




A MATLAB implementation of TASE-RK methods

D. Conte ¹, G. Pagano ^{1,*}, and B. Paternoster ¹

¹*Department of Mathematics, University of Salerno*

Received: 09/05/2023 – Published: 23/07/2024

Communicated by: R. Cavoretto and A. De Rossi

Abstract

In this paper, we analyze theoretical and implementation aspects of Time-Accurate and highly-Stable Explicit Runge-Kutta (TASE-RK) methods, which have been recently introduced by Bassenne et al. (2021) [5], for the numerical solution of stiff Initial Value Problems (IVPs). These methods are obtained by combining explicit RK schemes with suitable matrix operators, called TASE operators, involving in their expression a matrix J related to the Jacobian of the differential problem to be solved. By analyzing the formulation and order conditions of TASE-RK methods, we show that they can be interpreted as particular linearly implicit RK schemes, and that their consistency properties are independent of the choice of J . Using this information, we propose a MATLAB implementation of TASE-RK methods, which makes use of matrix factorizations and allows setting J according to user preferences.

Keywords: RK methods, TASE preconditioners, TASE-RK methods, MATLAB code, stiff problems (MSC2020: 65L04, 65L06, 65M06, 65Y99)

1 Introduction

In this manuscript, we focus on the numerical solution of IVPs of the form

$$\begin{cases} y'(t) = f(t, y(t)), \\ y(t_0) = y_0, \end{cases} \quad t \in [t_0, t_e], \quad f : \mathbb{R} \times \mathbb{R}^d \rightarrow \mathbb{R}^d, \quad (1)$$

characterized by severe stiffness, usually arising from the spatial semi-discretization of Partial Differential Equations (PDEs) in several applications. Stiffness is a well-known characteristic of differential equations, and several definitions have been given over the years to formalize

* Corresponding author: gpagano@unisa.it

this concept, see, e.g., [10, 11]. Roughly speaking, a system of differential equations is stiff when an explicit numerical method is forced to use very small discretization steps in order to furnish an accurate solution, thus becoming totally inefficient.

Research in the field of efficient methods for solving problems of this type has developed a lot over the years, and still continues. Indeed, most of the models of differential equations that derive from application in several contexts, e.g., corrosion [4, 15], biology [19], chemistry [8, 7], physics [3, 6, 20, 21], are characterized by severe stiffness. The first methods that have been proposed to deal with stiffness are the implicit ones, among which the most famous are the fully implicit RK schemes (e.g. the Gauss-Legendre and RADAU formulas [11, 28, 29]). Fully implicit methods manage to be particularly stable with quite large values of the discretization step. However, fully implicit RK methods are particularly expensive since they require the solution of systems of non-linear equations (of the size of the problem times the number of stages) at each time step, and this constitutes an obstacle especially for problems of large dimensions, such as semi-discretized PDEs. For this reason, several implementation procedures have been proposed in the scientific literature to optimize the efficiency of implicit methods by reducing the number of required operations or the size of the underlying non-linear system, see, e.g., [26]. To reduce the complexity of implicit methods, alternative RK schemes have been formulated, such as DIRK (Diagonally Implicit RK) (see, e.g., [27] and references therein contained), or IMEX (IMplicit EXplicit) (see, e.g. [14, 13] and references therein contained). Furthermore, particularly efficient and stable methods for stiff problems are the so-called linearly implicit RK schemes, which arise for example from a linearization of DIRK. Such methods require the solution of a fixed number of linear systems at each step.

The most famous linearly implicit RK schemes are the Rosenbrock and W-methods, see, e.g., [23, 25, 24, 34]. Moreover, recently Bassenne et al. [5] proposed a new class of RK methods, called TASE-RK methods. These numerical schemes have been subsequently improved by Calvo et al. [12]. As pointed out in [5, Introduction], the idea of the former is based on the fact that a user is not a-priori aware of the severity of the stiffness of a differential problem. Thus, a convenient approach may be to always start using an explicit RK method, which is very simple and fast to program. Then, if the numerical results are not good, to avoid using another code and reprogramming a new method, the user can keep the one already applied by pre-conditioning the problem to solve. In this way the stiffness of the problem is moderated and therefore the probability that the explicit RK method works well increases. TASE-RK methods are very interesting and promising, as shown by the large number of scientific articles that have been produced based on them [2, 18, 17, 22, 30, 33, 36]. Some of these manuscripts show that TASE operators are very efficient also when used with other classes of numerical schemes, such as peer methods [1, 16, 18, 31, 32, 35].

In this manuscript, we focus on the implementation aspects of TASE-RK methods. In particular, by analyzing their formulation, we express them in such a way that their implementation and also the study of the related properties of accuracy and stability simplify. Indeed, we revise TASE-RK methods as linearly implicit numerical schemes, and compute an efficient solution of the underlying linear systems. These systems involve matrices that depend on the Jacobian $J_f = f_y(t, y(t))$ of the differential problem. However, since the consistency analysis shows that the TASE-RK methods preserve their order regardless of the choice of these matrices, we propose an implementation that allows the user to fix them in a convenient way. Finally, we show two examples of use of the proposed MATLAB code.

Summarizing, this paper is organized as follows: in Section 2 we recall the original TASE-RK methods and formulate them as linearly implicit RK methods; in Section 3 we discuss the related properties of accuracy and stability; in Section 4 we show and explain the improvement

of TASE-RK methods performed by Calvo et al.; in Section 5 we propose and describe a MATLAB function for implementing TASE-RK methods; in Section 6 we show an example of use through two numerical tests concerning a system of Ordinary Differential Equations (ODEs) and a semi-discretized PDE; finally, some conclusions are drawn in Section 7.

2 Formulation

Let us fix, from now on, the discrete grid $\{t_n = t_0 + nh; n = 0, \dots, N; t_N = t_e\}$, $h > 0$. The idea of derivation of the TASE-RK methods arises from the following observations.

First, consider the implicit Euler method for solving the problem $y'(t) = Jy(t)$, with J representing a generic matrix of order d :

$$y_{n+1} = y_n + hJy_{n+1}. \quad (2)$$

Note that the method (2) can be rewritten as $(I_d - hJ)y_{n+1} = y_n$, where I_d indicates the Identity matrix of order d . Assuming the matrix $I_d - hJ$ to be invertible, we get

$$y_{n+1} = (I_d - hJ)^{-1}y_n \iff y_{n+1} = y_n + ((I_d - hJ)^{-1} - I_d)y_n.$$

Furthermore, using that

$$(I_d - hJ)^{-1} - I_d = (I_d - hJ)^{-1}[I_d - (I_d - hJ)] = (I_d - hJ)^{-1}hJ,$$

we can finally write

$$y_{n+1} = y_n + hT_1(hJ)Jy_n, \quad \text{with } T_1(hJ) = (I_d - hJ)^{-1}. \quad (3)$$

Note that the numerical scheme (3) corresponds to the explicit Euler method applied to the problem $y'(t) = T_1(hJ)Jy(t)$. Therefore, solving

$$y'(t) = Jy(t), \quad J \in \mathbb{R}^{d,d}, \quad (4)$$

with implicit Euler method is equivalent to solving

$$y'(t) = T_1(hJ)Jy(t), \quad T_1 : \mathbb{R}^{d,d} \rightarrow \mathbb{R}^{d,d}, \quad T_1(A) := (I_d - A)^{-1} \in \mathbb{R}^{d,d}, \quad (5)$$

using explicit Euler, which in principle has bad stability properties. However, the preconditioning made to the vector field of the problem (4) by means of the matrix T_1 definitely improves the stability of explicit Euler method, since we get implicit Euler.

Using the above observations, Bassenne et al. have proposed a general setting for constructing new RK schemes with s stages and order $p = s$ suitable for stiff problems. The general approach consists of the following two main steps.

First, the differential problem (1) is modified as follows:

$$\begin{cases} u'(t) = T_p(hJ_f)f(t, u(t)), & T_p : \mathbb{R}^{d,d} \rightarrow \mathbb{R}^{d,d}, \\ u(t_0) = y_0. \end{cases} \quad (6)$$

Here, T_p is a matrix operator which depends on the product between the step-size h and the Jacobian $J_f = f_y(t, y(t))$ of the problem, where in J_f for simplicity of notation we do not indicate

the time dependence. The only property required for T_p is that it must be an approximation of order p of the Identity. This means that

$$T_p(hJ_f) = I_d + O(h^p). \tag{7}$$

Subsequently, the perturbed problem (6) is solved through an explicit RK method of order p .

Therefore, more conveniently, we can express the TASE-RK methods directly in the following way:

$$\begin{cases} Y_{n,i} = y_n + h \sum_{j=1}^{i-1} a_{ij} T_p(hJ_n) f(t_n + c_j h, Y_{n,j}), & i = 1, \dots, s, \\ y_{n+1} = y_n + h \sum_{j=1}^s b_j T_p(hJ_n) f(t_n + c_j h, Y_{n,j}). \end{cases} \tag{8}$$

Here, $Y_{n,j} \approx y(t_n + c_j h)$, $y_n \approx y(t_n)$, and $A = (a_{ij})$, $b = (b_j)$, $c = (c_j)$ represent the coefficients of the underlying explicit RK method. Note that, in the discrete setting, the exact Jacobian J_f is replaced by $J_n = f_y(t_n, y_n)$. Hence, TASE-RK methods require the Jacobian to be updated at each integration time step.

The TASE operator T_p proposed by Bassenne et al. is a natural extension of the function T_1 in Equation (5). In particular, adding in T_1 the dependency on a real positive parameter α , the TASE operator of order one reads

$$T_1(\alpha, hJ_f) = (I_d - \alpha hJ_f)^{-1}, \quad \alpha > 0.$$

Using Richardson extrapolation, Bassenne et al. have then recursively defined a generic family of TASE operators of order p , as follows:

$$T_p(\alpha, hJ_f) = \begin{cases} (I_d - \alpha hJ_f)^{-1}, & \text{if } p = 1, \\ \frac{2^{p-1}}{2^{p-1} - 1} T_{p-1}(\alpha/2, hJ_f) - \frac{1}{2^{p-1} - 1} T_{p-1}(\alpha, hJ_f), & \text{if } p \geq 2. \end{cases} \tag{9}$$

The TASE operator T_p (9) can also be expressed as

$$T_p(\alpha, hJ_f) = \sum_{k=0}^{p-1} \beta_{p,k} (2^k - \alpha hJ_f)^{-1}, \tag{10}$$

where the coefficients $\beta_{p,k}$ must be suitably fixed (see [5, Table 2]).

From the formulation (8), and from the expression of the TASE operator (10), it is clear that TASE-RK methods are linearly implicit numerical schemes. In particular, these methods require the Jacobian to be updated at each step, and the solution of p linear systems depending on J_n for each stage $Y_{n,i}$, $i = 2, \dots, s$ (since $Y_{n,1} = y_n$, we do not take into account the first stage). Furthermore, there are p extra linear systems concerning the computation of the advancing solution. Thus, TASE-RK methods require the solution of sp linear systems at each step.

3 Consistency and stability analysis

In this section, we analyze the consistency and stability of TASE-RK methods.

We start by showing below that a TASE-RK scheme has the same order as the underlying explicit RK method.

Theorem 3.1. [5, 12] *Let us consider an explicit RK method of order p , and assume that the TASE operator T_p satisfies the property (7). Then, the corresponding TASE-RK method (8) is consistent of order p .*

This theorem is quite natural by observing that the exact solutions $y(t), u(t)$, of the original and perturbed problems (1), (6), respectively, satisfy $\|y(t) - u(t)\| = O(h^p)$, thanks to property (7). Using an explicit RK method of order p for the perturbed problem, we obtain that $\|u(t_n) - u_n\| = O(h^p)$, where $u_n \approx u(t_n)$, for each n . With u_n we here denote the numerical solution of the perturbed problem through the explicit RK method, i.e. the solution of the TASE-RK method. Therefore, it holds that $\|y(t_n) - u_n\| = O(h^p)$, i.e. the TASE-RK method has order p .

Several interesting observations can be made starting from Theorem 3.1.

We first write the Taylor series expansion of the TASE operator T_p (9) proposed by Bassenne et al. as follows:

$$T_p(\alpha, hJ_f) = I_d + Q_p(hJ_f)^p + O(h^{p+1}).$$

By making simple calculations, it can be shown that $Q_p = \alpha^p/c$, where c is a known positive constant whose value depends on p .

Remark 3.1. The smaller $|Q_p|$ is, the more T_p approximates the Identity matrix in an accurate way, and therefore the perturbed problem (6) gets closer to the initial one (1). Indeed, in the manuscripts [5, 12] it is observed that the smaller $|Q_p|$ is, the lower the error provided by the TASE-RK methods is.

Remark 3.2. In the manuscripts [5, 12], TASE-RK methods of order $p = s (\leq 4)$ have been derived. This choice allows to attain the maximum possible order using the minimum number of stages and simplifies the study of linear stability, as discussed below.

Now, we analyze the stability properties of TASE-RK methods. It is known that the stability function of explicit s -stage RK methods with order $p = s \leq 4$ is independent of their coefficients. In particular, it can be expressed as follows:

$$R(z) = 1 + z + \dots + \frac{1}{p!}z^p.$$

Here, $z = h\lambda$, where λ is a complex parameter with $\text{Re}(\lambda) < 0$ associated with the classical test equation $y'(t) = \lambda y(t)$. The stability function of TASE-RK methods can be easily derived from it. Indeed, considering the perturbed problem $y'(t) = T_p(\alpha, h\lambda)\lambda y(t)$, we get the following stability function:

$$RT_p(\alpha, z) = 1 + zT_p(\alpha, z) + \dots + \frac{1}{p!}(zT_p(\alpha, z))^p. \tag{11}$$

Note that for TASE-RK methods with order $p = s$ the stability function is independent of the coefficients of the underlying explicit RK scheme. The only parameter on which the stability function depends is that of the TASE operator α . Therefore, adequately fixing the free parameter α of T_p , using TASE technique it is possible to improve the stability properties of any explicit RK method with order $p = s \leq 4$.

Remark 3.3. Property (7) holds for the operator T_p (9) for any value of the parameter α and matrix J_f . Thus, according to Remark 3.1, the free parameter α can be set to minimize $|Q_p|$, in order to have a small error. Moreover, since the function RT_p (11) depends on α , this parameter can be determined in order to get A -stability (or at least $A(\theta)$ -stability) for the corresponding TASE-RK method. See, e.g., [11, p. 230] for the stability definitions. Furthermore, we can

choose a generic matrix other than J_n in the formulation of TASE-RK methods without altering their order of consistency; however, good stability properties are preserved provided that J_n is a suitable approximation of the Jacobian.

In the paper [5], Bassenne et al. have set, for the cases $p = s = 2, 3, 4$, the α parameter in order to obtain a good balance between the minimum $|Q_p|$ value, and the best possible stability properties for the corresponding TASE-RK method. The related results are reported in Table 1.

Table 1: Properties of TASE-RK methods in correspondence of the values of α proposed in the manuscript [5].

$p = s$	Stability properties	$ R(\infty) $	$ Q_p $	α
2	A-stability	1	1.13	1.5
2	Strong A-stability	0.5	4.50	3
3	$A(\theta)$ -stability, $\theta = 89.31^\circ$	1	2.70	2.7858
4	$A(\theta)$ -stability, $\theta = 88.36^\circ$	1	13.14	5.3854

4 Improved TASE-RK methods

The TASE-RK methods have been improved by Calvo et al., who in the paper [12] proposed the following generalization of the family (9):

$$T_p(\boldsymbol{\alpha}, hJ_f) = \sum_{j=1}^p \gamma_j (I_d - \alpha_j hJ_f)^{-1}, \tag{12}$$

$$\gamma_j = \left(\frac{1}{\alpha_j}\right)^{p-1} / \prod_{k \neq j} \left(\frac{1}{\alpha_j} - \frac{1}{\alpha_k}\right), \quad \boldsymbol{\alpha} = (\alpha_1, \dots, \alpha_p) \in \mathbb{R}^p.$$

Here, $\alpha_j > 0$ and $\alpha_j \neq \alpha_k$ for all $j \neq k$. It can be easily shown that the TASE family (12) generalizes the one given by Bassenne et al., which can be derived using $\alpha_j = \alpha/2^{j-1}$, $j = 1, \dots, p$. By construction it holds that (see [12, Eqs. (6)-(7)])

$$T_p(\boldsymbol{\alpha}, hJ_f) = I_d + Q_p (hJ_f)^p + O(h^{p+1}) \quad \text{with } Q_p = \prod_{j=1}^p \alpha_j.$$

The motivation that led Calvo et al. to this generalization is based on the fact that the first family of TASE operators depends on a single free parameter α . As seen in the previous section, α should be set to obtain the optimal balance between the minimum value of the error constant $|Q_p|$ and the best stability of the corresponding TASE-RK method. However, the results in Table 1 are not fully satisfactory, mainly because it is not possible to achieve strongly $A(\theta)$ -stable or even $L(\theta)$ -stable methods for $s = p = 3, 4$. For problems with high stiffness, these properties are very important.

Thanks to this generalization, there are p ($p > 1$) free parameters α_j to fix for stability and accuracy reasons, and therefore more possibilities to improve TASE-RK methods. We note that obviously for this generalization the stability function becomes

$$RT_p(\boldsymbol{\alpha}, z) = 1 + zT_p(\boldsymbol{\alpha}, z) + \dots + \frac{1}{p!} (zT_p(\boldsymbol{\alpha}, z))^p.$$

Furthermore, the computational cost of the TASE-RK methods (8) with the new T_p (12) remains the same. Indeed, it is still required the solution of sp linear systems at each step depending on J_n . In Table 2, we report the properties of the TASE-RK methods in correspondence of the α_j parameters of the operator T_p (12) proposed by Calvo et al. in the paper [12].

Table 2: Properties of TASE-RK methods in correspondence of the values of α proposed in the manuscript [12].

$p = s$	Stability properties	$ R(\infty) $	$ Q_p $	α
2	Strong A -stability	0.5	4.50	(3, 1.50)
3	$L(\theta)$ -stability, $\theta = 89.02^\circ$	0	6.88	(2.315, 1.880, 1.582)
4	Strong $A(\theta)$ -stability, $\theta = 87.34^\circ$	0.270	44.32	(3.940, 2.451, 2.227, 2.061)

By comparing Tables 1, 2, it is clear that the error constant $|Q_p|$ is of the same order of magnitude for both families of operators (9), (12), for $p = 2, 3, 4$. Also the angle of $A(\theta)$ -stability is more or less similar for the versions proposed in the two tables. However, TASE-RK methods with operator (12) have much better strong stability properties. Therefore, from now on we refer directly to the operators proposed by Calvo et al. (12).

5 MATLAB code and computational effort

From the analysis made in the previous sections, it is clear that the consistency of the TASE-RK methods is independent of the choice of J_n . Indeed, for TASE-RK methods, to get order p , it suffices that the underlying explicit RK method has order p and property (7) holds (see Theorem 3.1).

As observed in Remark 3.3, stability reasons require J_n to be a suitable approximation of the Jacobian, obtained, for example, by fixing J_n as a constant matrix (thus avoiding updating J_n at all points of the time grid). For this reason, we report below an implementation of TASE-RK methods allowing the user to choose whether to update the Jacobian at each step, or fix it as a constant matrix. Furthermore, with this implementation, methods can be applied with any number of stages and with a TASE operator having a generic number of terms (therefore also for $p \neq s$). Finally, referring to formulation (8) of TASE-RK methods, we exploit the MATLAB `lu` function for factorizing the coefficient matrices and then the command `backslash` for solving the required linear systems by means of the forward/backward substitution. At the end of this section, we also provide a brief analysis of the computational effort of TASE-RK methods with and without exact Jacobian, showing that the choice of constant J_n allows to remarkably reduce the number of required operations.

5.1 Function TASERK.m

Let us describe below the input and output arguments, together with the employed auxiliary MATLAB functions, of the code `TASERK.m`, which allows to apply the TASE-RK methods proposed by Calvo et al. [12] to a differential problem of the type (1) chosen by the user.

Input arguments

- N - integer scalar

Number of (equally spaced) discrete time intervals into which the user decides to subdivide the continuous grid $[t_0, t_e]$.

- `tspan` - double array
Row vector of length two containing the first and last grid points t_0, t_e , respectively.
- `y0` - double array
Column vector with the initial condition $y_0 \in \mathbb{R}^d$.
- `Fun` - function handle
Function which returns the vector field f , evaluated at (τ, y) , of the problem (1) that the user wants to solve; $\tau \in \mathbb{R}$, $y \in \mathbb{R}^d$, constitute the input arguments of `Fun`, and the column vector $f(\tau, y) \in \mathbb{R}^d$ is the output.
- `Jac` - function handle
Function which returns the Jacobian matrix of the problem (1) that the user wants to solve, evaluated at a point (τ, y) , or a suitable fixed approximation of J_f ; in the first case, $\tau \in \mathbb{R}$, $y \in \mathbb{R}^d$, constitute the input arguments of `Jac`, and the matrix $f_y(\tau, y) \in \mathbb{R}^{d,d}$ is the output.
- `Method` - integer array
Array with the TASE-RK methods to apply; in particular:
 - 20 corresponds to the TASE-RK with $s = p = 2$, using the midpoint rule as underlying explicit RK;
 - 30 corresponds to the TASE-RK with $s = p = 3$, using the Ralston's method as underlying explicit RK;
 - 40 corresponds to the TASE-RK with $s = p = 4$, using the Kutta's method as underlying explicit RK.

For example, if we want to use methods 20 and 30, then `Method=[20,30]`. We will later show an example where we apply all the TASE-RK methods using the same main program.

- `jacup` - integer scalar
Parameter which is equal to 0 if the user wants to use constant J_n , 1 otherwise.

Output arguments

- `yT` - double array
Column vector of length d with the numerical solution computed by the chosen TASE-RK method at last grid point t_e .
- `y` - double array
Matrix of size $d \times (N + 1)$ having in column $n + 1$ the numerical solution y_n computed by the chosen TASE-RK method.
- `τ` - double array
Row vector of length $N + 1$ with all the discrete grid points $\{t_n = t_0 + nh; n = 0, \dots, N; t_N = t_e\}$.

- CPUtime - double scalar
Total CPU time in seconds taken by the chosen TASE-RK method.

Auxiliary MATLAB functions

- backslash
Computes $A^{-1}B$, where A and B are input matrices.
- cputime
Returns the current CPU time in seconds.
- eye
Returns the Identity matrix of required dimension.
- kron
Computes, in general, the Kronecker product of two matrices.
- length
Returns the length of a vector.
- linspace
Generates a row vector of linearly equally spaced points.
- lu
Computes the LU factorization of a matrix.
- ones
Generates a matrix of the required size with all elements equal to one.
- sum
Computes the sum by rows or columns of the elements of a matrix.
- zeros
Generates a matrix of the required size with all elements equal to zero.

Below, we report the MATLAB function `TASERK.m`.

Code 1: function `TASERK.m`.

```

1 function [yT,y,t,CPUtime] = TASERK(N,tspan,y0,Fun,Jac,Method,jacup)
2
3 % Fixing explicit RK tableau and TASE operator coefficients
4 switch Method
5     case 20 % 20-midpoint method of order s=p=2
6         s = 2; p = 2; alpha = [3 3/2];
7         A = [0 0;1/2 0]; c = [0 1/2]; b = [0 1];
8
9     case 30 % 30-Ralston method of order s=p=3
10        s = 3; p = 3; alpha = [2.31469 1.87961 1.58222];
11        A = [0 0 0;1/2 0 0;0 3/4 0];
12        c = [0 1/2 3/4]; b = [2/9 1/3 4/9];
13
14        case 40 % 40-Kutta method of order s=p=4
15            s = 4; p = 4; alpha = [3.939556 2.450558 2.227083 2.061235];
16            A = [0 0 0 0;1/2 0 0 0;0 1/2 0 0;0 0 1 0];
17            c = [0 1/2 1/2 1]; b = [1/6 1/3 1/3 1/6];
18    end
19
20 % Computation of gamma
21 alphas = 1./alpha; gamma = [];
22 for i = 1:p

```

```

23     prod = 1;
24     for j = 1:p
25         if i ~= j
26             prod = prod*(alpham1(i)-alpham1(j));
27         end
28     end
29     gamma(i) = alpham1(i)^(p-1)/prod;
30 end
31
32 % Initialization
33 t = linspace(tspan(1),tspan(2),N+1); % t: discrete time grid
34 h = (tspan(2)-tspan(1))/N; % h: constant time step-size
35 d = length(y0); % d: dimension of the problem
36 Id = eye(d); % Id: Identity matrix of order d
37 n = 1; % n: index representing the current step
38 y = y0;
39
40 if (jacup == 0) % If we want a 'constant Jacobian'
41     C = cputime;
42     Jn = Jac();
43     for l = 1:p % Compute, outside the loop, the LU factorizations of Id-alpha(l)*(h*Jn)
44         [Ll(1:d, (l-1)*d+1:l*d), Ul(1:d, (l-1)*d+1:l*d)] = lu(Id-alpha(l)*(h*Jn));
45     end
46     for n = 2:N+1
47         Y = zeros(d,s); % Block matrix with stages in columns
48         Y(:,1) = y(:,n-1);
49         f = zeros(d,s); % Block matrix with h*fi in columns
50         F = zeros(d,s); % Block matrix with Tp*h*fi in columns
51         for i = 1:s-1 % Compute all the stages
52             f(:,i) = h*Fun(t(n-1)+h*c(i),Y(:,i));
53             for l = 1:p
54                 F(:,i) = F(:,i) + gamma(l)*(Ul(1:d, (l-1)*d+1:l*d) \ (Ll(1:d, (l-1)*d+1:l*d)\f
55                 (:,i)));
56                 end
57                 Y(:,i+1) = y(:,n-1) + sum(kron(A(i+1,:),ones(d,1)).*F,2);
58             end
59             f(:,s) = h*Fun(t(n-1)+h*c(s),Y(:,s));
60             for l = 1:p
61                 F(:,s) = F(:,s) + gamma(l)*(Ul(1:d, (l-1)*d+1:l*d) \ (Ll(1:d, (l-1)*d+1:l*d)\f(:,s)
62                 );
63             end
64             y(:,n) = y(:,n-1) + sum(kron(b,ones(d,1)).*F,2);
65         end
66         Cf = cputime;
67     elseif (jacup == 1) % If we want exact Jacobian
68         C = cputime;
69         for n = 2:N+1
70             Jn = Jac(t(n-1),y(:,n-1)); % Update Jn at each step
71             for l = 1:p % Compute, at each step, the LU factorizations of Id-alpha(l)*(h*Jn)
72                 [Ll(1:d, (l-1)*d+1:l*d), Ul(1:d, (l-1)*d+1:l*d)] = lu(Id-alpha(l)*(h*Jn));
73             end
74             Y = zeros(d,s); % Block matrix with the stages in column
75             Y(:,1) = y(:,n-1);
76             f = zeros(d,s); % Block matrix with h*fi in column
77             F = zeros(d,s); % Block matrix with Tp*h*fi in column
78             for i = 1:s-1 % Compute all the stages
79                 f(:,i) = h*Fun(t(n-1)+h*c(i),Y(:,i));
80                 for l = 1:p
81                     F(:,i) = F(:,i) + gamma(l)*(Ul(1:d, (l-1)*d+1:l*d) \ (Ll(1:d, (l-1)*d+1:l*d)\f
82                     (:,i)));
83                     end
84                     Y(:,i+1) = y(:,n-1) + sum(kron(A(i+1,:),ones(d,1)).*F,2);
85                 end
86                 f(:,s) = h*Fun(t(n-1)+h*c(s),Y(:,s));
87                 for l = 1:p
88                     F(:,s) = F(:,s) + gamma(l)*(Ul(1:d, (l-1)*d+1:l*d) \ (Ll(1:d, (l-1)*d+1:l*d)\f(:,s)
89                     );
90                 end
91                 y(:,n) = y(:,n-1) + sum(kron(b,ones(d,1)).*F,2);
92             end
93         Cf = cputime;
94     end
95 end

```

```

92 CPUtime = Cf - C;
93 yT = y(:,end);
94 end

```

5.2 Description of the code

Let us describe the lines of code of the function `TASERK.m`.

- From line 3 to line 18: we select the TASE-RK methods chosen by the user; method 20 corresponds to the case $s = p = 2$, using the midpoint rule as underlying explicit RK; method 30 corresponds to the case $s = p = 3$, using the Ralston's method as underlying explicit RK; method 40 corresponds to the case $s = p = 4$, using the Kutta's method as underlying explicit RK.
- From line 20 to line 30: we compute the values of γ_j , storing them in a vector `gamma`, starting from the alpha (α) vector according to Equation (12).
- From line 32 to line 38: we define the time grid `t`, the step-size `h`, the Identity matrix `Id` of size `d` (i.e. the size of the problem); we also initialize the time step `n`, the matrix `y`, which, at the end, will contain the numerical solution (in the columns) at all the discrete points, and the CPU time `C`.
- From line 40 to line 64: if `jacup=0`, i.e. we want to fix J_n avoiding updating it at each step, we first call the function `Jac.m`, which returns in this case a suitable constant approximation of J_n ; then we apply the TASE-RK method as described below.
- From line 43 to line 45: using the `lu` command, we compute the LU factorizations of the matrices $I_d - \alpha_j h J_n$, $j = 1, \dots, p$, whose summed inverses define the TASE operator T_p according to Equation (12); we allocate all the lower and upper triangular matrices L and U of size d thus obtained in successive blocks of the matrices `L1` and `U1`, respectively, which have dimension $d \times (pd)$; note that the matrices `L1` and `U1` are only computed here, and are not updated within the method being $I_d - \alpha_j h J_n$, $j = 1, \dots, p$, constants.
- From line 46 to line 64: we compute the numerical solution at all grid points using the TASE-RK method; we explain below the operations performed here.
 - From line 47 to line 50: we define the matrix `Y` which, at the end of the current step, will contain all the stages $Y_{n,j}$, $j = 1, \dots, s$, in columns; the matrix `f` which, at the end of the current step, will contain all the function evaluations $hf(t_n + c_j h, Y_{n,j})$, $j = 1, \dots, s$, in columns; the matrix `F` which, at the end of the current step, will contain the products between T_p and $hf(t_n + c_j h, Y_{n,j})$, $j = 1, \dots, s$, in columns.
 - From line 51 to line 61: we compute all the stages storing them in `Y`; note that the products between T_p and $hf(t_n + c_j h, Y_{n,j})$ are calculated by solving linear systems of the form $\tilde{A}x = \tilde{b}$ by means of the `backslash` command, with $\tilde{A} = I_d - \alpha_j h J_n$, $\tilde{b} = hf(t_n + c_j h, Y_{n,j})$, through the LU factorizations of the coefficient matrices stored in the arrays `L1` and `U1`.
 - Line 62: we compute the solution y_{n-1} , which we store in the `n`-th column of the matrix `y`; the index `n` is shifted by one with respect to n as the initial condition y_0 is stored in the first column of `y` (in MATLAB, array indexes start at 1).

- From line 65 to line 90: if $\text{jacup} = 1$, we update J_n and therefore the LU factorizations of the matrices $I_d - \alpha_j h J_n$, $j = 1, \dots, p$, at each step; in this case, the function `Jac.m` returns the exact Jacobian of the problem evaluated at the desired grid point; note that, of course, lines 72–89 correspond to lines 47–64 (in fact, the only change with respect to the case $\text{jacup} = 0$ is due to the fact that J_n and the matrix factorizations must be updated at each step and therefore appear inside the loop).
- From line 92 to line 94: we compute the total CPU time employed by the method and store in the vector `yT` the numerical solution at the final point of the grid.

Note that, of course, setting J_n constant, the number of operations required by the method drops drastically. In fact, for:

- $\text{jacup} = 0$, we have to compute only p LU factorizations, then using them in the solution of sp linear systems per step; thus, at the end we have p LU factorizations plus $N(sp)$ linear systems;
- $\text{jacup} = 1$, we have to compute p LU factorizations per step, using them in the solution of sp linear systems; thus, at the end we have Np LU factorizations plus $N(sp)$ linear systems.

Obviously, we underline that the cost of solving a linear system with already factorized matrix is considerably reduced. In particular, given d the size of the problem, an LU factorization costs $O(d^3/3)$, and solving a linear system with an already factorized matrix costs $O(d^2)$. Therefore, the approach we propose in the code is especially convenient for problems of big dimensions, and when a large number N of time grid points (i.e. small h) is required.

6 Examples of application

We report below two examples of application of the code `TASERK.m`. The first is quite simple and concerns a system of ODEs. The second concerns the numerical solution of the famous Burgers' PDE.

6.1 Euler's problem

In this subsection, we show the application of the function `TASERK.m` in solving the well known Euler's problem, given by the following system of coupled ODEs:

$$\begin{cases} \frac{dy_1}{dt} = -2y_2y_3, \\ \frac{dy_2}{dt} = \frac{5}{4}y_1y_3, \\ \frac{dy_3}{dt} = -\frac{1}{2}y_1y_2, \end{cases} \quad t \in [t_0, t_e]. \quad (13)$$

This model is related to the rotational motion of solid bodies. We take $t_0 = 0$, $t_e = 10$, and $y_0 = (1, 0, 0.9)$. Easily, note that

$$J_f = \begin{pmatrix} 0 & -2y_3 & -2y_2 \\ \frac{5}{4}y_3 & 0 & \frac{5}{4}y_1 \\ -\frac{1}{2}y_2 & -\frac{1}{2}y_1 & 0 \end{pmatrix}. \quad (14)$$

Since we can use a fixed approximation of the Jacobian to lower the computational cost of TASE-RK methods preserving their order of consistency, if $\text{jacup} = 0$ we impose that the function `Jac` returns the matrix J_f evaluated at the initial point (t_0, y_0) . Then, in this case we evaluate the Jacobian only at the initial grid point, without updating it during the integration.

We report and describe below the main code, which we have called `exampleEuler.m`. In line 2 we define the function `Fun`, which corresponds to `funEuler.m`. In line 3 we choose the TASE-RK methods to use; we apply in this case the method with $p = s = 4$. From line 5 to line 10, we choose the `jacup` parameter and fix the function `Jac` according to its value. From line 13 to line 16, we set the time grid, the initial conditions and the number N of discrete intervals. From line 19 to line 21 we also compute a reference solution by means of the MATLAB `ode15s` function. In line 23 we apply the selected TASE-RK method to the Euler's model. Finally, we print the error at the final time grid point and the employed CPU time.

Code 2: main code `exampleEuler.m`.

```

1  %% Main code: exampleEuler.m
2  Fun = @funEuler;
3  Tmethod = [40]; % Select the TASE-RK method
4  nTmethods = length(Tmethod);
5  jacup = 0; % We want 'constant Jacobian'
6  if (jacup == 0)
7      Jac = @jacEulerfix;
8  elseif (jacup == 1)
9      Jac = @jacEuler;
10 end
11
12 % Initial conditions
13 global y0
14 tspan = [0 10];
15 y0 = [1;0;0.9];
16 N = 5000; % Number of grid intervals
17
18 % Compute a reference solution with ode15s
19 options = odeset('RelTol', 5e-14, 'AbsTol', 5e-14);
20 [tode15s, yode15s] = ode15s(@funEuler, tspan, y0, options);
21 YrefT = yode15s(end, :)';
22
23 [yTTRK, yTRK, t, CPUtimeTRK] = TASERK(N, tspan, y0, Fun, Jac, Tmethod, jacup);
24
25 % Print results
26 format short e
27 errT_TRK = norm(yTTRK-YrefT, inf) % Error
28 CPUtimeTRK % CPU time

```

We also report the functions `funEuler.m`, `jacEulerfix.m`, `jacEuler.m`, respectively, which are recalled in the main algorithm. Obviously, `funEuler.m` returns the (column) vector field f given in Equation (13) evaluated at a point (t, y) . Furthermore, `jacEulerfix.m` returns the Jacobian (14) evaluated at the initial grid point. Finally, `jacEuler.m` returns the exact Jacobian (14) evaluated at a point (t, y) .

Code 3: function `funEuler.m`.

```

1  function yp = funEuler(t, y)
2
3  yp(1) = -2*y(2)*y(3);
4  yp(2) = 5/4*y(3)*y(1);
5  yp(3) = -1/2*y(1)*y(2);
6  yp = [yp(1); yp(2); yp(3)]';
7
8  end

```

Code 4: function `jacEulerfix.m`.

```

1  function J = jacEulerfix()

```

```

2
3 global y0
4 J = [0 -2*y0(3) -2*y0(2);
5       5/4*y0(3) 0 5/4*y0(1);
6       -1/2*y0(2) -1/2*y0(1) 0];
7
8 end

```

Code 5: function `jacEuler.m`.

```

1 function J = jacEuler(t,y)
2
3 J = [0 -2*y(3) -2*y(2);
4       5/4*y(3) 0 5/4*y(1);
5       -1/2*y(2) -1/2*y(1) 0];
6
7 end

```

To conclude this subsection, we also report the outputs obtained.

Code 6: outputs of the main code `exampleEuler.m`.

```

1 >> exampleEuler
2 errT_TRK =
3     3.3776e-08
4 CPUtimeTRK =
5     6.4062e-01

```

6.2 Burgers' equation

In this subsection, we show the application of the function `TASERK.m` in solving the Burgers' equation [6, 9], which can be expressed as follows:

$$\frac{\partial u}{\partial t} = \varepsilon \frac{\partial^2 u}{\partial x^2} - u \frac{\partial u}{\partial x}, \quad (x, t) \in [x_0, X] \times [t_0, t_e]. \quad (15)$$

The function u represents the speed of the fluid at the considered spatial (x) and temporal (t) coordinates, ε is related to a constant physical property of the fluid, generally the viscosity or something similar to it. When the diffusion term is absent, this PDE becomes the inviscid Burgers' equation. Furthermore, Equation (15) can also be expressed in the following conservative form:

$$\frac{\partial u}{\partial t} = \varepsilon \frac{\partial^2 u}{\partial x^2} - \frac{1}{2} \frac{\partial u^2}{\partial x}, \quad (x, t) \in [x_0, X] \times [t_0, t_e]. \quad (16)$$

Here, we consider the numerical solution of Equation (16). Regarding the initial conditions, the spatial semi-discretization and the boundary conditions we mainly refer to [12, Section 3.2]. In particular, we consider as initial conditions the function

$$y(x, 0) = \begin{cases} 1, & x \in [0, \pi], \\ 0, & x \in (\pi, 2\pi]. \end{cases}$$

Regarding the spatial semi-discretization of the equation, we take centred finite differences of order four for both the first and second order spatial derivatives. Then, by fixing the uniform spatial grid $\{x_n = x_0 + m\Delta x; m = 0, \dots, M; x_M = X\}$, the semi-discretized Burgers' equation (16) reads

$$y'(t) = \varepsilon L_1 y(t) - \frac{1}{2} L_2 y(t)^2. \quad (17)$$

By calling with $(d_{-2}, d_{-1}, d, d_1, d_2)$ the significant entries of the sub-, main-, and over-diagonals, L_1 and L_2 are the following pentadiagonal Toeplitz matrices:

$$L_1 = \frac{1}{12\Delta x^2}(-1, 16, -30, 16, -1), \quad L_2 = \frac{1}{12\Delta x}(1, -8, 0, 8, -1).$$

The Jacobian is in this case given by the non-constant matrix

$$J_f = \varepsilon L_1 - L_2 Y(t). \quad (18)$$

Here, $Y(t)$ is a square matrix of size M , having in each column the vector $y(t)$. For semi-discretized PDEs of this form, the diffusion part is mainly responsible for the stiffness of the problem. Thus, in our code, since we can use a fixed approximation of the Jacobian to lower the computational cost while maintaining order of TASE-RK methods, if `jacup=0` the function `Jac` returns the constant matrix εL_1 .

We report and describe below the main code, which we have called `exampleBurgers.m`. Note that, unlike the Euler's problem, we now apply all the schemes reported in Table 2. Furthermore, we perform the numerical integration using several values of N ; in particular we use $N = 2^8, 2^9, \dots, 2^{12}$ time grid points, as can be seen from line 11. From line 13 to line 25, we set the parameters, initial conditions and spatial semi-discretization of Burgers' equation. In the example, we have used $\varepsilon = 1/10$, $M = 32$ spatial grid points, $t_0 = x_0 = 0$, $X = 2\pi$, $t_e = 4$. From line 28 to line 30 we also compute, like before, a reference solution by means of the MATLAB `ode15s` function. From line 32 to line 46 we apply the selected TASE-RK methods to the chosen problem, and store the obtained results in the matrices `errT_TRK`, `CPUtime_TRK`, `pest_TRK`. In particular, the first two matrices contain in column i the absolute errors (at the last time grid point) and CPU times, respectively, of the TASE-RK method with $p = s = i + 1$. Each row corresponds to a different value of N . The first row corresponds to the case $N = 2^8$, the last to $N = 2^{12}$. The matrix `pest_TRK` is constructed in the same way and contains the estimated order of the methods. However, it has one row less since it is not possible to estimate the order for the first value of N .

Code 7: main code `exampleBurgers.m`.

```

1  %% Main code: exampleBurgers.m
2  Fun = @funBurgers;
3  Tmethod = [20 30 40]; % Select the TASE-RK methods
4  nTmethods = length(Tmethod);
5  jacup = 0; % We want 'constant Jacobian'
6  if (jacup == 0)
7      Jac = @jacBurgersfix;
8  elseif (jacup == 1)
9      Jac = @jacBurgers;
10 end
11 inN = 8; finN = 12; % We do the time integration using 2^(inN), ..., 2^(finN) time grid points
12
13 global epsilon M L1 L2
14 epsilon = 1/10; M = 32; % Spatial grid points
15 xspan = [0 2*pi]; tspan = [0 4];
16 Deltax = (xspan(2)-xspan(1))/M; % Spatial step-size
17
18 % Initial conditions
19 y0 = []; y0(1:M/2,1) = ones(M/2,1); y0(M/2+1:M,1) = zeros(M/2,1);
20
21 % Space-discetization: order-four FD and periodic BC
22 e = ones(M,1); r = 1/(12*Deltax^2);
23 L1 = spdiags([-r*e 16*r*e -30*r*e 16*r*e -r*e], -2:2, M, M); L1(1,M-1) = -r; L1(M-1,1) = -r;
    L1(M,2) = -r; L1(2,M) = -r; L1(1,end) = 16*r; L1(end,1) = 16*r;
24 r = 1/(12*Deltax);
25 L2 = spdiags([r*e -8*r*e 0*r*e 8*r*e -r*e], -2:2, M, M); L2(1,M-1) = r; L2(M-1,1) = -r; L2(M
    ,2) = -r; L2(2,M) = r; L2(1,end) = -8*r; L2(end,1) = 8*r;

```

```

26
27 % Compute a reference solution with ode15s
28 options = odeset('RelTol',5e-14,'AbsTol',5e-14);
29 [tode15s,yode15s] = ode15s(@funBurgers,tspan,y0,options);
30 YrefT = yode15s(end,:);
31
32 for nm = 1:nTmethods % We apply all the selected TASE-RK
33     i = 1;
34     for N = 2.^(inN:finN) % For the simulations done
35         [yTTRK,yTRK,t,CPUtimeTRK] = TASERK(N,tspan,y0,Fun,Jac,Tmethod(nm),jacup);
36         errT_TRK(i,nm) = norm(yTTRK-YrefT,inf); % Error
37         cd_TRK(i,nm) = -log10(errT_TRK(i,nm));
38         CPUtime_TRK(i,nm) = CPUtimeTRK; % CPU time
39         i = i + 1;
40     end
41
42     l = length(cd_TRK(:,nm));
43     for i = 2:l % Compute the estimated order
44         pest_TRK(i-1,nm) = (cd_TRK(i,nm)-cd_TRK(i-1,nm))/log10(2);
45     end
46 end
47
48 % Print results
49 format short e
50 errT_TRK
51 CPUtime_TRK
52 format short
53 pest_TRK

```

We also report below the function `funBurgers.m`, which obviously returns the vector field of the semi-discretized Burgers' equation (17). Furthermore, we report the functions `jacBurgersfix.m`, `jacBurgers.m`. The latter computes the exact Jacobian, given in Equation (18). The first returns only the diffusion part.

Code 8: function `funBurgers.m`.

```

1 function yp = funBurgers(t,y)
2
3 global epsilon L1 L2
4 yp = epsilon*L1*y - (1/2)*L2*(y.^2);
5
6 end

```

Code 9: function `jacBurgersfix.m`.

```

1 function J = jacBurgersfix()
2
3 global epsilon L1
4 J = epsilon*L1;
5
6 end

```

Code 10: function `jacBurgers.m`.

```

1 function J = jacBurgers(t,y)
2
3 global epsilon L1 L2 M
4 J = epsilon*L1-L2.*repmat(y,1,M)';
5
6 end

```

Finally, we report below the outputs obtained by running `exampleBurgers.m`. Note that, although we used `jacup=0`, the methods maintain their order, as expected from the consistency analysis done in the paper. This can be seen by looking at the columns of the matrices `errT_TRK` and `pest_TRK`. Indeed, we recall that the first column corresponds to the TASE-RK of order $p = s = 2$, the second to the method of order $p = s = 3$ and the third to the scheme with $p = s = 4$. Looking at the matrix with the CPU times, it is evident that the more the number of

grid points N increases and the more the number of stages is high, the greater the computational effort.

Code 11: outputs of the main code `exampleBurgers.m`.

```

1 >> exampleBurgers
2 errT_TRK =
3   3.2141e-04   2.5591e-05   8.8510e-06
4   8.9912e-05   3.9132e-06   9.0181e-07
5   2.3923e-05   5.4871e-07   7.5195e-08
6   6.1825e-06   7.2968e-08   5.5087e-09
7   1.5724e-06   9.4195e-09   3.7483e-10
8 CPUtime_TRK =
9   3.1250e-02   4.6875e-02   6.2500e-02
10  4.6875e-02   1.0938e-01   1.2500e-01
11  1.4062e-01   1.5625e-01   2.6562e-01
12  2.3438e-01   2.9688e-01   5.0000e-01
13  3.4375e-01   6.5625e-01   9.8438e-01
14 pest_TRK =
15   1.8378   2.7092   3.2949
16   1.9101   2.8342   3.5841
17   1.9521   2.9107   3.7708
18   1.9752   2.9535   3.8774

```

7 Conclusions

In this manuscript, we have analyzed in detail the consistency and stability properties of the recently introduced TASE-RK methods. Taking advantage of this analysis, we have proposed a MATLAB implementation of these methods allowing the user to make a flexible choice of the approximation of J_n to be used. Furthermore, by formulating TASE-RK schemes as linearly implicit methods, we have employed some MATLAB functions to efficiently solve the underlying linear systems, by computing a-priori the LU factorizations of the matrices involved. The proposed implementation is not limited to the cases of TASE-RK methods with $p = s \leq 4$, but can allow the user to test methods with a higher number of stages and such that $p \neq s$.

We were motivated by the growing interest in TASE-RK methods, which are quite promising in solving large stiff problems, especially coming from the spatial semi-discretization of PDEs. The results reported on the Euler’s problem and Burgers’ equation testify the correctness of the theoretical analysis and the functioning of the proposed MATLAB code.

Acknowledgments

The authors are members of the GNCS group. This work is supported by GNCS-INDAM project and by the Italian Ministry of University and Research (MUR), through the PRIN 2020 project (No. 2020JLWP23) “Integrated Mathematical Approaches to Socio–Epidemiological Dynamics” (CUP: E15F21005420006) and the PRIN 2017 project (No. 2017JYCLSF) “Structure preserving approximation of evolutionary problems”.

Declaration of interests

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

References

- [1] A. Abdi, G. Hojjati, Z. Jackiewicz, H. Podhaisky, and M. Sharifi. *On the implementation of explicit two-step peer methods with Runge-Kutta stability*. Appl. Numer. Math., vol. 186, (2023), 213–227. ISSN 0168-9274. URL <http://dx.doi.org/10.1016/j.apnum.2023.01.015>
- [2] L. Aceto, D. Conte, and G. Pagano. *On a generalization of time-accurate and highly-stable explicit operators for stiff problems*. Appl. Numer. Math. URL <http://dx.doi.org/10.1016/j.apnum.2023.04.001>
- [3] S. M. Allen and J. W. Cahn. *Ground State Structures in Ordered Binary Alloys with Second Neighbor Interactions*. Acta Metall., vol. 20, 3, (1972), 423–433. URL [http://dx.doi.org/10.1016/0001-6160\(72\)90037-5](http://dx.doi.org/10.1016/0001-6160(72)90037-5)
- [4] D. Aregba-Driollet, F. Diele, and R. Natalini. *A mathematical model for the sulphur dioxide aggression to calcium carbonate stones: numerical approximation and asymptotic analysis*. SIAM J. Appl. Math., vol. 64, 5, (2004), 1636–1667. ISSN 0036-1399. URL <http://dx.doi.org/10.1137/S003613990342829X>
- [5] M. Bassenne, L. Fu, and A. Mani. *Time-accurate and highly-stable explicit operators for stiff differential equations*. J. Comput. Phys., vol. 424, (2021), Paper No. 109847, 24. ISSN 0021-9991. URL <http://dx.doi.org/10.1016/j.jcp.2020.109847>
- [6] H. Bateman. *Some recent researches on the motion of fluids*. Monthly Weather Review, vol. 43, 3, (1915), 163–170. URL [http://dx.doi.org/10.1175/1520-0493\(1915\)43<163:SRROTM>2.0.CO;2](http://dx.doi.org/10.1175/1520-0493(1915)43<163:SRROTM>2.0.CO;2)
- [7] M. A. Budroni, G. Pagano, D. Conte, B. Paternoster, R. D’Ambrosio, S. Ristori, A. Abou-Hassan, and F. Rossi. *Synchronization scenarios induced by delayed communication in arrays of diffusively coupled autonomous chemical oscillators*. Phys. Chem. Chem. Phys., vol. 23, 32, (2021), 17606–17615. URL <http://dx.doi.org/10.1039/d1cp02221k>
- [8] M. A. Budroni, K. Torbensen, S. Ristori, A. Abou-Hassan, and F. Rossi. *Membrane Structure Drives Synchronization Patterns in Arrays of Diffusively Coupled Self-Oscillating Droplets*. J. Phys. Chem. Lett., vol. 11, 6, (2020), 2014–2020. URL <http://dx.doi.org/10.1021/acs.jpcclett.0c00072>
- [9] J. M. Burgers. *A mathematical model illustrating the theory of turbulence*. *Advances in Applied Mechanics*, (Academic Press, Inc., New York, N.Y.1948). 171–199. Edited by Richard von Mises and Theodore von Kármán, URL [http://dx.doi.org/10.1016/S0065-2156\(08\)70100-5](http://dx.doi.org/10.1016/S0065-2156(08)70100-5)
- [10] J. C. Butcher. *The numerical analysis of ordinary differential equations*. A Wiley-Interscience Publication, (John Wiley & Sons, Ltd., Chichester1987). ISBN 0-471-91046-5. Runge-Kutta and general linear methods, URL <http://dx.doi.org/10.1137/1031147>
- [11] J. C. Butcher. *Numerical methods for ordinary differential equations*, (John Wiley & Sons, Ltd., Chichester2008), second edn. ISBN 978-0-470-72335-7. URL <http://dx.doi.org/10.1002/9780470753767>

- [12] M. Calvo, J. I. Montijano, and L. Rández. *A note on the stability of time-accurate and highly-stable explicit operators for stiff differential equations*. J. Comput. Phys., vol. 436, (2021), Paper No. 110316, 13. ISSN 0021-9991. URL <http://dx.doi.org/10.1016/j.jcp.2021.110316>
- [13] A. Cardone, R. D’Ambrosio, and B. Paternoster. *Exponentially fitted IMEX methods for advection-diffusion problems*. J. Comput. Appl. Math., vol. 316, (2017), 100–108. ISSN 0377-0427. URL <http://dx.doi.org/10.1016/j.cam.2016.08.025>
- [14] A. Cardone, Z. Jackiewicz, A. Sandu, and H. Zhang. *Extrapolation-based implicit-explicit general linear methods*. Numer. Algorithms, vol. 65, 3, (2014), 377–399. ISSN 1017-1398. URL <http://dx.doi.org/10.1007/s11075-013-9759-y>
- [15] D. Conte and G. Frasca-Caccia. *A Matlab code for the computational solution of a phase field model for pitting corrosion*. Dolomites Res. Notes Approx., vol. 15, Special Issue SA2022—Software for Approximation 2022, (2022), 47–65. URL <http://dx.doi.org/10.14658/PUPJ-DRNA-2022-2-5>
- [16] D. Conte, G. Pagano, and B. Paternoster. *Two-step peer methods with equation-dependent coefficients*. Comput. Appl. Math., vol. 41, 4, (2022), Paper No. 140, 21. ISSN 2238-3603. URL <http://dx.doi.org/10.1007/s40314-022-01844-z>
- [17] D. Conte, G. Pagano, and B. Paternoster. *Nonstandard finite differences numerical methods for a vegetation reaction-diffusion model*. J. Comput. Appl. Math., vol. 419, (2023), Paper No. 114790, 17. ISSN 0377-0427. URL <http://dx.doi.org/10.1016/j.cam.2022.114790>
- [18] D. Conte, G. Pagano, and B. Paternoster. *Time-accurate and highly-stable explicit peer methods for stiff differential problems*. Commun. Nonlinear Sci. Numer. Simul., vol. 119, (2023), Paper No. 107136, 20. ISSN 1007-5704. URL <http://dx.doi.org/10.1016/j.cnsns.2023.107136>
- [19] L. Eigentler and J. A. Sherratt. *Metastability as a Coexistence Mechanism in a Model for Dryland Vegetation Patterns*. Bull. Math. Biol., vol. 81, (2019), 2290–2322. URL <http://dx.doi.org/10.1007/s11538-019-00606-z>
- [20] G. Frasca-Caccia and P. E. Hydon. *Locally conservative finite difference schemes for the modified KDV equation*. J. Comput. Dyn., vol. 6, 2, (2019), 307–323. ISSN 2158-2491. URL <http://dx.doi.org/10.3934/jcd.2019015>
- [21] G. Frasca-Caccia and P. E. Hydon. *Simple bespoke preservation of two conservation laws*. IMA J. Numer. Anal., vol. 40, 2, (2020), 1294–1329. ISSN 0272-4979. URL <http://dx.doi.org/10.1093/imanum/dry087>
- [22] S. González-Pinto, D. Hernández-Abreu, G. Pagano, and S. Pérez-Rodríguez. *Generalized TASE-RK methods for stiff problems*. Appl. Numer. Math., vol. 188, (2023), 129–145. ISSN 0168-9274. URL <http://dx.doi.org/10.1016/j.apnum.2023.03.007>
- [23] S. González-Pinto, D. Hernández-Abreu, and S. Pérez-Rodríguez. *Rosenbrock-type methods with inexact AMF for the time integration of advection-diffusion-reaction PDEs*. J. Comput. Appl. Math., vol. 262, (2014), 304–321. ISSN 0377-0427. URL <http://dx.doi.org/10.1016/j.cam.2013.10.050>

- [24] S. González-Pinto, D. Hernández-Abreu, and S. Pérez-Rodríguez. *W-methods to stabilize standard explicit Runge-Kutta methods in the time integration of advection-diffusion-reaction PDEs*. J. Comput. Appl. Math., vol. 316, (2017), 143–160. ISSN 0377-0427. URL <http://dx.doi.org/10.1016/j.cam.2016.08.026>
- [25] S. González-Pinto, D. Hernández-Abreu, S. Pérez-Rodríguez, and R. Weiner. *A family of three-stage third order AMF-W-methods for the time integration of advection diffusion reaction PDEs*. Appl. Math. Comput., vol. 274, (2016), 565–584. ISSN 0096-3003. URL <http://dx.doi.org/10.1016/j.amc.2015.10.013>
- [26] E. Hairer and G. Wanner. *Solving ordinary differential equations. II, Springer Series in Computational Mathematics*, vol. 14, (Springer-Verlag, Berlin1996), second edn. ISBN 3-540-60452-9. Stiff and differential-algebraic problems, URL <http://dx.doi.org/10.1007/978-3-642-05221-7>
- [27] I. Higuera and T. Roldán. *Starting algorithms for some DIRK methods*. Numer. Algorithms, vol. 23, 4, (2000), 357–369. ISSN 1017-1398. URL <http://dx.doi.org/10.1023/A:1019112419829>
- [28] A. Iserles. *A first course in the numerical analysis of differential equations*. Cambridge Texts in Applied Mathematics, (Cambridge University Press, Cambridge1996). ISBN 0-521-55376-8; 0-521-55655-4. URL <http://dx.doi.org/10.1017/CBO9780511995569>
- [29] Z. Jackiewicz. *General linear methods for ordinary differential equations*, (John Wiley & Sons, Inc., Hoboken, NJ2009). ISBN 978-0-470-40855-1. URL <http://dx.doi.org/10.1002/9780470522165>
- [30] W. Ji, W. Qiu, Z. Shi, S. Pan, and S. Deng. *Stiff-PINN: Physics-Informed Neural Network for Stiff Chemical Kinetics*. J. Phys. Chem. A, vol. 125, 36, (2021), 8098–8106. URL <http://dx.doi.org/10.1021/acs.jpca.1c05102>
- [31] B. A. Schmitt and R. Weiner. *Parallel two-step W-methods with peer variables*. SIAM J. Numer. Anal., vol. 42, 1, (2004), 265–282. ISSN 0036-1429. URL <http://dx.doi.org/10.1137/S0036142902411057>
- [32] B. A. Schmitt, R. Weiner, and H. Podhaisky. *Multi-implicit peer two-step W-methods for parallel time integration*. BIT, vol. 45, 1, (2005), 197–217. ISSN 0006-3835. URL <http://dx.doi.org/10.1007/s10543-005-2635-y>
- [33] H. Soomro, N. Zainuddin, H. Daud, J. Sunday, N. Jamaludin, A. Abdullah, M. Apriyanto, and E. Kadir. *Variable Step Block Hybrid Method for Stiff Chemical Kinetics Problems*. Appl. Sci., vol. 19, 9, (2022), Paper No. 4484. URL <http://dx.doi.org/10.3390/app12094484>
- [34] T. Steihaug and A. Wolfbrandt. *An attempt to avoid exact Jacobian and nonlinear equations in the numerical solution of stiff differential equations*. Math. Comp., vol. 33, 146, (1979), 521–534. ISSN 0025-5718. URL <http://dx.doi.org/10.2307/2006293>
- [35] R. Weiner, K. Biermann, B. A. Schmitt, and H. Podhaisky. *Explicit two-step peer methods*. Comput. Math. Appl., vol. 55, 4, (2008), 609–619. ISSN 0898-1221. URL <http://dx.doi.org/10.1016/j.camwa.2007.04.026>

- [36] H. Zhang, X. Qian, J. Xia, and S. Song. *Unconditionally maximum-principle-preserving parametric integrating factor two-step Runge-Kutta schemes for parabolic sine-Gordon equations*. CSIAM Trans. Appl. Math., vol. 4, 1, (2023), 177–224. ISSN 2708-0560. URL <http://dx.doi.org/10.4208/csiam-am.so-2022-0019>