Electronic Transactions on Numerical Analysis.
Volume 60, pp. 520–540, 2024.

ISSN 1068–9613.
DOI: 10.1553/etna_vol60s520

ETNA
Kent State University and
Johann Radon Institute (RICAM)

EFFICIENT THIRD-ORDER TENSOR-ORIENTED DIRECTIONAL SPLITTING FOR EXPONENTIAL INTEGRATORS*

FABIO CASSINI[§‡]

Abstract. Suitable discretizations of popular multidimensional operators (for instance of diffusion or diffusion-advection type) by tensor product formulas lead to matrices with d-dimensional Kronecker sum structure. For evolutionary partial differential equations containing such operators and when integrating in time with exponential integrators, it is then of paramount importance to efficiently approximate the actions of φ-functions of the arising matrices. In this work we show how to produce directional split approximations of third order with respect to the time step size. These approximations conveniently employ tensor-matrix products (the so-called μ-mode product and the related Tucker operator, realized in practice with high-performance level 3 BLAS operations) and allow for the effective usage of exponential Runge–Kutta integrators up to order three. The technique can also be efficiently implemented on modern computer hardware such as Graphic Processing Units. This approach is successfully tested against state-of-the-art techniques on two well-known physical models that lead to Turing patterns, namely the 2D Schnakenberg and the 3D FitzHugh–Nagumo systems, on different hardware and software architectures.

Key words. exponential integrators, μ-mode product, directional splitting, φ-functions, Kronecker sum, Turing patterns, Graphic Processing Units

AMS subject classifications. 65F60, 65L04, 65L05, 65M20

1. Introduction.

We are interested in the solution of stiff systems of ordinary differential equations (ODEs) of the type

$$\boldsymbol{u}'(t) = K\boldsymbol{u}(t) + \boldsymbol{g}(t, \boldsymbol{u}(t)) = \boldsymbol{f}(t, \boldsymbol{u}(t)), \qquad \boldsymbol{u}(0) = \boldsymbol{u}_0, \tag{1.1a}$$

using exponential integrators [22]. The stiff part is represented by the matrix $K \in \mathbb{C}^{N\times N}$, which we assume to have d-dimensional *Kronecker sum structure*

$$K = A_d \oplus \cdots \oplus A_1 = A_{\otimes 1} + \cdots + A_{\otimes d}, \tag{1.1b}$$

with

$$A_{\otimes\mu} = I_d \otimes \cdots \otimes I_{\mu+1} \otimes A_\mu \otimes I_{\mu-1} \otimes \cdots \otimes I_1, \qquad \mu = 1, \dots, d. \tag{1.1c}$$

Here, $A_\mu \in \mathbb{C}^{n_\mu \times n_\mu}$, and I_μ is the identity matrix of size n_μ. Moreover, $\boldsymbol{g}(t, \boldsymbol{u}(t))$ is a nonlinear function of t and of the unknown $\boldsymbol{u}(t) \in \mathbb{C}^N$, with $N = n_1 \cdots n_d$. Throughout the paper, the symbol $\otimes$ denotes the Kronecker product of matrices, while $\oplus$ is employed for the Kronecker sum of matrices. Systems of the form (1.1) naturally arise when discretizing in space some partial differential equations (PDEs) defined on tensor product domains and with appropriate boundary conditions. In those cases, A_μ are matrices that correspond to differential or fractional one-dimensional operators after a discretization with (nonuniform) finite differences [8, 9] or tensor product finite or spectral elements [17, 33], and they usually encapsulate boundary conditions. Typical examples are (systems of) evolutionary PDEs that contain diffusion-advection-absorption or Schrödinger operators, amongst others.

In the last years, exponential integrators proved to be a valuable alternative for the efficient integration of systems with Kronecker sum structure (1.1) [5, 7, 8, 9, 12, 13, 17, 18, 23, 25, 29,

*Received January 19, 2024. Accepted September 11, 2024. Published online on October 3, 2024. Recommended by Stéphane Gaudreault.
§Department of Computer Science, University of Verona. Strada le Grazie, 15, Verona, 37134, Italy (fabio.cassini@univr.it).
‡Istituto Nazionale di Alta Matematica "Francesco Severi". Piazzale Aldo Moro, 5, Roma, 00185, Italy (cassini@altamatematica.it).

ETNA
Kent State University and
Johann Radon Institute (RICAM)

30, 33, 34, 40]. In particular, it is shown in [7, 8] that the well-known exponential Runge–Kutta method of order two, ETD2RK,

$$\begin{aligned} \boldsymbol{u}_{n2} &= \boldsymbol{u}_n + \tau\varphi_1(\tau K)\boldsymbol{f}(t_n, \boldsymbol{u}_n), \\ \boldsymbol{u}_{n+1} &= \boldsymbol{u}_{n2} + \tau\varphi_2(\tau K)(\boldsymbol{g}(t_{n+1}, \boldsymbol{u}_{n2}) - \boldsymbol{g}(t_n, \boldsymbol{u}_n)), \end{aligned} \tag{1.2}$$

where τ denotes the time step size, can be directionally split, obtaining the so-called ETD2RKDS integrator. There, it is also shown that the latter strongly outperforms other well-established methods in the solution of stiff diffusion-advection-reaction models. The φ-functions appearing in exponential integrators are exponential-like matrix functions defined for a generic matrix $X \in \mathbb{C}^{N\times N}$ as

$$\varphi_0(X) = \exp(X), \quad \varphi_\ell(X) = \frac{1}{(\ell-1)!}\int_0^1 \exp((1-\theta)X)\theta^{\ell-1}d\theta, \qquad \ell > 0.$$

They can also be expressed in terms of the Taylor expansion

$$\varphi_j(X) = \sum_{k=0}^{\infty} \frac{X^k}{(k+j)!}, \qquad j \geq 0.$$

Concerning their computation and action to a vector, we refer to [1, 4, 15, 28, 37, 38] for algorithms suitable for small-sized matrices, to [2, 10, 11, 14, 20, 31, 35] for large and sparse matrices, and to [6, 9, 12, 13, 17, 29, 30, 33] when X is a Kronecker sum. The key points to develop the directional split integrator based on method (1.2) were the observation that

$$\begin{aligned} \varphi_1(\tau K) &= \varphi_1(\tau A_d) \otimes \cdots \otimes \varphi_1(\tau A_1) + \mathcal{O}(\tau^2), \\ \varphi_2(\tau K) &= 2^{d-1}\varphi_2(\tau A_d) \otimes \cdots \otimes \varphi_2(\tau A_1) + \mathcal{O}(\tau^2), \end{aligned} \tag{1.3}$$

and that the actions of the right-hand sides can be efficiently realized in tensor form with Tucker operators (see the next section and [9, 12] for insights on the relevant tensor-matrix operations). However, the second-order accuracy of the formulas with respect to the time step size essentially limits their applicability to schemes of at most order two.

In this paper we aim at introducing directional split approximations of third order with respect to τ for φ-functions of matrices with d-dimensional Kronecker sum structure. The formulas derived in Section 2 allow for the efficient construction and employment of, for instance, exponential Runge–Kutta integrators of order three for ODEs systems with Kronecker sum structure. In particular, since the realization of the approximations heavily exploits BLAS operations, the resulting schemes can also be efficiently implemented on modern hardware architectures such as multi-core Central Processing Units (CPUs) and Graphic Processing Units (GPUs). The effectiveness of the approach and the performance results for two popular systems of diffusion-reaction equations are shown in the numerical examples of Section 3. We finally draw conclusions in Section 4.

2. Third-order directional splitting. We focus here on three-stage exponential Runge–Kutta integrators of the form

$$\begin{aligned} \boldsymbol{u}_{n2} &= \boldsymbol{u}_n + c_2\tau\varphi_1(c_2\tau K)\boldsymbol{f}(t_n, \boldsymbol{u}_n), \\ \boldsymbol{u}_{n3} &= \boldsymbol{u}_n + c_3\tau\varphi_1(c_3\tau K)\boldsymbol{f}(t_n, \boldsymbol{u}_n) + \tau a_{32}(\tau K)\boldsymbol{d}_{n2}, \\ \boldsymbol{u}_{n+1} &= \boldsymbol{u}_n + \tau\varphi_1(\tau K)\boldsymbol{f}(t_n, \boldsymbol{u}_n) + \tau(b_2(\tau K)\boldsymbol{d}_{n2} + b_3(\tau K)\boldsymbol{d}_{n3}), \end{aligned} \tag{2.1a}$$

where

$$\boldsymbol{d}_{ni} = \boldsymbol{g}(t_n + c_i\tau, \boldsymbol{u}_{ni}) - \boldsymbol{g}(t_n, \boldsymbol{u}_n), \qquad i = 2, 3, \tag{2.1b}$$

ETNA
Kent State University and
Johann Radon Institute (RICAM)

c_2 and c_3 are scalars, while $a_{32}(\cdot)$, $b_2(\cdot)$, and $b_3(\cdot)$ are (linear combinations of) φ-functions. From [21, Section 5.2] we know that we can construct integrators of order three in which the coefficients $a_{32}(\cdot)$, $b_2(\cdot)$, and $b_3(\cdot)$ involve just the functions φ_1 and φ_2. Therefore, we will fully develop our approximation techniques using these two functions only. The generalization to higher-order φ-functions is straightforward and is briefly discussed in Remark 2.4.

The starting point for the realization of efficient directional split exponential integrators is the following formula for the matrix exponential of the Kronecker sum K [9, 12]:

$$\begin{aligned} \exp(\tau K)\boldsymbol{v} &= \exp(\tau A_{\otimes 1}) \cdots \exp(\tau A_{\otimes d})\boldsymbol{v} \\ &= (\exp(\tau A_d) \otimes \cdots \otimes \exp(\tau A_1))\boldsymbol{v} \\ &= \operatorname{vec}(\boldsymbol{V} \times_1 \exp(\tau A_1) \times_2 \cdots \times_d \exp(\tau A_d)). \end{aligned} \tag{2.2}$$

Here $\boldsymbol{V} \in \mathbb{C}^{n_1 \times \cdots \times n_d}$ is an order-d tensor such that $\operatorname{vec}(\boldsymbol{V}) = \boldsymbol{v}$, with vec representing the operator that stacks the input tensor by columns, and $\times_\mu$ denotes the μ-mode product, i.e., a tensor-matrix product along the direction μ. The concatenation of μ-mode products is referred to as Tucker operator. Since these are core concepts in the manuscript, we briefly describe them in the following. A thorough explanation with full details can be found, for instance, in [12, 26, 27]. Given a generic order-d tensor $\boldsymbol{T} \in \mathbb{C}^{n_1 \times \cdots \times n_d}$ (with elements denoted as $t_{i_1 \dots i_d}$) and a matrix $L_\mu \in \mathbb{C}^{n_\mu \times n_\mu}$ with elements ℓ^μ_{ij}, the *μ-mode product* of $\boldsymbol{T}$ with L_μ (denoted as $\boldsymbol{T} \times_\mu L_\mu$) is the tensor $\boldsymbol{S} \in \mathbb{C}^{n_1 \times \cdots \times n_d}$ defined elementwise as

$$s_{i_1 \dots i_d} = \sum_{j_\mu = 1}^{n_\mu} t_{i_1 \dots i_{\mu-1} j_\mu i_{\mu+1} \dots i_d} \ell^\mu_{i_\mu j_\mu}.$$

This corresponds to multiplying the matrix L_μ onto the *μ-fibers* of the tensor $\boldsymbol{T}$, i.e., the vectors along direction μ that are generalizations of columns and rows of a matrix to the tensor case. The concatenation of μ-mode products with the matrices $L_1, \dots, L_d$, that is, the tensor $\boldsymbol{S}$ with elements

$$s_{i_1 \dots i_d} = \sum_{j_d = 1}^{n_d} \cdots \sum_{j_1 = 1}^{n_1} t_{j_1 \dots j_d} \prod_{\mu = 1}^{d} \ell^\mu_{i_\mu j_\mu},$$

is denoted by $\boldsymbol{T} \times_1 L_1 \times_2 \cdots \times_d L_d$ and is called *Tucker operator*. In terms of computational cost, a single μ-mode product requires $\mathcal{O}(N n_\mu)$ floating-point operations (with $N = n_1 \cdots n_d$), and it can be implemented by a single (full) matrix-matrix product. In practice, this can be realized with a single GEMM (GEneral Matrix Multiply) call of level 3 BLAS (Basic Linear Algebra Subprograms), whose highly optimized implementations are available essentially for any kind of modern computer architecture (we mention, for instance, [24, 36, 39]). Consequently, the Tucker operator has an overall computational cost of $\mathcal{O}(N(n_1 + \cdots + n_d))$, and it can be realized with d GEMM calls of level 3 BLAS. Finally, the connection between the Kronecker product and the Tucker operator is given by the formula (see [12, Lemma 2.1])

$$(L_d \otimes \cdots \otimes L_1)\boldsymbol{t} = \operatorname{vec}(\boldsymbol{T} \times_1 L_1 \times_2 \cdots \times_d L_d), \qquad \boldsymbol{t} = \operatorname{vec}(\boldsymbol{T}). \tag{2.3}$$

Let us now return to the action of the matrix exponential. First of all, notice that in the case $d = 2$, formula (2.2) simply reduces to

$$\exp(\tau(A_{\otimes 1} + A_{\otimes 2}))\boldsymbol{v} = \operatorname{vec}\left(\exp(\tau A_1)\boldsymbol{V} \exp(\tau A_2)^{\mathsf{T}}\right), \tag{2.4}$$

ETNA
Kent State University and
Johann Radon Institute (RICAM)

since for order-2 tensors (i.e., matrices) 1- and 2-mode products are just the standard matrix-matrix and matrix-matrix-transpose multiplications, respectively. Hence, after computing the small-sized matrix exponential functions, formula (2.4) can be efficiently realized with two GEMM calls. In general, the advantage of the tensor approach in formula (2.2) is that it allows to compute the action $\exp(\tau K)\boldsymbol{v}$ through a single Tucker operator without assembling the matrix itself or computing Kronecker products. In fact, we just rely on high-performance BLAS after the computation of the small-sized matrix exponentials $\exp(\tau A_\mu)$. We remark that the main cost of the procedure lies in the computation of the Tucker operator, while computing the needed matrix exponentials is of negligible burden [9]. Concerning the functions φ_ℓ, with $\ell > 0$, which are needed for the exponential integrators considered in this work, we notice that the first equality in formula (2.2) is not valid anymore. However, in [7] it is shown that

$$\begin{aligned}\varphi_\ell(\tau K)\boldsymbol{v} &= \ell!^{d-1}\varphi_\ell(\tau A_{\otimes 1})\cdots\varphi_\ell(\tau A_{\otimes d})\boldsymbol{v} + \mathcal{O}(\tau^2)\\ &= \ell!^{d-1}\left(\varphi_\ell(\tau A_d)\otimes\cdots\otimes\varphi_\ell(\tau A_1)\right)\boldsymbol{v} + \mathcal{O}(\tau^2)\\ &= \operatorname{vec}\left(\ell!^{d-1}\boldsymbol{V}\times_1\varphi_\ell(\tau A_1)\times_2\cdots\times_d\varphi_\ell(\tau A_d)\right) + \mathcal{O}(\tau^2),\end{aligned} \tag{2.5}$$

of which the expressions (1.3) are particular cases. When $d = 2$, the formula simply reduces to

$$\varphi_\ell(\tau(A_{\otimes 1} + A_{\otimes 2}))\boldsymbol{v} = \operatorname{vec}\left(\varphi_\ell(\tau A_1)(\ell!\boldsymbol{V})\varphi_\ell(\tau A_2)^{\mathsf{T}}\right) + \mathcal{O}(\tau^2), \tag{2.6}$$

which can again be realized with two GEMM calls after computing the small-sized φ-functions. Similarly to the action of the matrix exponential, the generic d-dimensional formulation (2.5) requires one Tucker operator and can be implemented with d GEMM calls as floating-point operations. Clearly, the approximation (2.5) cannot be directly employed in the formulation (2.1) since it would lead to an order reduction of the resulting integrator. In the following sections we will then look for third-order directional split approximations of the φ-functions *similar* to formula (2.5), in the sense that their realization will require only few Tucker operators (which, we emphasize again, constitute the major computational cost of computing the approximation).

2.1. Two-term two-dimensional splitting. To increase the approximation order of formula (2.6), we look for a combination of the form

$$\begin{aligned}\varphi_\ell(\tau(A_{\otimes 1} + A_{\otimes 2})) = {}& \eta_{1,2}\varphi_{\ell_1}(\alpha_{1,2}\tau A_2)\otimes\varphi_{\ell_1}(\alpha_{1,1}\tau A_1)\\ &+ \eta_{2,2}\varphi_{\ell_2}(\alpha_{2,2}\tau A_2)\otimes\varphi_{\ell_2}(\alpha_{2,1}\tau A_1) + \mathcal{O}(\tau^3),\end{aligned} \tag{2.7}$$

where $\ell_i > 0$ and $\eta_{i,2}, \alpha_{i,\mu} \in \mathbb{C}$ (with $i = 1, 2$ and $\mu = 1, 2$) are parameters to be determined. By a Taylor expansion we directly obtain the conditions

$$\frac{\eta_{1,2}}{\ell_1!^2} + \frac{\eta_{2,2}}{\ell_2!^2} = \frac{1}{\ell!}, \tag{2.8a}$$

$$\frac{\eta_{1,2}\alpha_{1,1}}{\ell_1!(\ell_1+1)!} + \frac{\eta_{2,2}\alpha_{2,1}}{\ell_2!(\ell_2+1)!} = \frac{1}{(\ell+1)!},\quad \frac{\eta_{1,2}\alpha_{1,2}}{\ell_1!(\ell_1+1)!} + \frac{\eta_{2,2}\alpha_{2,2}}{\ell_2!(\ell_2+1)!} = \frac{1}{(\ell+1)!}, \tag{2.8b}$$

$$\frac{\eta_{1,2}\alpha_{1,1}^2}{\ell_1!(\ell_1+2)!} + \frac{\eta_{2,2}\alpha_{2,1}^2}{\ell_2!(\ell_2+2)!} = \frac{1}{(\ell+2)!},\quad \frac{\eta_{1,2}\alpha_{1,2}^2}{\ell_1!(\ell_1+2)!} + \frac{\eta_{2,2}\alpha_{2,2}^2}{\ell_2!(\ell_2+2)!} = \frac{1}{(\ell+2)!}, \tag{2.8c}$$

$$\frac{\eta_{1,2}\alpha_{1,1}\alpha_{1,2}}{(\ell_1+1)!^2} + \frac{\eta_{2,2}\alpha_{2,1}\alpha_{2,2}}{(\ell_2+1)!^2} = \frac{2}{(\ell+2)!}. \tag{2.8d}$$

Thus, we have the following result:

ETNA
Kent State University and
Johann Radon Institute (RICAM)

THEOREM 2.1. *The coefficients in Table 2.1 are the solutions of the system (2.8) for $\ell = 1, 2$, with $\ell_1 = 1$ and $\ell_2 = 2$.*

TABLE 2.1
Coefficients for the two-term two-dimensional splitting (2.7) with $\ell_1 = 1$ and $\ell_2 = 2$.

	$\ell = 1$		$\ell = 2$	
$\eta_{1,2}$	$-\frac{5}{4}$	$\frac{7}{4} \pm \frac{3\sqrt{2}}{2}\mathrm{i}$	$-\frac{4}{3}$	$\frac{2}{3} \pm \frac{2\sqrt{3}}{3}\mathrm{i}$
$\alpha_{1,1}$	$\pm\frac{4\sqrt{10}}{15} + \frac{4}{3}$	$\frac{12}{11} \mp \frac{4\sqrt{2}}{11}\mathrm{i}$	$\pm\frac{\sqrt{33}}{8} + \frac{9}{8}$	$\frac{3}{4} \mp \frac{\sqrt{3}}{4}\mathrm{i}$
$\alpha_{1,2}$	$\mp\frac{4\sqrt{10}}{15} + \frac{4}{3}$	$\frac{12}{11} \mp \frac{4\sqrt{2}}{11}\mathrm{i}$	$\mp\frac{\sqrt{33}}{8} + \frac{9}{8}$	$\frac{3}{4} \mp \frac{\sqrt{3}}{4}\mathrm{i}$
$\eta_{2,2}$	9	$-3 \mp 6\sqrt{2}\mathrm{i}$	$\frac{22}{3}$	$-\frac{2}{3} \mp \frac{8\sqrt{3}}{3}\mathrm{i}$
$\alpha_{2,1}$	$\pm\frac{2\sqrt{10}}{9} + \frac{16}{9}$	$\frac{4}{3} \mp \frac{2\sqrt{2}}{3}\mathrm{i}$	$\pm\frac{3\sqrt{33}}{22} + \frac{3}{2}$	$\frac{6}{7} \mp \frac{3\sqrt{3}}{7}\mathrm{i}$
$\alpha_{2,2}$	$\mp\frac{2\sqrt{10}}{9} + \frac{16}{9}$	$\frac{4}{3} \mp \frac{2\sqrt{2}}{3}\mathrm{i}$	$\mp\frac{3\sqrt{33}}{22} + \frac{3}{2}$	$\frac{6}{7} \mp \frac{3\sqrt{3}}{7}\mathrm{i}$

Proof. By writing $\eta_{2,2}$ in terms of $\eta_{1,2}$ from equation (2.8a) and $\alpha_{2,1}$ and $\alpha_{2,2}$ in terms of $\alpha_{1,1}$ and $\alpha_{1,2}$, respectively, from equations (2.8b), we get two quadratic equations in $\alpha_{1,1}$ and $\alpha_{1,2}$ from (2.8c). Their solution (dependent on $\eta_{1,2}$) gives four possible pairs. If we now substitute each pair into equation (2.8d), then we get one quartic equation in $\eta_{1,2}$ with no admissible solution, one quartic equation in $\eta_{1,2}$ with two complex conjugate solutions, and two linear equations in $\eta_{1,2}$, each of which has one real solution. Substituting back gives the desired coefficients. ∎

Notice that formula (2.7) allows for a third-order approximation of the φ-functions, which, thanks to equivalence (2.3), requires the computation of two Tucker operators in tensor form.

2.2. Two-term d-dimensional splitting with complex coefficients. In dimension d we consider an approximation of the form

$$\begin{aligned}\varphi_\ell(\tau(A_{\otimes 1} + \cdots + A_{\otimes d})) = \eta_{1,d}\varphi_{\ell_1}(\alpha_{1,d}\tau A_d) \otimes \cdots \otimes \varphi_{\ell_1}(\alpha_{1,1}\tau A_1) \\ + \eta_{2,d}\varphi_{\ell_2}(\alpha_{2,d}\tau A_d) \otimes \cdots \otimes \varphi_{\ell_2}(\alpha_{2,1}\tau A_1) + \mathcal{O}(\tau^3),\end{aligned} \tag{2.9}$$

which again can be realized in tensor form with two Tucker operators. Similarly to the previous case, the coefficients have to satisfy the nonlinear system

$$\frac{\eta_{1,d}}{\ell_1!^d} + \frac{\eta_{2,d}}{\ell_2!^d} = \frac{1}{\ell!}, \tag{2.10a}$$

$$\frac{\eta_{1,d}\alpha_{1,\mu}}{\ell_1!^{d-1}(\ell_1+1)!} + \frac{\eta_{2,d}\alpha_{2,\mu}}{\ell_2!^{d-1}(\ell_2+1)!} = \frac{1}{(\ell+1)!}, \qquad \mu = 1, \ldots, d, \tag{2.10b}$$

$$\frac{\eta_{1,d}\alpha_{1,\mu}^2}{\ell_1!^{d-1}(\ell_1+2)!} + \frac{\eta_{2,d}\alpha_{2,\mu}^2}{\ell_2!^{d-1}(\ell_2+2)!} = \frac{1}{(\ell+2)!}, \qquad \mu = 1, \ldots, d, \tag{2.10c}$$

$$\frac{\eta_{1,d}\alpha_{1,\mu}\alpha_{1,\nu}}{\ell_1!^{d-2}(\ell_1+1)!^2} + \frac{\eta_{2,d}\alpha_{2,\mu}\alpha_{2,\nu}}{\ell_2!^{d-2}(\ell_2+1)!^2} = \frac{2}{(\ell+2)!}, \qquad \mu, \nu = 1, \ldots, d,\ \mu < \nu. \tag{2.10d}$$

THEOREM 2.2. *For $\ell = 1, 2$, $d > 2$, $\ell_1 = 1$, and $\ell_2 = 2$, there exists no real solution to the system (2.10). On the other hand, the solutions of the system (2.10) are given by the complex coefficients in Table 2.2.*

ETNA
Kent State University and
Johann Radon Institute (RICAM)

TABLE 2.2
Complex coefficients for the two-term d-dimensional splitting (2.9) with $l_1 = 1$ and $l_2 = 2$.

	$\ell = 1$	$\ell = 2$
$\eta_{1,d}$	$\frac{7}{4} \pm \frac{3\sqrt{2}}{2}\mathrm{i}$	$\frac{2}{3} \pm \frac{2\sqrt{3}}{3}\mathrm{i}$
$\alpha_{1,\mu}$	$\frac{12}{11} \mp \frac{4\sqrt{2}}{11}\mathrm{i}$	$\frac{3}{4} \mp \frac{\sqrt{3}}{4}\mathrm{i}$
$\eta_{2,d}$	$2^{d-2}\left(-3 \mp 6\sqrt{2}\mathrm{i}\right)$	$2^{d-2}\left(-\frac{2}{3} \mp \frac{8\sqrt{3}}{3}\mathrm{i}\right)$
$\alpha_{2,\mu}$	$\frac{4}{3} \mp \frac{2\sqrt{2}}{3}\mathrm{i}$	$\frac{6}{7} \mp \frac{3\sqrt{3}}{7}\mathrm{i}$

Proof. Since $\ell_1 = 1$ and $\ell_2 = 2$, we have from equation (2.10a) that $\eta_{2,d} = 2^d\left(\frac{1}{\ell!} - \eta_{1,d}\right)$. Substituting in the expressions (2.10b)–(2.10d), we obtain a set of equations that does not depend on d anymore. Let us then consider, among them, the equations for $\mu = \mu_1$ and $\mu = \mu_2$, with $\mu_1 \neq \mu_2$. Their solutions correspond to the ones obtained for the two-dimensional case (see Table 2.1 for a summary), i.e., $\eta_{1,d} = \eta_{1,2}$, $\eta_{2,d} = 2^{d-2}\eta_{2,2}$, $\alpha_{1,\mu_1} = \alpha_{1,1}$, $\alpha_{1,\mu_2} = \alpha_{1,2}$, $\alpha_{2,\mu_1} = \alpha_{2,1}$, and $\alpha_{2,\mu_2} = \alpha_{2,2}$. This has to be valid for each $\mu_1 \neq \mu_2$, which implies $\alpha_{1,\mu} = \alpha_{1,1}$ and $\alpha_{2,\mu} = \alpha_{2,1}$, for $\mu = 1, \ldots, d$, and hence excludes the real solutions. ∎

Notice that in case $d = 2$ the coefficients in Table 2.2 reduce to the complex coefficients in Table 2.1.

2.3. Three-term *d*-dimensional splitting with real coefficients. The approximation in the d-dimensional case derived in the previous section required, in tensor form, two *complex* Tucker operators to be computed. Here, we present an alternative formula for $d > 2$ which works entirely in *real* arithmetic but needs three Tucker operators. We write our ansatz as

$$\begin{aligned}\varphi_\ell(\tau(A_{\otimes 1} + \cdots + A_{\otimes d})) = {}& \eta_{1,d}\varphi_{\ell_1}(\alpha_{1,d}\tau A_d) \otimes \cdots \otimes \varphi_{\ell_1}(\alpha_{1,1}\tau A_1) \\ &+ \eta_{2,d}\varphi_{\ell_2}(\alpha_{2,d}\tau A_d) \otimes \cdots \otimes \varphi_{\ell_2}(\alpha_{2,1}\tau A_1) \\ &+ \eta_{3,d}\varphi_{\ell_3}(\alpha_{3,d}\tau A_d) \otimes \cdots \otimes \varphi_{\ell_3}(\alpha_{3,1}\tau A_1) + \mathcal{O}(\tau^3)\end{aligned} \tag{2.11}$$

and look for the coefficients which satisfy the nonlinear system

$$\frac{\eta_{1,d}}{\ell_1!^d} + \frac{\eta_{2,d}}{\ell_2!^d} + \frac{\eta_{3,d}}{\ell_3!^d} = \frac{1}{\ell!}, \tag{2.12a}$$

$$\frac{\eta_{1,d}\alpha_{1,\mu}}{\ell_1!^{d-1}(\ell_1+1)!} + \frac{\eta_{2,d}\alpha_{2,\mu}}{\ell_2!^{d-1}(\ell_2+1)!} + \frac{\eta_{3,d}\alpha_{3,\mu}}{\ell_3!^{d-1}(\ell_3+1)!} = \frac{1}{(\ell+1)!}, \tag{2.12b}$$

$$\frac{\eta_{1,d}\alpha_{1,\mu}^2}{\ell_1!^{d-1}(\ell_1+2)!} + \frac{\eta_{2,d}\alpha_{2,\mu}^2}{\ell_2!^{d-1}(\ell_2+2)!} + \frac{\eta_{3,d}\alpha_{3,\mu}^2}{\ell_3!^{d-1}(\ell_3+2)!} = \frac{1}{(\ell+2)!}, \tag{2.12c}$$

$$\frac{\eta_{1,d}\alpha_{1,\mu}\alpha_{1,\nu}}{\ell_1!^{d-2}(\ell_1+1)!^2} + \frac{\eta_{2,d}\alpha_{2,\mu}\alpha_{2,\nu}}{\ell_2!^{d-2}(\ell_2+1)!^2} + \frac{\eta_{3,d}\alpha_{3,\mu}\alpha_{3,\nu}}{\ell_3!^{d-2}(\ell_3+1)!^2} = \frac{2}{(\ell+2)!}, \quad \mu < \nu. \tag{2.12d}$$

In this notation, the indices μ and ν run from 1 to d. Then, we have the following result:

THEOREM 2.3. *Let $\ell_1 = \ell_3 = 1$ and $\ell_2 = 2$. Consider the additional conditions*

$$\frac{\eta_{1,d}\alpha_{1,\mu}^3}{4!} + \frac{\eta_{2,d}\alpha_{2,\mu}^3}{2!^{d-1}5!} + \frac{\eta_{3,d}\alpha_{3,\mu}^3}{4!} = \frac{1}{(\ell+3)!}, \tag{2.12e}$$

$$\frac{\eta_{1,d}\alpha_{1,\mu}\alpha_{1,\nu}\alpha_{1,\xi}}{2!^3} + \frac{\eta_{2,d}\alpha_{2,\mu}\alpha_{2,\nu}\alpha_{2,\xi}}{2^{d-3}3!^3} + \frac{\eta_{3,d}\alpha_{3,\mu}\alpha_{3,\nu}\alpha_{3,\xi}}{2!^3} = \frac{6}{(\ell+3)!}, \quad \mu < \nu < \xi, \tag{2.12f}$$

which correspond to matching the third-order terms $A_{\otimes\mu}^3$ and $A_{\otimes\mu}A_{\otimes\nu}A_{\otimes\xi}$ in the Taylor expansion. Again, in this notation the indices μ, ν, and ξ run from 1 to d. Then, for $\ell = 1$ and $\ell = 2$, the coefficients in Table 2.3 are the only solutions of the system (2.12).

ETNA
Kent State University and
Johann Radon Institute (RICAM)

TABLE 2.3
Real coefficients for the three-term d-dimensional splitting (2.11) with $\ell_1 = \ell_3 = 1$ and $\ell_2 = 2$.

	$\ell = 1$	$\ell = 2$
$\eta_{1,d}$	$\frac{2243}{1350} \pm \frac{440521}{675\sqrt{2991111}}$	$\frac{19}{27} \pm \frac{151}{27\sqrt{2391}}$
$\alpha_{1,\mu}$	$\frac{3(5161 \pm \sqrt{2991111})}{15869}$	$\frac{3(121 \pm \sqrt{2391})}{490}$
$\eta_{2,d}$	$-\frac{12544}{675} \cdot 2^{d-3}$	$-\frac{196}{27} \cdot 2^{d-3}$
$\alpha_{2,\mu}$	$\frac{45}{28}$	$\frac{9}{7}$
$\eta_{3,d}$	$\frac{2243}{1350} \mp \frac{440521}{675\sqrt{2991111}}$	$\frac{19}{27} \mp \frac{151}{27\sqrt{2391}}$
$\alpha_{3,\mu}$	$\frac{3(5161 \mp \sqrt{2991111})}{15869}$	$\frac{3(121 \mp \sqrt{2391})}{490}$

Proof. We first consider the system (2.12) in the case $d = 3$. Then, to determine its solutions we use standard arguments of Gröbner basis theory [16]. We assume the lexicographic ordering

$$\alpha_{1,1} > \alpha_{1,2} > \alpha_{1,3} > \alpha_{2,1} > \alpha_{2,2} > \alpha_{2,3} > \alpha_{3,1} > \alpha_{3,2} > \alpha_{3,3} > \eta_1 > \eta_2 > \eta_3,$$

and we let $\mu = 1, 2, 3$. Then, for $\ell = 1$, a Gröbner basis of the ideal associated to the system is

$$\begin{aligned}\Big\{&570887639987 - 724578693084\eta_3 + 218051991900\eta_3^2, 12544 + 675\eta_2,\\ &-2243 + 675\eta_1 + 675\eta_3, -45 + 28a_{2,\mu},\\ &6486012633 + 13981255498\alpha_{3,\mu} - 12113999550\eta_3,\\ &-33768359205 + 13981255498\alpha_{1,\mu} + 12113999550\eta_3\Big\},\end{aligned}$$

while for $\ell = 2$ it is

$$\begin{aligned}\Big\{&840350 - 2453166\eta_3 + 1743039\eta_3^2, 196 + 27\eta_2, -38 + 27\eta_1 + 27\eta_3,\\ &81474 + 73990\alpha_{3,\mu} - 193671\eta_3, -9 + 7\alpha_{2,\mu}, -191100 + 73990\alpha_{1,\mu} + 193671\eta_3\Big\}.\end{aligned}$$

The desired solutions of the system (2.12) are hence equivalently given by the zeros of the polynomials in the Gröbner basis. Simple calculations lead to the coefficients summarized in Table 2.3. Finally, using arguments similar to the ones in the proof of Theorem 2.2, we get the result for the d-dimensional case. ∎

REMARK 2.4. As mentioned at the beginning of the section, we focused our attention to the functions φ_1 and φ_2 only since the class of integrators that we consider requires at most the latter. Clearly, using a similar approach, one could obtain formulas for different ℓ and ℓ_i.

2.4. Implementation details. The directional splitting approximations introduced above (i.e., the formulas (2.7), (2.9), and (2.11)) allow for an efficient tensor-oriented approximation of the actions of φ-functions needed in third-order exponential integrators of the form (2.1), thanks to the equivalence (2.3). For the numerical examples presented in Section 3, we will employ in particular the time marching scheme with coefficients $c_2 = \frac{1}{3}$, $c_3 = \frac{2}{3}$, $a_{32}(\cdot) = \frac{4}{3}\varphi_2(c_3\cdot)$, $b_2(\cdot) = 0$, and $b_3(\cdot) = \frac{3}{2}\varphi_2(\cdot)$ (see also tableau (5.8) in [21]). More

ETNA
Kent State University and
Johann Radon Institute (RICAM)

explicitly, we have

$$
\begin{aligned}
\boldsymbol{u}_{n2} &= \boldsymbol{u}_n + \frac{\tau}{3}\varphi_1\left(\frac{\tau}{3}K\right)\boldsymbol{f}(t_n,\boldsymbol{u}_n),\\
\boldsymbol{u}_{n3} &= \boldsymbol{u}_n + \frac{2\tau}{3}\varphi_1\left(\frac{2\tau}{3}K\right)\boldsymbol{f}(t_n,\boldsymbol{u}_n)\\
&\quad + \frac{4\tau}{3}\varphi_2\left(\frac{2\tau}{3}K\right)\left(\boldsymbol{g}\left(t_n+\frac{\tau}{3},\boldsymbol{u}_{n2}\right)-\boldsymbol{g}(t_n,\boldsymbol{u}_n)\right),\\
\boldsymbol{u}_{n+1} &= \boldsymbol{u}_n + \tau\varphi_1(\tau K)\boldsymbol{f}(t_n,\boldsymbol{u}_n)\\
&\quad + \frac{3\tau}{2}\varphi_2(\tau K)\left(\boldsymbol{g}\left(t_n+\frac{2\tau}{3},\boldsymbol{u}_{n3}\right)-\boldsymbol{g}(t_n,\boldsymbol{u}_n)\right).
\end{aligned}
\tag{2.13}
$$

We label this method EXPRK3 when no directional splitting approximations are used. In this case, in our experiments, the needed linear combinations of actions of φ-functions are computed using the very efficient incomplete orthogonalization Krylov-based technique described in [31].

When the directional splitting approximations are actually employed in the scheme (2.13), we label the resulting methods EXPRK3DS_REAL (if we use real coefficients) and EXPRK3DS_CPLX (if we use complex coefficients). In more detail, as set of directional splitting coefficients, we always employ the ones corresponding to the choice of the symbol '+' in $\alpha_{1,1}$ or $\alpha_{1,\mu}$ (see Tables 2.1, 2.2, and 2.3). The use of a different set of coefficients did not provide qualitatively different results in the examples. The pseudocodes of the directional splitting schemes EXPRK3DS_REAL and EXPRK3DS_CPLX (assuming a constant time step size) are given in Appendix A (Algorithms 1 and 2). Both schemes require the computation of small-sized matrix φ-functions of the matrices A_μ (once and for all before the actual time integration starts). This is done in practice by employing a rational Padé approach with modified scaling and squaring [38], and, as already mentioned above, the computational cost of this phase is negligible (see the timings reported in Section 3). We also remark that in the time integration all the relevant operations are performed in a tensor fashion, without the need to assemble the matrix K itself or to compute Kronecker products. In fact, the needed approximations of actions of φ-functions (i.e., the application of formulas (2.7), (2.9), and (2.11) to a vector) are realized in tensor form by employing Tucker operators (i.e., exploiting the equivalence (2.3)). Notice that even the action of the matrix K on a vector (needed to evaluate $\boldsymbol{f}(t_n,\boldsymbol{u}_n)$) is realized in tensor form thanks to the equivalence (see [12, formula (9)])

$$
K\boldsymbol{t} = (A_d \oplus \cdots \oplus A_1)\boldsymbol{t} = \operatorname{vec}\left(\sum_{\mu=1}^{d}(\boldsymbol{T}\times_\mu A_\mu)\right), \qquad \operatorname{vec}(\boldsymbol{T}) = \boldsymbol{t}. \tag{2.14}
$$

In practice, in our numerical experiments we compute the Tucker operator and the action of K by exploiting the high efficiency of level 3 BLAS, as thoroughly explained in [9, 12] and briefly summarized after formula (2.2). More specifically, when performing experiments in the MATLAB environment, we employ the functions `tucker` and `kronsumv` from the package KronPACK [12], which directly exploit the multithreaded MATLAB routines to perform GEMM. Concerning the experiments in CPU and GPU using C++ and CUDA, we directly call level 3 BLAS from efficient libraries available on the hardware (Intel MKL [24] and cuBLAS [36], respectively) to realize the needed tensor operations. Notice that, in this context, we expect consistent speedups in terms of wall-clock time by employing GPUs instead of CPUs, since these kinds of operations are very well implemented on the former (see also the discussions in [9]). In addition, the evaluation of the nonlinearity $\boldsymbol{g}$ and the

ETNA
Kent State University and
Johann Radon Institute (RICAM)

(pointwise) summation or multiplication operations, for instance, are performed by proper CUDA kernels, i.e., exploiting a massive parallelism. This is also an area in which GPUs greatly outperform the corresponding multithreaded version on CPUs. As a matter of fact, in our GPU implementations of Algorithms 1 and 2, the only communication with the CPU is for the solution of the linear systems in the Padé approximation of the φ-functions, which was faster if performed on the CPU. This is expected, since it is a small-sized task for which GPUs are not highly optimized. Apart from that, all the remaining code is executed directly on the GPU.

Finally, we remark that in order to perform a single integration step with Algorithms 1 and 2, we need an action of Kronecker sum and 10 and 15 Tucker operators, respectively. The cost of evaluating Tucker operators, independently of the considered hardware, is much lower compared to that of computing actions of exponentials and/or linear combinations of actions of φ-functions without directional splitting (as needed by the EXPRK3 integrator (2.13)). This has already been observed and discussed in full details, for instance in [9, 13]. Therefore, we may plausibly expect that the proposed integrators EXPRK3DS_REAL and EXPRK3DS_CPLX will perform well (compared to EXPRK3) in terms of simulation wall-clock time, even if in principle they may introduce a directional splitting error (see the next section for specific results on the performed numerical experiments).

3. Numerical examples. In this section we present an application of the proposed approximations for efficiently solving two popular systems of PDEs, namely the two-component 2D Schnakenberg and 3D FitzHugh–Nagumo models, using exponential integrators. Such models are important in the context of biochemical reactions and electric current flows, since they lead to the formation of so-called Turing patterns [3, 19, 32]. For both examples we first perform a semidiscretization in space using second-order uniform centered finite differences, with $n_\mu = n$ discretization points per direction, encapsulating the boundary conditions (homogeneous Neumann, in fact) directly into the relevant matrices. This leads to a system of ODEs in the form

$$\begin{bmatrix} \boldsymbol{u}'(t) \\ \boldsymbol{v}'(t) \end{bmatrix} = \begin{bmatrix} K_1 & 0 \\ 0 & K_2 \end{bmatrix} \begin{bmatrix} \boldsymbol{u}(t) \\ \boldsymbol{v}(t) \end{bmatrix} + \begin{bmatrix} \boldsymbol{g}^1(t, \boldsymbol{u}(t), \boldsymbol{v}(t)) \\ \boldsymbol{g}^2(t, \boldsymbol{u}(t), \boldsymbol{v}(t)) \end{bmatrix} = \begin{bmatrix} \boldsymbol{f}^1(t, \boldsymbol{u}(t), \boldsymbol{v}(t)) \\ \boldsymbol{f}^2(t, \boldsymbol{u}(t), \boldsymbol{v}(t)) \end{bmatrix}, \tag{3.1}$$

where $\boldsymbol{u}(t)$ and $\boldsymbol{v}(t)$ represent the two components of the system of PDEs, while K_1 and K_2 are matrices having Kronecker sum structure. Since the matrix of the system (3.1) is block diagonal, the needed actions of the matrix φ-functions can be performed separately for K_1 and K_2 using the techniques presented above (see also [8]).

For both examples, we perform several experiments which are briefly described in the following.

- First of all, we test the accuracy of the third-order directional split integrators EXPRK3DS_REAL and EXPRK3DS_CPLX and measure the performance against other methods in MATLAB (using the software MathWorks MATLAB® R2022a). To this aim, we fix the number of spatial discretization points, the final simulation time, and we let vary the number of time steps. As term of comparison we consider the second-order directional split integrator proposed in [8] (denoted as ETD2RKDS), which corresponds to the popular ETD2RK scheme (i.e., tableau (5.3) in [21] with $c_2 = 1$) with second-order directional splitting of the involved φ-functions. Moreover, we also present the results with the ETD2RK and the EXPRK3 integrators (without directional splitting), where, as mentioned in Section 2.4, the actions of the φ-functions are approximated with the incomplete orthogonalization Krylov-based technique [31] (input tolerance and incomplete orthogonalization parameter set to 1e-6 and 2, respectively). The error is always measured in the infinity norm with

ETNA
Kent State University and
Johann Radon Institute (RICAM)

respect to a reference solution computed with the EXPRK3DS_REAL method and a sufficiently large number of time steps. The hardware used for this experiment is a standard laptop equipped with an Intel Core i7-10750H CPU (6 physical cores) and 16GB of RAM. On the same hardware we perform a similar simulation using C++ to emphasize that the proposed procedures are effective also in this framework. In more detail, we test the accuracy and the performances of the directional split exponential integrators ETD2RKDS, EXPRK3DS_REAL, and EXPRK3DS_CPLX in double precision arithmetic. The gcc compiler version is 8.4.0, the Intel OneAPI MKL library version 2021.4.0 is used for BLAS, and OpenMP is employed for basic parallelization.

- Then, we perform a simulation with the proposed third-order integrators to show that we are able to retrieve the expected Turing pattern. For this aim we fix the number of spatial discretization points, we set a large final simulation time, and we let vary the number of time integration steps. We measure the performances on the same hardware employed previously using different software architectures, i.e., MATLAB and C++ with double precision arithmetic. We also present results using CUDA (single precision arithmetic, nvcc compiler and CUDA version 10.1, BLAS provided by cuBLAS, and CUDA kernels for massive parallelization). In this case the GPU employed is the mobile one provided with the laptop, i.e., an NVIDIA GeForce GTX 1650 Ti card (4GB of dedicated memory). Even if this is clearly a consumer-level GPU (and, in particular, not suitable for double precision arithmetic), it can still be effectively employed for the relatively small simulations under consideration at this stage.
- Finally, we present some results on professional CPU and GPU hardware. Once again, we first test the accuracy and the performances of ETD2RKDS, EXPRK3DS_REAL, and EXPRK3DS_CPLX by fixing the number of spatial discretization points, the final simulation time, and by letting vary the number of time steps. Double precision arithmetic is employed for both the CPU and the GPU. Then, we perform a simulation using the proposed third-order integrators for an increasing number of spatial discretization points (i.e., we increase the number of degrees of freedom). In this case we employ double precision arithmetic for the CPU results and both double and single precision arithmetic for the GPU ones. The CPU hardware is a dual socket Intel Xeon Gold 5118 with 2×12 cores, while the GPU is a single NVIDIA V100 card (equipped with 16 GB of RAM). When calling BLAS on the CPU, we use the Intel OneAPI MKL library version 2020.1.0, while on the GPU we employ cuBLAS from CUDA 11.2. For the parallelization we again employ OpenMP and CUDA kernels for the CPU and the GPU, respectively.

We remark that in all the examples presented here, the EXPRK3DS_CPLX integrator is outperformed by the EXPRK3DS_REAL scheme. Nevertheless, we decided to report also the results of the former for all the experiments. In fact, the complex method could still be convenient in other instances, such as the integration of complex-valued PDEs. We believe that having an idea of the overall computational cost may then be of scientific interest.

The MATLAB code to reproduce the examples (fully compatible with GNU Octave) is publicly available in a GitHub repository[1].

[1] Available at https://github.com/cassinif/Expds3, commit 1321521.

ETNA
Kent State University and
Johann Radon Institute (RICAM)

3.1. Two-dimensional Schnakenberg model. We consider the following Schnakenberg model in two space dimensions (see [3, 8, 18])

$$\begin{cases} \partial_t u = \delta^u \Delta u + \rho(a^u - u + u^2 v), \\ \partial_t v = \delta^v \Delta v + \rho(a^v - u^2 v), \end{cases} \tag{3.2}$$

defined in the spatial domain $\Omega = (0,1)^2$ with homogeneous Neumann boundary conditions. The parameters are set to $\delta^u = 1$, $\delta^v = 10$, $\rho = 1000$, $a^u = 0.1$, and $a^v = 0.9$ so that the equilibrium $(u_e, v_e) = (a^u + a^v, a^v/(a^u + a^v)^2)$ is susceptible of a 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 $\mathcal{U}(0,1)$ denotes the uniformly distributed random variable in $(0,1)$.

We start by performing simulations in MATLAB with the spatial domain discretized by $n = 150$ point per direction, i.e., the total number of degrees of freedom is $N = 2 \cdot 150^2$. The final time is set to $T = 0.25$. The number of time steps ranges from 3000 to 6000 for the second-order integrators, while from 1000 to 2500 for the third-order ones. The results are collected in an error decay plot and in a work-precision diagram in Figure 3.1. First

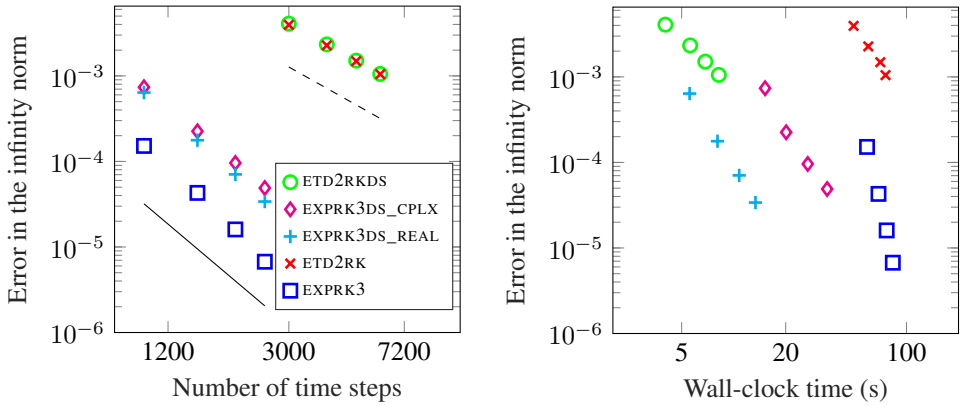

FIG. 3.1. *Results in MATLAB (standard laptop) for the simulation of the 2D Schnakenberg model (3.2) with $n = 150$ spatial discretization points per direction up to final time $T = 0.25$. Left plot: error decay, with reference slope lines of order two (dashed) and three (solid). Right plot: work-precision diagram.*

of all, notice that all the integrators show the expected order of convergence. Moreover, we observe that the third-order integrators EXPRK3DS_CPLX and EXPRK3DS_REAL have larger errors compared to the classical EXPRK3 method, plausibly because of the introduced directional splitting approximation. Nevertheless, the gain in efficiency in terms of wall-clock time is neat (see the right plot of Figure 3.1). In fact, the work-precision diagram is separated into two parts. On the right-hand side we have the most expensive integrators, i.e., ETD2RK and EXPRK3 implemented with a general-purpose technique for computing actions of φ-functions. On the left-hand side we have the directional split integrators, which always perform better with respect to their original counterparts. Considering in more detail the third-order directional split schemes, we observe that EXPRK3DS_CPLX and EXPRK3DS_REAL do not show a considerable difference in terms of achieved accuracy. Hence, as expected, the one that employs just real arithmetic is cheaper. Overall, for stringent accuracies, the best performant integrator is EXPRK3DS_REAL, and in particular it outperforms the ETD2RKDS scheme already available in the literature. We then perform a similar simulation using C++. We report the results of the directional split integrators ETD2RKDS, EXPRK3DS_REAL, and EXPRK3DS_CPLX in Figure 3.2. The conclusions are essentially the same as drawn from the MATLAB experiment, i.e., the integrator that performs best is EXPRK3DS_REAL.

ETNA
Kent State University and
Johann Radon Institute (RICAM)

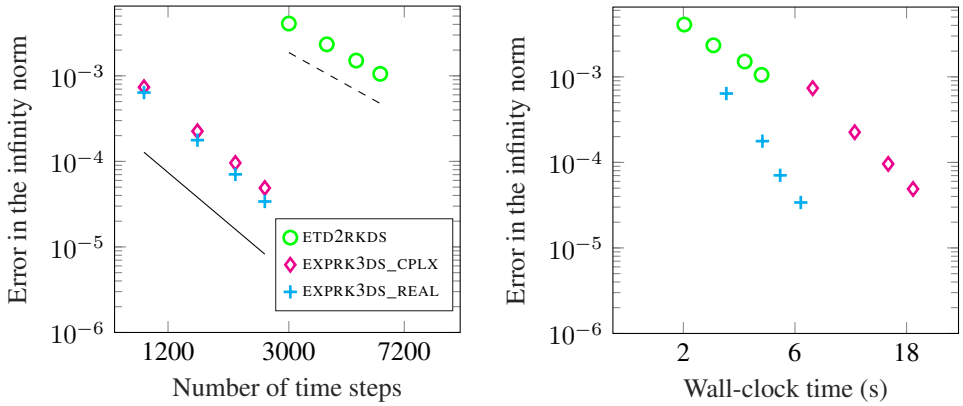

FIG. 3.2. *Results in C++ (standard laptop) for the simulation of the 2D Schnakenberg model (3.2) with $n = 150$ spatial discretization points per direction up to final time $T = 0.25$. Left plot: error decay, with reference slope lines of order two (dashed) and three (solid). Right plot: work-precision diagram.*

Then, we compare on different software architectures the results of the third-order integrators EXPRK3DS_REAL and EXPRK3DS_CPLX, i.e., the ones that employ the proposed approximations, to achieve the expected stationary pattern (a cos-like structure with modes $(3, 5), (5, 3)$ [18]). To this aim, we set the final simulation time to $T = 2$, and we let vary the number of time steps, while the space discretization is the same as for the previous experiments (i.e., $n = 150$ points per direction). A representative of the obtained pattern in the u component is shown in Figure 3.3, and the results of the experiment are collected in Table 3.1. Notice

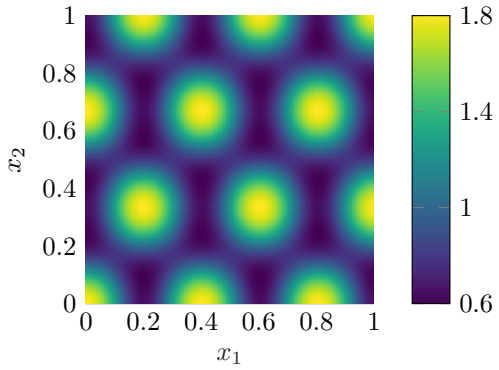

FIG. 3.3. *Turing pattern (u component) for the 2D Schnakenberg model (3.2) obtained at final time $T = 2$ with $n = 150$ spatial discretization points per direction.*

that, as expected, the wall-clock time of all the simulations is proportional to the number of time steps. In fact, after the computation of the small-sized φ-functions, the integrators are direct, and their total computational burden could be easily predicted by performing a single time step, similarly to what was stated for the ETD2RKDS scheme in [8]. Also, remark that the wall-clock time needed to compute the φ-functions is negligible compared to that of the time marching of the methods, and, again as expected, the EXPRK3DS_CPLX integrator is computationally more expensive than EXPRK3DS_REAL. We observe that the employment of GPUs in this context is really effective. In fact, even if we are using a basic consumer-level GPU card, we can still obtain the Turing pattern in a short amount of time.

ETNA
Kent State University and
Johann Radon Institute (RICAM)

TABLE 3.1
Wall-clock time (in seconds) for the simulation of the 2D Schnakenberg model (3.2) up to $T = 2$ with $n = 150$ spatial discretization points per direction (i.e., $N = 2 \cdot 150^2$ degrees of freedom). The time integrators are EXPRK3DS_REAL *(top) and* EXPRK3DS_CPLX *(bottom), with varying number of time steps, using different software architectures (standard laptop). In brackets we report the portion of time needed for computing the φ-functions.*

Number of steps	MATLAB *double*	C++ *double*	CUDA *single*
2000	10.03 (0.26)	4.38 (0.21)	0.93 (0.05)
4000	18.87 (0.25)	8.63 (0.20)	1.76 (0.06)
6000	28.69 (0.27)	13.09 (0.21)	2.61 (0.06)

Number of steps	MATLAB *double*	C++ *double*	CUDA *single*
2000	23.61 (0.74)	12.98 (0.61)	2.41 (0.12)
4000	46.11 (0.75)	26.12 (0.60)	4.77 (0.11)
6000	70.89 (0.77)	39.20 (0.62)	6.84 (0.11)

We proceed by presenting some performance results doing simulations with professional hardware (see the beginning of Section 3 for the details). Similarly to what was performed previously, we first test the directional split integrators on problem (3.2) with the spatial domain discretized by $n = 300$ point per direction, i.e., the total number of degrees of freedom is $N = 2 \cdot 300^2$. The final time is set to $T = 0.25$. The number of time steps ranges from 3000 to 6000 for the second-order integrators, while from 1000 to 2500 for the third-order ones. For the computations, we use double precision arithmetic. The outcome is collected in the error decay plot and in the work-precision diagram in Figure 3.4. As we can see, the

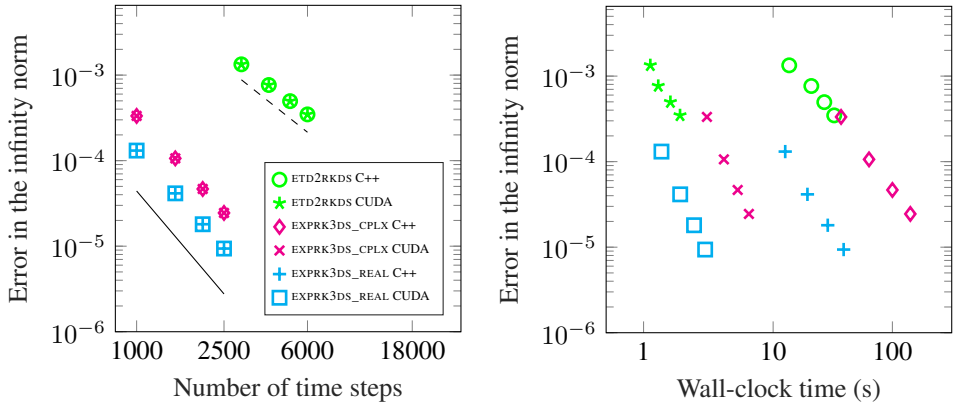

FIG. 3.4. *Results in C++ and CUDA (professional hardware) for the simulation of the 2D Schnakenberg model (3.2) with $n = 300$ spatial discretization points per direction up to final time $T = 0.25$. Left plot: error decay, with reference slope lines of order two (dashed) and three (solid). Right plot: work-precision diagram.*

results are coherent with what was observed in the previous experiments using a standard laptop. In particular, all the integrators have the expected order of convergence, and the specific hardware employed (CPU or GPU) does not influence the resulting error. On the other hand, the effectiveness of the usage of GPUs in this context is clear. In fact, we observe a speedup of roughly a factor of 10 in favour of the GPUs, and the integrator that performs best is EXPRK3DS_REAL. To conclude, we perform a simulation with increasing number of degrees of freedom N using the third-order directional split integrators EXPRK3DS_REAL and EXPRK3DS_CPLX. We set the final time to $T = 2$, while the number of time steps is fixed to 6000. The results are summarized in Table 3.2. Again, the integrator employing complex-valued directional splitting coefficients is more costly than that with real-valued ones. Comparing CPU and GPU simulations, the scaling in terms of computational time is still

ETNA
Kent State University and
Johann Radon Institute (RICAM)

TABLE 3.2
Wall-clock time (in seconds) for the simulation of the 2D Schnakenberg model (3.2) up to $T = 2$ with increasing number of degrees of freedom N, 6000 time steps, and different software architectures (professional hardware). The time integrators are EXPRK3DS_REAL *(top) and* EXPRK3DS_CPLX *(bottom). In brackets we report the portion of time needed for computing the φ-functions.*

Number of d.o.f. N	C++ *double*	CUDA *double*	CUDA *single*
$2 \cdot 300^2$	73.11 (1.85)	7.32 (0.31)	4.12 (0.26)
$2 \cdot 450^2$	196.52 (4.72)	16.39 (0.54)	7.44 (0.35)
$2 \cdot 600^2$	440.14 (8.73)	26.15 (0.83)	13.17 (0.51)

Number of d.o.f. N	C++ *double*	CUDA *double*	CUDA *single*
$2 \cdot 300^2$	192.14 (4.64)	16.33 (0.52)	7.90 (0.31)
$2 \cdot 450^2$	607.79 (13.06)	46.28 (1.01)	23.91 (0.64)
$2 \cdot 600^2$	1663.05 (23.32)	89.93 (1.73)	42.37 (0.98)

very good (roughly a factor of 10 to 18 for double precision). This is expected since the main computational cost in the time integration comes from the Tucker operators and hence scales favourably for the GPU.

3.2. Three-dimensional FitzHugh–Nagumo model. We now consider the FitzHugh–Nagumo model in three space dimensions

$$\begin{cases} \partial_t u = \delta^u \Delta u + \rho(-u(u^2 - 1) - v), \\ \partial_t v = \delta^v \Delta v + \rho a_1^v (u - a_2^v v), \end{cases} \tag{3.3}$$

defined in the spatial domain $\Omega = (0, \pi)^3$ with homogeneous Neumann boundary conditions. The parameters are $\delta^u = 1$, $\delta^v = 42.1887$, $\rho = 24.649$, $a_1^v = 11$, and $a_2^v = 0.1$. With this choice, the equilibrium $(u_e, v_e) = (0, 0)$ is susceptible of a Turing instability, and we expect to achieve in the long-time regime a stationary square pattern with modes $(2, 2, 2)$ (see also [8, 19]). The initial conditions are set to $u_0 = 10^{-3} \cdot \mathcal{U}(0, 1)$ and $v_0 = 10^{-3} \cdot \mathcal{U}(0, 1)$, where as in the previous example $\mathcal{U}(0, 1)$ denotes the uniform random variable in the interval $(0, 1)$.

For the first experiment, we discretize the spatial domain with a grid of $n = 64$ points per direction (total number of degrees of freedom $N = 2 \cdot 64^3$), and we simulate up to final time $T = 5$ with different integrators and a number of time steps ranging from 60000 to 75000 for the second-order method, while from 14000 to 20000 for the third-order ones. The results are graphically depicted in Figure 3.5. First of all notice that the results with the ETD2RK and EXPRK3 integrators are not reported in the plots since their simulation wall-clock time was too large. This is in line with what already observed in the two-dimensional example of Section 3.1. In fact, to obtain comparable accuracies, a simulation with ETD2RK took roughly 6300 seconds (60000 time steps), while EXPRK3 needed about 6100 seconds (14000 time steps). Again similarly to the 2D Schnakenberg experiment, the achieved accuracies of EXPRK3DS_CPLX and EXPRK3DS_REAL are similar, with an advantage of the latter in terms of execution time. Overall, for the range of accuracies under consideration, also in this case the most performant method is EXPRK3DS_REAL. We repeat the simulations with the directional split exponential integrators in C++ (double precision on a standard laptop). The results, summarized in Figure 3.6, lead essentially to the same conclusions.

Then, we proceed by comparing the results of EXPRK3DS_REAL and EXPRK3DS_CPLX to achieve the expected stationary pattern on different software architectures. To this aim, we set the final time to $T = 150$, we let vary the number of time steps, while the semidiscretization in space is performed with $n = 64$ points per direction. The obtained pattern in the u component is illustrated in Figure 3.7, while the outcome of the experiment is summarized in Table 3.3.

ETNA
Kent State University and
Johann Radon Institute (RICAM)

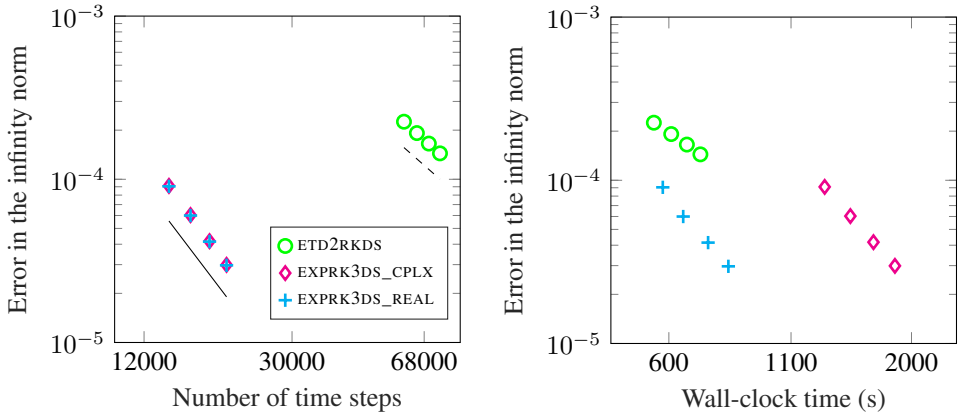

FIG. 3.5. *Results in MATLAB (standard laptop) for the simulation of the 3D FitzHugh–Nagumo model (3.3) with $n = 64$ spatial discretization points per direction up to final time $T = 5$. Left plot: error decay, with reference slope lines of order two (dashed) and three (solid). Right plot: work-precision diagram.*

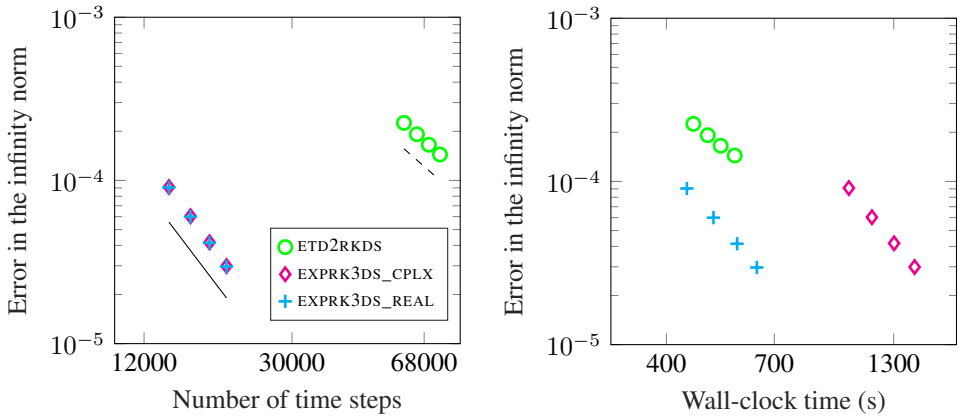

FIG. 3.6. *Results in C++ (standard laptop) for the simulation of the 3D FitzHugh–Nagumo model (3.3) with $n = 64$ spatial discretization points per direction up to final time $T = 5$. Left plot: error decay, with reference slope lines of order two (dashed) and three (solid). Right plot: work-precision diagram.*

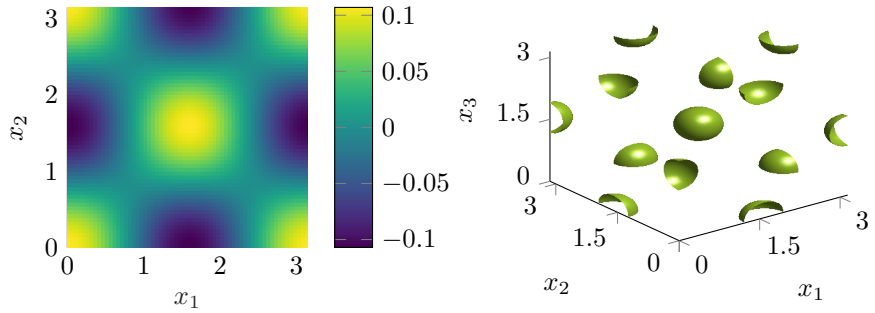

FIG. 3.7. *Turing pattern (u component) for the 3D FitzHugh–Nagumo model (3.3) obtained at final time $T = 150$ with $n = 64$ spatial discretization points per direction. The reported slice (left plot) corresponds to $x_3 = 1.55$ and the isosurface value (right plot) is 0.08.*

ETNA
Kent State University and
Johann Radon Institute (RICAM)

TABLE 3.3
Wall-clock time (in seconds) for the simulation of the 3D FitzHugh–Nagumo model (3.3) up to $T = 150$ with $n = 64$ spatial discretization points per direction (i.e., $N = 2 \cdot 64^3$ degrees of freedom). The time integrators are EXPRK3DS_REAL *(top) and* EXPRK3DS_CPLX *(bottom), with varying number of time steps, using different software architectures (standard laptop). In brackets we report the portion of time needed for computing the φ-functions.*

Number of steps	MATLAB *double*	C++ *double*	CUDA *single*
10000	439.47 (0.09)	340.36 (0.06)	26.90 (0.05)
15000	607.62 (0.11)	509.48 (0.07)	40.92 (0.07)
20000	850.41 (0.12)	677.91 (0.07)	54.54 (0.07)

Number of steps	MATLAB *double*	C++ *double*	CUDA *single*
10000	918.86 (0.13)	733.68 (0.07)	44.99 (0.06)
15000	1396.22 (0.14)	1087.42 (0.07)	65.87 (0.06)
20000	1855.47 (0.14)	1460.29 (0.07)	90.54 (0.06)

Again, as already observed for the 2D Schnakenberg model, the wall-clock time is proportional to the number of time steps for both the integrators and all the architectures under consideration. We recall that, in this three-dimensional setting, we need to compute two *complex* and three *real* Tucker operators for each action of the φ-function approximation in EXPRK3DS_CPLX and EXPRK3DS_REAL, respectively. Even if we need one Tucker operator less for the former, obviously the computational cost of forming the approximation is larger since we employ complex arithmetic. Overall, we then observe that for the example under consideration it is more convenient to employ the EXPRK3DS_REAL method. Also, notice that the employment of a GPU in this three-dimensional case is even more effective compared to the two-dimensional scenario.

Finally, we present the performance results by doing simulations with professional-level hardware. To this purpose, we first test the directional split integrators on problem (3.3) with the spatial domain discretized by $n = 100$ point per direction, i.e., the total number of degrees of freedom is $N = 2 \cdot 100^3$. The final time is set to $T = 5$. The number of time steps ranges from 60000 to 75000 for the second-order integrators, while from 14000 to 20000 for the third-order ones. We use double precision arithmetic, and the results are collected in Figure 3.8. As highlighted also in the previous experiments, we observe a clear order of convergence for all the integrators and the architectures under consideration. Moreover, from the work-precision diagram we conclude that also in this case it is more convenient to employ the integrator with real arithmetic rather than the one with complex coefficients, and overall the integrator that performs best in reaching stringent accuracies is EXPRK3DS_REAL implemented in CUDA (with an average speedup of 22 times passing from the CPU to the GPU). To conclude, we perform a simulation considering the third-order integrators EXPRK3DS_REAL and EXPRK3DS_CPLX with varying total number of degrees of freedom N. The final time is set to $T = 150$ and the number of time steps to 10000. The results are presented in Table 3.4. Also in this final experiment, as expected, the scaling is very good comparing the CPU and GPU simulations. In particular, the resulting speedup from CPU to GPU in double precision is given by a factor around 14 and 23, hence larger than the 2D case. Performing the experiment in single precision arithmetic on the GPU allows one to gain an additional factor of 2. The usage of the scheme EXPRK3DS_CPLX results in larger computational cost also for this example.

4. Conclusions. In this manuscript, we introduced third-order directional split approximations for matrix φ-functions with underlying d-dimensional Kronecker sum structure. The derived formulas allow for the efficient construction and employment of directional split exponential Runge–Kutta integrators of order three for the time integration of ODEs systems with

ETNA
Kent State University and
Johann Radon Institute (RICAM)

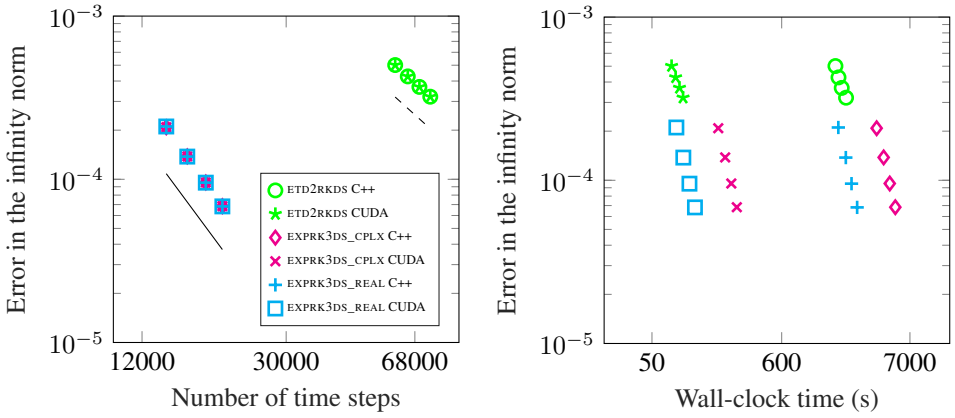

FIG. 3.8. *Results in C++ and CUDA (professional hardware) for the simulation of the 3D FitzHugh–Nagumo model (3.3) with $n = 100$ spatial discretization points per direction up to final time $T = 5$. Left plot: error decay, with reference slope lines of order two (dashed) and three (solid). Right plot: work-precision diagram.*

TABLE 3.4
Wall-clock time (in seconds) for the simulation of the 3D FitzHugh–Nagumo model (3.3) up to $T = 150$ with increasing number of degrees of freedom N, 10000 time steps, and different software architectures (professional hardware). The time marching integrators are EXPRK3DS_REAL *(top) and* EXPRK3DS_CPLX *(bottom). In brackets we report the portion of time needed for computing the φ-functions.*

Number of d.o.f. N	C++ *double*	CUDA *double*	CUDA *single*
$2 \cdot 100^3$	1261.13 (0.29)	57.31 (0.28)	32.59 (0.22)
$2 \cdot 150^3$	3686.32 (0.62)	251.62 (0.40)	119.60 (0.33)
$2 \cdot 200^3$	9746.87 (1.12)	685.89 (0.44)	321.81 (0.35)

Number of d.o.f. N	C++ *double*	CUDA *double*	CUDA *single*
$2 \cdot 100^3$	2682.31 (0.37)	127.47 (0.24)	67.46 (0.22)
$2 \cdot 150^3$	10732.53 (0.90)	470.61 (0.33)	241.18 (0.28)
$2 \cdot 200^3$	24640.98 (2.41)	1469.11 (0.45)	753.38 (0.32)

Kronecker structure. The efficiency of the proposed tensor approach has been tested against state-of-the-art techniques on two important physical models in the context of Turing patterns for diffusion-reaction systems of partial differential equations, namely the 2D Schnakenberg and the 3D FitzHugh–Nagumo models. In particular, it turns out that the investigated third-order directional split integrator EXPRK3DS_REAL outperforms the ETD2RKDS scheme that was already available in the literature. The numerical experiments also clearly show that the procedures scale very well on modern computer hardware such as Graphic Processing Units.

Acknowledgments. Fabio Cassini received financial support from the Italian Ministry of University and Research (MUR) with the PRIN Project 2022 No. 2022N9BM3N "Efficient numerical schemes and optimal control methods for time-dependent partial differential equations". The author is a member of the Gruppo Nazionale Calcolo Scientifico-Istituto Nazionale di Alta Matematica (GNCS-INdAM) and holds a post-doc fellowship funded by INdAM. We would like to thank Marco Caliari for the fruitful discussions and the valuable suggestions. Also, we are grateful to Lukas Einkemmer and the Research Area Scientific Computing of the University of Innsbruck for the availability to use their professional hardware resources.

Appendix A. Pseudocodes. In the following, in Algorithm 1 and Algorithm 2 we list the pseudocodes for the directional split exponential integrators EXPRK3DS_REAL and EXPRK3DS_CPLX, which employ the approximations presented in the manuscript.

ETNA
Kent State University and
Johann Radon Institute (RICAM)

Algorithm 1: Pseudocode for EXPRK3DS_REAL (if $d = 2$) and EXPRK3DS_CPLX. The relevant coefficients for the directional splitting approximations are given in Table 2.1 (real coefficients) for EXPRK3DS_REAL and in Table 2.2 for EXPRK3DS_CPLX. The notations $\alpha_{i,\mu}^{(\ell)}$ and $\eta_{i,d}^{(\ell)}$ mean $\alpha_{i,\mu}$ and $\eta_{i,d}$ for the approximation of φ_ℓ. The symbols $\mathcal{K}$ and $\mathcal{T}$ denote the action of the Kronecker sum in tensor form (2.14) and the Tucker operator (2.3), respectively, where the first input is the tensor and the second one is the sequence of matrices.

Input: Initial datum tensor ($\boldsymbol{U}_0$), ODEs nonlinear function ($\boldsymbol{g}$), matrices which constitute K ($A_1, \ldots, A_d$), final time (T), and number of time steps (m).
Output: Approximated solution $\boldsymbol{U}$ at final time T.

Compute $\tau = T/m$;
Needed φ-functions
for $\mu = 1, \ldots, d$ **do**
  Set $A\{\mu\} = A_\mu$;
  for $i = 1, 2$ **do**
    For stage U_{n2}
    Compute $P_{i,2}^{(1)}\{\mu\} = \varphi_{\ell_i}\left(\frac{\tau}{3}\alpha_{i,\mu}^{(1)}A\{\mu\}\right)$;
    for $\ell = 1, 2$ **do**
      For stage U_{n3}
      Compute $P_{i,3}^{(\ell)}\{\mu\} = \varphi_{\ell_i}\left(\frac{2\tau}{3}\alpha_{i,\mu}^{(\ell)}A\{\mu\}\right)$;
      For final approximation U_{n+1}
      Compute $P_{i,f}^{(\ell)}\{\mu\} = \varphi_{\ell_i}\left(\tau\alpha_{i,\mu}^{(\ell)}A\{\mu\}\right)$;
    end
  end
end
Set $t = 0$ and $\boldsymbol{U} = \boldsymbol{U}_0$;
Time integration
for $n = 0, \ldots, m-1$ **do**
  Compute $\boldsymbol{G} = \boldsymbol{g}(t, \boldsymbol{U})$ and $\boldsymbol{F} = \mathcal{K}(\boldsymbol{U}, A) + \boldsymbol{G}$;
  Stage U_{n2}
  Compute $\boldsymbol{U}_2 = \boldsymbol{U} + \frac{\tau}{3}\left(\eta_{1,d}^{(1)}\mathcal{T}(\boldsymbol{F}, P_{1,2}^{(1)}) + \eta_{2,d}^{(1)}\mathcal{T}(\boldsymbol{F}, P_{2,2}^{(1)})\right)$;
  Stage U_{n3}
  Compute $\boldsymbol{D}_2 = \boldsymbol{g}(t + \frac{\tau}{3}, \boldsymbol{U}_2) - \boldsymbol{G}$;
  Compute $\boldsymbol{U}_3 = \boldsymbol{U} + \frac{2\tau}{3}\left(\eta_{1,d}^{(1)}\mathcal{T}(\boldsymbol{F}, P_{1,3}^{(1)}) + \eta_{2,d}^{(1)}\mathcal{T}(\boldsymbol{F}, P_{2,3}^{(1)})\right) +$
  $\frac{4\tau}{3}\left(\eta_{1,d}^{(2)}\mathcal{T}(\boldsymbol{D}_2, P_{1,3}^{(2)}) + \eta_{2,d}^{(2)}\mathcal{T}(\boldsymbol{D}_2, P_{2,3}^{(2)})\right)$;
  Final approximation U_{n+1}
  Compute $\boldsymbol{D}_3 = \boldsymbol{g}(t + \frac{2\tau}{3}, \boldsymbol{U}_3) - \boldsymbol{G}$;
  Compute $\boldsymbol{U} = \boldsymbol{U} + \tau\left(\eta_{1,d}^{(1)}\mathcal{T}(\boldsymbol{F}, P_{1,f}^{(1)}) + \eta_{2,d}^{(1)}\mathcal{T}(\boldsymbol{F}, P_{2,f}^{(1)})\right) +$
  $\frac{3\tau}{2}\left(\eta_{1,d}^{(2)}\mathcal{T}(\boldsymbol{D}_3, P_{1,f}^{(2)}) + \eta_{2,d}^{(2)}\mathcal{T}(\boldsymbol{D}_3, P_{2,f}^{(2)})\right)$;
  Set $t = t + \tau$;
end

ETNA
Kent State University and
Johann Radon Institute (RICAM)

Algorithm 2: Pseudocode for EXPRK3DS_REAL (if $d > 2$). The relevant coefficients for the directional splitting approximations are given in Table 2.3. The notations $\alpha_{i,\mu}^{(\ell)}$ and $\eta_{i,d}^{(\ell)}$ mean $\alpha_{i,\mu}$ and $\eta_{i,d}$ for the approximation of φ_ℓ. The symbols $\mathcal{K}$ and $\mathcal{T}$ denote the action of the Kronecker sum in tensor form (2.14) and the Tucker operator (2.3), respectively, where the first input is the tensor and the second one is the sequence of matrices.

Input: Initial datum tensor ($\boldsymbol{U}_0$), ODEs nonlinear function ($\boldsymbol{g}$), matrices which constitute K ($A_1, \dots, A_d$), final time (T), and number of time steps (m).
Output: Approximated solution $\boldsymbol{U}$ at final time T.

Compute $\tau = T/m$;
Needed φ-functions
for $\mu = 1, \dots, d$ **do**
  Set $A\{\mu\} = A_\mu$;
  for $i = 1, 2, 3$ **do**
    For stage U_{n2}
    Compute $P_{i,2}^{(1)}\{\mu\} = \varphi_{\ell_i}\left(\frac{\tau}{3}\alpha_{i,\mu}^{(1)}A\{\mu\}\right)$;
    for $\ell = 1, 2$ **do**
      For stage U_{n3}
      Compute $P_{i,3}^{(\ell)}\{\mu\} = \varphi_{\ell_i}\left(\frac{2\tau}{3}\alpha_{i,\mu}^{(\ell)}A\{\mu\}\right)$;
      For final approximation U_{n+1}
      Compute $P_{i,f}^{(\ell)}\{\mu\} = \varphi_{\ell_i}\left(\tau\alpha_{i,\mu}^{(\ell)}A\{\mu\}\right)$;
    end
  end
end
Set $t = 0$ and $\boldsymbol{U} = \boldsymbol{U}_0$;
Time integration
for $n = 0, \dots, m-1$ **do**
  Compute $\boldsymbol{G} = \boldsymbol{g}(t, \boldsymbol{U})$ and $\boldsymbol{F} = \mathcal{K}(\boldsymbol{U}, A) + \boldsymbol{G}$;
  Stage U_{n2}
  Compute
  $$\boldsymbol{U}_2 = \boldsymbol{U} + \tfrac{\tau}{3}\left(\eta_{1,d}^{(1)}\mathcal{T}(\boldsymbol{F}, P_{1,2}^{(1)}) + \eta_{2,d}^{(1)}\mathcal{T}(\boldsymbol{F}, P_{2,2}^{(1)}) + \eta_{3,d}^{(1)}\mathcal{T}(\boldsymbol{F}, P_{3,2}^{(1)})\right);$$
  Stage U_{n3}
  Compute $\boldsymbol{D}_2 = \boldsymbol{g}(t + \frac{\tau}{3}, \boldsymbol{U}_2) - \boldsymbol{G}$;
  Compute
  $$\boldsymbol{U}_3 = \boldsymbol{U} + \tfrac{2\tau}{3}\left(\eta_{1,d}^{(1)}\mathcal{T}(\boldsymbol{F}, P_{1,3}^{(1)}) + \eta_{2,d}^{(1)}\mathcal{T}(\boldsymbol{F}, P_{2,3}^{(1)}) + \eta_{3,d}^{(1)}\mathcal{T}(\boldsymbol{F}, P_{3,3}^{(1)})\right) +$$
  $$\tfrac{4\tau}{3}\left(\eta_{1,d}^{(2)}\mathcal{T}(\boldsymbol{D}_2, P_{1,3}^{(2)}) + \eta_{2,d}^{(2)}\mathcal{T}(\boldsymbol{D}_2, P_{2,3}^{(2)}) + \eta_{3,d}^{(2)}\mathcal{T}(\boldsymbol{D}_2, P_{3,3}^{(2)})\right);$$
  Final approximation U_{n+1}
  Compute $\boldsymbol{D}_3 = \boldsymbol{g}(t + \frac{2\tau}{3}, \boldsymbol{U}_3) - \boldsymbol{G}$;
  Compute $\boldsymbol{U} = \boldsymbol{U} + \tau\left(\eta_{1,d}^{(1)}\mathcal{T}(\boldsymbol{F}, P_{1,f}^{(1)}) + \eta_{2,d}^{(1)}\mathcal{T}(\boldsymbol{F}, P_{2,f}^{(1)}) + \eta_{3,d}^{(1)}\mathcal{T}(\boldsymbol{F}, P_{3,f}^{(1)})\right) +$
  $$\tfrac{3\tau}{2}\left(\eta_{1,d}^{(2)}\mathcal{T}(\boldsymbol{D}_3, P_{1,f}^{(2)}) + \eta_{2,d}^{(2)}\mathcal{T}(\boldsymbol{D}_3, P_{2,f}^{(2)}) + \eta_{3,d}^{(2)}\mathcal{T}(\boldsymbol{D}_3, P_{3,f}^{(2)})\right);$$
  Set $t = t + \tau$;
end

ETNA
Kent State University and
Johann Radon Institute (RICAM)

REFERENCES

[1] A. H. AL-MOHY AND N. J. HIGHAM, *A new scaling and squaring algorithm for the matrix exponential*, SIAM J. Matrix Anal. Appl., 31 (2010), pp. 970–989.
[2] ———, *Computing the action of the matrix exponential, with an application to exponential integrators*, SIAM J. Sci. Comput., 33 (2011), pp. 488–511.
[3] A. ALLA, A. MONTI, AND I. SGURA, *Adaptive POD-DEIM correction for Turing pattern approximation in reaction-diffusion PDE systems*, J. Numer. Math., 31 (2023), pp. 205–229.
[4] J. M. ALONSO, J. IBÁÑEZ, E. DEFEZ, AND P. ALONSO-JORDÁ, *Euler polynomials for the matrix exponential approximation*, J. Comput. Appl. Math., 425 (2023), Paper No. 115074, 15 pages.
[5] E. O. ASANTE-ASAMANI, A. KLEEFELD, AND B. A. WADE, *A second-order exponential time differencing scheme for non-linear reaction-diffusion systems with dimensional splitting*, J. Comput. Phys., 415 (2020), Paper No. 109490, 18 pages.
[6] M. BENZI AND V. SIMONCINI, *Approximation of functions of large matrices with Kronecker structure*, Numer. Math., 135 (2017), pp. 1–26.
[7] M. CALIARI AND F. CASSINI, *Direction splitting of φ-functions in exponential integrators for d-dimensional problems in Kronecker form*, J. Approx. Softw., 1 (2024), Paper No. 1, 20 pages.
[8] ———, *A second order directional split exponential integrator for systems of advection-diffusion-reaction equations*, J. Comput. Phys., 498 (2024), Paper No. 112640, 17 pages.
[9] M. CALIARI, F. CASSINI, L. EINKEMMER, A. OSTERMANN, AND F. ZIVCOVICH, *A μ-mode integrator for solving evolution equations in Kronecker form*, J. Comput. Phys., 455 (2022), Paper No. 110989, 16 pages.
[10] M. CALIARI, F. CASSINI, AND F. ZIVCOVICH, *Approximation of the matrix exponential for matrices with a skinny field of values*, BIT Numer. Math., 60 (2020), pp. 1113–1131.
[11] ———, *BAMPHI: matrix-free and transpose-free action of linear combinations of φ-functions from exponential integrators*, J. Comput. Appl. Math., 423 (2023), Paper No. 114973, 16 pages.
[12] ———, *A μ-mode BLAS approach for multidimensional tensor-structured problems*, Numer. Algorithms, 92 (2023), pp. 2483–2508.
[13] ———, *A μ-mode approach for exponential integrators: actions of φ-functions of Kronecker sums*, Calcolo, 61 (2024), Paper No. 61, 28 pages.
[14] M. CALIARI, P. KANDOLF, A. OSTERMANN, AND S. RAINER, *The Leja method revisited: backward error analysis for the matrix exponential*, SIAM J. Sci. Comput., 38 (2016), pp. A1639–A1661.
[15] M. CALIARI AND F. ZIVCOVICH, *On-the-fly backward error estimate for matrix exponential approximation by Taylor algorithm*, J. Comput. Appl. Math., 346 (2019), pp. 532–548.
[16] D. A. COX, J. LITTLE, AND D. O'SHEA, *Ideals, Varieties, and Algorithms*, 4th ed., Springer, Cham, 2015.
[17] M. CROCI AND 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), Paper No. 101966, 10 pages.
[18] M. C. D'AUTILIA, I. SGURA, AND V. SIMONCINI, *Matrix-oriented discretization methods for reaction-diffusion PDEs: comparisons and applications*, Comput. Math. Appl., 79 (2020), pp. 2067–2085.
[19] G. GAMBINO, M. C. LOMBARDO, G. RUBINO, AND M. SAMMARTINO, *Pattern selection in the 2D FitzHugh–Nagumo model*, Ric. Mat., 68 (2019), pp. 535–549.
[20] S. GAUDREAULT, G. RAINWATER, AND M. TOKMAN, *KIOPS: a fast adaptive Krylov subspace solver for exponential integrators*, J. Comput. Phys., 372 (2018), pp. 236–255.
[21] M. HOCHBRUCK AND A. OSTERMANN, *Explicit exponential Runge-Kutta methods for semilinear parabolic problems*, SIAM J. Numer. Anal., 43 (2005), pp. 1069–1090.
[22] ———, *Exponential integrators*, Acta Numer., 19 (2010), pp. 209–286.
[23] J. HUANG, L. JU, AND B. WU, *A fast compact exponential time differencing method for semilinear parabolic equations with Neumann boundary conditions*, Appl. Math. Lett., 94 (2019), pp. 257–265.
[24] INTEL CORPORATION, *Intel Math Kernel Library*, 2024. https://software.intel.com/content/www/us/en/develop/tools/oneapi/components/onemkl.html,
[25] L. JU, J. ZHANG, L. ZHU, AND Q. DU, *Fast explicit integration factor methods for semilinear parabolic equations*, J. Sci. Comput., 62 (2015), pp. 431–455.
[26] T. G. KOLDA, *Multilinear operators for higher-order decompositions*, Tech. Rep. SAND2006-2081, Sandia National Laboratories, April 2006.
[27] T. G. KOLDA AND B. W. BADER, *Tensor decompositions and applications*, SIAM Rev., 51 (2009), pp. 455–500.
[28] D. LI, S. YANG, AND J. LAN, *Efficient and accurate computation for the φ-functions arising from exponential integrators*, Calcolo, 59 (2022), Paper No. 11, 24 pages.
[29] D. LI, X. ZHANG, AND R. LIU, *Exponential integrators for large-scale stiff Riccati differential equations*, J. Comput. Appl. Math., 389 (2021), Paper No. 113360, 17 pages.

ETNA
Kent State University and
Johann Radon Institute (RICAM)

[30] D. LI, Y. ZHANG, AND X. ZHANG, *Computing the Lyapunov operator φ-functions, with an application to matrix-valued exponential integrators*, Appl. Numer. Math., 182 (2022), pp. 330–343.
[31] V. T. LUAN, J. A. PUDYKIEWICZ, AND D. R. REYNOLDS, *Further development of efficient and accurate time integration schemes for meteorological models*, J. Comput. Phys., 376 (2019), pp. 817–837.
[32] H. MALCHOW, S. V. PETROVSKII, AND E. VENTURINO, *Spatiotemporal Patterns in Ecology and Epidemiology: Theory, Models, and Simulation*, Chapman & Hall, New York, 2008.
[33] J. MUÑOZ-MATUTE, D. PARDO, AND V. M. CALO, *Exploiting the Kronecker product structure of φ-functions in exponential integrators*, Internat. J. Numer. Methods Engrg., 123 (2022), pp. 2142–2161.
[34] Q. NIE, F. Y. M. WAN, Y.-T. ZHANG, AND X.-F. LIU, *Compact integration factor methods in high spatial dimensions*, J. Comput. Phys., 227 (2008), pp. 5238–5255.
[35] J. NIESEN AND W. M. WRIGHT, *Algorithm 919: a Krylov subspace algorithm for evaluating the ϕ-functions appearing in exponential integrators*, ACM Trans. Math. Software, 38 (2012), Paper No. 22, 19 pages.
[36] NVIDIA CORPORATION, *cuBLAS documentation*, 2024. https://docs.nvidia.com/cuda/cublas/index.html
[37] J. SASTRE, J. IBÁÑEZ, AND E. DEFEZ, *Boosting the computation of the matrix exponential*, Appl. Math. Comput., 340 (2019), pp. 206–220.
[38] B. SKAFLESTAD AND W. M. WRIGHT, *The scaling and modified squaring method for matrix functions related to the exponential*, Appl. Numer. Math., 59 (2009), pp. 783–799.
[39] Z. XIANYI, W. QIAN, AND Z. YUNQUAN, *Model-driven level 3 BLAS performance optimization on Loongson 3A processor*, in 2012 IEEE 18th international conference on parallel and distributed systems, IEEE Conference Proceedings, Los Alamitos, 2012, pp. 684–691.
[40] L. ZHU, L. JU, AND W. ZHAO, *Fast high-order compact exponential time differencing Runge-Kutta methods for second-order semilinear parabolic equations*, J. Sci. Comput., 67 (2016), pp. 1043–1065.