J. Approx. Softw., Vol. 1, No. 1 (2024) 1

Journal of Approximation Software

Direction splitting of φ-functions in exponential integrators for d-dimensional problems in Kronecker form

M. Caliari[1,⋆] and F. Cassini[1]

[1] *Department of Computer Science, University of Verona, Italy*

Received: 05/04/2023 – Published: 23/07/2024
Communicated by: R. Cavoretto and A. De Rossi

Abstract

In this manuscript, we propose an efficient, practical and easy-to-implement way to approximate actions of φ-functions for matrices with d-dimensional Kronecker sum structure in the context of exponential integrators up to second order. The method is based on a direction splitting of the involved matrix functions, which lets us exploit the highly efficient level 3 BLAS for the actual computation of the required actions in a μ-mode fashion. The approach has been successfully tested on two- and three-dimensional problems with various exponential integrators, resulting in a consistent speedup with respect to a technique designed to approximate actions of φ-functions for Kronecker sums.

Keywords: exponential integrators, μ-mode, φ-functions, direction splitting, Kronecker form (MSC2020: 65F60, 65L04, 65M20)

1 Introduction

The problem of computing actions of exponential and exponential-like functions with Kronecker sum structure received a lot of attention in the last years [6, 8, 10, 14, 21, 22, 25]. Indeed, the efficient approximation of such quantities allows to effectively employ exponential integrators for the time integration of large stiff systems of Ordinary Differential Equations (ODEs). More in detail, we suppose to work with the following system of ODEs

$$\begin{cases} \boldsymbol{u}'(t) = K\boldsymbol{u}(t) + \boldsymbol{g}(t,\boldsymbol{u}(t)), & t > 0, \\ \boldsymbol{u}(0) = \boldsymbol{u}_0. \end{cases} \tag{1a}$$

⋆ Corresponding author: marco.caliari@univr.it

The stiff part is represented by the matrix $K \in \mathbb{C}^{N\times N}$ which has d-dimensional Kronecker sum structure, i.e.,

$$K = A_d \oplus A_{d-1} \oplus \cdots \oplus A_1 = \sum_{\mu=1}^{d} A_{\otimes\mu}, \quad A_{\otimes\mu} = I_d \otimes \cdots \otimes I_{\mu+1} \otimes A_\mu \otimes I_{\mu-1} \otimes \cdots \otimes I_1. \quad (1b)$$

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 generic 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 standard Kronecker product of matrices, while $\oplus$ is employed for the Kronecker sum of matrices. Finally, we refer to system (1) as a system in *Kronecker form* or with *Kronecker sum structure*.

This kind of systems naturally arises in many contexts. For example, if $d = 2$, such a structure appears in constant coefficient matrix Riccati differential equations (see, for instance, Reference [1, Ch. 3])

$$\begin{cases} \boldsymbol{U}'(t) = A_1\boldsymbol{U}(t) + \boldsymbol{U}(t)A_2^{\mathsf{T}} + C + \boldsymbol{U}(t)B\boldsymbol{U}(t), \\ \boldsymbol{U}(0) = \boldsymbol{U}_0, \end{cases} \quad (2)$$

where $\boldsymbol{U}(t) \in \mathbb{C}^{n_1\times n_2}$, $B \in \mathbb{C}^{n_2\times n_1}$, and $C \in \mathbb{C}^{n_1\times n_2}$. Indeed, using the properties of the Kronecker product [31], we can rewrite equivalently such a matrix equation as a system of ODEs in Kronecker form (1), i.e.,

$$\begin{cases} \boldsymbol{u}'(t) = ((I_2 \otimes A_1) + (A_2 \otimes I_1))\boldsymbol{u}(t) + \operatorname{vec}(C + \boldsymbol{U}(t)B\boldsymbol{U}(t)), \\ \boldsymbol{u}(0) = \operatorname{vec}(\boldsymbol{U}_0), \end{cases} \quad (3)$$

where vec is the operator which stacks the columns of the input matrix in a single vector.

Systems with Kronecker sum structure often arise also when applying the method of lines to approximate numerically the solution of a Partial Differential Equation (PDE) defined on a tensor product domain and appropriate boundary conditions. Indeed, after semidiscretization in space of well-known parabolic equations such as Allen–Cahn, Brusselator, Gray–Scott, advection–diffusion–reaction [8, 10] or Schrödinger equations [6], we obtain a large stiff system of ODEs in form (1).

Once system (1) is given, many techniques can be employed to numerically integrate it in time, and in particular we are interested in the application of exponential integrators [19]. In fact, they are a prominent way to perform the required task since they enjoy favorable stability properties that make them suitable to work in the stiff regime. These kinds of schemes require the computation of the action of the matrix exponential and of exponential-like matrix functions (the so-called φ-functions) on vectors. They are defined, for a generic matrix $X \in \mathbb{C}^{N\times N}$, as

$$\varphi_0(X) = \mathrm{e}^X, \quad \varphi_\ell(X) = \int_0^1 \frac{\theta^{\ell-1}}{(\ell-1)!}\mathrm{e}^{(1-\theta)X}d\theta, \quad \ell > 0, \quad (4a)$$

and their Taylor series expansion is given by

$$\varphi_\ell(X) = \sum_{i=0}^{\infty} \frac{X^i}{(i+\ell)!}, \quad \ell \geq 0. \quad (4b)$$

When the size of X allows, it is common in practice to approximate such matrix functions by means of diagonal Padé approximations [2, 5, 30] or via polynomial approximations [12, 20,

29]. On the other hand, when X is large sized, this approach is computationally unfeasible, and many algorithms have been developed to perform directly the action of φ-functions on vectors. We mention, among the others, Krylov-based techniques [16, 23, 26], direct polynomial methods [3, 7, 11, 20], and hybrid techniques [9]. When X is in fact a matrix K with Kronecker sum structure (1b), it is possible to exploit this information to compute more efficiently the action of the φ-functions on a vector. Indeed, let us consider $\ell = 0$, so that $\varphi_0(K) = \mathrm{e}^K$. Then, it is easy to see [10] that computing

$$\boldsymbol{e} = \mathrm{e}^K \boldsymbol{v} = \mathrm{e}^{A_d \oplus A_{d-1} \oplus \cdots \oplus A_1} \boldsymbol{v} = \left(\mathrm{e}^{A_d} \otimes \mathrm{e}^{A_{d-1}} \otimes \cdots \otimes \mathrm{e}^{A_1} \right) \boldsymbol{v} \tag{5}$$

is mathematically equivalent to the *tensor formulation*

$$\boldsymbol{E} = \boldsymbol{V} \times_1 \mathrm{e}^{A_1} \times_2 \cdots \times_d \mathrm{e}^{A_d}. \tag{6}$$

Here, $\boldsymbol{E}$ and $\boldsymbol{V}$ are order-d tensors of size $n_1 \times \cdots \times n_d$ that satisfy $\mathrm{vec}(\boldsymbol{E}) = \boldsymbol{e}$ and $\mathrm{vec}(\boldsymbol{V}) = \boldsymbol{v}$, respectively, while vec is the operator which stacks the columns of the input tensor into a suitable single column vector. The symbol $\times_\mu$ denotes the tensor–matrix product along the mode μ, which is also known as μ-mode product, and the computation of consecutive μ-mode products (as it happens in formula (6)) is usually referred to as *Tucker operator*. Notice that the element $e_{i_1 \ldots i_d}$ of the tensor $\boldsymbol{E}$ turns out to be

$$e_{i_1 \ldots i_d} = \sum_{j_d=1}^{n_d} \cdots \sum_{j_1=1}^{n_1} v_{j_1 \ldots j_d} \prod_{\mu=1}^{d} e^{\mu}_{i_\mu j_\mu}, \quad 1 \le i_\mu \le n_\mu, \tag{7}$$

being $e^{\mu}_{i_\mu j_\mu}$ the generic element of e^{A_μ}. Although formulas (5), (6), and (7) are mathematically equivalent, the direct usage of both formulas (5) and (7) is much less efficient than formula (6), which is implemented by exploiting the highly performance level 3 BLAS after computing the *small* sized matrix exponentials e^{A_μ}. Indeed, for instance, formula (6) in two dimensions requires two matrix-matrix products, as it reduces to $\mathrm{e}^{A_1} \boldsymbol{V} \left(\mathrm{e}^{A_2} \right)^{\mathsf{T}}$, while in the d-dimensional case it requires d level 3 BLAS calls. This technique led to the so-called μ-mode integrator [6], and has been successfully used to integrate in time semidiscretizations of advection–diffusion–reaction and Schrödinger equations, eventually in combination with a splitting scheme. In particular, it is reported a consistent speedup with respect to state-of-the-art techniques to compute the action of the matrix exponential on a vector, as well as a very good scaling when performing GPUs simulations. We invite a reader interested in more details and applications of the Tucker operator to check References [6, 10].

When computing actions of φ-function of higher order, i.e., $\varphi_\ell(K)\boldsymbol{v}$ with $\ell > 0$, the last equality in formula (5) does not hold anymore. In Reference [8] the authors propose an approach to overcome this difficulty, by developing a method based on the application of a quadrature formula to the integral definition of the φ-functions (4a). In fact, it requires an action of the matrix exponential for each quadrature point, which is performed by a Tucker operator. In this way, it is possible to compute the required action of φ-functions at a given tolerance. The technique, which has been named PHIKS, has been developed for arbitrary dimension d and is designed to compute not only φ-functions applied to a vector but also linear combinations of actions of φ-functions. In addition the desired quantities can be made available simultaneously at suitable different time scales. These features allow to implement high stiff order exponential integrators, such as exponential Runge–Kutta schemes, in a more efficient way compared to the usage of state-of-the-art techniques to compute combinations of actions of φ-functions. Another very recent method based on quadrature rules applied to formula (4a) is presented

in Reference [14], where the technique is described only in dimension $d \leq 3$ for actions of single φ-functions at a given time scale. Other approaches for the action of φ-functions for matrices with Kronecker sum structure are available in the literature. We mention for instance Reference [25], whose algorithm is based on the solution of Sylvester equations and is currently limited to dimension $d = 2$. Another way to approximate the action of φ-functions of the Sylvester operator $A_1\boldsymbol{V} + \boldsymbol{V}A_2^{\mathsf{T}}$ or the Lyapunov operator $A\boldsymbol{V} + \boldsymbol{V}A^{\mathsf{T}}$ for the solution of Riccati differential equations, possibly in the context of low-rank approximation, is presented in References [21, 22].

In this manuscript we propose an alternative way to approximate $\varphi_\ell(K)\boldsymbol{v}$, with $\ell > 0$ and K a matrix with d-dimensional Kronecker sum structure, in the context of exponential integrators up to second order. The approach, that we call PHISPLIT, is based on a direction splitting of the matrix φ-functions of K, which generates an approximation error compatible with the one of the time marching numerical scheme. The evaluation of the required actions is performed in a μ-mode fashion by means of a *single* Tucker operator for each φ-function, exploiting the highly efficient level 3 BLAS. After recalling some popular exponential integrators in Section 2, we describe in Section 3 the proposed technique, as well as how to employ it to implement the just mentioned exponential schemes. Then, in Section 4 we present some numerical experiments that show the effectiveness of PHISPLIT, and we finally draw some conclusions in Section 5.

2 Recall of some exponential integrators up to order two

When numerically integrating stiff semilinear ODEs in form (1), a prominent approach is to use exponential integrators [19]. For convenience of the reader, we report here (for simplicity in a constant time step size scenario) a possible derivation of the exponential schemes that will be employed later in the numerical experiments of Section 4.

The starting point is the variation-of-constants formula

$$\begin{aligned} \boldsymbol{u}(t_{n+1}) &= \mathrm{e}^{\tau K}\boldsymbol{u}(t_n) + \int_{t_n}^{t_{n+1}} \mathrm{e}^{(t_{n+1}-s)K}\boldsymbol{g}(s,\boldsymbol{u}(s))ds \\ &= \mathrm{e}^{\tau K}\boldsymbol{u}(t_n) + \tau\int_0^1 \mathrm{e}^{(1-\theta)\tau K}\boldsymbol{g}(t_n+\tau\theta,\boldsymbol{u}(t_n+\tau\theta))d\theta \end{aligned} \tag{8}$$

which expresses the analytical solution of system (1a) at time $t_{n+1} = t_n + \tau$, where τ is the time step size. If we approximate the integral with the rectangle left rule, we get the scheme

$$\boldsymbol{u}_{n+1} = \mathrm{e}^{\tau K}\boldsymbol{u}_n + \tau\mathrm{e}^{\tau K}\boldsymbol{g}(t_n,\boldsymbol{u}_n) = \mathrm{e}^{\tau K}(\boldsymbol{u}_n + \tau\boldsymbol{g}(t_n,\boldsymbol{u}_n)), \tag{9}$$

which is known as Lawson–Euler scheme (see Reference [4, Sec. A.1.1]). It is of order one, exact if $\boldsymbol{g}(t,\boldsymbol{u}(t))$ is null. The linear part of system (1a) is solved exactly and thus no restriction on the time step size due to the stiffness is necessary. Instead, if the trapezoidal quadrature rule is applied to the integral in equation (8), we get the approximation

$$\boldsymbol{u}(t_{n+1}) \approx \mathrm{e}^{\tau K}\boldsymbol{u}(t_n) + \frac{\tau}{2}\left(\mathrm{e}^{\tau K}\boldsymbol{g}(t_n,\boldsymbol{u}(t_n)) + \boldsymbol{g}(t_{n+1},\boldsymbol{u}(t_{n+1}))\right).$$

An explicit time marching scheme is then obtained by creating an intermediate stage $\boldsymbol{u}_{n2}$ which approximates $\boldsymbol{u}(t_{n+1})$ in the right hand side by the Lawson–Euler scheme (9). Overall, we get

$$\begin{aligned} \boldsymbol{u}_{n2} &= \mathrm{e}^{\tau K}(\boldsymbol{u}_n + \tau\boldsymbol{g}(t_n,\boldsymbol{u}_n)), \\ \boldsymbol{u}_{n+1} &= \mathrm{e}^{\tau K}\left(\boldsymbol{u}_n + \frac{\tau}{2}\boldsymbol{g}(t_n,\boldsymbol{u}_n)\right) + \frac{\tau}{2}\boldsymbol{g}(t_{n+1},\boldsymbol{u}_{n2}), \end{aligned} \tag{10}$$

which is a Lawson method of order two, also known in literature as Lawson2b (see Reference [4, Sec. A.1.6]).

A different approach to the approximation of the integral in formula (8) leads to the so-called exponential Runge–Kutta methods. Indeed, if we approximate only the nonlinear function $\boldsymbol{g}(t_n+\tau\theta,\boldsymbol{u}(t_n+\tau\theta))$ by $\boldsymbol{g}(t_n,\boldsymbol{u}(t_n))$, by using the definition of φ_1 function in equation (4a) we get the scheme

$$\boldsymbol{u}_{n+1} = \mathrm{e}^{\tau K}\boldsymbol{u}_n + \tau\varphi_1(\tau K)\boldsymbol{g}(t_n,\boldsymbol{u}_n),$$

which can be equivalently rewritten as

$$\boldsymbol{u}_{n+1} = \boldsymbol{u}_n + \tau\varphi_1(\tau K)(K\boldsymbol{u}_n + \boldsymbol{g}(t_n,\boldsymbol{u}_n)) \tag{11}$$

and is known as exponential Euler (or Nørsett–Euler, see Reference [4, Sec. A.2.1]). It is a first order scheme, exact if $\boldsymbol{g}(t,\boldsymbol{u}(t))$ is constant. Another possibility is to interpolate $\boldsymbol{g}(t_n+\tau\theta,\boldsymbol{u}(t_n+\tau\theta))$ with a polynomial of degree one in θ at 0 and 1, thus obtaining the approximation

$$\boldsymbol{u}(t_{n+1}) \approx \mathrm{e}^{\tau K}\boldsymbol{u}(t_n) + \tau\int_0^1 \mathrm{e}^{(1-\theta)\tau K}(\theta\boldsymbol{g}(t_{n+1},\boldsymbol{u}(t_{n+1})) + (1-\theta)\boldsymbol{g}(t_n,\boldsymbol{u}(t_n)))d\theta.$$

By taking a stage $\boldsymbol{u}_{n2}$ which approximates $\boldsymbol{u}(t_{n+1})$ in the right hand side by the exponential Euler scheme, and using the definitions of φ_1 and φ_2 functions in formula (4a), we obtain the second order exponential Runge–Kutta scheme (also known in literature as ETD2RK, see Reference [4, Sec. A.2.5])

$$\begin{aligned}\boldsymbol{u}_{n2} &= \boldsymbol{u}_n + \tau\varphi_1(\tau K)(K\boldsymbol{u}_n + \boldsymbol{g}(t_n,\boldsymbol{u}_n)),\\ \boldsymbol{u}_{n+1} &= \boldsymbol{u}_{n2} + \tau\varphi_2(\tau K)(\boldsymbol{g}(t_{n+1},\boldsymbol{u}_{n2}) - \boldsymbol{g}(t_n,\boldsymbol{u}_n)).\end{aligned} \tag{12}$$

Finally, we consider the Rosenbrock–Euler method (see Reference [19, Ex. 2.20]) which, in the autonomous case, can be obtained from the application of the exponential Euler scheme to the linearized differential equation

$$\boldsymbol{u}'(t) = \left(\underbrace{K + \frac{\partial \boldsymbol{g}}{\partial \boldsymbol{u}}(\boldsymbol{u}_n)}_{K_n}\right)\boldsymbol{u}(t) + \left(\boldsymbol{g}(\boldsymbol{u}(t)) - \frac{\partial \boldsymbol{g}}{\partial \boldsymbol{u}}(\boldsymbol{u}_n)\boldsymbol{u}(t)\right),$$

where K_n is the Jacobian evaluated at $\boldsymbol{u}_n$. The resulting scheme is

$$\boldsymbol{u}_{n+1} = \boldsymbol{u}_n + \tau\varphi_1(\tau K_n)(K\boldsymbol{u}_n + \boldsymbol{g}(\boldsymbol{u}_n)). \tag{13}$$

It is a second order method and, in contrast to all the methods presented above, it requires the evaluation of a different matrix function $\varphi_1(\tau K_n)$ at *each time step*. The extension to non-autonomous systems is straightforward, see Reference [19, Ex. 2.21].

Remark 2.1. We considered here only a selected number of exponential integrators which require the action of φ-functions. Other exponential-type schemes of first or second order could benefit from the μ-mode splitting technique for computing φ-functions of Kronecker sums that we present in this work. We mention, among the others, exponential multistep schemes [13], corrected splitting schemes [15], low-regularity schemes [28], and Magnus integrators for linear time dependent coefficient non-homogeneous equations [17].

3 Direction splitting of φ-functions

As mentioned in the introduction, we suppose that we are dealing with a matrix K with Kronecker sum structure (1b), and we are interested in approximating efficiently $\varphi_\ell(\tau K)\boldsymbol{v}$, $\boldsymbol{v} \in \mathbb{C}^N$, $\tau \in \mathbb{R}$, in the context of exponential integrators. In particular, we know that by employing a scheme of order p, we make a local error $\mathscr{O}(\tau^{p+1})$, being τ the (constant) time step size. Hence, if the integrator requires to compute a quantity of the form $\tau^q \varphi_\ell(\tau K)$, with $q > 0$, it is sufficient to approximate $\varphi_\ell(\tau K)$ with an error $\mathscr{O}(\tau^{p+1-q})$ to preserve the order of convergence. For our schemes of interest, i.e., the ones presented in the previous section, we make use of the following result.

Theorem 3.1. *Let K be a matrix with d-dimensional Kronecker sum structure (1b). Then, for $\ell > 0$, we have*

$$\varphi_\ell(\tau K) = (\ell!)^{d-1} \left(\varphi_\ell(\tau A_d) \otimes \varphi_\ell(\tau A_{d-1}) \otimes \cdots \otimes \varphi_\ell(\tau A_1)\right) + \mathscr{O}(\tau^2). \tag{14}$$

Proof. For compactness of presentation, we employ the following notation

$$X_d \otimes X_{d-1} \otimes \cdots \otimes X_1 = \bigotimes_{\mu=d}^{1} X_\mu, \quad X_\mu \in \mathbb{C}^{n_\mu \times n_\mu}.$$

Then, by using the Taylor expansion of the φ_ℓ function (4b) and the properties of the Kronecker product (see Reference [31] for a comprehensive review) we obtain

$$\begin{aligned}
(\ell!)^{d-1} \bigotimes_{\mu=d}^{1} \varphi_\ell(\tau A_\mu) &= (\ell!)^{d-1} \bigotimes_{\mu=d}^{1} \left(\frac{I_\mu}{\ell!} + \frac{\tau A_\mu}{(\ell+1)!} + \mathscr{O}(\tau^2)\right) \\
&= (\ell!)^{d-1} \left(\frac{1}{(\ell!)^d} \bigotimes_{\mu=d}^{1} I_\mu + \frac{\tau}{(\ell!)^{d-1}(\ell+1)!} \sum_{\mu=1}^{d} A_{\otimes\mu} + \mathscr{O}(\tau^2)\right) \\
&= \frac{I}{\ell!} + \frac{\tau K}{(\ell+1)!} + \mathscr{O}(\tau^2) \\
&= \varphi_\ell(\tau K) + \mathscr{O}(\tau^2),
\end{aligned}$$

where I is the identity matrix of size $N \times N$. □

Formula (14) allows for an efficient μ-mode based implementation, similarly to the matrix exponential case (6). Indeed, given an order-d tensor $\boldsymbol{V}$, if we define the tensor formulation

$$\boldsymbol{P}_\ell^{(2)}(\boldsymbol{V}) = \left((\ell!)^{d-1}\boldsymbol{V}\right) \times_1 \varphi_\ell(\tau A_1) \times_2 \varphi_\ell(\tau A_2) \times_3 \cdots \times_d \varphi_\ell(\tau A_d), \tag{15}$$

we have

$$\varphi_\ell(\tau K)\boldsymbol{v} = \boldsymbol{p}_\ell^{(2)}(\boldsymbol{v}) + \mathscr{O}(\tau^2),$$

where $\boldsymbol{v} = \mathrm{vec}(\boldsymbol{V})$ and $\boldsymbol{p}_\ell^{(2)}(\boldsymbol{v}) = \mathrm{vec}(\boldsymbol{P}_\ell^{(2)}(\boldsymbol{V}))$.

This is precisely the formulation that we propose to employ in the above exponential integrators when actions of φ-functions of a matrix with Kronecker sum structure are required. From now on, we refer to this technique as the PHISPLIT approach. Notice that, after the computation of the *small* sized matrix functions $\varphi_\ell(\tau A_\mu)$, with $\mu = 1, \ldots, d$, a single Tucker operator is required to evaluate the approximation.

3.1 Evaluation of small sized matrix φ-functions

The matrices A_μ have a much smaller size compared to K, and the corresponding matrix φ-functions can be directly computed without much effort. In particular, for φ_0 (i.e., the exponential function), we employ the most popular technique for generic matrices, which is based on a diagonal rational Padé approximation coupled with a scaling and squaring algorithm (see Reference [2]). This procedure is encoded in the internal MATLAB function `expm`. Different algorithms that could be used as well, based on Taylor approximations of the matrix exponential, can be found in References [12, 29].

For the computation of higher order matrix φ-functions we rely on a quadrature formula applied to the integral definition (4a). For a generic matrix $X \in \mathbb{C}^{N\times N}$, we have then

$$\varphi_\ell(X) \approx \sum_{i=1}^{q} w_i \mathrm{e}^{(1-\theta_i)X} \frac{\theta_i^{\ell-1}}{(\ell-1)!}, \quad \ell > 0. \tag{16}$$

In order to avoid an impractically large number of quadrature points, we couple the procedure with a modified scaling and squaring algorithm (see Reference [30]). In fact, we scale the original matrix X by 2^s, where s is a natural number defined so that $\|X/2^s\|_1 < 1$, we approximate $\varphi_\ell(X/2^s)$ by means of formula (16) and we recover $\varphi_\ell(X)$ by the recurrence

$$\varphi_\ell(2z) = \frac{1}{2^\ell}\left[\mathrm{e}^z \varphi_\ell(z) + \sum_{k=1}^{\ell} \frac{\varphi_k(z)}{(\ell-k)!}\right]. \tag{17}$$

In order to compute the needed matrix exponentials, we again employ the internal MATLAB function `expm`. Notice that the squaring of φ_ℓ also requires the evaluation of all the φ_j functions, for $0 < j < \ell$. In particular, for the first squaring step, we compute themselves by formula (16) with $\ell = j$ and using the *same* set of matrix exponentials already available for the quadrature procedure of $\varphi_\ell(X/2^s)$. For all the subsequent squaring steps, we use formula (17) itself with $\ell = j$. As a consequence of this procedure, the computation of a single matrix function φ_ℓ makes available all the matrix functions φ_j, with $0 \le j \le \ell$. In practice, for the quadrature we employ the Gauss–Legendre–Lobatto formula, which allows for high precision with a moderate number of quadrature nodes. Moreover, since it uses the endpoints of the quadrature interval, we make use of the matrix exponential $\mathrm{e}^{X/2^s}$ ($\theta_1 = 0$), which is also required for the subsequent squaring procedure, and we avoid generating the last matrix exponential since $\theta_q = 1$. The overall procedure is implemented in MATLAB language in our function `phiquad`.

Alternatively, for the computation of the matrix φ-functions, it is possible to employ the MATLAB routine `phipade` (see Reference [5]), whose algorithm is based on a rational Padé approximation coupled with the squaring formula (17). Another recent technique that employs a polynomial Taylor approximation instead of rational Padé one is presented in Reference [20].

3.2 Practical implementation of the exponential integrators

The implementation of the Lawson methods introduced in Section 2, which require just actions of matrix exponentials, does not suffer from any direction splitting error, thanks to the equivalence between formulas (5) and (6). In particular, the tensor formulation of Lawson–Euler is

$$\boldsymbol{U}_{n+1} = (\boldsymbol{U}_n + \tau \boldsymbol{G}(t_n, \boldsymbol{U}_n)) \times_1 \mathrm{e}^{\tau A_1} \times_2 \cdots \times_d \mathrm{e}^{\tau A_d}, \tag{18}$$

while the Lawson2b scheme is given by

$$\begin{aligned} \boldsymbol{U}_{n2} &= (\boldsymbol{U}_n + \tau \boldsymbol{G}(t_n, \boldsymbol{U}_n)) \times_1 \mathrm{e}^{\tau A_1} \times_2 \cdots \times_d \mathrm{e}^{\tau A_d}, \\ \boldsymbol{U}_{n+1} &= \left(\boldsymbol{U}_n + \frac{\tau}{2}\boldsymbol{G}(t_n, \boldsymbol{U}_n)\right) \times_1 \mathrm{e}^{\tau A_1} \times_2 \cdots \times_d \mathrm{e}^{\tau A_d} + \frac{\tau}{2}\boldsymbol{G}(t_{n+1}, \boldsymbol{U}_{n2}). \end{aligned} \tag{19}$$

The remaining exponential integrators transform as follows. First of all, the action of the matrix K on $\boldsymbol{u}_n$ is computed in tensor form as

$$\sum_{\mu=1}^{d} \left(\boldsymbol{U}_n \times_\mu A_\mu\right) \tag{20}$$

without explicitly assembling the matrix K (see Reference [9]). The exponential Euler PHISPLIT method is

$$\boldsymbol{U}_{n+1} = \boldsymbol{U}_n + \tau \left(\sum_{\mu=1}^{d} (\boldsymbol{U}_n \times_\mu A_\mu) + \boldsymbol{G}(t_n, \boldsymbol{U}_n) \right) \times_1 \varphi_1(\tau A_1) \times_2 \cdots \times_d \varphi_1(\tau A_d). \tag{21}$$

Notice that an alternative version, which we will not consider in this work but could be more appropriate for the cases in which the contribute of $\boldsymbol{G}(t_n, \boldsymbol{U}_n)$ is particularly small, is

$$\boldsymbol{U}_{n+1} = \boldsymbol{U}_n \times_1 \mathrm{e}^{\tau A_1} \times_2 \cdots \times_d \mathrm{e}^{\tau A_d} + \tau \boldsymbol{G}(t_n, \boldsymbol{U}_n) \times_1 \varphi_1(\tau A_1) \times_2 \cdots \times_d \varphi_1(\tau A_d), \tag{22}$$

that is in fact an exact formula if $\boldsymbol{G}(t, \boldsymbol{U}(t))$ is null. The ETD2RK PHISPLIT scheme becomes

$$\begin{aligned} \boldsymbol{U}_{n2} &= \boldsymbol{U}_n + \tau \left(\sum_{\mu=1}^{d} (\boldsymbol{U}_n \times_\mu A_\mu) + \boldsymbol{G}(t_n, \boldsymbol{U}_n) \right) \times_1 \varphi_1(\tau A_1) \times_2 \cdots \times_d \varphi_1(\tau A_d), \\ \boldsymbol{U}_{n+1} &= \boldsymbol{U}_{n2} + \tau \left(2^{d-1} \left(\boldsymbol{G}(t_{n+1}, \boldsymbol{U}_{n2}) - \boldsymbol{G}(t_n, \boldsymbol{U}_n) \right) \right) \times_1 \varphi_2(\tau A_1) \times_2 \cdots \times_d \varphi_2(\tau A_d). \end{aligned} \tag{23}$$

Finally, concerning the exponential Rosenbrock–Euler method for autonomous systems, we assume that the Jacobian K_n can be written as a Kronecker sum, i.e.,

$$K_n = K + \frac{\partial \boldsymbol{g}}{\partial \boldsymbol{u}}(\boldsymbol{u}_n) = J_d(\boldsymbol{U}_n) \oplus J_{d-1}(\boldsymbol{U}_n) \oplus \cdots \oplus J_1(\boldsymbol{U}_n).$$

Therefore the exponential Rosenbrock–Euler PHISPLIT method is

$$\boldsymbol{U}_{n+1} = \boldsymbol{U}_n + \tau \left(\sum_{\mu=1}^{d} (\boldsymbol{U}_n \times_\mu A_\mu) + \boldsymbol{G}(\boldsymbol{U}_n) \right) \times_1 \varphi_1(\tau J_1(\boldsymbol{U}_n)) \times_2 \cdots \times_d \varphi_1(\tau J_d(\boldsymbol{U}_n)). \tag{24}$$

4 Numerical experiments

In this section we present numerical experiments to validate the proposed approach PHISPLIT. In particular, we will consider a two-dimensional example from linear quadratic control and a three-dimensional example which models an advection–diffusion–reaction equation. To perform the time marching, we will employ the exponential integrators of Section 2 as described in Section 3.2 for the PHISPLIT version.

As term of comparison, we will consider the approximation of actions of φ-functions for matrices with Kronecker sum structure using PHIKS[1] [8]. This algorithm operates in tensor

[1] https://github.com/caliarim/phiks

formulation using μ-mode products, too, but it requires an input tolerance, which we take proportional to the local temporal order of the method and to the norm of the current solution. The proportionality constant is chosen so that the accuracy of the routine does not affect the integration error.

To compute all the relevant tensor operations, i.e., Tucker operators and μ-mode products, we use the functions contained in the package KronPACK[2]. Moreover, to compute the needed matrix φ-functions, we employ the internal MATLAB function `expm` (for φ_0) and the function `phiquad` (for φ_ℓ, $\ell > 0$), as presented in Section 3.1. In terms of hardware, we run all the experiments employing an Intel® Core™ i7-10750H CPU with six physical cores and 16GB of RAM. As a software, we use MathWorks MATLAB® R2022a.

4.1 Example of code usage

All the codes to reproduce the following numerical experiments, together with our implementation of PHISPLIT and the function `phiquad`, can be found in a maintained GitHub repository[3]. They are written in MATLAB language and they are fully compatible with GNU Octave. As an example, we show in Code 1 how to perform a full integration of problem (1) by the ETD2RK PHISPLIT method (23) with constant time step size.

Code 1: Implementation of ETD2RK PHISPLIT

```
% compute once and for all phi1 and phi2 functions and store them
for mu = 1:d
    [phi_store.phi{1:2,mu}] = phiquad(tau * A{mu},2);
end

% time integration
U = U0;
t = 0;
for i = 1:m
    Gn = G(t,U);
    P1 = phisplit(tau,A,kronsumv(U,A) + Gn,1,phi_store); % phi1 approximation
    Un2 = U + tau * P1; % intermediate stage
    P2 = phisplit(tau,A,G(t + tau,U2) - Gn,2,phi_store); % phi2 approximation
    U = Un2 + tau * P2; % updated solution
    t = t + tau;
end
```

Here, we simply notice that the required φ-functions are computed once and for all before time integration by the function `phiquad`, the function `phisplit` computes approximation (15), while the function `kronsumv` of the KronPACK package realizes formula (20).

4.2 Linear quadratic control

We present in this section a classical example from linear quadratic control (see, for instance, References [24, 27]). We are interested in the minimization over the scalar control $v(t) \in \mathbb{R}$ of the functional

$$\mathscr{J}(v) = \frac{1}{2}\int_0^T \left(\alpha s(t)^2 + v(t)^2\right) dt$$

subject to the constraints

$$\begin{aligned} &w'(t) = Aw(t) + bv(t), \quad w(0) = w_0, \\ &s(t) = cw(t). \end{aligned}$$

[2] https://github.com/caliarim/KronPACK
[3] https://github.com/caliarim/phisplit

Here $w(t) \in \mathbb{R}^{n\times 1}$ is a column vector containing the state variables, $s(t) \in \mathbb{R}$ represents the scalar output, $A \in \mathbb{R}^{n\times n}$ is the system matrix, $b \in \mathbb{R}^{n\times 1}$ is the system column vector, $c \in \mathbb{R}^{1\times n}$ is a row vector, and $\alpha \in \mathbb{R}^{+}$ is a positive scalar.

Then, the solution of the constrained optimization problem is determined by the optimal control

$$v^*(t) = -b^{\mathsf{T}}\boldsymbol{U}(t)w(t),$$

where $\boldsymbol{U}(t) \in \mathbb{R}^{n\times n}$ satisfies the symmetric Riccati differential equation

$$\begin{cases} \boldsymbol{U}'(t) = A^{\mathsf{T}}\boldsymbol{U}(t) + \boldsymbol{U}(t)A + C + \boldsymbol{U}(t)B\boldsymbol{U}(t), \\ \boldsymbol{U}(0) = \boldsymbol{Z}, \end{cases} \tag{25}$$

where $C = \alpha c^{\mathsf{T}}c$ and $B = -bb^{\mathsf{T}}$ (see Reference [1, Ch. 4] for a comprehensive introduction to the subject). Here $\boldsymbol{Z} \in \mathbb{R}^{n\times n}$ is a matrix containing all zeros entries. Clearly, equation (25) is in form (2), which in turn can be seen as a problem with two-dimensional Kronecker sum structure (3) and integrated efficiently by means of the techniques described in Section 3. Notice also that the solution of equation (25) converges to a steady state determined by the algebraic Riccati equation

$$A^{\mathsf{T}}\boldsymbol{U}(t) + \boldsymbol{U}(t)A + C + \boldsymbol{U}(t)B\boldsymbol{U}(t) = 0. \tag{26}$$

For our numerical experiment, similarly to what previously done in the literature [21, 22, 24, 27], we take $A \in \mathbb{R}^{\hat{n}^2\times\hat{n}^2}$ as the matrix obtained by the discretization with second order centered finite differences of the operator

$$\partial_{xx} + \partial_{yy} - 10x\partial_x - 100y\partial_y \tag{27}$$

on the domain $[0,1]^2$ with homogeneous Dirichlet boundary conditions. Moreover, the components b_k of the vector b are defined as

$$b_k = \begin{cases} 1 \text{ if } 0.1 < x_i \leq 0.3, \\ 0 \text{ otherwise,} \end{cases} \qquad k = i + (j-1)\hat{n}, \quad i = 1,\ldots,\hat{n}, \quad j = 1,\ldots,\hat{n},$$

while for the components c_k of the vector c we take

$$c_k = \begin{cases} 1 \text{ if } 0.7 < x_i \leq 0.9, \\ 0 \text{ otherwise,} \end{cases} \qquad k = i + (j-1)\hat{n}, \quad i = 1,\ldots,\hat{n}, \quad j = 1,\ldots,\hat{n}.$$

Here x_i represents the ith (inner) grid point along the x direction. Finally, we set $\alpha = 100$.

For the temporal integration of equation (25) we use the exponential Rosenbrock–Euler method, already employed in References [21, 22], and reported in formula (13) (see formula (24) for the PHISPLIT version). In fact, the Jacobian matrix of system (25) has the following Kronecker sum structure

$$K_n = I \otimes (A^{\mathsf{T}} + \boldsymbol{U}_n B) + (A + B\boldsymbol{U}_n)^{\mathsf{T}} \otimes I,$$

where I is the identity matrix of size $n \times n$, with $n = \hat{n}^2$. We remark that the exponential Rosenbrock–Euler PHISPLIT method requires at *each time step* to evaluate the matrix function $\varphi_1(\tau(A^{\mathsf{T}} + \boldsymbol{U}_n B))$, to compute the action $K\boldsymbol{u}_n$ and to perform one Tucker operator. We will employ also the second order exponential Runge–Kutta method ETD2RK, reported in formula (12) and presented in PHISPLIT sense in formula (23). Although each time step of this

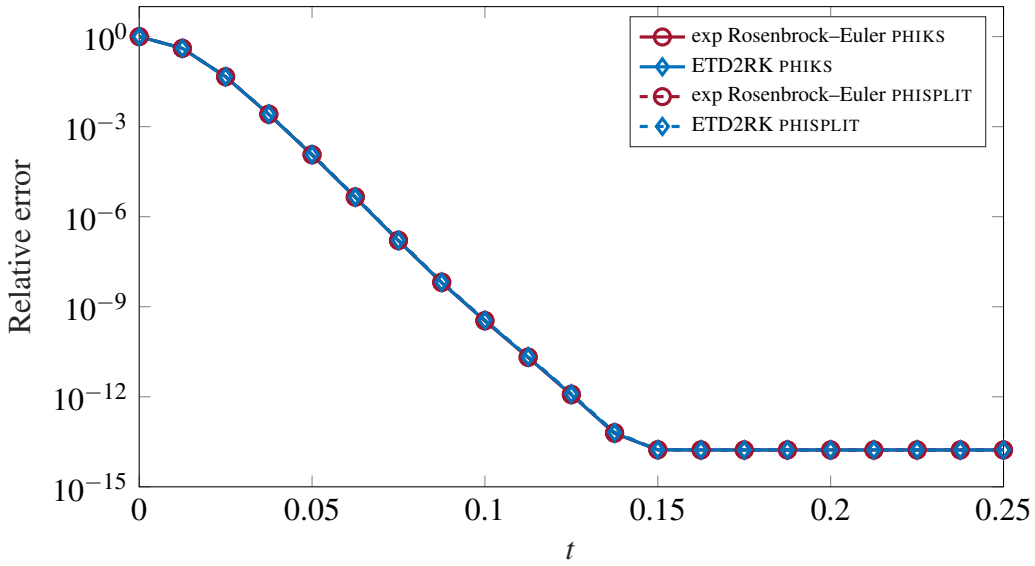

Figure 1: Convergence of the exponential Rosenbrock–Euler and of the ETD2RK methods, both in PHIKS and in PHISPLIT variants, to the steady state of Riccati differential equation (25). Here $\hat{n} = 20$, the integrators have been employed with 200 time steps, and the relative errors, measured in the Frobenius norm with respect to the solution of algebraic Riccati equation (26), are displayed each 10th time step.

integrator requires two Tucker operators plus the action $K\boldsymbol{u}_n$ for the PHISPLIT version, in a constant time step size implementation the needed matrix functions $\varphi_1(\tau A^{\mathsf{T}})$ and $\varphi_2(\tau A^{\mathsf{T}})$ can be computed once and for all at the beginning.

First of all, we verify the implementation of the involved exponential integrators for a long term simulation, i.e., until reaching the steady state. For this experiment, we employ $\hat{n} = 20$ inner discretization points for the x and the y variables. As confirmed by the plot in Figure 1, around time 0.15 the methods, both in their PHIKS and PHISPLIT implementation, approach the solution of equation (26), which is obtained with the MATLAB function `icare` from the Control System Toolbox.

exp Rosenbrock–Euler PHIKS	steps	10	20	30	40	50
	order	–	2.11	2.06	2.05	2.03
ETD2RK PHIKS	steps	7	14	21	28	35
	order	–	2.08	2.05	2.03	2.03
exp Rosenbrock–Euler PHISPLIT	steps	30	65	100	135	170
	order	–	2.05	2.03	2.02	2.02
ETD2RK PHISPLIT	steps	30	65	100	135	170
	order	–	2.05	2.03	2.02	2.02

Table 1: Number of time steps and observed convergence rates for the time integration of Riccati differential equation (25) up to final time $T = 0.025$, with different exponential integrators and $\hat{n} = 30$. The achieved errors and the wall-clock times are displayed in Figure 2.

Then, we compare the performances of the integrators for the solution of equation (25) with $\hat{n} = 30$ and final time $T = 0.025$. All methods are run with different time step sizes

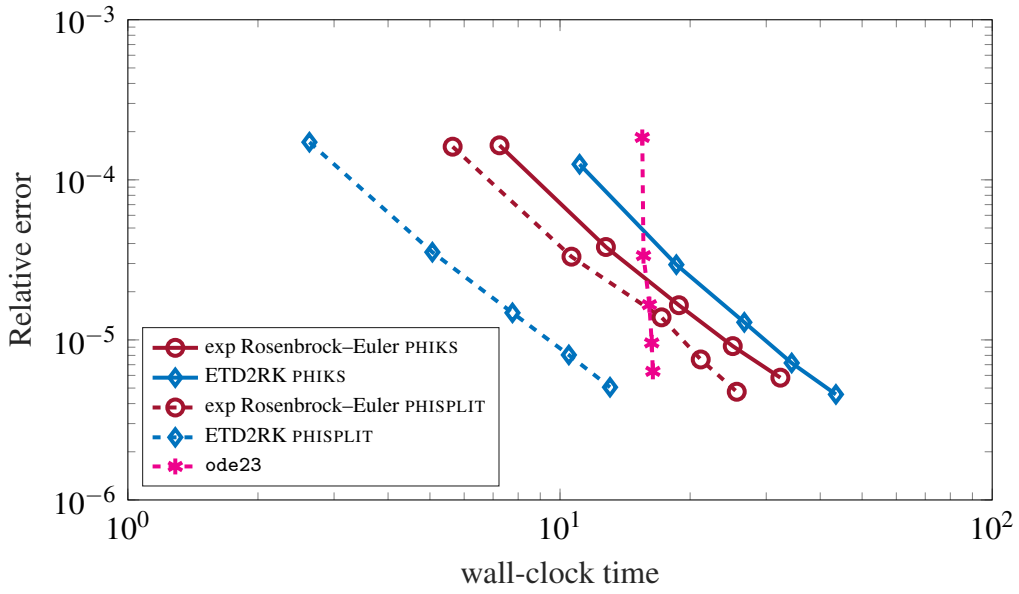

Figure 2: Achieved errors in the Frobenius norm and wall-clock times in seconds for the solution of Riccati differential equation (25) up to final time $T = 0.025$, with different integrators and $\hat{n} = 30$. The number of time steps for each exponential method is reported in Table 1. The input tolerances (both absolute and relative) for `ode23` are 5e−3, 1e−3, 1e−4, 5.5e−5, and 5e−5.

in such a way to reach comparable relative errors with respect to a reference solution. The number of time steps for each method and simulation, together with the numerically observed convergence rate, is reported in Table 1. All the methods appear to be of second order, as expected. In Figure 2 we report the relative errors and the corresponding wall-clock times of the simulations. Here, we also include the performance of the built-in MATLAB function `ode23`. This is an explicit Runge–Kutta method of order three with variable step size, suggested for not stringent tolerances and for moderately stiff problems. In fact, it turned out to be the fastest routine in the ODE suite to reach accuracies in the same range of the other methods. We notice first of all that the exponential Rosenbrock–Euler method is always faster than ETD2RK in the PHIKS implementation, that is with the action of matrix functions computed at a precision that does not affect the temporal error (see the discussion at the beginning of the section). On the other hand, the two implementations with PHISPLIT are always faster compared with their PHIKS counterparts, although they require a larger number of time steps to reach a comparable accuracy. Moreover, the ETD2RK method turns out to be faster with respect to the exponential Rosenbrock–Euler method. This is mainly due to the fact that the matrix functions in the Runge–Kutta case are computed only once before the time marching. This method is in fact at least twice as fast as the other exponential methods and faster than `ode23`, which anyway shows a good performance for the most stringent tolerances.

Finally, we repeat the same experiment with $\hat{n} = 40$. The results are presented in Table 2 and in Figure 3. The global behavior is similar with respect to the previous case, although the speed-ups of the PHISPLIT implementations with respect to their PHIKS counterparts is noticeably larger. In fact, ETD2RK PHISPLIT is still the most efficient method.

Remark 4.1. The discretization of the operator (27) has itself a Kronecker sum structure. Hence,

exp Rosenbrock–Euler PHIKS	steps	15	30	45	60	75
	order	–	2.10	2.06	2.04	2.03
ETD2RK PHIKS	steps	10	20	30	40	50
	order	–	2.12	2.07	2.05	2.04
exp Rosenbrock–Euler PHISPLIT	steps	30	65	100	135	170
	order	–	2.05	2.03	2.02	2.02
ETD2RK PHISPLIT	steps	30	65	100	135	170
	order	–	2.05	2.03	2.02	2.02

Table 2: Number of time steps and observed convergence rates for the time integration of Riccati differential equation (25) up to final time $T = 0.025$, with different exponential integrators and $\hat{n} = 40$. The achieved errors and the wall-clock times are displayed in Figure 3.

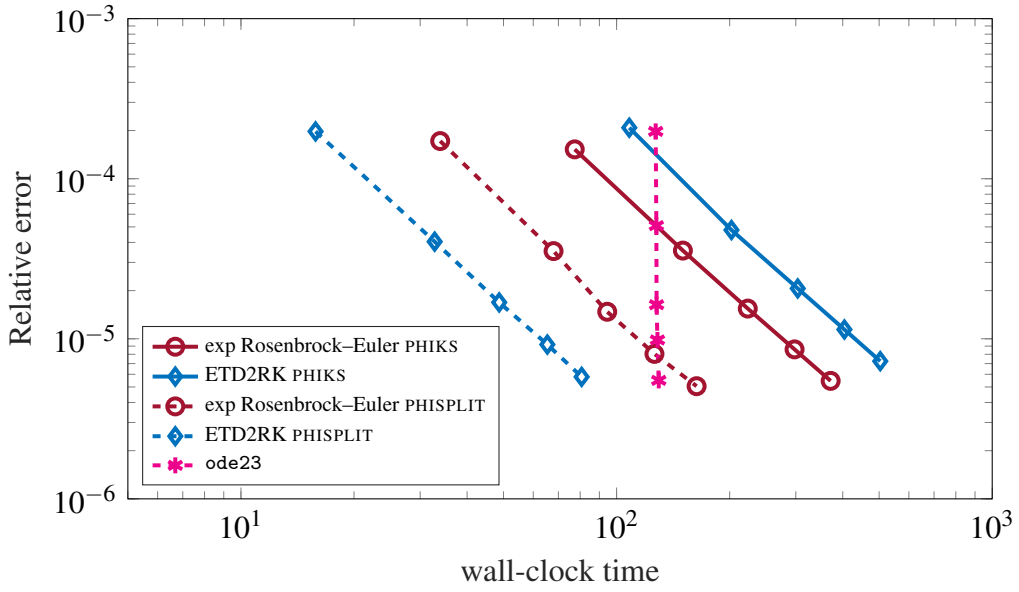

Figure 3: Achieved errors in the Frobenius norm and wall-clock times in seconds for the solution of Riccati differential equation (25) up to final time $T = 0.025$, with different integrators and $\hat{n} = 40$. The number of time steps for each exponential method is reported in Table 2. The input tolerances (both absolute and relative) for `ode23` are 3e−3, 4e−4, 2.3e−4, 9.5e−5, and 8.7e−5.

it is possible to write equation (25) (in vector formulation for simplicity of exposition) as

$$\begin{cases} \boldsymbol{u}'(t) = K\boldsymbol{u}(t) + \boldsymbol{g}(\boldsymbol{u}(t)), \\ \boldsymbol{u}(0) = \boldsymbol{z}, \end{cases}$$

where $\boldsymbol{g}$ and $\boldsymbol{z}$ are the vectorizations of the nonlinearity and of $\boldsymbol{Z}$, respectively, and K has the form

$$K = I \otimes I \otimes I \otimes D_1^{\mathsf{T}} + D_2^{\mathsf{T}} \otimes I \otimes I \otimes I.$$

Here I is an identity matrix of size $\hat{n} \times \hat{n}$ and $D_1 \in \mathbb{R}^{\hat{n}\times\hat{n}}$ and $D_2 \in \mathbb{R}^{\hat{n}\times\hat{n}}$ the discretizations of the operators $\partial_{xx} - 10x\partial_x$ and $\partial_{yy} - 100y\partial_y$, respectively. In the context of temporal integration with exponential Runge–Kutta schemes, we could then use both the PHIKS and the PHISPLIT

approaches with the even smaller sized matrices D_1 and D_2, forming then the approximations at every time steps using Tucker operators with order-4 tensors. However, as this is just possible because of the specific form of the operator (27), we do not pursue this approach here.

4.3 Advection–diffusion–reaction

We now consider the semidiscretization in space of the following three-dimensional evolutionary Advection–Diffusion–Reaction (ADR) equation (see Reference [8])

$$\begin{cases} \partial_t u(t,x_1,x_2,x_3) = \varepsilon\Delta u(t,x_1,x_2,x_3) + \alpha(\partial_{x_1}+\partial_{x_2}+\partial_{x_3})u(t,x_1,x_2,x_3) \\ \qquad\qquad + g(t,x_1,x_2,x_3,u(t,x_1,x_2,x_3)), \\ u_0(x_1,x_2,x_3) = 64x_1(1-x_1)x_2(1-x_2)x_3(1-x_3). \end{cases} \tag{28}$$

The nonlinear function g is given by

$$g(t,x_1,x_2,x_3,u(t,x_1,x_2,x_3)) = \frac{1}{1+u(t,x_1,x_2,x_3)^2} + \Psi(t,x_1,x_2,x_3),$$

where $\Psi(t,x_1,x_2,x_3)$ is such that the analytical solution of the equation is

$$u(t,x_1,x_2,x_3) = \mathrm{e}^t u_0(x_1,x_2,x_3).$$

The problem is solved up to final time $T = 1$ in the domain $[0,1]^3$ and completed with homogeneous Dirichlet boundary conditions. The remaining parameters are set to $\varepsilon = 0.75$ and $\alpha = 0.1$. By semidiscretizing in space with second order centered finite differences, we obtain a system of type (1) with K having three-dimensional Kronecker sum structure, where A_μ approximates $\varepsilon\partial_{x_\mu x_\mu} + \alpha\partial_{x_\mu}$. We first perform simulations with $n_1 = 40$, $n_2 = 41$, and $n_3 = 42$ inner discretization points for the x_1, x_2 and x_3 variables, respectively. The temporal integration is performed with four methods: the Lawson–Euler scheme (9), the exponential Euler method (11), the Lawson2b scheme (10) and the ETD2RK method (12) (see Section 3.2 for their practical implementation and the PHISPLIT versions). In particular, concerning the Lawson schemes, the needed matrix exponentials $\exp(\tau A_\mu)$, with $\mu = 1,2,3$, are computed once and for all at the beginning. Then, one and two Tucker operators per time step, for the first order and second order scheme, respectively, are required to form the approximations during the temporal integration. Concerning the PHISPLIT implementation of exponential Euler and ETD2RK, again we compute once and for all the needed matrix functions $\varphi_1(\tau A_\mu)$ and $\varphi_2(\tau A_\mu)$ before starting the temporal integration, and we then combine them suitably at each time step. This operation requires a single Tucker operator for the first order scheme and two for the second order one, as for the aforementioned Lawson schemes, plus the action $K\boldsymbol{u}_n$. The number of time steps for each method, for both the PHISPLIT and PHIKS implementations, is reported in Table 3, while the reached relative errors and the wall-clock times are summarized in Figure 4. First of all, we notice that all the methods show the expected convergence rate, reported in Table 3 as well. The Lawson–Euler method and the exponential Euler scheme in its PHIKS implementation (see top plot of Figure 4) perform equally well, even if the former requires much more time steps. Overall, the exponential Euler method in its PHISPLIT variant is roughly 10 times faster to reach the highest accuracy in this experiment. If we consider the second order methods (bottom plot of Figure 4), we observe that the Lawson2b scheme needs much more wall-clock time to reach the same level of accuracy of the other methods, and overall the most efficient method is ETD2RK in the PHISPLIT variant. In this plot we report also

Lawson–Euler	steps	800	8800	16800	24800	32800
	order	–	1.00	1.00	1.00	1.00
exp Euler PHIKS	steps	50	450	850	1250	1650
	order	–	1.03	1.00	1.00	1.00
exp Euler PHISPLIT	steps	50	450	850	1250	1650
	order	–	1.03	1.01	1.00	1.00
Lawson2b	steps	1500	5500	9500	13500	17500
	order	–	1.96	1.99	2.00	2.00
ETD2RK PHIKS	steps	20	80	140	200	260
	order	–	1.94	1.97	1.98	1.99
ETD2RK PHISPLIT	steps	40	140	240	340	440
	order	–	2.10	2.04	2.03	2.02

Table 3: Number of time steps and observed convergence rates for the time integration of the semidiscretization of the ADR equation (28) up to final time $T = 1$, with different exponential integrators. Here we considered $n_1 = 40$, $n_2 = 41$ and $n_3 = 42$ inner space discretization points for the x_1, x_2 and x_3 variables, respectively. The achieved errors and the wall-clock times are displayed in Figure 4.

Lawson–Euler	steps	800	8800	16800	24800	32800
	order	–	1.00	1.00	1.00	1.00
exp Euler PHIKS	steps	50	450	850	1250	1650
	order	–	1.02	1.00	1.00	1.00
exp Euler PHISPLIT	steps	50	450	850	1250	1650
	order	–	1.03	1.01	1.00	1.00
Lawson2b	steps	3000	4500	6000	7500	9000
	order	–	1.79	1.87	1.92	1.94
ETD2RK PHIKS	steps	20	80	140	200	260
	order	–	1.94	1.97	1.98	1.99
ETD2RK PHISPLIT	steps	40	140	240	340	440
	order	–	2.10	2.04	2.03	2.02

Table 4: Number of time steps and observed convergence rates for the time integration of the semidiscretization of the ADR equation (28) up to final time $T = 1$, with different exponential integrators. Here we considered $n_1 = 80$, $n_2 = 81$ and $n_3 = 82$ inner space discretization points for the x_1, x_2 and x_3 variables, respectively. The achieved errors and the wall-clock times are displayed in Figure 5.

the results obtained with the internal MATLAB `ode23t` implicit integrator. It is an implementation of the trapezoidal rule with variable step size, which is suggested for stiff problems at low accuracies. Nevertheless, it performs worse than the considered exponential integrators.

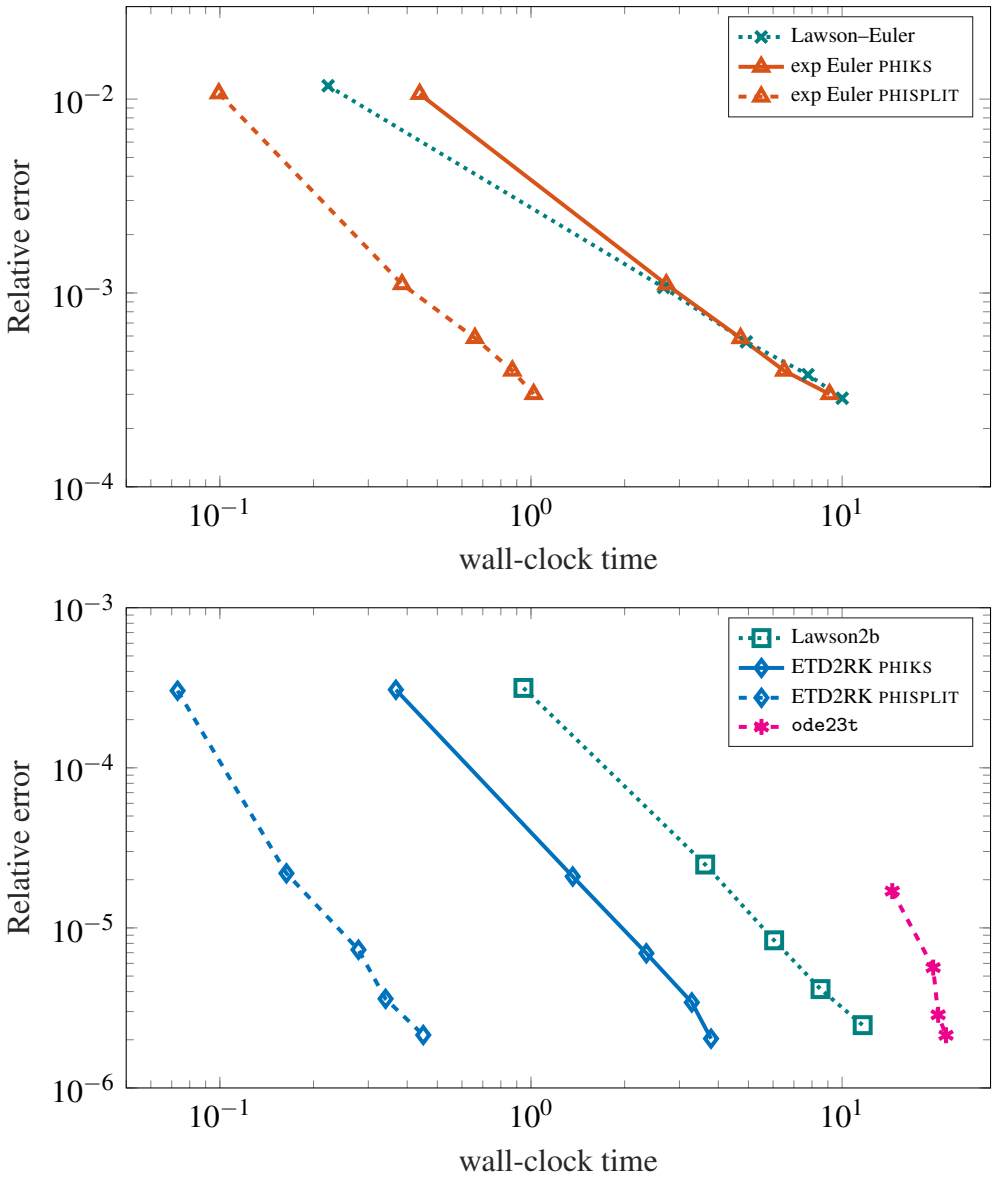

Figure 4: Achieved errors in the infinity norm and wall-clock times in seconds for the solution of the semidiscretization of the ADR equation (28) up to final time $T = 1$, with different exponential integrators of order one (top) and order two (bottom). Here we considered $n_1 = 40$, $n_2 = 41$ and $n_3 = 42$ inner space discretization points for the x_1, x_2 and x_3 variables, respectively. The number of time steps for each exponential method is reported in Table 3. The input tolerances (both absolute and relative) for `ode23t` are 8e−3, 4e−5, 1e−5, and 5e−6.

Finally, we repeat the experiment with $n_1 = 80$, $n_2 = 81$, and $n_3 = 82$ inner discretization points for the x_1, x_2 and x_3 variables, respectively. The number of time steps for each method is reported in Table 4, while the relative errors and the wall-clock times are summarized in Figure 5. Again, we notice that all the methods show the expected convergence rate, reported in Table 4 as well. In particular, for large time step sizes, the Lawson2b method suffers from an order reduction. This is expected, as in these cases the problem is more stiff, and schemes which employ just the exponential function are affected by this phenomenon (see, for instance, Reference [18]). Then, from Figure 5 we observe that the PHISPLIT approach is in any case the most efficient among all the methods and techniques considered, with an increasing speedup for

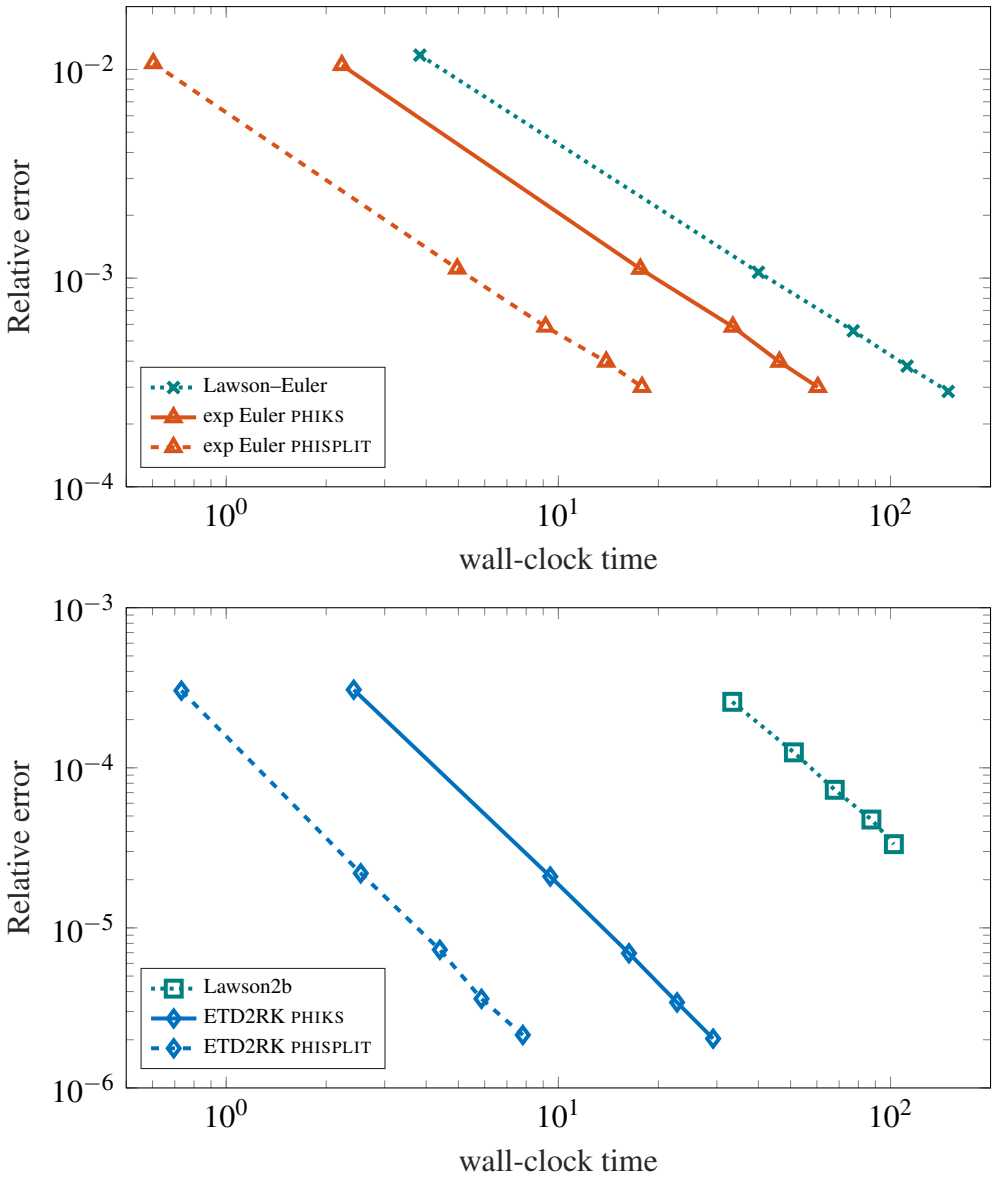

Figure 5: Achieved errors in the infinity norm and wall-clock times in seconds for the solution of the semidiscretization of the ADR equation (28) up to final time $T = 1$, with different exponential integrators of order one (top) and order two (bottom). Here we considered $n_1 = 80$, $n_2 = 81$ and $n_3 = 82$ inner space discretization points for the x_1, x_2 and x_3 variables, respectively. The number of time steps for each method is reported in Table 4.

more stringent accuracies. More in detail, compared with its PHIKS counterparts, the PHISPLIT implementations are roughly 3.5 time faster, even if (in general) they require more time steps to reach a comparable accuracy. On the other hand, the Lawson schemes perform poorly. This is mainly due to the requirement of a large number of time steps to reach the accuracy of the other methods, which is particularly evident for the second order schemes (see bottom of Table 4 and Figure 5). Moreover, while ETD2RK in its PHISPLIT variant reached the most stringent accuracy in less than 10 seconds, Lawson2b was not able to reach an accuracy 10 times larger in 100 seconds. Hence, we decided to stop the simulation with this integrator with a larger number of time steps. Finally, concerning the internal MATLAB ODE suite, none of the methods was able to output a solution within 10 minutes.

5 Conclusions

In this paper, we presented how it is possible to efficiently approximate actions of φ-functions for matrices with d-dimensional Kronecker sum structure using a μ-mode based approach. The technique, that we call PHISPLIT, is suitable when integrating initial valued ordinary differential equations with exponential integrators up to second order. It is based on an inexact direction splitting of the matrix functions involved in the time marching schemes which preserves the order of the method. The effectiveness and superiority of the approach, with respect to another technique to compute actions of φ-functions in Kronecker form, has been shown on a two-dimensional problem from linear quadratic control and on a three-dimensional advection–diffusion–reaction equation, using a variety of exponential integrators. Interesting future developments would be to generalize the approach for higher order integrators and performing GPU simulations with the PHISPLIT technique, possibly in single and/or half precision (which are compatible with the magnitude of the errors of the temporal integration), for different problems from science and engineering.

Acknowledgments

The authors would like to acknowledge partial support from the Program Ricerca di Base 2019 No. RBVR199YFL of the University of Verona entitled “Geometric Evolution of Multi Agent Systems”.

Declaration of interests

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

References

[1] H. Abou-Kandil, G. Freiling, V. Ionescu, and G. Jank. *Matrix Riccati Equations in Control and Systems Theory*, (Birkhäuser Springer, Basel2003). URL https://doi.org/10.1007/978-3-0348-8081-7

[2] A. H. Al-Mohy and N. J. Higham. *A New Scaling and Squaring Algorithm for the Matrix Exponential*. SIAM J. Matrix Anal. Appl., vol. 31, 3, (2010), 970–989. URL https://doi.org/10.1137/09074721X

[3] A. H. Al-Mohy and N. J. Higham. *Computing the Action of the Matrix Exponential, with an Application to Exponential Integrators*. SIAM J. Sci. Comput., vol. 33, 2, (2011), 488–511. URL https://doi.org/10.1137/100788860

[4] H. Berland, B. Skaflestad, and W. M. Wright. *EXPINT — A MATLAB package for exponential integrators*. Tech. Rep. 4, Norwegian University of Science and Technology (2005). URL https://www.math.ntnu.no/preprint/numerics/2005/N4-2005.pdf

[5] H. Berland, B. Skaflestad, and W. M. Wright. *EXPINT—A MATLAB Package for Exponential Integrators*. ACM Trans. Math. Softw., vol. 33, 1, (2007), Article 4. URL https://doi.org/10.1145/1206040.1206044

[6] 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., vol. 455, (2022), 110989. URL https://doi.org/10.1016/j.jcp.2022.110989

[7] M. Caliari, F. Cassini, and F. Zivcovich. *Approximation of the matrix exponential for matrices with a skinny field of values*. BIT Numer. Math., vol. 60, 4, (2020), 1113–1131. URL https://doi.org/10.1007/s10543-020-00809-0

[8] M. Caliari, F. Cassini, and F. Zivcovich. *A μ-mode approach for exponential integrators: actions of φ-functions of Kronecker sums*. arXiv preprint arXiv:2210.07667. URL https://doi.org/10.48550/arXiv.2210.07667

[9] M. Caliari, F. Cassini, and F. Zivcovich. *BAMPHI: Matrix-free and transpose-free action of linear combinations of φ-functions from exponential integrators*. J. Comput. Appl. Math., vol. 423, (2023), 114973. URL https://doi.org/10.1016/j.cam.2022.114973

[10] M. Caliari, F. Cassini, and F. Zivcovich. *A μ-mode BLAS approach for multidimensional tensor-structured problems*. Numer. Algorithms, vol. 92, 4, (2023), 2483–2508. URL https://doi.org/10.1007/s11075-022-01399-4

[11] M. Caliari, P. Kandolf, A. Ostermann, and S. Rainer. *The Leja Method Revisited: Backward Error Analysis for the Matrix Exponential*. SIAM J. Sci. Comput., vol. 38, 3, (2016), A1639–A1661. URL https://doi.org/10.1137/15M1027620

[12] M. Caliari and F. Zivcovich. *On-the-fly backward error estimate for matrix exponential approximation by Taylor algorithm*. J. Comput. Appl. Math., vol. 346, (2019), 532–548. URL https://doi.org/10.1016/j.cam.2018.07.042

[13] S. M. Cox and P. C. Matthews. *Exponential Time Differencing for Stiff Systems*. J. Comput. Phys., vol. 176, 2, (2002), 430–455. URL https://doi.org/10.1006/jcph.2002.6995

[14] 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., vol. 67, (2023), 101966. URL https://doi.org/10.1016/j.jocs.2023.101966

[15] L. Einkemmer and A. Ostermann. *Overcoming Order Reduction in Diffusion-Reaction Splitting. Part 1: Dirichlet Boundary Conditions*. SIAM J. Sci. Comput., vol. 37, 3, (2015), A1577–A1592. URL https://doi.org/10.1137/140994204

[16] S. Gaudreault, G. Rainwater, and M. Tokman. *KIOPS: A fast adaptive Krylov subspace solver for exponential integrators*. J. Comput. Phys., vol. 372, (2018), 236–255. URL https://doi.org/10.1016/j.jcp.2018.06.026

[17] C. González, A. Ostermann, and M. Thalhammer. *A second-order Magnus-type integrator for nonautonomous parabolic problems*. J. Comput. Appl. Math., vol. 189, 1–2, (2006), 142–156. URL https://doi.org/10.1016/j.cam.2005.04.036

[18] M. Hochbruck, J. Leibold, and A. Ostermann. *On the convergence of Lawson methods for semilinear stiff problems*. Numer. Math., vol. 145, (2020), 553–580. URL https://doi.org/10.1007/s00211-020-01120-4

[19] M. Hochbruck and A. Ostermann. *Exponential integrators*. Acta Numer., vol. 19, (2010), 209–286. URL https://doi.org/10.1017/S0962492910000048

[20] D. Li, S. Yang, and J. Lan. *Efficient and accurate computation for the φ-functions arising from exponential integrators*. Calcolo, vol. 59, (2022), Article 11. URL https://doi.org/10.1007/s10092-021-00453-2

[21] D. Li, X. Zhang, and R. Liu. *Exponential integrators for large-scale stiff Riccati differential equations*. J. Comput. Appl. Math., vol. 389, (2021), 113360. URL https://doi.org/10.1016/j.cam.2020.113360

[22] D. Li, Y. Zhang, and X. Zhang. *Computing the Lyapunov operator φ-functions, with an application to matrix-valued exponential integrators*. Appl. Numer. Math., vol. 182, (2022), 330–343. URL https://doi.org/10.1016/j.apnum.2022.08.009

[23] 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., vol. 376, (2019), 817–837. URL https://doi.org/10.1016/j.jcp.2018.10.018

[24] H. Mena, A. Ostermann, L.-M. Pfurtscheller, and C. Piazzola. *Numerical low-rank approximation of matrix differential equations*. J. Comput. Appl. Math., vol. 340, (2018), 602–614. URL https://doi.org/10.1016/j.cam.2018.01.035

[25] J. Muñoz-Matute, D. Pardo, and V. M. Calo. *Exploiting the Kronecker product structure of φ–functions in exponential integrators*. Int. J. Numer. Methods Eng., vol. 123, 9, (2022), 2142–2161. URL https://doi.org/10.1002/nme.6929

[26] J. Niesen and W. M. Wright. *Algorithm 919: A Krylov Subspace Algorithm for Evaluating the ϕ-Functions Appearing in Exponential Integrators*. ACM Trans. Math. Softw., vol. 38, 3, (2012), 1–19. URL https://doi.org/10.1145/2168773.2168781

[27] T. Penzl. *LYAPACK: A MATLAB Toolbox for Large Lyapunov and Riccati Equations, Model Reduction Problems, and Linear–Quadratic Optimal Control Problems. Users' Guide (Version 1.0)* (2000). URL https://www.netlib.org/lyapack/guide.pdf

[28] F. Rousset and K. Schratz. *A General Framework of Low Regularity Integrators*. SIAM J. Numer. Anal., vol. 59, 3, (2021), 1735–1768. URL https://doi.org/10.1137/20M1371506

[29] J. Sastre, J. Ibáñez, and E. Defez. *Boosting the computation of the matrix exponential*. Appl. Math. Comput., vol. 340, (2019), 206–220. URL https://doi.org/10.1016/j.amc.2018.08.017

[30] B. Skaflestad and W. M. Wright. *The scaling and modified squaring method for matrix functions related to the exponential*. Appl. Numer. Math., vol. 59, 3–4, (2009), 783–799. URL https://doi.org/10.1016/j.apnum.2008.03.035

[31] C. F. Van Loan. *The ubiquitous Kronecker product*. J. Comput. Appl. Math., vol. 123, 1–2, (2000), 85–100. URL https://doi.org/10.1016/S0377-0427(00)00393-9