Efficient computation of the matrix cosine ☆

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

Universitat Politècnica de València, Spain

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

Abstract

Trigonometric matrix functions play a fundamental role in second order differential equation systems. This work presents an algorithm for computing the cosine matrix function based on Taylor series and the cosine double angle formula. It uses a forward absolute error analysis providing sharper bounds than existing methods. The proposed algorithm had lower cost than state-of-the-art algorithms based on Hermite matrix polynomial series and Padé approximants with higher accuracy in the majority of test matrices.

Keywords: matrix sine and cosine, double angle formula scaling method, Taylor series, error analysis, Paterson-Stockmeyer's method.

1. Introduction

Many engineering processes are described by second order differential equations, whose exact solution is given in terms of trigonometric matrix functions sine and cosine. For example, the wave problem

$$v^2 \frac{\partial^2 \psi}{\partial x^2} = \frac{\partial^2 \psi}{\partial t^2} \ , \tag{1}$$

plays an important role in many areas of engineering and applied sciences. If the spatially semi-discretization method is used to solve (1), we obtain the

☆This work has been supported by *Universitat Politècnica de València* grant PAID-06-011-2020.

*Corresponding author
Email address: `jorsasma@iteam.upv.es` (J. Sastre)

matrix differential problem

$$X''(t) + AX(t) = 0 \ , \ X(0) = X_0 \ , \ X'(0) = Y_1 \ , \tag{2}$$

where A is a square matrix and Y_0 and Y_1 are vectors. The solution of (2) is

$$X(t) = \cos\left(\sqrt{A}t\right)X_0 + \left(\sqrt{A}\right)^{-1}\sin\left(\sqrt{A}t\right)X_1, \tag{3}$$

where $\sqrt{A}$ denotes any square root of a non-singular matrix A [1, p. 36]. More general problems of type (2), with a forcing term $F(t)$ on the right-hand side arise from mechanical systems without damping, and their solutions can be expressed in terms of integrals involving the matrix sine and cosine [2].

The most competitive algorithms for computing matrix cosine are based on Padé approximations [3, 4, 5], and recently on Hermite matrix polynomial series [6, 7], using scaling of matrix A by a power of two, i.e. $A/2^s$ with a nonnegative integer parameter s, and the double angle formula

$$\cos 2A = 2\cos^2 A - I. \tag{4}$$

Matrix sine can be computed using formula $\sin(A) = \cos\left(A - \frac{\pi}{2}I\right)$ and an algorithm to compute both cosine and sine with a lower cost than computing them separately has been proposed in [8, Algorithm 12.8].

In this work we present a competitive scaling algorithm for the computation of matrix cosine based on Taylor series. It uses matrix scaling based on sharp absolute forward error bounds of the types given in [9], and Paterson-Stockmeyer's method for the evaluation of Taylor matrix polynomial [10]. A MATLAB implementation of this algorithm is made available online and it is compared with MATLAB function `cosher` based on Hermite series [6, 7], and MATLAB function `cosm` implementing the Padé algorithm from [5].

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. This paper is organized as follows. Section 2 summarizes some existing results for efficient matrix polynomial evaluation based on Paterson-Stockmeyer's method [10]. Section 3 presents a general Taylor algorithm for computing matrix cosine. Section 4 deals with the error analysis in exact arithmetic. Section 5 describes the new scaling algorithm. Section 6 provides a rounding error analysis. Section 7 deals with numerical tests, and Section 8 gives the conclusions.

2. Matrix polynomial computation by Paterson-Stockmeyer's method

From [11, p. 6454-6455] matrix polynomial Taylor approximation $P_m(B) = \sum_{i=0}^{m} p_i B^i$, $B \in \mathbb{C}^{n\times n}$ can be computed optimally for m in the set

$$\mathbb{M} = \{1, 2, 4, 6, 9, 12, 16, 20, 25, 30, \ldots\}, \tag{5}$$

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 [1, p. 72–74] for a complete description. First, matrix powers B^2, $B^3, \cdots, B^q$ 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 [1, p. 74]. Then, evaluation formula (23) from [11, p. 6455] is computed

$$\begin{aligned} P_{m_k}(B) = \big(\big(\cdots\big(&B^q p_{m_k} + B^{q-1} p_{m_k-1} + \cdots + B p_{m_k-q+1} + I p_{m_k-q}\big) \\ \times\ & B^q + B^{q-1} p_{m_k-q-1} + B^{q-2} p_{m_k-q-2} + \cdots + B p_{m_k-2q+1} + I p_{m_k-2q}\big) \\ \times\ & B^q + B^{q-1} p_{m_k-2q-1} + B^{q-2} p_{m_k-2q-2} + \cdots + B p_{m_k-3q+1} + I p_{m_k-3q}\big) \\ & \cdots \\ \times\ & B^q + B^{q-1} p_{q-1} + B^{q-2} p_{q-2} + \cdots + B p_1 + I p_0. \end{aligned} \tag{6}$$

Taking into account Table 4.1 from [1, p. 74], the cost of evaluating $P_{m_k}(B)$ in terms of matrix products, denoted by Π_{m_k}, for $k = 0, 1, \ldots$ is

$$\Pi_{m_k} = k. \tag{7}$$

3. General Algorithm

Taylor approximation of order $2m$ of cosine of matrix $A \in \mathbb{C}^{n\times n}$ can be expressed as the polynomial of order m

$$P_m(B) = \sum_{i=0}^{m} p_i B^i, \tag{8}$$

where $p_i = \frac{(-1)^i}{(2i)!}$ and $B = A^2$. Since Taylor series is accurate only near the origin, in algorithms that use this approximation the norm of matrix A is reduced using techniques based on the double angle formula (4), similar to those employed in the scaling an squaring method for computing the matrix exponential [12]. Algorithm 1 `costay` computes the matrix cosine based on these ideas, considering the values of $m_k \in \mathbb{M}$ for the truncated Taylor series (8), with a maximum allowed value of m_k equal to m_M.

Algorithm 1 `costay`: Given a matrix $A \in \mathbb{C}^{n\times n}$ and a maximum order $m_M \in \mathbb{M}$, this algorithm computes $C = \cos(A)$ by a Taylor approximation of order $2m_k \leqslant 2m_M$ and the double angle formula (4).

```
1: Preprocessing of matrix A.
2: B = A^2                                ▷ The memory for A is reused for B
3: SCALING PHASE: Choose m_k ⩽ m_M, m_k ∈ 𝕄, and an adequate scaling
   parameter s ∈ ℕ ∪ {0} for the Taylor approximation with scaling.
4: Compute C = P_{m_k}(B/4^s) using (6)
5: for i = 1 : s do
6:     C = 2C^2 − I
7: end for
8: Postprocessing of matrix C.
```

The preprocessing and postprocessing are based on applying transformations to reduce the norm of matrix A and recover the matrix $C \cong \cos(A)$ from the result of Loop 5-7. The available techniques to reduce the norm of a matrix are argument translation and balancing [1, p. 299]. The argument translation is based on the formula

$$\cos(A - \pi jI) = (-1)^j \cos(A),\ k \in \mathbb{Z},$$

and on finding the integer j such that the norm of matrix $A-\pi jI$ is minimum. This value can be calculated by using Theorem 4.18 from [1]. Balancing is a heuristic that attempts to equalize the norms of the kth row and kth column, for each k, by a diagonal similarity transformation defined by a non singular matrix D. Balancing tends to reduce the norm, though this is not guaranteed, so it should be used only for matrices where the norm is really reduced. For those matrices, if $A = D^{-1}(A - \pi jI)D$ is the obtained matrix in the preprocessing step, the postprocessing consists of computing $(-1)^j DCD^{-1}$, where C is the matrix obtained after Loop $5-7$.

We consider as input argument the maximum Taylor order m_M that can be used for computing the matrix cosine. In the SCALING PHASE the optimal order of Taylor approximation $m_k \leqslant m_M$ and the scaling parameter s are chosen. Analogously to [9], in the proposed scaling algorithm it will be necessary that the same powers of B are used in (6) for the two last orders m_{M-1} and m_M, i.e. $B^i, i = 2, 3, \ldots, q$. 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 $q_4 = 3$.

For the evaluation of P_{m_k} in Step 4 the Paterson-Stockmeyer's method described in Section 2 is applied. Then, the double angle formula is used to obtain $\cos(A)$ in Steps $5-7$ from matrix C of Step 4. Thus, using (7) it follows that computational cost of Algorithm `costay` in terms of matrix products is

$$\mathrm{Cost}(m_k, s) = 1 + k + s. \tag{9}$$

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
12	1	2	2	3	3	3		
16	1	2	2	3	3	4	4	
20	1	2	2	3	3	4	4	4

4. Error analysis in exact arithmetic and practical considerations

Using (8), it follows that

$$E_{m_k,s} = \left\|\cos\left(A/2^s\right) - P_{m_k}(B/4^s)\right\| = \left\|\sum_{i=m_k+1}^{\infty} \frac{(-1)^i B^i}{(2i)!4^{si}}\right\| \tag{10}$$

represents the forward absolute error in exact arithmetic from the approximation of cosine matrix function of the scaled matrix $A/2^s$ by Taylor series truncation. Analogously to [5], for the evaluation of $\cos(A)$ in IEEE double precision arithmetic we consider an absolute error-based algorithm, by selecting the appropriate values of m_k and s such that

$$E_{m_k,s} \leqslant u, \tag{11}$$

where $u = 2^{-53}$ is the unit roundoff in IEEE double precision arithmetic, providing high accuracy with minimal computational cost. The application of the proposed scaling algorithm to IEEE single precision arithmetic is straightforward and it will not be included in this paper.

In order to bound the norm of the matrix power series in (10), we will use the following improved version of Theorem 1 from [9, p. 1835]:

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

$$\beta_t = \max\{b_j^{1/j} : j = t, l, l+1, \ldots, t_0-1, t_0+1, t_0+2, \ldots, l+t-1\}, \quad (12)$$

where b_j *is an upper bound for* $||B^j||$, $||B^j|| \leqslant b_j$, *then*

$$||h_l(B)|| \leqslant \tilde{h}_l(\beta_t). \quad (13)$$

PROOF. Note that $||B^{t_0}||^{1/t_0} \leqslant (||B^t||^{t_0/t})^{1/t_0} = ||B^t||^{1/t}$, and then, if we denote

$$\alpha_t = \max\left\{\left\|B^j\right\|^{\frac{1}{j}} : j = t, l, l+1, \cdots, l+t-1\right\}, \quad (14)$$

it follows that

$$\alpha_t = \max\left\{\left\|B^j\right\|^{\frac{1}{j}} : j = t, l, l+1, \ldots, t_0-1, t_0+1, t_0+2, \ldots, l+t-1\right\} \leqslant \beta_t. \quad (15)$$

Hence

$$||h_l(B)|| \leqslant \sum_{j\geqslant 0} \sum_{i=l}^{l+t-1} |p_{i+jt}| \, ||B^t||^j ||B^i|| \leqslant \sum_{j\geqslant 0} \sum_{i=l}^{l+t-1} |p_{i+jt}| \alpha_t^{i+jt} \leqslant \sum_{i\geqslant l} |p_i| \beta_t^i = \tilde{h}_l(\beta_t). \qquad \square \quad (16)$$

Theorem 1 simplifies Theorem 1 from [9, p. 1835] avoiding the need for the bound b_{t_0} for $||B^{t_0}||$ to obtain β_t, see (12).

Similarly to [9] we use three types of bounds for the absolute error. Using (10) and Theorem 1 it follows that

$$E_{m_k,s} \leqslant \sum_{i=m_k+1}^{\infty} \frac{(\beta_t^{(m_k)}/4^s)^i}{(2i)!}, \quad (17)$$

$$E_{m_k,s} \leqslant \sum_{i=m_k+1}^{\infty} \frac{\|(B/4^s)^i\|}{(2i)!}, \quad (18)$$

$$E_{m_k,s} \leqslant \left\|(B/4^s)^{m_k+1}\right\| \left\| \sum_{i=0}^{q_k} \frac{(-1)^{i+m_k+1}(B/4^s)^i}{(2(i+m_k+1))!} \right\| + \sum_{i=m_k+q_k+2}^{\infty} \frac{\|(B/4^s)^i\|}{(2i)!}, \quad (19)$$

where $m_k \in \mathbb{M}$ and $\beta_t^{(m_k)}$, $l = m_k + 1$, $1 \leqslant t \leqslant l$, are the values given in (12) from Theorem 1. The superscript on $\beta_t^{(m_k)}$ remarks the dependency on the order m_k through the value of l. In order to compute (12), bounds b_j for the norms of matrix powers $||B^j||$ are needed. Analogously to [9], first, $\|B^{m_k+1}\|_1$ will be estimated using the block 1-norm estimation algorithm of [13], taking $b_{m_k+1} = \|B^{m_k+1}\|_1$. For a $n \times n$ matrix, this algorithm carries out a 1-norm power iteration whose iterates are $n \times r$ matrices, where r is a parameter that has been taken to be 2, see [14, p. 983]. Hence, the estimation algorithm has $O(n^2)$ computational cost, negligible compared with a matrix product, whose cost is $O(n^3)$. Then, we compute bounds b_j for the rest of needed matrix power 1-norms involved in (12), (18) and (19) using products of the estimated 1-norm of matrix powers, and the norms of the matrix powers needed for the computation of P_{m_k}, taking for them $b_j = ||B^j||_1$, $j = 1, 2, \ldots, q_k$, see Section 2. Thus, if $b_{e_k} = \|B^{e_k}\|_1$, $k = 1, 2, \ldots, L$, are all the known norms of matrix powers we obtain the remaining bounds of norms of matrix powers using

$$\left\|B^j\right\|_1 \leqslant b_j = \min\left\{b_{e_1}^{i_1} \cdot b_{e_2}^{i_2} \cdots b_{e_L}^{i_L} :\ e_1 i_1 + e_2 i_2 + \cdots + e_L i_L = j\right\}. \quad (20)$$

Note that the minimum in (20) is desirable but not necessary. A simple Matlab function has been provided to obtain b_j, see nested function `powerbound` from `costay.m` available at [15], analogous to nested function `powerbound` in `exptayns.m` from [9].

Then, the values $\beta_t^{(m_k)}$ from (17) can be obtained using bounds b_j in (12) with $l = m_k + 1$. Taking into account (17), let

$$\Theta_{m_k} = \max\left\{\theta : \sum_{i=m_k+1}^{\infty} \frac{\theta^i}{(2i)!} \leqslant u\right\}. \quad (21)$$

To compute Θ_{m_k}, we follow Higham in [16] using the MATLAB Symbolic Math Toolbox to evaluate $\sum_{i=m_k+1}^{\infty} \frac{\theta^i}{(2i)!}$ in 250-digit decimal arithmetic for each m_k, summing the first 200 terms with the coefficients obtained symbolically. Then, a numerical zero-finder is invoked to determine the highest value of Θ_{m_k} such that $\sum_{i=m_k+1}^{\infty} \frac{\theta^i}{(2i)!} \leqslant u$ holds. Table 2 shows the values obtained for the first ten values of $m_k \in \mathbb{M}$. Using (17) and (21), if $\beta_t^{(m_k)} \leqslant \Theta_{m_k}$ for two given values of m_k and t, then $E_{m_k,0} \leqslant u$, and $s = 0$ can be used with order m_k. Otherwise, for using order m_k the appropriate minimum scaling parameter $s > 0$ such that $\beta_t^{(m_k)}/4^s \leqslant \Theta_{m_k}$ and $E_{m_k,s} \leqslant u$ should be taken.

Taking into account (7) it follows that $\Pi_{m_{k+1}} = \Pi_{m_k} + 1$, but this is offset by the larger allowed value of $\theta = \beta_t^{(m_{k+1})}/4^s$ if $\Theta_{m_{k+1}} > 4\Theta_{m_k}$, since decreasing s by 1 saves one matrix multiplication in the application of the double angle formula in Steps $5-7$ of `costay`. Table 2 shows that $\Theta_{m_{k+1}} > 4\Theta_{m_k}$ for $k \leqslant 3$. Therefore, taking into account (17) and (21) it follows that selecting $m_M < m_4 = 9$ as maximum order is not an efficient choice.

On the other hand, Table 2 shows that $\Theta_{m_{k+1}}/4 < \Theta_{m_k}$ for $k \geqslant 4$. Therefore, for $m_M \geqslant m_5 = 12$, if the following expression holds for certain values of $s \geqslant 0$ and t_1, $1 \leqslant t_1 \leqslant m_M + 1$

$$\Theta_{m_M}/4 \lesssim \beta_{t_1}^{(m_M)}/4^s \leqslant \Theta_{m_M}, \tag{22}$$

then, for those matrices where next expression also holds

$$\Theta_{m_M}/4 \leqslant \beta_{t_2}^{(m_{M-1})}/4^s \leqslant \Theta_{m_{M-1}}, \tag{23}$$

for certain value of t_2, $1 \leqslant t_2 \leqslant m_{M-1} + 1$, one can select s with m_{M-1} instead of m_M saving one matrix product, see (9). Therefore, if $m_M \geqslant 12$ the proposed scaling algorithm will consider both orders m_{M-1} and m_M, selecting the one that provides the lowest cost.

Table 2: Highest values Θ_{m_k} such that $\sum_{i=m_k+1}^{\infty} \frac{\theta^i}{(2i)!} \leqslant u$.

k	m_k	Θ_{m_k}	k	m_k	Θ_{m_k}
0	1	5.161913651490293e−8	5	12	6.592007689102032
1	2	4.307719974921524e−5	6	16	2.108701860627005e1
2	4	1.321374609245925e−2	7	20	4.735200196725911e1
3	6	1.921492462995386e−1	8	25	9.944132963297543e1
4	9	1.749801512963547	9	30	1.748690782129054e2

Bounds (18) and (19) are used to refine the results obtained with (17). In [9] it is shown that using the 1-norm a bound of type (18) can be lower or higher than a bound of type (19) for normal and also for nonnormal matrices, depending on the specific matrix, see (20) [9, p. 1839]. Therefore, both bounds are considered.

To approximate bounds (18) and (19) we use the matrix power 1-norm $b_{m_k+1} = \|B^{m_k+1}\|_1$ estimated previously, and the bounds b_j for matrix powers obtained from (20). Taking into account that B^i takes the values $I,\ B,\ldots,B^{q_k}$ in the first summation of (19), this summation can be evaluated with a cost $O(n^2)$ reusing the matrix powers needed to compute P_{m_k} from (8), see Section 2. Following [9], we will truncate adequately the infinite series of (18) and (19) determining in Section 5 the minimum number of terms needed to introduce negligible errors.

5. Scaling algorithm

Scaling Phase of Algorithm 1 first tests if any of the orders $m_k < m_{M-1} \in \mathbb{M}$, $m_M \geqslant 12$, verifies (11) with $s = 0$, using the error bounds described in Section 4. If no order $m_k < m_{M-1}$ verifies (11) with $s = 0$, the algorithm will use Theorem 1, (17), (21) and values Θ_{m_k} from Table 2, to obtain initial values of scaling parameter for m_{M-1} and m_M, denoted by $s_0^{(M-1)}$ and $s_0^{(M)}$, respectively. If $s_0^{(M-1)} > 0$ or $s_0^{(M)} > 0$, the algorithm will verify if $s_0^{(M-1)}$ or $s_0^{(M)}$ can be reduced using bounds (18) and (19), selecting finally the combination of order and scaling parameter that gives the lowest cost in Steps $4-7$ from `costay`, see (9). Next we describe the proposed algorithm.

First we test if $m_0 = 1$ can be used with $s = 0$. Note that order m_0 is not very likely to be used, given (17), (21) and the value of Θ_1 in Table 2. Then we do not waste work estimating $||B^2||_1$, and, using Theorem 1 with $l = m_0 + 1$, (17) and (21), order m_0 will be selected only if

$$\beta_1^{(1)} = \max\{||B||, ||B^2||^{1/2}\} = ||B|| \leqslant \Theta_1, \tag{24}$$

taking in practice

$$\beta_1^{(1)} = \min\{||B||_1, ||B||_\infty\}. \tag{25}$$

For order $m_1 = 2$, taking into account Table 1, it follows that $q_1 = 2$ and then B^2 is computed. Analogously this order is not very likely to be used. Then, taking $b_2 = ||B^2||$ and $b_3 = ||B^2||\,||B||$ and using Theorem 1 with $l = m_1 + 1$, from (17) and (21) we will only select m_1 if

$$\beta_2^{(2)} = \max\{b_2^{1/2}, b_3^{1/3}\} = b_3^{1/3} = (||B^2||\,||B||)^{1/3} \leqslant \Theta_2, \tag{26}$$

taking in practice

$$\beta_2^{(2)} = (\min\{||B^2||_1\,||B||_1, ||B^2||_\infty||B||_\infty\})^{1/3}. \tag{27}$$

Orders $m_k \geqslant m_2 = 4$ are more likely to be used given that $\Theta_{m_k} \geqslant \Theta_4 \approx 0.013$, see Table 2. Then, we will test successively each value $m_k \in \mathbb{M}$, from $m_2 = 4$ to m_M in increasing order. Whenever the corresponding value of q_k for the current value of m_k increases, the corresponding matrix power B^{q_k} will be computed and used for evaluating the different error bounds. In Subsection 5.1 we use Theorem 1 and (17) to obtain an initial value of the scaling parameter for m_k, denoted by $s_0^{(k)}$. In Subsection 5.2 we use (18) and (19) to verify if $s_0^{(k)}$ can be reduced when $s_0^{(k)} > 0$. The 1-norm will be used for all norms in both Subsections. The complete algorithm is given finally as Algorithm 2.

5.1. Initial value of the scaling parameter

For each value of $m_k \in \mathbb{M}$, $4 \leqslant m_k \leqslant m_M$, we search for the minimum value of $\beta_t^{(m_k)}$ values of (12) from Theorem 1 with $l = m_k + 1$. For that task we estimate $b_{m_k+1} = ||B^{m_k+1}||_1$ as explained in Section 4, and use the estimated 1-norms of powers of B that have been computed to test orders $m_2, m_3, \ldots, m_{k-1}$, i.e. $||B^{m_2+1}||_1, ||B^{m_3+1}||_1, \cdots, ||B^{m_{k-1}+1}||_1$, the 1-norms of the powers of B used for the computation of P_{m_k}, i.e. B, $B^2, \ldots, B^{q_k}$, and bounds b_j obtained from (20). Thus, for $t = 2, 3, \ldots, q_k, m_2 + 1, m_3 + 1, \cdots, m_k + 1$ we will obtain successively

$$\beta_t^{(m_k)} = \max\{b_j^{1/j} : j = t, m_k+1, m_k+2, \ldots, t_0-1, t_0+1, t_0+2, \ldots, m_k+t\}, \tag{28}$$

where t_0 is the multiple of t in $[m_k + 1, m_k + t]$, stopping the process for the lowest value $t = r$ such that, see (12),

$$b_r^{\,1/r} \leq \max\left\{b_j^{1/j} : j = m_k + 1, m_k + 2, \ldots, r_0 - 1, r_0 + 1, \ldots, m_k + r\right\}, \tag{29}$$

where r_0 is the multiple of $r \in \mathbb{N}$ such that $m_k + 1 \leqslant r_0 \leqslant m_k + r$. Next we show that if (29) is verified then it follows that $\beta_i^{(m_k)} \geqslant \beta_r^{(m_k)}$ for $i \geqslant r$. Note that by (20) one gets $b_{r_0} \leqslant b_r^{r_0/r}$, and then it follows that

$$b_{r_0}^{1/r_0} \leqslant b_r^{1/r} \leqslant \max\{b_j^{1/j} : j = m_k + 1, m_k + 2, \ldots, r_0 - 1, r_0 + 1, \ldots, m_k + r\}. \tag{30}$$

Thus, substituting t by r and t_0 by r_0 in (28), and using (30) it follows that

$$\begin{aligned} \beta_r^{(m_k)} &= \max\{b_j^{1/j} : j = m_k + 1, m_k + 2, \ldots, r_0 - 1, r_0 + 1, \ldots, m_k + r\} \\ &= \max\{b_j^{1/j} : j = m_k + 1, m_k + 2, \ldots, m_k + r\}. \end{aligned} \tag{31}$$

Hence, for $i > r$, if

$$b_i^{1/i} \leqslant \max\{b_j^{1/j} : j = m_k+1, m_k+2, \ldots, i_0-1, i_0+1, \ldots, m_k+i\}, \quad (32)$$

where i_0 is the multiple of i in $[m_k+1, m_k+i]$, then in a similar way and using (31) it follows that

$$\begin{aligned} \beta_i^{(m_k)} &= \max\{b_j^{1/j} : j = m_k+1, m_k+2, \ldots, m_k+i\} \\ &\geqslant \max\{b_j^{1/j} : j = m_k+1, m_k+2, \ldots, m_k+r\} = \beta_r^{(m_k)}. \end{aligned} \quad (33)$$

Otherwise, if (32) is not verified, since $i > r$ and by (20) one gets $b_i^{1/i} \geqslant b_{i_0}^{1/i_0}$, then it follows that

$$\begin{aligned} \beta_i^{(m_k)} &= b_i^{1/i} > \max\{b_j^{1/j} : j = m_k+1, m_k+2, \ldots, m_k+i\} \\ &\geqslant \max\{b_j^{1/j} : j = m_k+1, m_k+2, \ldots, m_k+r\} = \beta_r^{(m_k)}, \end{aligned} \quad (34)$$

and then, using (33) and (34), it follows that $\beta_i^{(m_k)} \geqslant \beta_r^{(m_k)}$ for $i > r$.

Let $\beta_{min}^{(m_k)}$ be the minimum value of all computed values $\beta_t^{(m_k)}$, $1 \leqslant t \leqslant r$,

$$\beta_{min}^{(m_k)} = \min\{\beta_t^{(m_k)},\ t = 2, 3, \ldots, q_k, m_2+1, m_3+1, \cdots, r\}. \quad (35)$$

Then the appropriate initial minimum scaling parameter $s_0^{(k)} \geqslant 0$ is obtained such that $4^{-s_0^{(k)}} \beta_{min}^{(m_k)} \leqslant \Theta_{m_k}$, i.e.

$$s_0^{(k)} = \max\left\{0, \lceil 1/2 \log_2(\beta_{min}^{(m_M)}/\Theta_{m_k}) \rceil\right\}. \quad (36)$$

Table 3 shows the order and initial scaling selection depending on the values of $\beta_1^{(1)}$, $\beta_2^{(2)}$ and $\beta_{min}^{(m)}$, $m = 4, 6, \ldots, m_M$, taking into account (22) and (23) to use the most efficient choice between m_M and m_{M-1} in each case represented in the last four rows of the table, where i is a nonnegative integer parameter. If $s_0^{(k)} = 0$ then the proposed scaling algorithm selects m_k and $s = 0$ for Steps 4-7 of Algorithm `costay`. Otherwise the refinement proposed in next subsection is carried out.

5.2. Refinement of the scaling parameter

If $s_0^{(k)} > 0$ for the current value of m_k, taking into account (19), bounds from (20) are used to verify if the following inequality holds:

$$\sum_{i=m_k+1}^{m_k+N} \frac{b_i}{c_i 4^{si}} \leqslant u, \quad (37)$$

Table 3: Selection of initial scaling $s_0^{(k)}$ and order $m \in \mathbb{M}$, $m \leqslant m_M$ using only the first part of the proposed scaling algorithm, described in Subsection 5.1, depending on the values of $\beta_{min}^{(m)}$, for $m_M = 9, 12, 16, 20$. Total cost, denoted by C_m^T, is also presented. The cost with $m_M = 12$ and 9 is the same and it is presented in one column. $\beta_{min}^{(1)}$ and $\beta_{min}^{(2)}$ are not calculated for first orders $m = 1$ and 2, using $\beta_1^{(1)}$ and $\beta_2^{(2)}$ instead. The tests are done from top to bottom: If the condition for current row is not verified then we test the condition for the next row. In last four rows i can take the values $i = 0, 1, \dots$

m_M	9	12	9,12	16	20
interval	s_0, m	s_0, m	C_m^T	s_0, m, C_m^T	s_0, m, C_m^T
$\beta_1^{(1)} \leqslant \Theta_1$	0,1	0,1	0	0,1,0	0,1,0
$\beta_2^{(2)} \leqslant \Theta_2$	0,2	0,2	1	0,2,1	0,2,1
$\beta_{min}^{(m)} \leqslant \Theta_4$	0,4	0,4	2	0,4,2	0,4,2
$\beta_{min}^{(m)} \leqslant \Theta_6$	0,6	0,6	3	0,6,3	0,6,3
$\beta_{min}^{(m)} \leqslant \Theta_9$	0,9	0,9	4	0,9,4	0,9,4
$\beta_{min}^{(m)} \leqslant \Theta_{12}$	1,9	0,12	5	0,12,5	0,12,5
$\beta_{min}^{(m)} \leqslant 4\Theta_9$	1,9	1,9	5	0,16,6	0,16,6
$\beta_{min}^{(m)} \leqslant \Theta_{16}$	2,9	1,12	6	0,16,6	0,16,6
$\beta_{min}^{(m)} \leqslant 4^{i+1}\Theta_{12}$	2+i,9	1+i,12	6+i	1+i,12,6+i	i,20,7+i
$\beta_{min}^{(m)} \leqslant 4^{i+2}\Theta_9$	2+i,9	2+i,9	6+i	1+i,16,7+i	i,20,7+i
$\beta_{min}^{(m)} \leqslant 4^{i}\Theta_{20}$	3+i,9	2+i,12	7+i	1+i,16,7+i	i,20,7+i
$\beta_{min}^{(m)} \leqslant 4^{i+1}\Theta_{16}$	3+i,9	2+i,12	7+i	1+i,16,7+i	1+i,16,7+i

where $c_i = (2i)!$, and $s = 0$ if $m_k < m_{M-1}$, and $s = s^{(k)} = s_0^{(k)} - 1 \geqslant 0$ if $m_k \geqslant m_{M-1}$. For testing (37), we have truncated the series in (19) by choosing $N \geqslant q_k + 2$ terms as this number of terms will be also used when computing (19), see (38). At the end of this subsection we provide justification for the number of terms N to select for a negligible truncation error. We stop the series summation in (37) if after summing one term the sum is greater than u. If the sum of one or more terms is lower than u but the complete truncated series sum is not, we can estimate $b_{m_k+2} = ||B^{m_k+2}||_1$ to verify if (37) holds. If (37) holds, using (18) it follows that the forward absolute error $E_{m_k,s}$ is approximately lower than or equal to u.

If (37) does not hold, then, from (19) we verify if next bound holds

$$\frac{\|B^{m_k+1}\|_1}{4^{s(m_k+2)}}\left\|\sum_{i=0}^{q_k}\frac{c_{m_k+2}}{c_{i+m_k+1}}\frac{(-1)^i B^i}{4^{s(i-1)}}\right\|_1+\sum_{i=m_k+q_k+2}^{m_k+N}\frac{c_{m_k+2}}{c_i}\frac{b_i}{4^{si}}\leq uc_{m_k+2}, \quad (38)$$

where $s=0$ if $m_k<m_{M-1}$, and $s=s^{(k)}=s_0^{(k)}-1\geqslant 0$ if $m_k\geqslant m_{M-1}$, and we truncate the series in (38) taking $N\geqslant q_k+2$. We multiply both sides of (19) by c_{m_k+2} to save the product of matrix B by a scalar. If the first term of the left-hand side of (38) is lower than uc_{m_k+2} but the sum of the two terms is not, we can estimate $b_{m_k+q_k+2}=||B^{m_k+q_k+2}||_1$ to verify if (38) holds then.

Next, we obtain lower bounds for expression (38) to avoid its computation in some cases, see (16)-(18) from [9, p. 1838]. Let

$$T_i^{(m_k)}=\frac{c_{m_k+2}\,\|B^i\|}{c_{i+m_k+1}4^{s(i-1)}}, i=0:q_k,$$

and

$$T_j^{(m_k)}=\max\left\{T_i^{(m_k)}, i=0:q_k\right\},$$

be. Since

$$\left\|\sum_{i=0}^{q_k}\frac{c_{m_k+2}}{c_{i+m_k+1}}\frac{(-1)^i B^i}{4^{s(i-1)}}\right\|\geqslant T_s^{(m_k)},$$

where

$$T_s^{(m_k)}=\max\left\{0, T_j^{(m_k)}-\sum_{i\neq j}T_i^{(m_k)}\right\},$$

then if

$$\frac{\|B^{m_k+1}\|}{4^{s(m_k+2)}}T_s^{(m_k)}+\sum_{i=m_k+q_k+2}^{i=m_k+N}\frac{c_{m_k+2}b_i}{c_i4^{si}}>uc_{m_k+2}, \quad (39)$$

then there is no need to test (38).

For the case where $m_k<m_{M-1}$ if (37) or (38) hold with $s=0$, the scaling algorithm selects m_k and $s=0$ for Steps $4-7$ of `costay`. Otherwise, the two stages of the scaling algorithm from previous and this subsections are repeated with the next value of $m_k\in\mathbb{M}$ until $m_k=m_{M-1}$.

For $m_k\geqslant m_{M-1}$, firstly, an initial value $s_0^{(k)}$ of the scaling parameter is obtained as explained in Subsection 5.1, see (36). If $s_0^{(k)}=0$, then m_k and $s^{(k)}=0$ are selected. Otherwise, (37) and (38) are tested with $s^{(k)}=s_0^{(k)}-1$.

If none of both bounds hold, then we take $s^{(k)} = s_0^{(k)}$, and finally the order m_{M-1} or m_M that provides the lowest cost is selected for Steps $4-7$ of `costay`. It is possible to evaluate $P_m(4^{-s}B)$ with optimal cost for both orders because we set in its evaluation that both use the same powers of B, see Section 3.

Note that the cost of evaluating (37) and (38) is $O(n^2)$ and if any of them is verified with $s^{(k)} < s_0^{(k)}$ matrix products are saved, whose cost is $O(n^3)$.

Finally, we provide justification for the truncation of the series in (37) and (38). From (17), (35) and (36) it follows that the remainder of the infinite series without the first N terms, denoted by $R_{m_k+N+1,s}(B)$, verifies

$$||R_{m_k+N+1,s}(B)|| \leqslant \sum_{i=m_k+N+1}^{\infty} \frac{\|B^i\|}{(2i)!4^{si}} \leqslant \sum_{i=m_k+N+1}^{\infty} \frac{(\beta_{min}^{(m_k)}/4^s)^i}{(2i)!}. \tag{40}$$

If the minimum value of s to be tested is $s = s^{(k)} = s_0^{(k)} - 1$ then by (36) it follows that $\beta_{min}^{(m_k)}/4^s \leqslant 4\Theta_{m_k}$. Hence, using (40) it follows that

$$||R_{m_k+N+1,s}(B)|| \leqslant \sum_{i=m_k+N+1}^{\infty} \frac{(4\Theta_{m_k})^i}{(2i)!}. \tag{41}$$

Using Matlab Symbolic Math Toolbox and computing the first 150 terms of the series in (41) with high precision arithmetic, Table 4 presents the values of bound (41) for m_k, $k = 2, 3, \ldots, 7$, with the corresponding values of q_k proposed in Section 3 for each value of $m_k \in \mathbb{M}$, and $N = q_k + 2, q_k + 3, \ldots, q_k + 8$. Since error $E_{m,s}$ must verify $E_{m,s} \leqslant u \approx 1.11 \cdot 10^{-16}$ and rounding errors of values at least nu are expected in the evaluation of P_{m_k} where n is the matrix dimension, see [1], the values of bound (41) from Table 4 are satisfactory taking $N = q_k + 2$ for $m_k \leqslant 12$, $N = q_k + 4$ for $m_k = 16$, and $N = q_k + 7$ for $m_k = 20$.

However, in numerical tests we have observed that if (37) or (38) hold with $N = q_k + 2$, usually the value of $\beta_{min}^{(m_k)}$ in (40) is nearer to Θ_{m_M} than to $4\Theta_{m_M}$, and bound (40) is usually much lower than bound (41), therefore $N = q_k + 2$ being a good selection for all $m \leqslant 20$, see Section 7.

On the other hand, note that the values obtained in Table 4 for (41) with $m_k = 4$ and 6 are much lower than the unit roundoff u. Thus, in order to test if we can permit values of the final scaling parameter up to $s = s_0^{(k)} - 2$ for those orders, we have taken $4^2\Theta_{m_k}$ instead of $4\Theta_{m_k}$ in (41) with

$N = q_k + 2$, resulting $||R_{4+q_2+2+1,s}(B)|| \leqslant 1.3\text{e}{-22}$ and $||R_{6+q_3+2+1,s}(B)|| \leqslant 1.2\text{e}{-18}$. Thus, for orders $m_k = 4$ and 6 we can take $s = s^{(k)} \geqslant s_0^{(k)} - 2$ with $N = q_k + 2$. Taking this into account Algorithm 2 describes the proposed scaling algorithm. The complete algorithm `costay` has been implemented in MATLAB and made available online in [15]. This version of `costay` permits the selection of maximum order m_M from 6 to 30 for testing purposes.

Table 4: Values of bound (41).

	N						
m_k, q_k	q_k+2	q_k+3	q_k+4	q_k+5	q_k+6	q_k+7	q_k+8
4,2	5.0e−28	7.0e−32	8.0e−36	7.7e−40	6.2e−44	4.4e−48	2.6e−52
6,3	6.9e−26	8.1e−29	8.2e−32	7.3e−35	5.6e−38	3.9e−41	2.4e−44
9,3	1.8e−20	1.3e−22	7.9e−25	4.4e−27	2.2e−29	9.8e−32	4.0e−34
12,3	1.0e−16	2.0e−18	3.3e−20	5.0e−22	7.0e−24	8.9e−26	1.0e−27
12,4	2.0e−18	3.3e−20	5.0e−22	7.0e−24	8.9e−26	1.0e−27	1.1e−29
16,4	3.8e−14	1.4e−15	4.8e−17	1.5e−18	4.5e−20	1.2e−21	3.1e−23
20,4	1.4e−10	8.7e−12	5.0e−13	2.7e−14	1.3e−15	6.2e−17	2.7e−18

Algorithm 2 SCALING PHASE: Given matrix B from Step 2 of `costay` and maximum order m_M with $12 \leqslant m_M \leqslant 20$, it computes $m \leqslant m_M$, $m \in \mathbb{M}$, q from Table 1, and the scaling parameter s to be used in Steps $4-7$. This algorithm uses the values m_k and q_k from Table 1, and Θ_{m_k} from Table 2.

$b_1 = ||B||_1$, $d_1 = ||B||_\infty$
if $\min\{b_1, d_1\} \leqslant \Theta_1$ **then** ▷ Test $m_0 = 1$
    return $s = 0$, $m = m_0$, $q = 1$
end if
$B_2 = B^2$, $b_2 = ||B_2||_1$, $d_2 = ||B_2||_\infty$
if $\min\{b_2 \cdot b_1, d_2 \cdot d_1\}^{1/3} \leqslant \Theta_2$ **then** ▷ Test $m_1 = 2$
    return $s = 0$, $m = m_1$, $q = 2$
end if
$q = q_2$, $k = 1$
while $m < m_M$ **do**
    $k = k + 1$, $m = m_k$
    if $q < q_k$ **then**
        $q = q_k$
        if $q = 3$ **then**

$B_3 = B_2 B,\ b_3 = ||B_3||_1$
else if $q = 4$ **then**
$B_4 = B_2^2,\ b_4 = ||B_4||_1$
end if
end if
Estimate $b_{m+1} = ||B^{m+1}||_1$.
Obtain $\beta_t^{(m)}$ from (12) with $l = m+1,\ t = 2, 3, \dots, q, m_2+1, m_3+1, \cdots, r$ where r is the lowest value such that condition (29) is verified, computing the needed bounds b_j for $||B^j||_1$ using (20).
$\beta_{min}^{(m)} = \min\{\beta_t^{(m)},\ t = 2, 3, \dots, q, m_2+1, m_3+1, \cdots, r\}$.
$s^{(k)} = s_0^{(k)} = \max\left\{0, \lceil 1/2 \log_2(\beta_{min}^{(m)}/\Theta_m)\rceil\right\}$.
if $s_0^{(k)} > 0$ **then**
if ($m \leqslant 6$ AND $s_0^{(k)} \leqslant 2$) OR ($m < m_{M-1}$ AND $s_0^{(k)} = 1$) **then**
if (37) is verified with $s = 0$ **then**
$s^{(k)} = 0$
else if (39) is not verified with $s = 0$ **then**
if (38) is verified with $s = 0$ **then**
$s^{(k)} = 0$
end if
end if
else if $m \geqslant m_{M-1}$ **then**
if (37) is verified with $s_0^{(k)} - 1$ **then**
$s^{(k)} = s_0^{(k)} - 1$
else if (39) is not verified with $s_0^{(k)} - 1$ **then**
if (38) is verified with $s_0^{(k)} - 1$ **then**
$s^{(k)} = s_0^{(k)} - 1$
end if
end if
end if
end if
if $s^{(k)} = 0$ **then** ▷ If $s^{(k)} = 0$ order m is selected directly
return $s^{(k)}$, m, q
end if
end while
if $s^{(M-1)} \geqslant s^{(M)} + 1$ **then** ▷ Select the combination with the lowest cost
return $s = s^{(M)},\ m = m_M,\ q = q_M$

```
else
    return s = s^(M-1), m = m_{M-1}, q = q_{M-1}        ▷ q_{M-1} = q_M
end if
```

6. Rounding error analysis

The analysis of rounding errors in Algorithm 1 is based on the results given in [7] and [1, p. 293-294]. In [7] it was justified that rounding errors in the evaluation of $P_m(B)$ are balanced with rounding errors in the double angle phase. If we define $C_i = \cos\left(2^{i-s}A\right)$ and $\hat{C}_i = fl\left(2\hat{C}_{i-1}^2 - I\right)$, where fl is the floating operator [17, p. 61], and we assume that$\|E_i\|_1 \leqslant 0.05||\hat{C}_i||_1$, then rounding error in Step 6 of Algorithm 1 verifies

$$\begin{aligned} ||E_i||_1 = ||C_i - \hat{C}_i||_1 \leqslant (4.1)^i ||E_0||_1 ||C_0||_1 ||C_1||_1 \cdots ||C_{i-1}||_1 + \\ \tilde{\gamma}_{n+1} \sum_{j=0}^{i-1} (4.1)^{i-j-1} \left(2.21||C_j||_1^2 + 1\right) ||C_{j+1}||_1 \cdots ||C_{i-1}||_1, \end{aligned}$$

where $\tilde{\gamma}_{n+1}$ is defined by $\tilde{\gamma}_{n+1} = \frac{c(n+1)u}{1-c(n+1)u}$ [1, p. 332]. Hence the error $||E_i||_1$ fundamentally depends on the norms of the matrices $||C_0||_1$ and $||E_0||_1$. From (11), values of m_k and s are chosen such that $\|E_0\|_1 \leqslant u$. Since $||4^{-s}A^2||$ is not bounded with the new proposed scaling algorithm, it follows that $||C_0||$ is not bounded. Taking into account that the values of Θ_{m_k} increase with m_k, it follows that the values of the scaling parameter s with high values of m_k will be typically lower, giving higher values of $||4^{-s}A^2||$, and then $||C_0||$ will usually be higher when permitting the use of higher orders. Hence, despite the error balancing between the evaluation of $P_m(B)$ and the double angle phase, orders not much higher than the approximately optimal $m_M = 12$ should be used in the proposed scaling algorithm.

7. Numerical experiments

In this section we compare MATLAB implementation `costay` with functions `cosm` and `cosher`. `cosm` is a MATLAB implementation of Algorithm 5.1 proposed in [5] which uses Padé approximants of cosine function and it is available online at http://www.maths.manchester.ac.uk/ higham/mftoolbox. `cosher` is a MATLAB function based on Hermite series proposed in [6] and available at http://personales.upv.es/∼jorsasma/cosher.m. Similarly to costay, cosher also allows different maximum orders m_M for the Hermite approximation, recommending $m_M = 16$ for best performance in numerical tests, see [6]. All MATLAB implementations (R2011a) were tested on an Intel Core 2 Duo processor at 2.52 GHz with 4 GB main memory. 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 $\cos(A)$ the exact solution. We used 101 of the 102 test matrices from [7]. Test matrix 61 was removed due to very large rounding errors produced when computing the powers of that matrix, making all the tested algorithms fail. The "exact" matrix cosine was calculated analytically when possible, and otherwise using MATLAB's Symbolic Math Toolbox with high precision arithmetic.

In the tests we did not use any preprocessing/postprocessing in the implemented algorithms. Analogously to the experiments in [16], we found that turning on preprocessing provided similar results to those presented in this section without preprocessing.

Figures 1a and 1b show the comparisons costay-cosm and costay-cosher for maximum orders $m_M \in \{9, 12, 16, 20\}$ in both costay and cosher. The first three rows show the percentages of times that the relative error of the first function is lower, equal or greater than the relative error of the second function. The fourth row shows the ratio of matrix products needed for computing the matrix cosine for over all the test matrices with costay divided by the number of matrix products for the compared function. The asymptotic cost in terms of matrix products for solving the multiple right-hand side linear system appearing in Padé algorithm has been taken 4/3, see [9].

As shown in Figures 1a and 1b, function costay presented more accurate results than cosm and cosher in the significant majority of tests, especially for $m_M = 16$ (in 91.09% of cases with respect to cosm function and 44.55% with respect to cosher). Moreover, costay has lower computational costs than the functions cosm and cosher. Note that the cost gains of using $m_M = 12$ and $m_M = 16$ are the same. Table 3 shows that both orders provide the same cost in almost all cases in the first stage of the scaling algorithm providing justification for that. However, $\Theta_{16} > \Theta_{12} > \Theta_9$ and then the values of s with $m_M = 16$ are lower in many cases than those with $m_M = 12$ and $m_M = 9$, see Table 3, reducing the number of double angle steps in Algorithm `costay`. Numerical results show that $m_M = 16$ provided the highest accuracy with similar cost to $m_M = 12$, being the best choice for m_M in tests.

We have observed that in a 94.50% of cases the final value of the scaling parameter is directly $s_0^{(k)}$ given by (36), and that bounds (37) and (38) reduce the scaling parameter in the majority of cases where $\beta_{min}^{(m_k)} 4^{-(s_0^{(k)}-1)}/\Theta_{m_k}$ is slightly greater than 1. There were only two matrices where $\beta_{min}^{(m_k)} 4^{-(s_0^{(k)}-1)}/\Theta_{m_k}$ was not approximately 1, taking values 2.72 with order $m_k = 12$, and 5.86 with $m_k = 4$. With respect to the selection of the number of terms N to be considered in bounds (37) and (38) we have observed that taking $N = q_k + 2$, the greatest value of the remainder (40) using the values of $\beta_{min}^{(m_k)}$ obtained in numerical tests with $m_M = 12, 16, 20$, was $1.3 \cdot 10^{-21}$, thus confirming that the selection $N = q_k + 2$ is enough in practice for all orders $m_k \leqslant 20$.

m_M :	9	12	16	20
L	59.41	83.17	91.09	77.23
E	0	0	0	0
G	40.59	16.83	8.91	22.77
R	0.84	0.83	0.83	0.84

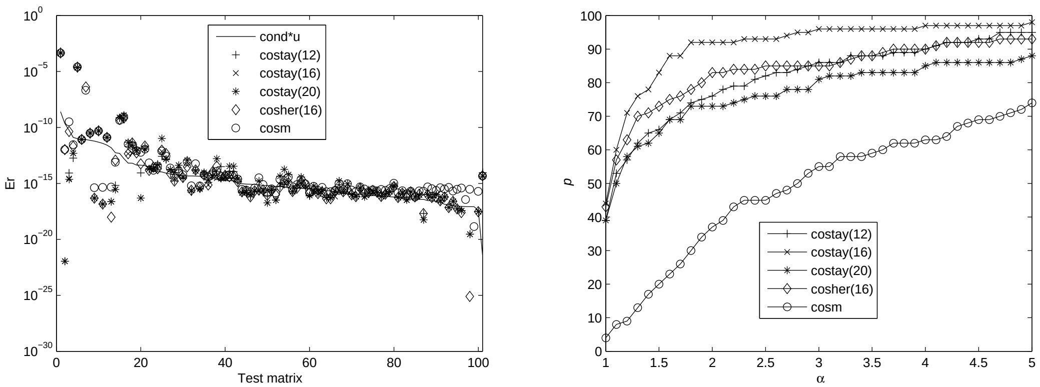
(a) Comparative `costay-cosm`.

m_M :	9	12	16	20
L	55.45	46.53	44.55	44.55
E	30.69	28.71	20.59	19.80
G	13.86	24.76	34.66	35.65
R	0.88	0.90	0.90	0.92

(b) Comparative `costay-cosher`.

Figure 1: Comparatives `costay-cosm` and `costay-cosher`. The first three rows show the percentage of times that relative error of `costay` is lower (L), equal (E) or greater (G) than relative error of `cosm` or `cosher`. The last row shows the ratio (R) of cost in terms of matrix products between `costay` divided by the cost of `cosm` in 1a, and `cosher` in 1b.

To test the numerical stability of functions we plotted the normwise relative errors of functions `cosm`, `cosher` and `costay` for $m_M = 12, 16, 20$. Figure 2a shows the relative errors of all implementations, and a solid line that represents the unit roundoff multiplied by the relative condition number of the cosine function at X [1, p. 55]. Relative condition number was computed using the MATLAB function `funm_condest1` from the Matrix Function Toolbox [1, Appendix D] (http://www.ma.man.ac.uk/ ~higham/mftoolbox). For a method to perform in a backward and forward stable manner, its error should lie not far above this line on the graph [16, p. 1188]. Figure 2a shows that in general the functions performed in a numerically stable way apart from some exceptions.



(a) Normwise relative error. (b) Performance profile.

Figure 2: Normwise relative errors and perfomance profile of `cosm`, `cosher(16)` and `costay` for $m_M = 12, 16, 20$.

Figure 2b shows the performances [18] of the functions compared, where α coordinate varies between 1 and 5 in steps equal to 0.1, and 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, where probabilities are defined over all matrices, showing that the most accurate function is `costay` with $m_M = 16$ followed by `cosher` with $m_M = 16$.

8. Conclusions

In this work an accurate Taylor algorithm to compute matrix cosine is proposed. The new algorithm uses a scaling technique based on the double angle formula and sharp bounds for the forward absolute error, and the Horner and Paterson-Stockmeyer's method for computing the Taylor approximation. A MATLAB implementation of this algorithm has been compared with MATLAB function `cosher`, based on Hermites series [6], and the MATLAB function `cosm`, based on the Padé algorithm given in [5]. Numerical experiments show that the new algorithm has lower computational costs and higher accuracy than both functions `cosher` and `cosm` in the majority of test matrices. The new proposed Taylor algorithm provided the highest accuracy and lowest cost when maximum order $m_M = 16$ was used in tests, and this maximum order is therefore recommended.

References

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

[2] S. Serbin, Rational approximations of trigonometric matrices with application to second-order systems of differential equations, Appl. Math. Comput. 5 (1) (1979) 75–92.

[3] S. M. Serbin, S. A. Blalock, An algorithm for computing the matrix cosine, SIAM J. Sci. Statist. Comput. 1 (2) (1980) 198–204.

[4] N. J. Higham, M. I. Smith, Computing the matrix cosine, Numer. Algorithms 34 (2003) 13–26.

[5] G. I. Hargreaves, N. J. Higham, Efficient algorithms for the matrix cosine and sine, Numer. Algorithms 40 (2005) 383–400.

[6] 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.

[7] E. Defez, J. Sastre, J. J. Ibáñez, P. A. Ruiz, Solving engineering models using matrix functions, in: Proc. Int. Conf. on Math. Model. in Engineering & Human Behaviour 2011, Sept. 6-9, pp. 1-17, 2011.

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

[9] J. Sastre, J. J. Ibáñez, E. Defez, P. A. Ruiz, Accurate matrix exponential computation to solve coupled differential, Math. Comput. Model. 54 (2011) 1835–1840.

[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] J. Sastre, J. J. Ibáñez, E. Defez, P. A. Ruiz, Efficient orthogonal matrix polynomial based method for computing matrix exponential, Appl. Math. Comput. 217 (2011) 6451–6463.

[12] 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.

[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] 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] http://personales.upv.es/∼jorsasma/costay.m.

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

[17] G. H. Golub, C. V. Loan, Matrix Computations, 3rd Edition, Johns Hopkins Studies in Mathematical Sciences, The Johns Hopkins University Press, 1996.

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