Efficient scaling and squaring method for the matrix exponential

S. Blanes[♭], N. Kopylov[♭, 1] and M. Seydaoğlu[♮]

(♭) I.U. de Matemática Multidisciplinar, Universitat Politècnica de València
Camí de Vera s/n, València, Spain.

(♮) Department of Mathematics, Faculty of Art and Science, Muş Alparslan University
Muş, Turkey.

1 Introduction

We present an algorithm for computing the matrix exponential $\exp(A)$ within a specified tolerance *tol*, given a matrix A of size $m \times m$. First, the algorithm computes a bound, θ, to the norm of the matrix, i.e., $\|A\| < \theta$, and then, according to θ, it chooses the scheme among a list of selected methods which provides an approximation to $\exp(A)$ within such tolerance. If no matching scheme is found, the algorithm employs scaling and squaring to find the cheapest method matching $\frac{\|A\|}{2^s} < \theta$. Different schemes for approximating the matrix exponential should be selected based on the specific problem at hand.

As an example of numerical methods where such adaptivity is beneficial, consider exponential integrators for solving differential equations. They have been shown to be very useful for a significant number of problems [10]. For instance, most Lie-group methods (see also [12] for a review) like Magnus integrators (see [3] and references therein), Crouch–Grossman methods [5], Runge–Kutta–Munthe-Kaas methods [13], etc. require the calculation of one or several matrix exponentials per step. In most cases, it suffices to approximate each matrix exponential only up to a given tolerance lower than the round-off accuracy. When the Lie-group structure must be preserved, it is also possible to provide cheaper approximations to the exponential while still preserving the structure.

2 Methodology

The goal of this work is to construct an algorithm that, when provided with a matrix $A \in \mathbb{C}^{N\times N}$ and a tolerance *tol*, computes a function $w_\alpha(A)$, where α refers to a label to identify the method, capable of approximating the matrix exponential e^A so that

$$relerr := \frac{\|w_\alpha(A) - \mathrm{e}^A\|_1}{\|\mathrm{e}^A\|_1} < tol \cdot \|A\|_1. \tag{1}$$

Although there are various for the function $w_\alpha(A)$, we analyse an extensive set of methods from two families:

[1] nikop1@upv.es

- Taylor polynomials: $w_\alpha(A) = t_m(A)$, where $t_m(x) = \sum_{n=0}^{m} \frac{x^n}{n!}$, $x \in \mathbb{C}$, and $|t_m(x) - \mathrm{e}^x| = \mathcal{O}(x^{m+1})$.
- Rational Padé approximants: $w_\alpha(A) = r_{k,m}(A)$, where $r_{k,m}(x) = p_{k,m}(x)/q_{k,m}(x)$ with

$$p_{k,m}(x) = \sum_{j=0}^{k} \frac{(k+m-j)!k!}{(k+m)!(k-j)!} \frac{x^j}{j!}, \quad q_{k,m}(x) = \sum_{j=0}^{m} \frac{(k+m-j)!m!}{(k+m)!(m-j)!} \frac{(-x)^j}{j!}, \tag{2}$$

  which is an approximation of order $s = k + m$ with the leading error term given [11, 7] by

$$\mathrm{e}^x - r_{k,m}(x) = (-1)^m \frac{k!m!}{(k+m)!(k+m+1)!} x^{k+m+1} + \mathcal{O}(x^{k+m+2}). \tag{3}$$

  Notice that the Taylor polynomial is a special case of Padé approximant, $t_s(x) = r_{s,0}(x)$, and has a larger leading error than $r_{k,m}(x)$ with $s = k + m$, $k, m > 0$. However, the performance of a method strongly depends on the computational cost[2] and it should also be considered.

Taking into account their computational cost, we select a list of methods which our algorithm uses according to the provided matrix A and the tolerance tol.

Among all possible choices, one should use the method that provides the desired accuracy at the lowest computational cost, and this requires to carry the following analysis:

Backward error. Given a particular method w_α, we look for an associated scalar function for that method, say $\theta_\alpha(y)$, such that, given a matrix A and a positive integer number, s, such that if $\|A\|_1 \leq 2^s \theta_\alpha(tol)$, the method provides an approximation with a relative error below the tolerance when used with s squarings. Forward and backward error analysis are frequently used in the literature, and we will consider the backward error analysis in this work, similarly to the analysis carried out in [6, 9] and implemented in the function (`expm`) in MATLAB.

Computational cost. For the analysis of the cost we assume that one dense matrix–matrix multiplication cost is $C \coloneqq 1$, which we will take as the reference cost. We will say that the cost of a method is $kC \equiv k$ if its computational cost is approximately k times the cost of a matrix–matrix product. Then, given two matrices A and B, the cost to compute A^{-1} or $A^{-1}B$ will be taken as $\frac{4}{3}C$.

Computational cost is less straightforward. Over the past few decades, the most effective approach for addressing a specific issue has evolved, with advancements in reducing algorithmic costs (a trend that is expected to persist with the emergence of novel algorithms and computer designs). For example, Taylor methods were originally discarded since, when computed with the standard Horner's algorithm, they require $m - 1$ products to compute t_m. However, t_m with $m = 2, 4, 6, 9, 16, 20$ can be computed with $k = 1, 2, 3, 4, 5, 6$ products, respectively, a significant saving which made the Taylor methods competitive [14, 16]. In addition, in [1, 2, 17] it is shown that further reduction can be carried out such that the Taylor polynomial with $m = 2, 4, 8, 12, 18$ can be computed with $k = 1, 2, 3, 4, 5$ products, respectively, making them the methods of choice for many problems.

On the other hand, diagonal Padé methods have the property that $q_{m,m}(x) = p_{m,m}(-x)$, and this symmetry allows finding a procedure to compute both polynomials $p_{m,m}(x)$ and $q_{m,m}(x)$ simultaneously at a reduced cost for $m = 1, 2, 3, 5, 7, 9, 13$ with $k = 0, 1, 2, 3, 4, 5, 6$ products, respectively, in addition to one inverse to compute $r_{m,m}$. In [6] it is claimed that, since $r_{m,m}$ can be computed with the same cost as $r_{k,m}$ or $r_{m,k}$ with $k < m$ and provide higher accuracy

[2] In some cases round off accuracy must be taken into account.

(see eq. (3)), only diagonal Padé methods are considered. However, in [15] it is shown that this is not necessarily the case, showing that a number of approximants $r_{k,m}$, with $k > m$ and with an appropriate fractional decomposition, can be computed at the same cost as $r_{m,m}$ while providing higher accuracy.

Avoiding complex coefficients. In this work, we decompose some rational Padé methods into simpler fractions, avoiding schemes with complex coefficients. Our goal is to propose methods optimized for real matrices yet efficient for complex matrices, favoring real coefficients.

Lie group methods. If A belongs to a Lie algebra, then $\exp(A)$ belongs to the associated Lie group. Our algorithm facilitates computing the exponential using only diagonal Padé approximants, preserving this property to round-off accuracy if necessary.

2.1 Selected methods sorted by computational cost

We collect now the list of the cost-efficient methods w_α we have found, sorted by their computational cost k_α. We give the structure of the schemes with appropriate values for the coefficients $\alpha_i, \beta_i, \ldots,\ i = 1, 2, \ldots$, which take different values in each case.

$k = 1$ **product** $t_2(x) = 1 + x + \frac{1}{2}x^2$.

$k = 1\frac{1}{3}$ **product** $r_{2,1}(x) = p_0(x) + \frac{p_1(x)}{p_2(x)}$, $p_i(x) = \alpha_i + \beta_i x$ with $p_0(0) = 0, \alpha_2 = 1$.

$k = 2$ **products** $t_4(x) = 1 + x + x^2(\frac{1}{2!} + \frac{1}{3!}x + \frac{1}{4!}x^2)$.

$k = 2\frac{1}{3}$ **products** $r_{4,2}(x) = p_0(x) + \frac{p_1(x)}{p_2(x)}$, $p_i(x) = \alpha_i + \beta_i x + \gamma_i x^2$, $\alpha_0 = 0, \alpha_2 = 1$.

$k = 3$ **products** $t_8(x)$.

$k = 3\frac{1}{3}$ **products** $r_{6,3}(x) = p_0(x) + \frac{p_1(x)}{p_2(x)}, \quad p_i(x) = \alpha_i + \beta_i x + \gamma_i x^2 + \delta_i x^3$ with $\alpha_0 = 0, \alpha_2 = 1$.

$k = 3\frac{2}{3}$ **products** $r_{6,4}(x) = p_0(x) + \frac{p_1(x)}{p_2(x)} + \frac{p_3(x)}{p_4(x)}, \quad p_i(x) = \alpha_i + \beta_i x + \gamma_i x^2$ with $\alpha_0 = 0$ and $\frac{p_1(0)}{p_2(0)} = \frac{p_3(0)}{p_4(0)} = \frac{1}{2},\ \alpha_2 = \alpha_4 = 1$.

$k = 4$ **products** $t_{12}(x)$ and $t_{15}^{[16]}(x)$.

$k = 4\frac{1}{3}$ **products** $r_{8,4}(x) = p_0(x) + \frac{p_1(x)}{p_2(x)}, \quad p_i(x) = \alpha_i + \beta_i x + \gamma_i x^2 + \delta_i x^3 + \sigma_i x^4$ with $\alpha_0 = 0, \alpha_2 = 1$.

$k = 4\frac{2}{3}$ **products** $r_{8,5}(x) = p_0(x) + \frac{p_1(x)}{p_2(x)} + \frac{q_1(x)}{q_2(x)}, \quad p_i(x) = \alpha_i + \beta_i x + \gamma_i x^2 + \delta_i x^3$, $q_i(x) = \tilde{\alpha}_i + \tilde{\beta}_i x + \tilde{\gamma}_i x^2 + \tilde{\delta}_i x^3$ with $\alpha_0 = 0$ and $\frac{p_1(0)}{p_2(0)} = \frac{q_1(0)}{q_2(0)} = \frac{1}{2}$.

$k = 5$ **products** $t_{18}(x)$ and $t_{21}^{[24]}(x)$.

$k = 5\frac{1}{3}$ **products** $r_{10,5}(x) = p_0(x) + \frac{p_1(x)}{p_2(x)}, \quad p_i(x) = \alpha_i + \beta_i x + \gamma_i x^2 + \delta_i x^3 + \sigma_i x^4 + \mu_i x^5,\ \alpha_0 = 0,\ \alpha_2 = 1$. This method is not used because for all the considered tolerance values its θs are less than the corresponding ones of $(r_{8,4}(x))^2$.

$k = 5\frac{2}{3}$ **products** $r_{12,8}(x) = p_0(x) + \frac{p_1(x)}{p_2(x)} + \frac{p_3(x)}{p_4(x)}, \quad p_i(x) = \alpha_i + \beta_i x + \gamma_i x^2 + \delta_i x^3 + \sigma_i x^4$ with $\alpha_0 = 0$ and $\frac{p_1(0)}{p_2(0)} = \frac{p_3(0)}{p_4(0)} = \frac{1}{2},\ \alpha_2 = \alpha_4 = 1$. Unfortunately, in general it suffers from round-off errors, although smaller than those of $r_{16,12}(x)$. However for lower tolerance values it can compete with $r_{8,4}(x)$ and $r_{8,5}(x)$.

$k = 7$ **products** $r_{16,12}(x) = p_0(x) + \frac{p_1(x)}{p_2(x)} + \frac{p_3(x)}{p_4(x)} + \frac{p_5(x)}{p_6(x)}, \quad p_i(x) = \alpha_i + \beta_i x + \gamma_i x^2 + \delta_i x^3 + \sigma_i x^4$. This scheme would be the method of choice when very high accuracy is desired and the scaling has to be used. Unfortunately, for all choices in the free parameters of the method that we have tried, it suffers severe round-off errors and this scheme is under investigation at this moment and not used in the algorithm.

$k = 7\frac{1}{3}$ **products** $r_{13,13}(x)$.

2.2 Adaptive scheme selection

Taking into account the considerations above, our experimental implementation uses the following algorithm to select a method when provided a matrix, a tolerance value, and Table 1 (ordered by tolerance and method cost in ascending order):

1. Calculate $m = \lfloor \log_{10}(tol) \rfloor$, where $\lfloor \cdot \rfloor$ denotes the rounding-down operation.
2. Select the column with θ_α corresponding to the condition $10^m < tol$.
3. For each θ_α calculate the corresponding scaling $s_\alpha = \max\left\{0, \left\lceil \log_2 \frac{\|A\|_1}{\theta_\alpha(10^m)} \right\rceil\right\}$, where $\lceil \cdot \rceil$ denotes the rounding-up operation.
4. For each method w_α with the cost k_α (provided in the list above) calculate the total cost $k_\alpha + 1.1 \cdot s_\alpha$. To break ties, methods that need scaling are penalized by multiplying s_α by the factor of 1.1.
5. Select the method with the lowest total cost.

2.3 Numerical performance

In the numerical experiments we show the behavior of the proposed algorithm and the individual schemes that comprise it.

Consider a random diagonally dominant matrix $A = D + R$, $A \in \mathbb{R}^{101\times101}$, such that $D = diag(-50, -49, \ldots, 50)$ and the elements of R are sampled uniformly from $[-1; 1]$, and the matrix is normalized (i.e. $A := A/\|A\|_1$).

Then, for every norm–tolerance pair in $h = 10^m$, $m = -3, -2, \ldots, 2$ and $tol = 10^{-k}$, $k = 0, 1, \ldots, 16$ the algorithm returns $w_\alpha(hA)$ and the corresponding computational cost in terms of matrix–matrix products, C. For each norm h, we calculate the normalized relative error

$$\frac{\|w_\alpha(hA) - \mathrm{e}^{hA}\|_1}{\|A\|_1 \cdot \|\mathrm{e}^{hA}\|_1}, \tag{4}$$

where the reference e^{hA} is computed numerically in arbitrary arithmetic, obtaining error–cost pairs. Then in Figure 1, we plot these pairs alongside with the theoretical estimates from Table 1.

In Figure 1 one can see that the average error of superdiagonal methods from Table 1 is below the tolerance predicted by the backward error analysis. The only exception is $r_{12,8}(x)$ for low tolerance, where $r_{13,13}(x)$ should be preferred.

The results that one would obtain using the schemes from the current numerical software are included for comparison as vertical dotted lines, which correspond to unchanging approximation to the round-off error. The specific scheme used by the algorithm is annotated above the error line. More specifically, if we take $m = -1$, i.e. $\|A\|_1 \leq 0.1$, one of six methods seen in the Figure 1 will be used, while $r_{5,5}$ would be used at a higher cost in MATLAB or Julia.

Similar results can be observed for in the case of diagonal Padé methods that preserve the Lie group properties.

Table 1: Comparison of backward errors of superdiagonal methods. The asterisk ($*$) stands for methods with no cost estimate. Grayed-out rows and columns are provided for completeness.

Cost	Method	Tolerance						
		2^{-11}	10^{-4}	2^{-24}	10^{-8}	10^{-12}	2^{-53}	10^{-16}
1	t_2	5.31e−2	2.43e−2	5.98e−4	2.45e−4	2.45e−6	2.58e−8	2.45e−8
$1\frac{1}{3}$	$r_{2,1}$	3.18e−1	1.90e−1	1.62e−2	8.96e−3	4.16e−4	2.00e−5	1.93e−5
2	t_4	4.48e−1	3.10e−1	5.12e−2	3.29e−2	3.31e−3	3.40e−4	3.31e−4
$2\frac{1}{3}$	$r_{4,2}$	1.66	1.30	3.98e−1	2.97e−1	6.48e−2	1.42e−2	1.40e−2
3	t_8	1.59	1.35	5.80e−1	4.70e−1	1.54e−1	4.99e−2	4.93e−2
$3\frac{1}{3}$	$r_{6,3}$	3.28	2.81	1.31	1.09	4.01e−1	1.47e−1	1.45e−1
$3\frac{2}{3}$	$r_{6,4}$	4.10	3.57	1.79	1.51	6.12e−1	2.48e−1	2.46e−1
4	t_{12}	2.79	2.50	1.46	1.28	6.24e−1	3.00e−1	2.97e−1
$*$	t_{15}	3.68	3.38	2.22	2.00	1.14	6.41e−1	6.37e−1
4	$t_{15}^{[16]}$	3.91	3.59	2.35	2.11	1.20	4.92e−1	4.63e−1
$4\frac{1}{3}$	$r_{8,4}$	4.95	4.43	2.55	2.22	1.07	5.07e−1	5.03e−1
$4\frac{2}{3}$	$r_{8,5}$	5.83	5.25	3.14	2.76	1.40	7.05e−1	6.99e−1
5	t_{18}	4.57	4.26	3.01	2.76	1.75	1.09	1.08
$*$	t_{21}	5.45	5.13	3.82	3.56	2.42	1.62	1.62
5	$t_{21}^{[24]}$	5.62	5.29	3.95	3.67	2.50	4.54e−1	4.21e−1
$5\frac{1}{3}$	$r_{10,5}$	6.64	6.08	3.95	3.54	2.00	1.11	1.10
$5\frac{2}{3}$	$r_{12,8}$	1.02e1	9.54	6.91	6.37	4.16	2.69	2.68
7	$r_{16,12}$	1.55e1	1.48e1	1.18e1	1.12e1	8.27	6.09	6.07
$7\frac{1}{3}$	$r_{13,13}$	1.53e1	1.45e1	1.12e1	1.06e1	7.55	5.37	5.35

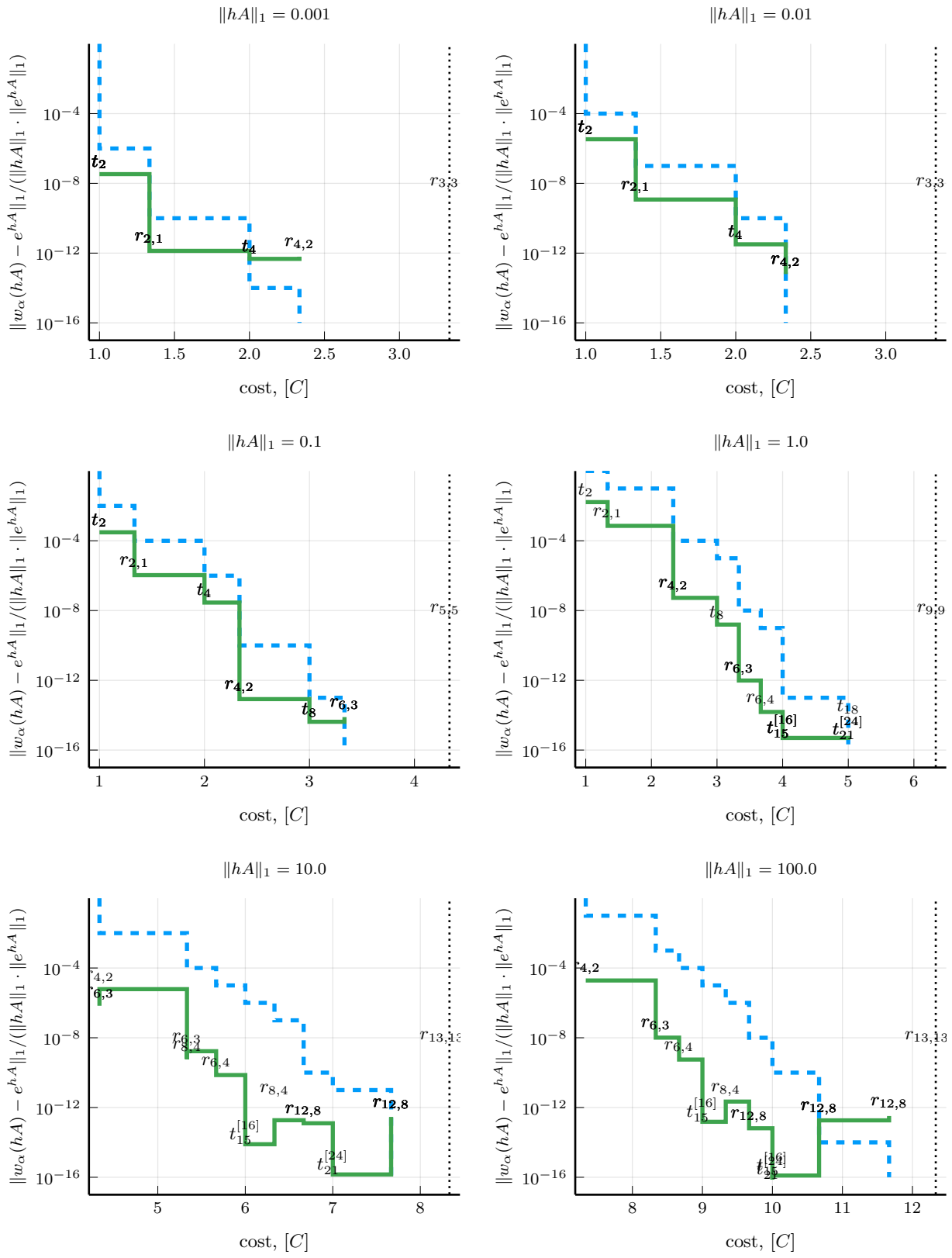

Figure 1: Comparison of tolerance (dashed) and practical average cost plotted vs. total computational cost of superdiagonal methods. The thin vertical dotted line represents the cost in the current software that implements [7]. Note that the distance between the dotted vertical line and the solid one represents cost savings.

3 Conclusions

We introduce a novel algorithm for approximating the matrix exponential function with adjustable tolerance, offering reduced computational costs compared to conventional methods across a broad spectrum of matrices. This algorithm readily accommodates customization, allowing for the exclusive use of Taylor methods (i.e., matrix-matrix products without inverses) in scenarios where products are significantly more economical than inverses, or the exclusive use of diagonal Padé approximants to maintain Lie group structure.

Furthermore, we have decomposed several Padé approximants into lower-degree fractions, facilitating independent computation and rendering them amenable to parallelization. We will conduct an in-depth investigation into parallel computation strategies for matrix exponential evaluation following the approach outlined in [4].

Acknowledgments

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

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

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

[4] S. Blanes, Parallel computation of functions of matrices and their action on vectors arXiv:2210.03714 [math.NA], (2022), preprint.

[5] P.E. Crouch and R. Grossman, Numerical integration of ordinary differential equations on manifolds, J. Nonlinear Sci. 3 (1993), 1–33.

[6] N.J. Higham, The scaling and squaring method for the matrix exponential revisited, SIAM J. Matrix Anal. Appl., **26**, (2005), pp. 1179–1193.

[7] N.J. Higham, The scaling and squaring method for the matrix exponential revisited, SIAM Review **51** (2009), pp. 747-764.

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

[9] N.J. Higham and A.H. Al-Mohy, Computing matrix functions, Acta Numerica, **51**, (2010), pp. 159–208.

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

[11] A. Iserles, Composite exponential approximations. Mathematics Of Computation. **38**, 99-112 (1982), http://dx.doi.org/10.1090/S0025-5718-1982-0637289-7

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

[13] H. Munthe-Kaas, Runge–Kutta methods on Lie groups, BIT 38 (1998), 92–111.

[14] M.S. Paterson and L.J. Stockmeyer, On the number of nonscalar multiplications necessary to evaluate polynomials, SIAM J. Comput., **2** (1973), pp. 60–66.

[15] J. Sastre, Efficient mixed rational and polynomial approximations of matrix functions, Appl. Math. Comput., **218** (2012), pp. 11938–11946.

[16] J. Sastre, J. Ibáñez, P. Ruiz, and E. Defez, New scaling-squaring Taylor algorithms for computing the matrix exponential, SIAM J. Sci. Comp., **37** (2015), pp. A439–A455.

[17] J. Sastre, Efficient evaluation of matrix polynomials, Linear Algebra Appl., **539**, (2018), pp. 229–250.