Parallel Computation of functions of matrices and their action on vectors for exponential integrators

Sergio Blanes*

October 12, 2024

Abstract

We present a novel class of methods to compute functions of matrices or their action on vectors that are suitable for parallel programming and can improve the performance of exponential integrators. Solving appropriate simple linear systems of equations in parallel (or computing the inverse of several matrices) and with a proper linear combination of the results, allows us to obtain new high order approximations to the desired functions of matrices. An error analysis to obtain forward and backward error bounds is presented. The coefficients of each method, which depends on the number of processors, can be adjusted to improve the accuracy, the stability or to reduce the round off errors of the methods. We illustrate this procedure by explicitly constructing some methods which are then tested on several numerical examples.

1 Introduction

The numerical integration of differential equations is an area of great research interest. During the last few decades a number of new families of methods with special properties tailored to solve different classes of problems have emerged like, for example, exponential methods for stiff or oscillatory problems, originated from partial differential equations (see, for example, [31] and references therein). Some of these new methods require either the evaluation of functions of matrices or their action on vectors so, their performance strongly depend on the existence of efficient algorithms to compute them.

We present novel algorithms to compute functions of matrices and their action on vectors which, for example, can improve the performance of exponential integrators and, in addition, can be computed in parallel. We believe that new algorithms which are designed to be computed in parallel will become more frequent and useful in the near future. For instance, about 25 five years ago, in one of the most relevant books in numerical methods [38] it is written: "In recent years we Numerical Recipes authors have increasingly become convinced that a certain revolution, cryptically denoted by the words "parallel programming,"

*Instituto Universitario de Matemática Multidisciplinar, Universitat Politècnica de València, E-46022 Valencia, Spain. serblaza@imm.upv.es

is about to burst forth from its gestation and adolescence in the community of supercomputer users, and become the mainstream methodology for all computing... With multiprocessor workstations right around the corner, we think that now is the right time for scientists and engineers who use computers to start thinking parallel." See also e.g. [6, 11, 18, 20, 21, 23].

The performance of a parallel algorithm will depend on the particular code in which it is written (Fortran, C++, Matlab, Python, Julia, etc.), the compiler used, the number of processors available, how efficiently they communicate, etc., and this will change in the future. For this reason, we will mainly focus on the structure of the algorithm rather than in its particular implementation for solving a given problem in a parallel computer. We first introduce the idea, show how to build some simple methods and illustrate their interest to be used as exponential integrators for solving stiff differential equations originated from spatial discretisation of parabolic problems.

Then, we preswent how one can easily analyse the accuracy and stability of the methods with an error bound analysis and, in some cases, we will show the performance of the new algorithms in the two extreme cases, i.e. in the worst scenario when it is evaluated as a sequential method versus the ideal one in which the cost of the whole algorithm can be taken as the cost for the computations in one single processor and neglecting the cost in the communication between processors. These results will be compared with the results obtained with sequential algorithms from the literature to solving the same problem. These results will illustrate the benefits one can achieve when the parallel algorithms are used under different conditions.

The efficient computation of an important number of functions of matrices of moderate size is of great interest in many different fields [1, 3, 4, 5, 7, 8, 9, 24, 26, 27, 28, 29, 35, 36, 39, 40, 41, 42, 43]. Frequently, it suffices to compute their action on a vector [2, 21, 29, 30, 31, 37] allowing to solve problems of large dimensions, or using an appropriate filtering technique the previous methods can also be used to compute functions of large sparse matrices [45]. For example, exponential integrators have shown to be highly efficient numerical schemes to solve linear systems of differential equations [6, 13, 14, 21, 22, 31, 32] but their performance depend on the existence of algorithms to compute accurately and cheaply the matrix exponential and related matrix functions, or their action on vectors.

Given a matrix $A \in \mathbb{C}^{d\times d}$, the goal is to compute $f(A)$ where $f(x)$, with $x \in \mathbb{C}$, is an analytic function near the origin, e.g. $e^x, \cos(x), \sin(x), \log(1+x)$ or the φ-functions, i.e. $\varphi_k(x) = (e^x - p_k(x))/x^k$ where p_k is the kth order Taylor polynomial to the exponential.

To compute $f(A)$, in this work we consider the case in which

$$(I+\alpha A)^{-1} \qquad \text{or} \qquad (I+\alpha A)^{-1}\mathbf{v},$$

with α a sufficiently small scalar and $\mathbf{v}$ a given vector, can be efficiently computed. Notice that:

- If A, B are two dense matrices: It is well known that $(I+\alpha A)^{-1}$ can be computed at the same cost as the product AB while the product

$(I + \alpha A)^{-1}B$ can be computed at 4/3 times the cost of AB (using an LU decomposition and solving d upper and lower triangular systems with a total number of $\frac{8}{3}d^3$ flops or, equivalently, a total cost similar to 4/3 matrix-matrix products).

- If A is a dense triangular matrix: It is well known that $(I + \alpha A)^{-1}$ can also be computed at the same cost as the product of two dense triangular matrices. The cost to solve the linear system $(I+\alpha A)\mathbf{w} = \mathbf{v}$, by backward or forward substitution, is exactly the same as the product $A\mathbf{v}$.

- If A is a large and sparse matrix: the solution of $(I + \alpha A)^{-1}\mathbf{v}$ can be efficiently carried out in many cases using, for example, incomplete LU or Choleski factorization, the conjugate (or bi-conjugate) gradient method with preconditioners, etc. [24, 44]. For example, if A is tridiagonal (or pentadiagonal) then $(I + \alpha A)$ is also tridiagonal (or pentadiagonal) and, as we will see in more detail, the system $(I + \alpha A)\mathbf{x} = \mathbf{v}$ can be solved with only $8d$ flops ($15d$ flops for pentadiagonal matrices) which can be considered very cheap since the product $A\mathbf{v}$ already needs $5n$ flops ($9d$ flops for pentadiagonal matrices).

- Quantum computation emerged about two decades ago and recently this is a field of enormous research interest. It is claimed that a high performance can be achieved for solving linear algebra problems of large dimension [10, 25]. For instance, in [10] it is mentioned: *"...quantum computers have been shown to perform common linear algebraic operations such as Fourier transforms, finding eigenvectors and eigenvalues, and solving linear sets of equations over* $2n$*-dimensional vector spaces in time that is polynomial in* n*, exponentially faster than their best known classical counterparts."*

Two steps are frequently considered when computing most functions, $f(A)$, or their action on vectors:

- If the norm of the matrix A is not sufficiently small, an scaling is usually applied that depends on the function to be computed. For example, to compute e^A one can consider $e^{A/N}$ with N such that the norm of A/N is smaller than a given value and then $e^{A/N}$ is accurately approximated. If $N = 2^s$, then s squaring are finally applied. For trigonometric functions, alternative recurrences like the double angle formula can be applied, etc. If, given $\mathbf{v} \in \mathbb{C}^d$, one is interested to compute $e^{A/N}\mathbf{v}$ the recurrence $\mathbf{v}_n = e^{A/N}\mathbf{v}_{n-1}, n = 1, 2, \ldots, N$, with $\mathbf{v}_0 = \mathbf{v}$ is applied.

- One has to compute the scaled function, say $f(B)$ with B depending on A, e.g. $B = A/N$, that can be written as a power series expansion

$$f(B) = \sum_{k=0}^{\infty} a_k B^k. \tag{1}$$

Then, high order rational Chebyshev or Padé approximants, or polynomial approximations are frequently considered following some tricks that

allow to carry their computations with a reduced number of operations [1, 8, 22, 40]. For example, a Taylor polynomial of degree 18 can be computed with only 5 matrix-matrix products [8] or a diagonal Padé approximation, that approximates e^x up to order x^{26}, can be computed with only 6 matrix-matrix products and one inverse [1]. On the other hand, the computation $f(B)\mathbf{v}$ is frequently carried out using Taylor or Krylov methods [2, 22, 30, 31] because the scaling-squaring technique can not be used in this case.

In some cases the numerical methods to solve these problems have some stages which can be computed in parallel and this is considered as an extra bonus of the method. However, we are interested on numerical schemes that are built from the very beginning to be used in parallel. The goal of this work is to present a procedure that allows to approximate any function as a linear combination of simple functions that can be evaluated independently so, they can be computed in parallel. In addition, they can be used to approximate simultaneously several functions of matrices too. We will also show how similar schemes can be used to compute the action of these functions on vectors.

1.1 Fractional decomposition

A technique that has already been used in the literature is the approximation of functions by rational approximations [17, 18, 20, 21, 22, 23, 43] in which we can apply a fractional decomposition. Given a function $f(x)$, it can be approximated by a rational function, $r_{n,m}(x)$, such that $r_{n,m}(x) \simeq f(x)$ for a range of values of x, where $r_{n,m}(x) = \frac{p_n(x)}{q_m(x)}$, and $p_n(x), q_m(x)$ are polynomials of degree n and m, respectively. One can then consider the fractional decomposition

$$f(x) \simeq r_{n,m}(x) = \frac{p_n(x)}{q_m(x)} = \sum_{i=1}^{m} w_i \frac{1}{x - c_i} + s_{n-m}(x),$$

where $s_{n-m}(x)$ is a polynomial of degree $n - m$ if $n \geq m$ or 0 otherwise, and the right hand side can be computed in parallel. If $n - m > 2$ the cost can be dominated by the cost to evaluate the polynomial $s_{n-m}(x)$.

The choice of the polynomials $p_n(x), q_m(x)$ depends on the particular methods used, i.e. rational Padé or Chebyshev approximations, and the main trouble is that for most functions of practical interest the roots of $q_m(x)$ (the coefficients c_i) are complex, making the computational cost about four times more expensive. This can be partially solved if one considers an incomplete fractional decomposition [17]. Since the complex roots of $q_m(x)$ occur in pairs, one can decompose it as follows

$$f(x) \simeq r_{n,m}(x) = \frac{p_n(x)}{q_m(x)} = \sum_{i=1}^{m_1} w_i \frac{1}{x - c_i} + \sum_{i=m_1+1}^{m_1+m_2} \frac{2Re(w_i)x - 2Re(w_i c_i^*)}{x^2 - 2Re(c_i)x + |c_i|^2} + s_{n-m}(x),$$

where $m = m_1 + 2m_2$ and $c_1, \ldots, c_{m_1}$ are the real roots. Then, some processors have to compute one product, x^2, and one inverse which altogether is nearly

twice the cost of one inverse. To compute the action of the matrix function on a vector, the incomplete fractional decomposition is, in general, not appropriate.

Notice that for each function one has to find the polynomials $p_n(x), q_m(x)$ for different values of n and m, and then to evaluate the fractional decomposition, making this procedure less attractive

We simplify this procedure using only real coefficients and making the search for the coefficients of the fractional decomposition trivial for most functions as well as to show how to adapt the procedure when the matrix A has different properties. The paper is organised as follows: Section 2 presents the main idea to build the methods and we show their interest on some numerical examples for stiff problems originated from parabolic PDEs. An error analysis is presented in Section 3. Section 4 illustrates how to build some particular methods which are numerically tested in Section 5. Finally, Section 6 collects the conclusions as well as future work. This work is an updated version of the unpublished report [12]

2 Approximating functions by simple fractions

The main idea of this work is quite simple, notice that

$$F_i(B) = (I - c_i B)^{-1} = \sum_{k=0}^{\infty} c_i^k B^k$$

for sufficiently small values of $\|c_i B\|$ and whose computational cost, as previously mentioned, can be considered, for general dense matrices, as one matrix-matrix product. Notice that each function F_i, for different values of c_i, can be computed in parallel and then, if P processors are available, we can compute

$$r_s^{(P)}(x) = \sum_{i=1}^{P} b_i F_i(x) \tag{2}$$

where

$$f(x) - r_s^{(P)}(x) = \mathcal{O}(x^{s+1}) \tag{3}$$

with $s = P - 1$ if the coefficients are chosen such that $c_i \neq c_j$ for $i \neq j$ and the coefficients b_i solve the following simple linear system of equations

$$\sum_{i=1}^{P} b_i c_i^k = a_k, \qquad k = 0, 1, \ldots, P-1, \tag{4}$$

where the coefficients a_k are known from (1). Alternatively, one can also choose the coefficients b_i such that

$$|f(x) - r_s^{(P)}(x)| < \varepsilon, \qquad x \in [a, b] \tag{5}$$

similarly to Chebyshev methods, or a combination of both conditions (high accuracy near the origin and a bounded error in a given interval).

Obviously, once the functions F_i are computed, different functions $f(B)$ (with different values for the coefficients a_k) can be simultaneously approximated just by looking for a new set of coefficients b_i in (2), say $\hat{b}_i$, such the corresponding equations (4) with the new coefficients a_k are satisfied. It remains the problem on how to choose the best set of coefficients, c_i, for each function, and this will depend on the number of processors available, P, the accuracy or stability desired, etc. If we choose $s+1 < P$ then $P-s-1$ coefficients b_i can be taken as free parameters for optimization purposes.

For example, the 4th-order diagonal Padé approximation to the exponential is given by

$$\begin{aligned} r_{2,2}(x) &= \frac{1+\frac{1}{2}x+\frac{1}{12}x^2}{1-\frac{1}{2}x+\frac{1}{12}x^2} = 1+\frac{6+i\,6\sqrt{6}}{x-(3-i\,\sqrt{3})}+\frac{6-i\,6\sqrt{6}}{x-(3+i\,\sqrt{3})} \\ &= 1+2\mathrm{Re}\left(\frac{6+i\,6\sqrt{6}}{x-(3-i\,\sqrt{3})}\right) \end{aligned} \tag{6}$$

with $r_{2,2}(x) = e^x + \mathcal{O}(x^5)$, but complex arithmetic is involved, and $r_{2,2}(x)$ is only valid to approximate the exponential function. Obviously, $r_{2,2}(x)$ can also be computed at the cost of one product, x^2, plus one inverse, $(1-x/2+x^2/12)^{-1}$. On the other hand, a 4th-order Padé approximation to the function $\varphi_1(x)$ is given by

$$\tilde{r}_{2,2}(x) = \frac{1+\frac{1}{10}x+\frac{1}{60}x^2}{1-\frac{2}{5}x+\frac{1}{20}x^2} = \frac{1}{3}+2\mathrm{Re}\left(\frac{\frac{7}{3}+i\,8}{x-(4-i\,2)}\right) \tag{7}$$

with $\tilde{r}_{2,2}(x) = \varphi_1(x) + \mathcal{O}(x^5)$ which has different complex roots.

However, if five processors are available we can take, for example

$$r_4^{(5)}(x) = \sum_{i=1}^{5} b_i \frac{1}{1-c_i x},$$

where the coefficients c_i can be chosen to optimise the performance of the method. One set of coefficients c_i can be optimal to get accurate results while other set of coefficients can be more appropriate when, for example, the matrix B is positive or negative definite (the coefficients c_i with appropriate size and sign can be chosen to optimize stability). If we take $c_1 = 0$ and $b_1 = a_0 - (b_2 + b_3 + b_4 + b_5)$ then only four processors are required. Once the values for c_i are fixed, the coefficients b_i are trivially obtained.

For example, if we choose $c_i = 1/(i+1),\ i = 1, \ldots, 5$ then the system (4) with $a_k = 1/k!,\ k = 0, 1, \ldots, 4$ has the solution

$$b_1 = \frac{1}{3}, \quad b_2 = -18, \quad b_3 = 128, \quad b_4 = -\frac{625}{3}, \quad b_5 = 99, \tag{8}$$

which corresponds to an approximation to the exponential that is already more accurate than the previous 4th-order Padé approximation (6). In addition, if one takes $a_k = 1/(k+1)!$, then we will obtain the solution

$$b_1 = \frac{7}{18}, \quad b_2 = -9, \quad b_3 = \frac{128}{3}, \quad b_4 = -\frac{500}{9}, \quad b_5 = \frac{45}{2}, \tag{9}$$

which corresponds to a 4th-order approximation to the function $\varphi_1(x)$, or a 4th-order approximation to the function $\log(1-x)$ for sufficiently small x can be obtained if we take

$$b_1 = -\frac{35}{3}, \quad b_2 = \frac{153}{2}, \quad b_3 = -160, \quad b_4 = \frac{625}{6}, \quad b_5 = -9,$$

although this set of coefficients c_i is not necessarily the optimal one for these functions.

If one is interested to compute the action of the function on a vector one has to consider

$$f(B)\mathbf{v} \simeq \sum_{i=1}^{P} b_i \mathbf{v}_i, \qquad \text{where} \qquad (I - c_i B)\mathbf{v}_i = \mathbf{v}$$

which, as previously, can be computed by solving linear systems of equations in each processor in parallel.

2.1 Functions of triangular and tridiagonal matrices acting on vectors

If B is a triangular matrix then it is well known that the cost to solve the triangular system $(I - c_i B)\mathbf{v}_i = \mathbf{v}$ is d^2 flops, exactly the same as the product $B\mathbf{v}$, making the new procedure of great interest, even in the worst case in which all computations are carried out sequentially.

Next, and just as an illustration of functions of sparse matrices acting on vectors, let us consider the computation $f(B)\mathbf{v}$ where B is a tridiagonal matrix. The product $B\mathbf{v}$ is done with $5d$ flops so, a Krylov or Taylor polynomial of degree K, requires $5d \times K$ flops, in addition to the evaluation of the function f for a matrix of dimension $K \times K$ for the Krylov methods. However, since $(I - c_i B)$ is also tridiagonal then $(I - c_i B)\mathbf{v}_i = \mathbf{v}$ can be solved with only $8d$ flops (using, for example, the Thomas algorithm). Then, for problems of relatively large size this is a significant saving in the computational cost (if the cost to communicate between processors can be neglected or is not dominant). Linear systems for tridiagonal matrices can also be solved in parallel (see, e.g. [38]), but this would require a second level of parallelism which is not considered in this work.

2.2 Illustrative numerical examples

To illustrate the interest of this procedure to build tailored methods for different classes of problems, let us consider the following parabolic PDE with source term and Dirichlet boudary conditions

$$\begin{cases} \dfrac{\partial u}{\partial t} = a^2 \dfrac{\partial^2 u}{\partial x^2} + f(x,t) \\ u(0,t) = g_L(t) \ , \quad u(1,t) = g_R(t) \\ u(x,0) = u_0(x) \end{cases} \qquad x \in [0,1], \quad t \geq 0. \tag{10}$$

We discretize the problem using a mesh of size $\Delta x = \frac{1}{N+1}$, and second order finite differences for the spatial derivative. This leads to a linear system of ODEs

$$\mathbf{u}' = A\mathbf{u} + \mathbf{b}(t), \qquad \mathbf{u}(0) = \mathbf{u}_0$$

$\mathbf{u}, \mathbf{b} \in \mathbb{C}^N$, $A = \frac{a^2}{\Delta x^2} trid\{1, -2, 1\} \in \mathbb{C}^{N\times N}$, and $\mathbf{b}(t)$ contains the source and boundary conditions. The solution can be written as

$$\mathbf{u}(t) = e^{tA}\mathbf{u}_0 + \int_0^t e^{(t-s)A}\mathbf{b}(s)ds, \tag{11}$$

and, if $\mathbf{b}$ is time independent, the solution can be written as

$$\mathbf{u}(t) = e^{tA}\mathbf{u}_0 + t\varphi_1(tA)\mathbf{b}.$$

We can measure the error due to the spatial discretisation at a given instant, t_f, as follows

$$E_x(t_f) = \Delta x \left(\sum_{i=1}^{N} \Big(\mathbf{u}_i(t_f) - u(x_i, t_f) \Big)^2 \right)^{1/2}.$$

The exact solution of the PDE, $u(x_i, t_f)$, can be easily evaluated to a sufficiently high accuracy e.g. using the separation of variables method and $\mathbf{u}(t_f)$ can be numerically computed to sufficiently high accuracy.

The homogeneous case. Let us first consider the case in which there are null boundary conditions and no source term, i.e. $\mathbf{b} = \mathbf{0}$. In this case, the solution is given by the action of the matrix exponential on the initial conditions. We then, consider different numerical methods, say R_h, to advance the solution as follows

$$\mathbf{u}_{n+1} = R_h\mathbf{u}_n \simeq e^{hA}\mathbf{u}_n$$

where $\mathbf{u}_n \simeq \mathbf{u}(t_n)$, $t_n = nh$, being h the time step.

Notice that the matrix A is tridiagonal with eigenvalues in the interval $\sigma(A) \in (-4\frac{a^2}{\Delta x^2}, 0)$ and contains very large negative eigenvalues making the problem very stiff.

We take $a^2 = \frac{5}{100}$ (so $\sigma(A) \in (-\frac{1}{5\Delta x^2}, 0)$) and initial conditions $u_0(x) = \frac{1}{2}(1 - \cos(2\pi x))$ so, the exact solution is

$$u(x,t) = \frac{8}{3\pi} e^{-a^2\pi^2 t} \sin(\pi x) + \sum_{n=3}^{\infty} \frac{4((-1)^n - 1)}{(n^3\pi - 4n\pi)} e^{-a^2 n^2 \pi^2 t} \sin(n\pi x).$$

We compute it with a sufficiently large number of terms and the numerical solution of the ODE system to sufficiently high accuracy at $t_f = 2$ and measure the error due to the spatial discretisation for different values of Δx.

We also evaluate the two-norm error at $t_f = 2$, with respect to the "exact" numerical solution of the ODE, for the following methods with their corresponding cost:

- $r_{2,2}(hA)$: the 4th-order Padé approximant.
- $r_4^{(5)}(hA)$: the new 4th-order fractional decomposition, with coefficients given in (8)
- $c_{4,4}(hA)$ the rational Chebyshev approximant. It is a quotient of two polynomials of degree four with coefficients given in [19]. The scheme has two complex poles and a bounded error to the function e^x in the whole negative real line.
- `ode15s`: The most frequently used MATLAB function to solve stiff differential equatrions. It uses a varialble order and variable time step implicit method for stiff differential equations. The function is used with different relative and absolute errors and the cost is measured taking into account the number of function evaluations as well as the number of LU factorisations and linear systems which are solved (considering them as tridiagonal systems).

Since the matrix A is tridiagonal, one matrix vector requires $5N$ flops which we take as the reference unit to measure the cost. Then, to solve a linear system is considered has a cost of $\frac{8}{5}$ products, and we consider fractional decomposition for the rational methods so, only tridiagonal systems have to be solved. This number is multiplied by a factor of four when complex coefficients are involved.

The computational cost is measured as if the algorithms were carried out in a sequential computer, i.e. to solve 5 linear systems per step for $r_4^{(5)}$, so the relative cost of this method would be considerably reduced if it was efficiently computed in a parallel computer.

Figure 1 (left) shows the results for $\Delta x = \frac{1}{20}$ (i.e. $N = 19$). Notice that for this spatial mesh Padé and the new method perform similarly and the rational Chebyshev approximation would be a good choice. We repeat the numerical experiments with $\Delta x = \frac{1}{200}$ (i.e. $N = 199$ and $\sigma(A) \in (-8000, 0)$) and the results are shown in the right pannel. The performance of the Padé method as well as the results with `ode15s` strongly depend on the chosen mesh while the Chebyshev as well as the new method is not much afected by this mesh refinement. While the Chebyshev method is valid in the first case, if the mesh size is reduced, the spatial error reduces and, in order to reach similar accuracy as the spatial error, rational Chebyshev approximants with higher degree polynomials should be used while the new very simple fractional decomposition with positive real poles still can be used.

Time dependent boundaries and non-homogeneous term. Let us now consider the case in which there are time-dependent boundary conditions and a time-dependent source term. In this case one has to approximate the solution (11). If we approximate the integral by the midpoint rule we get the method

$$\mathbf{u}_{n+1} = e^{hA}\mathbf{u}_n + he^{\frac{h}{2}A}\mathbf{b}_{n+1/2}$$

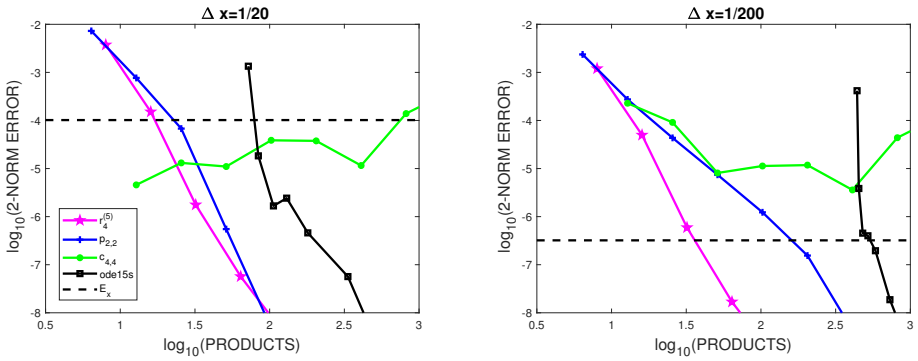

Figure 1: Error versus number of vector-matrix products for the homogeneous case at $t_f = 2$ using the 4th-order Padé, $r_{2,2}(hA)$, the fractional decomposition, $r_4^{(5)}(hA)$ with coefficients given in (8), and the rational Chebyshev $c_{4,4}$. Horizontal lines correspond to the spatial errors: for $\Delta x = \frac{1}{20}$ (left) and for $\Delta x = \frac{1}{200}$ (right). We also include the results provided by the MATLAB function for stiff problems `ode15s`.

with $\mathbf{b}_{n+1/2} = \mathbf{b}(t_n + h/2)$. This scheme coincides with a second order exponential Runge-Kutta method reccommended in [31] when applied to this non-homogeneous linear problem. Improved results can be obtained using a 4th-order quadrature rule. If we take the Gaussian nodes, $c_1 = \frac{1}{2} - \frac{\sqrt{3}}{6}$, $c_1 = \frac{1}{2} + \frac{\sqrt{3}}{6}$, we get the method

$$\mathbf{u}_{n+1} = e^{hA}\mathbf{u}_n + \frac{h}{2}\left(e^{h(1-c_1)A}\mathbf{b}_1 + e^{h(1-c_2)A}\mathbf{b}_2\right). \tag{12}$$

with $\mathbf{b}_i = \mathbf{b}(t_n + c_i h)$, $i = 1, 2$. Notice that each term in the right hand side can be computed in separately and, at the same time, each matrix exponential acting on a vector can also be computed in aprallel using the fractional decomposition.

Alternatively, one can also use a 4-th-order commutator-free Magnus integrator. For example, a 2-exponential commutator-free Magnus integrator applied to this problem reads as follows [16]

$$\mathbf{v} = e^{\frac{h}{2}A}\mathbf{u}_n + \frac{h}{2}\varphi_1(\frac{h}{2}A)\Big(\alpha_1\mathbf{b}_1 + \alpha_2\mathbf{b}_2\Big) \tag{13}$$

$$\mathbf{u}_{n+1} = e^{\frac{h}{2}A}\mathbf{v} + \frac{h}{2}\varphi_1(\frac{h}{2}A)\Big(\alpha_2\mathbf{b}_1 + \alpha_1\mathbf{b}_2\Big) \tag{14}$$

with $\alpha_1 = \frac{3-2\sqrt{3}}{12}$, $\alpha_2 = \frac{3+2\sqrt{3}}{12}$, which requires to compute the action of the matrix exponential and the φ_1-function on vectors in two consecutive stages

Notice also that in the computation of (13)-(14) using the fractional decomposition methods one can compute both functions simultaneously. Given $\mathbf{v}, \mathbf{w} \in \mathbb{R}^N$ we have that

$$e^{\alpha A}\mathbf{v} + \varphi_1(\alpha A)\mathbf{w} \simeq \sum_{i=1}^{5} (I - c_i\alpha A)^{-1}\left(b_i\mathbf{v} + \hat{b}_i\mathbf{w}\right)$$

where the coeficients b_i are given in (8) while the coeficients $\hat{b}_i$ are the coefficients given in (9) so, both contributions can be simultaneously computed without the requirement of additional processors. In [34] the author presents high order exponential Runge-Kutta methods where this simultaneously computations can be applied. Since the roots for the Padé approximations to the exponential and φ-functions are different, this property can not be applied.

We take for the numerical experiments $a = 1$ and

$$f(x,t) = 50x(1-x)e^{-t}, \quad u(0,t) = 1-e^{-t}, \quad u(1,t) = 2(1-e^{-t}), \quad u(x,0) = x(1-x)$$

so, the exact solution is given by

$$u(x,t) = \sum_{n=1}^{\infty} (a_n e^{-n^2\pi^2 t} + b_n(t)) \sin(n\pi x)$$

where

$$a_n = \frac{4(1-(-1)^n)}{n^3\pi^3}, \quad b_n(t) = \frac{2(100 - n^2\pi^2 + 2(-1)^n(n^2\pi^2 - 50)}{n^3\pi^3(n^2\pi^2 - 1)}(e^{-t} - e^{-n^2\pi^2 t}).$$

We consider the 4th-order CF Magnus integrator since it can be easily adapted to the case in which the matrix A is also time-dependent and its structure is closer to the frequently used exponential Runge-Kutta methods. The matrix exponential is approximated both using $r_{2,2}$ and $r_4^{(5)}$ and their corresponding 4th-order approximations for the φ_1 function. We measure their cost as the cost for solving one linear system per stage (ideal parallel computation) but the cost of the Padé is multiplied by four since it involves complex coefficients.

For this particular problem, the main error is dominated by the action of the φ_1 function on vectors (originated from the source and boundary conditions). This error can be reduced using a more accurate approximation to this function, up to the point in which the error is dominated by the method, which is larger than the error from the approximations to the matrix exponential. Obviously, a round off error analysis for the method $r_4^{(5)}$ due to the relatively large values of its coefficients b_i is not relevant for this application.

The Chebyshev method is not included since it requires an approximation to the φ-function. The results from the `ode15s` are neither included since this method uses variable order and time step and it is difficult to make a fair comparision for this problem. Finally, as a reference we also include the results obtained by (12) when the matrix exponentials are approximated by the $r_4^{(5)}$ method and at the cost of solving only one linear system (ideal parallel computation with 15 processors), $R_4^{(5)}$. Figure 2 shows he results at $t_f = \frac{1}{10}$ obtained for $\Delta x = \frac{1}{20}$ and $\Delta x = \frac{1}{80}$.

For this problem it is clear that the new method shows the best performance since it provides sufficient accuracy at a cheapest price than other approximaitons to the matrix exponential.

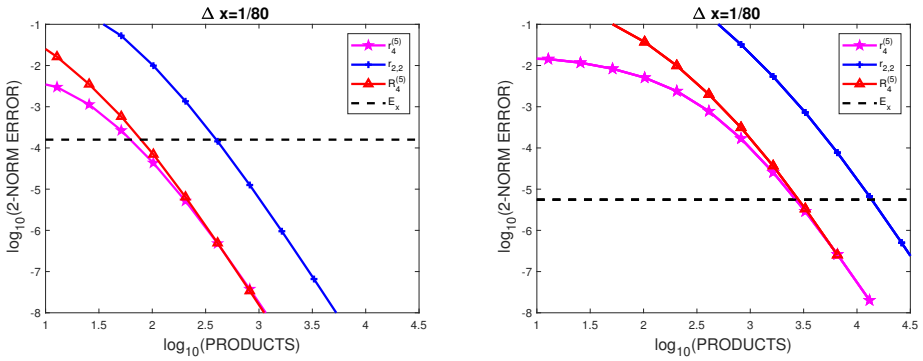

Figure 2: Error versus number of vector-matrix products for the non-homogeneous case at $t_f = \frac{1}{10}$ using $r_{2,2}$ and $\tilde{r}_{2,2}$ for the exponential and φ_1 functions and $r_4^{(5)}$ with coefficients given in (8) and (9). Horizontal lines correspond to the spatial errors. Left: for $\Delta x = \frac{1}{20}$ and Right: for $\Delta x = \frac{1}{80}$. We also include the results provided when considering (12) and the matrix exponentials are approximated by the $r_4^{(5)}$ method taking its the cost as solving only one linear system per step, $R_4^{(5)}$.

3 Error analysis

Since this new procedure can be used to build tailored methods for different classes of problems, it is convenient to consider an error analysis. Forward and backward error analysis can be easily carried out for the new classes of approximations. For example, if one is interested in forward error bounds, we have that

$$\|f(B) - r_s^{(P)}(B)\| = \|\sum_{k=0}^{\infty}(a_k - \alpha_k)B^k\| \leq \sum_{k=0}^{\infty}|a_k - \alpha_k|\|B\|^k$$

where $\alpha_k = \sum_{i=1}^{P} b_i c_i^k$. Since, in general, the coefficients a_k, b_i, c_i are known with any desired accuracy, one can take a sufficiently large number of terms in the summation to compute the error bounds with several significant digits. For different values of the tolerance, ϵ, we can numerically find values, say θ_s, such that if $\|B\| < \theta_s$ the error is below ϵ. Obviously, sharper error bounds can be obtained if additional information of the matrix is known, e.g. when the bounds are written in terms of $\|B^k\|^{1/k}$ for some values of k (see, e.g. [1]).

Alternatively, error bounds from the backward error analysis can be obtained if one considers that $r_s^{(P)}(B)$ can formally be written as

$$r_s^{(P)}(B) = f(B + \Delta B)$$

where

$$\Delta B = f^{-1}(r_s^{(P)}(B)) - B = h_s(B) = \sum_{k=0}^{\infty}\beta_k B^k$$

and $\beta_k = 0$ for $k = 0, 1, \ldots, s$ since $r_s^{(P)}(x)$ coincides with the Taylor expansion of $f(B)$ up to order s, but this is not necessarily the case in schemes like Chebyshev approximations. Then, it is clear that $\|h_P(B)\| \leq \tilde{h}_s(\|B\|)$ where

$$\tilde{h}_P(x) = \sum_{k=0}^{\infty} |\beta_k| x^k,$$

and thus

$$\frac{\|\Delta B\|}{\|B\|} = \frac{\|h_P(B)\|}{\|B\|} \leq \frac{\tilde{h}_P(\|B\|)}{\|B\|}.$$

These previous error analysis can be easily carried for different functions $f(B)$ and for different choices of the coefficients c_i and their associated values for the coefficients b_i.

However, one has to take into account that the coefficients b_i will strongly depend on the choice of the coefficients c_i. On one side, we have that taking very small values for the coefficients c_i usually allows to apply the methods for matrices with large norms, but then the solution of the linear system of equations (4) will have, in general, large values for the coefficients b_i, and this can cause badly conditioned methods when high accuracy (say, near to round off errors) is desired. This can be partially solved taking additional processors or slightly modifying the methods as we will see.

4 Illustrative examples: Exponential and φ-functions

We illustrate how to build and optimise some approximations of the matrix exponential and the φ_1-function taking into account the error analysis when only a small number of processors are available and relatively low order methods are considered. For simplicity in the presentation we will only consider forward error bounds.

4.1 4th-order approximations

One of the best 4th-order methods to approximate the the matrix exponential is the $r_{2,2}(x)$ Padé approximation which in a serial algorithm needs one product and one inverse and similarly for $\tilde{r}_{2,2}(x)$ to approximate the φ_1-function. A forward error bound is given by

$$\|e^B - r_{2,2}(B)\| \leq \epsilon_{4,pad}(\|B\|), \qquad \|\varphi_1(B) - \tilde{r}_{2,2}(B)\| \leq \tilde{\epsilon}_{4,pad}(\|B\|)$$

where

$$\epsilon_{4,pad}(x) = \sum_{k=5}^{\infty} \left| \frac{1}{k!} - d_k \right| x^k, \qquad \tilde{\epsilon}_{4,pad}(x) = \sum_{k=5}^{\infty} \left| \frac{1}{(k+1)!} - \tilde{d}_k \right| x^k.$$

Here, d_k and $\tilde{d}_k$ are the coefficients from the Taylor expansion of $r_{2,2}(x)$ and $\tilde{r}_{2,2}(x)$, respectively.

Let us now consider, for simplicity in the search of an optimized parallel method for a general problem, the following choice for the coefficients c_i for an scheme using four processors:

$$c_1 = 0, \quad c_2 = \frac{1}{\alpha}, \quad c_3 = -\frac{1}{\alpha}, \quad c_4 = \frac{1}{2\alpha}, \quad c_5 = -\frac{1}{2\alpha}, \tag{15}$$

where the coefficients b_i are trivially obtained. Then, the proposed method to be used with four processors is written in terms of one free parameter, α. The forward error bound is given by

$$\|e^B - r_4^{(4)}(B)\| \leq \epsilon_4(\|B\|), \qquad \|\varphi_1(B) - \tilde{r}_4^{(4)}(B)\| \leq \tilde{\epsilon}_4(\|B\|)$$

where

$$\epsilon_4(x) = \sum_{k=5}^{\infty} \left| \frac{1}{k!} - d_k(\alpha) \right| x^k, \qquad \tilde{\epsilon}_4(x) = \sum_{k=5}^{\infty} \left| \frac{1}{(k+1)!} - \tilde{d}_k(\alpha) \right| x^k.$$

Here, $d_k(\alpha) = \sum_{i=2}^{5} b_i c_i^k$ and $\tilde{d}_k(\alpha) = \sum_{i=2}^{5} \tilde{b}_i c_i^k$ are the coefficients from the Taylor expansion of $r_4^{(4)}(x)$ and $\tilde{r}_4^{(4)}(x)$, respectively, which depend on α. A simple search shows that the optimal solution which minimises $\epsilon_4(x)$ for most values of x about the origin occurs (approximately) for $\alpha = 5$, and then

$$b_1 = \frac{128}{3}, \quad b_2 = \frac{85}{3}, \quad b_3 = \frac{20}{9}, \quad b_4 = -\frac{515}{9}, \quad b_5 = -15. \tag{16}$$

For this choice of α we can still get an approximation for the φ_1 function that is nearly as accurate as the previous Padé approximation. The optimal choice for this function is however (approximately) for $\alpha = 6$, and then

$$b_1 = \frac{71}{5}, \quad b_2 = \frac{117}{10}, \quad b_3 = \frac{7}{10}, \quad b_4 = -\frac{104}{5}, \quad b_5 = -\frac{24}{5},$$

which gives more accurate results than the Padé scheme.

In Fig. 3, left panel, we show the values of the error bounds, $\epsilon_{4,pad}(x)$, $\tilde{\epsilon}_{4,pad}(x)$, $\epsilon_4(x)$ and $\tilde{\epsilon}_4(x)$, versus x. The thick lines correspond to the 4th-order approximations to the exponential function and the thin lines correspond to the φ_1-function when using Padé approximations (dashed lines) and the new fractional approximations (solid lines). Then, given a tolerance, tol, one can easily find the value θ such that $\epsilon(x) < tol$ for $x < \theta$, i.e. $\epsilon(\|B\|) < tol$ for $\|B\| < \theta$ where ϵ denotes the desired error bound. It is clear that the new (not fully optimized) methods are more accurate for all values of $\|B\|$ and are also faster to be computed when done in parallel.

4.2 8th-order approximations

In a serial computer, the 8th-order Taylor approximation can be computed with only 3 matrix-matrix products as follows [8]

$$\begin{aligned} B_2 &= B^2, \\ B_4 &= B_2(x_1 B + x_2 B_2), \\ B_8 &= (x_3 B_2 + B_4)(x_4 I + x_5 B + x_6 B_2 + x_7 B_4), \\ T_8(B) &= y_0 I + y_1 B + y_2 B_2 + B_8, \end{aligned} \tag{17}$$

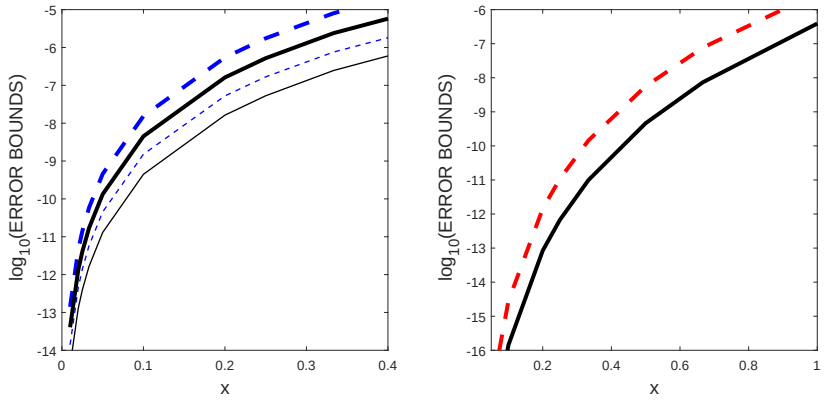

Figure 3: Left: Error bounds for the 4th-order approximations to the exponential function (thick lines) and to the φ_1-function (thin lines) using Padé approximations (dashed lines) and the fractional approximations (solid lines). Right: Error bounds for the 8th-order approximations to the exponential function using the Taylor approximation (dashed line) and the fractional approximation (solid line).

with

$$x_1 = x_3\frac{1+\sqrt{177}}{88}, \qquad x_2 = \frac{1+\sqrt{177}}{352}x_3, \qquad x_4 = \frac{-271+29\sqrt{177}}{315x_3},$$
$$x_5 = \frac{11(-1+\sqrt{177})}{1260x_3}, \quad x_6 = \frac{11(-9+\sqrt{177})}{5040x_3}, \quad x_7 = \frac{89-\sqrt{177}}{5040x_3^2},$$
$$y_0 = 1, \qquad y_1 = 1, \qquad y_2 = \frac{857-58\sqrt{177}}{630},$$
$$x_3 = 2/3,$$

being the most efficient scheme for this order of accuracy. In spite the coefficients x_i, y_i are not rational numbers one can check that T_8 is exactly the Taylor expansion to order 8, and a similar method can be obtained for the φ_1-function which we do not consider for simplicity. The forward error bound is given by

$$\|e^B - T_8(B)\| \le \epsilon_8^T(\|B\|), \qquad \text{where} \qquad \epsilon_8^T(x) = \sum_{k=9}^{\infty}\frac{1}{k!}x^k.$$

Let us take, for example

$$r_8^{(8)} = b_1 + \sum_{i=2}^{9} b_i\frac{1}{1-c_i x}$$

(with $c_1 = 0$) and

$$c_2 = \frac{1}{\alpha}, \quad c_3 = -\frac{1}{\alpha}, \quad c_4 = \frac{2}{3\alpha}, \quad c_5 = -\frac{2}{3\alpha},$$
$$c_6 = \frac{1}{2\alpha}, \quad c_7 = -\frac{1}{2\alpha}, \quad c_8 = \frac{2}{5\alpha}, \quad c_9 = -\frac{2}{5\alpha}, \tag{18}$$

to be used with eight processors and written in terms of one free parameter, α. The optimal value for this choice of coefficients c_i which minimize the forward error bound, $\epsilon_8(x)$ corresponds (approximately) to $\alpha = 5$, and then

$$b_1 = -\frac{9979069}{32256}, \quad b_2 = -\frac{1995521}{254016}, \qquad b_3 = -\frac{392009}{254016},$$

$$b_4 = \frac{520866369}{802816}, \quad b_5 = \frac{48898161}{802816}, \qquad b_6 = -\frac{26686735}{11907},$$

$$b_7 = -\frac{3892615}{11907}, \quad b_8 = \frac{353067578125}{195084288}, \quad b_9 = \frac{71873828125}{195084288}.$$

In the right panel of Fig. 3 we show the values of $\epsilon_8^T(x)$ (dashed line) and $\epsilon_8(x)$ for $\alpha = 5$ (solid line) for different values of x. We observe that this method, with such a simple optimization search, already provides more accurate results than the Taylor approximation being up to slightly more than twice faster.

4.3 High order approximations

From the error bounds analysis we can deduce that when high accuracy is desired, it is usually more efficient to consider higher order approximations rather than taking a larger value of N in the approximation $B = A/N$ and next to consider the recurrence like the squaring. This usually occurs up to a relatively high order.

However, in the construction of different methods we have noticed that in the optimization process for the fractional approximation, the best error bounds are obtained when small values of the coefficients c_i are taken and this usually leads to large values for the coefficients b_i whose absolute values typically grow with the order of accuracy (the coefficients b_i of our 8th-order approximation are considerably larger in absolute value than the corresponding coefficients for the 4th-order method). It is well known that the equations for the coefficients b_i is badly conditioned, but it still can allow to obtain methods of practical interest as we have already seen.

Then, if nearly round off accuracy is desired, e.g. double precision, the new schemes could not be well conditioned and can suffer from large round off errors. This problem can be partially reduced in different ways. If the computation of $(I - c_i B)^{-1} I$ has similar cost as $(I - c_i B)^{-1} B$ we can take into account that $\frac{1}{1-c_i x} = 1 + \frac{c_i x}{1-c_i x}$ and we have that

$$r_s^{(P)}(x) = a_0 + \sum_{i=1}^{P} b_i \frac{c_i x}{1 - c_i x} \tag{19}$$

where we have considered that $\sum_{i=1}^{P} b_i = a_0$. This decomposition usually has smaller round-off errors for relatively small values of $|c_i x|$ which is usually the case when close to round off errors are desired.

Alternatively, if there is no restriction on the number of processors, one can consider in (3) $P > s + 1$ allowing for $P - s - 1$ free coefficients b_i that can

be chosen jointly with appropriate values for the c_i's to reduce round off errors. We can end up with an optimisation problem with many free parameters (the coefficients c_i in addition to $P-s-1$ coefficients b_i) and we leave this as an interesting open problem to be analysed for different functions of matrices, number of processors used and choices of the order of accuracy. In [33] the authors show how to optimise the search of the coefficients in some approximations of functions of matrices that hopefully could be used for this problem too.

There are also hybrid methods that could be used to reduce round off errors. For example, we can consider

$$r_s^{(P)}(x) = d_0 + d_1 x + d_2 x^2 + \sum_{i=1}^{P-1} b_i \frac{1}{1-c_i x}, \tag{20}$$

which can be considered as a generalisation of the fractional decomposition of $r_{n+2,n}$ rational Padé approximations (rational Padé approximations to the exponential, $r_{n,m}(x)$, with $n > m$ have recently shown to be superior to diagonal Padé schemes in most cases [15]). Here, $d_0 + d_1 x + d_2 x^2$ can be computed in one processor (at the cost of one matrix-matrix product, that is slightly cheaper than the inverse of a matrix) and the partial fractions are computed in the remaining $P-1$ processors. The coefficients b_i must satisfy (for $P \geq 2$)

$$\sum_{i=1}^{P-1} b_i c_i^k = a_k, \qquad k = 3, 4, \ldots, P+1$$

and

$$d_k = a_k - \sum_{i=1}^{P-1} b_i c_i^k, \qquad k = 0, 1, 2,$$

which allows to get methods up to order $s = P+1$ with P processors and, hopefully, smaller coefficients b_i in absolute value. It is then expected that a more careful search will lead to new schemes with considerably reduced round off errors.

We illustrate this procedure in the search of some 10th-order methods for the matrix exponential.

4.3.1 10th-order approximations to the exponential

The 10th-order diagonal Padé approximation which is given by [26] and is used as one of the methods implemented in the function `expm` of MATLAB is:

$$(-u_5+v_5) r_{5,5}(A) = (u_5+v_5), \quad u_5 = A[b_5 A_4 + b_3 A_2 + b_1 I], \quad v_5 = b_4 A_4 + b_2 A_2 + b_0 I, \tag{21}$$

where $A_2 = A^2, A_4 = A_2^2$, $(b_0, b_1, b_2, b_3, b_4, b_5) = (1, \frac{1}{2}, \frac{1}{9}, \frac{1}{72}, \frac{1}{1008}, \frac{1}{30240})$. It requires 3 products and one inverse (approximately three times more expensive than the inverse to be evaluated by each processor on a parallel method). Note that the Padé approximation for the φ_1-function has not the same symmetry for the numerator and denominator and has to be computed with four products

and one inverse, i.e. four times more expensive that one inverse (and it has different roots).

An approximation by using the proposed fractional methods to order ten requires 11 processors (10 processors if one takes, e.g. $c_1 = 0$ or 9 processors if one considers (20)). Suppose we have two extra free coefficients b_i in the composition (20) (11 processors in total) to have some freedom to improve the accuracy of the method (this is just a very simple illustrative example that has not been exhaustively optimised)

$$r_{10}^{(11)}(x) = d_0 + d_1 x + d_2 x^2 + \sum_{i=1}^{10} b_i \frac{1}{1 - c_i x}, \tag{22}$$

where, given some values for the coefficients c_i we take as free parameters b_9, b_{10} and solve the following linear systems of equations for the remaining coefficients:

$$\sum_{i=1}^{10} b_i c_i^k = \frac{1}{k!}, \qquad k = 3, 4, \ldots, 10$$

which provides $b_1, \ldots, b_8$, and

$$d_k = \frac{1}{k!} - \sum_{i=1}^{10} b_i c_i^k, \qquad k = 0, 1, 2.$$

We have taken

$$c_i = \frac{1}{6+i}, \quad i = 1, 2, \ldots, 10; \qquad b_9 = -50000, \quad b_{10} = 350000, \tag{23}$$

being the remaining coefficients

$$\begin{aligned}
d_1 &= -\frac{34244933704346617}{14676708556800}, & d_2 &= -\frac{18541933026870559}{428070666240000}, \\
d_3 &= -\frac{6798371473106351}{15410543984640000}, & & \\
b_1 &= -\frac{2385751622325187262153}{1585084524134400000}, & b_2 &= \frac{3752603696192081}{82668600000}, \\
b_3 &= -\frac{87230538798639033213}{187904819200000}, & b_4 &= \frac{14789110319838821875}{6934744793088}, \\
b_5 &= -\frac{10558563753676365149296981}{2219118333788160000}, & b_6 &= \frac{3520134037769971}{716800000}, \\
b_7 &= -\frac{22630572997141151810195 09}{1585084524134400000}, & b_8 &= -\frac{2275831141003773874927}{3095868211200000}.
\end{aligned} \tag{24}$$

This method has an error bound smaller than the error bound provided by the Padé approximation by more than one order of magnitude while being up to three times cheaper in a parallel computer. However, note that

$$\max\{|b_i|, |d_i|\} \simeq 4.9 \cdot 10^6$$

so, one can expect large cancellations leading to large round off errors.

This problem can be partially solved by choosing different values for the coefficients c_i. Note that we have much freedom in this choice. For example, if we take the following values:

$$c_{2i-1} = -\frac{1}{6+2i},\ c_{2i} = \frac{1}{6+2i}, \quad i = 1,2,3,4,5, \qquad b_9 = 2000, \quad b_{10} = -3500, \tag{25}$$

then we get

$$d_1 = -\frac{781562376863}{94371840}, \qquad d_2 = -\frac{54849495983}{330301440}, \qquad d_3 = -\frac{8034429391}{587202560}, \tag{26}$$

$$b_1 = -\frac{57383239}{760320}, \qquad b_2 = -\frac{115498838729}{239500800}, \qquad b_3 = \frac{1648441938671875}{1255673954304},$$
$$b_4 = \frac{56790060546875}{4227858432}, \qquad b_5 = -\frac{1476772203681}{298188800}, \qquad b_6 = -\frac{31012455666807}{656015360},$$
$$b_7 = \frac{4891212112962371}{1295536619520}, \qquad b_8 = \frac{171190903245297593}{3886609858560},$$

where

$$\max\{|b_i|, |d_i|\} \simeq 4.7 \cdot 10^4.$$

The forward error bound is slightly worst for this method but, as we will observe in the numerical experiments, round off errors are reduced by nearly three orders of magnitude.

Obviously, a deeper search taking into account the freedom in the choice of the coefficients c_i as well as the possibility to take more processors with more free parameters b_i and c_i should allow us to considerably reduce both the error bounds as well as the round off error.

5 Numerical examples

Given a matrix, A, to compute a function, $f(A)$, most algorithms first evaluate a bound to a norm, say $\|A\|$ or $\|A^k\|^{1/k}$ for some values of k and then, according to its value and a tabulated set of values obtained from the error bound analysis, it is chosen the method to be used that gives a result with (hopefully) an error below the desired tolerance.

To test the algorithms on particular problems allows us to check if the error bounds are sufficiently sharp as well as to observe how big the undesirable round off errors are.

In the following numerical experiments we only compute approximations to e^{hA} or its action on vectors for different values of h using the following methods

- $r_{2,2}$: the 4th-order Padé approximation (6). Cost: one product and one inverse.
- T_8: the 8th-order Taylor approximation, $T_8(x)$ (17). Cost: three products.
- T_5, T_{10}: the 5th- and 10th-order Taylor approximations used for the action of the exponential on a vector. Cost: five and ten vector-matrix products, respectively.

- $r_{5,5}$: the 10th-order Padé approximation (21). Cost: three products and one inverse.

- R_4: the 4th-order rational approximation

$$r_4^{(4)}(x) = \sum_{i=1}^{5} b_i \frac{1}{1 - c_i x},$$

  with coefficients given in (15) and (16) for $\alpha = 5$, designed to be used with four processors.

- R_5: the 5th-order rational approximation, $r_5^{(5)}(x)$, with coefficients given in (28), designed to be used with five processors.

- R_8: the 8th-order rational approximation

$$r_8^{(8)}(x) = \sum_{i=1}^{9} b_i \frac{1}{1 - c_i x},$$

  with coefficients given in (18) for $\alpha = 5$, designed to be used with eight processors.

- R_{10}^*: the 10th-order rational approximation (22) with coefficients given in (23) and (24), designed to be used with eleven processors.

- R_{10}: the 10th-order rational approximation (22) with coefficients given in (25) and (26), designed to be used with eleven processors.

Exponential of a dense matrix. As a first numerical test we take

$$A = \texttt{randn(100)}$$

i.e. a 100×100 matrix whose elements are normally distributed random numbers and we compute e^{hA} for different values of h. We have done this numerical experiment repeatedly many times with very similar results.

Figure 4 shows in the left panel the two-norm error (the exact solution is computed numerically with sufficiently high accuracy) versus $h\|A\|$ for these methods. Dashed lines correspond to the results obtained with the diagonal Padé and Taylor methods and solid lines are obtained with the new rational methods. We observe that the new methods have in all cases similar or higher accuracy than the methods of the same order and built to be computed in a sequential algorithm. There is only a minor drawback in the 8th-order rational method when high accuracy is desired due to round off errors.

Note that the computational cost is not taken into account in the figures since this depends on how efficiently is implemented the rational method in a parallel computer, where the new methods can be up to several times faster to compute than the methods designed for serial algorithms.

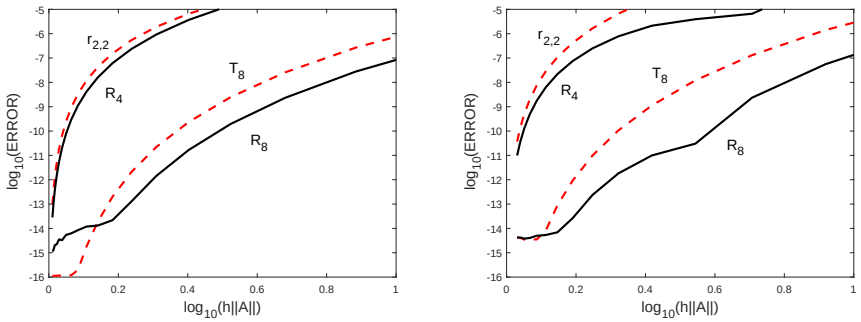

Figure 4: Left:Two-norm error (in logarithmic scale) to compute e^{hA} with A a random matrix of dimension $d = 100$ versus $h\|A\|$. Dashed lines correspond to the results obtained with the 4th-order diagonal Padé method and the 8th-order Taylor method and solid lines are obtained with the new rational methods of the same order. Right: The same with the matrix (27) of dimension $d = 100$.

We have repeated the numerical experiments with another 100×100 symmetric dense matrix with elements

$$A_{i,j} = \frac{1}{1 + (i - j)^2} \tag{27}$$

and the results are shown in the right panel in Figure 4. We observe that for this problem round off errors are similar for both 8th-order methods. We have repeated these numerical experiments for different dimensions of the matrices and other matrices with different structures and the results are, in general, qualitatively similar. Notice the results are in agreement with the error bounds shown in Figure 3.

We have repeated the numerical experiments using the 10th-order methods. Figure 5 shows the results obtained in a double logarithmic scale: $r_{5,5}$ (dashed lines), R^*_{10} (dotted lines) and R_{10} (solid lines). We observe the high accuracy of the method R^*_{10}, but with large round off errors. The scheme R_{10} shows slightly less accurate results but the round off errors are reduced about three orders of magnitude. Notice that the methods R^*_{10} and R_{10} can be up to three times faster than $r_{5,5}$ and this is not shown in the figure.

5.1 Performance of the exponential of triangular or tridiagonal matrices acting on vectors

Let us now consider the particular case (but of great practical interest) of computing the exponential of a tridiagonal matrix acting on a vector, $e^A\mathbf{v}$, with single precision accuracy and where we assume that, say $\|A\| \leq 1.5$. The same algorithms with minor changes in the computational cost of the methods apply to the computation of the exponential of pentadiagonal or banded matrices acting on vectors.

One of the most used schemes to compute the action of the matrix exponential is proposed in [2]. The algorithm works as follows: To compute $e^A\mathbf{v}$,

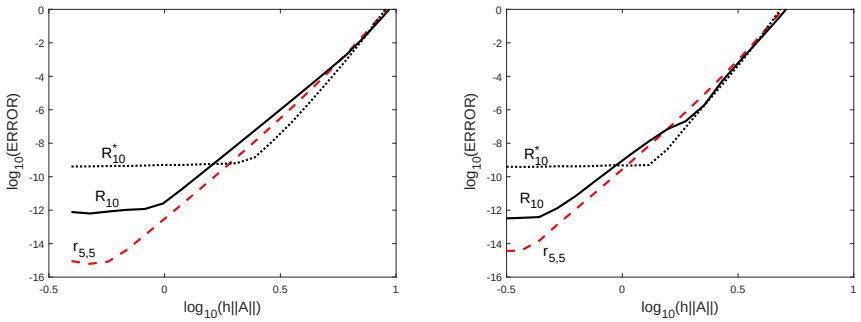

Figure 5: Left: Two-norm error to compute e^{hA} with A a random matrix of dimension $d = 100$ versus $h\|A\|$ in double logarithmic scale. Dashed lines correspond to the results obtained with the 10th-order diagonal Padé method, $r_{5,5}$, doted lines correspond to the 10th-order rational method R^*_{10} and solid lines correspond to the 10th-order rational method R_{10} (solid lines). Right: The same with the matrix (27) of dimension $d = 100$.

Table 1: Values of θ_m obtained from the forward error analysis for the Taylor approximation t_m of order m for single precision. The values for the backward error analysis used in the algorithm proposed in [2] are quite close but slightly shifted, but this also occurs for the rational methods.

m :	5	10	15
$u \leq 2^{-24}$	0.186	1.073	2.382

a set of Taylor polynomials of degree $m = 5k,\ k = 1, 2, \ldots, 11$ are considered. Given a tolerance (single or double precision), an estimation to the norm $\|A\|$ is computed (frequently $\|A\|_1$ is used), and from the error analysis, it is chosen the lowest degree Taylor polynomial among the list such that the desired accuracy is guaranteed. In [2], the choice of the method is done from the results obtained by the backward error analysis. Very similar results are obtained with the forward error analysis, which is simpler and for the convenience of the reader we will consider it. In Table 1 we collect the relevant values for our problem, i.e. if $\|A\| \leq 0.186$ the algorithm chooses the Taylor approximation of order five, if $0.186 < \|A\| \leq 1.073$ the tenth order method is used, etc. which guarantee that $\|e^A\mathbf{v} - T_m(A)\mathbf{v}\| < u \leq 2^{-24}$ where $T_m(A)$ is the Taylor polynomial approximation of degree m.

We build a similar scheme in order to analyse the interest of the new algorithms versus the State of the Art algorithm. We then need a 5th-order approximation that, without an exhaustive analysis, we take as the following one:

$$r_5^{(5)}(x) = b_1 + \sum_{i=2}^{6} b_i \frac{1}{1 - c_i x}.$$

If we take $c_1 = 0,\ c_i = 1/(i+1),\ i = 2, \ldots, 6$ then the system (4) has the

Table 2: Values of θ_m obtained from the forward error analysis for the fractional approximations $r_5^{(5)}$ and $r_{10}^{(11)}$ of order 5 and 10, respectively, for single precision. The cost m^* (for tridiagonal and triangular matrices) is measured as an interval in terms of the cost of the product $A\mathbf{v}$ to make easier the comparison with the Taylor methods. The lower limit of the interval corresponds to the ideal case where all calculations are carried in parallel and there is no cost to communicate between processors, and the upper limit corresponds to the same scheme fully computed as a serial method.

$m^*_{trid}:$	$(\frac{8}{5}, 8)$	$(2, 18)$
$m^*_{triang}:$	$(1, 5)$	$(2, 12)$
$u \leq 2^{-24}$	0.298	1.734

solution

$$b_1 = -\frac{43}{12}, \quad b_2 = \frac{81}{32}, \quad b_3 = -\frac{704}{9}, \quad b_4 = \frac{23125}{48}, \quad b_5 = -810, \quad b_6 = \frac{117649}{288}. \tag{28}$$

The forward error analysis tells us that the method provides an error below the tolerance,$u \leq 2^{-24}$, for $\|A\| \leq \theta_5 = 0.298$, as indicated in Table 2, being significantly greater than with the Taylor method. The computational cost to evaluate $r_5^{(5)}(A)\mathbf{v}$ corresponds to solve five tridiagonal systems. If we consider that each system is solved with $8d$ flows, this will correspond to the cost of $8/5$ times the cost of the products $A\mathbf{v}$. If the method is computed sequentially, we observe that the cost would be equivalent to 8 matrix-vector products, and these numbers correspond to the interval $(\frac{8}{5}, 8)$ shown in Table 2 as a measure of the cost in comparison with the Taylor method. For the 10th-order method we take the scheme (22)-(24) (the scheme with coefficients given in (25)-(26) have smaller round off errors but on the other hand it is slightly less accurate and it has a smaller value of θ_{10}). The cost and value of θ_{10} for this method is collected in Table 2.

The results from Tables 1 and 2 are illustrated in the left panel of Figure 6 where we plot the computational cost of each algorithm, measured in terms of the cost of a matrix-vector product, for different values of $\|A\|$ in the interval of interest. For the fractional methods (solid lines) the cost is measured both considering they are computed as a sequential scheme as well as if they were computed in an ideal parallel computer (the cost for one single processor). The Taylor methods correspond to the dashed line. We observe that the new methods are competitive even in the worst scenario and much better once some cost is saved due to the parallel programing. The excellent results obtained motivated us to make deeper analysis for this problem in order to find an optimised set of methods for different orders of accuracy similarly to the actual sequential scheme and to test them in multiprocessor workstations. This work will be carried out in the future.

We conclude this section with some remarks:

- If one is interested to compute the exponential e^A acting on several vectors

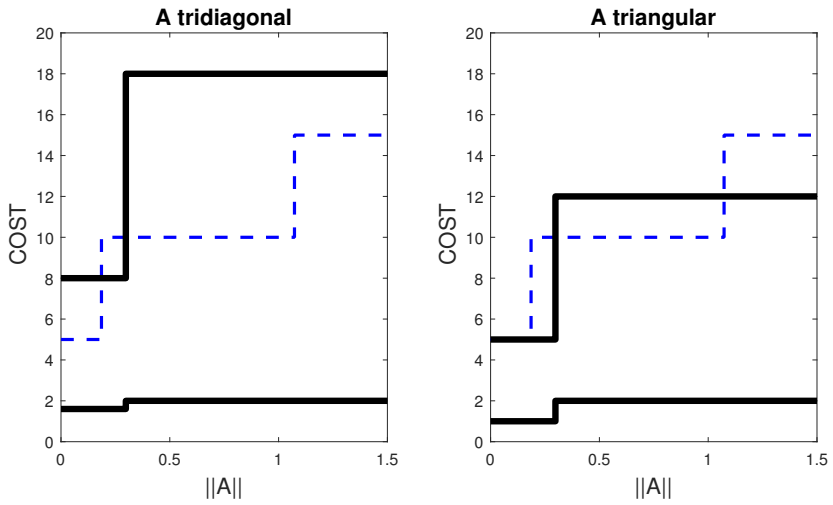

Figure 6: Computational cost to evaluate $e^A\mathbf{v}$ with either A a tridiagonal (left) or a triangular (right) matrix (measured as times the cost of one matrix-vector product) versus $\|A\|$ for the algorithm using Taylor approximations of degree 5, 10 and 15 as given in [2] (dashed line) and the new rational approximations R_5 and R_{10} of order 5 and 10, respectively. The upper solid line corresponds to the worst case in which the cost is measured as if the methods were computed sequentially while the lower one corresponds to the ideal case in which they were computed in parallel with no cost in the communication between processors.

(or the same exponential is applied on several steps for a given integrator) then the LU factorization needs to be done only once making the new algorithms cheaper and competitive even when used as sequential algorithms.

- If the matrix A is pentadiagonal and we take into account that the vector-matrix product, $A\mathbf{v}$, can be done with $9d$ flops and that the pentadiagonal system $(I - c_i A)\mathbf{x} = \mathbf{v}$ can be solved with only $15d$ flops using an LU factorization then, the same conclusions remain approximately valid for this problem because the relative cost is very similar to the case of tridiagonal matrices.

- If the matrix A is banded and the linear system $(I - c_i A)\mathbf{x} = \mathbf{v}$ can be solved accurately with few iterations using, for example, a conjugate gradient method with an incomplete LU factorization as a preconditioner, then the previous results provide a good picture of the performance of the new methods for banded matrices.

- The computational cost from the linear combination of vectors should also to be considered in the simple case of the exponential of tridiagonal matrices, making all methods slightly more costly. We have not included this extra cost to provide some results which can also be applied to pentadiagonal or banded matrices where this extra cost is marginal.

We have repeated the same analysis for the evaluation of the exponential of triangular matrices acting on vectors and taking into account that to solve a triangular systems requires the same computational cost as a triangular matrix multiplying a vector. The results are collected in Table 2 and illustrated in the right panel of Figure 6. We observe that the new algorithms are superior to the Taylor method even in the worst case in which they are computed sequentially! These results motivated us for a deeper analysis for building new methods for approximating functions of triangular matrices for different tolerances, which will be carried in the future.

6 Conclusions

The performance of an important number of exponential integrators for solving differential equations strongly depend on the existence of efficient algorithms to compute functions of matrices or their action on vectors. In this work we have presented a new procedure to compute functions of matrices as well as their action on vectors designed for parallel programming that can be significantly more efficient than existing sequential methods. Given a dense matrix, A, the computation of $(I - c_i A)^{-1}$, for sufficiently small constant c_i, can be evaluated at the cost of one matrix-matrix product and can formally be written as a series expansion that contains all powers of A. Then, a proper linear combination of this matrix evaluated in parallel for different values of c_i can allow to approximate any function inside the radius of convergence. If the computation can be carried in parallel with a reduced cost in the communication between processors then the new methods can be up to several times faster than conventional serial algorithms. For large dimensional problems in which one is only interested in the matrix function acting on a vector, the performance of the new methods depend on the existence of a fast algorithm to solve the linear system of equations, $(I - c_i A)\mathbf{v}_i = \mathbf{v}$. We have illustrated with some examples that to construct new methods as well as to carry an error analysis is quite simple. The preliminary results are very much promising and it deserves to be further investigated to get optimal methods for different classes of problems and number of processors available in order to get accurate and stable solutions with small round off errors.

We conclude that one can easily build tailored methods for different classes of problems. For example, if A has negative real eigenvalues (or close to it) of large norm, one can choose $c_i > 0$ and the coefficients b_i can also be chosen such that the rational funciton interpolates the exponential e^x at some appropriate values of x in the negative real line such that the error of the method is bounded by a given tolerance in the whole negative real line. This is similar to the rational Chebyshev methods but with positive real roots and can be easily done for other functions as the φ-functions, and then they would be appropriate for parbolic problems, and can be valuable tools for exponential integrators.

Acknowledgements

This work has been funded by Ministerio de Ciencia e Innovacion (Spain) through project PID2022-136585NB-C21, MCIN/AEI/10.13039/501100011033/FEDER, UE, and also by Generalitat Valenciana (Spain) through project CIAICO/2021/180.

References

[1] A. H. Al-Mohy and N. J. Higham, A new Scaling and Squaring Algorithm for the Matrix Exponential, SIAM J. Matrix Anal. Appl., 31, (2009), pp. 970–989.

[2] 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. 33 (2011), pp. 488-511.

[3] A.H. Al-Mohy, N.J. Higham and S.D. Relton, New algorithms for computing the matrix sine and cosine separately or simultaneously, SIAM J. Sci. Comput. 37 (2015) A456 - A487.

[4] A. H. Al-Mohy and N. J. Higham, and X. Liu, Arbitrary Precision Algorithms for Computing the Matrix Cosine and its Fréchet Derivative, SIAM J. Matrix Anal. Appl., 43, (2022), pp. 233–256.

[5] M. Arioli, B. Codenotti, and C. Fassino, The Padé method for computing the matrix exponential. Lin. Alg. Applic. 240 (1996), pp. 111–130.

[6] T. Auckenthaler, M. Bader, T. Huckle, A. Spörl, and K. Waldherr, Matrix exponentials and parallel prefix computation in a quantum control problem, Parallel Computing 36 (2010), pp. 359–369.

[7] P. Bader, S. Blanes, and F. Casas, An improved algorithm to compute the exponential of a matrix. arXiv:1710.10989 [math.NA], 2017.

[8] P. Bader, S. Blanes, and F. Casas, Computing the matrix exponential with an optimized Taylor polynomial approximation, Mathematics, 7 (2019), p. 1174.

[9] P. Bader, S. Blanes, F. Casas, and M. Seydaoğlu, An efficient algorithm to compute the exponential of skew-Hermitian matrices for the time integration of the Schrödinger equation, Math. Comput. Sim. 194 (2022), pp. 383–400.

[10] J. Biamonte, P. Wittek, N. Pancotti, P. Rebentrost, N. Wiebe, S. Lloyd, Quantum Machine Learning, Nature, 549, (2017), pp. 195–202.

[11] S. Blanes, Novel parallel in time integrators for ODEs, Appl. Math. Lett., 122 (2021) 107542.

[12] S. Blanes, Parallel Computation of functions of matrices and their action on vectors, arXiv:2210.03714

[13] S. Blanes, F. Casas, J. A. Oteo, J. Ros, The Magnus expansion and some of its applications, Phys. Rep., 470, (2009), pp. 151–238.

[14] S. Blanes, F. Casas, and M. Thalhammer, High-order commutator-free quasi-magnus exponential integrators for non-autonomous linear evolution equations, Comput. Phys. Comm., 220 (2017), pp. 243–262.

[15] S. Blanes, N. Kopylov and M. Seydaoğlu, Efficient scaling and squaring method for the matrix exponential, SIAM J. Matrix Anal. To appear (arXiv:submit/5546993).

[16] S. Blanes and P.C. Moan, Fourth- and sixth-order commutator-free Magnus integrators for linear and non-linear dynamical systems, Appl. Numer. Math. 56 (2006), pp. 1519-1537.

[17] D. Calvetti, E. Gallopoulos, and L. Reichel, Incomplete partial fractions for parallel evaluation of rational matrix functions, J. Comput. Appl. Math., 59 (1995) pp. 349-380.

[18] J.-P. Chehab and M. Petcu, Parallel matrix function evaluation via initial value ODE modeling Comput. Math. Appl., 72 (2016), pp. 76–91.

[19] W.J. Cody, G. Meinardus, and R. S. Varga, Chebyshev rational approximation to e^{-x} in $[0, +\infty)$ and applications to heat-conduction problems, J. Approximation Theory, 2 (1969), pp. 50–65.

[20] E. Gallopoulos, B. Philippe, and A.H. Sameh, Parallelism in matrix computations, Springer, 2016.

[21] E. Gallopoulos and Y. Saad, On the parallel solution of parabolic equations, in Proc. 1989 ACM Int'l. Conference on Supercomputing, Herakleion, Greece, June 1989, pp. 17–28.

[22] E. Gallopoulos and Y. Saad, Efficient Solution of Parabolic Equations by Krylov Approximation Methods, SIAM J. Sci. Statist. Comput., 13 (1992), pp. 1236–1264.

[23] M.J. Gander and S. Güttel, PARAEXP: A Parallel integrator for linear initial-value problems, SIAM J. Sci. Comput., 35 (2013), pp. C123–C142.

[24] G. H. Golub and C. F. Van Loan, Matrix Computations, John Hopkins Univ. Press, Baltimore, Maryland, 1993.

[25] A.W. Harrow, A. Hassidim, and S. Lloyd, Quantum Algorithm for Linear Systems of Equations, Phys. Rev. Lett., 103, (2009), 150502.

[26] N. J. Higham, The Scaling and Squaring Method for the Matrix Exponential, SIAM J. Matrix Anal. Appl., 26, (2005), pp. 1179–1193.

[27] N. J. Higham, The Scaling and Squaring Method for the Matrix Exponential Revisited, SIAM Review, 51, (2009), pp. 747–764.

[28] N. J. Higham, Functions of Matrices: Theory and Computation, Society for Industrial and Applied Mathematics, Philadelphia, PA, USA (2008).

[29] N. J. Higham and A. H. Al-Mohy, Computing Matrix Functions, Acta Numerica, 51, (2010), pp. 159–208.

[30] M. Hochbruck, Ch. Lubich, On Krylov subspace approximations to the matrix exponential operator, SIAM J. Numer. Anal. 34 (1997), pp. 1911-1925

[31] M. Hochbruck and A. Ostermann, Exponential integrators, Acta Numerica, 19 (2010), pp. 209–286.

[32] A. Iserles, H.Z. Munthe-Kaas, S.P. Nørsett, and A. Zanna, Lie group methods. Acta Numerica *9* (2000), pp. 215–365.

[33] E. Jarlebring, M. Fasi, and E. Ringh, Computational graphs for matrix functions, arXiv preprint arXiv:2107.12198, (2021).

[34] V.T. Luan, Efficient exponential Runge–Kutta methods of high order: construction and implementation, BIT Numer. Math. 61 (2021), pp. 535-560.

[35] C. B. Moler and C. F. Van Loan, Nineteen dubious ways to compute the exponential of a matrix, twenty-five years later, SIAM Review 45 (2003), pp. 3–49.

[36] I. Najfeld and T.F. Havel, Derivatives of the matrix exponential and their computation. Adv. Appl. Math. *16* (1995), pp. 321–375.

[37] J. Niesen and W. M. Wright, Algorithm 919: A Krylov subspace algorithm for evaluating the φ-functions appearing in exponential integrators, ACM Trans. Math. Software, 38 (2012), Art. 22, 19 pp.

[38] W.H. Press, B.P. Flannery, S.A. Teukolsky, and W.T. Vetterling, Numerical Recipes in Fortran 90: The Art of Parallel Scientific Computing, Volume 2 of Fortran Numerical Recipes, Second Ed., Cambridge Univ. Press, New York 1997.

[39] J. Sastre, Efficient evaluation of matrix polynomials, Lin. Alg. Applic., 539 (2018), pp. 229–250.

[40] J. Sastre, J. Ibáñez, and E. Defez, Boosting the computation of the matrix exponential, Appl. Math. Comput., 340 (2019), pp. 206–220.

[41] J. Sastre, J. Ibáñez, P. Ruiz, and E. Defez, Accurate and efficient matrix exponential computation, Int. J. Comput. Math., 91, (2014), pp. 97–112.

[42] M. Seydaoğlu, P. Bader, S. Blanes, and F. Casas, Computing the matrix sine and cosine simultaneously with a reduced number of products, Appl. Numer. Math., 163 (2021), pp. 96–107.

[43] R. B. Sidje, Expokit: a software package for computing matrix exponentials, ACM Trans. Math. Software 24 (1998), pp. 130–156.

[44] L.N. Trefethen and D. Bau III, Numerical linear algebra, SIAM, Philadelphia, 1997.

[45] F. Wu, K. Zhang, L. Zhu, and J. Hu, High-performance computation of the exponential of a large sparse matrix, SIAM J. Matrix Anal. Appl., 42, (2021), pp. 1636–1655.