P. Masarati, M. Morandini, P. Mantegazza
An Efficient Formulation for General-Purpose Multibody/Multiphysics Analysis
Journal of Computational and Nonlinear Dynamics, Vol. 9, N. 4, 2014, 041001 (9 pages)
doi:10.1115/1.4025628

An Efficient Formulation for General-Purpose Multibody/Multiphysics Analysis

Pierangelo Masarati*, Marco Morandini, Paolo Mantegazza
Politecnico di Milano, Dipartimento di Scienze e Tecnologie Aerospaziali
mail: {pierangelo.masarati,marco.morandini,paolo.mantegazza}@polimi.it

This paper presents a formulation for the efficient solution of general-purpose multibody/multiphysics problems. The formulation has been implemented in MBDyn, a free general-purpose multibody solver continuously developed over several years at the Department of Aerospace Engineering, Politecnico di Milano. The core equations and details on structural dynamics and finite rotations handling are presented. The solution phases are illustrated. Highlights of the implementation structure are presented, and special features are discussed.

1 INTRODUCTION

Multibody dynamics originated from the need to analyze complex, multidisciplinary dynamics problems. Several formulations have been proposed, derived from classical approaches to analytical mechanics. After the pioneering development of the 1960s and 1970s, pushed by space race needs and subsequently by a rapid spread into the aerospace, robotics, mechanical and specifically automotive industry, multibody dynamics became an industrial tool of general use.

This work does not have the ambition to present a detailed history of the development of multibody dynamics; for this purpose, the interested reader should consider for example [1, 2] or [3]. It is intended to discuss a general-purpose formulation developed at the Department of Aerospace Engineering of Politecnico di Milano and implemented in the free solver MBDyn[1]. This solver originated from academia, like many others (for example DYMORE [4], CHRONO::Engine[2] [5], and HOTINT[3] [6]), and has grown to maturity as a full-featured research tool.

The project started in the early 1990s as a research tool for students to familiarize with the dynamics of systems of constrained rigid bodies and their numerical integration. Mechanical problems were initially formulated as systems of mixed algebraic and first- and second-order differential equations exploiting the redundant coordinates approach, which expressed the Newton-Euler equations of motion, with constraints enforced using Lagrange multipliers. The initial implementation was in Fortran 77, and made limited use of external libraries for basic operations like linear algebra on sparse matrices.

Later on, the need to model significantly more complex multidisciplinary problems led to a substantial rewriting of the formulation, expressing all differential equations in first-order form. Starting from 1996, the implementation was substantially refactored, moving to C++ (despite the language and free compilers being in an early stage of maturity at that time), to extensively exploit the inherent abstraction of object-oriented programming.

In 2001, MBDyn was formally released as free software under GNU's General Public License[4]. Over the years it has been continuously maintained and developed by the core developers at Politecnico di Milano, with significant contributions coming from independent users, thanks to the open development paradigm granted by the choice to "go free". Development addressed several areas: parallelization, real-time simulation, co-simulation ranging from loose interaction with block diagram solvers including multirate integration strategies to tight coupling with computational fluid-dynamics. Development was, and is, often driven by applied research needs, which are mainly related but not limited to the aerospace field.

The core formulation is described in Section 2. Highlights of the implementation are discussed in Section 3. The core problem solver is described in Section 4. Special features are described in Section 5 and strengths and weaknesses are discussed in Section 6.

*Corresponding author: Politecnico di Milano, Dipartimento di Scienze e Tecnologie Aerospaziali, via La Masa 34, 20156 Milano, Italy

[1] http://www.mbdyn.org/

[2] http://www.chronoengine.info/

[3] http://www.hotint.org/

[4] http://www.gnu.org/licenses/licenses.html

ASME J. Comput. Nonlinear Dyn. MONTH YYYY V(N), doi:10.1115/1.4025628

2 FORMULATION

A generic dynamical problem is formulated as a system of implicit Differential-Algebraic Equations (DAE),

$$\mathbf{r}(\mathbf{y}, \dot{\mathbf{y}}, t) = \mathbf{0} \tag{1}$$

integrated in time using implicit A/L stable linear multistep integration schemes,

$$\mathbf{y}_k = \sum_{i=1,2} a_i \mathbf{y}_{k-i} + h \sum_{i=0,2} b_i \dot{\mathbf{y}}_{k-i}. \tag{2}$$

A prediction-correction approach is used. The state $\mathbf{y}_k^{(0)}$ is predicted using Eq. (2), based on an extrapolation of its derivative $\dot{\mathbf{y}}_k^{(0)}$. The perturbation of Eq. (2),

$$\partial \mathbf{y}_k = h b_0 \partial \dot{\mathbf{y}}_k, \tag{3}$$

is used in the iterative solution of the correction phase,

$$\left(h b_0 \mathbf{r}_{/\mathbf{y}} + \mathbf{r}_{/\dot{\mathbf{y}}}\right) \partial \dot{\mathbf{y}} = -\mathbf{r}(\mathbf{y}_k^{(j)}, \dot{\mathbf{y}}_k^{(j)}, t_k) \tag{4a}$$

$$\dot{\mathbf{y}}_k^{(j+1)} \mathrel{+}= \partial \dot{\mathbf{y}} \tag{4b}$$

$$\mathbf{y}_k^{(j+1)} \mathrel{+}= h b_0 \partial \dot{\mathbf{y}} \tag{4c}$$

(operator $(\cdot)_{/\mathbf{y}}$ indicates partial derivative with respect to $\mathbf{y}$; operator $+=$, increment-assignment, is mutuated from the C/C++ programming languages).

A model is logically formulated in terms of nodes and elements, resembling the structure of Finite Elements. Nodes contribute to "public" entries of the state $\mathbf{y}$; elements contribute to equations, i.e. provide $\mathbf{r}$ (residual) and $h b_0 \mathbf{r}_{/\mathbf{y}} + \mathbf{r}_{/\dot{\mathbf{y}}}$ (Jacobian matrix). Elements may require, and thus instantiate, "private" entries of $\mathbf{y}$ (for example, elements implementing kinematic constraints require Lagrange multipliers) and write the corresponding contributions to equations.

Specialized scalar nodes are used for discrete scalar fields (temperature, pressure, electric voltage) and for abstract states. Structural nodes, described in the next Section, originally provided both displacement and rotation in space; displacement-only nodes have been recently added, mainly to support membrane elements. They will be used to investigate absolute nodal coordinate formulations.

2.1 Structural Dynamics

Structural dynamics are dealt with using specialized nodes. Both *static* and *dynamic structural nodes* create the corresponding force (and moment) equilibrium equations. Dynamic nodes also include momentum (and momenta moment) in the state, and create equations for momentum (and momenta moment) definition. The displacement field is discretized using *structural displacement nodes*. Their state is the position of the node, $\mathbf{x}$; that of dynamic nodes also includes the momentum $\boldsymbol{\beta}$. The rotation field is discretized using *structural* nodes, which inherit from the displacement-only nodes and add rotation-related variables and methods. Their state is specially dealt with by considering the rotation matrix $\mathbf{R}$ in incremental manner, which is updated using the (incremental) Cayley-Gibbs-Rodriguez (CGR) [7] vector rotation parameters $\mathbf{g}$ in the state vector $\mathbf{y}$. The momenta moment is also present in the state of dynamic nodes. The handling of rotations is a key aspect in the formulation; it is discussed in detail in Section 2.2.

Each dynamic node adds to vector $\mathbf{r}$ the definitions of momentum and momenta moment, namely

$$m\dot{\mathbf{x}} + \boldsymbol{\omega} \times \mathbf{s} = \boldsymbol{\beta} \tag{5a}$$

$$\mathbf{s} \times \dot{\mathbf{x}} + \mathbf{J}\boldsymbol{\omega} = \boldsymbol{\gamma} \tag{5b}$$

using the mass m, the static moment $\mathbf{s} = m\mathbf{b}_{\mathrm{CM}}$ ($\mathbf{b}_{\mathrm{CM}}$ being the center of mass offset), and the inertia moment with respect to the node, $\mathbf{J} = \mathbf{J}_{\mathrm{CM}} + m\mathbf{b}_{\mathrm{CM}} \times \mathbf{b}_{\mathrm{CM}} \times^T$. The inertia contributions are written in Eqs. (5) by those elements that contribute to the inertia of the node (e.g. point masses, rigid bodies). The handling of the angular velocity, $\boldsymbol{\omega}$, is discussed in Section 2.2.

Moreover, each structural node adds the force and moment equilibrium equations to vector $\mathbf{r}$. They are formulated according to the Virtual Work Principle (VWP); the contribution of moment $\mathbf{m}_i$ and force $\mathbf{f}_i$ for the virtual rotation $\boldsymbol{\theta}_{\delta i}$ and displacement $\delta\mathbf{x}_i$ of point i, in $\mathbf{x}_i = \mathbf{x} + \mathbf{b}_i$, associated with that of the corresponding node, $\boldsymbol{\theta}_\delta$ and $\delta\mathbf{x}$, namely $\boldsymbol{\theta}_{\delta i} = \boldsymbol{\theta}_\delta$ and $\delta\mathbf{x}_i = \delta\mathbf{x} + \boldsymbol{\theta}_\delta \times \mathbf{b}_i$, is

$$\begin{aligned} \delta\mathcal{W} &= \delta\mathbf{x}_i^T \mathbf{f}_i + \boldsymbol{\theta}_{\delta i}^T \mathbf{m}_i \\ &= \delta\mathbf{x}^T \mathbf{f}_i + \boldsymbol{\theta}_\delta^T (\mathbf{b}_i \times \mathbf{f}_i + \mathbf{m}_i) \end{aligned} \tag{6}$$

where $\boldsymbol{\theta}_\delta = \mathrm{ax}(\delta\mathbf{R}\mathbf{R}^T)$ is the node virtual rotation; then

$$\dot{\boldsymbol{\beta}} = \sum_i \mathbf{f}_i \tag{7a}$$

$$\dot{\boldsymbol{\gamma}} + \dot{\mathbf{x}} \times \boldsymbol{\beta} = \sum_i (\mathbf{b}_i \times \mathbf{f}_i + \mathbf{m}_i). \tag{7b}$$

The generic forces and moments $\mathbf{f}_i$ and $\mathbf{m}_i$ may depend on the states of the system (for example in the case of elastic forces); $\mathbf{b}_i$ is the distance between the node and the point of application of force $\mathbf{f}_i$. Configuration-dependent forces and moments include contributions associated with deformable components, like lumped linear and angular springs (including an original spring element with the unique feature of being truly independent from the ordering of the connectivity, unlike other known implementations, see [8, 9] for details), an original finite volume formulation for geometrically exact beams [10], nonlinear finite element shells [11], and Component Mode Synthesis (CMS) elements. Unlike the previously mentioned elements, membranes [12] may use displacement-only nodes. Displacement-only nodes do not create Eqs. (5b)

ASME J. Comput. Nonlinear Dyn. MONTH YYYY V(N), doi:10.1115/1.4025628

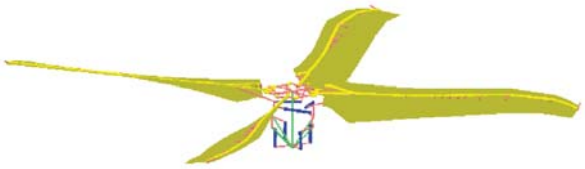

Fig. 1. Tiltrotor rotor model.

and (7b). The left-hand side of Eqs. (7) is zero for static nodes. Placing momentum and momenta moment in the state confines the contribution of inertia-related elements into Eqs. (5); Eqs. (7) are unaffected.

The reason for choosing Newton-Euler equations written through the VWP instead of other, perhaps more elegant, formalisms, is motivated by the need to accommodate rather different problems in a monolithic solver. Essential configuration-dependent non-conservative generalized forces (e.g. those originating from aerodynamics) contribute to the Jacobian matrix in an intrinsically non-symmetric manner, thus breaking any possibility to exploit noteworthy structures of conservative mechanical problems.

2.2 Handling of Rotations

Rotations are handled in incremental manner. The orientation of each node is stored in an orientation matrix, $\mathbf{R}$. During the prediction phase from time t_{k-1} to t_k, the relative CGR orientation parameters $\mathbf{g}$ and their time derivatives are computed from the relative orientation between times t_{k-2} and t_{k-1} and the corresponding angular velocities. The predicted value of the orientation parameters is used to predict the orientation matrix at time t_k as $\mathbf{R}_k^{(0)}$. During the correction phase, the orientation of the node is expressed as $\mathbf{R}_k = \mathbf{R}_\Delta \mathbf{R}_k^{(0)}$, with $\mathbf{R}_\Delta = \mathbf{R}(\mathbf{g}_\Delta)$ and

$$\mathbf{R}(\mathbf{g}) = \mathbf{I} + \frac{4}{4+\mathbf{g}^T\mathbf{g}}\left(\mathbf{g}\times + \frac{1}{2}\mathbf{g}\times\mathbf{g}\times\right). \qquad (8)$$

As a consequence, the rotation associated with $\mathbf{R}_\Delta$ is of the order of magnitude of the *correction* parameters, which in principle is $o(h^N)$, where N is the order of accuracy of the multistep method ($N = 2$ in the present case).

This approach has been termed *updated-updated*, since it can be interpreted as an updated Lagrangian approach in which the metrics of the problem are referred to the *predicted* orientation for the current time step rather than to the *converged* orientation for the previous time step. The advantage of referring orientations to the predicted value may not be apparent; consider the case of systems with non-negligible reference angular velocity, $\mathbf{\Omega}$ (from the beginning, the driving application of MBDyn has been the aeroelasticity of helicopter rotors [13–15]; for example, Figure 1 shows a detailed aeroelastic model of the rotor of a tiltrotor). In such case, the order of magnitude of the relative orientation parameters referred to the *converged* orientation for the previous time step would be $O(\|\mathbf{\Omega}\|h)$. For accuracy reasons, typical values of time step h are of the order of 1/100 of the rotation period, T; as a consequence, $\|\boldsymbol{\omega}\|h \approx 2\pi/100 \cong 0.06$, a number that can be much larger than h^3, the order of the correction.

The angular velocity, $\boldsymbol{\omega} = \mathrm{ax}(\dot{\mathbf{R}}\mathbf{R}^T) = \mathbf{G}(\mathbf{g})\dot{\mathbf{g}}$, is

$$\boldsymbol{\omega}_k = \boldsymbol{\omega}_\Delta + \mathbf{R}_\Delta \boldsymbol{\omega}_k^{(0)}, \qquad (9)$$

with

$$\mathbf{G}(\mathbf{g}) = \frac{4}{4+\mathbf{g}^T\mathbf{g}}\left(\mathbf{I} + \frac{1}{2}\mathbf{g}\times\right), \qquad (10)$$

and $\boldsymbol{\omega}_\Delta = \mathbf{G}(\mathbf{g}_\Delta)\dot{\mathbf{g}}_\Delta$; $\boldsymbol{\omega}_k^{(0)}$ is the predicted angular velocity.

Consider now the perturbations of the rotation-related entities, which are required for the computation of the Jacobian matrices $\mathbf{r}_{/\mathbf{y}}$ and $\mathbf{r}_{/\dot{\mathbf{y}}}$. The perturbation of the orientation matrix yields

$$\mathbf{G}(\mathbf{g}_\Delta)\partial\mathbf{g}_\Delta = \mathrm{ax}(\partial\mathbf{R}_k\mathbf{R}_k^T) = \mathrm{ax}(\partial\mathbf{R}_\Delta\mathbf{R}_\Delta^T). \qquad (11)$$

Considering that $\mathbf{g}_\Delta$ is $o(h^2)$, higher-order terms can be neglected when the Jacobian matrices are computed. This simplification is highlighted in the formulas using the operator $\overset{\text{uu}}{=}$, which indicates an equality approximated thanks to the updated-updated approach assumptions. For example, $\partial\mathbf{R}_k \overset{\text{uu}}{=} \partial\mathbf{g}_\Delta \times \mathbf{R}_k^{(0)}$, since both $\mathbf{R}_\Delta \overset{\text{uu}}{=} \mathbf{I}$ and $\mathbf{G}(\mathbf{g}_\Delta) \overset{\text{uu}}{=} \mathbf{I}$. Similarly, $\partial\boldsymbol{\omega}_k \overset{\text{uu}}{=} \partial\dot{\mathbf{g}}_\Delta - \boldsymbol{\omega}_k^{(0)} \times \partial\mathbf{g}_\Delta$, since $\partial(\mathbf{G}(\mathbf{g}_\Delta)\dot{\mathbf{g}}_\Delta) = \mathbf{H}(\mathbf{g}_\Delta,\dot{\mathbf{g}}_\Delta)\partial\mathbf{g}_\Delta + \mathbf{G}(\mathbf{g}_\Delta)\partial\dot{\mathbf{g}}_\Delta$, and $\mathbf{H}(\mathbf{g}_\Delta,\dot{\mathbf{g}}_\Delta) \overset{\text{uu}}{=} \mathbf{0}$ (operator $\mathbf{H}$ is defined for example in [16,17]). Furthermore, considering Eq. (3), $\partial\mathbf{g}_\Delta = hb_0\partial\dot{\mathbf{g}}_\Delta$ and thus $\partial\boldsymbol{\omega}_k \overset{\text{uu}}{=} (\mathbf{I} - hb_0\boldsymbol{\omega}_k^{(0)}\times)\partial\dot{\mathbf{g}}_\Delta$. The updated-updated simplifications are used only for the computation of contributions to the Jacobian matrices; the residual is computed using the exact formulas.

Solution prediction at time t_k is performed setting $\mathbf{g}_{k-1} = \mathbf{0}$ and $\dot{\mathbf{g}}_{k-1} = \boldsymbol{\omega}_{k-1}$, since $\mathbf{G}(\mathbf{g}_{k-1}) = \mathbf{G}(\mathbf{0}) \equiv \mathbf{I}$. Then $\mathbf{g}_{k-2} = \mathbf{g}(\mathbf{R}_{k-2}\mathbf{R}_{k-1}^T)$ and $\dot{\mathbf{g}}_{k-2} = \mathbf{G}^{-1}(\mathbf{g}_{k-2})\boldsymbol{\omega}_{k-2}$. The backward relative orientation $\mathbf{R}_{k-2}\mathbf{R}_{k-1}^T$ must be limited (formally, less than $\pi/2$ radian to avoid the singularity of $\mathbf{G}^{-1}$, but practically much less); this assumption holds in practice as long as the amount of rotation between two time steps is limited by accuracy considerations. The reference orientation is computed using the predicted $\mathbf{g}_k^{(0)}$, $\mathbf{R}_k^{(0)} = \mathbf{R}(\mathbf{g}_k^{(0)})\mathbf{R}_{k-1}$, as well as $\boldsymbol{\omega}_k^{(0)} = \mathbf{G}(\mathbf{g}_k^{(0)})\dot{\mathbf{g}}_k^{(0)}$. At this point, the orientation parameters and their time derivatives are reset again, as for the correction their updated-updated counterparts, $\mathbf{g}_\Delta$, are used.

Moment equilibrium equations $\sum\mathbf{m} = \mathbf{0}$ are energetically conjugated with virtual rotations, $\boldsymbol{\theta}_\delta = \mathbf{G}_\Delta(\mathbf{g}_\Delta)\delta\mathbf{g}_\Delta$. By producing equations energetically conjugated with the

ASME J. Comput. Nonlinear Dyn. MONTH YYYY V(N), doi:10.1115/1.4025628

virtual perturbation of the rotation parameters, $\delta\mathbf{g}_\Delta$, one would need to write moment equilibrium as $\boldsymbol{\theta}_\delta^T \sum \mathbf{m} = \delta\mathbf{g}_\Delta^T \mathbf{G}_\Delta^T(\mathbf{g}_\Delta) \sum \mathbf{m}$. The linearization of the right-hand side would require the linearization of $\mathbf{G}(\mathbf{g})$ as well. By directly considering moment equilibrium, this is avoided at the cost of breaking the symmetry of the matrices, which is lost in any case when non-conservative loads and general multidisciplinary terms, e.g. hydraulic elements, are present.

3 IMPLEMENTATION STRUCTURE

The code is mainly written in C++; the main classes deserve a mention. The `DataManager` class handles the model. It stores all nodes and elements, and provides the methods `AssRes()` and `AssJac()` that respectively assemble the residual vector $\mathbf{r}$ and the Jacobian matrix $(c\mathbf{r}_{/\mathbf{y}} + \mathbf{r}_{/\dot{\mathbf{y}}})$. The class `ImplicitStepIntegrator` defines the integration scheme. Its method `Advance()` drives the integration from a time step to the next one. Through the `Residual()` and `Jacobian()` methods it provides access to the `DataManager`'s `AssRes()` and `AssJac()`. The base class `NonlinearSolver`, through its method `Solve()`, defines the interface that must be provided by the iterative nonlinear problem solution methods, such as the `NewtonRaphsonSolver` or the `Gmres` matrix free solver. To actually solve the problem, a `NonlinearSolver` requires a pointer to a (linear) `SolutionManager` class that abstracts the concept solving a linear system of equations. For direct solvers, this class actually stores the (often sparse) matrix and makes use of appropriate `LinearSolver` subclasses, i.e. the classes that actually encapsulate one of the many linear solver libraries supported by MBDyn. Significant examples are UMFPACK [18], KLU [19], and a dense storage sparse solver specialized for relatively small problems [20]. Each linear solver requires a specific storage scheme for the Jacobian matrix. For this reason the `SparseMatrixHandler` base class is in charge of abstracting the concept of sparse matrix, with optimizations aiming at saving the matrix compaction process cost across subsequent assemblies when the sparsity pattern does not change, and supporting multi-thread assembly on Symmetric Multi-Processor (SMP) architectures [21].

All the required classes are owned by a `Solver` class that drives the solution of each specific problem following the scheme sketched in Fig. 2. The *initial value* problem solver is discussed in detail in the next Section, along with its *real-time* variant, whereas the *inverse dynamics* solver is briefly discussed in Section 5.3.

For isolation reasons, elements do not directly access the Jacobian matrix (which, in principle, may not exist) nor the residual vector. Elements act on temporary storage (either dense or sparse) which are later assembled in the global storage. This can also occur in a concurrent manner exploiting multi-threading on SMP architectures.

```
Solver S;
  stores:
    DataManager PDM;
    ImplicitStepIntegrator ISI;
    SolutionManager SM;
      stores:
        SparseMatrix SPM;
        LinearSolver LS;
    NonlinearSolver NS;
  calls: ISI.Advance();
    calls: NS.Solve();
      calls:
        ISI.Residual();
          calls: DM.AssRes();
        ISI.Jacobian();
          calls: DM.AssJac(SM.SPM);
        SM.Solve();
          calls: LS.Solve(SM.SPM);
```

Fig. 2. Solution call graph

Facilities are provided in support to common functionalities. Constitutive laws are abstracted and generalized, exploiting C++ templates to provide similar laws for different dimensionalities (e.g. scalar for unidimensional viscoelastic elements, three-dimensional for linear and angular ones, six-dimensional for beams).

Generic parameter-dependent functions are abstracted in *drives*, either scalar or multi-dimensional. The default parameter is the current time. Drives are used for example as the multipliers of external forces, the prescribed motion in rheonomic constraints, the prestrain in constitutive laws, and more. Drives can be pipelined to produce complex functions of functions.

The model is described in terms of nodes and elements through a textual input file that is parsed at the beginning of the analysis. The parser embeds a mathematical expression evaluator that provides simple yet extensible mathematical capabilities, including strongly typed variable declaration. Fully parametric, hierarchical models can be defined using mathematical expression evaluation. The possibility to hierarchically define reference systems greatly supports the construction of kinematically consistent models. Repetitive models can be generated by duplicating the definition of the repeated components. Several types of entities, including constitutive laws and drives, can be declared and reused throughout the model.

4 PROBLEMS AND SOLUTION PHASES

Different types of analyses can be performed. The main one solves an Initial Value Problem (IVP), either static or dynamic. All analyses require a model to operate on; the model is encapsulated in a `DataManager` object.

ASME J. Comput. Nonlinear Dyn. MONTH YYYY V(N), doi:10.1115/1.4025628

The constrained mechanics IVP is formulated as

$$\mathbf{M}\dot{\mathbf{x}} - \boldsymbol{\beta} = \mathbf{0} \tag{12a}$$

$$\dot{\boldsymbol{\beta}} + \boldsymbol{\phi}_{/\mathbf{x}}^T \boldsymbol{\lambda}_\phi + \boldsymbol{\psi}_{/\dot{\mathbf{x}}}^T \boldsymbol{\lambda}_\psi = \mathbf{f}(\mathbf{x}, \dot{\mathbf{x}}, t) \tag{12b}$$

$$\boldsymbol{\phi}(\mathbf{x}, t) = \mathbf{0} \tag{12c}$$

$$\boldsymbol{\psi}(\mathbf{x}, \dot{\mathbf{x}}, t) = \mathbf{0}, \tag{12d}$$

where vector $\mathbf{f}$ contains generic forces and moments, possibly dependent on the configuration (e.g. produced by viscoelastic elements), and $\boldsymbol{\lambda}_\phi$, $\boldsymbol{\lambda}_\psi$ are the Lagrange multipliers associated with holonomic and non-holonomic constraint equations. Its linearization, required for the Newton-like iterative solution process, yields

$$\begin{bmatrix} \mathbf{M} & -hb_0\mathbf{I} & \mathbf{0} & \mathbf{0} \\ -(hb_0\mathbf{f}_{/\mathbf{x}} + \mathbf{f}_{/\dot{\mathbf{x}}}) & \mathbf{I} & hb_0\boldsymbol{\phi}_{/\mathbf{x}}^T & hb_0\boldsymbol{\psi}_{/\dot{\mathbf{x}}}^T \\ hb_0\boldsymbol{\phi}_{/\mathbf{x}} & \mathbf{0} & \mathbf{0} & \mathbf{0} \\ \boldsymbol{\psi}_{/\dot{\mathbf{x}}} & \mathbf{0} & \mathbf{0} & \mathbf{0} \end{bmatrix} \begin{Bmatrix} \partial\dot{\mathbf{x}} \\ \partial\dot{\boldsymbol{\beta}} \\ \partial\dot{\boldsymbol{\lambda}}_\phi \\ \partial\dot{\boldsymbol{\lambda}}_\psi \end{Bmatrix} = \text{RHS} \tag{13}$$

In the symbolic representation of Eq. (13) terms like $(\mathbf{M}\dot{\mathbf{x}})_{/\mathbf{x}}\partial\mathbf{x}$, related to the possible dependence of the mass matrix $\mathbf{M}$ on the configuration $\mathbf{x}$, have been omitted for clarity, although they are present in the implementation. In order to improve the scaling of the matrix, the linearized problem is actually reformulated as

$$\begin{bmatrix} \mathbf{M} & -hb_0\mathbf{I} & \mathbf{0} & \mathbf{0} \\ -(hb_0\mathbf{f}_{/\mathbf{x}} + \mathbf{f}_{/\dot{\mathbf{x}}}) & \mathbf{I} & \boldsymbol{\phi}_{/\mathbf{x}}^T & \boldsymbol{\psi}_{/\dot{\mathbf{x}}}^T \\ \boldsymbol{\phi}_{/\mathbf{x}} & \mathbf{0} & \mathbf{0} & \mathbf{0} \\ \boldsymbol{\psi}_{/\dot{\mathbf{x}}} & \mathbf{0} & \mathbf{0} & \mathbf{0} \end{bmatrix} \begin{Bmatrix} \partial\dot{\mathbf{x}} \\ \partial\dot{\boldsymbol{\beta}} \\ \partial\dot{\boldsymbol{\lambda}}_\phi \\ \partial\dot{\boldsymbol{\lambda}}_\psi \end{Bmatrix} = \text{RHS}', \tag{14}$$

where the equations that express holonomic constraints are divided by hb_0, as suggested for example in [22]. Moreover, the integrals of the Lagrange multipliers, which do not explicitly appear in the equations, are actually used as primary unknowns. As a consequence, hb_0 is removed from the last two columns of the matrix.

The `DataManager` directly produces the matrix and the right-hand side of Eq. (14) when the `AssJac()` and `AssRes()` methods are called.

Each analysis may need to perform specific solution phases. Typical solution phases are:

1. initial assembly: ensure consistency of model's state;
2. derivatives: compute the derivatives of the state;
3. first step: use a self-starting algorithm;
4. subsequent steps.

In most cases, the first solution phase, called *initial assembly*, is dedicated to checking the consistency of the model as input by the user. It is dealt with internally by the data manager to ensure that the initial state of the system complies with the algebraic equations (e.g. the kinematic constraints). The remaining phases are dealt with by specialized solvers for each type of problem. In the following, the phases associated with IVP are discussed; other problem types are discussed in Section 5.

4.1 Initial Assembly

Kinematic constraint equations that express holonomic constraints are collected in vector $\boldsymbol{\phi}(\mathbf{x}, t) = \mathbf{0}$. They need to be satisfied up to the first derivative. The equations that express non-holonomic constraints are collected in vector $\boldsymbol{\psi}(\dot{\mathbf{x}}, \mathbf{x}, t) = \mathbf{0}$. The constraint equations need to be satisfied by the initial position and velocity $\mathbf{x}_0$, $\dot{\mathbf{x}}_0 = \mathbf{v}_0$. This requires that consistent initial conditions be provided from the beginning. However, in many cases this is not practical; an automatic procedure is needed to restore consistency by determining a relaxed initial configuration. For this purpose, a dedicated procedure is envisaged, in which the constraint equations are enforced and, at the same time, the departure from the initial configuration is penalized by matrices $\mathbb{K}_p$, $\mathbb{K}_v$, namely

$$\mathbb{K}_p\,(\mathbf{x} - \mathbf{x}_0) + \boldsymbol{\phi}_{/\mathbf{x}}^T \boldsymbol{\lambda}_\phi = \mathbf{f}_p \tag{15a}$$

$$\mathbb{K}_v\,(\dot{\mathbf{x}} - \mathbf{v}_0) + \boldsymbol{\phi}_{/\mathbf{x}}^T \boldsymbol{\lambda}'_\phi + \boldsymbol{\psi}_{/\dot{\mathbf{x}}}^T \boldsymbol{\lambda}'_\psi = \mathbf{f}_v \tag{15b}$$

$$\boldsymbol{\phi}(\mathbf{x}, t_0) = \mathbf{0} \tag{15c}$$

$$\boldsymbol{\psi}(\dot{\mathbf{x}}, \mathbf{x}, t_0) = \mathbf{0} \tag{15d}$$

$$\boldsymbol{\phi}_{/\mathbf{x}}\dot{\mathbf{x}} + \boldsymbol{\phi}_{/t} = \mathbf{0}. \tag{15e}$$

Equation (15a) penalizes departure from the prescribed initial position and orientation, whereas Eq. (15b) penalizes departure from the prescribed initial linear and angular velocities. The right-hand side terms $\mathbf{f}_p$, $\mathbf{f}_v$ can be interpreted as external loads (e.g. dead weight, springs, and so on), that may be used to bias and thus further influence this solution phase. The position and velocity are solved simultaneously to give the solver the opportunity to trade position and velocity adjustments, since all constraint functions, Eqs. (15c) and (15d), depend on the position, and the non-holonomic and the derivative of the holonomic ones, Eqs. (15d) and (15e), also depend on the velocity. The resulting $\mathbf{x}$, $\dot{\mathbf{x}}$ represent the new, consistent initial values of the configuration.

4.2 Derivatives

In an explicit ODE problem $\dot{\mathbf{y}} = \mathbf{r}(\mathbf{y}, t)$, the initial value of the state derivative is directly $\dot{\mathbf{y}}(t_0) = \mathbf{r}(\mathbf{y}(t_0), t_0)$. When written in implicit form, $\mathbf{r}(\mathbf{y}, \dot{\mathbf{y}}, t) = \mathbf{0}$, a nonlinear problem must be solved, e.g. by resorting to a Newton-like iterative process, $\mathbf{r}_{/\dot{\mathbf{y}}}\partial\dot{\mathbf{y}} = -\mathbf{r}$, $\dot{\mathbf{y}} \mathrel{+}= \partial\dot{\mathbf{y}}$. This is not possible when the problem is DAE, since by definition $\mathbf{r}_{/\dot{\mathbf{y}}}$ is structurally singular, whereas the matrix pencil $(s\mathbf{r}_{/\mathbf{y}} + \mathbf{r}_{/\dot{\mathbf{y}}})$ is not for $|s| > 0$.

In a constrained mechanics problem of the form of Eq. (12) the "derivatives" are actually $\boldsymbol{\beta}$ (according to

ASME J. Comput. Nonlinear Dyn. MONTH YYYY V(N), doi:10.1115/1.4025628

Eq. (12a)), $\dot{\boldsymbol{\beta}}$, such that the corresponding accelerations $\ddot{\mathbf{x}}$ comply with the constraints, and the Lagrange multipliers $\boldsymbol{\lambda}_\phi$ and $\boldsymbol{\lambda}_\psi$. The direct solution requires one to differentiate the constraint equations (twice the holonomic, once the non-holonomic) [23]. To avoid the complexity of this approach, and significantly the need to formulate and implement the derivatives of the constraint equations, a different procedure has been designed. The original problem of Eq. (4a) is solved, but only the entries of $\mathbf{y}$ and $\dot{\mathbf{y}}$ corresponding to "derivatives" (i.e. $\boldsymbol{\beta}$, $\dot{\boldsymbol{\beta}}$, $\boldsymbol{\lambda}_\phi$, $\boldsymbol{\lambda}_\psi$) are updated according to Eqs. (4b) and (4c), the others are left untouched.

A linear analogy can be found by modifying the original problem with the addition and subtraction of the Jacobian matrix of the problem,

$$\left(c\mathbf{r}_{/\mathbf{y}}+\mathbf{r}_{/\dot{\mathbf{y}}}\right)\partial\dot{\mathbf{y}} = -\mathbf{r}+\left(\left(c\mathbf{r}_{/\mathbf{y}}+\mathbf{r}_{/\dot{\mathbf{y}}}\right)-\mathbf{r}_{/\dot{\mathbf{y}}}\right)\partial\dot{\mathbf{y}}, \quad (16)$$

which is then solved in an iterative manner,

$$\begin{aligned}\partial\dot{\mathbf{y}}^{(j+1)} = &-\left(c\mathbf{r}_{/\mathbf{y}}+\mathbf{r}_{/\dot{\mathbf{y}}}\right)^{-1}\mathbf{r}\\ &+\left(\mathbf{I}-\left(c\mathbf{r}_{/\mathbf{y}}+\mathbf{r}_{/\dot{\mathbf{y}}}\right)^{-1}\mathbf{r}_{/\dot{\mathbf{y}}}\right)\partial\dot{\mathbf{y}}^{(j)}. \quad (17)\end{aligned}$$

The iterative process converges to the correct value as long as the spectral radius[5] of matrix $\mathbf{I}-(c\mathbf{r}_{/\mathbf{y}}+\mathbf{r}_{/\dot{\mathbf{y}}})^{-1}\mathbf{r}_{/\dot{\mathbf{y}}} = (c\mathbf{r}_{/\mathbf{y}}+\mathbf{r}_{/\dot{\mathbf{y}}})^{-1}c\mathbf{r}_{/\mathbf{y}}$ is less than unity, and convergence is faster the smaller the spectral radius is. The value of c is tuned to improve the convergence rate.

4.3 First Step and Subsequent Ones

The first step requires a self-starting integration scheme. The Crank-Nicolson formula is used,

$$\mathbf{y}_k = \mathbf{y}_{k-1}+\frac{h}{2}\left(\dot{\mathbf{y}}_k+\dot{\mathbf{y}}_{k-1}\right), \quad (18)$$

i.e. Eq. (2) with $a_1 = 1$ and $b_0 = b_1 = 1/2$, which is unconditionally stable and second-order accurate. This method is not suitable for the integration of DAEs because it does not produce any algorithmic dissipation; however, in this context it is only used for one step.

The subsequent steps are performed using an original A/L stable linear multistep algorithm. The algorithm is formulated according to Eq. (2). The coefficients are chosen in order to guarantee second-order accuracy (three conditions) and asymptotic roots of the characteristic polynomial

$$(1-h\lambda b_0)\rho^2-(a_1+h\lambda b_1)\rho-(a_2+h\lambda b_2) = 0 \quad (19)$$

coincident for $|h\lambda|\to\infty$ and converging to a prescribed value $|\rho_\infty|$ (two conditions) when solving a purely oscillatory problem $\dot{y} = \lambda y = j\omega y$. This makes the tuning of algorithmic dissipation possible. The resulting coefficients

[5] The modulus of the largest eigenvalue.

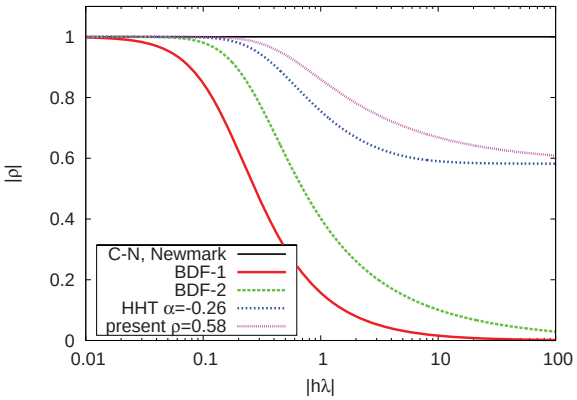

Fig. 3. Spectral radius of integration scheme.

are

$$\begin{aligned} & & b_0 &= \delta/\alpha+\alpha/2\\ a_1 &= 1-\beta & b_1 &= \beta/2+\alpha/2-(1+\alpha)\delta/\alpha \quad (20)\\ a_2 &= \beta & b_2 &= \beta/2+\delta\end{aligned}$$

with

$$\alpha = (t_k-t_{k-1})/(t_{k-1}-t_{k-2}) \quad (21)$$

$$\beta = \alpha\frac{(2+\alpha)(1-|\rho_\infty|)^2+2(1+\alpha)(2|\rho_\infty|-1)}{2(1+\alpha)-(1-|\rho_\infty|)^2} \quad (22)$$

$$\delta = \frac{\alpha^2(1-|\rho_\infty|)^2}{2(2(1+\alpha)-(1-|\rho_\infty|)^2)} \quad (23)$$

A-stability (unconditional stability) is obtained by choosing $0\le|\rho_\infty|\le 1$. When $|\rho_\infty|\equiv 0$ the method is equivalent to second-order Backward Difference Formulas (BDF), which are L-stable; $|\rho_\infty| = \sqrt{21}-4\cong 0.58$ minimizes the third-order remainder. Numerical experiments show that such value represents an optimal trade-off between accuracy and algorithmic dissipation. Figure 3 compares the spectral radius of the present method with that of well-known methods: Crank-Nicolson (C-N), Newmark, first and second order Backward Difference Formulas (BDF-1, BDF-2), and Hilbert-Hughes-Taylor (HHT) with the same $|\rho_\infty|$.

5 SPECIAL PROBLEMS

5.1 Real-Time Solver

Support for Real-Time Simulation (RTS) has been introduced in the initial value solver. RTS indicates the capability to perform time marching analysis within strict scheduling. The motivation is to exploit general-purpose simulation capabilities, generality of model topology and model, and software reuse without the need to resort

to specialized solvers in a hardware-in-the-loop environment, for example to interact with an actual control system, where the simulation replaces the process to be controlled. To this end, the simulation needs to be scheduled periodically in tight real time, and at the same time to interact with external processes.

Scheduling is obtained either using POSIX primitives (specifically, the `clock_nanosleep()` system call), or primitives provided by RTAI[6], the Real-Time Application Interface for Linux (specifically, the `rt_task_wait_period()` call).

Inter-process communication is obtained using UNIX sockets with POSIX scheduling, or real-time mailboxes with RTAI, either locally or over the Ethernet, in the latter case possibly using RT-Net[7]. Non-blocking communication is used to allow each process to stick as much as possible to its own scheduling, possibly at the cost of occasional overruns and missed communication.

In principle, RTS requires fixed cost to advance by one time step. This is no longer guaranteed when iteratively solving implicit nonlinear problems, as in the present case. Implicit integration is however needed when dealing with DAE problems, as mandated by the redundant coordinates approach. Moreover, redundant coordinates may result in large sets of equations (the size of a constrained mechanics problem is $12 \times n_{\text{bodies}}$ plus $6 \times n_{\text{bodies}} - n_{\text{dof}}$ for the constraint equations, as opposed to n_{dof} for minimal coordinates approaches). As a consequence, the size and the complexity of the model need to be carefully tailored, considering the required time step and the computational resources.

The built-in dense-storage sparse solver [20] has been specifically developed for this application to guarantee the maximum efficiency in matrix access and linear solution of problems with 50 to 2000 equations (in many tests, the LAPACK dense solver can be more efficient below 50 equations, whereas UMFPACK and KLU can be more efficient above 2000 equations). Further details on implementation and applications are given in [24, 25].

5.2 Cosimulation

Cosimulation is the capability to perform a cooperative task among multiple peers, exchanging information required by each task to complete the analysis. Several forms of cosimulation are supported. A very general form consists in communicating with a peer task using unidirectional sockets. MBDyn provides input and output sockets, respectively wrapped under the drive and the element interface, which can be interpreted as multi-channel streams (either blocking or non-blocking). UNIX *local* and *inet* sockets are supported for batch simulation, whereas real-time mailboxes are supported for hard real-time scheduling of the execution. Local sockets (and RT mailboxes) are used in cosimulation with tasks running on the same host machine (possibly SMP), whereas inet sockets (and RT mailboxes) are used in cosimulation with tasks running on different machines. Such cosimulation is inherently explicit, since the input values remain constant for the duration of the iterative solution of each time step; similarly, values are output only after convergence. Cosimulation applications with this type of communication have been developed between one or multiple concurrent multibody simulations and block diagram simulation tools like Simulink and Scicos/Scicoslab, also implementing several multirate integration strategies [26, 27].

A rather sophisticated interface has been developed for multidomain cosimulation, like fluid-structure interaction, with possibly incompatible interfaces between domains. Typically, the multibody domain is made of sets of nodes (e.g. those of a flexible helicopter or wind turbine rotor blade modeled using beams). The fluid domain is made of structured or unstructured grids. The physical interface is a surface. Domain interfacing requires the capability to map the configuration of the multibody domain into that of the interface of the other domain. The problem is formulated as an optimization using moving least squares on a compact meshless support provided by radial basis functions. The mapping provides a robust means to interpolate the configuration of points belonging to the two domains, $\mathbf{x}_p = \mathcal{H}\mathbf{x}_m$ (subscript $(\cdot)_m$ refers to the multibody domain; $(\cdot)_p$ refers to the peer domain). Given the linear nature of the mapping operator $\mathcal{H}(\cdot)$, only Cartesian point mapping is considered. On the structural domain side, whenever rotations are involved (e.g. when using structural nodes), they are mapped by considering a set of dummy points rigidly offset from each multibody node. Velocities (and accelerations, if needed) are mapped accordingly, $\dot{\mathbf{x}}_p = \mathcal{H}\dot{\mathbf{x}}_m$. By prescribing that the virtual work is conserved across the mapping one obtains $\delta\mathcal{W} = \delta\mathbf{x}_m^T\mathbf{f}_m = \delta\mathbf{x}_p^T\mathbf{f}_p$, which implies $\mathbf{f}_m = \mathcal{H}^T\mathbf{f}_p$. Matrix $\mathcal{H}$ only needs to be computed once based on the nominal domain interfaces, using the algorithm proposed in [28]. It is subsequently used for matrix-vector multiplications, either direct or transposed, exploiting its often considerable sparsity. Figure 4 illustrates a flexible flapping wing, modeled using nonlinear shell elements, in cosimulation with a CFD solver [29].

Interaction requires to:

- (optional) compute the configuration of offset points;
- map the configuration onto the peer points;
- send the configuration of mapped points;
- receive the forces acting on mapped points;
- map back forces on multibody domain;
- (optional) reduce the forces to the multibody nodes;

[6] `http://www.rtai.org/`
[7] `http://www.rtnet.org/`

ASME J. Comput. Nonlinear Dyn. MONTH YYYY V(N), doi:10.1115/1.4025628

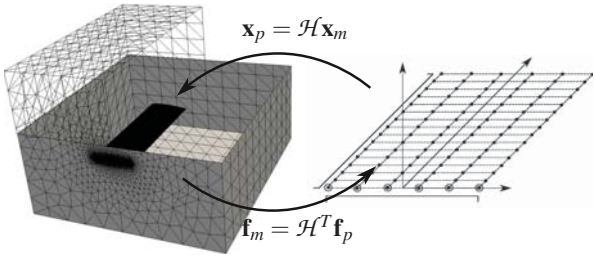

Fig. 4. Mesh (left) to multibody model (right) communication pattern for a typical flapping wing fluid-structure coupled simulation.

Communication can occur once per time step, either using the configuration at the end of the previous step or the predicted (i.e. extrapolated) one for the current time step, in what is called loose (i.e. explicit) coupling; alternatively, communication can also occur during the iterations of the nonlinear problem solution, thus realizing a thoroughly tight (thus implicit) coupling.

Data are communicated using a simple native protocol implemented in MBDyn and in a free peer communication library, `libmbc`, distributed with the code. The library is implemented in C, with interfaces in C++ and Python. Embedding the communication library in peer software is straightforward; interface with several CFD solvers has been developed for applications in aerospace [29, 30] and wind energy [31, 32].

5.3 Inverse Kinematics/Inverse Dynamics

Specific solution strategies have been developed for robotics applications, which encompass the capability to perform inverse kinematics and inverse dynamics (IK/ID) of mechanisms with rather general topology. The focus is on solving IK problems and, in some cases, on solving ID problems for actuator sizing and mechanism control based on feedforward and feedback linearization. Two approaches have been considered, one based on control constraint [33, 34]. and the other on a staggered IK/ID solution [35, 36]. A comparison between the two approaches on a real case of industrial interest is discussed in [37].

A control constraint problem is formulated as

$$\mathbf{M}\ddot{\mathbf{x}} - \mathbf{B}^T\mathbf{u} = \mathbf{f} \tag{24}$$

$$\boldsymbol{\vartheta}(\mathbf{x}) = \boldsymbol{\alpha}(t), \tag{25}$$

where the generalized control forces $\mathbf{u}$ need to be designed in order to prescribe the control constraint $\boldsymbol{\vartheta}(\mathbf{x}) = \boldsymbol{\alpha}(t)$ ("passive" constraints $\boldsymbol{\phi}(\mathbf{x}) = \mathbf{0}$ are omitted for simplicity). The theoretical details of the problem are discussed in [33,34] and references therein; the realization of the control has been obtained by developing two companion elements. The first element implements the prescribed motion $\boldsymbol{\vartheta}(\mathbf{x}) = \boldsymbol{\alpha}(t)$, whereas the second one applies the related generalized force through the Lagrange multiplier associated with the control constraint by introducing an appropriate matrix $\mathbf{B}$. The resulting problem is DAE; the corresponding differential index[8] is at least three, in case the so-called orthogonal realization condition of matrix $\boldsymbol{\vartheta}_{/\mathbf{x}}\mathbf{M}^{-1}\mathbf{B}^T$ being non-singular is met. Otherwise, a tangent realization is needed, and the index of the DAE is greater than three. The problem is treated as a normal initial value one, although of possibly high index.

The staggered approach is treated as a special problem, called *inverse dynamics*, with a dedicated solver. The IK phase requires the solution of three specialized kinetostatic problems that are trivial in case of fully determined problems, whereas in case of underdetermined problems they can be cast into the minimization of cost functions augmented by kinematic constraints, namely

$$J(\mathbf{x}) = \frac{1}{2}\mathcal{C}(\mathbf{x} - \mathbf{x}_{\text{ergo}}) + \boldsymbol{\lambda}_\phi \cdot \boldsymbol{\phi} + \boldsymbol{\lambda}_\vartheta \cdot (\boldsymbol{\vartheta} - \boldsymbol{\alpha}) \tag{26a}$$

$$J'(\dot{\mathbf{x}}) = \frac{1}{2}\Delta\dot{\mathbf{x}}^T\mathbf{M}\Delta\dot{\mathbf{x}} + \boldsymbol{\lambda}'_\phi \cdot \dot{\boldsymbol{\phi}} + \boldsymbol{\lambda}'_\vartheta \cdot \left(\dot{\boldsymbol{\vartheta}} - \dot{\boldsymbol{\alpha}}\right) \tag{26b}$$

$$J''(\dot{\mathbf{x}}) = \frac{1}{2}\Delta\ddot{\mathbf{x}}^T\mathbf{M}\Delta\ddot{\mathbf{x}} + \boldsymbol{\lambda}''_\phi \cdot \ddot{\boldsymbol{\phi}} + \boldsymbol{\lambda}''_\vartheta \cdot \left(\ddot{\boldsymbol{\vartheta}} - \ddot{\boldsymbol{\alpha}}\right) \tag{26c}$$

where $\mathcal{C}$ is a cost function usually constructed from analogy with potential energy associated with straining of elastic components that penalize undesired motions like departing from a reference configuration or approaching obstacles, and $\Delta\dot{\mathbf{x}} = \dot{\mathbf{x}} - \dot{\mathbf{x}}_{\text{ref}}$, with reference velocity and acceleration predicted using backward finite differences. Underdetermined problems can be solved, eliminating the indetermination through a careful choice of the cost functions. Details on Eqs. (26) are given in [36]. As long as kinematics are computed up to second-order derivatives, the ID problem is solved as

$$\left\{ \begin{array}{c} \boldsymbol{\lambda}_\phi \\ \mathbf{u} \end{array} \right\} = \left[\boldsymbol{\phi}^T_{/\mathbf{x}} \ \mathbf{B}^T \right]^{-1} (\mathbf{f} - \mathbf{M}\ddot{\mathbf{x}}), \tag{27}$$

which is defined as long as all joints are actuated. This approach has been successfully used for robotics [35–37] and biomechanical applications [38, 39]. Figure 5 shows an example robotic application in which the staggered IK/ID procedure was used to compute the joint trajectories required to perform a prescribed end effector task. The robot has 7 degrees of freedom, and the position and the axis of the tool are prescribed, making the system underdetermined by 2 degrees of freedom. The bottom plot shows the same analysis with ergonomy functions penalizing the proximity to a point in space, to induce obstacle avoidance. ID is used to implement feedforward control.

Figure 6 shows a model of the left arm of a helicopter pilot, with detailed muscles, holding the collective control

[8] For a definition of differential index see for example [22].

ASME J. Comput. Nonlinear Dyn. MONTH YYYY V(N), doi:10.1115/1.4025628

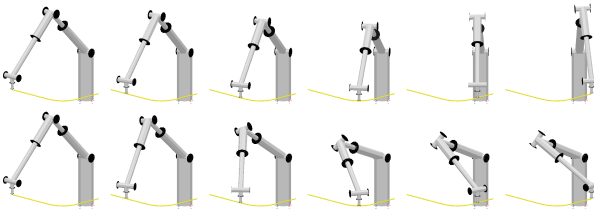

Fig. 5. PA10-like robot performing prescribed nominal (top) and obstacle avoidance (bottom) trajectories.

Fig. 6. Helicopter pilot arm holding the collective control inceptor at 10%, 50%, and 90%.

```
SolutionManager SM;
ElementsContainer E;
VectorHandler x, xp, Dxp, r;
MatrixHandler J;

foreach (e in E) e.BeforePredict(x, xp);
Predict(x);
foreach (e in E) e.AfterPredict(x, xp);

while(1) {  // solve one time step
 foreach (e in E) r += e.AssRes(x, xp);
 if (Test(r)) break;

 if (new_matrix) {  // only if needed
  foreach (e in E) J += e.AssJac(x, xp, c);
 }
 Dxp = SM.Solve(J, r); xp += Dxp; x += c*Dxp;
 foreach (e in E) e.Update(x, xp);
}

foreach (e in E) e.AfterConvergence(x);
if (output) {
 foreach (e in E) e.Output();
}
```

Fig. 7. Pseudo-code of nonlinear problem iterative solution.

inceptor. The IK procedure is used to determine the configuration of the arm. The ID procedure is used to determine the joint torques and contribute to the determination of the muscular activation required to perform the task.

5.4 Customization

Entities of several types can be implemented as run-time loadable modules, whose portability is taken care of by GNU's LTDL library[9]. Elements, drives, constitutive laws and other features are stored in containers that associate their name with a pointer to a functional object that instantiates an entity of the given type. User-defined modules must export a C function, `module_init()`, that is invoked when the module is loaded and is typically used to register those functional objects in the containers.

Among the methods that user-defined classes derived from the `Elem` class can provide, those directly related to solution phases are illustrated in (simplified) pseudo-code in Fig. 7. Missed convergence is treated separately, e.g. when adapting the time step.

Elements (and nodes) can access the state and its derivative prior to and after prediction. Apart from contributing to the residual and Jacobian matrix, they can update their internal state after each iteration and upon convergence, and contribute to the output.

Other methods make it possible to extract information during the analysis, for example to produce custom output or to feed information about the state of an element into other elements. Such features have been generalized through a common interface, with textual names for each published value, that can be referenced in the input, in drives, and in expressions evaluated run-time.

Several non-core functionalities are provided using run-time modules distributed with the software. Such modules provide several specialized constitutive laws (hydraulic and elastomeric dampers, continuous contact [40], muscles [38, 39]) and elements (aircraft wheel, helicopter and cycloidal rotor induced velocity, contact with friction as nonsmooth event formulated in co-simulation as a linear complementarity problem [40]). A special module independently developed by Reinhard Resch, an ingenious user, supports the definition of drives, constitutive laws and elements as Octave[10] functions.

6 POINTS OF STRENGTH AND LIMITATIONS

Among the points of strength of the formulation are undoubtedly its multidisciplinarity and efficiency. The integrated solution of aerodynamic, electric, thermal and hydraulic domains along with the core multibody domain, support the analysis of rather complex multidisciplinary problems. Real-time and cosimulation capabilities support easy and seamless interfacing with problems ranging from complex control systems to external fluid-dynamic codes.

However, the otherwise excellent core multibody library lags behind other codes with respect to the simulation of contacts. Contact detection and simulation is being addressed, but the current capabilities are rather poor

[9] `libltdl`, `http://www.gnu.org/software/libtool/`

[10] `http://www.gnu.org/software/octave/`

compared for example with CHRONO::Engine.

The lack of a graphical pre-processor and the somewhat primitive graphical post-processor is another limitation. Recent efforts from independent users to exploit Blender as pre- and postprocessor[11] are partially addressing the issue. This gap is well compensated by the versatility of the input file structure, which makes the definition of really complex models possible without the need to resort to their graphical representation.

Novice users sometimes find the initial assembly and derivative solution phases difficult to understand and manage, especially when confronted with errors resulting from inconsistently input models. The redesign and improvement of these phases is under consideration.

All in all, the object-oriented implementation so far made the maintenance and improvement of the code possible over the years. The decision to store state-dependent data inside the element classes appeared to be a double-edged sword. In fact, it is one of the reasons that complicate the task of migrating an element between different processes during a MPI run, and that so far prevented the reliable support of the restart functionality.

References

[1] Schiehlen, W., 1997, "Multibody system dynamics: Roots and perspectives," Multibody System Dynamics, **1**(2), pp. 149–188, doi:10.1023/A:1009745432698.

[2] Shabana, A. A., 1997, "Flexible multibody dynamics: Review of past and recent developments," Multibody System Dynamics, **1**(2), pp. 189–222, doi:10.1023/A:1009773505418.

[3] Shabana, A. A., 1998, *Dynamics of Multibody Systems*, Cambridge University Press, Cambridge, MA, second ed.

[4] Bauchau, O. A. and Kang, N. K., 1993, "A multibody formulation for helicopter structural dynamic analysis," Journal of the American Helicopter Society, **38**(2), pp. 3–14.

[5] Anitescu, M. and Tasora, A., 2010, "An iterative approach for cone complementarity problems for nonsmooth dynamics," Computational Optimization and Applications, **47**(2), pp. 207–235, doi:10.1007/s10589-008-9223-4.

[6] Gerstmayr, J., Dorninger, A., Eder, R., Gruber, P., Reischl, D., Saxinger, M., Schörgenhumer, M., Humer, A., Nachbagauer, K., Pechstein, A., and Vetyukov, Y., 2013, "HOTINT — a script language based framework for the simulation of multibody dynamics systems," *Proceedings of ASME IDETC/CIE*, Portland, OR.

[7] Bauchau, O. A. and Trainelli, L., 2003, "The vectorial parameterization of rotation," Nonlinear Dynamics, **32**(1), pp. 71–92, doi:10.1023/A:1024265401576.

[8] Masarati, P. and Morandini, M., 2010, "Intrinsic deformable joints," Multibody System Dynamics, **23**(4), pp. 361–386, doi:10.1007/s11044-010-9194-y.

[9] Bauchau, O. A., Li, L., Masarati, P., and Morandini, M., 2011, "Tensorial deformation measures for flexible joints," J. of Computational and Nonlinear Dynamics, **6**(3), doi:10.1115/1.4002517.

[10] Ghiringhelli, G. L., Masarati, P., and Mantegazza, P., 2000, "A multi-body implementation of finite volume beams," AIAA Journal, **38**(1), pp. 131–138, doi:10.2514/2.933.

[11] Masarati, P., Morandini, M., Quaranta, G., and Vescovini, R., 2011, "Multibody analysis of a micro-aerial vehicle flapping wing," *Multibody Dynamics 2011*, J. C. Samin and P. Fisette, eds., Brussels, Belgium.

[12] Masarati, P., Morandini, M., and Solcia, T., 2012, "A membrane element for micro-aerial vehicle fluid-structure interaction," *2nd Joint International Conference on Multibody System Dynamics*, P. Eberhard and P. Ziegler, eds., Stuttgart, Germany.

[13] Ghiringhelli, G. L., Masarati, P., Mantegazza, P., and Nixon, M. W., 1999, "Multi-body analysis of the 1/5 scale wind tunnel model of the V-22 tiltrotor," *American Helicopter Society 55th Annual Forum*, vol. 2, Montreal, Canada, pp. 1087–1096.

[14] Ghiringhelli, G. L., Masarati, P., Mantegazza, P., and Nixon, M. W., 1999, "Multi-body analysis of a tiltrotor configuration," Nonlinear Dynamics, **19**(4), pp. 333–357, doi:10.1023/A:1008386219934.

[15] Masarati, P., Piatak, D., Quaranta, G., Singleton, J., and Shen, J., 2008, "Soft-inplane tiltrotor aeromechanics investigation using two comprehensive multibody solvers," Journal of the American Helicopter Society, **53**(2), pp. 179–192, doi:10.4050/JAHS.53.179.

[16] Merlini, T. and Morandini, M., 2004, "The helicoidal modeling in computational finite elasticity. part II: multiplicative interpolation," International journal of solids and structures, **41**(18–19), pp. 5383–5409, doi:10.1016/j.ijsolstr.2004.02.026.

[17] Merlini, T. and Morandini, M., 2013, "On successive differentiations of the rotation tensor. An application to nonlinear beam elements," Journal of Mechanics of Materials and Structures, **in press**.

[18] Davis, T. A., 2004, "Algorithm 832: Umfpack, an unsymmetric-pattern multifrontal method," ACM Transactions on Mathematical Software, **30**(2), pp. 196–199.

[19] Davis, T. A. and Palamadai Natarajan, E., 2010, "Algorithm 907: KLU, a direct sparse solver for circuit simulation problems," ACM Transactions on Mathematical Software, **37**(3), pp. 36:1–17, doi:10.1145/1824801.1824814.

[20] Morandini, M. and Mantegazza, P., 2007, "Using dense storage to solve small sparse linear systems," ACM Transactions on Mathematical Software, **33**(1), pp. 5:1–12, doi:10.1145/1206040.1206045.

[21] Mantegazza, P., Masarati, P., Morandini, M., and Quaranta, G., 2007, "Computational and design aspects in multibody software development," *Multibody Dynamics*, J. C. Garcia Orden, J. M. Goicolea, and J. Cuadrado, eds., vol. 4, Springer, pp. 137–158, doi:10.1007/978-1-4020-5684-0_7.

[22] Brenan, K. E., Campbell, S. L. V., and Petzold, L. R., 1989, *Numerical Solution of Initial-Value Problems in*

[11] `http://sourceforge.net/projects/blenderandmbdyn/`, another example of the fruitful interaction with users granted by the free software development model.

Differential-Algebraic Equations, North-Holland, New York.

[23] Leimkuhler, B., Petzold, L. R., and Gear, C. W., 1991, "Approximation methods for the consistent initialization of differential-algebraic equations," SIAM Journal on Numerical Analysis, **28**(1), pp. 205–226.

[24] Masarati, P., Attolico, M., Nixon, M. W., and Mantegazza, P., 2004, "Real-time multibody analysis of wind-tunnel rotorcraft models for virtual experiment purposes," *American Helicopter Society 4th Decennial Specialists' Conference on Aeromechanics*, Fisherman's Wharf, San Francisco, CA.

[25] Morandini, M., Masarati, P., and Mantegazza, P., 2005, "Performance improvements in real-time general-purpose multibody virtual experimenting of rotorcraft systems," *31st European Rotorcraft Forum*, Firenze, Italy.

[26] Cavagna, L., Fumagalli, A., Masarati, P., Morandini, M., and Mantegazza, P., 2011, "Real-time aeroservoelastic analysis of wind-turbines by free multibody software," *Multibody Dynamics: Computational Methods and Applications*, W. Blajer, K. Arczewski, J. Fraczek, and M. Wojtyra, eds., vol. 23, Springer, pp. 69–86, doi:10.1007/978-90-481-9971-6_4.

[27] Solcia, T. and Masarati, P., 2011, "Efficient multirate simulation of complex multibody systems based on free software," *ASME IDETC/CIE 2011*, Washington, DC, USA, DETC2011-47306.

[28] Quaranta, G., Masarati, P., and Mantegazza, P., 2005, "A conservative mesh-free approach for fluid structure interface problems," *Coupled Problems 2005*, Santorini, Greece.

[29] Alioli, M., Morandini, M., and Masarati, P., 2013, "Coupled multibody-fluid dynamics simulation of flapping wings," *Proceedings of ASME IDETC/CIE*, Portland, OR, DETC2013-12198.

[30] Malhan, R., Baeder, J., Chopra, I., and Masarati, P., 2013, "Investigation of aerodynamics of flapping wings for MAV applications using experiments and CFD-CSD analysis," *American Helicopter Society 5th International Specialists Meeting on Unmanned Rotorcraft and Network Centric Operations*, Scottsdale, Arizona, USA.

[31] Masarati, P. and Sitaraman, J., 2011, "Tightly coupled CFD/multibody analysis of NREL unsteady aerodynamic experiment phase VI rotor," *49th AIAA Aerospace Sciences Meeting*, Orlando, Florida.

[32] Sitaraman, J., Gundling, C., Roget, B., and Masarati, P., 2013, "Computational study of wind turbine performance and loading response to turbulent inflow conditions," *American Helicopter Society 69th Annual Forum*, Phoenix, Arizona, Paper No. 339.

[33] Fumagalli, A., Masarati, P., Morandini, M., and Mantegazza, P., 2011, "Control constraint realization for multibody systems," J. of Computational and Nonlinear Dynamics, **6**(1), p. 011002 (8 pages), doi:10.1115/1.4002087.

[34] Masarati, P., Morandini, M., and Fumagalli, A., in press, "Control constraint of underactuated aerospace systems," J. of Computational and Nonlinear Dynamics.

[35] Fumagalli, A. and Masarati, P., 2009, "Real-time inverse dynamics control using general-purpose multibody software," Multibody System Dynamics, **22**(1), pp. 47–68, doi:10.1007/s11044-009-9153-7.

[36] Masarati, P., in press, "Computed torque control of redundant manipulators using general-purpose software in real-time," Multibody System Dynamics, doi:10.1007/s11044-013-9377-4.

[37] Morandini, M., Masarati, P., Bargigli, L., and Vaccani, L., 2012, "Feedforward control design from general-purpose multibody analysis for an original parallel robot concept," *2nd Joint International Conference on Multibody System Dynamics*, P. Eberhard and P. Ziegler, eds., Stuttgart, Germany.

[38] Masarati, P. and Quaranta, G., 2013, "Coupled bioaeroservoelastic rotorcraft-pilot simulation," *Proceedings of ASME IDETC/CIE*, Portland, OR, DETC2013-12035.

[39] Masarati, P., Quaranta, G., and Zanoni, A., in press, "Dependence of helicopter pilots' biodynamic feedthrough on upper limbs' muscular activation patterns," Proc. IMechE Part K: J. Multi-body Dynamics, doi:10.1177/1464419313490680.

[40] Fancello, M., Masarati, P., and Morandini, M., 2013, "Adding non-smooth analysis capabilities to general-purpose multibody dynamics by co-simulation," *Proceedings of ASME IDETC/CIE*, Portland, OR, DETC2013-12208.