Simulation

Methodology

Simulation: Transactions of the Society for Modeling and Simulation International
1–13

DOI: 10.1177/0037549719835026
journals.sagepub.com/home/sim
SAGE SCS

An implementation of geometric integration within Matlab

Guillaume Chauvon[1,2], Philippe Saucez[1] and Alain Vande Wouwer[2]

Abstract

Geometric integrators allow preservation of specific geometric properties of the exact flow of differential equation systems, such as energy, phase-space volume, and time-reversal symmetry, and are particularly reliable for long-run integration. In this study, variable step size composition methods and Gauss methods are implemented in Matlab library integrators, and are tested with several representative problems, including the Kepler problem, the outer solar system and a conservative Lotka–Volterra system. Variable step size integrators often perform better than their fixed step size counterparts and the numerical results show excellent long time preservation of the Hamiltonian in these examples.

Keywords

Geometric numerical integrator, Hamiltonian systems, reversible systems, conservative Lotka–Volterra systems, numerical simulation

1. Introduction

Although many excellent, general purpose codes for the numerical solution of ordinary differential equations (ODEs) are currently available, they have mostly been developed based on accuracy and stability concerns, and do not intrinsically preserve specific properties or invariants of the underlying problems. As numerical errors accumulate over time, these classical integrators will usually not be able to compute solutions over long periods of time (required in some fields of investigation such as celestial mechanics). Geometric numerical integration is a relatively new class of methods which focuses on the preservation of geometric properties associated with the exact solution of differential equation systems,[1] and, in particular, the symplecticity and the reversibility of Hamiltonian systems. These numerical schemes offering preservation of geometric properties will have better long-run behavior.

The theory of geometric numerical integration was developed in the 1980s and has been well studied since then by many authors. More details can be found in two monographs,[2,3] or in the overview by McLachlan et al.[1] Applications are also diverse, including celestial mechanics,[4] particle accelerators,[5] molecular dynamics,[6] and quantum mechanics, to name just a few.

To the best of our knowledge, only fixed step size implementations of geometric integrators are readily available.[7] It is the purpose of this work to develop a variable step size implementation of the composition methods and Gauss methods in the form of library integrators coded in Matlab m-files. Variable step size integrators are often more efficient than fixed step size integrators but their implementation is more complex. Matlab is a well-known numerical computing environment, with multiple applications in simulation of dynamic systems,[8,9], and is well suited to the coding of ODE solvers. Our Matlab implementation of geometric integrators therefore comes as a complement to the high-quality, general purpose, ODE suite in Matlab,[10] which does not include integrators with such capabilities. Besides, we also propose a new fixed step size implementation of linear multistep methods with improved performance. All the codes are made available as part of the MATMOL library (http://www.matmol.org/),[11] which is a Matlab toolbox containing various methods for the solution of partial differential equations using the method of lines (for instance, finite differences, finite element methods, and slope limiters for the resolution of steep moving fronts), and for time integration of differential equations. MATMOL has been used for

[1]Service de Mathématique et Recherche opérationnelle, Université de Mons, Belgium
[2]Service d'Automatique, Université de Mons, Belgium

Corresponding author:
Alain Vande Wouwer, Service d'Automatique, Université de Mons, 31 boulevard Dolez, 7000 Mons, Belgium.
Email: alain.vandewouwer@umons.ac.be

solving a wide spectrum of problems in chemical and environmental engineering.[12,13] The inclusion of geometric integrators in the set of ODE solvers currently available in the toolbox extends the range of problems that can be considered. In this connection, several examples, including the Kepler problem, the outer solar system, and a conservative Lotka–Volterra system, serve as illustration to demonstrate the use of the geometric solvers and their performance.

This paper is organized as follows. In the next section, the basic concepts, i.e., Hamiltonian and reversible systems, as well as geometric integration are introduced. The several methods considered in this study, i.e., composition methods, Gauss methods, and linear multistep methods, are then described in more details. Matlab implementation together with a simple application example is discussed and finally, the several solvers are tested with three case studies. The last section draws some conclusions.

2. Geometric integration and a few other basic concepts

This section introduces the main concepts, i.e., Hamiltonian and reversible systems, as well as geometric integration.

2.1. Hamiltonian systems

In a mechanical system, the Lagrangian is given by:

$$L(q,\dot{q}) = T(q,\dot{q}) - U(q) \tag{1}$$

where $T(q,\dot{q})$ is the kinetic energy and $U(q)$ is the potential energy, which are expressed in terms of the generalized coordinates and velocities $q = (q_1, \ldots, q_d)^{\mathrm{T}}$ and $\dot{q} = (\dot{q}_1, \ldots, \dot{q}_d)^{\mathrm{T}}$. The system motion is described by Lagrange equation:

$$\frac{d}{dt}\left(\frac{\partial L}{\partial \dot{q}}\right) = \frac{\partial L}{\partial q} \tag{2}$$

These equations can be reformulated by replacing the velocities $\dot{q}$, which are kinematic variables, by the momenta $p = (p_1, \ldots, p_d)^{\mathrm{T}}$, which are dynamic variables, and by introducing a new function $H(q,p) : \mathbb{R}^{2d} \to \mathbb{R}$, the Hamiltonian,

$$H(q,p) = p^T\dot{q} - L(q,\dot{q}) \tag{3}$$

The transformation $\dot{q} \to p$ leads to the equivalent Hamiltonian equations of movement:

$$\dot{q}_i = \frac{\partial H}{\partial p_i}(q,p) \qquad \dot{p}_i = -\frac{\partial H}{\partial q_i}(q,p) \qquad i = 1, \ldots, d \tag{4}$$

If the kinetic energy is a quadratic form $T(q,p) = 1/2\dot{q}^{\mathrm{T}}M(q)\dot{q}$, where $M(q)$ is a symmetric and positive definite matrix, then

$$\begin{aligned} p &= M\dot{q} \\ H(q,p) &= \tfrac{1}{2}p^{\mathrm{T}}M(q)^{-1}p + U(q) = T(q,p) + U(q) \end{aligned} \tag{5}$$

and the Hamiltonian represents the total energy of the system.

A classic example is the pendulum with equations:

$$\dot{q} = p, \quad \dot{p} = -\sin(q) \tag{6}$$

and a Hamiltonian given by:

$$H(q,p) = \frac{1}{2}p^2 - \cos(q) \tag{7}$$

Figure 1 shows solutions corresponding to several initial conditions. This system has interesting properties:

- It is a time-invariant system (no explicit time dependency), and $H(q,p)$ is constant along the solution at all times t. In this case, H is an *invariant* of the system, i.e., a quantity always preserved by the exact solution.
- For a system with $d = 1$, the area is conserved (volume conservation for $d \geqslant 2$), i.e.

$$area(S_0) = area(S_t),$$

where S_0 is a set of initial values and S_t the corresponding exact solutions at time t.

Figure 1(a) shows area preservation. Using a phase plane representation,[2] a set of initial values S_0 and the corresponding exact solutions at time $t = \pi$, given by set S_π, have exactly the same area. The same is true for the sets S_0' and S_π' although their shapes are very different.

Area preservation is a facet of a more general property called *symplecticity*, which is defined in detail by Hairer et al.[2] In particular it is shown that all systems of the form of Equations (4) with a twice continuously differentiable function $H(q,p)$ have (solution) flows that are symplectic transformations.

Excellent coverage of Hamiltonian systems and their integration is available in the literature.[2,3,14]

2.2. Reversible systems

The equations of a mechanical system, with Equation (5) as Hamiltonian, are as follows:

$$\begin{aligned} \dot{q} &= M^{-1}p \\ \dot{p} &= -\nabla_q H(q,p) = g(q,p), \end{aligned} \tag{8}$$

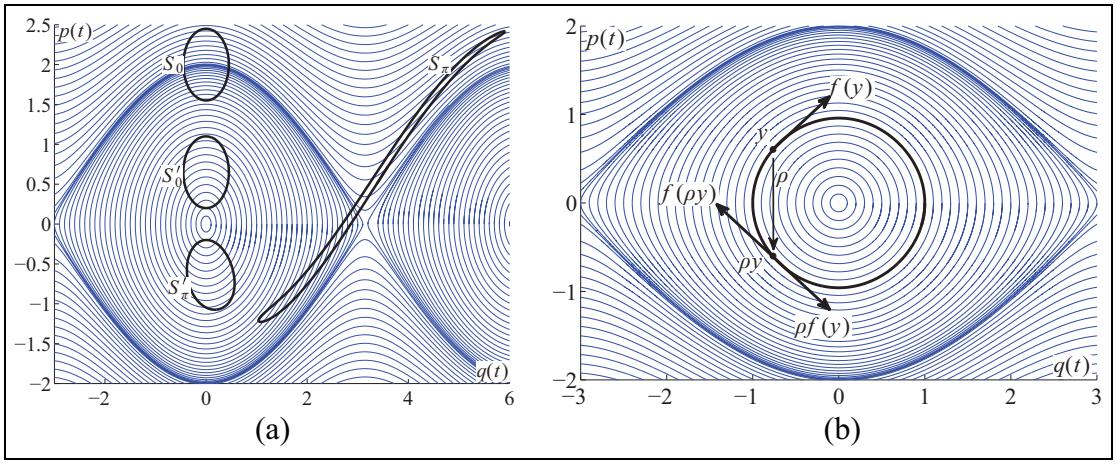

Figure 1. Geometric properties of the pendulum: symplecticity or area preservation (a) and reversibility (b).

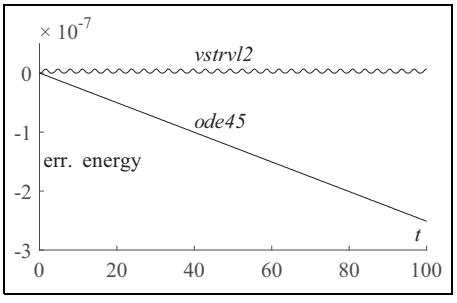

Figure 2. Energy preservation (error in the Hamiltonian of Equation (6)) for *ode45* and *vstrvl2*. Initial values $(q_0,p_0) = (0.4,0)$.

which satisfy $g(q, -p) = g(q,p)$. This implies that if $(q(t),p(t))$ is a solution of Equations (8) then $(q(-t), -p(-t))$ is also a solution. If the signs of the velocities are changed at a given time, the system will go back to its initial position after some time.[3] Such a system is called time-reversible.

More generally, if we consider a differential equation $\dot{y} = f(y)$ and a linear transformation ρ, the differential equation is said *ρ-reversible* if:

$$f(\rho y) = -\rho f(y) \qquad \text{for all } y \tag{9}$$

Figure 1(b) illustrates this geometric property of the pendulum. A second-order differential equation $\ddot{q} = g(q)$ is always ρ-reversible with the transformation $\rho(q,p) = (q, -p)$. A Hamiltonian system is reversible if $H(q,p) = H(q, -p)$, i.e., if H is quadratic in p.[3]

2.3. Geometric integration

A numerical integrator which preserves one of the above-mentioned properties is called a *geometric integrator*. Such an integrator is generally reliable on the long run. Figure 2 shows the preservation of energy of the pendulum when using a Störmer–Verlet method (as introduced in the next subsection and implemented in a code named *vstrvl2*), which is a geometric integrator for this problem, and a non-geometric integrator such as *ode45* from Matlab ODE suite.[10] The error in the energy grows linearly with *ode45*, but remains bounded with the geometric integrator.

A numerical integrator with step size h is a function $\phi_h(y)$ that gives an approximation of the exact solution $y(t)$ of a differential equation. For a one-step method, the numerical solution is given by $y_{n+1} = \phi_h(y_n)$. A numerical method ϕ_h is said to be

- symplectic if the function ϕ_h is symplectic;
- ρ-reversible if ϕ_h satisfies the two following conditions:

$$\begin{aligned} \rho \circ \phi_h &= \phi_{-h} \circ \rho \\ \phi_h &= \phi_{-h}^{-1} \end{aligned} \tag{10}$$

where ϕ_h^{-1} is the inverse function. The majority of numerical methods satisfy the first condition.

The second condition corresponds to the *symmetry* of the method. A one-step method is symmetric if changing $h \leftrightarrow -h$ and $y_n \leftrightarrow y_{n+1}$ leaves the method unchanged. For example, the implicit midpoint rule defined by the formula $y_{n+1} = y_n + hf((y_n + y_{n+1})/2)$ is symmetric, while the Euler explicit method $y_{n+1} = y_n + hf(y_n)$ is not.

2.4. Störmer–Verlet method

The Störmer-Verlet method,[15] or leap-frog method is one of the earliest method, often used in astronomy or molecular dynamics for its important properties.

For a second-order differential equation $\ddot{q} = g(q)$, the Störmer–Verlet method is a one-step method $\phi_h : (q_n, p_n) \rightarrow (q_{n+1}, p_{n+1})$ defined by:

$$\begin{aligned} q_{n+\frac{1}{2}} &= q_n + \frac{h}{2} p_n \\ p_{n+1} &= p_n + h g(q_{n+\frac{1}{2}}) \\ q_{n+1} &= q_{n+\frac{1}{2}} + \frac{h}{2} p_{n+1} \end{aligned} \tag{11}$$

where $q_{n+1/2}$ is an approximation of $q(t)$ at time $t_n + \frac{h}{2}$. This method is explicit, symplectic, symmetric and order 2 and uses only one function evaluation per step.[16]

3. Implementation of composition methods

The underlying idea is to exploit a simple, low-order method as a basic element of a composition process leading to a higher-order method that performs better than the basic one. If ϕ_h is the basic method of order p with a step h, and if $\gamma_1, \gamma_2, \ldots, \gamma_s$ are real numbers defining the individual steps $\gamma_1 h, \ldots, \gamma_s h$, a *composition* method ψ_h with s stages and step size h is given by:

$$\psi_h = \phi_{\gamma_s h} \circ \phi_{\gamma_{s-1} h} \circ \cdots \circ \phi_{\gamma_2 h} \circ \phi_{\gamma_1 h} \tag{12}$$

$$= \phi_{\gamma_s h}\left(\phi_{\gamma_{s-1} h}\left(\cdots\left(\phi_{\gamma_1 h}\right)\right)\right) \tag{13}$$

where the notation $f \circ g$ or $(f \circ g)(y)$ means $f(g(y))$. This method is at least of order $p + 1$ if:

$$\gamma_1 + \gamma_2 + \cdots + \gamma_s = 1 \tag{14}$$

$$\gamma_1^{p+1} + \gamma_2^{p+1} + \cdots + \gamma_s^{p+1} = 1 \tag{15}$$

These equations only have a real solution if p is even.

The interest of composition methods is to increase the order of accuracy while preserving certain properties of the basic method, such as[2]:

- if ϕ_h is symplectic, then the composition method ψ_h is symplectic too,
- if ϕ_h is symmetric and if γ_i satisfies the symmetry condition:

$$\gamma_i = \gamma_{s+1-i} \qquad \text{for all } i \tag{16}$$

then the method ψ_h is symmetric,

- if ϕ_h preserves an invariant (for example the angular momentum of a mechanical system), then the method ψ_h preserves it as well.

The basic method considered in our implementation is the Störmer–Verlet method, used in a composition process with different integration steps. For example the following method is of order 6 with 9 stages and satisfies the symmetry condition of Equation (16):

$$\begin{aligned} \gamma_1 = \gamma_9 &= 0.39216144400731413927925056 \\ \gamma_2 = \gamma_8 &= 0.33259913678935943859974864 \\ \gamma_3 = \gamma_7 &= -0.70624617255763935980996482 \\ \gamma_4 = \gamma_6 &= 0.08221359629355080023149045 \\ \gamma_5 &= 0.79854399093482996339895035 \end{aligned} \tag{17}$$

3.1. Step size selection

A variable step size integrator is usually more efficient than a fixed step size integrator. However, the standard step size control with a geometric integrator, i.e., based on a local error, is not recommended due to performance degradation.[2] There are several algorithms which allow extending geometric integrators to variable step size integration without losing the good long-time behavior of Figure 2. To implement the Störmer–Verlet method we have opted for the following step size selection because it is entirely explicit.[17,18] With the notation $y = (q, p)^{\mathrm{T}}$ we have the following:

$$\begin{aligned} y_{n+1} &= \psi_{h_{n+1/2}}(y_n) \\ h_{n+1/2} &= \varepsilon s_{n+1/2} \end{aligned} \tag{18}$$

where ψ is a composition method, $h_{n+1/2}$ is the step size at iteration n, and ε a constant accuracy parameter. Furthermore $s_{n+1/2}$ is given by the following (symmetric) recurrence relation:

$$\frac{1}{s_{n+1/2}} + \frac{1}{s_{n-1/2}} = \frac{2}{\sigma(y_n)}. \tag{19}$$

with $s_{1/2} = \sigma(y_0)$. For a correct long-time behavior the function $\sigma(y)$ must satisfy the two following conditions:

$$\sigma(y) > 0 \tag{20a}$$

$$\sigma(\rho y) = \rho \sigma(y) \tag{20b}$$

The condition given by Equation (20b) allows preservation of reversibility. The function σ directly influences the step size: if $\sigma(y)$ increases (or decreases) the step size does the same. In our implementation we use the following:

$$\sigma(y) = \frac{1}{\| f(y) \|} \tag{21}$$

which has the advantage to not require any knowledge of the solution. The condition of Equation (20b) is verified if ρ is orthogonal (because the determinant of an orthogonal matrix is ± 1), which is the case for many Hamiltonian systems. In particular, the condition is satisfied for a second-

order differential equation. Indeed, as explained before, the transformation is $\rho(q,p) = (q, -p)$ and the determinant is clearly equal to ± 1.

3.2. Round-off errors

In high accuracy long-time integration of differential equations, round-off errors may dominate truncation errors. A direct implementation of a numerical integrator shows that the sum of round-off errors typically increases linearly with time. Use of the Kahan algorithm,[19,20] also known as *compensated summation*, reduces considerably this sum which then grows as the square root of time. This algorithm consists of introducing a variable to accumulate small errors, and one can show that the results are close to those produced by an extended precision calculation.

3.3. Solver vstrvl2

The algorithm *vstrvl2* (*v*: the solver uses a variable step size strategy, *strvl*: the basic solver is the Störmer–Verlet method; *2*: the solver applies to second-order ODEs) is reversible and symplectic for second-order differential equations:

$$\ddot{q} = g(q) \quad q(0) = q_0 \quad \dot{q}(0) = \dot{q}_0 \tag{22}$$

It implements:

- a composition method (Equation (13)), up to order 10, with the Störmer–Verlet method (Equation (11)) as basic integrator,
- a selection of the order of the composition method through the coefficients γ_i in Equation (13),
- an accuracy parameter ε as in Equation (18),
- the step size control defined by Equations (18), (19), and (22),
- the compensated summation technique for reducing round-off errors.

The user can select the order of the method and the numbers of stages via the solver options (see Table 1).

The reader interested in complete implementation details can download the codes of the MATMOL library (http://www.matmol.org/).[11]

4. Implementation of Gauss methods

One of the most important families of methods in the context of geometric integration is that of *Gauss methods*, which are symmetric and symplectic. For an arbitrary number of stages s, they have the highest possible order $r = 2s$. They belong to the class of Runge–Kutta methods for first-order differential equations:

Table 1. Different composition methods.

Method	Order	Stages
21	2	1
43	4	3
45	4	5
67	6	7
69	6	9
815	8	15
817	8	17
1031	10	31
1033	10	33
1035	10	35

$$\dot{y} = f(y) \tag{23}$$

For a general problem $\dot{y} = f(t,y)$, an s-stage Runge–Kutta method is a one-step method $y_n \rightarrow y_{n+1}$ defined by:

$$Z_{i,n} - h\sum_{j=1}^{s} a_{ij} f(y_n + Z_{j,n}) = 0 \qquad i = 1, \ldots, s \tag{24a}$$

$$y_{n+1} = y_n + h\sum_{i=1}^{s} b_i f(t_n + c_i h, y_n + Z_{i,n}) \tag{24b}$$

No explicit Runge–Kutta method can be symplectic or symmetric,[2] and only a few of the implicit Runge–Kutta methods have both properties.

The implicit character implies that the nonlinear system of Equation (24a) for the values $Z_{1,n}, \ldots, Z_{s,n}$ has to be solved at every step.

To construct a Gauss method,[21] the coefficients $c_1, \ldots, c_s$ are first defined as the roots of the Legendre polynomial of order s:

$$\frac{d^s}{dx^s}(x^s(x-1)^s) \tag{25}$$

where s is the number of stages.

The coefficients a_{ij} and b_i can then be obtained by solving the following two linear equations which represent the order conditions:

$$\sum_{j=1}^{s} a_{ij} c_j^{k-1} = \frac{c_i^k}{k}, \qquad k = 1, \ldots, s, \qquad \forall i \tag{26a}$$

$$\sum_{i=1}^{s} b_i c_i^{k-1} = \frac{1}{k}, \qquad k = 1, \cdots, s. \tag{26b}$$

4.1. Step size selection

The main idea to construct an appropriate step size control for Gauss methods is to introduce a time transformation $t \leftrightarrow \tau$ given as the solution of a differential equation

$\frac{dt}{d\tau} = \sigma(y)$ and to write the differential system in terms of the new independent variable τ.[2]

From the chain rule $\frac{dy}{d\tau} = \frac{dy}{dt}\frac{dt}{d\tau}$, the transformed system is given by:

$$y' = \sigma(y) f(y) \tag{27a}$$

$$t' = \sigma(y) \tag{27b}$$

where prime indicates a derivative with respect to τ. Integrating, with fixed step size ε, Equation (27a) yields an approximation y_n of $y(\tau_n) = y(t_n)$, where $\tau_n = n\varepsilon$. Integrating Equation (27b) gives the corresponding non-equidistant times $t_{n+1} \approx t_n + \varepsilon\sigma(y_n)$. This algorithm therefore provides variable step size numerical solutions to Equation (23).

Again, the function $\sigma(y)$ has to satisfy the conditions of Equation (20) and, in this work, we use the function in Equation (21). The long-time behavior is preserved as long as ρ is orthogonal.

4.2. Starting approximation

As recommended by Hairer et al.,[2] fixed-point iteration is used to solve the nonlinear system (Equation (24a)).

To ensure convergence, the iterations are pursued as long as the relative Euclidean norm of the difference between two successive iterations $\underline{Z}_k$ and $\underline{Z}_{k+1}$ (the vector $\underline{Z} = [Z_1 \cdots Z_s]$) is larger than an accuracy parameter ε_p related to the machine precision:

$$\frac{1}{s}\left\| \frac{\underline{Z}_k - \underline{Z}_{k+1}}{\underline{Z}_k} \right\| \geqslant \varepsilon_p \tag{28}$$

Two safeguards complete this test:

1. When the norm of Equation (28) tends to grow while remaining less than $10\varepsilon_p$, the iterations are stopped, this trend being attributed to the accumulation of the rounding errors.
2. A stop test is also provided with respect to the maximum number of iterations, which is imposed a priori by the user (when this number is reached, a warning or error message is displayed according to whether the norm of Equation (28) is lower or no to $10^5\varepsilon_p$).

To achieve satisfactory performance, it is essential to have a good starting approximation to initiate the iterations of the fixed-point method. As the transformed system (Equation (27)) is integrated with a fixed step size, the underlying idea is to use the starting approximation of Laburta.[22]

Denoting the transformed system (Equation (27)) $y' = F(y)$, the starting approximation $Z^0_{i,n}$ is given by:

$$\begin{aligned} Z^0_{i,n} &= h\sum_{j=1}^{s} \alpha_{ij} F(y_{n-1} + Z_{j,n-1}) \\ &+ h\sum_{j=1}^{m} \nu_{ij} F(y_{n-1} + Z_{s+j,n-1}), \end{aligned} \tag{29}$$

where the quantity $F(y_{n-1} + Z_{i,n-1})$ is already known from the previous step and the several terms $F(y_{n-1} + Z_{s+i,n-1})$ are m additional function evaluations, which can be computed from:

$$Z_{s+i,n-1} = h\sum_{j=1}^{s+i-1} \mu_{ij} F(y_{n-1} + Z_{j,n-1}). \tag{30}$$

The coefficients $\alpha_{ij}, \nu_{ij}, \mu_{ij}$ are obtained from a Vandermonde-type linear system.[22] In our codes we use $m = 3$, which allows constructing a starting approximation of order $s + 2$ and reducing the number of iterations of the fixed-point method.

4.3. Solver vgauss1

The Matlab solver *vgauss1* (*v*: the solver uses a variable step size strategy; *gauss*: the solver is based on the Gauss method; *1*: the solver applies to first-order ODEs) is intended for reversible and symplectic first-order differential equations:

$$\dot{y} = f(y) \quad y(0) = y_0. \tag{31}$$

It implements:

- Gauss methods up to order 14 defined by Equations (24), (25), and (26),
- step size control defined by the transformed system (Equation (27)) and function $\sigma(y)$ given by Equation (21),
- accuracy parameters, which are the fixed step size ε used to integrate the transformed system (Equation (27)) as well as the order of the Runge–Kutta method (coefficients a_{ij}, b_i and c_i in Equation (24)),
- a starting approximation of Laburta defined by Equation (29),
- the compensated summation technique.

The reader interested in complete implementation details can download the codes of the MATMOL library (http://www.matmol.org/).[11]

5. Implementation of linear multistep methods

Linear multistep methods are appealing because they can achieve high order while requiring only one function evaluation per step. They have an excellent long-time behavior when applied to second-order Hamiltonian systems.

For the problem:

$$\ddot{q} = f(q) \tag{32}$$

a linear multistep method is defined by the formula:

$$\sum_{j=0}^{k} \alpha_j q_{n+j} = h^2 \sum_{j=0}^{k} \beta_j f(q_{n+j}) \tag{33}$$

where α_j, β_j are real parameters, $\alpha_k \neq 0$, and $|\alpha_0| + |\beta_0| > 0$. If $\beta_k = 0$ the method is explicit, otherwise it is implicit. Aside from an initial value $q(t_0) = q_0$, a starting procedure is required to get approximations $q_1, \ldots, q_{k-1}$ to $q(t_0 + h), \ldots, q(t_0 + (k-1)h)$.

The generating polynomials are characteristic of the method, and are given by:

$$\rho(\xi) = \sum_{i=0}^{k} \alpha_i \xi^i \tag{34}$$

$$\vartheta(\xi) = \sum_{i=0}^{k} \beta_i \xi^i \tag{35}$$

The method of Equation (33) has order r if the coefficients α_j and β_j satisfy the following:

$$\vartheta(\xi) = \frac{\rho(\xi)}{\log^2 \xi} + \mathcal{O}((\xi - 1)^r) \quad \text{for } \xi \to 1 \tag{36}$$

The method of Equation (33) is *stable* if all roots of $\rho(\xi)$ satisfy $|\xi| \leqslant 1$, and those on the unit circle are at most double zeros.

If the coefficients of Equation (33) satisfy the following:

$$\alpha_{k-j} = \alpha_j, \qquad \beta_{k-j} = \beta_j \qquad \text{for all } j \tag{37}$$

then the method is *symmetric*. It is easy to show that the condition of Equation (37) implies that for all zeros ξ of $\rho(\xi)$, its inverse ξ^{-1} is also a zero. Hence, for stable symmetric methods all zeros of $\rho(\xi)$ lie on the unit circle and they are at most of multiplicity two.

A symmetric multistep method (Equation (33)) is called *s-stable* if, except for a double zero in 1, all zeros of $\rho(\xi)$ are simple and of modulus one. One can show that k is always even for a *s*-stable method.

Although those methods can have high order with only one function evaluation per step, they are characterized by resonance phenomena and instability (in particular if large steps are used). Undesired oscillations called parasitic solutions are also observed. It was discovered that *s*-stable methods have excellent performance when applied to a second-order differential equation.[23] In this case, the parasitic solution get under control and the error in the energy and in the angular momentum remain bounded (as in Figure 2). A long-time analysis is known only for a problem of the form[2]:

$$\ddot{q} = -\nabla U(q). \tag{38}$$

Among the system (Equation (38)), there are Hamiltonian systems with constant mass matrix.

To construct a symmetric *s*-stable method of order k (with k even), we first define the polynomial ρ by:

$$\rho(\xi) = (\xi - 1)^2 \prod_{j=1}^{k/2-1} (\xi^2 + 2a_j\xi + 1) \tag{39}$$

where a_j are real distinct numbers which satisfy $-1 < a_j < 1$. The method is thus *s*-stable: there is a double zero in $\xi = 1$, and the other zeros are complex conjugates and lie on the unit circle. The unique polynomial $\vartheta(\xi)$ of order $k-1$ can be found thanks to the order condition of Equation (36). A Taylor series expansion in $\xi = 1$ truncated at order k gives the coefficients β_i of Equation (33) (this computation can be automated using Matlab symbolic computation and the function *taylor*). Because Equation (35) is of order $k-1$ we have $\beta_k = 0$ and the method is explicit.

If the derivatives $p = \dot{q}$ are required, they can be computed by finite differences of order r:

$$p_n = \frac{1}{h} \sum_{j=-l}^{l} \delta_j q_{n+j}. \tag{40}$$

The algorithm of Fornberg allows derivatives of arbitrary order to be obtained.[24]

5.1. Solver *flmm2*

The Matlab solver *flmm2* (*f*: the solver uses a fixed step size strategy; *lmm*: the solvers uses a linear multistep method; *2*: the solver applies to second-order ODEs) is dedicated to second-order differential equations:

$$\ddot{q} = g(q) \quad q(0) = q_0 \quad \dot{q}(0) = \dot{q}_0 \tag{41}$$

It implements:

- fixed step size linear multistep methods (Equation (33)) up to order 12 using the generating polynomials (Equations (39) and (36)),
- an initialization (computation of the first steps) with a high-order Gauss method,
- the stabilization process and compensated summation technique described by Console and Hairer.[25]

The reader interested in complete implementation details can download the codes of the MATMOL library (http://www.matmol.org/).[11]

Note that variable step size methods are recommended to solve specific problems in astronomy.[26]

6. Tutorial introduction to the use of the library integrators

In this section we provide a tutorial introduction to the use of the solvers based on the simple pendulum example. The solvers can be called in a way similar to the standard solvers of the Matlab ODE suite.[10] The general calling sequence is:

```
[ T,Q,P] = ...
solver('odefun',tspan,y0,options,varargin)
```

with the followings arguments:

- "odefun" is a string that refers to the function evaluating the right-hand side of the differential equations $\dot{y} = f(y)$.
- tspan=[t0 tf] is a row vector of dimension 2 defining the time span.
- y0 is a column vector containing the initial conditions $y_0 = (q_0\ \dot{q}_0)^{\mathrm{T}}$.
- options allows modifying default parameters. It is a structure created by the function *gni_set*. The syntax is the same as *odeset* in Matlab but the available options are different.
- varargin contains an arbitrary number of optional arguments which are passed over to the function.
- The output [T,Q,P] contains, respectively, the times at which the solution is evaluated and the values of q and $\dot{q}$ at these times.

An additional function allows adjusting the default integration parameters of each solver. It is used in the same way as *odeset* in Matlab. The list of available options depends on the solver.

For each solver, it is possible to select the order of the method and a parameter influencing accuracy: ε in Equation (18) of *vstrvl2*, the fixed step size used to integrate the modified system (Equation (27)) in *vgauss1* and the fixed step size h in Equation (33) of *flmm2*. It is also possible to use the output Matlab functions to visualize the solution and to change the number of output points (*Refine* option in the Matlab ODE suite).

An important option of *vgauss1* is the possibility to use a *vectorized* ODE function (Equation (23)). (When the ODE function $\dot{y} = f(y)$ is *vectorized* then $f([y_1 \ldots y_n])$ returns $[f(y_1) \ldots f(y_n)]$.) This allows the solver to reduce the required number of function evaluations, and might significantly decrease the computation time.

Finally, the solver *vstrvl2* offers the possibility to replace the default function (Equation (21)) for the step size selection, by another function tailored to the problem under consideration.[17]

6.1. A simple illustrative example

The pendulum problem is now solved using *vstrvl2*. The equations are given by Equation (6), or equivalently $\ddot{q} = -\sin q$ which is defined in a Matlab function *G_pendul.m*. This function has two inputs: the scalar t and the column vector q:

```
function out = G_pendul(t,q)
out = -sin(q);
```

This problem is solved over the time interval [0 100] with method "69" corresponding to Equation (17). The accuracy parameter in Equation (18) is fixed to $\varepsilon = 0.1$ and the initial conditions are taken as $q_0 = 0.4$ and $\dot{q}_0 = 0$. The corresponding calling sequence is:

```
options =
gni_set('method','69','precision',0.1);
[ T,Q,P] = gni_vstrvl2('G_pendul',[ 0 100],[ 0.4
...
0],options);
```

tspan, y0, and options are optional and can also be defined in function *G_pendul.m*:

```
function[ out,out2,out3] = G_pendul(t,q,flag)
  if (nargin < 3) || isempty(flag)
    out = -sin(q);
  else
    switch flag
      case 'init',
        out =[ 0 100];
        out2 =[ 0 0.4];
        out3 = gni_set('Precision',0.1);
    end
  end
```

At the initialization stage, out, out2, and out3 define, respectively, the time span for integration, the initial conditions, and accuracy options. This way, the solver has an even simpler calling sequence:

```
[ T,Q,P] = gni_vstrvl2('G_pendul');
```

It is also possible to use the graphical output functions of Matlab (*odeplot, odephas2*) as follows (options are the same for the Matlab ODE suite), to for instance, visualize the first component $q(t)$ as a function of time:

```
options =...
gni_set('OutputFcn','odeplot',
'OutputSel',1);
gni_vstrvl2('G_pendul',[],[],options);
```

Some numerical results for this example are illustrated in Figure 1, which takes the form of phase plane plot.

The step size selection function used by the solver is by default $\sigma(q,p) = \| f(q,p)\|^{-1}$ (Equation (21)). This function is suitable for the majority of problems but is not necessarily optimal. It is possible to tailor this function to a specific problem. For example, $\sigma(q,p) = \| q\|^{\alpha}$ could be a very good choice in a two attracting body problem (the closer the bodies the smaller the step size). In this case, we first create a Matlab file *resc_2body.m*

```
function out = resc_2body(t,q,p,varargin)
alpha = 3/2;
out = norm(q)^alpha;
```

and add the line:

```
options = ...
gni_set(options,'RescalFun','resc_2body');
```

7. Test examples and numerical results

In this section, the several solvers are tested with three case studies, e.g. the Kepler problem, the outer solar system, and a conservative Lotka–Volterra system.

7.1. Kepler problem

For computing the motion of two bodies which attract each other, one is placed at the center of the coordinate system, while the coordinates of the second are (q_1, q_2), which are defined by:

$$\ddot{q}_1 = \frac{-q_1}{(q_1^2 + q_2^2)^{2/3}} \tag{42a}$$

$$\ddot{q}_2 = \frac{-q_2}{(q_1^2 + q_2^2)^{2/3}} \tag{42b}$$

The corresponding Hamiltonian is as follows:

$$H(q_1, q_2, p_1, p_2) = \frac{1}{2}(p_1^2 + p_2^2) - \frac{1}{\sqrt{q_1^2 + q_2^2}}, \qquad p_i = \dot{q}_i \tag{43}$$

It is a reversible and periodic system. The trajectories are elliptic orbits, with the reference body in one of the foci. With the following initial values:

$$\begin{aligned} q_1(0) &= 1 - e, \qquad q_2(0) = 0, \\ \dot{q}_1(0) &= 0, \qquad \dot{q}_2(0) = \sqrt{\tfrac{1+e}{1-e}} \end{aligned} \tag{44}$$

the trajectory is an ellipse with eccentricity e ($0 \leqslant e < 1$) and period 2π.

The Kepler problem is solved with the three proposed solvers as well as with *ode113* of the Matlab ODE suite,[10] which is known to be very efficient at stringent tolerances but which is not a geometric integrator. We choose $e = 0.8$ and a time interval of 200 periods. We compute the error $err = \| y_0 - y_{fin} \|$ where y_0 is the initial condition and y_{fin} the final solution. Because the problem is 2π-periodic, we should have $err = 0$ with an integrator with infinite precision.

Figure 3 compares the results of the several solvers. We observe that the solver *ode113* requires fewer function evaluations. However, the three proposed solvers outperform *ode113* in terms of CPU time. This can be explained by a larger overhead (total CPU time minus that used for the function evaluations) in *ode113*. Nonetheless, only the variable step size integrators (*vstrvl2* and *vgauss1*) have a clear advantage over *ode113* in terms of CPU time. This is related to the relatively high eccentricity of the problem ($e = 0.8$) which favors variable step size methods.

Notice that with *vgauss1* solver, the s function evaluations of the fixed-point iteration can be achieved in parallel (option *vectorized*), which can reduce considerably the CPU time. This is not possible with the three other solvers.

To illustrate the advantage of our variable step size integrators (*vstrvl2* and *vgauss1*), we compare them with fixed step size versions. To this end, we use the codes of Hairer and Hairer[7]: the integrator *gni_irk2* is a fixed step size implementation of Gauss methods for second-order differential equations, while *gni_comp* is a fixed step size implementation of composition methods with Störmer–Verlet as a basic method.

Figure 4 shows the error at the end of the integration interval as a function of the total numbers of function evaluations as well as a function of the CPU time. As we can see, the variable step size methods are more efficient than their fixed step size counterparts. Note that the difference between the Gauss methods is much smaller than between the composition methods. As explained by Hairer et al.,[2] the implementation of a second-order differential equation allows the use of a Gauss–Seidel iteration which

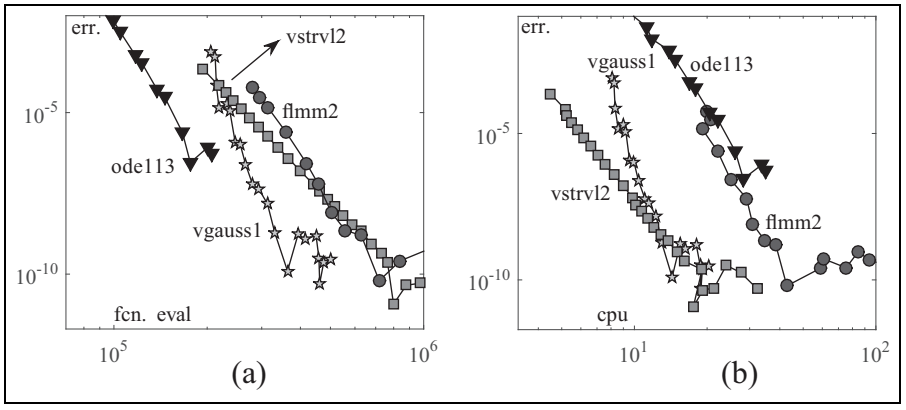

Figure 3. Accuracy–workload diagram for the Kepler problem using *ode113* from Matlab and the three proposed solvers (*vgauss1*, *vstrvl2*, and *flmm2*). Each point corresponds to an accuracy parameter.

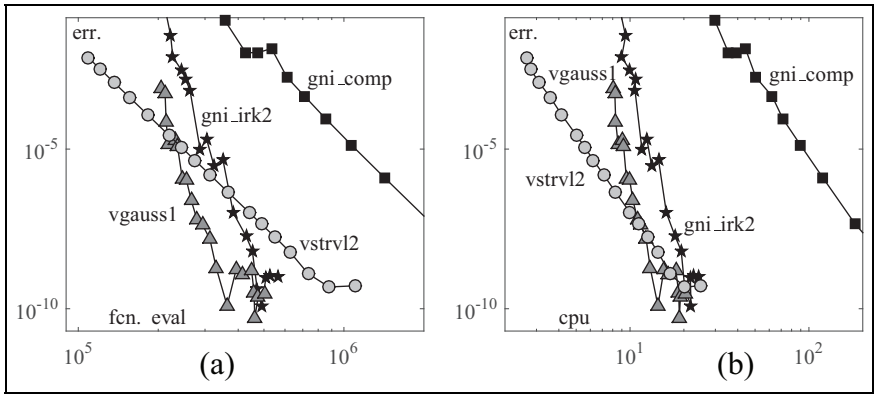

Figure 4. Accuracy–workload diagram for the Kepler problem. Comparison between our variable step size integrators (*vgauss1*, *vstrvl2*) and the fixed versions of Hairer and Hairer (*gni_irk2* and *gni_comp*).[7]

converges faster that a fixed-point iteration. Our implementation, however, can be used to solve first-order differential equations or problems which require variable step size integration.

7.2. The outer solar system

The outer solar system is a six-bodies system: Sun, Jupiter, Saturn, Neptune, Uranus, and Pluto. If we denote by m_i the mass of the ith body, this system, for the Hamiltonian of Equation (5) where $(M_{ii}) = m_i$, is diagonal and the potential function is given by:

$$U(q) = -G \sum_{i=1}^{5} \sum_{j=0}^{i-1} \frac{m_i m_j}{\| q_i - q_j \|}$$

with G the gravitational constant. Initial values and planet masses can be found in Hairer et al.[2]

We integrate this problem on a time interval of one million years (which is not so much in astronomy). Figure 5 shows the comparative results. Because of the low eccentricity and the costly function evaluation of this problem, the linear multistep method is now clearly the best.

7.3. Conservative Lotka–Volterra systems

Generalized Lotka–Volterra models describe competition and predation among n interacting species. They are useful in mathematical ecology for modeling food webs. Here we consider the general formulation of conservative Lotka–Volterra systems[27]:

$$\dot{x}_i = x_i(r_i + \sum_{j=1}^{n} a_{ij} x_j), \qquad a_{ij} = -a_{ji}, \qquad i = 1, \ldots, n. \tag{45}$$

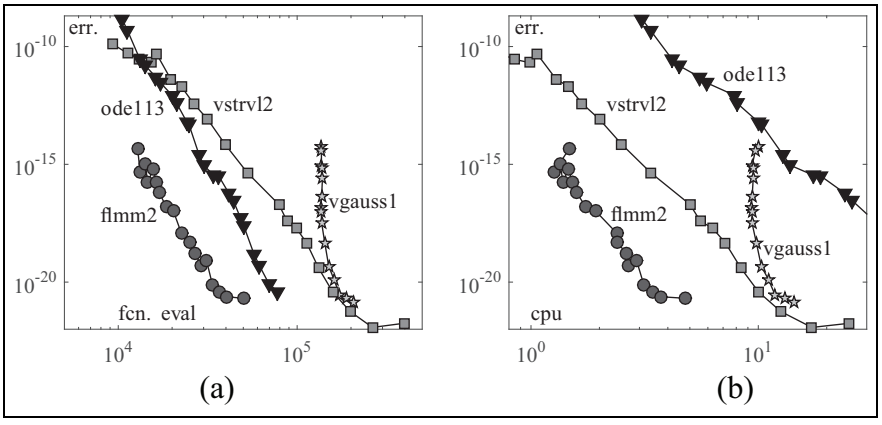

Figure 5. Work–precision diagram for the outer solar system using *ode113* and the three proposed solvers (*vstrvl2, vgauss1*, and *flmm2*).

where x_i represents the number of individuals of species i ($x_i > 0$). The coefficients a_{ij} describe the interactions between the populations and $A = (a_{ij})$ is the *interaction matrix*. The parameters r_i represent the growth rates of the populations, subsuming birth and mortality, and depending on the environment in the following way: if $r_i > 0$ the species i is able to reproduce alone, using food from the environment, and if $r_i < 0$, the species i cannot survive when left alone in the environment (as a predator that cannot persist without its preys).

The condition $a_{ij} = -a_{ji}$ is required for a conservative system, i.e., a system that has at least an invariant.[27] This condition also implies that the interaction matrix A is *skew-symmetric*.

If x^* is a steady state of Equation (45), i.e., a solution of $r + Ax^* = 0$ where $r = (r_i)$ and $A = (a_{ij})$, a transformation $(x_i \rightarrow e^{y_i})$ can be applied to the system (Equation (45)) so as to put it in the following form:

$$\dot{y}_i = \sum_{j=1}^{n} a_{ij} \frac{\partial H(y)}{\partial y_j}, \tag{46}$$

with

$$H(y) = \sum_{j=1}^{n} (e^{y_j} - x_j^* y_j). \tag{47}$$

The Hamiltonian structure of the Lotka–Volterra equations is discussed in details in several papers, for example.[27,28] In particular it is shown that $H(y)$ is an invariant for the system given by Equation (46).

In the paper by Jay,[29] it is shown that symplectic Runge–Kutta methods (such as Gauss methods) preserve the structure of the transformed Lotka–Volterra system (Equation (46)). Consequently, in order to integrate the system (Equation (46)), we use a fixed step size version of Gauss methods for first-order differential equations (not discussed in this article but also available in our Matlab package).

Examples of conservative Lotka–Volterra systems of dimension 2, 3, or 4 can already be found in the literature,[2,27] so that we consider here an example of dimension $n = 5$. Arbitrarily, we choose the following parameters:

$$A = \begin{pmatrix} 0 & 2 & -0.5 & 0 & 0.2 \\ -2 & 0 & 3 & 0.3 & -0.5 \\ 0.5 & -3 & 0 & 0.6 & -0.4 \\ 0 & -0.3 & -0.6 & 0 & 1 \\ -0.2 & 0.5 & 0.4 & -1 & 0 \end{pmatrix}, \tag{48}$$

$$r = (-5.8, 5.55, 7.2, -1.9, 0.2)^T, \tag{49}$$

$$x^* = (5, 3, 2, 1.5, 4)^T. \tag{50}$$

In this test, we compare the efficiency of the Matlab integrators *ode45* and *ode113* and a fixed step size implementation of Gauss method of order 14. Figure 6(a) shows the maximal error in the invariant (Equation (47)) as a function of CPU time. The Gauss method is definitively more efficient than *ode45* and appears to be more efficient than *ode113* from a certain tolerance on. From our experience, this integrator appears to be very efficient at stringent tolerances.

Figure 6(b) shows the error in the invariant as a function of time. A linear drift in the invariant is visible with *ode45* whereas the Gauss method shows the correct behavior. The error remains bounded and small. For an explanation of this long-time behavior, we refer to the theory of *backward error analysis*.[2]

8. Conclusion

It is natural to use numerical integrators that preserve geometric properties when solving differential equations. This

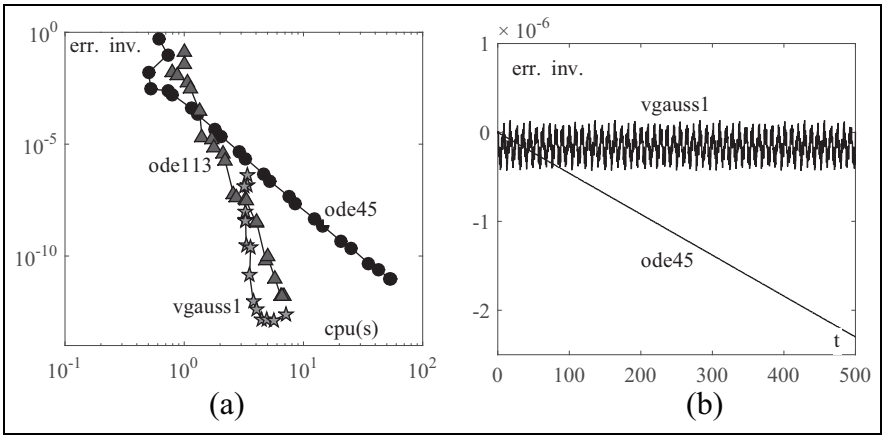

Figure 6. Comparison of the efficiency and behavior of Matlab integrators *ode45* and *ode113* and a fixed step size implementation of Gauss method of order 14: maximal error in the invariant (Equation (47)) as a function of CPU time (a) and as a function of time (b). The problem of Equation (46) with initial values $x^0 = (9,4,3,1.7,6)$ and time interval $[0; 500]$.

has been shown very successful in many fields, such as molecular dynamics, celestial mechanics.

We have implemented a Matlab library of dedicated solvers for Hamiltonian and reversible systems. Variable step size integrators are often better than their fixed step size counterparts and the numerical results show excellent long time preservation of the Hamiltonian. The codes can be downloaded from http://www.matmol.org/, and we hope that the availability of these geometric solvers will contribute to the popularity of these methods among non-specialist users.

Funding

This research received no specific grant from any funding agency in the public, commercial, or not-for-profit sectors.

References

1. McLachlan RI and Quispel GRW. Geometric integrators for ODEs. *J Phys A Math Gen* 2006; 39: 5251–5285.
2. Hairer E, Lubich C and Wanner G. *Geometric numerical integration. Structure-preserving algorithms for ordinary differential equations*. 2nd ed. Springer Series in Computational Mathematics 31. Berlin: Springer-Verlag, 2006.
3. Leimkuhler B and Reich S. *Simulating Hamiltonian dynamics*. Cambridge Monographs on Applied and Computational Mathematics 14. Cambridge: Cambridge University Press, 2004.
4. Gladman B, Duncan M and Candy J. Symplectic integrators for long-term integrations in celestial mechanics. *Celestial Mech Dyn Astron* 1991; 52: 221–240.
5. Forest E. Geometric integration for particle accelerators. *J Phys A Math Gen* 2006; 39: 5321–5377.
6. Leimkuhler B and Reich S. A reversible averaging integrator for multiple time-scale dynamics. *J Comput Phys* 2001; 171: 95–114.
7. Hairer E and Hairer M. GniCodes – Matlab programs or geometric numerical integration. In: *Frontiers in numerical analysis*. Berlin: Springer, 2003, pp.199–240.
8. Klee H and Allen R. *Simulation of dynamic systems with Matlab® and Simulink ®*. Boca Raton, FL: CRC Press, 2012.
9. Vande Wouwer A, Saucez P, Vilas C, et al. *Simulation of Ode/Pde models with Matlab, Octave and Scilab*. Berlin: Springer, 2014.
10. Shampine LF and Reichelt MW. The Matlab ODE suite. *J Sci Comput* 1997; 18: 1–22.
11. Vande Wouwer A, Saucez P and Schiesser W. Simulation of distributed parameter systems using a Matlab-based method of lines toolbox chemical engineering applications. *Ind Eng Chem Res* 2004; 43: 3496–3477.
12. David R, Vasel J-L and Vande Wouwer A. Settler dynamic modeling and Matlab simulation of the activated sludge process. *Chem Eng J* 2009; 146: 174–183.
13. Logist F, Saucez P, Van Impe J, et al. Simulation of (bio) chemical processes with distributed parameters using Matlab. *Chem Eng J* 2009; 155: 603–616.
14. Sanz-Sernar JM and Calvo MP. *Numerical Hamiltonian problems*. London: Chapman & Hall, 1994.
15. Verlet L. Computer experiments on classical fluids. I. Thermodynamical properties of Lennard-Jones molecules. *Phys Rev* 1967; 159: 98–103.
16. Hairer E, Lubich C and Wanner G. Geometric numerical integration illustrated by the Störmer–Verlet method. *Acta Numer* 2003; 12: 399–450.
17. Hairer E. Variable time step integration with symplectic methods. *Appl Numer Math* 1997; 25: 219–227.
18. Huang W and Leimkuhler B. The adaptive Verlet method. *SIAM J Sci Comput* 1997; 18: 239–256.
19. Higham NJ. The accuracy of floating point summation. *SIAM J Sci Comput* 1993; 14: 783–799.
20. Kahan W. Further remarks on reducing truncation errors. *Commun ACM* 1965; 8: 40.

21. Hairer E, Nørsett SP and Wanner G. *Solving ordinary differential equations. I. Nonstiff problems*. 2nd ed. Springer Series in Computational Mathematics 8. Berlin: Springer 1993.
22. Laburta MP. Starting algorithms for IRK methods. *J Comput* 1997; 83: 269–288.
23. Quinlan GD and Tremaine S. Symmetric multistep methods for the numerical integration of planetary orbits. *Astron J* 1990; 100: 1694–1700.
24. Fornberg B. Generation of finite difference formulas. *Math Comput* 1988; 51: 699–706.
25. Console P and Hairer E. Reducing round-off errors in symmetric multistep methods. *Comput Appl Math* 2014; 262: 217–222.
26. Cano B and Durán A. A technique to construct symmetric variable-stepsize linear multistep methods for second-order systems. *Math Comput* 2003; 72: 1803–1816.
27. Duarte P, Fernandes RL and Oliva WM. Dynamics of the attractor in the Lotka–Volterra equations. *J Differ Equations* 1998; 149: 143–189.
28. Idema T. *The behaviour and attractiveness of the Lotka–Volterra equations*. PhD Thesis, Universiteit Leiden, Belgium, 2005.
29. Jay LO. Preserving Poisson structure and orthogonality in numerical integration of differential equations. *Comput Math Appl* 2004; 48: 237–255.

Author biographies

Guillaume Chauvon graduated in engineering from University of Mons in 2013.

Philippe Saucez is a retired professor at the University of Mons, Department of Mathematics and Operational Research. His research interests include numerical analysis of partial differential equations.

Alain Vande Wouwer is a professor at the University of Mons, Department of Automatic Control. His research is in the area of system dynamics, estimation and control, with applications in biological systems and mechatronics.