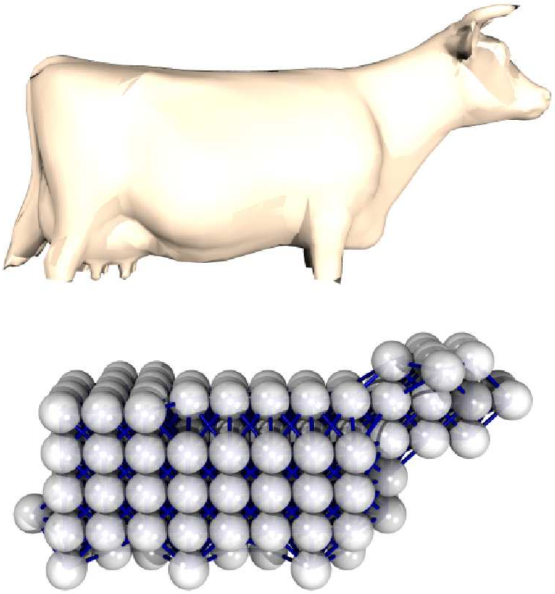
CHALMERS

FINITE ELEMENT CENTER



PREPRINT 2004–14

Simulation of mechanical systems with individual time steps

Johan Jansson and Anders Logg

Chalmers Finite Element Center
CHALMERS UNIVERSITY OF TECHNOLOGY
Göteborg Sweden 2004

CHALMERS FINITE ELEMENT CENTER

Preprint 2004–14

Simulation of mechanical systems with individual time steps

Johan Jansson and Anders Logg

CHALMERS

Chalmers Finite Element Center
Chalmers University of Technology
SE–412 96 Göteborg Sweden
Göteborg, April 2004

Simulation of mechanical systems with individual time steps
Johan Jansson and Anders Logg
NO 2004–14
ISSN 1404–4382

Chalmers Finite Element Center
Chalmers University of Technology
SE–412 96 Göteborg
Sweden
Telephone: +46 (0)31 772 1000
Fax: +46 (0)31 772 3595
`www.phi.chalmers.se`

Printed in Sweden
Chalmers University of Technology
Göteborg, Sweden 2004

SIMULATION OF MECHANICAL SYSTEMS WITH INDIVIDUAL TIME STEPS

JOHAN JANSSON AND ANDERS LOGG

Abstract. The simulation of a mechanical system involves the formulation of a differential equation (modeling) and the solution of the differential equation (computing). The solution method needs to be efficient as well as accurate and reliable. This paper discusses multi-adaptive Galerkin methods in the context of mechanical systems. The primary type of mechanical system studied is an extended mass–spring model. A multi-adaptive method integrates the mechanical system using individual time steps for the different components of the system, adapting the time steps to the different time scales of the system, potentially allowing enormous improvement in efficiency compared to traditional mono-adaptive methods.

1. Introduction

Simulation of mechanical systems is an important component of many technologies of modern society. It appears in industrial design, for the prediction and verification of mechanical products. It appears in virtual reality, both for entertainment in the form of computer games and movies, and in the simulation of realistic environments such as surgical training on virtual and infinitely resurrectable patients. Common to all these applications is that the computation time is critical. Often, an application is real-time, which means that the time inside the simulation must reasonably match the time in the real world.

Simulating a mechanical system involves both *modeling* (formulating an equation describing the system) and *computation* (solving the equation). The model of a mechanical system often takes the form of an initial value problem for a system of ordinary differential equations of the form

$$\begin{aligned} \dot{u}(t) &= f(u(t), t), \quad t \in (0, T], \\ u(0) &= u_0, \end{aligned} \tag{1.1}$$

where $u : [0, T] \to \mathbb{R}^N$ is the solution to be computed, $u_0 \in \mathbb{R}^N$ a given initial value, $T > 0$ a given final time, and $f : \mathbb{R}^N \times (0, T] \to \mathbb{R}^N$ a given function that is Lipschitz-continuous in u and bounded.

Date: April 27, 2004.

Key words and phrases. Multi-adaptivity, individual time steps, local time steps, ODE, continuous Galerkin, discontinuous Galerkin, mcgq, mdgq, mechanical system, mass–spring model.

Johan Jansson, *email*: johanjan@math.chalmers.se. Anders Logg, *email*: logg@math.chalmers.se. Department of Computational Mathematics, Chalmers University of Technology, SE–412 96 Göteborg, Sweden.

The simulation of a mechanical system thus involves the formulation of a model of the form (1.1) and the solution of (1.1) using a time-stepping method. We present below *multi-adaptive Galerkin* methods for the solution of (1.1) with individual time steps for the different parts of the mechanical system.

1.1. **Mass–spring systems.** A mass–spring system consists of a set of point masses connected by springs, typically governed by Hooke's law with other laws optionally present, such as viscous damping and external forces. Mass–spring systems appear to encompass most of the behaviors of elementary mechanical systems and thus represent a simple, intuitive, and powerful model for the simulation of mechanical systems. This is the approach taken in this paper.

However, to obtain a physically accurate model of a mechanical system, we believe it is necessary to solve a system of partial differential equations properly describing the mechanical system, in the simplest case given by the equations of linear elasticity. Discretizing the system of PDEs in space, for example using the Galerkin finite element method, an initial value problem for a system of ODEs of the form (1.1) is obtained. The resulting system can be interpreted as a mass–spring system and thus the finite element method in combination with a PDE model represents a systematic methodology for the generation of a mass–spring model of a given mechanical system.

1.2. **Time-stepping methods.** Numerical methods for the (approximate) solution of (1.1) are almost exclusively based on time-stepping, i.e., the step-wise integration of (1.1) to obtain an approximation U of the solution u satisfying

$$u(t_j) = u(t_{j-1}) + \int_{t_{j-1}}^{t_j} f(u(t), t)\, dt, \quad j = 1, \dots, M, \tag{1.2}$$

for a partition $0 = t_0 < t_1 < \cdots < t_M = T$ of $[0, T]$. The approximate solution $U \approx u$ is obtained by an appropriate approximation of the integral $\int_{t_{j-1}}^{t_j} f(u(t), t)\, dt$.

Selecting the appropriate size of the time steps $\{k_j = t_j - t_{j-1}\}_{j=1}^{M}$ is essential for efficiency and accuracy. We want to compute the solution U using as little work as possible, which means using a small number of large time steps. At the same time, we want to compute an accurate solution U which is close to the exact solution u, which means using a large number of small time steps. Often, the accuracy requirement is given in the form of a *tolerance* TOL for the size of the *error* $e = U - u$ in a suitable norm. The competing goals of efficiency and accuracy can be met using an *adaptive* algorithm, determining a sequence of time steps $\{k_j\}_{j=1}^{M}$ which produces an approximate solution U satisfying the given tolerance with minimal work.

Galerkin finite element methods present a general framework for the numerical solution of (1.1), including adaptive algorithms for the automatic construction of an optimal time step sequence, see [7, 8]. The Galerkin finite element method for (1.1) reads: Find $U \in V$, such that

$$\int_0^T (\dot{U}, v)\, dt = \int_0^T (f, v)\, dt \quad \forall v \in \hat{V}, \tag{1.3}$$

where $(\cdot,\cdot)$ denotes the $\mathbb{R}^N$ inner product and $(V,\hat{V})$ denotes a suitable pair of finite dimensional subspaces (the *trial* and *test spaces*).

Typical choices of approximating spaces include

$$
\begin{aligned}
V &= \{v \in [\mathcal{C}([0,T])]^N : v|_{I_j} \in [\mathcal{P}^q(I_j)]^N, \quad j=1,\ldots,M\},\\
\hat{V} &= \{v : v|_{I_j} \in [\mathcal{P}^{q-1}(I_j)]^N, \quad j=1,\ldots,M\},
\end{aligned}
\tag{1.4}
$$

i.e., V represents the space of continuous and piecewise polynomial vector-valued functions of degree $q \geq 1$ and $\hat{V}$ represents the space of discontinuous piecewise polynomial vector-valued functions of degree $q-1$ on a partition of $[0,T]$. We refer to this as the cG(q) method. With both V and $\hat{V}$ representing discontinuous piecewise polynomials of degree $q \geq 0$, we obtain the dG(q) method. Early work on the cG(q) and dG(q) methods include [6, 19, 10, 9].

By choosing a constant test function v in (1.3), it follows that both the cG(q) and dG(q) solutions satisfy the relation

$$
U(t_j) = U(t_{j-1}) + \int_{t_{j-1}}^{t_j} f(U(t),t)\,dt, \quad j=1,\ldots,M, \tag{1.5}
$$

corresponding to (1.2).

In the simplest case of the dG(0) method, we note that $\int_{t_{j-1}}^{t_j} f(U(t),t)\,dt \approx k_j f(U(t_j),t_j)$, since U piecewise constant, with equality if f does not depend explicitly on t. We thus obtain the method

$$
U(t_j) = U(t_{j-1}) + k_j f(U(t_j),t_j), \quad j=1,\ldots,M, \tag{1.6}
$$

which we recognize as the backward (or implicit) Euler method. In general, a cG(q) or dG(q) method corresponds to an implicit Runge–Kutta method, with details depending on the choice of quadrature for the approximation of the integral of $f(U,\cdot)$.

1.3. **Multi-adaptive time-stepping.** Standard methods for the discretization of (1.1), including the cG(q) and dG(q) methods, require that the same time steps $\{k_j\}_{j=1}^M$ are used for all components $U_i = U_i(t)$ of the approximate solution U of (1.1). This can be very costly if the system exhibits multiple time scales of different magnitudes. If the different time scales are localized to different components, efficient representation and computation of the solution thus requires that this difference in time scales is reflected in the choice of approximating spaces $(V,\hat{V})$. We refer to the resulting methods, recently introduced in a series of papers [20, 21, 22, 23, 16], as *multi-adaptive Galerkin methods*.

Surprisingly, individual time-stepping (multi-adaptivity) has previously received little attention in the large literature on numerical methods for ODEs, see e.g. [3, 12, 13, 2, 27, 1], but has been used to some extent for specific applications, including specialized integrators for the n-body problem [24, 4, 26], and low-order methods for conservation laws [25, 18, 5].

1.4. **Obtaining the software.** The examples presented below have been obtained using **DOLFIN** version 0.4.11 [14]. **DOLFIN** is licensed under the GNU General Public License [11], which means that anyone is free to use or modify the software, provided these rights are preserved. The source code of **DOLFIN**, including numerous example programs, is available at the **DOLFIN** web page, `http://www.phi.chalmers.se/dolfin/`, and each new release is announced on freshmeat.net. Alternatively, the source code can be obtained through anonymous CVS as explained on the web page. Comments and contributions are welcome.

The mechanical systems presented in the examples have been implemented using *Ko*, which is a software system for the simulation of mass–spring models, based on **DOLFIN**s multi-adaptive ODE-solver. Ko will be released shortly under the GNU General Public License and will be available at `http://www.phi.chalmers.se/ko/`.

1.5. **Outline of the paper.** We first describe the basic mass–spring model in Section 2 and then give a short introduction to multi-adaptive Galerkin methods in Section 3. In Section 4, we discuss the interface of the multi-adaptive solver and its application to mass–spring models. In Section 5, we investigate and analyze the performance of the multi-adaptive methods for a set of model problems. Finally, we present in Section 6 results for a number of large mechanical systems to demonstrate the potential and applicability of the proposed methods.

2. Mass–spring model

We have earlier in [17] described an extended mass–spring model for the simulation of systems of deformable bodies.

The mass–spring model represents bodies as systems of discrete *mass elements*, with the forces between the mass elements transmitted using explicit *spring connections*. (Note that "spring" is a historical term, and is not limited to pure Hookean interactions.) Given the forces acting on an element, we can determine its motion from Newton's second law,

$$F = ma, \tag{2.1}$$

where F denotes the force acting on the element, m is the mass of the element, and $a = \ddot{x}$ is the acceleration of the element with $x = (x_1, x_2, x_3)$ the position of the element. The motion of the entire body is then implicitly described by the motion of its individual mass elements.

The force given by a standard spring is assumed to be proportional to the elongation of the spring from its rest length. We extend the standard model with contact, collision and fracture, by adding a radius of interaction to each mass element, and dynamically creating and destroying spring connections based on contact and fracture conditions.

In Table 1 and Figure 1, we give the basic properties of the mass–spring model consisting of mass elements and spring connections. With these definitions, a mass–spring model may thus be given by just listing the mass elements and spring connections of the model.

A *mass element* e is a set of parameters $\{x,\ v,\ m,\ r,\ C\}$:

x	:	position
v	:	velocity
m	:	mass
r	:	radius
C	:	a set of spring connections

A *spring connection* c is a set of parameters $\{e_1,\ e_2,\ \kappa,\ b,\ l_0,\ l_f\}$:

e_1	:	the first mass element connected to the spring
e_2	:	the second mass element connected to the spring
κ	:	Hooke spring constant
b	:	damping constant
l_0	:	rest length
l_f	:	fracture length

TABLE 1. Descriptions of the basic elements of the mass–spring model: mass elements and spring connections.

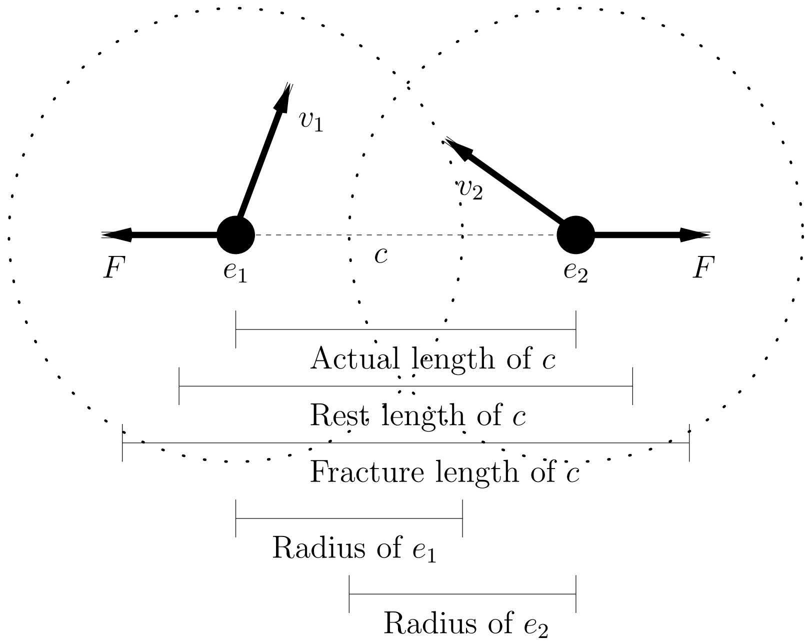


FIGURE 1. Schema of two mass elements e_1 and e_2, a spring connection c, and important quantities.

3. Multi-adaptive Galerkin methods

The multi-adaptive methods mcG(q) and mdG(q) used for the simulation are obtained as extensions of the standard cG(q) and dG(q) methods by enriching the trial and test spaces $(V, \hat{V})$ of (1.3) to allow each component U_i of the approximate solution U to be piecewise polynomial on an individual partition of $[0, T]$.

3.1. **Definition of the methods.** To give the definition of the multi-adaptive Galerkin methods, we introduce the following notation: Subinterval j for component i is denoted by $I_{ij} = (t_{i,j-1}, t_{ij}]$, and the length of the subinterval is given by the local *time step* $k_{ij} = t_{ij} - t_{i,j-1}$ for $j = 1, \ldots, M_i$. This is illustrated in Figure 2. We also assume that the interval $[0, T]$ is partitioned into blocks between certain synchronized time levels $0 = T_0 < T_1 < \cdots < T_M = T$. We refer to the set of intervals $\mathcal{T}_n$ between two synchronized time levels T_{n-1} and T_n as a *time slab*.

With this notation, we can write the mcG(q) method for (1.1) in the following form: Find $U \in V$, such that

$$\int_0^T (\dot{U}, v)\, dt = \int_0^T (f, v)\, dt \quad \forall v \in \hat{V}, \tag{3.1}$$

where the trial space V and test space $\hat{V}$ are given by

$$\begin{aligned} V &= \{v \in [\mathcal{C}([0,T])]^N : v_i|_{I_{ij}} \in \mathcal{P}^{q_{ij}}(I_{ij}), \quad j = 1, \ldots, M_i, \quad i = 1, \ldots, N\}, \\ \hat{V} &= \{v : v_i|_{I_{ij}} \in \mathcal{P}^{q_{ij}-1}(I_{ij}), \quad j = 1, \ldots, M_i, \quad i = 1, \ldots, N\}. \end{aligned} \tag{3.2}$$

The mcG(q) method is thus obtained as a simple extension of the standard cG(q) method by allowing each component to be piecewise polynomial on an individual partition of $[0, T]$. Similarly, we obtain the mdG(q) method as a simple extension of the standard dG(q) method. For a detailed description of the multi-adaptive Galerkin methods, we refer the reader to [20, 21, 22, 23, 16]. In particular, we refer to [20] or [22] for the full definition of the methods.

3.2. **Adaptivity.** The individual time steps $\{k_{ij}\}_{j=1,i=1}^{M_i,N}$ are chosen adaptively based on an a posteriori error estimate for the global error $e = U - u$ at final time $t = T$, as discussed in [20, 21]. The a posteriori error estimate for the mcG(q) method takes the form

$$\|e(T)\|_{l_2} \leq C_q \sum_{i=1}^N S_i^{[q_i]}(T) \max_{[0,T]} |k_i^{q_i} R_i(U, \cdot)|, \tag{3.3}$$

where C_q is an interpolation constant, $S_i^{[q_i]}(T)$ are the individual *stability factors*, $k_i = k_i(t)$ are the individual time steps, and $R_i(U, \cdot) = \dot{U}_i - f_i(U, \cdot)$ are the individual *residuals* for $i = 1, \ldots, N$. The individual stability factors $S_i^{[q_i]}(T)$, measuring the effect of local errors introduced by a nonzero local residual on the global error, are obtained from the solution ϕ of an associated *dual problem*, see [7] or [20].

Thus, to determine the individual time steps, we measure the individual residuals and take each individual time step k_{ij} such that

$$k_{ij}^{q_{ij}} \max_{I_{ij}} |R_i(U, \cdot)| = \mathrm{TOL}/(N C_q S_i^{[q_i]}(T)), \tag{3.4}$$

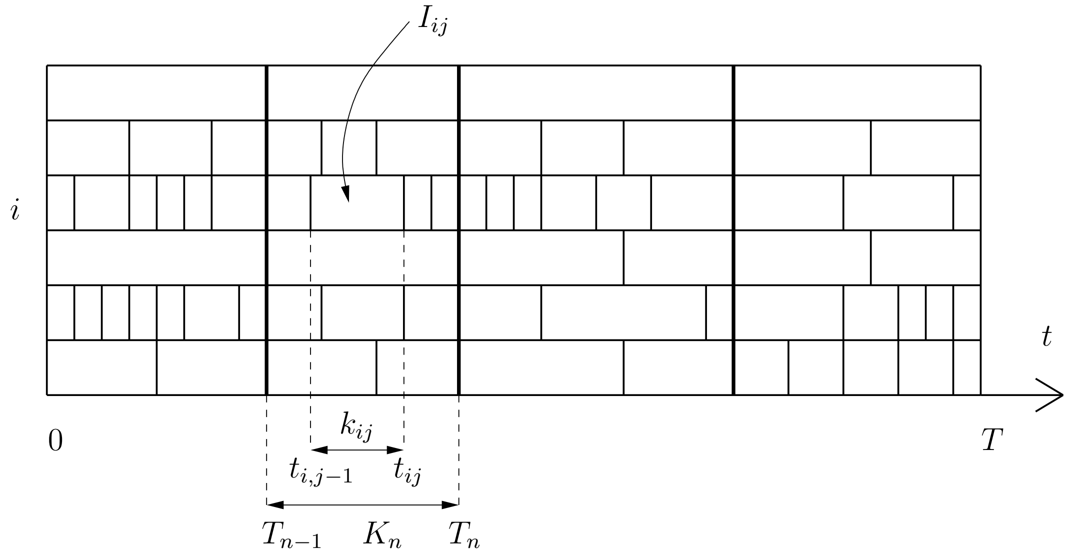


FIGURE 2. Individual partitions of the interval $[0, T]$ for different components. Elements between common synchronized time levels are organized in time slabs. In this example, we have $N = 6$ and $M = 4$.

where TOL is a tolerance for the error at final time. See [21] or [15] for a detailed discussion of the algorithm for the automatic selection of the individual time steps.

3.3. **Iterative methods.** The system of discrete nonlinear equations defined by (3.1) is solved by fixed point iteration on time slabs, as described in [16]. For a stiff problem, the fixed point iteration is automatically stabilized by introducing a damping parameter which is adaptively determined through feed-back from the computation. We refer to this as *adaptive fixed point iteration.*

4. Multi-adaptive simulation of mass–spring systems

The simulation of a mechanical system involves the formulation of a differential equation (modeling) and the solution of the differential equation (computing). Having defined these two components in the form of the mass–spring model presented in Section 2 and the multi-adaptive solver presented in Section 3, we comment briefly on the user interface of the multi-adaptive solver.

The user interface of the multi-adaptive solver is specified in terms of an ODE base class consisting of a right hand side f, a time interval $[0, T]$, and an initial value u_0, as shown in Table 2. To solve an ODE, the user implements a subclass which inherits from the ODE base class.

The mass–spring model presented above has been implemented using Ko, a software system for the simulation and visualization of mass–spring models. Ko automatically generates a mass–spring model from a geometric representation of a given system, as shown in Figure 3. The mass–spring model is then automatically translated into a system

```
class ODE
{
  ODE(int N);

  virtual real u0(int i);
  virtual real f(Vector u, real t, int i);
}
```

TABLE 2. Sketch of the C++ interface of the multi-adaptive ODE-solver.

of ODEs of the form (1.1). Ko specifies the ODE system as an ODE subclass and uses **DOLFIN** to compute the solution.

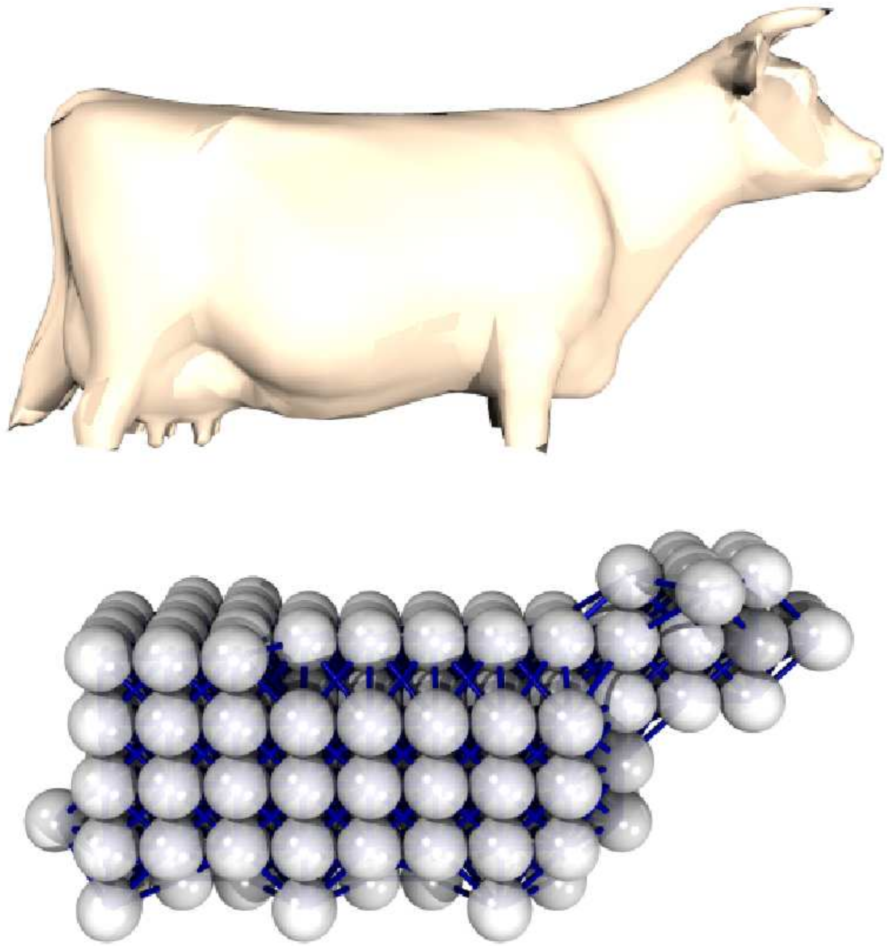


FIGURE 3. A geometric representation of a cow is automatically translated into a mass–spring model.

Ko represents a mass–spring model internally as lists of mass elements and spring connections. To evaluate the right-hand side f of the corresponding ODE system, a translation or mapping is thus needed between a given mass element and a component number in the system of ODEs. This mapping may take a number different forms; Ko uses the mapping presented in Algorithm 1.

Algorithm 1 FromComponents(Vector u, Mass m)

$i \leftarrow \text{index}(m)$
$N \leftarrow \text{size}(u)$

$m.x_1 \leftarrow u(3(i-1)+1)$
$m.x_2 \leftarrow u(3(i-1)+2)$
$m.x_3 \leftarrow u(3(i-1)+3)$

$m.v_1 \leftarrow u(N/2+3(i-1)+1)$
$m.v_2 \leftarrow u(N/2+3(i-1)+2)$
$m.v_3 \leftarrow u(N/2+3(i-1)+3)$

5. Performance

We consider a simple model problem consisting of a long string of n point masses connected with springs as shown in Figure 4. The first mass on the left is connected to a fixed support through a hard spring with large spring constant $\kappa \gg 1$. All other springs are connected together with soft springs with spring constant $\kappa = 1$. As a result, the first mass oscillates at a high frequency, with the rest of the masses oscillating slowly. In Figure 5, we plot the coordinates for the first three masses on $[0, 1]$.

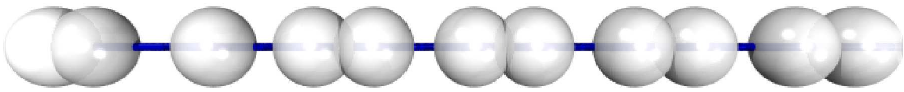


FIGURE 4. The mechanical system used for the performance test. The system consists of a string of masses, fixed at the left end. Each mass has been slightly displaced to initialize the oscillations.

To compare the performance of the multi-adaptive solver (in the case of the mcG(1) method) with a mono-adaptive method (the cG(1) method), we choose a fixed small time step k for the first mass and a fixed large time step $K > k$ for the rest of the masses in the multi-adaptive simulation, and use the same small time step k for all masses in the mono-adaptive simulation. We let $M = K/k$ denote the number of small time steps per each large time step.

We run the test for $M = 100$ and $M = 200$ with large spring constant $\kappa = 10M$ for the hard spring connecting the first mass to the fixed support. We use a large time step of size

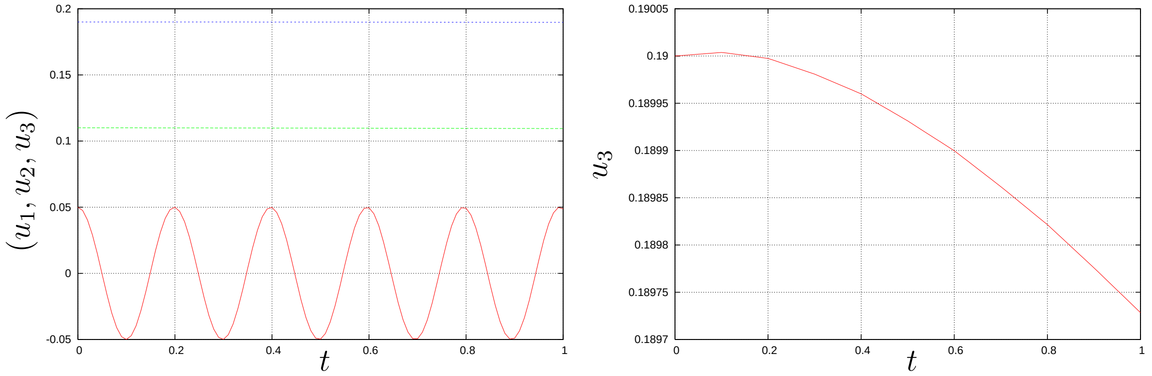


FIGURE 5. Coordinates for the first three masses of the simple model problem (left) and for the third mass (right).

$K = 0.1$ and, consequently, a small time step of size $k = 0.1/M$. The computation time T_c is recorded as function of the number of masses n.

As shown in Figure 6, the computation time for the multi-adaptive solver grows slowly with the number of masses n, practically remaining constant; small time steps are used only for the first rapidly oscillating mass and so the work is dominated by frequently updating the first mass, independent of the total number of masses. On the other hand, the work for the mono-adaptive method grows linearly with the total number of masses, as expected.

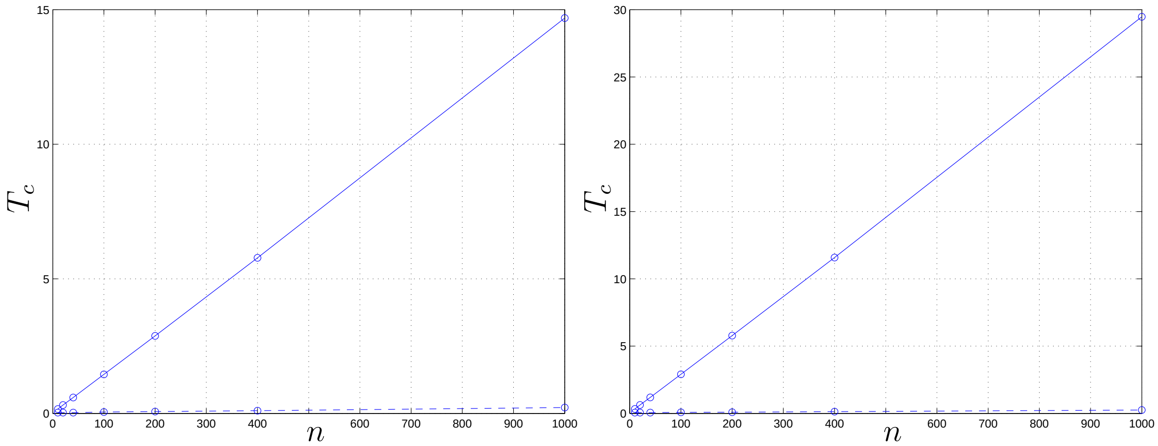


FIGURE 6. Computation time T_c as function of the number of masses n for the multi-adaptive solver (dashed) and a mono-adaptive method (solid), with $M = 100$ (left) and $M = 200$ (right).

More specifically, the complexity of the mono-adaptive method may be expressed in terms of M and n as follows:

$$T_c(M, n) = C_1 + C_2 M n, \tag{5.1}$$

while for the multi-adaptive solver, we obtain

$$T_c(M, n) = C_3M + C_4n. \tag{5.2}$$

Our general conclusion is that the multi-adaptive solver is more efficient than a mono-adaptive method for the simulation of a mechanical system if M is large, i.e., when small time steps are needed for a part of the system, and if n is large, i.e, if large time steps may be used for a large part of the system.

The same result is obtained if we add damping to the system in the form of a damping constant of size $b = 100$ for the spring connections between the slowly oscillating masses, resulting in gradual damping of the slow oscillations, while keeping the rapid oscillations of the first mass largely unaffected. With $b = 100$, adaptive fixed point iteration is automatically activated for the solution of the discrete equations, as discussed in Section 3.3.

6. Large problems and applications

To demonstrate the potential and applicability of the proposed mass–spring model and the multi-adaptive solver, we present results for a number of large mechanical systems.

6.1. **Oscillating tail.**

For the first example, we take the mass–spring model of Figure 3 representing a heavy cow and add a light mass representing its tail, as shown in Figure 7. The cow is given a constant initial velocity and the tail is given an initial push to make it oscillate. A sequence of frames from an animation of the multi-adaptive solution is given in Figure 8.

We compare the performance of the multi-adaptive solver (in the case of the mcG(1) method) with a mono-adaptive method (the cG(1) method) using the same time steps for all components. We also make a comparison with a simple non-adaptive implementation of the cG(1) method, with minimal overhead, using constant time steps equal to the smallest time step selected by the mono-adaptive method.

As expected, the multi-adaptive solver automatically selects small time steps for the oscillating tail and large time steps for the rest of the system. In Figure 9, we plot the time steps as function of time for relevant components of the system. We also plot the corresponding solutions in Figure 11. In Figure 10, we plot the time steps used in the mono-adaptive simulation.

The computation times are given in Table 3. The speed-up of the multi-adaptive method compared to the mono-adaptive method is a factor 70. Compared to the simple non-adaptive implementation of the cG(1) method, using a minimal amount of work, the speed-up is a factor 3. This shows that the speed-up of a multi-adaptive method can be significant. It also shows that the overhead is substantial for the current implementation of the multi-adaptive solver, including the organization of the multi-adaptive time slabs, interpolation of solution values within time slabs, and the evaluation of residuals for multi-adaptive time-stepping. However, we believe it is possible to remove a large part of this overhead.

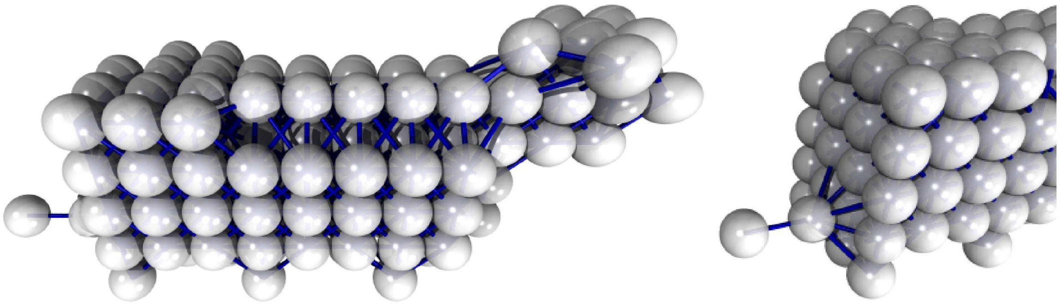


FIGURE 7. A cow with an oscillating tail (left) with details of the tail (right).

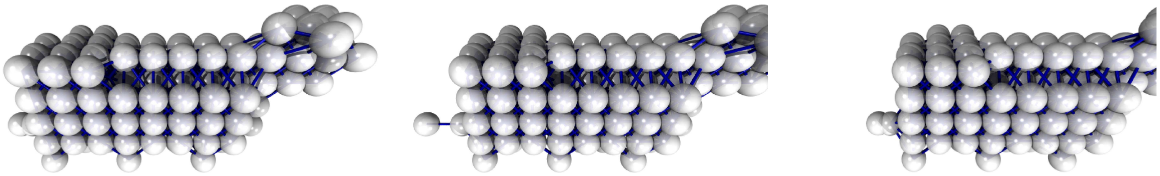


FIGURE 8. The tail oscillates rapidly while the rest of the cow travels at a constant velocity to the right.

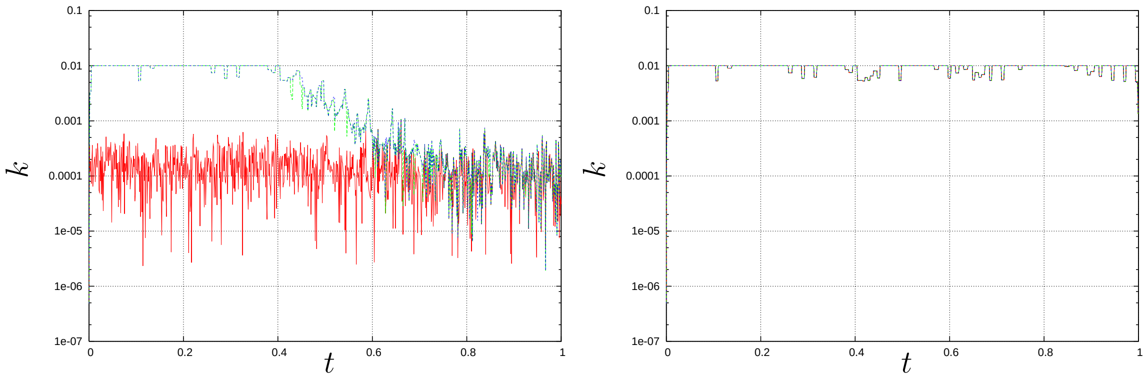


FIGURE 9. Multi-adaptive time steps used in the simulation of the cow with oscillating tail. The plot on the left shows the time steps for components 481–483 corresponding to the velocity of the tail, and the plot on the right shows the time steps for components 13–24 corresponding to the positions for a set of masses in the interior of the cow.

6.2. **Local manipulation.** For the next example, we fix a damped cow shape at one end and repeatedly push the other end with a manipulator in the form of a large sphere, as illustrated in Figure 12.

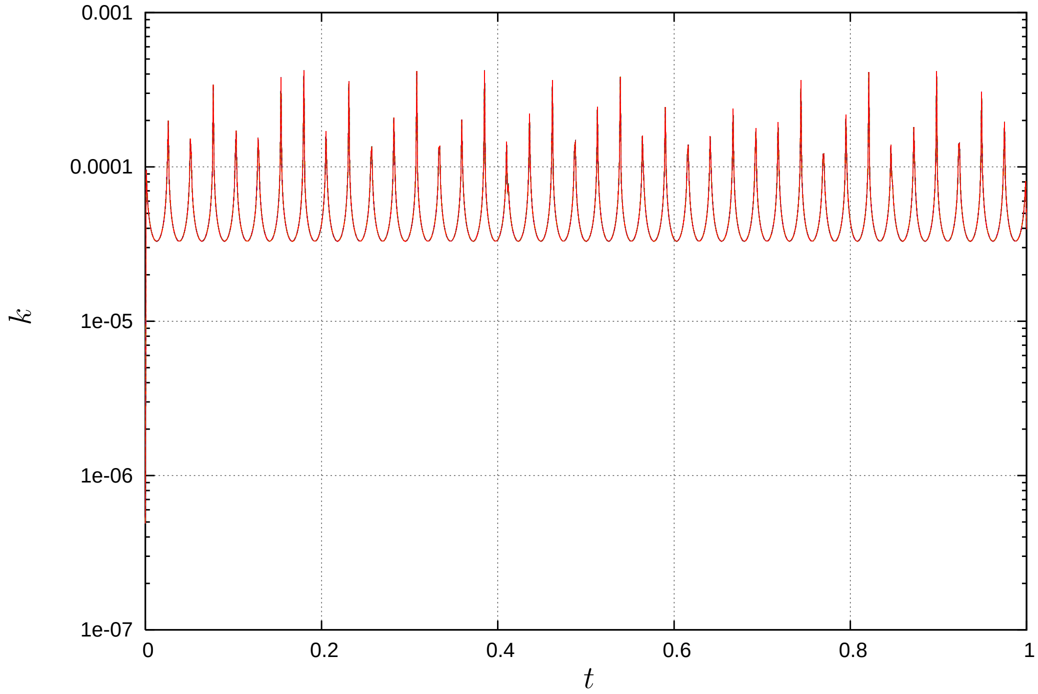


FIGURE 10. Mono-adaptive time steps for the cow with oscillating tail.

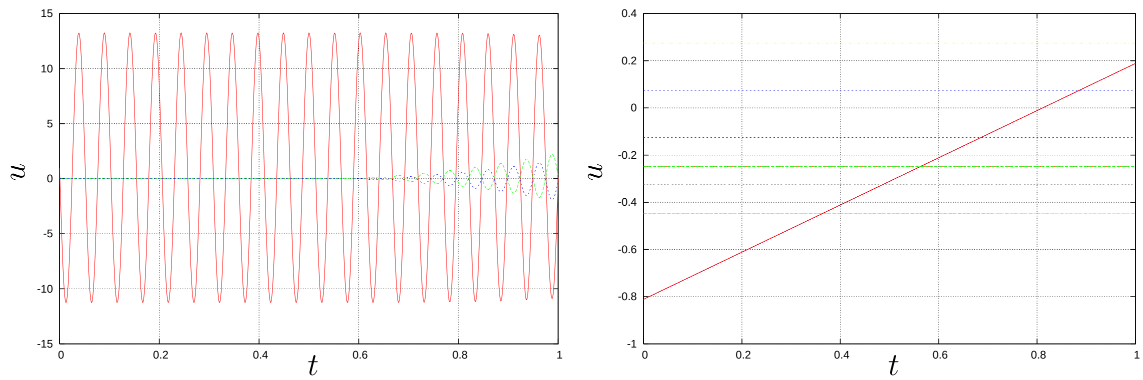


FIGURE 11. Solution for relevant components of the cow with oscillating tail. The plot on the left shows the solution for components 481–483 corresponding to the velocity of the tail, and the plot on the right shows the solution for components 13–24 corresponding to the positions for a set of masses in the interior of the cow.

Algorithm	Time / s
Multi-adaptive	40
Mono-adaptive	2800
Non-adaptive	130

Table 3. Computation times for the simulation of the cow with oscillating tail for three different algorithms: multi-adaptive mcG(1), mono-adaptive cG(1), and a simple implementation of non-adaptive cG(1) with fixed time steps and minimal overhead.

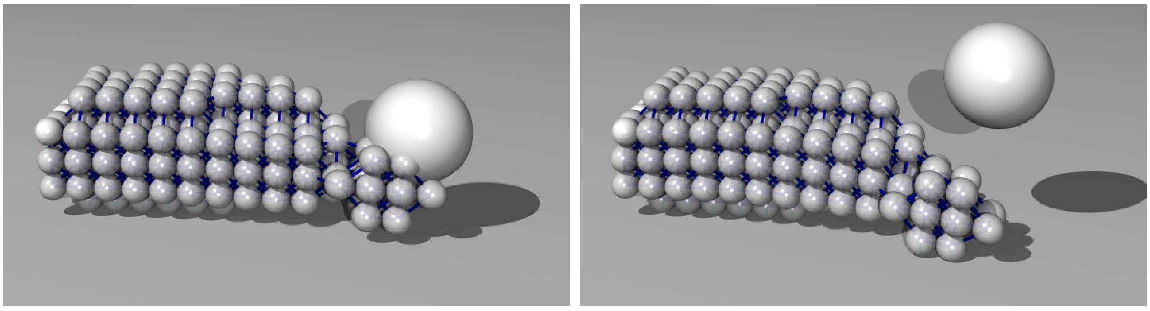


Figure 12. A cow shape is locally manipulated. Small time steps are automatically selected for the components affected by the local manipulation, with large time steps for the rest of the system.

As shown in Figure 13, the multi-adaptive solver automatically selects small time steps for the components directly affected by the manipulation. This test problem also illustrates the basic use of adaptive time-stepping; the time steps are drastically decreased at each impact to accurately track the effect of the impact.

6.3. **A stiff beam.** Our final example demonstrates the applicability of the multi-adaptive solver to a stiff problem consisting of a block being dropped onto a stiff beam, as shown in Figure 14. The material of both the block and the beam is very hard and very damped, with spring constant $\kappa = 10^7$ and damping constant $b = 2 \cdot 10^5$ for each spring connection. The multi-adaptive time steps for the simulation are shown in Figure 15. Note that the time steps are drastically reduced at the time of impact, with large time steps before and after the impact.

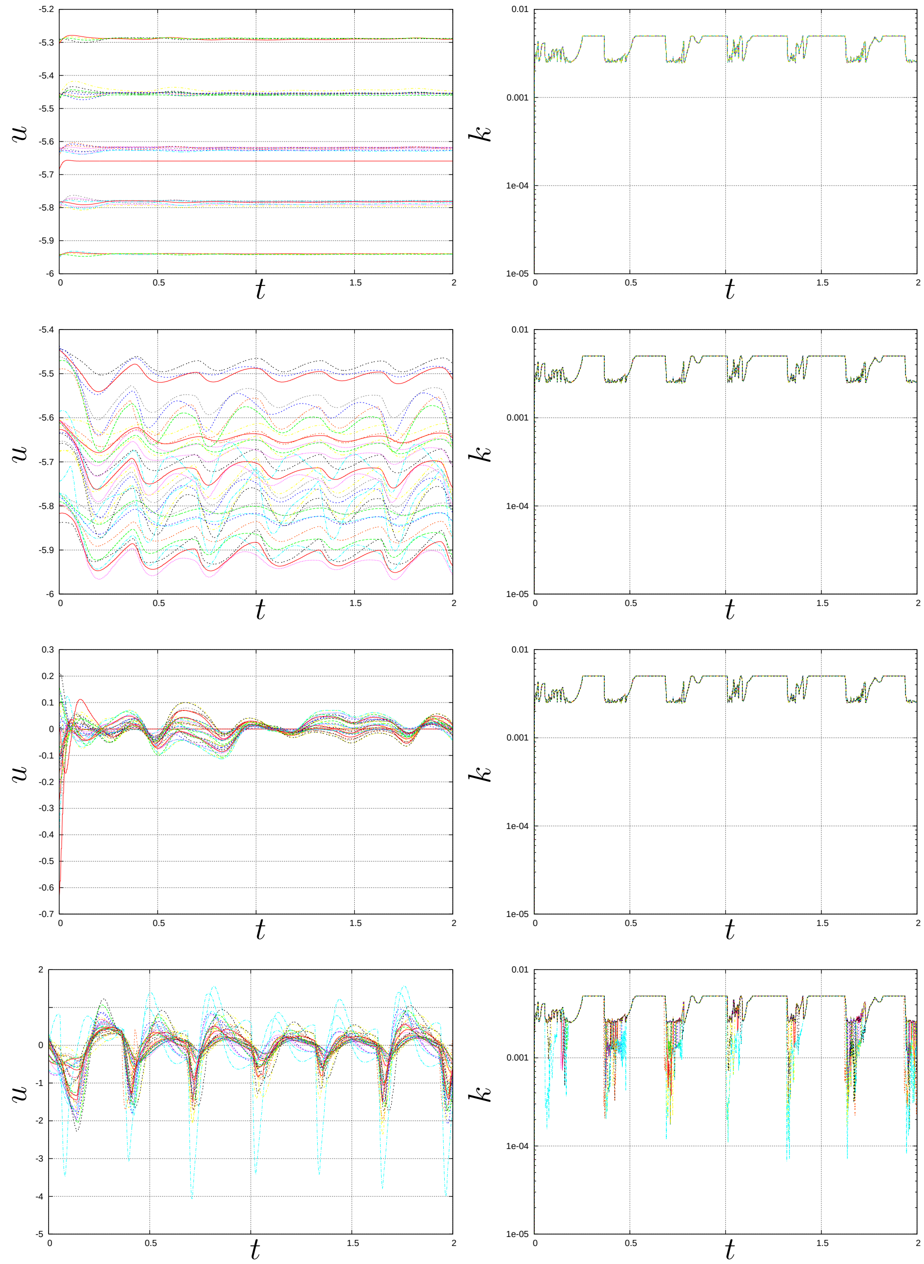


Figure 13. Solution (left) and multi-adaptive time steps (right) for selected components of the manipulated cow. The two top rows correspond to the positions of the left- and right-most masses, respectively, and the two rows below correspond to the velocities of the left- and right-most masses, respectively. Note that smaller time steps are used for the components mostly affected by the manipulation, in particular at the point of impact, while larger time steps are used for other components.

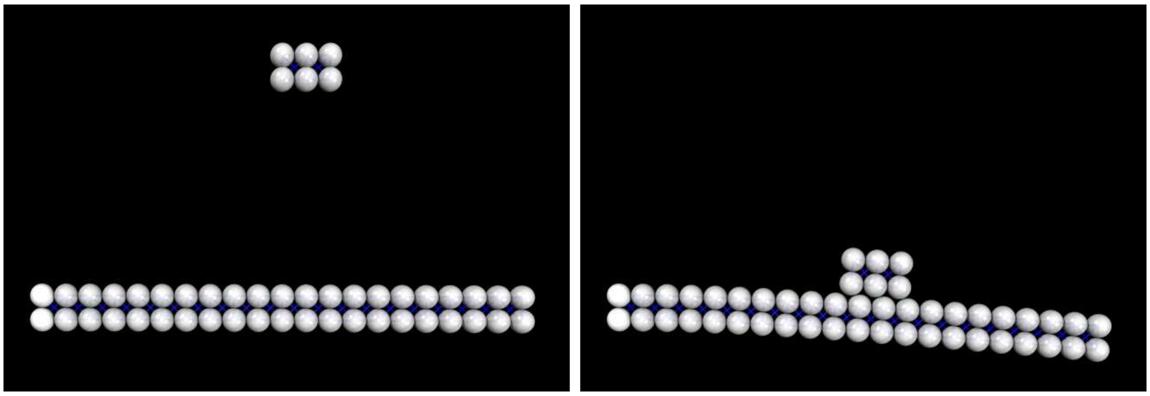


FIGURE 14. A block is dropped onto a beam. The material of both the block and the beam is very hard and very damped, with spring constant $\kappa = 10^7$ and damping constant $b = 2 \cdot 10^5$.

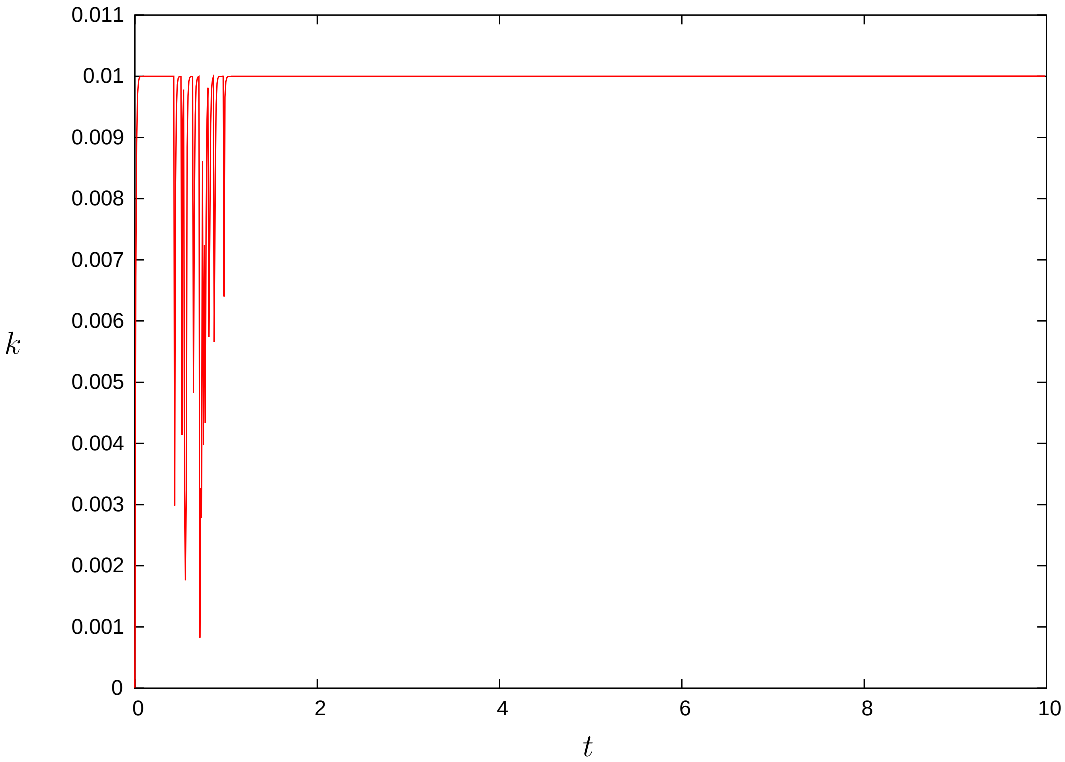


FIGURE 15. Multi-adaptive time steps for the block and beam. Note that the time steps are drastically reduced at the time of impact. The maximum time step is set to 0.01 to track the contact between the block and the beam.

7. Conclusions

From the results presented above, we make the following conclusions regarding multi-adaptive time-stepping:

- A multi-adaptive method outperforms a mono-adaptive method for systems containing different time scales if there is a significant separation of the time scales and if the fast time scales are localized to a relatively small part of the system.
- Multi-adaptive time-stepping, and in particular the current implementation, works in practice for large and realistic problems.

References

[1] U. Ascher and L. Petzold, *Computer Methods for Ordinary Differential Equations and Differential-Algebraic Equations*, SIAM, 1998.

[2] J. Butcher, *The Numerical Analysis of Ordinary Differential Equations — Runge–Kutta and General Linear Methods*, Wiley, 1987.

[3] G. Dahlquist, *Stability and Error Bounds in the Numerical Integration of Ordinary Differential Equations*, PhD thesis, Stockholm University, 1958.

[4] R. Davé, J. Dubinski, and L. Hernquist, *Parallel treeSPH*, New Astronomy, 2 (1997), pp. 277–297.

[5] C. Dawson and R. Kirby, *High resolution schemes for conservation laws with locally varying time steps*, SIAM J. Sci. Comput., 22, No. 6 (2001), pp. 2256–2281.

[6] M. Delfour, W. Hager, and F. Trochu, *Discontinuous Galerkin methods for ordinary differential equations*, Math. Comp., 36 (1981), pp. 455–473.

[7] K. Eriksson, D. Estep, P. Hansbo, and C. Johnson, *Introduction to adaptive methods for differential equations*, Acta Numerica, (1995), pp. 105–158.

[8] ———, *Computational Differential Equations*, Cambridge University Press, 1996.

[9] D. Estep, *A posteriori error bounds and global error control for approximations of ordinary differential equations*, SIAM J. Numer. Anal., 32 (1995), pp. 1–48.

[10] D. Estep and D. French, *Global error control for the continuous Galerkin finite element method for ordinary differential equations*, M^2AN, 28 (1994), pp. 815–852.

[11] Free Software Foundation, *GNU GPL*, http://www.gnu.org/copyleft/gpl.html.

[12] E. Hairer and G. Wanner, *Solving Ordinary Differential Equations I — Nonstiff Problems*, Springer Series in Computational Mathematics, vol 8, 1991.

[13] ———, *Solving Ordinary Differential Equations II — Stiff and Differential-Algebraic Problems*, Springer Series in Computational Mathematics, vol 14, 1991.

[14] J. Hoffman and A. Logg et al., *DOLFIN*, http://www.phi.chalmers.se/dolfin/.

[15] J. Jansson and A. Logg, *Algorithms for multi-adaptive time-stepping*, submitted to ACM Trans. Math. Softw., (2004).

[16] ———, *Multi-adaptive Galerkin methods for ODEs V: Stiff problems*, submitted to BIT, (2004).

[17] J. Jansson and J. Vergeest, *A discrete mechanics model for deformable bodies*, Computer-Aided Design, 34 (2002).

[18] J.E. Flaherty, R.M. Loy, M.S. Shephard, B.K. Szymanski, J.D. Teresco, and L.H. Ziantz, *Adaptive local refinement with octree load balancing for the parallel solution of three-dimensional conservation laws*, Journal of Parallel and Distributed Computing, 47 (1997), pp. 139–152.

[19] C. Johnson, *Error estimates and adaptive time-step control for a class of one-step methods for stiff ordinary differential equations*, SIAM J. Numer. Anal., 25 (1988), pp. 908–926.

[20] A. Logg, *Multi-adaptive Galerkin methods for ODEs I*, SIAM J. Sci. Comput., 24 (2003), pp. 1879–1902.

[21] ———, *Multi-adaptive Galerkin methods for ODEs II: Implementation and applications*, SIAM J. Sci. Comput., 25 (2003), pp. 1119–1141.

[22] ———, *Multi-adaptive Galerkin methods for ODEs III: Existence and stability*, Submitted to SIAM J. Numer. Anal., (2004).

[23] ———, *Multi-adaptive Galerkin methods for ODEs IV: A priori error estimates*, Submitted to SIAM J. Numer. Anal., (2004).

[24] J. Makino and S. Aarseth, *On a Hermite integrator with Ahmad-Cohen scheme for gravitational many-body problems*, Publ. Astron. Soc. Japan, 44 (1992), pp. 141–151.

[25] S. Osher and R. Sanders, *Numerical approximations to nonlinear conservation laws with locally varying time and space grids*, Math. Comp., 41 (1983), pp. 321–336.

[26] S.G. Alexander and C.B. Agnor, *n-body simulations of late stage planetary formation with a simple fragmentation model*, ICARUS, 132 (1998), pp. 113–124.

[27] L. Shampine, *Numerical Solution of Ordinary Differential Equations*, Chapman & Hall, 1994.

Chalmers Finite Element Center Preprints

2003–01 *A hybrid method for elastic waves*
Larisa Beilina

2003–02 *Application of the local nonobtuse tetrahedral refinement techniques near Fichera-like corners*
L. Beilina, S. Korotov and M. Křížek

2003–03 *Nitsche's method for coupling non-matching meshes in fluid-structure vibration problems*
Peter Hansbo and Joakim Hermansson

2003–04 *Crouzeix–Raviart and Raviart–Thomas elements for acoustic fluid–structure interaction*
Joakim Hermansson

2003–05 *Smoothing properties and approximation of time derivatives in multistep backward difference methods for linear parabolic equations*
Yubin Yan

2003–06 *Postprocessing the finite element method for semilinear parabolic problems*
Yubin Yan

2003–07 *The finite element method for a linear stochastic parabolic partial differential equation driven by additive noise*
Yubin Yan

2003–08 *A finite element method for a nonlinear stochastic parabolic equation*
Yubin Yan

2003–09 *A finite element method for the simulation of strong and weak discontinuities in elasticity*
Anita Hansbo and Peter Hansbo

2003–10 *Generalized Green's functions and the effective domain of influence*
Donald Estep, Michael Holst, and Mats G. Larson

2003–11 *Adaptive finite element/difference method for inverse elastic scattering waves*
Larisa Beilina

2003–12 *A Lagrange multiplier method for the finite element solution of elliptic domain decomposition problems using non-matching meshes*
Peter Hansbo, Carlo Lovadina, Ilaria Perugia, and Giancarlo Sangalli

2003–13 *A reduced P^1–discontinuous Galerkin method*
R. Becker, E. Burman, P. Hansbo, and M.G. Larson

2003–14 *Nitsche's method combined with space–time finite elements for ALE fluid–structure interaction problems*
Peter Hansbo, Joakim Hermansson, and Thomas Svedberg

2003–15 *Stabilized Crouzeix–Raviart element for the Darcy-Stokes problem*
Erik Burman and Peter Hansbo

2003–16 *Edge stabilization for the generalized Stokes problem: a continuous interior penalty method*
Erik Burman and Peter Hansbo

2003–17 *A conservative flux for the continuous Galerkin method based on discontinuous enrichment*
Mats G. Larson and A. Jonas Niklasson

2003–18 *CAD–to–CAE integration through automated model simplification and adaptive modelling*
K.Y. Lee, M.A. Price, C.G. Armstrong, M.G. Larson, and K. Samuelsson

2003–19 *Multi-adaptive time integration*
Anders Logg

2003–20 *Adaptive computational methods for parabolic problems*
Kenneth Eriksson, Claes Johnson, and Anders Logg

2003–21 *The FEniCS project*
T. Dupont, J. Hoffman, C. Johnson, R.C. Kirby, M.G. Larson, A. Logg, and R. Scott

2003–22 *Adaptive finite element methods for LES: Computation of the mean drag coefficient in a turbulent flow around a surface mounted cube using adaptive mesh refinement*
Johan Hoffman

2003–23 *Adaptive DNS/LES: a new agenda in CFD*
Johan Hoffman and Claes Johnson

2003–24 *Multiscale convergence and reiterated homogenization of parabolic problem*
Anders Holmbom, Nils Svanstedt, and Niklas Wellander

2003–25 *On the relationship between some weak compactnesses with different numbers of scales*
Anders Holmbom, Jeanette Silfver, Nils Svanstedt, and Niklas Wellander

2003–26 *A posteriori error estimation in computational inverse scattering*
Larisa Beilina and Claes Johnson

2004–01 *Computability and adaptivity in CFD*
Johan Hoffman och Claes Johnson

2004–02 *Interpolation estimates for piecewise smooth functions in one dimension*
Anders Logg

2004–03 *Estimates of derivatives and jumps across element boundaries for multi-adaptive Galerkin solutions of ODEs*
Anders Logg

2004–04 *Multi-adaptive Galerkin methods for ODEs III: Existence and stability*
Anders Logg

2004–05 *Multi-adaptive Galerkin methods for ODEs IV: A priori error estimates*
Anders Logg

2004–06 *A stabilized non-conforming finite element method for incompressible flow*
Erik Burman and Peter Hansbo

2004–07 *On the uniqueness of weak solutions of Navier-Stokes equations: Remarks on a Clay Institute prize problem*
Johan Hoffman and Claes Johnson

2004–08 *A new approach to computational turbulence modeling*
Johan Hoffman and Claes Johnson

2004–09 *A posteriori error analysis of the boundary penalty method*
Kenneth Eriksson, Mats G. Larson, and Axel Målqvist

2004–10 *A posteriori error analysis of stabilized finite element approximations of the helmholtz equation on unstructured grids*
Mats G. Larson and Axel Målqvist

2004–11 *Adaptive variational multiscale methods based on a posteriori error estimation*
Mats G. Larson and Axel Målqvist

2004–12 *Multi-adaptive Galerkin methods for ODEs V: Stiff problems*
Johan Jansson and Anders Logg

2004–13 *Algorithms for multi-adaptive time-stepping*
Johan Jansson and Anders Logg

2004–14 *Simulation of mechanical systems with individual time steps*
Johan Jansson and Anders Logg

These preprints can be obtained from

`www.phi.chalmers.se/preprints`