Sastre, J.; Ibáñez González, JJ.; Ruíz Martínez, PA.; Defez Candel, E. (2014). Accurate and efficient matrix exponential computation. International Journal of Computer Mathematics. 91(1):97-112. doi:10.1080/00207160.2013.791392.

Accurate and efficient matrix exponential computation[†]

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

Universitat Politècnica de València (UPV), Spain

[a]*Instituto de Telecomunicaciones y Aplicaciones Multimedia*; [b]*Instituto de Instrumentación para Imagen Molecular*; [b]*Instituto de Matemática Multidisciplinar*

(*released November 2012*)

This work gives a new formula for the forward relative error of matrix exponential Taylor approximation and proposes new bounds for it depending on the matrix size and the Taylor approximation order, providing a new efficient scaling and squaring Taylor algorithm for the matrix exponential. A Matlab version of the new algorithm is provided and compared with Padé state-of-the-art algorithms obtaining higher accuracy in the majority of tests at similar or even lower cost.

Keywords: Matrix exponential; scaling and squaring; Taylor series; error analysis.

1. Introduction

Matrix exponential plays a fundamental role in linear systems arising in many areas of science, and a large number of methods for its computation have been proposed [1, 2]. This work improves the scaling and squaring algorithm presented in [3] providing a competitive scaling and squaring algorithm for matrix exponential computation. The new algorithm employs an improved version of Theorem 1 from [3] to bound the norm of matrix power series. A new formula for the forward relative error of Taylor approximation in exact arithmetic and new sharp bounds for forward and backward relative errors are given. Moreover, taking into account that the roundoff error in the computation of Taylor matrix polynomial tends to increase with the matrix size and the approximation order, we propose increasing the allowed bounds for the error in exact arithmetic with both parameters. A Matlab version of the new algorithm is given. Numerical tests showed that it provided higher accuracy than Padé algorithms from [5, 6] at similar or even lower cost.

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, and $||\cdot||_\infty$ and $||\cdot||_1$ denote the ∞-norm and the 1-norm, respectively. Both norms are simple to compute, so they have been very used in the matrix function computation literature; particularly, the 1-norm is used in recent studies on matrix exponential computation [5, 6], and [4] provides an algorithm for its estimation which will be used in this paper.

This paper is organized as follows. Section 2 presents the scaling and squaring error analysis and the improved algorithm. Section 3 deals with numerical tests

[†]This work has been supported by the Programa de Apoyo a la Investigación y el Desarrollo of the *Universitat Politècnica de València* grant PAID-06-11-2020.

*Corresponding author. Email: jorsasma@iteam.upv.es

and gives some conclusions. The following theorem will be used to bound the norm of matrix power series, see [3, Th. 1].

THEOREM 1.1 *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*

$$\alpha_p = \max\{a_k^{\frac{1}{k}} : k = p, l, l+1, l+2, \ldots, p_0 - 1, p_0 + 1, p_0 + 2, \ldots, l + p - 1\}, \quad (1)$$

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

Proof . Since p_0 is a multiple of p, then $p_0/p \in \mathbb{N}$ and $||A^{p_0}|| = ||A^{pp_0/p}|| \leq ||A^p||^{p_0/p}$. Hence, it follows that

$$\begin{aligned}
&||h_l(A)|| \leq \sum_{k\geq l} |b_k| ||A^k|| \leq \sum_{j\geq 0} \sum_{i=l}^{l+p-1} |b_{i+jp}| ||A^p||^j ||A^i|| \\
&\leq \sum_{j\geq 0} \left[\sum_{i=l}^{p_0-1} |b_{i+jp}| ||A^p||^j ||A^i|| + |b_{p_0+jp}| ||A^p||^{j+p_0/p} + \sum_{i=p_0+1}^{l+p-1} |b_{i+jp}| ||A^p||^j ||A^i|| \right] \\
&\leq \sum_{j\geq 0} \sum_{i=l}^{l+p-1} |b_{i+jp}| \alpha_p^{pj+i} = \sum_{k\geq l} |b_k| \alpha_p^k = \tilde{h}_l(\alpha_p). \quad (2)
\end{aligned}$$

■

Theorem 1.1 unifies the two cases in which Theorem 1 from [3, p. 1835] is divided, and avoids needing a bound for $||A^{p_0}||$ to obtain α_p, see (1).

2. Taylor Algorithm

Taylor approximation of order m of exponential of matrix $A \in \mathbb{C}^{n\times n}$ can be expressed as the matrix polynomial $T_m(A) = \sum_{k=0}^{m} A^k/k!$. The scaling and squaring algorithms in Taylor approximations are based on the approximation $e^A = \left(e^{2^{-s}A}\right)^{2^s} \approx (T_m(2^{-s}A))^{2^s}$ [1], where the nonnegative integers m and s are chosen with the aim of achieve full machine accuracy at minimal cost. Similarly, this method is applied in Padé approximation.

In [2, p. 241], the author states that Padé approximations are preferred to Taylor series approximations in the context of scaling and squaring methods because they provide a given accuracy with lower computational cost. However, in [3] the authors presented a scaling and squaring Taylor algorithm based on an improved mixed backward and forward error analysis, which was more accurate than existing state-of-the-art Padé algorithms [5, 6] in the majority of test matrices with a lower or similar cost. Moreover, modifications to Padé algorithm had to be carried out in [6, p. 983] to improve the denominator conditioning, whereas Taylor algorithms have no denominator.

In [2, p. 247-248], an analysis about rounding errors and numerical stability of the scaling and squaring methods are performed. The author states that the overall effect of rounding errors in the computation by repeated squaring may be large

relative to the computed exponential. This may or may not indicate instability of the algorithm, depending on the conditioning of the e^A problem at the matrix A. If A is normal, then the scaling and squaring method is guaranteed to be forward stable; hence, the square phase is innocuous and the error in the computed exponential is consistent with the conditioning of the problem. Another case where the scaling and squaring method is forward stable corresponds to matrices with nonnegative nondiagonal entries as shown in[12].

The scaling and squaring method has also a weakness when applied to block triangular matrices [13, 14]. The exponential of a block 2×2 block triangular matrix A can be computed as

$$\exp\left(\begin{bmatrix} A_{11} & A_{12} \\ 0 & A_{22} \end{bmatrix}\right) = \begin{bmatrix} e^{A_{11}} & \int_0^1 e^{A_{11}(1-s)} A_{12} e^{A_{22}s} ds \\ 0 & e^{A_{22}} \end{bmatrix},$$

where matrices A_{11} and A_{22} are square matrices. However, since matrix A_{12} appears only in the (1,2) block of e^A, then e^A depends linearly of A_{12}, and the accuracy of computing e^A should be unaffected by $\|A_{12}\|$ and should depend only on $\|A_{11}\|$ and $\|A_{22}\|$. Since s depends on $\|A\|$, when $\|A_{12}\| >> \max\{\|A_{11}\|, \|A_{22}\|\}$ the diagonal blocks A_{11} and A_{22} are overscaled with regard to the computation of $e^{A_{11}}$ and $e^{A_{22}}$, and this can affect the accuracy of computing e^A.

In [14] L. Diecci and A. Papini obtain improved error bounds for Padé approximations to e^A when A is block triangular. As a result, improved scaling strategies ensue which avoid some common overscaling difficulties. Later, [6] presents an algorithm that reduces the overscaling problem choosing parameter s, based on the norms of low powers of matrix A instead of in $||A||$, and computes the diagonal elements in the squaring phase as exponentials instead of from powers of the diagonal Padé approximation for the case of triangular matrices. In [3], estimations of norms of higher powers of matrix A (greater than or equal to m+1) are used to obtain the scaling parameter s, and similar ideas to those in [6] can be used in the case of Taylor approximations to compute the diagonal elements for triangular matrices.

Algorithm *1* presents a general scaling and squaring Taylor algorithm for computing the matrix exponential, where the maximum allowed value of m is denoted by m_M.

Algorithm *1* Given a matrix $A \in \mathbb{C}^{n\times n}$ and a maximum order m_M, this algorithm computes $C = e^A$ by a Taylor approximation of order $m \leqslant m_M$.

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

The preprocessing and postprocessing steps are based on applying transformations to reduce the norm of matrix A, see [2], and will not be studied in this paper.

In Step 2, the scaling phase, the optimal order of Taylor approximation $m_k \leqslant m_M$ and the scaling parameter s are chosen.

Table 1. Values of q_k depending on the selection of m_M.

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

For the evaluation of $T_m\left(2^{-s}A\right)$ in Step 3 we use the modified Horner and Paterson–Stockmeyer's method proposed in [3, p. 1836-1837]. From [3, p. 6454-6455] matrix polynomial $T_m(2^sA)$ can be computed optimally for m in the set $\mathbb{M} = \{1, 2, 4, 6, 9, 12, 16, 20, 25, 30, \ldots\}$, where we denote the elements of $\mathbb{M}$ as m_0, m_1, $m_2, \ldots$, respectively, by using Paterson-Stockmeyer's method [10], see [2, p. 72–74] for a complete description. First, matrix powers A^2, $A^3, \cdots, A^q$ are computed, where $q = \lceil\sqrt{m_k}\rceil$ or $\lfloor\sqrt{m_k}\rfloor$, both values dividing m_k and giving the same cost [2, p. 74].Then, the truncated Taylor series is computed using (10) of [3, p. 1837], which includes the matrix scaling in Taylor series coefficients and saves some divisions of matrix A by scalar in the next way

$$
\begin{aligned}
T_m\left(2^{-s}A\right) = \Bigg\{\Bigg\{\cdots\Bigg\{\frac{A_q}{2^s m} + A_{q-1}\Bigg\}/\left[2^s(m-1)\right] + A_{q-2}\Bigg\}/\left[2^s(m-2)\right] + \cdots A_2\Bigg\}/\left[2^s(m-q+2)\right] + A \\
+ 2^s(m-q+1)I\Bigg\}\frac{A_q}{2^{2s}(m-q+1)(m-q)} + A_{q-1}\Bigg\}/\left[2^s(m-q-1)\right] + A_{q-2}\Bigg\} \\
/\left[2^s(m-q-2)\right] + \cdots + A_2\Bigg\}/\left[2^s(m-2q+2)\right] + A + 2^s(m-2q+1)I\Bigg\} \\
\times \frac{A_q}{2^{2s}(m-2q+1)(m-2q)} + \cdots + A_2\Bigg\}/\left[2^s(q+2)\right] + A + 2^s(q+1)I\Bigg\} \\
\times \frac{A_q}{2^{2s}(q+1)q} + A_{q-1}\Bigg\}/\left[2^s(q-1)\right] + \cdots + A_2\Bigg\}/\left[2^s 2\right] + A\Bigg\}/2^s + I. \qquad (3)
\end{aligned}
$$

Analogously to Sastre *et al.* [3], in the proposed scaling algorithm it will be necessary that the same powers of A are used for the two last orders m_{M-1} and m_M, i.e. $A^i, i = 2, 3, \ldots, q$. For each value of m_M Table 1 shows the selected optimal values of q for orders m_k, $k = 0, 1, 2, \ldots, M$, denoted by q_k. For example, if $m_M = 20$ and $m_4 = 9$ is the optimal order obtained in the scaling phase, then $q_4 = 3$.

Taking into account Table 4.1 from [2, p. 74], the total cost of evaluating $T_{m_k}(2^{-s}A)$ in terms of matrix products for $k = 0, 1, \ldots$, denoted by Π_{m_k}, is $\Pi_{m_k} = k$. Finally, after the evaluation of $T_m\left(2^{-s}A\right)$, s repeated squarings are applied in Steps 4-6. Thus, the computational cost of Algorithm *1* in terms of matrix products is $\mathrm{Cost}(m_k, s) = k + s$.

2.1 *Roundoff error analysis*

The main contributions of this paper are concerned with the selection of m and s in the scaling phase, where the roundoff error in the computation of (3) will play an important role. The roundoff error can be studied by using a componentwise analysis [11, pp. 18-19]. If we denote $|A| = (|a_{ij}|)_{n\times n}$, then

(1) $|fl(A+B) - (A+B)| < u\,|A+B|\,,\ A, B \in \mathbb{C}^{n\times n}$,
(2) $|fl(AB) - AB| < nu\,|A|\,|B|$,
(3) $\left|fl\left(\sum_{k=0}^{m} p_k A^k\right) - \sum_{k=0}^{m} p_k A^k\right| \dot{\leq}\ m(n+1)\sum_{k=0}^{m} |p_k|\,|A|^k$,

where $\dot{\leq}$ denotes the inequality avoiding the terms of order greater than or equal to u^2, where $u = 2^{-53}$ is the unit roundoff in IEEE double precision arithmetic. Taking into account these properties, it is straightforward to prove that the roundoff error for computing $T_m\left(2^{-s}A\right)$ by using (3) verifies $|fl\left(T_m(2^{-s}A)\right) - T_m(2^{-s}A)| \dot{\leqslant} \varphi(m,n)uT_m(2^{-s}\,|A|)$, where u is the unit roundoff, $m \in \mathbb{M}$, and for large n, asymptotically it follows that

$$\varphi(m,n) = mn, m \geq 2. \tag{4}$$

Hence if we use the 1–norm, then

$$\frac{\|fl\left(T_m(2^{-s}A)\right) - T_m(2^{-s}A)\|_1}{T_m(2^{-s}\|A\|_1)} \leqslant \varphi(m,n)u. \tag{5}$$

This is a worst-case bound. If we denote the actual roundoff error in computing (3) from (5) as $\phi(m,n)u$, and noting that minimum roundoff errors of value u are expected, by (4), for large n in the majority of cases we can assume that

$$1 \leq \phi(m,n) \leq mn. \tag{6}$$

From [8, p. 52] a well-known rule of thumb to obtain realistic error estimates in (5) is that if the bound is $f(n)u$ then the error will be typically of order $\sqrt{f(n)}u$, and this rule of thumb can be supported by assuming that the rounding errors are independent random variables and applying the central limit theorem. By (4) and (5) the application of this rule gives

$$\phi(m,n) \approx \sqrt{mn}. \tag{7}$$

Rounding errors do not necessarily behave like independent random variables [8, p. 52] and there will be cases where the application of this rule will be pessimistic and others where it will be optimistic. However, in numerical results we will see that the use of (7) allows to reduce the cost of Algorithm *1* with no important effects in accuracy in the majority of cases.

2.2 *Analysis of truncation error*

Following [3, p. 1836], if we denote the remainder of the truncated exponential Taylor series of $A \in \mathbb{C}^{n\times n}$ as

$$R_m(A) = \sum_{k\geq m+1} A^k/k!, \tag{8}$$

for a scaled matrix $2^{-s}A$, $s \in \mathbb{N} \cup \{0\}$, we can write

$$\left(T_m(2^{-s}A)\right)^{2^s} = e^A\left(I + g_{m+1}(2^{-s}A)\right)^{2^s} = e^{A+2^s h_{m+1}(2^{-s}A)}, \tag{9}$$

where

$$g_{m+1}(2^{-s}A) = -e^{-2^{-s}A}R_m(2^{-s}A), \tag{10}$$

$$h_{m+1}\left(2^{-s}A\right) = \log\left(I+g_{m+1}(2^{-s}A)\right), \tag{11}$$

where log denotes the principal logarithm, $h_{m+1}(X)$ is defined in the set

$$\Omega_m = \left\{ X \in \mathbb{C}^{n\times n} : \rho\left(e^{-X}T_m(X) - I\right) < 1 \right\}, \tag{12}$$

and both $g_{m+1}(2^{-s}A)$ and $h_{m+1}\left(2^{-s}A\right)$ are holomorphic functions of A in Ω_m and then commute with A. Using scalar Taylor series in (10) and (11) one gets

$$g_{m+1}(x) = \sum_{k\geq m+1} b_k^{(m)} x^k = \left(e^{h_{m+1}(x)} - 1\right) = \sum_{k\geq 1} \left(h_{m+1}\left(x\right)\right)^k / k!, \tag{13}$$

$$h_{m+1}(x) = \sum_{k\geq 1} \frac{(-1)^{k+1}(g_{m+1}(x))^k}{k} = \sum_{k\geq m+1} c_k^{(m)} x^k, \tag{14}$$

where $b_k^{(m)}$ and $c_k^{(m)}$ depend on order m. From (13) and (14) it follows that

$$b_k^{(m)} = c_k^{(m)}, \qquad k = m+1, m+2, \ldots, 2m+1, \tag{15}$$

and if $\|h_{m+1}(2^{-s}A)\| \ll 1$ then

$$g_{m+1}(2^{-s}A) = h_{m+1}(2^{-s}A) + (h_{m+1}(2^{-s}A))^2/2 + \cdots \approx h_{m+1}(2^{-s}A), \tag{16}$$

and, similarly, if $\|g_{m+1}(2^{-s}A)\| \ll 1$

$$h_{m+1}(2^{-s}A) = g_{m+1}(2^{-s}A) + (g_{m+1}(2^{-s}A))^2/2 + \cdots \approx g_{m+1}(2^{-s}A). \tag{17}$$

Using (8) and the exponential Taylor series in (10) it follows that

$$\begin{aligned} g_{m+1}(x) = \frac{-x^{m+1}}{(m+1)!}\Bigg\{ &(-1)^0\frac{1}{0!} + (-1)^1\left[\frac{1}{1!} - \frac{1}{0!(m+2)}\right]x + (-1)^2\left[\frac{1}{2!}\right. \\ &\left. - \frac{1}{1!(m+2)} + \frac{1}{0!(m+2)(m+3)}\right]x^2 + \cdots + (-1)^k a_k^{(m)} x^k + \cdots, \end{aligned} \tag{18}$$

where by induction the general term of coefficient $a_k^{(m)}$ is

$$\begin{aligned} a_k^{(m)} = \frac{1}{k!} &- \frac{1}{(k-1)!(m+2)} + \frac{1}{(k-2)!(m+2)(m+3)} - \cdots \\ &+ \frac{(-1)^{k-3}}{3!(m+2)(m+3)\cdots(m+k-2)} + \frac{(-1)^{k-2}}{2!(m+2)(m+3)\cdots(m+k-1)} \\ &+ \frac{(-1)^{k-1}}{1!(m+2)(m+3)\cdots(m+k)} + \frac{(-1)^k}{0!(m+2)(m+3)\cdots(m+k+1)}. \end{aligned} \tag{19}$$

Summing from the last term to the initial terms of $a_k^{(m)}$ it is easy to show that the sum of the last i terms of $a_k^{(m)}$ becomes

$$\frac{(-1)^{k-(i-1)}}{(i-1)!(m+2)(m+3)\cdots(m+k-(i-1))(m+k+1)} \tag{20}$$

and then it follows that

$$a_k^{(m)} = \frac{m+1}{k!\,(m+1+k)}. \tag{21}$$

Hence, using (18) it follows that

$$g_{m+1}(x) = -\sum_{k\geqslant 0} \frac{(-1)^k x^{m+1+k}}{k!m!\,(m+1+k)}, \tag{22}$$

and then

$$\frac{b_{m+k+1}^{(m)}}{b_{m+k}^{(m)}} = \frac{-1}{k\left(1+\frac{1}{m+k}\right)}. \tag{23}$$

This expression confirms the observation from [3, p. 1838], where for the orders m_k that were proposed in the algorithm presented therein and the first 1000 series terms, using Matlab's Symbolic Math Toolbox we checked experimentally that the following expression holds

$$\frac{1}{k+1} < \left|\frac{b_{m+k+1}^{(m)}}{b_{m+k}^{(m)}}\right| < \frac{1}{k}, \quad k \geqslant 1. \tag{24}$$

Once obtained the closed form (22) for $g_{m+1}(x)$, $h_{m+1}(x)$ can be obtained using (14), and now we set the basis for the scaling algorithm. If we choose s so that $2^{-s}A \in \Omega_m$, see (12), then from (9) one gets that

$$\Delta A = 2^s h_{m+1}\left(2^{-s}A\right), \tag{25}$$

represents the backward absolute error in exact arithmetic from the approximation of e^A by Taylor series truncation with the scaling and squaring technique. If the minimum value of s is chosen so that

$$\left\|h_{m+1}\left(2^{-s}A\right)\right\| \leq \left\|2^{-s}A\right\| u, \tag{26}$$

or

$$\left\|g_{m+1}\left(2^{-s}A\right)\right\| \leq \phi(m,n)u. \tag{27}$$

where $\phi(m,n)u$ is the roundoff error in computing (3), see Section 2.1.

- If the minimum value of s is given by (26), then from (25) it follows that $\Delta A \leq \|A\|\,u$ and using (9) it follows that

$$\left(T_m\left(2^{-s}A\right)\right)^{2^s} = e^{A+\Delta A} \approx e^A. \tag{28}$$

- If the minimum value of s is given by (27), using (9), (10) and the exponential Taylor series it follows that

$$\left\|R_m\left(2^{-s}A\right)\right\| = \left\|e^{2^{-s}A} g_{m+1}\left(2^{-s}A\right)\right\| \leq \left\|e^{2^{-s}A}\right\| \phi(m,n)u, \tag{29}$$

Table 2. Maximal values Θ_m such that $\tilde{h}_{m+1}(\Theta_m) \leq \Theta_m u$, maximal values $\tilde{\Theta}_m$ such that $\tilde{g}_{m+1}\left(\tilde{\Theta}_m\right) \leq \phi(m,n)u$ for $\phi(m,n)=1$, and values $\vartheta_m = \max\left\{\Theta_m, \tilde{\Theta}_m\right\}$.

m	Θ_m	$\tilde{\Theta}_m$ for $\phi(m,n)=1$	ϑ_m
1	2.220446049250264e-16	1.490116111983279e-8	1.490116111983279e-8
2	2.580956802971767e-8	8.733457513635361e-6	8.733457513635361e-6
4	3.397168839976962e-4	1.678018844321752e-3	1.678018844321752e-3
6	9.065656407595101e-3	1.773082199654024e-2	1.773082199654024e-2
9	8.957760203223343e-2	1.137689245787824e-1	1.137689245787824e-1
12	2.996158913811581e-1	3.280542018037257e-1	3.280542018037257e-1
16	7.802874256626574e-1	7.912740176600240e-1	7.912740176600240e-1
20	1.438252596804337	1.415070447561532	1.438252596804337
25	2.428582524442827	2.353642766989427	2.428582524442827
30	3.539666348743690	3.411877172556771	3.539666348743690

and, taking into account that a relative roundoff error $\phi(m,n)u$ will be introduced in the numerical evaluation of the matrix polynomial $T_m\left(2^{-s}A\right)$, there is no point in increasing the scaling parameter s or the approximation order m to reduce further the norm of Taylor remainder $R_m\left(2^{-s}A\right)$.

Using (14) and (22), Matlab's Symbolic Math Toolbox, high precision arithmetic, 200 series terms and a zero finder we obtained the maximal values Θ_m of $\Theta = \|2^{-s}A\|$, shown in Table 2, such that, using notation of Theorem 1.1

$$||h_{m+1}\left(2^{-s}A\right)|| \leq \tilde{h}_{m+1}(\Theta) = \sum_{k\geq m+1} c_k^{(m)}\Theta^k \leq \Theta u. \tag{30}$$

In a similar way, for a given value of $\phi(n,m)$ the maximal values $\tilde{\Theta}_m$ such that

$$||g_{m+1}\left(2^{-s}A\right)|| \leq \tilde{g}_{m+1}(\Theta) = \sum_{k\geq m+1} b_k^{(m)}\Theta^k \leq \phi(m,n)u, \tag{31}$$

can be easily obtained. Table 2 shows $\tilde{\Theta}_m$ values for the more restrictive case from (6), i.e. when $\phi(m,n)=1$, which is the case used in the error analysis of [3, p. 1836]. Hence, if $||2^{-s}A|| \leq \vartheta_m$ where $\vartheta_m = \max\left\{\Theta_m, \tilde{\Theta}_m\right\}$ then (26) or (27) hold. For the values of m where $\Theta_m > 1$ in Table 2, i.e. $m = 20$, 25 and 30, it follows that $\vartheta_m = \Theta_m$. We recall bound (9) of [3, p. 1836], which holds for those values of m and will be needed in the scaling algorithm:

$$||h_{m+1}\left(2^{-s}A\right)|| \leq \tilde{h}_{m+1}\left(||2^{-s}A||\right) = \tilde{h}_{m+1}(\Theta) \leq \Theta u, \quad 0 \leq \Theta \leq \Theta_m. \tag{32}$$

2.3 *Scaling algorithm*

The new proposed scaling algorithm has some improvements with respect to that proposed in [3, p. 1837-1838] that we describe in this section. For all norms appearing in the scaling algorithm we will use the 1–norm, and from Algorithm *1* recall that m_M is the maximum allowed Taylor order. Using the same bounds and a similar process that we use in the proposed scaling algorithm described below, we will first verify if any of the lower Taylor optimal orders $m_k = 1, 2, 4, \ldots, m_{M-1}$ satisfy (26) or (27) without scaling, i.e. with $s = 0$. If not, the optimal scaling parameter $s \geq 0$ for order m_M is computed. This computation has two phases: Calculation of an initial value of the scaling parameter s_0, and the refinement of this value to test if it can be reduced. In this paper the main improvement with respect to the algorithm from [3] is applied to the refinement of the scaling parameter, where roundoff error and function $\phi(m,n)$ from (7) will play an important role. The calculation of the initial scaling parameter s_0 remains practically unchanged except for the use of

the new Theorem 1.1 instead of Theorem 1 from [3], and it is described for clarity in the following section.

2.3.1 Calculation of initial scaling s_0

First, the 1–norm estimate of $||A^{m_M+1}||$ is computed using the block 1–norm estimation algorithm of [4]. For a $n \times n$ matrix this algorithm carries out a 1-norm power iteration whose iterates are $n \times t$ matrices, where t is a parameter that has been taken to be 2, see [6, p. 983]. Hence, the estimation algorithm has $O(n^2)$ computational cost, negligible compared to matrix products, whose cost is $O(n^3)$.

Similarly to [3, p. 1837], the upper bounds a_k for $||A^k||$ needed to apply Theorem 1.1 in (30) and (31) are obtained using products of norms of matrix powers estimated for current and previous tested orders, i.e. $||A^{m_k+1}||,\ k = 0, 1, 2, \ldots, M$, and the powers of A computed for evaluation of $T_{m_M}(2^{-s}A)$, $A^i,\ i = 1, 2, \ldots, q$, as

$$\begin{aligned}\left\|A^k\right\| \le a_k = \min\Big\{ &\|A\|^{i_1} \left\|A^2\right\|^{i_2} \cdots \|A^q\|^{i_q} \left\|A^{m_1+1}\right\|^{i_{m_1+1}} \left\|A^{m_2+1}\right\|^{i_{m_2+1}} \cdots \\ &\times \left\|A^{m_M+1}\right\|^{i_{m_M+1}} : i_1 + 2i_2 + \cdots + qi_q + (m_1+1)i_{m_1+1} \\ &+ (m_2+1)i_{m_2+1} + \cdots + (m_M+1)i_{m_M+1} = k\Big\}, \end{aligned} \tag{33}$$

and a simple Matlab function was provided in [3] to obtain a_k, see nested function **powerbound** from **exptayns.m** available at `http://personales.upv.es/~jorsasma/Software/exptayns.m`.

Then, we seek for the minimum value of α_p from Theorem 1.1 with $l = m_M + 1$, denoted by $\alpha_{\min}$, obtaining successively α_p for $p = 2, 3, \ldots, q, m_1 + 1, m_2 + 1, \ldots, m_M + 1$, stopping the process for that value of p such that

$$a_p^{1/p} \le \max\{a_k^{1/k} : k = m+1, m+2, \ldots, p_0 - 1, p_0 + 1, p_0 + 2, \ldots, m+p\}, \tag{34}$$

where p_0 is the multiple of p with $m + 1 \le p_0 \le m + p$. In the following we show that if condition (34) holds, then for $p' > p$ it follows that $\alpha_{p'} \ge \alpha_p$:

By (33) one gets $a_{p_0} \leqslant a_p^{p_0/p}$, and then using (34) it follows that

$$a_{p_0}^{1/p_0} \le a_p^{1/p} \le \max\{a_k^{1/k} : k = m+1, m+2, \ldots, p_0 - 1, p_0 + 1, p_0 + 2, \ldots, m+p\}, \tag{35}$$

and then

$$\begin{aligned}\alpha_p &= \max\{a_k^{1/k} : k = m+1, m+2, \ldots, p_0 - 1, p_0 + 1, p_0 + 2, \ldots, m+p\} \\ &= \max\{a_k^{1/k} : k = m+1, m+2, \ldots, m+p\}. \end{aligned} \tag{36}$$

Hence, for $p' > p$, if

$$a_{p'}^{1/p'} \le \max\{a_k^{1/k} : k = m+1, m+2, \ldots, p_0' - 1, p_0' + 1, p_0' + 2, \ldots, m+p'\}, \tag{37}$$

where p_0' is a multiple of p', then in a similar way it follows that

$$\begin{aligned}\alpha_{p'} &= \max\{a_k^{1/k} : k = m+1, m+2, \ldots, m+p'\} \\ &\ge \max\{a_k^{1/k} : k = m+1, m+2, \ldots, m+p\} = \alpha_p. \end{aligned} \tag{38}$$

Otherwise, if (37) is not verified, as $a_{p'}^{1/p'} \geq a_{p'_0}^{1/p'_0}$ it follows that

$$\begin{aligned}\alpha_{p'} = a_{p'}^{1/p'} &> \max\{a_k^{1/k} : k = m+1, m+2, \ldots, m+p'\} \\ &\geq \max\{a_k^{1/k} : k = m+1, m+2, \ldots, m+p\} = \alpha_p, \end{aligned} \tag{39}$$

and then, by (38) and (39) $\alpha_{p'} \geq \alpha_p$.

Once obtained $\alpha_{\min}$, we take the appropriate initial minimum scaling parameter $s_0 \geq 0$ so that $2^{-s_0}\alpha_{\min} \leq \vartheta_{m_M}$, i.e.

$$s_0 = \begin{cases} 0, & \text{if } \alpha_{\min} \leq \vartheta_{m_M}, \\ \lceil \log_2(\alpha_{\min}/\vartheta_{m_M}) \rceil, & \text{if } \alpha_{\min} > \vartheta_{m_M}. \end{cases} \tag{40}$$

Then, if $\vartheta_{m_M} = \max\{\Theta_{m_M}, \tilde{\Theta}_{m_M}\} = \tilde{\Theta}_{m_M}$ using Theorem 1.1 and (31), taking for simplicity from now on $m = m_M$, it follows that

$$||g_{m+1}\left(2^{-s_0}A\right)|| \leq \tilde{g}_{m+1}\left(2^{-s_0}\alpha_{\min}\right) \leq \tilde{g}_{m+1}\left(\tilde{\Theta}_m\right) \leq u, \tag{41}$$

and (27) holds. Taking into account that $||A^k||^{1/k} \leq ||A||$, from (33) it follows that $a_k^{1/k} \leq (||A||^k)^{1/k} = ||A||$. Thus, $\alpha_{\min}$ from Theorem 1.1 satisfies $\alpha_{\min} \leq ||A||$. Hence, if $\vartheta_m = \Theta_m$ from Table 2 it follows that $m = 20, 25$ or 30, and using (32) one gets

$$||h_{m+1}\left(2^{-s_0}A\right)|| \leq \tilde{h}_{m+1}\left(2^{-s_0}\alpha_{\min}\right) \leq 2^{-s_0}\alpha_{\min}\, u \leq 2^{-s_0}||A||u, \tag{42}$$

and (26) holds.

After obtaining the initial value of the scaling parameter s_0, it will be refined as described in the following section.

2.3.2 Scaling refinement

In this section bounds for $\|g_{m+1}(2^sA)\|$ and $\|h_{m+1}(2^{-s}A)\|$ of the same type of (14) and (15) from [3, p. 1838] are used for the scaling parameter refinement. Both series from (14) and (22) will be truncated with the same number of terms as in Section 3 of [3, p. 1839], i.e. $q+2$ where q takes the values from Table 1. Taking into account (15) one gets that the first $m+1$ coefficients of $g_{m+1}(2^sA)$ and $h_{m+1}(2^{-s}A)$ are equal, and if q takes the values from Table 1 it follows that $m+1 \geq q+2$ for $m \geq 4$. Then, using (13)–(15) we take

$$\|g_{m+1}(2^sA)\| \approx \left\| \sum_{k=m+1}^{m+q+2} b_k^{(m)} (2^sA)^k \right\| = \left\| \sum_{k=m+1}^{m+q+2} c_k^{(m)} (2^sA)^k \right\| \approx \|h_{m+1}(2^sA)\|, \tag{43}$$

and the refinement will be considered for $m \geq 4$.

Hence, once the initial value s_0 of the scaling parameter is obtained, if $s_0 \geq 1$ we test if (26) or (27) hold with the reduced scaling parameter $s = s_0 - 1$, using the bounds for $\left\|A^k\right\| \leq a_k$ from (33) and testing if bound

$$\sum_{k=m+1}^{m+q+2} \left| c_k^{(m)} \right| a_k / 2^{sk} \leq \max\left\{\phi(m,n), \left\|2^{-s}A\right\|\right\} u, \tag{44}$$

holds taking $\phi(m,n) = \sqrt{nm}$ from (7). Note that we stop the series summation if the first term does not verify the bound. If the sum of one or more terms is lower

than the bound but the complete truncated series sum is not, we can estimate $||A^{m+2}||$ to improve the bound a_{m+2} and check if (44) holds then, see [3]. If (44) does not hold with $s = s_0 - 1$, then we check if next bound holds

$$\frac{\left\|A^{m+1}\right\|}{2^{s(m+1)}} \left\| \sum_{k=m+1}^{m+q+1} c_k^{(m)} (2^{-s}A)^{k-m-1} \right\| + \left| c_{m+q+2}^{(m)} \right| \frac{a_{m+q+2}}{2^{(m+q+2)s}} \leq \max\left\{\phi(m,n), \left\|2^{-s}A\right\|\right\} u, \tag{45}$$

where $\phi(m,n) = \sqrt{nm}$, matrix powers A^i, $i = 2, 3, \ldots, q$ from the computation of $T_m\left(2^{-s}A\right)$ by (3) are reused, and we can divide by the coefficient of A to save the product of matrix A by a scalar. In a similar way to [3, p. 1838], lower bounds for expression (45) to avoid its unnecessary evaluation can be easily obtained.

If any of both bounds (44) or (45) holds with $s_0 - 1$ then repeat the process with $s = s_0 - 2, s_0 - 3, \ldots$ Note that computational cost of evaluating (44) and (45) is $O(n^2)$, negligible when compared to matrix product costs, which are $O(n^3)$.

Note that from Table 2 for $m_k \geq 20$ it follows that $\vartheta_{m_k}/2 < \vartheta_{m_{k-1}}$. Thus, if the last value of s where (44) or (45) hold is $s \geq 1$ and $\vartheta_{m_M}/2 < \vartheta_{m_{M-1}}$ it is possible that order m_{M-1} also verifies (44) or (45) with the same value of scaling s, see [3]. Hence, if final resulting scaling is $s \geq 1$ bounds (44) and/or (45) should be tested with the same scaling parameter s and order m_{M-1}. Finally, the algorithm returns s and the minimum order m_M or m_{M-1} satisfying (44) or (45). It is possible to evaluate $T_m(2^{-s}A)$ with both orders with optimal number of matrix products at this point because we set in its evaluation that both last orders used the same matrix powers of A [3].

Summarizing, the complete scaling algorithm from Step 2 in Algorithm *1* consists of:

(1) Check if one of optimal orders $m = 4, 6, \ldots, m_{M-1}$, $m \in \mathbb{M}$ satisfies (44) or (45) with $s = 0$ using the 1–norm estimate of $||A^{m+1}||$ from [4] and reusing the matrix powers A^i, $i = 2, 3, \ldots, q$ needed to compute Taylor matrix polynomial for each tested value of m, returning $s = 0$ and the order m which satisfies (44) or (45) if it exists.
(2) If there is no value of $m \leq m_{M-1}$ that satisfies (44) or (45) with $s = 0$ then obtain an initial value s_0 of the scaling parameter with order m_M as described in Section 2.3.1.
(3) If $s_0 > 0$ refine the selection of s_0 testing if (44) or (45) hold with $s = s_0 - 1$, $s_0 - 2, \ldots$, returning the lowest value of s that verifies (44) or (45).
(4) If the final selection of s is not zero, test if (44) or (45) hold with s and m_{M-1}, returning m_{M-1} if (44) or (45) hold with it, or otherwise returning m_M.

Maximum order $m_M = 30$ is recommended as it obtained the highest accuracy results in [3]. For more details about the implementation we made available online a fully commented Matlab version of the complete Algorithm *1*, denoted by **exptaynsv2**, in `http://personales.upv.es/~jorsasma/Software/exptaynsv2.m`, where changes in the selection of order m and scaling parameter s are avoided when no cost reductions are achieved with respect to the original algorithm **exptayns** from [3].

2.4 *New bounds for* $\|g_{m+1}(2^s A)\|$ *and* $\|h_{m+1}(2^{-s}A)\|$

This section provides bounds for the complete series of $\|g_{m+1}(2^{-s}A)\|$ without truncation. These bounds are used to provide bounds for $\|h_{m+1}(2^{-s}A)\|$. From

(22), if $B \in \mathbb{C}^{n\times n}$ and $r \in \mathbb{N} \cup \{0\}$, then

$$||g_{m+1}(B)|| \leq \frac{1}{m!} \left\| \sum_{k=0}^{k=r} \frac{(-1)^k B^{m+1+k}}{k!\,(m+1+k)} \right\| + \frac{\left\|B^{m+r+2}\right\|}{m!(r+1)!(m+r+2)} + ||f_{m+2+r}(B)||, \tag{46}$$

where

$$||f_{m+2+r}(B)|| = \frac{1}{m!} \left\| \sum_{k\geq r+2} \frac{(-1)^k B^{m+1+k}}{k!\,(m+1+k)} \right\| \leq \frac{1}{m!(m+r+3)} \sum_{k\geq r+2} \frac{\left\|B^{m+k}\right\|}{k!\left(\frac{m+1+k}{m+r+3}\right)}. \tag{47}$$

If

$$\vartheta_{m_M} = \max\{\Theta_{m_M}, \tilde{\Theta}_{m_M}\} = \tilde{\Theta}_{m_M}, \tag{48}$$

using α_{min} from Section 2.3.1, it follows that

$$||f_{m+2+r}(2^{-s}A)|| \leq \frac{1}{m!(m+r+3)} \sum_{k\geq r+2} \frac{(2^{-s}\alpha_{min})^{m+k}}{k!\left(\frac{m+1+k}{m+r+3}\right)} \tag{49}$$

$$\leq \frac{(2^{-s}\alpha_{min})^{m+r+2}}{m!(m+r+3)(r+2)!} \left[1 + \frac{2^{-s}\alpha_{min}}{(r+3)\left(1+\frac{1}{m+r+3}\right)} + \frac{(2^{-s}\alpha_{min})^2}{(r+3)(r+4)\left(1+\frac{2}{m+r+3}\right)} + \cdots \right]$$

$$\leq \frac{(2^{-s}\alpha_{min})^{m+r+2}}{m!(m+r+3)(r+2)!} \sum_{j\geq 0} \beta^j = \frac{(2^{-s}\alpha_{min})^{m+r+2}}{m!(m+r+3)(r+2)!} \frac{1}{1-\beta}, \tag{50}$$

where

$$\beta = \frac{2^{-s}\alpha_{min}}{(r+3)\left(1+\frac{1}{m+r+3}\right)}, \tag{51}$$

must verify $\beta < 1$, and the binomial theorem has been used taking into account $(1+x)^j \geq 1 + jx$ for $x > 0$. If a sharper bound is needed, using (49) note that

$$||f_{m+2+r}(2^{-s}A)|| \leq \frac{1}{m!(m+r+3)} \sum_{k\geq r+2} \frac{(2^{-s}\alpha_{min})^{m+k}}{k!\left(\frac{m+1+k}{m+r+3}\right)}$$

$$\leq \frac{(2^{-s}\alpha_{min})^{m+1}}{m!(m+r+3)} \sum_{k\geq r+2} \frac{(2^{-s}\alpha_{min})^k}{k!}$$

$$\leq \frac{(2^{-s}\alpha_{min})^{m+1}}{m!(m+r+3)} \left[e^{2^{-s}\alpha_{min}} - \sum_{k=0}^{r+1} \frac{(2^{-s}\alpha_{min})^k}{k!} \right], \tag{52}$$

and an accurate enough approximation to this bound can be obtained using for instance Matlab's exponential function **exp** if r is not large and $2^{-s}\alpha_{min}$ is not

very small. Taking $r = q$, as in Section 2.3.2, and using Table 1, it follows that $q \leq 5$. On the other hand, for $m_M \geq 16$, taking into account (40), (48) and Table 2 it follows that

$$2^{-s}\alpha_{min} \geq \tilde{\Theta}_{16}/2 \approx 0.3956, \tag{53}$$

which is not so small to produce problems in the evaluation of bound (49). For instance, with the maximum value of q and the minimum value of $2^{-s}\alpha_{min}$ using Matlab we obtained that

$$\exp\left(\tilde{\Theta}_{16}/2\right) - \sum_{k=0}^{6} \left(\tilde{\Theta}_{16}/2\right)^k /k! = 3.16626365 \times 10^{-7}, \tag{54}$$

obtaining the same result for the 9 decimal digits shown using Matlab's Symbolic Math Toolbox and high precision arithmetic.

If $\vartheta_{m_M} = \max\{\Theta_{m_M}, \tilde{\Theta}_{m_M}\} = \Theta_{m_M}$ then using (14), for $||g_{m+1}(2^{-s}A)|| \ll 1$ it follows that

$$||h_{m+1}(2^{-s}A)|| \leq ||g_{m+1}(2^{-s}A)|| + O\left[||g_{m+1}(2^{-s}A)||^2\right] \approx ||g_{m+1}(2^{-s}A)||, \tag{55}$$

If $||g_{m+1}(2^{-s}A)|| < 1$ is not small, using Taylor series of function $\log(1-x)$ and (14) it follows that

$$||h_{m+1}(2^{-s}A)|| \leq -\log(1 - ||g_{m+1}(2^{-s}A)||). \tag{56}$$

3. Numerical experiments and conclusions

This section compares the Matlab implementation of the proposed Taylor algorithm, denoted by **extaynsv2**, with the original function **exptayns** from [3] (`http://personales.upv.es/~jorsasma/Software/exptayns.m`), and functions **expm** and **expm_new** which implement Padé algorithms from [5] and [6], respectively. Algorithm accuracy was tested by computing the relative error $\mathrm{E} = \|e^A - \tilde{Y}\|_1 / \left\|e^A\right\|_1$, where $\tilde{Y}$ is the computed approximation. Cost was given in terms of matrix products. The asymptotic cost in terms of matrix products for solving the multiple right-hand side linear system appearing in Padé algorithms was taken 4/3 [15]. We were interested in testing a wide range of matrices (diagonalizable and nondiagonalizable matrices) with a considerable dimension (between 128 and 1024), and whose exponentials could be computed accurately using orthogonal transformations. For this reason we chose the sets of matrices (1) and (2). The matrices of set (3) appear in the state of the art in the exponential matrix computation [5, 6]. These sets of matrices are described below:

(1) 3 sets of one hundred diagonalizable matrices of sizes 128, 256 and 1024, respectively. These matrices have the form $V^T DV$, 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) 3 sets of one hundred matrices with multiple eigenvalues of sizes 128, 256 and 1000, respectively. These matrices have the form $V^T DV$, 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 invertible matrix with random values in $[-0.5, 0.5]$ for size 1000, and

Table 3. Comparison of functions **exptaynsv2** (labelled 2) and **exptayns** (labelled 1) with maximum order $m_M = 30$: Maximum and minimum values of the matrix norm for each matrix set, maximum and minimum relative error ratios E_2/E_1, maximum and minimum relative error ratio E_1/u where u is the unit roundoff, number N of matrices where **exptaynsv2** cost is one matrix product lower than **exptayns**.

	diag 128	diag 256	diag 1024	Jord 128	Jord 256	Jord 1024
$\max\{\|A\|_1\}$	6.07e1	3.75e2	7.28e2	2.14e2	2.43e2	1.91e6
$\min\{\|A\|_1\}$	4.65e1	3.59	7.21	5.27	5.93	1.20e3
$\max\{E_2/E_1\}$	1.02	1.01	1.00	1.12	1.06	1
$\min\{E_2/E_1\}$	0.99	0.98	1.00	0.99	0.89	1
$\max\{E_1\}/u$	2.28e1	6.34e1	2.13e2	1.54e3	6.89e3	7.22e11
$\min\{E_1\}/u$	9.82	1.14	8.71	2.38	3.53	4.03e2
$N(\%)$	67	10	6	11	12	0

an orthogonal matrix obtained as $V = H/16$, where H is the Hadamard matrix for sizes 128 and 256.

(3) 43 25×25 matrices, 39 100×100 and 32 1000×1000 from the function **matrix** from the Matrix Computation Toolbox [9]. Matrices whose exponential cannot be represented in double precision due to overflow were excluded from all the matrices given by function **matrix** for each size.

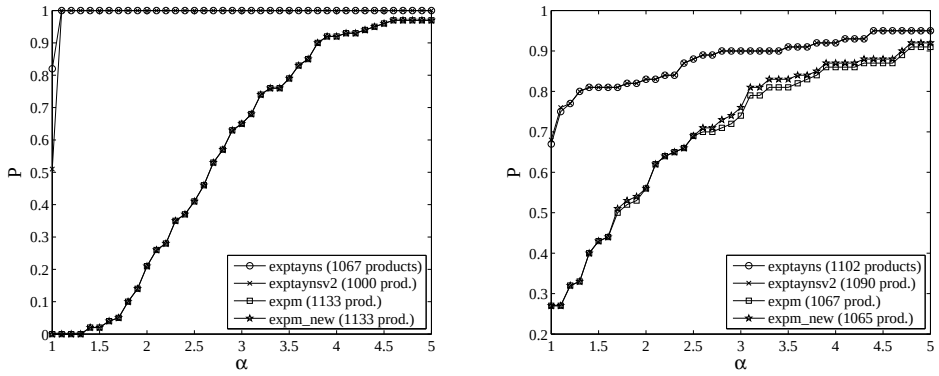
The "exact" value of matrix exponential e^A was computed using Matlab's Symbolic Math Toolbox or quadruple precision in Fortran, by using the following methods:

- For matrix sets 1 and 2: $e^A = V^T e^D V$, where $V^T e^D V$ was computed by using the **vpa** function of Matlab with 32 decimal digit precision.
- For matrix set 3: Taylor method with different orders and scaling parameters for each matrix to check the result correctness, using quadruple precision for 1000×1000 matrices, and 128 decimal digit precision for the remaining matrices.

Table 3 presents the results for sets (1) and (2) showing that maximum and minimum **exptaynsv2** and **exptayns** relative error ratios E_2/E_1, where E_2 is the relative error with function **exptaynsv2** and E_1 is the relative error for **exptayns**, are very close to unity for all matrix sets. This is verified even for the first set, whose norm range is such that **exptaynsv2** saves one matrix product for 67% matrices. In the remaining sets, which have a greater norm variability, the percentage of matrices where **exptaynsv2** saves products is lower, between 0 and 12%. Table 3 also shows that the relative error tends to increase with the matrix size, as expected. Figures 1(a) and 1(b) show the costs in terms of matrix products and the performance profiles [7] of **exptaynsv2**, **exptayns**, **expm_new** and **expm** for the two sets of the test matrices, where α coordinate varies between 1 and 5 in steps equal to 0.1, and p coordinate is the probability that the considered method has a relative error lower than or equal to α-times the smallest error over all the methods, where probabilities are defined over all matrices. Taylor functions had the highest accuracies with costs similar or even lower than **expm_new**.

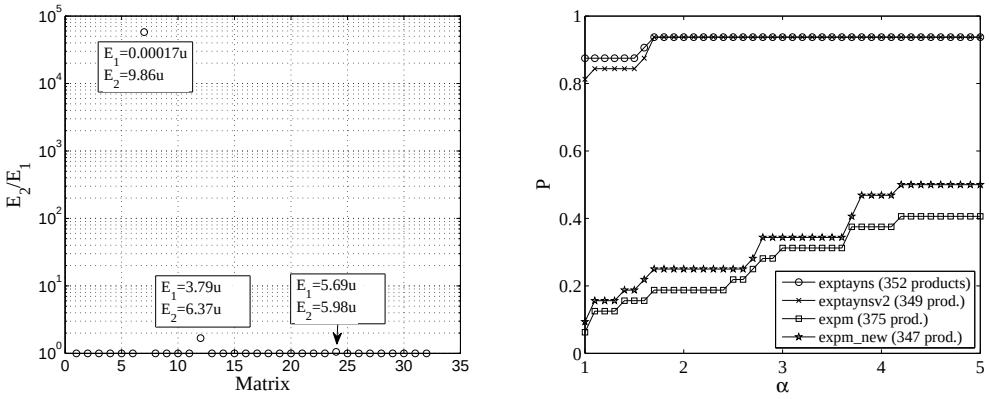
Figure 2(a) presents the relative error ratios E_2/E_1 for the 1000×1000 matrices from set 3, showing that there are three matrices (9.38%) where **exptaynsv2** saved one matrix product. It also shows that there is one matrix where E_2/E_1 was significant. It can be explained because for that matrix $E_1 = 0.00017u$ was much lower than the minimum expected error, i.e. the roundoff error u. In the remaining two cases E_2 and E_1 had the same order. Figure 2(b) gives the same conclusions as Figures 1(a) and 1(b).

In order to test the most critical cases for function **exptaynsv2** with respect to error, we multiplied each matrix A_i from the 25×25 matrices from test set 3, $1 \leq i \leq 43$, by a different constant $t_i \geq 1$ such that **exptaynsv2** cost for matrix $t_i \times A_i$ was one matrix product lower than the cost of the same function with matrix $(t_i + 0.01) \times A_i$. The same was done with the 100×100 matrices from the same



(a) Diagonalizable matrices $n = 128$. (b) Jordan matrices $n = 256$.

Figure 1. Performance profile and cost in terms of matrix products for the 128×128 diagonalizable and 256×256 Jordan matrices from sets 1 and 2, and $m_M = 30$ in Taylor functions.



(a) Error comparison $n = 1000$. (b) Performance profile $n = 1000$.

Figure 2. Relative error ratios E_2/E_1 for functions **exptayns** (E_1) and **exptaynsv2** (E_2), performance profile and cost in terms of matrix products, for $m_M = 30$ and the 1000×1000 matrices from the Matrix Computation Toolbox.

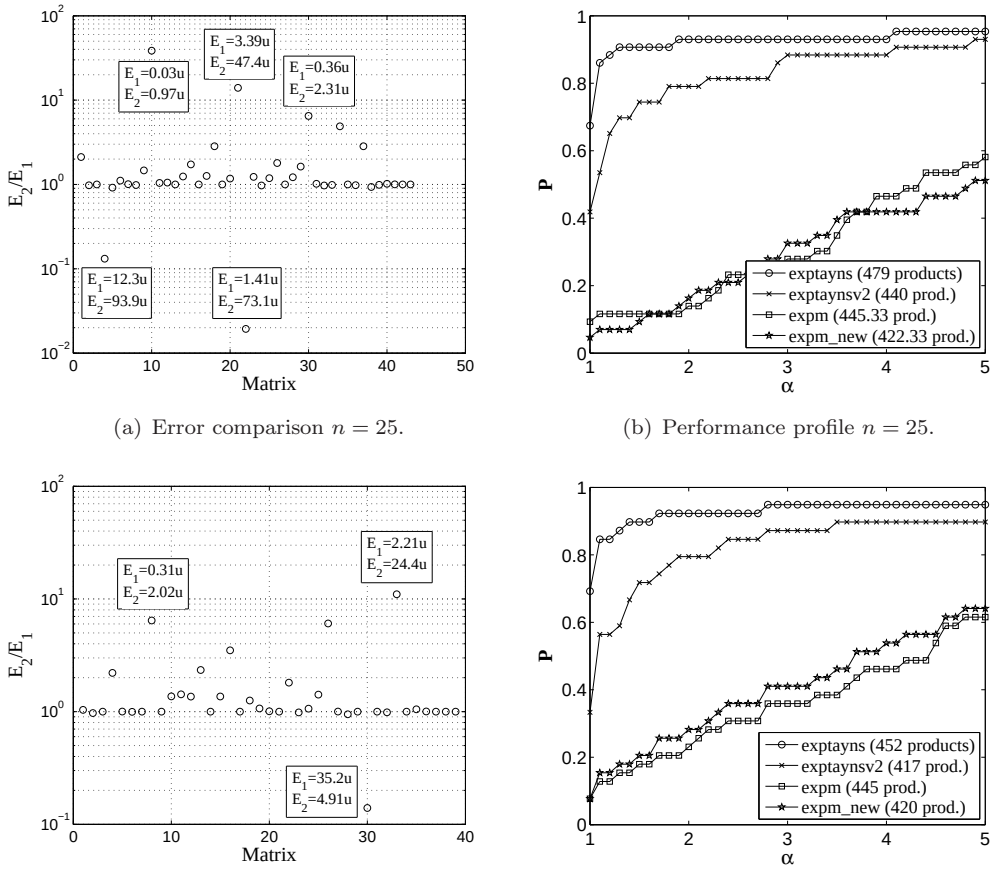
set. For the majority of the new matrices $t_i \times A_i$, **exptaynsv2** cost was one matrix product lower than **exptayns** cost, showing that the new proposed bounds based on $\phi(n, m)u = \sqrt{mn}u$ were verified, while the original bounds from [3] based on u were not. Maximum relative error differences between both functions were then expected for those matrices. Figure 3 presents the results for the modified sets of matrices sized 25 and 100, and similar conclusions to those from Figures 2(a) and 2(b) can be obtained, with the only difference that in the new matrix sets there were some cases where E_2 was significantly lower than E_1.

To sum up, a competitive modification of the Taylor algorithm from [3] has been proposed based on increasing the allowed forward error bound depending on the matrix size and Taylor order. The proposed modification reduces the cost with a small impact on accuracy, being more accurate than Padé existing state-of-the-art algorithms in the majority of tests with similar or even lower cost.

References

[1] 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.

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



(a) Error comparison $n = 25$.

(b) Performance profile $n = 25$.

(c) Error comparison $n = 100$.

(d) Performance profile $n = 100$.

Figure 3. Relative error ratios E_2/E_1 for functions **exptayns** (E_1), **exptaynsv2** (E_2), performance profiles and cost in terms of matrix products, for $m_M = 30$ and matrices with sizes 25 and 100 from the Matrix Computation Toolbox multiplied by constants t_i, see text.

[3] J. Sastre, J. Ibáñez, E. Defez and P. Ruiz, Accurate matrix exponential computation to solve coupled differential models in engineering, Math. Comput. Model., 54 (2011) 1835-1840.

[4] N. J. Higham, F. Tisseur, A block algorithm for matrix 1-norm estimation, with an application to 1-norm pseudospectra, SIAM J. Matrix Anal. Appl. 21 (2000) 1185–1201.

[5] N. J. Higham, The scaling and squaring method for the matrix exponential revisited, SIAM J. Matrix Anal. Appl. 26 (4) (2005) 1179–1193.

[6] 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.

[7] E. D. Dolan, J.J. Moré, Benchmarking optimization software with performance profiles, Math. Programming 91 (2002) 201–213.

[8] N. J. Higham. Accuracy and Stability of Numerical Algorithms. Society for Industrial and Applied Mathematics, Philadelphia, PA, USA, 2002.

[9] N. J. Higham, The Matrix Computation Toolbox, `http://www.ma.man.ac.uk/~higham/mctoolbox`.

[10] M. S. Paterson, L. J. Stockmeyer, On the number of nonscalar multiplications necessary to evaluate polynomials, SIAM J. Comput. 2 (1) (1973) 60–66.

[11] C. Fassino, Computation of Matrix Functions, PhD Thesis TD-7/93, Università di Pisa, Genova, 1993.

[12] M. Arioli, B. Codenotti, and C. Fassino The Padé method for computing the matrix exponential. Linear Algebra Appl., 240 (1996), 111–130.

[13] L. Dieci and A. Papini, Padé approximation for the exponential of a block triangular matrix, Linear Algebra Appl., 308 (2000), 183–202.

[14] L. Dieci and A. Papini, Conditioning of the exponential of a block triangular matrix, Numer. Algorithms 28 (2001), 137–150.

[15] S. Blackford and J. Dongarra. Installation guide for LAPACK. LAPACK Working Note 411, Department of Computer Science University of Tenessee, 1999.