Automatic Partitioning and Simulation of Weakly Coupled Systems

Alessandro Vittorio Papadopoulos, Johan Åkesson, Francesco Casella, and Alberto Leva

***Abstract*— Many control engineering tasks nowadays rely on the simulation of complex multi-physics systems. Modern tools allow to build the required dynamic models conveniently, thanks to Object-Oriented Modelling languages, e.g., Modelica, and to perform simulations with hardly any additional effort on the part of the analyst. However, when simulation speed is of concern, the same tools fall short of exploiting some useful properties of the model, namely – to focus on the subject of this work – the possibility of partitioning said model in "weakly" coupled submodels. This work proposes an automatic method to perform a structural analysis aimed at identifying weak couplings in the system, providing the information needed for the mentioned partition. This information is here used to feed a mixed-mode integration method, leading to a significant improvement in terms of simulation speed.**

I. Introduction

Modelling and simulation of complex physical systems have received increasing attention in the last years, in particular in the control domain. Even if the control design is usually based on simple models – most often a linearised one is sufficient – the controller should be validated on a more accurate description of the real plant.

Moreover, several important control applications require to simulate accurate models in real time. Model reference adaptive controllers, for example, make use of a reference plant model as part of the online control law [1]. This requires the model to be simulated in real time in parallel with the plant, both being driven simultaneously by the same input signals.

The main problem, however, is that simulating accurate models usually takes a lot of time and computational resources, which in some cases is, e.g., in real-time applications, is not acceptable. Therefore, finding a way to improve simulation efficiency is thus crucial and worth investigating. In this work, a novel method to improve simulation performance is presented.

The paper is outlined as follows. Section II describes the related work, and specifies the scope of the paper. Section III presents a brief overview on classical integration methods used in real-time simulation. The proposed partitioning method is described in Section IV, while the mixed-mode integration method is described in Section VI. In Section VII the method is applied to some physical examples and the obtained results are discussed. Section VIII concludes the paper.

II. Related work and motivation

In recent years the introduction of new modelling techniques, e.g., Object-Oriented Modelling (OOM), has simplified and streamlined the task of building accurate description of complex dynamic systems. Such dynamic models usually yield large Differential Algebraic Equations (DAEs) systems, which, in many cases, have dynamic variables that evolve within different time-scales (a property often referred to as *stiffness*). There exist solvers able to cope with stiffness, e.g., Backward Differentiation Formulas (BDFs) methods [1], [2], but their computational burden most often prevents their use in real-time applications.

Moreover, neither implicit nor explicit methods can cope with stiff systems efficiently. Typically, explicit methods have to employ too small steps, since the fastest time-constant governs the stability of the integration method, while implicit methods have to solve, at each step, large nonlinear systems of equations.

An interesting solution has been presented in [3], based on the idea of zeroing out certain elements of the Jacobian, i.e., neglecting some couplings in the system, leading to a structure that can be exploited to improve simulation performance. Another interesting work is [4], which proposes a *mixed-mode* integration method, i.e., one that is in between of Explicit Euler and Implicit Euler, for real-time simulation. In [4] the system is partitioned into two subsystems through a heuristic criterion based on eigenvalues analysis of the linearised model. However, both the mentioned methods require some insight on the model in order to partition it in an effective manner.

On the other hand, there is a vast literature in the field of co-simulation [5]–[9], which is based on the idea that *monolithic* systems can be partitioned in different subsystems on the basis of some physical considerations, e.g., exploiting weak couplings. This allow to numerically integrate the subsystems separately having some communication infrastructure in charge of synchronizing the subsystems and managing the overall simulation.

The main limitation to this approach is that the partition has to be defined *a priori*, trying both to identify which are the parts of the model that are weakly coupled and to minimize the number of "communication variables" or "interfaces" among the subsystems. Moreover, it is not trivial to fit this framework to existing OOM and simulation tools [10].

Other methods aimed at simulation speedup, exploiting weakly coupled model components have been already proposed in the literature [7], [11], [12], but they are often domain-specific and the structural analysis needed is far from being automatic.

In this paper, an automatic procedure to identify subsystems evolving within different time-scales is presented. Since the proposed method is based on the analysis of the "dependency cycles" among the model dynamic variables, it is called *cycle analysis* method. The information coming

A.V. Papadopoulos, F. Casella and A. Leva are with Dipartimento di Elettronica, Informazione e Bioingegneria, Politecnico di Milano, Milano, Italy. {papadopoulos,casella,leva}@elet.polimi.it

J. Åkesson is with Department of Automatic Control, Lund University, Lund, Sweden. johan.akesson@control.lth.se

from the cycle analysis is thus used to improve simulation efficiency through a mixed-mode integration method.

III. Integration methods overview

To better understand some of the results presented in this work, a brief review of explicit and implicit integration methods and of their properties is in order.

Explicit single-step methods are widely used in real-time simulation, since they are keen to comply with real-time requirements. Their computational effort is relatively low and constant and the number of calculations per step can be easily estimated. Those methods can deal fairly well with discontinuous input signals, since they do not use information from the past. Thus, for non-stiff ordinary differential equations, explicit single-step methods are by common opinion the best choice.

Among them, Explicit Runge-Kutta (ERK) methods are the most widely adopted; the simplest ERK method is the Explicit Euler (EE) one. The numerical stability region of ERK methods are the interior of the curves shown in Fig. 1.

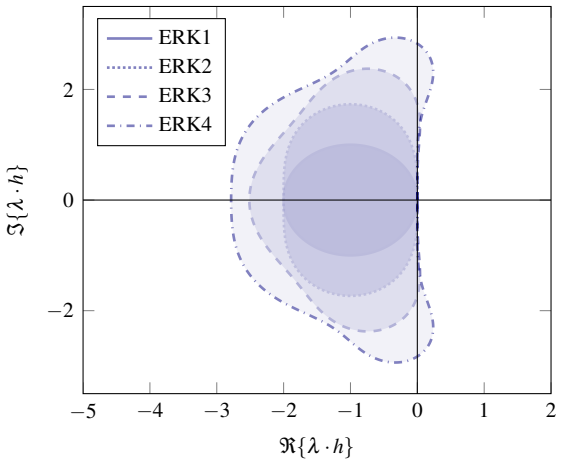

Fig. 1. Numerical stability domains of ERK (interior of the curves).

On the other hand, implicit methods require solving a system of (in general) nonlinear equations at each integration step, which implies the use of iterative methods, such as the Newton one. The computational effort for each step cannot be estimated reliably, as it depends on the (theoretically unbounded) number of iterations. Hence, implicit methods are not really suited for of real-time simulation, unless the number of iterations is kept bounded to a fixed value. Implicit Runge-Kutta (IRK) methods are widely adopted as implicit single-step integration methods and are suited for dealing with stiff systems; the simplest IRK method is Implicit Euler (IE).

IV. The partitioning method

Summarising for brevity, only low-order explicit methods seem well suited for real-time simulation. In the absence of stiffness, discontinuities, or badly nonlinear implicit equations, those methods work properly. But, as already stated in Section II, multi-physics systems are usually made of components exhibiting different time-scales. For example in mechatronical systems a slow mechanical part is often controlled by fast electric circuits or by a hydraulic drive.

The main idea of this work is to find a systematic way to separate the different dynamics present in the model and to numerically integrate them with different methods, trying to exploit the advantages of both explicit and implicit methods. The most important contribution of this paper is the proposal of an automatic procedure to identify the possible submodels.

A. Overview

Referring to Fig. 1, and focusing for a moment to linear systems, a first possible idea to perform the partitioning is the following. Acting on the choice of the integration step h of an explicit method, the system eigenvalues can be moved inside or outside its numerical stability region.

If one wants all the eigenvalues to lie inside the stability region, then the choice of h is limited by the faster dynamics. On the other hand, if a larger value of h is chosen, then some eigenvalues lie outside the numerical stability region. In this case two subsystems can be easily identified: a fast subsystem, associated with the outer eigenvalues, and a slow subsystem associated with the inner ones.

A straightforward solution to partition the model is thus to cut the subspace spanned by the eigenvectors associated with the fast eigenvalues. However, it is not always possible, because in the nonlinear case it involves a coordinate transformation at each integration step, as the linearisation is state-dependent. Moreover, the partition properties may change in time and this is is hard to fit with the *a priori* partition needed for co-simulation. As a consequence, state selection criteria are preferable, also in accordance with [4].

The methodology proposed herein takes a different viewpoint, and is aimed at associating a specific time-constant with each dynamic variable in the system, without performing an eigenvalue analysis.

Coming to the proposed method, let the state-space form of a continuous-time system

$$\dot{\mathbf{x}} = f(\mathbf{x}, \mathbf{u}) \tag{1}$$

that discretised with EE with an integration step h yields

$$\mathbf{x_{k+1}} = \mathbf{x_k} + h \cdot f(\mathbf{x_k}, \mathbf{u_k}). \tag{2}$$

Suppose now that (2) is at an asymptotic stable equilibrium, i.e., $\mathbf{x_{k+1}} = \mathbf{x_k}$. If a small perturbation is applied to a single state variable $x_\mathbf{k}$, a transient occurs, and two things may happen:

1) the perturbation affects the other state variables, without in turn re-affecting $x_\mathbf{k}$;
2) the perturbation, after some integration steps, re-affects $x_\mathbf{k}$.

In the first case, no numerical instability can be introduced by the numerical integration process, but in the second case there is a "dependency cycle" among some state variables that may lead to unstable behaviours of the integration algorithm, depending on how the perturbation propagates. Intuitively, if the perturbation is amplified along the dependency cycle, a numerical instability may occur.

The proposed method detects the dependency cycles that are present in the system, and defines conditions under which

this perturbation cannot lead to numerical instability, i.e., it is attenuated.

B. Preliminary definitions

Before going into the details of the method, some preliminary definitions are in order.

Definition 1: A *path* p in a digraph (or directed graph) $G=(N,E)$ is an ordered sequence of nodes such that from each of its nodes there is an edge to the next node in the sequence. Formally, letting L the length of the path, and $p(i)$ the i-th node in the path p,

$$\forall i \in \{1,\dots,L-1\} \forall p(i) \in p \quad \exists! \, (p(i),p(i+1)) \in E.$$

A path is denoted as $p = \langle n_{\text{start}}, \dots, n_i, \dots, n_{\text{end}} \rangle$.

Definition 2: A path with no repeated nodes is called a *simple path* (or *walk*).

Definition 3: A *simple cycle* c is a simple path which starts from a root node $i \in N$ and ends with the same node. The length of a simple cycle is L, where L is also the number of non-repeated nodes in the cycle.

It is intuitive from Definition 3 that the longest possible simple cycle in a digraph G with $|N|$ nodes, has length $|N|$ (Hamiltonian circuit). In the following, the terms *simple cycle* and *cycle* will be used indifferently.

It is also worth noticing that, given a cycle c, the cycles

$$\langle c(i), c(i+1), \dots, c(j), c(i) \rangle \equiv \langle c(j), c(i), c(i+1), \dots, c(j) \rangle$$

are equivalent. In fact, they actually represent the same cycle, since the same edges (and the same nodes) are involved.

C. Cycle analysis

The first step in the cycle analysis is to build the dependency digraph $G=(N,E)$ associated with the model. In particular, there is a node $n \in N$ for each state, and the set of edges $E \subseteq N \times N$ is formed as

$$e_{i,i} = 1 + h \cdot \frac{\partial f_i}{\partial x_j}, \quad e_{i,j} = h \cdot \frac{\partial f_i}{\partial x_j}, \; \forall i \neq j$$

where h is the integration step and the $\partial f_i / \partial x_j$ are the elements of the Jacobian of the continuous-time system. In other words, the Jacobian of (1) multiplied by h is the adjacency matrix of the weighted digraph, and accounts for the propagation entity of the disturbance from the i-th variable to the j-th one.

The second step in the cycle analysis is to detect the set $\mathscr{C}$ of all cycles contained in the directed graph. Hence, for every cycle $c \in \mathscr{C}$ detected in G a *cycle gain* is computed.

Definition 4: A *cycle gain* $\mu_c(h)$ of a cycle $c \in \mathscr{C}$ is

$$\mu_c(h) = \prod_{x_i, x_j \in c} e_{i,j} = \begin{cases} 1 + h \dfrac{\partial f_i}{\partial x_i} & \text{if } L = 1 \\ h^L \cdot \prod_{x_i, x_j \in c} \dfrac{\partial f_i}{\partial x_j} & \text{if } L > 1 \end{cases}$$

where $e_{i,j}$ are the edges involved in the cycles and L is the length of the cycle.

In other words, the cycle gain quantifies how much the disturbance given to a single state variable $x_i \in c$ is amplified along one cycle c. Apparently, by suitably constraining this quantity, we ensure that no numerical unstable behaviours can occur.

In fact, starting from the computed $\mu_c(h)$, *for each cycle* inequalities of the form $|\mu_c(h)| \leq \alpha$, yielding

$$\begin{cases} 0 < h \leq (1+\alpha) \left| \dfrac{\partial f_i}{\partial x_i} \right|^{-1} & \text{if } L = 1 \text{ and } \dfrac{\partial f_i}{\partial x_i} < 0 \\ 0 < h \leq \sqrt[L]{\alpha} \cdot \left| \prod_{x_i, x_j \in c} \dfrac{\partial f_i}{\partial x_j} \right|^{-\frac{1}{L}} & \text{otherwise.} \end{cases} \tag{3}$$

are introduced, where α is an upper bound on the allowed amplification of the disturbance entering the cycle (see Section IV-A). As a result, every cycle $c \in \mathscr{C}$ has been associated with a constraint on h, i.e., with an upper bound on the integration step which allows the perturbation not to be amplified along c.

Finally, each variable x_i is associated with the most restrictive constraint on h among the set of cycles $\mathfrak{C}_{x_i} = \{c \in \mathscr{C} | x_i \in c\}$. Formally,

$$\begin{aligned} &\text{maximize} \quad \overline{h}_i \\ &\text{subject to} \quad |\mu_c(h)| \leq \alpha, \quad \forall c \in \mathfrak{C}_{x_i}. \end{aligned}$$

The result of the cycle analysis is that each dynamic variable is associated with an upper bound of the integration step needed for a numerically stable integration, thus with a quantity related to its time-scale. As a result, the variables can be ordered, for example, by increasing value of $\overline{h}_i$ and presented to the analyst, who can decide how to cut the model.

In the following, to put the idea to work, a mixed-mode integration method is considered, where the model needs to be split into two subsystems integrated with a single step h (details in Section VI). The cut is thus defined with the choice of the integration step which practically defines a threshold over which the state variables can be considered "slow" – thus integrated with an explicit method – and under which the variables are considered "fast".

D. Remarks

A first remark is related to the information coming from the cycle analysis. In particular, it is worth noticing that cycle analysis dictates not only the time-scales associated with each state variable, but also which are the variables that are mutually interacting together, i.e., the cycles. This information is not exploited in this work for space limitation, but it can be used to identify independent components in the model – the strongly connected components of the dependency graph – to make the simulation code parallel.

A second remark deals with the computational complexity of the method. Unfortunately the problem of finding all the cycles in a digraph has complexity $\mathscr{O}\left(2^{|E|-|N|+1}\right)$, and is a well-known and studied problem in the operation research community [13]–[15]. This is of course a limitation with strongly connected digraphs, but sparse ones are more common in real applications, especially if weakly couplings exist. This is due to the fact that in many physical models there is a neat separation among the states coming from different physical domains.

On the other hand, it is worth stressing that cycle analysis must be done only once for a given model, during an offline phase, before the simulation is started. Spending more time

for an automatic structural analysis aimed at speeding up the simulation is an acceptable tradeoff.

A last remark is related to the choice of α. This is the unique parameter of the method. It controls the tradeoff between the accuracy of the resulting simulation, and how much the system can be decoupled. In particular, the lower the α, the higher the accuracy of the resulting simulation, the lower the capability of the method to identify weakly coupled components, and vice-versa.

A good "default" choice of α is intuitively 1, since we want the perturbation to be attenuated along the cycle, i.e., we want the cycle gain $|\mu_c| < 1$. There are, however, some cases in which the choice of α can be critical, and in particular in the case of loosely damped systems.

V. Loosely damped models and stability issues

Referring again to the stability region in Fig. 1 for EE (the smallest one), it is apparent that complex eigenvalues with a low damping factor can easily lead to numerical instability.

Consider, for example, a linear, time-invariant and autonomous system of the form:

$$\dot{\mathbf{x}} = \begin{bmatrix} -\omega_n\xi & -\omega_n\sqrt{1-\xi^2} \\ \omega_n\sqrt{1-\xi^2} & -\omega_n\xi \end{bmatrix} \mathbf{x} \tag{4}$$

with the natural frequency $\omega_n > 0$ and damping factor $0 \leq \xi \leq 1$, which has two complex conjugate eigenvalues

$$\lambda_{1,2} = -\omega_n \cdot \left(\xi \pm \imath\sqrt{1-\xi^2}\right).$$

If the eigenvalues of the corresponding discrete-time system computed with EE are computed, the stability condition is

$$h < 2\frac{\xi}{\omega_n} := \overline{h}_s \tag{5}$$

On the other hand, the obtained cycle gains are

$$\begin{aligned} \mu_{\langle 1,1\rangle} &= e_{1,1} = 1 + h\frac{\partial f_1}{\partial x_1} = 1 - h\omega_n\xi \\ \mu_{\langle 2,2\rangle} &= e_{2,2} = 1 + h\frac{\partial f_2}{\partial x_2} = 1 - h\omega_n\xi \\ \mu_{\langle 1,2,1\rangle} &= e_{1,2}\cdot e_{2,1} = h^2 \cdot \frac{\partial f_1}{\partial x_2}\cdot\frac{\partial f_2}{\partial x_1} = -h^2\omega_n^2(1-\xi^2). \end{aligned}$$

which lead to the following constraints

$$\begin{aligned} |\mu_{\langle 1,1\rangle}| < \alpha \quad &\Rightarrow \quad h \leq \frac{1+\alpha}{\omega_n\xi} := \overline{h}_{c,1} \\ |\mu_{\langle 2,2\rangle}| < \alpha \quad &\Rightarrow \quad h \leq \frac{1+\alpha}{\omega_n\xi} := \overline{h}_{c,2} \\ |\mu_{\langle 1,2,1\rangle}| < \alpha \quad &\Rightarrow \quad h \leq \frac{1}{\omega_n}\cdot\sqrt{\frac{\alpha}{1-\xi^2}} := \overline{h}_{c,3} \end{aligned} \tag{6}$$

Comparing the upper bounds as $\overline{h}_s \leq \overline{h}_{c,i}$, i.e., investigating when the cycle analysis gives an upper bound greater than the actual stability condition as a function of α, leads to

$$\begin{cases} 2\dfrac{\xi}{\omega_n} \leq \dfrac{1+\alpha}{\omega_n\xi} \\ 2\dfrac{\xi}{\omega_n} \leq \dfrac{1}{\omega_n}\cdot\sqrt{\dfrac{\alpha}{1-\xi^2}} \end{cases} \quad \Rightarrow \quad \begin{cases} \alpha \geq 2\xi^2 - 1 \\ \alpha \geq 4\xi^2\left(1-\xi^2\right) \end{cases}$$

The conditions on α depend only on the damping factor ξ. In particular, for $\xi \to 0$ also $\alpha \to 0$, i.e., the integration step should tend to 0 as the damping factor decreases. The highlighted area in Fig. 2 represents the region of values α that preserve the stability of the system.

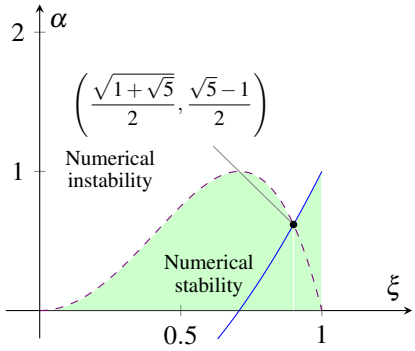

Fig. 2. Stability conditions on the parameter α w.r.t. ξ.

VI. Mixed-mode integration method

One of the possibilities opened by the information coming from the cycle analysis is to perform a mixed-mode integration method. In fact, if the states are ordered by decreasing value of the associated time-constant, the model can be cut in "slow" and "fast" dynamics, as anticipated, by simply choosing a the fixed integration step used in the simulation. Accordingly to the remarks of Section III, the fast dynamics can be integrated with an implicit method (IE is used in the following as an example), while for the slow dynamics an explicit method (here, EE) can be used with a larger h with respect to a pure explicit one. Therefore, with these example choice, the discrete-time system associated with the continuous-time one reads as

$$\begin{cases} \mathbf{x}^s_{\mathbf{k}+1} = \mathbf{x}^s_{\mathbf{k}} + h\cdot f\left(\mathbf{x}^s_{\mathbf{k}}, \mathbf{x}^f_{\mathbf{k}}, \mathbf{u}_{\mathbf{k}}\right) \\ \mathbf{x}^f_{\mathbf{k}+1} = \mathbf{x}^f_{\mathbf{k}} + h\cdot f\left(\mathbf{x}^s_{\mathbf{k}+1}, \mathbf{x}^f_{\mathbf{k}+1}, \mathbf{u}_{\mathbf{k}+1}\right) \end{cases}$$

This means that the fast component $\mathbf{x}^f_{\mathbf{k}+1}$ can be computed considering $\mathbf{x}^s_{\mathbf{k}+1}$ as an input. Fig. 3 shows the resulting mixed-mode integration scheme.

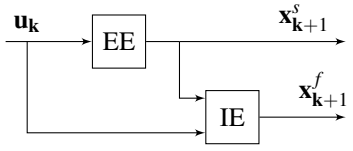

Fig. 3. Explicit/Implicit Euler integration scheme.

VII. Physical examples

In the following two examples coming from physical modelling are presented, showing the effectiveness of the approach. The design parameter α is set to 1. The simulations were performed using Assimulo[1], fed with models written in Modelica and exported through JModelica into Functional Mock-up interface (FMI) descriptions [16].

[1] http://www.jmodelica.org/assimulo

A. DC motor

Consider a simple physical system, i.e., a DC motor. The motor can be represented by a third order model of the form:

$$\begin{cases} L\cdot\dot{I} &= -R\cdot I - k_m\cdot\omega + u(t) \\ J\cdot\dot{\omega} &= k_m\cdot I - b\cdot\omega - \tau(t) \\ \dot{\varphi} &= \omega \end{cases} \tag{7}$$

where $L = 3\,\text{mH}$ is the armature inductance, $R = 50\,\text{m}\Omega$ is the armature resistance, $J = 1500\,\text{kg}\,\text{m}^2$ is the inertia, $b = 0.001\,\text{kg}\,\text{m}^2/\text{s}$ is the friction coefficient, and $k_m = 6.785\,\text{V}\,\text{s}$ is the electro-motorical force (EMF) constant of the motor. These parameter values correspond to those of a real system. The inputs are the armature voltage, $u(t)$, and the torque load, $\tau(t)$, respectively. In the given example, $u(t)$ is 500 V, and the torque is of 2500 N m.

The cycle analysis leads to the following constraints:

$$\begin{aligned} I:&\quad h \le 0.032 \\ \omega:&\quad h \le 0.221 \\ \varphi:&\quad h \le 0.500 \end{aligned}$$

hence, choosing an integration step $h = 0.2$ leads to a partition of the system that is natural, separating the electrical components from the mechanical variables. Fig. 4 shows the simulation results.

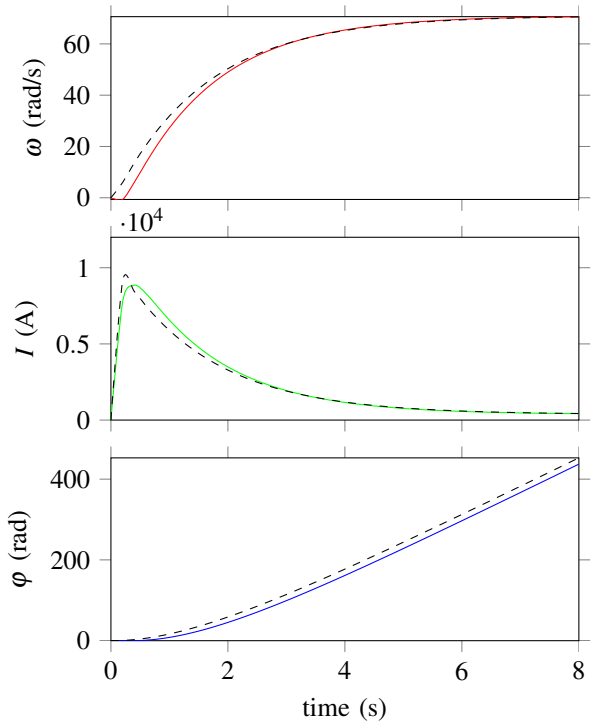

Fig. 4. Simulation results of Model (7). Dashed lines represent the real trajectories, while the solid lines the mixed-mode ones.

Table I shows the simulation statistics for different integration methods. It is worth noticing that the dimension of the system the Newton iteration has to solve is reduced from 3 to 1 in the mixed-mode method. Notice, also that the EE method needs a smaller step size ($h = 0.01$) for numerical stability reasons. Apparently, the mixed-mode method performs better than the others also in this very simple case.

TABLE I
SIMULATION STATISTICS FOR MODEL (7).

	Mixed-mode	**BDF**	**IE**	**EE**
# Steps	**40**	255	**40**	800
# Function ev.	123	283	**122**	–
# Jacobian ev.	**2**	5	**2**	–
# Fun. ev. in Jac. ev.	**4**	15	8	–
# Newton iterations	83	279	**82**	–
# Newton fail	0	0	0	–
Accuracy	**1.139**	–	1.531	1.876
Sim time	**0.03s**	0.09s	0.05s	0.22s

B. Counterflow heat exchanger

This example refers to a counterflow heat exchanger with two incompressible streams (Fig. 5). Both streams and

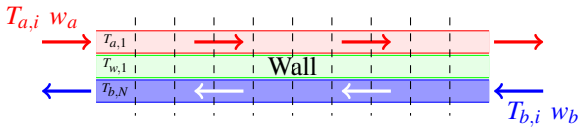

Fig. 5. Counterflow heat exchanger scheme.

the interposed wall are spatially discretised with the finite volume approach, neglecting axial diffusion in the wall – as is common practice – and also in the streams, as zero-flow operation is not considered for simplicity. Taking ten volumes for both streams and the wall, with the same spatial division (again, for simplicity) leads to a nonlinear dynamic system of order 30, having as boundary conditions the four pressures at the stream inlets and outlets, and the two temperatures at the inlets. More precisely, the system is given by

$$\begin{cases} c_a\dfrac{M_a}{N}\dot{T}_{a,i} = w_a c_a\cdot(T_{a,i-1}-T_{a,i}) + \dfrac{G_a}{N}\cdot(T_{w,i}-T_{a,i}) \\ c_w\dfrac{M_w}{N}\dot{T}_{w,i} = -\dfrac{G_a}{N}\cdot(T_{w,i}-T_{a,i}) - \dfrac{G_b}{N}\cdot(T_{w,i}-T_{b,N-i+1}) \\ c_b\dfrac{M_b}{N}\dot{T}_{b,i} = w_b c_b\cdot(T_{b,i-1}-T_{b,i}) + \dfrac{G_b}{N}\cdot(T_{w,N-i+1}-T_{b,i}) \end{cases} \tag{8}$$

where T stands for temperature, w for mass flowrate, c for (constant) specific heat, M for mass, and G for thermal conductance; the a, b and w subscripts denote respectively the two streams and the wall, while $i \in [1, N]$ ($i = 0$ for boundary conditions) is the volume index, counted for both streams from inlet to outlet, the wall being enumerated like stream a.

TABLE II
PARAMETER VALUES OF MODEL (8).

Parameters					
N	10	M_b	1 kg	c_w	3500 J/(kg K)
$T_{a,\text{in}}$	323.15 K	M_w	10 kg	G_a	8000 W/K
$T_{b,\text{in}}$	288.15 K	c_a	4200 J/(kg K)	G_b	8000 W/K
M_a	0.1 kg	c_b	3500 J/(kg K)		

Notice that in this example there is no neat physical separation between the dynamics. In fact, the cycle analysis

leads to the following constraints:

$$\begin{aligned} T_{a,i}: &\quad h \leq 0.0084 \\ T_{b,i}: &\quad h \leq 0.0478 \\ T_{w,1,10}: &\quad h \leq 0.0478 \\ T_{w,2,9}: &\quad h \leq 0.0498 \end{aligned} \qquad \begin{aligned} T_{w,3,8}: &\quad h \leq 0.0524 \\ T_{w,4,7}: &\quad h \leq 0.0559 \\ T_{w,5,6}: &\quad h \leq 0.0606 \end{aligned}$$

By choosing an integration step $h = 0.04$ yields a partition of the system that considers the $T_{a,i}$ as the fast while the $T_{b,i}$ and $T_{w,i}$ as the slow states. Fig. 6 shows the simulation results — notice that the temperatures are reported with different scales.

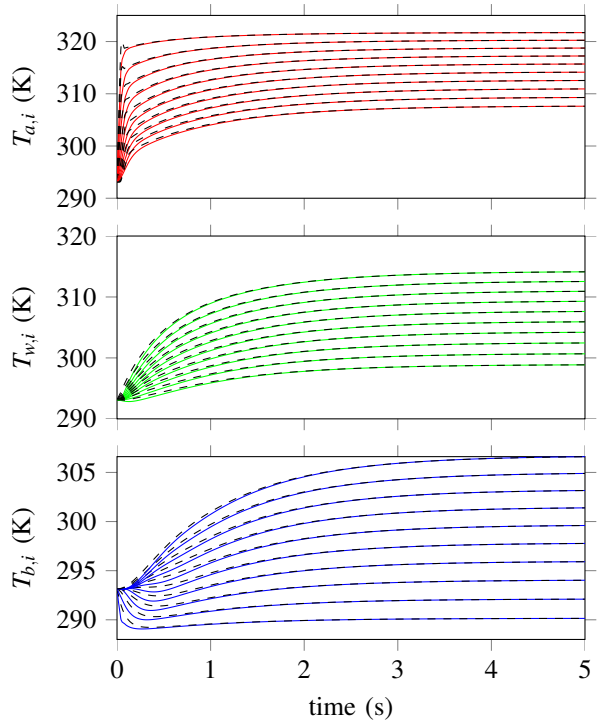

Fig. 6. Simulation results of (8). Dashed lines represent the real trajectories, while the solid lines the mixed-mode ones.

Table III shows the simulation statistics for different integration methods. It is worth noticing that, since the complexity of IE is $\mathcal{O}(n^3)$, where n is the dimension of the model, integrating implicitly only $T_{a,i}$ instead of the whole model, reduces the computations from $30^3 = 27000$ to $10^3 = 1000$, leading to a significant improvement in terms of simulation efficiency. Notice also that the EE method needs a smaller step size ($h = 0.01$) for numerical stability reasons.

VIII. Conclusion and Future Work

A method to automatically partition a dynamic model for efficient simulation was presented based on the idea of *cycle analysis*. Peculiar to the method is the capability of providing the analyst with a clearly interpretable parameter to govern the partition procedure, related to the time-scale of the different model's dynamics. Once that parameter is set, the selection of fast and slow states is straightforward and automatic. As a result, the partitioned system can be used either to take advantage of mixed-mode integration, i.e., for real-time simulation, or even as a baseline for co-simulation.

TABLE III
Simulation statistics for Model (8) ($h = 0.04$).

	Mixed-mode	**BDF**	**IE**	**EE**
# Steps	**125**	260	**125**	500
# Function ev.	375	**296**	375	–
# Jacobian ev.	6	**5**	6	–
# Fun. ev. in Jac. ev.	**66**	150	186	–
# Newton iterations	**250**	292	**250**	–
# Newton fail	0	0	0	–
Accuracy	0.019	–	**0.018**	0.107
Sim time	**0.06s**	0.21s	0.12s	0.15s

The method was implemented in python and used jointly with JModelica through the FMI standard, complementing the classical manipulation toolchain. This also demonstrates how it can be seamlessly interpreted in OOM tools [10]. Examples were reported to show the effectiveness of the proposed ideas.

Future research directions will be to better investigate the role of parameter α with respect to accuracy and numerical stability issues, to exploit the information coming from the cycle analysis to parallelise the simulation, and to explore possible uses in co-simulation contexts.

References

[1] F. Cellier and E. Kofman, *Continuous system simulation*. Springer, 2006.

[2] E. Hairer and G. Wanner, *Solving Ordinary Differential Equations II: Stiff and Differential-Algebraic Problems*, ser. Springer Series in Computational Mathematics. Springer, 2004.

[3] A. Schiela and F. Bornemann, "Sparsing in real time simulation," *ZAMM - Journal of Applied Mathematics and Mechanics*, vol. 83, no. 10, pp. 637–647, 2003.

[4] A. Schiela and H. Olsson, "Mixed-mode integration for real-time simulation," in *Modelica Workshop 2000 Proc.*, 2000, pp. 69–75.

[5] B. Gu and H. Asada, "Co-simulation of algebraically coupled dynamic subsystems," in *American Control Conference, 2001. Proceedings of the 2001*, vol. 3, 2001, pp. 2273–2278.

[6] R. Kübler and W. Schiehlen, "Two methods of simulator coupling," *Mathematical and Computer Modelling of Dynamical Systems*, vol. 6, no. 2, pp. 93–113, 2000.

[7] S. Schops, H. De Gersem, and A. Bartel, "A cosimulation framework for multirate time integration of field/circuit coupled problems," *IEEE Transactions on Magnetics*, vol. 46, no. 8, pp. 3233–3236, 2010.

[8] J. Bastian, C. Clauß, S. Wolf, and P. Schneider, "Master for co-simulation using FMI," in *Proc. of the 8th Int. Modelica Conf.*, 2011.

[9] T. Schierz, M. Arnold, and C. Clauß, "Co-simulation with communication step size control in an fmi compatible master algorithm," in *Proc. of the 9th Int. Modelica Conf.*, 2012.

[10] A. V. Papadopoulos and A. Leva, "Automating dynamic decoupling in object-oriented modelling and simulation tools," in *5th International Workshop on Equation-Based Object-Oriented Modeling Languages and Tools.*, 2013.

[11] F. Casella, A. Leva, and C. Maffezzoni, "Dynamic simulation of a condensation plate column by dynamic decoupling," in *Proc. EUROSIM '98, Espoo 1998*, 1998, pp. 368–374.

[12] F. Casella and C. Maffezzoni, "Exploiting weak interactions in object-oriented modeling," *Simulation News Europe*, vol. 22, pp. 8–10, 1998.

[13] R. Tarjan, "Enumeration of the elementary circuits of a directed graph," *SIAM Journal on Computing*, vol. 2, no. 3, pp. 211–216, 1972.

[14] D. Johnson, "Finding all the elementary circuits of a directed graph," *SIAM Journal on Computing*, vol. 4, no. 1, pp. 77–84, 1975.

[15] L. Goldberg and G. Ann, *Efficient algorithms for listing combinatorial structures*. Cambridge Univ Pr, 2009, vol. 5.

[16] C. Andersson, J. Andreasson, C. Führer, and J. Åkesson, "A workbench for multibody systems ODE and DAE solvers," in *Proceedings of the IMSD2012 - The 2nd Joint International Conference on Multibody System Dynamics*, 2012.