Numer Algor (2018) 78:63–86
DOI 10.1007/s11075-017-0367-0

ORIGINAL PAPER

Efficient implementation of symplectic implicit Runge-Kutta schemes with simplified Newton iterations

Mikel Antoñana[1] · Joseba Makazaga[1] · Ander Murua[1]

Received: 22 March 2017 / Accepted: 19 June 2017 / Published online: 28 June 2017

Abstract We are concerned with the efficient implementation of symplectic implicit Runge-Kutta (IRK) methods applied to systems of Hamiltonian ordinary differential equations by means of Newton-like iterations. We pay particular attention to time-symmetric symplectic IRK schemes (such as collocation methods with Gaussian nodes). For an s-stage IRK scheme used to integrate a d-dimensional system of ordinary differential equations, the application of simplified versions of Newton iterations requires solving at each step several linear systems (one per iteration) with the same $sd \times sd$ real coefficient matrix. We propose a technique that takes advantage of the symplecticity of the IRK scheme to reduce the cost of methods based on diagonalization of the IRK coefficient matrix. This is achieved by rewriting one step of the method centered at the midpoint on the integration subinterval and observing that the resulting coefficient matrix becomes similar to a skew-symmetric matrix. In addition, we propose a C implementation (based on Newton-like iterations) of Runge-Kutta collocation methods with Gaussian nodes that make use of such a rewriting of the linear system and that takes special care in reducing the effect of round-off errors. We report some numerical experiments that demonstrate the reduced round-off error propagation of our implementation.

✉ Mikel Antoñana
Mikel.Antonana@ehu.eus

Joseba Makazaga
Joseba.Makazaga@ehu.eus

Ander Murua
Ander.Murua@ehu.eus

[1] Computer Science and Artificial Intelligence Department, UPV/EHU (University of the Basque Country), Donostia, Spain

Keywords Symplectic implicit Runge-Kutta schemes · Simplified Newton iteration · Efficient implementation · Round-off error propagation

1 Introduction

The main goal of the present work is the efficient implementation of symplectic implicit Runge-Kutta schemes for stiff Hamiltonian ordinary differential equation (ODE) problems. Our primary interest is on geometric numerical integration, which motivates us to solve the implicit equations determining each step to full machine precision. The stiff character of the target problems leads us to solving the implicit equations by some modified version of Newton method. This typically requires repeatedly solving linear systems of equations with coefficient matrices of the form

$$(I_s \otimes I_d - h\, A \otimes J) \in \mathbb{R}^{sd \times sd} \tag{1}$$

where $A \in \mathbb{R}^{s \times s}$ is the coefficient matrix of the RK scheme and J is some common approximation of the Jacobian matrices evaluated at the stage values.

A standard approach, independently introduced in [13], [5], and [3], to efficiently solve such linear systems takes advantage of the special structure of the matrix (1). More specifically, (1) is similar to a block-diagonal matrix with s blocks of size $d \times d$ of the form $I_d - h\,\lambda_j J$ ($j = 1, \ldots, s$), one per eigenvalue λ_j of A. Typically, the coefficient matrix A of standard high-order implicit RK schemes has $[s/2]$ complex conjugate pairs of eigenvalues (plus a real one for odd s).

The main contribution of the present paper is a technique for transforming s d-dimensional systems with coefficient matrix (1) into an equivalent $(s+1)d$-dimensional systems that is similar to a matrix with a blockwise sparse structure that can be favorably exploited. We pay particular attention to implicit Runge-Kutta schemes that are both time-symmetric and symplectic. (However, our technique is also applicable for symplectic IRK schemes that are not time-symmetric, and also for some time-symmetric non-symplectic IRK schemes; see last paragraph in Section 3.3 for more details.)

Compared to the standard approach based on the complex diagonalization of the RK matrix, our technique allows us completely avoiding complex arithmetic, and according to our complexity analysis for dense linear algebra (Section 3.6), it requires fewer arithmetic operations (less than half for Hamiltonian problems). We also show that our technique has potential advantages when the linear systems are solved by iterative methods preconditioned by means of incomplete LU factorizations.

A second contribution of the paper is a C code that implements time-symmetric symplectic IRK schemes (such as RK collocation methods with Gaussian nodes) based in Newton-like iterations (with the arising linear equations solved with the proposed new technique) that takes special care of reducing the effect of round-off errors by adapting some techniques used (for the implementation of symplectic IRK schemes with fixed-point iterations) in [1].

The plan of the paper is as follows: Section 2 summarizes some standard material about implicit Runge-Kutta methods and Newton-like iterations and fixes some notation. Section 3 presents our new technique to solve the simplified linear system

of Newton iterations for symplectic IRK schemes. Section 4 is devoted to describing our implementation of symplectic IRK methods with Newton-like iterations. Some numerical results are reported in Section 5. A few concluding remarks can be found in Section 6.

2 Implementation of implicit Runge-Kutta schemes with Newton-like iterations

2.1 Implicit Runge-Kutta schemes

We consider initial value problems of systems of ODEs of the form

$$\frac{d}{dt}y = f(t, y), \quad y(t_0) = y_0, \tag{2}$$

where $f : \mathbb{R}^{d+1} \to \mathbb{R}^d$ is a sufficiently smooth map and $y_0 \in \mathbb{R}^d$.

Given a time discretization $t_0 < t_1 < t_2 < \cdots$, the numerical approximations $y_n \approx y(t_n)$, $(n = 1, 2, \ldots)$ to the solution $y(t)$ of the initial value problem (2) is obtained by means of a one-step integrator as

$$y_{n+1} = \Phi(y_n, t_n, t_{n+1} - t_n), \quad n = 0, 1, 2, \ldots, \tag{3}$$

for a map $\Phi : \mathbb{R}^{d+2} \to \mathbb{R}^d$ determined in some way from $f : \mathbb{R}^{d+1} \to \mathbb{R}^d$.

In the case of a s-stage implicit Runge-Kutta method, the map Φ is determined in terms of the real coefficients a_{ij} $(1 \leqslant i, j \leqslant s)$ and b_i, c_i $(1 \leqslant i \leqslant s)$ as

$$\Phi(y, t, h) := y + h \sum_{i=1}^{s} b_i \, f(t + c_i h, Y_i), \tag{4}$$

where the *stage vectors* Y_i are implicitly defined as functions of $(y, t, h) \in \mathbb{R}^{d+2}$ by

$$Y_i = y + h \sum_{j=1}^{s} a_{ij} \, f(t + c_j h, Y_j), \quad i = 1, \ldots, s. \tag{5}$$

Typically,

$$c_i = \sum_{j=1}^{s} a_{ij}, \quad i = 1, \ldots, s.$$

The equations (5) can be solved for the stage vectors Y_i by means of some iterative procedure, starting, for instance, from $Y_i^{[0]} = y$, $i = 1, 2, \ldots, s$. (In the non-stiff case, it is usually more efficient initializing the stage vectors with some other procedure that uses the stage values of the previous steps [6]).

A very simple iterative procedure is fixed-point iteration. For stiff problems, fixed-point iteration is not appropriate, and Newton iteration may be used to compute the stage vectors Y_i from (5). For non-stiff problems, Newton iteration may still be an attractive option in some cases, in particular for very-high-precision computations (for quadruple precision or in arbitrary precision arithmetic calculations) if implemented with mixed-precision strategies [2] (which reduce the cost of the linear

algebra and the evaluation of the Jacobians, performed in lower-precision arithmetic than the evaluations of the right-hand side of the system of ODEs).

In any case, since at each Newton iteration s evaluations of the Jacobian matrix $\frac{\partial f}{\partial y}$ and a LU decomposition of a $sd \times sd$ matrix are required, some computationally cheaper variants are often used instead.

2.2 Newton-like iterations

Recall that a Newton iteration can be used to compute for $k = 1, 2, \ldots$ the approximations $Y_i^{[k]}$ of Y_i $(i = 1, \ldots, s)$ in (5) as follows:

$$(1) \quad r_i^{[k]} := -Y_i^{[k-1]} + y + h \sum_{j=1}^{s} a_{ij} f(t + c_j h, Y_j^{[k-1]}), \quad i = 1, \ldots, s, \tag{6}$$

(2) Solve $\Delta Y_i^{[k]}$ from

$$\Delta Y_i^{[k]} - h \sum_{j=1}^{s} a_{ij} J_j^{[k]} \Delta Y_j^{[k]} = r_i^{[k]} \quad i = 1, \ldots, s, \tag{7}$$

$$\text{where} \quad J_i^{[k]} = \frac{\partial f}{\partial y}(t + c_i h, Y_i^{[k-1]}) \quad \text{for} \quad i = 1, \ldots, s,$$

$$(3) \quad Y_i^{[k]} := Y_i^{[k-1]} + \Delta Y_i^{[k]}, \quad i = 1, \ldots, s. \tag{8}$$

Observe that, s evaluations of the Jacobian matrix $\frac{\partial f}{\partial y}$ and a LU decomposition of a $sd \times sd$ matrix are required (in addition to s evaluations of f) at each iteration. This is typically computationally too expensive, and some variants of the full Newton algorithm are implemented instead. Among others, the following alternatives are possible:

– Application of simplified Newton iterations. This consists on replacing the Jacobian matrices $J_i^{[k]}$ in (7) by $J_i^{[0]} = \frac{\partial f}{\partial y}(t + c_i h, Y_i^{[0]})$. In that case, LU decomposition is done only once and the linear system

$$\Delta Y_i^{[k]} - h \sum_{j=1}^{s} a_{ij} J_j^{[0]} \Delta Y_j^{[k]} = r_i^{[k]} \quad i = 1, \ldots, s, \tag{9}$$

has to be solved at each of the simplified Newton iterations. If the simple initialization $Y_i^{[0]} = y$ $(i = 1, \ldots, s)$ is considered (this is typically the case when solving stiff systems) and f does not depend on t, then $J_i^{[0]} = J := \frac{\partial f}{\partial y}(y)$ for each $i = 1, \ldots, s$, and the linear system (9) reduces to

$$(I_s \otimes I_d - h\, A \otimes J)\, \Delta Y^{[k]} = r^{[k]}, \tag{10}$$

where

$$Y^{[k]} = \begin{pmatrix} Y_1^{[k]} \\ \vdots \\ Y_s^{[k]} \end{pmatrix} \in \mathbb{R}^{sd}, \quad r^{[k]} = \begin{pmatrix} r_1^{[k]} \\ \vdots \\ r_s^{[k]} \end{pmatrix} \in \mathbb{R}^{sd}.$$

Even in the case where some initialization procedure other than $Y_i^{[0]} = y$ is used, in practice, the linear system (9) is often replaced by (10), where J is some common approximation of $\frac{\partial f}{\partial y}(t + c_i h, Y_i^{[0]})$, $i = 1, \ldots, s$. An appropriate choice [17] is $J := \frac{\partial f}{\partial y}(t + \bar{c}\, h, \bar{y})$, where $\bar{c} = \frac{1}{s}\sum_{i=1}^{s} c_i$ (which for methods that are symmetric in time gives $\bar{c} = \frac{1}{2}$) and $\bar{y} = \frac{1}{s}\sum_{i=1}^{s} Y_i^{[0]}$. Often, it will be sufficient to evaluate instead of $\frac{\partial f}{\partial y}$ a computationally cheaper approximation of it.

– Applying the original Newton iteration by solving the linear systems (7) with some iterative method [15] preconditioned by the inverse of the matrix

$$I_s \otimes I_d - h\, A \otimes J. \tag{11}$$

In practice, the linear systems (7) are only approximately solved with the iterative method. In such case, the resulting scheme is sometimes referred to as inexact Newton iteration [15]. Further variants of Newton-like iterations will be obtained if the Jacobian matrices are not updated at each iteration.

In any of the two alternatives above, one needs to repeatedly solve linear systems of the form

$$(I_s \otimes I_d - h\, A \otimes J)\, \Delta Y = r, \tag{12}$$

for given $r \in \mathbb{R}^{sd}$. From now on, we will refer to (12) as simplified linear system (of Newton-like iterations).

Of course, (12) may be solved by previously computing the LU decomposition of the full $sd \times sd$ matrix (11), but this may be done more efficiently.

A standard approach [3, 5, 13] consists on diagonalizing the matrix A as

$$\Lambda = S^{-1} A S = \mathrm{diag}(\lambda_1, \ldots, \lambda_s), \tag{13}$$

and computing the LU decomposition of the matrix

$$I_s \otimes I_d - h\, \Lambda \otimes J = (S^{-1} \otimes I_d)\, (I_s \otimes I_d - h\, A \otimes J)\, (S \otimes I_d). \tag{14}$$

In that case, one needs to compute the LU decomposition of a real (resp. complex) $d \times d$ matrix for each distinct real eigenvalue (for each distinct pair of complex eigenvalues) of A.

Alternatively, some authors [4, 11] propose solving (12) by an iterative procedure preconditioned by the inverse of

$$I_s \otimes I_d - h\, \tilde{A} \otimes J, \tag{15}$$

where $\tilde{A} \in \mathbb{R}^{s \times s}$ is a matrix chosen so that the LU decomposition of (15) can be more efficiently computed than that of (11).

In next section, we propose a new technique to efficiently solve simplified linear systems (12) of Newton iterations, provided that the IRK scheme is symplectic.

3 Efficient solution of simplified linear systems for symplectic IRK schemes

3.1 Symplectic IRK schemes

In what follows, we consider symplectic IRK schemes, that is [16], IRK schemes whose coefficients satisfy

$$b_i a_{ij} + b_j a_{ji} - b_i b_j = 0, \quad 1 \leqslant i, j \leqslant s. \tag{16}$$

Condition (16) guarantees that the discrete flow resulting from the application of the IRK scheme to an autonomous Hamiltonian system is symplectic, with important favorable consequences in the long-term behavior of the numerical solution [6]. Condition (16) also implies that, when applied to an ODE system with a quadratic invariant, then it is also a conserved quantity for the numerical solution provided by the IRK scheme.

Nevertheless, our interest in condition (16) is of a completely different nature: we will see that such a condition allows to solve efficiently linear systems of the form (12) for a given $d \times d$ real matrix J and a given $r \in \mathbb{R}^{sd}$. We will pay particular attention to symplectic IRK schemes that additionally satisfy (possibly after some reordering of the stage values Y_i) the time-symmetry condition [6]

$$\begin{aligned} b_{s+1-i} &= b_i, \quad c_{s+1-i} = 1 - c_i, \quad 1 \leqslant i \leqslant s, \\ b_j &= a_{s+1-i,s+1-j} + a_{i,j}, \quad 1 \leqslant i, j \leqslant s, \end{aligned} \tag{17}$$

In particular, the IRK schemes of collocation type with Gaussian nodes are both symplectic and symmetric in time.

3.2 Alternative symplecticity and time-symmetry characterizations

The map Φ determining the steps (3) of a IRK scheme can be alternatively written as

$$\Phi(y, t, h) := y + z,$$

where the stage vectors $Y_i \in \mathbb{R}^d$ and the increment $z \in \mathbb{R}^d$ are implicitly defined as functions of $(y, t, h) \in \mathbb{R}^{d+2}$ by

$$Y_i = y + \frac{z}{2} + h \sum_{j=1}^{s} \bar{a}_{ij} \, f(t + c_j h, Y_j), \quad i = 1, \ldots, s, \tag{18}$$

$$z = h \sum_{i=1}^{s} b_i \, f(t + c_i h, Y_i), \tag{19}$$

where

$$\bar{a}_{ij} = a_{ij} - \frac{b_j}{2}, \quad 1 \leqslant i, j \leqslant s. \tag{20}$$

Condition (16) may be equivalently characterized in terms of the matrix $\bar{A} = (\bar{a}_{ij})_{i,j=1}^{s}$ and the diagonal matrix B with diagonal entries $b_1, \ldots, b_s$. Indeed, (16) is equivalent to the requirement that the real $s \times s$ matrix $(B\bar{A})$ be skew-symmetric.

As for the time-symmetry condition (17), it reads

$$\begin{aligned} b_{s+1-i} &= b_i, \quad \bar{c}_{s+1-i} = -\bar{c}_i, \quad 1 \leqslant i \leqslant s, \\ \bar{a}_{s+1-i,s+1-j} &= -\bar{a}_{i,j}, \quad 1 \leqslant i, j \leqslant s, \end{aligned} \tag{21}$$

where $\bar{c}_i = c_i - \frac{1}{2}$ for $i = 1, \ldots, s$.

3.3 Efficient solution of the linear systems of the form (12)

From now on, we will only consider, without loss of generality[1], symplectic IRK schemes with invertible B. Since $B\bar{A}$ is skew-symmetric, so is $B^{\frac{1}{2}}\bar{A}B^{-\frac{1}{2}}$, which implies that $\bar{A}$ is diagonalizable with all eigenvalues in the imaginary axis. This is equivalent to the existence of a $s \times s$ invertible matrix Q such that

$$Q^{-1}\bar{A}Q = \begin{pmatrix} 0 & D \\ -D^T & 0 \end{pmatrix} \tag{22}$$

where D is a real diagonal matrix (with non-negative diagonal entries) of size $m \times (s-m)$, where $m = [(s+1)/2]$ (and $s - m = [s/2]$).

We will next show that (22) may be exploited to solve efficiently linear systems of the form (12). Consider the implicit equations (18)–(19). Application of simplified Newton iteration to such implicit equations leads to linear systems of the form

$$\begin{aligned} \left(I_s \otimes I_d - h\,\bar{A} \otimes J\right)\Delta Y - \tfrac{1}{2}(e_s \otimes I_d)\,\Delta z &= r, \\ \left(-h\,e_s^T B \otimes J\right)\Delta Y + \Delta z &= 0, \end{aligned} \tag{23}$$

where $e_s = (1, \ldots, 1)^T \in \mathbb{R}^s$. Clearly, if $(\Delta Y, \Delta z)$ is solution of (23), then ΔY is solution of (12).

By virtue of (22), the linear system (23) is equivalent, with the change of variables $\Delta Y = (Q \otimes I_d)W$ to

$$\begin{aligned} \begin{pmatrix} I_m \otimes I_d & -h\,D \otimes J \\ h\,D^T \otimes J & I_{s-m} \otimes I_d \end{pmatrix} W - \tfrac{1}{2}\,(Q^{-1}e_s \otimes I_d)\,\Delta z &= (Q^{-1} \otimes I_d)\,r, \\ -h\,(e_s^T B Q \otimes J)\,W + \Delta z &= 0, \end{aligned} \tag{24}$$

The blockwise sparsity pattern of the system (24) allows obtaining its LU decomposition by computing, in addition to several multiplications of matrices of size $d \times d$, the LU decompositions of $[s/2] + 1$ real matrices of size $d \times d$: the matrices

$$I_d + h^2\sigma_i^2 J^2, \quad i = 1, \ldots, [s/2],$$

[1] Any symplectic IRK method with $b_i = 0$ for some i is equivalent to a symplectic IRK scheme with fewer stages and $b_i \neq 0$ for all i [9]

where $\sigma_1, \ldots, \sigma_{[s/2]} \geqslant 0$ are the diagonal entries in D, and an additional $d \times d$ matrix obtained from the former. We will give more details in Section 3.4 in the case of time-symmetric symplectic IRK schemes.

It is worth remarking that such a technique for solving linear systems of the form (12) is not restricted to symplectic IRK schemes. It is enough that the corresponding matrix $\bar{A}$ be diagonalizable with all its eigenvalues in the imaginary axis. This seems to be the case of several families of (non-symplectic) time-symmetric IRK methods of collocation type, in particular, for the nodes of Lobatto quadrature formulas, or if the nodes are either the zeros or the extrema of Chebyshev polynomials of the first kind.

3.4 The case of time-symmetric symplectic IRK schemes

In the present section, in addition to the symplecticity conditions, that guarantee that the matrix $B^{\frac{1}{2}}\bar{A}B^{-\frac{1}{2}}$ is skew-symmetric, we assume that the symmetry conditions (21) hold.

Consider the $s \times s$ orthogonal matrix $P = (P_1 \ P_2)$ such that, for $x = (x_1, \ldots, x_s)^T \in \mathbb{R}^s$, $P_1^T x = (y_1 \cdots y_m)^T$, and $P_2^T x = (y_{m+1}, \cdots y_s)^T$, where

$$\begin{aligned} y_i &= \frac{\sqrt{2}}{2}(x_{s+1-i} + x_i), \qquad \text{for} \quad i = 1, \ldots, [s/2], \\ y_m &= x_m, \qquad \text{if} \quad s \text{ is odd}, \\ y_i &= \frac{\sqrt{2}}{2}(x_{s+1-i} - x_i), \qquad \text{for} \quad i = m+1, \ldots, s, \end{aligned}$$

with $m = [(s+1)/2]$.

The time-symmetry condition (21) implies that $P_i^T B^{\frac{1}{2}}\bar{A}B^{-\frac{1}{2}}P_i = 0$ for $i = 1, 2$, and since by symplecticity $B^{\frac{1}{2}}\bar{A}B^{-\frac{1}{2}}$ is a skew-symmetric matrix, we conclude that the matrix $\bar{A}$ is similar to

$$P^T B^{\frac{1}{2}}\bar{A}B^{-\frac{1}{2}}P = \begin{pmatrix} 0 & K \\ -K^T & 0 \end{pmatrix} \tag{25}$$

where $K = P_1^T B^{\frac{1}{2}}\bar{A}B^{-\frac{1}{2}}P_2$ (which is a real matrix of size $m \times (s-m) = [(s+1)/2] \times [s/2]$). Let $K = UDV^T$ be the singular value decomposition of K, (where $U \in \mathbb{R}^{m\times m}$ and $V \in \mathbb{R}^{(s-m)\times(s-m)}$ are orthonormal matrices, and $D \in \mathbb{R}^{m\times(s-m)}$ is a diagonal matrix with the singular values $\sigma_1, \ldots, \sigma_{s-m}$ of K as diagonal entries). We have that (22) holds with

$$Q = (Q_1 \ Q_2) = B^{-1/2}(P_1 \ P_2)\begin{pmatrix} U & 0 \\ 0 & V \end{pmatrix} = B^{-1/2}\begin{pmatrix} P_1 U & P_2 V \end{pmatrix},$$

and $Q^{-1} = Q^T B$. This implies that the linear system (23), with the change of variables

$$\Delta Y = (Q \otimes I_d)W = (Q_1 \otimes I_d)W' + (Q_2 \otimes I_d)W'' \tag{26}$$

is equivalent to (24). Due to the first symmetry conditions in (21), $e_s^T B P_2 = 0$, and hence $e_s^T B Q_2 = e_s^T B P_2 V = 0$, so that (24) reads

$$\begin{aligned} W' - h\,(D \otimes J)\,W'' - \tfrac{1}{2}\,(Q_1^T B e_s \otimes I_d)\,\Delta z &= (Q_1^T B \otimes I_d)\,r, \\ h\,(D^T \otimes J)\,W' + W'' &= (Q_2^T B \otimes I_d)\,r, \\ -h\,(e_s^T B Q_1 \otimes J)\,W' + \Delta z &= 0. \end{aligned}$$

By solving for W'' from the second equation of the linear system above,

$$W'' = -h\,(D^T \otimes J)\,W' + (Q_2^T B \otimes I_d)\,r. \tag{27}$$

and substitution in the remaining two equations, one obtains

$$\begin{aligned} (I_m \otimes I_d + h^2\,DD^T \otimes J^2)W' - \tfrac{1}{2}\,(Q_1^T B e_s \otimes I_d)\,\Delta z &= R, \\ -h\,(e_s^T B Q_1 \otimes J)\,W' + \Delta z &= 0. \end{aligned}$$

where $R = (Q_1^T B \otimes I_d)\,r + h\,(D Q_2^T B \otimes J)\,r \in \mathbb{R}^{md}$.

The linear system above can be rewritten in terms of

$$R = \begin{pmatrix} R_1 \\ \vdots \\ R_m \end{pmatrix}, \quad W' = \begin{pmatrix} W_1 \\ \vdots \\ W_m \end{pmatrix}$$

with $R_i, W_i \in \mathbb{R}^d$, $i = 1, \ldots, m$, as follows:

$$(I_d + h^2\sigma_i^2 J^2)\,W_i - \frac{\alpha_i}{2}\,\Delta z = R_i, \quad i = 1, \ldots, m, \tag{28}$$

$$-h\,J \sum_{i=1}^{m} \alpha_i\,W_i + \Delta z = 0. \tag{29}$$

where

$$\begin{pmatrix} \alpha_1 \\ \vdots \\ \alpha_m \end{pmatrix} = Q_1^T B e_s,$$

and $\sigma_1 \geqslant \cdots \geqslant \sigma_{[s/2]}$ are the singular values of K, and if s is odd (in which case $m = [(s+1)/2] = [s/2] + 1$), then $\sigma_m = 0$.

Thus, the unknown $\Delta z \in \mathbb{R}^d$ can be obtained by solving the linear system

$$M\,\Delta z = h\,J \sum_{i=1}^{m} \alpha_i (I_d + h^2\sigma_i^2 J^2)^{-1} R_i, \tag{30}$$

where

$$M = I_d + J\,\frac{h}{2} \sum_{i=1}^{m} \alpha_i^2 (I_d + h^2\sigma_i^2 J^2)^{-1} \in \mathbb{R}^{d\times d}. \tag{31}$$

The unknowns in $W' \in \mathbb{R}^{md}$ are then solved from (28), while $W'' \in \mathbb{R}^{(s-m)d}$ may be obtained from (27).

The required solution ΔY of the original linear system (12) may finally be obtained from (26).

3.5 Alternative reformulation of symplectic IRK schemes

If the coefficients b_i, a_{ij} determining a symplectic IRK are replaced by floating point numbers $\tilde{b}_i, \tilde{a}_{ij}$ that approximate them, then the resulting IRK scheme typically fails to satisfy the symplecticity conditions (16). This results [8] in a method that exhibits a linear drift in the value of quadratic invariants of the system and in the Hamiltonian function when applied to autonomous Hamiltonian systems.

Motivated by that, the map $\Phi : \mathbb{R}^{d+2} \to \mathbb{R}^d$ of the one-step integrator (3) corresponding to the IRK scheme, defined by (4)–(5), is rewritten in [1] in the following equivalent form:

$$\Phi(y, t, h) := y + \sum_{i=1}^{s} L_i,$$

where $L_i \in \mathbb{R}^d$, $i = 1, \ldots, s$ are implicitly defined as functions of $(t, y, h) \in \mathbb{R}^{d+2}$ by

$$L_i = h\, b_i\, f\left(t + c_i h,\, y + \sum_{j=1}^{s} \mu_{ij}\, L_j\right), \quad i = 1, \ldots, s, \tag{32}$$

where

$$\mu_{ij} = a_{ij}/b_j, \quad 1 \leqslant i, j \leqslant s.$$

The symplecticity condition (16) is equivalent to

$$\mu_{ij} + \mu_{ji} - 1 = 0, \quad 1 \leqslant i, j \leqslant s. \tag{33}$$

The main advantage of the proposed formulation over the standard one is that the absence of multiplications in the symplecticity condition (33) makes possible to find machine number approximations μ_{ij} of a_{ij}/b_j satisfying exactly the symplecticity condition (33).

With that alternative formulation, the Newton iteration reads as follows: initialize $L_i^{[0]} = 0$ $(i = 1, \ldots, s)$ and compute for $k = 1, 2, \ldots$

$$\begin{aligned}
&(1) \quad Y_i^{[k]} := y + \sum_{j=1}^{s} \mu_{ij}\, L_j^{[k-1]}, \quad i = 1, \ldots, s.\\
&\quad g_i^{[k]} := -L_i^{[k-1]} + h\, b_i\, f(t + c_i h, Y_i^{[k]}), \quad i = 1, \ldots, s,\\
&(2) \quad \text{Solve } \Delta L_i^{[k]} \text{ from}\\
&\quad \Delta L_i^{[k]} - h b_i J_i^{[k]} \sum_{j=1}^{s} \mu_{ij}\, \Delta L_j^{[k]} = g_i^{[k]}, \quad i = 1, \ldots, s,\\
&\text{where} \quad J_i^{[k]} = \frac{\partial f}{\partial y}(t + c_i h, Y_i^{[k]}) \quad \text{for} \quad i = 1, \ldots, s,\\
&(3) \quad L^{[k]} := L^{[k-1]} + \Delta L^{[k]}.
\end{aligned} \tag{34}$$

In the simplified version of the Newton iteration where the Jacobian matrices $J_i^{[k]}$ are replaced by a common approximation J (say, $J = \frac{\partial f}{\partial y}(t + h/2, y)$), the linear system in (34) is replaced by

$$\Delta L^{[k]} = \left(I_s \otimes I_d - h\, BAB^{-1} \otimes J \right)^{-1} \begin{pmatrix} g_1^{[k]} \\ \vdots \\ g_s^{[k]} \end{pmatrix}, \tag{35}$$

In that case, we need to repeatedly solve systems of the form

$$\left(I_s \otimes I_d - h\, BAB^{-1} \otimes J \right) \Delta L = g, \tag{36}$$

for prescribed $g \in \mathbb{R}^{sd}$. Repeated solution of linear systems of this form is also required if the linear system in (34) is iteratively solved as described in Section 4.2 below.

Of course, (36) can be solved by adapting the technique described in Sections 3.3 and 3.4 for the solutions of systems of the form (12). We next describe, for the time-symmetric case (i.e., when the symmetry condition (17) holds), the corresponding procedure (with the notation adopted in Section 3.4) to compute the solution ΔL of (36).

Proposed algorithm to solve the linear systems of the form (36)

1. Computations only depending on the coefficient matrix of the linear system:
   - Compute the inverses of the $\mathbb{R}^{d\times d}$ matrices
   $$I_d + h^2 \sigma_i^2\, J^2, \quad i = 1, \ldots, [s/2], \tag{37}$$
   - Compute the matrix $M \in \mathbb{R}^{d\times d}$ given in (31) (recall that $\sigma_m = 0$ when s is odd), and obtain its LU decomposition.
2. Computations depending on the right-hand side vector g:
   - Compute $R \in \mathbb{R}^{md}$ from
   $$R = (Q_1^T \otimes I_d)\, g + h\, (D Q_2^T \otimes J)\, g,$$
   - Compute
   $$d = h\, J \sum_{i=1}^{m} \alpha_i (I_d + h^2 \sigma_i^2 J^2)^{-1} R_i,$$
   - Compute $\Delta z \in \mathbb{R}^d$ as the solution of the linear system $M\, \Delta z = d$,
   - Next, compute $W_1, \ldots, W_m \in \mathbb{R}^d$ from
   $$(I_d + h^2 \sigma_i^2 J^2)\, W_i - \frac{\alpha_i}{2}\, J\, \Delta z = R_i, \quad i = 1, \ldots, m.$$
   - Follow by computing $W_{m+1}, \ldots, W_s \in \mathbb{R}^d$ from
   $$\begin{pmatrix} W_{m+1} \\ \vdots \\ W_s \end{pmatrix} = - \begin{pmatrix} h\, \sigma_1\, J\, W_1 \\ \vdots \\ h\, \sigma_{s-m}\, J\, W_{s-m} \end{pmatrix} + (Q_2^T \otimes I_d)\, g.$$

– And finally, $\Delta L \in \mathbb{R}^{sd}$ is obtained from

$$\Delta L = (BQ \otimes I_d) \begin{pmatrix} W_1 \\ \vdots \\ W_s \end{pmatrix}.$$

3.6 Potential advantages of the new solution technique for simplified linear systems

We will next compare the standard approach based on the decomposition (14) of the matrix $(I_s \otimes I_d - h\,A \otimes J)$ with the algorithm presented above.

In the standard approach, for even number of stages $s = 2m$, one has to solve m linear systems, each of them with a different complex coefficient matrix of size $d \times d$. For odd number of stages, $s = 2m + 1$, one has to solve one additional linear system with a real coefficient matrix of size $d \times d$. The main computational work is dominated by m complex LU decompositions (resp. m complex and one real LU decompositions). In our new approach, for $s = 2m$ or $s = 2m + 1$, the main computational work is dominated by the computation of m inverses and one LU decomposition of real matrices of dimension $d \times d$, and two additional real matrix-matrix multiplications of size $d \times d$ required to compute the matrix M in (31).

One obvious advantage of our approach is that complex arithmetic is completely avoided. For instance, not all linear algebra routines (in particular, sparse direct solvers) are available for complex arithmetic. Of course, the real block-diagonal equivalent of (14) could be used in the standard approach but it is at the cost of not exploiting the special structure of the 2×2 blocks in the diagonal (automatically exploited in the complex formulation by the definition of the complex arithmetic operations, see for instance Section 4.8 in [7]), which typically will imply increasing the computational cost.

Another advantage of our approach is that allows us exploiting the special structure of the Jacobian matrix of Hamiltonian systems to reduce the computational cost of the algorithm. Indeed, consider a general Hamiltonian systems of the form

$$\frac{d}{dt} y = K\,\nabla H(y),$$

where K is a constant skew-symmetric matrix of size $d \times d$, so that $J = K\,\nabla^2 H(y)$, where $\nabla^2 H(y)$ represents the Hessian matrix of the Hamiltonian function $H(y)$. One can check that

$$K^{-1}(I_d + h^2\sigma_j^2 J^2) = K^{-1} + h^2\sigma_j^2\,\nabla^2 H(y) K \nabla^2 H(y),$$

which is a skew-symmetric matrix. Its inverse can be efficiently computed with approximately half the arithmetic operations needed to invert a general matrix of the same size, for instance, by applying the block LDL^T factorization for skew-symmetric matrices described in [10].

The inversion of the relevant matrices can be simplified for Hamiltonian systems with Hamiltonian function of the form $H(q, p) = 1/2 p^T M^{-1} p + U(q)$, with $q, p \in \mathbb{R}^{d/2}$ and an invertible (typically diagonal) mass matrix M. In that

case, the inverses of the matrices $I_d + h^2\sigma_j^2 J^2$ (resp. the LU decompositions of the matrices $I_d + h\gamma_j J$) can be computed by inverting the real symmetric matrices $I_{d/2} + h^2\sigma_j M^{-1/2}\nabla^2 U(q)M^{-1/2}$ (resp. computing the LU decomposition of the complex non-symmetric matrices $I_{d/2} + h^2\gamma_j^2 M^{-1/2}\nabla^2 U(q)M^{-1/2}$).

We next compare the complexity of the two approaches for dense linear algebra. It is well known that for relatively large dimension d each real LU decomposition requires approximately $2d^3/3$ (real) arithmetic operations, and each complex LU decomposition approximately needs the equivalent of $8d^3/3$ real arithmetic operations. We thus have that for $s = 2m$ (resp. $s = 2m + 1$), approximately $8md^3/3$ (resp. $(8m + 2)d^3/3$) arithmetic operations are performed for the computations of the LU decompositions required at each integration step.

The inversion of each real matrix of size $d \times d$ approximately costs $2d^3$ arithmetic operations for general matrices. However, we have seen above that the inversion of skew-symmetric (and in some particular case, symmetric) matrices is required in the case of Hamiltonian systems. In that case, the number of arithmetic operations of each matrix inversion is approximately halved. We thus have that, when $s = 2m$ or $s = 2m + 1$, the computation of m inverses and one LU decomposition of real matrices of dimension $d \times d$, and two additional real matrix-matrix multiplications of size $d \times d$ require approximately $(m + 2/3 + 4)d^3 = (3m + 14)d^3/3$ arithmetic operations.

For sparse matrices, similar improvements may be obtained when sparse direct solvers are used to perform the matrix inversions required in our algorithm. As for the application of iterative solvers typically applied for systems of very large dimension, our approach could be particularly exploited for a more efficient preconditioning based on incomplete LU decompositions. Indeed, the squares σ_j^2 of the singular values σ_j, $j = 1, \ldots, m$ decrease approximately at a geometric rate of $1/10$ (unlike the modulus $|\gamma_i|$ of the eigenvalues of the RK matrix A, that are all similar in size). Motivated by that, it makes sense to apply an iterative solver to the linear system (28)–(29), preconditioned by the application of an incomplete LU decompositions to the system resulting from replacing in (28)–(29) most of the σ_j (the smallest ones) by zero.

4 Implementation of symplectic IRK schemes with Newton-like iterations

In this section, we present an algorithm (Algorithm 4 below) that implements symplectic IRK schemes by making use of the techniques in previous section. Special care is taken to try to reduce the effect of round-off errors by adapting some techniques used in [1] for the implementation of symplectic IRK schemes with fixed point iterations. Our algorithm is intended to be applied with the 64-bit IEEE double-precision floating-point arithmetic.

We originally implemented the standard simplified Newton iteration obtained by replacing the linear system in (34) by (35) (efficiently solved with our new

algorithm). However, we observed in our numerical experiments with the double pendulum problem that some linear drift of the energy error (due to accumulation of round-off errors) arises in some cases. We already observed similar linear drift of the energy error for our fixed-point implementation of symplectic IRK methods in [1], but expected this phenomenon to disappear in implementations based on Newton iterations. On the other hand, we observed that our simplified Newton implementation required more iterations per step for increasing values of the stiffness constant. These observations motivated us to modify the standard simplified Newton iteration to produce Algorithm 4 below, which requires, at each step, s additional evaluations of the Jacobian matrix. (It is worth noting that such Jacobian evaluations can be computed in parallel provided that s processors or cores are available.) We have observed in our numerical experiments with the stiff pendulum problem, that the additional computational cost of Algorithm 4 does improve efficiency (the number of iterations is substantially reduced) and robustness (unlike simplified Newton iteration, the number of iterations of Algorithm 4 does not increase for increasing stiffness constant), and in addition, the linear drift of the energy error mentioned above completely disappears.

4.1 Auxiliary techniques

In this subsection, we summarize some techniques associated to the use of finite precision arithmetic that we applied in the fixed-point iteration implementation of symplectic IRK schemes proposed in [1], and will be used in the algorithm proposed in Sections 4.3.

Let $\mathbb{F} \subset \mathbb{R}$ be the set of machine numbers of the 64-bit IEEE double-precision floating-point arithmetic. We consider the map $\mathrm{fl} : \mathbb{R} \longrightarrow \mathbb{F}$ that sends each real number x to a nearest machine number $\mathrm{fl}(x) \in \mathbb{F}$.

4.1.1 Kahan's compensated summation

The application of any one-step integrator of the form (3) requires computing sums of the form

$$y_{n+1} = y_n + x_n, \quad n = 0, 1, 2, \ldots, \tag{38}$$

For an actual implementation that only uses a floating-point arithmetic with machine numbers in $\mathbb{F}$, special care must be taken with the additions (38). The naive recursive algorithm $\hat{y}_{n+1} := \mathrm{fl}(\hat{y}_n + \mathrm{fl}(x_n))$, $(n = 0, 1, 2, 3 \ldots)$, typically suffers, for large n, a significant loss of precision due to round-off errors. It is well known that such a round-off error accumulation can be greatly reduced with the use of Kahan's compensated summation algorithm [12] (see also [10, 14]).

Given $y_0 \in \mathbb{R}^d$ and a sequence $\{x_0, x_1, \ldots, x_n, \ldots\} \subset \mathbb{F}^d$ of machine numbers, Kahan's algorithm is aimed to compute the sums $y_n = y_0 + \sum_{\ell=0}^{n-1} x_\ell$, $(n \geqslant 1,)$ using

a prescribed floating point arithmetic, more precisely than with the naive recursive algorithm. The actual algorithm reads as follows:

Algorithm 1: Compensated summation algorithm

$\tilde{y}_0 = \mathrm{fl}(y_0);\ e_0 = \mathrm{fl}(y_0 - \tilde{y}_0);$

for $l \leftarrow 0$ **to** n **do**

$X_l = \mathrm{fl}(x_l + e_l);$
$\tilde{y}_{l+1} = \mathrm{fl}(\tilde{y}_l + X_l);$
$\hat{X}_l = \mathrm{fl}(\tilde{y}_{l+1} - \tilde{y}_l);$
$e_{l+1} = \mathrm{fl}(X_l - \hat{X}_l);$

end

The sums $\tilde{y}_l + e_l \in \mathbb{R}^d$ are more precise approximations of the exact sums y_l than $\tilde{y}_l \in \mathbb{F}$. Algorithm 1 can be interpreted as a family of maps parametrized by n and d,

$$S_{n,d} : \mathbb{F}^{(n+3)d} \to \mathbb{F}^{2d},$$

that given the arguments $\tilde{y}_0, e_0, x_0, x_1, \ldots, x_n \in \mathbb{F}^d$, returns $\tilde{y}_{n+1}, e_{n+1} \in \mathbb{F}^d$ such that $\tilde{y}_{n+1} + e_{n+1} \approx (\tilde{y}_0 + e_0) + x_0 + x_1 + \cdots + x_n$ with some small error.

4.1.2 Stopping criterion for iterative processes

Given a smooth map $F : \mathbb{R}^D \to \mathbb{R}^D$ and $Z^{[0]} = (Z_1^{[0]}, \ldots, Z_D^{[0]}) \in \mathbb{R}^D$ assume that the iteration

$$Z^{[k]} = F(Z^{[k-1]}), \quad \text{for} \quad k = 1, 2, \ldots \tag{39}$$

produces a sequence $\{Z^{[0]}, Z^{[1]}, Z^{[2]}, \ldots\} \subset \mathbb{R}^D$ that converges to a fixed point $Z^{[\infty]}$ of F.

Assume now that instead of the original map F, we have a computational substitute

$$\widetilde{F} : \mathbb{F}^D \to \mathbb{F}^D. \tag{40}$$

Ideally, for each $Z \in \mathbb{F}^D$, $\widetilde{F}(Z) := \mathrm{fl}(F(Z))$. In practice, the intermediate computations to evaluate $\widetilde{F}$ are typically made using the floating-point arithmetic corresponding to $\mathbb{F}$, which will result in some additional error caused by the accumulated effect of several round-off errors.

The resulting sequence $\widetilde{Z}^{[k]} = \widetilde{F}(\widetilde{Z}^{[k-1]})$, $k = 1, 2, \ldots$ (started with $\widetilde{Z}^{[0]} = \mathrm{fl}(Z^{[0]})$) will either converge to a fixed point of $\widetilde{F}$ in a finite number K of iterations or will fail to converge. In the former case, the fixed point $\widetilde{Z}^{[K]} \in \mathbb{F}^D$ of $\widetilde{F}$ may be expected to be a good approximation of the fixed point $Z^{[\infty]} \in \mathbb{R}^D$ of F. In the later case, one would expect that there exists an index K such that the approximations

$\widetilde{Z}^{[k]} \approx Z^{[\infty]}$ improves for increasing k up to $k = K$, but the quality of the approximations $\widetilde{Z}^{[k]}$ does not improve for $k > K$. It then make sense to apply an algorithm of the form

Algorithm 2: Stopping criterion

$k = 0$;
$\widetilde{Z}^{[0]} = \mathrm{fl}(Z^{[0]})$;
while (ContFcn$(\widetilde{Z}^{[0]}, \cdots, \widetilde{Z}^{[k]})$) **do**
  $k = k + 1$;
  $\widetilde{Z}^{[k]} = \widetilde{F}(\widetilde{Z}^{[k-1]})$;
end

where ContFcn$(\widetilde{Z}^{[0]}, \cdots, \widetilde{Z}^{[k]})$ gives either true if it is judged appropriate to continue iterating, and false otherwise. In [1], we propose defining this function so that ContFcn$(\widetilde{Z}^{[0]}, \cdots, \widetilde{Z}^{[k]})$ returns,

$$\begin{cases} \text{false} \;\rightarrow\; \text{if} \quad (\widetilde{Z}^{[k]} = \widetilde{Z}^{[k-1]}) \text{ or} \\ \qquad\qquad k > 1 \text{ and } k = K - 1 \text{ and } k = K \text{ and } \forall j \in \{1, \ldots, D\}, \\ \qquad\qquad \min\left(\{|\widetilde{Z}_j^{[1]} - \widetilde{Z}_j^{[0]}|, \cdots, |\widetilde{Z}_j^{[k-1]} - \widetilde{Z}_j^{[k-2]}|\} \,/\{0\}\right) \leqslant |\widetilde{Z}_j^{[k]} - \widetilde{Z}_j^{[k-1]}| \\ \text{true} \;\rightarrow\; \text{otherwise.} \end{cases} \tag{41}$$

The output $\widetilde{Z}^{[K]}$ of the algorithm will be a fixed point of $\widetilde{F}$ when it stops because $\widetilde{Z}^{[K]} = \widetilde{Z}^{[K-1]}$, and in any case, it is not expected that $\widetilde{Z}^{[k]}$ for $k > K$ be a better approximation of the fixed point $Z^{[\infty]} \in \mathbb{R}^D$ of F than $\widetilde{Z}^{[K]}$.

4.2 An inexact Newton iteration

In our implementation of symplectic IRK schemes to be described in Section 4.3, we consider a modified version of the Newton iteration (34). First of all, we will consider fixed approximations J_i of the Jacobian matrices $\frac{\partial f}{\partial y}(t + c_i\, h, Y_i)$ $(i = 1, \ldots, s)$ for each Newton-like iteration, so that in each iteration, we have to solve the linear system

$$\Delta L_i^{[k]} - h b_i\, J_i \sum_{j=1}^{s} \mu_{ij}\, \Delta L_j^{[k]} = g_i^{[k]}, \quad i = 1, \ldots, s, \tag{42}$$

where

$$g_i^{[k]} = -L_i^{[k-1]} + h\, b_i\, f\left(t + c_i h, y + \sum_{j=1}^{s} \mu_{ij}\, L_j^{[k-1]}\right), \quad i = 1, \ldots, s, \tag{43}$$

and

$$\Delta L^{[k]} = \begin{pmatrix} \Delta L_1^{[k]} \\ \vdots \\ \Delta L_s^{[k]} \end{pmatrix} \in \mathbb{R}^{sd}, \quad g^{[k]} = \begin{pmatrix} g_1^{[k]} \\ \vdots \\ g_s^{[k]} \end{pmatrix} \in \mathbb{R}^{sd},$$

In addition, instead of exactly solving for $\Delta L_i^{[k]}$ $(i = 1, \ldots, s)$, we iteratively compute a sequence $\Delta L_i^{[k,0]}, \Delta L_i^{[k,1]}, \Delta L_i^{[k,2]}, \ldots$ of approximation of the solution $\Delta L^{[k]} \in \mathbb{R}^{sd}$ of (42) as shown below (Algorithm 3).

Algorithm 3: Inner iteration

$\Delta L^{[k,0]} = (I_s \otimes I_d - h\, BAB^{-1} \otimes J)^{-1}\, g^{[k]}$;

while (ContFcn($\mathrm{fl}_{32}(\Delta L^{[k,0]}), \cdots, \mathrm{fl}_{32}(\Delta L^{[k,\ell]})$)) **do**

$l = l + 1$;

$G_i^{[k,\ell]} = g_i^{[k]} - \Delta L_i^{[k,\ell-1]} + hb_i\, J_i \sum_{j=1}^{s} \mu_{ij}\, \Delta L_j^{[k,\ell-1]}, \quad i = 1, \ldots, s$;

$\Delta L^{[k,l]} = \Delta L^{[k,l-1]} + (I_s \otimes I_d - h\, BAB^{-1} \otimes J)^{-1}\, G^{[k,l]}$;

end

Here, $\mathrm{fl}_{32}(x)$ represents the 32-bit IEEE single-precision machine number that is closest to $x \in \mathbb{R}$, and we let fl_{32} act componentwise on vectors.

In the algorithm we propose in Section 4.3, the Jacobian matrices J_i $(i = 1, \ldots, s)$ in (42) will be evaluated (only once at each step) in approximations of the stage values Y_i that are not very accurate. This implies that it does not make sense to apply the iteration (Algorithm 3) until an accurate double-precision approximation $\Delta L^{[k,\ell]}$ of the solution $\Delta L^{[k]}$ of (42) is obtained. Motivated by that, we will stop the iteration when ContFcn($\mathrm{fl}_{32}(\Delta L^{[k,0]}), \cdots, \mathrm{fl}_{32}(\Delta L^{[k,\ell]})$) returns false (i.e., typically, when $\mathrm{fl}_{32}(\Delta L^{[k,\ell]}) = \mathrm{fl}_{32}(\Delta L^{[k,\ell-1]})$.

4.3 Algorithm for one step of the IRK scheme

In our implementation, the numerical solution $y_n \approx y(t_n) \in \mathbb{R}^d$, $n = 1, 2, \ldots$, is obtained as the sum $\tilde{y}_n + e_n$ of two vectors in $\mathbb{F}^d$. In particular, the initial value $y_0 \in \mathbb{R}^d$ is (approximately) represented as $\tilde{y}_0 + e_0$, where $\tilde{y}_0 = \mathrm{fl}(y_0)$ and $e_0 = \mathrm{fl}(y_n - \tilde{y}_n)$. Instead of (3), we actually have

$$(\tilde{y}_{n+1}, e_{n+1}) = \tilde{\Phi}(\tilde{y}_n, e_n, t_n, t_{n+1} - t_n),$$

where $\tilde{\Phi} : \mathbb{F}^{2d+2} \to \mathbb{F}^{2d}$.

Our proposed implementation of one step

$$(\tilde{y}^*, e^*) = \tilde{\Phi}(\tilde{y}, e, t, h)$$

of the IRK scheme is performed in five substeps:

1. Starting from $L^{[0]} = 0 \in \mathbb{R}^{sd}$, we apply several simplified Newton iterations

(i.e., the simplified version of Newton iterations (34) where the linear system is replaced by (35)) to compute

$$L^{[1]} = L^{[0]} + \Delta L^{[1]}, \quad L^{[2]} = L^{[1]} + \Delta L^{[2]}, \ldots$$

until $\mathrm{fl}_{32}(L^{[k]}) = \mathrm{fl}_{32}(L^{[k-1]})$ (or rather, by using the notation introduced in paragraph 4.1.2, until $\mathrm{ContFcn}(\mathrm{fl}_{32}(L^{[0]}), \cdots, \mathrm{fl}_{32}(L^{[k]}))$ returns false).

2. Use $L^{[k]}$ to compute the Jacobian matrices

$$J_i = \frac{\partial f}{\partial y}\left(t + c_i\, h,\ \tilde{y} + \sum_{j=1}^{s} \mu_{ij}\, L_j^{[k]}\right), \quad i = 1, \ldots, s,$$

3. Then consider the increment $\Delta L^{[k]} \in \mathbb{F}^{sd}$ obtained in first substep as an approximation $\Delta L^{[k,0]}$ of the exact solution $\Delta L^{[k]}$ of the linear system in (34), and apply the inner iterations (Algorithm 3) to obtain as output an approximation $\Delta L^{[k,\ell]}$ (accurate at least at single-precision level).
4. Follow by updating $L^{[k]} = L^{[k-1]} + \Delta L^{[k,\ell]}$, and $k = k+1$, and applying a final inexact Newton iteration with the Jacobian matrices J_i computed in the second substep. More precisely, compute an approximation $\Delta L^{[k,\ell]}$ (again accurate at least at single-precision level) of the solution $\Delta L^{[k]}$ of (42)–(43) by applying Algorithm 3.
5. Finally, the increment $\tilde{\Phi}(\tilde{y}, e, t, h)$, defined as the sum $(\tilde{y}+e)+\sum_{i=1}^{s}(L_i^{[k-1]} + \Delta L_i^{[k,\ell]})$ is accurately obtained as the (unevaluated) sum of the double-precision vectors $\tilde{y}^*, e^* \in \mathbb{F}^d$ with the help of Kahan's compensated summation algorithm (summarized in paragraph 4.1.1) as follows: First, perform the sum $\delta := e + \sum_{i=1}^{s} \Delta L_i^{[k,\ell]})$ of the vectors with relatively smaller size in the double-precision floating-point arithmetic, and then compute $(\tilde{y}^*, e^*) = S_{s,d}(\tilde{y}, \delta, L_1^{[k-1]}, \ldots, L_s^{[k-1]})$.

Some remarks about our actual implementation are in order:

- All the linear system with coefficient matrix $(I_s \otimes I_d - h\, BAB^{-1} \otimes J)$ are solved by means of the algorithm at the end of Section 3.
- The coefficients μ_{ij} are machine numbers in $\mathbb{F}$ (i.e., in the target precision floating-point system) satisfying exactly the symplecticity condition (33) and the symmetry conditions $\mu_{j,i} = \mu_{s+1-i,s+1-j}$.
- The remainders (43) ($i = 1, \ldots, s$, $k \geqslant 1$) should in principle be computed with $y \in \mathbb{R}^d$ replaced by $\tilde{y} + e$ ($\tilde{y}, e \in \mathbb{F}^d$). However, the effect of ignoring the extra digits of y that may be contained in e is expected to be so small that it should be enough to take it into account only in the final inexact Newton iteration (substep 4 above). That is, it should be enough considering (43) with $y \in \mathbb{R}^d$ replaced by $\tilde{y} \in \mathbb{F}^d$ in all the Newton-like iterations with the exception of the final one. And in the final inexact Newton iteration, rather than computing (43) with $y \in \mathbb{R}^d$

replaced by $\tilde{y}+e$, we make use of the Jacobian matrices J_i to obtain the following approximation:

$$h\, b_i\, f\left(t + c_i h,\ \tilde{y} + e + \sum_{j=1}^{s} \mu_{ij}\, L_j^{[k-1]}\right) - L_i^{[k-1]} \approx \left(h\, b_i\, f_i^{[k]} - L_i^{[k-1]}\right) + h\, b_i\, J_i\, e,$$

where $f_i^{[k]} = f\left(t + c_i h,\ \tilde{y} + \sum_{j=1}^{s} \mu_{ij}\, L_j^{[k-1]}\right)$.

– If the FMA (fused-multiply-add) instruction is available, it should be used to compute $h\, b_i\, f_i^{[k]} - L_i^{[k-1]}$ (with precomputed coefficients $hb_i \in \mathbb{F}$ satisfying the symmetry conditions $hb_{s+1-i} = hb_i$).

Our final implementation is summarized in Algorithm 4.

5 Numerical experiments

We next report some numerical experiments with our implementation of the 6-stage Gauss collocation method of order 12 based on Newton-like iterations (Algorithm 4) with 64-bit IEEE double-precision floating-point arithmetic. We are particularly interested in assessing the quality of our implementation with respect to the propagation of round-off errors.

5.1 The stiff double pendulum problem

We consider the planar stiff double pendulum problem: a double bob pendulum with masses m_1 and m_2 attached by rigid massless rods of lengths l_1 and l_2 and spring of elastic constant k between both rods (the rods are aligned at equilibrium). For $k = 0$, the problem is non-stiff, and the system's stiffness arises through increasing the value of k.

The configuration of the pendulum is described by two angles $q = (\phi, \theta)$: while ϕ is the angle (with respect to the vertical direction) of the first bob, the second bob's angle is defined by $\psi = \phi + \theta$. We denote the corresponding momenta as $p = (p_\phi, p_\theta)$.

Its Hamiltonian $H(q, p)$ is

$$-\frac{{l_1}^2\,(m_1+m_2)\,{p_\theta}^2 + {l_2}^2\,m_2\,(p_\theta - p_\phi)^2 + 2\,l_1\,l_2\,m_2\,p_\theta\,(p_\theta - p_\phi)\,\cos(\theta)}{{l_1}^2\,{l_2}^2\,m_2\,(-2\,m_1 - m_2 + m_2\,\cos(2\theta))}$$
$$-\,g\,\cos(\phi)\,(l_1\,(m_1+m_2) + l_2\,m_2\,\cos(\theta)) + g\,l_2\,m_2\,\sin(\theta)\sin(\phi) + \frac{k}{2}\,\theta^2. \quad (44)$$

We consider the following fixed parameter values

$$g = 9.8\,\frac{m}{s^2}\ ,\ \ l_1 = 1.0\,m\ ,\ l_2 = 1.0\,m\ ,\ m_1 = 1.0\,kg\ ,\ m_2 = 1.0\,kg,$$

Algorithm 4: Implementation of one step of the IRK scheme

$L^{[0]} = 0$;
$J = \frac{\partial f}{\partial y}(t + h/2, \tilde{y})$;
$M = I_d + J \, \frac{h}{2} \, \sum_{i=1}^{m} \alpha_i^2 (I_d + h^2 \sigma_i^2 J^2)^{-1}$;
Compute the LU decomposition of M;
/********** 1st substep **********/;
$k = 0$;
while ContFcn($\mathrm{fl}_{32}(L^{[0]}), \ldots, \mathrm{fl}_{32}(L^{[k]})$) **do**
  $k = k + 1$;
  $Y_i^{[k]} = \tilde{y} + \sum_{j=1}^{s} \mu_{ij} \, L_j^{[k-1]}, \ i = 1, \ldots, s$;
  $f_i^{[k]} = f\left(t + c_i h, Y_i^{[k]}\right), \ i = 1, \ldots, s$;
  $g_i^{[k]} = h \, b_i \, f_i^{[k]} - L_i^{[k-1]}, \ i = 1, \ldots, s$;
  $\Delta L^{[k]} = \left(I_s \otimes I_d - h \, BAB^{-1} \otimes J\right)^{-1} g^{[k]}$;
  $L^{[k]} = L^{[k-1]} + \Delta L^{[k]}$;
end
/********** 2nd substep **********/;
$J_i = \frac{\partial f}{\partial y}\left(t + c_i h, \ \tilde{y} + \sum_{j=1}^{s} \mu_{ij} \, L_j^{[k]}\right), \ i = 1, \ldots, s$;
/********** 3rd substep **********/;
$\ell = 0$;
$\Delta L^{[k,0]} = \Delta L^{[k]}$;
while ContFcn($\mathrm{fl}_{32}(\Delta L^{[k,0]}), \ldots, \mathrm{fl}_{32}(\Delta L^{[k,\ell]})$) **do**
  $\ell = \ell + 1$;
  $G_i^{[k,\ell]} = g_i^{[k]} - \Delta L_i^{[k,\ell-1]} + h b_i J_i \sum_{j=1}^{s} \mu_{ij} \, \Delta L_j^{[k,\ell-1]}, \ i = 1, \ldots, s$;
  $\Delta L^{[k,\ell]} = \Delta L^{[k,\ell-1]} + \left(I_s \otimes I_d - h \, BAB^{-1} \otimes J\right)^{-1} G^{[k,\ell]}$;
end
$L^{[k]} = L^{[k-1]} + \Delta L^{[k,\ell]}$;
/********** 4th substep **********/;
$k = k + 1$;
$Y_i^{[k]} = \tilde{y} + \sum_{j=1}^{s} \mu_{ij} \, L_j^{[k-1]}, \ i = 1, \ldots, s$;
$f_i^{[k]} = f\left(t + c_i h, Y_i^{[k]}\right), \ i = 1, \ldots, s$;
$g_i^{[k]} = \left(h \, b_i \, f_i^{[k]} - L_i^{[k-1]}\right) + h \, b_i \, J_i \, e, \ i = 1, \ldots, s$;
$\ell = 0$;
$\Delta L^{[k,0]} = \left(I_s \otimes I_d - h \, BAB^{-1} \otimes J\right)^{-1} g^{[k]}$;
while ContFcn($\mathrm{fl}_{32}(\Delta L^{[k,0]}), \ldots, \mathrm{fl}_{32}(\Delta L^{[k,\ell]})$) **do**
  $\ell = \ell + 1$;
  $G_i^{[k,\ell]} = g_i^{[k]} - \Delta L_i^{[k,\ell-1]} + h b_i J_i \sum_{j=1}^{s} \mu_{ij} \, \Delta L_j^{[k,\ell-1]}, \ i = 1, \ldots, s$;
  $\Delta L^{[k,\ell]} = \Delta L^{[k,\ell-1]} + \left(I_s \otimes I_d - h \, BAB^{-1} \otimes J\right)^{-1} G^{[k,\ell]}$;
end
/********** 5th substep **********/;
$\delta = e + \sum_{i=1}^{s} \Delta L_i^{[k,\ell]}$;
$(\tilde{y}^*, e^*) = S_{s,d}(\tilde{y}, \delta, L_1^{[k-1]}, \ldots, L_s^{[k-1]})$;

and we choose various values for the elastic constant k to study different levels of stiffness in the double pendulum. The largest eigenvalues of the Jacobian matrix of that system are $\sim \pm i\sqrt{k}$. The initial values are chosen as follows: For $k = 0$, we choose the initial values considered in [1], which gives rise to a non-chaotic trajectory, $q(0) = (1.1, -1.1)$ and $p(0) = (2.7746, 2.7746)$. The initial values for $k \neq 0$ are chosen as

$$q(0) = \left(1.1, \frac{-1.1}{\sqrt{1+100k}}\right), \quad p(0) = (2.7746, 2.7746)$$

so that the total energy of the system is bounded as $k \to \infty$.

All the integrations are performed with step-size $h = 2^{-7}$, which is small enough for round-off errors to dominate over truncation errors in the case $k = 0$. The truncation errors dominate over the round-off errors for large enough stiffness constant $k > 0$, even before than becoming truly stiff. We integrate over $T_{\text{end}} = 2^{12}$ s and sample the numerical results every $m = 2^{10}$ steps.

5.2 Round-off error propagation

First, we check the performance of round-off error propagation of our new implementation based on Newton-like iterations. In [1], we proposed an implementation based on fixed-point iterations for non-stiff problems that takes special care of reducing the propagation of round-off errors. We will compare the round-off error of both implementations of the 6-stage Gauss collocation method.

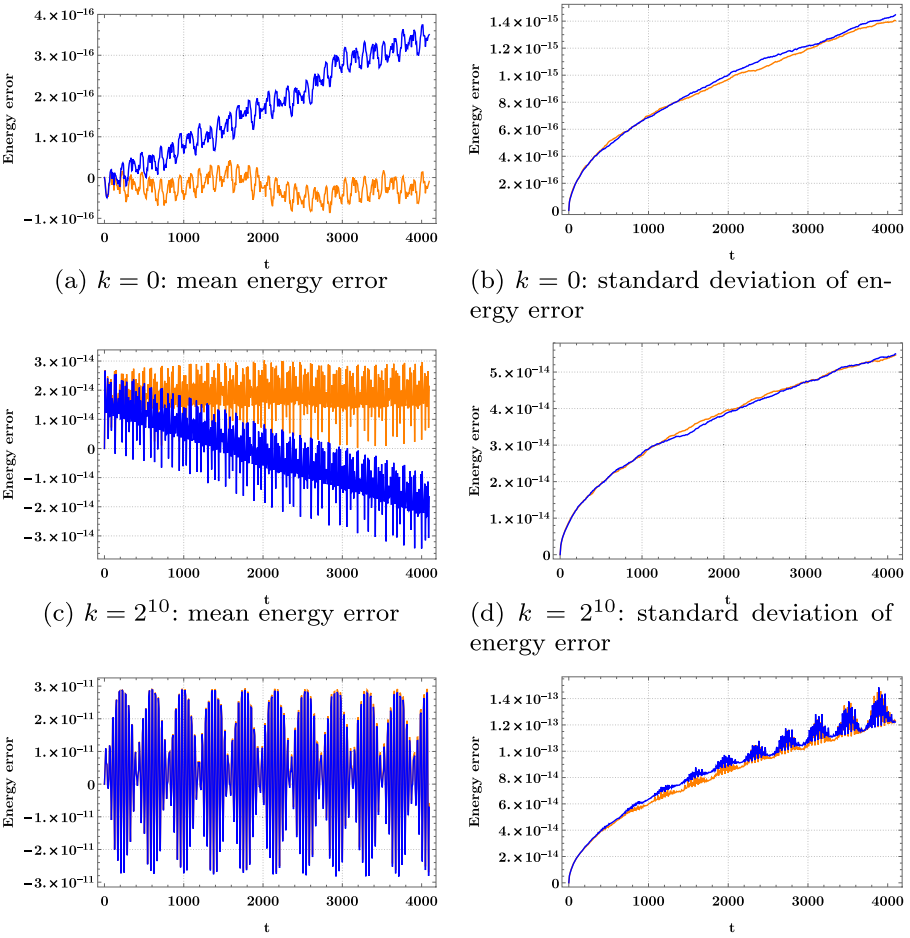
We have studied in detail the errors in energy of the double pendulum problem for three values of the stiffness constant: $k = 0$, where the round-off errors dominate over truncation errors, $k = 2^{10}$, where both kinds of errors are similar in size, and $k = 2^{12}$, where truncation errors dominate over round-off errors. In order to make a more robust comparison of the numerical errors due to round-off errors, we adopt (as in [8]) an statistical approach. We have considered for each of the three initial value problems, $P = 1000$ perturbed initial values by randomly perturbing each component of the initial values with a relative error of size $\mathcal{O}(10^{-6})$.

The numerical tests in Fig. 1 seem to confirm the good performance of round-off error propagation of our new implementation. On one hand, one can observe that, as in [1], the fixed-point implementation exhibits a small linear drift of the mean energy error for $k = 0$ and $k = 2^{10}$, while in the Newton implementation, this energy drift does not appear at all. On the other hand, the standard deviation of the energy errors are of similar size and grow proportionally to $t^{\frac{1}{2}}$ in both implementations.

5.3 Fixed-point versus Newton iteration

We summarize in Table 1 the main results of numerical integrations for both implementations: the fixed-point iteration and Newton-like iteration for four different values of k.

We have compared their efficiency by sequential execution of each iteration method and reported the cpu-time of each numerical integration. In addition, we have reported the number of iterations per step (It. per step) in both implementations and



(a) $k = 0$: mean energy error

(b) $k = 0$: standard deviation of energy error

(c) $k = 2^{10}$: mean energy error

(d) $k = 2^{10}$: standard deviation of energy error

(e) $k = 2^{12}$: mean energy error

(f) $k = 2^{12}$: standard deviation of energy error

Fig. 1 Evolution of mean (*left*) and standard deviation (*right*) of relative errors in energy for fixed-point implementation (*blue*), and Newton implementation (*orange*). Increasing values of stiffness constant: $k = 0$ in (**a**, **b**), $k = 2^{10}$ in (**c**, **d**), and $k = 2^{12}$ in (**e**, **f**)

the number of linear systems solved in the Newton implementation. To check the precision of the numerical solution, we have reported the maximum relative energy error,

$$\max \left| \frac{E(t_n) - E(t_0)}{E(t_0)} \right|, \quad t_n = t_0 + nh, \quad n = 1, 2, \ldots$$

We can see that, as expected, the fixed-point implementation is more efficient than Newton implementation for low values of k. But as we increase the stiffness of the double pendulum, the number of iteration needed at each step in the

Table 1 Summary of numerical integrations with fixed-point iteration- and Newton iteration-based implementation for the following elastic constant values: $k = 0$, $k = 2^6$, $k = 2^{12}$, and $k = 2^{16}$

k	0	2^6	2^{12}	2^{16}
E_0	-14.39	-5.75	-5.64	-5.64
Fixed-points it.				
Cpu-time (s)	10	12	19	51
It. per step	8.58	11.1	22.	64.2
Max energy error	2.96×10^{-15}	1.81×10^{-14}	2.94×10^{-11}	6.33×10^{-5}
Newton it.				
Cpu-time (s)	18	20	19	18
It. per step	5.09	5.53	5.58	5.01
L. solves per step	11.37	12.92	12.72	11.04
Max energy error	1.6×10^{-15}	1.74×10^{-14}	2.94×10^{-11}	6.33×10^{-5}

E_0 indicates the initial energy of the system. We show the cpu-time, the number of iteration per step (It. per step), the number of linear system solving operations (L. solves per step), and maximum energy error for each numerical computation

fixed-point implementation grows up notably, while in the Newton implementation, the number of iterations even becomes slightly lower for higher values of k. Hence, the Newton implementation eventually becomes more efficient as the stiffness increases. For k values higher than $k = 2^{18}$, the fixed-point iteration fails to converge, while the Newton implementations succeeds while keeping approximately the same number of iterations per step (cpu-time, 17 s; iterations per step, 4.95; linear solves per step, 10.94). It must be pointed out that the example considered here is of very low dimension. For systems of higher dimension with dense Jacobian matrix, the complexity of each Newton-like iteration may be largely dominated by the linear algebra, so the fixed-point implementation may still be more efficient than the Newton implementation for larger values of k. Of course, eventually, fixed-point iteration will fail to converge for large enough values of the stiffness constant k.

6 Conclusions

Our main contribution is a technique to solve efficiently the simplified linear systems of symplectic IRK schemes. This technique can be adapted for some time-symmetric non-symplectic schemes as well. Such technique could also be exploited for the numerical solution of boundary value problems with collocation methods with Gaussian quadrature nodes.

In addition, an efficient and robust algorithm for implementing symplectic IRK methods with reduced round-off error propagation is provided. A C code with our

implementation for s-stage Gauss collocation methods of order $2s$ in 64-bit IEEE double-precision floating-point arithmetic can be downloaded from IRK-Newton Github software repository, whose URL is as follows: https://github.com/mikelehu/IRK-Newton.

Acknowledgements M. Antoñana, J. Makazaga, and A. Murua have received funding from the Project of the Spanish Ministry of Economy and Competitiveness with reference MTM2016-76329-R (AEI/FEDER, EU), from the project MTM2013-46553-C3-2-P from Spanish Ministry of Economy and Trade, and as part of the Consolidated Research Group IT649-13 by the Basque Government.

References

1. Antoñana, M., Makazaga, J., Murua, A.: Reducing and monitoring round-off error propagation for symplectic implicit Runge-Kutta schemes. Numerical Algorithms 1–20. doi:10.1007/s11075-017-0287-z (2017)
2. Baboulin, M., Buttari, A., Dongarra, J., Kurzak, J., Langou, J., Langou, J., Luszczek, P., Tomov, S.: Accelerating scientific computations with mixed precision algorithms. Comput. Phys. Commun. **180**(12), 2526–2533 (2009). doi:10.1016/j.cpc.2008.11.005
3. Bickart, T.A.: An efficient solution process for implicit Runge-Kutta methods. SIAM J. Numer. Anal. **14**(6), 1022–1027 (1977). doi:10.1137/0714069
4. Brugnano, L., Caccia, G.F., Iavernaro, F.: Efficient implementation of Gauss collocation and Hamiltonian boundary value methods. Numerical Algorithms **65**(3), 633–650 (2014). doi:10.1007/s11075-014-9825-0
5. Butcher, J.C.: On the implementation of implicit Runge-Kutta methods. BIT Numer. Math. **16**(3), 237–240 (1976). doi:10.1007/BF01932265
6. Hairer, E., Lubich, C., Wanner, G.: Geometric numerical integration: structure-preserving algorithms for ordinary differential equations, vol. 31. Springer Science & Business Media. doi:10.1007/3-540-30666-8 (2006)
7. Hairer, E., Wanner, G. Solving ordinary differential equations, 2nd edn., vol. II. Springer, Berlin (1996). doi:10.1007/978-3-642-05221-7
8. Hairer, E., McLachlan, R.I., Razakarivony, A.: Achieving Brouwer's law with implicit Runge-Kutta methods. BIT Numer. Math. **48**(2), 231–243 (2008). doi:10.1007/s10543-008-0170-3
9. Hairer, E., Murua, A., Sanz-Serna, J.M.: The non-existence of symplectic multi-derivative Runge-Kutta methods. BIT Numer. Math. **34**(1), 80–87 (1994). doi:10.1007/BF01935017
10. Higham, N.J.: Accuracy and stability of numerical algorithms. Siam. doi:10.1137/1.9780898718027 (2002)
11. Jay, L.O.: Preconditioning of implicit Runge-Kutta methods. Scalable Computing: Practice and Experience. **10** (2009)
12. Kahan, W.: Further remarks on reducing truncation errors. Commun. ACM **8**(1), 40 (1965)
13. Liniger, W., Willoughby, R.A.: Efficient integration methods for stiff systems of ordinary differential equations. SIAM J. Numer. Anal. **7**(1), 47–66 (1970). doi:10.1137/0707002
14. Muller, J., Brisebarre, N., De Dinechin, F., Jeannerod, C., Lefevre, V., Melquiond, G., Revol, N., Stehlé, D., Torres, S.: Handbook of floating-point arithmetic. Springer Science & Business Media. doi:10.1007/978-0-8176-4705-6 (2009)
15. Saad, Y.: Iterative methods for sparse linear systems. SIAM. doi:10.1137/1.9780898718003.bm (2003)
16. Sanz Serna, J., Calvo, M.: Numerical Hamiltonian problems. Chapman and Hall (1994)
17. Xie, D.: A new numerical algorithm for efficiently implementing implicit Runge-Kutta methods. Department of Mathematical Sciences. University of Wisconsin, Milwaukee, Wisconsin USA (2009)