PADOVA UNIVERSITY PRESS

Dolomites Research Notes on Approximation

Special Issue SA2022 – *Software for Approximation 2022*, Volume 15 · 2022 · Pages 66–80

Two classes of linearly implicit numerical methods for stiff problems: analysis and MATLAB software*

Dajana Conte[a] · Giovanni Pagano[a] · Beatrice Paternoster[a]

Abstract

The purpose of this work lies in the writing of efficient and optimized Matlab codes to implement two classes of promising linearly implicit numerical schemes that can be used to accurately and stably solve stiff Ordinary Differential Equations (ODEs), and also Partial Differential Equations (PDEs) through the Method Of Lines (MOL). Such classes of methods are the Runge-Kutta (RK) [28] and the Peer [17], and have been constructed using a variant of the Exponential-Fitting (EF) technique [27]. We carry out numerical tests to compare the two methods with each other, and also with the well known and very used Gaussian RK method, by the point of view of stability, accuracy and computational cost, in order to show their convenience.

1 Introduction

In this paper, we propose optimized Matlab codes for solving initial values problems of the type

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

For the numerical solution of ODEs of the type (1), which can also arise from PDEs semi-discretized in space using the MOL, there are several methods in the scientific literature. The choice of the numerical method must be weighed according to the characteristics of the problem to be solved. For example, if all components of the problem are characterized by severe stiffness, it is advisable to use totally implicit methods [2, 3, 23]. If, on the other hand, the vector field f is characterized by a stiff part and a non-stiff one, it can be used an Implicit-Explicit approach [6, 7, 8, 9], in which stiff components are treated with implicit methods, and non-stiff components with explicit methods. Finally, if the ODEs system is not affected by stiffness, explicit methods [20, 29, 30, 34] can be used, which are much cheaper than implicit ones (see, e.g, [4, 22] for an overview on these approaches).

However, the characteristics of the problem (1) are not always a-priori known. Therefore, given that the use of a totally implicit method can be very expensive, as the solution of a big dimension non-linear system is required at each step, and that of an explicit method can lead to numerical instability, one can resort to the use of linearly implicit methods [5, 12, 15, 18, 24, 25, 31, 32]. In fact, linearly implicit methods, requiring the resolution of linear systems at each step, are much cheaper than implicit methods, especially as the size d of the problem to be solved increases. In addition, they usually have very good stability properties. Hence, they are also advantageous over explicit methods, whereby the use of very small time steps may be required.

In this work, we analyze two different promising linearly implicit numerical methods, belonging to the class of RK and Peer, proposing for them an alternative formulation to reduce the computational cost of the related Matlab code. Fixing the uniform discrete grid $\{t_n = t_0 + nh; n = 0, ..., N; t_N = T\}$, one-step RK methods are s-stage numerical schemes of the form

$$\begin{cases} Y_{n,i} = y_n + h\sum_{j=1}^{s} a_{ij} f(t_n + c_j h, Y_{n,j}), & 1 \le i \le s, \\ y_{n+1} = y_n + h\sum_{j=1}^{s} b_j f(t_n + c_j h, Y_{n,j}), & n = 0, ..., N-1, \end{cases} \tag{2}$$

where the stages are $Y_{n,i} \approx y(t_n + c_i h)$, $i = 1, ..., s$, and the advancing solution is $y_{n+1} \approx y(t_{n+1})$. Peer methods are instead two-step numerical schemes formulated as

$$Y_{n,i} = \sum_{j=1}^{s} b_{ij} Y_{n-1,j} + h\sum_{j=1}^{s} a_{ij} f(t_{n-1} + c_j h, Y_{n-1,j}) + \sum_{j=1}^{s} q_{ij} Y_{n,j} + h\sum_{j=1}^{s} r_{ij} f(t_n + c_j h, Y_{n,j}), \quad 1 \le i \le s, \qquad n = 1, ..., N-1, \tag{3}$$

where the stages approximate the exact solution at the same points as the RK. Usually, and this is the choice made for the method discussed in this work, the advancing solution corresponds to the last stage, i.e. $c_s = 1$.

The RK methods (2), see, e.g., [3], are the best known numerical schemes that can be used to solve first order ODEs of type (1). In fact, their coefficients, depending on the related order conditions, can be easily obtained by means of the Butcher tree

*The preface of this special issue to which the article belongs is given in [10].

[a]Department of Mathematics, University of Salerno, Via Giovanni Paolo II 132, 84084 - Fisciano, Italy. E-mail: {dajconte,gpagano,beapat}@unisa.it.

theory. Furthermore, they are one-step numerical schemes, which means that no starting procedure is required. Finally, implicit RK methods enjoy excellent stability properties and thus can be used to solve highly stiff ODEs and PDEs. On the other hand, Peer methods, see, e.g., [34], are newer and less known than RK methods, but they are advantageous over the latter in two aspects. First of all, compared to RK [33], they are not affected by order reduction if applied to very stiff problems, and this happens thanks to the way in which the related order conditions are defined. Furthermore, they can be parallelized. In fact, observing formulation (3), note that if $(q_{ij})_{i,j=1}^{s}$ and $(r_{ij})_{i,j=1}^{s}$ are diagonal matrices, the subsequent stages depend only on those calculated in the previous interval. Since Peer methods are two-step numerical schemes, however, they require a starting procedure. In this work, we also show that the starting procedure for Peer methods can be performed efficiently through the linearly implicit RK schemes derived in [28].

The two classes of linearly implicit methods we analyze have been derived using the EF technique [26, 27]. Usually, the EF technique is used when the oscillating trend and the exact frequency of the exact solution of the problem to be solved are known [11, 13, 14, 16]. An EF inspired approach has been proposed in [28] to derive the principal term of the local truncation error of explicit two- and three-stage one-step RK methods, imposing the greatest possible accuracy to determine the stages and the advancing solution. By doing so, new methods coefficients are obtained, which appear to depend on the Jacobian of the ODEs system (1), and it is possible for them a choice that leads to an increase in the order of consistency with respect to the explicit case and to A-stable schemes. Following what has been done in [28] for RK methods, in [17] the authors extended the EF inspired procedure for explicit two-stage Peer methods. However, Peer methods are two-step numerical schemes, and this, as we will see, leads to different calculations and results than those obtained for RK. Even for Peer methods an improvement in stability and accuracy is obtained.

This paper is organized as follows. In Section 2 we recall the linearly implicit RK methods proposed in [28], showing in detail the derivation of the two-stage version. Furthermore, we propose a related Matlab code based on an equivalent formulation of these methods, leading to a reduction in computing times. In Section 3 we recall the linearly implicit Peer methods derived in [17], then doing the same as in Section 2. In Section 4 we show in detail the writing of a Matlab script to apply the codes reported in Sections 2 and 3. In Section 5 we perform numerical tests to compare the two classes of methods also with another well known numerical scheme, showing their utility. In Section 6 we discuss the obtained results.

2 Linearly implicit Runge-Kutta methods

The linearly implicit RK methods we consider have been derived through the EF in [28]. Before showing the Matlab code, we describe these methods and the technique of derivation of the coefficients.

Since in [28] the author shows the adopted procedure mainly for the three-stage case, we now show in detail the computations for the two-stage case. Consider an explicit two-stage RK method of order two, whose coefficients can be summarized in the following Butcher tableau:

$$\begin{array}{c|cc} 0 & & \\ c_2 & a_{21} & \\ \hline & b_1 & b_2 \end{array}, \qquad c_2 \in (0,1], \quad a_{21} = c_2, \quad b_1 = 1 - \frac{c_2}{2}, \quad b_2 = 1 - b_1. \tag{4}$$

The coefficients values of Tableau 4 can be easily derived using the famous Butcher tree theory.

In [28], the author derives instead different coefficients for an explicit two-stage RK method, using the EF approach. First of all, consider the first RK stage

$$Y_{n,1} = y_n, \qquad y_n \approx y(t_n).$$

Moreover, consider the second RK stage, which is computed as follows:

$$Y_{n,2} = y_n + h a_{21} f(t_n, Y_{n,1}). \tag{5}$$

Since $Y_{n,2} \approx y(t_n + c_2 h)$, it makes sense to derive a value of a_{21} in (5) for which one gets the best possible approximation. The EF approach foresees to consider the continuous expression of the second stage

$$\mathcal{Y}_2(t) = y(t) + h a_{21} y'(t), \tag{6}$$

then defining the related error operator

$$\underline{L_2} y(t) = y(t + c_2 h) - \mathcal{Y}_2(t). \tag{7}$$

The operator $\underline{L_2} y(t)$ measures the error associated with the calculation of the second stage.

Following the EF approach, to make this error small, as many moments as possible have to be annihilated. The moments associated with $\underline{L_2} y(t)$, which we indicate with L_{2k}, $k \geq 0$, correspond to the evaluations of $\underline{L_2} y(t)$, for $y(t) = t^k$, $k \geq 0$, at $t = 0$. In fact, the following property is exploited, which binds the moments to the related error operator:

$$\underline{L_2} y(t) = \sum_{k=0}^{\infty} L_{2k} \frac{y^{(k)}(t)}{k!}.$$

From (6)-(7), it holds that

$$\underline{L_2} t^k = (t + c_2 h)^k - (t^k + h a_{21} k t^{k-1}), \quad k \geq 0.$$

Therefore, the moments are

$$L_{20} = 1 - 1 = 0, \quad L_{21} = h(c_2 - a_{21}), \quad L_{22} = (c_2 h)^2, \ldots, L_{2k} = (c_2 h)^k, \quad k \geq 2.$$

In our case, the first moment is already null, and to annihilate the second one it is necessary to set $a_{21} = c_2$. It is not possible to annihilate the subsequent ones, as it makes no sense to choose $c_2 = 0$. Therefore, the following writing holds:

$$y(t + c_2h) = \mathcal{Y}_2(t) + err_2(t), \tag{8}$$

where

$$err_2(t) = \sum_{k=0}^{\infty} L_{2k} \frac{y^{(k)}(t)}{k!} = t_{err_2}(t) + O(h^4),$$
$$t_{err_2}(t) = \frac{1}{2}(c_2h)^2 y^{(2)}(t) + \frac{1}{6}(c_2h)^3 y^{(3)}(t).$$

The final step lies in the computation of the advancing solution, given by

$$y_{n+1} = y_n + h(b_1 f(t_n, y_n) + b_2 f(t_n + c_2h, Y_2)). \tag{9}$$

As previously done, the related error operator

$$\underline{L_{adv}} y(t) = y(t + h) - \mathcal{Y}(t), \tag{10}$$

is introduced, where

$$\mathcal{Y}(t) = y(t) + h(b_1 f(t_n, Y_1) + b_2 f(t_n + c_2h, Y_2)).$$

Taking into account that, from (8), $err_2(t) \sim O(h^2)$, and using Taylor series expansions, it holds that

$$y'(t + c_2h) = f(t + c_2h, y(t + c_2h)) = f(t + c_2h, \mathcal{Y}_2(t) + err_2(t)) = f(t + c_2h, \mathcal{Y}_2(t)) + J_2(t) err_2(t) + O(err_2(t)^2),$$

where $J_2(t) = f_y(t + c_2h, y)_{|y=\mathcal{Y}_2(t)}$ is the Jacobian of the ODEs system (1) computed at the second stage $Y_{n,2}$ during the numerical integration. In fact, using the localizing assumption $y(t_n) = y_n$ (then $y'(t_n) = f(t_n, Y_{n,1})$), which is supposed to be valid for obtaining the order conditions of a numerical method, from (5)-(6), it follows that $\mathcal{Y}_2(t_n) = Y_{n,2}$. Finally, we can write

$$y'(t + c_2h) = f(t + c_2h, \mathcal{Y}_2(t)) + J_2(t) t_{err_2}(t) + O(h^4).$$

From (10), ignoring the terms of order equal to or greater than four, it is obtained that

$$\underline{L_{adv}} t^k = (t + h)^k - t^k - h(b_1 k t^{k-1} + b_2(k(t + c_2h)^{k-1} - J_2(t) t_{err_2*}(t))),$$

where

$$t_{err_2*}(t) = \frac{1}{2}(c_2h)^2 k(k-1)t^{k-2} + \frac{1}{6}(c_2h)^3 k(k-1)(k-2)t^{k-3}, \quad k \geq 0.$$

The moments now are

$$L_{adv0} = 0, \quad L_{adv1} = h(1 - b_1 - b_2), \quad L_{adv2} = h^2(1 - b_2c_2(2 - c_2hJ_2(t))), \quad L_{adv3} = h^3(1 - b_2c_2^2(3 - c_2hJ_2(t))), \dots \quad .$$

Annihilating L_{adv1} and L_{adv2} leads to

$$b_2 = \frac{1}{2c_2}\Big(I - \frac{c_2}{2}hJ_2(t_n)\Big)^{-1}, \quad b_1 = 1 - b_2, \tag{11}$$

where we have indicated with I the Identity matrix of dimension d. Setting the coefficients as in (11) leads to a two-stage RK method that has at least order two. Note therefore that the coefficients b_1 and b_2 are no longer scalars and constants as in the explicit case, but they are matrices dependent on the Jacobian of the ODEs system (1). This is the reason why, from now on, we will indicate them with capital letters, i.e. with B_2 and B_1, respectively. Although the considered RK method is explicit by structure, it is linearly implicit, as we will show that the dependence of the coefficients on the equation to be solved practically leads to the solution of some linear systems.

Making similar considerations and calculations for a three-stage explicit RK with the Butcher tableau

$$\begin{array}{c|ccc} 0 & & & \\ c_2 & a_{21} & & \\ c_3 & 0 & a_{32} & \\ \hline & b_1 & b_2 & b_3 \end{array},$$

one gets a method that has at least order three by setting $a_{21} = c_2$, $a_{32} = c_3$, and

$$B_1 = Q^{-1}T, \quad B_3 = \frac{1}{c_2^2} B_3\Big((1 - c_2 - c_3(3c_3 - 2c_2))I + (-c_3^2 + c_2c_3 + c_2^2)hJ_3(t_n)\Big), \quad B_3 = I - B_1 - B_2,$$

where

$$T = (3c_2 - 2)I - c_2(c_2 - 1)hJ_2(t_n),$$
$$Q = c_3\Big(6(c_2 - c_3)I + c_2(3c_3 - 2c_2)hJ_2(t_n) + c_3(2c_3 - 3c_2)hJ_3(t_n) + c_3c_2(c_2 - c_3)h^2J_3(t_n)J_2(t_n)\Big),$$
$$J_2(t) = f_y(t + c_2h, y)_{|y=Y_{n,2}}, \quad J_3(t) = f_y(t + c_3h, y)_{|y=Y_{n,3}}.$$

Table 1: Accuracy and stability properties of the two- and three-stage linearly implicit RK methods described in Section 2 by setting specific values for the coefficients.

Two-stage linearly implicit RK			Three-stage linearly implicit RK		
Coefficients	Order	Stability	Coefficients	Order	Stability
$c_2 = 1$	2	A-stability	$c_2 = 1/2$, $c_3 = 1$	4	A-stability
$c_2 = 2/3$	3	No A-stability	$c_2 = 1$, $c_3 = 1/2$	4	A-stability

Note therefore that we have directly reported the coefficients b_i, $i = 1,2,3$, using capital letters, as they are Jacobian-dependent matrices.

In [28], the author derives values of the coefficients of the two- and three-stage linearly implicit RK methods that allow to obtain A-stable numerical schemes and/or with increased order of consistency. The mentioned results are summarized in Table 1. For the three-stage method, we will directly use the A-stable version of order four with $c_2 = 1/2$ and $c_3 = 1$, whose matrix coefficients can be simplified as

$$B_1 = B_{den}^{-1}\big(2I - 3hJ_2(t)\big), \quad B_2 = B_{den}^{-1}\big(8I - 2hJ_3(t) + h^2J_2(t)J_3(t)\big), \quad B_3 = B_{den}^{-1}\big(2 - hJ_2(t)\big), \tag{12}$$

where

$$B_{den} = 12I - 4hJ_2(t) - 2hJ_3(t) + h^2J_2(t)J_3(t). \tag{13}$$

2.1 Matlab implementation

In this subsection, we show an optimized Matlab code for the linearly implicit RK methods analyzed in this paper. To optimize the code while avoiding the inversion of matrices, we observe that such RK methods can be reformulated.

2.1.1 Optimization of the two-stage version

Considering the two-stage case, from (9)-(11), in the classic formulation the advancing solution is calculated as

$$y_{n+1} = y_n + h(B_1 f(t_n, y_n) + B_2 f(t_n + c_2 h, Y_{n,2})),$$

where

$$B_1 = I - Q^{-1}, \quad B_2 = Q^{-1}, \quad Q = 2c_2 I - c_2^2 hJ_2(t).$$

Note that all the other coefficients are scalars. By setting $k_1 = hf(t_n, y_n)$ and $k_2 = hf(t_n + c_2 h, Y_{n,2})$, we can write

$$y_{n+1} = y_n + k_1 - Q^{-1}k_1 + Q^{-1}k_2.$$

Hence, we have to practically solve two linear systems of the form

$$\tilde{A}x = \tilde{b},$$

where, for both, $\tilde{A} = Q$, and $\tilde{b} = k_1$ or $\tilde{b} = k_2$, respectively. Since the coefficient matrix of the two linear systems is the same, rather than solving the two systems separately, we can a-priori calculate the LU factorization of Q using the Matlab function `lu`, then computing $x_1 = U\backslash(L\backslash k_1)$ and $x_2 = U\backslash(L\backslash k_2)$, where \ is the Matlab *backslash* command. Finally, we compute the advancing solution as

$$y_{n+1} = y_n + k_1 - x_1 + x_2.$$

By reformulating the method in this way, we are able to obtain lower CPU times than the classic formulation, as we will show in numerical tests.

Remark 1. It is not certain that a square matrix always admits a pure LU factorization, where L is a lower triangular matrix and U is upper triangular. However, a square matrix A always admits a factorization of LUP type, i.e. $PA = LU$, where P is known as permutation matrix. The Matlab function `lu` returns two matrices L and U linked precisely to a factorization of this type, in which L is not exactly lower triangular, but is the product between a transposed permutation matrix and a lower triangular matrix.

2.1.2 Optimization of the three-stage version

Using similar idea to that of the two-stage case, also for the three-stage case we now propose an alternative formulation of the linearly implicit RK methods to avoid inverting matrices, considerably lowering the computing times.

Note that all the coefficients are scalars, except B_i, $i = 1,2,3$, which are involved in the computation of the advancing solution

$$y_{n+1} = y_n + h(B_1 f(t_n, y_n) + B_2 f(t_n + c_2 h, Y_{n,2}) + B_3 f(t_n + c_3 h, Y_{n,3})).$$

As previously said, we fix the matrix coefficients B_i, $i = 1,2,3$, as in (12)-(13). Setting $k_i = B_i^{num} hf(t_n + c_i h, Y_i)$, $i = 1,2,3$, where B_i^{num} is the numerator of B_i, $i = 1,2,3$ (12), then computing the LU factorization of B_{den} (13), the advancing solution becomes

$$y_{n+1} = y_n + x_1 + x_2 + x_3,$$

where $x_i = U\backslash(L\backslash k_i)$, $i = 1,2,3$. This reformulation allows a significant reduction in the cost of the method.

Table 2: Input arguments of the functions `lin_imp_RK.m` and `lin_imp_peer.m`.

Argument	Description
N - integer scalar	Number of discrete time intervals
tspan - double array	Two-component row vector containing first and last grid point, respectively
y0 - double array	Column vector of length d containing the problem exact solution at the initial time point t_0
p - integer scalar	Parameter selected by the user, related to chosen problem to be solved

Table 3: Output arguments of the functions `lin_imp_RK.m` and `lin_imp_peer.m`.

Argument	Description
CPUtime - double scalar	Total CPU time in seconds taken by the method
yT - double array	Column vector of length d containing the problem numerical solution at the final time point t_N
y - double array	Matrix of size $d \times (N+1)$ containing the numerical solution y_n in column $n+1$
t - double array	Row vector of length $N+1$ containing all the time grid points

2.1.3 Matlab code

The main program `lin_imp_RK.m` is shown below. The Input and Output arguments are described in Tables 2 and 3, respectively. The auxiliary functions used by it are briefly summarized in Table 4. Furthermore, the called built in Matlab functions are listed in Table 5.

From line 2 to 4 of the function `lin_imp_RK.m`, we memorize in a variable called `dim` the dimension of the problem, then defining the Identity matrix `Id` and deciding if we want to work with the two- or three-stage linearly implicit RK method. Therefore, the parameter `s` of line 4 must be set by the user according to his preferences. In line 5, the user can decide whether to use the classic formulation, setting `form=1`, or the optimized one, setting `form=2`. From line 7 to 14, we define the scalar coefficients of the RK method. In line 16, we initialize the matrix `y`, which contains in column $n+1$ the approximation of the numerical solution at t_n. In fact, the first column of `y` corresponds to the vector y_0. We then define other quantities of interest, such as the step-size in time `h`, the discrete grid `t`, and the variable `n`, which represents the current integration step. From line 25 to 43, we compute the advancing solution at each step by means of the two-stage linearly implicit RK method, if selected. From line 46 to 78, we compute the advancing solution at each step by means of the three-stage linearly implicit RK method, if selected. Finally, we calculate the CPU time and store the solution at the last grid point in a column vector called `yT`. By default, the code is setted on the three-stage version (see line 4, where `s=3`) with the optimized formulation (see line 5, where `form=2`).

```
1  function [CPUtime,yT,y,t] = lin_imp_RK(N,tspan,y0,p)
2  dim = length(y0);
3  Id = eye(dim);
4  s = 3;
5  form = 2;
6  % % Selecting the matrix A and vector c of the Runge-Kutta
7  switch s
8      case 2
9          c = [0;1];
10         A = [0 0;c(2) 0];
11     case 3
12         c = [0;1/2;1];
13         A = [0 0 0;c(2) 0 0;0 c(3) 0];
14 end
15 % % Initialization
16 y = [y0];
17 h = (tspan(2)-tspan(1))/N;
18 t = linspace(tspan(1),tspan(2),N+1);
19 n = 1;
20 % % Method
21 C = cputime;
22 switch s
23     case 2
24         % % Using the two-stage Runge-Kutta
25         for n = 2:N+1
26             Y1 = y(:,n-1);
27             Y2 = y(:,n-1)+h*c(2)*funz(t(n-1),Y1,p);
28             M2 = h*jacob(t(n-1)+c(2)*h,Y2,p);
29             Q = 2*c(2)*Id-c(2)*c(2)*M2;
30             switch form
31                 case 1
32                     % % Using classic method formulation
33                     B1 = Id-inv(Q); B2 = Id-B1;
```

```
34                      y(:,n) = y(:,n-1)+h*(B1*funz(t(n-1),Y1,p)+...
35                          B2*funz(t(n-1)+c(2)*h,Y2,p));
36                  case 2
37                      % % Using LU decomposition
38                      [L,U] = lu(Q);
39                      k1 = h*funz(t(n-1),Y1,p); x1 = U\(L\k1);
40                      k2 = h*funz(t(n-1)+c(2)*h,Y2,p); x2 = U\(L\k2);
41                      y(:,n) = y(:,n-1)+k1-x1+x2;
42              end
43          end
44      case 3
45          % % Using the three-stage Runge-Kutta
46          for n = 2:N+1
47              Y1 = y(:,n-1);
48              Y2 = y(:,n-1)+h*c(2)*funz(t(n-1),Y1,p);
49              Y3 = y(:,n-1)+h*c(3)*funz(t(n-1)+c(2)*h,Y2,p);
50              M2 = h*jacob(t(n-1)+c(2)*h,Y2,p);
51              M3 = h*jacob(t(n-1)+c(3)*h,Y3,p);
52              switch form
53                  case 1
54                      % % Using classic method formulation
55                      Bden = Id-(1/3)*M2-(1/6)*M3+(1/12)*M3*M2;
56                      Q = inv(Bden);
57                      B1num = Id-(3/2)*M2;
58                      B2num = Id-(1/4)*M3+(1/8)*M3*M2;
59                      B3num = Id-(1/2)*M2;
60                      B1 = (1/6)*Q*B1num;
61                      B2 = (2/3)*Q*B2num;
62                      B3 = (1/6)*Q*B3num;
63                      y(:,n) = y(:,n-1)+h*(B1*funz(t(n-1),Y1,p)+...
64                          B2*funz(t(n-1)+c(2)*h,Y2,p)+...
65                          B3*funz(t(n-1)+c(3)*h,Y3,p));
66                  case 2
67                      % % Using LU decomposition
68                      Bden = Id-(1/3)*M2-(1/6)*M3+(1/12)*M3*M2;
69                      [L,U] = lu(Bden);
70                      B1num = (1/6)*(Id-(3/2)*M2);
71                      B2num = (2/3)*(Id-(1/4)*M3+(1/8)*M3*M2);
72                      B3num = (1/6)*(Id-(1/2)*M2);
73                      k1 = B1num*h*funz(t(n-1),Y1,p); x1 = U\(L\k1);
74                      k2 = B2num*h*funz(t(n-1)+c(2)*h,Y2,p); x2 = U\(L\k2);
75                      k3 = B3num*h*funz(t(n-1)+c(3)*h,Y3,p); x3 = U\(L\k3);
76                      y(:,n) = y(:,n-1)+x1+x2+x3;
77              end
78          end
79  end
80  Cf = cputime;
81  CPUtime = Cf-C;
82  yT = y(:,end);
83  end
```

Table 4: Short description of the functions `funz.m` and `jacob.m` recalled in the algorithms `lin_imp_RK.m` and `lin_imp_peer.m`.

Input of both	Output of funz	Output of jacob
t - double scalar y - double array p - integer scalar	Column vector f (t,y) of length d	Jacobian matrix f_y(t,y) of size $d \times d$

Note that, in the main algorithm `lin_imp_RK.m`, the functions `funz.m` and `jacob.m` are recalled, which are briefly described in Table 4. Observe that they, like the main function `lin_imp_RK.m`, have the parameter `p` as Input argument. In fact, the idea followed in the implementation of the code is to consider a test-set of problems of the type (1), which can also derive from PDEs semi-discretized in space. Each choice of `p` corresponds to a different problem of the mentioned test-set. Therefore, `funz(t,y,p)` returns the value of the relative vector field f evaluated at the point `(t,y)`, and `jacob(t,y,p)` does the same thing for the Jacobian of f.

Table 5: Auxiliary Matlab functions adopted in the algorithms `lin_imp_RK.m` and `lin_imp_peer.m` in alphabetical order.

Function	Task
backslash	Computes in a different way $A^{-1}B$, where A and B are matrices
cputime	Returns the CPU time in seconds of the Matlab process
eye	Returns the Identity matrix of required dimension
inv	Computes the inverse of a square matrix
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

3 Linearly implicit Peer methods

A similar EF-based approach to improve the stability of explicit two-stage Peer methods has been proposed in [17]. However, the difference between the RK methods and Peer methods is substantial, as the latter are two-step numerical schemes. Let us quickly recap the made calculations to derive the coefficients values of the linearly implicit Peer methods.

The continuous expressions of the stages $Y_{n,1}$ and $Y_{n,2}$ in the interval $[t_n, t_{n+1}]$ are

$$\begin{aligned}\mathcal{Y}_1(t) =& b_{11}\mathcal{Y}_1(t-h) + b_{12}y(t) + ha_{11}f(t+h(c_1-1), \mathcal{Y}_1(t-h)) + ha_{12}y'(t),\\ \mathcal{Y}_2(t) =& b_{21}\mathcal{Y}_1(t-h) + b_{22}y(t) + ha_{21}f(t+h(c_1-1), \mathcal{Y}_1(t-h)) + ha_{22}y'(t) + hr_{21}f(t+hc_1, \mathcal{Y}_1(t)).\end{aligned}$$

In this case, assuming that the first stage is affected by error (as done for the RK methods in the previous section), means assuming that $\mathcal{Y}_1(t-h)$ also does not represent the exact value of the first stage in $[t_{n-1}, t_n]$. Therefore, unlike what has been done in [28] the authors of [17] had to work also with the moments associated with $\mathcal{Y}_1(t-h)$, annihilating them until order two, then doing the same for $\mathcal{Y}_1(t)$. Finally, the advancing solution, which corresponds with the second stage, has been computed by requiring for it order three.

Since for Peer methods the overall consistency order is equal to the minimum order in which all stages are calculated [34], the linearly implicit Peer methods determined with the EF technique still have order two, even if, as we will see in numerical tests, they initially exhibit order three and are characterized by a very low error. The coefficients of these methods, expressed in matrix form, are as follows:

$$\begin{aligned}A_{21} =& \frac{a_{21}^{cnum}}{a_{21}^{cden}}\Big(I + \frac{1}{a_{21}^{cden}}\big(A_{21}^{den1}hJ_1(t-h) + A_{21}^{den2}hJ_1(t)\big)\Big)^{-1}\Big(I + \frac{1}{a_{21}^{cnum}}A_{21}^{num}hJ_1(t)\Big),\\ A_{22} =& \frac{a_{22}^{cnum}}{a_{21}^{cden}}\Big(I + \frac{1}{a_{21}^{cden}}\big(A_{21}^{den1}hJ_1(t-h) + A_{21}^{den2}hJ_1(t)\big)\Big)^{-1}\Big(I + \frac{1}{a_{22}^{cnum}}\big(A_{22}^{num1}hJ_1(t-h) + A_{22}^{num2}hJ_1(t)\big)\Big),\\ R_{21} =& \frac{r_{21}^{cnum}}{r_{21}^{cden}}\Big(I + \frac{1}{r_{21}^{cden}}\big(R_{21}^{den1}hJ_1(t-h) + R_{21}^{den2}hJ_1(t)\big)\Big)^{-1}\Big(I + \frac{1}{r_{21}^{cnum}}R_{21}^{num}hJ_1(t-h)\Big),\end{aligned} \tag{14}$$

where

$$\begin{aligned}A_{21}^{num} =& -\big(-1+b_{21}(-1+c_1)^2\big)\big(b_{11}(-1+c_1)^3 - (-3+c_1)c_1^2\big)A_{21}^{num2},\\ A_{21}^{num2} =& \big(2I + b_{11}(-1+c_1)\big(-2I + (-1+c_1)hJ_1(t-h)\big) - c_1\big(2I + c_1hJ_1(t-h)\big)\big),\\ A_{21}^{den1} =& 2(-1+c_1)\big(b_{11}(-1+c_1)^3 - (-3+c_1)c_1^2\big)\big(b_{11}hJ_1(t) + c_1\big(2I + b_{11}(-2+c_1)hJ_1(t) - c_1hJ_1(t)\big)\big),\\ A_{21}^{den2} =& 4\big((-1+c_1)^2\big)\big(b_{11} - b_{11}^2(-1+c_1)^3 - 3b_{11}c_1 + (-3+c_1)c1^2\big),\\ A_{22}^{num1} =& 2(-1+c_1)(-b_{11}(-1+c_1)^3 + (-3+c_1)c_1^2)\big(1 - 2c_1 + b_{21}(-1+c_1^2)\big),\\ A_{22}^{num2} =& -(3 + b_{21}(-1+c_1)^2 - 2c_1)(b_{11}(-1+c_1)^3 - (-3+c_1)c_1^2)A_{21}^{num2},\\ a_{21}^{cnum} =& -4(-1+c_1)c_1\big(2 - 3c_1 + b_{21}(2 + b_{11}(-1+c_1)^3 + 3(-1+c_1)c_1)\big),\\ a_{21}^{cden} =& 24c_1(-1+c_1)^2,\\ a_{22}^{cnum} =& 4(-1+c_1)\big(5 + 6(-2+c_1)c_1 + b_{21}(-1 + b_{11}(-1+c_1)^3 - 3(-2+c_1)c_1^2)\big),\\ R_{21}^{num} =& \big(1 - b_{21}(-1+c_1)^2\big)\big(b_{11}(-1+c_1)^3 - (-3+c_1)c_1^2\big),\\ R_{21}^{den1} =& 2c_1(b_{11}(-1+c_1)^3 - (-3+c_1)c_1^2),\\ R_{21}^{den2} =& (b_{11}(-1+c_1)^3 - (-3+c_1)c_1^2)A_{21}^{num2},\\ r_{21}^{cnum} =& 2\big(-5 + (8-3c_1)c_1 + b_{21}(-1+c_1)(-1 + b_{11}(-1+c_1)^3 + 3c_1)\big),\\ r_{21}^{cden} =& 12(-1+c_1)c_1.\end{aligned} \tag{15}$$

Therefore, note that the coefficients a_{21}, a_{22} and r_{21} are Jacobian-dependent matrices, and for this reason we have indicated

them with a capital letter in (14), respectively. Instead, the coefficients a_{11} and a_{12} are scalars and correspond to

$$\begin{aligned} a_{11} &= \Big(-\big(b_{11}(-1+c_1)^2\big) + c_1^2 \Big) / \Big(2(-1+c_1) \Big), \\ a_{12} &= \Big(-\big(b_{11}(-1+c_1)^2\big) + (-2+c_1)c_1 \Big) / \Big(2(-1+c_1) \Big). \end{aligned} \tag{16}$$

The other free coefficients are set so as to obtain the best possible stability properties. In particular, $b_{11} = -0.24$, $b_{21} = -0.31$, and $c_1 = 0.2$. As a matter of consistency, $b_{12} = 1 - b_{11}$, and $b_{22} = 1 - b_{21}$. As mentioned in the Introduction, the advancing solution is calculated with the second stage, then $c_2 = 1$. The linearly implicit Peer method recalled in this section has much better stability properties and exhibit better accuracy than classic explicit Peer methods with the same number of stages.

3.1 Matlab implementation

In this subsection, we show the optimized Matlab code of the linearly implicit Peer method analyzed in this paper. As done in the previous section, we need to reformulate the method avoiding the inversion of matrices.

3.1.1 Optimization

We first observe, from the coefficients of the method (14)-(15), that the matrices to be inverted in the classical formulation are

$$Q_1 = I + \frac{1}{a_{21}^{cden}} \big(A_{21}^{den1} h J_1(t-h) + A_{21}^{den2} h J_1(t) \big), \quad Q_2 = I + \frac{1}{r_{21}^{cden}} \big(R_{21}^{den1} h J_1(t-h) + R_{21}^{den2} h J_1(t) \big).$$

Note also that the advancing solution, which corresponds with the second stage, is

$$Y_{n,2} = b_{21} Y_{n-1,1} + b_{22} Y_{n-1,2} + h(A_{21} f(t_n + h(c_1 - 1), Y_{n-1,1}) + A_{22} f(t_n, Y_{n-1,2}) + R_{21} f(t_n + h c_1, Y_{n,1})).$$

Thus, while the first stage involves only scalar coefficients, the second stage involves the three matrix coefficients A_{21}, A_{22} and R_{21}.

Then, we store in L_1 and U_1 the matrices of the factorization of Q_1 carried out with the Matlab function `lu`, respectively. Furthermore, calling with A_{21}^{NUM}, A_{22}^{NUM} and R_{21}^{NUM} the numerators of A_{21}, A_{22} and R_{21} (14), respectively, we compute

$$k_1 = A_{21}^{NUM} h f(t_n + h(c_1 - 1), Y_{n-1,1}), \quad k_2 = A_{22}^{NUM} h f(t_n, Y_{n-1,2}), \quad k_3 = R_{21}^{NUM} h f(t_n + h c_1, Y_{n,1}).$$

Finally, using the Matlab *backslash* command, we solve the linear systems

$$Q_1 x_i = k_i, \quad i = 1, 2,$$

as $x_i = U_1 \backslash (L_1 \backslash k_i)$, $i = 1, 2$, then computing $x_3 = Q_2 \backslash k_3$. Therefore, the second stage is computed as

$$Y_{n,2} = b_{21} Y_{n-1,1} + b_{22} Y_{n-1,2} + x_1 + x_2 + x_3.$$

3.1.2 Matlab code

The main program `lin_imp_peer.m` is shown below. The Input and Output arguments are the same as the linearly implicit RK methods described in the previous section (see Tables 2 and 3, respectively). The same holds for the used auxiliary functions and the called built in Matlab functions (see Tables 4 and 5, respectively).

From line 2 to 21 of the function `lin_imp_peer.m`, we store in the variable `dim` the dimension of the problem, then defining the Identity matrix, deciding if we want to work with the classic or optimized formulation (as in the RK case, by default `form=2`, i.e. we are working with the optimized formulation), and fixing the scalar and constant coefficients of the linearly implicit Peer method. From line 23 to 25, we define the step-size in time `h`, the discrete grid, and the variable `n` indicating the current integration step, respectively. In lines 27, 28, and 29, we carry out the stages initialization. Indeed, as said several times so far, Peer methods are two-step schemes, and it is therefore necessary to use a one-step method to determine the initial value of the stages. In this case, we need for the two stages an approximation of the solution at the points $[t_0 + c_1 h]$ and $[t_0 + c_2 h]$, respectively. Note, from line 28, that to compute $Y_{0,1}$, we call the function `lin_imp_RK.m`, i.e. we use the linearly implicit RK methods shown in the previous section. We do the same for $Y_{0,2}$ in line 29. Since, as previously said, the consistency order of Peer methods corresponds to the minimum order with which each stage is calculated, in this case the starting procedure must be done with a numerical method that has at least order two, as the Peer method we are considering has order two. Thus, we can choose both the linearly implicit two-stage and three-stage RK (by default we are considering the latter).

We are considering $c_2 = 1$, i.e. the second stage also represents the advancing solution. So, we store the column vector $Y_{0,2}$ in `y`. Initially, for index consistency (i.e., to make the initial index of the `for` loop starting in line 35 to be 2 and not 3), we do not insert y_0 in the vector `y`. This is done after applying the method, in line 79. Since for the considered linearly implicit Peer methods we need to know the Jacobian and the stages also in the previous grid interval, we store them in the arrays `M1tmh` (initialized in line 31 and updated in line 73), `Y1nm1` and `Y2nm1` (initialized in line 32 and updated in line 74), respectively. Analogously, the arrays containing such quantities in the current grid interval are named with `M1t` (computed in line 39), `Y1n` (computed in line 38) and `Y2n` (computed in line 58 or 71, depending on which formulation we are using), respectively. From line 35 to 76, we compute the solution at all the grid points. Finally, we compute the total CPU time in line 78, including that of the stages initialization, then storing the solution at the last grid point in the column vector `yT`.

```
function [CPUtime,yT,y,t] = lin_imp_peer(N,tspan,y0,p)
dim = length(y0);
Id = eye(dim);
form = 2;
% % Fixing method constant coefficients
B = [-0.24 1.24; -0.31 1.31]; c = [0.2;1];
A = [(-(B(1,1)*(-1+c(1))^2)+c(1)^2)/(2*(-1+c(1)))...
    (-(B(1,1)*(-1+c(1))^2)+(-2+c(1))*c(1))/(2*(-1+c(1)));0 0];
a21cnum = -4*(-1+c(1))*c(1)*(2-3*c(1)+B(2,1)*(2+B(1,1)*(-1+c(1))^3+...
    3*(-1+c(1))*c(1)));
a21cden = 24*c(1)*(-1+c(1))^2;
a22cnum = 4*(-1+c(1))*(5+6*(-2+c(1))*c(1)+B(2,1)*(-1+B(1,1)*(-1+c(1))^3-...
    3*(-2+c(1))*c(1)^2));
r21cnum = 2*(-5+(8-3*c(1))*c(1)+B(2,1)*(-1+c(1))*(-1+B(1,1)*(-1+c(1))^3+3*c(1)));
r21cden = 12*(-1+c(1))*c(1);
A21den2 = 4*(-1+c(1))^2*(B(1,1)-B(1,1)^2*(-1+c(1))^3-3*B(1,1)*c(1)+...
    (-3+c(1))*c(1)^2);
A22num1 = 2*(-1+c(1))*(-B(1,1)*(-1+c(1))^3+(-3+c(1))*c(1)^2)*(1-2*c(1)+...
    B(2,1)*(-1+c(1)^2));
R21num = (1-B(2,1)*(-1+c(1))^2)*(B(1,1)*(-1+c(1))^3-(-3+c(1))*c(1)^2);
R21den1 = 2*c(1)*(B(1,1)*(-1+c(1))^3-(-3+c(1))*c(1)^2);
% % Initialization
h = (tspan(2)-tspan(1))/N;
t = linspace(tspan(1),tspan(2),N+1);
n = 1;
% % Stages initialization with linearly implicit Runge-Kutta
tspan1 = [tspan(1) tspan(1)+c(1)*h]; tspan2 = [tspan(1) tspan(1)+c(2)*h];
[CPUtime_Y1,Y1n,Y1vect,tstage1] = lin_imp_RK(1,tspan1,y0,p);
[CPUtime_Y2,Y2n,Y2vect,tstage2] = lin_imp_RK(1,tspan2,y0,p);
y = [Y2n];
M1tmh = h*jacob(t(n)+c(1)*h,Y1n,p);
Y1nm1 = Y1n; Y2nm1 = Y2n;
% % Method
C = cputime;
for n = 2:N
    f1 = funz(t(n-1)+h*c(1),Y1nm1,p);
    f2 = funz(t(n-1)+h*c(2),Y2nm1,p);
    Y1n = B(1,1)*Y1nm1+B(1,2)*Y2nm1+h*A(1,1)*f1+h*A(1,2)*f2;
    M1t = h*jacob(t(n)+c(1)*h,Y1n,p);
    A21num2 = (2*Id+B(1,1)*(-1+c(1))*(-2*Id+(-1+c(1))*M1tmh)-...
        c(1)*(2*Id+c(1)*M1tmh));
    A21num = -(-1+B(2,1)*(-1+c(1))^2)*(B(1,1)*(-1+c(1))^3-...
        (-3+c(1))*c(1)^2)*A21num2;
    A21den1 = 2*(-1+c(1))*(B(1,1)*(-1+c(1))^3-(-3+c(1))*c(1)^2)*(B(1,1)*M1t+...
        c(1)*(2*Id+B(1,1)*(-2+c(1))*M1t-c(1)*M1t));
    A22num2 = -(3+B(2,1)*(-1+c(1))^2-2*c(1))*(B(1,1)*(-1+c(1))^3-...
        (-3+c(1))*c(1)^2)*A21num2;
    R21den2 = (B(1,1)*(-1+c(1))^3-(-3+c(1))*c(1)^2)*A21num2;
    switch form
        case 1
            % % Using classic method formulation
            Iden = inv(Id+(1/a21cden)*(A21den1*M1tmh+A21den2*M1t));
            A21 = (a21cnum/a21cden)*Iden*(Id+(1/a21cnum)*A21num*M1t);
            A22 = (a22cnum/a21cden)*Iden*(Id+(1/a22cnum)*...
                (A22num1*M1tmh+A22num2*M1t));
            R21 = (r21cnum/r21cden)*inv(Id +(1/r21cden)*...
                (R21den1*M1tmh+R21den2*M1t))*(Id+(1/r21cnum)*R21num*M1tmh);
            Y2n = B(2,1)*Y1nm1+B(2,2)*Y2nm1+h*A21*f1+h*A22*f2+...
                h*R21*funz(t(n)+h*c(1),Y1n,p);
        case 2
            % % Using LU decomposition
            Q1 = Id+(1/a21cden)*(A21den1*M1tmh+A21den2*M1t);
            [L1,U1] = lu(Q1);
            A21NUM = (a21cnum/a21cden)*(Id+(1/a21cnum)*A21num*M1t);
            A22NUM = (a22cnum/a21cden)*(Id+(1/a22cnum)*(A22num1*M1tmh+A22num2*M1t));
            k1 = A21NUM*h*f1; x1 = U1\(L1\k1);
            k2 = A22NUM*h*f2; x2 = U1\(L1\k2);
            R21NUM = (r21cnum/r21cden)*(Id+(1/r21cnum)*R21num*M1tmh);
            Q2 = Id +(1/r21cden)*(R21den1*M1tmh+R21den2*M1t);
            k3 = R21NUM*h*funz(t(n)+h*c(1),Y1n,p); x3 = Q2\k3;
```

```
71              Y2n = B(2,1)*Y1nm1+B(2,2)*Y2nm1+x1+x2+x3;
72      end
73      M1tmh = M1t;
74      Y1nm1 = Y1n; Y2nm1 = Y2n;
75      y = [y Y2n];
76  end
77  Cf = cputime;
78  CPUtime = CPUtime_Y1+CPUtime_Y2+Cf-C;
79  y = [y0 y];
80  yT = Y2n;
81  end
```

4 Example of usage

In this section, we show the application of the previously illustrated codes on the following test ODEs system:

$$\begin{cases} \dfrac{dy_1}{dt} = -2y_2y_3, \\ \dfrac{dy_2}{dt} = \dfrac{5}{4}y_1y_3, \\ \dfrac{dy_3}{dt} = -\dfrac{1}{2}y_1y_2. \end{cases} \tag{17}$$

The ODEs system (17) is known as Euler's model [19] and is related to the rotational motion of solid bodies. We take $t \in [0, 10]$, and $y(0) = (1, 0, 0.9)$ as initial condition. Easily, note that the Jacobian related to the Euler's model is

$$J = \begin{pmatrix} 0 & -2y_3 & -2y_2 \\ \dfrac{5}{4}y_3 & 0 & \dfrac{5}{4}y_1 \\ \dfrac{-1}{2}y_2 & \dfrac{-1}{2}y_1 & 0 \end{pmatrix}. \tag{18}$$

Then, we create a script named `example.m` in which we apply the linearly implicit RK and Peer methods Matlab codes described in Sections 2 and 3, respectively, as reported. This script, which we report below, can in principle contain a list of test problems to apply the methods to, as mentioned previously. In this case, since we solve Euler's model, we report only this test problem in the script, which corresponds to set `p=1`. From line 7 to 10, we initialize the integration interval, the solution at the starting grid point and the reference solution at the end point, respectively. This latter solution can be computed, for example, using the Matlab function `ode15s` requiring maximum accuracy. In line 12 we choose the number of grid intervals into which we divide $[t_0, T]$. In this case, we will apply the methods using 2^4 intervals, then 2^5, and so on, up to 2^{15}. So, we define `inN=4` and `finN=15`. In the vectors defined in lines 13 and 14, initially empty, we will put the error and CPU time of each method for each number of selected grid intervals. For example, `errT_RK` will contain in the first component the absolute error of the RK method for $N = 2^4$, in the second for $N = 2^5$, and so on up to $N = 2^{15}$ in the last component. From line 15 to 25 we apply the RK and Peer methods by calling the functions `lin_imp_RK.m` and `lin_imp_peer.m` shown and described in the previous sections, for several values of N. Finally, we print the vectors with the errors and the CPU times of the methods, then plotting the numerical solution computed by them in correspondence of the last value of N.

```
1  %% Script example.m to apply the linearly implicit Runge-Kutta and Peer methods
2  %% Selecting the problem to solve
3  p = 1;
4  switch p
5      case 1
6          fprintf('Euler''s model\n')
7          tspan = [0 10];
8          y0 = [1;0;0.9];
9          YR = [0.89018057222794;0.36018966256315;0.87069246166083];
10 end
11 %% Initialization
12 inN = 4; finN = 15;
13 errT_RK = []; errT_peer = [];
14 CPUtime_RK = []; CPUtime_peer = [];
15 for N = 2.^(inN:finN)
16     %% Application of linearly implicit Runge-Kutta and Peer methods
17     [CPU_RK,yT_RK,y_RK,t] = lin_imp_RK(N,tspan,y0,p);
18     [CPU_peer,yT_peer,y_peer,t] = lin_imp_peer(N,tspan,y0,p);
19     %% Computation of absolute errors in norm two
20     errT_RK = [errT_RK norm(yT_RK-YR,2)];
21     errT_peer = [errT_peer norm(yT_peer-YR,2)];
22     %% CPU time spent by the methods
```

```
    CPUtime_RK = [CPUtime_RK CPU_RK];
    CPUtime_peer = [CPUtime_peer CPU_peer];
end
%% Printing vectors with methods error and CPU time
format short e
errT_RK
errT_peer
CPUtime_RK
CPUtime_peer
%% Plots of the numerical solution computed by the methods
switch p
    case 1
        figure(1)
        plot(t,y_RK)
        title('RK')
        xlabel('t')
        ylabel('y(t)')
        legend('y_1','y_2','y_3')
        figure(2)
        plot(t,y_peer)
        title('Peer')
        xlabel('t')
        ylabel('y(t)')
        legend('y_1','y_2','y_3')
end
```

As mentioned before, `lin_imp_RK.m` and `lin_imp_peer.m` call the functions `funz.m` and `jacob.m`, which are illustrated in Table 4. These functions return respectively the vector field f and the Jacobian related to the selected problem `p`, evaluated at the point `(t,y)`. Below we show the function `funz.m`, which returns the column vector field `yp` (17) related to Euler's model.

```
function yp = funz(t,y,p)
switch p
    case 1
        %% Euler's model
        yp(1) = -2*y(2)*y(3);
        yp(2) = 5/4*y(3)*y(1);
        yp(3) = -1/2*y(1)*y(2);
        yp = [yp(1);yp(2);yp(3)];
end
end
```

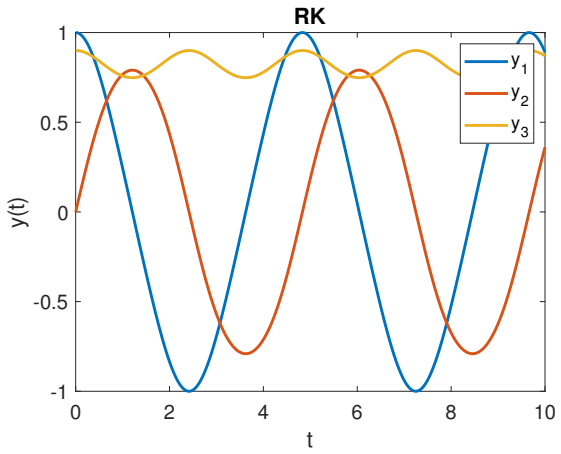
Furthermore, we also show the function `jacob.m`, which returns the Jacobian (18) of Euler's model.

```
function J = jacob(t,y,p)
switch p
    case 1
        %% Euler's model
        J = [0 -2*y(3) -2*y(2);
            5/4*y(3) 0 5/4*y(1);
            -1/2*y(2) -1/2*y(1) 0];
end
end
```

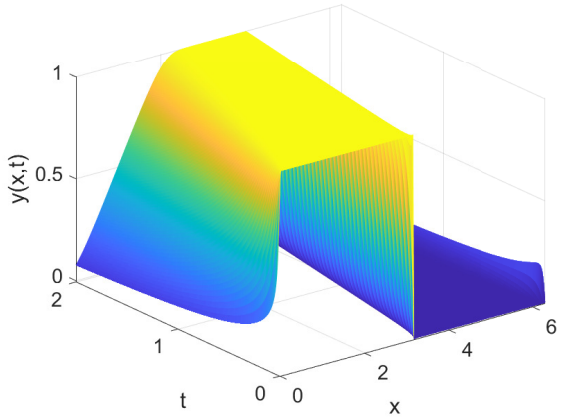
Finally, we show below the Matlab Command Window with the execution of the script `example.m`. We also report in Figure 1(a) the solution plot related to the three-stage linearly implicit RK method.

```
>> example
Euler's model
errT_RK =
  Columns 1 through 5
   8.3031e-03   3.9712e-04   2.2997e-05   1.3836e-06   8.5131e-08
  Columns 6 through 10
   5.2863e-09   3.2934e-10   2.0478e-11   1.1941e-12   8.6611e-14
  Columns 11 through 12
   2.4126e-13   9.1259e-14
errT_peer =
  Columns 1 through 5
   6.2815e-01   7.2235e-02   9.4716e-03   1.2136e-03   1.5428e-04
```

```
  Columns 6 through 10
    2.0180e-05   3.0052e-06   5.7452e-07   1.3315e-07   3.2973e-08
  Columns 11 through 12
    8.2752e-09   2.0878e-09
CPUtime_RK =
  Columns 1 through 5
    9.3750e-02   1.5625e-02            0   3.1250e-02            0
  Columns 6 through 10
    2.0312e-01   4.0625e-01   4.5312e-01   6.2500e-01   5.3125e-01
  Columns 11 through 12
    1.0469e+00   1.4219e+00
CPUtime_peer =
  Columns 1 through 5
    1.5625e-02            0            0   3.1250e-02   4.6875e-02
  Columns 6 through 10
    2.1875e-01   2.9688e-01   3.1250e-01   5.6250e-01   6.5625e-01
  Columns 11 through 12
    7.3438e-01   1.4844e+00
```



(a) Numerical solution of (17) computed by the three-stage RK.



(b) Reference solution of (19) computed by the Matlab function `ode15s`.

Figure 1: Numerical solution of Euler's model computed by the three-stage linearly implicit RK method in correspondence of $N = 2^{15}$ on the left, and profile of the reference solution of the Burgers' PDE (19) computed by the Matlab function `ode15s` on the right.

5 Numerical results

We apply the methods considered in this paper to the following Burgers' PDE [1], endowed with periodic boundary conditions:

$$\frac{\partial y}{\partial t} = \epsilon \frac{\partial^2 y}{\partial x^2} - \frac{1}{2}\frac{\partial y^2}{\partial x}, \qquad (x,t) \in [x_0, X] \times [t_0, T] = [0, 2\pi] \times [0, 2]. \tag{19}$$

Furthermore, we consider as initial conditions

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

Obviously, in order to apply the linearly implicit RK and Peer methods, we need to perform a spatial semi-discretization of the equation. In our case, we consider central finite differences of order four for both the first order spatial derivative and the second order spatial derivative. Then, by fixing the uniform spatial grid $\{x_n = x_0 + mk; m = 0, ..., M; x_M = X\}$, the Burgers' PDE (19) becomes the following ODEs system of size $M + 1$:

$$y'(t) = \epsilon L_1 y(t) - \frac{1}{2} L_2 y(t)^2. \tag{20}$$

Then, 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:

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

Note also that trivially, from (20), the Jacobian is the non-constant matrix

$$J(t, y(t)) = \epsilon L_1 - L_2 y(t).$$

To perform the numerical tests, we set $\epsilon = 10^{-2}$ and $M = 2^8$, which are parameters corresponding to a stiff case.

In addition to comparing the Peer and RK methods in their classical and optimized formulation, we also apply the famous two-stage fully implicit RK based on Gaussian collocation points [21] of order four, which is one of the most used methods to solve first order initial value problems, due to its excellent accuracy and stability properties. This method has the following Butcher tableau:

$$\begin{array}{c|cc} \frac{3-\sqrt{3}}{6} & \frac{1}{4} & \frac{3-2\sqrt{3}}{12} \\ \frac{3+\sqrt{3}}{6} & \frac{3+2\sqrt{3}}{12} & \frac{1}{4} \\ \hline & \frac{1}{2} & \frac{1}{2} \end{array} . \tag{21}$$

Since the RK of Gauss is fully implicit, we use Newton's iterations to solve the non-linear system of stages at each step, requiring accuracy equal to 10^{-8} for the computation of the related solution. We apply the RK of Gauss because, in addition to showing the good properties of the numerical schemes analyzed in this work and the efficiency of the related proposed Matlab codes, we also want to show their advantages over existing good and well known methods. Then, let us recall in Table 6 the schemes that are used in this section.

Table 6: Methods used in numerical tests.

Method	Coefficients
Fully implicit RK of Gauss (RKG)	(21)
Classical linearly implicit two-stage RK (CRK2)	$c_2 = 1$ (see Table 1), (11)
Optimized linearly implicit two-stage RK (ORK2)	$c_2 = 1$ (see Table 1), (11), Section 2.1.1
Classical linearly implicit three-stage RK (CRK3)	$c_2 = 1/2$, $c_3 = 1$ (see Table 1), (12)-(13)
Optimized linearly implicit three-stage RK (ORK3)	$c_2 = 1/2$, $c_3 = 1$ (see Table 1), (12)-(13), Section 2.1.2
Classical linearly implicit Peer (CP)	$b_{11} = -0.24$, $b_{21} = -0.31$, $c_1 = 0.2$, (14)-(15)-(16)
Optimized linearly implicit Peer (OP)	$b_{11} = -0.24$, $b_{21} = -0.31$, $c_1 = 0.2$, (14)-(15)-(16), Section 3.1.1

For each method, we determine the absolute error at the end time grid point $T = 2$, evaluated as the difference between the numerical solution $yT \approx y(T)$ and the one (that we call yR, as it represents the reference solution) computed by the Matlab function `ode15s` applied to semi-discretized ODEs system (20), requiring maximum accuracy. We also estimate the order of each method, calculated using the formula

$$p(h) = \frac{cd(h) - cd(2h)}{log_{10}(2)},$$

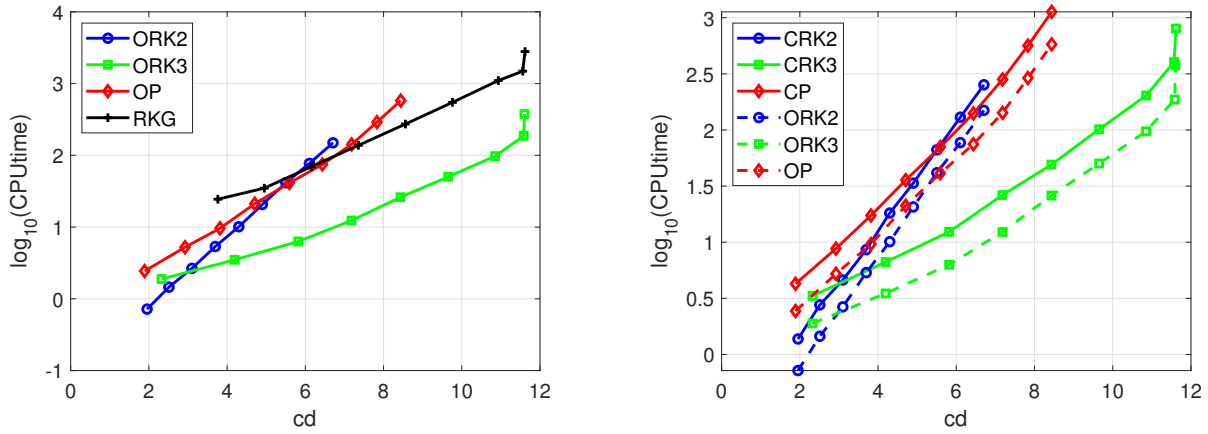
where $cd(h) = -log_{10}(|yT - yR|)$ is the number of correct digits obtained with time step-size h.

Table 7: Absolute error and estimated order on the semi-discretized Burgers' PDE (20) in correspondence of several values of the time-step size h (and then of the number of time grid intervals N), by the optimized linearly implicit RK and Peer methods and the RKG.

	ORK2		ORK3		OP		RKG	
h (N)	$\|yT - yR\|$	$p(h)$	$\|yT - yR\|$	$p(h)$	$\|yT - yR\|$	$p(h)$	$\|yT - yR\|$	$p(h)$
3.1250e-02 (2^6)	1.1130e-02	-	4.7498e-03	-	1.2861e-02	-	1.7270e-04	-
1.5625e-02 (2^7)	3.0787e-03	1.8541	6.4084e-05	6.2118	1.1918e-03	3.4319	1.1170e-05	3.9505
7.8125e-03 (2^8)	7.9555e-04	1.9523	1.5232e-06	5.3948	1.5171e-04	2.9737	7.0399e-07	3.9880
3.9062e-03 (2^9)	2.0055e-04	1.9880	6.5970e-08	4.5291	1.9620e-05	2.9510	4.4090e-08	3.9970
1.9531e-03 (2^{10})	5.0243e-05	1.9970	3.6852e-09	4.1620	2.5629e-06	2.9365	2.7576e-09	3.9990
9.7656e-04 (2^{11})	1.2567e-05	1.9992	2.2191e-10	4.0537	3.6533e-07	2.8105	1.7293e-10	3.9951
4.8828e-04 (2^{12})	3.1422e-06	1.9998	1.4013e-11	3.9852	6.5451e-08	2.4807	1.1612e-11	3.8965
2.4414e-04 (2^{13})	7.8558e-07	2.0000	2.5999e-12	2.4302	1.4724e-08	2.1523	2.7597e-12	2.0731
1.2207e-04 (2^{14})	1.9640e-07	2.0000	2.4929e-12	-	3.6335e-09	2.0187	2.4156e-12	-

In Figure 1(b) we represent the profile of the reference solution. Note that the initial discontinuity at $(\pi, 0)$ contributes to the stiffness of the problem, as there is a sudden change in the behavior of the function. We report in Table 7 and in Figure 2 the results obtained by applying all the discussed methods. Specifically, in Table 7 we report the absolute error and the estimated order related to the optimized linearly implicit methods and the RKG, by varying the time step-size h, and therefore the number of grid intervals N. In Figure 2 we report two work precision diagrams with the estimated number of correct digits on the abscissa axis, and with the CPU time in logarithmic basis on the ordinate axis, comparing all the methods of Table 6.

From Table 7, note that all methods exhibit their theoretical order. In fact, the RKG and the OR3 have order four. The OR2 method has order two. However, it is interesting to note the behavior of the order of the OP method. In fact, although this method formally has order two, it exhibits order three for most of the values of h used. This happens because, as mentioned in



(a) ORK2, ORK3, OP and RKG.

(b) CRK2, CRK3, CP versus ORK2, ORK3, OP.

Figure 2: Work precision diagrams with the estimated number of correct digits on the abscissa axis, and with the CPU time in logarithmic basis on the ordinate axis; on the left we compare the analyzed linearly implicit methods with the RKG, while on the right we compare the classic formulations of these methods with the optimized ones derived in this paper.

Section 3 and extensively explained in [17], the second stage is calculated by requiring order three. Hence, since the first stage has order two, as seen asymptotically the overall order remains two, but the behavior of this method is more similar to that of a numerical scheme of order three. Finally, note that the RKG and ORK3 methods produce very similar errors.

From Figure 2, note that on the left we compare the RKG with the linearly implicit methods discussed in this work directly in the optimized formulation, while on the right we show the comparison between the linearly implicit methods in the classical formulation and those in the optimized formulation. It can be deduced that, in relation to the computational cost and the related error, the ORK3 method is decidedly better than the others. Furthermore, the ORK2 and OP methods also produce, at least for the first values of h, results comparable to the RKG, despite the latter having a higher order. Finally, we observe that the optimized formulation of the linearly implcit methods actually allows to lower the computing times. For example, the three-stage RK method reaches, in the optimized formulation, more than four correct digits at the same time as, in the classical formulation, it reaches just two.

To perform numerical tests, we have used a laptop with a RAM of 8GB and an operative system of 64 bit. The processor is AMD Ryzen 7 3700U with Radeon Vega Mobile Gfx 2.30 GHz. The version of Matlab used is R2022a.

6 Conclusions

In this work we have provided the Matlab codes for the application of two classes of linearly implicit methods derived by means of the EF approach for the numerical solution of ODEs, and then also semi-discretized PDEs, characterized by stiffness. We have shown that, by manipulating the methods in their classical formulation, it is possible to obtain an equivalent formulation which considerably reduces their computational cost.

The linearly implicit RK methods [28] that we have discussed are very promising, as they are much cheaper than totally implicit methods, requiring only the resolution of s linear systems at each step (where s is the number of stages), and despite that they result A-stable. In fact, in numerical tests we have shown that the three-stage RK can be more beneficial than the well known RKG. Furthermore, we have made a comparison of such linearly implicit RK methods with a class of linearly implicit Peer methods [17] obtained with the same EF-based idea. By firstly providing for the latter an adequate starting procedure, we have shown that they have very good accuracy and stability properties, resulting also competitive with the linearly implicit RK and the RKG by the point of view of the computing times.

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*.

References

[1] Burgers, J.M.: A Mathematical Model Illustrating the Theory of Turbulence. Adv. Appl. Mech., **1**, 171–199 (1948)

[2] Butcher, J.C.: Implicit Runge-Kutta processes. Math. Comp., **18**, 50–64 (1964)

[3] Butcher, J.C.: Numerical Methods for Ordinary Differential Equations. 2nd edn. Wiley, Chichester (2008)

[4] Butcher, J.C.: General linear methods. Acta Numer., **15**, 157–256 (2006)

[5] Calvo, M.P., Gerisch, A.: Linearly implicit Runge-Kutta methods and approximate matrix factorization. Appl. Numer. Math., **53**(2-4), 183–200 (2005)

[6] Cardone, A., Jackiewicz, Z., Sandu, A., Zhang, H.: Extrapolated Implicit-Explicit Runge-Kutta Methods. Math. Model. Anal., **19**(1), 18–43 (2014)

[7] Cardone, A., Jackiewicz, Z., Sandu, A., Zhang, H.: Extrapolation-based implicit-explicit general linear methods. Numer. Algorithms, **65**(3), 377–399 (2014)

[8] Cardone, A., D'Ambrosio, R., Paternoster, B.: Exponentially fitted IMEX methods for advection-diffusion problems. J. Comput. Appl. Math., **316**, 100–108 (2017)

[9] Bras, M., Cardone, A., Jackiewicz, Z., Pierzchala, P.: Error propagation for implicit-explicit general linear methods. Appl. Numer. Math., **131**, 207–231 (2018)

[10] R. Cavoretto, A. De Rossi. Software for Approximation 2022 (SA2022). *Dolomites Res. Notes Approx.*, Special Issue SA2022, 15:i–ii, 2022.

[11] Conte, D., D'Ambrosio, R., Moccaldi, M., Paternoster, B.: Adapted explicit two-step peer methods. J. Numer. Math., **27**(2), 69–83 (2018)

[12] Conte, D., D'Ambrosio, R., Pagano, G., Paternoster, B.: Jacobian-dependent vs Jacobian-free discretizations for nonlinear differential problems. Comput. Appl. Math., **39**(3), 171 (2020)

[13] Conte, D., Mohammadi, F., Moradi, L., Paternoster, B.: Exponentially fitted two-step peer methods for oscillatory problems. Comput. Appl. Math., **39**(3), 174 (2020)

[14] Conte, D., D'Ambrosio, R., Giordano, G., Ixaru, L.G., Paternoster, B.: User-Friendly Expressions of the Coefficients of Some Exponentially Fitted Methods. Lect. Notes Comput. Sci. (249529), 47–62 (2020)

[15] Conte, D., Pagano, G., Paternoster, B.: Jacobian-dependent two-stage peer method for ordinary differential equations. Lect. Notes Comput. Sci. (12949), 309–324 (2021)

[16] Conte, D., Frasca-Caccia, G.: Exponentially fitted methods that preserve conservation laws. Commun. Nonlinear. Sci. Numer. Simul., **109**, 106334 (2022)

[17] Conte, D., Pagano, G., Paternoster, B.: Two-step peer methods with equation-dependent coefficients. Comput. Appl. Math., **41**, 140 (2022)

[18] D'Ambrosio, R., Ixaru, L.G., Paternoster, B.: Construction of the EF-based Runge-Kutta methods revisited. Comput. Phys. Commun., **182**(2), 322–329 (2011)

[19] Euler, L.: Du mouvement de rotation des corps solides autour d'un axe variable. Memoires de l'Academie des Sciences der Berlin, **14**, 154–193 (1758)

[20] Hairer, E., Norsett, S.P., Wanner, G.: Solving Ordinary Differential Equations I: Nonstiff Problems. 2nd edn. Springer, Berlin (1993)

[21] Hairer, E., Wanner, G.: Solving Ordinary Differential equations II: Stiff and Differential-Algebraic Problems. Springer, Berlin (2002)

[22] Hundsdorfer, W., Verwer, J.: Numerical Solution of Time-Dependent Advection-Diffusion-Reaction Equations. Springer Series in Computational Mathematics, Berlin (2003)

[23] Gonzalez-Pinto, S., Hernandez-Abreu, D., Montijano, J.I.: An efficient family of strongly A-stable Runge-Kutta collocation methods for stiff systems and DAEs. Part I: Stability and order results. J. Comput. Appl. Math., **234**(4), 1105–1116 (2010)

[24] Gonzalez-Pinto, S., Hernandez-Abreu, D., Perez-Rodriguez, S.: W-methods to stabilize standard explicit Runge-Kutta methods in the time integration of advection-diffusion-reaction PDEs. J. Comput. Appl. Math., **316**, 143–160 (2017)

[25] Gonzalez-Pinto, S., Hernandez-Abreu, D., Perez-Rodriguez, S.: AMFR-W-methods for parabolic problems with mixed derivatives. Applications to the Heston model. J. Comput. Appl. Math., **387**, 112518 (2021)

[26] Ixaru, L.Gr.: Operations on oscillatory functions. Comput. Phys. Commun., **105**, 1–19 (1997)

[27] Ixaru, L.Gr., Vanden Berghe, G.: Exponential fitting. Springer (2004)

[28] Ixaru, L.Gr.: Runge-Kutta methods with equation dependent coefficients. Comput. Phys. Commun., **183**(1), 63–69 (2012)

[29] Jebens, S., Weiner, R., Podhaisky, H., Schmitt, B.: Explicit multi-step peer methods for special second-order differential equations. Appl. Math. Comput., **202**(2), 803–813 (2008)

[30] Klinge, M., Weiner, R., Podhaisky, H.: Optimally zero stable explicit peer methods with variable nodes. BIT Numer. Math., **58**(2), 331–345 (2017)

[31] Klinge, M., Hernandez-Abreu, D., Wiener, R.: A comparison of one-step and two-step W-methods and peer methods with approximate matrix factorization. J. Comput. Appl. Math., **387**, 112519 (2021)

[32] Rosenbrock, H.H.: Some general implicit processes for the numerical solution of differential equations. Comput. J., **5**(4), 329–330 (1963)

[33] Sanz-Serna, J.M., Verwer, J.G., Hundsdorfer, W.H.: Convergence and Order Reduction of Runge-Kutta Schemes Applied to Evolutionary Problems in Partial Differential Equations. Numer. Math., **50**, 405–418 (1986)

[34] Weiner, R., Biermann, K., Schmitt, B., Podhaisky, H.: Explicit two-step peer methods. Comput. Math. with Appl., **55**(4), 609–619 (2008)