Applied Mathematics and Computation 340 (2019) 206–220

Contents lists available at ScienceDirect

Applied Mathematics and Computation

journal homepage: www.elsevier.com/locate/amc

Boosting the computation of the matrix exponential[☆]

J. Sastre[a,*], J. Ibáñez[b], E. Defez[c]

[a] Instituto de Telecomunicaciones y Aplicaciones Multimedia, Universitat Politècnica de València, Spain
[b] Instituto de Instrumentación para Imagen Molecular, Universitat Politècnica de València, Spain
[c] Instituto de Matemática Multidisciplinar, Universitat Politècnica de València, Spain

ARTICLE INFO

Keywords:
Matrix exponential
Scaling and squaring
Taylor series
Efficient matrix polynomial evaluation

ABSTRACT

This paper presents new Taylor algorithms for the computation of the matrix exponential based on recent new matrix polynomial evaluation methods. Those methods are more efficient than the well known Paterson–Stockmeyer method. The cost of the proposed algorithms is reduced with respect to previous algorithms based on Taylor approximations. Tests have been performed to compare the MATLAB implementations of the new algorithms to a state-of-the-art Padé algorithm for the computation of the matrix exponential, providing higher accuracy and cost performances.

1. Introduction

The computation of matrix functions has received remarkable attention in the last decades because of its numerous applications in science and engineering [1]. From all the matrix functions the matrix exponential has been the most studied function, and a large number of methods for its computation have been proposed [1,2].

In 2009 the authors submitted their first work with Taylor based algorithms for computing the matrix exponential [3]. Until then, Padé approximants for the matrix exponential were preferred to Taylor approximations because Padé algorithms were more efficient than the existing Taylor algorithms, for similar accuracy [1]. Applying and improving the algorithms for Padé approximants from [15] to Taylor approximations, the Taylor algorithms from [3] showed to be generally more accurate than the Padé algorithm from [15] in tests, with a slightly higher cost.

In [4] the authors presented a scaling and squaring Taylor algorithm for computing the matrix exponential based on an improved mixed backward and forward error analysis. It was more accurate than the state-of-the-art Padé algorithm from [14] in the majority of tests, with a slightly higher cost. Subsequently, Sastre et al. [5] provided a formula for the forward relative error of the matrix exponential Taylor approximation, and proposed to increase the allowed error bounds depending on the matrix size and the Taylor approximation order. This algorithm reduced the computational cost in exchange for a small impact in accuracy. The method proposed in [6] simplified the algorithm of [4], preserving accuracy, and showing to be more accurate than the Padé algorithm from [14] in the majority of tests, being also more efficient in some cases. Finally, Defez et al. [8] used Taylor approximations combined with spline techniques to increase accuracy, also increasing the cost. In this work, we present new Taylor algorithms based on the efficient matrix polynomial evaluation methods from [9],

☆ This work has been supported by Spanish Ministerio de Economía y Competitividad and European Regional Development Fund (ERDF) grant TIN2014-59294-P.

* Corresponding author at: Instituto de Telecomunicaciones y Aplicaciones Multimedia, Spain.
E-mail address: jsastrem@upv.es (J. Sastre).

https://doi.org/10.1016/j.amc.2018.08.017
0096-3003/© 2018 Elsevier Inc. All rights reserved.

increasing significantly the efficiency of the previous Taylor methods. We will show that the new algorithms are generally both more accurate and efficient than the state-of-the-art Padé algorithm from [14].

Throughout this paper $\mathbb{C}^{n\times n}$ denotes the set of complex matrices of size $n\times n$, I denotes the identity matrix for this set, $\rho(A)$ is the spectral radius of matrix A, and $\mathbb{N}$ denotes the set of positive integers. The matrix norm $\|\cdot\|$ denotes any subordinate matrix norm; in particular $\|\cdot\|_1$ is the 1-norm. The symbols $\lceil\cdot\rceil$ and $\lfloor\cdot\rfloor$ denote the smallest following and the largest previous integer, respectively. The cost of the Taylor algorithms will be given in terms of the number of evaluations of matrix products, denoting the cost of one matrix product by M. Note that the multiplication by the matrix inverse in Padé approximations is calculated as the solution of a multiple right-hand side linear system. The cost of the solution of multiple right-hand side linear systems $AX=B$, where matrices A and B are $n\times n$ will be denoted by D. Taking into account that, see [10, Appendix C]

$$D\approx 4/3M, \tag{1}$$

the cost of evaluating rational approximations will be also given in terms of M. All the given algorithms are intended for IEEE double precision arithmetic. Their extension to different precision arithmetics is straightforward.

This paper is organized as follows: Section 2 presents a general scaling and squaring Taylor algorithm. Section 3 introduces efficient evaluation formulas for the Taylor matrix polynomial approximation of the matrix exponential based on [9]. Section 4 presents the scaling and squaring error analysis. The new algorithm is given in Section 5. Section 6 shows numerical results and Section 7 gives some conclusions. Next theorem from [5] will be used in Section 4 to bound the norm of matrix power series.

Theorem 1. *Let $h_l(x)=\sum_{k\geq l} b_k x^k$ be a power series with radius of convergence R, and let $\tilde{h}_l(x)=\sum_{k\geq l}|b_k|x^k$. For any matrix $A\in\mathbb{C}^{n\times n}$ with $\rho(A)<R$, if a_k is an upper bound for $\|A^k\|$ ($\|A^k\|\leq a_k$), $p\in\mathbb{N}$, $1\leq p\leq l$, $p_0\in\mathbb{N}$ is the multiple of p with $l\leq p_0\leq l+p-1$, and*

$$\alpha_p=\max\{a_k^{\frac{1}{k}}: k=p,l,l+1,l+2,\dots,p_0-1,p_0+1,p_0+2,\dots,l+p-1\}, \tag{2}$$

then $\|h_l(A)\|\leq\tilde{h}_l(\alpha_p)$.

2. General Taylor algorithm

The Taylor approximation of order m of the matrix exponential of $A\in\mathbb{C}^{n\times n}$, denoted by $T_m(A)$, is defined by the expression

$$T_m(A)=\sum_{k=0}^{m}\frac{A^k}{k!}. \tag{3}$$

The scaling and squaring algorithms with Taylor approximation (3) are based on the approximation $e^A=\left(e^{2^{-s}A}\right)^{2^s}\approx(T_m(2^{-s}A))^{2^s}$ [2], where the nonnegative integers m and s are chosen to achieve full machine accuracy at a minimum cost.

A general scaling and squaring Taylor algorithm for computing the matrix exponential is presented in Algorithm 1, where m_M is the maximum allowed value of m.

Algorithm 1 General scaling and squaring Taylor algorithm for computing $B=e^A$, where $A\in\mathbb{C}^{n\times n}$ and m_M is the maximum approximation order allowed.

```
1: Preprocessing of matrix $A$.
2: Choose $m_k \leqslant m_M$, and an adequate scaling parameter $s\in\mathbb{N}\cup\{0\}$ for the Taylor approximation with scaling.
3: Compute the matrix polynomial $B=T_{m_k}(A/2^s)$
4: for $i=1:s$ do
5:     $B=B^2$
6: end for
7: Postprocessing of matrix $B$.
```

In this paper the evaluation of the Taylor matrix polynomial of Step 3 is improved. The preprocessing and postprocessing steps (1 and 7) are based on applying transformations to reduce the norm of matrix A, see [1], and will not be discussed in this paper. In Step 2, the optimal order of Taylor approximation $m_k\leq m_M$ and the scaling parameter s will be chosen improving the algorithm from [6].

In [6] the matrix polynomial $T_m(2^{-s}A)$ was evaluated using the Paterson–Stockmeyer method evaluation formula (7) of [6], see [11]. The optimal Taylor orders m for that method were in the set $m_k=\{1,2,4,6,9,12,16,20,25,30,\dots\}$, $k=0,1,\dots$, respectively, where the matrix powers $A^2, A^3,\dots,A^q$ were evaluated and stored to be used in all the computations. Table 1, see [6, Table 1], shows some optimal values of q, denoted by q_k, used in [6] for orders m_k, $k=0,\ 1,\ 2,\dots,M$, and $m_M=20$, 25 or 30. In this work $T_m(2^{-s}A)$ will be computed using new evaluation methods based on [9], more efficient than Paterson–Stockmeyer method.

Table 1
Values of q_k depending on the selection of m_M in [6].

k	0	1	2	3	4	5	6	7	8	9
$m_M \diagdown m_k$	1	2	4	6	9	12	16	20	25	30
20	1	2	2	3	3	4	4	4		
25	1	2	2	3	3	4	4	5	5	
30	1	2	2	3	3	4	4	5	5	5

Table 2
Coefficients for computing the matrix exponential Taylor approximation of order $m = 8$ using (8) and (9), see [9, Table 4].

c_1	$4.980119205559973 \times 10^{-3}$
c_2	$1.992047682223989 \times 10^{-2}$
c_3	$7.665265321119147 \times 10^{-2}$
c_4	$8.765009801785554 \times 10^{-1}$
c_5	$1.225521150112075 \times 10^{-1}$
c_6	$2.974307204847627 \times 10^{0}$

Finally, after the evaluation of $T_m(2^{-s}A)$, s repeated squarings are applied in Steps 4–6. The postprocessing is applied in Step 7 to obtain the matrix exponential approximation of the original matrix A. The computational cost of [6, Algorithm 1] in terms of matrix products is

$$\mathrm{Cost}(m_k, s) = k + s. \tag{4}$$

Note that if $s = 0$ then $\mathrm{Cost}(m_k, 0) = k$. Using matrix polynomial evaluation methods based on [9] the costs from (4) will be reduced.

3. Efficient evaluation of the matrix exponential Taylor matrix approximation

In this section formulas for evaluating Taylor based approximations of orders $m = 8, 15, 21, 24$ and 30 are given based on the results from [9]. MATLAB R2017a Symbolic Math Toolbox with 200 decimal digit arithmetic was used in all the calculations. Note that for $T_m(A)$ with orders $m = 1, 2, 4$, the same evaluation formulas as in [6] will be used. i.e.

$$T_1(A) = A + I, \tag{5}$$

$$T_2(A) = A^2/2 + A + I, \tag{6}$$

$$T_4(A) = ((A^2/4 + A)/3 + I)A^2/2 + A + I. \tag{7}$$

3.1. Evaluation of $T_8(A)$

Following [9, Example 3.1] we can evaluate $T_8(A)$ with the following evaluation formulas

$$y_{02}(A) = A^2(c_1A^2 + c_2A), \tag{8}$$

$$T_8(A) = (y_{02}(A) + c_3A^2 + c_4A)(y_{02}(A) + c_5A^2) + c_6y_{02}(A) + A^2/2 + A + I, \tag{9}$$

where the coefficients c_i, $i = 1, 2, \ldots, 6$, numbered correlatively, are given in IEEE double precision arithmetic in Table 2, see [9, Table 4], A^2 is computed once and stored to be reused in all the computations, and $T_8(A)$ can be evaluated with a cost $3M$. Note that with a cost $3M$, taking $s = 0$ in (4) and using Table 1 the maximum Taylor order available with Paterson–Stockmeyer method is $m = 6$.

3.2. Evaluation of a Taylor based approximation of order 15

Following [9, Example 5.1] we can evaluate a Taylor based approximation of order $m = 15$ of the matrix exponential with the following evaluation formulas

$$y_{02}(A) = A^2(c_1A^2 + c_2A), \tag{10}$$

Table 3
Coefficients of y_{02}, y_{12}, y_{22} from (10)–(12) for computing a Taylor based approximation of the matrix exponential of order $m = 15$.

c_1	$4.018761610201036 \times 10^{-4}$	c_9	$2.224209172496374 \times 10^{0}$
c_2	$2.945531440279683 \times 10^{-3}$	c_{10}	$-5.792361707073261 \times 10^{0}$
c_3	$-8.709066576837676 \times 10^{-3}$	c_{11}	$-4.130276365929783 \times 10^{-2}$
c_4	$4.017568440673568 \times 10^{-1}$	c_{12}	$1.040801735231354 \times 10^{1}$
c_5	$3.230762888122312 \times 10^{-2}$	c_{13}	$-6.331712455883370 \times 10^{1}$
c_6	$5.768988513026145 \times 10^{0}$	c_{14}	$3.484665863364574 \times 10^{-1}$
c_7	$2.338576034271299 \times 10^{-2}$	c_{15}	1
c_8	$2.381070373870987 \times 10^{-1}$	c_{16}	1

$$y_{12}(A) = (y_{02}(A) + c_3A^2 + c_4A)(y_{02}(A) + c_5A^2) + c_6y_{02}(A) + c_7A^2, \tag{11}$$

$$\begin{aligned} y_{22}(A) = {} & (y_{12}(A) + c_8A^2 + c_9A)(y_{12}(A) + c_{10}y_{02}(A) + c_{11}A) \\ & + c_{12}y_{12}(A) + c_{13}y_{02}(A) + c_{14}A^2 + c_{15}A + c_{16}I, \end{aligned} \tag{12}$$

where, analogously to [9, Example 5.1], A^2 is computed once and stored to be reused in all the computations, the degree of polynomial $y_{22}(A)$ is $m = 16$ and it can be evaluated with a cost $4M$. Note that with Paterson–Stockmeyer method and cost $4M$, using (4) and Table 1, the maximum available order for $T_m(A)$ is $m = 9$.

Similarly to [9, Example 5.1], if we rewrite $y_{22}(A)$ as a matrix polynomial of degree $m = 16$, and equate the coefficients of the matrix powers A^i, $i = 0, 1, \ldots, 15$, of the rewritten $y_{22}(A)$ to the corresponding coefficients of $T_{15}(A)$, we obtain a nonlinear system of 16 equations with 16 unknown coefficients c_i, $i = 1, 2, \ldots, 16$.

Note that MATLAB Symbolic Math Toolbox `solve` function could not give the general solution of the expressions (57)–(59) proposed in [9, Example 5.1], returning one numerical approximation of the coefficients instead. Moreover, we had to use a special ordering of the coefficients so that `solve` obtained this numerical approximation, see [12, Section 4.2]. In the following we describe a method to obtain more solutions for the coefficients c_i. The stability recommendations from [9, p. 243] propose to select the solution giving the lesser error in the system of equations, when substituting the coefficients c_i in IEEE double precision arithmetic.

$y_{12}(A)$ from (11) can be written as a polynomial of degree 8

$$y_{12}(A) = \sum_{i=2}^{8} a_iA^i. \tag{13}$$

Taking into account the four existing solutions from [9, Example 1] for evaluating polynomials of degree 8 with cost $3M$, see (25) and (26) from [9], it follows that

$$y_{02}(A) = \pm A^2(\sqrt{a_8}A^2 + a_7/(2\sqrt{a_8})A). \tag{14}$$

If we write $y_{22}(A)$ as a polynomial of degree $m = 16$ and equate the coefficients of the matrix powers A^i to the Taylor coefficients $1/i!$, for $i = 0, 1, \ldots, 15$, a system of 16 equations with 16 variables a_i, $i = 2, 3, \ldots, 8$, and c_i, $i = 8, 9, \ldots, 16$, arises. This system can be solved using `solve` function from MATLAB Symbolic Math Toolbox obtaining coefficients a_i, $i = 2, 3, \ldots, 8$, and c_i, $i = 8, 9, \ldots, 16$. Then, [9, Example 3.1] gives four solutions for the coefficients c_i, $i = 1, 2, \ldots, 7$, from (10) and (11) that allow to evaluate (13) with such coefficients a_i. From now on, we will call these different solutions of coefficients c_i based on coefficients a_i as nested solutions. From all the four solutions we selected the one giving the lesser error in the original system of equations when rounding the coefficients to IEEE double precision arithmetic, see Table 3. For that solution if we write $y_{22}(A)$ as a polynomial of degree $m = 16$ the coefficient of A^{16}, denoted by b_{16}, and its relative error with respect to the corresponding Taylor polynomial coefficient are

$$b_{16} = 2.608368698098254 \times 10^{-14}, \qquad |b_{16} - 1/16!|16! \approx 0.454, \tag{15}$$

where the coefficient b_{16} is presented in IEEE double precision arithmetic. Note that there is another real solution that gives $|b_{16} - 1/16!|16! \approx 2.510 > 0.454$, and then we discarded it. Note also that the relative error in (15) is the same as in [9, Example 5.1], see (60) from [9].

The fact that

$$y_{22}(A) = T_{15}(A) + b_{16}A^{16}, \tag{16}$$

will have interesting implications in the error analysis of the algorithm. The evaluation formulas (11) and (12) are slightly different from (58) and (59) of [9, Example 5.1] so that the coefficient of the last matrix A in (12) is 1, instead of c_2 from [9, Table 9].

Table 4
Coefficients of y_{03}, y_{13}, y_{23} from (17)–(19) for computing a Taylor based matrix exponential approximation of order $m = 21$.

c_1	$1.161658834444880\times10^{-6}$	c_{11}	$1.392249143769798\times10^{-1}$
c_2	$4.500852739573010\times10^{-6}$	c_{12}	$-2.269101241269351\times10^{-3}$
c_3	$5.374708803114821\times10^{-5}$	c_{13}	$-5.394098846866402\times10^{-2}$
c_4	$2.005403977292901\times10^{-3}$	c_{14}	$3.112216227982407\times10^{-1}$
c_5	$6.974348269544424\times10^{-2}$	c_{15}	$9.343851261938047\times10^{0}$
c_6	$9.418613214806352\times10^{-1}$	c_{16}	$6.865706355662834\times10^{-1}$
c_7	$2.852960512714315\times10^{-3}$	c_{17}	$3.233370163085380\times10^{0}$
c_8	$-7.544837153586671\times10^{-3}$	c_{18}	$-5.726379787260966\times10^{0}$
c_9	$1.829773504500424\times10^{0}$	c_{19}	$-1.413550099309667\times10^{-2}$
c_{10}	$3.151382711608315\times10^{-2}$	c_{20}	$-1.638413114712016\times10^{-1}$

3.3. Evaluation of a Taylor based approximation of order 21

In a similar way to [9, Example 5.1] we developed the following formulas to evaluate a Taylor approximation of order $m = 21$ of the matrix exponential

$$y_{03}(A) = A^3(c_1A^3 + c_2A^2 + c_3A), \tag{17}$$

$$y_{13}(A) = (y_{03}(A) + c_4A^3 + c_5A^2 + c_6A)(y_{03}(A) + c_7A^3 + c_8A^2) + c_9y_{03}(A) + c_{10}A^3 + c_{11}A^2, \tag{18}$$

$$\begin{aligned} y_{23}(A) = {} & (y_{13}(A) + c_{12}A^3 + c_{13}A^2 + c_{14}A)(y_{13}(A) + c_{15}y_{03}(A) + c_{16}A) \\ & + c_{17}y_{13}(A) + c_{18}y_{03}(A) + c_{19}A^3 + c_{20}A^2 + A + I, \end{aligned} \tag{19}$$

where A^2 and A^3 are computed once and reused in all the computations, the degree of polynomial $y_{23}(A)$ is $m = 24$, and it can be evaluated with a cost $5M$. Note that with cost $5M$ the maximum available order for $T_m(A)$ using Paterson–Stockmeyer method is $m = 12$, see (4) and Table 1.

Proceeding analogously to Section 3.2, writing

$$y_{13}(A) = \sum_{i=2}^{12} a_iA^i, \tag{20}$$

and taking into account the solutions of (38) from [9], it follows that

$$y_{03}(A) = \pm A^3(\sqrt{a_{12}}A^3 + a_{11}/(2\sqrt{a_{12}})A^2 + (4a_{10}a_{12} - a_{11}^2)/(8a_{12}^{3/2})A). \tag{21}$$

Then, similarly to Section 3.2, we use function `solve` from the MATLAB Symbolic Math Toolbox to obtain a_i, $i = 2, 3, \ldots, 12$, and c_i, $i = 12, 13, \ldots, 20$, so that $y_{23}(x) = T_{21}(x) + O(x^{22})$, for scalar x. Then, we use the solutions from [9, pp. 237–240] to evaluate (20) with (17) and (18), obtaining different nested solutions for coefficients c_i, $i = 1, 2, \ldots, 11$. Again, according to the stability recommendations from [9, p. 243], we selected the solution given in Table 4. For that solution, if we write $y_{23}(A)$ from (19) as a polynomial of degree $m = 24$, the coefficients of A^{22}, A^{23} and A^{24}, denoted by b_{22}, b_{23} and b_{24}, respectively, and their relative errors with respect to the corresponding Taylor polynomial coefficients are

$$b_{22} = 5.010366348377648\times10^{-22}, \qquad |b_{22} - 1/22!|22! \approx 0.437, \tag{22}$$

$$b_{23} = 2.822218236752230\times10^{-23}, \qquad |b_{23} - 1/23!|23! \approx 0.270, \tag{23}$$

$$b_{24} = 1.821018669767511\times10^{-24}, \qquad |b_{24} - 1/24!|24! \approx 0.130, \tag{24}$$

where the coefficients b_{22}, b_{23} and b_{24} are presented in IEEE double precision arithmetic. Again, the fact that

$$y_{23}(A) = T_{21}(A) + b_{22}A^{22} + b_{23}A^{23} + b_{24}A^{24}, \tag{25}$$

will have interesting implications in the error analysis of the algorithm.

3.4. Evaluation of Taylor approximations $T_{24}(A)$ and $T_{30}(A)$

The coefficients for the evaluation of $T_{24}(A)$ corresponding to

$$y_{04}(A) = A^4(c_1A^4 + c_2A^3 + c_3A^2 + c_4A), \tag{26}$$

$$\begin{aligned} y_{14}(A) =& (y_{04}(A)+c_5A^4+c_6A^3+c_7A^2+c_8A)(y_{04}(A)+c_9A^4+c_{10}A^3 \\ &+c_{11}A^2)+c_{12}y_{04}(A)+c_{13}A^4+c_{14}A^3+c_{15}A^2+c_{16}A, \end{aligned} \tag{27}$$

$$\begin{aligned} T_{24}(A) =& y_{14}(A)(y_{04}(A)+c_{17}A^4+c_{18}A^3+c_{19}A^2+c_{20}A) \\ &+c_{21}A^4+c_{22}A^3+c_{23}A^2+A+I, \end{aligned} \tag{28}$$

can be obtained in a similar way to Sections 3.2 and 3.3. For the values of the coefficients in IEEE double precision arithmetic see case $m=24$ from the nested function EFFEVPOL from the MATLAB implementation of the proposed algorithm `expmpol` at http://personales.upv.es/~jorsasma/software/expmpol.m

Note that using (26)–(28), $T_{24}(A)$ can be evaluated with a cost $6M$. For $6M$ the maximum available order with Paterson–Stockmeyer method is 16, see Table 1.

Similarly, the coefficients for the evaluation of $T_{30}(A)$ corresponding to

$$y_{05}(A) = A^5(c_1A^5+c_2A^4+c_3A^3+c_4A^2+c_5A), \tag{29}$$

$$\begin{aligned} y_{15}(A) =& (y_{05}(A)+c_6A^5+c_7A^4+c_8A^3+c_9A^2+c_{10}A) \\ &\times(y_{05}(A)+c_{11}A^5+c_{12}A^4+c_{13}A^3+c_{14}A^2) \\ &+c_{15}y_{05}(A)+c_{16}A^5+c_{17}A^4+c_{18}A^3+c_{19}A^2+c_{20}A, \end{aligned} \tag{30}$$

$$\begin{aligned} T_{30}(A) =& y_{15}(A)(y_{05}(A)+c_{21}A^5+c_{22}A^4+c_{23}A^3+c_{24}A^2+c_{25}A) \\ &+c_{26}A^5+c_{27}A^4+c_{28}A^3+c_{29}A^2+A+I, \end{aligned} \tag{31}$$

can be obtained. For the coefficient values see case $m=30$ from the nested function EFFEVPOL from `expmpol`.

Note that using (29)–(31), $T_{30}(A)$ can be evaluated with a cost $7M$, whereas for that cost the maximum available order with Paterson–Stockmeyer method is 20, see Table 1.

Finally, note that in the evaluation of Padé approximants from [14] many of the products of the numerator can be reused for the denominator. In general, this is not so advantageous if we compute the numerator and denominator with the evaluation algorithms from [9]. For instance, to evaluate r_{88} using the method of Section 3.1 to evaluate both numerator and denominator polynomials of degree 8 we need one matrix product to evaluate A^2, two more products to evaluate the numerator and two more for the denominator. Therefore, the cost for evaluating r_{88} is $5M+1D$ providing an order of the approximation 16. Note that with a lower cost of $5M$, y_{23} from Section 3.3 gives an order of approximation 21.

4. Error analysis

For completeness of the exposition we summarize some results for the error analysis of [6]. Denoting the remainder of the Taylor series as $R_m(A)=\sum_{k\geq m+1}A^k/k!$, for a scaled matrix $2^{-s}A$, $s\in\mathbb{N}\cup\{0\}$, we can write

$$\left(T_m(2^{-s}A)\right)^{2^s} = e^A\left(I+g_{m+1}(2^{-s}A)\right)^{2^s} = e^{A+2^s h_{m+1}(2^{-s}A)}, \tag{32}$$

$$g_{m+1}(2^{-s}A) = -e^{-2^{-s}A}R_m(2^{-s}A), \;\; h_{m+1}\left(2^{-s}A\right) = \log\left(I+g_{m+1}(2^{-s}A)\right), \tag{33}$$

where log denotes the principal logarithm, $h_{m+1}(X)$ is defined in the set $\Omega_m=\left\{X\in\mathbb{C}^{n\times n}:\rho\left(e^{-X}T_m(X)-I\right)<1\right\}$, and both $g_{m+1}(2^{-s}A)$ and $h_{m+1}(2^{-s}A)$ are holomorphic functions of A in Ω_m and then commute with A. As showed in [4], $\Delta A=h_{m+1}(2^{-s}A)$ is the backward absolute error, and $\Delta E=g_{m+1}(2^{-s}A)$ is the forward relative error, from the approximation of e^A by the Taylor series with scaling and squaring in exact arithmetic.

Using the scalar Taylor series in (33) one gets

$$g_{m+1}(x) = \sum_{k\geq m+1} b_k^{(m)}x^k, \quad h_{m+1}(x) = \sum_{k\geq 1}\frac{(-1)^{k+1}(g_{m+1}(x))^k}{k} = \sum_{k\geq m+1} c_k^{(m)}x^k, \tag{34}$$

where $b_k^{(m)}$ and $c_k^{(m)}$ depend on the order m. Moreover, $b_k^{(m)}=c_k^{(m)}$, $k=m+1,m+2,\ldots,2m+1$, and if $\|h_{m+1}(2^{-s}A)\|\ll 1$ or $\|g_{m+1}(2^{-s}A)\|\ll 1$, then

$$\Delta A = h_{m+1}(2^{-s}A) \approx g_{m+1}(2^{-s}A) = \Delta E, \tag{35}$$

see [5].

Choosing s so that

$$\left\|h_{m+1}\left(2^{-s}A\right)\right\| \leq \max\left\{1,\left\|2^{-s}A\right\|\right\}u, \tag{36}$$

Table 5
Maximal values $\Theta_m = \|2^{-s}A\|$ such that $\tilde{h}_{m+1}(\Theta_m) \le \max\{1, \Theta_m\}u$, coefficient ratios $c_{m+1}^{(m)}/c_{m+2}^{(m)}$ and values $u/c_{m+2}^{(m)}$. Note that $m = 15+$ and $m = 21+$ correspond to approximations (16) and (25) that are not exactly $T_{15}(A)$ and $T_{21}(A)$.

m	Θ_m	$\lvert c_{m+1}^{(m)}/c_{m+2}^{(m)}\rvert$	$\lvert u/c_{m+2}^{(m)}\rvert$
1	$1.490116111983279\times10^{-8}$	3/2	3.33×10^{-16}
2	$8.733457513635361\times10^{-6}$	4/3	8.88×10^{-16}
4	$1.678018844321752\times10^{-3}$	6/5	1.60×10^{-14}
8	$1.773082199654024\times10^{-2}$	10/9	4.48×10^{-11}
15+	$6.950240768069781\times10^{-1}$	1.15	5.87×10^{-3}
21+	1.682715644786316	1.03	2.93×10^{5}
24	2.219048869365090	26/25	1.79×10^{9}
30	3.539666348743690	32/31	9.42×10^{17}

where $u=2^{-53}$ is the unit roundoff in IEEE double precision arithmetic, then:

- If $2^{-s}\|A\| \ge 1$, then $\Delta A \le \|A\|u$ and using (32) one gets $(T_m(2^{-s}A))^{2^s} = e^{A+\Delta A} \approx e^A$,
- If $2^{-s}\|A\| < 1$, using (32)–(35) and the Taylor series, if (36) holds one gets

$$\begin{aligned}\left\|R_m\left(2^{-s}A\right)\right\| &= \left\|e^{2^{-s}A}g_{m+1}(2^{-s}A)\right\| \\ &\approx \left\|T_m\left(2^{-s}A\right)h_{m+1}(2^{-s}A)\right\| \le \left\|T_m\left(2^{-s}A\right)\right\|u.\end{aligned} \tag{37}$$

Hence, by (37), in IEEE double precision arithmetic one gets that $T_m(2^{-s}A) + R_m(2^{-s}A) \approx T_m(2^{-s}A)$.

Using MATLAB symbolic Math Toolbox, high precision arithmetic, 200 series terms and a zero finder we obtained the maximal values Θ_m of $\Theta = \|2^{-s}A\|$, see Table 5, such that, using the notation of Theorem 1

$$||h_{m+1}(2^{-s}A)|| \le \tilde{h}_{m+1}(\Theta) = \textstyle\sum_{k\ge m+1} |c_k^{(m)}|\Theta^k \le \max\{1, \Theta\}\, u. \tag{38}$$

Hence, if $||2^{-s}A|| \le \Theta_m$ then (36) holds. For orders $m = 20, 25, 30$ where $\Theta_m > 1$ by (14) of [6] the next bound holds

$$||h_{m+1}\left(2^{-s}A\right)|| \le \tilde{h}_{m+1}\left(||2^{-s}A||\right) = \tilde{h}_{m+1}(\Theta) \le \Theta u, \quad 0 \le \Theta \le \Theta_m. \tag{39}$$

The previous analysis is valid for the orders m_k used in [6], and also for the new evaluation formulas for $T_8(A)$, $T_{24}(A)$ and $T_{30}(A)$ from Section 3. For $y_{22}(A)$ from (16) it is easy to show that all the previous error analysis is valid substituting $R_m(2^{-s}A)$ in (33) for

$$\tilde{R}_{15+}(2^{-s}A) = \exp(2^{-s}A) - y_{22}(2^{-s}A) = R_{15}(2^{-s}A) - b_{16}A^{16}. \tag{40}$$

Similarly, for $y_{23}(A)$ from (25) all the previous error analysis is also valid substituting $R_m(2^{-s}A)$ in (33) for

$$\tilde{R}_{21+}(2^{-s}A) = \exp(2^{-s}A) - y_{22}(2^{-s}A) = R_{21}(2^{-s}A) - \sum_{i=22}^{24} b_i A^i. \tag{41}$$

Therefore, the new Θ_{15+} and Θ_{21+} values are obtained in a similar way to the other values obtained in [6], where the suffix " + " has been added. The corresponding values are given in Table 5. Comparing those values to the actual values obtained for $T_{15}(A)$, i.e. $\Theta_{15} \approx 0.658$ and $T_{21}(A)$, i.e. $\Theta_{21} \approx 1.624$, we see that both values are lower than the corresponding values for Θ_{15+} and Θ_{21+}. That is the reason for adding the suffix " + " in $m = 15+$ and $m = 21+$, notation that will be used from now on. The higher values Θ_{15+} and Θ_{21+} will imply a lower order or scaling for certain matrices. For instance, this may be the case for matrices with $\Theta_{15} \le \|A\| \le \Theta_{15+}$ and $\Theta_{21} \le \|A\| \le \Theta_{21+}$, see Section 5. We have checked that bound (39) also holds for the approximation $y_{23}(A)$ from (25) and for $T_{24}(A)$. Hence, the optimal orders m to be used are in the set

$$m_k^* = \{1, 2, 4, 8, 15+, 21+, 24, 30\}. \tag{42}$$

5. New Taylor algorithms

This section summarizes the results from the scaling algorithm from [6] and gives the new algorithms. The scaling algorithm from [6] can be applied directly to the Taylor approximations $T_8(A)$, $T_{24}(A)$, $y_{22}(A)$ from (16) and $y_{23}(A)$ from (25) from Section 3 using the new values for all the parameters from Table 5, see [6, Table 2]. For the values corresponding to $m = 15+$ and $21+$ the remainders (40) and (41) were used in (33) and the coefficients $c_{m+1}^{(m)}$ and $c_{m+2}^{(m)}$ from (34) were obtained symbolically.

For all norms appearing in the scaling algorithm we will use the 1-norm, and m_M will be the maximum allowed Taylor order. In [6, Section 4.1] the minimum value of α_p from Theorem 1, denoted by α_{min}, was used to determine an initial scaling parameter. It was taken as, see (16) from [6],

$$\alpha_{min} \approx \max\{a_{m+1}^{1/(m+1)}, a_{m+2}^{1/(m+2)}\}, \tag{43}$$

where a_{m+1} and a_{m+2} were the 1–norm estimation of $||A^{m+1}||$ and $||A^{m+2}||$ using the block 1–norm estimation algorithm of [13]. This algorithm has a cost $O(n^2)$, negligible compared to the cost of a matrix product, i.e. $O(n^3)$ if the matrix A is big enough. Alternatively, we can also use bounds for $||A^{m+1}||$ and $||A^{m+2}||$ based on products of norms of the matrix powers computed at each step of the algorithm, similarly to [7, Algorithm 3] and [8, Algorithm 3]. Then, no norm estimations are used. For instance, if $m = 4$ and A^2 is known, then $\|A^{m+1}\| = \|A^5\| \le \|A^2\|^2\|A\|$.

In numerical tests with matrices 128×128, see Section 6, we detected that the cost of the 1–norm estimation algorithm of [13] is not negligible. Then, a MATLAB implementation of Algorithm 1 with no norm estimations is given. Moreover, an implementation that reduces the number of norm estimations compared to that of [6, Algorithm 2] is also given. Both options are possible in function `expmpol(A,kmax,NormEst)` from http://personales.upv.es/∼jorsasma/software/expmpol.m

This function can use $m_M = 24$ (`kmax=6`) and $m_M = 30$ (`kmax=7`), and selecting the use of the norm estimation algorithms from [13] (`NormEst=1`), or not (`NormEst=0`). `expmpol` does not consider preprocessing and postprocessing of matrix A, as indicated in Section 2.

Algorithm 2, called ms_selectNoNormEst, presents the proposed algorithm for the selection of the order m and the scaling parameter s to compute the matrix exponential using no norm estimations and $m_M = 24$. It corresponds to Step 2 of

Algorithm 2 Algorithm ms_selectNoNormEst: Given a matrix $A \in \mathbb{C}^{n\times n}$ this algorithm obtains the order m and scaling parameter s for Taylor approximation of order $m \leqslant m_M = 24$ using no norm estimations of matrix powers.

Input: $A \in \mathbb{C}^{n\times n}$
Output: Order m, scaling parameter s, and some matrixpowers A^i, $1 \le i \le 3$.

```
1: Set $\Theta_m$, $c^{(m)}_{m+1}/c^{(m)}_{m+2}$ and$u/c^{(m)}_{m+2}$ values from Table 5
2: $a_1 = ||A||$
3: $s = 0$                                                          ▷ Null scaling tests
4: if $a_1 < \Theta_1$, $m = 1$, quit                                ▷ Test $m = 1$ with $s = 0$
5: $A_2 = A^2$, $a_2 = \|A_2\|$
6: $a_3 = a_2a_1$, $a_4 = a_2^2$, $b = \max\{1, ||A||\} \cdot u/c^{(2)}_4$
7: if $c^{(2)}_3/c^{(2)}_4 \cdot a_3 + a_4 \le b$, $m = 2$, quit        ▷ Test $m = 2$ with $s = 0$
8: $a_5 = a_2^2a_1$, $a_6 = a_2^3$, $b = \max\{1, ||A||\} \cdot u/c^{(4)}_6$
9: if $c^{(4)}_5/c^{(4)}_6 \cdot a_5 + a_6 \le b$, $m = 4$, quit        ▷ Test $m = 4$ with $s = 0$
10: $a_9 = a_2^4a_1$, $a_{10} = a_2^5$, $b = \max\{1, ||A||\} \cdot u/c^{(8)}_{10}$
11: if $c^{(8)}_9/c^{(8)}_{10} \cdot a_9 + a_{10} \le b$, $m = 8$, quit  ▷ Test $m = 8$ with $s = 0$
12: $a_{16} = a_2^8$, $a_{17} = a_2^8a_1$, $b = \max\{1, ||A||\} \cdot u/c^{(15)}_{17}$
13: if $c^{(15)}_{16}/c^{(15)}_{17} \cdot a_{16} + a_{17} \le b$, $m = 15$, quit   ▷ Test $m = 15+$ with $s = 0$
14: $A_3 = A_2A$, $a_3 = \|A_3\|$
15: $a_{22} = \min\{a_2^{11}, a_3^6a_2^2, a_3^7a_1\}$,$a_{23} = \min\{a_2^{10}a_3, a_3^7a_2\}$, $b = \max\{1, ||A||\} \cdot u/c^{(21)}_{23}$
16: if $c^{(21)}_{22}/c^{(21)}_{23} \cdot a_{22} + a_{23} \le b$, $m = 21$, quit   ▷ Test $m = 21+$ with $s = 0$
17: $a_{25} = \min\{a_2^{11}a_3, a_3^7a_2^2, a_3^8a_1\}$,$a_{26} = \min\{a_2^{13}, a_3^8a_2\}$, $b = \max\{1, ||A||\} \cdot u/c^{(24)}_{26}$
18: if $c^{(24)}_{25}/c^{(24)}_{26} \cdot a_{25} + a_{26} \le b$, $m = 24$, quit   ▷ Test$m = 24$ with $s = 0$
19: $\alpha_{min} = \max\{a_{25}^{1/25}, a_{26}^{1/26}\}$
20: $s = \lceil \log_2(\alpha_{min}/\Theta_m)\rceil$                    ▷ Calculate initial scaling $s$
21: if $s > 0$ then
22:     $s_{red} = s - 1$                                   ▷ Check if (45)holds reducing the scaling
23:     $b = \max\{1, ||A||/2^{s_{red}})\} \cdot u/c^{(24)}_{26}$
24:     if $c^{(24)}_{25}/c^{(24)}_{26} \cdot a_{25}/2^{25s_{red}} + a_{26}/2^{26s_{red}} \le b$ then
25:         $s = s_{red}$                                  ▷ (45) holds, then $s = s_{red}$
26:     end if
27:     $b = \max\{1, ||A||/2^s\} \cdot u/c^{(21)}_{23}$     ▷ Testif the scaled matrix allows using $m_{M-1} = 21+$
28:     if $c^{(21)}_{22}/c^{(21)}_{23} \cdot a_{22}/2^{22s} + a_{23}/2^{23s} \le b$ then
29:         $m = 21$, quit
30:     else
31:         $m = 24$, quit                      ▷ The scaled matrix does not allow using $m_{M-1}$
32:     end if
33: end if
```

Algorithm 1. The selection of m and s is analogous to that from [6, Algorithm 2], but using bounds for $||A^{m+1}||$ and $||A^{m+2}||$ based on products of norms of matrix powers. The version of Algorithm 2 allowing both $m_M = 24$ and $m_M = 30$ is straight-

Table 6
Cost of expmpol in terms of matrix products, denoted by $C_0 = C_{\texttt{expmpol}}$, and relative cost comparison $R = C_{\texttt{function}}/C_0(\%)$ between expmpol and functions expmpoln, expmpol_orig, expmspl, exptayns and expm_new. All the Taylor approximations use maximum order $m_M = 30$.

	C_0	$R_{\texttt{expmpoln}}$	$R_{\texttt{expmpol\_orig}}$	$R_{\texttt{expmpspl}}$	$R_{\texttt{exptayns}}$	$R_{\texttt{expm\_new}}$
Test 1	948	104.01	100	125.84	119.41	127.14
Test 2	703	104.84	100	132.01	121.48	116.31
Test 3	270	108.52	100	132.84	128.15	121.73

forward, and a MATLAB implementation allowing both values of m_M is given as the nested function ms_selectNoNormEst of expmpol.

In Steps 1–18, Algorithm 2 checks if any of the Taylor optimal orders $m_k = 1, 2, 4, 8, 15+, 21+$ and $m_M = 24$, satisfies (36) without scaling, i.e. $s = 0$, using the bounds provided in Section 4 and considering the two first terms of the series of $h_{m+1}(x)$ from (34), in the same way as (19) from [6]

$$\frac{\|h_{m+1}(2^{-s}A)\|}{|c_{m+2}^{(m)}|} \le \sum_{k=m+1}^{m+2} \left| \frac{c_k^{(m)}}{c_{m+2}^{(m)}} \right| \frac{a_k}{2^{sk}} \le \max\left\{1, \left\|2^{-s}A\right\|\right\} \frac{u}{|c_{m+2}^{(m)}|}, \tag{44}$$

As mentioned above, we obtain bounds of $||A^{m+1}||$ and $||A^{m+2}||$ using products of norms of matrix powers previously computed, see Steps 6, 8, etc.

If no value of $m_k \le m_M$ satisfies (44) with $s = 0$, the algorithm computes α_{min} using only the bounds for $||A^{m+1}||$ and $||A^{m+2}||$, and determines the initial scaling parameter s in Steps 19–20. Then, if $s > 0$, the algorithm checks in Steps 22–26 if the initial scaling parameter can be reduced, checking if (44) holds with $s_{red} = s - 1$.

Finally, In Steps 27–32 we check if $m_{M-1} = 21+$ can be used with the same scaling s, reducing the cost, similarly to [6, Algorithm 2].

After selecting m and s, the matrix polynomial evaluation formulas from Section 3 will be used. The implementation of this part is straightforward using the formulas and coefficients from Section 3. A MATLAB implementation can be seen at function EFFEVPOL of expmpol. And, finally, the s squarings from Steps 4–6 are done. See function expmpol for a complete implementation of the algorithm.

For simplicity, the special case of nilpotent matrices, i.e. $\|A^i\| = 0 \times I$, $i = 2, 3, 4$, or 5, is not included in Algorithm 2, but they are considered in the MATLAB implementation ms_selectNoNormEst in expmpol.

The selection of m and s in Algorithm 2 is analogous to that of [6, Algorithm 2] with the new optimal orders m from Section 3, and using no norm estimations for $\|A^{m+1}\|$ and $\|A^{m+2}\|$.

With respect to algorithms for the selection of m and s using norm estimations of matrix powers [13], it is easy to implement a direct adaptation of [6, Algorithm 2] that selects the optimal m and s for the new values m_k^* from (42) using the corresponding parameter values from Table 5. We will denote this algorithm as expmpol_orig. However, [6, Algorithm 2] computes norm estimations of matrix powers that can be avoided. MATLAB function ms_selectNormEst has been developed reducing the number of norm estimations of matrix powers with respect to expmpol_orig, see expmpol. It is similar to function ms_selectNormEst from [7] for selecting m and s for computing the matrix cosine with a Taylor algorithm. The reduction of estimations will increase the efficiency for small matrices, see Section 6. The selection of m and s in ms_selectNormEst is based essentially on testing if the order m_k^* satisfies (44) using products of known norms of matrix powers and, if so, then, testing if m_{k-1}^* also satisfies (44) using estimations of $\|A^{m_{k-1}^*+1}\|$ and $\|A^{m_{k-1}^*+2}\|$. General steps for ms_selectNormEst with $m_M = 24$ are:

1. Set $s = 0$.
2. Test if one order $m = m_k^*$ from (42), with $k = 2, 3, 4$, i.e. $m = \{4, 8, 15\}$, satisfies (44) with $s = 0$, using bounds of matrix powers based on products of $\|A\|$ and $\|A^2\|$. If so, then: if (44) holds with $m = m_{k-1}^*$ using the estimation of $\|A^{m+1}\|$ and $\|A^{m+2}\|$, then $m = m_{k-1}^*$, else $m = m_k$, and the algorithm quits. Note that, similarly to [6, Algorithm 2], we will not estimate $\|A^{m+2}\|$ if
$$\left| \frac{c_k^{(m)}}{c_{m+2}^{(m)}} \right| \frac{\|A^{m+1}\|}{2^{s(m+1)}} > \max\left\{1, \left\|2^{-s}A\right\|\right\} \frac{u}{|c_{m+2}^{(m)}|}, \tag{45}$$
see (44).
3. If $m = 15+$ satisfies (44) with $s = 0$ using estimations for $\|A^{16}\|$ and $\|A^{17}\|$ then: if (44) holds with $m = 8$ using estimations of $\|A^9\|$ and $\|A^{10}\|$ then $m = 8$, else $m = 15+$, and the algorithm quits. Testing $m = 15+$ with norm estimations is necessary in this step because A^3 is computed in the following step, and A^3 is not used for the evaluation of the approximation $y_{22}(A)$ of order $m = 15+$ from (10)–(12).
4. A^3 is computed.

Table 7
Execution times of `expmpol` in seconds, denoted by $t_0 = t_{\texttt{expmpol}}$, and relative execution time comparison $R = t_{\texttt{function}}/t_0(\%)$ between `expmpol` and functions `expmpoln`, `expmpol_orig`, `expmspl`, `exptayns` and `expm_new`. All the Taylor approximations use maximum order $m_M = 30$.

	t_0	$R_{\texttt{expmpoln}}$	$R_{\texttt{expmpol\_orig}}$	$R_{\texttt{expmpspl}}$	$R_{\texttt{exptayns}}$	$R_{\texttt{expm\_new}}$
Test 1	0.2244	60.39	141.10	156.80	123.88	232.08
Test 2	0.1798	57.62	146.25	161.32	127.49	229.63
Test 3	0.0825	65.68	154.13	184.78	143.71	206.19

Table 8
Cost of `expmpol` with $m_M = 24$ in terms of matrix products, denoted by $C_0 = C_{\texttt{expmpol}}$, and relative cost comparison $R = C_{\texttt{function}}/C_0(\%)$ between `expmpol` ($m_M = 24$) and functions `expmpoln` ($m_M = 24$), `expmpol_orig` ($m_M = 24$), `expmspl` ($m_M = 30$), `exptayns` ($m_M = 25$) and `expm_new`.

	C_0	$R_{\texttt{expmpoln}}$	$R_{\texttt{expmpol\_orig}}$	$R_{\texttt{expmpspl}}$	$R_{\texttt{exptayns}}$	$R_{\texttt{expm\_new}}$
Test 1	887	107.22	100	134.50	127.62	135.89
Test 2	662	104.38	100	140.18	129.00	123.51
Test 3	259	109.27	100	138.48	133.59	126.90

Table 9
Execution times of `expmpol` with $m_M = 24$ in seconds, denoted by $t_0 = t_{\texttt{expmpol}}$, and relative execution time comparison $R = t_{\texttt{function}}/t_0(\%)$ between `expmpol` ($m_M = 24$) and functions `expmpoln` ($m_M = 24$), `expmpol_orig` ($m_M = 24$), `expmspl` ($m_M = 30$), `exptayns` ($m_M = 25$) and `expm_new`.

	t_0	$R_{\texttt{expmpoln}}$	$R_{\texttt{expmpol\_orig}}$	$R_{\texttt{expmpspl}}$	$R_{\texttt{exptayns}}$	$R_{\texttt{expm\_new}}$
Test 1	0.2112	56.55	122.08	165.38	119.29	244.65
Test 2	0.1730	51.08	126.39	166.27	120.43	236.35
Test 3	0.0820	60.44	137.85	183.76	135.12	204.12

5. If $m = 21+$ satisfies (44) with $s = 0$ using products of known norms of matrix powers, including the estimations from Step 3, then $m = 21+$ is selected and the algorithm quits. For instance, a bound for $\|A^{m+1}\|$ is $\|A^{22}\| \le \|A^{16}\| \min\{\|A^3\|^2, \|A^2\|^3\}$, where $\|A^{16}\|$ was estimated in Step 3.
6. If $m = 24$ satisfies (44) using products of known norms of matrix powers then: if (44) holds with $m = 21+$ and estimations of $\|A^{22}\|$ and $\|A^{23}\|$, then $m = 21+$, else $m = 24$, and the algorithm quits.
7. If $m = 24$ satisfies (44) using estimations of $\|A^{25}\|$ and $\|A^{26}\|$ then: if (44) holds with $m = 21+$ and estimations of $\|A^{22}\|$ and $\|A^{23}\|$, then $m = 21+$, else $m = 24$, and the algorithm quits.
8. The rest is similar to Steps 22–32 from Algorithm 2, combining norm estimations of matrix powers and bounds of norms of matrix powers based on products of known norms of matrix powers.

The extension of the previous algorithm to $m_M = 30$ is straightforward and a complete MATLAB implementation of the algorithm can be seen in MATLAB function `ms_selectNormEst` of `expmpol`, allowing $m_M = 24$ and $m_M = 30$.

6. Numerical experiments and conclusions

In this section we compare a MATLAB implementation of the new algorithm `expmpol`, available at http://personales.upv.es/~jorsasma/software/expmpol.m with the following three legacy MATLAB implementations for the computation of the matrix exponential:

- `expmspl`: MATLAB implementation based on matrix splines from [8], available at http://personales.upv.es/~jorsasma/software/expmspl.m. This function uses Taylor approximations with a fixed maximum order $m_M = 30$.
- `exptayns`: MATLAB implementation of exptaynsv3 from [6], available at http://personales.upv.es/~jorsasma/software/exptaynsv3.m. For this function the argument denoted as kmax determines the maximum Taylor approximation order used.
- `expm_new`: MATLAB implementation based on Padé approximation from [14]. For this function the argument denoted as schur_fact is not used, i.e. an initial transformation to complex or real Schur form is not used. Anyway, this kind of transformations can be also applied to our algorithms, see Test 4 of [7, Section 3] for the application of a Schur form to a Taylor algorithm for the matrix cosine. The accuracy results were similar to those of a state-of-the-art Padé algorithm that used the same Schur form, but with a lower cost.

We use the following notations for different versions of the algorithms proposed in this paper:

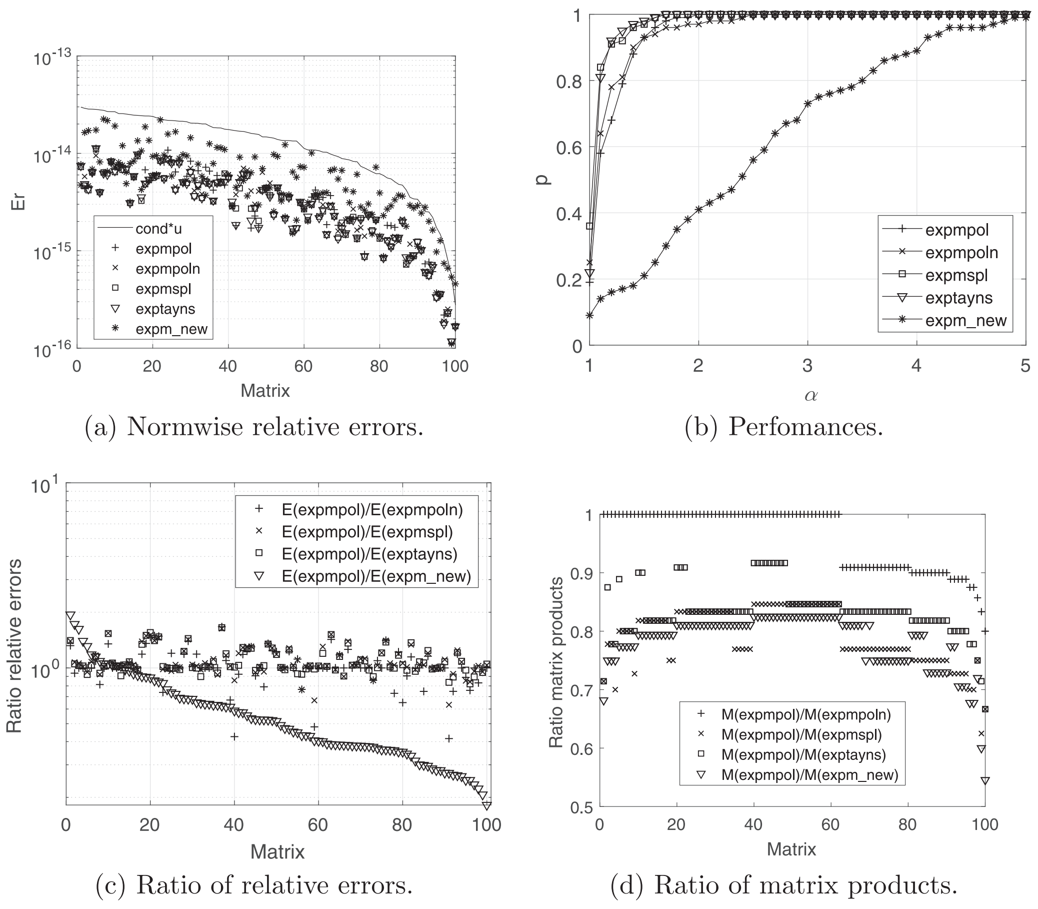
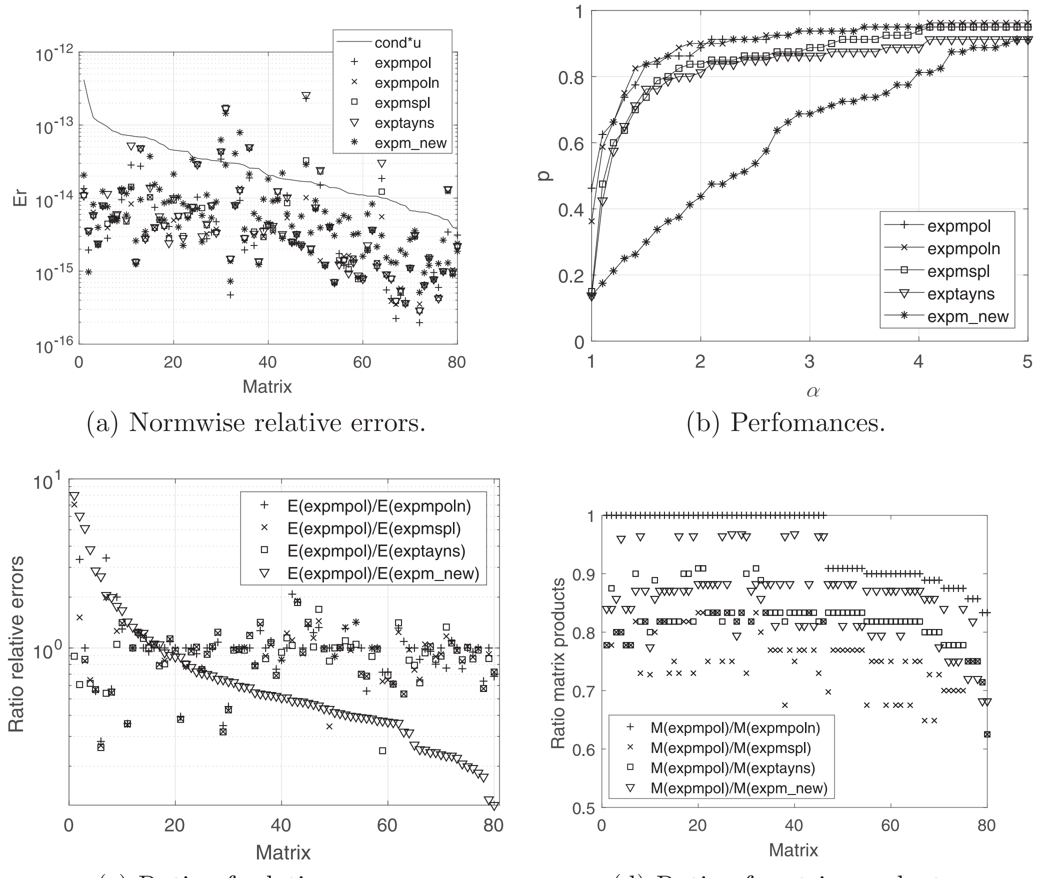


Fig. 1. Experimental results with Test 1. All the Taylor functions use maximum order $m_M = 30$.

- `expmpol_orig`: Direct MATLAB implementation of `expmpol` based on [6, Algorithm 2], i.e. evaluating the Taylor approximations with the methods from Section 3, changing the optimal orders m_k for m_k^* from (42) in [6, Algorithm 2] and using norm estimations.
- `expmpol`: MATLAB implementation using `ms_selectNormEst` from Section 5 to select m and s (argument `NormEst` = 1 in function `expmpol`), which reduces the number of norm estimations with respect to `expmpol_orig`.
- `expmpoln`: MATLAB implementation using `ms_selectNoNormEst` from Section 5 to select m and s (Algorithm 2, with no norm estimations, argument `NormEst` = 0 in function `expmpol`).

The experimental results have been carried out on an Intel i7-6700HQ @2.60 GHz, 32 GB RAM, using MATLAB R2017a. The accuracy was tested by computing the relative error $\mathrm{E} = \|e^A - \tilde{X}\|_1 / \left\|e^A\right\|_1$, where $\tilde{X}$ is the computed approximation and the cost is given in terms of matrix products. We used the following sets of matrices for testing:

1. One hundred diagonalizable matrices of size 128. These matrices have the form V^TDV, where D is a diagonal matrix whose diagonal elements are random values between $-k$ and k with different integer values of k, and V is an orthogonal matrix obtained as $V = H/16$, where H is the Hadamard matrix.
2. Eighty matrices with multiple eigenvalues of size 128. These matrices have the form V^TDV, where D is a block diagonal matrix whose diagonal blocks are Jordan blocks with random dimension and random eigenvalues between -50 and 50, and V is an orthogonal matrix obtained as $V = H/16$, where H is the Hadamard matrix.
3. Matrices 128×128, from the function `matrix` from the Matrix Computation Toolbox [17], and matrices from the MATLAB Eigtool package available at http://www.cs.ox.ac.uk/pseudospectra/eigtool/. These matrices appear in the state of the art in the exponential matrix computation [14,15]. Matrices whose exponential cannot be represented in double precision arithmetic due to overflow were excluded from all the test matrices. One matrix from the Eigtool package where MATLAB gave the warning “Matrix is singular to working precision” with function `expm_new` was also excluded. Note that in Taylor methods there are no matrix inversions. Finally, the test consisted of 31 matrices from the Matrix Computation Toolbox and 7 matrices from the Eigtool package.



(a) Normwise relative errors. (b) Perfomances.

(c) Ratio of relative errors. (d) Ratio of matrix products.

Fig. 2. Experimental results with Test 2. All the Taylor functions use maximum order $m_M = 30$.

The “exact” value of the matrix exponential for matrix sets 1 and 2 was computed by using transformations $e^A = V^T e^D V$, where $V^T e^D V$ was computed using vpa function from MATLAB’s Symbolic Math Toolbox with 256 decimal digit precision. For matrix set 3, we used a quadruple precision Taylor algorithm in Fortran with different orders and scaling parameters for each matrix to check the result correctness.

Table 6 shows the total cost in terms of matrix products for function expmpol, and the cost comparison in % between this function and expmpoln, expmspl, exptayns and expm_new for Tests 1–3, taking $m_M = 30$ in all the Taylor based functions. Table 7 shows the same comparison for the execution times.

Table 8 shows the total cost in terms of matrix products for function expmpol with kmax=6 ($m_M = 24$), and the cost comparison in % between this function and expmpoln with $m_M = 24$, expmspl ($m_M = 30$), exptayns ($m_M = 25$) and expm_new for Tests 1–3. Table 9 shows the same comparison for the execution times. For the execution times in MATLAB to be accurate the experiments were repeated 100 times and the mean values were used. The standard deviation of the values in all cases was less than or equal to 6.56%.

Figs. 1–3 show the results obtained in the three case studies. Figs. 1a, 2 a and 3 a show the graphics of the normwise relative errors of functions expmpol, expmpoln, expmspl, exptayns and expm_new, with all the Taylor functions using $m_M = 30$. These graphics show the numerical stability of the four functions, the relative errors of all implementations, and a solid line that represents the unit roundoff multiplied by the relative condition number of the exponential function at X [1, p. 55], respectively. The relative condition number was computed using the MATLAB function funm_condest1 from the Matrix Function Toolbox [1, Appendix D]. For a method to perform in a backward and forward stable manner, its error should lie not far above this line on the graph [15, p. 1188]. Figs. 1b, 2 b and 3 b show the performances of the four functions [16]. The α value varies between 1 and 5 with a step size equal to 0.1. Value p is the probability that the considered algorithm has a relative error lower than or equal to α-times the smallest error over all the methods. Figs. 1c, 2 c and 3 c show the ratios of errors of expmpol with respect to expmpoln, expmspl, exptayns and expm_new, ordering the matrices according to the ratio of relative errors $E_{\texttt{expmpol}}/E_{\texttt{expm\_new}}$. Figs. 1d, 2 d and 3 d show the ratios of the matrix products of expmpol with respect to expmspl, exptayns and expm_new.

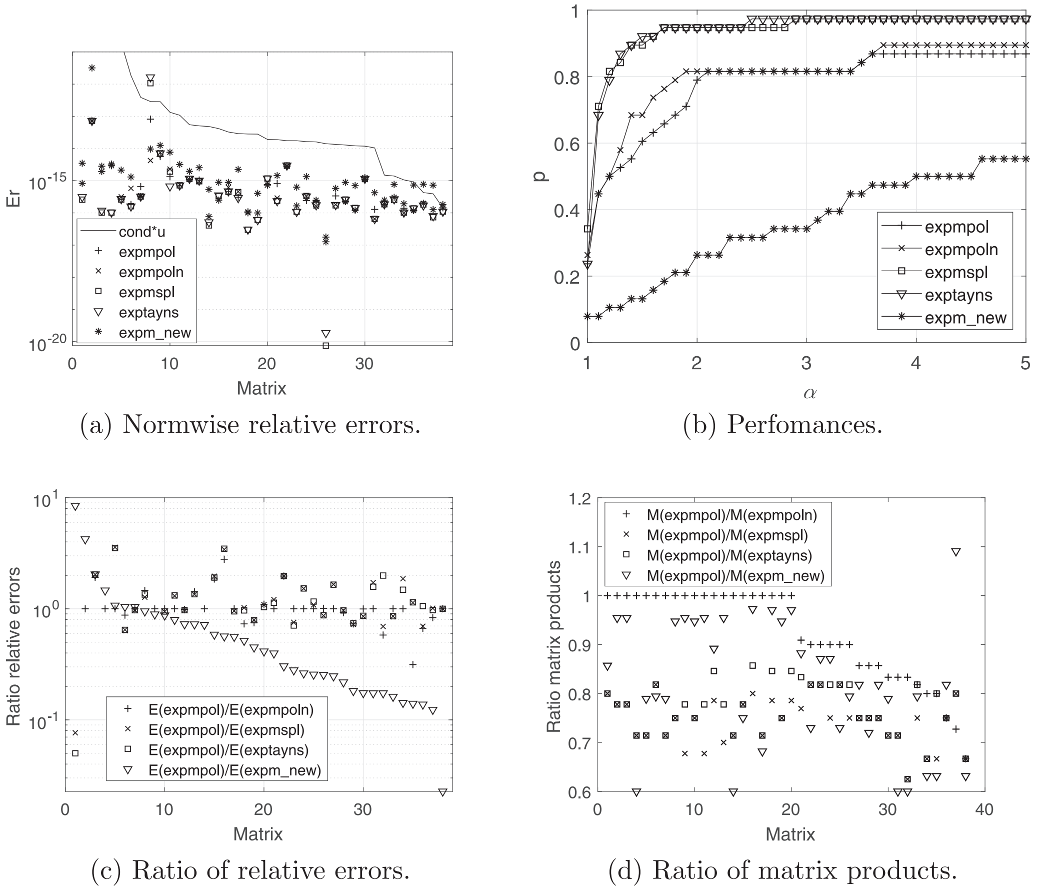


Fig. 3. Experimental results with Test 3. All the Taylor functions use maximum order $m_M = 30$.

Fig. 4 shows the results obtained in the case study 3 when kmax = 6 ($m = 24$) is considered for `expmpol` and `expmpoln`, and kmax=8 for `exptayns` giving a similar maximum order ($m_M = 25$). As mentioned above, `expmspl` has a fixed maximum order $m_M = 30$.

According to the results shown in the above tables and figures we can outline the following conclusions:

- All the functions performed in a numerically stable way in the three case studies.
- The implementations based on Taylor series are more accurate in general than the implementation based on Padé approximants. `expmpol` and `expmpoln` were the most accurate functions in Test 2, behaving similarly. In Test 1 and 3 the most accurate functions were `expmspl` and `expmtayns`, behaving also similarly.
- The execution processing times for Tests 1–3 (matrices 128×128) and function `expm_new` were between 204.12% and 244.65% of the `expmpol` times, and higher than all the Taylor functions.
- `expmpol` with $m_M = 24$ gives the lowest cost in terms of matrix products from all the functions, being the cost of `expm_new` between 123.51% and 135.89% of the cost of `expmpol`, and the cost of the Paterson–Stockmeyer based functions `exptayns` and `expmspl` between 127.62% and 140.18% of the cost of `expmpol`.
- The cost in terms of matrix products of `expmpoln` (without estimation) is between a 104.01% and a 109.27% of the cost of `expmpol` (with estimation), but the execution times are between 51.08% and 65.68% of the execution times of `expmpol`. Therefore, the cost of the estimation algorithm is not negligible for matrices sized 128×128, and it is important to reduce the number of estimations.
- The cost of `expmpol_orig` in terms of matrix products is equal to the cost of `expmpol`, but the execution times are between 122.08% and 154.13% of the execution times of `expmpol`. Therefore, the reduction of the estimations in `expmpol` is noticeable with respect to the original algorithm `expmpol_orig`, with no increase of the number of matrix products.
- in Test 3 with `expmpol` and `expmpol` with $m_M = 24$, `exptayns` with $m_M = 25$ and `expmspl` with $m_M = 30$, `expmspl` was the most accurate function at a higher cost, and the second function was `exptayns` with $m_M = 25$.
- According to Tables 6–9, in `expmpol` maximum order $m_M = 24$ is recommended for maximum efficiency. Similarly to [4] and [3], we checked that using $m_M = 30$ provides a higher accuracy at a slightly higher cost.

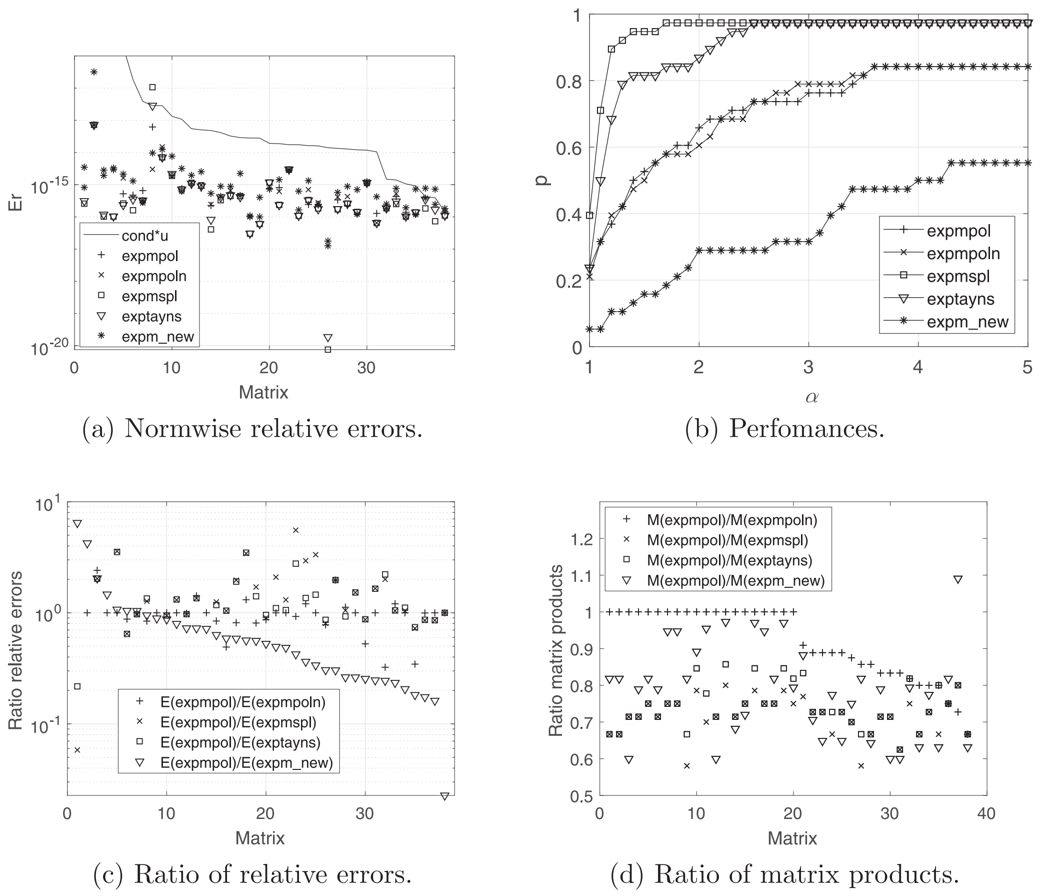


Fig. 4. Experimental results with Test 3: `expmpol` and `expmpol` with $m_M = 24$, `exptayns` with $m_M = 25$ and `expmspl` with $m_M = 30$.

7. Conclusions

In this paper we have given two Taylor algorithms for the computation of the matrix exponential. They are based on the matrix polynomial evaluation methods from [9] and on an improved version of the Taylor scaling algorithm from [4], simplified in [6]. These algorithms achieve maximum efficiency using maximum order $m_M = 24$, and maximum accuracy with $m_M = 30$.

In the last years Taylor algorithms have shown to be significantly more accurate than Padé algorithms, being also more efficient in some cases. With the matrix polynomial evaluation methods from [9] they are now considerably more efficient, and the proposed Taylor algorithms were superior in tests to the state-of-the-art Padé algorithm from [14] in both accuracy and efficiency.

Taylor methods based on the Paterson–Stockmeyer matrix polynomial evaluation method seem to be more accurate in certain cases than the proposed algorithms, based on the methods from [9]. Future work is addressed to:

- Increasing the accuracy of the methods based on [9].
- Increasing the efficiency searching Taylor approximations based on evaluation formulas (62)–(65) from [9] of the type $y_{kj}(A)$ with $k \geq 2$ and $j \geq 2$ with higher orders of approximation than the ones given in this paper for the same cost.

References

[1] N.J. Higham, Functions of matrices: Theory and computation, in: Society for Industrial and Applied Mathematics, Philadelphia, PA, USA, 2008.
[2] C.B. Moler, C.V. Loan, Nineteen dubious ways to compute the exponential of a matrix, twenty-five years later, SIAM Rev. 45 (2003) 3–49.
[3] J. Sastre, J. Ibáñez, E. Defez, P.A. Ruiz, Efficient scaling-squaring Taylor method for computing matrix exponential, SIAM J. Sci. Comput. 37 (1) (2015). A439–A455
[4] J. Sastre, J. Ibáñez, E. Defez, P. Ruiz, Accurate matrix exponential computation to solve coupled differential models in engineering, Math. Comput. Model. 54 (2011) 1835–1840.
[5] J. Sastre, J. Ibáñez, E. Defez, P. Ruiz, Accurate and efficient matrix exponential computation, Int. J. Comput. Math. 91 (1) (2014) 97–112.
[6] P. Ruiz, J. Sastre, J. Ibáñez, E. Defez, High performance computing of the matrix exponential, J. Comput. Appl. Math. 291 (2016) 370–379.
[7] J. Sastre, J. Ibáñez, P. Alonso, J. Peinado, E. Defez, Two algorithms for computing the matrix cosine function, Appl. Math. Comput. 312 (2017) 66–77.

[8] E. Defez, J. Ibáñez, J. Sastre, J. Peinado, P. Alonso, A new efficient and accurate spline algorithm for the matrix exponential computation, J. Comput. Appl. Math. 373 (2018) 354–365.
[9] J. Sastre, Efficient evaluation of matrix polynomials, Linear Algebra Appl. 539 (2018) 229–250.
[10] S. Blackford, J. Dongarra, Installation Guide for LAPACK, LAPACK, Working Note 41, Department of Computer Science University of Tennessee, 1999.
[11] M.S. Paterson, L.J. Stockmeyer, On the number of nonscalar multiplications necessary to evaluate polynomials, SIAM J. Comput. 2 (1) (1973) 60–66.
[12] J. Sastre, On the polynomial approximation of matrix functions, early unpublished version of [9] submitted to AMC, Feb. 2016, available at http://personales.upv.es/~jorsasma/AMC-S-16-00951.pdf.
[13] N.J. Higham, F. Tisseur, Fortran codes for estimating the one-norm of a real or complex matrix, with applications to condition estimation, ACM Trans. Math. Softw 14 (4) (2000) 381–396.
[14] A.H. Al-Mohy, N.J. Higham, A new scaling and squaring algorithm for the matrix exponential, SIAM J. Matrix Anal. Appl. 31 (3) (2009) 970–989.
[15] N.J. Higham, The scaling and squaring method for the matrix exponential revisited, SIAM J. Matrix Anal. Appl. 26 (4) (2005) 1179–1193.
[16] E.D. Dolan, J.J. Moré, Benchmarking optimization software with performance profiles, Math. Program. 91 (2002) 201–213.
[17] N.J. Higham, The matrix computation toolbox, http://www.ma.man.ac.uk/~higham/mctoolbox.