Efficient implementation of Radau collocation methods

Luigi Brugnano[a], Felice Iavernaro[b], Cecilia Magherini[c,*]

[a]*Dipartimento di Matematica e Informatica "U. Dini", Università di Firenze, Italy*
[b]*Dipartimento di Matematica, Università di Bari, Italy*
[c]*Dipartimento di Matematica, Università di Pisa, Italy*

Abstract

In this paper we define an efficient implementation of Runge-Kutta methods of Radau IIA type, which are commonly used when solving stiff ODE-IVPs problems. The proposed implementation relies on an alternative low-rank formulation of the methods, for which a splitting procedure is easily defined. The linear convergence analysis of this splitting procedure exhibits excellent properties, which are confirmed by its performance on a few numerical tests.

Keywords: Radau IIA collocation methods, W-transformation, Implicit Runge-Kutta methods,, Singly Implicit Runge-Kutta methods, Splitting, Hamiltonian BVMs.
2010 MSC: 65L04, 65L05, 65L06, 65L99

1. Introduction

The efficient numerical solution of implicit Runge-Kutta methods has been the subject of many investigations in the last decades, starting from the seminal paper of Butcher [17, 18] (see also [3]). An s-stage R-K method applied to the initial value problem

$$y' = f(y), \qquad y(t_0) = y_0 \in \mathbb{R}^m, \tag{1}$$

yields a nonlinear system of dimension sm which takes the form

$$G(\boldsymbol{y}) \equiv \boldsymbol{y} - \boldsymbol{e} \otimes y_0 - hA \otimes I_m\, f(\boldsymbol{y}) = \boldsymbol{0}, \tag{2}$$

where, in general, $I_r \in \mathbb{R}^{r\times r}$ is the identity matrix of order r,

$$\boldsymbol{e} = \begin{pmatrix} 1 \\ \vdots \\ 1 \end{pmatrix} \in \mathbb{R}^s, \qquad \boldsymbol{y} = \begin{pmatrix} y_1 \\ \vdots \\ y_s \end{pmatrix}, \qquad f(\boldsymbol{y}) = \begin{pmatrix} f(y_1) \\ \vdots \\ f(y_s) \end{pmatrix}, \tag{3}$$

*Corresponding author
Email addresses: luigi.brugnano@unifi.it (Luigi Brugnano), felix@dm.uniba.it (Felice Iavernaro), magherini@dm.unipi.it (Cecilia Magherini)

$y_1, \ldots, y_s$ being the internal stages. It is common to solve (2) by a simplified Newton iteration, namely, for $k = 0, 1, \ldots,$

$$\begin{aligned} &(I - hA \otimes J)\, \Delta^{(k)} = -G(\boldsymbol{y}^{(k)}), \\ &\boldsymbol{y}^{(k+1)} = \boldsymbol{y}^{(k)} + \Delta^{(k)}, \end{aligned} \tag{4}$$

where $I \equiv I_{sm}$, J is the Jacobian of f evaluated at some intermediate point and $\boldsymbol{y}^{(0)}$ an initial approximation of the stage vector: for instance, $J = \frac{\partial f}{\partial y}(y_0)$ and $\boldsymbol{y}^{(0)} = \boldsymbol{e} \otimes y_0$. To reduce the computational efforts associated with the solution of (4), a suitable linear change of variables on the s stages of the method is often introduced with the goal of simplifying the structure of the system itself. This is tantamount to performing a similarity transformation, commonly referred to as *Butcher transformation*, that puts the coefficient matrix A of the R-K method in a simpler form, i.e. a diagonal or triangular matrix. Let $B = TAT^{-1}$ be such a transformation. System (4) becomes

$$(I - h(B \otimes J))\, (T \otimes I_m)\Delta^{(k)} = -(T \otimes I_m)G(\boldsymbol{y}^{(k)}), \tag{5}$$

with the obvious advantage that the costs associated with the LU factorizations decrease from $O(s^3m^3)$ to $O(sm^3)$ *flops*.[1] In particular, if A has a one-point spectrum one only needs a single LU decomposition and the cost further reduces to $O(m^3)$ *flops* [15]. However, for many fully implicit methods of interest, the matrix A possesses complex conjugate pairs of eigenvalues which will appear as diagonal entries in the matrix B. In such a case, it is computationally more advantageous to allow B to be block-diagonal, with each 2×2 diagonal block corresponding to a complex conjugate pair of eigenvalues of A. Each subsystem of dimension $2m$ is then turned into an m-dimensional complex system. This is the standard procedure used in the codes RADAU5 [29, 36] and RADAU [30, 36], the former a variable-step fixed-order code, and the latter a variable-order variant, both based upon Radau-IIA formulae (of orders 5, 9, and 13).

Subsequent attempts to derive implicit high-order methods, for which the discrete problem to be solved can be cast in a simplified form, have been made, e.g., in [1, 19]. This line of investigation has been further refined in later papers (see, e.g., [20, 21, 22, 23]). Sometimes, the formulation of the discrete problem has been suitably modified, in order to induce a corresponding "natural splitting" procedure, as is done, e.g., in [4, 10, 11] (see also [12, 14]).

A different approach to the problem is that of considering suitable splitting procedures for solving the generated discrete problems [2, 24, 25, 26, 27, 28, 31, 32, 33, 34, 35]. A particularly interesting splitting scheme, first introduced in

[1] One *flop* is an elementary *fl*oating-point *op*eration.

[32], is that induced by the Crout factorization of the coefficient matrix A, namely $A = LU$, with L lower triangular and U upper triangular with unit diagonal entries. After observing that, for many remarkable R-K methods, the lower triangular part of A is *dominant*, in [32] the authors suggest to replace the matrix A in (4) with the matrix L, thus obtaining the scheme

$$\begin{aligned} &(I - hL \otimes J)\,\Delta^{(k)} = -G(\boldsymbol{y}^{(k)}),\\ &\boldsymbol{y}^{(k+1)} = \boldsymbol{y}^{(k)} + \Delta^{(k)}. \end{aligned} \tag{6}$$

Compared to (4) this scheme only requires the sequential solution of s subsystems of dimension m and therefore a global cost of $O(sm^3)$ elementary operations. Moreover, the s LU factorizations of the matrices $I_m - hl_{ii}J$ (l_{ii} being the ith diagonal entry of L), and the evaluations of the components of $G(\boldsymbol{y}^{(k)})$ may be done in parallel. This is why the corresponding methods have been named *parallel triangularly implicit iterated R-K methods* (PTIRK).

On the other hand, if the original modified Newton process (4) converges in one iterate on linear problems, the same no longer holds true for (6), due to the approximation $A \simeq L$. Applying the method to the linear test equation $y' = \lambda y$ yields the following estimation for the error $e^{(k)} = y^{(k)} - y$:

$$e^{(k+1)} = M(q)e^{(k)}, \qquad M(q) = q(I_s - qL)^{-1}(A - L), \tag{7}$$

with $q = h\lambda$. Matrix $M(q)$ is referred to as the *amplification matrix* associated with the method and its properties influence the rate of convergence of the scheme (6) according to a first order convergence analysis (see Section 4).

In this paper we wish to combine both the approaches described above and epitomized at formulae (5) and (6), to derive an efficient implementation of Radau IIA methods on sequential computers. In fact, the above discussion begs the following question: is it possible to perform a change of variables of the stage vector such that, for the new system (5), the matrix B admits a LU factorization with constant diagonal entries? In the affirmative, a single LU factorization would be needed to solve (6), with a cost of only $O(m^3)$ *flops*. A first positive answer in this direction has been given in [2] for general R-K methods. Later on, in [35], an optimal splitting of the form (6) has been devised for the Radau IIA method of order three (two stages), with $l_{11} = l_{22}$.

In this paper, we follow a different route, which relies on a *low-rank* formulation of Radau IIA collocation methods. Low-rank R-K methods have been recently introduced in a series of papers in the context of numerical geometric integration [5, 6, 7, 8, 9] (see also [16] for an application of low-rank R-K methods to stochastic differential equations).

Furthermore, our aim is not to destroy the overall convergence features of the simplified Newton method (4). Thus, instead of (6), we first recast system (4) as

$$(I - h(L \otimes J))\,\Delta^{(k)} = h\,((A - L) \otimes J)\,\Delta^{(k)} - G(\boldsymbol{y}^{(k)}), \quad k = 0, 1, \ldots,$$

and then, we retrieve an approximation of the unknown vector $\Delta^{(k)}$ by means of the *inner iteration*

$$(I - h(L \otimes J))\,\Delta_{\nu+1}^{(k)} = h\,((A - L) \otimes J)\,\Delta_{\nu}^{(k)} - G(\boldsymbol{y}^{(k)}), \quad \nu = 0, 1, \ldots, \tag{8}$$

starting at $\Delta_0^{(k)} = 0$. The inner scheme (8) could be iterated to convergence or stopped after a suitable number, say r, of steps. We see that (6) corresponds to (8) performed with one single inner iteration. Considering that no function evaluations are needed during the implementation of (8), we aim to perform the minimum number r of inner iterations that does not alter the convergence rate of the outer iteration (5).

The convergence properties of the purely linear scheme (8) continue to be described by the amplification matrix $M(q)$ defined at (7). In fact, its iteration matrix is

$$h(I - h(L \otimes J))^{-1}((A - L) \otimes J),$$

which reduces to $M(q)$ for the individual components corresponding to the eigenvalues λ of J. An advantage of the change of variable we propose is that a fast convergence rate is guaranteed at the very first steps of the process, and we will show that, in many practical situations, choosing $\nu \leq s$ produces very good results (see Table 3).

The paper is organized as follows. The low-rank formulation of Gauss Radau IIA methods is presented in Section 2, while the splitting procedure is defined in Section 3. Its convergence analysis and some comparisons with similar splitting procedures are reported in Section 4. Section 5 is devoted to some numerical tests with the fortran 77 code RADAU5 [29, 36], modified according to the presented procedure. Finally, a few conclusions are reported in Section 6, along with future directions of investigations.

2. Augmented low-rank implementation of Radau IIA methods

The discrete problem generated by the application of an s-stage ($s \geq 2$) Radau IIA method to problem (1) may be cast in vector form, by using the W-transformation [29], as:

$$\boldsymbol{y} = \boldsymbol{e} \otimes y_0 + h\mathcal{P}X_s\mathcal{P}^{-1} \otimes I_m\, f(\boldsymbol{y}), \tag{9}$$

where $\boldsymbol{e}$, $\boldsymbol{y}$ and $f(\boldsymbol{y})$ are defined at (3), while the matrices $\mathcal{P}$ and X_s are defined as

$$\mathcal{P} = \begin{pmatrix} P_0(c_1) & \dots & P_{s-1}(c_1) \\ \vdots & & \vdots \\ P_0(c_s) & \dots & P_{s-1}(c_s) \end{pmatrix}, \quad X_s = \begin{pmatrix} \frac{1}{2} & -\xi_1 & & & \\ \xi_1 & 0 & \ddots & & \\ & \ddots & \ddots & -\xi_{s-1} & \\ & & \xi_{s-2} & 0 & -\xi_{s-1} \\ & & & \xi_{s-1} & \beta_s \end{pmatrix}, \tag{10}$$

with $\{P_j\}$ the shifted and normalized Legendre polynomials on the interval $[0, 1]$,

$$\int_0^1 P_i(x)P_j(x)\mathrm{d}x = \delta_{ij}, \qquad i, j \geq 0,$$

and

$$\xi_i = \frac{1}{2\sqrt{4i^2-1}}, \quad i = 1, \dots, s-1, \qquad \beta_s = \frac{1}{4s-2}.$$

Clearly, h is the stepsize and the abscissae $\{c_1, \dots, c_s\}$ are the Gauss-Radau nodes in $[0, 1]$. In particular, $c_s = 1$, so that y_s is the approximation to the true solution at the time $t_1 = t_0 + h$.

We now derive an augmented low-rank Runge-Kutta method, which is equivalent to (9), by following an approach similar to that devised in [5] to introduce *Hamiltonian boundary value methods* (HBVMs), a class of energy-preserving R-K methods. In more detail, we choose an auxiliary set of distinct abscissae,

$$0 < \hat{c}_1 < \cdots < \hat{c}_s = 1, \tag{11}$$

and define the following change of variables involving the internal stages y_i:

$$\hat{\boldsymbol{y}} = \hat{\mathcal{P}}\mathcal{P}^{-1} \otimes I_m \, \boldsymbol{y}, \tag{12}$$

with

$$\hat{\boldsymbol{y}} = \begin{pmatrix} \hat{y}_1 \\ \vdots \\ \hat{y}_s \end{pmatrix}, \qquad \hat{\mathcal{P}} = \begin{pmatrix} P_0(\hat{c}_1) & \dots & P_{s-1}(\hat{c}_1) \\ \vdots & & \vdots \\ P_0(\hat{c}_s) & \dots & P_{s-1}(\hat{c}_s) \end{pmatrix}.$$

The vectors $\{\hat{y}_i\}$, $i = 1, \dots, s$, called *auxiliary stages*,[2] are nothing but the values at the abscissae (11) of the polynomial interpolating the internal stages $\{y_i\}$. Substituting (12) into (2) yields the new nonlinear system in the unknown $\hat{\boldsymbol{y}}$ (notice

[2]They are called *silent stages* in the HBVMs terminology, since their presence does not alter the complexity of the resulting nonlinear system. Similarly, the abscissae (11) are called *silent abscissae*.

that $\hat{\mathcal{P}}\mathcal{P}^{-1}\boldsymbol{e} = \boldsymbol{e}$):

$$\hat{G}(\hat{\boldsymbol{y}}) \equiv \hat{\boldsymbol{y}} - \boldsymbol{e} \otimes y_0 - h\hat{\mathcal{P}} X_s \mathcal{P}^{-1} \otimes I_m \, f\left(\mathcal{P}\hat{\mathcal{P}}^{-1} \otimes I_m \, \hat{\boldsymbol{y}}\right) = \boldsymbol{0}. \tag{13}$$

Of course, after computing $\hat{\boldsymbol{y}}$, the solution must be advanced in the standard manner, that is by means of the last component, y_s, of the original stage vector $\boldsymbol{y}$. However notice that $\hat{c}_s = c_s \Rightarrow \hat{y}_s = y_s$, so that this step of the procedure is costless.

In the next section, we show that the auxiliary abscissae (11) can be chosen so that the solution of the corresponding simplified Newton iteration (see (14) below) is more efficient than solving (4). We end this section by noticing that system (13) is actually identified by a R-K method with rank deficient coefficient matrix.

Theorem 1. *The method (12)-(13) can be cast as a Runge-Kutta method with* $2s$*-stages, defined by the following Butcher tableau:*

$$\begin{array}{c|cc} \hat{\boldsymbol{c}} & O & \hat{\mathcal{P}} X_s \mathcal{P}^{-1} \\ \boldsymbol{c} & O & \mathcal{P} X_s \mathcal{P}^{-1} \\ \hline & \boldsymbol{0}^T & \boldsymbol{b}^T \end{array}$$

where $O \in \mathbb{R}^{s\times s}$ *and* $\boldsymbol{0} \in \mathbb{R}^s$ *are the zero matrix and vector, respectively,* $\boldsymbol{c}$*,* $\hat{\boldsymbol{c}}$ *are the vectors with the Radau abscissae and the auxiliary abscissae (11), respectively, and* $\boldsymbol{b}$ *contains the weights of the Radau quadrature.*

3. The splitting procedure

The simplified Newton iteration (see (4)) applied to (13) reads

$$\begin{array}{l} \left(I - h\hat{\mathcal{P}} X_s \hat{\mathcal{P}}^{-1} \otimes J\right) \hat{\Delta}^{(k)} = -\hat{G}(\hat{\boldsymbol{y}}^k), \\ \hat{\boldsymbol{y}}^{(k+1)} = \hat{\boldsymbol{y}}^{(k)} + \hat{\Delta}^{(k)}. \end{array} \tag{14}$$

As we can see, its structure is precisely the same as that we would obtain by applying the simplified Newton iteration directly to the original system (9), with the only difference that the matrix $\hat{\mathcal{P}}$ in (14) should be replaced by $\mathcal{P}$.

As was emphasized in the introduction, to simplify the structure of systems such as (14), van der Houwen and de Swart [32, 33] proposed to replace the matrix $(\mathcal{P} X_s \mathcal{P}^{-1})$ in (9) with the lower triangular matrix L arising from its Crout factorization. The advantage is that, in such a case, to perform the iteration, one has to factorize s matrices having the same size m as that of the continuous problem with a noticeable saving of work. They showed that on parallel computers this approach gives very interesting speedups over more standard approaches based on the use of the LU factorization: indeed, the leading term lowers from $\frac{2}{3}(sm)^3$ to $s\frac{2}{3}m^3$

flops which, moreover, can be performed in parallel on s processors. This is symptomatic of the fact that LU factorizations generally give a relevant contribution to the overall execution time of a given code.

Similarly, here we want to take advantage from both the Crout factorization of $(\hat{\mathcal{P}} X_s \hat{\mathcal{P}}^{-1})$ appearing in (14) and the freedom of choosing the auxiliary abscissae $\{\hat{c}_i\}$, to devise an iteration scheme that only requires a single LU factorization of a system of dimension m which is, therefore, suitable for sequential programming. Differently from [32], we continue to adopt the iteration (14) (outer iteration) and retrieve an approximation to $\hat{\Delta}^{(k)}$ via the linear inner iteration

$$\left(I - h\hat{L} \otimes J\right) \hat{\Delta}_{\nu+1}^{(k)} = h \left((\hat{\mathcal{P}} X_s \hat{\mathcal{P}}^{-1} - \hat{L}) \otimes J\right) \hat{\Delta}_{\nu}^{(k)} - \hat{G}(\hat{\boldsymbol{y}}^{(k)}), \qquad \nu = 0, 1, \ldots, \tag{15}$$

where

$$\hat{\mathcal{P}} X_s \hat{\mathcal{P}}^{-1} = \hat{L}\hat{U},$$

with $\hat{L}$ lower triangular and $\hat{U}$ upper triangular with unit diagonal entries. Our purpose is to choose the auxiliary abscissae (11) so that all the diagonal entries of $\hat{L}$ are equal to each other, i.e.,

$$(\hat{L})_{jj} = \sqrt[s]{\det(X_s)}, \qquad j = 1, \ldots, s. \tag{16}$$

In so doing, one has to factor only *one* $m \times m$ matrix, to carry out the inner iteration (15). Concerning the diagonal entry in (16), the following result can be proved by induction. For later use, we define the function

$$\ell_j(\hat{c}_1, \ldots, \hat{c}_s) \equiv (\hat{L})_{jj}, \tag{17}$$

in order to make clear that $(\hat{L})_{jj}$ does depend on the choice of the auxiliary abscissae (11).

Theorem 2. *Let X_s be defined according to (10) and let*

$$\eta = \begin{cases} 0, & \text{if } s \text{ is even,} \\ 1, & \text{if } s \text{ is odd.} \end{cases}$$

Then

$$\det(X_s) = \frac{2^{1-s}}{\prod_{i=1+\eta}^{s-1}(4i^2 - 1)}. \tag{18}$$

Consequently, from (16) one has:

$$d_s := (\hat{L})_{jj} = \frac{2^{\frac{1}{s}-1}}{\left(\prod_{i=1+\eta}^{s-1}(4i^2 - 1)\right)^{\frac{1}{s}}}, \qquad j = 1, \ldots, s. \tag{19}$$

Table 1: Auxiliary abscissae for the s-stage Radau method, $s = 2, \ldots, 5$, and the diagonal entry d_s (see (19) of the corresponding factor $\hat{L}$.

$s = 2$	
$\hat{c}_1$	$(6 - \sqrt{6})/(6 + 2\sqrt{6})$
$\hat{c}_2$	1
d_2	0.40824829046386301636621401245098
$s = 3$	
$\hat{c}_1$	0.18589230221764097222357873465176
$\hat{c}_2$	0.50022434784008286059148415923632
$\hat{c}_3$	1
d_3	0.25543647746451770219954184281099
$s = 4$	
$\hat{c}_1$	0.12661575733255931078112184952036
$\hat{c}_2$	0.34154548143311325099490740728171
$\hat{c}_3$	0.56937072098419698874387077046544
$\hat{c}_4$	1
d_4	0.18575057999133599176307088298897
$s = 5$	
$\hat{c}_1$	0.09527975140867214336447374571157
$\hat{c}_2$	0.28143874673988994521203045137949
$\hat{c}_3$	0.38152142820340929736570124768463
$\hat{c}_4$	0.60680555490108389442461323421422
$\hat{c}_5$	1
d_5	0.14591154019899779261811749554182

In Table 1 we list the auxiliary abscissae $\{\hat{c}_i\}_{i=1,\ldots,s}$ and the diagonal entries d_s, given by (19), for the Radau IIA methods with $s = 2, \ldots, 5$ stages. Notice that, having set $\hat{c}_s = 1$, the free parameters are $s - 1$, namely $\hat{c}_i$, $i = 1, \ldots, s - 1$. We have formally derived the expression (17) of the first $s - 1$ diagonal entries of the matrix $\hat{L}$ as a function of these unknowns, and then we have solved the $(s - 1)$-dimensional system

$$\ell_j(\hat{c}_1, \ldots, \hat{c}_s) = d_s, \qquad j = 1, \ldots, s - 1,$$

with the aid of the symbolic computation software Maple. From (18) it is clear that the last diagonal element of $\hat{L}$ will be automatically equal to d_s, too.

3.1. Further reduction of the computational cost

As was observed in [20] in the context of singly implicit R-K methods, the implementation of a formula such as (15) consists of a block-forward substitution

which requires the computation of $(T \otimes J)\hat{\Delta}_{\nu+1}^{(k)}$, with

$$T = \hat{L} - d_s I_s$$

(i.e., the strictly lower triangular part of matrix $\hat{L}$), at a cost of $O(s^2m + m^2s)$ operations. The $O(m^2s)$ term, as well as the m^2 multiplications for computing $(hd_s)J$ before the factorization of the matrix $I_m - hd_sJ$, may be eliminated by multiplying both sides of (15) by

$$h^{-1}\hat{L}^{-1} \otimes I_m.$$

Considering that

$$\hat{L}^{-1} = d_s^{-1} I_s - S,$$

with S strictly lower triangular, system (15) then takes the form

$$\left(\frac{1}{hd_s} I - I_s \otimes J\right) \hat{\Delta}_{\nu+1}^{(k)} = \frac{1}{h}(S \otimes I_m)\hat{\Delta}_{\nu+1}^{(k)} + (C \otimes J)\ \hat{\Delta}_{\nu}^{(k)} + R^{(k)}, \qquad (20)$$

where

$$C = \hat{L}^{-1}(\hat{\mathcal{P}} X_s \hat{\mathcal{P}}^{-1} - \hat{L}) = \hat{U} - I_s \quad \text{and} \quad R^{(k)} = -\frac{1}{h}(\hat{L}^{-1} \otimes I_m)\hat{G}(\hat{\boldsymbol{y}}^{(k)}). \qquad (21)$$

Notice that, since C is strictly upper triangular, the multiplication of J by the first block-component of $\hat{\Delta}_{\nu}^{(k)}$ may be skipped. But we can go another step beyond and completely eliminate any $O(m^2)$ term in the computation of the term

$$(C \otimes J)\ \hat{\Delta}_{\nu}^{(k)}$$

at right-hand side of (20). This is true at the very first step, since, by definition,

$$\hat{\Delta}_0^{(k)} = 0.$$

Let us set

$$\boldsymbol{w}_\nu := (C \otimes J)\ \hat{\Delta}_{\nu}^{(k)} + R^{(k)},$$

which is part of the right-hand side of (20). Thus $\boldsymbol{w}_0 = R^{(k)}$ and the first step of (20) is equivalent to the system

$$I_s \otimes \left[(hd_s)^{-1} I_m - J\right] \hat{\Delta}_1^{(k)} = h^{-1}(S \otimes I_m)\hat{\Delta}_1^{(k)} + \boldsymbol{w}_0. \qquad (22)$$

After solving for the unknown $\hat{\Delta}_1^{(k)}$, we set $\boldsymbol{v}_1$ equal to the right-hand side of (22), which can be exploited to compute the term

$$(I_s \otimes J)\hat{\Delta}_1^{(k)} = (hd_s)^{-1}\hat{\Delta}_1^{(k)} - \boldsymbol{v}_1,$$

at a cost of $O(ms)$ operations. It follows that

$$(C \otimes J)\,\hat{\Delta}_1^{(k)} = (C \otimes I_m)\left[(I_s \otimes J)\,\hat{\Delta}_1^{(k)}\right] = (C \otimes I_m)\left[(hd_s)^{-1}\hat{\Delta}_1^{(k)} - \boldsymbol{v}_1\right],$$

and thus $\boldsymbol{w}_1 = (C \otimes J)\ \hat{\Delta}_1^{(k)} + R^{(k)}$ may be computed with $O(s^2 m)$ floating point operations. This trick may be repeated at the subsequent steps, thus resulting in the following algorithm:

$\boldsymbol{w}_0 := R^{(k)}$

solve: $I_s \otimes \left[(hd_s)^{-1} I_m - J\right]\hat{\Delta}_1^{(k)} = h^{-1}(S \otimes I_m)\hat{\Delta}_1^{(k)} + \boldsymbol{w}_0$ (23)

do $\nu = 1, 2, \ldots$

$\boldsymbol{v}_\nu \ := \ h^{-1}(S \otimes I_m)\hat{\Delta}_\nu^{(k)} + \boldsymbol{w}_{\nu-1}$ (24)

$\boldsymbol{w}_\nu \ := \ (C \otimes I_m)\left[(hd_s)^{-1}\hat{\Delta}_\nu^{(k)} - \boldsymbol{v}_\nu\right] + R^{(k)}$ (25)

solve: $I_s \otimes \left[(hd_s)^{-1} I_m - J\right]\hat{\Delta}_{\nu+1}^{(k)} = h^{-1}(S \otimes I_m)\hat{\Delta}_{\nu+1}^{(k)} + \boldsymbol{w}_\nu$ (26)

end do

Notice that $\boldsymbol{v}_\nu$ is just the right-hand side of the preceding linear system and thus it is freely available as soon as the system has been solved. For sake of completeness, we also compute the complexity of the above algorithm (the strictly lower and upper triangular structure of the matrices S and C, respectively, is taken into account):

LU factorization of matrix $[(hd_s)^{-1} I_m - J]$: $\frac{2}{3}m^3 - \frac{1}{2}m^2 - \frac{1}{6}m$ *flops*;

(23): $2sm^2 + s^2 m - sm$ *flops*;

(24): no additional *flops* required, besides those needed for (23) or (26);

(25): $(s-1)(s+2)m$ *flops*;

(26): the same as (23).

Consequently, by also considering (13)-(21), and if ν_{in} inner iterations are performed for solving (14), and ν_{out} such iterations are needed, the total cost of the step amounts to:

$$\frac{2}{3}m^3 - \frac{1}{2}m^2 - \frac{1}{6}m + \nu_{out}\left\{4(s^2+1)m - 5sm + \nu_{in}\left[2sm^2 + 2(s^2-1)m\right]\right\}\ \textit{flops}, \tag{27}$$

that is, for m large,

$$\approx \quad 2m^2\left[\frac{m}{3} + (\nu_{out}\nu_{in})s\right] \quad \textit{flops},$$

plus 1 Jacobian and $\nu_{out} s$ function evaluations.

In order to make a comparison, we consider the traditional approach in [17] (see also [3]), modified as suggested in [29, pag. 121-122], for Radau IIA methods, sketched below.

Preliminarly, for the s-stage methods, one performs the factorization of $\lfloor s/2 \rfloor$ complex matrices and $\lceil s/2 \rceil - \lfloor s/2 \rfloor$ real matrices, having dimension $m \times m$, with m the size of the continuous problem.

Then, for each iteration, one needs:

- two variable transformations by the real matrices $T \otimes I_m$ and $T^{-1} \otimes I_m$, with $T \in \mathbb{R}^{s \times s}$;
- 3s real *axpy*-s of length m;[3]
- s function evaluations;
- $\lfloor s/2 \rfloor$ back-substitutions with the complex factors and $\lceil s/2 \rceil - \lfloor s/2 \rfloor$ back-substitution with the real factors.

Consequently, by considering that:

- a complex factorization costs $\frac{8}{3}m^3 + \frac{7}{3}m - 5$ *flops*;
- a complex back-substitution costs $8m^2 + 3m$ *flops*;

one obtains that, if ν iterations are needed for the step, then the total cost amounts to

$$\lfloor s/2 \rfloor \left(\frac{8}{3}m^3 + \frac{7}{3}m - 5 \right) + (\lceil s/2 \rceil - \lfloor s/2 \rfloor) \left(\frac{2}{3}m^3 - \frac{1}{2}m^2 - \frac{1}{6}m \right) + \nu \left[4s^2 m + 4sm + \lfloor s/2 \rfloor (6m^2 + 4m) + \lceil s/2 \rceil (2m^2 - m) \right] \ \textit{flops}, \tag{28}$$

which, for large m becomes approximately

$$m^2 \left[6 \lfloor s/2 \rfloor \left(\frac{m}{3} + \nu + \frac{1}{12} \right) + 2 \lceil s/2 \rceil \left(\frac{m}{3} + \nu - \frac{1}{4} \right) \right] \quad \textit{flops},$$

plus 1 Jacobian and νs function evaluations.

[3] As is well known, *axpy* is the acronym for "*alpha x plus y*", with $\boldsymbol{x}$ and $\boldsymbol{y}$ vectors and α a scalar.

4. Convergence analysis and comparisons

In this section we briefly analyze the splitting procedure (15). This will be done according to the linear analysis of convergence in [32] (see also [13]). In such a case, problem (1) becomes the celebrated test equation

$$y' = \lambda y, \qquad y(t_0) = y_0.$$

By setting, as usual, $q = h\lambda$, one then obtains that the error equation associated with (15) is given by

$$e_{\nu+1} = \hat{M}(q)e_\nu, \qquad \hat{M}(q) := q(I_s - q\hat{L})^{-1}\hat{L}(\hat{U} - I_s), \qquad \nu = 0, 1, \ldots,$$

where we have set $e_\nu = \Delta_\nu^{(k)} - \Delta^{(k)}$, that is the error vector at step ν (we neglect, for sake of simplicity, the index k of the *outer* iteration) and $\hat{M}(q)$ is the iteration matrix induced by the splitting procedure. This latter will converge if and only if its spectral radius,

$$\rho(q) := \rho(\hat{M}(q)),$$

is less than 1. The *region of convergence* of the iteration is then defined as

$$\mathbb{D} = \{q \in \mathbb{C} : \rho(q) < 1\}.$$

The iteration is said to be *A-convergent* if $\mathbb{C}^- \subseteq \mathbb{D}$. If, in addition, the *stiff amplification factor*,

$$\rho^\infty := \lim_{q\to\infty} \rho(q),$$

is null, then the iteration is said to be *L-convergent*. Clearly, A-convergent iterations are appropriate for A-stable methods, and L-convergent iterations are appropriate for L-stable methods. In our case, since

$$\hat{M}(q) \to (\hat{U} - I_s), \qquad q \to \infty, \tag{29}$$

which is a nilpotent matrix of index s, the iteration is L-convergent if and only if it is A-convergent. Since the iteration is well defined for all $q \in \mathbb{C}^-$ (due to the fact that the diagonal entry of $\hat{L}$, d_s, is positive) and $\rho(0) = 0$, A-convergence, in turn, is equivalent to require that the *maximum amplification factor*,

$$\rho^* = \max_{x\in\mathbb{R}} \rho(ix),$$

is not larger than 1. Another useful parameter is the *nonstiff amplification factor*,

$$\tilde{\rho} = \rho(\hat{L}(\hat{U} - I_s)), \tag{30}$$

that governs the convergence for small values of q since

$$\rho(q) \approx \tilde{\rho} q, \qquad \text{for } q \approx 0.$$

Clearly, the smaller ρ^* and $\tilde{\rho}$, the better the convergence properties of the iteration. In Table 2 we list the nonstiff amplification factors and the maximum amplification factors for the following L-convergent iterations applied to the s-stage Radau IIA methods:

(i) the iteration obtained by the original triangular splitting in [32];

(ii) the iteration obtained by the modified triangular splitting in [2];

(iii) the *blended* iteration obtained by the *blended implementation* of the methods, as defined in [10];

(iv) the iteration defined by (15).

We recall that the scheme (i) (first column) requires s real factorizations per iteration, whereas (ii)–(iv) only need one factorization per iteration. From the parameters listed in the table, one concludes that the proposed splitting procedure is the most effective among all the considered ones.

It is worth mentioning that the above amplification factors are defined in terms of the eigenvalues of the involved matrices. Therefore, they are significant if a large number of inner iterations are performed or if the initial guess is accurate enough. However, the number of inner iteration is usually small, so that it is also useful to check the so called *averaged amplification factors* over ν iterations, defined as follows (see (30) and (29)):

$$\tilde{\rho}_\nu = \sqrt[\nu]{\left\| \left[\hat{L}(\hat{U} - I_s) \right]^\nu \right\|}, \qquad \rho^*_\nu = \max_{x \in \mathbb{R}} \sqrt[\nu]{\|M(ix)^\nu\|}, \qquad \rho^\infty_\nu = \sqrt[\nu]{\left\| (\hat{U} - I_s)^\nu \right\|}.$$

Clearly,

$$\rho^\infty_\nu = 0, \qquad \forall \nu \geq s,$$

since matrix $\hat{U} - I_s$ is nilpotent of index s. Moreover,

$$\tilde{\rho}_\nu \to \tilde{\rho}, \qquad \rho^*_\nu \to \rho^*, \qquad \text{as} \qquad \nu \to \infty.$$

For this reason, in Table 3 we compare the asymptotic parameters $\tilde{\rho}$ and ρ^* (columns 2 and 3) with the averaged ones over s iterations (columns 4 and 5), for $s = 2, \ldots, 5$. As one can see, the iterations are still L-convergent after s iterations (the norm $\|\cdot\|_\infty$ has been considered). In the last three columns of the table, we list the amplification factors after just 1 inner iteration: in such a case, the iterations are no more L-convergent, though still A-convergent, up to $s = 4$.

Table 2: Amplification factors for the triangular splitting in [32], the modified triangular splitting in [2], the *blended* iteration in [10], and the splitting (15), for the s-stage Radau IIA methods.

	(i): triangular splitting in [32]		(ii): triangular splitting in [2]		(iii): *blended* iteration in [10]		(iv): triangular splitting (15)	
s	$\tilde{\rho}$	ρ^*	$\tilde{\rho}$	ρ^*	$\tilde{\rho}$	ρ^*	$\tilde{\rho}$	ρ^*
2	0.1500	0.1837	0.1498	0.1835	0.1498	0.1835	0.1498	0.1835
3	0.1853	0.3726	0.1375	0.3138	0.1674	0.3398	0.1333	0.3134
4	0.1728	0.5064	0.1236	0.4137	0.1535	0.4416	0.1174	0.3826
5	0.1496	0.6103	0.1090	0.4949	0.1367	0.5123	0.0787	0.3963

Table 3: Amplification factors, and averaged amplification factors after s inner iterations and 1 inner iteration, for the triangular splitting (15), for the s-stage Radau IIA methods.

s	$\tilde{\rho}$	ρ^*	$\tilde{\rho}_s$	ρ_s^*	$\tilde{\rho}_1$	ρ_1^*	ρ_1^∞
2	0.1498	0.1835	0.1498	0.1835	0.1498	0.2020	0.2020
3	0.1333	0.3134	0.1407	0.3378	0.1513	0.3984	0.3440
4	0.1174	0.3826	0.1316	0.4363	0.2169	0.6643	0.5172
5	0.0787	0.3963	0.1200	0.5841	0.2959	1.1141	0.9945

5. Numerical Tests

In this section, we report the results of a few numerical tests on some stiff problems, which have been obtained by using the RADAU5 code [29, 36] and a suitable modification of it which implements the splitting procedure with a fixed number of inner iterations, namely $\nu = 1, 2, 3$. Moreover, a simplified version of both codes, implementing a fixed-step integration procedure, has been also implemented. Clearly, further improvements could be obtained by dynamically varying the number of inner iterations as well as by implementing a suitable strategy, well tuned for the new iterative procedure, to decide whether the evaluation of the Jacobian can be avoided. In absence of such refinements, in order to verify the effectiveness of the proposed approach, we have forced the evaluation of the Jacobian after every accepted step by setting in input `work(3)=-1D0`. As a consequence, the factorization of the involved matrices is computed at each integration step. All the experiments have been done on a PC with an Intel Core2 Quad Q9400 @ 2.66GHz processor under Linux by using the GNU Fortran compiler `gfortran` with optimization flag `-Ofast`. We shall report two sets of numerical tests:

- the first set of tests contains the application of the simplified version of the codes, implementing a fixed-step strategy, in order to asses the complexity analysis made in Section 3.1;

- the second set of tests is made by using the original variable-step strategy implemented in the code RADAU5.

Concerning the first set of numerical tests, we have considered the following linear problem:

$$\begin{cases} y'(t) = J(t)\,(y(t) - \varphi(t)\boldsymbol{e}_m) + \varphi'(t)\boldsymbol{e}_m, & t \in (0,4], \\ y(0) = \varphi(0)\boldsymbol{e}_m, \end{cases} \tag{31}$$

where

$$\varphi(t) = 16(16+t^2)^{-1}, \qquad \boldsymbol{e}_m = (1,\ldots,1)^T \in \mathbb{R}^m,$$

whose solution is $y(t) = \varphi(t)\boldsymbol{e}_m$. Moreover,

$$J(t) = D^{-1}(t) F \hat{D} F^T D(t),$$

with

$$\begin{aligned} D(t) &= \mathrm{diag}\left(d_1(t), d_2(t), \ldots, d_m(t)\right), & d_i(t) &= \frac{m^2 + 4(it)^2}{m^2 + 5(it)^2}, \\ \hat{D} &= \mathrm{diag}\left(\hat{d}_1, \hat{d}_2, \ldots, \hat{d}_m\right), & \hat{d}_i &= \begin{cases} -10^4, & \text{if } i \bmod 10 = 1, \\ -1, & \text{otherwise,} \end{cases} \\ F &= \frac{1}{8}\begin{pmatrix} 8 & & & \\ 1 & \ddots & & \\ \vdots & \ddots & \ddots & \\ 1 & \ldots & 1 & 8 \end{pmatrix}. \end{aligned}$$

As one may easily realize, the complexity of one function evaluation can be made linear, in the dimension m of the problem, whereas the computation of the Jacobian is quadratic. This allows us to use the complexity analysis in Section 3.1, to predict the speed-up, by means of (27) and (28) (also taking into account function and Jacobian evaluations). We have solved problem (31) for increasing values of the dimension m, ranging from 50 to 400, by using a fixed step-size $h = 1/32$, thus performing 128 integration steps. In this series of tests, we compare a simplified version of the standard code RADAU5 (implementing a diagonalization of the Butcher matrix of the 3-stage Radau IIA method), with a modified version of the same simplified code, implementing a prescribed number of inner iterations. For the problem at hand, 2 inner iterations are the best choice, in order to produce a comparably accurate numerical approximation to the solution. The accuracy

Table 4: Statistics for the linear problem (31) concerning diagonalization (Diag) vs. splitting with 2 inner iterations (Split 2).

	m	mescd	feval	jeval	real LU	complex LU	real backsubs	complex backsubs	CPU time
Diag	100	11.82	2013	128	128	128	671	671	2.36E-01
	200	11.38	2055	128	128	128	685	685	1.61E+00
	300	11.29	2085	128	128	128	695	695	5.25E+00
	400	11.12	2112	128	128	128	704	704	1.25E+01
Split 2	100	12.04	2076	128	128	–	4152	–	1.18E-01
	200	11.85	2178	128	128	–	4356	–	6.23E-01
	300	11.63	2226	128	128	–	4452	–	1.77E+00
	400	11.57	2262	128	128	–	4524	–	3.89E+00

of the computed solution, in turn, is measured in terms of *mixed-error significant correct digits (mescd)*, defined as

$$mescd = -\log_{10} \|(y - y_{ref})./(\texttt{artol} + |y_{ref}|)\|_\infty , \qquad \texttt{artol} = \frac{\texttt{atol}}{\texttt{rtol}}, \tag{32}$$

where y is the final point in the computed solution, y_{ref} is the reference solution (which is known), `atol` and `rtol` are the input absolute and relative tolerances, respectively, and ./ denotes the componentwise division between vectors. In this case, since the variable step-size strategy has been disabled (we use, indeed, a constant step-size), we have set `artol` = 1 in (32). In Table 4 we report a few statistics concerning selected runs for increasing values of the dimension m of the problem. As one may see, the accuracy of the computed solution is similar (in the range 11-12 *mescd*), as well as the number of function evaluations (*feval*). Also the number of Jacobian evaluations (*jeval*) is the same since, as said above, a Jacobian evaluation is forced at each integration step, as well as the factorization of the involved matrices.
On the left of Figure 1, we plot the measured speed-up (stars) and, in solid line, that predicted by the complexity analysis made in Section 3.1. As one may see, the measured speed-up is smaller than the expected one: as an example, at $m = 400$ the measured speed-up is approximately 3.2, whereas the expected one is approximately 4. This fact can be explained by considering that the most time-consuming operation, for each of the two codes, is:

- the factorization of a $m \times m$ complex matrix, for the original code using diagonalization, and
- the factorization of a $m \times m$ real matrix, for the modified code using the splitting.

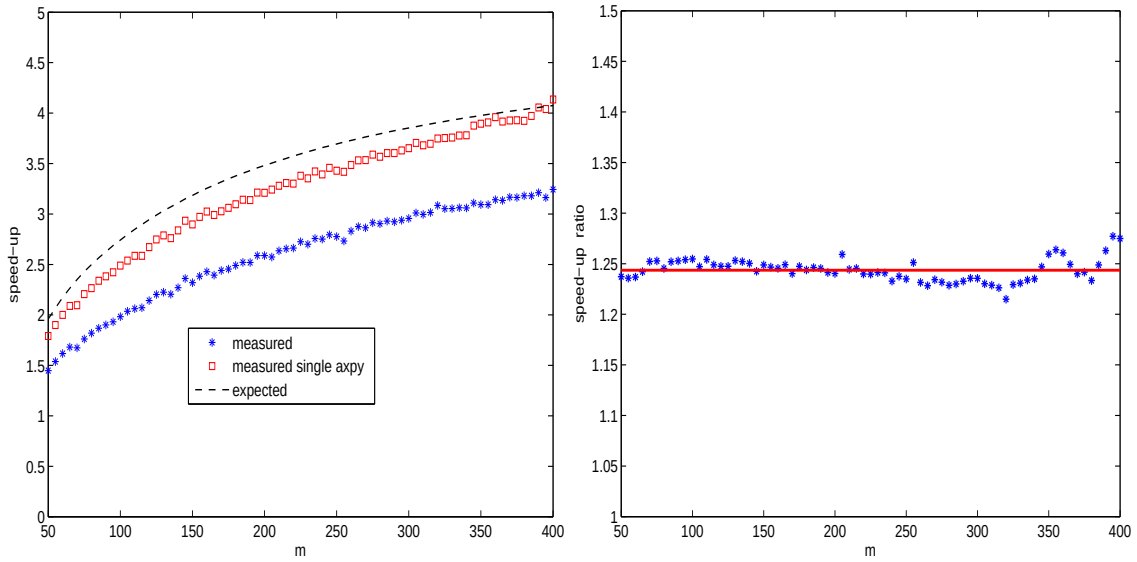

Figure 1: Speed-up for the linear problem (31) corresponding to the splitting procedure with two inner iterations and 128 integration steps (left-plot), and speed-up ratio when using the *single axpy* implementation (35) in the code (right-plot), showing an improvement of about 24%.

In the real factorization, the core operation is a (real) *axpy*,

$$\boldsymbol{y} \leftarrow \alpha \boldsymbol{x} + \boldsymbol{y}, \tag{33}$$

whereas for the complex factorization (which is implemented by computing separately real and imaginary parts, thus using real vectors) it is a *double* (real) *axpy*,

$$\boldsymbol{y} \leftarrow \alpha \boldsymbol{x} + \beta \boldsymbol{v} + \boldsymbol{y}. \tag{34}$$

This latter is optimized, by exploiting locality of data, through the use of the cache. In other words, even though all the vectors appearing in (33) and (34) have the same length, in general the execution time of (34) is smaller than twice the execution time for (33). This can be easily verified by splitting the double *axpy* (34) into two separate *single axpy*-s, i.e.,

$$\boldsymbol{y} \leftarrow \alpha \boldsymbol{x} + \boldsymbol{y}, \qquad \boldsymbol{y} \leftarrow \beta \boldsymbol{v} + \boldsymbol{y}. \tag{35}$$

Clearly (34) and (35) have the same complexity, in terms of *flops*, but different execution times. As matter of fact, by modifying the code implementing the diagonalization according to (35), the execution times increase, and we obtain a measured speed-up, which is plotted in squares in the left-plot of Figure 1. This latter curve, is now much closer to the expected one. Moreover, one obtains that, for the used computing platform, the performance improvement, deriving from

the use of (34) in place of (35), is more than 24%, as is shown in the right-plot of Figure 1.

In the second set of numerical tests, we compare the original code RADAU5 and a modification of it, implementing the splitting procedure, on three stiff problems taken from the *Test Set for IVP Solvers* [36]:

- *Elastic Beam problem*, of dimension $m = 80$;
- *Emep problem*, of dimension $m = 66$;
- *Ring Modulator problem*, of dimension $m = 15$.

In such a case, the variable step-size strategy implemented in RADAU5 has been used. The following input tolerances for the relative (*rtol*) and absolute (*atol*) errors and initial stepsizes (h_0) have been used:

- Elastic Beam problem: $rtol = atol = h_0 = 10^{-4-i/4}, \ \ i = 0, \ldots, 16$;
- Emep problem: $\quad rtol = 10^{-4-i/4}, \ \ i = 0, \ldots, 28, \quad atol = 1, \quad h_0 = 10^{-7}$;
- Ring Modulator problem: $\ rtol = atol = h_0 = 10^{-7-i/4}, \ \ i = 0, \ldots, 20$.

Figures 2, 3, and 4 show the obtained results as *work-precision diagrams*, where the CPU-time (in seconds) is plotted versus accuracy, measured in terms of *mescd* (see (32)).[4]

For the first two problems, the work-precision diagrams suggest that the splitting version of the RADAU5 code is more efficient than the original one, even starting with 1 inner iteration. Moreover, in Tables 5–8 we list a few statistics for the Elastic Beam problem, from which one deduces that, by using 2–3 inner iterations, the number of steps is approximately the same as the original code: in other words, the convergence rate of the outer iteration is preserved.

For the last problem (Ring Modulator), which has a much smaller size, the splitting with 2 and 3 inner iterations is less efficient than the original RADAU5 code. Nevertheless, when using a single inner iteration the algorithm uses a larger number of steps (8-10% more), as is shown in Tables 9 and 10, resulting into a much more accurate solution. In our understanding, this behaviour may be explained by considering that computing the vector field $f(t, \boldsymbol{y})$ of this problem is extremely cheap and, hence, accuracy is more conveniently obtained by acting on the number of function evaluations, rather than on the number of inner iterations.

[4]Indeed, a reference solution is known, for all problems in the *Test Set*.

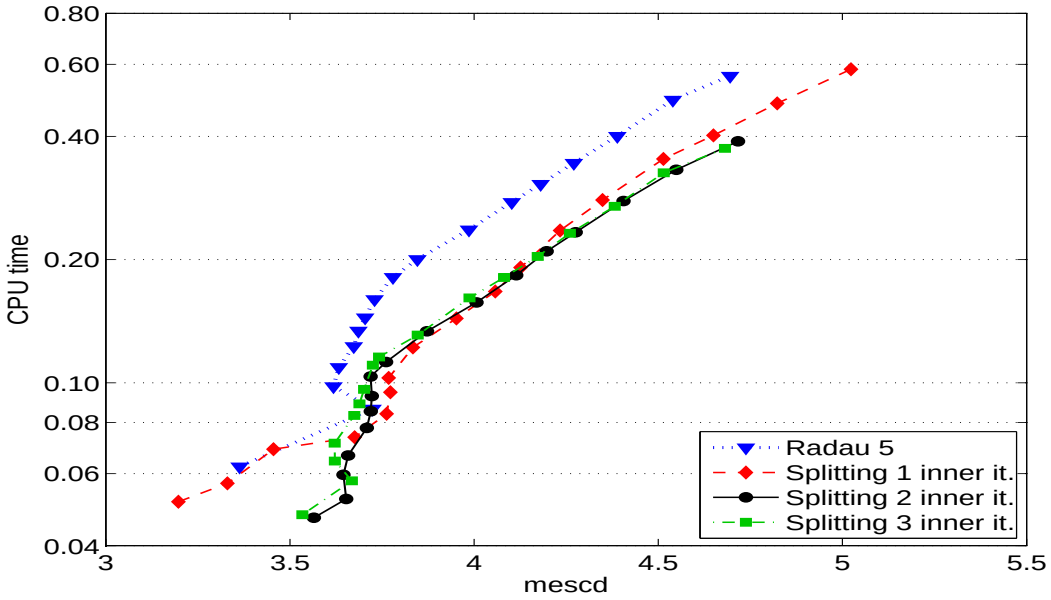

Figure 2: Work precision diagram for the Elastic Beam problem.

Even though it is difficult to exactly compute the obtained speed-ups, since the curves in the work-precision diagrams are not monotone, in general, it is possible to infer a mean speed-up of 1.43, for the Elastic Beam problem, a mean speed-up 1.84, for the Emep problem, and a mean speed-up 1.44, for the Ring Modulator problem. In the latter case, however, the speed-up increases with the required accuracy, reaching a value of approximately 2. Moreover, as explained above, these speed-ups could be larger by a 24% amount, if hardware/software optimizations were not present.

6. Conclusions

In this paper we have defined a splitting procedure for Radau IIA methods, derived by an *augmented low-rank* formulation of the methods. In such formulation, a set of *auxiliary abscissae* are determined such that the Crout factorization of a corresponding matrix associated with the method has constant diagonal entries, which leads to optimal complexity. Moreover, the presented iteration compares favorably with all previously defined iterative procedures for the efficient implementation of Radau IIA methods. The presented technique can be straightforwardly extended to other classes of implicit Runge-Kutta methods (e.g., collocation methods) and this will be the subject of future investigations.

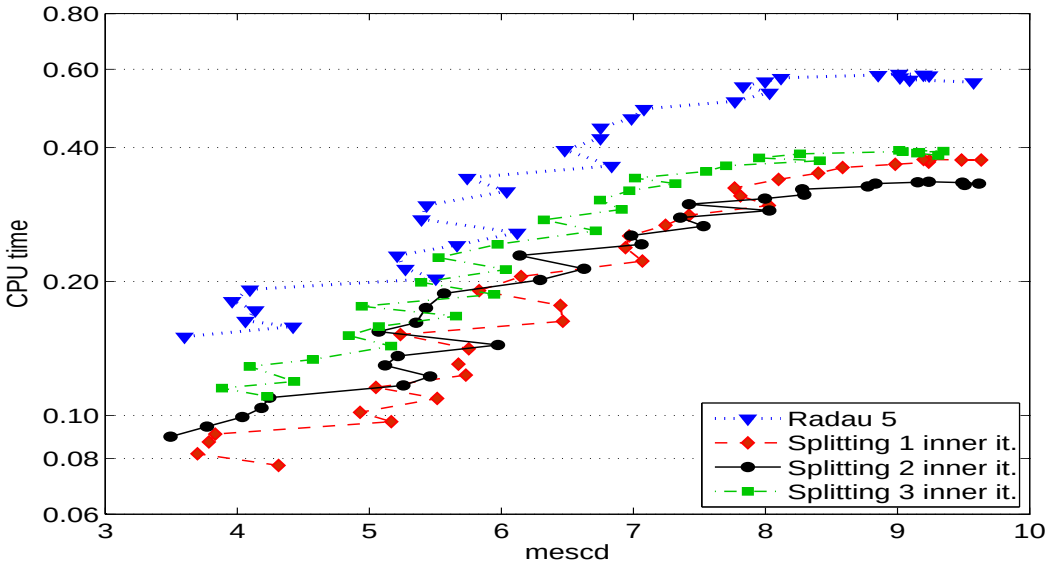

Figure 3: Work precision diagram for the Emep problem.

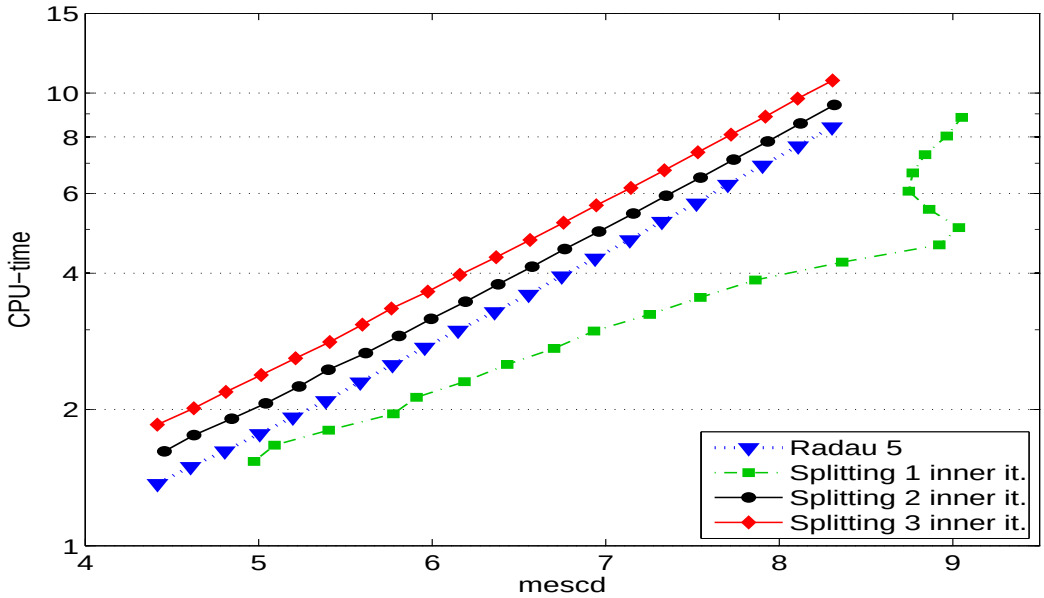

Figure 4: Work precision diagram for the Ring Modulator problem.

Table 5: Statistics for the Elastic Beam problem, RADAU5.

rtol	mescd	steps	accept	feval	jeval	LU	CPU-time
1.00E-04	3.36	55	49	380	49	55	6.24E-02
1.00E-05	3.67	112	95	764	95	112	1.23E-01
1.00E-06	3.78	162	146	1103	146	162	1.81E-01
1.00E-07	4.18	275	251	1853	251	275	3.06E-01
1.00E-08	4.69	507	459	3417	459	507	5.62E-01

Table 6: Statistics for the Elastic Beam problem, RADAU5, split 1.

rtol	mescd	steps	accept	feval	jeval	LU	CPU-time
1.00E-04	3.20	74	66	870	66	74	5.12E-02
1.00E-05	3.76	117	105	1443	105	117	8.40E-02
1.00E-06	3.95	193	177	2769	177	193	1.44E-01
1.00E-07	4.35	374	330	5925	330	374	2.80E-01
1.00E-08	5.02	801	655	12814	655	801	5.84E-01

Table 7: Statistics for the Elastic Beam problem, RADAU5, split 2.

rtol	mescd	steps	accept	feval	jeval	LU	CPU-time
1.00E-04	3.57	66	56	548	56	66	4.68E-02
1.00E-05	3.71	112	96	879	96	112	7.76E-02
1.00E-06	3.76	152	144	1290	144	152	1.12E-01
1.00E-07	4.20	284	260	2603	260	284	2.10E-01
1.00E-08	4.72	517	481	5044	481	517	3.89E-01

Table 8: Statistics for the Elastic Beam problem, RADAU5, split 3.

rtol	mescd	steps	accept	feval	jeval	LU	CPU-time
1.00E-04	3.53	64	54	454	54	64	4.76E-02
1.00E-05	3.67	115	96	810	96	115	8.32E-02
1.00E-06	3.74	154	141	1104	141	154	1.16E-01
1.00E-07	4.17	273	249	1959	249	273	2.04E-01
1.00E-08	4.68	502	456	3654	456	502	3.74E-01

Table 9: Statistics for the Ring Modulator problem, RADAU5.

rtol	mescd	steps	accept	feval	jeval	LU	CPU-time
1.00E-07	4.42	98754	89346	510295	89346	98754	1.37E+00
1.00E-08	5.20	137823	128316	727506	128316	137823	1.92E+00
1.00E-09	5.96	194463	185008	1046747	185008	194463	2.74E+00
1.00E-10	6.75	277830	268414	1525756	268414	277830	3.94E+00
1.00E-11	7.52	399846	390508	2234881	390508	399846	5.71E+00
1.00E-12	8.30	580535	571309	3365783	571309	580535	8.42E+00

Table 10: Statistics for the Ring Modulator problem, RADAU5, split 1.

rtol	mescd	steps	accept	feval	jeval	LU	CPU-time
1.00E-07	4.97	110376	95269	958749	95269	110376	1.54E+00
1.00E-08	5.91	152526	136231	1328822	136231	152526	2.13E+00
1.00E-09	6.93	212686	195982	1855438	195982	212686	2.98E+00
1.00E-10	8.36	301719	283921	2635810	283921	301719	4.23E+00
1.00E-11	8.75	432000	412643	3785978	412643	432000	6.07E+00
1.00E-12	9.05	624708	602385	5524392	602385	624708	8.84E+00

Acknowledgements

The authors want to thanks the anonymous referees, for their comments and suggestions.

References

[1] R. Alexander. Diagonally implicit Runge-Kutta methods for stiff ODE's. *SIAM J. Numer. Anal.* **14** (1977) 1006–1021.

[2] P. Amodio, L. Brugnano. A Note on the Efficient Implementation of Implicit Methods for ODEs. *J. Comput. Appl. Math.* **87** (1997) 1–9.

[3] T.A. Bickart. An efficient solution process for implicit Runge–Kutta methods. *SIAM J. Numer. Anal.* **14** 6 (1977), 1022–1027.

[4] L. Brugnano. Blended Block BVMs (B_3VMs): A Family of Economical Implicit Methods for ODEs. *J. Comput. Appl. Math.* **116** (2000) 41–62.

[5] L. Brugnano, F. Iavernaro, D. Trigiante. Analysis of Hamiltonian Boundary Value Methods (HBVMs) for the numerical solution of polynomial Hamiltonian dynamical systems, 2009. `arXiv:0909.5659v1`

[6] L. Brugnano, F. Iavernaro, D. Trigiante. Hamiltonian Boundary Value Methods (Energy Preserving Discrete Line Methods). *JNAIAM, J. Numer. Anal. Ind. Appl. Math.* **5** 1-2 (2010) 17–37.

[7] L. Brugnano, F. Iavernaro, D. Trigiante. A note on the efficient implementation of Hamiltonian BVMs. *J. Comput. Appl. Math.* **236** (2011) 375–383.

[8] L. Brugnano, F. Iavernaro, D. Trigiante. The Lack of Continuity and the Role of Infinite and Infinitesimal in Numerical Methods for ODEs: the Case of Symplecticity. *Appl. Math. Comput.* **218** (2012) 8053–8063.

[9] L. Brugnano, F. Iavernaro, D. Trigiante. A simple framework for the derivation and analysis of effective one-step methods for ODEs. *Appl. Math. Comput.* **218** (2012) 8475–8485.

[10] L. Brugnano, C. Magherini. Blended Implementation of Block Implicit Methods for ODEs. *Appl. Numer. Math.* **42** (2002) 29–45.

[11] L. Brugnano, C. Magherini. The BiM Code for the Numerical Solution of ODEs. *J. Comput. Appl. Math.* **164-165** (2004) 145–158.

[12] L. Brugnano, C. Magherini. Blended Implicit Methods for solving ODE and DAE problems, and their extension for second order problems. *J. Comput. Appl. Math.* **205** (2007) 777–790.

[13] L. Brugnano, C. Magherini. Recent Advances in Linear Analysis of Convergence for Splittings for Solving ODE problems. *Appl. Numer. Math.* **59** (2009) 542–557.

[14] L. Brugnano, C. Magherini, F. Mugnai. Blended Implicit Methods for the Numerical Solution of DAE Problems. *J. Comput. Appl. Math.* **189** (2006) 34–50.

[15] K. Burrage. A special family of Runge-Kutta methods for solving stiff differential equations. *BIT* **18** (1978) 22–41.

[16] K. Burrage, P.M. Burrage. Low rank Runge-Kutta methods, symplecticity and stochastic Hamiltonian problems with additive noise. *J. Comput. Appl. Math.* **236** (2012) 3920–3930.

[17] J.C. Butcher. On the implementation of implicit Runge-Kutta methods. *BIT* **16** (1976) 237–240.

[18] J.C. Butcher. A transformed implicit Runge-Kutta method. *J. Assoc. Comput Mach.* **26** (1979) 237–240.

[19] J.R. Cash. The integration of stiff initial value problems in ODEs using mod-

ified extended backward differentiation formulae. *Comput. Math. Appl.* **9** (1983) 645–657.

[20] G.J. Cooper. On the implementation of Singly Implicit Runge-Kutta methods. *Math. Comp.* **57**, 196 (1991) 663–672.

[21] G.J. Cooper. Some linear stability results for iterative schemes for implicit Runge-Kutta methods. *BIT* **36** (1996) 77–85.

[22] G.J. Cooper, J.C. Butcher. An iteration scheme for implicit Runge-Kutta methods. *IMA J. Numer. Anal.* **3** (1983) 127–140.

[23] G.J. Cooper, R. Vignesvaran. A scheme for the implementation of implicit Runge-Kutta methods. *Computing* **45** (1990) 321–332.

[24] S. González-Pinto, S. González-Conceptión, J.I. Montijano. Iterative Schemes for Gauss Methods. *Computers Math. Appl.* **27** (1994) 67–81.

[25] S. González-Pinto, J.I. Montijano, L. Rández. Iterative schemes for three-stage implicit Runge-Kutta methods. *Appl. Numer. Math.* **17** (1995) 363–382.

[26] S. González-Pinto, J.I. Montijano, L. Rández. Improving the efficiency of the iterative schemes for implicit Runge-Kutta methods. *J. Comput. Appl. Math.* **66** (1996) 227–238.

[27] S. González-Pinto, S. Pérez-Rodríguez, J.I. Montijano, Implementation of High-Order Implicit Runge-Kutta Methods. *Computers Math. Appl.* **41** (2001) 1009–1024.

[28] S. González-Pinto, S. Pérez-Rodríguez, R. Rojas-Bello. Efficient iterations for Gauss methods on second order problems. *J. Comput. Appl. Math.* **189** (2006) 80–97.

[29] E. Hairer, G. Wanner. *Solving Ordinary Differential Equations II. Stiff and Differential-Algebraic Equations,* 2^{nd} *Ed.* Springer, Berlin, 2002.

[30] E. Hairer, G. Wanner, Stiff differential equations solved by Radau methods, *J. Comput. Appl. Math.* **111**, 1-2 (1999) 93–111.

[31] P.J. van der Houwen, B.P. Sommeijer. Iterated Runge–Kutta methods on parallel computers. *SIAM J. Sci. Stat. Comput.* **12**, 5 (1991) 1000–1028.

[32] P.J. van der Houwen, J.J.B. de Swart. Triangularly implicit iteration methods for ODE-IVP solvers. *SIAM J. Sci. Comput.* **18** (1997) 41–55.

[33] P.J. van der Houwen, J.J.B. de Swart. Parallel linear system solvers for Runge-

Kutta methods. *Adv. Comput. Math.* **7**, 1-2 (1997) 157–181.

[34] F. Iavernaro, F. Mazzia. Solving ordinary differential equations by generalized Adams methods: properties and implementation techniques. *Appl. Numer. Math.* **28**, 2-4 (1998) 107–126.

[35] J.J.B. de Swart. A simple ODE solver based on 2-stage Radau IIA. *J. Comput. Appl. Math.* **84** (1997) 227–280.

[36] *Test Set for IVP Solvers*:
`http://www.dm.uniba.it/~testset/testsetivpsolvers/`