Exploiting the Kronecker product structure of $\varphi-$functions in Exponential Integrators

Judit Muñoz-Matute[a,b], David Pardo[c,a,d], Victor M. Calo[e]

[a] *Basque Center for Applied Mathematics (BCAM), Bilbao, Spain*
[b] *Oden Institute for Computational Engineering and Sciences, The University of Texas at Austin, Austin, USA*
[c] *University of the Basque Country (UPV/EHU), Leioa, Spain*
[d] *IKERBASQUE, Basque Foundation for Science, Bilbao, Spain*
[e] *School of Electrical Engineering, Computing and Mathematical Sciences, Curtin University, Perth, WA 6845, Australia*

Abstract

Exponential time integrators are well-established discretization methods for time semilinear systems of ordinary differential equations. These methods use $\varphi-$functions, which are matrix functions related to the exponential. This work introduces an algorithm to speed up the computation of the $\varphi-$function action over vectors for 2D matrices expressed as a Kronecker sum. For that, we present an auxiliary exponential-related matrix function that we express using Kronecker products of 1D matrices. We exploit state-of-the-art implementations of $\varphi-$functions to compute this auxiliary function's action and then recover the original $\varphi-$action by solving a Sylvester equation system. Our approach allows us to save memory and solve exponential integrators of 2D+time problems in a fraction of the time traditional methods need. We analyze the method's performance considering different linear operators and with the nonlinear 2D+time Allen-Cahn equation.

Keywords: $\varphi-$functions, Kronecker sum, exponential integrators, Sylvester equation, Finite Element Method, semilinear parabolic problems

1. Introduction

Exponential integrators [22, 24, 31] are a class of time-marching methods that can deal with stiff systems of ordinary differential equations (ODEs) of the form $u'(t) = Au(t) + f(t, u(t))$. Here, A is a linear operator and f is nonlinear and possibly autonomous, i.e., it does not depend explicitly on time. These type of methods include: exponential Runge-Kutta methods [23], exponential Rosenbrock methods [8, 26], exponential multistep methods [25], exponential splitting schemes [19], and Lawson methods [27], among others. All the exponential integrator methods are built in terms of the *$\varphi-$functions*

$$\varphi_p(A) = \int_0^1 e^{(1-\theta)A} \frac{\theta^{p-1}}{(p-1)!} d\theta, \ \forall p \geq 1,$$

which are functions related to the exponential of the matrix A, let $\varphi_0(A) = e^A$. Although the idea of exponential integrators is classical, the increasing availability of robust and reliable software to compute the action of matrix functions over vectors [21] has promoted their use in the last decade in many areas, including: linear partial differential equations (PDEs) [30, 32, 33], nonlinear advection-diffusion-reaction problems [38], Vlasov-Poisson equations [12], Navier-Stokes equations [28], financial applications [36], stochastic PDEs [29], phase-field models [14], nonlinear Schödinger equation [5], and shallow water equations [15].

There exists a wide variety of methods to compute the action of exponential functions on vectors. We refer to Higham's recent catalogue [21] of the existing software. These methods include Krylov subspace algorithms [16, 34], scaling-and-squaring algorithms [1, 6, 13], truncated Taylor series [2], and Laguerre polynomials [37], among others.

Herein, we focus on systems of ODEs where matrix A can be decomposed in *Kronecker sum*[1] structure [18], i.e., $A = A^x \otimes I^y + I^x \otimes A^y$. Here, A^x and A^y are one-dimensional matrices. We can find this setting after semidiscretizing in space certain 2D+time PDEs with tensor-product finite element method (FEM) [41], isogeometric analysis (IGA) [3, 9, 11] or finite differences [35]. In this setting, finite differences and certain spectral element methods or FEM with tailored quadrature rules lead Kronecker sum structure with identity matrices. In general, for FEM and IGA the linear operator is of the form $A = M^{-1}K$ where M and K are the mass and stiffness matrices, respectively. Here, we obtain the Kronecker sum structure with identities by inverting the 1D mass matrices and, therefore, the 1D operators are of the form $A^x = (M^x)^{-1}K^x$ and $A^y = (M^y)^{-1}K^y$.

This decomposition is possible when the spatial domain is a conformal mapping of a rectangle, the material properties are constant, and the linear part of the system is the advection-reaction-diffusion operator. For brevity, we focus on rectangular domains where the coordinate axis of the domain aligns with the main axis of the rectangle. Among the applications of interest that fullfil this requirement are the Schödinger equation [6], and several phase-field models [17, 40] like the Burger's equation or the Allen-Cahn equation.

The exponential function (φ_0) preserves the Kronecker product structure [4], that is, $\varphi_0(A) = \varphi_0(A^x) \otimes \varphi_0(A^y)$. We generalize this decomposition in Kronecker products to any $\varphi-$function in order to speed up the computations of the action $\varphi_p(A)b$ where b is a vector and $p \geq 1$. For that, we proceed in the following steps:

a. we introduce an auxiliary matrix function $\Phi_p(A) = A^p\varphi_p(A)$;

b. using recurrence formulas, we express $\Phi_p(A)$ in terms of $\Phi_q(A^x)$, $\Phi_q(A^y)$ with $q \leq p$;

c. we compute the actions $\Phi_q(A^x)b$ and $\Phi_q(A^y)b$ employing the existing software; and

d. we recover the original action $\varphi_p(A)b$ from $\Phi_p(A)b$ solving Sylvester equations [20].

Our approach develops the mathematical tools to compute $\varphi_p(A)b$ in terms of the action of smaller 1D matrices using the existing software for evaluating $\varphi-$functions in the literature.

[1]The Kronecker sum is defined in the literature as $A^x \oplus A^y = A^x \otimes I^y + I^x \otimes A^y$, where I^x and I^y are identity matrices.

We analyze the method's performance for different linear and nonlinear 2D+time problems and different sizes of matrix A. We show that our approach leads to significant memory and computational time savings as we only operate with 1D matrices. The extension of the Kronecker product decomposition we propose to 3D problems is straightforward. In [7], the authors recently exploit the case $p = 0$ for both CPU and GPUs. However, our method's reliance on solving Sylvester-like systems of equations has shown the limitations of the state-of-the-art solution strategies. First, the solution of Sylvester systems with large non-symmetric matrices introduce significant numerical pollution due to the ill-conditioning errors when $p \geq 4$. We analyze this phenomenon in detail in Section 5. The ill-conditioning affects the solution, making our approach's extension to 3D difficult due to the lack of robust and reliable generalized Sylvester solvers. We partially overcome these limitations by using a direct solver for the 2D and 3D equation systems rather than solving the Sylvester system. This alternative involves assembling the 2D/3D matrix, which reduces the memory savings. In terms of computational times, this alternative is slightly slower than solving the Sylvester equations while it is still orders of magnitude faster than solving the original exponential action of the 2D/3D matrix. Nevertheless, this alternative still induces some round-off error for large matrices with higher p values. As the 3D case would require further analysis, we limit our description and analysis to 2D+time problems.

This article is organized as follows: Section 2 presents our model problem and the definition of $\varphi-$functions in exponential time integrators. In Section 3, we exemplify a semi-discretization in space by tensor product finite elements. We also provide two alternatives for exploiting the Kronecker product structure of $\varphi-$functions when we express the matrix system as a Kronecker sum. Section 4 describes the Algorithm for computing $\varphi-$functions of Kronecker sum matrices using matrix products and solving Sylvester equations. In Section 5 we discuss the implementation and analyze the performance of our method comparing with the classical routines to compute the action of $\varphi-$functions over vectors. We apply our method to different linear operators and to solve the Allen-Cahn equation. In Section 6, we summarize the conclusions and future work. Finally, in Appendix A we summarize the definition and main properties of the Kronecker product of matrices and in Appendix B we introduce the Sylvester equations.

2. Model problem and time discretization

We consider the following class of parabolic semilinear and autonomous system of ordinary differential equations (ODEs)

$$\begin{cases} U'(t) + AU(t) = F(U(t)), \quad \forall t \in I, \\ U(0) = U_0, \end{cases} \tag{1}$$

where $I = (0, T] \subset \mathbb{R}$, A is a square matrix, and $F(\cdot)$ is a nonlinear vector.

Exponential integrators [24] are a wide-class of time integration methods to solve semi-

linear problems (1). We build them in terms of the $\varphi-$functions

$$\begin{cases} \varphi_0(A) = e^A, \\ \varphi_p(A) = \displaystyle\int_0^1 e^{(1-\theta)A} \frac{\theta^{p-1}}{(p-1)!} d\theta, \ \forall p \geq 1, \end{cases} \tag{2}$$

where e^A is

$$e^A = \sum_{k=0}^{\infty} \frac{A^k}{k!}. \tag{3}$$

An essential property of the exponential of a matrix is the following: If A and B are two commutative matrices, it holds that

$$AB = BA \Longrightarrow e^{A+B} = e^A e^B = e^B e^A. \tag{4}$$

The converse is not true in general.

These exponential functions (2) satisfy the following recurrence relation

$$\varphi_{p+1}(A) = A^{-1} \left(\varphi_p(A) - \frac{1}{p!} I \right). \tag{5}$$

We consider a partition of the time interval

$$0 = t_0 < t_1 < \ldots < t_{m-1} < t_m,$$

with time step size $\tau_k = t_k - t_{k-1}, \ \ \forall k = 1, \ldots, m$. There exist many exponential integrators that employ the functions defined in (2), such as exponential multistage or multistep methods. The simplest exponential time integrator is the exponential Euler method

$$U^k = \varphi_0(-\tau_k A) U^{k-1} + \tau_k \varphi_1(-\tau_k A) F(U^{k-1}).$$

We refer to [24] for an extensive overview of these methods and to [21] for an overview of the existing software.

3. Kronecker product structure

We focus on problems where the matrix A in (1) has a 2D Kronecker sum structure,

$$A = A^x \otimes I^y + I^x \otimes A^y. \tag{6}$$

Here, A^x and A^y are two smaller matrices than A, I^x and I^y are the identity matrices of the appropriate size, and $\otimes$ denotes the Kronecker product of matrices (see Appendix A for definitions and properties). In the following subsection, we show how we can obtain a system like (1) with matrix A of form (6) after semi-discretization in space of 2D+time partial differential equations (PDEs). Then, we exploit the Kronecker structure of the $\varphi-$functions for these matrices.

3.1. Semidiscretization in space

Let $\Omega = \Omega_x \times \Omega_y \subset \mathbb{R}^2$. We seek to solve the following problem

$$u_t + \beta \cdot \nabla u - \epsilon \Delta u + \alpha u = f(u), \text{ in } \Omega \times I, \tag{7}$$

with suitable boundary and initial conditions. Here, the linear operator is advection-diffusion-reaction with constant coefficients $\beta = (\beta_1, \beta_2)$, ϵ, and α.

We obtain a system like (1) after semidiscretizing (7) in space by a tensor-product finite element method (FEM). We multiply (7) by suitable test functions $v \in V$, where V is a Hilbert space and integrate over Ω

$$\int_\Omega (u_t v + \beta \cdot \nabla u v - \epsilon \Delta u v + \alpha u v) d\Omega = \int_\Omega f(u) v d\Omega.$$

Integrating by parts the diffusion term, we obtain

$$\int_\Omega (u_t v + \beta \cdot \nabla u v + \epsilon \nabla u \cdot \nabla v + \alpha u v) d\Omega = \int_\Omega f(u) v d\Omega. \tag{8}$$

We select tensor-product trial and test functions; that is, we approximate the solution as

$$u(x, y, t) = \sum_{i=1}^{n_x} \sum_{j=1}^{n_y} U_{ij}(t) N_i(x) N_j(y)$$

and consider test functions of the form $\{N_k(x) N_l(y),\ k = 1, \ldots, n_x,\ l = 1, \ldots, n_y\}$. Therefore, we obtain in (8)

$$(M^x \otimes M^y) U' + \left(\left(\beta_1 R^x + \epsilon K^x + \frac{\alpha}{2} M^x \right) \otimes M^y + M^x \otimes \left(\beta_2 R^y + \epsilon K^y + \frac{\alpha}{2} M^y \right) \right) U = F(U). \tag{9}$$

The entries of the matrices in (9) are

$$\begin{aligned} M^x_{ik} &= \int_{\Omega_x} N_i(x) N_k(x) dx, \\ R^x_{ik} &= \int_{\Omega_x} N_i'(x) N_k(x) dx, \\ K^x_{ik} &= \int_{\Omega_x} N_i'(x) N_k'(x) dx, \end{aligned}$$

where M^y, R^y, and K^y have similar definitions. The entries of vector $F(U)$ are

$$F(U)_{ij} = f(U_{ij}) M^x_{ik} M^y_{jl},$$

where we interpolate the given external forcing using the FEM basis functions.

We denote

$$A = (M^x \otimes M^y)^{-1} \left(\left(\beta_1 R^x + \epsilon K^x + \frac{\alpha}{2} M^x \right) \otimes M^y + M^x \otimes \left(\beta_2 R^y + \epsilon K^y + \frac{\alpha}{2} M^y \right) \right),$$

and from the Kronecker product properties, we have

$$A = (M^x)^{-1}\left(\beta_1 R^x + \epsilon K^x + \frac{\alpha}{2}M^x\right) \otimes I^y + I^x \otimes (M^y)^{-1}\left(\beta_2 R^y + \epsilon K^y + \frac{\alpha}{2}M^y\right).$$

Finally, we obtain the ODE system (1) where A has Kronecker sum structure (6) with

$$A^x = (M^x)^{-1}\left(\beta_1 R^x + \epsilon K^x + \frac{\alpha}{2}M^x\right),$$

$$A^y = (M^y)^{-1}\left(\beta_2 R^y + \epsilon K^y + \frac{\alpha}{2}M^y\right).$$

3.2. Kronecker product structure of $\varphi-$functions

We now express $\varphi-$functions using Kronecker products when the matrix A is of the form (6). First, we introduce an auxiliary exponential-related function. If we multiply (5) by A^{p+1}, we obtain

$$A^{p+1}\varphi_{p+1}(A) = A^p\varphi_p(A) - \frac{1}{p!}A^p. \tag{10}$$

We denote $\Phi_p(A) = A^p\varphi_p(A)$ and therefore relation (10) reads

$$\Phi_{p+1}(A) = \Phi_p(A) - \frac{1}{p!}A^p. \tag{11}$$

We now express $\Phi_p(A)$ in terms of $\Phi_q(A^x)$ and $\Phi_q(A^y)$ with $q \leq p$ and A defined in (6). For that, we employ the definition of the exponential of a matrix (3), the recurrence relation (11), and the Kronecker product properties defined in Appendix A.

For the lowest order case ($p = 0$), property (4) and the exponential definition (3) imply

$$e^A = e^{A^x\otimes I^y + I^x\otimes A^y} = e^{A^x\otimes I^y}e^{I^x\otimes A^y} = (e^{A^x}\otimes I^y)(I^x\otimes e^{A^y}) = e^{A^x}\otimes e^{A^y},$$

therefore, we have

$$\Phi_0(A) = \Phi_0(A^x)\otimes\Phi_0(A^y). \tag{12}$$

For $p \geq 1$, we give two alternatives in the following theorems.

Theorem 1. *Let $A = A^x \otimes I^y + I^x \otimes A^y$, it holds that*

$$\begin{aligned}\Phi_p(A) &= \Phi_p(A^x)\otimes\Phi_p(A^y)\\ &+ \Phi_p(A^x)\otimes\left(\sum_{j=0}^{p-1}\frac{(A^y)^j}{j!}\right) + \left(\sum_{i=0}^{p-1}\frac{(A^x)^i}{i!}\right)\otimes\Phi_p(A^y)\\ &+ \sum_{i=1}^{p-1}\sum_{j=p-i}^{p-1}\frac{(A^x)^i}{i!}\otimes\frac{(A^y)^j}{j!},\quad \forall p \geq 1.\end{aligned} \tag{13}$$

Proof. By induction over p (see Appendix C). □

Theorem 2. *Let $A = A^x \otimes I^y + I^x \otimes A^y$, it holds that:*

- *If $p = 2r + 1$ with $r \geq 0$, then*

$$\Phi_p(A) = \sum_{j=0}^{r} \frac{(A^x)^j}{j!} \otimes \Phi_{p-j}(A^y) + \sum_{j=0}^{r} \Phi_{p-j}(A^x) \otimes \frac{(A^y)^j}{j!} + \Phi_{r+1}(A^x) \otimes \Phi_{r+1}(A^y), \quad (14)$$

- *If $p = 2r$ with $r \geq 1$, then*

$$\Phi_p(A) = \sum_{j=0}^{r-1} \frac{(A^x)^j}{j!} \otimes \Phi_{p-j}(A^y) + \sum_{j=0}^{r-1} \Phi_{p-j}(A^x) \otimes \frac{(A^y)^j}{j!} + \Phi_r(A^x) \otimes \Phi_r(A^y), \quad (15)$$

Proof. By induction over p (see Appendix C). □

4. Algorithm

This section proposes an algorithm to compute the action $\varphi_p(A)b$ when

$$A = A^x \otimes I^y + I^x \otimes A^y$$

and b is a vector. First, we express

$$A^p \underbrace{\varphi_p(A)b}_{z} = \Phi_p(A)b. \quad (16)$$

Our goal is to solve (16) for $z = \varphi_p(A)b$ so we proceed in the two steps: (a) to compute the righ-hand side of (16) using either results from Theorem 1 or Theorem 2; and (b) to solve for z in (16) by solving p Sylvester equations. For step (a), we employ the properties of a Kronecker-product matrix acting a vector stated in Appendix A. We reshape the source vector b that comes from the semi-discretization of the full 2D problem in space into a matrix B where each column is a 1D source vector. For step (b) we employ the definition of Sylvester equations in Appendix B.

Algorithm 1 shows how to compute the action $\varphi_p(A)b$ employing the factorizations of Theorem 2. The implementation of Algorithm 1 using Theorem 1 is straightforward. Here, we reduce the computation of the action $\varphi_p(A)b$ to products of 1D matrices and the resolution of p Sylvester equations. Therefore, we achieve huge savings in both memory and computational time as we are only computing the $\varphi-$functions of one dimensional matrices. Finally, a simple strategy to accelerate and reduce the memory footprint of Algorithm 1 is to merge steps 6-7, 9-10, 13-14 and 16-17 together to avoid the storage of matrices $\Phi_p(A^x)$ and $\Phi_p(A^y)$ that are dense.

Algorithm 1: Compute the action $z = \varphi_p(A)b$ using Theorem 2

Input data: A^x, A^y, b, p, r;
`// Compute the right-hand-side`
Reshape vector b into matrix B;
$Z = 0$;
if $p = 2r + 1$ **then**
  for $j = 0, \ldots, r$ **do**
    Compute $\Phi_{p-j}(A^x) = (A^x)^{p-j}\varphi_{p-j}(A^x)$ and $\Phi_{p-j}(A^y) = (A^y)^{p-j}\varphi_{p-j}(A^y)$;
    $Z = Z + \Phi_{p-j}(A^y)B\left(\frac{(A^x)^j}{j!}\right)^T + \frac{(A^y)^j}{j!}B\Phi_{p-j}(A^x)^T$;
  end
  Compute $\Phi_{r+1}(A^x) = (A^x)^{r+1}\varphi_{r+1}(A^x)$ and $\Phi_{r+1}(A^y) = (A^y)^{r+1}\varphi_{r+1}(A^y)$;
  $Z = Z + \Phi_{r+1}(A^y)B\Phi_{r+1}(A^x)^T$;
else if $p = 2r$ **then**
  for $j = 0, \ldots, r-1$ **do**
    Compute $\Phi_{p-j}(A^x) = (A^x)^{p-j}\varphi_{p-j}(A^x)$ and $\Phi_{p-j}(A^y) = (A^y)^{p-j}\varphi_{p-j}(A^y)$;
    $Z = Z + \Phi_{p-j}(A^y)B\left(\frac{(A^x)^j}{j!}\right)^T + \frac{(A^y)^j}{j!}B\Phi_{p-j}(A^x)^T$;
  end
  Compute $\Phi_r(A^x) = (A^x)^r\varphi_r(A^x)$ and $\Phi_r(A^y) = (A^y)^r\varphi_r(A^y)$;
  $Z = Z + \Phi_r(A^y)B\Phi_r(A^x)^T$;
end
`// Solve the Sylvester equations`
for $i = 1, \ldots, p$ **do**
  $Z = sylvester(A^y, (A^x)^T, Z)$;
end
Reshape matrix Z into vector z;
Return: z;

Remark 1. *In the 3D case, we can decompose A into*

$$A = A^x \otimes I^y \otimes I^z + I^x \otimes A^y \otimes I^z + I^x \otimes I^y \otimes A^z,$$

and we have that $\varphi_0(A) = \varphi_0(A^x) \otimes \varphi_0(A^y) \otimes \varphi_0(A^z)$. Thus, the expressions given in Theorems 1 and 2 can also be extended to 3D. However, in the last step of Algorithm 1, we need to solve a generalized Sylvester equation with three terms instead of two. This question is out of the scope of this article and we will address in the future.

5. Numerical results

In this section, we first test the performance of our approach in terms of computational times. For that, we compute the action $U_1 = \varphi_p(-\tau A)U_0$ for different matrices A coming

from different spatial operators. We measure the error of our method when approximating these actions with the MATLAB routine *expmv*() from Higham et al. [2]. The latter computes the action of φ-functions on vectors for the full 2D matrix. For Algorithm 1, similarly to [7], we require to explicitly compute $\varphi-$functions of small 1D matrices so we employ the routine *phipade*() from [6]. We also employ the built-in MATLAB routine *sylvester*() to solve the Sylvester equations. Then, we solve the 2D+time Allen-Cahn equation with the exponential Euler method to test our method for approximating a non-linear transient PDE. In all examples, we discretize the space variable with a FEM with piecewise linear functions and use Lobatto quadrature to obtain diagonal mass matrices. The simulations use a 2,4 GHz Intel Core i5 CPU with 16GB of RAM using MATLAB R2017b.

5.1. Heat equation

We consider the following 2D+time heat equation

$$\frac{\partial u}{\partial t} - \epsilon \left(\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2} \right) = f(x, y, t),$$

over $\Omega = (0, 1) \times (0, 1)$ and $I = (0, 1]$. We consider homogeneous Dirichlet boundary condition, $\epsilon = 1$ and the source term corresponding to the following solution

$$u(x, y, t) = e^{-\pi^2 t} \sin(\pi x) \sin(\pi y).$$

In this case, matrix A is symmetric and has a Kronecker sum structure with A^x and A^y having the same dimension. We track the time required to compute $U_1 = \varphi_p(-\tau A)U_0$ with $p = 0, 1, 3, 5$. Here, we set $\tau = 1/16$ and U_0, the vector containing the spatial nodal points of the exact solution when $t = 0$. Table 1 shows the times needed to compute U_1 when we employ the full 2D matrix and the expressions obtained for the 1D Kronecker product. From these results, we can conclude that our approach significantly reduces the computational times. The time reduces from one hour to four seconds (i.e., a reduction factor of 792). Also, for the biggest matrix and lower order, we have a reduction factor of $27,716$.

	$p=0$			$p=1$			$p=3$			$p=5$		
Size A	2D	1D⊗1D	saving	2D	1D⊗1D	saving	2D	1D⊗1D	saving	2D	1D⊗1D	saving
49	0.01	0.016	0.6	0.01	0.04	0.3	0.01	0.05	0.2	0.01	0.08	0.2
225	0.008	0.001	8.1	0.04	0.01	4.2	0.04	0.03	1.4	0.04	0.02	1.9
961	0.011	0.003	3.3	0.04	0.02	1.9	0.04	0.01	3.1	0.04	0.01	4.1
3,969	0.35	0.001	373	0.67	0.01	45	0.49	0.01	48	0.5	0.02	24
16,129	2.16	0.003	785	10.26	0.03	384	8.57	0.04	203	9.32	0.08	117
65,025	40.33	0.006	6,341	159.81	0.06	2,868	156.87	0.19	838	163.01	0.44	370
261,121	572.75	0.021	**27,716**	3,017.82	0.46	**6,518**	2,965.42	1.61	**1,839**	3,002.46	3.79	**792**

Table 1: Computational times in seconds for $U_1 = \varphi_p(-\tau A)U_0$ with $p = 0, 1, 3, 5$ solving the full 2D problem and 1D Kronecker products for different sizes of A.

Figure 1 shows the following relative error

$$\frac{||U_1^{full} - U_1^{kron}||_2}{||U_1^{full}||_2}, \tag{17}$$

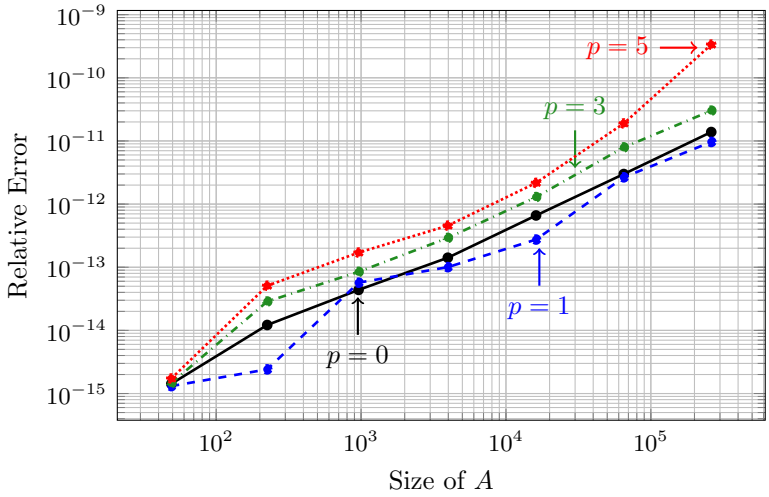

Figure 1: Relative approximation error to U_1 with $p = 0, 1, 3, 5$ for 1D Kronecker products.

where $||\cdot||_2$ denotes the L^2 error, U_1^{full} is the result we obtain with the 2D matrix and U_1^{kron} is the solution obtained with Kronecker products of 1D matrices. In this case, the error of computing the action with 1D Kronecker products is negligible. Figure 1 shows errors between high accuracy methods; thus, it becomes more difficult to achieve the same accuracy as the pollution due to condition number increases as the problem size increases.

Size A	Memory A (MB)	Size A^x, A^y	Memory A^x, A^y (MB)	saving
49	0.0037	7	0.00037	4.94
225	0.018	15	0.0017	5.26
961	0.079	31	0.0073	5.37
3,969	0.33	63	0.0303	5.44
16,129	1.35	127	0.123	5.47
65,025	5.44	255	0.496	5.48
261,121	21.88	511	1.99	5.49

Table 2: Memory in MB used to store the full 2D matrices and the 1D matrices for different problem sizes.

Table 2 shows the memory used in MB to store the 2D and 1D matrices of different sizes. Routine *expmv*() allows sparse matrices while routine *phipade*() only admits dense matrices. Therefore, the full 2D matrix is sparse, while we store the 1D matrices as dense matrices. As a result of this implementation limitation, we observe a saving factor of memory storage of 5.5; however, there is room for even larger memory savings using sparse 1D matrices.

5.2. Transient Eriksson-Johnson problem

We consider the 2D + time Eriksson-Johnson problem [33]

$$\frac{\partial u}{\partial t} + \frac{\partial u}{\partial x} - \epsilon \left(\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2} \right) = f(x, y, t),$$

over $\Omega = (-1, 0) \times (-0.5, 0.5)$ and $I = (0, 1]$. We select the data of the problem in such a way that the exact solution is

$$u(x, y, t) = Ce^{-lt} x \left(y^2 - 0.25 \right) + \frac{e^{r_1 x} - e^{r_2 x}}{e^{-r_1} - e^{-r_2}} \cos(\pi y),$$

where $r_{1,2} = \dfrac{1 \pm \sqrt{4\pi^2 \epsilon^2}}{2\epsilon}$. Therefore, we have

$$f(x, y, t) = Ce^{-lt} \left((y^2 - 0.25)(1 - lx) - 2\epsilon x \right),$$

with the following boundary and initial conditions

$$\begin{cases} \dfrac{\partial}{\partial x} u(-1, y, t) = Ce^{-lt} \left(y^2 - 0.25 \right) + \dfrac{r_1 e^{-r_1} - r_2 e^{-r_2}}{e^{-r_1} - e^{-r_2}} \cos(\pi y), \\ u(0, y, t) = u(x, -0.5, t) = u(x, 0.5, t) = 0, \\ u(x, y, 0) = Cx \left(y^2 - 0.25 \right) + \dfrac{e^{r_1 x} - e^{r_2 x}}{e^{-r_1} - e^{-r_2}} \cos(\pi y). \end{cases}$$

We set $C = 10$, $l = 4$ and $\epsilon = 10^{-2}$. For the space discretization, we select a non-uniform mesh and therefore A is a sparse non-symmetric matrix. We compute the action $U_1 = \varphi_p(-\tau A)U_0$, where U_0 is the initial condition and $\tau = 1/16$ is the time step size.

	$p = 0$			$p = 1$			$p = 3$			$p = 5$		
Size A	2D	1D⊗1D	saving	2D	1D⊗1D	saving	2D	1D⊗1D	saving	2D	1D⊗1D	saving
56	0.01	0.038	0.3	0.05	0.04	1.1	0.06	0.06	0.9	0.05	0.09	0.6
240	0.003	0.003	1	0.01	0.02	0.5	0.01	0.03	0.3	0.01	0.01	0.6
992	0.055	0.006	9	0.02	0.03	0.6	0.02	0.01	1.3	0.02	0.02	1
4,032	0.064	0.005	13	0.17	0.01	16	0.16	0.02	8	0.15	0.04	4
16,256	0.781	0.021	37	2.28	0.03	71	2.24	0.08	27	2.31	0.18	13
65,280	13.32	0.056	237	40.04	0.16	244	40.33	0.55	73	40.33	1.26	32
261,632	184.35	0.206	**895**	799.03	1.16	**687**	792.33	4.04	**196**	775.39	9.36	**83**

Table 3: Computational times in seconds for $U_1 = \varphi_p(-\tau A)U_0$ with $p = 0, 1, 3, 5$ solving the full 2D problem and 1D Kronecker products and Sylvester equations for different sizes of matrix A.

Table 3 shows the computational times in seconds for computing U_1 with the action of the full 2D matrix and Kronecker products of 1D matrices together with Sylvester equations for different sizes of matrix A. Figure 2 reflects the relative error defined in (17). In this case, we conclude that our approach is significantly faster; the largest A is 895 times faster for the lowest p and 83 times faster for the higher p. However, relative errors perform

	$p=0$			$p=1$			$p=3$			$p=5$		
Size A	2D	1D⊗1D	saving	2D	1D⊗1D	saving	2D	1D⊗1D	saving	2D	1D⊗1D	saving
56	0.01	0.019	0.5	0.05	0.06	0.8	0.06	0.06	1	0.05	0.11	0.5
240	0.003	0.003	1	0.01	0.01	0.6	0.01	0.03	0.3	0.01	0.02	0.5
992	0.055	0.005	11	0.02	0.05	0.4	0.02	0.02	0.9	0.02	0.03	0.6
4,032	0.064	0.003	19	0.17	0.02	9	0.16	0.04	4	0.15	0.08	2
16,256	0.781	0.027	29	2.28	0.07	35	2.24	0.18	13	2.31	0.43	5
65,280	13.32	0.05	267	40.04	0.31	127	40.33	0.96	42	40.33	1.87	22
261,632	184.35	0.217	**849**	799.03	2.47	**323**	792.33	6.05	**131**	775.39	12.39	**63**

Table 4: Computational times in seconds for $U_1 = \varphi_p(-\tau A)U_0$ with $p = 0, 1, 3, 5$ solving the full 2D problem and 1D Kronecker products and 2D linear systems for different sizes of matrix A.

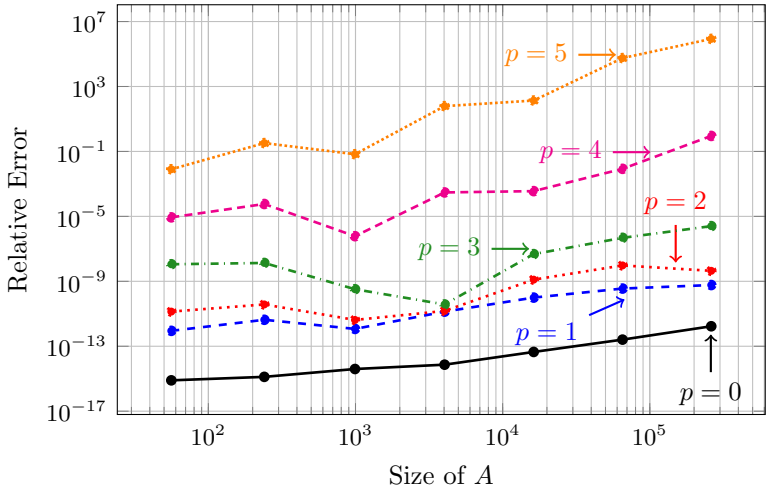

Figure 2: Relative approximation error to U_1 with $p = 0, \ldots, 5$ for 1D Kronecker products and Sylvester equations.

properly for $p \leq 3$, but we observe significant errors for $p \geq 4$ due to the ill-conditioning of the Sylvester systems.

In order to reduce these errors, we now solve directly the 2D system in Algorithm 1 (step 20) instead of a Sylvester equation. As we assemble the full 2D matrix, we lose the memory savings. Table 4 and Figure 3 show the computational times and relative errors employing this approach and the original action of the full 2D matrix. Solving directly the 2D system in Algorithm 1 is slightly slower than solving the Sylvester equations but significantly faster than the original routine (63 times faster in the last experiment). In terms of relative errors, they are small for $p \leq 3$, and moderate for $p = 4$. We will further study our method's performance and how to solve the ill-conditioning problem for high p and large non-symmetric matrices in the future.

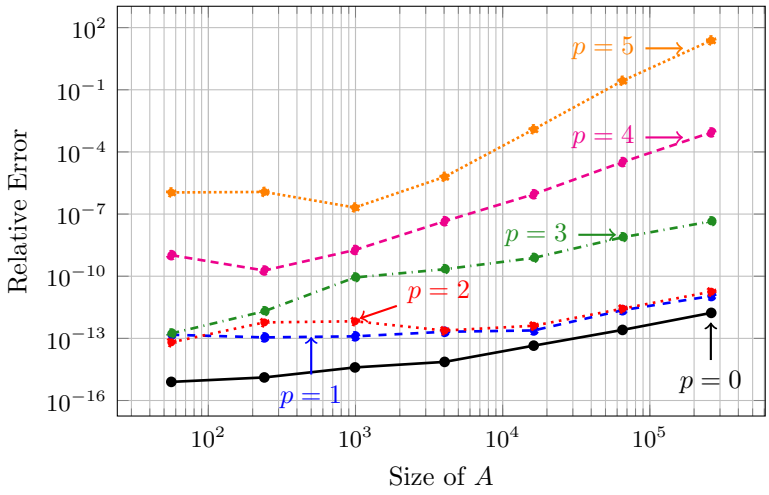

Figure 3: Relative approximation error to U_1 with $p = 0, \ldots, 5$ for 1D Kronecker products and 2D linear equations.

5.3. Allen-Cahn equation

We consider the 2D Allen-Cahn equation from [10, 39]

$$u_t = \Delta u + \frac{1}{\epsilon^2} f(u),$$

where $f(u) = u - u^3$. We set homogeneous Neumman boundary conditions over $\Omega = (0,1) \times (0,1)$, $T = 0.02$, $\epsilon = 0.01$, and initial condition

$$u_0(x,y) = \tanh \frac{0.25 + 0.1\cos(N\theta) - \sqrt{(x-0.5)^2 + (y-0.5)^2}}{\sqrt{2}\alpha},$$

where

$$\theta(x,y) = \begin{cases} \tan^{-1}\dfrac{y-0.5}{x-0.5}, & \text{if } x > 0.5, \\ \pi + \tan^{-1}\dfrac{y-0.5}{x-0.5}, & \text{otherwise.} \end{cases}$$

The top-left subfigure of Figure 4 shows the initial condition for $N = 7$ and $\alpha = 0.75$. We solve the full 2D problem with the exponential Euler method [24]

$$U^k = \varphi_0(-\tau A)U^{k-1} + \tau\varphi_1(-\tau A)F(U^{k-1}), \tag{18}$$

where, after the spatial discretization, A is the matrix of the Laplacian operator, F is the nonlinear vector of $f(u)$, and τ is the time-step size.

As we have homogeneous Neumann boundary conditions, the matrix A after the spatial discretization with FEM is singular. Therefore, the Sylvester equation we solve to compute

the action of φ_1 has no unique solution. To overcome this problem, we add and subtract a small reaction term to the problem

$$u_t = \Delta u + \epsilon^2 u + \frac{1}{\epsilon^2} f(u) - \epsilon^2 u,$$

and instead of (18), for the Kronecker approach, we solve the following time-marching scheme

$$U^k = \varphi_0(-\tau \tilde{A}) U^{k-1} + \tau \varphi_1(-\tau \tilde{A}) \tilde{F}(U^{k-1}), \tag{19}$$

where $\tilde{A}$ is the matrix we obtain after the spatial discretization of the reaction-difussion operator $\Delta + \epsilon^2 I$ and $\tilde{F}$ is the vector corresponding to the modified non-linear term $\tilde{f}(u) := \frac{1}{\epsilon^2} f(u) - \epsilon^2 u$.

We solve (18) and (19) for a mesh of $2^9 + 2$ elements for space dimension and $\tau = 10^{-4}$. Therefore, the full 2D matrix size is $264,196^2$, whereas the 1D matrices are 514^2. Figure 4 shows some snapshots of the solution we obtained with Algorithm 1. Here, we have a star-shaped initial condition that converges to a circle and, finally, its radius shrinks. In Figure 5 we can see that the relative error at each time step is around $1.7 \cdot 10^{-6}$. Finally, we conclude that the algorithm solving the full 2D matrix lasted around 57 minutes (3423 seconds) for the whole simulation and our method only 1.6 minutes (100 seconds), that is, 34 times faster.

6. Conclusions

This work proposes an algorithm for the fast computation of the action of $\varphi-$functions over vectors when the matrix of the system has Kronecker sum structure. We introduce an auxiliary matrix function $\Phi_p(A) = A^p \varphi_p(A)$ expressed as Kronecker products of smaller matrices. We employ existing software to compute the action of this auxiliary function and recover the original $\varphi-$action by solving Sylvester equations. Our approach computes the action of $\varphi_p(A)$ on a vector by computing only actions of smaller 1D matrices. We show the performance of our method for different linear operators and the 2D+time Allen-Cahn equation. We conclude that our approach is orders of magnitude faster (in some cases, up to 27,000 times faster) than solving the full 2D problem, and it is cheaper in terms of memory (about 5.5 times) because we only deal with 1D matrices.

However, we observe high errors in the solution for $p \geq 4$ in large non-symmetric matrices when we solve the Sylvester equations, which are ill-conditioned. These errors decrease orders of magnitude when we solve the corresponding 2D equations instead of the Sylvester equations but are still considerable. We plan to study this case more in detail in the future. We also plan to improve the memory savings by extending the software implementations to allow sparse matrices rather than dense ones. Finally, our decomposition in Kronecker products of the $\varphi-$functions is easily extendable to 3D problems. However, this approach involves the resolution of a generalized Sylvester equation. We will explore this possibility also as future work. Possible extensions of this work include: (a) a further study of our approach for high-order methods ($p \geq 4$); (b) decomposition in Kronecker products for

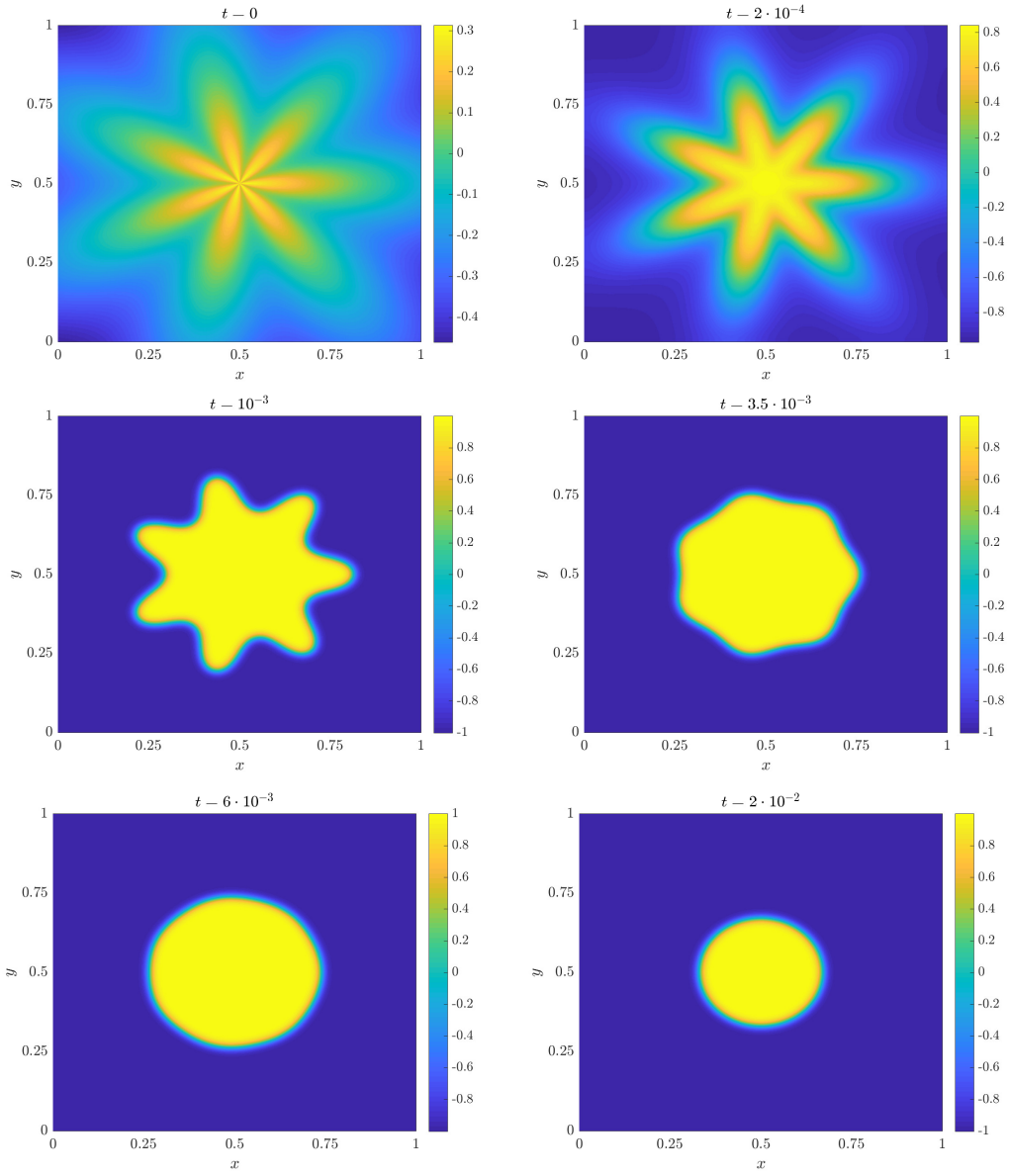

Figure 4: Allen-Cahn solution snapshots obtained with Algorithm 1 for the exponential Euler time-marching scheme.

$\varphi-$functions of 3D matrices with Kronecker sum structure; (d) combination of our method with isogeometric analysis in space; (e) extension of our method to hyperbolic problems.

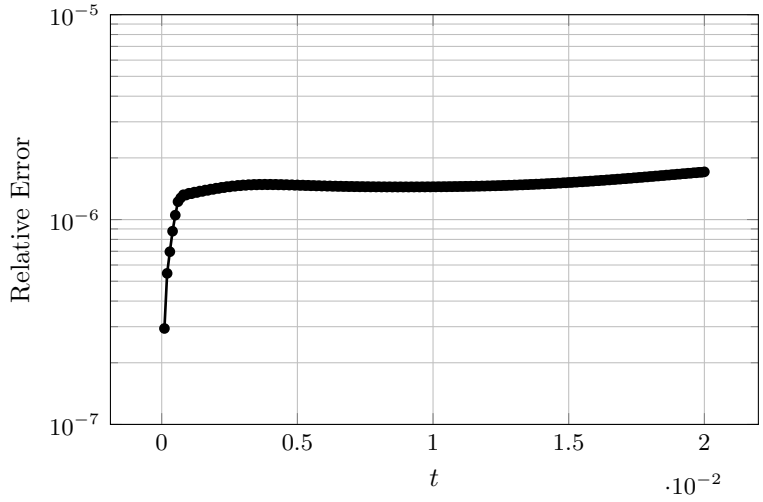

Figure 5: Relative error at each time step for $\tau = 10^{-4}$.

Appendix A. Kronecker product

Definition. Let $A \in \mathbb{R}^{p\times q}$ and $B \in \mathbb{R}^{r\times s}$, the Kronecker product $A \otimes B \in \mathbb{R}^{pr\times qs}$ is defined as

$$A \otimes B = \begin{bmatrix} a_{11}B & a_{12}B & \cdots & a_{1q}B \\ a_{21}B & a_{22}B & \cdots & a_{2q}B \\ \vdots & \vdots & \ddots & \vdots \\ a_{p1}B & a_{p1}B & \cdots & a_{pq}B \end{bmatrix}.$$

Properties. The Kronecker product satisfies the following properties:

- $A \otimes (B + C) = A \otimes B + A \otimes C$,
- $(B + C) \otimes A = B \otimes A + C \otimes A$,
- $(\lambda A) \otimes B = A \otimes (\lambda B) = \lambda(A \otimes B)$, $\forall \lambda \in \mathbb{R}$,
- $(A \otimes B) \otimes C = A \otimes (B \otimes C)$,
- $(A \otimes B)(C \otimes D) = (AC) \otimes (BD)$,
- $(A \otimes B)^{-1} = A^{-1} \otimes B^{-1}$,
- $(A \otimes B)^T = A^T \otimes B^T$.

Kronecker product times a vector. Let $x \in \mathbb{R}^{qs\times 1}$, in the following matrix-vector multiplication

$$y = (A \otimes B)x, \tag{A.1}$$

we have $y \in \mathbb{R}^{pr\times 1}$. In order to avoid forming $A \otimes B$ explicitly, we need to reshape the vectors x and y as

$$x = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_q \end{bmatrix},\ x_i \in \mathbb{R}^s, \quad y = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_p \end{bmatrix},\ y_i \in \mathbb{R}^r.$$

and form the following matrices

$$X = \begin{bmatrix} x_1 & x_2 & \cdots & x_q \end{bmatrix} \in \mathbb{R}^{s\times q},\ Y = \begin{bmatrix} y_1 & y_2 & \cdots & y_p \end{bmatrix} \in \mathbb{R}^{r\times p}.$$

It holds the following property

$$y = (A \otimes B)x \Longleftrightarrow Y = BXA^T.$$

Therefore, to compute (A.1), we first reshape x into X, then we compute $Y = BXA^T$ and finally, we reshape Y into y.

Appendix B. Sylvester equation

In (16), we need to compute $z = \varphi_p(A)b$ from $\Phi_p(A)b$, where b is a vector. After computing the righ-hand-side of (13), we obtain an expression of the form

$$A^p \underbrace{\varphi_p(A)b}_{z} = w,$$

where w is another vector. For $p = 1$, we have $Az = w$, or equivalently

$$(A^x \otimes I^y + I^x \otimes A^y)z = w.$$

If we reshape vectors z and w into matrices U and W, we obtain

$$Z(A^x)^T + A^y Z = W, \tag{B.1}$$

which is a Sylvester equation [20]. For $p > 1$, we have to solve p Sylvester equations of type (B.1).

Appendix C. Proofs of Theorems 1 and 2

Proof of Theorem 1: By induction over p:

- We prove (13) for $p = 1$. From (11) and (12), we have

$$\begin{aligned} \Phi_1(A) &= \Phi_0(A) - I = \Phi_0(A^x) \otimes \Phi_0(A^y) - I^x \otimes I^y \\ &= (\Phi_1(A^x) + I^x) \otimes (\Phi_1(A^y) + I^y) - I^x \otimes I^y \\ &= \Phi_1(A^x) \otimes \Phi_1(A^y) + \Phi_1(A^x) \otimes I^y + I^x \otimes \Phi_1(A^y). \end{aligned}$$

- We assume (13) true for $p-1$ and prove it for p. From (11) acting on A^x and A^y, and the induction hypothesis

$$\begin{aligned}
\Phi_p(A) &= \Phi_{p-1}(A) - \frac{1}{(p-1)!}A^{p-1} \\
&= \Phi_{p-1}(A^x) \otimes \Phi_{p-1}(A^y) \\
&+ \Phi_{p-1}(A^x) \otimes \left(\sum_{j=0}^{p-2} \frac{(A^y)^j}{j!} \right) + \left(\sum_{i=0}^{p-2} \frac{(A^x)^i}{i!} \right) \otimes \Phi_{p-1}(A^y) \\
&+ \sum_{i=1}^{p-2} \sum_{j=p-1-i}^{p-2} \frac{(A^x)^i}{i!} \otimes \frac{(A^y)^j}{j!} - \frac{1}{(p-1)!}(A^x \otimes I^y + I^x \otimes A^y)^{p-1},
\end{aligned}$$

again from (11) we have

$$\begin{aligned}
\Phi_p(A) &= \left(\Phi_p(A^x) + \frac{(A^x)^{p-1}}{(p-1)!} \right) \otimes \left(\Phi_p(A^y) + \frac{(A^y)^{p-1}}{(p-1)!} \right) \\
&+ \left(\Phi_p(A^x) + \frac{(A^x)^{p-1}}{(p-1)!} \right) \otimes \left(\sum_{j=0}^{p-2} \frac{(A^y)^j}{j!} \right) \\
&+ \left(\sum_{i=0}^{p-2} \frac{(A^x)^i}{i!} \right) \otimes \left(\Phi_p(A^y) + \frac{(A^y)^{p-1}}{(p-1)!} \right) \\
&+ \sum_{i=1}^{p-2} \sum_{j=p-1-i}^{p-2} \frac{(A^x)^i}{i!} \otimes \frac{(A^y)^j}{j!} - \frac{1}{(p-1)!}(A^x \otimes I^y + I^x \otimes A^y)^{p-1},
\end{aligned}$$

and equivalently

$$\begin{aligned}
\Phi_p(A) &= \Phi_p(A^x) \otimes \Phi_p(A^y) \\
&+ \Phi_p(A^x) \otimes \left(\sum_{j=0}^{p-1} \frac{(A^y)^j}{j!} \right) + \left(\sum_{i=0}^{p-1} \frac{(A^x)^i}{i!} \right) \otimes \Phi_p(A^y) \\
&+ \frac{(A^x)^{p-1}}{(p-1)!} \otimes \frac{(A^y)^{p-1}}{(p-1)!} \\
&+ \frac{(A^x)^{p-1}}{(p-1)!} \otimes \left(\sum_{j=0}^{p-2} \frac{(A^y)^j}{j!} \right) + \left(\sum_{i=0}^{p-2} \frac{(A^x)^i}{i!} \right) \otimes \frac{(A^y)^{p-1}}{(p-1)!} \\
&+ \sum_{i=1}^{p-2} \sum_{j=p-1-i}^{p-2} \frac{(A^x)^i}{i!} \otimes \frac{(A^y)^j}{j!} - \frac{1}{(p-1)!}(A^x \otimes I^y + I^x \otimes A^y)^{p-1}.
\end{aligned}$$

Employing the classical binomial expansion and the properties of the Kronecker prod-

uct, we write the last term as

$$\frac{1}{(p-1)!}(A^x \otimes I^y + I^x \otimes A^y)^{p-1} = \frac{1}{(p-1)!}\sum_{i=0}^{p-1}\binom{p-1}{i}(A^x \otimes I^y)^i(I^x \otimes A^y)^{p-1-i}$$
$$= \sum_{j=0}^{p-1}\frac{(A^x)^i}{i!} \otimes \frac{(A^y)^{p-1-i}}{(p-1-i)!}. \tag{C.1}$$

Eliminating the identity terms, we have

$$\begin{aligned}\Phi_p(A) = {} & \Phi_p(A^x) \otimes \Phi_p(A^y) \\ & + \Phi_p(A^x) \otimes \left(\sum_{j=0}^{p-1}\frac{(A^y)^j}{j!}\right) + \left(\sum_{i=0}^{p-1}\frac{(A^x)^i}{i!}\right) \otimes \Phi_p(A^y) \\ & + \frac{(A^x)^{p-1}}{(p-1)!} \otimes \frac{(A^y)^{p-1}}{(p-1)!} \\ & + \frac{(A^x)^{p-1}}{(p-1)!} \otimes \left(\sum_{j=1}^{p-2}\frac{(A^y)^j}{j!}\right) + \left(\sum_{i=1}^{p-2}\frac{(A^x)^i}{i!}\right) \otimes \frac{(A^y)^{p-1}}{(p-1)!} \\ & + \sum_{i=1}^{p-2}\sum_{j=p-1-i}^{p-2}\frac{(A^x)^i}{i!} \otimes \frac{(A^y)^j}{j!} - \sum_{j=1}^{p-2}\frac{(A^x)^i}{i!} \otimes \frac{(A^y)^{p-1-i}}{(p-1-i)!}.\end{aligned}$$

The last term above cancels out fixing $j = p-1-i$ in the previous term to obtain

$$\begin{aligned}\Phi_p(A) = {} & \Phi_p(A^x) \otimes \Phi_p(A^y) \\ & + \Phi_p(A^x) \otimes \left(\sum_{j=0}^{p-1}\frac{(A^y)^j}{j!}\right) + \left(\sum_{i=0}^{p-1}\frac{(A^x)^i}{i!}\right) \otimes \Phi_p(A^y) \\ & + \frac{(A^x)^{p-1}}{(p-1)!} \otimes \frac{(A^y)^{p-1}}{(p-1)!} \\ & + \frac{(A^x)^{p-1}}{(p-1)!} \otimes \left(\sum_{j=1}^{p-2}\frac{(A^y)^j}{j!}\right) + \left(\sum_{i=1}^{p-2}\frac{(A^x)^i}{i!}\right) \otimes \frac{(A^y)^{p-1}}{(p-1)!} \\ & + \sum_{i=1}^{p-2}\sum_{j=p-i}^{p-2}\frac{(A^x)^i}{i!} \otimes \frac{(A^y)^j}{j!}.\end{aligned}$$

Collecting terms, we have

$$\begin{aligned}\Phi_p(A) &= \Phi_p(A^x)\otimes\Phi_p(A^y)\\ &+ \Phi_p(A^x)\otimes\left(\sum_{j=0}^{p-1}\frac{(A^y)^j}{j!}\right)+\left(\sum_{i=0}^{p-1}\frac{(A^x)^i}{i!}\right)\otimes\Phi_p(A^y)\\ &+ \frac{(A^x)^{p-1}}{(p-1)!}\otimes\left(\sum_{j=1}^{p-1}\frac{(A^y)^j}{j!}\right)+\sum_{i=1}^{p-2}\sum_{j=p-i}^{p-1}\frac{(A^x)^i}{i!}\otimes\frac{(A^y)^j}{j!}.\end{aligned}$$

Finally, adding the last two terms, we obtain (13).

Proof of Theorem 2: By induction over p:

- We first prove (14) and (15) for $p=1$ and $p=2$, respectively. We observe that (14) coincides with (13) for $p=1$, so we already proved that it holds. Now, from (11) and (14) with $p=1$, we obtain

$$\begin{aligned}\Phi_2(A) &= \Phi_1(A)-A\\ &= \Phi_1(A^x)\otimes\Phi_1(A^y)+\Phi_1(A^x)\otimes I^y+I^x\otimes\Phi_1(A^y)-A^x\otimes I^y-I^x\otimes A^y\\ &= \Phi_1(A^x)\otimes\Phi_1(A^y)+(\Phi_2(A^x)+A^x)\otimes I^y+I^x\otimes(\Phi_2(A^y)+A^y)-A^x\otimes I^y-I^x\otimes A^y\\ &= \Phi_1(A^x)\otimes\Phi_1(A^y)+\Phi_2(A^x)\otimes I^y+I^x\otimes\Phi_2(A^y).\end{aligned}$$

- If $p=2r+1$, we suppose it is true for $p-1=2r$. From (11) and (15), we have

$$\begin{aligned}\Phi_{2r+1}(A) &= \Phi_{2r}(A)-\frac{1}{(2r)!}A^{2r}\\ &= \sum_{j=0}^{r-1}\frac{(A^x)^j}{j!}\otimes\Phi_{2r-j}(A^y)+\sum_{j=0}^{r-1}\Phi_{2r-j}(A^x)\otimes\frac{(A^y)^j}{j!}+\Phi_r(A^x)\otimes\Phi_r(A^y)\\ &- \frac{1}{(2r)!}(A^x\otimes I^y+I^x\otimes A^y)^{2r},\end{aligned}$$

again from (11) and (C.1), we obtain

$$\begin{aligned}\Phi_{2r+1}(A) &= \sum_{j=0}^{r-1}\frac{(A^x)^j}{j!}\otimes\left(\Phi_{2r+1-j}(A^y)+\frac{1}{(2r-j)!}(A^y)^{2r-j}\right)\\ &+ \sum_{j=0}^{r-1}\left(\Phi_{2r+1-j}(A^x)+\frac{1}{(2r-j)!}(A^x)^{2r-j}\right)\otimes\frac{(A^y)^j}{j!}\\ &+ \left(\Phi_{r+1}(A^x)+\frac{1}{r!}(A^x)^r\right)\otimes\left(\Phi_{r+1}(A^y)+\frac{1}{r!}(A^y)^r\right)\\ &- \sum_{i=0}^{2r}\frac{(A^x)^i}{i!}\otimes\frac{(A^y)^{2r-i}}{(2r-i)!},\end{aligned}$$

or equivalently, rearranging terms, we obtain

$$\begin{aligned}\Phi_{2r+1}(A) = &\sum_{j=0}^{r} \frac{(A^x)^j}{j!} \otimes \Phi_{2r+1-j}(A^y) + \sum_{j=0}^{r} \Phi_{2r+1-j}(A^x) \otimes \frac{(A^y)^j}{j!} + \Phi_{r+1}(A^x) \otimes \Phi_{r+1}(A^y) \\ &+ \sum_{j=0}^{r-1} \frac{(A^x)^j}{j!} \otimes \frac{(A^y)^{2r-j}}{(2r-j)!} + \sum_{j=0}^{r-1} \frac{(A^x)^{2r-j}}{(2r-j)!} \otimes \frac{(A^y)^j}{j!} + \frac{(A^x)^r}{r!} \otimes \frac{(A^y)^r}{r!} \\ &- \sum_{i=0}^{2r} \frac{(A^x)^i}{i!} \otimes \frac{(A^y)^{2r-i}}{(2r-i)!}.\end{aligned}$$

Adding the 4th and 6th terms together and performing a change of indexes ($2r-j=i$) in the 5th term, we obtain

$$\begin{aligned}\Phi_{2r+1}(A) = &\sum_{j=0}^{r} \frac{(A^x)^j}{j!} \otimes \Phi_{2r+1-j}(A^y) + \sum_{j=0}^{r} \Phi_{2r+1-j}(A^x) \otimes \frac{(A^y)^j}{j!} + \Phi_{r+1}(A^x) \otimes \Phi_{r+1}(A^y) \\ &+ \sum_{j=0}^{r} \frac{(A^x)^j}{j!} \otimes \frac{(A^y)^{2r-j}}{(2r-j)!} + \sum_{i=r+1}^{2r} \frac{(A^x)^i}{i!} \otimes \frac{(A^y)^{2r-i}}{(2r-i)!} \\ &- \sum_{i=0}^{2r} \frac{(A^x)^i}{i!} \otimes \frac{(A^y)^{2r-i}}{(2r-i)!}.\end{aligned}$$

Finally, the last three terms above cancel out, which results in (14) as claimed.

- If $p = 2r$, we suppose it is true for $p-1 = 2r-1$. From (11) and (14), we have

$$\begin{aligned}\Phi_{2r}(A) &= \Phi_{2r-1}(A) - \frac{1}{(2r-1)!} A^{2r-1} \\ &= \sum_{j=0}^{r-1} \frac{(A^x)^j}{j!} \otimes \Phi_{2r-1-j}(A^y) + \sum_{j=0}^{r-1} \Phi_{2r-1-j}(A^x) \otimes \frac{(A^y)^j}{j!} + \Phi_r(A^x) \otimes \Phi_r(A^y) \\ &- \frac{1}{(2r-1)!} (A^x \otimes I^y + I^x \otimes A^y)^{2r-1}.\end{aligned}$$

Employing (11) in the first two terms and (C.1) in the last one, we have

$$\begin{aligned}\Phi_{2r}(A) = &\sum_{j=0}^{r-1} \frac{(A^x)^j}{j!} \otimes \left(\Phi_{2r-j}(A^y) + \frac{1}{(2r-1-j)!} (A^y)^{2r-1-j} \right) \\ &+ \sum_{j=0}^{r-1} \left(\Phi_{2r-j}(A^x) + \frac{1}{(2r-1-j)!} (A^x)^{2r-1-j} \right) \otimes \frac{(A^y)^j}{j!} \\ &+ \Phi_r(A^x) \otimes \Phi_r(A^y) - \sum_{i=0}^{2r-1} \frac{(A^x)^i}{i!} \otimes \frac{(A^y)^{2r-1-i}}{(2r-1-i)!},\end{aligned}$$

or equivalently,

$$\begin{aligned}\Phi_{2r}(A) = &\sum_{j=0}^{r-1} \frac{(A^x)^j}{j!} \otimes \Phi_{2r-j}(A^y) + \sum_{j=0}^{r-1} \Phi_{2r-j}(A^x) \otimes \frac{(A^y)^j}{j!} + \Phi_r(A^x) \otimes \Phi_r(A^y) \\ &+ \sum_{j=0}^{r-1} \frac{(A^x)^j}{j!} \otimes \frac{(A^y)^{2r-1-j}}{(2r-1-j)!} + \sum_{j=0}^{r-1} \frac{(A^x)^{2r-1-j}}{(2r-j-1)!} \otimes \frac{(A^y)^j}{j!} \\ &- \sum_{i=0}^{2r-1} \frac{(A^x)^i}{i!} \otimes \frac{(A^y)^{2r-1-i}}{(2r-1-i)!}.\end{aligned}$$

Finally, by changing the indexes $i = 2r-1-j$ in the 5th term above, we obtain

$$\begin{aligned}\Phi_{2r}(A) = &\sum_{j=0}^{r-1} \frac{(A^x)^j}{j!} \otimes \Phi_{2r-j}(A^y) + \sum_{j=0}^{r-1} \Phi_{2r-j}(A^x) \otimes \frac{(A^y)^j}{j!} + \Phi_r(A^x) \otimes \Phi_r(A^y) \\ &+ \sum_{j=0}^{r-1} \frac{(A^x)^j}{j!} \otimes \frac{(A^y)^{2r-1-j}}{(2r-1-j)!} + \sum_{i=r}^{2r-1} \frac{(A^x)^i}{i!} \otimes \frac{(A^y)^{2r-1-i}}{(2r-1-i)!} \\ &- \sum_{i=0}^{2r-1} \frac{(A^x)^i}{i!} \otimes \frac{(A^y)^{2r-1-i}}{(2r-1-i)!}.\end{aligned}$$

The last three terms above cancel out, and we obtain (15).

Acknowledgements

We received funding from: the European Union's Horizon 2020 research and innovation program under the Marie Sklodowska-Curie grant agreement No 777778 (MATHROCKS). JMM and DP received funding from the Projects of the the Spanish Ministry of Science and Innovation with references PID2019-108111RB-I00 (FEDER/AEI) and PDC2021-121093-I00, the "BCAM Severo Ochoa" accreditation of excellence (SEV-2017-0718), the Basque Government through the BERC 2018-2021 program, and the Consolidated Research Group MATHMODE (IT1294-19) given by the Department of Education. JMM has also received funding from the Basque Government through the postdoctoral program for the improvement of doctor research staff (POS_2019_1_0001) and the European Union's Horizon 2020 research and innovation programme under the Marie Sklodowska-Curie individual fellowship grant agreement No 101017984 (GEODPG). DP has also received funding from the European Regional Development Fund (ERDF) through the Interreg V-A Spain-France-Andorra program POCTEFA 2014-2020 Project PIXIL (EFA362/19) and the three Elkartek projects 3KIA (KK-2020/00049), EXPERTIA (KK-2021/00048), and SIGZE (KK-2021/00095). VMC was funded in part by the Professorial Chair in Computational Geoscience at Curtin University and the Deep Earth Imaging Enterprise Future Science Platforms of the Commonwealth Scientific Industrial Research Organisation, CSIRO, of Australia. The Curtin Corrosion Centre and the Curtin Institute for Computation kindly provide ongoing support for VMC.

Data Availability Statement

The codes to reproduce the numerical results of this articles can be found in: `https://github.com/jmunoz022/Kronecker_EI`.

References

[1] A. H. Al-Mohy and N. J. Higham. A new scaling and squaring algorithm for the matrix exponential. *SIAM Journal on Matrix Analysis and Applications*, 31(3):970–989, 2010.

[2] A. H. Al-Mohy and N. J. Higham. Computing the action of the matrix exponential, with an application to exponential integrators. *SIAM Journal on Scientific Computing*, 33(2):488–511, 2011.

[3] Y. Bazilevs, V. M. Calo, J. A. Cottrell, J. A. Evans, T. J. R. Hughes, S. Lipton, M. A. Scott, and T. W. Sederberg. Isogeometric analysis using T-splines. *Computer Methods in Applied Mechanics and Engineering*, 199(5-8):229–263, 2010.

[4] M. Benzi and V. Simoncini. Approximation of functions of large matrices with Kronecker structure. *Numerische Mathematik*, 135(1):1–26, 2017.

[5] H. Berland and B. Skaflestad. Solving the nonlinear Schrödinger equation using exponential integrators. *Modeling, Identification and Control*, 27(4):201–207, 2006.

[6] H. Berland, B. Skaflestad, and W. M. Wright. EXPINT—A MATLAB package for exponential integrators. *ACM Transactions on Mathematical Software (TOMS)*, 33(1):4–es, 2007.

[7] M. Caliari, F. Cassini, L. Einkemmer, A. Ostermann, and F. Zivcovich. A μ-mode integrator for solving evolution equations in Kronecker form. *arXiv preprint arXiv:2103.01691*, 2021.

[8] M. Caliari and A. Ostermann. Implementation of exponential Rosenbrock-type integrators. *Applied Numerical Mathematics*, 59(3-4):568–581, 2009.

[9] V. Calo, Q. Deng, and V. Puzyrev. Dispersion optimized quadratures for isogeometric analysis. *Journal of Computational and Applied Mathematics*, 355:283–300, 2019.

[10] V. Calo, P. Minev, and V. Puzyrev. Splitting schemes for phase-field models. *Applied Numerical Mathematics*, 156:192–209, 2020.

[11] J. A. Cottrell, T. J. Hughes, and Y. Bazilevs. *Isogeometric analysis: toward integration of CAD and FEA*. John Wiley & Sons, 2009.

[12] N. Crouseilles, L. Einkemmer, and J. Massot. Exponential methods for solving hyperbolic problems with application to collisionless kinetic equations. *Journal of Computational Physics*, 420:109688, 2020.

[13] M. Fasi and N. J. Higham. An arbitrary precision scaling and squaring algorithm for the matrix exponential. *SIAM Journal on Matrix Analysis and Applications*, 40(4):1233–1256, 2019.

[14] H. Gao, L. Ju, X. Li, and R. Duddu. A space-time adaptive finite element method with exponential time integrator for the phase field model of pitting corrosion. *Journal of Computational Physics*, 406:109191, 2020.

[15] S. Gaudreault and J. A. Pudykiewicz. An efficient exponential time integration method for the numerical solution of the shallow water equations on the sphere. *Journal of Computational Physics*, 322:827–848, 2016.

[16] S. Gaudreault, G. Rainwater, and M. Tokman. KIOPS: A fast adaptive Krylov subspace solver for exponential integrators. *Journal of Computational Physics*, 372:236–255, 2018.

[17] H. Gómez, V. M. Calo, Y. Bazilevs, and T. J. Hughes. Isogeometric analysis of the Cahn–Hilliard phase-field model. *Computer methods in applied mechanics and engineering*, 197(49-50):4333–4352, 2008.

[18] A. Graham. *Kronecker products and matrix calculus with applications*. Courier Dover Publications, 2018.

[19] E. Hansen and A. Ostermann. Exponential splitting for unbounded operators. *Mathematics of Computation*, 78(267):1485–1496, 2009.

[20] N. J. Higham. Perturbation theory and backward error for AX-XB=C. *BIT Numerical Mathematics*, 33(1):124–136, 1993.

[21] N. J. Higham and E. Hopkins. A catalogue of software for matrix functions. Version 3.0. Technical report. MIMS EPrint 2020.7, The University of Manchester, 2020.

[22] M. Hochbruck, C. Lubich, and H. Selhofer. Exponential integrators for large systems of differential equations. *SIAM Journal on Scientific Computing*, 19(5):1552–1574, 1998.

[23] M. Hochbruck and A. Ostermann. Explicit exponential Runge–Kutta methods for semilinear parabolic problems. *SIAM Journal on Numerical Analysis*, 43(3):1069–1090, 2005.

[24] M. Hochbruck and A. Ostermann. Exponential integrators. *Acta Numerica*, 19:209–286, 2010.

[25] M. Hochbruck and A. Ostermann. Exponential multistep methods of Adams-type. *BIT Numerical Mathematics*, 51(4):889–908, 2011.

[26] M. Hochbruck, A. Ostermann, and J. Schweitzer. Exponential Rosenbrock-type methods. *SIAM Journal on Numerical Analysis*, 47(1):786–803, 2009.

[27] J. D. Lawson. Generalized Runge-Kutta processes for stable systems with large Lipschitz constants. *SIAM Journal on Numerical Analysis*, 4(3):372–380, 1967.

[28] S. J. Li, L. S. Luo, Z. J. Wang, and L. Ju. An exponential time-integrator scheme for steady and unsteady inviscid flows. *Journal of Computational Physics*, 365:206–225, 2018.

[29] G. J. Lord and A. Tambue. Stochastic exponential integrators for the finite element discretization of SPDEs for multiplicative and additive noise. *IMA Journal of Numerical Analysis*, 33(2):515–543, 2013.

[30] M. A. Medvedeva, T. Simos, and C. Tsitouras. Exponential integrators for linear inhomogeneous problems. *Mathematical Methods in the Applied Sciences*, 44(1):937–944, 2021.

[31] B. V. Minchev and W. Wright. A review of exponential integrators for first order semi-linear problems. PhD thesis. PREPRINT NUMERICS NO. 2/2005, The Norwegian University of Science and Technology, 2005.

[32] J. Muñoz-Matute, D. Pardo, and L. Demkowicz. A DPG-based time-marching scheme for linear hyperbolic problems. *Computer Methods in Applied Mechanics and Engineering*, 373:113539, 2021.

[33] J. Muñoz-Matute, D. Pardo, and L. Demkowicz. Equivalence between the DPG method and the exponential integrators for linear parabolic problems. *Journal of Computational Physics*, 429:110016, 2021.

[34] J. Niesen and W. M. Wright. Algorithm 919: A Krylov subspace algorithm for evaluating the φ-functions appearing in exponential integrators. *ACM Transactions on Mathematical Software (TOMS)*, 38(3):1–19, 2012.

[35] M. N. Özişik, H. R. Orlande, M. J. Colaco, and R. M. Cotta. *Finite difference methods in heat transfer*. CRC press, 2017.

[36] N. Rambeerich, D. Tangman, A. Gopaul, and M. Bhuruth. Exponential time integration for fast finite element solutions of some financial engineering problems. *Journal of Computational and Applied Mathematics*, 224(2):668–678, 2009.

[37] B. N. Sheehan, Y. Saad, and R. B. Sidje. Computing exp(-τa)b with Laguerre polynomials. *Electronic Transactions on Numerical Analysis*, 37:147–165, 2010.

[38] A. Tambue, G. J. Lord, and S. Geiger. An exponential integrator for advection-dominated reactive transport in heterogeneous porous media. *Journal of Computational Physics*, 229(10):3957–3969, 2010.

[39] P. Vignal, N. Collier, L. Dalcin, D. Brown, and V. Calo. An energy-stable time-integrator for phase-field models. *Computer Methods in Applied Mechanics and Engineering*, 316:1179–1214, 2017.

[40] X. Wu, G. Van Zwieten, and K. Van der Zee. Stabilized second-order convex splitting schemes for Cahn–Hilliard models with application to diffuse-interface tumor-growth models. *International journal for numerical methods in biomedical engineering*, 30(2):180–203, 2014.

[41] O. C. Zienkiewicz, R. L. Taylor, P. Nithiarasu, and J. Zhu. *The finite element method*, volume 3. McGraw-hill London, 1977.