Journal of Computational Physics 498 (2024) 112640

Contents lists available at ScienceDirect

Journal of Computational Physics

journal homepage: www.elsevier.com/locate/jcp

A second order directional split exponential integrator for systems of advection–diffusion–reaction equations

Marco Caliari *, Fabio Cassini

Department of Computer Science, University of Verona, Italy

ARTICLE INFO

Keywords:
Exponential integrators
Advection–diffusion–reaction systems
μ-mode product
Directional splitting
Turing patterns

ABSTRACT

We propose a second order exponential scheme suitable for two-component coupled systems of stiff evolutionary advection–diffusion–reaction equations in two and three space dimensions. It is based on a directional splitting of the involved matrix functions, which allows for a simple yet efficient implementation through the computation of small sized exponential-like functions and tensor-matrix products. The procedure straightforwardly extends to the case of an arbitrary number of components and to any space dimension. Several numerical examples in 2D and 3D with physically relevant (advective) Schnakenberg, FitzHugh–Nagumo, DIB, and advective Brusselator models clearly show the advantage of the approach against state-of-the-art techniques.

1. Introduction

Efficiently solving systems of Advection–Diffusion–Reaction (ADR) equations is of paramount importance in practical applications. In fact, these kinds of models effectively capture many physical, chemical, and biological phenomena, such as electrodeposition [9], biochemical reactions [28,39], electric current flows [18], tumor growth [36], and epidemic dynamics [29], among the others. From a numerical point of view, the computation of approximated solutions gives rise to many challenges. For example, in this context there is often the intrinsic need of a fine spatial grid in order to correctly capture the dynamics of the system (the formation of the so-called Turing patterns [17], for instance). This, in turn, typically translates into working in a *stiff* regime [20], which requires a careful choice of the underlying numerical schemes.

In particular, in this work we focus on the numerical integration of *two-component* systems of ADR equations in the form

$$\begin{cases} \partial_t u(t,\boldsymbol{x}) = \mathcal{K}^u u(t,\boldsymbol{x}) + g^u(u(t,\boldsymbol{x}), v(t,\boldsymbol{x})), \\ \partial_t v(t,\boldsymbol{x}) = \mathcal{K}^v v(t,\boldsymbol{x}) + g^v(u(t,\boldsymbol{x}), v(t,\boldsymbol{x})), \end{cases} \tag{1}$$

although the method presented later on in the manuscript easily extends to an arbitrary number of components. Here $u, v : [0,T] \times \Omega \subset \mathbb{R} \times \mathbb{R}^d \to \mathbb{R}$ represent the unknowns, $\mathcal{K}^u, \mathcal{K}^v$ are linear advection–diffusion operators, while g^u, g^v are the nonlinear reaction terms. The choice of the latter basically determines the model and phenomenon under investigation. We assume that the spatial domain Ω is the Cartesian product of one-dimensional intervals, that is $\Omega = [a_1, b_1] \times \cdots \times [a_d, b_d]$, as it is commonly considered

* Corresponding author.
E-mail addresses: marco.caliari@univr.it (M. Caliari), fabio.cassini@univr.it (F. Cassini).

https://doi.org/10.1016/j.jcp.2023.112640
Received 19 July 2023; Received in revised form 13 November 2023; Accepted 13 November 2023

Available online 20 November 2023

in ADR examples from the literature (see, for instance, References [2,5,7,17,18]). The system is finally completed with appropriate initial conditions and with homogeneous Neumann boundary conditions, which appear to be widely used by the scientific community. We introduce a spatial grid of size $n_1 \times \cdots \times n_d$ and apply the Method Of Lines (MOL) to model (1) to get a system with *Kronecker sum* structure. In fact, we suppose to obtain a system of stiff ordinary differential equations in the form

$$\begin{cases} \boldsymbol{u}'(t) = K^{\boldsymbol{u}}\boldsymbol{u}(t) + \boldsymbol{g}^{\boldsymbol{u}}(\boldsymbol{u}(t), \boldsymbol{v}(t)), \\ \boldsymbol{v}'(t) = K^{\boldsymbol{v}}\boldsymbol{v}(t) + \boldsymbol{g}^{\boldsymbol{v}}(\boldsymbol{u}(t), \boldsymbol{v}(t)), \end{cases} \tag{2}$$

where $K^{\boldsymbol{u}}$ and $K^{\boldsymbol{v}}$ are matrices of size $N \times N$, with $N = n_1 \cdots n_d$, that discretize the linear operators $\mathcal{K}^u$ and $\mathcal{K}^v$, respectively. The matrices $K^{\boldsymbol{u}}$ and $K^{\boldsymbol{v}}$ are Kronecker sums. By definition, a matrix $K \in \mathbb{C}^{N\times N}$ is a Kronecker sum if it can be decomposed as

$$K = A_d \oplus A_{d-1} \oplus \cdots \oplus A_1 = \sum_{\mu=1}^{d} A_{\otimes\mu}, \tag{3a}$$

where

$$A_{\otimes\mu} = I_d \otimes \cdots \otimes I_{\mu+1} \otimes A_\mu \otimes I_{\mu-1} \otimes \cdots \otimes I_1. \tag{3b}$$

In our context I_μ and A_μ, for $\mu = 1, \ldots, d$, are matrices of *small* size $n_\mu \times n_\mu$ and represent the identity and a one-dimensional linear differential operator along the direction μ, respectively. The symbol $\otimes$ denotes the standard Kronecker product between matrices. The Kronecker sum structure typically arises when applying finite differences discretizations [10,11] to the differential operators involved in the equations, but for instance tensor product finite elements [16,31] fit into this class as well. Notice that, for the method we are going to propose, we do not assume the matrices A_μ having a specific sparsity pattern.

Several methods can be employed for the time integration of system (2). In presence of stiffness, an attractive choice are integrators of exponential or implicit type, that are useful to alleviate the stability restrictions on the time step size that come with explicit methods. Of course, ad-hoc techniques have to be adopted in order to retain efficiency of computation. For instance, among schemes of exponential type [22], Lawson integrators [25] (also known as Integration Factor methods [24]) are appealing, since they require just the action of the matrix exponential function and thus they can directly exploit the Kronecker sum structure of the system (see Reference [10], and Reference [11] for some insights on GPU scalability). On the other hand, it is well-known that they may suffer from a bad convergence behavior, in particular when the boundary conditions are non-periodic (see Reference [21]). Different exponential integrators rely on general purpose Krylov methods for the approximation of the matrix exponential or related functions with a fixed size of the Krylov space (see Reference [7]) or with incomplete orthogonalization (see References [19,27]). More recent techniques that exploit the underlying Kronecker sum structure are presented, for instance, in References [4,13,16,17,31]. Concerning implicit methods, in the two-dimensional case low-rank or other Model Order Reduction techniques combined with a matrix formulation of the problem can be applied to classical schemes such as IMplicit EXplicit methods (see References [2,17]). We finally mention also the possibility to employ explicit schemes such as a Total Variation Diminishing explicit Runge–Kutta scheme (see Reference [37]), provided that a sufficiently small time step size is taken.

In this manuscript, we propose an exponential scheme of second order for system (2). It is a *directional split* version of the well-known exponential Runge–Kutta method of order two ETD2RK, and it approximates the required four actions of matrix φ_ℓ functions by means of four tensor-matrix products (using the so-called Tucker operator). This approach allows for a simple yet very efficient implementation, thanks to the high performance level 3 BLAS available for basically any kind of modern computer system. A preliminary study of the effectiveness of the directional splitting of the matrix φ_ℓ functions in exponential integrators has been done in Reference [10], but there the numerical examples were limited to recover the steady state of an algebraic Riccati equation and to measure the performance on a prototypical evolutionary equation (whose explicit analytical solution was known). Here, the focus is to show that the proposed exponential scheme is highly competitive with state-of-the-art methods in the solution of important ADR models, and in particular for the challenging task of recovering the arising patterns.

The remaining part of paper is organized as follows. In Section 2 we will briefly recall integrators of exponential type and the tensor framework needed to implement them efficiently. In Section 3 we will present ETD2RKds, which is the proposed integrator with directional splitting. Although the method is presented in arbitrary dimension d, we proceed in Section 4 by testing it on a wide variety of two-dimensional and three-dimensional numerical examples with popular models in the literature, i.e., the (advective) Schnakenberg, the FitzHugh–Nagumo, the DIB, and the advective Brusselator models. We will see that the proposed approach is able to effectively and efficiently integrate numerically the problems, both in terms of achieved accuracy and wall-clock time, in comparison with other popular techniques. Moreover, we will also show that we can attain the expected patterns of the underlying models in a short amount of time. Finally, we will draw the conclusions in Section 5.

2. Exponential integrators for systems of partial differential equations

The starting point for the exponential integrators under consideration is the variation-of-constants formula applied to system (2) with time step size $\tau = t_{n+1} - t_n$, i.e.,

$$\begin{aligned}\boldsymbol{u}(t_{n+1}) &= \mathrm{e}^{\tau K^{\boldsymbol{u}}}\boldsymbol{u}(t_n) + \tau\int_0^1 \mathrm{e}^{(1-\theta)\tau K^{\boldsymbol{u}}}\boldsymbol{g}^{\boldsymbol{u}}(\boldsymbol{u}(t_n+\theta\tau),\boldsymbol{v}(t_n+\theta\tau))d\theta,\\ \boldsymbol{v}(t_{n+1}) &= \mathrm{e}^{\tau K^{\boldsymbol{v}}}\boldsymbol{v}(t_n) + \tau\int_0^1 \mathrm{e}^{(1-\theta)\tau K^{\boldsymbol{v}}}\boldsymbol{g}^{\boldsymbol{v}}(\boldsymbol{u}(t_n+\theta\tau),\boldsymbol{v}(t_n+\theta\tau))d\theta.\end{aligned} \tag{4}$$

We then apply the trapezoidal quadrature rule, by introducing an explicit intermediate stage to approximate the evaluation at t_{n+1} (see also Reference [10]). Putting everything together, we get the second order Lawson scheme

$$\begin{aligned}\boldsymbol{u}_{n2} &= \mathrm{e}^{\tau K^{\boldsymbol{u}}}(\boldsymbol{u}_n + \tau\boldsymbol{g}^{\boldsymbol{u}}(\boldsymbol{u}_n,\boldsymbol{v}_n)),\\ \boldsymbol{v}_{n2} &= \mathrm{e}^{\tau K^{\boldsymbol{v}}}(\boldsymbol{v}_n + \tau\boldsymbol{g}^{\boldsymbol{v}}(\boldsymbol{u}_n,\boldsymbol{v}_n)),\\ \boldsymbol{u}_{n+1} &= \mathrm{e}^{\tau K^{\boldsymbol{u}}}\left(\boldsymbol{u}_n + \frac{\tau}{2}\boldsymbol{g}^{\boldsymbol{u}}(\boldsymbol{u}_n,\boldsymbol{v}_n)\right) + \frac{\tau}{2}\boldsymbol{g}^{\boldsymbol{u}}(\boldsymbol{u}_{n2},\boldsymbol{v}_{n2}),\\ \boldsymbol{v}_{n+1} &= \mathrm{e}^{\tau K^{\boldsymbol{v}}}\left(\boldsymbol{v}_n + \frac{\tau}{2}\boldsymbol{g}^{\boldsymbol{v}}(\boldsymbol{u}_n,\boldsymbol{v}_n)\right) + \frac{\tau}{2}\boldsymbol{g}^{\boldsymbol{v}}(\boldsymbol{u}_{n2},\boldsymbol{v}_{n2}).\end{aligned}$$

The same method can be recovered also by multiplying the two equations of system (2) by the integration factors $\mathrm{e}^{-tK^{\boldsymbol{u}}}$ and $\mathrm{e}^{-tK^{\boldsymbol{v}}}$, respectively, and applying the explicit Heun method (see for instance Reference [24]). The just mentioned Lawson scheme can be efficiently implemented by directly exploiting the Kronecker sum structure of the involved matrices. Indeed, in the two-dimensional case (that is, for a matrix $\tau K = \tau A_2 \oplus \tau A_1$), we can use the well-known formula (see, for instance, Reference [32])

$$\mathrm{e}^{\tau K}\boldsymbol{w} = \mathrm{vec}\left(\mathrm{e}^{\tau A_1}\boldsymbol{W}\left(\mathrm{e}^{\tau A_2}\right)^{\mathsf{T}}\right),$$

where $\boldsymbol{W}$ is a matrix of size $n_1 \times n_2$ and vec is the operator which stacks the columns of the input matrix into a suitable column vector such that $\mathrm{vec}(\boldsymbol{W}) = \boldsymbol{w}$. This allows for computing the action of the matrix exponential without assembling the matrix K and by performing two products with the dense matrices $\mathrm{e}^{\tau A_1}$ and $\mathrm{e}^{\tau A_2}$, respectively. For a matrix τK with d-dimensional Kronecker sum structure (3), the formula above can be generalized by employing the representation

$$\mathrm{e}^{\tau K}\boldsymbol{w} = \mathrm{vec}\left(\boldsymbol{W}\times_1 \mathrm{e}^{\tau A_1}\times_2\cdots\times_d \mathrm{e}^{\tau A_d}\right) = \mathrm{vec}\left(\boldsymbol{W}\bigtimes_{\mu=1}^{d}\mathrm{e}^{\tau A_\mu}\right), \tag{5}$$

see Reference [11], where $\boldsymbol{W}$ is an order-d tensor of size $n_1\times\cdots\times n_d$. Here, $\times_\mu$ denotes the *μ-mode product*, which multiplies the matrix $\mathrm{e}^{\tau A_\mu}$ onto the μ-fibers (i.e., the generalizations to tensors of matrix columns and rows) of the tensor $\boldsymbol{W}$. Notice that also in the d-dimensional case it can be efficiently realized by means of high performance level 3 BLAS operations at a computational cost $\mathcal{O}(n_\mu N)$, since the matrix exponentials $\mathrm{e}^{\tau A_\mu}$ are dense matrices. The concatenation of d different μ-mode products, that we denoted as $\bigtimes_{\mu=1}^{d}$, is called a *Tucker operator*, and clearly has computational cost $\mathcal{O}((n_1+\cdots+n_d)N)$. The exponentials of the matrices τA_μ can be efficiently computed by polynomial or rational approximations (see References [1,3,15,34]) together with a scaling and squaring algorithm, or even by a spectral decomposition (see Reference [17]). We refer the reader also to Reference [14] for more details about formula (5), the Tucker operator and related operations (implemented in a MATLAB package[1]).

Using the tensor formalism just introduced, it is also possible to equivalently rewrite vector ODE system (2) as

$$\begin{cases}\boldsymbol{U}'(t) = \displaystyle\sum_{\mu=1}^{d}(\boldsymbol{U}(t)\times_\mu A_\mu^{\boldsymbol{u}}) + \boldsymbol{G}^{\boldsymbol{u}}(\boldsymbol{U}(t),\boldsymbol{V}(t)),\\ \boldsymbol{V}'(t) = \displaystyle\sum_{\mu=1}^{d}(\boldsymbol{V}(t)\times_\mu A_\mu^{\boldsymbol{v}}) + \boldsymbol{G}^{\boldsymbol{v}}(\boldsymbol{U}(t),\boldsymbol{V}(t)),\end{cases} \tag{6}$$

where

$$\begin{aligned}&\mathrm{vec}(\boldsymbol{U}(t)) = \boldsymbol{u}(t), \qquad &&\mathrm{vec}(\boldsymbol{G}^{\boldsymbol{u}}(\boldsymbol{U}(t),\boldsymbol{V}(t))) = \boldsymbol{g}^{\boldsymbol{u}}(\boldsymbol{u}(t),\boldsymbol{v}(t)),\\ &\mathrm{vec}(\boldsymbol{V}(t)) = \boldsymbol{v}(t), \qquad &&\mathrm{vec}(\boldsymbol{G}^{\boldsymbol{v}}(\boldsymbol{U}(t),\boldsymbol{V}(t))) = \boldsymbol{g}^{\boldsymbol{v}}(\boldsymbol{u}(t),\boldsymbol{v}(t)).\end{aligned}$$

Notice that, in the formulation above, the actions of $K^{\boldsymbol{u}}$ and $K^{\boldsymbol{v}}$ are expressed in tensor form thanks to the identities

$$K^{\boldsymbol{u}}\boldsymbol{u}(t) = \mathrm{vec}\left(\sum_{\mu=1}^{d}\boldsymbol{U}(t)\times_\mu A_\mu^{\boldsymbol{u}}\right) \quad\text{and}\quad K^{\boldsymbol{v}}\boldsymbol{v}(t) = \mathrm{vec}\left(\sum_{\mu=1}^{d}\boldsymbol{V}(t)\times_\mu A_\mu^{\boldsymbol{v}}\right).$$

Each of these two evaluations has a computational cost equivalent to that of a single Tucker operator. Then, the aforementioned second order Lawson scheme in tensor formulation is

[1] Available at https://github.com/caliarim/KronPACK, commit `562a9da`.

$$
\begin{aligned}
\boldsymbol{U}_{n2} &= \left(\boldsymbol{U}_n + \tau \boldsymbol{G}^{\boldsymbol{u}}(\boldsymbol{U}_n, \boldsymbol{V}_n)\right) \bigtimes_{\mu=1}^{d} \mathrm{e}^{\tau A_\mu^{\boldsymbol{u}}}, \\
\boldsymbol{V}_{n2} &= \left(\boldsymbol{V}_n + \tau \boldsymbol{G}^{\boldsymbol{v}}(\boldsymbol{U}_n, \boldsymbol{V}_n)\right) \bigtimes_{\mu=1}^{d} \mathrm{e}^{\tau A_\mu^{\boldsymbol{v}}}, \\
\boldsymbol{U}_{n+1} &= \left(\boldsymbol{U}_n + \frac{\tau}{2} \boldsymbol{G}^{\boldsymbol{u}}(\boldsymbol{U}_n, \boldsymbol{V}_n)\right) \bigtimes_{\mu=1}^{d} \mathrm{e}^{\tau A_\mu^{\boldsymbol{u}}} + \frac{\tau}{2} \boldsymbol{G}^{\boldsymbol{u}}(\boldsymbol{U}_{n2}, \boldsymbol{V}_{n2}), \\
\boldsymbol{V}_{n+1} &= \left(\boldsymbol{V}_n + \frac{\tau}{2} \boldsymbol{G}^{\boldsymbol{v}}(\boldsymbol{U}_n, \boldsymbol{V}_n)\right) \bigtimes_{\mu=1}^{d} \mathrm{e}^{\tau A_\mu^{\boldsymbol{v}}} + \frac{\tau}{2} \boldsymbol{G}^{\boldsymbol{v}}(\boldsymbol{U}_{n2}, \boldsymbol{V}_{n2}).
\end{aligned}
\tag{7}
$$

For the two-dimensional case, the scheme can be written without introducing tensor notation as

$$
\begin{aligned}
\boldsymbol{U}_{n2} &= \mathrm{e}^{\tau A_1^{\boldsymbol{u}}} \left(\boldsymbol{U}_n + \tau \boldsymbol{G}^{\boldsymbol{u}}(\boldsymbol{U}_n, \boldsymbol{V}_n)\right) \left(\mathrm{e}^{\tau A_2^{\boldsymbol{u}}}\right)^{\mathsf{T}}, \\
\boldsymbol{V}_{n2} &= \mathrm{e}^{\tau A_1^{\boldsymbol{v}}} \left(\boldsymbol{V}_n + \tau \boldsymbol{G}^{\boldsymbol{v}}(\boldsymbol{U}_n, \boldsymbol{V}_n)\right) \left(\mathrm{e}^{\tau A_2^{\boldsymbol{v}}}\right)^{\mathsf{T}}, \\
\boldsymbol{U}_{n+1} &= \mathrm{e}^{\tau A_1^{\boldsymbol{u}}} \left(\boldsymbol{U}_n + \frac{\tau}{2} \boldsymbol{G}^{\boldsymbol{u}}(\boldsymbol{U}_n, \boldsymbol{V}_n)\right) \left(\mathrm{e}^{\tau A_2^{\boldsymbol{u}}}\right)^{\mathsf{T}} + \frac{\tau}{2} \boldsymbol{G}^{\boldsymbol{u}}(\boldsymbol{U}_{n2}, \boldsymbol{V}_{n2}), \\
\boldsymbol{V}_{n+1} &= \mathrm{e}^{\tau A_1^{\boldsymbol{v}}} \left(\boldsymbol{V}_n + \frac{\tau}{2} \boldsymbol{G}^{\boldsymbol{v}}(\boldsymbol{U}_n, \boldsymbol{V}_n)\right) \left(\mathrm{e}^{\tau A_2^{\boldsymbol{v}}}\right)^{\mathsf{T}} + \frac{\tau}{2} \boldsymbol{G}^{\boldsymbol{v}}(\boldsymbol{U}_{n2}, \boldsymbol{V}_{n2}).
\end{aligned}
$$

We will refer to method (7) as Lawson2b.

Different approximations in the integrals of formula (4) lead to other exponential integrators. In particular, if we interpolate at $\theta = 0$ and $\theta = 1$ the nonlinear functions $\boldsymbol{g}^{\boldsymbol{u}}$ and $\boldsymbol{g}^{\boldsymbol{v}}$ by a polynomial, and we introduce an intermediate stage for the approximation at $\theta = 1$, we get the exponential Runge–Kutta scheme of order two

$$
\begin{aligned}
\boldsymbol{u}_{n2} &= \boldsymbol{u}_n + \tau \varphi_1(\tau K^{\boldsymbol{u}})(K^{\boldsymbol{u}} \boldsymbol{u}_n + \boldsymbol{g}^{\boldsymbol{u}}(\boldsymbol{u}_n, \boldsymbol{v}_n)), \\
\boldsymbol{v}_{n2} &= \boldsymbol{v}_n + \tau \varphi_1(\tau K^{\boldsymbol{v}})(K^{\boldsymbol{v}} \boldsymbol{v}_n + \boldsymbol{g}^{\boldsymbol{v}}(\boldsymbol{u}_n, \boldsymbol{v}_n)), \\
\boldsymbol{u}_{n+1} &= \boldsymbol{u}_{n2} + \tau \varphi_2(\tau K^{\boldsymbol{u}}) \left(\boldsymbol{g}^{\boldsymbol{u}}(\boldsymbol{u}_{n2}, \boldsymbol{v}_{n2}) - \boldsymbol{g}^{\boldsymbol{u}}(\boldsymbol{u}_n, \boldsymbol{v}_n)\right), \\
\boldsymbol{v}_{n+1} &= \boldsymbol{v}_{n2} + \tau \varphi_2(\tau K^{\boldsymbol{v}}) \left(\boldsymbol{g}^{\boldsymbol{v}}(\boldsymbol{u}_{n2}, \boldsymbol{v}_{n2}) - \boldsymbol{g}^{\boldsymbol{v}}(\boldsymbol{u}_n, \boldsymbol{v}_n)\right),
\end{aligned}
\tag{8}
$$

known as ETD2RK (see, for instance, Reference [6]). The matrix φ_ℓ functions, for a generic matrix $X \in \mathbb{C}^{N \times N}$, are defined by

$$
\varphi_\ell(X) = \int_0^1 \frac{\theta^{\ell-1}}{(\ell-1)!} \mathrm{e}^{(1-\theta)X} d\theta, \quad \ell > 0. \tag{9}
$$

Several techniques are available in the literature to compute the matrix φ_ℓ functions (see References [6,10,26] for methods based on rational or polynomial approximations or quadrature formulas, suitable for small sized matrices) or their action to vectors (see References [12,19,27] for Krylov methods with incomplete orthogonalization, suitable for large sized and sparse matrices).

Unfortunately, even if $X = \tau K$, property (5) is not valid for the matrix φ_ℓ functions. However, it is still possible to exploit the Kronecker sum structure of K to compute the action of a matrix φ_ℓ function on a vector $\boldsymbol{w}$ efficiently. In fact, if we introduce a quadrature formula for integral definition (9), we can approximate the desired quantity by computing the actions $\mathrm{e}^{(1-\theta_i)\tau K} \boldsymbol{w}$ using Tucker operators, being θ_i the quadrature nodes. In this way, the approximation via quadrature formula requires one Tucker operator for each quadrature node. The total number of quadrature nodes can be kept reasonably small by employing a modified scaling and squaring technique for the φ_ℓ functions. Each iteration of the squaring procedure requires then the action of a matrix exponential, which can be realized again by a Tucker operator. Such an approach is employed in Reference [13] and is called PHIKS (PHI-functions of Kronecker Sums), while a similar technique is presented in Reference [16]. We refer to those manuscripts for the details on the choice of the quadrature formulas and of the scaling parameter.

3. Directional splitting of the matrix φ_ℓ functions

As already mentioned, the splitting property

$$
\mathrm{e}^{\tau K} \boldsymbol{w} = \mathrm{e}^{\tau A_{\otimes 1}} \cdots \mathrm{e}^{\tau A_{\otimes d}} \boldsymbol{w},
$$

which is at the basis of equivalence (5), does not extend directly to the φ_ℓ functions. However, it has been observed in Reference [10] that a simple directional splitting of the matrix φ_ℓ functions, that is

$$
\varphi_\ell(\tau K)\boldsymbol{w} = (\ell!)^{d-1} \varphi_\ell(\tau A_{\otimes 1}) \cdots \varphi_\ell(\tau A_{\otimes d}) \boldsymbol{w} + \mathcal{O}(\tau^2), \tag{10a}
$$

gives an approximation compatible with second order integrators. The advantage of using the above approximation is that it can be efficiently computed in tensor form (similarly to the matrix exponential case) as

$$\varphi_\ell(\tau A_{\otimes 1})\cdots\varphi_\ell(\tau A_{\otimes d})\boldsymbol{w} = \mathrm{vec}\left(\boldsymbol{W}\bigtimes_{\mu=1}^{d}\varphi_\ell(\tau A_\mu)\right). \tag{10b}$$

The approximation introduced by the directional splitting can affect in general the magnitude of the error when applied in exponential integrators. However, we will see later in the numerical examples that the overall computational cost required to reach the same level of accuracy is smaller. This is possible since only small sized matrix φ_ℓ functions have to be computed, and the Tucker operator (which can be efficiently realized as explained in Section 2) has to be applied.

In particular, if we embed splitting formulas (10) in ETD2RK integrator (8), we obtain a scheme that we call *directional split ETD2RK*, and more concisely ETD2RKds. The complete expression of the proposed scheme for the solution of tensor ODE system (6) is then

$$\begin{aligned}
\boldsymbol{U}_{n2} &= \boldsymbol{U}_n + \tau\left(\sum_{\mu=1}^{d}(\boldsymbol{U}_n\times_\mu A_\mu^{\boldsymbol{u}}) + \boldsymbol{G}^{\boldsymbol{u}}(\boldsymbol{U}_n,\boldsymbol{V}_n)\right)\bigtimes_{\mu=1}^{d}\varphi_1(\tau A_\mu^{\boldsymbol{u}}),\\
\boldsymbol{V}_{n2} &= \boldsymbol{V}_n + \tau\left(\sum_{\mu=1}^{d}(\boldsymbol{V}_n\times_\mu A_\mu^{\boldsymbol{v}}) + \boldsymbol{G}^{\boldsymbol{v}}(\boldsymbol{U}_n,\boldsymbol{V}_n)\right)\bigtimes_{\mu=1}^{d}\varphi_1(\tau A_\mu^{\boldsymbol{v}}),\\
\boldsymbol{U}_{n+1} &= \boldsymbol{U}_{n2} + 2^{d-1}\tau(\boldsymbol{G}^{\boldsymbol{u}}(\boldsymbol{U}_{n2},\boldsymbol{V}_{n2}) - \boldsymbol{G}^{\boldsymbol{u}}(\boldsymbol{U}_n,\boldsymbol{V}_n))\bigtimes_{\mu=1}^{d}\varphi_2(\tau A_\mu^{\boldsymbol{u}}),\\
\boldsymbol{V}_{n+1} &= \boldsymbol{V}_{n2} + 2^{d-1}\tau(\boldsymbol{G}^{\boldsymbol{v}}(\boldsymbol{U}_{n2},\boldsymbol{V}_{n2}) - \boldsymbol{G}^{\boldsymbol{v}}(\boldsymbol{U}_n,\boldsymbol{V}_n))\bigtimes_{\mu=1}^{d}\varphi_2(\tau A_\mu^{\boldsymbol{v}}),
\end{aligned} \tag{11}$$

which, for the two-dimensional case, can be written as

$$\begin{aligned}
\boldsymbol{U}_{n2} &= \boldsymbol{U}_n + \tau\varphi_1(\tau A_1^{\boldsymbol{u}})\left(A_1^{\boldsymbol{u}}\boldsymbol{U}_n + \boldsymbol{U}_n(A_2^{\boldsymbol{u}})^{\mathsf{T}} + \boldsymbol{G}^{\boldsymbol{u}}(\boldsymbol{U}_n,\boldsymbol{V}_n)\right)\left(\varphi_1(\tau A_2^{\boldsymbol{u}})\right)^{\mathsf{T}},\\
\boldsymbol{V}_{n2} &= \boldsymbol{V}_n + \tau\varphi_1(\tau A_1^{\boldsymbol{v}})\left(A_1^{\boldsymbol{v}}\boldsymbol{V}_n + \boldsymbol{V}_n(A_2^{\boldsymbol{v}})^{\mathsf{T}} + \boldsymbol{G}^{\boldsymbol{v}}(\boldsymbol{U}_n,\boldsymbol{V}_n)\right)\left(\varphi_1(\tau A_2^{\boldsymbol{v}})\right)^{\mathsf{T}},\\
\boldsymbol{U}_{n+1} &= \boldsymbol{U}_{n2} + 2\tau\varphi_2(\tau A_1^{\boldsymbol{u}})(\boldsymbol{G}^{\boldsymbol{u}}(\boldsymbol{U}_{n2},\boldsymbol{V}_{n2}) - \boldsymbol{G}^{\boldsymbol{u}}(\boldsymbol{U}_n,\boldsymbol{V}_n))\left(\varphi_2(\tau A_2^{\boldsymbol{u}})\right)^{\mathsf{T}},\\
\boldsymbol{V}_{n+1} &= \boldsymbol{V}_{n2} + 2\tau\varphi_2(\tau A_1^{\boldsymbol{v}})(\boldsymbol{G}^{\boldsymbol{v}}(\boldsymbol{U}_{n2},\boldsymbol{V}_{n2}) - \boldsymbol{G}^{\boldsymbol{v}}(\boldsymbol{U}_n,\boldsymbol{V}_n))\left(\varphi_2(\tau A_2^{\boldsymbol{v}})\right)^{\mathsf{T}}.
\end{aligned}$$

Notice that ETD2RKds scheme (11) is second order accurate in time, as can be easily seen by comparing it with the original ETD2RK method and using formulas (10). The expected numerical rate of convergence of the scheme will also be verified later on in the numerical examples, see the tables in Section 4. Clearly, once the relevant small sized matrix functions have been computed, the realization of a time step requires four Tucker operators and two actions of matrices in Kronecker form. Hence, its computational cost mainly depends on the number of degrees of freedom N, and not on the time step size τ. Finally, we remark that scheme (11) can be straightforwardly generalized to systems of semilinear equations with more than two components.

4. Numerical examples

In this section, we show the effectiveness of the proposed directional split exponential integrator ETD2RKds by applying it to several two-dimensional and three-dimensional models of great interest. All the problems are discretized in space by second order centered finite differences, both for the diffusion and the advection terms, on a grid of equispaced points with $n_1 = \ldots = n_d = n$ (for a total number of degrees of freedom equal to $N = n^d$). The usage of centered finite differences for the advection terms is justified by the fact that the cell Péclet numbers are much less than one in all the relevant examples. The discretization of the homogeneous Neumann boundary conditions is directly embedded into the matrices. Notice that this choice gives rise to tridiagonal matrices $A_\mu \in \mathbb{R}^{n\times n}$. We remark, however, that this structure is not required by our method. Concerning the computation of the small sized matrix φ_ℓ functions needed by ETD2RKds, since we will use it in a constant time step size scenario, we compute them at the beginning of the time integration by the `phiquad` function[2] (introduced in Reference [10]). In short, it approximates the φ_ℓ functions by employing the Gauss–Legendre–Lobatto quadrature rule applied to formula (9), in combination with the modified scaling and squaring algorithm [38]. The matrix exponentials needed in this procedure are computed by the built-in MATLAB function `expm`, that, for nonsymmetric matrices, implements a variable order rational Padé algorithm with scaling and squaring [1]. Notice that, instead of using `phiquad`, other techniques could also be employed, such as direct Padé approximation of the φ_ℓ functions (as it is done in the function `phipade` of Reference [6]). As already observed in Reference [11], also for the examples presented here this initial phase has a negligible computational cost with respect to the total integration time, since the size of the directional

[2] Available at https://github.com/caliarim/phisplit, commit `c67abe3`.

matrices A_μ is small. In fact, the maximum size of exponential-like matrix functions computed is 200×200 in the two-dimensional DIB model of Section 4.4.

We compare our scheme with other popular exponential or implicit integrators, developed both for the two-dimensional and the three-dimensional case. The exponential schemes are employed with constant time step size. All the methods are implemented in MATLAB language, and the arising matrix functions or linear systems are computed by *direct* methods (that is, without using iterative procedures). In particular, we consider the following schemes.

- An implementation of Lawson2b method (7) in tensor formulation, in which the needed small sized matrix exponentials are computed, once and for all at the beginning of the time integration, by the built-in MATLAB function `expm`. As for the ETD2RKds scheme, this initial phase has a negligible computational cost. Notice also that this scheme has no directional splitting error.
- An implementation of the standard ETD2RK method in which the action of the relevant matrix functions is realized in tensor formulation by the PHIKS routine[3] (see end of Section 2 and Reference [12] for a detailed explanation). The required small sized matrix exponentials are computed using the function `expm`. The scaling parameter and the number of quadrature nodes are automatically chosen by PHIKS, based on the input tolerance. In fact, the tolerances range from 5e−6 to 1e−3, depending on the specific numerical example.
- An implementation of the Exponential Time Differencing Real Distinct Poles Integrating Factor method (ETD-RDP-IF, see References [4,30] and the accompanying MATLAB software[4]) for the solution of vector ODE system (2). The exponential functions within the predictor and corrector terms are approximated by a first-order Padé expansion and a second-order rational approximation with simple real distinct poles. After dimensional splitting, thanks to the chosen space discretization, the arising linear systems involving the large sized matrices $A_{\otimes\mu} \in \mathbb{R}^{N\times N}$ (that have three nonzero diagonals) are solved by an adapted Thomas algorithm. We remark that this time integrator could benefit from a three-core parallel implementation, which we do not consider here.
- The MATLAB solver `ode23tb`, which implements a variable step size diagonally implicit Runge–Kutta pair 2(3) (DIRK23, see Reference [23]) for vector system (2), suggested for stiff problems with "crude error tolerances", fed with the exact Jacobian of the system.

Moreover, we also test the performance of the MATLAB solver `ode23`, which implements a variable time step explicit Runge–Kutta pair 3(2) (RK32, see Reference [8]) for ODE system (2). This method is considered in the comparisons to be sure that the examples, with the selected parameters and space discretizations, are in a stiff regime in which explicit methods do suffer from a time step size restriction. As a consequence, we expect that the variable step size mechanism will shrink the time step size to make the scheme operate in its stability region. Hence, the method will be able to reach different accuracies with approximately the same number of time steps and thus of computational load (see, for instance, the first experiment of Section 4.1 for more details). Whenever calling the built-in MATLAB integrators we always use sparse matrices and, since we are just interested in the solution at final times, we employ a proper output function (through the option `OutputFcn`) in order not to save the solutions at intermediate steps and hence waste memory. We believe that running the experiments with well-established built-in MATLAB integrators is valuable and can constitute a common and useful benchmark for the community working in the field of ADR systems using this popular scientific language.

In all the numerical examples, we first compare the performances of the methods in reaching a common range of accuracies with respect to a reference solution computed with ETD2RKds and a sufficiently large number of time steps. We therefore select, for the different methods, some sequences of time steps (for the fixed time step size implementations) or input tolerances (for the variable time step size implementations), and measure the overall needed wall-clock time. The error is computed as the 2-norm of the relative Frobenius norm of the solutions $\boldsymbol{U}$ and $\boldsymbol{V}$ at final time. First of all, this experiment is useful to check that all the constant time step size methods exhibit the expected rate of convergence. In addition, we compute the computational load of a single step of our proposed exponential scheme. Then, in a second experiment, we run ETD2RKds up to a larger final time, plot the corresponding component $\boldsymbol{U}$, and report the wall-clock time. Moreover, for the systems of diffusion–reaction equations that lead to the formation of a stationary Turing pattern, we show the evolution dynamics by reporting the discretized spatial mean of $u(t_n, \boldsymbol{x})$

$$\langle \boldsymbol{U}_n \rangle \approx \frac{1}{|\Omega|} \int_\Omega u(t_n, \boldsymbol{x}) d\boldsymbol{x},$$

and the time increment $\|\boldsymbol{U}_{n+1} - \boldsymbol{U}_n\|_{\mathrm{F}}$ in the Frobenius norm, as it is typically done in the literature (see References [2,17], for instance).

All the experiments are performed on an Intel® Core™ i7-10750H CPU with six physical cores and 16 GB of RAM. As a software, we use MathWorks MATLAB® R2022a. All the codes to reproduce the examples can be found in a maintained GitHub repository.[5]

[3] Available at https://github.com/caliarim/phiks, commit `97ae469`.

[4] Available at https://github.com/kleefeld80/ETDRDPIF, commit `2647b6e`.

[5] Available at https://github.com/cassinif/ExpADRds.

Table 1
Number of time steps (or input tolerance), wall-clock time (in seconds), relative error at final time, and observed numerical order of convergence for the solution of 2D Schnakenberg model (12) up to $T = 0.25$ with different integrators. See also Fig. 1a for a graphical representation.

ETD2RKds				ETD2RK			
steps	time (s)	error	order	steps	time (s)	error	order
3000	3.82	1.78e−3	—	3000	176.98	1.65e−3	—
4000	5.17	1.01e−3	1.98	4000	227.38	9.33e−4	1.97
5000	6.43	6.42e−4	2.02	5000	282.61	5.95e−4	2.02
6000	7.73	4.41e−4	2.06	6000	334.67	4.09e−4	2.06
Lawson2b				ETD-RDP-IF			
steps	time (s)	error	order	steps	time (s)	error	order
14000	10.56	3.58e−3	—	14000	97.40	1.99e−3	—
18000	13.33	2.16e−3	2.01	18000	125.23	1.21e−3	2.00
22000	16.07	1.44e−3	2.02	22000	152.67	8.05e−4	2.02
26000	19.01	1.03e−3	2.03	26000	180.81	5.71e−4	2.05
DIRK23				RK32			
tolerance	time (s)	error		tolerance	time (s)	error	
1e−6	10.58	5.12e−3		1e−2	366.16	4.37e−3	
5e−7	11.57	2.91e−3		8e−3	371.76	2.93e−3	
1e−7	16.40	9.37e−4		4e−3	373.26	1.10e−3	
5e−8	18.66	5.54e−4		1e−3	377.94	6.47e−4	

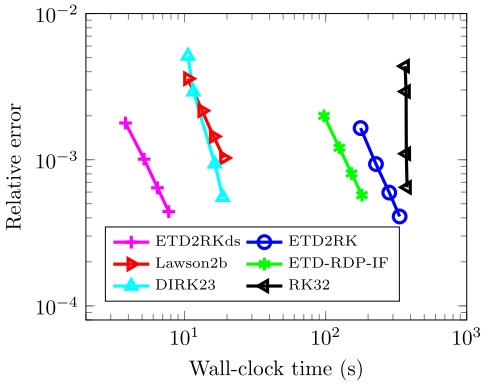

Fig. 1a. Results for the simulation of 2D Schnakenberg model (12) with $N = 150^2$ spatial discretization points. The number of time steps (or input tolerance) for each integrator is reported in Table 1. The final simulation time is $T = 0.25$.

4.1. Two-dimensional Schnakenberg model

We consider the Schnakenberg model (see References [2,5,17,35])

$$\begin{cases} \partial_t u = \delta^u \Delta u + \rho(a^u - u + u^2 v), \\ \partial_t v = \delta^v \Delta v + \rho(a^v - u^2 v), \end{cases} \tag{12}$$

in the spatial domain $\Omega = [0,1]^2$. The unknowns u and v represent two chemical concentrations in autocatalytic reactions. The parameters, taken from Reference [2], are $\delta^u = 1$, $\delta^v = 10$, $\rho = 1000$, $a^u = 0.1$, and $a^v = 0.9$. The equilibrium $(u_e, v_e) = (a^u + a^v, a^v/(a^u + a^v)^2)$ is susceptible of Turing instability. The initial data are $u_0 = u_e + 10^{-5} \cdot \mathcal{U}(0,1)$ and $v_0 = v_e + 10^{-5} \cdot \mathcal{U}(0,1)$, where here and throughout the experiments $\mathcal{U}(0,1)$ denotes the uniformly distributed random variable in $(0,1)$. The MATLAB random generator seed is set to the value 0. The spatial domain is discretized with a grid of $N = 150^2$ points. The final simulation time is $T = 0.25$. The detailed outcome of the experiment is reported in Table 1, in which we also indicate the numbers of time steps (or input tolerances) and the obtained numerical order of convergence for the different integrators. In particular, concerning the MATLAB variable step size Runge–Kutta integrators, in this and in all the subsequent examples the options `AbsTol` and `RelTol` are set to the values of the reported tolerances. The results are also graphically presented in a precision diagram in Fig. 1a.

First of all, we observe that all the constant time step size methods exhibit the expected rate of convergence (see the relevant columns in Table 1). Then, concerning Fig. 1a, the plot is clearly divided into two parts. On the right we have the more expensive methods. As expected, the RK32 method performs poorly in terms of computational time. Moreover, we observe that it requires about the same computational load for the four different input tolerances. In fact, the numbers of time steps performed by the

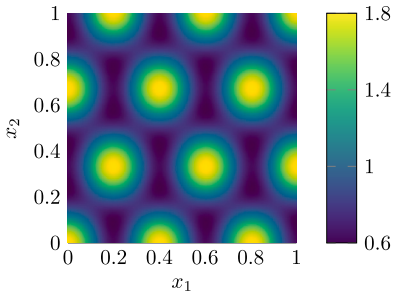

Fig. 1b. Turing pattern (u component) for 2D Schnakenberg model (12) obtained at final time $T = 2$ with $N = 150^2$ space discretization points with ETD2RKds. The time step size employed is $\tau = 5\text{e}{-4}$. The simulation wall-clock time is 5 seconds.

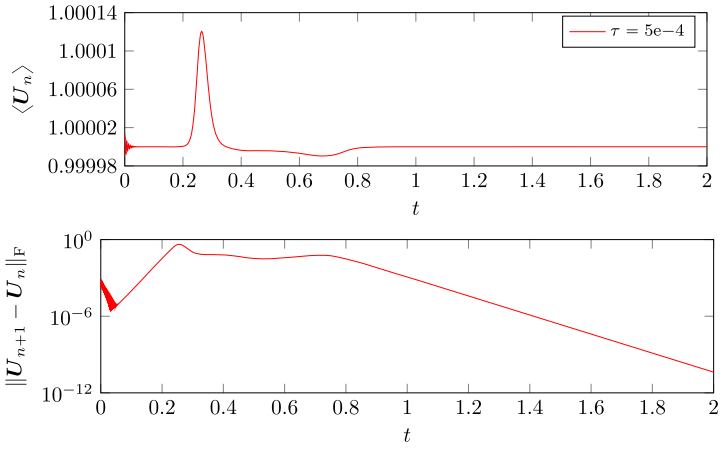

Fig. 1c. Indicators for the time dynamics of 2D Schnakenberg model (12) solved up to final time $T = 2$ with ETD2RKds and time step size $\tau = 5\text{e}{-4}$. The top plot refers to the spatial mean $\langle \boldsymbol{U}_n \rangle$ while the bottom plot depicts the time increment $\|\boldsymbol{U}_{n+1} - \boldsymbol{U}_n\|_F$.

RK32 method are 176797, 176798, 176800, and 176802, respectively. This behavior is expected, since the scheme is explicit and is forced to take a large number of time steps due to stability constraints. Nevertheless, due to the automatic time step selection, slightly distinct sequences of time steps are produced, and hence different final accuracies are obtained. The ETD-RDP-IF scheme (specifically designed for this type of models) performs slightly better than the ETD2RK method, both schemes being however not competitive with the remaining methods. On the left hand side of the plot, we find the implicit Runge–Kutta method, which is more than ten times faster than the explicit one, and with performances similar to the Lawson2b method. Finally, an overall neat advantage is obtained by applying the proposed method ETD2RKds, which requires an average wall-clock time of 1.3e−3 seconds per time step.

In the second experiment, we perform a simulation with ETD2RKds up to final time $T = 2$ with 4000 time steps. The observed pattern is in agreement with that already reported in the literature (compare Fig. 1b and Reference [2, Fig. 8(a)]). The overall wall-clock time of the simulation is roughly 5 seconds, which corresponds to the just mentioned average time step cost multiplied by the number of steps, as expected. Finally, we report in Fig. 1c two useful indicators for the dynamics of the system, i.e., the spatial mean and the time increments. In particular, it is clearly visible the distinction between the initial reactivity phase (up to about $t = 0.25$) and the stabilization to a stationary pattern.

4.2. Two-dimensional FitzHugh–Nagumo model

We consider the FitzHugh–Nagumo model (see Reference [2])

$$\begin{cases} \partial_t u = \delta^u \Delta u + \rho(-u(u^2 - 1) - v), \\ \partial_t v = \delta^v \Delta v + \rho a_1^v (u - a_2^v v), \end{cases} \tag{13}$$

in $\Omega = [0, \pi]^2$. The model describes the flow of an electric current through a nerve fiber. The unknowns u and v represent the electric potential and the recovery variable, respectively. The parameters, taken from Reference [2], are $\delta^u = 1$, $\delta^v = 42.1887$, $\rho = 65.731$, $a_1^v = 11$, and $a_2^v = 0.1$. With these choices, the equilibrium $(u_e, v_e) = (0, 0)$ is susceptible of Turing instability. The initial solutions are $u_0 = 10^{-3} \cdot \mathcal{U}(0, 1)$ and $v_0 = 10^{-3} \cdot \mathcal{U}(0, 1)$ with MATLAB random generator seed set to the value 0. The spatial domain is discretized by a grid of $N = 100^2$ points. In the first experiment we simulate up to $T = 10$. The numbers of time steps (or input tolerances) for each integrator, as well as the detailed outcome of the experiment, can be found in Table 2. The results are graphically presented in Fig. 2a.

Table 2
Number of time steps (or input tolerance), wall-clock time (in seconds), relative error at final time, and observed numerical order of convergence for the solution of 2D FitzHugh–Nagumo model (13) up to $T = 10$ with different integrators. See also Fig. 2a for a graphical representation.

ETD2RKds				ETD2RK			
steps	time (s)	error	order	steps	time (s)	error	order
20000	13.50	9.16e−3	—	20000	499.28	8.34e−3	—
22500	17.66	7.36e−3	1.85	22500	560.11	6.73e−3	1.83
25000	19.35	6.04e−3	1.88	25000	627.55	5.53e−3	1.86
27500	21.21	5.03e−3	1.91	27500	686.38	4.61e−3	1.89
Lawson2b				ETD-RDP-IF			
steps	time (s)	error	order	steps	time (s)	error	order
600000	277.90	3.86e−2	—	600000	1794.51	2.57e−2	—
675000	307.72	3.04e−2	2.02	675000	2025.12	2.03e−2	2.01
750000	337.09	2.46e−2	2.02	750000	2263.78	1.64e−2	2.02
825000	362.99	2.03e−2	2.02	825000	2476.61	1.35e−2	2.02
DIRK23				RK32			
tolerance	time (s)	error		tolerance	time (s)	error	
5e−7	33.59	1.33e−2		8e−7	1548.62	2.29e−2	
1e−7	47.48	7.89e−3		6e−7	1601.73	1.48e−2	
8e−8	50.95	6.39e−3		4e−7	1641.75	6.16e−3	
6e−8	53.07	5.70e−3		2e−7	1654.41	2.53e−3	

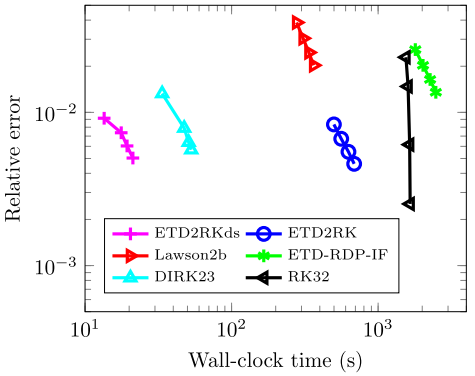

Fig. 2a. Results for the simulation of 2D FitzHugh–Nagumo model (13) with $N = 100^2$ spatial discretization points. The number of time steps (or input tolerance) for each integrator is reported in Table 2. The final simulation time is $T = 10$.

First of all, we notice that the Lawson2b and ETD-RDP-IF methods require many more time steps than the exponential Runge–Kutta methods, and still they cannot reach the same accuracies. Their wall-clock time is in fact at least one order of magnitude larger than the most efficient methods. On the other hand, as in the previous example, the ETD2RKds method is not heavily affected by the directional splitting error. Indeed, it reaches almost the same accuracies obtained by the ETD2RK method but with a much smaller computational time. In this case, the average wall-clock time per time step of ETD2RKds is 7.5e−4 seconds.

In the second experiment, we set the final simulation time to $T = 50$ and integrate the problem with the ETD2RKds method with 30000 time steps. The overall computational time is about 22 seconds. Again, the obtained pattern agrees with that reported in the literature (compare Fig. 2b and Reference [2, Fig. 4(a)]), i.e., a square pattern corresponding to the cosine modes $(4, 4)$. Also, as expected both the indicators reported in Fig. 2c tend to zero.

4.3. Three-dimensional FitzHugh–Nagumo model

We consider again the FitzHugh–Nagumo model

$$\begin{cases} \partial_t u = \delta^u \Delta u + \rho(-u(u^2 - 1) - v), \\ \partial_t v = \delta^v \Delta v + \rho a_1^v (u - a_2^v v), \end{cases} \tag{14}$$

but now in a three-dimensional domain $\Omega = [0, \pi]^3$. The parameters $\delta^u = 1$, $\delta^v = 42.1887$, $\rho = 24.649$, $a_1^v = 11$, and $a_2^v = 0.1$ were obtained by following the analysis performed in Reference [18] to achieve a stationary square pattern with modes $(2, 2, 2)$. With

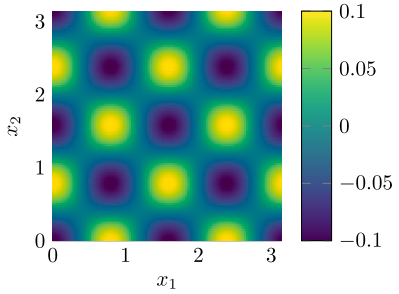

Fig. 2b. Turing pattern (u component) for 2D FitzHugh–Nagumo model (13) obtained at final time $T = 50$ with $N = 100^2$ space discretization points with ETD2RKds. The time step size employed is $\tau = 1.67\text{e}{-3}$. The simulation wall-clock time is 22 seconds.

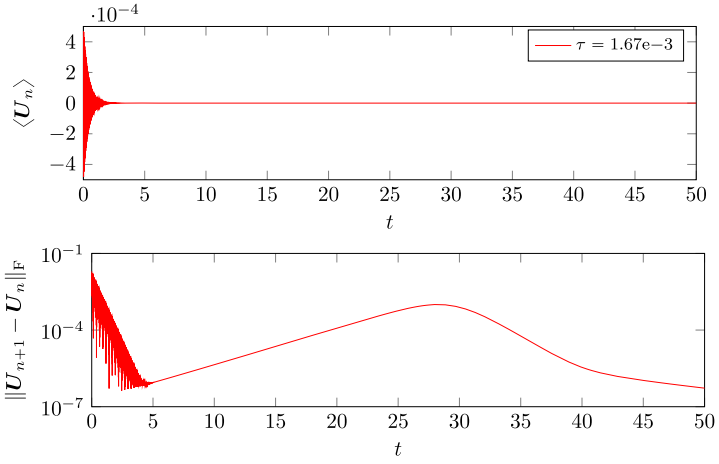

Fig. 2c. Indicators for the time dynamics of 2D FitzHugh–Nagumo model (13) solved up to final time $T = 50$ with ETD2RKds and time step size $\tau = 1.67\text{e}{-3}$. The top plot refers to the spatial mean $\langle U_n \rangle$ while the bottom plot depicts the time increment $\|U_{n+1} - U_n\|_F$.

Table 3
Number of time steps, wall-clock time (in seconds), relative error at final time, and observed numerical order of convergence for the solution of 3D FitzHugh–Nagumo model (14) up to $T = 10$ with different integrators. The DIRK23 method interrupted due to excessive memory requirements with tolerance 1e−1. The RK32 and the ETD-RDP-IF methods did not output a solution within 10^4 seconds. See also Fig. 3a for a graphical representation.

ETD2RKds				ETD2RK			
steps	time (s)	error	order	steps	time (s)	error	order
12000	101.14	3.01e−3	—	12000	626.16	2.67e−3	—
14000	119.30	2.20e−3	2.03	14000	718.69	1.96e−3	2.00
16000	138.89	1.67e−3	2.05	16000	816.36	1.49e−3	2.03
18000	154.95	1.31e−3	2.07	18000	889.12	1.17e−3	2.06

Lawson2b			
steps	time (s)	error	order
250000	1429.71	1.13e−2	—
300000	1729.65	7.86e−3	2.00
350000	2013.62	5.77e−3	2.00
400000	2320.22	4.42e−3	2.01

this choice of parameters, the equilibrium $(u_e, v_e) = (0, 0)$ is indeed susceptible of Turing instability. The initial solutions are $u_0 = 10^{-3} \cdot \mathcal{U}(0, 1)$ and $v_0 = 10^{-3} \cdot \mathcal{U}(0, 1)$, with MATLAB random generator seed set to the value 0. The spatial domain is discretized by a grid of $N = 64^3$ points. We simulate up to final time $T = 10$ with the number of time steps given in Table 3. The results, reported in the table, are also graphically depicted in Fig. 3a.

First of all, despite the aforementioned cares in the usage of the DIRK23 method from the MATLAB ODE suite, the `ode23tb` function stopped to run after some seconds due to too large memory requirements. Moreover, the RK32 and ETD-RDP-IF methods

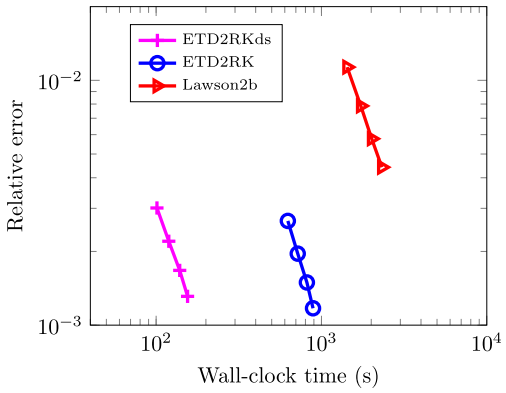

Fig. 3a. Results for the simulation of 3D FitzHugh–Nagumo model (14) with $N = 64^3$ spatial discretization points. The number of time steps for each integrator is reported in Table 3. The final simulation time is $T = 10$.

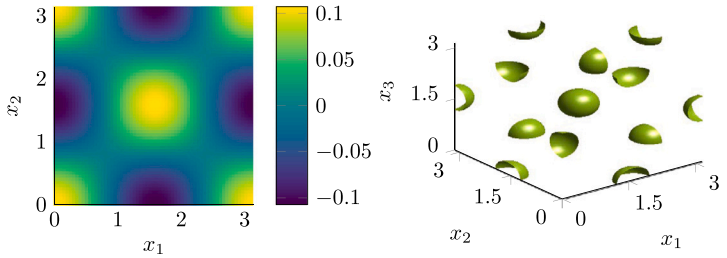

Fig. 3b. Turing pattern (u component) for 3D FitzHugh–Nagumo model (14) obtained at final time $T = 150$ with $N = 64^3$ space discretization points and ETD2RKds. The time step size employed is $\tau = 6\text{e}{-3}$. The wall-clock simulation time is 230 seconds. The reported slice (left plot) corresponds to $x_3 = 1.55$ and the isosurface value (right plot) is 0.08.

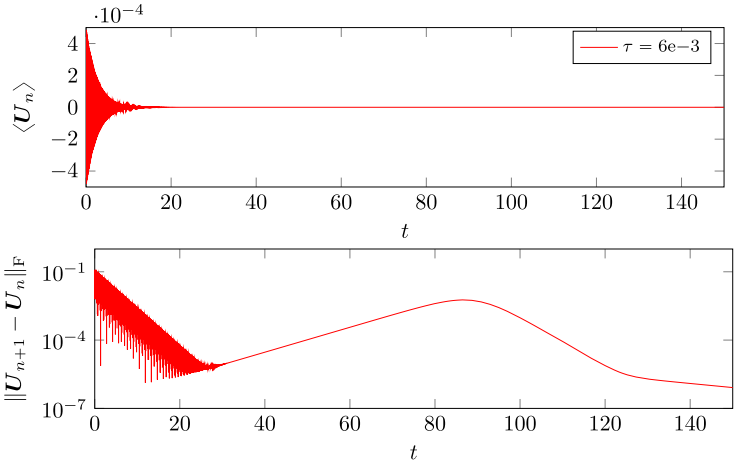

Fig. 3c. Indicators for the time dynamics of 3D FitzHugh–Nagumo model (14) solved up to final time $T = 150$ with ETD2RKds and time step size $\tau = 6\text{e}{-3}$. The top plot refers to the spatial mean $\langle U_n \rangle$ while the bottom plot depicts the time increment $\|U_{n+1} - U_n\|_F$.

did not reach accuracies comparable with the other schemes within 10^4 seconds. Among the remaining methods, ETD2RKds is again the best one, with a speedup of almost one order of magnitude with respect to the second one, that is the exponential Runge–Kutta method with the φ_ℓ functions approximated by PHIKS. The average wall-clock time per time step for ETD2RKds is 8.6e−3 seconds.

We then simulate the system up to $T = 150$ with 25000 time steps and the ETD2RKds method. We report in Fig. 3b the obtained pattern and in Fig. 3c the relevant indicators, which agree with the theoretical expectations. The overall wall-clock time is about 230 seconds.

Table 4
Number of time steps (or input tolerance), wall-clock time (in seconds), relative error at final time, and observed numerical order of convergence for the solution of 2D DIB model (15) up to $T = 2.5$ with different integrators. See also Fig. 4a for a graphical representation.

ETD2RKds				ETD2RK			
steps	time (s)	error	order	steps	time (s)	error	order
1250	5.51	1.13e−2	—	1250	133.63	9.48e−3	—
1500	6.72	7.80e−3	2.04	1500	146.90	6.59e−3	1.99
1750	7.82	5.69e−3	2.04	1750	167.88	4.84e−3	2.00
2000	9.13	4.34e−3	2.04	2000	190.90	3.70e−3	2.01
Lawson2b				**ETD-RDP-IF**			
steps	time (s)	error	order	steps	time (s)	error	order
3750	11.52	1.62e−2	—	5000	62.78	2.29e−2	—
4500	13.23	1.16e−2	1.83	6000	75.46	1.62e−2	1.91
5250	16.40	8.72e−3	1.86	7000	88.46	1.20e−2	1.92
6000	18.74	6.78e−3	1.88	8000	101.04	9.29e−3	1.94
DIRK23				**RK32**			
tolerance	time (s)	error		tolerance	time (s)	error	
5e−5	15.21	2.90e−2		9e−4	58.74	1.83e−2	
3e−5	17.04	2.04e−2		8e−4	59.27	1.40e−2	
2e−5	18.56	1.19e−2		7e−4	58.62	1.07e−2	
1e−5	23.36	5.64e−3		6e−4	58.32	7.56e−3	

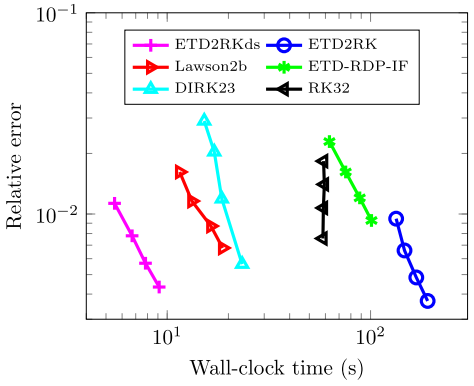

Fig. 4a. Results for the simulation of 2D DIB model (15) with $N = 200^2$ spatial discretization points. The number of time steps (or input tolerance) for each integrator is reported in Table 4. The final simulation time is $T = 2.5$.

4.4. Two-dimensional morpho-chemical DIB model

We consider the morpho-chemical DIB model (see References [2,17])

$$\begin{cases} \partial_t u = \delta^u \Delta u + \rho(a_1^u(1-v)u - a_2^u u^3 - a_3^u(v - a_4^u)), \\ \partial_t v = \delta^v \Delta v + \rho(a_1^v(1 + a_2^v u)(1-v)(1 - a_3^v(1-v)) - a_4^v v(1 + a_5^v u)(1 + a_3^v v)), \end{cases} \tag{15}$$

in $\Omega = [0, 20]^2$. The model describes the electrodeposition process for metal growth. The unknowns u and v represent the morphology of the metal deposit and its surface chemical composition, respectively. The parameters, taken from Reference [2], are $\delta^u = 1$, $\rho = 25/4$, $a_1^u = 10$, $a_2^u = 1$, $a_3^u = 66$, $a_4^u = 0.5$, $\delta^v = 20$, $a_1^v = 3$, $a_2^v = 2.5$, $a_3^v = 0.2$,

$$a_4^v = \frac{a_1^v(1 - a_4^u)(1 - a_3^v + a_3^v a_4^u)}{a_4^u(1 + a_3^v a_4^u)},$$

and $a_5^v = 1.5$. The particular choice of the parameter a_4^v makes the equilibrium $(u_e, v_e) = (0, a_4^u)$ susceptible of Turing instability. The initial conditions are $u_0 = u_e + 10^{-5} \cdot \mathcal{U}(0,1)$ and $v_0 = v_e + 10^{-5} \cdot \mathcal{U}(0,1)$. The MATLAB random generator seed is fixed to the value 123. The spatial domain is discretized with a grid of $N = 200^2$ points. We first integrate the system up to final time $T = 2.5$ with a number of time steps, or input tolerance, as reported in Table 4. The detailed results are reported in the table and also presented in the precision diagram in Fig. 4a. The plot is again divided into two parts. On the right we have the more expensive methods, with an overall wall-clock time of roughly 100 seconds. In particular, the ETD-RDP-IF method and the ETD2RK method do not perform

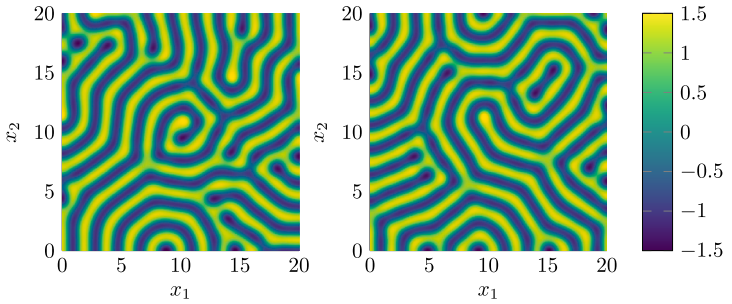

Fig. 4b. Turing pattern (u component) for 2D DIB model (15) obtained at final time $T = 100$ with $N = 200^2$ space discretization points with ETD2RKds and with time step size $\tau = 2\text{e}{-3}$ (left plot). The simulation wall-clock time is 219 seconds. We observe the same pattern also by using a smaller time step size $\tau = 1.25\text{e}{-3}$. However, if a larger time step size $\tau = 2\text{e}{-2}$ is used, a different labyrinth appears (right plot).

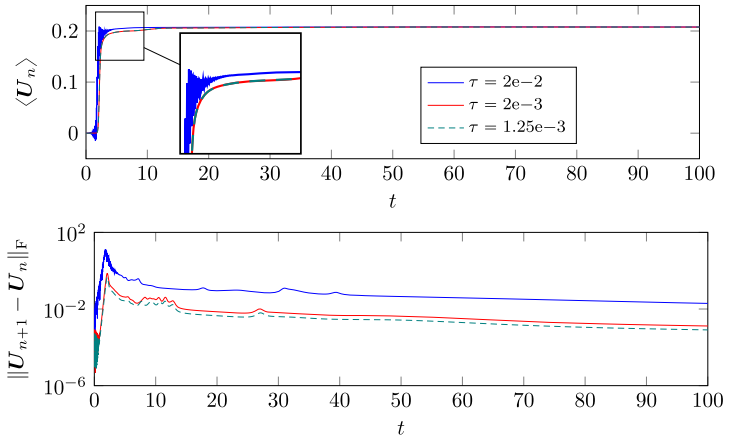

Fig. 4c. Indicators for the time dynamics of 2D DIB model (15) solved up to final time $T = 100$ with ETD2RKds and different time step sizes. The top plot refers to the spatial mean $\langle \boldsymbol{U}_n \rangle$ while the bottom plot depicts the time increment $\|\boldsymbol{U}_{n+1} - \boldsymbol{U}_n\|_F$.

better than the explicit RK23 method. This means that for the considered parameters and spatial discretization the problem does not appear to be excessively stiff. In fact, the implicit method is roughly five times faster than the explicit one and with performances similar to the Lawson2b method. The ETD2RKds method is the fastest among all and, in particular, ten times faster than ETD2RK. In this experiment, the average wall-clock time of ETD2RKds is 4.5e−3 seconds per time step.

Then, we integrate the system up to $T = 100$ with ETD2RKds and 50000 time steps. The observed steady Turing pattern is shown in Fig. 4b, left plot. The wall-clock time needed for this experiment is 219 seconds. It was observed in Reference [17] that a spatial domain not large enough or a number of discretization points too small can prevent from clearly detecting the labyrinth Turing pattern. In our experiment, where the domain and the number of discretization points have been properly chosen, we notice that if we take 5000 time steps instead of 50000, a clear steady labyrinth pattern still appears (see Fig. 4b, right plot, and the indicators in Fig. 4c). Instead, if we increase to 80000 the number of time steps, the first labyrinth pattern shows up again.

4.5. Three-dimensional advective Schnakenberg system

The diffusion–reaction Schnakenberg system models the limit cycle behaviors of two-component chemical reactions. An advection term was introduced in Reference [33] in order to study its effect on patterns (see also Reference [7]). We therefore study the system

$$\begin{cases} \partial_t u = \delta^u \Delta u - \alpha^u (\partial_{x_1} u + \partial_{x_2} u + \partial_{x_3} u) + \rho(a^u - u + u^2 v), \\ \partial_t v = \delta^v \Delta v - \alpha^v (\partial_{x_1} v + \partial_{x_2} v + \partial_{x_3} v) + \rho(a^v - u^2 v). \end{cases} \tag{16}$$

The computational domain is $\Omega = [0,1]^3$, the advection parameters are $\alpha^u = \alpha^v = 0.01$, the diffusion parameters are $\delta^u = 0.05$ and $\delta^v = 1$, the reaction parameter is $\rho = 100$, and the concentration parameters are $a^u = 0.1305$ and $a^v = 0.7695$, respectively. As initial solution we take

$$\begin{cases} u_0(x_1, x_2, x_3) = a^u + a^v + 10^{-5} \cdot \mathrm{e}^{-100((x_1-1/3)^2 + (x_2-1/2)^2 + (x_3-1/3)^2)}, \\ v_0(x_1, x_2, x_3) = \dfrac{a^v}{(a^u + a^v)^2}, \end{cases}$$

Table 5
Number of time steps (or input tolerance), wall-clock time (in seconds), relative error at final time, and observed numerical order of convergence for the solution of 3D advective Schnakenberg model (16) up to $T = 0.4$ with different integrators. The DIRK23 method interrupted due to excessive memory requirements with tolerance 1e−1. See also Fig. 5a for a graphical representation.

ETD2RKds				ETD2RK			
steps	time (s)	error	order	steps	time (s)	error	order
50	1.35	2.34e−3	—	50	10.76	1.48e−3	—
150	3.84	3.19e−4	1.81	150	25.34	2.79e−4	1.52
250	6.74	1.26e−4	1.82	250	38.71	1.14e−4	1.76
350	9.70	6.64e−5	1.91	350	51.45	6.04e−5	1.88
Lawson2b				ETD-RDP-IF			
steps	time (s)	error	order	steps	time (s)	error	order
400	6.10	6.90e−4	—	200	52.04	9.98e−4	—
800	12.16	1.89e−4	1.87	450	116.11	2.59e−4	1.66
1200	18.45	8.59e−5	1.94	700	181.09	1.17e−4	1.79
1600	24.89	4.81e−5	2.01	950	245.61	6.61e−5	1.88

RK32		
tolerance	time (s)	error
8e−3	1075.21	8.12e−3
4e−3	1033.12	3.79e−3
8e−4	1026.94	8.60e−4
4e−4	1126.25	2.03e−4

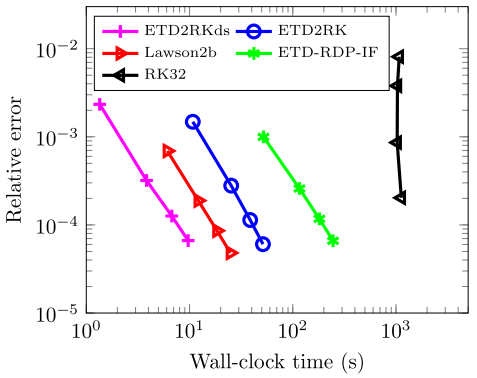

Fig. 5a. Results for the simulation of 3D advective Schnakenberg model (16) with $N = 80^3$ spatial discretization points. The number of time steps (or input tolerance) for each integrator is reported in Table 5. The final simulation time is $T = 0.4$.

which corresponds to a small deviation from the steady state solution $(u_e, v_e) = (a^u + a^v, a^v/(a^u + a^v)^2)$. We integrate the system up to final time $T = 0.4$, with the numbers of time steps (or input tolerances) given in Table 5. The degrees of freedom in space are $N = 80^3$. The detailed results are given in the table and depicted in Fig. 5a.

As in the previous three-dimensional example, the implicit Runge–Kutta method DIRK23 could not terminate due to excessive memory requirements. Concerning the other methods, we observe that explicit Runge–Kutta scheme performs poorly in terms of wall-clock time. Better performances can be gradually obtained by employing the ETD-RDP-IF method, the ETD2RK scheme and the Lawson2b integrator. Overall, the best method turns out to be again ETD2RKds, with an average wall-clock time per time step of 2.7e−2 seconds.

Then, we repeat the experiment up to final times $T = 0.8$, $T = 8$, and $T = 80$ with again $N = 80^3$ degrees of freedom in space and using as time integrator ETD2RKds with $\tau = 8e{-}3$ (i.e., performing 100, 1000 and 10000 time steps, respectively). As already observed in the literature (see Reference [7]), the initial condition evolves to a spot-like pattern, as depicted in Fig. 5b. The overall simulation times are roughly 3, 28, and 275 seconds for the final times $T = 0.8$, $T = 8$, and $T = 80$, respectively.

4.6. Three-dimensional advective Brusselator system

We consider the advective Brusselator system

$$\begin{cases} \partial_t u = \delta^u \Delta u - \alpha^u(\partial_{x_1} u + \partial_{x_2} u + \partial_{x_3} u) + u^2 v - (a_1^u + 1)u + a_2^u, \\ \partial_t v = \delta^v \Delta v - \alpha^v(\partial_{x_1} v + \partial_{x_2} v + \partial_{x_3} v) - u^2 v + a_1^u u, \end{cases} \tag{17}$$

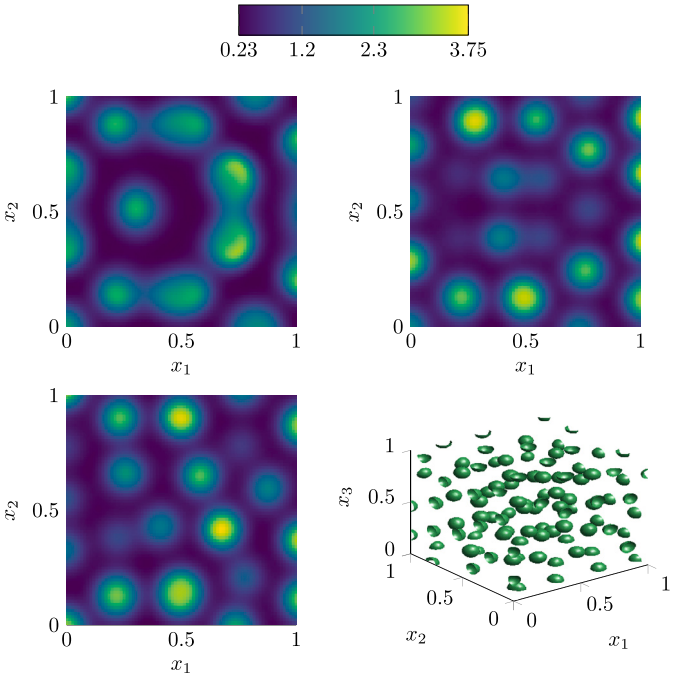

Fig. 5b. Spot-like pattern (u component) for 3D advective Schnakenberg model (16) with $N = 80^3$ space discretization points and ETD2RKds at final time $T = 0.8$ (top left), $T = 8$ (top right) and $T = 80$ (bottom left). The simulation wall-clock time is 3, 28, and 275 seconds, respectively. The time step size employed is $\tau = 8\text{e}{-3}$, and the slices correspond to $x_3 = 0.49$. The common colorbar is displayed at the top. We report also the isosurface of level 2.8 at $T = 80$ (bottom right plot).

Table 6
Number of time steps (or input tolerance), wall-clock time (in seconds), relative error at final time, and observed numerical order of convergence for the solution of 3D advective Brusselator model (17) up to $T = 1$ with different integrators. The DIRK23 method interrupted due to excessive memory requirements with tolerance 1e−1. See also Fig. 6a for a graphical representation.

ETD2RKds				ETD2RK			
steps	time (s)	error	order	steps	time (s)	error	order
50	0.48	3.46e−4	—	50	4.76	3.46e−4	—
100	0.94	7.94e−5	2.13	100	7.84	7.93e−5	2.12
150	1.25	3.43e−5	2.07	150	11.38	3.43e−5	2.07
200	1.82	1.90e−5	2.05	200	13.62	1.90e−5	2.05
Lawson2b				ETD-RDP-IF			
steps	time (s)	error	order	steps	time (s)	error	order
50	0.32	3.43e−4	—	50	6.59	3.35e−4	—
100	0.63	7.85e−5	2.13	100	12.91	7.68e−5	2.12
150	0.81	3.40e−5	2.07	150	19.24	3.32e−5	2.07
200	1.31	1.88e−5	2.05	200	25.39	1.84e−5	2.05

RK32		
tolerance	time (s)	error
8e−4	16.08	8.73e−4
6e−4	16.65	1.11e−4
1e−4	16.54	5.48e−5
5e−5	16.64	2.43e−5

presented in Reference [7]. The unknowns u and v represent the concentration of the activator and inhibitor in the chemical reaction, respectively. The spatial domain is $\Omega = [0, 1]^3$, the diffusion coefficients are $\delta^u = 0.01$, $\delta^v = 0.02$, the advection coefficients are $a_1^u = 1$, and the remaining parameters are $a_2^u = 2$, $\alpha^u = \alpha^v = 0.1$. The space discretization is performed using $N = 64^3$ points, and the initial datum is $u_0(x_1, x_2, x_3) = 1 + \sin(2\pi x_1)\sin(2\pi x_2)\sin(2\pi x_3)$ and $v_0(x_1, x_2, x_3) = 3$. The number of steps for each integrator is reported in Table 6, as well as the detailed outcome of the experiment for final time $T = 1$. Similarly to the previous examples, we also present the precision diagram in Fig. 6a.

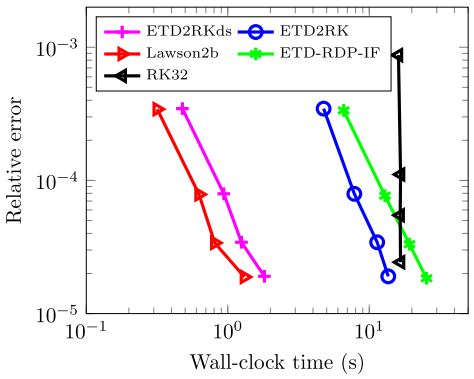

Fig. 6a. Results for the simulation of 3D advective Brusselator model (17) with $N = 64^3$ spatial discretization points. The number of time steps (or input tolerance) for each integrator is reported in Table 6. The final simulation time is $T = 1$.

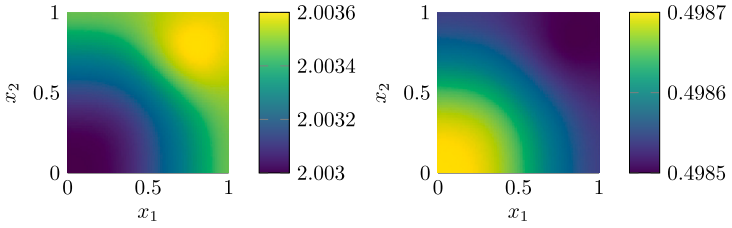

Fig. 6b. Equilibrium state of 3D advective Brusselator model (17) obtained at final time $T = 5$ with $N = 64^3$ space discretization points and ETD2RKds. The time step size employed is $\tau = 5\text{e}{-2}$. The simulation wall-clock time is 1 second. The reported slices correspond to $x_3 = 1$ for the component u (left plot) and the component v (right plot).

Again, the plot is split into two parts. Concerning the most efficient schemes, in this experiment the Lawson2b method is slightly faster than the ETD2RKds scheme, while reaching the same set of accuracies. The average wall-clock time per time step of ETD2RKds is 9.1e−3 seconds. All the remaining methods are almost one order of magnitude slower, with the DIRK23 method not able to produce an approximation due to excessive memory requirements.

We then simulate the system up to $T = 5$ with 100 time steps with ETD2RKds (the overall computational time is about 1 second). As already observed in Reference [7], the solution approaches the equilibrium state $(u_e, v_e) = (a_2^u, a_1^u/a_2^u)$, see Fig. 6b.

5. Conclusions

In this paper, we show how it is possible to effectively exploit the Kronecker sum structure for the time integration of semidiscretized two-component systems of coupled advection–diffusion–reaction equations. The proposed second order exponential-type time marching scheme, which is based on a directional splitting of the involved matrix functions and named ETD2RKds, is shown to outperform well-established techniques on a variety of physically relevant models from the literature, such as two-dimensional Schnakenberg, FitzHugh–Nagumo, DIB, and three-dimensional FitzHugh–Nagumo, advective Schnakenberg, and advective Brusselator models. The procedure is able to capture the formation of Turing patterns, easily extends to models with any number of components, and can be applied in more than three spatial dimensions. As future work, we plan to investigate and perform simulations at HPC level (i.e., using server multi-core CPUs and GPUs) to further enhance the performances of the directional splitting procedure.

CRediT authorship contribution statement

Marco Caliari: Conceptualization, Formal analysis, Investigation, Methodology, Project administration, Software, Supervision, Validation, Writing – original draft, Writing – review & editing. **Fabio Cassini:** Conceptualization, Formal analysis, Investigation, Methodology, Software, Validation, Visualization, Writing – original draft, Writing – review & editing.

Declaration of competing interest

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

Data availability

We have made available the code in a public GitHub repository.

References

[1] A.H. Al-Mohy, N.J. Higham, A new scaling and squaring algorithm for the matrix exponential, SIAM J. Matrix Anal. Appl. 31 (2010) 970–989.
[2] A. Alla, A. Monti, I. Sgura, Adaptive POD-DEIM correction for Turing pattern approximation in reaction–diffusion PDE systems, J. Numer. Math. 31 (2023) 205–229.
[3] J.M. Alonso, J. Ibáñez, E. Defez, P. Alonso-Jordá, Euler polynomials for the matrix exponential approximation, J. Comput. Appl. Math. 425 (2023) 115074.
[4] E.O. Asante-Asamani, A. Kleefeld, B.A. Wade, A second-order exponential time differencing scheme for non-linear reaction-diffusion systems with dimensional splitting, J. Comput. Phys. 415 (2020) 109490.
[5] S. Ben Tahar, J.J. Muñoz, S.J. Shefelbine, E. Comellas, Turing pattern prediction in three-dimensional domains: the role of initial conditions and growth, bioRxiv (2023), 2023.03.29.534782.
[6] H. Berland, B. Skaflestad, W.M. Wright, EXPINT — a MATLAB package for exponential integrators, Technical Report 4, Norwegian University of Science and Technology, 2005.
[7] H.P. Bhatt, A.Q.M. Khaliq, B.A. Wade, Efficient Krylov-based exponential time differencing method in application to 3D advection-diffusion-reaction systems, Appl. Math. Comput. 338 (2018) 260–273.
[8] P. Bogacki, L.F. Shampine, A 3(2) pair of Runge – Kutta formulas, Appl. Math. Lett. 2 (1989) 321–325.
[9] B. Bozzini, D. Lacitignola, I. Sgura, Spatio-temporal organization in alloy electrodeposition: a morphochemical mathematical model and its experimental validation, J. Solid State Electrochem. 17 (2013) 467–479.
[10] M. Caliari, F. Cassini, Direction splitting of φ-functions in exponential integrators for d-dimensional problems in Kronecker form, J. Approx. Softw. (2023), in press.
[11] M. Caliari, F. Cassini, L. Einkemmer, A. Ostermann, F. Zivcovich, A μ-mode integrator for solving evolution equations in Kronecker form, J. Comput. Phys. 455 (2022) 110989.
[12] M. Caliari, F. Cassini, F. Zivcovich, BAMPHI: matrix-free and transpose-free action of linear combinations of φ-functions from exponential integrators, J. Comput. Appl. Math. 423 (2023) 114973.
[13] M. Caliari, F. Cassini, F. Zivcovich, A μ-mode approach for exponential integrators: actions of φ-functions of Kronecker sums, arXiv:2210.07667 [math.NA], 2023.
[14] M. Caliari, F. Cassini, F. Zivcovich, A μ-mode BLAS approach for multidimensional tensor-structured problems, Numer. Algorithms 92 (2023) 2483–2508.
[15] M. Caliari, F. Zivcovich, On-the-fly backward error estimate for matrix exponential approximation by Taylor algorithm, J. Comput. Appl. Math. 346 (2019) 532–548.
[16] M. Croci, J. Muñoz-Matute, Exploiting Kronecker structure in exponential integrators: fast approximation of the action of φ-functions of matrices via quadrature, J. Comput. Sci. 67 (2023) 101966.
[17] M.C. D'Autilia, I. Sgura, V. Simoncini, Matrix-oriented discretization methods for reaction–diffusion PDEs: comparisons and applications, Comput. Math. Appl. 79 (2020) 2067–2085.
[18] G. Gambino, M.C. Lombardo, G. Rubino, M. Sammartino, Pattern selection in the 2D FitzHugh–Nagumo model, Ric. Mat. 68 (2019) 535–549.
[19] S. Gaudreault, G. Rainwater, M. Tokman, KIOPS: a fast adaptive Krylov subspace solver for exponential integrators, J. Comput. Phys. 372 (2018) 236–255.
[20] E. Hairer, G. Wanner, Solving Ordinary Differential Equations II: Stiff and Differential-Algebraic Problems, second ed., Springer Series in Computational Mathematics, vol. 14, Springer Berlin, Heidelberg, 1996.
[21] M. Hochbruck, J. Leibold, A. Ostermann, On the convergence of Lawson methods for semilinear stiff problems, Numer. Math. 145 (2020) 553–580.
[22] M. Hochbruck, A. Ostermann, Exponential integrators, Acta Numer. 19 (2010) 209–286.
[23] M.E. Hosea, L.F. Shampine, Analysis and implementation of TR-BDF2, Appl. Numer. Math. 20 (1996) 21–37.
[24] T. Jiang, Y.-T. Zhang, Krylov single-step implicit integration factor WENO methods for advection–diffusion–reaction equations, J. Comput. Phys. 311 (2016) 22–44.
[25] J.D. Lawson, Generalized Runge-Kutta processes for stable systems with large Lipschitz constants, SIAM J. Numer. Anal. 4 (1967) 372–380.
[26] D. Li, S. Yang, J. Lan, Efficient and accurate computation for the φ-functions arising from exponential integrators, Calcolo 59 (2022) 1–24.
[27] V.T. Luan, J.A. Pudykiewicz, D.R. Reynolds, Further development of efficient and accurate time integration schemes for meteorological models, J. Comput. Phys. 376 (2019) 817–837.
[28] A. Madzvamuse, A.J. Wathen, P.K. Maini, A moving grid finite element method applied to a model biological pattern generator, J. Comput. Phys. 190 (2003) 478–500.
[29] H. Malchow, S.V. Petrovskii, E. Venturino, Spatiotemporal Patterns in Ecology and Epidemiology: Theory, Models, and Simulation, first ed., CRC Mathematical Biology Series, Chapman & Hall/CRC, Boca Raton, 2008.
[30] B. Müller, Investigation of Exponential Time Differencing schemes for advection-diffusion-reaction problems in the presence of significant advection, Master's thesis, University of Louisiana at Lafayette, 2022.
[31] J. Muñoz-Matute, D. Pardo, V.M. Calo, Exploiting the Kronecker product structure of φ–functions in exponential integrators, Int. J. Numer. Methods Eng. 123 (2022) 2142–2161.
[32] H. Neudecker, A note on Kronecker matrix products and matrix equation systems, SIAM J. Appl. Math. 17 (1969) 603–606.
[33] A.J. Perumpanani, J.A. Sherratt, P.K. Maini, Phase differences in reaction–diffusion–advection systems and applications to morphogenesis, IMA J. Appl. Math. 55 (1995) 19–33.
[34] J. Sastre, J. Ibáñez, E. Defez, Boosting the computation of the matrix exponential, Appl. Math. Comput. 340 (2019) 206–220.
[35] J. Schnakenberg, Simple chemical reaction systems with limit cycle behaviour, J. Theor. Biol. 81 (1979) 389–400.
[36] J.A. Sherratt, M.A.J. Chaplain, A new mathematical model for avascular tumour growth, J. Math. Biol. 43 (2001) 291–312.
[37] S. Singh, R.C. Mittal, S.R. Thottoli, M. Tamsir, High-fidelity simulations for Turing pattern formation in multi-dimensional Gray–Scott reaction-diffusion system, Appl. Math. Comput. 452 (2023) 128079.
[38] B. Skaflestad, W.M. Wright, The scaling and modified squaring method for matrix functions related to the exponential, Appl. Numer. Math. 59 (2009) 783–799.
[39] E.H. Twizell, A.B. Gumel, Q. Cao, A second-order scheme for the "Brusselator" reaction–diffusion system, J. Math. Chem. 26 (1999) 297–316.