An efficient and accurate algorithm for computing the matrix cosine based on New Hermite approximations[☆]

Emilio Defez[⋆], Javier Ibáñez[♮], Jesús Peinado[‡], Jorge Sastre[†], Pedro Alonso-Jordá[‡]

⋆ Instituto de Matemática Multidisciplinar.
♮ Instituto de Instrumentación para Imagen Molecular.
‡ Departmento de Sistemas Informáticos y Computación.
† Instituto de Telecomunicaciones y Aplicaciones Multimedia.
Universitat Politècnica de València, Camino de Vera s/n, 46022, Valencia, Spain.
edefez@imm.upv.es, jjibanez@dsic.upv.es, jpeinado@dsic.upv.es, jsastrem@upv.es, palonso@upv.es

Abstract

In this work we introduce new rational-polynomial Hermite matrix expansions which allows us to obtain a new accurate and efficient method for computing the matrix cosine. This method is compared with other state-of-the-art methods for computing the matrix cosine, including a method based on Padé approximants, showing a far superior efficiency, and higher accuracy. The algorithm implemented on the basis of this method can also be executed either in one or two NVIDIA GPUs, which demonstrates its great computational capacity.

Keywords: matrix cosine, scaling and squaring method, Hermite series, backward error, parallel implementation, GPUs, CUDA.

1. Introduction and Notation

The computation of matrix trigonometric functions has received remarkable attention in the last decades due to its usefulness in the solution of

[☆]This work has been partially supported by Spanish Ministerio de Economía y Competitividad and European Regional Development Fund (ERDF) grants TIN2014-59294-P, and TIN2017-89314-P.

systems of second order linear differential equations. Several state-of-the-art algorithms have been provided for computing these matrix functions, see for instance [1, 2, 3, 4, 5, 6, 7] and the references therein.

Matrix polynomials are a very active area of research [8, 9, 10, 11], becoming more and more relevant in the last decades. Orthogonal matrix polynomials and, in particular, Hermite matrix polynomials, introduced and studied in [12, 13], have received considerable attention for its application in the solution of matrix differential equations [14]. Series of Hermite matrix polynomials have been studied for its application in the approximation of different matrix functions: exponential matrix [15], matrix cosine [6, 16, 17], and the hyperbolic matrix sine and cosine [18, 19].

In the scalar case, Hermite polynomials $H_n(x)$ are widely used in quantum mechanics, mathematical physics, nucleon physics and quantum optics. Recently, new formulas for series of Hermite polynomials

$$\sum_{n\geq 0} \frac{H_{2n+l}(x)}{n!} t^n, \quad \text{with} \quad l = 1, 2, 3, \ldots,$$

have been obtained in [20], having been these formulas applied, e.g. in the theory of quantum optics. The generalization of this class of formulas for Hermite matrix polynomials $H_n(x, A)$ can be found in [21].

Hermite matrix polynomial $H_n(x, A)$ has the following generating function [12]:

$$e^{xt\sqrt{2A}} = e^{t^2} \sum_{n\geq 0} \frac{H_n(x, A)}{n!} t^n,$$

from which the following expressions are derived:

$$\left.\begin{array}{rcl} \cos\left(xt\sqrt{2A}\right) & = & e^{-t^2} \displaystyle\sum_{n\geq 0} \frac{(-1)^n H_{2n}(x, A)}{(2n)!} t^{2n} \\ \sin\left(xt\sqrt{2A}\right) & = & e^{-t^2} \displaystyle\sum_{n\geq 0} \frac{(-1)^n H_{2n+1}(x, A)}{(2n+1)!} t^{2n+1} \end{array}\right\}, \; x \in \mathbb{R}, |t| < \infty. \tag{1}$$

for the matrix sine and cosine.

In this paper we derive a functional form for the new Hermite matrix polynomial series as:

$$\sum_{n\geq 0} \frac{(-1)^n H_{2n+1}(x, A)}{(2n)!} t^{2n} := \mathcal{A}(x, t; A), \tag{2}$$

and

$$\sum_{n\geq 0} \frac{(-1)^n H_{2n+3}(x,A)}{(2n+1)!} t^{2n+1} := \mathcal{B}(x,t;A), \tag{3}$$

which are generalizations of formulas (1). We will use formulas (2) and (3) to obtain a new rational expansion in Hermite matrix polynomials of the matrix cosine.

The organization of this paper is as follows: In Section 2 we present a proof of formulas (2) and (3). Section 3 deals with new rational-polynomial Hermite matrix expansions for the matrix cosine. The proposed algorithm and its MATLAB implementation are both presented in Section 4. The implementation of the accelerated algorithm is described in Section 5 being the numerical results presented in Section 6. Finally, the conclusions are given in Section 7.

Throughout this paper, we denote by $\mathbb{C}^{r\times r}$ the set of all the complex square matrices of size r. With Θ and I we denote the zero and the identity matrices in $\mathbb{C}^{r\times r}$, respectively. If $A \in \mathbb{C}^{r\times r}$, we denote by $\sigma(A)$ the set of all the eigenvalues of A. For a real number x, $\lfloor x \rfloor$ denotes the lowest integer not less than x and $\lceil x \rceil$ denotes the highest integer not exceeding x.

If $f(z)$ and $g(z)$ are holomorphic functions in an open set Ω of the complex plane, and if $\sigma(A) \subset \Omega$, we denote by $f(A)$ and $g(A)$ the image by the Riesz-Dunford functional calculus of the functions $f(z)$ and $g(z)$, respectively, acting on matrix A. Also, $f(A)g(A) = g(A)f(A)$ [22, p. 558]. We say that matrix A is positive stable if $Re(z) > 0$ for every eigenvalue $z \in \sigma(A)$. In this case, let us denote $\sqrt{A} = A^{1/2} = \exp\left(\frac{1}{2}\log(A)\right)$ the image of function $z^{1/2} = \exp\left(\frac{1}{2}\log(z)\right)$ by the Riesz-Dunford functional calculus acting on matrix A, where $\log(z)$ denotes the principal branch of the complex logarithm.

In this paper, we use consistent matrix norms, in particular $\|A\|_2$ is the 2-norm. In tests we use the 1-norm of a matrix $A \in \mathbb{C}^{r\times r}$ defined by $\|A\|_1 = \sup_{x\neq 0} \frac{\|Ax\|_1}{\|x\|_1}$, where $\|\cdot\|_1$ denotes the vector 1-norm defined as $\|y\|_1 = |y_1| + \cdots + |y_r|$, $y \in \mathbb{C}^r$, see [23, Chapter 2]. For a positive stable matrix $A \in \mathbb{C}^{r\times r}$ the nth Hermite matrix polynomial is defined in [12] by:

$$H_n(x,A) = n! \sum_{k=0}^{\lfloor \frac{n}{2} \rfloor} \frac{(-1)^k \left(\sqrt{2A}\right)^{n-2k}}{k!(n-2k)!} x^{n-2k}, \tag{4}$$

which satisfies the three-term matrix recurrence:

$$\left.\begin{array}{l} H_m(x,A) = x\sqrt{2A}H_{m-1}(x,A) - 2(m-1)H_{m-2}(x,A)\ ,\ m \geq 1, \\ H_{-1}(x,A) = \Theta\ ,\ H_0(x,A) = I\ . \end{array}\right\} \quad (5)$$

The following upper bound of Hermite matrix polynomials was demonstrated in [24]:

$$\left.\begin{array}{c} \|H_{2n}(x,A)\|_2 \leq g_n(x)\ ,\ n \geq 1 \\ \|H_{2n+1}(x,A)\|_2 \leq |x| \left\| \left(\frac{A}{2}\right)^{-\frac{1}{2}} \right\|_2 \frac{2g_n(x)}{n+1}\ ,\ n \geq 0 \end{array}\right\} \quad (6)$$

where function $g_n(x)$ is defined as

$$g_n(x) = \frac{(2n+1)!2^{2n}}{n!} \exp\left(\frac{5}{2}\|A\|_2 x^2\right), n \geq 0. \quad (7)$$

2. A proof of the new formulas (2) and (3)

We calculate the exact value of the series of matrices $\mathcal{A}(x,t;A)$ and $\mathcal{B}(x,t;A)$ defined by (2) and (3). Firstly, we prove that both matrix series are convergent. Taking into account (6) one gets that

$$\left\| \frac{(-1)^n H_{2n+1}(x,A)}{(2n)!} t^{2n} \right\| \leq |x| \left\| \left(\frac{A}{2}\right)^{-\frac{1}{2}} \right\|_2 \frac{2g_n(x)}{(n+1)(2n)!} |t|^{2n}.$$

Using (7) we get that $\sum_{n\geq 0} \frac{g_n(x)}{(n+1)(2n)!}|t|^{2n}$ is convergent for $|t| < \infty$ so the matrix series $\mathcal{A}(x,t;A)$ defined by (2) is convergent in any compact real interval. Analogously and taking into account (6) again, we have

$$\left\| \frac{(-1)^n H_{2n+3}(x,A)}{(2n+1)!} t^{2n+1} \right\| \leq |x| \left\| \left(\frac{A}{2}\right)^{-\frac{1}{2}} \right\|_2 \frac{2g_{n+1}(x)}{(n+2)(2n+1)!} |t|^{2n+1}.$$

Using (7) again we have that $\sum_{n\geq 0} \frac{g_{n+1}(x)}{(n+2)(2n+1)!}|t|^{2n+1}$ is convergent for $|t| < \infty$ so the matrix series $\mathcal{B}(x,t;A)$ defined by (3) is convergent in any

compact real interval. Using now (2), (5) and the fact that $H_1(x,A)=\sqrt{2A}x$, we can write:

$$\begin{aligned}
\mathcal{A}(x,t;A) &= \left(x\sqrt{2A}\right)\sum_{n\geq 0}\frac{(-1)^n H_{2n}(x,A)}{(2n)!}t^{2n}-4\sum_{n\geq 1}\frac{(-1)^n n H_{2n-1}(x,A)}{(2n)!}t^{2n}\\
&= H_1(x,A)e^{t^2}\cos\left(xt\sqrt{2A}\right)-2t\sum_{n\geq 1}\frac{(-1)^n H_{2n-1}(x,A)}{(2n-1)!}t^{2n-1}\\
&= H_1(x,A)e^{t^2}\cos\left(xt\sqrt{2A}\right)-2t\sum_{n\geq 0}\frac{(-1)^{n+1} H_{2n+1}(x,A)}{(2n+1)!}t^{2n+1}\\
&= H_1(x,A)e^{t^2}\cos\left(xt\sqrt{2A}\right)+2te^{t^2}\sin\left(xt\sqrt{2A}\right). \qquad (8)
\end{aligned}$$

Working similarly we can write:

$$\begin{aligned}
\mathcal{B}(x,t;A) &= x\sqrt{2A}\sum_{n\geq 0}\frac{(-1)^n H_{2n+2}(x,A)}{(2n+1)!}t^{2n+1}-2\sum_{n\geq 0}\frac{(-1)^n(2n+2)H_{2n+1}(x,A)}{(2n+1)!}t^{2n+1}\\
&= x\sqrt{2A}\left(x\sqrt{2A}\sum_{n\geq 0}\frac{(-1)^n H_{2n+1}(x,A)}{(2n+1)!}t^{2n+1}-2\sum_{n\geq 0}\frac{(-1)^n(2n+1)H_{2n}(x,A)}{(2n+1)!}t^{2n+1}\right)\\
&- 2\left(\sum_{n\geq 0}\frac{(-1)^n(2n+1)H_{2n+1}(x,A)}{(2n+1)!}t^{2n+1}+\sum_{n\geq 0}\frac{(-1)^n H_{2n+1}(x,A)}{(2n+1)!}t^{2n+1}\right)\\
&= x\sqrt{2A}\left(x\sqrt{2A}e^{t^2}\sin\left(xt\sqrt{2A}\right)-2t\sum_{n\geq 0}\frac{(-1)^n H_{2n}(x,A)}{(2n)!}t^{2n}\right)\\
&- 2\left(\sum_{n\geq 0}\frac{(-1)^n H_{2n+1}(x,A)}{(2n)!}t^{2n+1}+e^{t^2}\sin\left(xt\sqrt{2A}\right)\right)\\
&= \left(2x^2A-2I\right)e^{t^2}\sin\left(xt\sqrt{2A}\right)-2xt\sqrt{2A}\,e^{t^2}\cos\left(xt\sqrt{2A}\right)\\
&- 2\sum_{n\geq 0}\frac{(-1)^n H_{2n+1}(x,A)}{(2n)!}t^{2n+1}\\
&= e^{t^2}\left[\left(2x^2A-2I\right)\sin\left(xt\sqrt{2A}\right)-2xt\sqrt{2A}\cos\left(xt\sqrt{2A}\right)\right]-2t\mathcal{A}(x,t;A).
\end{aligned}$$

Taking into account (8) one gets

$$\mathcal{B}(x,t;A)=\left(2x^2A-2I-4t^2I\right)e^{t^2}\sin\left(xt\sqrt{2A}\right)-4xt\sqrt{2A}e^{t^2}\cos\left(xt\sqrt{2A}\right).$$

By using (4), we have that $H_1(x,A)=\sqrt{2A}x, H_2(x,A)=2x^2A-2I$ so we can rewrite the last expression of $\mathcal{B}(x,t;A)$ in the following form:

$$\mathcal{B}(x,t;A):=e^{t^2}\left[\left(H_2(x,A)-4t^2I\right)\sin\left(xt\sqrt{2A}\right)-4tH_1(x,A)\cos\left(xt\sqrt{2A}\right)\right].$$

Summarizing, we establish the following result:

Lemma 2.1. *Let $A\in\mathbb{C}^{r\times r}$ be a positive stable matrix, $x\in\mathbb{R},|t|<+\infty$, then*

$$\begin{aligned}\sum_{n\geq 0}\frac{(-1)^nH_{2n+1}(x,A)}{(2n)!}t^{2n} &= e^{t^2}\left[H_1(x,A)\cos\left(xt\sqrt{2A}\right)+2t\sin\left(xt\sqrt{2A}\right)\right],\\ \sum_{n\geq 0}\frac{(-1)^nH_{2n+3}(x,A)}{(2n+1)!}t^{2n+1} &= e^{t^2}\left[\left(H_2(x,A)-4t^2I\right)\sin\left(xt\sqrt{2A}\right)-4tH_1(x,A)\cos\left(xt\sqrt{2A}\right)\right].\end{aligned}\tag{9}$$

Consequently, the following corollary follows:

Corollary 2.1. *Let $\{H_n(x)\}_{n\geq 0}$ be the sequence of Hermite polynomials, then*

$$\begin{aligned}\sum_{n\geq 0}\frac{(-1)^nH_{2n+1}(x)}{(2n)!}t^{2n} &= e^{t^2}\left[H_1(x)\cos(2xt)+2t\sin(2xt)\right],\\ \sum_{n\geq 0}\frac{(-1)^nH_{2n+3}(x)}{(2n+1)!}t^{2n+1} &= e^{t^2}\left[(H_2(x)-4t^2)\sin(2xt)-4tH_1(x)\cos(2xt)\right],\end{aligned}$$

for $x\in\mathbb{R},|t|<+\infty$.

Proof. The proof is a consequence of Lemma 2.1 since the Hermite matrix polynomial $H_n(x,A)$ coincides with the Hermite polynomial $H_n(x)$ taking $r=1$ and $A=2$.

3. On new rational-polynomial Hermite matrix expansions for the matrix cosine

Let $A\in\mathbb{C}^{r\times r}$ be a positive stable matrix, then the matrix polynomial $H_1(x,A)=\sqrt{2A}x$ is invertible if $x\neq 0$. Substituting $\sin\left(xt\sqrt{2A}\right)$, given

in (1), into the first expression of (9) we obtain a new rational expression for the matrix cosine in terms of Hermite matrix polynomials:

$$\begin{aligned}\cos\left(xt\sqrt{2A}\right) &= e^{-t^2}\left(\sum_{n\geq 0}\frac{(-1)^n H_{2n+1}(x,A)}{(2n)!}\left(1-\frac{2t^2}{2n+1}\right)t^{2n}\right)\left[H_1(x,A)\right]^{-1},\\ &\quad x\in\mathbb{R}\sim\{0\},\ |t|<+\infty.\end{aligned} \tag{10}$$

Similarly, replacing $\cos\left(xt\sqrt{2A}\right)$, given by (1), into the second expression of (9), the expression obtained is reduced to that given in (1). On the other hand, replacing the expression of $\sin\left(xt\sqrt{2A}\right)$ given in (1) into the second expression of (9), we obtain another new rational expression for the matrix cosine in terms of Hermite matrix polynomials:

$$\begin{aligned}\cos\left(xt\sqrt{2A}\right) &= \frac{-e^{-t^2}}{4}\left[\sum_{n\geq 0}\frac{(-1)^n H_{2n+3}(x,A)}{(2n+1)!}t^{2n}\right.\\ &- \left.\left(H_2(x,A)-4t^2 I\right)\star\left(\sum_{n\geq 0}\frac{(-1)^n H_{2n+1}(x,A)}{(2n+1)!}t^{2n}\right)\right]\left[H_1(x,A)\right]^{-1},\\ &\quad x\in\mathbb{R}\sim\{0\},\ |t|<+\infty.\end{aligned} \tag{11}$$

Comparing (10) with (11), we observe that there exists always one matrix product more when we evaluate (11)—the matrix product remarked by symbol "$\star$". Due to the importance of reducing the number of matrix products, we will focus mainly on the expansion (10).

Replacing matrix A by $A^2/2$ in (10), we can avoid the square roots of matrices. In addition, if $x\neq 0$ (4), it follows that

$$\begin{aligned}\left[H_1\left(x,\frac{1}{2}A^2\right)\right]^{-1}H_{2n+1}\left(x,\frac{1}{2}A^2\right) &= (Ax)^{-1}H_{2n+1}\left(x,\frac{1}{2}A^2\right)\\ &= (2n+1)!\sum_{k=0}^{n}\frac{(-1)^k x^{2(n-k)}A^{2(n-k)}}{k!(2(n-k)+1)!}\\ &= \widetilde{H}_{2n+1}\left(x,\frac{1}{2}A^2\right),\end{aligned} \tag{12}$$

so the right side of (12) is still defined in case matrix A is singular. In this way, we can re-write the relation (10) in terms of the matrix polynomial $\widetilde{H}_{2n+1}\left(x,\frac{1}{2}A^2\right)$ and, taking $x=\lambda$, $\lambda\neq 0$, and $t=1/\lambda$, we obtain

$$\cos(A) = e^{-\frac{1}{\lambda^2}} \sum_{n\geq 0} \frac{(-1)^n \widetilde{H}_{2n+1}\left(\lambda, \frac{1}{2}A^2\right)}{(2n)!\lambda^{2n}} \left(1 - \frac{2}{(2n+1)\lambda^2}\right). \quad (13)$$

Note that expansion (13) is really a polynomial series in matrix A. Truncating the given series (13) until order m, we obtain the approximation $C_m(\lambda, A) \approx \cos(A)$ defined by

$$C_m(\lambda, A) = e^{-\frac{1}{\lambda^2}} \sum_{n=0}^{m} \frac{(-1)^n \widetilde{H}_{2n+1}\left(\lambda, \frac{1}{2}A^2\right)}{(2n)!\lambda^{2n}} \left(1 - \frac{2}{(2n+1)\lambda^2}\right), 0 < \lambda < +\infty,$$

for any matrix $A \in \mathbb{C}^{r\times r}$.

Working analogously to the proof of equation (3.3) of [19], we have

$$\left\| \widetilde{H}_{2n+1}\left(x, \frac{1}{2}A^2\right) \right\|_2 \leq (2n+1)! \frac{e \sinh\left(|x| \left\|A^2\right\|_2^{1/2}\right)}{|x| \left\|A^2\right\|_2^{1/2}}, x \neq 0, \quad (14)$$

so we can obtain the following expression for the approximation error:

$$\|\cos(A) - C_m(\lambda, A)\|_2 \leq e^{-\frac{1}{\lambda^2}} \sum_{n\geq m+1} \frac{\left\| \widetilde{H}_{2n+1}\left(\lambda, \frac{1}{2}A^2\right) \right\|_2}{(2n)!\lambda^{2n}} \left|1 - \frac{2}{(2n+1)\lambda^2}\right| \quad (15)$$

$$\leq \frac{e^{1-\frac{1}{\lambda^2}} \sinh\left(\lambda \left\|A^2\right\|_2^{1/2}\right)}{\lambda \left\|A^2\right\|_2^{1/2}} \sum_{n\geq m+1} \frac{2n+1}{\lambda^{2n}} \left|1 - \frac{2}{(2n+1)\lambda^2}\right|.$$

Taking $\lambda > \sqrt{2}$ it follows that $\frac{2}{(2n+1)\lambda^2} < 1$, and one gets

$$\sum_{n\geq m+1} \frac{2n+1}{\lambda^{2n}} \left|1 - \frac{2}{(2n+1)\lambda^2}\right| < \sum_{n\geq m+1} \frac{2n+1}{\lambda^{2n}} = \frac{2 + (2m+3)(\lambda^2-1)}{\lambda^{2m}(\lambda^2-1)^2}.$$

Thus, from (15) we finally obtain:

$$\|\cos(A) - C_m(\lambda, A)\|_2 \leq \frac{e^{1-\frac{1}{\lambda^2}} \sinh\left(\lambda \left\|A^2\right\|_2^{1/2}\right) \left(2 + (2m+3)(\lambda^2-1)\right)}{\left\|A^2\right\|_2^{1/2} \lambda^{2m+1} (\lambda^2-1)^2}. \quad (16)$$

Table 1: Values of z_m and λ_m

m	2	4	6	9	12	16
z_m	$2.0612 \cdot 10^{-5}$	$8.1180 \cdot 10^{-3}$	$1.2996 \cdot 10^{-1}$	$1.3290 \cdot 10^{-1}$	5.2620	$1.7686 \cdot 10^{1}$
λ_m	1518.9764	118.9737	35.9520	17.9304	10.9977	8.3117

From expression (16) we can derive the optimal pair of values $(\lambda_m; z_m)$ such that

$$z_m = \max\left\{ z = \left\|A^2\right\|_2 ; \frac{e^{1-\frac{1}{\lambda^2}} \sinh\left(\lambda z^{1/2}\right)\left(2+(2m+3)(\lambda^2-1)\right)}{z^{1/2}\lambda^{2m+1}\left(\lambda^2-1\right)^2} \leq u \right\}, \tag{17}$$

where u is the unit roundoff in IEEE double precision arithmetic ($u = 2^{-53}$). For each m, the optimal values of z_m and λ_m have been obtained with a MATLAB script. This script iterates through two nested loops varying z and λ in increments of order 10^{-6}. Of those pairs (λ, z) that satisfy (17), we take the one that maximizes z_m. The values of z_m and λ_m obtained for $m \in \{2, 4, 6, 9, 12, 16\}$ are given in Table 1.

4. The proposed MATLAB implementations

The matrix cosine of a matrix $A \in \mathbb{C}^{n\times n}$ be computed with the expression

$$P_m(B) = \sum_{i=0}^{m} p_i B^i, \tag{18}$$

where $B = A^2$, and p_i is the coefficient polynomial of the Hermite expression (3). Since Hermite and Taylor series are accurate only near the origin, the algorithms that use these approximations must reduce the norm of matrix B by scaling the matrix. Then, once the cosine of the *scaled* matrix has been computed, the approximation of $\cos(A)$ is recovered by means of the double angle formula

$$\cos(2X) = 2\cos^2(X) - I. \tag{19}$$

Algorithm 1 shows a general algorithm for computing the matrix cosine based on Hermite 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 Hermite series.

1: Select adequate values of $m_k \in \{2,4,6,9,12,16\}$ and $s \in \mathbb{N}\cup\{0\}$ ▷ Phase I
2: $B = 4^{-s}A^2$
3: $C = P_{m_k}(B)$ ▷ Phase II: Compute Hermite approximation (18)
4: **for** $i = 1 : s$ **do** ▷ Phase III: Recovering $\cos(A)$
5: $\quad C = 2C^2 - I$
6: **end for**

In Phase I of Algorithm 1, the suitable values of m and s are calculated so that the Hermite of the scaled matrix is computed accurately and efficiently. Phase II consists of computing the approximations (3) (or (18)). The Hermite matrix polynomial approximation (18) can be computed with optimal cost by the Paterson-Stockmeyer's method [25] choosing m from the set

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

where the elements of $\mathbb{M}$ are denoted as $m_1, m_2, m_3, \dots$ The algorithm computes first the powers B^i, $2 \le i \le q$ 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, for $k \ge 1$, both values giving the same cost in terms of matrix products. Therefore, equation (18) can be computed efficiently as

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

Taking into account Table 4.1 from [26] and that $B = A^2$, the computational cost in terms of matrix products of (18) is $\Pi m_k = k+1$ (see Table 3).

Finally, in Phase III, the approximation of $\cos(A)$ is recovered by using the double angle formula (19).

Table 2: Values $\bar{m}_k$, $\tilde{m}_k$, and $f_{\max}$.

	$m_1 = 2$	$m_2 = 4$	$m_3 = 6$	$m_4 = 9$	$m_5 = 12$	$m_6 = 16$
$\bar{m}_k$	1	2	3	5	7	11
$\tilde{m}_k$	1	2	4	10	13	17
$f_{m_k}(\max)$	0	0	$1.9 \cdot 10^{-17}$	$6.0 \cdot 10^{-19}$	$1.4 \cdot 10^{-26}$	$1.3 \cdot 10^{-35}$

The computation of the scaling factor s and the order of Hermite approximation m_k (Phase II) is based on the error analysis that we present below. The following theorem is used in the study:

Theorem 1 ([27]). *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* $1 \leqslant t \leqslant l$. *If* t_0 *is the multiple of* t *such that* $l \leqslant t_0 \leqslant l + t - 1$ *and*

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

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

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

If $\cos(A)$ is calculated from the Taylor series, then the absolute forward error of the Hermite approximation (18) of $\cos(A)$, denoted by E_f, can be computed as

$$E_f = \|\cos(A) - P_{m_k}(B)\| = \left\| \sum_{i \geqslant \bar{m}_k} f_{m_k,i} B^i \right\|,$$

where the values of $\bar{m}_k$, for each $m_k \in \{2, 4, 6, 9, 12, 16\}$, are those appearing in Table 2. We have empirically verified that by ignoring the coefficients whose absolute difference is lower than u, the results of efficiency are far superior to the state-of-the-art Padé and Taylor algorithms [4, 5], with also an excellent accuracy. Hence, we have applied the following approximation:

$$E_f = \|\cos(A) - P_{m_k}(B)\| \cong \left\| \sum_{i \geqslant \tilde{m}_k} f_{m_k,i} B^i \right\|,$$

where the values $\tilde{m}_k$ given in Table 2 are the first values that are accounted for. Values $f_{m_k}(\max)$ in Table 2 correspond to the maximum absolute values of the coefficients that have been neglected for each order. For each

Table 3: Products Π_{m_k} and values Θ_{m_k} corresponding to m_k.

	$m_1 = 2$	$m_2 = 4$	$m_3 = 6$	$m_4 = 9$	$m_5 = 12$	$m_6 = 16$
Π_{m_k}	2	3	4	5	6	7
Θ_{m_k}	$3.7247 \cdot 10^{-5}$	$1.1723 \cdot 10^{-2}$	$1.7002 \cdot 10^{-1}$	1.6237	6.1627	$2.0113 \cdot 10^{1}$

polynomial order, the values $f_{m_k}(\max)$ shown in Table 2 correspond to the maximum absolute values of the coefficients that have been discarded.

If we define $f_{\tilde{m}_k}(x) = \sum_{i \geqslant \tilde{m}_k} f_{m_k,i} x^i$ and $\tilde{f}_{\tilde{m}_k}(x) = \sum_{i \geq \tilde{m}_k} |f_{m_k,i}|\, x^i$, when Theorem 1 is applied we have

$$E_f = \|f_{\tilde{m}_k}(B)\| \leqslant \tilde{f}_{\tilde{m}_k}(\beta_t),$$

for every t, $1 \leq t \leq \tilde{m}_k$. Let Θ_{m_k} be

$$\Theta_{m_k} = \max\left\{\theta \geqslant 0 : \tilde{f}_{\tilde{m}_k}(\theta) = \sum_{i \geqslant \tilde{m}_k} |f_{m_k,i}|\, \theta^i \leqslant u\right\}, \tag{20}$$

where $u = 2^{-53}$ is the unit roundoff in double precision floating-point arithmetic. Using MATLAB (R2017b) Symbolic Math Toolbox with 200 series terms and a zero finder, we obtained the values Θ_{m_k} that verify (20). These values (Table 3) have similar magnitude to the values of Θ_{m_k} of Table 2 of [6].

The optimal values m_k and s of Algorithm 1 (Phase I) are obtained from the values of β_t of Theorem 1 and from the values Θ_{m_k} of Table 3. A complete study of this question was developed by the authors in [5, Subsection 2.3], and it is reproduced below. Let be

$$\beta_{\min}^{(\tilde{m}_k)} = \min_{1 \leqslant t \leqslant \tilde{m}_k} \{\beta_t\},$$

if there exists a value $m_k \leq 16$ such that $\beta_{\min}^{(\tilde{m}_k)} \leq \Theta_{m_k}$, then the forward error E_f is lower than u. In this case, we choose the lower order m_k such that $\beta_{\min}^{(\tilde{m}_k)} \leq \Theta_{m_k}$ and the scaling factor is $s = 0$. Otherwise, we choose the Hermite approximation of order 12 or 16 providing the lower cost, with

$$s = \max\left\{0, \left\lceil \frac{1}{2} \log\left(\frac{\beta_{\min}^{(\tilde{m}_k)}}{\Theta_{m_k})}\right)\right\rceil\right\}, \ m_k = 12 \text{ or } 16.$$

The following approximation is used for computing $\beta_{\min}^{(\tilde{m}_k)}$ (see (16) from [28]):

$$\beta_{\min}^{(\tilde{m}_k)} \approx \max\left\{d_{\tilde{m}_k}^{1/\tilde{m}_k},\ d_{\tilde{m}_k+1}^{1/(\tilde{m}_k+1)}\right\}$$

being $d_{\tilde{m}_k}$ and $d_{\tilde{m}_k+1}$ bounds of $\left\|B^{\tilde{m}_k}\right\|$ and $\left\|B^{\tilde{m}_k+1}\right\|$, respectively. The bounds d_l, $l = \tilde{m}_k, \tilde{m}_{k+1}$ can be computed using products of norms of matrix powers previously computed. This algorithm is analogous to Algorithm 2 from [5]. For example, for $m_k = 6$ the powers B^2 and B^3 must be computed, hence $\beta_{min}^{(3)}$ ($\tilde{m}_k = 3$) can be computed as

$$\beta_{min}^{(3)} = \max\left\{\left\|B^3\right\|^{1/3}, \min\left\{\left\|B^3\right\|\|B\|, \left\|B^2\right\|^2\right\}^{1/4}\right\}.$$

A MATLAB implementation of Algorithm 1, called `cosmtayher`, has been developed, using the values Θ_{m_k} of Table 3 to obtain the approximation degree m_k and the scaling factor s of matrix B, and using the Hermite matrix polynomial approximation obtained from the values λ_{m_k} of Table 1.

5. The implementation of the parallel algorithms

In addition, we have implemented an "*accelerated*" version of `cosmtayher`. This version uses NVIDIA GPUs (Graphics Processing Units) to accelerate the computations. The MATLAB version of `cosmtayher` is modified in such a way that the most time consuming operations are replaced by calls to a function which launches these operations to the GPU. The operations replaced are mainly those based on matrix products, e.g. the computation of polynomial factors and the evaluation of the polynomial carried out in Phase II of Algorithm 1, or the double angle formula computation (Phase III). This function, implemented and presented in [29], uses MATLAB C++ `mex` files and CUDA, a parallel computing platform and programming model developed by NVIDIA to implement general purpose applications for NVIDIA GPUs. Through a `mex` file we implemented a single function that is called at different points of the MATLAB script. At each point, a given operation to be executed by the `mex` function is selected according to a string argument. With this strategy we get persistence of data among different calls to the `mex` function and provide, in turn, easiness to the developer of the MATLAB function. There exist two `mex` files, to use one or two GPUs, respectively.

6. Numerical experiments

In this section, we compare the new MATLAB function developed in this paper, `cosmtayher`, with other two functions:

- `cosm`. Code based on the Padé rational approximation for the matrix cosine [4]. The MATLAB function `cosm` has an argument which allows us to compute $\cos(A)$ by means of just Padé approximants, or also using the real Schur and the complex Schur decompositions. In these tests we did not use the Schur decomposition since, as it was shown in the tests presented in [5], using the Schur decomposition in `cosmtay` provides higher efficiency than the Padé method with the Schur decomposition [4] and with similar accuracy. The MATLAB code can be found in: `http://github.com/sdrelton/cosm_sinm`.

- `cosmtay`. Code based on Taylor series to compute the matrix cosine [5]. This MATLAB function has an argument which allows to choose between two different methods to compute bounds of norms of matrix powers. One method uses norm estimation of matrix powers while the other one does not. For this function, we have used the same method as that used for `cosmtayher`. The MATLAB code can be found in: `http://personales.upv.es/jorsasma/software/cosmtay.m`

- `cosmtayher`. The new code presented in this contribution to compute the matrix cosine that is based on formula (3), available at `http://personales.upv.es/jorsasma/software/cosmtayher.m`.

6.1. Sequential tests

In this sequential tests we used MATLAB (R2017b) runing on an Apple Macintosh iMac 27" (iMac retina 5K 27" late 2015) with a quadcore INTEL i7-6700K 4Ghz processor and 16 Gb of RAM. The following tests were made using different matrices:

- Test 1: 100 diagonalizable 128×128 real matrices with 1-norms varying from 2.32 to 220.04. These matrices have the form $A = V^T D V$, where D is diagonal with real and complex eigenvalues, and V is an orthogonal matrix obtained as $V = H/\sqrt{128}$, where H is the Hadamard matrix. A Hadamard matrix of order n is a matrix of 1's and -1's whose columns are orthogonal, i.e. $H^T H = nI$. These matrices can be obtained by using the MATLAB function `hadamard`.

- Test 2: 100 non diagonalizable 128×128 real matrices whose 1-norms vary from 6.5 to 249.5. These matrices have the form $A = V^T J V$, where J is a Jordan matrix with real and complex eigenvalues. The eigenvalues have an algebraic multiplicity that varies between 1 and 3. Matrix V is orthogonal and has been obtained as $V = H/\sqrt{128}$, where H is a Hadamard matrix.

- Test 3: Sixteen matrices with dimensions lower than or equal to 128 from the Eigtool MATLAB package [30], forty three matrices from the matrix function literature with dimensions lower than or equal to 128, and 128×128 real matrices generated with the function `matrix` of the Matrix Computation Toolbox [31].

The *"exact"* matrix cosine was computed as $\cos(A) = V^T \cos(D) V$, for matrices of Test 1, and $\cos(A) = V^T \cos(J) V$, for matrices of Test 2 (see [26, pp. 10]), by using the MATLAB's Symbolic Math Toolbox with 256 decimal digit arithmetic in all the computations. For the other matrices we followed [7, Sec. 4.1], i.e. we used MATLAB symbolic versions of a scaled Padé rational approximation from [4] and a scaled Taylor Paterson-Stockmeyer approximation [5, pp. 67], 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, and checking that their relative difference was small enough. Similar results were obtained by using the method proposed in [32]. 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.

We show the accuracy and computational cost of `cosm`, `cosmtay`, and `cosmtayher`, for the three tests.

Table 4 shows the computational cost of each routine cast in terms of number of matrix products (M(routine)). The rest of the operations are considered negligible compared to matrix products, specially for big enough matrices. Since the computational cost of solving a linear system $AX = B$, $A, B \in \mathbb{R}^{n \times n}$, which appears in the code based on Padé approximations, is $8n^3/3$ flops and the cost of a matrix product AB is $2n^3$ [26, Table C.1, pp. 336], we assume that solving $AX = B$ is equivalent to perform 4/3 matrix products.

Table 4: Matrix products for the three tests using the three MATLAB functions.

	M(`cosmtayher`)	M(`cosmtay`)	M(`cosm`)
Test 1	671	948	1129
Test 2	681	964	1146
Test 3	399	558	675

Table 5: Relative error comparison between `cosmtayher` vs `cosm` (row 1) and `cosmtayher` vs `cosmtay` (row 2) for the three tests.

	Test 1	Test 2	Test 3
E(`cosmtayher`)<E(`cosm`)	92%	81%	77.97%
E(`cosmtayher`)<E(`cosmtay`)	53%	65%	69.49%

Table 5 shows a comparison of relative errors of the three functions. The table shows the percentage of matrices in which the relative error of `cosmtayher` is lower than the relative error `cosm` and `cosmtay`, respectively.

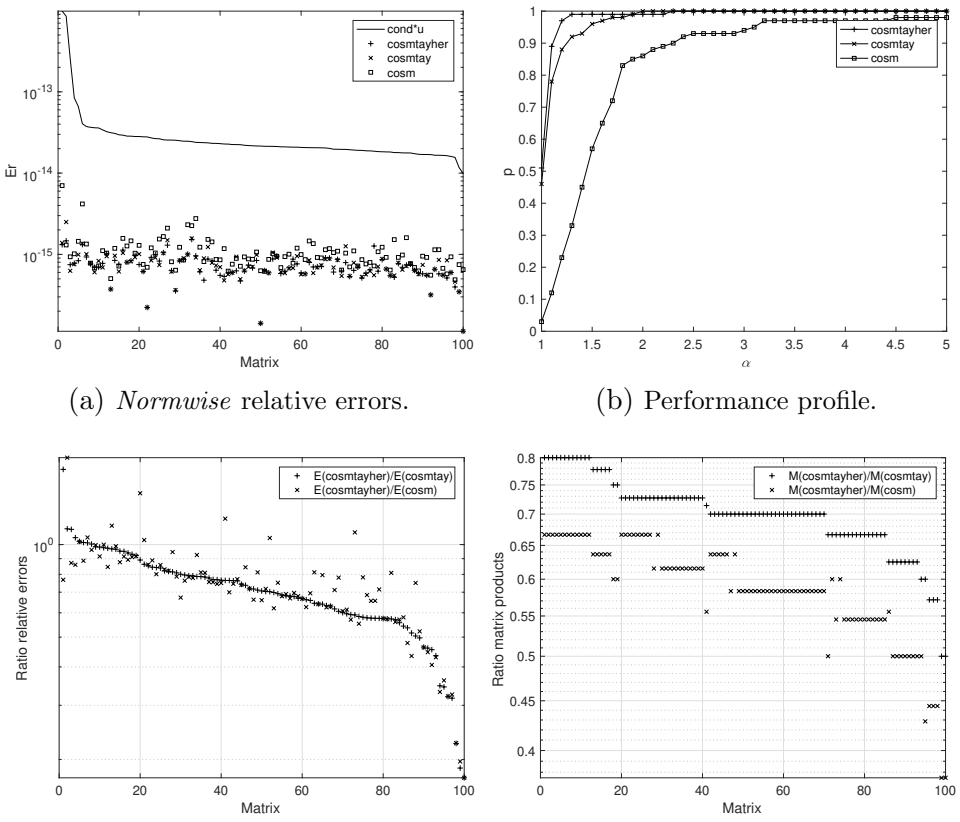
Moreover, we plotted in Figures 1, 2 and 3 the *normwise* relative errors (a), the Performance Profiles (b), and the ratio of relative errors (c) to show if these ratios are significant:

$$\text{E(cosmtayher)/E(cosmtay)}, \ \text{E(cosmtayher)/E(cosm)},$$

and the ratios of matrix products (d):

$$\text{M(cosmtayher)/M(cosmtay)}, \ \text{M(cosmtayher)/M(cosm)},$$

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 methods. The ratios of relative errors are presented in decreasing order with respect to E(`cosmtayher`)/E(`cosmtay`) and E(`cosmtayher`)/E(`cosm`). The solid lines in figures 1a, 2a and 3a is the function $k_{\cos}u$, where $k_{\cos}$ is the condition number of matrix cosine function [26, Chapter 3] and $u = 2^{-53}$ is the unit roundoff in the double precision floating-point arithmetic. Our conclusions are:



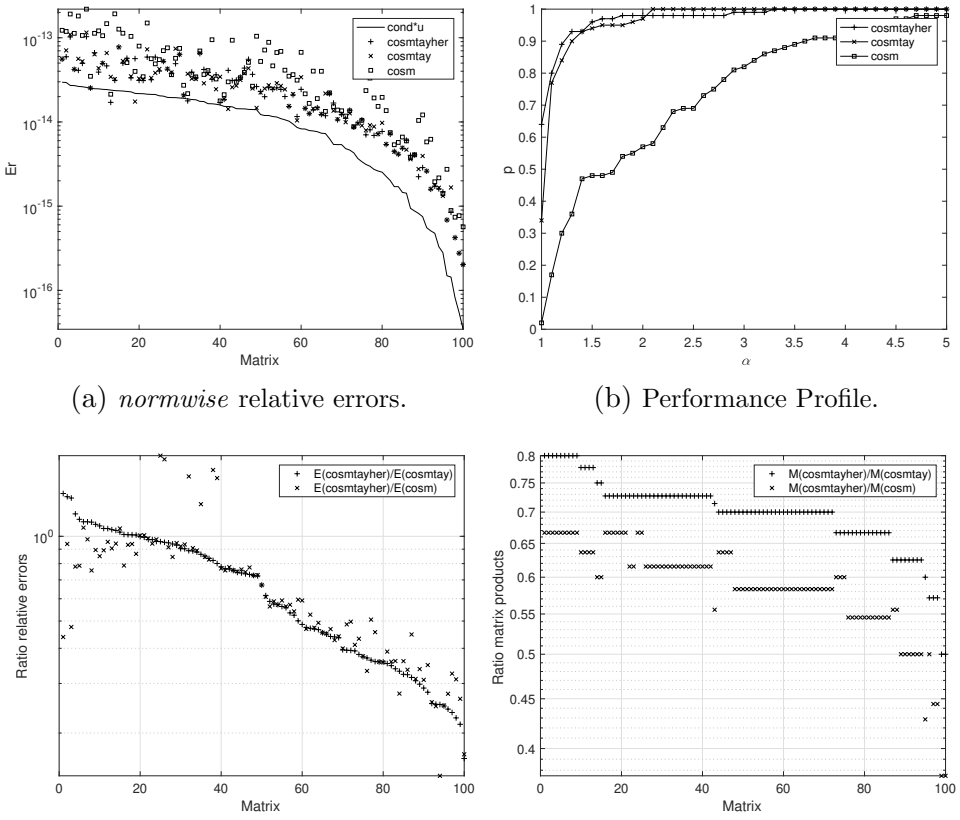
(a) *Normwise* relative errors. (b) Performance profile.

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

Figure 1: Experimental results for Test 1.

- Figures 1a, 2a, and 3a, which display the *normwise* relative error, show that all the implementations have, in general, a similar numerical stability. Only for two matrices of Test 3 all the functions present certain numerical instability (see Subfigure 3a). This can be appreciated taking into account the distance from each matrix *normwise* relative error to the $cond * u$ line.

- The functions based on polynomial approximations are more accurate than the one based on Padé approximants, being the new function `cosmtayher` the more accurate one. The performance profile (Figures 1b, 2b, 3b) shows the graph of `cosmtayher` above the graphs of the other two functions, which means that `cosmtayher` is the most accurate routine. Table 5 shows that function `cosmtayher` has a lower



(a) *normwise* relative errors. (b) Performance Profile.

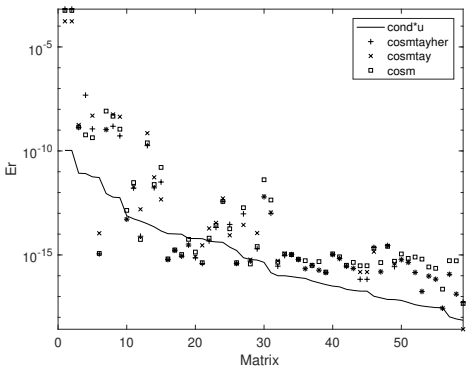
(c) Ratio of relative errors. (d) Ratio of matrix products.

Figure 2: Experimental results for Test 2

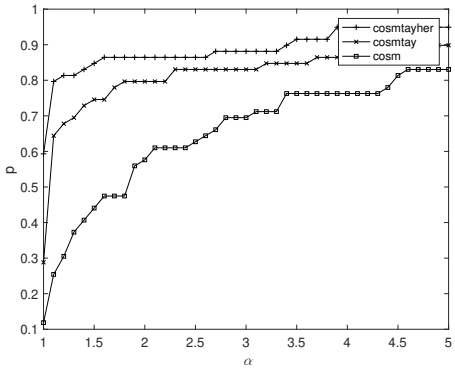
relative error, between 77.97%−92% of the matrices (row 1), than `cosm`, which it is between 53%−69.49% of the matrices, and than `cosmtay` (row 2). Subfigures 1c, 2c and 3c show the ratio of relative errors. Subfigures 1c, 2c and 3c show that E(`cosmtayher`)<0.8 `E(cosm)` for approximately 70 of 100 matrices in Test 1, 58 of 100 matrices in Test 2, and 36 of 59 matrices in Test 3. The values in these Subfigures 1c, 2c and 3c show the more accurate method is `cosmtayher`.

- Table 4 shows that function `cosmtayher` has a significantly lower computational cost than the other two functions. This can also be verified at the sight of Subfigures 1d, 2d and 3d, from which we can see that:

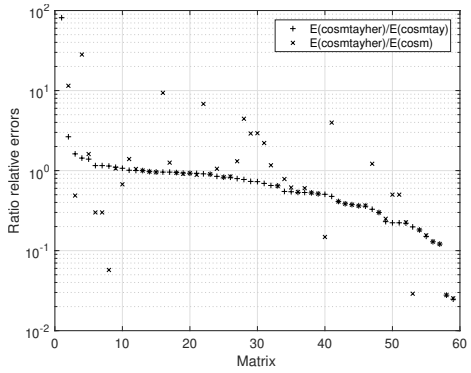
  – Subfigure 1d (Test 1):



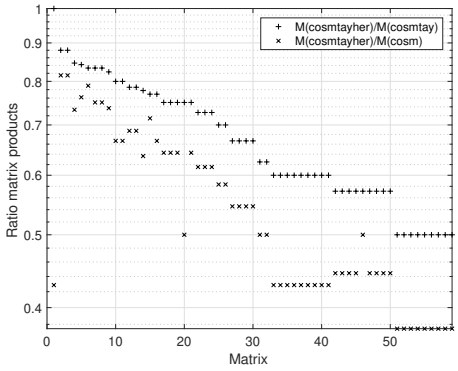
(a) *normwise* relative error.



(b) Performance Profile.



(c) Ratio of relative errors.



(d) Ratio of matrix products.

Figure 3: Experimental results for Test 3.

M(`cosmtayher`) ∈[0.5M(`cosmtay`),0.8M(`cosmtay`)],
M(`cosmtayher`) ∈[0.38M(`cosm`),0.67M(`cosm`)].

– Subfigure 2d (Test 2):
M(`cosmtayher`) ∈[0.57M(`cosmtay`),0.8M(`cosmtay`)],
M(`cosmtayher`) ∈[0.38M(`cosm`),0.67M(`cosm`)].

– Subfigure 3d (Test 3):
M(`cosmtayher`) ∈[0.5M(`cosmtay`),0.88M(`cosmtay`)],
M(`cosmtayher`) ∈[0.38M(`cosm`),0.8M(`cosm`)].

6.2. Results in GPU

Matrix multiplication is a highly optimized operation in different computing environments. The NVIDIA implementation for its GPUs (included in

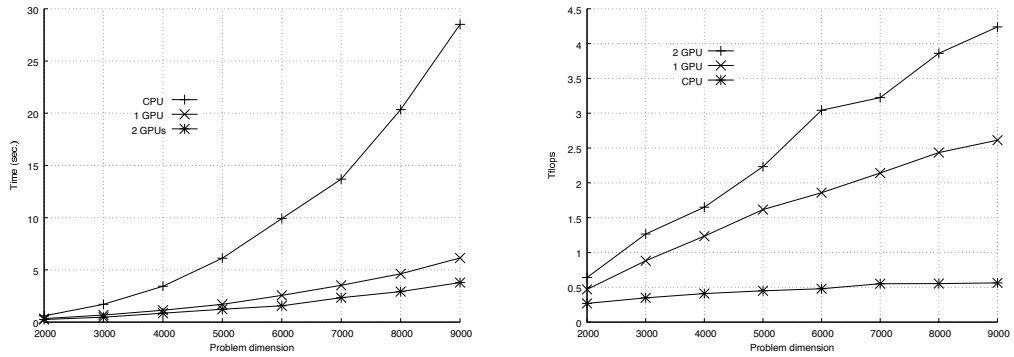


Figure 4: Execution time (sec.) and performance (Tflops) of algorithm `cosmtayher` on CPU, 1 GPU, and 2 GPUs for large problem sizes.

the CUBLAS [33] package) takes full advantage of the computational power of these devices. Therefore, algorithms based on matrix polynomials evaluation and, in turn, based on matrix products, benefit of this.

To analyze the performance of `cosmtayher` on GPUs we carried out our experiments on a computer equipped with two processors Intel Xeon CPU E5-2698 at 2.20 GHz featuring 20 cores each. Attached to the PCI of this board there are four NVIDIA Tesla P100 SMX2 (Pascal architecture) with 16 GB of memory each. Each one of these GPUs features 56 multiprocessors with 64 CUDA cores each, resulting in a total of 3584 CUDA cores. All the GPUs are interconnected through NVIDIA NVLink.

Figure 4 left shows the reduction in execution time when we use GPUs to accelerate the computations and how the execution time decreases when increasing the matrix dimension. We should note that the CPU uses all the 40 cores to execute the matrix multiplications. Around 80% of efficient is achieved when two GPUs are used, i.e. some performance is lost in communications between the two GPUs when they cooperate to solve the problem. If we account for the theoretical cost of the algorithm, we can analyze the performance. Figure 4 right shows the performance of the algorithm in Tflops in the same scenarios, and how this performance steadily increases with the problem size.

7. Conclusions

We have proposed in this work an accurate algorithm to compute the matrix cosine based on new rational-polynomial Hermite matrix expansions, using an absolute forward error analysis. The algorithm has been compared with other state-of-the-art MATLAB implementations.

The new MATLAB function `cosmtayher` presents a higher accuracy in the majority of tests than the other two functions. The numerical stability of `cosmtayher` is similar to the other algorithms.

The numerical experiments also show that, in general, the computational cost of the new algorithm is significantly lower than the other algorithms used in the comparison, i.e. `cosm` and `cosmtay`. Furthermore, our accelerated implementation for one GPU and our parallel implementation for two GPUs is useful to work with large scale problems, indeed, we obtain a performance that steadily increases with the problem size. Like our previous method (`cosmtay`), the fundamental computational kernel of the method proposed here is matrix multiplication. This operation is highly optimized in different computational devices, namely CPUs, GPUs, FPGAs, etc. We have exploited this fact and our previous experience to demonstrate how we can benefit from the latest generation of NVIDIA GPUs to accelerate the computation of the cosine of a matrix.

8. Acknowledgements

We would like to thank the anonymous referees for their helpful comments addressed to improve this manuscript.

References

[1] S. M. Serbin, S. A. Blalock, An algorithm for computing the matrix cosine, SIAM Journal on Scientific and Statistical Computing 1 (1980) 1984–204.

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

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

[4] 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.

[5] J. Sastre, J. Ibáñez, P. Alonso, J. Peinado, E. Defez, Two algorithms for computing the matrix cosine function, Applied Mathematics and Computation 312 (2017) 66–77.

[6] J. Sastre, J. Ibáñez, P. Ruiz, E. Defez, Efficient computation of the matrix cosine, Applied Mathematics and Computation 219 (14) (2013) 7575–7585.

[7] P. Alonso, J. Ibáñez, J. Sastre, J. Peinado, E. Defez, Efficient and accurate algorithms for computing matrix trigonometric functions, Journal of Computational and Applied Mathematics 309 (2017) 325–332.

[8] T. Netzer, A. Thom, About the solvability of matrix polynomial equations, Bulletin of the London Mathematical Society 49 (4) (2017) 670–675.

[9] P. Alonso, M. Boratto, J. Peinado, J. Ibáñez, J. Sastre, On the evaluation of matrix polynomials using several GPGPUs, Tech. rep., Departament of Information Systems and Computation, Universitat Politècnica de València. http://hdl.handle.net/10251/39615 (2014).

[10] J. Sastre, Efficient evaluation of matrix polynomials, Linear Algebra and its Applications 539 (2018) 229–250.

[11] E. Gallopoulos, B. Philippe, A. Sameh, Parallelism in matrix computations, Springer, 2016.

[12] J. Jódar, R. Company, Hermite matrix polynomials and second order matrix differential equations, Approximation Theory and its Applications 12 (2) (1996) 20–30.

[13] L. Jódar, E. Defez, On Hermite matrix polynomials and Hermite matrix functions, Approximation Theory and its Applications 14 (1) (1998) 36–48.

[14] E. Defez, L. Jódar, Some applications of the Hermite matrix polynomials series expansions, Journal of Computational and Applied Mathematics 99 (1) (1998) 105–117.

[15] J. Sastre, J. Ibáñez, E. Defez, P. Ruiz, Efficient orthogonal matrix polynomial based method for computing matrix exponential, Applied Mathematics and Computation 217 (14) (2011) 6451–6463.

[16] E. Defez, J. Sastre, J. Ibáñez, P. Ruiz, Computing matrix functions solving coupled differential models, Mathematical and Computer Modelling 50 (5) (2009) 831–839.

[17] E. Defez, J. Sastre, J. Ibáñez, P. Ruiz, Computing matrix functions arising in engineering models with orthogonal matrix polynomials, Mathematical and Computer Modelling 57 (7) (2013) 1738–1743.

[18] E. Defez, J. Sastre, J. Ibáñez, J. Peinado, M. M. Tung, A method to approximate the hyperbolic sine of a matrix, International Journal of Complex Systems in Science 4 (1) (2014) 41–45.

[19] E. Defez, J. Sastre, J. Ibáñez, J. Peinado, Solving engineering models using hyperbolic matrix functions, Applied Mathematical Modelling 40 (4) (2016) 2837–2844.

[20] F. Hong-Yi, Z. De-Hui, New generating function formulae of even-and odd-Hermite polynomials obtained and applied in the context of quantum optics, Chinese Physics B 23 (6) (2014) 060301.

[21] E. Defez, M. M. Tung, A new type of Hermite matrix polynomial series, Quaestiones Mathematicae (2017) 1–8.

[22] N. Dunford, J. T. Schwartz, Linear operators, vol. I, Interscience, New York.

[23] G. H. Golub, C. F. Van Loan, Matrix computations, Johns Hopkins University Press, 1996.

[24] E. Defez, A. Hervás, L. Jódar, A. Law, Bounding Hermite matrix polynomials, Mathematical and Computer Modelling 40 (1) (2004) 117–125.

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

[26] N. J. Higham, Functions of Matrices: Theory and Computation, SIAM, Philadelphia, PA, USA, 2008.

[27] J. Sastre, J. Ibáñez, P. Ruiz, E. Defez, Efficient computation of the matrix cosine, Appl. Math. Comput. 219 (2013) 7575–7585.

[28] P. Ruiz, J. Sastre, J. Ibáñez, E. Defez, High perfomance computing of the matrix exponential, J. Comput. Appl. Math. 291 (2016) 370–379.

[29] P. Alonso, J. Peinado, J. J. Ibáñez, J. Sastre, E. Defez, A fast implementation of matrix trigonometric functions sine and cosine, in: Proceedings of the 17th International Conference on Computational and Mathematical Methods in Science and Engineering (CMMSE 2017), Vol. 1, 2017, pp. 51–55.

[30] T. G. Wright, Eigtool, version 2.1 (2009).
URL `web.comlab.ox.ac.uk/pseudospectra/eigtool`.

[31] N. J. Higham, The Test Matrix Toolbox for MATLAB, Numerical Analysis Report No. 237, Manchester, England (Dec. 1993).

[32] E. B. Davies, Approximate diagonalization, SIAM J. Matrix Analysis Applications 29 (4) (2007) 1051–1064.

[33] NVIDIA, CUDA. CUBLAS library (2009).