}$$

Differentiating Eq. (10) with respect to stage accelerations,

$$\Phi_{\mathbf{u}} \mathbf{x}_{i_{\ddot{\mathbf{w}}_i}} = -\Phi_{\mathbf{v}} \mathbf{z}_{i_{\ddot{\mathbf{w}}_i}} \tag{38}$$

Solving for the desired derivatives and using Eq. (37),

$$\begin{aligned} \mathbf{x}_{i_{\ddot{\mathbf{w}}_i}} &= -\Phi_{\mathbf{u}}^{-1} \Phi_{\mathbf{v}} \mathbf{z}_{i_{\ddot{\mathbf{w}}_i}} = \mathbf{H} \mathbf{z}_{i_{\ddot{\mathbf{w}}_i}} \\ &= h^2 \bar{a}_{ii} \mathbf{H} \mathbf{P} \end{aligned} \tag{39}$$

Combining these results yields

$$\mathbf{w}_{i_{\ddot{\mathbf{w}}_i}} = h^2 \bar{a}_{ii} \begin{bmatrix} \mathbf{HP} \\ \mathbf{P} \end{bmatrix} = h^2 \bar{a}_{ii} \hat{\mathbf{H}} \tag{40}$$

where

$$\hat{\mathbf{H}} = \begin{bmatrix} \mathbf{HP} \\ \mathbf{P} \end{bmatrix} \tag{41}$$

From Eq. (31),

$$\dot{\mathbf{z}}_{i_{\ddot{\mathbf{w}}_i}} = h a_{ii} \mathbf{P} \tag{42}$$

Differentiating Eq. (11) with respect to stage accelerations and using Eqs. (40) and (42) yields

$$\begin{aligned} \Phi_{\mathbf{u}} \dot{\mathbf{x}}_{i_{\ddot{\mathbf{w}}_i}} &= -\Phi_{\mathbf{v}} \dot{\mathbf{z}}_{i_{\ddot{\mathbf{w}}_i}} - \left(\Phi_{\mathbf{q}} \dot{\mathbf{w}}_i\right)_{\mathbf{q}} \mathbf{w}_{i_{\ddot{\mathbf{w}}_i}} \\ &= -h a_{ii} \Phi_{\mathbf{v}} \mathbf{P} - h^2 \bar{a}_{ii} \left(\Phi_{\mathbf{q}} \dot{\mathbf{w}}_i\right)_{\mathbf{q}} \hat{\mathbf{H}} \end{aligned} \tag{43}$$

Solving for the desired derivatives yields

$$\begin{aligned} \dot{\mathbf{x}}_{i_{\ddot{\mathbf{w}}_i}} &= h a_{ii} \mathbf{HP} - h^2 \bar{a}_{ii} \Phi_{\mathbf{u}}^{-1} \left(\Phi_{\mathbf{q}} \dot{\mathbf{w}}_i\right)_{\mathbf{q}} \hat{\mathbf{H}} \\ &= h a_{ii} \mathbf{HP} + h^2 \bar{a}_{ii} \mathbf{J} \end{aligned} \tag{44}$$

where **J** is defined as the solution of

$$\boldsymbol{\Phi}_{\mathbf{u}}\mathbf{J} = -\left(\boldsymbol{\Phi}_{\mathbf{q}}\dot{\mathbf{w}}_i\right)_{\mathbf{q}}\hat{\mathbf{H}} \tag{45}$$

Combining Eqs. (42) and (44) yields

$$\dot{\mathbf{w}}_{i_{\ddot{\mathbf{w}}_i}} = ha_{ii}\hat{\mathbf{H}} + h^2\overline{a}_{ii}\hat{\mathbf{J}} \tag{46}$$

where

$$\hat{\mathbf{J}} = \begin{bmatrix} \mathbf{J} \\ \mathbf{0} \end{bmatrix} \tag{47}$$

With the results of Eqs. (40) and (46), all derivatives required to iteratively solve Eqs. (35) and (36) are available. To make explicit the derivative calculations required, the discretized equations are written in descriptor form as

$$\boldsymbol{\Psi} \equiv \begin{bmatrix} \mathbf{M}(\mathbf{w}_i)\ddot{\mathbf{w}}_i + \boldsymbol{\Phi}_{\mathbf{q}}^T(\mathbf{w}_i)\boldsymbol{\lambda}_i - \mathbf{Q}^A(\mathbf{w}_i,\dot{\mathbf{w}}_i) \\ \boldsymbol{\Phi}_{\mathbf{q}}(\mathbf{w}_i)\ddot{\mathbf{w}}_i - \boldsymbol{\tau}(\mathbf{w}_i,\dot{\mathbf{w}}_i) \end{bmatrix} = \mathbf{0} \tag{48}$$

Derivatives of $\boldsymbol{\Psi}$ with respect to stage accelerations $\ddot{\mathbf{w}}_i$ are obtained, using the chain rule of differentiation and Eqs. (40) and (46), as

$$\boldsymbol{\Psi}_{\ddot{\mathbf{w}}_i} = \begin{bmatrix} \mathbf{M} + h^2\overline{a}_{ii}\left[\left(\mathbf{M}\ddot{\mathbf{w}}_{\mathrm{i}}\right)_{\mathbf{q}} + \left(\boldsymbol{\Phi}_{\mathbf{q}}^T\boldsymbol{\lambda}_i\right)_{\mathbf{q}} - \mathbf{Q}_{\mathbf{q}}^A\right]\hat{\mathbf{H}} - \mathbf{Q}_{\dot{\mathbf{q}}}^A\left[ha_{ii}\hat{\mathbf{H}} + h^2\overline{a}_{ii}\hat{\mathbf{J}}\right] \\ \boldsymbol{\Phi}_{\mathbf{q}} + h^2\overline{a}_{ii}\left[\left(\boldsymbol{\Phi}_{\mathbf{q}}\ddot{\mathbf{w}}_{\mathrm{i}}\right)_{\mathbf{q}} - \boldsymbol{\tau}_{\mathbf{q}}\right]\hat{\mathbf{H}} - \boldsymbol{\tau}_{\dot{\mathbf{q}}}\left[ha_{ii}\hat{\mathbf{H}} + h^2\overline{a}_{ii}\hat{\mathbf{J}}\right] \end{bmatrix} \tag{49}$$

More directly, the derivative of $\boldsymbol{\Psi}$ with respect to the unknown Lagrange multiplier is

$$\boldsymbol{\Psi}_{\boldsymbol{\lambda}_{\mathrm{i}}} = \begin{bmatrix} \boldsymbol{\Phi}_{\mathbf{q}}^{\mathrm{T}} \\ \mathbf{0} \end{bmatrix} \tag{50}$$

These results may be combined, using the notation

$$\mathbf{J}_0 \equiv \begin{bmatrix} \mathbf{M} + \mathrm{h}^2 \bar{a}_{ii}\left[\left\{(\mathbf{M}\ddot{\mathbf{q}})_{\mathbf{q}} + \left(\mathbf{\Phi}_{\mathbf{q}}^T \lambda_i\right)_{\mathbf{q}} - \mathbf{Q}_{\mathbf{q}}^{\mathrm{A}}\right\}\hat{\mathbf{H}} - \mathbf{Q}_{\dot{\mathbf{q}}}^{\mathrm{A}}\hat{\mathbf{J}}\right] - h a_{ii}\mathbf{Q}_{\dot{\mathbf{q}}}^{\mathrm{A}}\hat{\mathbf{H}} & \mathbf{\Phi}_{\mathbf{q}}^T \\ \mathbf{\Phi}_{\mathbf{q}} + \mathrm{h}^2 \bar{a}_{ii}\left[\left\{(\mathbf{\Phi}_{\mathbf{q}}\ddot{\mathbf{q}})_{\mathbf{q}} - \tau_{\mathbf{q}}\right\}\hat{\mathbf{H}} - \tau_{\dot{\mathbf{q}}}\hat{\mathbf{J}}\right] - h a_{ii}\tau_{\dot{\mathbf{q}}}\hat{\mathbf{H}} & \mathbf{0} \end{bmatrix} \tag{51}$$

where the integration Jacobian $\mathbf{J}_0$ is evaluated in the configuration $(\mathbf{q}_n, \dot{\mathbf{q}}_n)$ from the beginning of the macro-step. A quasi-Newton method is applied to iterate for the stage accelerations $\ddot{\mathbf{w}}_i$ and Lagrange multipliers $\lambda_i$ as

$$\begin{gathered} \mathbf{J}_0 \begin{bmatrix} \Delta\ddot{\mathbf{w}}_i \\ \Delta\lambda_{\mathrm{i}} \end{bmatrix}^{(j)} = -\mathbf{\Psi}^{(\mathrm{j}-1)} \\ \begin{bmatrix} \ddot{\mathbf{w}}_i \\ \lambda_{\mathrm{i}} \end{bmatrix}^{(j)} = \begin{bmatrix} \ddot{\mathbf{w}}_i \\ \lambda_{\mathrm{i}} \end{bmatrix}^{(j-1)} + \begin{bmatrix} \Delta\ddot{\mathbf{w}}_i \\ \Delta\lambda_{\mathrm{i}} \end{bmatrix}^{(j)} \end{gathered} \tag{52}$$

At each stage of the algorithm, during each iteration for the solution $(\ddot{\mathbf{w}}_i, \lambda_i)$ of Eq. (48), the dependent stage positions are iteratively computed as

$$\begin{aligned} \Phi_{\mathbf{u}}\, \Delta\mathbf{x}_i^{(\ell)} &= -\Phi(\mathbf{x}_i^{(\ell-1)}, \mathbf{z}_i) \\ \mathbf{x}_i^{(\ell)} &= \mathbf{x}_i^{(\ell-1)} + \Delta\mathbf{x}_i^{(\ell)} \end{aligned} \tag{53}$$

whereas the dependent stage velocities are computed as the solution of the linear system

$$\mathbf{\Phi}_{\mathbf{u}}\dot{\mathbf{x}}_i = -\mathbf{\Phi}_{\mathbf{v}}\dot{\mathbf{z}}_i \tag{54}$$

A challenge that the descriptor form method poses is the computation of the derivatives $(\mathbf{M}\ddot{\mathbf{q}})_{\mathbf{q}}$, $\left(\mathbf{\Phi}_{\mathbf{q}}^{\mathrm{T}}\lambda\right)_{\mathbf{q}}$, $\mathbf{Q}_{\mathbf{q}}^{\mathrm{A}}$, $\mathbf{Q}_{\dot{\mathbf{q}}}^{\mathrm{A}}$, $\tau_{\mathbf{q}}$, and $\tau_{\dot{\mathbf{q}}}$. Details about how these quantities are obtained for the case when the mechanical system is modeled using Cartesian coordinates with Euler parameters for body orientation are given by Serban and Haug (1998).

## 5. Integration Formulas

The descriptor form algorithm introduced in the previous Section is first implemented with a singly diagonal implicit Runge-Kutta (SDIRK) formula (Hairer and Wanner, 1996). For SDIRK formulas, the associated Butcher's tableau assumes the form

Table 1. Butcher's Tableau For SDIRK Formulas

| | | | | | |
|---|---|---|---|---|---|
| $c_1$ | $\gamma$ | $0$ | $\ldots$ | $\ldots$ | $0$ |
| $c_2$ | $a_{21}$ | $\gamma$ | $\ldots$ | $\ldots$ | $0$ |
| $\ldots$ | $\ldots$ | $\ldots$ | $\ldots$ | $\ldots$ | $\ldots$ |
| $c_s$ | $a_{s1}$ | $a_{s2}$ | $\ldots$ | $\ldots$ | $\gamma$ |
| $y_1$ | $b_1$ | $b_2$ | $\ldots$ | $\ldots$ | $b_s$ |
| $\hat{y}_1$ | $\hat{b}_1$ | $\hat{b}_2$ | $\ldots$ | $\ldots$ | $\hat{b}_s$ |

With these notations, the stage values $w_i$ are computed as

$$w_i = f(t_n + c_i h, y_n + h\sum_{j=1}^{i} a_{ij} w_j) \tag{55}$$

and the solution at time $t_{n+1}$ is obtained as

$$y_{n+1} = y_n + h\sum_{i=1}^{s} b_i w_i \tag{56}$$

For SDIRK formulas, $a_{ii} = \gamma$, $i = 1, \ldots, s$. The matrix $\mathbf{A} = [\mathrm{a}_{\mathrm{ij}}]$ is called the coefficient matrix of the formula, and for SDIRK methods it is nonsingular.

The SDIRK formula sought for the descriptor form method should be L-stable (Hairer and Wanner, 1996), and of average order. The L-stability attribute ensures good stability properties and order-preservation, even for extremely stiff problems (no stiffness-based order reduction). Thus, the formula chosen is of order $p = 4$, with $s = 5$ stages. This order is high enough to ensure good efficiency for tolerances typically used in simulations of engineering application, namely $10^{-2}$ to $10^{-5}$. Since the coefficient matrix $\mathbf{A}$ of the formula is nonsingular, if the condition

$$b_i = a_{si}, \qquad i = 1, \ldots, s \tag{57}$$

is satisfied, then the SDIRK formula becomes stiffly accurate (Hairer and Wanner, 1996). A stiffly-accurate formula is automatically A- and L-stable, and good stability properties and order preservation are guaranteed.

Error control is based on adjusting the integration step-size such that an approximation of the local truncation error is always kept smaller than a user-prescribed tolerance. The approximation of the local truncation error is obtained by means of a second numerical approximation of the solution that is provided by a different, usually lower order, Runge-Kutta formula. To make the process efficient, the second Runge-Kutta method is designed to use information generated during the process of finding the actual numerical solution with the original integration formula. Typically this translates in the second formula to using some or all of the stage values $w_i$ computed by the original formula. In this context, the last row of Table 1 contains the coefficients of the embedded formula. Thus, a second approximation of the solution at time $t_{n+1}$ is obtained as

$$\hat{y}_{n+1} = y_n + h\sum_{i=1}^{s} \hat{b}_i w_i \tag{58}$$

and the approximation of the local truncation error is given by $y_{n+1} - \hat{y}_{n+1}$. Componentwise, this error is kept smaller than a composite error tolerance $sc_i$,

$$| y_{(n+1)i} - \hat{y}_{(n+1)i} | \le sc_i \tag{59}$$

where $sc_i = Atol_i + \max(|y_{ni}|, |y_{(n+1)i}|) \cdot Rtol_i$, and $Atol_i$ and $Rtol_i$ are user prescribed integration error tolerances. As a measure of the error, the value

$$err = \sqrt{\frac{1}{k}\sum_{i=1}^{k}\left(\frac{y_{(n+1)i} - \hat{y}_{(n+1)i}}{sc_i}\right)^2} \tag{60}$$

is considered here. This value is compared to 1, in order to find an optimal step-size. From asymptotic error behavior, $err \approx C \cdot h^{q+1}$, and from $1 \approx C \cdot h_{opt}^{q+1}$ (where $q = \min(p, \hat{p})$, with $p$ and $\hat{p}$ being the order of the formulas used), the optimal step-size is obtained as

$$h_{opt} = h\left(\frac{1}{err}\right)^{\frac{1}{q+1}} \tag{61}$$

A safety factor $fac$ usually multiplies $h_{opt}$, such that the error is acceptable at the end of the next step with high probability. Further, $h$ is not allowed to increase or decrease too fast. Thus, the value used for the new step-size is

$$h_{new} = h \cdot \min(\text{facmax}, \max(\text{facmin}, fac \cdot (1/err)^{1/(q+1)}))$$

If, at the end of the current step, $err \leq 1$, the step is accepted. The solution is then advanced with $y_{n+1}$ and a new step is computed, with $h_{new}$ as step-size. Otherwise, the step is rejected and computations for the current step are repeated with the new step-size $h_{new}$. The maximal step-size increase $\text{facmax}$, usually chosen between 1.5 and 5, prevents the code from taking too large a step and contributes to its reliability. When chosen too small, it may unnecessarily increase the computational work. Finally, it is advisable to put $\text{facmax} = 1$ in steps after a step-rejection (Shampine and Watts, 1979).

The stiffly-accurate, L-stable, 5 stage, order 4 singly diagonal Runge-Kutta formula implemented with the descriptor form method of Section 4 is defined in Table 2 (Hairer and Wanner, 1996). Step-size control is based on an order 3 embedded formula whose weights $\hat{b}_i$ are provided as the last row in Table 2.

For consistency with the descriptor form method presented in Section 4, the trapezoidal formula is presented as a 2 stage Runge-Kutta method. The associated Butcher tableau is provided in Table 3. Note that since the first row of coefficients are zeros, only one stage variable needs to be computed.

Table 2. SDIRK Formula

| | | | | | |
|---|---|---|---|---|---|
| 1/4 | 1/4 | 0 | 0 | 0 | 0 |
| 3/4 | 1/2 | 1/4 | 0 | 0 | 0 |
| 11/20 | 17/50 | -1/25 | 1/4 | 0 | 0 |
| 1/2 | 371/1360 | -137/2720 | 15/544 | 1/4 | 0 |
| 1 | 25/24 | -49/48 | 125/16 | -85/12 | 1/4 |
| $y_1 =$ | 25/24 | -49/48 | 125/16 | -85/12 | 1/4 |
| $\hat{y}_1 =$ | 59/48 | -17/96 | 225/32 | -85/12 | 0 |

Table 3 Trapezoidal Formula

| | | |
|---|---|---|
| 0 | 0 | 0 |
| 1 | 1/2 | 1/2 |
| $y_1 =$ | 1/2 | 1/2 |
| $\hat{y}_1 =$ | 0 | 1 |

The trapezoidal formula is an order two, A-stable method. The embedded formula used for step-size control is backward Euler. Trapezoidal formula is often used, and it is analyzed in detail by Atkinson (1989). The important thing to point out about it is that, although the condition in Eq. (57) is satisfied; i.e. the formula is stiffly accurate, this does not result in L-stability, since the coefficient matrix **A** of the formula (see Table 3) is singular (Hairer and Wanner, 1996).

## 6. Computational Algorithms

Two computational algorithms have been developed. They are based on the proposed implicit method for the solution of the differential-algebraic equations of multibody dynamics of Section 4 and the integration formulas of Section 5. The first algorithm, denoted by *InflSDIRK*, is based on the five stage, order four, L-stable stiffly-accurate SDIRK formula (Hairer and Wanner, 1996) provided in Table 2. The second algorithm, denoted by *InflTrap*, uses the trapezoidal integration formula, whose coefficients are provided in Table 3. Both algorithms use error control mechanisms based on step-size selection. Pseudo-code for the *InflSDIRK* algorithm is provided in Table 4 and is discussed as follows.

Step 1 initializes the simulation. A consistent set of initial conditions is determined, simulation starting and ending times are defined, and an initial step-size is provided. User set integration tolerances are read during Step 2. At Step 3, the simulation loop is started and the code proceeds after saving the current system configuration. This is the configuration that is used in the event of a rejected time step, in which case integration is restarted from Step 4 with a new step-size computed by the error control mechanism.

Step 5 is a pivotal point of the implementation. If the current time step has not been rejected, the integration Jacobian is computed as in Eq. (51). Forming and factoring the integration Jacobian is the CPU intensive part of the code. If the call to integration Jacobian computation comes after an unsuccessful time step, the step-size is changed, but all other matrix quantities appearing in the expression for the integration Jacobian are available from the previous call.

Table 4. Pseudo-code for *InflSDIRK*

```
1.  Initialize Simulation
2.  Set Integration Tolerance
3.  While (t < tend) do
4.      Setup Macro-step
5.      Get Integration Jacobian
6.      Sparse Factor Integration Jacobian
7.      Do stage 1 to 5
8.          Setup Stage
9.          Do while (.NOT. converged)
10.             Integrate
11.             Recover Positions and Velocities
12.             Evaluate Error Residual.  Compute Corrections
13.             Verify Stopping Criteria
14.             Correct Accelerations and Lagrange Multipliers
15.         End do
16.     End do
17.     Check Accuracy.  Determine New Step-size
18.     Check Partition
19. End do
```

Harwell (1995) sparse linear algebra routines are used to factor the integration Jacobian. Since its sparsity pattern does not change during integration, the factorization process is efficient once Harwell routine *ma48ad* has analyzed its structure and a factorization sequence has been determined. Subsequent calls to integration Jacobian factorization use the much faster *ma48bd* factorization routine.

Step 7 initiates a loop for evaluation of the stage values $\ddot{\mathbf{w}}_i$. At Step 8, starting values for generalized accelerations and Lagrange multipliers are provided, and the iteration counter is reset to zero. If during the iterative process this counter exceeds a limit value, the time step is deemed rejected, the integration step-size is halved, and the code proceeds to Step 4.

The solution of the discretized non-linear algebraic equations is obtained during the loop that starts at Step 9 and ends at Step 15. Based on the SDIRK formula of Table 2 and Eqs. (30) and (31), accelerations are integrated in Step 10 to obtain generalized velocities, which are in turn integrated to obtain generalized positions. In the numerical implementation, the dependent coordinates are also integrated such as to provide a good starting configuration for dependent position recovery in Eq. (53). Dependent velocities are computed using the velocity kinematic constraint equation, as indicated in Eq. (54). The use of Eqs. (53) and (54) is the reason for which, although the discretization is done at the index 1 DAE (Hairer and Wanner, 1996) level, *InflSDIRK* is a state-space-based algorithm.

At Step 12, corrections in generalized accelerations and Lagrange multipliers are computed, as in Eq. (52). In Step 13, stopping criteria are checked. If norms of the residual and correction are small enough, the iteration process is stopped. Otherwise, the values of generalized accelerations and Lagrange multipliers are corrected, and the iterative process is continued.

During Step 17, based on the embedded formula provided in Table 2, the accuracy of the numerical solution is assessed. If accuracy is satisfactory, the configuration at the current grid point is accepted, and integration proceeds with a step-size that is obtained as a by-product of the accuracy check, as in Eq. (61). Otherwise, with the newly computed step-size, the code proceeds to Step 4 to restart integration.

During Step 18, the partitioning of the vector of generalized coordinates is checked and, if necessary, a new dependent/independent partitioning is determined. A repartitioning is triggered by a large value of the condition number of the dependent sub-Jacobian $\mathbf{\Phi}_{\mathbf{u}}$. Here, large means a condition number that exceeds by a factor of $\alpha = 1.25$ the value of the first condition number associated with the current partition. The value $\alpha = 1.25$ was determined as a result of numerical experiments. The proposed strategy has not caused unjustified repartitioning requests, and has been reliable. Computing the condition number of the dependent sub-Jacobian is inexpensive, since a factorization of this matrix is available after the last call to dependent position recovery. Finally, Step 19 is the end of the simulation loop.

The pseudo-code for *InflTrap* is similar to that for *InflSDIRK*. The only difference is in the number of stages of the formulas. For *InflTrap* there is no need to have the loop starting at Step 7 and ending at Step 16.

## 6.1. Description of Numerical Experiments

In order to validate *InflTrap* and *InflSDIRK*, simulation results for a US Army High Mobility Multipurpose Wheeled Vehicle (HMMWV) model are compared with reference results obtained with a third implicit method. The reference results are generated with an algorithm proposed by Negrut (1998), which is based on a first order reduction method, in conjunction with an SDIRK code of Hairer and Wanner (1996). Validation of the latter algorithm (denoted here by *ForSDIRK*) has been made by Negrut (1998).

The HMMWV in Fig. 1 is modeled using 14 bodies, as shown in Fig. 2. The bodies of the model are described in Table 5. A total of 98 generalized coordinates are used to model the vehicle. The initial partitioning of coordinates was valid for 40 seconds of simulation, and no repartitioning request was made.

Figure 1. US Army HMMWV

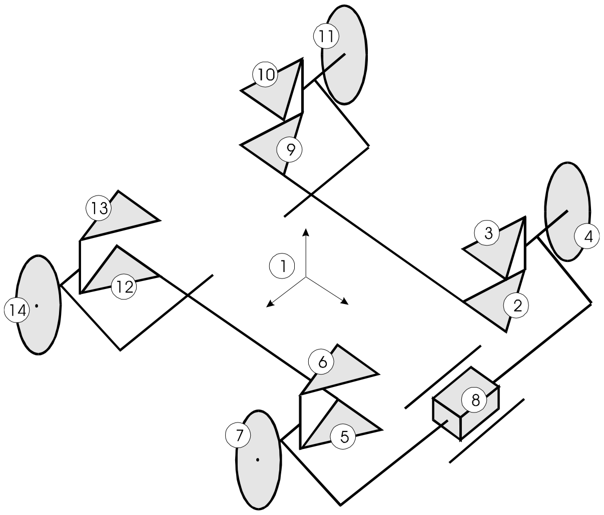


Figure 2. 14 Body Model of HMMWV

Table 5. HMMWV14 Model - Component Bodies

| No. | Body | No. | Body |
|---|---|---|---|
| 1 | Chassis | 8 | Rack |
| 2 | Left front lower control arm | 9 | Left rear lower control arm |
| 3 | Left front upper control arm | 10 | Left rear upper control arm |
| 4 | Left front wheel spindle | 11 | Left rear wheel spindle |
| 5 | Right front lower control arm | 12 | Right rear lower control arm |
| 6 | Right front upper control arm | 13 | Right rear upper control arm |
| 7 | Right front wheel spindle | 14 | Right rear wheel spindle |

Figure 3(a) shows the original topology graph of HMMWV14. Stiffness in the model is induced by replacing the revolute joints between upper control arms and chassis in the original model with spherical joints as shown in Fig. 3(b). Each joint replacement results in two additional degrees of freedom. For each such spherical joint, two translational-spring-damper-actuators (TSDA), acting in complementary directions, model bushings that control the extra degrees of freedom. The number of degrees of freedom is 19. The stiffness coefficient of each TSDA is $2.0 \cdot 10^7$ N/m, while the damping coefficient is $2.0 \cdot 10^6$ Ns/m. Tires are modeled as vertical TSDA elements, with stiffness coefficient 296,325 N/m and damping coefficient 3,502 Ns/m. The dominant eigenvalue for this example has a small imaginary part, while the real part is of the order $-2.6 \cdot 10^5$.

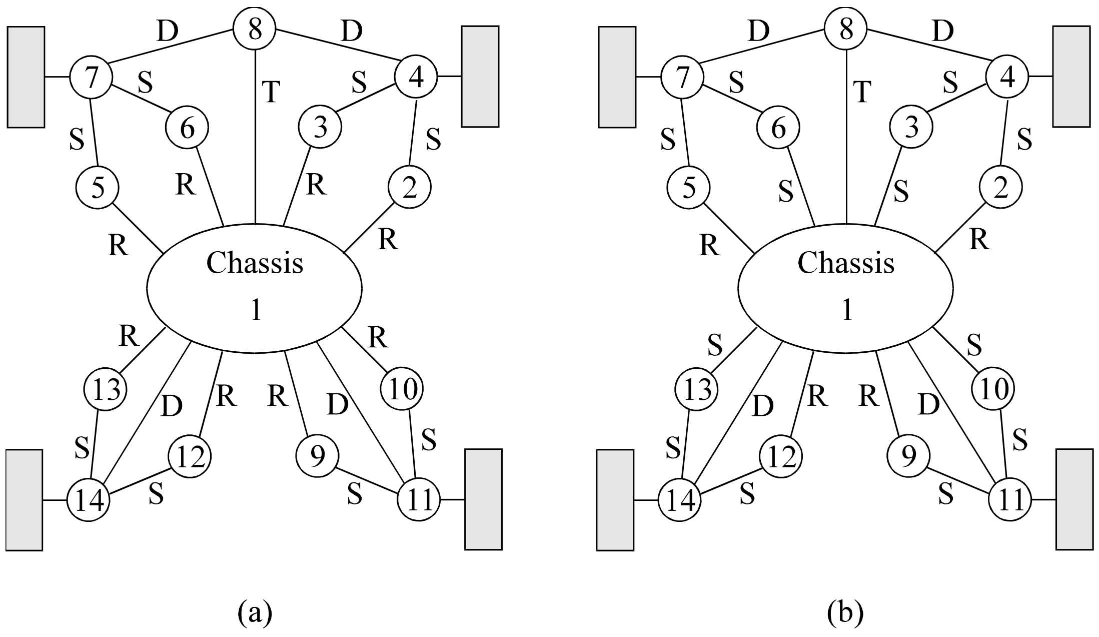


Figure 3. Topology Graph HMMWV14

The vehicle is driven at 10 mph and hits a bump. The bump's shape is a half cylinder of diameter 0.1 m. The steering rack is locked, to assure the vehicle drives straight. This reduces the number of degrees of freedom to 18. Figure 4 shows the time variation of chassis height. The front wheels hit the bump at $T \approx 0.5$ s, and the rear wheels hit the bump at $T \approx 1.2$ s. The length of the simulation is 5 seconds. Toward the end of the simulation (after approximately 4 seconds), due to overdamping, the chassis height stabilizes at approximately $z_1 = 0.71$ m.

Error analysis is carried out for the first 2 seconds. This time period is the most critical of the simulation, since after the wheels clear the bump, the vehicle does not experience any external excitation, and the motion stabilizes, due to suspension and bushing damping.

The reference solution was generated by imposing absolute and relative tolerances of $10^{-8}$ for positions and velocities. Data generated by the algorithm *ForSDIRK* are used to compare results obtained using *InflTrap* and *InflSDIRK* for the same simulation.

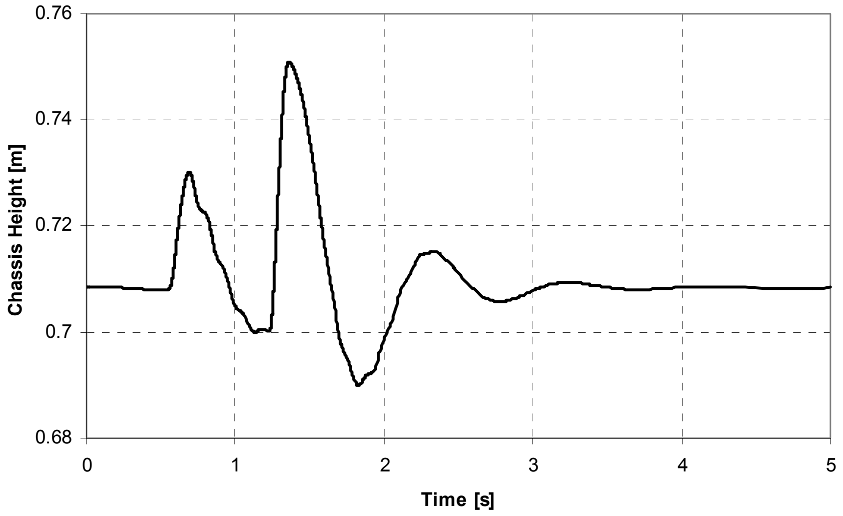


Figure 4. Chassis Height HMMWV14

## 6.2 Comparison of Accuracy

Suppose that $n$ time steps are taken during the current simulation, and the variable used for error analysis is denoted by $e$. The grid points of the simulation are denoted by $t_{init} = t_1 < t_2 < \ldots < t_n = t_{end}$, and results of the current simulation are obtained as $e_i$, $1 \le i \le n$. If $N$ is the number of time steps taken during the reference simulation, it is expected that $N >> n$. Let $T_{init} = T_1 < \ldots < T_N = T_{end}$ be the reference simulation time steps, and $E_j$, $1 \le j \le N$, be the corresponding reference values. For each $i$, $1 \le i \le n$, an integer $\mathbf{r}(i)$ is defined such that $T_{\mathbf{r}(i)} \le t_i \le T_{\mathbf{r}(i)+1}$. Based on reference data $E_{\mathbf{r}(i)-1}$, $E_{\mathbf{r}(i)}$, $E_{\mathbf{r}(i)+1}$, and $E_{\mathbf{r}(i)+2}$, spline cubic interpolation is used to generate an interpolated value $E_i^*$ at time $t_i$. If $\mathbf{r}(i)-1\le 0$, the first four reference points are considered for interpolation, whereas if $\mathbf{r}(i)+2\ge N$, the last four reference points are considered for interpolation. The error at time step $i$ is defined as

$$\Delta_i = |E_i^* - e_i| \tag{62}$$

The infinity norm of the simulation error is defined as

$$\Delta^{(k)} = \max_{1\le i\le n} \Delta_i \tag{63}$$

and it is obtained after setting the integration tolerance to $10^k, k<0$. Tables 6 and 7 contain results of the error analysis for *InflTrap* and *InflSDIRK*, respectively. The tables list in the first column the absolute and relative tolerances set for the simulation given as powers of 10; i.e., $Atol_i = Rtol_i = 10^k$. The same tolerances are imposed for both position and velocity. The variable $e$ for which error analysis is carried out is the global $x$-position of the chassis; i. e., the distance traveled by the vehicle. Thus, the second column contains the value of $\Delta^{(k)}$ position, while the third column contains the largest error for the longitudinal velocity of the vehicle

These results suggest that the algorithms perform well. The error-control mechanisms of both algorithms are reliable, and accuracy requirements imposed are met in every case. In fact, it can be observed that the step-size controllers are generally conservative; i. e., the accuracy obtained is approximately one order of magnitude better than requested. Although this contributes to reliability of the overall algorithm, it can cause unnecessary computational effort, due to selection of smaller step-sizes than are necessary. This conservative tendency of step-size selection is typical for very stiff problems, and has been explained by Shampine (1994) and mentioned by Hairer and Wanner (1996).

Table 6. *InflTrap* Errors

| Tol. ($k$) | Max. Error Position | Max. Error Velocity |
|---|---|---|
| -2 | 0.00198767020 | 0.00329997828 |
| -3 | 0.00041249616 | 0.00057043579 |
| -4 | 0.00004450907 | 0.00006135104 |
| -5 | 0.00000520341 | 0.00000641223 |

Table 7. *InflSDIRK* Errors

| Tol. ($k$) | Max. Error Position | Max. Error Velocity |
|---|---|---|
| -2 | 0.00065708165 | 0.00507871695 |
| -3 | 0.00004549833 | 0.00207167794 |
| -4 | 0.00000337006 | 0.00013460534 |
| -5 | 0.00000069689 | 0.00004154424 |

The plots in Figs. 5 and 6 are based on results presented in Tables 6 and 7. The ordinate is the value $k$ of the tolerance with which the simulation is run. On the abscissa are displayed the values of the simulation errors $\Delta^{(k)}$, both on a logarithmic scale. It can be seen that the precision in positions is better, and that the gap between the accuracy of results at the position and velocity levels is wider for *InflSDIRK*.

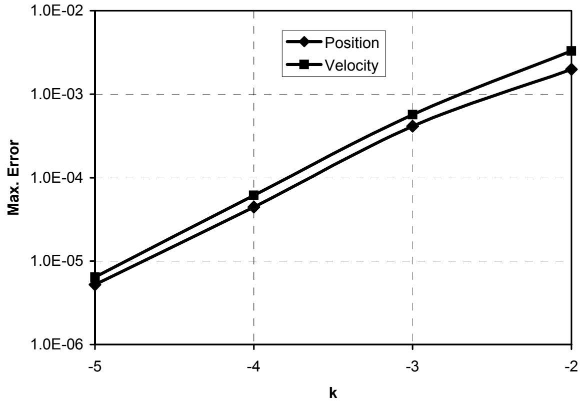


Figure 5. Error Analysis Results for *InflTrap*

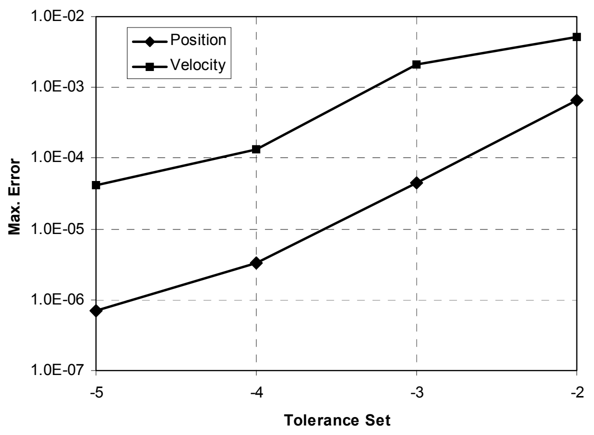


Figure 6. Error Analysis Results for *InflSDIRK*

## 6.3 Comparison of Efficiency

In this section, the algorithms proposed are compared in terms of efficiency. Four error tolerances are considered, ranging from $10^{-2}$ to $10^{-5}$, and CPU times are recorded for simulation time periods ranging from 1 s to 4 s. The same absolute and relative tolerances are considered, for both position and velocity. Timing results reported in Tables 8 and 9 are in CPU seconds. The results are obtained on an SGI Onyx computer with R10000 processors.

Table 8. Timing Results *InflTrap*

| *TOL* | $10^{-2}$ | $10^{-3}$ | $10^{-4}$ | $10^{-5}$ |
|---|---|---|---|---|
| 1 s | 42 | 61 | 158 | 463 |
| 2 s | 79 | 155 | 420 | 1198 |
| 3 s | 92 | 189 | 524 | 1521 |
| 4 s | 100 | 206 | 552 | 1568 |

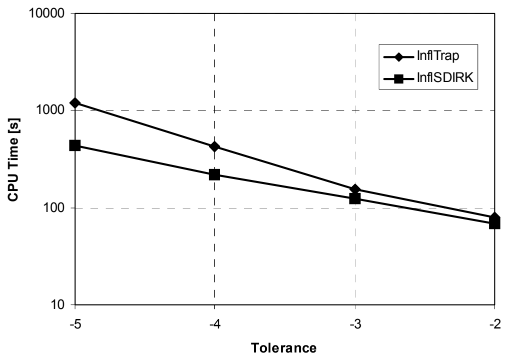
Table 9. Timing results *InflSDIRK*

| *TOL* | $10^{-2}$ | $10^{-3}$ | $10^{-4}$ | $10^{-5}$ |
|---|---|---|---|---|
| 1 s | 33 | 52 | 90 | 170 |
| 2 s | 69 | 124 | 218 | 433 |
| 3 s | 81 | 150 | 248 | 493 |
| 4 s | 84 | 155 | 256 | 500 |

When compared to *InflTrap*, the algorithm *InflSDIRK* is more efficient, because of its better integration formula. The step-size control mechanism for the SDIRK formula is well designed, and the simulation takes larger step-sizes. This results in fewer costly matrix factorizations. Furthermore, the order of the formula is 4, and its stability properties are very good (L-stable). These attributes strongly recommend this algorithm, especially when very stiff models are integrated with medium to high accuracy requirements. For low accuracy, the difference between the algorithms is not significant.

To better see the impact of error tolerance on efficiency, in Fig. 7 is shown the CPU time necessary for each of the algorithms to complete a 2 second simulation. The same model and the same simulation conditions are considered. The absolute and relative tolerances are identical, assuming values between $10^{-2}$ and $10^{-5}$. The same accuracy is imposed at position and velocity levels. The slope of the timing curve for *InflTrap* is smaller than the slope of *InflSDIRK*. This is a consequence of the different orders of the integration formulas used in these algorithms. The results confirm the recommendation made earlier that a higher order formula should be used when more stringent accuracy is imposed on the simulation.

Finally, in Fig. 8 is shown the variation of the integration step-size, for a four second simulation. The error tolerance for this simulation is $10^{-3}$. As noted earlier, at $T = 0.5$ and $T = 1.2$ the front and rear wheels hit the obstacle, respectively. The error control mechanism senses the strong excitation induced in the system as a result of these events, and the step-size is decreased to ensure that accuracy requirements are met. Once the vehicle clears the bump, transients damp out and the error controller increases the step-size. Note that step-sizes for *InflSDIRK* are substantially larger than for *InflTrap.*



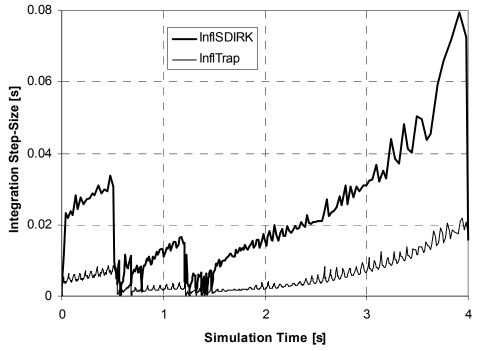


Figure 7. CPU Time vs. Tolerance

Figure 8. Integration Step-Size History

## 6.4 Implicit versus Explicit Integration Comparison

In order to see the impact of the proposed algorithms on simulation timing results, a comparison with an explicit algorithm is done in terms of efficiency. The explicit algorithm used is denoted *ExplDEABM*. It is based on the code DEABM from the suite DEPAC of explicit integrators of Shampine and Gordon (1975). *ExplDEABM* is an algorithm based on a state-space reduction method, which uses the code DDEABM to integrate independent accelerations to obtain independent velocities and positions. Dependent positions and velocities are recovered using the position and velocity kinematic constraint equations of Eqs. (1) and (2), as indicated in Eqs. (53) and (54), respectively. Efficient means for acceleration computation are used in *ExplDEABM*, as proposed by Serban, Negrut, Haug, and Potra (1997).

Table 6 presents simulation timing results in CPU seconds for *ExplDEABM*, which can be compared with the results in Tables 8 and 9. The first row contains the absolute and relative error tolerances imposed on both positions and velocities. The first column contains simulation lengths in seconds.

Table 6. Timing Results for *ExplDEABM*

| *TOL* | $10^{-2}$ | $10^{-3}$ | $10^{-4}$ | $10^{-5}$ |
|---|---|---|---|---|
| 1 s | 3618 | 3641 | 3667 | 3663 |
| 2 s | 7276 | 7348 | 7287 | 7276 |
| 3 s | 10865 | 11122 | 10949 | 10965 |
| 4 s | 14480 | 14771 | 14630 | 14592 |

The explicit algorithm requires significantly larger CPU times to complete simulations, independent of simulation duration and error tolerances. Results obtained with *ExplDEABM* also conform to theoretical predictions. For each imposed error tolerance, the algorithm required the same CPU time to complete a simulation run. This is typical of situations in which explicit algorithms are limited to small step-sizes by stability considerations when integrating stiff systems. Furthermore, for an imposed tolerance, the CPU time increases linearly with the simulation length. Even after the vehicle clears the obstacle and there is no significant source of external excitation, the explicit algorithm is constrained to take very small steps, because otherwise it would become unstable.

When compared to Figs. 8 and 9, the results obtained with *ExplDEABM* displayed in Figs. 10 and 11, support the observations made above. The plot in Fig. 9 contains CPU times in seconds for the case in which a two second simulation is run with tolerances between $10^{-2}$ and $10^{-5}$, whereas Fig. 10 displays CPU times obtained running 1 to 4 second simulations, with error tolerances (absolute and relative) set to $10^{-3}$ for both positions and velocities

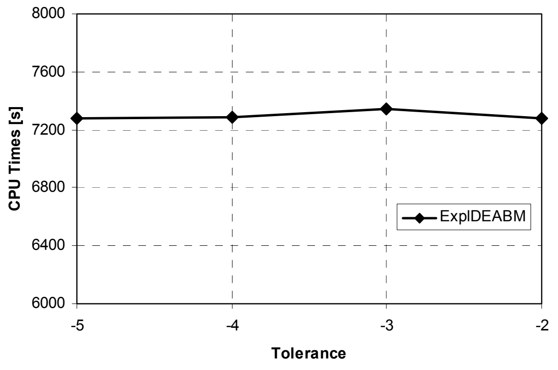


Figure 9. Different Tolerances

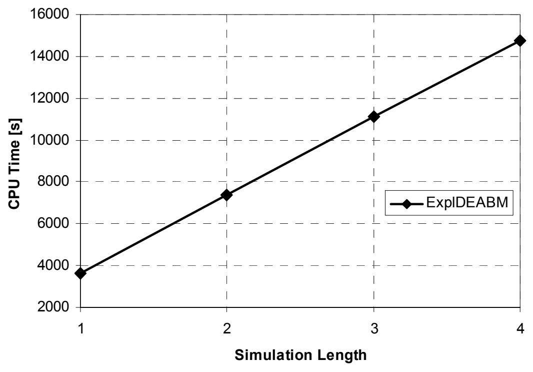


Figure 10. Different Simulation Lengths

## 7. Conclusions

The descriptor form implicit integration method proposed for the solution of stiff differential-algebraic equations of multibody dynamics is shown to be reliable and efficient. Compared to previously used explicit integrators, a speed-up of approximately two orders of magnitude is obtained. The algorithm, implemented with an order four, five stage, L-stable, stiffly accurate SDIRK formula, is shown to be superior to the algorithm based on the trapezoidal formula.

Stopping criteria and sparse factorization of the integration Jacobian are issues that must be addressed in the future to improve reliability and efficiency of the proposed method. Efficiency of the algorithms is expected to further improve once the current conservative error controller is adapted to use scaled local truncation errors for step-size selection.

## References

Atkinson, K. E., An Introduction to Numerical Analysis, New York: John Wiley & Sons, 2nd Edition, 1989

Corwin, L., J., Szczarba, R., H., Multivariable Calculus, New York: Marcel Dekker, 1982

Hairer, E., Nørsett S., P., Wanner, G., Solving Ordinary Differential Equations I. Nonstiff Problems, Berlin Heidelberg New York: Springer-Verlag, 1993

Hairer, E., Wanner, G., Solving Ordinary Differential Equations II. Stiff and Differential-Algebraic Problems, Berlin Heidelberg New York: Springer-Verlag, 1996

Harwell subroutine library – Specifications, AEATechology, Harwell Laboratory, Oxfordshire, England, 1995

Haug, E., J., Computer-Aided Kinematics and Dynamics of Mechanical Systems. Boston, London, Sydney, Toronto: Allyn and Bacon, 1989

Haug, E., J., Iancu, M., Negrut, D., "Implicit Integration of the Equations of Multibody Dynamics in Descriptor Form," in Advances in Design Automation-Proceedings of the ASME Design Automation Conference, Sacramento, 1997

Haug E., J., Negrut, D., Iancu, M., "A State-Space Based Implicit Integration Algorithm for Differential-Algebraic Equations of Multibody Dynamics," *Mech. Struct.&Mach.*, vol. 25(3), pp. 311-334, 1997

Hughes, T., J., The Finite Element Method, Prentice-Hall, Englewood Cliffs, New Jersey, 1987

Negrut, D., "On the Implicit Integration of Differential-Algebraic Equations of Multibody Dynamics", Ph.D. Thesis, The University of Iowa, 1998

Petzold, L., R., "Differential/Algebraic Equations are not ODE's", *SIAM Journal of Scientific and Statistical Computing* 3(3), pp. 367-384, 1982

Serban, R., and Haug, E.J., "Kinematic and Kinetic Derivatives in Multibody System Analysis," Mechanics of Structures and Machines, Vol. 26, No. 2, pp. 145-173, 1998.

Serban, R., Negrut, D., Haug, E. J., Potra, F. A., "A Topology Based Approach for Exploiting Sparsity in Multibody Dynamics in Cartesian Formulation," *Mech. Struct.&Mach.*, vol. 25(3), pp. 379-396, 1997

Shampine, L. ,F., Numerical Solution of Ordinary Differential Equations, Chapmann & Hall, New York, 1994

Shampine, L., F., Gordon, M., K., Computer Solution of Ordinary Differential Equations. The Initial Value Problem, Freeman and Company, San Francisco, 1975

Shampine, L. F., Watts, H. A., “The art of writing a Runge-Kutta code.II,” *Appl. Math. Comput.*, vol. 5, pp. 93-121, 1979