Multibody System Dynamics
https://doi.org/10.1007/s11044-026-10158-5

RESEARCH

A co-simulation algorithm for the efficient real-time parallel solution of multibody systems

Andrea Fontana[1] · **Marco Morandini**[2] · **Pierangelo Masarati**[2]

Received: 30 April 2025 / Accepted: 9 March 2026

Abstract
This study presents an original formulation for the co-simulation of multibody systems based on partitioning at interface nodes or joints where motion and loads are exchanged between subsystems. The proposed approach shows improved algorithmic stability compared to existing formulations when implemented in explicit, loose coupling, due to the retention of interface mass on both sides of the partition and the explicit exchange of inertia load contributions along with constraint reactions. The stability properties of the proposed approach are illustrated through benchmark problems. Its effectiveness is further validated in complex mechanical systems, including the structural dynamics of beams and shells and the real-time driver-in-the-loop dynamics of a race car.

Keywords Co-simulation · Parallelization

1 Introduction

Simulation is a cornerstone of modern engineering, offering a powerful means to model, analyze, and optimize complex systems. Among its many applications, real-time simulation has become increasingly essential in domains requiring immediate feedback, including control systems, simulators, hardware-in-the-loop testing, and game engines [1–3]. The challenge is to achieve real-time performance with high-fidelity models [4, 5], which requires consistently short time steps, and, consequently, high computational efficiency. The present work addresses this challenge by proposing a method to enhance the efficiency of the solution of multibody systems without compromising the fidelity of the model.

The proposed solution relies on co-simulation (also known as cooperative or coupled simulation) [6, 7], a widely used approach for addressing multidomain problems. An exhaustive review of co-simulation can be found in the work of Gomes et al. [8] and Hafner

✉ P. Masarati
pierangelo.masarati@polimi.it

A. Fontana
andrea8.fontana@mail.polimi.it

M. Morandini
marco.morandini@polimi.it

[1] Dipartimento di Meccanica, Politecnico di Milano, Milan 20156, Italy

[2] Dipartimento di Scienze e Tecnologie Aerospaziali, Politecnico di Milano, Milan 20156, Italy

Published online: 23 March 2026

and Popper [9]. In the present work, co-simulation is used to parallelize the solution of a mechanical system by partitioning it into subsystems, each solved on a dedicated core. At each time step, the subsystems exchange coupling variables (namely, positions and forces), whose formulation is detailed in this article. The proposed co-simulation scheme requires the exchange of extrapolated data; therefore, it should be possible to implement it within the latest version of the FMI standard, 3.0.2 [10], which introduced the `Intermediate Update Mode`, "enabl[ing] advanced Co-Simulation with interpolation/extrapolation techniques of inputs and outputs".

This approach preserves the original solvers and equation formulations, resulting in a noninvasive, general-purpose method. It avoids the need for internal data manipulation and instead relies on straightforward communication protocols, such as Internet or Unix sockets using TCP or UDP, which are widely available in many computational and simulation environments. These include mathematical environments such as Matlab, Simulink, and Octave, programming environments such as Python, and multibody solvers, provided they support the output of internal states and the input of prescribed kinematics and loads from external sources, which is a common feature. When solvers or mathematical environments are available as free or open-source software, the integration of such basic primitives is typically straightforward; however, proprietary solvers and environments often allow the integration of user-defined routines to achieve similar functionality.

Despite their advantages in terms of effectiveness and ease of implementation, coupling schemes (particularly explicit ones) may introduce challenges related to stability and accuracy, which depend on the choice of exchanged variables. To this end, the method proposed in Sect. 2 is benchmarked using a simple linear model for stability analysis and compared in Sect. 3 with reference algorithms from the literature. To assess its performance in more complex scenarios, the proposed algorithm is implemented in MBDyn[1] [11]. Results from an independent implementation, in the form of a Python script using standard integration schemes, are also presented to illustrate the versatility of the proposed method. Implementation details are discussed in Sect. 4.

Co-simulations are evaluated against monolithic simulations concerning computational efficiency and accuracy. A full-car vehicle model for Driver-in-the-Loop (DiL) simulations is developed and executed in real-time in Sect. 5, followed by conclusions in Sect. 6.

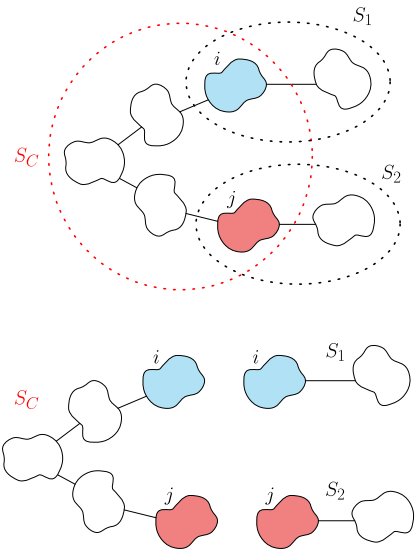
2 Co-simulation algorithm

The proposed algorithm is inspired by the one presented by Zhang et al. in [12], which was developed to study coupled systems composed of smooth multibody systems and non-smooth particle dampers. That co-simulation aims to extract and combine the best characteristics of rather different solvers—each specialized in its respective domain (MBDyn for smooth dynamics and Chrono::Engine[2] for non-smooth dynamics, in [12])—within a robust and algorithmically stable framework. It is implemented as a tightly coupled scheme, requiring the subsystems to exchange information—i.e., the pose from the smooth to the non-smooth solver and the corresponding reaction force and moment from the non-smooth to the smooth solver—multiple times at each time step through an iterative refinement process until convergence criteria are met for both solvers.

[1] A free software, general-purpose solver for multibody and multiphysics simulations, https://www.mbdyn.org.

[2] An Open Source general-purpose multibody library, specialized in non-smooth dynamics, https://projectchrono.org/.

Fig. 1 Node partition



The algorithm presented here has a different purpose: increasing the efficiency of multibody simulations by distributing the solution over multiple CPU cores, each dedicated to computing a small portion of the original system. To achieve this goal, the proposed algorithm deviates from the original formulation in two major areas. First, the tight coupling scheme is replaced by a loosely coupled approach, which not only reduces computational cost but also ensures a more consistent load throughout the simulation. In a loosely coupled system, information is exchanged only once per time step, and no correction iterations take place. Furthermore, the procedure described in the referenced paper features a sequential integration of the subsystems, whereas in the proposed algorithm, the integration of the so-called real subsystems is performed in parallel, as demonstrated in the following examples.

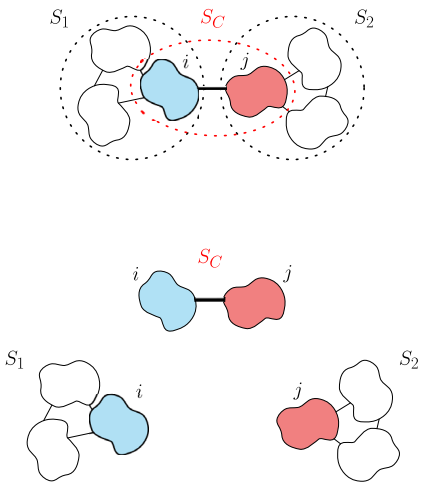
2.1 Partitioning

One crucial aspect of the proposed algorithm is the way the multibody model is partitioned into subsystems. This section addresses two common scenarios: splitting at constraints and splitting at nodes that logically define the interface between subsystems, as illustrated in Figs. 1 and 2. Both cases involve the duplication of the interface nodes across each cut and the presence of coupling subsystems. Despite the resulting slight increase in system size, this approach has a positive impact on the stability of the coupling, as will be shortly discussed.

2.1.1 Partitioning at a node

The simplest partition is obtained using a node as the interface between two (or more) subsystems. This approach can be employed in tree-like structures, such as the system in Fig. 1 to isolate individual branches. The subsystem that collects all the interface nodes (in this simple example, bodies i and j) is named the coupling subsystem, indicated as S_C. A possible application of this approach is shown in Fig. 3. A flexible blade is clamped to a rigid hub at its root node, which splits the system in two parts. If multiple blades are present, the same operation can be repeated, each time creating a new subsystem and adding the interface node to the coupling subsystem, represented by the hub.

Fig. 2 Constraint partition



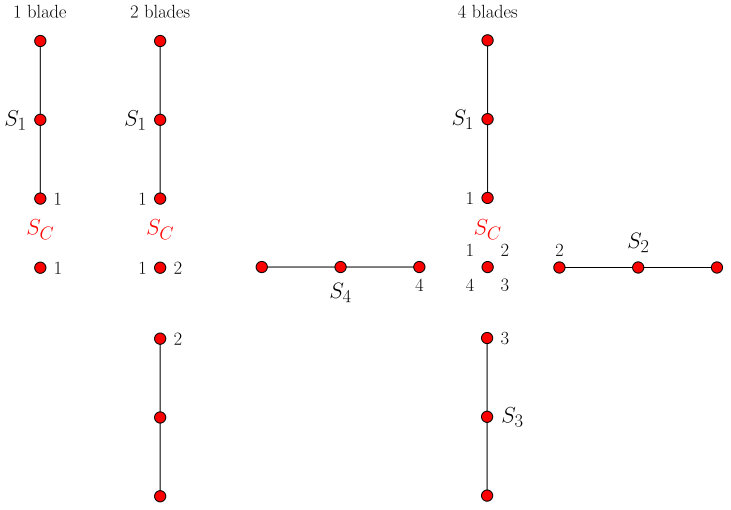


Fig. 3 Simplified helicopter rotor representation (1, 2, and 4 blades); example application of partitioning at a node

2.1.2 Partitioning at a constraint

If a constraint is chosen to split two subsystems, a slightly different approach is required. Two real subsystems, S_1 and S_2 in Fig. 2, contain the two sections of the model up to the two coupling bodies. The coupling subsystem, S_C, includes the two coupling bodies linked by the original constraint.

Any algebraic constraint or connection between two nodes, such as springs and dampers, can be employed. In the upcoming examples, clamp constraints frequently arise, as they can be conveniently used to split a single node in two. In the following example, a revolute hinge is used to link the coupling nodes of the two halves of a flexible pendulum, modeled with beam elements (Fig. 4).

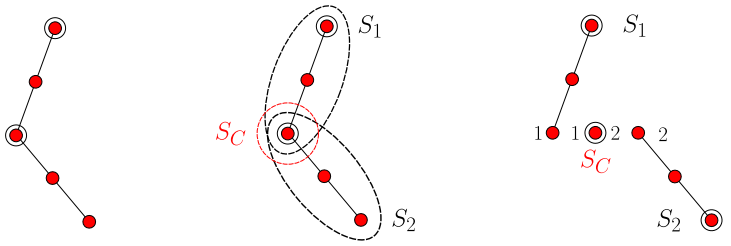


Fig. 4 Flexible double pendulum; example application of partitioning at a constraint

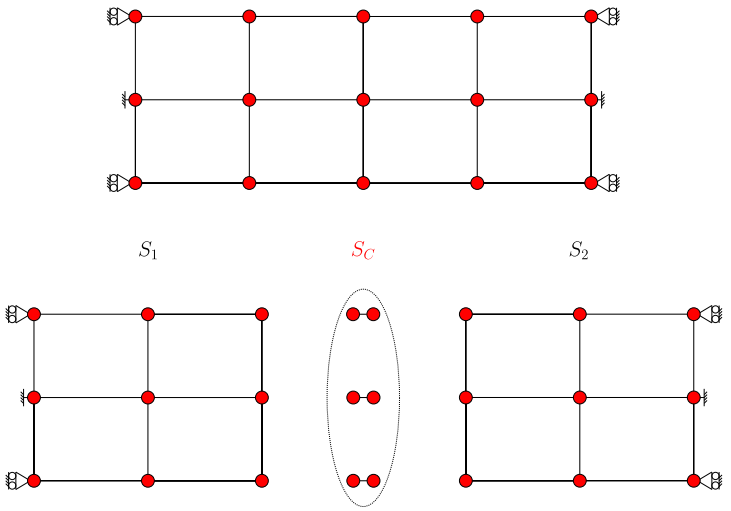


Fig. 5 Shell; example application of partitioning at (discretized) one-dimensional interface

2.1.3 Partitioning at one- (and multi-)dimensional interfaces

The algorithm remains effective for subsystems separated by one- or multi-dimensional interfaces, where discretization involves multiple nodes or constraints. The model in Fig. 5 is a rectangular-shaped solid discretized by the 4-node shell elements presented in [13]. The interface occurs along the vertical line that splits the mesh in two (equal) parts. The nodes of the centerline are split in pairs of nodes, each accounting for half the mass and the corresponding inertia properties. Each pair of nodes is clamped together by a dummy joint, which is used as the partitioning element. As described in the previous section, two real subsystems and a coupling subsystem are generated, the latter containing all pairs of interface nodes. The interface of a partitioned solid would be a surface, discretized by a surface mesh and a grid of interface nodes. The latter would then be split into pairs of nodes, each accounting for half the mass and the corresponding inertia properties. Each pair of nodes would be clamped together, and all clamped pairs would belong to the coupling subsystem.

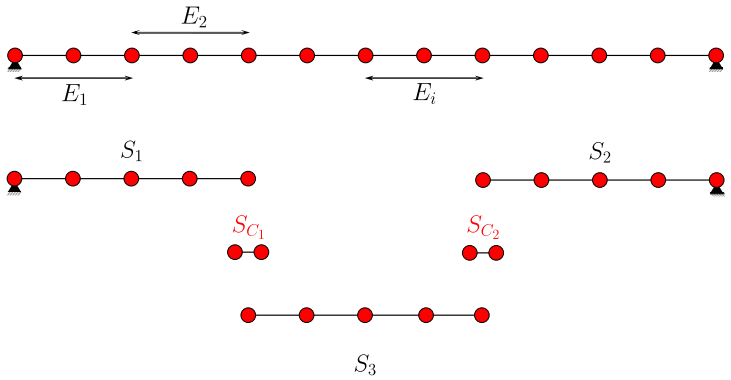


Fig. 6 Multiply partitioned beam; example application of multiple partitions

2.1.4 Multiple partitioning and coupling subsystems

As the number of subsystems increases, multiple coupling subsystems may arise. This case can be handled by integrating all real subsystems in parallel, followed by the concurrent integration of all coupling subsystems, without altering the solution process. Figure 6 shows a slender beam divided into three subsystems, S_1, S_2, and S_3. Two independent coupling subsystems are generated, S_{C_1} (which links S_1 and S_3) and S_{C_2} (which links S_2 and S_3).

2.2 Algorithm definition

The algorithm works as follows.

i. At the beginning of each time step, the state of the coupling bodies i and j of subsystem S_C is predicted and substituted in the integration scheme to yield $\mathbf{y}_i^G$ and $\mathbf{y}_j^G$.
ii. The predicted states are communicated to the corresponding subsystems, where a dedicated kinematic constraint prescribes the motion of the coupling bodies by enforcing the predicted kinematic quantities, $\mathbf{y}_i^G \rightarrow S_1$ and $\mathbf{y}_j^G \rightarrow S_2$.
iii. Subsystems S_1 and S_2 are then independently integrated one time step forward in parallel. As a result of the new configuration of the system, reaction forces and torques arise for each subsystem, $\mathbf{f}_{C,i}$ and $\mathbf{f}_{C,j}$.
iv. All constraint forces are sent back to the coupling subsystem, S_C, which is integrated one step forward to yield the updated state of the coupling bodies.
v. The loop can be repeated until convergence criteria are met by sending the updated states of bodies i and j to subsystems S_1 and S_2, i.e., by repeating the process from step (ii.), in the case of the tight coupling scheme, or the obtained value can be accepted and the solver proceeds with the next time step, in the case of loose coupling scheme.

The algorithm is structured as a Gauss-Seidel scheme. The connected subsystems, S_1 and S_2, use the interface kinematics predicted by the connection subsystem, S_C, to advance their states and send back to the connection subsystem the interface forces it subsequently uses to advance its own states.

2.3 Coupling forces determination

The definition of the coupling forces first requires the introduction of some kinematic quantities. All the kinematic and dynamic entities discussed in the following are assumed to be

expressed in a common fixed reference frame indicated as the *absolute reference frame*. For each of the coupling bodies i and j,

- $\mathbf{x}$ is the node's position vector (not necessarily coincident with the center of gravity of the corresponding rigid body);
- $\boldsymbol{\theta}$ is the node's orientation vector (the Euler vector), representing the orientation of the node in the absolute reference frame, such that the orientation matrix of the body, $\mathbf{R}$, can be expressed through Rodriguez's formula

$$\mathbf{R} = \mathbf{I} + \frac{\sin\theta}{\theta}\boldsymbol{\theta}\times + \frac{1-\cos\theta}{\theta^2}\boldsymbol{\theta}\times\boldsymbol{\theta}\times \tag{1}$$

  where $\theta = \|\boldsymbol{\theta}\|$ and operator $(\cdot)\times$ represents the 3×3 matrix that generates the vector product of its argument by another vector at its right;[3]
- $\mathbf{q}$ is the vector collecting the coupling body's position and orientation vectors, $\mathbf{q} = \{\mathbf{x};\ \boldsymbol{\theta}\}$;
- $\boldsymbol{\beta} = m\dot{\mathbf{x}} + \mathbf{S}\times^T\boldsymbol{\omega}$ is the momentum vector of the coupling body, where $\boldsymbol{\omega}$ is the body's angular velocity vector, m its mass, and $\mathbf{S} = m\mathbf{o}_{\mathrm{CM}}$ its static — or first-order inertia — moment, with $\mathbf{S}\times^T\boldsymbol{\omega} = -\mathbf{S}\times\boldsymbol{\omega} = \boldsymbol{\omega}\times\mathbf{S}$, being $\mathbf{o}_{\mathrm{CM}}$ the location of the body's center of mass with respect to the node, in the absolute frame;
- $\boldsymbol{\gamma} = \mathbf{S}\times\dot{\mathbf{x}} + \mathbf{J}\boldsymbol{\omega}$ is the momenta moment vector of the coupling body, where $\mathbf{J}$ is the (second-order) inertia tensor of the body;
- $\mathbf{f}_{\mathrm{ine}}$ is the vector collecting the coupling body's inertia force and moment vectors, $\mathbf{f}_{\mathrm{ine}} = \{-\dot{\boldsymbol{\beta}};\ -\dot{\boldsymbol{\gamma}} - \dot{\mathbf{x}}\times\boldsymbol{\beta}\}$;
- $\mathbf{f}_{\mathrm{constr}} = \boldsymbol{\phi}_{/\mathbf{q}}^T\boldsymbol{\lambda}$ is the vector collecting the constraint force and moment vectors arising from the enforcement of the position and orientation of the coupling body, where $\boldsymbol{\phi} = \mathbf{0}$ is the array of constraint equations associated with subsystem coupling, $\boldsymbol{\phi}_{/\mathbf{q}} = \partial\boldsymbol{\phi}/\partial\mathbf{q}$, and $\boldsymbol{\lambda}$ the array of the corresponding Lagrange multipliers.

The expression of the generalized coupling force vector, $\mathbf{f}_C$, is finally obtained as:

$$\mathbf{f}_C = -\left(\mathbf{f}_{\mathrm{constr}} + \mathbf{f}_{\mathrm{ine}}\right) \tag{2}$$

The generalized coupling force vector is then specialized to each of the connected subsystems, e.g., $\mathbf{f}_{C,i}$ and $\mathbf{f}_{C,j}$ in the example considered for the algorithm definition.

So far, interfaces have been treated as rigid-body-like, even when connecting subsystems that exhibit flexibility. This approach aligns with the goal of minizing the interface to simplify coupling and reduce the amount of exchanged information. When flexibility is explicitly accounted for in non-local formulations, such as the Floating Frame of Reference (FFR) formulation combined with Component Mode Synthesis (CMS), the proposed approach remains applicable and retains its effectiveness, provided that each FFR-CMS element is confined within a single subsystem. Although MBDyn supports FFR-CMS elements through the `modal joint`, problems of this type have not been considered so far.

[3] Namely, matrix $\mathbf{a}\times$ multiplied by vector $\mathbf{b}$ yields vector $\mathbf{a}\times\mathbf{b}$, with

$$\mathbf{a}\times = \begin{bmatrix} 0 & -a_3 & a_2 \\ a_3 & 0 & -a_1 \\ -a_2 & a_1 & 0 \end{bmatrix}.$$

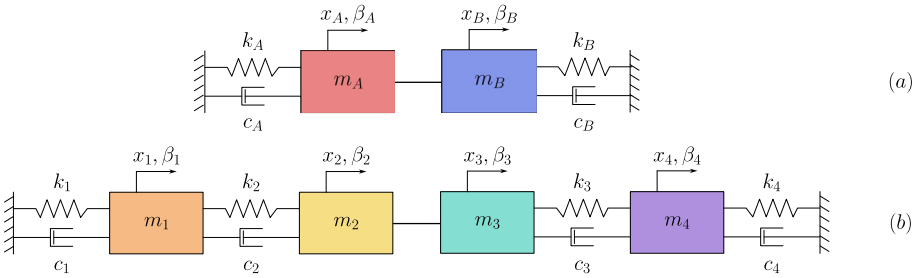


Fig. 7 Reference problem for stability analysis

Table 1 Baseline values of linear multi-oscillator parameters

Property	Symbol	Value
Mass	$m_1 = m_2$	2 kg
Mass	$m_3 = m_4$	1 kg
Spring stiffness	$k_1 = k_2 = k_3 = k_4$	10^3 N · m^{-1}
Damper constant	$c_1 = c_2 = c_3 = c_4$	10^1 N · s · m^{-1}

An advantage of the proposed approach is that the coupling variables are obtained though algebraic operations on quantities typically available to the user within simulation environments. No manipulation of the underlying structure of the equations or of the solver is required, allowing its application in a wide variety of solvers and computational environments. A detailed exemplification is given in Sect. 4 about the proposed implementations.

In the formulation of the algorithm, the kinematic quantities exchanged between subsystems include position, orientation (expressed in terms of the orientation vector), and linear and angular velocity. When enforcing the state of the coupling bodies, either the full set of kinematic quantities or only position and orientation may be employed. The two approaches are outlined and compared in the next section, while implementation details are reported in the Appendix.

3 Stability assessment

Figure 7(a) shows two linear systems used to evaluate the stability of the coupling. The system composed by two masses, m_A and m_B, presented in [12], is a variant of the typical systems used to study the stability of co-simulations, usually consisting of multiple single-mass oscillators coupled by additional springs and dampers representing the connection between subsystems (see for example [14]).

In the present system, bodies A and B are split in two parts (bodies 1 and 2, and 3 and 4, respectively), each pair connected by a spring and a damper, to produce a more general case in which multiple subsystems, comprising at least two bodies, interact (Fig. 7(b)). As shown in Table 1, the value of masses 1 and 2 is increased to 2 kg to make the problem more general, avoiding perfect symmetry and forcing the Lagrange multiplier to be generally non-zero; all other physical parameters are unchanged. Its monolithic first-order Differential-Algebraic

Table 2 Eigenvalues with the nominal parameter values of Table 1

Number	Eigenvalue, Radian/s	Frequency, Hz	Damping factor
1, 2	−1.9236e+00 ± j1.5787e+01	2.5126	0.12095
3, 4	−1.1525e+01 ± j3.2535e+01	5.1782	0.33389
5, 6	−1.1552e+01 ± j4.5594e+01	7.2565	0.24560

Equations (DAE) of motion are

$$\begin{aligned} \mathbf{M}\dot{\mathbf{x}} &= \boldsymbol{\beta} \\ \dot{\boldsymbol{\beta}} + \mathbf{C}\dot{\mathbf{x}} + \mathbf{K}\mathbf{x} + \phi_{/\mathbf{x}}^{T}\lambda &= \mathbf{0} \\ \phi &= 0 \quad \text{which for linearity corresponds to } \phi_{/\mathbf{x}}\mathbf{x} = 0 \end{aligned} \tag{3}$$

where

$$\mathbf{M} = \operatorname{diag}(m_1, m_2, m_3, m_4) \tag{4a}$$

$$\mathbf{K} = \begin{bmatrix} k_1 + k_2 & -k_2 & 0 & 0 \\ -k_2 & k_2 & 0 & 0 \\ 0 & 0 & k_3 & -k_3 \\ 0 & 0 & -k_3 & k_3 + k_4 \end{bmatrix} \tag{4b}$$

$$\mathbf{C} = \begin{bmatrix} c_1 + c_2 & -c_2 & 0 & 0 \\ -c_2 & c_2 & 0 & 0 \\ 0 & 0 & c_3 & -c_3 \\ 0 & 0 & -c_3 & c_3 + c_4 \end{bmatrix} \tag{4c}$$

$$\phi_{/\mathbf{x}} = \begin{bmatrix} 0 & 1 & -1 & 0 \end{bmatrix} \tag{4d}$$

This is a system of index-3 DAEs [15]. Table 1 reports the default parameters used throughout the analysis. The problem has 3 degrees of freedom and 6 (3 pairs of complex conjugated) eigenvalues, whose values with nominal parameters are shown in Table 2.

Without loss of generality, the equations of motion of the co-simulated problem are integrated using the linear, multistep (two-step) method (LMS2) with tunable algorithmic dissipation[4] originally presented in [16] and detailed in [11, 17].

The co-simulation system is generated following the procedure described in Sect. 2.1.2, using the rigid connection between bodies 2 and 3 as the partitioning element for the subsystems (Fig. 8).

Owing to linearity, the system can be reformulated as a recurrence equation,

$$\mathbf{z}_{n+1} = \boldsymbol{\Lambda}\mathbf{z}_n \tag{5}$$

where $\mathbf{z}$ collects the state of the problem at two consecutive time steps. The core of the process is outlined in the Appendix, with details fully developed in [18].

The spectral radius of matrix $\boldsymbol{\Lambda}$ is evaluated for different combinations of physical parameters and coefficients of the solver, to obtain the stability region of the coupling. Each

[4]The default integration method designed for, and implemented in, MBDyn.

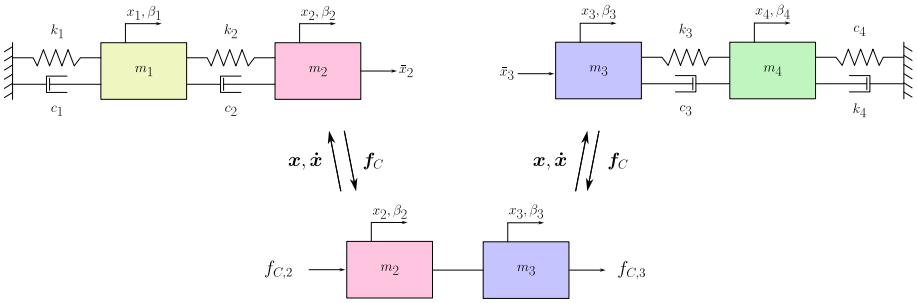


Fig. 8 Reference problem: co-simulation partitioning

plot shows the influence on stability of two parameters, while the others are set according to their baseline values, indicated in Table 1. A time step $h = 10^{-3}$ s and an asymptotic spectral radius $\rho_\infty = 0.6$ are used in the analysis.

The stability region corresponds to sets of parameters that satisfy the inequality

$$\rho = \max(|\lambda_i|) \leq 1 \qquad \forall i = 1..n \tag{6}$$

where λ_i are the eigenvalues of matrix $\mathbf{\Lambda}$.

Before addressing the stability analysis, two types of kinematic constraints are compared to evaluate the influence of the set of coupling variables on stability and accuracy. In the first case, the exchanged quantities include both position and velocity, whereas in the second case only position is exchanged. Accuracy is assessed by comparing the error in selected reference quantities against the analytical solution of the system, while stability is evaluated through the computation of the spectral radius for several combinations of physical and integration-scheme parameters. The corresponding results are not reported, as the two approaches exhibit nearly identical behavior for all the configurations considered. On the basis of this equivalent performance, the position-only formulation is retained in the following. This choice does not limit the generality of the proposed formulation. Rather, it reflects an implementation-oriented design decision, motivated by the additional complexity associated with enforcing both position and velocity constraints in general, and in the current MBDyn implementation in particular, discussed later, as well as by the increased data exchange required by the full-state approach.

Figure 9 shows an example of the output of the stability analysis. The plot on the left (labeled 'Literature') shows the stability region of the algorithm presented in [12], while the one on the right ('Proposed') illustrates the stability region obtained using the proposed algorithm. In this particular case, the parameter under analysis is the ratio between the masses of the interface bodies, which, as one can notice, has a significant impact in both cases on the algorithmic stability. Each mass value spans from $10^{-3} \times m_0$ to $10^3 \times m_0$, where m_0 is the baseline value. The substantial difference between the two stability regions can also be obtained for other combinations of physical parameters.

The effect of stiffness and damping on the instability region is shown in Fig. 10, where all stiffness and damping characteristics are scaled by 10^{-1} and 10^1 relative to their baseline values. The trend can be explained by considering the original system shown in Fig. 7(a) evaluated with the reference algorithm [12]. By studying the stability of the implicit problem, the following inequality can be obtained, which expresses the necessary condition for

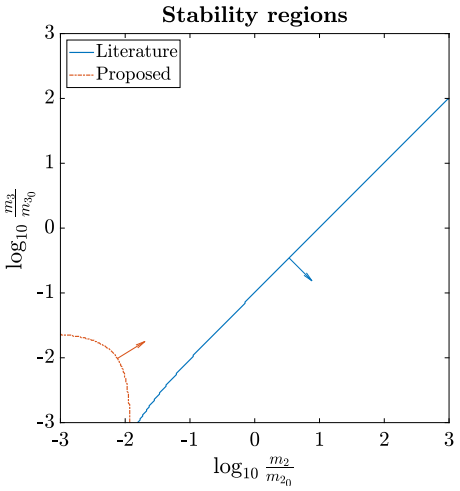


Fig. 9 Stability matrix, m_2 vs. m_3; the arrows indicate the stable region

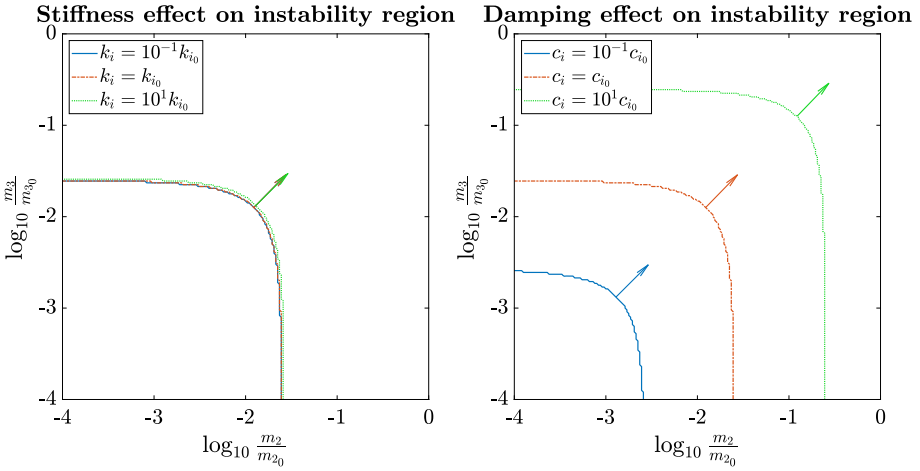


Fig. 10 Instability region variation with stiffness (left) and damping (right); the arrows indicate the stable region

all the eigenvalues of the recurrence matrix to be less than (or equal to) one:

$$m_2 + hb_0c_2 + h^2b_0^2k_2 < m_1 + hb_0c_1 + h^2b_0^2k_1 \tag{7}$$

When the physical parameters and the timestep are chosen in the same range of the baseline values of Table 1, the relative influence of mass, stiffness, and damping on the coupling stability becomes clear. Figure 11 shows the stability region obtained by changing both interface masses values along with stiffness and damping, both for the loose and tight schemes. Finally, the effect of the integrator parameters, namely, timestep and spectral radius at infinity (ρ_∞), is studied and the result is shown in Fig. 12.

Two relevant facts emerge from this analysis.

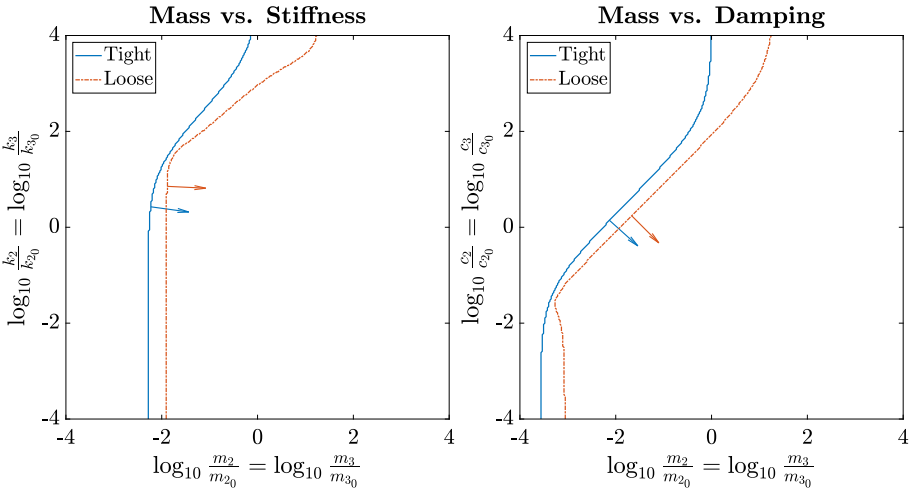


Fig. 11 Stability matrix varying mass, stiffness and damping; the arrows indicate the stable region

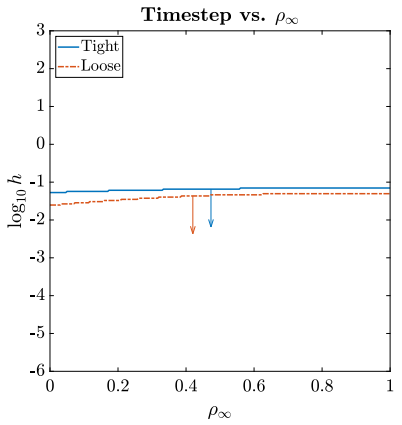


Fig. 12 Stability matrix timestep and ρ_∞; the arrows indicate the stable region

1. The stable region of the proposed algorithm is broader than that shown by similar algorithms found in the literature.
2. The difference between the proposed tight and loose schemes is marginal, making the much more efficient loose scheme a promising candidate for real-time applications.

Another important parameter affecting both stability and accuracy is the order of extrapolation of the state derivative. Higher-order extrapolation improves the accuracy of the coupling, but has a detrimental effect on stability.

4 Implementation

4.1 MBDyn

This section presents relevant details of the implementation of the proposed method in the free, general-purpose multibody solver MBDyn. As discussed in the following, only few

aspects required specific software development, which could be incorporated as run-time loadable user-defined functions, without the need to modify or recompile the core code. The rest was obtained programmatically using existing modeling and execution scheduling functionalities.

4.1.1 Inter-process communication

MBDyn provides primitives for interprocess communication via UNIX `local` and `inet` sockets, suitable for co-simulation by exchanging information for each time step:

- to receive input at the beginning of the solution phase, right after prediction and right before residual assembly and testing, based on `stream` drives;
- to send output at the end of the solution, right after reaching convergence and right before preparing for a new time step, based on `output elements`.[5]

Both primitives operate in terms of abstract channels. Both primitives support the implementation of user-defined functions as run-time loadable modules, to manipulate the contents of the channels without the need to modify the core code of the solver.

Each stream drive can receive an arbitrary number of channels and make them available to any entity (e.g., force components, prescribed displacement and rotation components, prescribed anelastic strain components in constitutive laws, etc.) that supports an input signal. The desired signal is identified by the name of the stream and the number of the channel.

In the present context, a stream that exposes the predicted prescribed pose of a structural node receives 6 channels, 3 for the position components and 3 for the orientation vector components. A stream that exposes the reaction and inertia forces acting on an interface node as computed by a subsystem according to Eq. (2) receives 6 channels, 3 for the force and 3 for the moment components.

Each output element can send an arbitrary number of channels. The values can be defined with great flexibility, from raw values of states to complex mathematical and logical expressions involving the values of internal states and virtually any other parameter of the simulation.

In the present context, an element that exports the predicted state of a structural node produces 6 channels, made available by a user-defined function that manipulates the state of the node at step k to consistently predict its position and orientation vector at step $k+1$, as discussed in Sect. 4.1.2. An element that exports the reaction and inertia forces acting on an interface node as computed by a subsystem according to Eq. (2) constructs the values of the components of vector $\mathbf{f}_C$ from the reaction forces of the related `total pin joint` (See Sect. 4.1.3) and the inertia forces obtained from the momentum and momenta moment derivatives of the related structural node.

4.1.2 Node state prediction

MBDyn exposes — and supports pushing to a peer solver — the solution of a time step after solution convergence and before prediction for a new time step. The proposed co-simulation needs predicted values for step $k+1$; however, when after step k's convergence the solver has the opportunity to expose its state, the predicted value is not yet available. Fortunately, MBDyn also supports the implementation of user-defined components as run-time loadable modules. A specific user-defined module, `module-cosim-output`, has

[5]The output primitive was originally derived from the `element` class because it provides versatile entry points to most simulation phases, although it does not directly contribute to the equations of the problem.

been developed to expose the predicted state of structural nodes. The module predicts the state of a structural node at step $k+1$ based on the state at step k and previous steps and makes such information available as signals for output streams. Its code is now included in MBDyn's public repository.[6]

The state derivative is predicted using an extrapolation based on cubic Hermitian functions from the values of the state and its derivative at steps k and $k-1$ as

$$\dot{\mathbf{y}}_{k+1}^{(0)} = \frac{1}{h}\left(m'_f(1)\mathbf{y}_k + m'_i(1)\mathbf{y}_{k-1}\right) + n'_f(1)\dot{\mathbf{y}}_k + n'_i(1)\dot{\mathbf{y}}_{k-1} \tag{8}$$

where the cubic Hermitian functions m_f, m_i, n_f, and n_i of the non-dimensional time $\tau \in [-1, 0]$ are

$$\begin{aligned} m_f(\tau) &= -2\tau^3 - 3\tau^2 + 1 & m_i(\tau) &= 2\tau^3 + 3\tau^2 \\ n_f(\tau) &= \tau^3 + 2\tau^2 + \tau & n_i(\tau) &= \tau^3 + \tau^2 \end{aligned} \tag{9}$$

The prime operator, $(\cdot)'$, indicates derivative with respect to τ, and h is the time step, $h = t_{k+1} - t_k$, such that $\clubsuit = \mathrm{d}\clubsuit/\mathrm{d}t = (\mathrm{d}\clubsuit/\mathrm{d}\tau)\cdot(\mathrm{d}\tau/\mathrm{d}t) = \clubsuit'/h$. The derivatives of the functions in Eqs. (9), evaluated at $\tau = 1$, i.e., at time t_{k+1}, yield

$$m'_f(1) = -12 \qquad m'_i(1) = 12 \qquad n'_f(1) = 8 \qquad n'_i(1) = 5 \tag{10}$$

The prediction of the state is performed using the second-order accurate, linear, multistep (two-step) method (LMS2) with tunable algorithmic dissipation presented in [11, 17], namely

$$\mathbf{y}_{k+1}^{(0)} = a_1\mathbf{y}_k + a_2\mathbf{y}_{k-1} + h\left(b_0\dot{\mathbf{y}}_{k+1}^{(0)} + b_1\dot{\mathbf{y}}_k + b_2\dot{\mathbf{y}}_{k-1}\right) \tag{11}$$

where the predicted state derivative of Eq. (8), $\dot{\mathbf{y}}_{k+1}^{(0)}$, is used. The coefficients of the method are

$$a_1 = 1 - \beta \qquad a_2 = \beta \qquad b_0 = \delta + \frac{1}{2} \qquad b_1 = \frac{\beta}{2} + \frac{1}{2} - 2\delta \qquad b_2 = \frac{\beta}{2} + \delta \tag{12}$$

with

$$\beta = \frac{3\,(1-\rho_\infty)^2 + 4\,(2\rho_\infty - 1)}{4 - (1-\rho_\infty)^2} \qquad \delta = \frac{(1-\rho_\infty)^2}{2\left(4 - (1-\rho_\infty)^2\right)} \tag{13}$$

where ρ_∞ is the method's asymptotic spectral radius, namely the largest norm root of the characteristic polynomial of the method applied to the problem $\dot{y} = \mathrm{j}\omega y$ for $\mathrm{j} = \sqrt{-1}$, $\omega \in \mathbb{R}$, and $h \to +\infty$.

It is worth recalling that this method corresponds to the (L-stable) second-order Backward Differentiation Formula (BDF2) when $\rho_\infty = 0$; according to [11], $\rho_\infty \approx 0.6$ is a valid trade-off between accuracy and algorithmic dissipation.

The above described prediction requires the converged state and state derivative at steps k and $k-1$. When the simulation starts, only the state and its derivative at step $k = 0$, the initial time, are available, a common problem of multistep methods, which earned them the

[6] https://public.gitlab.polimi.it/DAER/mbdyn.

non self-starting definition. In that case, the time derivative of a linear prediction is used to predict the state derivative, namely

$$\dot{\mathbf{y}}_1 = \dot{\mathbf{y}}_0 \tag{14}$$

and the Crank-Nicolson method is used to predict the state

$$\mathbf{y}_1 = \mathbf{y}_0 + h\frac{\dot{\mathbf{y}}_0 + \dot{\mathbf{y}}_1}{2} = \mathbf{y}_0 + h\dot{\mathbf{y}}_0 \tag{15}$$

preserving the second-order accuracy of state prediction. Owing to the particular choice of the state derivative prediction, the state prediction corresponds to integrating with explicit Euler, which is not algorithmically stable. However, this is only used for the first step, preserving the second-order accuracy of the prediction.

Orientation prediction requires special handling to guarantee that the resulting orientation matrix is orthogonal. For this purpose,

- the orientation at step k is considered as a reference; as a consequence, the corresponding relative orientation vector is by definition $\tilde{\boldsymbol{\theta}}_k \equiv \mathbf{0}$;
- the orientation at step $k-1$ is expressed in the reference frame of step k as the relative orientation matrix $\tilde{\mathbf{R}}_{k-1} = \mathbf{R}_k^T \mathbf{R}_{k-1}$ from the orientation matrices $\mathbf{R}_k$ and $\mathbf{R}_{k-1}$ at steps k and $k-1$;
- the corresponding relative orientation vector $\tilde{\boldsymbol{\theta}}_{k-1}$ is computed from the relative orientation matrix, $\tilde{\mathbf{R}}_{k-1}$, as $\tilde{\boldsymbol{\theta}}_{k-1} = \mathrm{ax}(\log(\tilde{\mathbf{R}}_{k-1}))$, where operator $\mathrm{ax}(\clubsuit\times) \equiv \clubsuit$ extracts the underlying vector $\clubsuit$ from the skew-symmetric matrix $\clubsuit\times$;
- the time derivative of the relative orientation vector, $\dot{\tilde{\boldsymbol{\theta}}}_{k-1}$, is computed from the corresponding angular velocity vector, $\boldsymbol{\omega}_{k-1}$ transformed in the reference frame of step k, as $\dot{\tilde{\boldsymbol{\theta}}}_{k-1} = \mathbf{G}(\tilde{\boldsymbol{\theta}}_{k-1})^{-1}\mathbf{R}_k^T\boldsymbol{\omega}_{k-1}$, where matrix $\mathbf{G}$ maps the angular velocity to the derivative of the orientation parameters, $\boldsymbol{\omega} = \mathbf{G}(\boldsymbol{\theta})\dot{\boldsymbol{\theta}}$; since matrix $\mathbf{G}$ is singular for $\|\boldsymbol{\theta}\| = 2n\pi$, with $n \in \mathbb{Z}$, using the orientation at step k as a reference is intended to keep $\left\|\tilde{\boldsymbol{\theta}}\right\|$ limited ($\left\|\tilde{\boldsymbol{\theta}}\right\| \ll 1$);
- the time derivative of $\tilde{\boldsymbol{\theta}}_k$ is $\dot{\tilde{\boldsymbol{\theta}}}_k = \mathbf{G}(\tilde{\boldsymbol{\theta}}_k)^{-1}\mathbf{R}_k^T\boldsymbol{\omega}_k \equiv \mathbf{R}_k^T\boldsymbol{\omega}_k$, since $\tilde{\boldsymbol{\theta}}_k \equiv \mathbf{0}$ and by construction $\mathbf{G}(\mathbf{0}) \equiv \mathbf{I}$;
- the time derivative of the relative orientation vector at step $k+1$, $\dot{\tilde{\boldsymbol{\theta}}}_{k+1}^{(0)}$, is predicted using Eq. (8), with $\mathbf{0}$, $\tilde{\boldsymbol{\theta}}_{k-1}$, $\boldsymbol{\omega}_k$, and $\dot{\tilde{\boldsymbol{\theta}}}_{k-1}$ in lieu of $\mathbf{y}_k$, $\mathbf{y}_{k-1}$, $\dot{\mathbf{y}}_k$, and $\dot{\mathbf{y}}_{k-1}$;
- the relative orientation vector at step $k+1$, $\tilde{\boldsymbol{\theta}}_{k+1}^{(0)}$, is predicted using Eq. (11), with the above predicted value for $\dot{\tilde{\boldsymbol{\theta}}}_{k+1}^{(0)}$;
- the predicted relative orientation matrix at step $k+1$ is constructed as $\tilde{\mathbf{R}}_{k+1}^{(0)} = \mathbf{R}(\tilde{\boldsymbol{\theta}}_{k+1}^{(0)}) = \exp(\tilde{\boldsymbol{\theta}}_{k+1}^{(0)}\times)$ using Rodriguez's formula;
- the predicted absolute orientation matrix at step $k+1$ is constructed as $\mathbf{R}_{k+1}^{(0)} = \mathbf{R}_k\tilde{\mathbf{R}}_{k+1}^{(0)}$;
- the predicted absolute orientation vector at step $k+1$, $\boldsymbol{\theta}_{k+1}^{(0)}$, as needed in Sect. 4.1.3, is finally extracted from the corresponding absolute orientation matrix, $\boldsymbol{\theta}_{k+1}^{(0)} = \mathrm{ax}(\log(\mathbf{R}_{k+1}^{(0)}))$.

4.1.3 Predicted pose enforcement

The position and orientation of the interface nodes in the subsystems — in short, their pose — are prescribed using the `total pin joint` element. This constraint prescribes a

node's absolute position and orientation, or some components of the corresponding vectors. Prescribing the position is relatively straightforward. Orientation prescription is a bit more tricky.

The solver computes the prescribed orientation matrix, $\mathbf{R}^{(0)}$, from the corresponding prescribed orientation vector, $\boldsymbol{\theta}^{(0)}$ (i.e., the predicted orientation vector of Sect. 4.1.2), namely $\mathbf{R}^{(0)} = \mathbf{R}(\boldsymbol{\theta}^{(0)})$, using Rodriguez's formula. The relative orientation matrix between the node's orientation matrix, $\mathbf{R}$, and the prescribed orientation matrix, $\mathbf{R}^{(0)}$, is computed as $\tilde{\mathbf{R}} = \mathbf{R}^T\mathbf{R}^{(0)}$, and the corresponding relative orientation vector, $\tilde{\boldsymbol{\theta}}$, is then extracted. This relative orientation vector is finally used to construct the orientation constraint equation as

$$\tilde{\boldsymbol{\theta}} = \mathbf{0} \tag{16}$$

More details are available in [19].

4.2 Other solvers

To assess the behavior of the proposed co-simulation algorithm in a mixed environment, where different solvers employing different integration schemes are involved, an alternative formulation of the coupling subsystem has been prototyped as a Python script and integrated using various numerical schemes. The connected subsystems are simulated in MBDyn using the second-order accurate, linear, two-step method (LMS2) with tunable algorithmic dissipation presented in [11, 17]. For this example, the selected asymptotic spectral radius is 0.6; in the following plots, this integration scheme is denoted as 'LMS2, 0.6'.

The coupling subsystem is handled outside MBDyn and solved using different numerical schemes. First, the same scheme employed for the connected subsystems (i.e., 'LMS2, 0.6') is applied to verify the consistency of the solution with respect to a co-simulation involving three MBDyn instances. Two additional schemes are then considered: the first-order accurate Backward-Euler method (denoted as 'BE') and a fourth-order Runge-Kutta scheme (denoted as 'RK4', with and without force interpolation). Figure 13 compares the solutions obtained in the mixed environment, the MBDyn-only co-simulation, and the monolithic solution, all evaluated against the analytical solution of the problem. As expected, the monolithic solution, the MBDyn-only co-simulation, and the mixed solution using 'LMS2, 0.6' produce nearly identical results. In contrast, the use of the other two integration schemes has a significant impact on the error magnitude, in accordance with their nominal order of accuracy: the error increases by approximately four orders of magnitude when using BE, while RK4 yields a slight reduction when the force is linearly interpolated within the time step, where the RK algorithm evaluates the problem at intermediate times, or a substantial increase without interpolation.[7]

5 Applications

The following case studies are investigated to test the algorithm on more complex problems and quantify the time gain achievable by the coupling. In addition, for each application,

[7] Similar results were obtained using higher-order explicit Runge-Kutta integration schemes in *Simscape Multibody*, where the absence of force interpolation yielded errors larger than those obtained with 'LMS2, 0.6' in monolithic, MBDyn-only co-simulation, and mixed solution configurations.

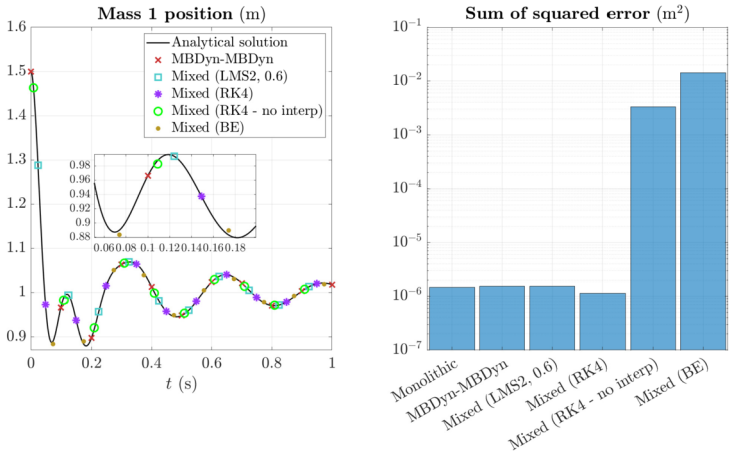


Fig. 13 Reference problem, monolithic vs. co-simulated (MBDyn-MBDyn and Python-MBDyn with different integrators)

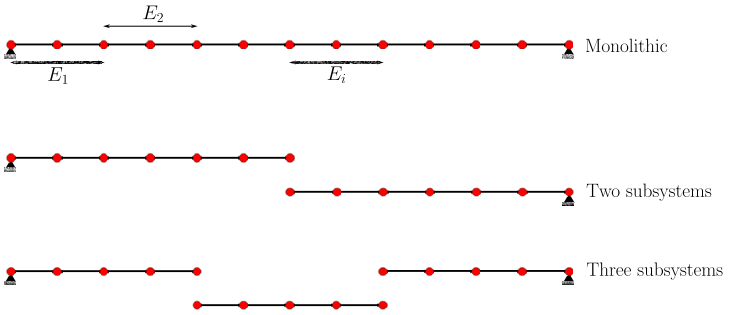


Fig. 14 Beam model, partitioned into two and three subsystems

different tests are conducted to validate and assess the accuracy of the simulation by comparing the results with the monolithic solution. All problems are tested on machines of different performance levels: an Asus Vivobook Pro (Intel i7-8750H @ 2.2-4.1 GHz, 6 cores), an Alienware Aurora R13 (Intel i9-12900KF @ 2.4-5.2 GHz, 16 cores) and an assembled desktop (AMD Ryzen 9 7950X @ 4.5-5.7 GHz, 16 cores)

5.1 Beam model

The first problem is a simply supported beam, illustrated in Fig. 14. Its geometric and material properties and the corresponding stiffness and inertia properties are summarized in Table 3. The number of equations of the model is varied by increasing the number of beam elements used to discretize the structure, respectively $n_b = 6$, 12, and 24. MBDyn's geometrically exact three-node finite-volume beam element is used [20, 21]. A monolithic model is developed for each mesh refinement, along with two co-simulation arrangements, dividing the system into two and three parts, respectively. In the three cases, the monolithic model respectively requires $n_n = 2n_b + 1 = 13$, 25, and 49 structural nodes and produces

Table 3 Beam properties

Property	Symbol	Value
Length	ℓ	24 m
Section moment of inertia	i_{xx}	0.1 kg · m^2/m
	i_{yy}	0.05 kg · m^2/m
	i_{zz}	0.05 kg · m^2/m
Axial stiffness	EA	10^6 N
Shear stiffness	$GA_y = GA_z$	$0.6 \cdot 10^6$ N
Torsional stiffness	GJ	10^3 N m^2
Bending stiffness, vertical	EJ_y	$2 \cdot 10^3$ N m^2
Bending stiffness, lateral	EJ_z	10^4 N m^2
Linear density	$m = \rho A$	5 kg m^{-1}
Damping factor	ξ	0.05

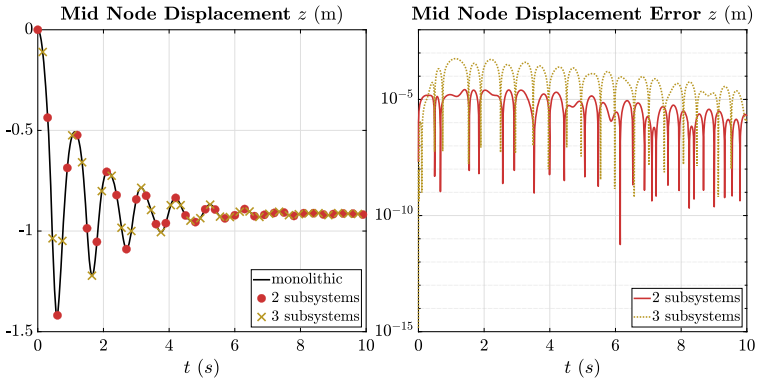


Fig. 15 Beam simulation, monolithic vs. co-simulation

$12n_n + 10 = 166$, 310, and 598 equations, respectively, where 10 is the number of constraints corresponding to the two revolute joints at the ends of the beam, each eliminating 5 degrees of freedom. The objective is to verify how the execution time scales as the number of subsystems is varied, while comparing the accuracy of the partitioned solutions with the monolithic one.

The beam starts in its undeformed configuration and reaches equilibrium under its own weight. Figure 15 shows the displacement of the middle node of the beam and the error between the monolithic and the partitioned solutions. Table 4 reports the required CPU time. Each simulation lasts 10 s, with a time step $h = 10^{-4}$ s and 10^5 time steps.

This example shows how a substantial speedup can be achieved with co-simulation. Each subsystem's size is smaller than that of the monolithic system; the subsystems are solved in parallel on multiple cores. The simulation wall clock time is nearly inversely proportional to the number of parts in which the system is partitioned. On the slower machine, Table 4a, the gain is more noticeable.

The wall clock time increases almost linearly with the problem size because a very efficient sparse linear solver [22] is used for the factorization of the problem's Jacobian matrix within the Newton-Raphson correction phase of the solution. Considering the structure of

Table 4 Beam model results

n. of beams, n_b	6	12	24
(a) Asus — CPU time, s.			
monolithic	28.19	52.19	100.26
2 subsystems	14.64	24.08	49.29
3 subsystems	10.69	18.04	34.11
(b) Alienware — CPU time, s.			
monolithic	8.81	15.40	30.74
2 subsystems	5.63	9.68	18.11
3 subsystems	4.09	7.20	13.02

Table 5 Shell properties

Property	Symbol	Value
Length	ℓ	1 m
Width	w	0.2 m
Thickness	t	0.001 m
Young's modulus	E	$6.825 \cdot 10^7$ Pa
Poisson's ratio	ν	0.3

the problem and the bandedness and sparsity of the Jacobian matrix, its latter property is very well preserved in the factored matrix, justifying the observed scalability.

5.2 Shell model

The shell model of Fig. 5 has been implemented as an example of partitioning at a discretized one-dimensional interface. Table 5 shows the physical properties of the system. The middle nodes, located at $x = 0$, are loaded with a total force $F(t)$ oriented along the positive z axis: half, i.e., $F(t)/2$, on the mid-node and one quarter, i.e., $F(t)/4$, on each of the side nodes. $F(t)$ is a smooth step, going from zero to 20 N in 1 s as $[1 - \cos(\pi t)]/2$. The deformation and the difference between the monolithic and partitioned solutions are shown in Fig. 16. The execution times of Table 6 show the same essentially linear speedup observed for the beam of Sect. 5.1.

5.3 Vehicle model

The main case study is a vehicle model for DiL simulations, implemented in MBDyn (Fig. 17(a)). The selected vehicle is the 2022 Formula Student car from Dynamis PRC[8]. The DP12evo was chosen for its extensive set of available parameters. Formula Student is an international competition born in the 1980s that challenges students in the field of engineering to design, manufacture, and race single-seater Formula-style cars.[9] The difference in mass and inertia between parts, e.g., between the car body and the wheel sets, and the high rigidity of several components make the system numerically stiff, making this a challenging testbench for the algorithm's stability.

[8]Dynamis PRC, https://www.dynamisprc.com, last accessed April 2025.

[9]Formula Student Germany, https://www.formulastudent.de/fsg, last accessed April 2025.

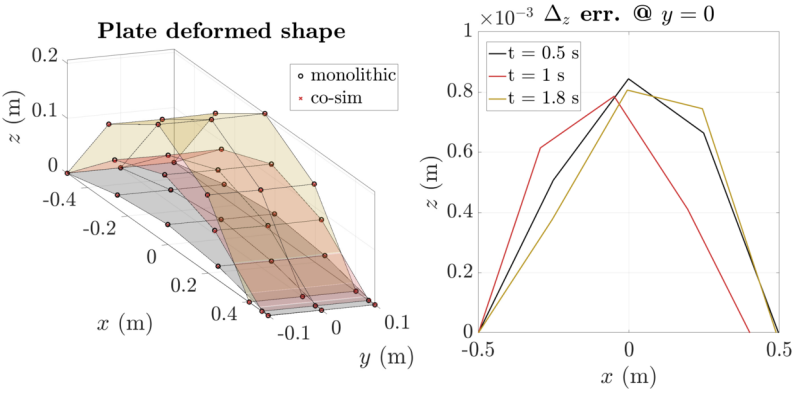


Fig. 16 Shell simulation, monolithic vs. co-simulation

Table 6 Shell model results

n. of shells, $n_x \times n_y$	4×2
(a) Asus — CPU time, s.	
monolithic	48.50 s
2 subsystems	23.76 s
(b) AMD desktop — CPU time, s.	
monolithic	28.4 s
2 subsystems	13.7 s

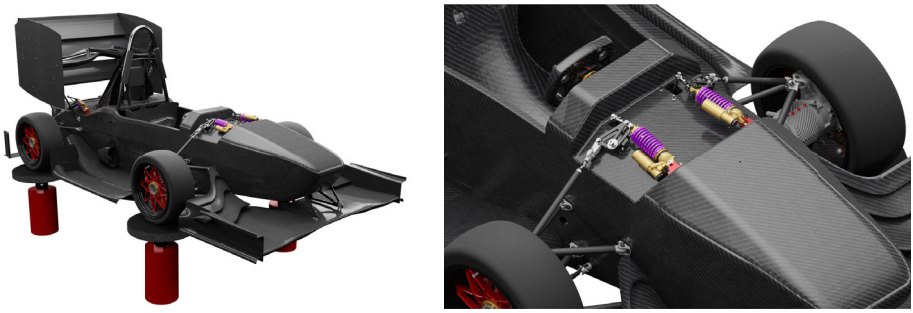
The modeling is essential, but it comprises all the fundamental elements of a vehicle model. Each subsystem is optimized to minimize the number of equations while providing a good representation of the underlying physical system.

All suspended masses (including the driver) are collected in a single rigid body (the chassis), while non-suspended masses are divided in two parts, comprising rotating (rims, hubs, brake disks, electric motor rotors etc.) and non-rotating components (hub carriers, brake calipers, motor stators and cooling jackets, etc.). All suspension members (wishbones, rods, and links) are modeled by means of distance constraints due to their negligible mass and corresponding negligible contribution to inertia forces. Elastic members include linear springs, anti-roll bars, and nonlinear shock absorbers (Fig. 17(b)).

The rims are connected to the wheel hubs by revolute joints. The steering mechanism is achieved by connecting the two inner ball joints of the tie rods to a rack, which is then linked to the chassis with a prismatic joint. Brakes and motor torques act as internal torques between each rim and the corresponding wheel hub. A specific built-in joint, the `brake`, is used to achieve the desired braking behavior through a LuGre friction model [23].

The tire model, named *MFtire*, has been developed from scratch as a user-defined element and included in MBDyn's source code[10] as `module-MFtire`. A Pacejka-like model [24] is used for the calculation of forces and moments. The contributions of the tire to the problem's Jacobian matrix have been developed to improve convergence and stability.

[10] https://public.gitlab.polimi.it/DAER/mbdyn.



(a) Full vehicle model. (b) Detail of front suspensions.

Fig. 17 Views of the vehicle

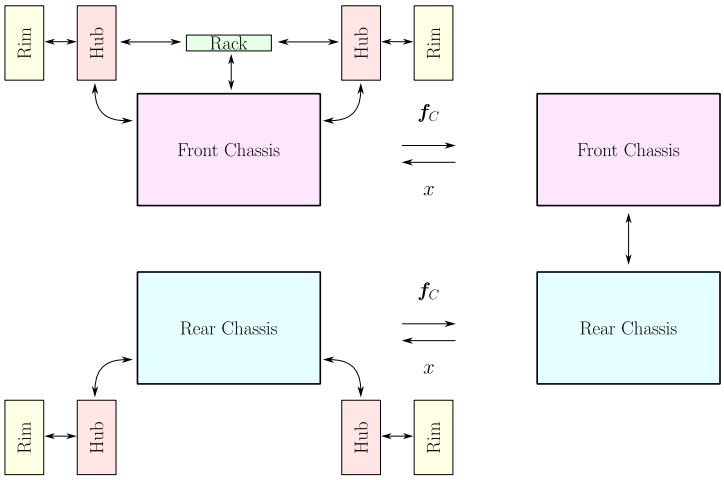


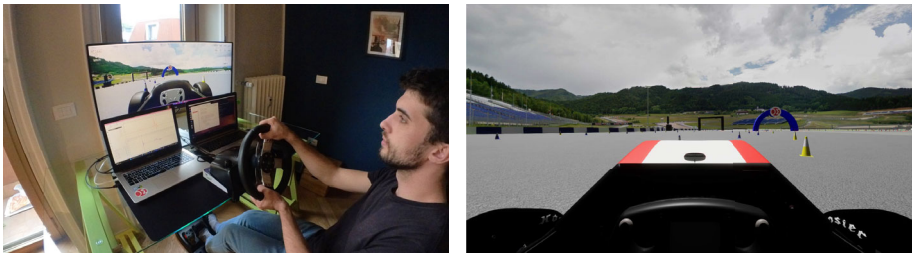
Fig. 18 Vehicle model co-simulation scheme

Aerodynamic forces and moments are computed as functions of the front and rear ride heights and applied to the chassis.

An open-loop step steer manoeuvre is conducted to investigate the lateral transient response of the vehicle following the procedure described in ISO 7401:2011 [25]. The result of the simulation is shown in Fig. 20. The vehicle is accelerated up to the test speed, 100 km/h, and then a step input is applied, whose magnitude is selected to obtain a target steady-state lateral acceleration of 0.8 g.

In this example, the chassis is divided in two parts to simplify the architecture of the co-simulation, since multiple elements connect the left and right side of the vehicle, as shown in Fig. 18. For further performance improvement, the chassis could be partitioned into four parts.

A Simulink model allows to visualize simulation results offline and for interaction with the operator in real-time simulations. Simulink's *Vehicle Dynamics Blockset* provides blocks to create customized scenes using *Unreal Engine*. The only input required is the vehicle motion (position and orientation of the chassis and each wheel). Due to the additional overhead



(a) DiL setup.

(b) On-board view of the vehicle.

Fig. 19 System implementation

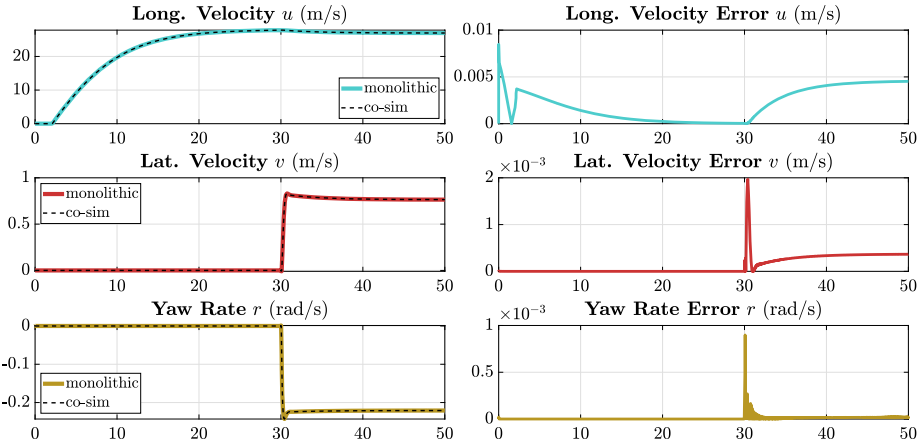


Fig. 20 Comparison of velocity and angular velocity components during a step steer manoeuver

introduced by the graphics engine, two computers connected via TPC socket are used, one for simulation and one for visualization, as shown in Fig. 19a.

For DiL simulations, MBDyn's Human Interface Devices (HID) module (`module-hid`) has been successfully integrated in the model and configured and tested with two devices that were not previously available, a PlayStation 4 joystick and a Logitech G27 steering wheel and pedals kit. The steering wheel offers greater accuracy and the possibility of returning force feedback, making it ideal for more realistic simulations, while the joystick controller is better suited for rapid testing and less complex setups.

As anticipated, co-simulating the vehicle model saves nearly 50% of simulation time. The model has been consistently executed in soft real-time at a $3 \cdot 10^{-4}$ s time step (Alienware Aurora R13). The time step must be chosen considering that at max speed the number of steps per wheel revolution, n_{rev}, must be sufficiently large for accuracy (e.g., $n_{\text{rev}} \geq 100$, thus $h_{\max} \leq T_{\min}/n_{\text{rev}} = \Omega_{\max}/(2\pi n_{\text{rev}}) = v_{\max}/(2\pi R n_{\text{rev}})$, where R is the wheel radius, Ω the angular velocity and v the speed of the vehicle).

MBDyn supports real-time scheduling using POSIX primitives. Periodic scheduling is achieved by waiting on calls to `clock_nanosleep()` before advancing to the next step.

6 Conclusions

This work presents the use of co-simulation to parallelize the solution of mechanical systems (primarily multibody systems), aiming to reduce solution times in particular for models that require fast, real-time execution, such as simulators and Human- and Hardware-in-the-Loop (HiL) problems. The case studies demonstrate that, with an efficient solver like the one implemented in MBDyn (whose time complexity can be almost linear with respect to model size), the scheme achieves time savings nearly proportional to the number of subsystems into which the system is divided. A comprehensive stability analysis of the coupling was conducted on a test model to identify the most influential parameters, revealing an ample stability region.

Several applications are presented to demonstrate the effects of the coupling scheme on complex multibody systems. By appropriately dividing the system into subsystems (typically using symmetries or repeated elements) significant time savings are achieved while maintaining stability.

Appendix: Derivation of recurrence matrix

To obtain the recurrence matrix $\mathbf{\Lambda}$, the equations of motion are integrated using LMS2, Eq. (11), and further discussed in Sect. 4.1.2. The state vector $\mathbf{y}$ collects the position, linear momentum and Lagrange multiplier variables. The subscript n indicates the time step. All constant terms are collected into matrices A_1, A_2 and A_3,

$$\mathbf{y}_{1,n} = \mathbf{A}_1\mathbf{y}_{1,n-1} + \mathbf{A}_2\mathbf{y}_{1,n-2} + \mathbf{A}_3\mathbf{y}_{C,n} \tag{17}$$

Similarly, for subsystem 2:

$$\mathbf{y}_{2,n} = \mathbf{B}_1\mathbf{y}_{2,n-1} + \mathbf{B}_2\mathbf{y}_{2,n-2} + \mathbf{B}_3\mathbf{y}_{C,n} \tag{18}$$

$\mathbf{y}_{C,n}$ is the state vector of the coupling subsystem, which contains the position of the coupling bodies enforced by the algebraic constraint. The coupling force vector is assembled by means of Eq. (2) and the resulting expression is the following:

$$\mathbf{F}_n = \begin{bmatrix} \mathbf{A}_4 \\ \mathbf{0} \end{bmatrix} \mathbf{y}_{1,n} + \begin{bmatrix} \mathbf{0} \\ \mathbf{B}_4 \end{bmatrix} \mathbf{y}_{2,n} \qquad = \mathbf{A}_5\mathbf{y}_{1,n} + \mathbf{B}_5\mathbf{y}_{2,n} \tag{19}$$

The same operation is carried out for the coupling subsystem, which yields:

$$\mathbf{y}_{C,n} = \mathbf{C}_1\mathbf{y}_{C,n-1} + \mathbf{C}_2\mathbf{y}_{C,n-2} + \mathbf{C}_3\mathbf{F}_{n-1} + \mathbf{C}_4\mathbf{F}_{n-2} + \mathbf{C}_5\mathbf{F}_n \tag{20}$$

To obtain the expression of the loose coupling recurrence matrix, the starting point is the computation of an initial guess for the coupling subsystem's state, which is computed by means of the first derivatives of Hermite polynomials (similarly to how prediction is treated in MBDyn, as discussed in Sect. 4.1.2).

$$\begin{aligned} \dot{\mathbf{y}}_C^G = -\frac{12}{h}\left(\mathbf{y}_{C,n-1} - \mathbf{y}_{C,n-1}\right) + 8\left(\mathbf{M}_{C1}\mathbf{y}_{C,n-1} + \mathbf{M}_{C2}\mathbf{F}_{n-1}\right) \\ +5\left(\mathbf{M}_{C1}\mathbf{y}_{C,n-2} + \mathbf{M}_{C2}\mathbf{F}_{n-2}\right) \end{aligned} \tag{21}$$

The predicted state derivative is substituted in the integration scheme and the state, $\mathbf{y}_{C,n-i}$, and coupling force vector, $\mathbf{F}_{n-i}$, at the previous time steps $i = 1, 2$ are collected, yielding

$$\mathbf{y}_C^G = \mathbf{G}_1\mathbf{y}_{C,n-1} + \mathbf{G}_2\mathbf{y}_{C,n-2} + \mathbf{G}_3\mathbf{F}_{n-1} + \mathbf{G}_4\mathbf{F}_{n-2} \tag{22}$$

The resulting system of equations is

$$\begin{aligned}
\mathbf{y}_{1,n} &= \mathbf{A}_1\mathbf{y}_{1,n-1} + \mathbf{A}_2\mathbf{y}_{1,n-2} + \mathbf{A}_3\left(\mathbf{G}_1\mathbf{y}_{C,n-1} + \mathbf{G}_2\mathbf{y}_{C,n-2} + \mathbf{G}_3\mathbf{F}_{n-1} + \mathbf{G}_4\mathbf{F}_{n-2}\right)\\
\mathbf{y}_{2,n} &= \mathbf{B}_1\mathbf{y}_{2,n-1} + \mathbf{B}_2\mathbf{y}_{2,n-2} + \mathbf{B}_3\left(\mathbf{G}_1\mathbf{y}_{C,n-1} + \mathbf{G}_2\mathbf{y}_{C,n-2} + \mathbf{G}_3\mathbf{F}_{n-1} + \mathbf{G}_4\mathbf{F}_{n-2}\right)\\
\mathbf{F}_n &= \mathbf{A}_5\mathbf{y}_{1,n} + \mathbf{B}_5\mathbf{y}_{2,n}\\
\mathbf{y}_{C,n} &= \mathbf{C}_1\mathbf{y}_{C,n-1} + \mathbf{C}_2\mathbf{y}_{C,n-2} + \mathbf{C}_3\mathbf{F}_{n-1} + \mathbf{C}_4\mathbf{F}_{n-2} + \mathbf{C}_5\mathbf{F}_n
\end{aligned} \tag{23}$$

Constant terms are collected as follows:

$$\begin{aligned}
\begin{bmatrix} \mathbf{I} & \mathbf{0} & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{I} & \mathbf{0} & \mathbf{0} \\ -\mathbf{A}_5 & -\mathbf{B}_5 & \mathbf{I} & \mathbf{0} \\ \mathbf{0} & \mathbf{I} & -\mathbf{C}_5 & \mathbf{I} \end{bmatrix} \mathbf{y}_n = &\begin{bmatrix} \mathbf{A}_1 & \mathbf{0} & \mathbf{A}_3\mathbf{G}_3 & \mathbf{A}_3\mathbf{G}_1 \\ \mathbf{0} & \mathbf{B}_1 & \mathbf{B}_3\mathbf{G}_3 & \mathbf{B}_3\mathbf{G}_1 \\ \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \mathbf{C}_3 & \mathbf{C}_1 \end{bmatrix} \mathbf{y}_{n-1}\\
&+ \begin{bmatrix} \mathbf{A}_2 & \mathbf{0} & \mathbf{A}_3\mathbf{G}_4 & \mathbf{A}_3\mathbf{G}_2 \\ \mathbf{0} & \mathbf{B}_1 & \mathbf{B}_3\mathbf{G}_4 & \mathbf{B}_3\mathbf{G}_2 \\ \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \mathbf{C}_4 & \mathbf{C}_2 \end{bmatrix} \mathbf{y}_{n-2}
\end{aligned} \tag{24}$$

which can be reorganized as

$$\mathbf{\Lambda}_1\mathbf{y}_n = \mathbf{\Lambda}_2\mathbf{y}_{n-1} + \mathbf{\Lambda}_3\mathbf{y}_{n-2} \tag{25}$$

where $\mathbf{y}_n = [\mathbf{y}_{1,n};\ \mathbf{y}_{2,n};\ \mathbf{F}_n;\ \mathbf{y}_{C,n}]$. After defining $\mathbf{z}_n = [\mathbf{y}_n;\ \mathbf{y}_{n-1}]$, Eq. (5) becomes

$$\mathbf{z}_n = \begin{bmatrix} \mathbf{\Lambda}_1^{-1}\mathbf{\Lambda}_2 & \mathbf{\Lambda}_1^{-1}\mathbf{\Lambda}_3 \\ \mathbf{I} & \mathbf{0} \end{bmatrix} \mathbf{z}_{n-1} \tag{26}$$

Author contributions A.F. developed the formulation, performed the numerical analysis, and drafted the manuscript; P.M. contributed to the conceptualization, supervised the research, and drafted and revised the manuscript; M.M. co-supervised the study and revised the manuscript.

Funding information Open access funding provided by Politecnico di Milano within the CRUI-CARE Agreement. The research described in this article received no funding.

Data availability No datasets were generated or analysed during the current study.

Declarations

Competing interests The third author, Pierangelo Masarati, is an Associate Editor of Multibody System Dynamics but has not been involved in the review of this manuscript. The authors have no other competing interests to declare that are relevant to the content of this article.

References

1. Rulka, W., Pankiewicz, E.: MBS approach to generate equations of motions for HiL-simulations in vehicle dynamics. Multibody Syst. Dyn. **14**, 367–386 (2005). https://doi.org/10.1007/s11044-005-1144-8
2. Macklin, M., Müller, M., Chentanez, N.: XPBD: position-based simulation of compliant constrained dynamics. In: Proceedings of the 9th International Conference on Motion in Games, pp. 49–54 (2016). https://doi.org/10.1145/2994258.2994272
3. Attolico, M., Masarati, P.: A multibody user-space hard real-time environment for the simulation of space robots. In: Fifth Real-Time Linux Workshop, Valencia, Spain, pp. 9–11 (2003)
4. Bae, D.-S., Hwang, R.-S., Haug, E.J.: A recursive formulation for real-time dynamic simulation of mechanical systems. J. Mech. Des. **113**, 158–166 (1991). https://doi.org/10.1115/1.2912764
5. García de Jalón, J., Bayo, E.: Kinematic and Dynamic Simulation of Multibody Systems: The Real Time Challenge. Springer, New York (1994). https://doi.org/10.1007/978-1-4612-2600-0
6. Busch, M.: Zur effizienten Kopplung von Simulationsprogrammen. Kassel University Press GmbH (2012)
7. Hafner, I., Popper, N.: On the terminology and structuring of co-simulation methods. In: Proceedings of the 8th International Workshop on Equation-Based Object-Oriented Modeling Languages and Tools, pp. 67–76 (2017). https://doi.org/10.1145/3158191.3158203
8. Gomes, C., Thule, C., Broman, D., Larsen, P.G., Vangheluwe, H.: Co-simulation: a survey. ACM Comput. Surv. **51**(3) (2018). https://doi.org/10.1145/3179993
9. Hafner, I., Popper, N.: An overview of the state of the art in co-simulation and related methods. Simul. Notes Eur. **31**(4), 185–200 (2021). https://doi.org/10.11128/sne.31.on.10582
10. Association, M.: FMI Functional Mock-up Interface Specification, 3.0.2 edn. (2024). https://github.com/modelica/fmi-standard/releases/download/v3.0.2/FMI-Standard-3.0.2.zip
11. Masarati, P., Morandini, M., Mantegazza, P.: An efficient formulation for general-purpose multibody/multiphysics analysis. J. Comput. Nonlinear Dyn. **9**(4), 041001 (2014). https://doi.org/10.1115/1.4025628
12. Zhang, R., Zhang, H., Zanoni, A., Wang, Q., Masarati, P.: A tight coupling scheme for smooth/non-smooth multibody co-simulation of a particle damper. Mech. Mach. Theory **161**, 104181 (2021). https://doi.org/10.1016/j.mechmachtheory.2020.104181
13. Morandini, M., Masarati, P.: Implementation and validation of a 4-node shell finite element. In: ASME IDETC/CIE 2014, Buffalo, NY, August 17–20 (2014). https://doi.org/10.1115/DETC2022-88995
14. Schweizer, B., Li, P., Lu, D.: Explicit and implicit cosimulation methods: stability and convergence analysis for different solver coupling approaches. J. Comput. Nonlinear Dyn. **10**(5), 051007 (12 pages) (2015). https://doi.org/10.1115/1.4028503
15. Brenan, K.E., La, S., Campbell, V., Petzold, L.R.: Numerical Solution of Initial-Value Problems in Differential-Algebraic Equations. North-Holland, New York (1989). https://doi.org/10.1137/1.9781611971224
16. Masarati, P., Lanz, M., Mantegazza, P.: Multistep integration of ordinary, stiff and differential-algebraic problems for multibody dynamics applications. In: XVI Congresso Nazionale AIDAA, Palermo, 24–28 September, pp. 71.1–10 (2001)
17. Zhang, H., Zhang, R., Zanoni, A., Masarati, P.: Performance of implicit A-stable time integration methods for multibody system dynamics. Multibody Syst. Dyn. **54**(3), 263–301 (2022). https://doi.org/10.1007/s11044-021-09806-9
18. Fontana, A.: A real-time capable co-simulation algorithm for the solution parallelization of multibody systems. Master's thesis, Politecnico di Milano (2024)
19. Masarati, P.: A formulation of kinematic constraints imposed by kinematic pairs using relative pose in vector form. Multibody Syst. Dyn. **29**(2), 119–137 (2013). https://doi.org/10.1007/s11044-012-9320-0
20. Ghiringhelli, G.L., Masarati, P., Mantegazza, P.: Multibody implementation of finite volume C^0 beams. AIAA J. **38**(1), 131–138 (2000). https://doi.org/10.2514/2.933

21. Bauchau, O.A., Betsch, P., Cardona, A., Gerstmayr, J., Jonker, B., Masarati, P., Sonneville, V.: Validation of flexible multibody dynamics beam formulations using benchmark problems. Multibody Syst. Dyn. **37**(1), 29–48 (2016). https://doi.org/10.1007/s11044-016-9514-y
22. Morandini, M., Mantegazza, P.: Using dense storage to solve small sparse linear systems. ACM Trans. Math. Softw. **33**(1), 5, 1–12 (2007). https://doi.org/10.1145/1206040.1206045
23. Dupont, P., Hayward, V., Armstrong, B., Altpeter, F.: Single state elasto-plastic friction models. IEEE Trans. Autom. Control **47**(5), 787–792 (2002). https://doi.org/10.1109/TAC.2002.1000274
24. Pacejka, H.: Tire and Vehicle Dynamics. Elsevier, Amsterdam (2012). https://doi.org/10.1016/C2010-0-68548-8
25. Road vehicles—Lateral transient response test methods—Open-loop test methods. ISO 7401:2011