Solving nonlinear parabolic PDEs in several dimensions: parallelized ESERK codes

J. Martín-Vaquero[a,*], A. Kleefeld[b]

[a] *ETS Ingenieros industriales, Universidad de Salamanca. E37700, Bejar, Spain*
[b] *Forschungszentrum Jülich GmbH, Jülich Supercomputing Centre. Wilhelm-Johnen-Straße, 52425 Jülich, Germany*

Abstract

There is a very large number of very important situations which can be modeled with nonlinear parabolic partial differential equations (PDEs) in several dimensions. In general, these PDEs can be solved by discretizing in the spatial variables and transforming them into huge systems of ordinary differential equations (ODEs), which are very stiff. Therefore, standard explicit methods require a large number of iterations to solve stiff problems. But implicit schemes are computationally very expensive when solving huge systems of nonlinear ODEs. Several families of Extrapolated Stabilized Explicit Runge-Kutta schemes (ESERK) with different order of accuracy (3 to 6) are derived and analyzed in this work. They are explicit methods, with stability regions extended, along the negative real semi-axis, quadratically with respect to the number of stages s, hence they can be considered to solve stiff problems much faster than traditional explicit schemes. Additionally, they allow the adaptation of the step length easily with a very small cost.

Two new families of ESERK schemes (ESERK3 and ESERK6) are derived, and analyzed, in this work. Each family has more than 50 new schemes, with up to 84.000 stages in the case of ESERK6. For the first time, we also parallelized all these new variable step length and variable number of stages algorithms (ESERK3, ESERK4, ESERK5, and ESERK6). These parallelized strategies allow to decrease times significantly, as it is discussed and also shown numerically in two problems. Thus, the new codes provide very good results compared to other well-known ODE solvers. Finally, a new strategy is proposed to increase the efficiency of these schemes, and it is discussed the idea of combining ESERK families in one code, because typically, stiff problems have different zones and according to them and the requested tolerance the optimum order of convergence is different.

Keywords: Higher-order codes, Multi-dimensional partial differential

*Corresponding author
Email addresses: jesmarva@usal.es (J. Martín-Vaquero), a.kleefeld@fz-juelich.de (A. Kleefeld)

equations, Nonlinear PDEs

1. Introduction

There is a very large number of areas where many important problems are modeled with nonlinear partial differential equations (PDEs) in several dimensions. Perhaps, one the most common type of PDEs is the second-order parabolic one. Some examples of areas (but not limited) where complex problems of this type may be needed are: atmospheric phenomena, biology, combustion problems (where the reaction is particularly very explosive), fluid mechanics (including Navier-Stokes problems), lasers, molecular dynamics, nuclear kinetics, in industrial processes of different types, in medicine, in financial mathematics (as PDEs, or SPDEs), in problems related to heat transfer (a dozen sub-areas are mentioned in [4]), and chemical reactions. In fact, perhaps this is the area that appears more extensively in the scientific bibliography [4, 8, 9]. In many cases, to solve this type of nonlinear PDEs, spatial variables are discretized [9] (through finite differences, spectral techniques, alternating direction implicit techniques,...). PDEs are transformed into nonlinear ordinary differential equations (ODEs) of very high dimension, often called semi-discrete systems.

These ODE systems are usually very stiff, because of the spatial discretization of elliptic operators. Hence, traditional explicit methods are usually very slow (it is necessary to use very small length steps, see [8, 9]), otherwise the algorithms are not stable and therefore do not converge to the solution). Therefore, in this type of problems, researchers often consider implicit schemes based on BDF and Runge-Kutta methods with good stability properties. However, if the dimension of the ODE system is very high (usual in several dimensions) it is necessary to solve very large nonlinear systems at each iteration. Recently, numerous techniques have also been proposed based on approximations of matrix exponentials, and explicit-implicit algorithms. However, in both cases there are very costly operations when the system dimension is high (and other considerations should be taken into account): either it is necessary to approximate functions related to exponentials of matrices or decompose very large matrices.

In many of these cases, it is known that the Jacobian eigenvalues of the function are all in a certain type of region. In the case of parabolic problems of the second order, the most common is that these values are all real negative or are very close to this semi-axis. When the Laplacian is discretized using finite differences or some spectral techniques the associated matrix has this type of eigenvalues. In this type of problem, where the nonlinear ODE system has a very high dimension and the eigenvalues of the Jacobian are of this class, stabilized explicit Runge-Kutta methods (also called Runge-Kutta-Chebyshev methods) are a very powerful tool (see [1, 5, 6, 9, 10, 19, 21, 22] and references cited therein).

These types of algorithms are totally explicit, and they have regions of stability extended along the real negative axis. Although schemes typically have order 2 (some of them 3, 4), these integrators have many steps; several of them

are intended to meet the conditions of consistency, and the rest seek to extend as much as possible the region of stability along the negative real axis. In this way, these regions of stability increase quadratically with the number of stages. Thus, the number of steps per step, s, is greater than in a classic Runge-Kutta. However, fewer steps are needed (for stability issues). They have been reduced proportionally with s^2, thus the total computational cost is reduced proportionally with s.

Logically we are looking for simple procedures that allow us to construct algorithms with high order, up to a high number of stages, and with a region of stability as long as possible. But we also need to construct Runge-Kutta methods that have optimal internal stability properties. Lebedev and Finogenov showed that such algorithms often suffer from two types of difficulties [14]: internal stability and error propagation. Later, many research papers have developed algorithms trying to reduce these difficulties: DUMKA [19], RKC [21], ROCK2 [2], ROCK4 [1], or SERK2 [15], for example. However, most of them are second-order, or have some problems with error propagation and internal stability when the number of stages is large.

In [18], recurrence formulas similar to those of [15, 21] were used and combined with extrapolation techniques to obtain methods of order higher than two. In particular, a family of fourth-order methods, and an algorithm with these schemes were derived. In [17], a similar procedure was employed to develop a family with more than 40 different fifth-order methods, and excellent stability properties and numerical results.

In this work, we continue with the derivation and analysis of more families of extrapolated stabilized Runge-Kutta (ESERK) methods: we obtained 4 families, with third, fourth, fifth and sixth orders of convergence. In total they are more than 200 algorithms since, at each family, we built the algorithms for $s = 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 25, 30, 35, 40, 45, 50, 60, 70, 80, 90, 100, 150, 200, 250, 300, 350, 400, 450, 500, 600, 700, 800, 900, 1000, 1200, 1400, 1600, 1800, 2000, 2200, 2400, 2600, 2800, 3000, 3200, 3400, 3600, 3800, 4000$, s being the number of stages of each first-order stabilized Runge-Kutta (SERK) algorithm (that we extrapolate to increase the order of convergence), $n_t = n_p \times s$ the number of stages of the ESERK method, and $n_p = 6, 10, 15, 21$ respectively for orders $p = 3, 4, 5, 6$ of convergence. Two of these families (those with third- and sixth-order) schemes are completely new. We provide details for their construction and analysis in Section 2. In Section 3, we explain how to develop efficient codes in parallel to decrease computational times. These new codes automatically choose the length step and the optimum number of stages at each step of the Runge-Kutta method. It is tested in Section 4, comparing the new parallel algorithm with the previous sequential ones, and also with other excellent ODE solvers such us RKC, ROCK4, IRKC, and PIROCK. These four parallel codes are freely available under:
https://github.com/kleefeld80/ESERK3-6parallel.git so anybody can use them.

Finally, we compare numerically the parallel algorithms for one problem and several different zones of integration: (i) one interval where solutions change very fast because of the transient terms, (ii) and another where solutions are

smoother. We check how our codes behave in a different way depending on the order of convergence and the type of interval. This fact suggests that it might be interesting to combine all these families in one code. Some discussion and conclusions are provided in Section 5.

2. Construction of extrapolated explicit methods

There is a large amount of areas where many situations are modeled with nonlinear parabolic equations satisfying

$$u_t = \operatorname{div}(a(x,t)\nabla u) + f(x,t,u), \quad \forall (x,t) \in Q_T = \Omega \times [0,T],$$

together with some initial and boundary conditions. This is the reason many researchers have developed and analyzed so many numerical procedures to solve these PDEs. When $f(x,t,u)$ clearly affects the numerical solution of the PDE, and Ω is not a complicated region, most often spatial discretizations are considered, and the PDE is transformed into a system of ODEs. However, $\Omega \subset R^n$ and for most of the real problems $n = 2, 3$, and therefore the system of ODEs has a very large dimension. Alternating Direction Implicit (ADI) schemes might be another different option, however the order of convergence of these methods is low.

However, in some cases, we would like to obtain more accurate solutions. Hence, our goal is developing higher-order methods for stiff, very large systems of ODEs, which can be used after semi-discretizing (discretizing in the spatial variables) with higher-order schemes the previous parabolic PDEs. In this section, we shall explain how to build higher-order ESERK methods.

The main ingredient is the stabilized explicit Runge-Kutta algorithms (SERK) derived in [11, 12, 15, 16], but these previous SERK schemes were only second-order. In [18], first-order SERK schemes were combined with extrapolation techniques to build a fourth-order ESERK scheme (ESERK4). And in [17], a similar procedure was employed to develop a family with more than 40 different fifth-order methods (ESERK5). For this new work, we added new algorithms to both previous families ESERK4 and ESERK5, and constructed other two whole new families: ESERK3 and ESERK6 with more than 50 new methods each one.

2.1. Construction of first-order stabilized explicit methods

First-order stabilized explicit Runge-Kutta (SERK) methods have been derived in several previous works (see [20, 14, 13] and references therein). Most of them use Chebyshev polynomials of the first kind of order s (s = stages), which are defined by the recursion:

$$T_0(x) = 1, \quad T_1(x) = x, \quad T_s(x) = 2xT_{s-1}(x) - T_{s-2}(x). \tag{1}$$

If we denote

$$R_s(z) = \frac{T_s(w_{0,s} + w_{1,s}z)}{T_s(w_{0,s})}, \quad w_{0,s} = 1 + \frac{\mu_p}{s^2}, \quad w_{1,s} = \frac{T_s(w_{0,s})}{T_s'(w_{0,s})}, \tag{2}$$

(s being the number of stages of the first-order method, z is a function of x depending on this values s) we obtain polynomials (the so-called shifted Chebyshev polynomials) oscillating between $-\lambda_p$ and λ_p (for a value $0 < \lambda_p < 1$ which depends on μ_p) in a region which is $O(s^2)$, and $R_s(z) = 1 + z + O(z^2)$ (as it is explained, for example in [8]). In the following subsection, we will address how to calculate these values of λ_p and μ_p to guarantee the stability not only of the first-order SERK methods, but also the higher-order ESERK schemes.

Obviously, we also have to construct explicit Runge-Kutta schemes with these $R_s(z)$ as stability functions. If we are able to develop these methods, they would be first-order numerical solvers (because $R_s(z) = 1 + z + \ldots$, and therefore $\sum_{j=1}^{s} b_j = 1$ (b_j being the solutions of (5), see Theorem 1 in [11]) with stability regions extended quadratically along the negative real semi-axis. Actually $|R_s(z)| < \lambda_p < 1$ for $z \in [-l_{s,p}, 0]$, which will be very useful in Section 2.2 when we derive higher-order methods through extrapolation.

These explicit Runge-Kutta methods, with internal stability regions given by the Chebyshev polynomials of the first kind, can be obtained through a three-term recurrence formula similar to Equation (1). We only have to remember that the stability functions of identity operator, g_s and $hf(\cdot)$ are respectively 1, $T_s(x)$ and $\bar{x}$ ($\bar{x}$ defined from $z = 1 + \alpha_p \bar{x}$, for α_p values calculated in Section 2.2), thus we obtain the Runge-Kutta methods with internal stability functions given by Equation (1):

$$\begin{aligned}
g_0 &= y_n, \\
g_1 &= g_0 + \alpha_p h f(g_0), \\
g_j &= 2g_{j-1} - g_{j-2} + 2\alpha_p h f(g_{j-1}), \qquad j = 2, \ldots, m, \\
g_{m+1} &= g_m + \alpha_p h f(g_m), \\
g_j &= 2g_{j-1} - g_{j-2} + 2\alpha_p h f(g_{j-1}), \qquad j = m+2, \ldots, 2m, \\
&\ldots \\
g_{qm+1} &= g_{qm} + \alpha_p h f(g_{qm}), \\
g_j &= 2g_{j-1} - g_{j-2} + 2\alpha_p h f(g_{j-1}), \qquad j = qm+2, \ldots, s.
\end{aligned} \tag{3}$$

Finally, we only need to calculate the value of the new approximation $y_{n+1} \simeq y(t_{n+1})$ as

$$y_{n+1} = \sum_{j=0}^{s} b_j g_j \tag{4}$$

where b_j are the solutions of the linear system

$$R_s(z) = b_0 T_0 + \sum_{j=1}^{q} \sum_{i=1}^{m} \left(b_{i+m(j-1)} T_i T_m^{j-1} \right) + \sum_{j=mq+1}^{s} \left(b_j T_{j-mq} T_m^q \right), \tag{5}$$

where $T_i = T_i(1 + z/(\alpha_p s^2)$. And, in this way, it is easy to demonstrate that the stability regions of these methods (Equation (4)) are given by $R_s(z)$, in a similar way as it was done for ESERK4 in Theorem 1 [11].

2.2. Construction of higher-order ESERK schemes

Previously, we did not explain how to calculate λ_p, nor how to choose the α_p and μ_p values. All of these values are related with the procedure to derive higher-order methods using Richardson's extrapolation:

Let us suppose that we have to compute the numerical results of any initial value problem (IVP), $y' = f(t, y)$, $y(x_0) = y_0$, and we want to approximate $y(x_0 + h)$. We can do it for various different, and constant, step sizes $h_1 > h_2 > h_3 > \ldots$ (taking $h_i = h/n_i$, n_i being a positive integer), until we obtain $y_{h_i}(x_0 + h) := S_{i,1}$.

If the method employed has order p, then the global error of any of these approximations has an asymptotic expansion of the form

$$y(x) - y_h(x) = e_p h^p + e_{p+1} h^{p+1} + \ldots + e_N h^N + \ldots .$$

In our case, we have a procedure to obtain first-order methods, and we would like to increase the order of the methods. The idea is to eliminate as many terms as possible from the asymptotic expansion by solving k linear equations for the unknowns y, e_p, ..., e_{p+k-2}.

Actually, this technique has been analyzed before, and it is well-known that the most economic one is the "harmonic sequence" $(1, 2, 3, 4, 5, \ldots)$, for this reason it is the one employed in this work. When the first scheme is first-order, the most economical algorithm to calculate the $(k+1)-$th method $S_{j,k+1}$ is the Aitken-Neville one:

$$S_{j,k+1} = S_{j,k} + \frac{(j-k)(S_{j,k} - S_{j-1,k})}{k}.$$

In this way, once the first-order approximation is calculated $S_{i,1}$ as g_s in Equation (3) for $h_i = h/i$, $S_{p,p}$ is an approximation with order p. We provided formulae for orders $2, 3, 4, 5$ in [18], page 143. The formula for the sixth-order method can be simplified as

$$\begin{array}{c} S_{6,6} = \frac{6480 y_{h/6}(x_0+h) - 14329 y_{h/5}(x_0+h) + 10240 y_{h/4}(x_0+h)}{120} \\ + \frac{-2430 y_{h/3}(x_0+h) + 160 y_{h/2}(x_0+h) - y_h(x_0+h)}{120}. \end{array} \tag{6}$$

Hence, we can calculate the polynomial of the sixth-order extrapolated method as $P_{6s}(z)$, then:

$$\begin{array}{c} P_{6s}(z) = \frac{-R_s(z) + 160 (R_s(z/2))^2 - 2430 (R_s(z/3))^3}{120} \\ + \frac{10240 (R_s(z/4))^4 - 14329 (R_s(z/5))^5 + 6480 (R_s(z/6))^6}{120}. \end{array} \tag{7}$$

And therefore

$$|P_{6s}(z)| \leq \frac{|R_s(z)| + 160 |R_s(z/2)|^2}{120}$$

$$+\frac{2430|R_s(z/3)|^3+10240|R_s(z/4)|^4+14329|R_s(z/5)|^5+6480|R_s(z/6)|^6}{120}.$$

We calculate λ_6 as the positive real root of the equation

$$\frac{x+160x^2+2430x^3+10240x^4+14329x^5+6480x^6}{120}=0.95.$$

In this way, whenever $|R_s(z)|<\lambda_6\approx 0.25658$, then $|P_{6s}(z)|<0.95$. In a similar way, we calculate the other λ_p values: $\lambda_5=0.277923$, $\lambda_4=0.311688$, $\lambda_3=0.368008$, $\lambda_2\leq\frac{\sqrt{215}-5}{20}=0.483144$.

Once, we know the procedure that we will use to construct ESERK schemes from SERK methods, and we know the λ_p values, we estimate the μ_p values such as $R_s(z)$ obtained from Equation (2) satisfies that $|R_s(x)|<\lambda_p,\ \forall x\in[-l_{s,p},0]$, with $l_{s,p}$ as large as possible. Finally, we take $\alpha_p\approx d_p/2$ (the value $l_{s,p}/s^2$ is almost a constant for large values s, thus we take $d_p=l_{4000,p}/(4000)^2$).

In Table 1, we provide the numerical values of the parameters λ_p, μ_p, d_p and α_p, that you we have used to derive the numerical methods for ESERK3, ESERK4, ESERK5, and ESERK6.

Order	λ_p	μ_p	d_p	α_p
3	0.368008	1.38	1.12006	0.56
4	0.311688	1.6875	1.03479	0.5
5	0.277923	1.92	0.980877	0.49
6	0.25658	2.08	0.94795	0.47

Table 1: Parameters values λ_p, μ_p, d_p and α_p for the derivation of the coefficients of the ESERK methods.

With these values for the parameters λ_p, μ_p, d_p and α_p, with $p=3,\ldots,6$, it is possible to obtain the following results:

Theorem 1. *The sixth-order extrapolated stabilized explicit Runge-Kutta methods, derived through equations (3), (4) and (6), are stable in the interval* $[-2\alpha_6 s^2,0]$ *(not only, but stability zone includes this region), with* $s\leq 4000$. *Additionally, the internal stability, at all the stages, includes this region.*

Proof. Demonstration of this theorem is analogous to the proof of the first part of Theorem 1 in [17]. We simply use that $T_i=T_i(1+z/(\alpha_p s^2))$ satisfy that $|T_i(z)|<1$ in $[-2\alpha_6 s^2,0]$ and $|R_s(z)|<|P_{6s}(z)|<1,\ \forall x\in[-l_{s,p},0]$. Finally $[-2\alpha_6 s^2,0]\subset[-l_{s,p},0]$. □

Remark: In a similar way, it is possible to obtain that $p-$th-order extrapolated stabilized explicit Runge-Kutta methods derived above are stable in the interval $[-2\alpha_p s^2,0]$.

Theorem 2. *Since the values* $S_{j,k}$ *represent a numerical method of order* k*, the schemes obtained with equations (3), (4), and (6) (analogously formulas given in [18], page 143, to obtain* $S_{3,3}$*,* $S_{4,4}$*, and* $S_{5,5}$*) converge with sixth-order (third-, fourth- and fifth-order of convergence), whenever the numerical*

methods are stable, and the right hand term in the system of ODEs is seven times continuously differentiable, $\mathcal{C}^7$ ($\mathcal{C}^{p+1}$).

Proof. The demonstration can be done in a similar way as in Theorems 9.1 and 9.2 in [7]. In this case, we know that $S_{1,1}$ given by SERK methods are first-order since $R_s(z) = 1 + z + a_{2,s}z^2 + \ldots$. □

3. Parallel, variable-step and number of stages ESERK algorithm

In this section, we shall explain how to use the methods derived above to build parallel and sequential, variable-step and variable-number of stages ESERK codes.

3.1. Decreasing memory demand

One can easily store the g_j values, with $j = 0, \ldots, s_{\max}$ (see Equation (3)) in a two-dimensional array of size $N \times s_{\max}$ where N is the length of the system of ODEs (obtained after spatial discretization of the PDE) and $s_{\max}$ is the maximal possible stage of any of the ESERK scheme (until now $s_{\max} = 4001$ stages are possible). To compute the first-order SERK approximation at the next time step, one only has to evaluate Equation (4), which means linearly combining all the previous g_j for each discretization point in space to obtain y at the next time step. However, this procedure may cause serious memory demanding problems if we want to use parallel codes or in 3D PDE problems, where N is huge after spatial discretizations.

However, the actual calculation of a next time step only involves a three-term recurrence relation (again see Equation (3)), whenever

$$sum_0 = b_0 g_0, \quad sum_j = sum_{j-1} + b_j g_j, \quad y_{n+1} = sum_s.$$

This means, one only has to store the previous two time steps to compute the next time step and at the same time update Equation (4). Precisely, this means that it suffices to use four arrays of length N, which is a very important progress for the parallel codes in some PDEs.

3.2. Parallelization

The idea of the parallelization for ESERK schemes is very simple, since they are constructed through Richardson extrapolation from first-order SERK methods for different h_i length steps, and each one of these SERK approximations ($S_{i,1}$) can be computed separately. At the same time, we know that the computational cost of calculating $S_{i,1}$ is proportional to the number of function evaluations necessary to calculate it: $i \times s$.

Thus, for traditional non-stiff ODE solvers, whenever the final order of the ODE solver is even, $S_{p,1}$ was calculated in one processor, $S_{p-1,1}$ and $S_{1,1}$ in another one, etc., and finally $S_{p/2,1}$ in the latest one. If the final order of the ODE integrator is odd, $S_{p,1}$ can be calculated in one processor, $S_{p-1,1}$ and $S_{1,1}$ in another one, etc. As we will check in the numerical section, this provides

us an optimum reduction of the computational cost also in our codes, instead of $s \times \frac{p(p+1)}{2}$ function evaluations per step, we would need only $s \times p$ function evaluations, and therefore the theoretical reduction in the computational cost is $\frac{p+1}{2}$.

Thus, parts of the ESERK codes have been parallelized with OpenMP. As it was commented previously, precisely, we have parallized the Richardson extrapolation, since the calculation of each $S_{i,1}$ (i goes from one to the order of the ESERK scheme) is independent. The numerical calculation of one of the $S_{i,1}$ involves the numerical calculation of i time steps using the recurrence relation in formulae (3) and (4).

We have considered several different values for the number of threads for each order $p = 3, 4, 5$ and 6, to numerically check the reduction in computational costs. In the following, we list the work balance between the number of threads using OMP SECTIONS for the different ESERK schemes to compute the $S_{i,1}$.

ESERK3:
1 thread: computes $6s$ functions ($S_{1,1}$, $S_{2,1}$, $S_{3,1}$).
2 threads: first computes 3 time steps of first-order SERK codes ($S_{3,1}$) and second computes other 3 time steps ($S_{1,1}$, $S_{2,1}$). This is well-balanced and no further threads can decrease the computational time for the extrapolation scheme using OMP SECTIONS. Theoretical reduction in the computational cost should be 2 in comparison to sequential ESERK3 code.

ESERK4:
1 thread: computes $10s$ functions ($S_{1,1}$, $S_{2,1}$, $S_{3,1}$, $S_{4,1}$).
2 threads: first computes $5s$ functions ($S_{1,1}$, $S_{4,1}$) and second computes 5 time steps of SERK codes ($S_{2,1}$, $S_{3,1}$). This is well-balanced.
4 threads: first computes 4 time steps ($S_{4,1}$), second computes 3 time steps ($S_{3,1}$), third computes 2 time steps ($S_{2,1}$), fourth computes 1 time step ($S_{1,1}$). This is not well-balanced, but cannot be improved due to the calculation of $S_{4,1}$. CPU times should be at most 2.5 times smaller than in the 1 thread.

ESERK5:
1 thread: computes $15s$ functions ($S_{1,1}$, $S_{2,1}$, $S_{3,1}$, $S_{4,1}$, $S_{5,1}$).
2 threads: first computes 8 time steps ($S_{3,1}$, $S_{5,1}$) and second computes 7 time steps ($S_{1,1}$, $S_{2,1}$, $S_{4,1}$). This is nearly well-balanced.
4 threads: first computes 5 time steps ($S_{5,1}$), second computes 4 time steps ($S_{4,1}$), third computes 3 time steps ($S_{3,1}$), fourth computes 3 time step ($S_{1,1}$, $S_{2,1}$). This is almost well-balanced, but cannot be improved due to the calculation of $S_{5,1}$. Theoretical reduction is 3.

ESERK6:
1 thread: computes $21s$ functions ($S_{1,1}$, $S_{2,1}$, $S_{3,1}$, $S_{4,1}$, $S_{5,1}$, $S_{6,1}$).
2 threads: first computes 11 time steps ($S_{5,1}$, $S_{6,1}$) and second computes 10

time steps ($S_{1,1}$, $S_{2,1}$, $S_{3,1}$, $S_{4,1}$). This is nearly well-balanced.
4 threads: first computes 6 time steps ($S_{6,1}$), second computes 5 time steps ($S_{5,1}$), third computes 5 time steps ($S_{2,1}$, $S_{3,1}$), fourth computes 5 time step ($S_{1,1}$, $S_{4,1}$). This is nearly well-balanced.

All this information is summarized in Table 2. However, we would like to notice that these are theoretical bounds, because some small calculations are done sequentially with the information received from the different processors, and additionally, we are checking that when the dimension of the system is huge numerical speed-up factors decrease.

Once, $S_{i,1}$ are all calculated in parallel, we employ Equation (6) to calculate $S_{6,6}$ in the case of ESERK6, and similarly for ESERK3 to ESERK5 with formulae (for orders $3, 4, 5$) in [18], page 143.

ESERK	2 threads	4 threads
ESERK3	2	—
ESERK4	2	2.5
ESERK5	1.875	3
ESERK6	1.909	3.5

Table 2: Theoretical reduction factors in CPU times employing parallelization versus the sequential codes.

3.3. Deriving variable-step and number of stages algorithm

The step size estimation and stage number selection are very similar to the ones obtained for the ESERK4 algorithm described in [18] or ESERK5 in [17]. First, we select the step size in order to control the local error, and later we choose the minimum number of stages such that the stability properties are satisfied.

1. To select the new step size h_{new} we use those techniques described in [7] for (traditional) p-th-order extrapolated methods:

$$h_{new} = h_{old} \min\left(\text{facmax}, \max\left(\text{facmin}, \text{fac} \cdot (1/err)^{1/p}\right)\right), \tag{8}$$

with fac= 0.8, facmax= 10 (except after a rejection), facmin= 10^{-3}.

During the study of the code ESERK4 [18], we also considered the PI-controller described in [8], with the parameters suggested on pages 27-31, but more steps and function evaluations were necessary, in general.

Comparison between the estimated error and the prescribed tolerance, err, is calculated as usual:

$$err = \sqrt{\frac{1}{n}\sum_{i=1}^{n}\left(\frac{(S_{p,p} - S_{p,p-1})_i}{sc_i}\right)^2}, \tag{9}$$

where

$$sc_i = \left(Atol_i + \max\left(|y_{0,i}|, |S_{p,p,i}|\right) \cdot Rtol_i\right)/2,$$

$y_{0,i}$ is i-th component of the solution at the previous step, and $S_{p,p}$ the solution previously obtained through extrapolation. *Atol* and *Rtol* are prescribed tolerances that depend on the accuracy that every researcher wants to use, *Atol* for the absolute tolerance, and *Rtol* is the relative one. In our test problems we considered $Atol = Rtol$.

Additionally, we try to decrease the number of rejected steps when solving problems in very stiff regions or models with non-smooth data. Hence, we employ techniques also used in [12]:

$$\frac{h_{n+i+1}^{(j)}}{h_{n+i}^{(k)}} \leq 1$$

for the two steps following the rejection ($i = 0, 1$) and

$$\frac{h_{n+i+1}^{(j)}}{h_{n+i}^{(k)}} \leq 2.5$$

for the three steps after that ($i = 2, 3, 4$), unless the interval where there could be jumps has passed. When the risk of rejections has decreased we allow again that

$$\frac{h_{n+i+1}}{h_{n+i}} \leq 10.$$

2. We utilize, as usual in other Chebyshev codes (or stabilized explicit methods), a family of pth-order methods with different numbers of stages, and therefore we choose the minimum number of stages such that the stability properties are satisfied to optimize the codes

$$s > \sqrt{\frac{h_{new}\rho\left(\frac{\partial f}{\partial y}\right)}{2\alpha_p}},$$

where $\rho\left(\frac{\partial f}{\partial y}\right)$ is a bound for the spectral radius (the largest eigenvalue in absolute value of the Jacobian of the function $f(y)$) and $2\alpha_p s^2$ is the estimate of the bound of the stability interval.

For the estimation of the spectral radius several procedures have traditionally been considered. If it is not possible to get an estimate of the spectral radius easily, then a non-linear power method (see [21], for example) is usually considered.

In Figure 1 a summary of the parallel ESERK6 method using four threads is given as a flow chart. After the spectral radius is computed via a non-linear power method as well as the stage and internal stage parameter are initialized, the main loop starts. It ends when $t = T$.

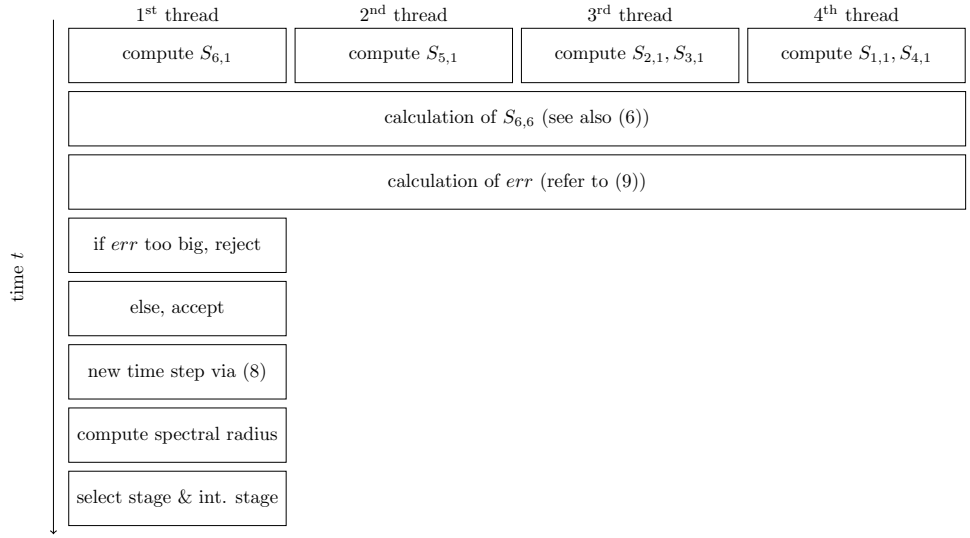


Figure 1: Exemplary flow chart of the main loop (as long as time $t \leq T$) for the parallel ESERK6 scheme using four threads. The spectral radius as well as the stage and internal stage parameter are initialized before the main loop.

4. Numerical results

All numerical results were performed on a regular PC with 32GB of memory and four Intel i7-4790 CPU @ 3.60GHz cores on a socket each of which can have two threads (architecture: x86_64, CPU operation modes: 32-bit and 64-bit, and byte order: little endian). We used the Fortran compiler *gfortran* gcc version 7.4.0 on SUSE Linux (version 15.1) specifying the optimization option `-O3` and the *OpenMP* (version 4.5) option `-fopen_mp`.

We consider two numerical examples. The first example under consideration is the 2D-combustion example that has previously been solved in [17, p. 31]. The second example is the well-known 2D-Brusselator example also considered (as Test 3) in [17, p. 32].

4.1. 2D-Combustion problem

In this section, we consider the 2D-combusion model on a unit square. The non-linear problem from combustion theory (see also [9, 23]) is given by

$$u_t = 2.5\Delta u + \frac{1}{4}(2-u)\mathrm{e}^{20(1-1/u)}\,. \tag{10}$$

The initial condition is given by constant one. We specify homogeneous Neumann boundary conditions on the south and west edge and constant Dirichlet boundary conditions with value one on the north and east edge of the square,

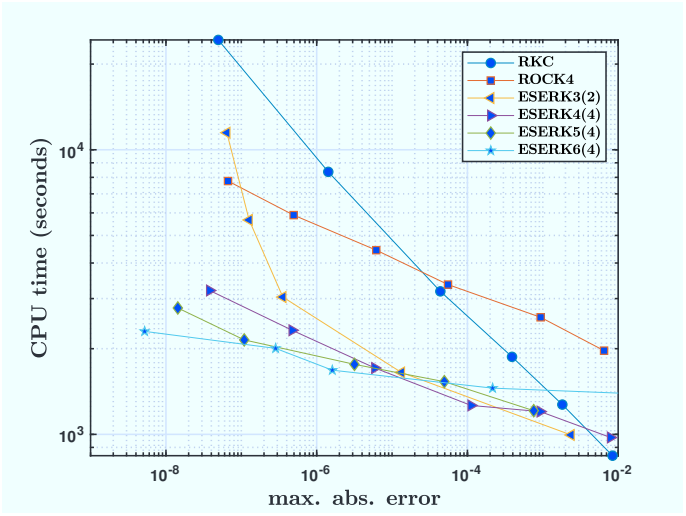


Figure 2: Maximal absolute errors versus CPU times in seconds for the second example using RKC, ROCK4, ESERK3, ESERK4, ESERK5, and ESERK6.

respectively. In total, we used $N = 600$ equidistant nodes in each spatial direction for the second-order discretization in space. The Neumann boundary condition is approximated by a second order approximation as well. The final time is 1.48.

In Fig. 2, we compare errors versus CPU times employed by the parallelized versions of ESERK3, ESERK4, ESERK5, and ESERK6, and the well-known sequential codes RKC [21] and ROCK4 [1]. Obviously, since these well-known codes are sequential, we need to be careful with the comparison among all the codes.

We check that for larger tolerances, lower-order codes are very efficient. When errors are between $O(10^{-2})$ and $O(10^{-4})$, RKC and ESERK3 are very fast, however for $O(10^{-6}) - O(10^{-8})$, ESERK5 and ESERK6 become safer and faster compared to lower-order algorithms. Therefore, the best order of convergence will depend on the sought errors, but this is not the only issue to observe when we are choosing the optimum order of convergence for our schemes.

4.1.1. Numerical study of parallelization

Let us now study the improvement provided by the parallelization of the codes in comparison to the sequential schemes.

The CPU times for the different ESERK methods using either 1, 2, or 4 threads with the parameter $N = 600$ are shown in Table 3, these CPU times are for the whole process. Since, errors (with the same tolerance, but comparing sequential and parallelized versions) are basically the same (normally, at least the first 3 or 4 digits), we have only compared the CPU times required among using 1, 2, and 4 threads, and calculated the improvement between using 4

threads (in ESERK4, ESERK5 and ESERK6) and 1 thread, or between using 2 threads and 1 thread in ESERK3, this is what we called speed-up factor in the table. We can notice that numerical speed-up factor are smaller than theoretical reductions in the computational cost described in the previous section. Also, the ratios observed in this table are lower when tolerances are smaller. Obviously, theoretical factors are always bounds, but communications among processors are necessary, and some small calculations are done sequentially before doing the loops, therefore numerical factors are clearly smaller, especially for the highest-order codes. Also, the memory access between the different threads lowers the theoretical bound. This can be seen when $N = 600$ is large compared to $N = 300$.

ESERK	Tolerance	1 thread	2 threads	4 threads	Speed-up factor
3	10^{-6}	2865	1650	—	1.74
3	10^{-8}	9671	5667	—	1.71
3	10^{-10}	37947	25236	—	1.50
4	10^{-6}	2518	1396	1203	2.09
4	10^{-8}	3593	1992	1711	2.10
4	10^{-10}	6584	3593	3203	2.06
5	10^{-6}	2928	1693	1211	2.42
5	10^{-8}	4262	2440	1762	2.42
5	10^{-10}	6687	3799	2780	2.41
6	10^{-6}	3873	2189	1454	2.66
6	10^{-8}	5341	3074	2005	2.66
6	10^{-10}	7741	4409	2892	2.68

Table 3: CPU times in seconds (in the whole process) for ESERK 3–6 for 1, 2, and 4 threads using the tolerances 10^{-6}, 10^{-8}, and 10^{-10} with $N = 600$. In this case the dimension of the system of ODEs is approximately 3.6×10^5.

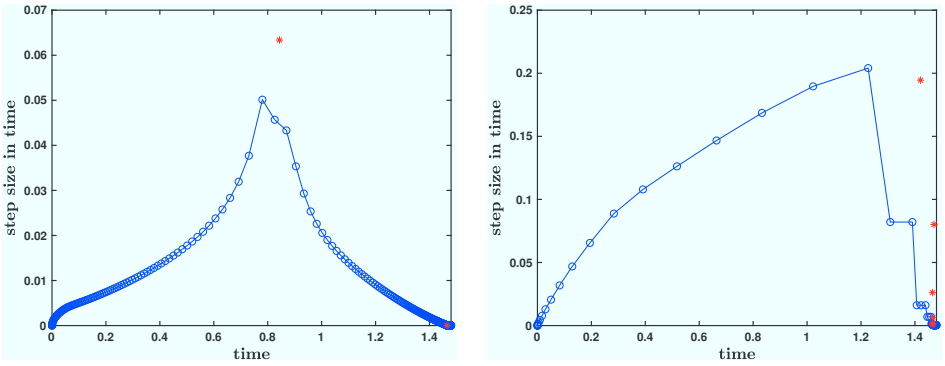
Readers might wonder what happens if we focus only on the parallel part, if these ratios are closer to theoretical bounds given in Table 2, and they are. Between a 2% (with larger tolerances, and smaller dimension cases) and a 20% of the total CPU time (in the sequential code) is required to tasks that they are done sequentially in all the cases (also with several threads): estimation of the error, calculation of the next step length, and time of preparation before the first iteration. This percentage is small for large tolerances, and it grows when tolerances are smaller, because the average number of stages is smaller for lower tolerances. This explains that ratios in the previous Table 3 usually decrease for small tolerances.

CPU times for the different ESERK methods, with $N = 300$ are shown in Table 4. When $N < 300$ ratios (speed-up factors) get closer to theoretical bounds. And, when tolerances decrease, the differences between theoretical and numerical rations are, now, not so large as with the previous Table 3.

Step sizes for problem 1 obtained through Eq. (8) in the paper are given in Fig. 3, with $tol = 10^{-6}$, 4 threads, and $N = 300$, for ESERK3 and ESERK6, respectively. We can check that most of the rejections are in the interval where

ESERK	Tolerance	1 thread	2 threads	4 threads	Speed-up factor
3	10^{-6}	347	184	—	1.97
3	10^{-8}	1041	537	—	1.94
3	10^{-10}	4156	2187	—	1.90
4	10^{-6}	287	145	122	2.34
4	10^{-8}	406	207	173	2.34
4	10^{-10}	751	381	323	2.33
5	10^{-6}	338	181	127	2.64
5	10^{-8}	491	261	182	2.69
5	10^{-10}	800	433	306	2.61
6	10^{-6}	447	235	148	3.03
6	10^{-8}	620	326	205	3.02
6	10^{-10}	908	482	300	3.03

Table 4: CPU times, only in the parallel part, in seconds for ESERK 3–6 for 1, 2, and 4 threads using the tolerances 10^{-6}, 10^{-8}, and 10^{-10} with $N = 300$.



ESERK3 *ESERK6*

Figure 3: Step sizes for problem 1, with $tol = 10^{-6}$, and $N = 300$, for ESERK3 and ESERK6.

the reaction term is very stiff (combustion part). The value of the accepted and rejected step size at time t is marked by a blue circle and a red asterisk, respectively.

However, we also repeated the calculations but, again with $N = 600$. Now, ratios are not as poor as in Table 3, but speed-up factors are lower (they are shown in Table 5). As we can see, the dimension of the system of ODEs is clearly much higher now (approximately four times larger than before).

4.1.2. Developing one code that allows the change of order of convergence

Extrapolation methods have the advantage that in addition to the length step, also the order of convergence can be changed at each step. Obviously, this will make the computation of the whole code (where we will include all the ESERK methods with different order) much more complicated and challenging

ESERK	Tolerance	1 thread	2 threads	4 threads	Speed-up factor
3	10^{-6}	2797	1584	—	1.77
3	10^{-8}	8949	4955	—	1.81
3	10^{-10}	30086	17492	—	1.72
4	10^{-6}	2505	1384	1190	2.11
4	10^{-8}	3560	1959	1678	2.12
4	10^{-10}	6479	3489	3100	2.09
5	10^{-6}	2920	1685	1203	2.43
5	10^{-8}	4243	2421	1743	2.43
5	10^{-10}	6640	3751	2733	2.43
6	10^{-6}	3866	2183	1447	2.67
6	10^{-8}	5327	3060	1991	2.68
6	10^{-10}	7710	4378	2861	2.70

Table 5: CPU times, only in the parallel part, in seconds for ESERK 3–6 for 1, 2 and 4 threads using the tolerances 10^{-6}, 10^{-8}, and 10^{-10} with $N = 600$.

than for any other fixed-order Runge–Kutta method. However, it is well-known that higher-order methods solve more efficiently and accurately ODEs where the solution is very stiff than lower-order methods.

As an example, we can do the following test: first we will divide the interval of integration in two,

- we solve the combustion problem in $[0, 1.46]$ with the parallelized versions of ESERK3 (2 threads) and ESERK6 (4 threads),
- we solve the combustion problem in $[1.46, 1.48]$ with ESERK3 and ESERK6, and the same tolerances as in the previous interval.

It is explained in [17] (see Fig. 3) that the solution of this problem varies very slowly in the interval $[0, 1.45]$, and therefore larger length steps are employed typically and lower-order methods are efficient for moderate tolerances. In $[1.45, 1.46]$ the solution changes more rapidly, and, from 1.46 to 1.48, approximately, the solution changes very fast and higher-order methods are more efficient in this part (it is also explained in the book by Hundsdorfer and Verwer [9]).

In Table 6, we show the results with both parallelized versions of ESERK3 (2 threads) and ESERK6 (4 threads). In $[0, 1.46]$, ESERK3 is faster for larger tolerances, ESERK6 has more rejections for $tol \geq 10^{-6}$. With errors $\sim O(10^{-6})$, ESERK3 is able to obtain faster an accurate solution. This fact changes clearly in the interval $[1.46, 1.48]$, for larger tolerances ESERK6 has still some difficulties and more rejections than ESERK3. However, when errors are between $\sim O(10^{-4})$ and $O(10^{-5})$, ESERK6 is now able to obtain faster an accurate solution than ESERK3. Hence, the stiffness of the problem influences in choosing the optimum order of convergence.

Successful extrapolation codes which allow changing the order, have been previously described for non-stiff equations (see [7, Ch. II.9]). But results

Interval	Tolerance	Method	max. err.	Time (s)	NFE
$[0, 1.46]$	10^{-6}	ESERK3	0.1561_{-5}	1314.56	286598
		ESERK6	0.1261_{0}	2414.85	375779
$[0, 1.46]$	10^{-7}	ESERK3	0.3517_{-7}	2414.85	517508
		ESERK6	0.3345_{-6}	1465.92	469425
$[0, 1.46]$	10^{-8}	ESERK3	0.6993_{-8}	4286.51	897502
		ESERK6	0.3682_{-7}	1687.77	539231
$[1.46, 1.48]$	10^{-6}	ESERK3	0.1330_{-3}	335.65	59311
		ESERK6	0.2158_{-3}	257.01	100013
$[1.46, 1.48]$	10^{-7}	ESERK3	0.3556_{-6}	622.5	107312
		ESERK6	0.1612_{-5}	215.05	100882
$[1.46, 1.48]$	10^{-8}	ESERK3	0.1269_{-6}	1380.	213967
		ESERK6	0.2852_{-6}	317.43	115707

Table 6: Maximal absolute error, CPU times and number of function evaluations for the methods ESERK3, and ESERK6 for $[0, 1.46]$ (up), and $[1.46, 1.48]$ (bottom).

shown in Table 6 suggest that, precisely in stiff problems, the combination in one code, of Runge–Kutta methods with different order of convergence is a very interesting tool to solve faster and more accurately these types of models, because stiff equations normally possesses regions where the solution is smooth (and therefore lower-order schemes produce accurate solutions fast), and regions where the solution vary very fast and therefore smaller length steps and higher-order algorithms are necessary.

The process of choosing step length and order of convergence in extrapolation codes for non-stiff equations is known. In general, it follows several stages (see [7, p. 233–237] for the DIFEX1 code based on the GBS-algorithm):

1. The choice of step size, if we are not changing the order of the method is done in a similar way as it was described in Section 3.3.

   In our case, for ESERK methods, at this stage, we also need to obtain the minimum number of stages to guarantee stability.

2. An "optimal order" is obtained. For this task, first an amount called work per unit step, W_k is defined, and a procedure divided in 5 stages is derived (stages (a)–(e) in pages 234 and 235 in [7]). However, our ESERK codes require different first-order SERK approximations, with different stability regions, for their construction. Additionally with parallelization the improvement given by stages (a), (b), and (e) is minimal. Hence, we can basically reduce this procedure to stage (c) for ESERK methods.

   (a) *Convergence in line $k-1$:* We first compute the $k-1$ lines of the extrapolation, and calculate err_{k-1}, W_{k-1} (W_k is defined as in Equation (14)). If $err_{k-1} \leq 1$, then $T_{k-1,k-1}$ (the extrapolation approximation similar to our $S_{k-1,k-1}$) is accepted as numerical solution, i.e. the last line of the extrapolation was avoided.

And k_{new}, the following order of convergence, is chosen as k (the previous one) if $W_{k-1} < 0.9W_{k-2}$ or $k-1$ otherwise.

(b) *Convergence monitor:* In this stage, the authors studied if the last step should be rejected. Therefore if $err_{k-1} > \left(\frac{n_{k+1}n_k}{n_1^2}\right)^2$ (n_i being the positive integers described in 2.2), then there was a rejection, and they restart with $k_{new} \leq k-1$ and calculate the new h_{new}. If this is not correct, they can compute the next line of the extrapolation method.

(c) *Convergence in line k:* We will calculate the extrapolation approximation $S_{k,k}$, err_k (as it was explained previously in this paper), W_k (for ESERK methods, W_k is calculated through Equation (14)), and also $S_{k-1,k-1}$, err_{k-1} and W_{k-1} . If $err_k \leq 1$, then $S_{k,k}$ is accepted and we continue the integration with the following values for the next step:

$$k_{new} = \begin{cases} k-1 & \text{if } W_{k-1} < 0.9W_k \\ k+1 & \text{if } W_k < 0.9W_{k-1} \\ k & \text{in all other cases,} \end{cases} \tag{11}$$

$$h_{new} = \begin{cases} h_{k_{new}} & \text{if } k_{new} \leq k \\ h_k \frac{A_{k+1}}{A_k} & \text{if } k_{new} = k+1, \end{cases} \tag{12}$$

where A_k is given by Equation (16) for the parallel codes and (15) for the sequential ones.

(d) *Second convergence monitor:* If $err_k > \left(\frac{n_{k+1}}{n_1}\right)^2$ then the authors in [7] proposed a rejection, and restart with $k_{new} \leq k$ (and calculate the new h_{new}). Otherwise, the code continues with the following step.

(e) *Hope for convergence in line k + 1:* In this stage, $T_{k+1,k+1}$, err_{k+1} and W_{k+1} were calculated (see [7]). If $err_{k+1} \leq 1$, then $T_{k+1,k+1}$ is accepted, and the code continues the integration with the new order:

$$k_{new} = \begin{cases} k-1 & \text{if } W_{k-1} < 0.9W_k \\ k+1 & \text{if } W_{k+1} < 0.9W_k \\ k & \text{in all other cases,} \end{cases} \tag{13}$$

If $err_{k+1} > 1$, then there is a rejection, and the authors in [7] propose to restart with $k_{new} \leq k$ and calculate the new h_{new} through Equation (8).

In our case, we should skip stages (d) and (e) of the procedure since our ESERK codes require different first-order SERK approximations, with different stability regions, if we take a higher-order, instabilities may appear.

Again, after a rejection, we avoid to increase the length step (and now) the order of convergence during several steps.

However, there are several important differences between ESERK methods and the GBS-extrapolation algorithm, and we should take care with them before developing our code that allows to change the order of convergence:

- In GBS-extrapolation algorithm, $T_{k,k}$ defined by Equation (9.10) in [7] (the extrapolation approximation) has order $2k$. In ESERK methods, the corresponding $S_{k,k}$ has order k. Additionally, as it was commented before, we should be very careful with instabilities if we try to employ a higher-order method $(k+1)$, with a $S_{1,1}$ derived for a lower order scheme (k).

- GBS-extrapolation algorithm is sequential, and therefore stages (a), (b), (d), and (e) previously mentioned are useful to reduce the computational cost. In our case, we have developed two types of codes, one sequential and another in parallel. With the sequential codes, we can proceed in a similar way as in [7]. But, in the case of the parallel algorithm, we will directly go to the third stage (c).

  Additionally, the way to calculate W_k (work per unit step) will be different.

- The work for computing $S_{k,k}$ (in the ESERK methods) can be computed with W_k, the so-called work per unit step, which can be defined again as

$$W_k = \frac{A_k}{h_k} \tag{14}$$

  where h_k will be the length step (calculated with Equation (8)) if we employ the k–th-order method, and A_k will be an estimation of the number of function evaluations required to calculate $S_{k,k}$ (as it was the case with $T_{k,k}$).

  Obviously, the way we calculate A_k for ESERK methods is quite different to the way it was calculated for DIFEX1 and other extrapolation codes for non-stiff ODEs. In our case, we need to take into account that the first-order SERK approximation requires s calls of the function.

  Therefore, for the sequential ESERK scheme of $k-$th order A_k can be calculated recursively through

$$\begin{aligned} A_1 &= s \\ A_k &= A_{k-1} + ks, \end{aligned} \tag{15}$$

  i.e., $A_k = \frac{sk(k+1)}{2}$.

  As for the parallel ESERK codes (in practice they are a better option), our idea is studying the general code using stage (c) previously described, and we need to take care with the numerical reduction factor obtained by parallelization in the calculus of A_k. Speed-up factors in Table 3, suggest that A_k can be calculated as

$$A_k = \frac{sk(k+1)}{2\,\mathrm{suf}_k}, \tag{16}$$

  with $\mathrm{suf}_3 = 1.8$, $\mathrm{suf}_4 = 2.1$, $\mathrm{suf}_5 = 2.4$ and $\mathrm{suf}_6 = 2.7$.

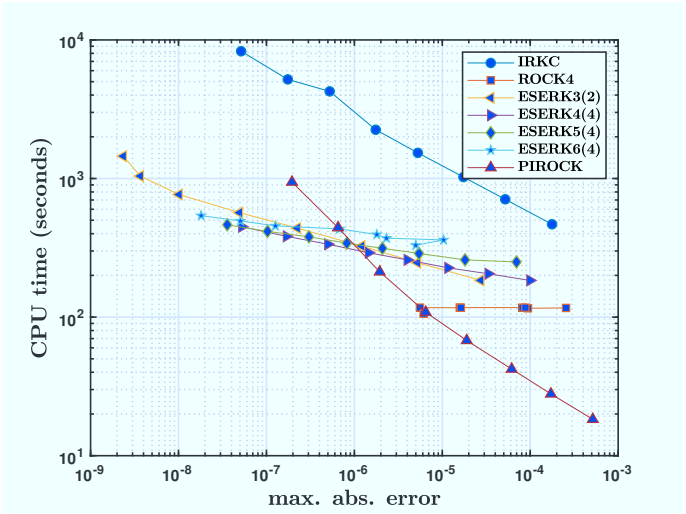


Figure 4: Maximal absolute error versus CPU times in seconds for the second example using ROCK4, IRKC, PIROCK, ESERK3, ESERK4, ESERK5 and ESERK6.

4.2. 2D-Brusselator example

The second example considered in this paper is a two-dimensional Brusselator reaction-diffusion problem

$$\begin{aligned} \frac{\partial u}{\partial t} &= A + u^2 v - (B+1)u + \alpha \left(\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2} \right), \\ \frac{\partial v}{\partial t} &= Bu - u^2 v + \alpha \left(\frac{\partial^2 v}{\partial x^2} + \frac{\partial^2 v}{\partial y^2} \right). \end{aligned} \tag{17}$$

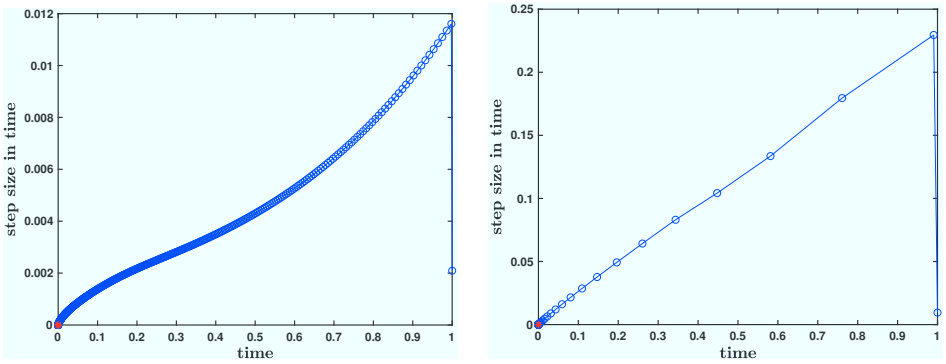
We solve this problem for $0 \le x \le 1$, $0 \le y \le 1$, $0 \le t \le t_{end} = 1$, using $A = 1.3$, $B = 2 \times 10^6$, and $\alpha = 0.1$, with periodic boundary conditions $u(x+1, y, t) = u(x, y, t) = u(x, y+1, t)$ and the initial condition chosen as in [3]:

$$u(x, y, 0) = 22y(1-y)^{3/2}, \quad v(x, y, 0) = 27x(1-x)^{3/2}.$$

We discretized u, v in space with two $N \times N$ uniform meshes, where $N = 500$. Thus, $\rho \sim 2.4 \times 10^6$ and $\rho_D \sim 8\alpha N^2 = 2 \times 10^5$ (the spectral radius of the diffusion term).

In Fig. 4, we compare errors versus CPU times employed by the parallelized versions of ESERK3, ESERK4, ESERK5, and ESERK6, and the well-known sequential codes ROCK4, IRKC, and PIROCK (developed for diffusion-reaction problems like this one).

As we can check, ESERK codes are very efficient compared to IRKC and PIROCK for small tolerances, although those algorithms are specifically constructed for reaction-diffusion problems.



ESERK3 *ESERK6*

Figure 5: Step sizes for problem 2 with ESERK3, and ESERK6.

ESERK5 and ESERK6 are faster than ESERK3 and ESERK4 only when $errors \simeq O(10^{-6}) - O(10^{-7})$ in this case, this is because the solution of this problem is less stiff than in the previous one, and stiffness of the problem is close to the origin.

For this t_{end}, ROCK4 obtains a solution very fast in this numerical example, but errors are clearly higher than prescribed tolerances. For $tol = 10^{-7}, 3\times10^{-8}$, $errors > 5 \times 10^{-6}$ (more than 100 times larger). This might happen because ROCK4 has some problems with propagation of errors when the maximum number of stages is utilized [9].

Additionally, ROCK4 is utilizing the maximum step allowed in time for many of the tolerances, because the stiffness of the problem is very close to the origin, and later solution is smoother. For longer t_{end} values, ROCK4 would require the maximum step during many steps, since the solution gets smoother. The advantage of ESERK codes is their maximum length step in time is more than 40 times bigger than for ROCK4. Thus, when t_{end} is bigger, they are able to obtain accurate solutions faster.

Step sizes for problem 2 obtained through Eq. (8) are given in Fig. 5, with $tol = 10^{-6}$, for ESERK3 and ESERK6 (the new families), respectively. We can check that step lengths are very stable, and there is a small number of rejections.

5. Conclusions

Two new whole families of ESERK schemes are derived, and analyzed, in this work, with more than 100 new schemes, and up to several thousand of stages in each family. For the first time, we also parallelized these codes: the four families with orders from 3 to 6. These parallelized strategies allow to decrease times significantly as it is explained theoretical and numerically, and therefore, the new codes provide very good results in relation to other well-known ODE solvers. These four parallel algorithms are now freely available in:

https://github.com/kleefeld80/ESERK3-6parallel.git and therefore anybody can use them. The four algorithms are tested for several problems and compared with other well-known codes obtaining very good results.

Finally, for the first time with these ESERK methods, it is discussed the idea of combining all these families in one code. We numerically showed that the optimum order of convergence usually depends on the prescribed tolerance (the sought errors), but also the stiffness of the problem. Hence, the combination of schemes with different order of convergence is specially interesting in such kind of problems that ESERK algorithms solve. We also theoretically stated a procedure to combine all these methods in one. We would like to continue with this new line in the future, and check numerically the behavior of this idea.

Acknowledgements

The authors would like to thank Mamen Borrego and Luisa M. López for their help to obtain the coefficients of the codes. The authors acknowledges support from the University of Salamanca through its own "Programa Propio I, Modalidad C2" grant 18.KB2B.

References

[1] A. Abdulle. Fourth order Chebyshev methods with recurrence relation. *SIAM J. Sci. Comput.*, 23(6):2041–2054, 2001.

[2] A. Abdulle and A. A. Medovikov. Second order Chebyshev methods based on orthogonal polynomials. *Numerische Mathematik*, 90(1):1–18, 2001.

[3] A. Abdulle and G. Vilmart. PIROCK: a swiss-knife partitioned implicit–explicit orthogonal Runge–Kutta–Chebyshev integrator for stiff diffusion–advection–reaction problems with or without noise. *Journal of Computational Physics*, 242:869–888, 2013.

[4] R. C. Aiken. *Stiff Computation.* Oxford University Press, Inc., New York, NY, USA, 1985.

[5] P. Bocher, J. I. Montijano, L. Rández, and M. Daele. Explicit Runge—Kutta methods for stiff problems with a gap in their eigenvalue spectrum. *J. Sci. Comput.*, 77(2):1055–1083, Nov. 2018.

[6] R. D'Ambrosio, E. Hairer, and C. Zbinden. G-symplecticity implies conjugate-symplecticity of the underlying one-step method. *BIT Numerical Mathematics*, 53:867–872, 2013.

[7] E. Hairer, S. P. Nørsett, and G. Wanner. *Solving Ordinary Differential Equations I (2nd Revised. Ed.): Nonstiff Problems.* Springer-Verlag New York, Inc., NY, USA, 1993.

[8] E. Hairer and G. Wanner. *Solving ordinary differential equations. II: Stiff and differential-algebraic problems.* Springer, Berlin, 1996.

[9] W. H. Hundsdorfer and J. G. Verwer. *Numerical solution of time-dependent advection-diffusion-reaction equations*. Springer, Berlin, Heidelberg, 2007.

[10] D. I. Ketcheson and U. bin Waheed. A comparison of high order explicit Runge-Kutta, extrapolation, and deferred correction methods in serial and parallel. *CAMCoS*, 9(2):175–200, 2014.

[11] B. Kleefeld and J. Martín-Vaquero. SERK2v2: A new second-order stabilized explicit Runge-Kutta method for stiff problems. *Numerical Methods for Partial Differential Equations*, 29(1):170–185, 2013.

[12] B. Kleefeld and J. Martín-Vaquero. SERK2v3: Solving mildly stiff nonlinear partial differential equations. *Journal of Computationald & Applied Mathematics*, 299:194–206, 2016.

[13] V. I. Lebedev. A new method for determining the roots of polynomials of least deviation on a segment with weight and subject to additional conditions. part I. *Russian Journal of Numerical Analysis and Mathematical Modelling*, 8(3):195–222, 1993.

[14] V. I. Lebedev and S. A. Finogenov. Solution of the parameter ordering problem in Chebyshev iterative methods. *USSR Computational Mathematics and Mathematical Physics*, 13(1):21–41, 1974.

[15] J. Martín-Vaquero and B. Janssen. Second-order stabilized explicit Runge-Kutta methods for stiff problems. *Computer Physics Communications*, 180(10):1802–1810, 2009.

[16] J. Martín-Vaquero, A. Q. M. Khaliq, and B. Kleefeld. Stabilized explicit Runge-Kutta methods for multi-asset American options. *Computers & Mathematics with Applications*, 67(6):1293–1308, 2014.

[17] J. Martín-Vaquero and A. Kleefeld. ESERK5: A fifth-order extrapolated stabilized explicit Runge-–Kutta method. *Journal of Computational and Applied Mathematics*, 356:22–36, 2019.

[18] J. Martín-Vaquero and B. Kleefeld. Extrapolated stabilized explicit Runge–Kutta methods. *Journal of Computational Physics*, 326:141–155, 2016.

[19] A. A. Medovikov. High order explicit methods for parabolic equations. *BIT Numerical Mathematics*, 38(2):372–390, 1998.

[20] V. Saul'yev, G. Tee, and E. Stewart, K.L. Integration of equations of parabolic type by the method of nets.

[21] B. Sommeijer, L. Shampine, and J. Verwer. RKC: An explicit solver for parabolic PDEs. *Journal of Computational and Applied Mathematics*, 88(2):315–326, 1997.

[22] M. Torrilhon and R. Jeltsch. Essentially optimal explicit Runge-Kutta methods with application to hyperbolic-parabolic equations. *Numerische Mathematik*, 106(2):303–334, 2007.

[23] J. G. Verwer. Explicit Runge-Kutta methods for parabolic partial differential equations. *Appl. Numer. Math.*, 22(1–3):359–379, 1996.