Applied Mathematics and Computation 312 (2017) 66–77

Contents lists available at ScienceDirect

Applied Mathematics and Computation

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

Two algorithms for computing the matrix cosine function[☆]

Jorge Sastre[a], Javier Ibáñez[b,*], Pedro Alonso[c], Jesús Peinado[b], Emilio Defez[d]

[a] Instituto de Telecomunicaciones y Aplicaciones Multimedia, Universitat Politècnica de València, Camino de Vera s/n, Valencia 46022, España
[b] Instituto de Instrumentación para Imagen Molecular, Universitat Politècnica de València, Camino de Vera s/n, Valencia 46022, España
[c] Department of Information Systems and Computation, Universitat Politècnica de València, Camino de Vera s/n, Valencia 46022, España
[d] Instituto de Matemática Multidisciplinar, Universitat Politècnica de València, Camino de Vera s/n, Valencia 46022, España

A R T I C L E   I N F O

Keywords:
Matrix cosine
Scaling and recovering method
Taylor series
Forward error analysis
Backward error analysis
MATLAB

A B S T R A C T

The computation of matrix trigonometric functions has received remarkable attention in the last decades due to its usefulness in the solution of systems of second order linear differential equations. Several state-of-the-art algorithms have been provided recently for computing these matrix functions. In this work, we present two efficient algorithms based on Taylor series with forward and backward error analysis for computing the matrix cosine. A MATLAB implementation of the algorithms is compared to state-of-the-art algorithms, with excellent performance in both accuracy and cost.

1. Introduction

Many engineering processes are described by second order differential equations, whose solution is given in terms of the trigonometric matrix functions sine and cosine. Examples arise in the spatial semi-discretization of the wave equation or in mechanical systems without damping, where their solutions can be expressed in terms of integrals involving the matrix sine and cosine [1,2]. Several state-of-the-art algorithms have been provided recently for computing these matrix functions using polynomial and rational approximations with scaling and recovering techniques [3–6]. In order to reduce computational costs Paterson–Stockmeyer method [7] is used to evaluate the matrix polynomials arising in these approximations.

In the Taylor algorithm proposed in [4] we used sharp absolute forward error bounds. In the Taylor algorithm proposed in [6] we improved the previous algorithm using relative error bounds based on backward error bounds of the matrix exponentials involved in $\cos(A)$. Those error bounds do not guarantee that the cosine backward error bound in exact arithmetic is less than the unit roundoff in double precision arithmetic [6, Section 2]. However, according to the tests, that algorithm improved the accuracy with respect to the previous Taylor algorithm at the expense of some increase in cost (measured in flops). The algorithm proposed in [6] was also superior in both accuracy and cost to the version of the scaling and recovering Padé state-of-the-art algorithm in [5] not using the Schur decomposition.

Other algorithms based on approximations on L_∞ for normal and nonnegative matrices have been presented recently in [8]. In this work, we focus on general matrices and algorithms using approximations at the origin. We present two

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

* Corresponding author.

E-mail addresses: jsastrem@upv.es (J. Sastre), jjibanez@dsic.upv.es (J. Ibáñez), palonso@dsic.upv.es (P. Alonso), jpeinado@dsic.upv.es (J. Peinado), edefez@imm.upv.es (E. Defez).

http://dx.doi.org/10.1016/j.amc.2017.05.019
0096-3003/© 2017 Elsevier Inc. All rights reserved.

algorithms based on Taylor series that use Theorem 1 from [4] for computing the matrix cosine. We provide relative forward and backward error analysis for the matrix cosine Taylor approximation that improves even more the comparison to the algorithm in [5] with and without Schur decomposition in both accuracy and cost tests.

Throughout this paper $\mathbb{C}^{n\times n}$ denotes the set of complex matrices of size $n \times n$, I the identity matrix for this set, $\rho(X)$ the spectral radius of matrix X, and $\mathbb{N}$ the set of positive integers. In this work, we use the 1-norm to compute the actual norms. This paper is organized as follows. Section 2 presents a Taylor algorithm for computing the matrix cosine function. Section 3 deals with numerical tests and, finally, Section 4 gives some conclusions.

2. Algorithms for computing matrix cosine

The matrix cosine can be defined for all $A \in \mathbb{C}^{n\times n}$ by

$$\cos(A) = \sum_{i=0}^{\infty} \frac{(-1)^i A^{2i}}{(2i)!},$$

and let

$$T_{2m}(A) = \sum_{i=0}^{m} \frac{(-1)^i B^i}{(2i)!} \equiv P_m(B), \tag{1}$$

be the Taylor approximation of order $2m$ of $\cos(A)$, where $B = A^2$. Since Taylor series are accurate only near the origin, in algorithms that use this approximation the norm of matrix B is reduced by scaling the matrix. Then, a Taylor approximation is computed, and finally the approximation of $\cos(A)$ is recovered by means of the double angle formula $\cos(2X) = 2\cos^2(X) - I$. Algorithm 1 shows a general algorithm for computing the matrix cosine based on Taylor approximation. By using the fact that $\sin(A) = \cos(A - \frac{\pi}{2}I)$, Algorithm 1 also can be easily used to compute the matrix sine.

Algorithm 1 Given a matrix $A \in \mathbb{C}^{n\times n}$, this algorithm computes $C = \cos(A)$ by Taylor series.

```
1: Select adequate values of $m$ and $s$                    ▷ Phase I
2: $B = 4^{-s}A^2$
3: $C = P_m(B)$                     ▷ Phase II: Compute Taylor approximation
4: for $i = 1 : s$ do                    ▷ Phase III: Recovering $\cos(A)$
5:     $C = 2C^2 - I$
6: end for
```

In Phase I of Algorithm 1, m and s must be calculated so that the Taylor approximation of the scaled matrix is computed accurately and efficiently. In this phase some powers B^i, $i \geq 2$, are usually computed for estimating m and s and if so they are used in Phase II.

Phase II consists of computing the Taylor approximation (1). For clarity of the exposition we recall some results summarized in [6, Section 2]. Taylor matrix polynomial approximation (1), expressed as $P_m(B) = \sum_{i=0}^{m} p_i B^i$, $B \in \mathbb{C}^{n\times n}$, can be computed with optimal cost by the Paterson–Stockmeyer's method [7] choosing m from the set

$$\mathbb{M} = \{1, 2, 4, 6, 9, 12, 16, 20, 25, 30, 36, 42, \dots\},$$

where the elements of $\mathbb{M}$ are denoted as $m_1, m_2, m_3, \dots$ The algorithm computes first the powers B^i, $2 \leq i \leq q$ not computed in the previous phase, being $q = \lceil \sqrt{m_k} \rceil$ or $q = \lfloor \sqrt{m_k} \rfloor$ an integer divisor of m_k, $k \geq 1$, both values giving the same cost in terms of matrix products. Therefore, (1) can be computed efficiently as

$$\begin{aligned}
P_{m_k}(B) \; = & \\
(((p_{m_k}B^q \; + & \; p_{m_k-1}B^{q-1} + p_{m_k-2}B^{q-2} + \dots + p_{m_k-q+1}B + p_{m_k-q}I)B^q \\
+ & \; p_{m_k-q-1}B^{q-1} + p_{m_k-q-2}B^{q-2} + \dots + p_{m_k-2q+1}B + p_{m_k-2q}I)B^q \\
+ & \; p_{m_k-2q-1}B^{q-1} + p_{m_k-2q-2}B^{q-2} + \dots + p_{m_k-3q+1}B + p_{m_k-3q}I)B^q \\
\dots & \\
+ & \; p_{q-1}B^{q-1} + p_{q-2}B^{q-2} + \dots + p_1B + p_0I.
\end{aligned} \tag{2}$$

Table 1 (page 11) shows the values of q for different values of m. From Table 4.1 from [9, p. 74] the cost of computing (1) with (2) is $\Pi_{m_k} = k$ matrix products, $k = 1, 2, \dots$

Finally, Phase III is necessary to obtain the cosine of matrix A from $\cos(4^{-s}B)$ computed previously in Phase II. If m_k is the order used and s is the scaling parameter, then the computational cost of Algorithm 1 is $2(k+s)n^3$ flops, and the storage cost is $(2+q_k)n^2$.

The difficulty of Algorithm 1 is to find appropriate values of m_k and s such that $\cos(A)$ is computed accurately with minimum cost. For that, in the following sections we will use Theorem 1:

Table 1
Values of Θ_{m_k} (forward analysis), $\bar{\Theta}_{m_k}$ (backward analysis) and q_k used to compute (1) by Paterson–Stockmeyer method (2).

k	m_k	q_k	Θ_{m_k}	k	m_k	q_k	$\bar{\Theta}_{m_k}$
1	1	1	5.161913593731081e−8	5	9	3	1.798505876916759
2	2	2	4.307691256676447e−5	6	12	4	6.752349007371135
3	4	2	1.319680929892753e−2	7	16	4	9.971046342716772
4	6	3	1.895232414039165e−1	8	20	5	10.177842844012551

Theorem 1 ([4]). *Let $h_l(x) = \sum_{i\geq l} p_i x^i$ be a power series with radius of convergence w, $\tilde{h}_l(x) = \sum_{i\geq l} |p_i| x^i$, $B \in \mathbb{C}^{n\times n}$ with $\rho(B) < w$, $l \in \mathbb{N}$ and $t \in \mathbb{N}$ with $1 \leq t \leq l$. If t_0 is the multiple of t such that $l \leqslant t_0 \leqslant l+t-1$ and*

$$\beta_t = \max\{d_j^{1/j} : j = t, l, l+1, \dots, t_0-1, t_0+1, t_0+2, \dots, l+t-1\},$$

where d_j is an upper bound for $||B^j||$, $d_j \geq ||B^j||$, then

$$||h_l(B)|| \leqslant \tilde{h}_l(\beta_t).$$

2.1. Relative forward error in Taylor approximation

The following proposition gives a necessary and sufficient condition for the existence of $\cos^{-1}(A)$.

Proposition 1. *Let A be a matrix in $\mathbb{C}^{n\times n}$ and let $B = A^2$. If $\|B\| < a^2$ where $a = \operatorname{acosh}(2)$ then $\cos(A)$ is invertible.*

Proof. Since

$$\|I - \cos(A)\| = \left\| \sum_{k\geq 1} \frac{(-1)^k A^{2k}}{(2k)!} \right\| \leq \sum_{k\geq 1} \frac{\left\|A^2\right\|^k}{(2k)!} < \sum_{k\geq 1} \frac{a^{2k}}{(2k)!} = \cosh(a) - 1 = 1,$$

then, by applying Lema 2.3.3 from [10, pp. 58], we obtain that $I - (I - \cos(A)) = \cos(A)$ is invertible. □

Using Proposition 1, if

$$\|B\| = \|A^2\| < \operatorname{acosh}^2(2) \approx 1.7343, \tag{3}$$

then $\cos^{-1}(A)$ exists and it follows that the relative forward error of computing $\cos(A)$ by means of (1), denoted by E_f, is

$$E_f = \left\|\cos^{-1}(A)(\cos(A) - T_{2m}(A))\right\| = \left\| \sum_{i\geq m+1} e_i^{(2m)} A^{2i} \right\| = \left\| \sum_{i\geq m+1} e_i^{(2m)} B^i \right\|,$$

where the coefficients $e_i^{(2m)}$ depend on Taylor approximation order $2m$. If we define $g_{m+1}(x) = \sum_{i\geqslant m+1} e_i^{(2m)} x^i$ and $\tilde{g}_{m+1}(x) = \sum_{i\geq m+1} |e_i^{(2m)}| x^i$, and we apply Theorem 1, then

$$E_f = \|g_{m+1}(B)\| \leq \tilde{g}_{m+1}(\beta_t^{(m)}), \tag{4}$$

for every t, $1 \leq t \leq m+1$. Following [4, Section 5.1], in (4) we denote by $\beta_t^{(m)}$ the corresponding value of β_t from Theorem 1 for order m, and from now on we will use that nomenclature.

Let Θ_m be

$$\Theta_m = \max\left\{ \theta \geq 0 : \sum_{i\geq m+1} \left|e_i^{(2m)}\right| \theta^i \leq u \right\}, \tag{5}$$

where $u = 2^{-53}$ is the unit roundoff in double precision floating-point arithmetic. We have used MATLAB Symbolic Math Toolbox to evaluate

$\sum_{i\geq m+1} |e_i^{(2m)}| \theta^i$ for each m in 250-digit decimal arithmetic, adding the first 250 series terms with the coefficients obtained symbolically. Then, a numerical zero-finder is invoked to determine the highest value of Θ_m such that $\sum_{i\geq m+1} |e_i^{(2m)}| \Theta_m^i \leq u$ holds. For this analysis to hold it is necessary that $\cos(A)$ is invertible. Hence, if condition (3) holds and $\beta_t^{(m)} \leq \Theta_m$, then

$$E_f \leq u.$$

Some values of Θ_m are given in Table 1.

2.2. Backward error in Taylor approximation

In [5], a backward error analysis is made for computing sine and cosine matrix functions. For each matrix function, two analysis were made. In the cosine function, the first one is based on considering the function

$$h_{2m}(x) := \arccos(r_m(x)) - x,$$

where $r_m(x)$ is the $[m/m]$ Padé approximant to the cosine function, and the authors conclude that different restrictions make this analysis unusable [5, Section 2.2]. We checked that an error analysis for the matrix cosine Taylor approximation similar to that on [5, Section 2.2] yields analogous results. Therefore, in order to calculate the backward error ΔX of approximating $\cos(X)$ by Taylor polynomial $T_{2m}(X)$ such that

$$T_{2m}(X) = \cos(X + \Delta X), \tag{6}$$

we propose a different approach that holds for any matrix $X \in \mathbb{C}^{r\times r}$ and uses the following result whose proof is trivial:

Lemma 1. *If A and B are matrices in $\mathbb{C}^{r\times r}$ and $AB = BA$ then*

$$\cos(A + B) = \cos(A)\cos(B) - \sin(A)\sin(B). \tag{7}$$

Note that the backward error ΔX from (6) is a holomorphic function of X and then $X\Delta X = \Delta X X$. Therefore, using (6) and Lemma 1

$$\begin{aligned} T_{2m}(X) &= \cos(X + \Delta X) = \cos(X)\cos(\Delta X) - \sin(X)\sin(\Delta X) \\ &= \cos(X)\sum_{i\geq 0}\frac{(-1)^i \Delta X^{2i}}{(2i)!} - \sin(X)\sum_{i\geq 0}\frac{(-1)^i \Delta X^{2i+1}}{(2i+1)!}. \end{aligned}$$

Hence,

$$\begin{aligned} \cos(X) - T_{2m}(X) &= \sum_{i\geq m+1}(-1)^i\frac{X^{2i}}{(2i)!} \\ &= \sin(X)\sum_{i\geq 0}\frac{(-1)^i \Delta X^{2i+1}}{(2i+1)!} - \cos(X)\sum_{i\geq 1}\frac{(-1)^i \Delta X^{2i}}{(2i)!}, \end{aligned} \tag{8}$$

and consequently the backward error ΔX can be expressed by

$$\Delta X = \sum_{i\geq m} c_i^{(2m)} X^{2i+1}, \tag{9}$$

where coefficients $c_i^{(2m)}$ depend on Taylor approximation order $2m$, and ΔX commutes with X. Note that an expression similar to (8) can be obtained for other approximations of the matrix cosine such as Padé approximation. Using (8) and (9) it follows that

$$\sin(X)\Delta X + O(X^{4m+2}) = \sum_{i\geq m+1}(-1)^i\frac{X^{2i}}{(2i)!}. \tag{10}$$

Hence, coefficients $c_i^{(2m)}$, $i = m, m+1, \dots, 2m-1$ can be computed obtaining symbolically the Taylor series of $\sin(X)\Delta X$ from the left-hand side of (10) and solving the system of equations that arise when equating the coefficients of X^{2i}, $i = m+1, m+2, \dots, 2m$, from both sides of (10) using function `solve` from the MATLAB Symbolic Math Toolbox.

Analogously, using (8) and (9) it follows that

$$\sin(X)\Delta X + \cos(X)\frac{\Delta X^2}{2!} + O(X^{6m+4}) = \sum_{i\geq m+1}(-1)^i\frac{X^{2i}}{(2i)!}, \tag{11}$$

and coefficients $c_i^{(2m)}$, $i = 2m, 2m+1, \dots, 3m+1$, can be calculated by using coefficients $c_i^{(2m)}$, $i = m, m+1, \dots, 2m-1$, obtained previously, computing symbolically the Taylor series of $\sin(X)\Delta X + \cos(X)\frac{\Delta X^2}{2!}$ in the left-hand side of (11) and solving the system of equations that arise when equating the coefficients of X^{2i}, $i = 2m+1, 2m+2, \dots, 3m+1$, from both sides of (11). By proceeding analogously, $c_i^{(2m)}$, $i > 3m+1$ can be computed. Then, we compute the relative backward error of approximating $\cos(A)$ by $T_{2m}(A)$, denoted by E_b, as

$$E_b = \frac{\|\Delta A\|}{\|A\|} = \frac{\left\|\sum_{i\geq m} c_i^{(2m)} A^{2i+1}\right\|}{\|A\|} \leq \left\|\sum_{i\geq m} c_i^{(2m)} A^{2i}\right\| = \left\|\sum_{i\geq m} c_i^{(2m)} B^{i}\right\|.$$

If we define $h_m(x) = \sum_{i\geq m} c_i^{(2m)} x^i$ and $\tilde{h}_m(x) = \sum_{i\geq m} |c_i^{(2m)}| x^i$, and we apply Theorem 1, then

$$E_b \leq \|h_m(B)\| \leq \tilde{h}_m(\beta_t^{(m)}). \tag{12}$$

Let $\bar{\Theta}_m$ be

$$\bar{\Theta}_m = \max\left\{\theta \geq 0 : \sum_{i\geq m} \left|c_i^{(2m)}\right|\theta^i \leq u\right\}. \tag{13}$$

For computing $\bar{\Theta}_m$, we have used MATLAB Symbolic Math Toolbox to evaluate $\sum_{i\geq m}|c_i^{(2m)}|\theta^i$ in 250-digit decimal arithmetic for each m adding a different number of series terms depending on m with the coefficients obtained symbolically, and a numerical zero-finder was invoked to determine the highest value of $\bar{\Theta}_m$ such that $\sum_{i\geq m}|c_i^{(2m)}|\bar{\Theta}_m^i \leq u$ holds. We have checked that for $m = 1, 2, 4, 6, 9, 12$ the values of $\bar{\Theta}_m$ obtained in double precision arithmetic do not vary if we take more than 256 series terms (and even fewer terms for the lower orders).

Note that the values $\bar{\Theta}_1 = 2.66 \cdot 10^{-15}$, $\bar{\Theta}_2 = 2.83 \cdot 10^{-7}$, $\bar{\Theta}_4 = 4.48 \cdot 10^{-3}$ and $\bar{\Theta}_6 = 1.45 \cdot 10^{-1}$, presented with two significant digits, are lower than the corresponding Θ_m values, $m = 1, 2, 4, 6$, from Table 1, for the forward error analysis, respectively.

On the other hand, considering 1880 and 1946 series terms for $m = 16$ the corresponding $\bar{\Theta}_{16}$ values have a relative difference of $4.0010 \cdot 10^{-4}$. Considering 1880 and 1980 series terms for $m = 20$ the corresponding $\bar{\Theta}_{20}$ values have a relative difference of $1.6569 \cdot 10^{-3}$. The process of computing those values for so many terms was very time consuming and we took as final values the ones shown in Table 1 corresponding to 1946 series terms for $m = 16$ and 1980 series terms for $m = 20$.

With the final selected values of $\bar{\Theta}_m$ in Table 1 it follows that if $\beta_t^{(m)} \leq \bar{\Theta}_m$, then relative backward error is lower than the unit roundoff in double precision floating-point arithmetic, i.e.,

$$E_b \leq u \text{ for } m_k = 9, 12, \text{ and } E_b \lesssim u \text{ for } m_k = 16, 20.$$

2.3. Backward error in double angle formula of the matrix cosine

We are interested in the backward error in Phase III of Algorithm 1. In the previous section, we have shown that it is possible to obtain a small backward error in the Taylor approximation of the matrix cosine. As in [5, Section 2.3], it can be shown that the backward error propagates linearly through the double angle formula. A result for the backward error similar to Lemma 2.1 from [5] can be obtained for polynomial approximations of the matrix cosine.

Lemma 2. *Let $A \in \mathbb{C}^{n\times n}$ and $X = 2^{-s}A$, s integer non negative, and suppose that $t(X) = \cos(X + \Delta X)$ for a polynomial function t. Then the approximation Y by applying the double angle formula satisfies $Y = \cos(A + \Delta A)$ in exact arithmetic, and hence*

$$\frac{\|\Delta A\|}{\|A\|} = \frac{\|\Delta X\|}{\|X\|}.$$

Proof. The proof is similar to that given in Lemma 2.1 from [5]. □

Lemma 2 shows that if we choose m and s, such that $\|\Delta X\|/\|X\| \leq u$, with $X = 4^{-s}A$, then if $s \geq 1$ the total backward error in exact arithmetic after Phase III of Algorithm 1 is bounded by u, producing no error growth.

2.4. Determining the values of the Taylor approximation order m and the scaling parameter s

Since $\Theta_6 \simeq 0.1895 < \operatorname{acosh}^2(2) \simeq 1.7343 < \bar{\Theta}_9 \simeq 1.7985$, see (3), and the values Θ_{m_k} for the forward error analysis and $m_k \leq 6$ are greater than the corresponding $\bar{\Theta}_{m_k}$ values for the backward error analysis, we use the relative forward analysis for $m_k \leq 6$, and the relative backward analysis for $m_k \geq 9$.

Therefore, by Theorem 1, for $m_k = 1, 2, 4, 6$, if there exists t such that $\beta_t^{(m_k)} \leq \Theta_{m_k}$, one gets that (2.1) holds, and for $m_k = 9, 12, 16, 20$, if there exists t such that $\beta_t^{(m_k)} \leq \bar{\Theta}_{m_k}$ it follows that (2.2) holds. Table 1 shows the values Θ_{m_k}, $m_k = 1, 2, 4, 6$ and $\bar{\Theta}_{m_k}$, $m_k = 9, 12, 16, 20$. For simplicity of notation, from now on we will denote by Θ_{m_k} both Θ_{m_k}, $m_k \leq 6$ and $\bar{\Theta}_{m_k}$, $m_k \geq 9$.

The selection of the order m and scaling parameter s is as follows. If there exists t and m_k such that $\beta_t^{(m_k)} \leq \Theta_9$ then it is not necessary to scale B and the Taylor approximation order m_k, where $k = \min\{k : \beta_t^{(m_k)} \leq \Theta_{m_k}\}$ is selected, i.e., the order m_k providing the minimum cost. Since in this case no scaling is applied then the double angle formula of Phase III of Algorithm 1 is not applied. Else, we scale the matrix B by the scaling parameter

$$s = \max\left\{0, \left\lceil \frac{1}{2}\log_2\left(\frac{\beta_t^{(m_k)}}{\Theta_{m_k}}\right)\right\rceil\right\}, \quad m_k \in \{9, 12, 16\}, \tag{14}$$

such that the matrix cosine is computed with minimum cost. Lemma 2 ensures that the backward error propagates linearly through the double angle formula in exact arithmetic if $s \geq 1$. Following [11, Section3.1], the explanation for the minimum and maximum orders m_k to be used in (14) for scaling (giving the minimum cost) is as follows: Since $\Theta_9/4 > \Theta_6$ and $\Theta_9 \cdot 4 > \Theta_{12}$ the minimum order to select for scaling is $m = m_5 = 9$. On the other hand, since $\Theta_{20}/4 < \Theta_{12}$ and $\Theta_{16}/4 > \Theta_9$, if $||B|| > \Theta_9$ the maximum order to select for scaling is $m = m_7 = 16$. Following [11, Section3.1] the final selection of m_k is the maximum order $m_k \in \{9, 12, 16\}$ giving also the minimum cost. This selection provides the minimum scaling parameter s over all selections of m_k that provide the minimum cost.

Then the Taylor approximation of order m_k of $\cos(B/4^s)$ is computed, and if $s \geq 1$ the recovering Phase III of Algorithm 1 is applied.

For computing the parameters $\beta_t^{(m_k)}$ for $1 \leq m_k \leq 16$, from Theorem 1, it is necessary to calculate upper bounds d_k for $||B^k||$. We have developed two different algorithms to obtain the order m_k and the scaling parameter s. Algorithm 2 uses an estimation $\beta_{min}^{(m_k)}$ of the minimum of the values $\beta_t^{(m_k)}$ from Theorem 1 obtained using Algorithm 3. In order to calculate the upper bounds d_k of $||B^k||$ for obtaining $\beta_{min}^{(m_k)}$ Algorithm 3 uses only products of norms of matrix powers previously computed in Algorithm 2, i.e., $||B^i||$, $i \leq 4$. For instance, note that in order to compute $\beta_2^{(m)}$ for the relative forward error bound and $m = 2$, using (1) we need only bounds $d_2^{1/2}$ and $d_3^{1/3}$. From Table 1 for $m = 2$ one gets $q = q_2 = 2$ and B^i, $i = 1, 2$, are available, and we take $d_1 = \|B\|$ and $d_2 = \|B^2\|$. Since $||B^2||^{1/2} \leq ||B||$ it follows that $\beta_{min}^{(2)} = \beta_2^{(2)} = \max\{d_2^{1/2}, (d_2 d_1)^{1/3} = (d_2 d_1)^{1/3}$, (Step 3 of Algorithm 3). Something similar happens with $\beta_2^{(4)}$, resulting in $\beta_{min}^{(4)} = \beta_2^{(4)} = \max\{d_2^{1/2}, (d_2^2 d_1)^{1/5}\} = (d_2^2 d_1)^{1/5}$, (Step 5 of Algorithm 3).

Algorithm 2 Given a matrix $A \in \mathbb{C}^{n\times n}$, this algorithm determines the order m, scaling parameter s, and powers of $B = A^2$ needed for computing Taylor approximation of $\cos(A)$ using no estimation of norms of matrix powers.

```
1: $B_1 = A^2$
2: if $\|B_1\| \leq \Theta_1$ then $m = 1$, $s = 0$, quit
3: $B_2 = B_1^2$, obtain $\beta_{min}^{(2)}$ using Algorithm 3 with $m = 2$, $q = 2$
4: if $\beta_{min}^{(2)} \leq \Theta_2$ then $m = 2$, $s = 0$, quit
5: $\beta_{min}^{(4)} = \min\{\beta_{min}^{(2)}, \beta_{min}^{(4)}$ using Algorithm 3 with $m = 4$, $q = 2\}$
6: if $\beta_{min}^{(4)} \leq \Theta_4$ then $m = 4$, $s = 0$, quit
7: $B_3 = B_2 B_1$
8: $\beta_{min}^{(6)} = \min\{\beta_{min}^{(4)}, \beta_{min}^{(6)}$ using Algorithm 3 with $m = 6$, $q = 3\}$
9: if $\beta_{min}^{(6)} \leq \Theta_6$ then $m = 6$, $s = 0$, quit
10: $\beta_{min}^{(9)} = \min\{\beta_{min}^{(6)}, \beta_{min}^{(9)}$ using Algorithm 3 with $m = 9$, $q = 3\}$
11: if $\beta_{min}^{(9)} \leq \Theta_9$ then $m = 9$, $s = 0$, quit
12: $\beta_{min}^{(12)} = \min\{\beta_{min}^{(9)}, \beta_{min}^{(12)}$ using Algorithm 3 with $m = 12$, $q = 3\}$
13: if $\beta_{min}^{(12)} \leq \Theta_{12}$ then $m = 12$, $s = 0$, quit
14: $s_9 = \lceil 1/2 \log_2(\beta_{min}^{(9)}/\Theta_9)\rceil$
15: $s_{12} = \lceil 1/2 \log_2(\beta_{min}^{(12)}/\Theta_{12})\rceil$
16: if $s_9 \leq s_{12}$ then $s = s_9$, $m = 9$ quit        ▷ $m = 9$ used for scaling only if providing less cost than $m = 12$
17: $B_4 = B_3 B_1$
18: $\beta_{min}^{(12)} = \min\{\beta_{min}^{(12)}, \beta_{min}^{(12)}$ using Algorithm 3 with $m = 12$, $q = 4\}$
19: if $\beta_{min}^{(12)} \leq \Theta_{12}$ then $m = 12$, $s = 0$, quit
20: $s_{12} = \lceil 1/2 \log_2(\beta_{min}^{(12)}/\Theta_{12})\rceil$
21: $\beta_{min}^{(16)} = \min\{\beta_{min}^{(12)}, \beta_{min}^{(16)}$ using Algorithm 3 with $m = 16$, $q = 4\}$
22: $s_{16} = \max\{0, \lceil 1/2 \log_2(\beta_{min}^{(16)}/\Theta_{16})\rceil\}$
23: if $s_{12} \leq s_{16}$ then $m = 12$,$s = s_{12}$ else $m = 16$, $s = s_{16}$ quit        ▷ $m = 12$ only used if provides less cost than $m = 16$
```

Using Table 1, for $m = 6$ one gets $q = q_4 = 3$, being now also available B^3, and we take $d_3 = \|B^3\|$. Using (1), if $d_2^{1/2} \leq d_3^{1/3}$ we select $\beta_{min}^{(6)} = \beta_2^{(6)} = \max\{d_2^{1/2}, (d_2^2 d_3)^{1/7}\} = (d_2^2 d_3)^{1/7}$, (Step 8 of Algorithm 3). Else, we select

$$\beta_{min}^{(6)} = \beta_3^{(6)} = \max\{d_3^{1/3}, \min\{d_2^2 d_3, d_1 d_3^2\}^{1/7}, (d_3^2 d_2)^{1/8}\}$$
$$= \max\{\min\{d_2^2 d_3, d_1 d_3^2\}^{1/7}, (d_3^2 d_2)^{1/8}\},$$

(Step 10 of Algorithm 3). Value $\beta_{min}^{(m_k)}$ is obtained analogously for $m_k = 9, 12, 16$ (Steps 12–37 of Algorithm 3).

On the other hand, in order to reduce the value $\beta_{min}^{(m_k)}$ and therefore the scaling parameter s and/or order m given by Algorithm 2, using (4) and (2.2), similarly to (16) from [12] we approximated $\beta_{min}^{(m_k)} = \min\{\beta_t^{(m_k)}\}$ from Theorem 1 by

$$\beta_{min}^{(m_k)} \approx \max\left\{d_{m_k+1}^{1/(m_k+1)},\ d_{m_k+2}^{1/(m_k+2)}\right\},\ m_k \leq 6 \text{ (forward bound)}, \tag{15}$$

$$\beta_{min}^{(m_k)} \approx \max\left\{d_{m_k}^{1/m_k},\ d_{m_k+1}^{1/(m_k+1)}\right\}\ m_k \geq 9 \text{ (backward bound)}, \tag{16}$$

Algorithm 3 beta_NoNormEst: determines value $\beta_{min}^{(m)} = \min\{\beta_t^{(m)}\}$ from Theorem 1 given $m \in \{2, 4, 6, 9, 12, 16\}$, $d_i = ||B^i||$, $b_i = ||B^i||^{1/i}$, $i = 1, 2, \ldots, q$, for $B \in \mathbb{C}^{n\times n}$, using bounds $d_i \geq ||B^i||$, $i > q$, based on products of $||B^i||$, $i \leq q$.

```
switch $m$ do
    case 2                                                              ▷ $m = 2$
        $\beta_{min}^{(2)} = (d_2 d_1)^{1/3}$
    case 4                                                              ▷ $m = 4$
        $\beta_{min}^{(4)} = (d_2^2 d_1)^{1/5}$
    case 6                                                              ▷ $m = 6$
        if $b_2 \leq b_3$ then
            $\beta_{min}^{(6)} = \min\{d_2^2 d_3, d_1 d_3^2\}^{1/7}$
        else
            $\beta_{min}^{(6)} = \max\{\min\{d_2^2 d_3, d_1 d_3^2\}^{1/7}, (d_3^2 d_2)^{1/8}\}$
        end if
    case 9                                                              ▷ $m = 9$
        if $b_2 \leq b_3$ then
            $\beta_{min}^{(9)} = (d_2^3 d_3)^{1/9}$
        else
            $\beta_{min}^{(9)} = \max\{\min\{d_2^2 d_3^2, d_3^3 d_1\}^{1/10}, (d_3^3 d_2)^{1/11}\}$
        end if
    case 12                                                             ▷ $m = 12$
        if $q = 3$ then
            if $b_2 \leq b_3$ then
                $\beta_{min}^{(12)} = (d_2^5 d_3)^{1/13}$
            else
                $\beta_{min}^{(12)} = \max\{\min\{d_3^4 d_1, d_3^3 d_2^2\}^{1/13}, (d_3^4 d_2)^{1/14}$
            end if
        else if $q = 4$ then
            if $b_3 \leq b_4$ then
                $\beta_{min}^{(12)} = \max\{(d_3^3 d_4)^{1/13}, \min\{d_3^2 d_4^2, d_3^4 d_2\}^{1/14}$
            else
                $\beta_{min}^{(12)} = \max\{(d_4^2 \min\{d_3 d_2, d_4 d_1\}\}^{1/13}, (d_4^2 \min\{d_3^2, d_4 d_2\})^{1/14}$
            end if
        end if
    case 16                                                             ▷ $m = 16$
        if $b_3 \leq b_4$ then
            $\beta_{min}^{(16)} = \max\{(d_3^4 d_4)^{1/16}, \min\{d_3^5 d_2, d_3^3 d_4^2\}^{1/17}\}$
        else
            $\beta_{min}^{(16)} = \max\{(d_4^3 \min\{d_4 d_1, d_3 d_2\})^{1/17}, (d_4^3 \min\{d_3^2, d_4 d_2\})^{1/18}\}$
        end if
```

computing the 1–norms of the corresponding matrix powers $\|B^i\|$ by the estimation algorithm from [13] and taking bounds $d_i = \|B^i\|$. Eqs. (15) and (16) may give values $\beta_{min}^{(m_k)}$ lower than the ones given by Algorithm 3, especially for nonnormal matrices, since

$$\|B^p\| \leq \|B\|^{i_1}\|B^2\|^{i_2}\|B^3\|^{i_3}\|B^4\|^{i_4}, \quad i_1 + 2i_2 + 3i_3 + 4i_4 = p.$$

Then it is possible to substitute Algorithm 3 by a new algorithm that computes $\beta_{min}^{(m_k)}$ using (15) and (16) with norm estimations of matrix powers [13] for the corresponding d_i. For a complete MATLAB implementation see the nested function `ms_selectNormEst` from function `cosmtay.m` available in http://personales.upv.es/jorsasma/Software/cosmtay.m.

Function `cosmtay.m` also implements the option without norm estimation: The MATLAB implementation of Algorithm 2 can be seen in the nested function `ms_selectNoNormEst` and Algorithm 3 in `beta_NoNormEst` from `cosmtay.m`. Both functions are slightly different from Algorithms 2 and 3 to be compatible with the version with norm estimation `ms_selectNoNormEst`.

3. Numerical experiments

In this section, we compare the MATLAB functions `cosmtay`, `cosm` and `costaym`:

Function `cosmtay(A,NormEst)` (http://personales.upv.es/jorsasma/software/cosmtay.m), is the MATLAB implementation of Algorithm 2 with determination of the order m and the scaling parameter s using the 1-norm estimator [13] (`NormEst`=1 corresponding in `cosmtay` to the nested function `ms_selectNormEst`) with `cosm` [5, Algorithm 4.2] (http://github.com/sdrelton/cosm_sinm). The tests using Algorithm 2 for determining m and s (`NormEst`=0 in `cosmtay(A,NormEst)` corresponding to the nested function `ms_selectNoNormEst` in `cosmtay`) gave similar accuracy results and a relative increase of the number of matrix products less than or equal to 3% in tests, therefore we omitted them. We also noted that for small matrices the cost of the norm estimation algorithm is not negligible compared to a matrix product. Therefore, the algorithm using no norm estimation is typically more efficient. For large matrices the norm estimation algorithm is negligible, and then, the algorithm with norm estimation is faster when it really saves matrix products.

The MATLAB function `cosm` has an argument which allows to compute $\cos(A)$ by means of just Padé approximants, or also using the real Schur decomposition and the complex Schur decomposition. For this function, we have used Padé approximants with and without real Schur decomposition, denoted by `cosmSchur` and `cosm`, respectively.

Finally, function `costaym` is the MATLAB implementation of Algorithm 1 from [6] (http://personales.upv.es/jorsasma/software/costaym.m).

In tests we used MATLAB (R2014b) running on an Intel Core 2 Duo processor at 3.00 GHz with 4 GB main memory. The following tests were made:

- Test 1: 100 diagonalizable 128×128 real matrices with real and complex eigenvalues and 1-norms varying from 2.32 to 220.04.
- Test 2: 100 non diagonalizable 128×128 real matrices with eigenvalues whose algebraic multiplicity vary between 1 and 128 and 1-norms varying from 5.27 to 21.97.
- Test 3: Seventeen matrices with dimensions lower than or equal to 128 from the Eigtool MATLAB package [14], twenty eight matrices from the matrix function literature with dimensions lower than or equal to 128, fifty one 128×128 real matrices from the function `matrix` of the Matrix Computation Toolbox [15] and fifty two 8×8 real matrices obtained scaling the matrices from the Matrix Computation Toolbox [15] such that their norms vary between 0.000145 and 0.334780. This last group of matrices sized 8×8 were used for testing specifically the forward error analysis from Section 2.1, since for those matrices the lower orders $m = 1, 2, 4, 6,$ were only used.
- Test 4: Fifty matrices from the semidiscretization of the wave equation from [5, Section 7.5] [16, Problem 4].

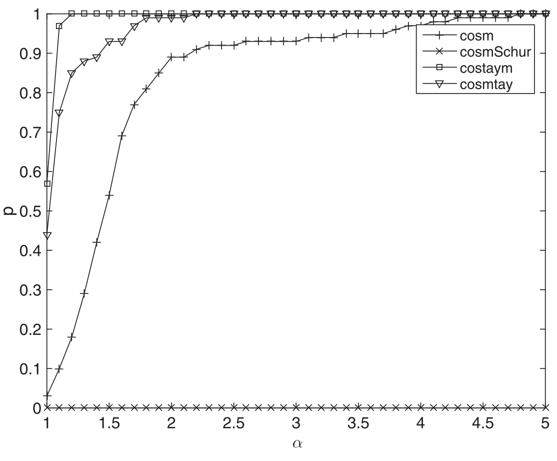
The "exact" matrix cosine was computed exactly for the matrices of Tests 1 and 2. Following [6, Section 4.1], for the other matrices we used MATLAB symbolic versions of a scaled Padé rational approximation from [5] and a scaled Taylor Paterson–Stockmeyer approximation (2), both with 4096 decimal digit arithmetic and several orders m and/or scaling parameters s higher than the ones used by `cosm` and `cosmtay`, respectively, checking that their relative difference was small enough. The algorithm accuracy was tested by computing the relative error

$$\mathrm{E} = \frac{\|\cos(A) - \tilde{Y}\|_1}{\|\cos(A)\|_1},$$

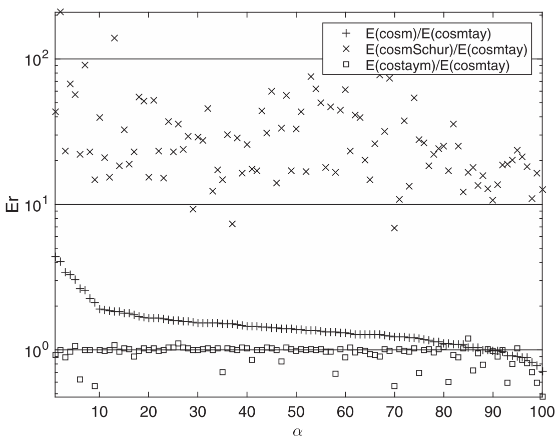
where $\tilde{Y}$ is the computed solution and $\cos(A)$ is the exact solution.

To compare the relative errors of the functions, we plotted in Fig. 1 the performance profiles and the ratio of relative errors E(`cosm`)/E(`cosmtay`) and E(`cosmSchur`)/E(`cosmtay`) for the three tests. In the performance profile, the α coordinate varies between 1 and 5 in steps equal to 0.1, and the p coordinate 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. The ratios of relative errors are presented in decreasing order of E(`cosm`)/E(`cosmtay`). The results were:

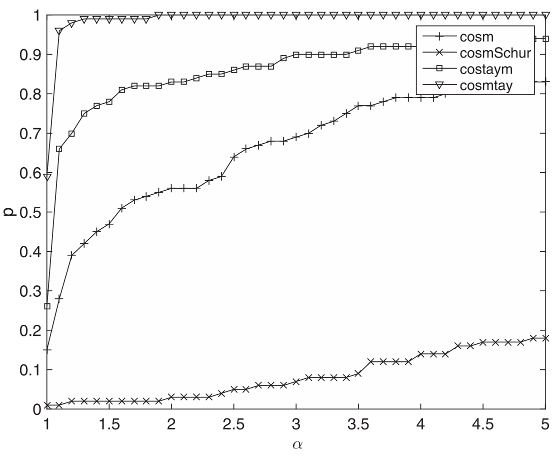
- Fig. 1 shows the performance profile and the relative error ratios giving the accuracy of the tested functions. Fig. 1a, c and e show that the most accurate functions in Tests 1, 2 and 3 were `costaym` from [6] and `cosmtay`. In Test 1 function `costaym` was slightly more accurate than `cosmtay` and this function was more accurate than `cosm` for 96 of the 100 matrices of Test 1 and more accurate than `cosmSchur` for all of them (see Fig. .1b). The graph of `cosmSchur` does not appear in Fig. 1a because for all matrices of Test 1 the relative error of this function was greater than 5 times the error by the other functions. In Test 2 `cosmtay` was the most accurate function, being more accurate than `cosm` for 93 of the 100 matrices and more accurate than `cosmSchur` for 98 matrices (see Fig. 1d). Finally, in Text 3 `costaym` from [6] was the most accurate function, and `cosmtay` was more accurate than `cosm` for 114 of the 135 matrices of Test 3 and more accurate than `cosmSchur` for 109 matrices of that test (see Fig. 1f).
- The ratios of flops from Fig. 2 show that the computational costs of `cosmtay` are always lower than `cosm`, `cosmSchur` and `costaym`. In the majority of matrices of Test 3, the ratios of flops of `cosmSchur` and `cosmtay` are between 2 and 4 and the ratios of flops of `cosm` and `cosmtay` are between 1 and 2 (Fig. 2a, c and e). In the majority of matrices of Test 3, the execution time ratios of `cosm` and `cosmtay` are between 1 and 5, the ratios of `cosmSchur` and `cosmtay` are greater than 5 (Fig. 2b, c and e), and the execution time ratios of `costaym` and `cosmtay` are between 1 and 2. Then, `cosmtay` provided always a lower cost than `costaym` but giving a lower accuracy for some test matrices.
- Test 4: Section 7.5 from [5] showed that for the matrices which appear in a wave equation problem the version of `cosm` using Schur decomposition (`cosmSchur`) was increasingly more accurate than the MATLAB function `costay` for the



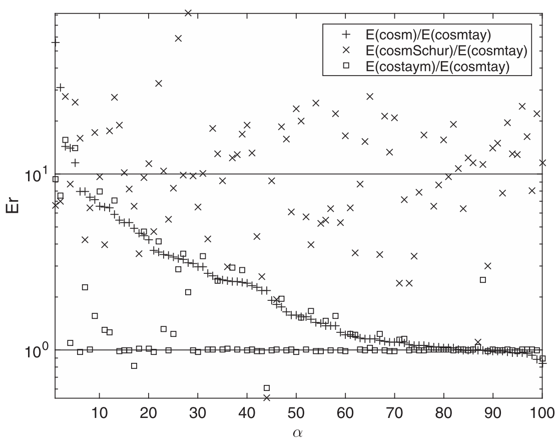
(a) Perfomance profile Test 1.



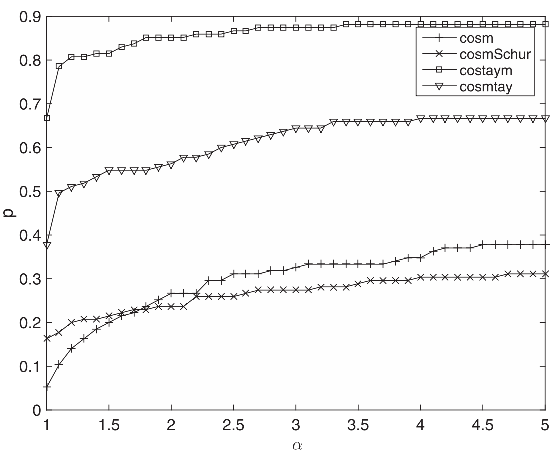
(b) Ratio of relative errors Test 1.



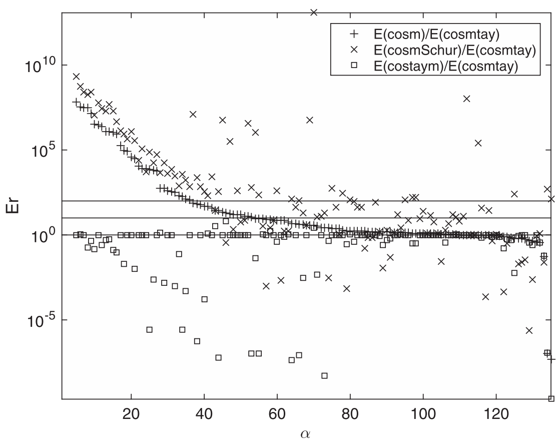
(c) Perfomance profile Test 2.



(d) Ratio of relative errors Test 2.

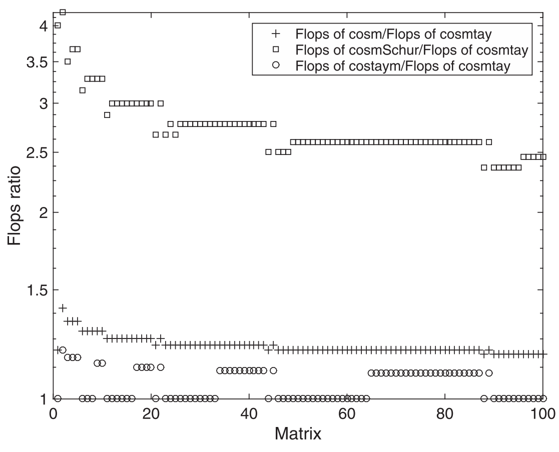


(e) Perfomance profile Test 3.

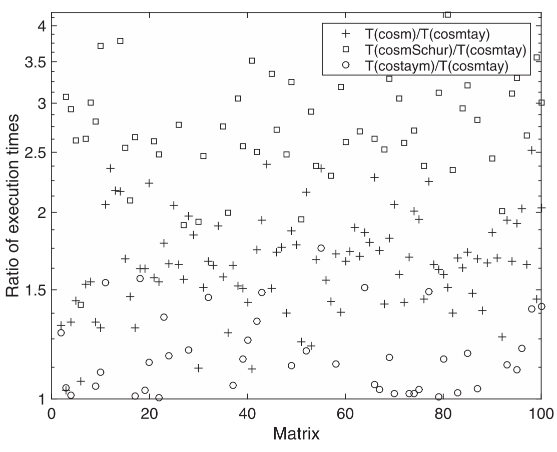


(f) Ratio of relative errors Test 3.

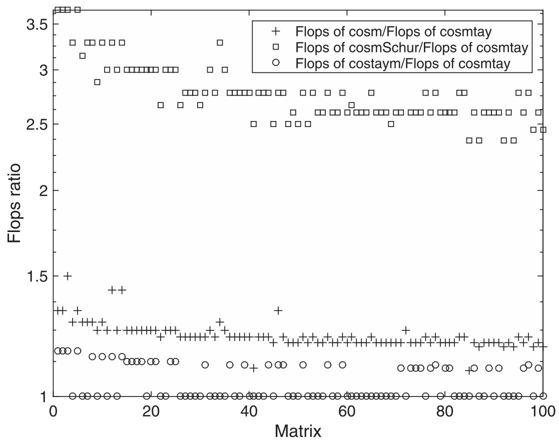
Fig. 1. Accuracy in Tests 1, 2 and 3.



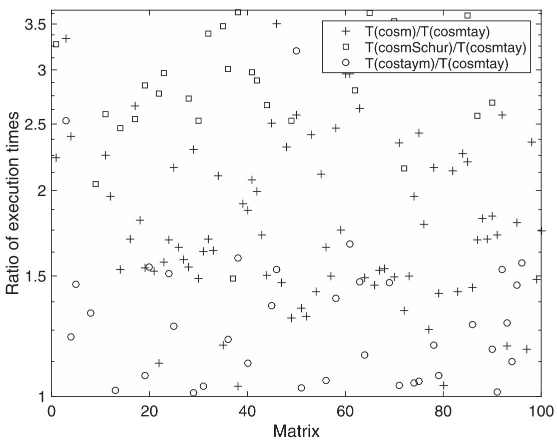
(a) Ratio of flops Test 1.



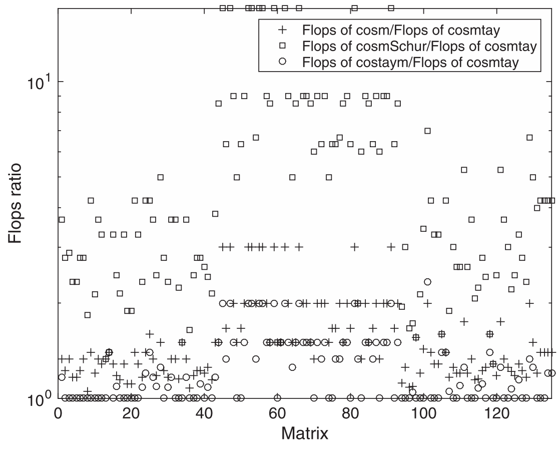
(b) Ratio of execution times Test 1.



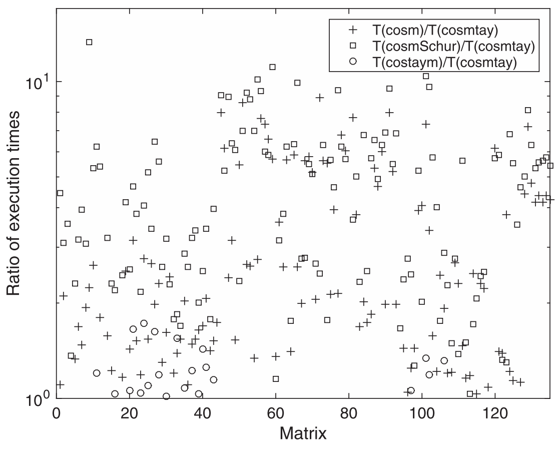
(c) Ratio of flops Test 2.



(d) Ratio of execution times Test 2.



(e) Ratio of flops Test 3.



(f) Ratio of execution times Test 3.

Fig. 2. Computational costs in Tests 1, 2 and 3.

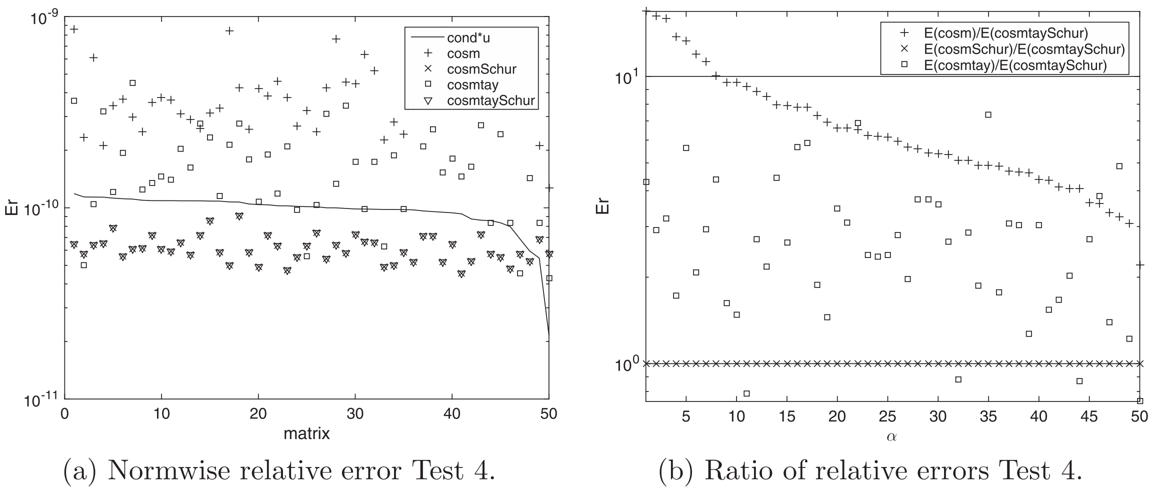


Fig. 3. Accuracy in Test 4.

biggest matrix dimensions in that test. This function was our first MATLAB implementation for computing the matrix cosine [4]. In Test 4 we compared the MATLAB implementations `cosmSchur`, `cosm`, `cosmtay` and `cosmtaySchur` for the biggest matrix size given in Section 7.5 from [5]. `cosmtaySchur` was based on the real Schur decomposition given by a modified implementation of Algorithm 4.2 from [5], where the Algorithm 1 is used for computing the cosine of the real Schur matrix of matrix A. Fig. 3 shows that the accuracy of `cosmSchur` and cosmtaySchur are similar, and both implementations are more accurate than the other implementations not based on the real Schur of a matrix. In [5] the authors claimed that `costay` had signs of instability. Note that `cosm` without Schur decomposition also shows signs of instability, even greater than those from `cosmtay`. We have tested that function `cosmtaySchur` is more accurate than `cosmSchur` for 28 of the 50 matrices of Test 4 and less accurate for 22 matrices of that test. Anyway, the differences are negligible and the main result from this test is that algorithms `cosmtay`, `cosm`, `cosmtaySchur`, `cosmSchur` had cost 1100, 1200, 1800 and 1900 matrix products, respectively. Therefore, `cosmtay` is 8.33% more efficient than `cosm` [5], and `cosmtaySchur` is an 5.26% more efficient than `cosmSchur` [5].

We have used `cosmtay` with the 1-norm estimator (`cosmtay(A,NormEst)` with `NormEst=1`) in the four tests above, because the results obtained with the variant that does not use the 1-norm estimator are similar in accuracy and we found that the computational cost in terms of matrix products for the implementation that uses the 1-norm estimator is only 1.90%, 1.85%, 3.00% and 0% lower than the implementation without estimation in Tests 1, 2, 3 and 4, respectively. However, the execution time was greater due to the overhead of using the 1-norm estimator, since the matrix sizes are not large enough so that the cost of the estimation ($O(n^2)$ for $n \times n$ matrices [13]) is negligible compared to the cost of matrix products ($O(n^3)$).

4. Conclusions

In this work, two accurate Taylor algorithms have been proposed to compute the matrix cosine. These algorithms use the scaling technique based on the double angle formula of the cosine function, the Paterson–Stockmeyer's method for computing the Taylor approximation, and new forward and backward relative error bounds for the matrix cosine Taylor approximation, which allow to calculate the optimal scaling parameter and the optimal order of the Taylor approximation. The two algorithms differ only in the use or not of the 1-norm estimation of norms of matrix powers [13]. The algorithm with no norm estimation uses Theorem 1 using the norms of matrix powers used for computing the matrix cosine to obtain bounds on the norms of matrix powers involved, giving in tests small relative cost differences in terms of matrix products with the version with norm estimation. The accuracy of both algorithms is similar, and the norm estimation algorithm has a cost negligible only for large matrices. Therefore, we recommend using the algorithm with no norm estimation for small matrices, and the algorithm with norm estimation for large matrices.

The MATLAB implementation that uses estimation was compared with other state-of-the-art MATLAB implementations for matrices sized up to 128 × 128 (analogous results were obtained with the other implementation). Numerical experiments show in general that our Taylor implementations have higher accuracy and less cost than the Padé state-of-the-art implementation `cosm` from [5] in the majority of tests. In particular, when the real Schur decomposition was used the ratio of flops between `cosmSchur` and `cosmtay` was flops(`cosmSchur`)/flops(`cosmtay`)> 5 for some matrices, and using the Schur decomposition in our algorithms gave the same accuracy results as `cosmSchur` with less cost. Numerical experiments also showed that function `cosmtay` was slightly less accurate than `costaym` from [6] in some tests but `cosmtay` provided always a lower computational cost.

Acknowledgments

The authors are very grateful to the anonymous referees, whose comments greatly improved this paper.

References

[1] S. Serbin, Rational approximations of trigonometric matrices with application to second-order systems of differential equations, Appl. Math. Comput. 5 (1) (1979) 75–92.
[2] S.M. Serbin, S.A. Blalock, An algorithm for computing the matrix cosine, SIAM J. Sci. Stat. Comput. 1 (2) (1980) 198–204.
[3] E. Defez, J. Sastre, J.J. Ibáñez, P.A. Ruiz, Computing matrix functions arising in engineering models with orthogonal matrix polynomials, Math. Comput. Model. 57 (7–8) (2013) 1738–1743.
[4] J. Sastre, J. Ibáñez, P. Ruiz, E. Defez, Efficient computation of the matrix cosine, Appl. Math. Comput. 219 (2013) 7575–7585.
[5] A.H. Al-Mohy, N.J. Higham, S.D. Relton, New algorithms for computing the matrix sine and cosine separately or simultaneously, SIAM J. Sci. Comput. 37 (1) (2015) A456–A487.
[6] P. Alonso, J. Ibáñez, J. Sastre, J. Peinado, E. Defez, Efficient and accurate algorithms for computing matrix trigonometric functions, J. Comput. Appl. Math. 309 (2017) 325–332, doi:10.1016/j.cam.2016.05.015.
[7] M.S. Paterson, L.J. Stockmeyer, On the number of nonscalar multiplications necessary to evaluate polynomials, SIAM J. Comput. 2 (1) (1973) 60–66.
[8] C. Tsitouras, V.N. Katsikis, Bounds for variable degree rational L_∞ approximations to the matrix cosine, Comput. Phys. Commun. 185 (11) (2014) 2834–2840.
[9] N.J. Higham, Functions of Matrices: Theory and Computation, SIAM, Philadelphia, PA, USA, 2008.
[10] G.H. Golub, C.V. Loan, Matrix computations, Johns Hopkins Studies in Mathematical Sciences, third, The Johns Hopkins University Press, 1996.
[11] J. Sastre, J.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.
[12] P. Ruiz, J. Sastre, J. Ibáñez, E. Defez, High performance computing of the matrix exponential, J. Comput. Appl. Math. 291 (2016) 370–379.
[13] 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.
[14] T.G. Wright, Eigtool, Version 2.1, 16, March 2009. Available online at: http://www.comlab.ox.ac.uk/pseudospectra/eigtool/.
[15] N.J. Higham, The Test Matrix Toolbox for MATLAB, Numerical Analysis Report No. 237, The University of Manchester, England, 1993.
[16] J.M. Franco, New methods for oscillatory systems based on ARKN methods, Appl. Numer. Math. 56 (8) (2006) 1040–1053, doi:10.1016/j.apnum.2005.09.005.