Contents lists available at ScienceDirect

Journal of Computational and Applied Mathematics

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

New Hermite series expansion for computing the matrix hyperbolic cosine

E. Defez [a,*], J. Ibáñez [a], J. Peinado [b], P. Alonso-Jordá [b], José M. Alonso [c]

[a] *Instituto de Matemática Multidisciplinar, Universitat Politècnica de València, Camino de Vera s/n, 46022, Valencia, Spain*
[b] *Departamento Sistemas Informáticos y Computación, Universitat Politècnica de València, Camino de Vera s/n, 46022, Valencia, Spain*
[c] *Instituto de Instrumentación para Imagen Molecular, Universitat Politècnica de València, Camino de Vera s/n, 46022, Valencia, Spain*

A R T I C L E   I N F O

Article history:
Received 27 June 2019
Received in revised form 27 November 2021

Keywords:
Hermite matrix approximation
Matrix hyperbolic cosine
Error analysis
GPU computing

A B S T R A C T

There are, currently, very few implementations to compute the hyperbolic cosine of a matrix. This work tries to fill this gap. To this end, we first introduce both a new rational-polynomial Hermite matrix expansion and a formula for the forward relative error of Hermite approximation in exact arithmetic with a sharp bound for the forward error. This matrix expansion allows obtaining a new accurate and efficient method for computing the hyperbolic matrix cosine. We present a MATLAB implementation, based on this method, which shows a superior efficiency and a better accuracy than other state-of-the-art methods. The algorithm developed on the basis of this method is also able to run on an NVIDIA GPU thanks to a MEX file that connects the MATLAB implementation to the CUDA code.

1. Introduction

Functions of a square matrix A frequently arise in many areas of science and technology, especially those that require the resolution of first and second order differential systems [1, pp. 35–37]. In particular, the hyperbolic matrix functions $\cosh(A)$ and $\sinh(A)$, defined in terms of the exponential matrix e^A as

$$\cosh(A) = \frac{e^A + e^{-A}}{2}, \quad \sinh(A) = \frac{e^A - e^{-A}}{2},$$

are involved in the solution of coupled hyperbolic systems of partial differential equations [2]. Moreover, we also can find applicability for these functions in other fields of science and engineering, e.g., communicability analysis in complex networks [3–6].

A way of computing these matrix functions is to use the well-known relations:

$$\cosh(A) = \cos(iA), \quad \text{and} \quad \sinh(A) = i\cos\left(A - \frac{i\pi}{2}I\right), \; i^2 = -1,$$

provided there exists a method to compute $\cos(A)$. Notwithstanding, this approach has the disadvantage of requiring complex arithmetic even when A is a real matrix.

* Corresponding author.
E-mail addresses: edefez@imm.upv.es (E. Defez), jjibanez@dsic.upv.es (J. Ibáñez), jpeinado@dsic.upv.es (J. Peinado), palonso@upv.es (P. Alonso-Jordá), jmalonso@dsic.upv.es (J.M. Alonso).

https://doi.org/10.1016/j.cam.2022.114084

There exist, however, alternative and more practical ways to evaluate these matrix functions. One of them uses Hermite matrix polynomials series expansions [7]. Other methods based on Taylor series have been studied to evaluate the action of these functions on vectors [8,9].

1.1. Notation

Throughout this paper, we denote by $\mathbb{C}^{r\times r}$ the set of all the complex square matrices of size r. We denote by Θ and I, respectively, the zero and the identity matrix in $\mathbb{C}^{r\times r}$. 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, $\lceil x \rceil$ denotes the ceiling function, that is, the least integer greater than or equal to x and $\lfloor x \rfloor$ denotes the floor function, that is, the largest integer less than or equal to 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 $g(A)$, respectively, the image by the Riesz–Dunford functional calculus of functions $f(z)$ and $g(z)$ acting on the matrix A, being $f(A)g(A) = g(A)f(A)$ [10, pp. 558]. We say that the 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 the function $z^{1/2} = \exp\left(\frac{1}{2}\log(z)\right)$ by the Riesz–Dunford functional calculus, acting on the 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 $\|y\|_1 = |y_1| + \cdots + |y_r|$, $y \in \mathbb{C}^r$ [11, Chapter 2]. When a concrete norm is not indicated in the text, any consistent norm can be applied.

Although originally introduced by Laplace in 1810, Hermite polynomials get their name from the French mathematician Charles Hermite, who wrote about them in 1864. Hermite polynomials and their generalizations are an active research area in the field of special functions, and its applications are numerous, see for example the Refs. [12–14].

For a positive stable matrix $A \in \mathbb{C}^{r\times r}$, the nth Hermite matrix polynomial is defined in [15] by

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

which satisfies the three-term matrix recurrence:

$$\left.\begin{aligned} &H_n(x, A) = x\sqrt{2A}H_{n-1}(x, A) - 2(n-1)H_{n-2}(x, A)\ ,\ n \geq 1, \\ &H_{-1}(x, A) = \Theta\ ,\ H_0(x, A) = I\ . \end{aligned}\right\} \tag{2}$$

The following upper bounds of Hermite matrix polynomials,

$$\left.\begin{aligned} &\|H_{2n}(x, A)\|_2 \leq g_n(x)\ ,\ n \geq 1, \\ &\|H_{2n+1}(x, A)\|_2 \leq |x| \left\| \left(\frac{A}{2}\right)^{-\frac{1}{2}} \right\|_2 \frac{2g_n(x)}{n+1}\ ,\ n \geq 0, \end{aligned}\right\} \tag{3}$$

were demonstrated in [16], where the function $g_n(x)$ is defined as

$$g_n(x) = \frac{(2n+1)!2^{2n}}{n!} \exp\left(\frac{5}{2}\|A\|_2 x^2\right), n \geq 0. \tag{4}$$

The Hermite matrix polynomial sequence $\{H_n(x, A)\}_{n\geq 0}$ has the following generating function [15]:

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

from which we can derive the following expressions for the matrix hyperbolic sine and cosine [17]:

$$\left.\begin{aligned} \cosh\left(xt\sqrt{2A}\right) &= e^{t^2} \sum_{n\geq 0} \frac{H_{2n}(x, A)}{(2n)!} t^{2n} \\ \sinh\left(xt\sqrt{2A}\right) &= e^{t^2} \sum_{n\geq 0} \frac{H_{2n+1}(x, A)}{(2n+1)!} t^{2n+1} \end{aligned}\right\}\ ,\ x \in \mathbb{R}, |t| < \infty. \tag{5}$$

Recent polynomial and rational developments in series of Hermite polynomials have been obtained in [18]. These results have led to the development of a more accurate and efficient method, compared with others proposed in the literature, for the computation of the matrix cosine function. In this paper, we calculate the exact value of the following Hermite matrix polynomial series:

$$\sum_{n\geq 0} \frac{H_{2n+1}(x, A)}{(2n)!} t^{2n} := \mathcal{A}(x, t; A)\ , \tag{6}$$

$$\sum_{n\geq 0} \frac{H_{2n+2}(x,A)}{(2n+1)!} t^{2n+1} := \mathcal{B}(x,t;A)\,, \tag{7}$$

and

$$\sum_{n\geq 0} \frac{H_{2n+3}(x,A)}{(2n+1)!} t^{2n+1} := \mathcal{C}(x,t;A)\,, \tag{8}$$

which result in a new expansion of the hyperbolic matrix cosine in Hermite matrix polynomials. This method aims to improve the one proposed in [7] and to achieve more accuracy in the results, for a large variety of matrices, with a lower computational cost in terms of matrix products.

The organization of this paper is as follows. Section 2 presents the proofs for the formulas (6)–(8) introduced in this paper. Section 3 deals with the new rational-polynomial Hermite matrix expansions for the hyperbolic matrix cosine. The proposed algorithm and its MATLAB implementation is described in Section 4. The numerical results are exposed in Section 5. Some final conclusions are given in Section 6.

2. A proof of formulas (6) –(8)

The aim is to calculate the exact value of matrix series $\mathcal{A}(x,t;A)$, $\mathcal{B}(x,t;A)$ and $\mathcal{C}(x,t;A)$ defined by (6)–(8). Next, we prove that all matrix series are convergent. Taking into account (3), we have

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

Since using (4), $\sum_{n\geq 0} \frac{g_n(x)}{(n+1)(2n)!} |t|^{2n}$ is convergent for $|t| < \infty$, the matrix series $\mathcal{A}(x,t;A)$ defined by (6) is convergent in any compact real interval. In the same way, we have

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

Since using (4), $\sum_{n\geq 0} \frac{g_{n+1}(x)}{(2n+1)!} |t|^{2n+1}$ is convergent for $|t| < \infty$, the matrix series $\mathcal{B}(x,t;A)$ defined by (7) is convergent in any compact real interval. Analogously and taking into account (3) again, we have

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

Since using (4), $\sum_{n\geq 0} \frac{g_{n+1}(x)}{(n+2)(2n+1)!} |t|^{2n+1}$ is convergent for $|t| < \infty$, the matrix series $\mathcal{C}(x,t;A)$ defined by (8) is convergent in any compact real interval. Using now (2), (6) and the fact that $H_1(x,A) = \sqrt{2A}x$, we can write

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

The proof of (7) is similar:

$$\begin{aligned}
\mathcal{B}(x,t;A) &= \left(x\sqrt{2A}\right) \sum_{n\geq 0} \frac{H_{2n+1}(x,A)}{(2n+1)!} t^{2n+1} - 2\sum_{n\geq 0} \frac{(2n+1)H_{2n}(x,A)}{(2n+1)!} t^{2n} \\
&= H_1(x,A) e^{-t^2} \sinh\left(xt\sqrt{2A}\right) - 2t \sum_{n\geq 0} \frac{H_{2n}(x,A)}{(2n)!} t^{2n}
\end{aligned}$$

$$= H_1(x,A)e^{-t^2}\sinh\left(xt\sqrt{2A}\right) - 2te^{-t^2}\cosh\left(xt\sqrt{2A}\right)$$
$$= e^{-t^2}\left[H_1(x,A)\sinh\left(xt\sqrt{2A}\right) - 2t\cosh\left(xt\sqrt{2A}\right)\right].$$

Working analogously for (8):

$$\begin{aligned}
\mathcal{C}(x,t;A) &= x\sqrt{2A}\sum_{n\ge 0}\frac{H_{2n+2}(x,A)}{(2n+1)!}t^{2n+1} - 2\sum_{n\ge 0}\frac{(2n+2)H_{2n+1}(x,A)}{(2n+1)!}t^{2n+1}\\
&= H_1(x,A)\mathcal{B}(x,t;A) - 2\left(\sum_{n\ge 0}\frac{(2n+1)H_{2n+1}(x,A)}{(2n+1)!}t^{2n+1}\right.\\
&\left.+\sum_{n\ge 0}\frac{H_{2n+1}(x,A)}{(2n+1)!}t^{2n+1}\right)\\
&= H_1(x,A)\mathcal{B}(x,t;A) - 2\left(\sum_{n\ge 0}\frac{H_{2n+1}(x,A)}{(2n)!}t^{2n+1} + e^{-t^2}\sinh\left(xt\sqrt{2A}\right)\right)\\
&= H_1(x,A)\mathcal{B}(x,t;A) - 2t\mathcal{A}(x,t;A) - 2e^{-t^2}\sinh\left(xt\sqrt{2A}\right).
\end{aligned}$$

Taking into account the values of $\mathcal{A}(x,t;A)$ and $\mathcal{B}(x,t;A)$, we get

$$\begin{aligned}
\mathcal{C}(x,t;A) &= H_1(x,A)\mathcal{B}(x,t;A) - 2t\mathcal{A}(x,t;A) - 2e^{-t^2}\sinh\left(xt\sqrt{2A}\right)\\
&= e^{-t^2}\left[\left(H_1(x,A)^2 + (4t^2-2)I\right)\sinh\left(xt\sqrt{2A}\right) - 4tH_1(x,A)\cosh\left(xt\sqrt{2A}\right)\right].
\end{aligned}$$

By (1), we have that $H_1(x,A) = \sqrt{2A}x$, $H_2(x,A) = 2x^2A - 2I$, and we can rewrite the last expression of $\mathcal{C}(x,t;A)$ in the form

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

Summarizing, the following result has been established:

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

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

Having in mind that the Hermite matrix polynomial $H_n(x,A)$ coincides with the Hermite polynomial $H_n(x)$, taking $r = 1$ and $A = 2$ (see [15] for more details), we get the following corollary:

Corollary 1. *Let $\{H_n(x)\}_{n\ge 0}$ be the sequence of Hermite polynomials, $x \in \mathbb{R}$, $|t| < +\infty$. Then*

$$\left.\begin{aligned}
\sum_{n\ge 0}\frac{H_{2n+1}(x)}{(2n)!}t^{2n} &= e^{-t^2}\left[H_1(x)\cosh(2xt) - 2t\sinh(2xt)\right],\\
\sum_{n\ge 0}\frac{H_{2n+2}(x)}{(2n+1)!}t^{2n+1} &= e^{-t^2}\left[H_1(x)\sinh(2xt) - 2t\cosh(2xt)\right],\\
\sum_{n\ge 0}\frac{H_{2n+3}(x)}{(2n+1)!}t^{2n+1} &= e^{-t^2}\left[\left(H_2(x) + 4t^2\right)\sinh(2xt) - 4tH_1(x)\cosh(2xt)\right].
\end{aligned}\right\} \quad (10)$$

Formulas (10) are new in the literature of Hermite polynomials and special functions.

3. Some new Hermite matrix series expansions for the hyperbolic 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 $\sinh\left(xt\sqrt{2A}\right)$ given in (5) into the first expression of (9), we obtain a new rational expression for the hyperbolic matrix cosine in terms of Hermite matrix polynomials:

$$\begin{aligned}\cosh\left(xt\sqrt{2A}\right) &= e^{t^2}\left(\sum_{n\geq 0}\frac{H_{2n+1}(x,A)}{(2n)!}\left(1+\frac{2t^2}{2n+1}\right)t^{2n}\right)\left[H_1(x,A)\right]^{-1},\\ & x\neq 0,\ |t|<+\infty.\end{aligned} \tag{11}$$

Substituting $\sinh\left(xt\sqrt{2A}\right)$ given in (5) into the second expression of (9) and using the three-term matrix recurrence (2), we obtain the expression of $\cosh\left(xt\sqrt{2A}\right)$ given in (5).

On the other hand, replacing the expression of $\sin\left(xt\sqrt{2A}\right)$ given in (5) into the third expression of (9), we get another new rational expression for the hyperbolic matrix cosine in terms of Hermite matrix polynomials:

$$\begin{aligned}&\cosh\left(xt\sqrt{2A}\right) =\\ &= \frac{-e^{t^2}}{4}\left[\sum_{n\geq 0}\frac{H_{2n+3}(x,A)}{(2n+1)!}t^{2n} - \left(H_2(x,A)+4t^2 I\right)\star\left(\sum_{n\geq 0}\frac{H_{2n+1}(x,A)}{(2n+1)!}t^{2n+1}\right)\right]\left[H_1(x,A)\right]^{-1},\\ &\quad x\neq 0,\ |t|<+\infty.\end{aligned} \tag{12}$$

Comparing (12) with (11), we observe that there is always an extra matrix product when evaluating (12), the matrix product indicated by the symbol "⋆". Due to the importance of reducing the number of matrix products, see [19–21] for more details, we will focus mainly on the expansion (11).

From (1), it follows that, for $x \neq 0$:

$$\begin{aligned}H_{2n+1}(x,A)\left[H_1(x,A)\right]^{-1} &= (2n+1)!\sum_{k=0}^{n}\frac{(-1)^k x^{2(n-k)}(2A)^{n-k}}{k!(2(n-k)+1)!}\\ &= \widetilde{H}_{2n+1}(x,A),\end{aligned} \tag{13}$$

where

$$\widetilde{H}_n(x,A) = n!\sum_{k=0}^{\lfloor \frac{n}{2}\rfloor}\frac{(-1)^k\left(\sqrt{2A}\right)^{n-2k-1}}{k!(n-2k)!}x^{n-2k-1}, \tag{14}$$

so the right hand side of (13) is still defined in the case where matrix A is singular. In this way, we can rewrite the relation (11) in terms of the matrix polynomial $\widetilde{H}_{2n+1}(x,A)$ as

$$\begin{aligned}\cosh\left(xt\sqrt{2A}\right) &= e^{t^2}\left(\sum_{n\geq 0}\frac{\widetilde{H}_{2n+1}(x,A)}{(2n)!}\left(1+\frac{2t^2}{2n+1}\right)t^{2n}\right),\\ & x\in\mathbb{R},\ |t|<+\infty.\end{aligned} \tag{15}$$

Replacing matrix A by $A^2/2$ in (15) we can avoid the square roots of matrices, and taking $x=\lambda$, $\lambda\neq 0$, $t=1/\lambda$, we finally obtain the expression

$$\cosh(A) = e^{\frac{1}{\lambda^2}}\left(\sum_{n\geq 0}\frac{\widetilde{H}_{2n+1}\left(\lambda,\frac{1}{2}A^2\right)}{(2n)!\lambda^{2n}}\left(1+\frac{2}{(2n+1)\lambda^2}\right)\right),\ 0<\lambda<+\infty. \tag{16}$$

4. Numerical approximations

Truncating the given series (16) until order m, we obtain the approximation $CH_m(\lambda,A)\approx\cosh(A)$ defined by

$$CH_m(\lambda,A) = e^{\frac{1}{\lambda^2}}\left(\sum_{n=0}^{m}\frac{\widetilde{H}_{2n+1}\left(\lambda,\frac{1}{2}A^2\right)}{(2n)!\lambda^{2n}}\left(1+\frac{2}{(2n+1)\lambda^2}\right)\right),\ 0<\lambda<+\infty. \tag{17}$$

Now, from the definition (13), it follows that $\widetilde{H}_{2n+1}\left(x,\frac{1}{2}A^2\right)$ is given by

$$\widetilde{H}_{2n+1}\left(x,\frac{1}{2}A^2\right) = (2n+1)!\sum_{k=0}^{n}\frac{(-1)^k x^{2(n-k)}\left(A^2\right)^{n-k}}{k!(2(n-k)+1)!}$$

and, therefore, considering the 2−norm, we find that

$$\begin{aligned}\left\|\widetilde{H}_{2n+1}\left(x, \frac{1}{2}A^2\right)\right\|_2 &\le (2n+1)! \sum_{k=0}^{n} \frac{|x|^{2(n-k)}\left(\left\|A^2\right\|_2\right)^{n-k}}{k!(2(n-k)+1)!} \\ &= (2n+1)! \sum_{k=0}^{n} \frac{|x|^{2(n-k)}\left(\left\|A^2\right\|_2^{1/2}\right)^{2(n-k)}}{k!(2(n-k)+1)!} \\ &= (2n+1)! \sum_{k=0}^{n} \frac{\left(|x|\,\left\|A^2\right\|_2^{1/2}\right)^{2(n-k)}}{k!(2(n-k)+1)!}. \end{aligned} \tag{18}$$

If $\mathcal{A}(k, n)$ is a matrix in $\mathbb{C}^{r\times r}$, for $n \ge 0, k \ge 0$, then from [22, p. 57] we get that

$$\sum_{n\ge 0}\sum_{k\ge 0}\mathcal{A}(k,n) = \sum_{n\ge 0}\sum_{k=0}^{n}\mathcal{A}(k,n-k). \tag{19}$$

Indeed, thanks to (19), it can be asserted that

$$\begin{aligned}\frac{e\sinh\left(|x|\,\left\|A^2\right\|_2^{1/2}\right)}{|x|\,\left\|A^2\right\|_2^{1/2}} &= \sum_{n\ge 0}\frac{\left(|x|\,\left\|A^2\right\|_2^{1/2}\right)^{2n}}{(2n+1)!}\sum_{k\ge 0}\frac{1}{k!} \\ &= \sum_{n\ge 0}\sum_{k\ge 0}\underbrace{\frac{\left(|x|\,\left\|A^2\right\|_2^{1/2}\right)^{2n}}{k!(2n+1)!}}_{=\mathcal{A}(k,n)} \\ &= \sum_{n\ge 0}\sum_{k=0}^{n}\frac{\left(|x|\,\left\|A^2\right\|_2^{1/2}\right)^{2(n-k)}}{k!(2(n-k)+1)!},\end{aligned}$$

from which it is deduced that

$$\sum_{k=0}^{n}\frac{\left(|x|\,\left\|A^2\right\|_2^{1/2}\right)^{2(n-k)}}{k!(2(n-k)+1)!} \le \frac{e\sinh\left(|x|\,\left\|A^2\right\|_2^{1/2}\right)}{|x|\,\left\|A^2\right\|_2^{1/2}}. \tag{20}$$

If the Expression (20) is multiplied by $(2n+1)!$ and the inequality (18) is applied, we finally conclude that, for $x \ne 0$,

$$\left\|\widetilde{H}_{2n+1}\left(x, \frac{1}{2}A^2\right)\right\|_2 \le (2n+1)!\frac{e\sinh\left(|x|\,\left\|A^2\right\|_2^{1/2}\right)}{|x|\,\left\|A^2\right\|_2^{1/2}}. \tag{21}$$

Now, the following expression for the approximation error can be obtained:

$$\begin{aligned}\left\|\cosh(A) - CH_m(\lambda, A)\right\|_2 &\le e^{\frac{1}{\lambda^2}}\sum_{n\ge m+1}\frac{\left\|\widetilde{H}_{2n+1}\left(\lambda, \frac{1}{2}A^2\right)\right\|_2}{(2n)!\lambda^{2n}}\left(1+\frac{2}{(2n+1)\lambda^2}\right) \\ &\le \frac{e^{1+\frac{1}{\lambda^2}}\sinh\left(\lambda\left\|A^2\right\|_2^{1/2}\right)}{\lambda\left\|A^2\right\|_2^{1/2}}\sum_{n\ge m+1}\frac{2n+1}{\lambda^{2n}}\left(1+\frac{2}{(2n+1)\lambda^2}\right).\end{aligned} \tag{22}$$

For $\lambda > 1$, it follows that $\dfrac{2}{(2n+1)\lambda^2} < 1$, and

$$\begin{aligned}\sum_{n\ge m+1}\frac{2n+1}{\lambda^{2n}}\left(1+\frac{2}{(2n+1)\lambda^2}\right) &\le 2\sum_{n\ge m+1}\frac{2n+1}{\lambda^{2n}} \\ &= \frac{4+(4m+6)(\lambda^2-1)}{\lambda^{2m}\left(\lambda^2-1\right)^2},\end{aligned}$$

```
function [Lopt,Zopt]=compute_Lz_cosh(m,Lmin,Lmax,incL,Zmin,Zmax,incZ)
Zopt=0;
for L=Lmin:incL:Lmax
  for z=Zmin:incZ:Zmax
    if z>Zopt
       f=(exp(1+1/L^2)*sinh(L*sqrt(z))*(4+(4*m+6)*(L^2-1)))/(sqrt(z)*L^(2*m+1)*(L^2-1)^2);
       if f<eps/2
          Zopt=z;
          Lopt=L;
       end
    end
  end
end
end
```

Fig. 1. MATLAB code for computing optimal values of λ and z.

thus, from (22) we finally deduce that:

$$\left\|\cosh(A) - CH_m(\lambda, A)\right\|_2 \leq \frac{e^{1+\frac{1}{\lambda^2}} \sinh\left(\lambda \left\|A^2\right\|_2^{1/2}\right)\left(4+(4m+6)(\lambda^2-1)\right)}{\left\|A^2\right\|_2^{1/2} \lambda^{2m+1}\left(\lambda^2-1\right)^2}. \tag{23}$$

From expression (23), we can derive the optimal values (λ_m, z_m) such that

$$z_m = \max\left\{ z = \left\|A^2\right\|_2 ; \frac{e^{1+\frac{1}{\lambda^2}} \sinh\left(\lambda z^{1/2}\right)\left(4+(4m+6)(\lambda^2-1)\right)}{z^{1/2}\lambda^{2m+1}\left(\lambda^2-1\right)^2} < u \right\},$$

where u is the unit roundoff in IEEE double precision arithmetic ($u = 2^{-53}$). The optimal values of z and λ, for each m, have been obtained with the MATLAB code which appears in Fig. 1. Given the order of the Taylor approximation m, this code determines all pairs (L, z) so that it is verified that the right hand side of (23) is lower than u, varying L between a minimum value *Lmin* and a maximum value *Lmax* in steps equal to *incL*. In the same way, the code increments the z variable from the minimum value *zmin* to the maximum value *zmax* by *incz*, on each iteration. This allows reducing the search time by choosing appropriately these parameters according to the value of m. The precision to be achieved is determined by the values of *incL* and *incz*. Once all possible pairs have been calculated, the chosen pair is the one that has a maximum value of z. The results are given in Table 1.

Approximation CH_m (see (17)) can be expressed as a polynomial with only even powers of matrix A:

$$CH_m(\lambda, A) = \sum_{i=0}^{m} p_i^{(\lambda)} A^{2i} = \sum_{i=0}^{m} p_i^{(\lambda)} B^i \equiv P_m^{(\lambda)}(B), \tag{24}$$

where $B = A^2$. We present Algorithm 1 which computes the hyperbolic cosine of a matrix A by means of the Paterson–Stockmeyer method [23]. The computational cost of this Algorithm is $k+s$ matrix products, i.e., $2(k+s)n^3$ flops, where k represents the position of the vector m_k used.

Algorithm 1 Scaling and recovering algorithm for computing $C = \cosh(A)$, where $A \in \mathbb{C}^{r\times r}$, with $m_M = 16$ the maximum approximation order allowed.

```
1: $B = A^2$.
2: Choose optimal values of $m_k \in \{2, 4, 6, 9, 12, 16\} \leqslant m_M$ and the scaling parameter $s \in \mathbb{N} \cup \{0\}$.
3: $B = 4^{-s}B$                                                    ▷ Scaling matrix $B$
4: Choose the corresponding $\lambda_{m_k}$ from Table 1.
5: Compute $C = P_{m_k}^{(\lambda_{m_k})}(B)$ by the Paterson–Stockmeyer method (see [24, Section 2]) .
6: for $i = 1 : s$ do                                   ▷ Recovering the approximation of $\cosh(A)$
7:     $C = 2C^2 - I$                                       ▷ Double angle formula of $\cosh(A)$
8: end for
```

The basic steps of this algorithm are the choosing of m_k and s (step 2), the computation of $P_{m_k}^{(\lambda_{m_k})}(B)$ (a complete study of how to compute $P_{m_k}^{(\lambda_{m_k})}(B)$ can be seen in Section 2 from [24]) (step 5), and the recovering (steps 6-7).

Next, let us show how to compute the values of m_k and s. If $\cosh(A)$ is computed from the Taylor series, then the absolute forward error of the Hermite approximation of $\cosh(A)$, denoted by E_f, can be computed as

$$E_f = \left\|\cosh(A) - P_{m_k}^{(\lambda_k)}(B)\right\| = \left\| \sum_{i \geq \hat{m}_k} f_i B^i \right\|,$$

Table 1
Values of z_{m_k}, λ_{m_k} and Θ_{m_k} of the matrix function $\cosh(A)$.

m_k	z_{m_k}	λ_{m_k}	Θ_{m_k}
2	$1.085438916 \times 10^{-5}$	3645.569817	$1.8509243149007247 \times 10^{-6}$
4	$7.072119354 \times 10^{-2}$	130.7978189	$3.810252709308867 \times 10^{-3}$
6	$1.024769681 \times 10^{-1}$	31.00030100	$8.9416635239106868 \times 10^{-2}$
9	1.232994877952250	17.607040100	$1.1838963351971854 \times 10^{0}$
12	4.840136411698479	10.200005000	$5.0162962795121144 \times 10^{0}$
16	16.851353484604754	7.9080200400	$1.7588311877511131 \times 10^{1}$

Table 2
Values of $\hat{m}_k$, $\tilde{m}_k$, and $f_{\max}(m_k)$.

	$m_1 = 2$	$m_2 = 4$	$m_3 = 6$	$m_4 = 9$	$m_5 = 12$	$m_6 = 16$
$\hat{m}_k$	1	2	2	4	6	9
$\tilde{m}_k$	1	2	3	10	13	17
$f_{\max}(m_k)$	0	0	1.9×10^{-17}	5.3×10^{-19}	3.1×10^{-26}	3.4×10^{-39}

where $\hat{m}_k \le m_k$. If $f_{\tilde{m}_k}$ is the first value of the above series greater than u, then we obtain the following approximation:

$$E_f \cong \left\| \sum_{i \geqslant \tilde{m}_k} f_i B^i \right\| .$$

Table 2 shows values $\hat{m}_k$, $\tilde{m}_k$ and $f_{\max}(m_k) = \max\left\{ f_i : \hat{m}_k \le i \le \tilde{m}_k - 1 \right\}$, for each $m_k \in \{2, 4, 6, 9, 12, 16\}$.

The scaling factor s and the order of the Hermite approximation m_k are obtained by simplifying the following theorem:

Theorem 4.1 ([25]). *Let $h_l(x) = \sum_{i \ge l} p_i x^i$ be a power series with radius of convergence w, $\tilde{h}_l(x) = \sum_{i \ge l} |p_i| x^i$, $B \in \mathbb{C}^{n \times n}$ 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 we apply Theorem 4.1 to the series $f_{\tilde{m}_k}(x) = \sum_{i \geqslant \tilde{m}_k} f_i x^i$ and $\tilde{f}_{\tilde{m}_k}(x) = \sum_{i \ge \tilde{m}_k} |f_i| x^i$, then

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

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

$$\Theta_{m_k} = \max\left\{ \theta \geqslant 0 : \tilde{f}_{\tilde{m}_k}(\theta) = \sum_{i \geqslant \tilde{m}_k} |f_i| \, \theta^i \leqslant u \right\}, \tag{25}$$

then using MATLAB (R2017b) Symbolic Math Toolbox with 200 series terms and a zero finder, we obtained the values Θ_{m_k} that verify (25) (see Table 1).

The optimal values m_k and s are obtained from the values of β_t of Theorem 4.1 and from the values Θ_{m_k} of Table 1. A complete study of this question was developed by the authors in [18,24]. Next, we reproduce that study.

Let $\beta_{\min}^{(\tilde{m}_k)} = \min_{1 \leqslant t \leqslant \tilde{m}_k} \{\beta_t\}$. If there exists a value $m_k \le 16$ such that $\beta_{\min}^{(\tilde{m}_k)} \le \Theta_{m_k}$, then the forward error E_f is lower than u. In this case, we choose the lower order m_k such that $\beta_{\min}^{(\tilde{m}_k)} \le \Theta_{m_k}$ and the scaling factor is $s = 0$. Otherwise, we choose the Hermite approximation of order 12 or 16 providing the lower cost, with

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

For computing $\beta_{\min}^{(\tilde{m}_k)}$, we have used the following approximation:

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

where $d_{\tilde{m}_k}$ and $d_{\tilde{m}_k+1}$ are bounds of $\left\| B^{\tilde{m}_k} \right\|$ and $\left\| B^{\tilde{m}_k+1} \right\|$, respectively (see (16) from [26]). The bounds d_l, $l = \tilde{m}_k, \tilde{m}_{k+1}$ can be computed using products of norms of matrix powers previously calculated. For example, for $m_k = 6$ the powers

B^2 and B^3 must be obtained, hence $\beta_{min}^{(3)}$ ($\tilde{m}_k = 3$) can be figured as follows

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

The algorithm for computing the values m and s is analogous to Algorithm 2 from [24].

5. Experiments

In this section, we show the results of numerical accuracy and performance of the proposed algorithm to compute the hyperbolic cosine function. By means of a CUDA implementation, we also show its performance using an NVIDIA GPU.

5.1. Numerical experiments

Our MATLAB implementation, named `coshmtayher`, has been developed by modifying the `coshher` MATLAB code given in [7], by replacing the original Hermite approximation by the new Hermite matrix polynomial developed in this paper and derived from (16). We have compared the new MATLAB function, `coshmtayher`, with `coshher` and `funmcosh` functions defined as:

- `coshmtayher`: Novel code based on the new developments of Hermite matrix polynomials (16).
- `coshher`: Code based on the Hermite series for the matrix hyperbolic cosine [7].
- `funmcosh`: `funm` MATLAB function to compute matrix functions, such as the matrix hyperbolic cosine.

5.2. Experiments description

The tests have been carried out using MATLAB (R2017b) running on an Apple Macintosh iMac 27" (iMac retina 5K 27" late 2015) with a quadcore INTEL i7-6700K 4 GHz processor and 16 Gb of RAM. The following tests were made using different matrices:

(a) Test 1: One hundred diagonalizable 128×128 randomly generated real matrices with 1-norm varying from 2.32 to 220.04. These matrices have the form $A = VDV^T$, where D is a diagonal matrix with real eigenvalues and V is an orthogonal matrix obtained as $V = H/\sqrt{128}$, where H is the Hadamard matrix. The *"exact"* matrix hyperbolic cosine was computed as $\cosh(A) = V\cosh(D)V^T$ (see [1, pp. 10]), by using the MATLAB Symbolic Math Toolbox with 128 decimal digit arithmetic in all the computations.

(b) Test 2: One hundred non-diagonalizable 128×128 random real matrices whose 1-norm range from 6.52 to 249.61. These matrices have the form $A = VJV^T$, where J is a Jordan matrix with real eigenvalues with algebraic multiplicity varying between 1 and 4, and V is an orthogonal matrix obtained as $V = H/\sqrt{128}$, where H is the Hadamard matrix. The *"exact"* matrix hyperbolic cosine was computed as $\cosh(A) = V\cosh(J)V^T$.

(c) Test 3: Thirteen test matrices from the Eigtool MATLAB package [27] with size 128×128 and thirty-nine matrices from the matrix function literature with dimensions lower than or equal to 128 from the function `matrix` of the Matrix Computation Toolbox [28]. These matrices have been scaled so that they have 1-norm not exceeding 1024. The *"exact"* matrix hyperbolic cosine was obtained by computing first the eigenvalue decomposition of matrix A, $A = VDV^{-1}$, by using the MATLAB function `eig`, and then computing $\cosh(A) = V\cosh(D)V^{-1}$, by means of the MATLAB's Symbolic Math Toolbox with 128 decimal digit arithmetic in all the computations.

Tables 3 and 4 show, respectively, the computational costs and the algorithm accuracy for the functions under comparison, i.e. `coshmtayher`, `coshher`, and `funmcosh`, in the three tests described. The algorithm accuracy is tested by computing the normwise relative error (`Er`) as

$$\mathrm{Er} = \frac{\|\cosh(A) - \widetilde{\cosh}(A)\|_1}{\|\cosh(A)\|_1},$$

where $\widetilde{\cosh}(A)$ is the computed solution and $\cosh(A)$ is the exact solution.

Table 3 shows the computational costs represented in terms of the number of matrix products (`P()`) of each code, since the cost of the rest of the operations is negligible compared to matrix multiplications for large enough matrices. The `funmcosh` routine has no matrix products, but a Real Schur Form reduction as the main cost, with a computational cost of $28n^3$ [29] for an $n \times n$ matrix. This can be expressed as a minimum of 14 matrix products, having in mind that the cost of a matrix multiplication is $2n^3$ [1, p. 336].

Table 4, on the other hand, shows the percentage of cases in which the relative errors of `coshmtayher` are lower than, greater than or equal to the relative errors of `coshher` and `funmcosh`.

We have plotted in Figs. 2, 3, and 4, for the three tests, respectively, the normwise relative errors (a), the performance profiles (b), the ratio of relative errors (c) to show if these ratios are significant:

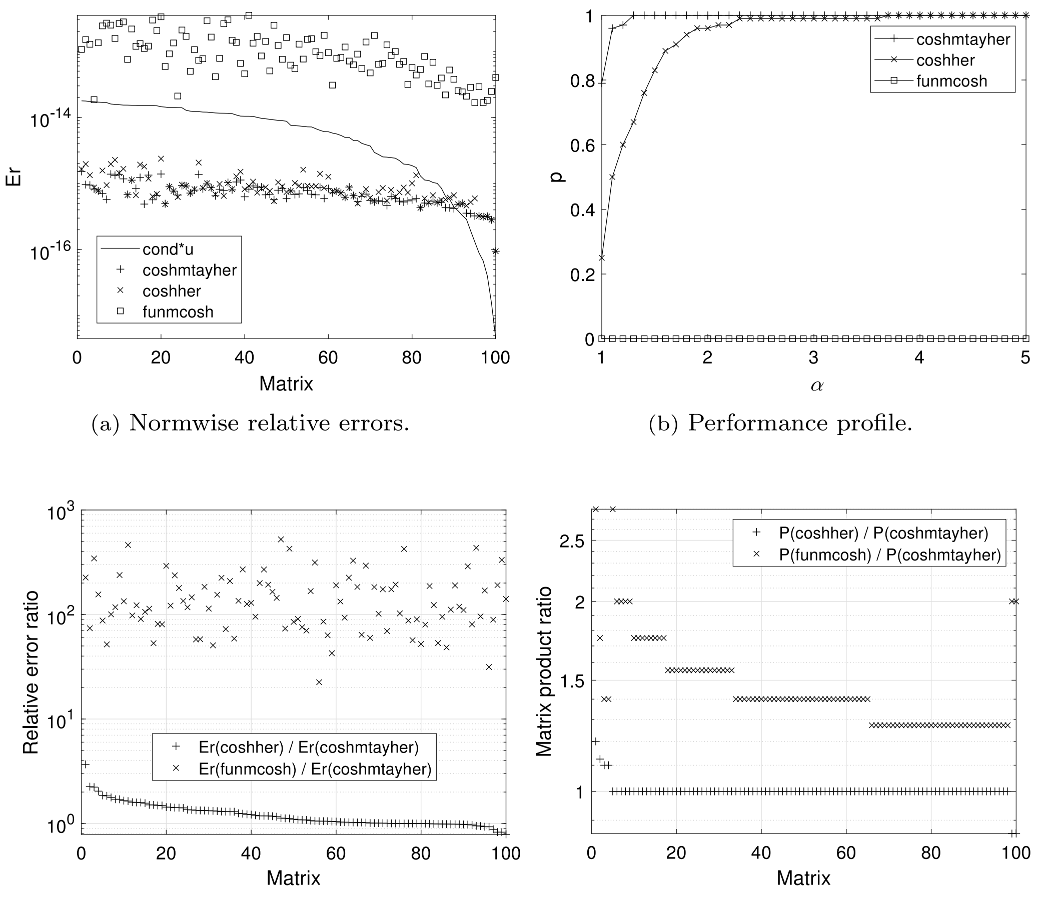
Er(`coshher`)/Er(`coshmtayher`), Er(`funmcosh`)/Er(`coshmtayher`),

Table 3
Matrix products (P) for Tests 1, 2, and 3 using `coshmtayher`, `coshher` and `funmcosh` MATLAB functions.

	P(coshmtayher)	P(coshher)	P(funmcosh)
Test 1	971	973	1400
Test 2	976	988	1400
Test 3	310	317	560

Table 4
Relative error comparison between `coshmtayher` vs `coshher` (rows 1 to 3) and `coshmtayher` vs `funmcosh` (rows 4 to 6) for Test 1, Test 2 and Test 3. The Table entries show the percentage of cases in which the relative errors of `coshmtayher` (New Hermite) are lower than, greater than or equal to the relative errors of `coshher` and `funmcosh`.

	Test 1	Test 2	Test 3
Er(coshmtayher)<Er(coshher)	76%	79%	65%
Er(coshmtayher)>Er(coshher)	21%	20%	25%
Er(coshmtayher)=Er(coshher)	3%	1%	10%
Er(coshmtayher)<Er(funmcosh)	100%	100%	97.5%
Er(coshmtayher)>Er(funmcosh)	0%	0%	2.5%
Er(coshmtayher)=Er(funmcosh)	0%	0%	0%



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

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

Fig. 2. Experimental results for Test 1.

and the ratio of the matrix products (d):

P(`coshher`)/P(`coshmtayher`), P(`funmcosh`)/P(`coshmtayher`).

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 Er(`coshmtayher`)/Er(`coshher`).

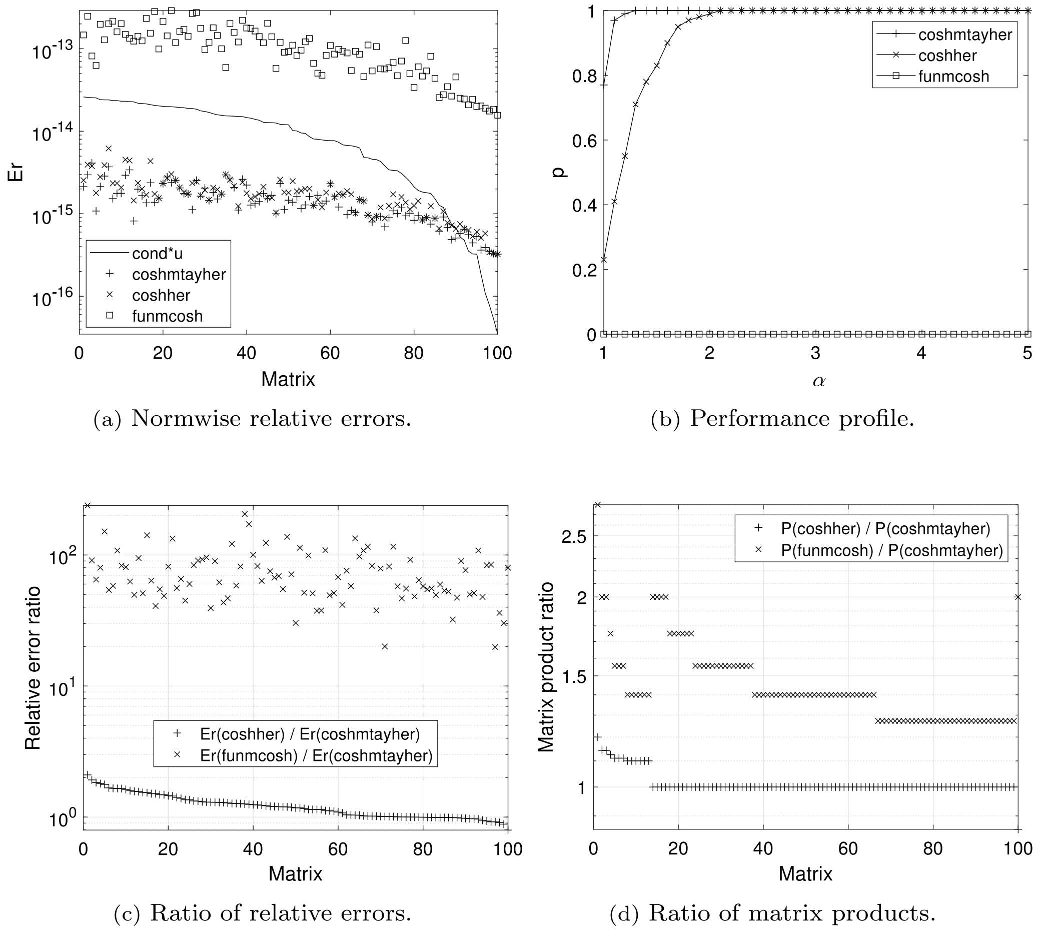
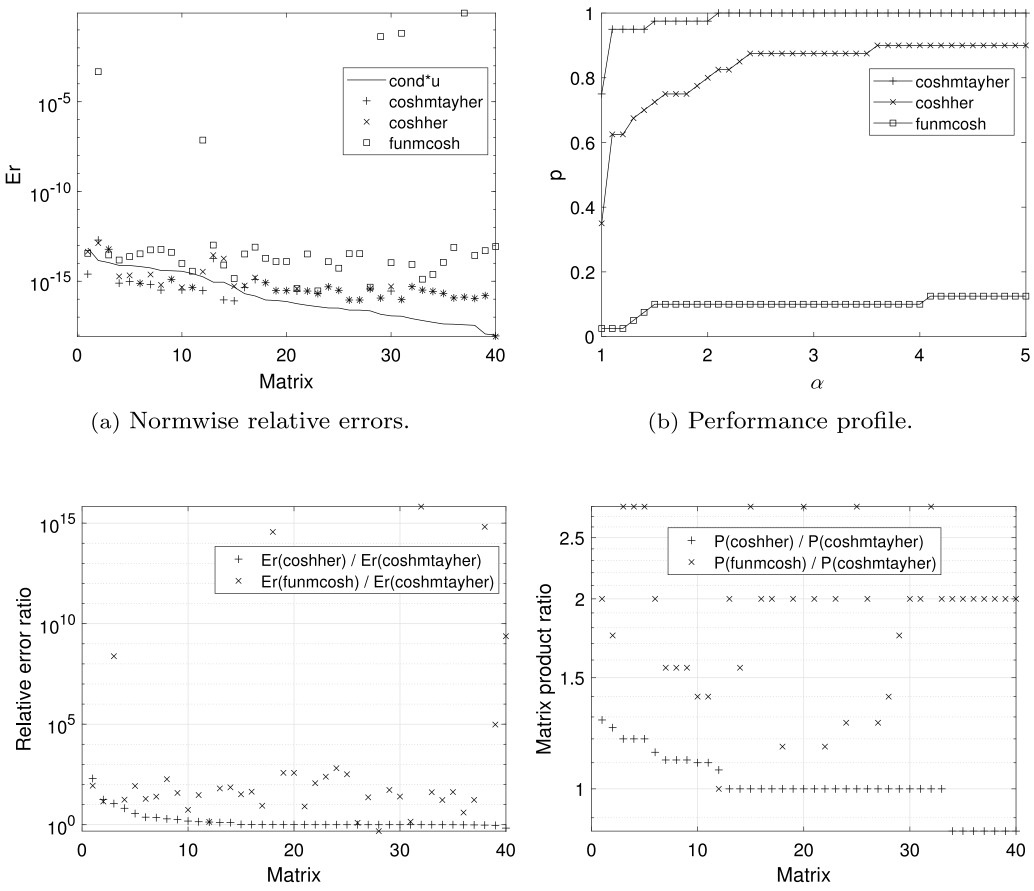


Fig. 3. Experimental results for Test 2.

The solid lines in Figs. 2(a), 3(a) and 4(a) are the function $k_{\cosh}u$, where $k_{\cosh}$ is the condition number of matrix hyperbolic cosine function [1, Chapter 3] and $u = 2^{-53}$ is the unit roundoff in the double precision floating-point arithmetic. Our conclusions are:

- Regarding the normwise relative error shown in Figs. 2(a), 3(a) and 4(a), in general, `coshmtayher` and `coshher` functions have a very good numerical stability. This can be appreciated seeing the distance from each matrix normwise relative error to the $cond * u$ line. In Figs. 2(a) and 3(a), the numerical stability is better because the relative errors are below the $cond * u$ line.
- The performance profile for the first two tests (Figs. 2(b), 3(b)) shows that accuracy of the `coshmtayher` and the `coshher` methods is similar as α increases. Moreover, both of them have much better accuracy than the `funmcosh` method. For the third test, Fig. 4(b) shows that the accuracy of `coshmtayher` function is considerably better than the accuracy of `coshher` function.
- Quantitatively, Table 4 indicates that `coshmtayher` code offers a relative error lower than `coshher` function in 76%, 79% and 65% of the cases, respectively, for Tests 1, 2 and 3. Similarly, `coshmtayher` method improves `funmcosh` function in terms of the relative error incurred in the calculation of the 100% of the matrices that comprise Tests 1 and 2, or in the 97.5% of the ones belonging to Test 3. These numerical values are corroborated by the error ratios depicted in Figs. 2(c), 3(c) and 4(c). As we can notice, this ratio is greater than one for a percentage of matrices that coincides with the values exposed in Table 4, which obviously indicates that the `coshmtayher` code is the most accurate and reliable one. Whereas this error ratio takes values not very far from unity between `coshmtayher` and `coshher` functions for the matrices of the first two Tests, this factor reaches more distant values from unity for many matrices that are part of Test 3. Special mention deserves the value of this factor between `coshmtayher` and `funmcosh` codes, which takes very high values particularly in the matrices of Test 3. More in detail:



(a) Normwise relative errors.

(b) Performance profile.

(c) Ratio of relative errors.

(d) Ratio of matrix products.

Fig. 4. Experimental results for Test 3.

Test 1 (Fig. 2(c)):

Er(coshher) ∈ [0.79 Er(coshmtayher), 3.68 Er(coshmtayher)],
Er(funmcosh) ∈ [22.49 Er(coshmtayher), 523.3 Er(coshmtayher)].

Test 2 (Fig. 3(c)):

Er(coshher) ∈ [0.80 Er(coshmtayher), 2.10 Er(coshmtayher)],
Er(funmcosh) ∈ [19.84 Er(coshmtayher), 238.6 Er(coshmtayher)]

Test 3 (Fig. 4(c)):

Er(coshher) ∈ [0.68 Er(coshmtayher), 199.0 Er(coshmtayher)],
Er(funmcosh) ∈ [0.48 Er(coshmtayher), $6.65e+15$ Er(coshmtayher)].

- As it was shown in Table 3, coshmtayher function has a computational cost significantly lower than funmcosh code and slightly lower than coshher function. As expected, the numerical tests also confirm this result:

Test 1 (Fig. 2(d)):

P(coshher) ∈ [0.86 P(coshmtayher), 1.20 P(coshmtayher)],
P(funmcosh) ∈ [1.27 P(coshmtayher), 2.80 P(coshmtayher)].

Test 2 (Fig. 3(d)):

P(coshher) ∈ [0.86 P(coshmtayher), 1.20 P(coshmtayher)],
P(funmcosh) ∈ [1.27 P(coshmtayher), 2.80 P(coshmtayher)].

Test 3 (Fig. 4(d)):

P(coshher) ∈ [0.85 P(coshmtayher), 1.29 P(coshmtayher)],
P(funmcosh) ∈ [1.00 P(coshmtayher), 2.80 P(coshmtayher)].

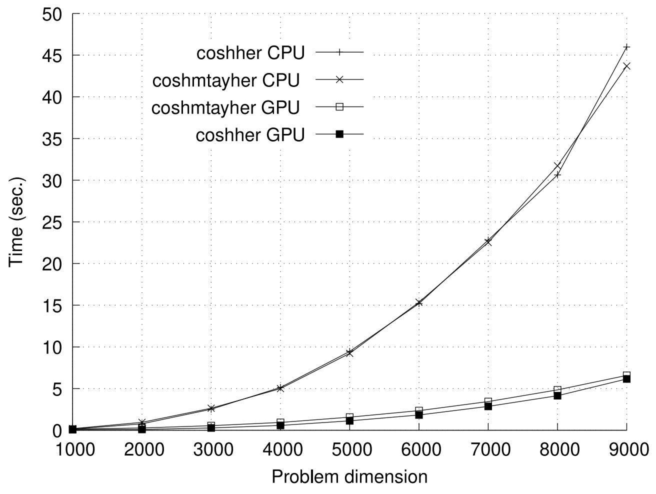


Fig. 5. Execution times (sec.) to compute the matrix hyperbolic cosine in CPU and GPU by means of `coshmtayher` and `coshher` codes on randomly generated large matrices.

5.3. Results in GPU

We have implemented an "accelerated" version that allows to execute our algorithm to compute the matrix hyperbolic cosine on NVIDIA GPUs. Current GPUs are computational devices that allow to boost performance on data parallelism applications, i.e. applications that operate over many independent data. This is the case of matrix multiplication, which is a highly optimized operation for GPUs in its current implementation routine included in the CUBLAS package [30]. Our GPU algorithms are all based on polynomial evaluations which, in turn, results in intensive use of matrix products. The basic MATLAB algorithm is used in this case with some very costly operations (those based on matrix multiplication) addressed to the GPU through CUDA language by a means of implementing a MEX file.

We have carried out our experimental results for this subsection on a computer equipped with two processors Intel Xeon CPU E5-2698 v4 @ 2.20 GHz featuring 20 cores each. To obtain the algorithm performance on GPU we used one NVIDIA Tesla P100-SXM2 (Pascal architecture) attached to the PCI of this workstation. This GPU features 3584 CUDA cores and 16 GB of memory.

Fig. 5 shows the reduction in execution time when we use a GPU to accelerate the computations. The figure also indicates that, for large randomly generated matrices, the former version of the matrix hyperbolic cosine (`coshher`), presented in [7], and the new algorithm (`coshmtayher`) show a similar performance in both the CPU and GPU subsystems. Experimental results with `funmcosh` function are not shown in the figure, since they are very large in comparison. For instance, for a matrix A of order $n = 2000$, the execution time to obtain $\cosh(A)$ is 83 s whilst it is 0.71 s and 0.73 s with `coshmtayher` and `coshher`, respectively.

6. Conclusions

A new polynomial Hermite matrix algorithm has been developed in this work for computing the matrix hyperbolic cosine. We have implemented a MATLAB routine that is also capable of using an existing GPU in the system. This new algorithm has been compared with other MATLAB implementations and, at the light of the tests carried out, we have verified that the new algorithm behaves in a numerically stable manner showing very good results. One of the main conclusions is that MATLAB implementations based on the Hermite series (`coshher` and `coshmtayher`) have turned out to be much more accurate and efficient than others focused on the `funm` MATLAB function when using `funmcosh` code. In addition, the new implementation based on Hermite series (`coshmtayher`), proposed here, offers much better numerical accuracy than that of `coshher` algorithm, which is also based on the Hermite series, with a slightly lower computational cost in terms of matrix products.

Acknowledgments

This work has been partially supported by Spanish Ministerio de Economía, Industria *y* Competitividad and European Regional Development Fund (ERDF) grants TIN2017-89314-P and by the Programa de Apoyo a la Investigación *y* Desarrollo 2018 of the Universitat Politècnica de València, Spain (PAID-06-18) grants SP20180016.

References

[1] N.J. Higham, Functions of Matrices: Theory and Computation, SIAM, Philadelphia, PA, USA, 2008.
[2] L. Jódar, E. Navarro, A. Posso, M. Casabán, Constructive solution of strongly coupled continuous hyperbolic mixed problems, Appl. Numer. Math. 47 (3–4) (2003) 477–492.
[3] E. Estrada, D.J. Higham, N. Hatano, Communicability and multipartite structures in complex networks at negative absolute temperatures, Phys. Rev. E 78 (2) (2008) 026102.
[4] E. Estrada, J.A. Rodríguez-Velázquez, Spectral measures of bipartivity in complex networks, Phys. Rev. E 72 (4) (2005) 046105.
[5] E. Estrada, J. Gómez-Gardeñes, Network bipartivity and the transportation efficiency of european passenger airlines, Physica D 323 (2016) 57–63.
[6] J. Kunegis, G. Gröner, T. Gottron, Online dating recommender systems: The split-complex number approach, in: Proceedings of the 4th ACM RecSys Workshop on Recommender Systems and the Social Web, RSWeb '12, Association for Computing Machinery, 2012, pp. 37–44.
[7] E. Defez, J. Sastre, J. Ibáñez, J. Peinado, Solving engineering models using hyperbolic matrix functions, Appl. Math. Model. 40 (4) (2016) 2837–2844.
[8] N.J. Higham, P. Kandolf, Computing the action of trigonometric and hyperbolic matrix functions, SIAM J. Sci. Comput. 39 (2) (2017) A613–A627.
[9] A.H. Al-Mohy, A truncated taylor series algorithm for computing the action of trigonometric and hyperbolic matrix functions, SIAM J. Sci. Comput. 40 (3) (2018) A1696–A1713.
[10] N. Dunford, J.T. Schwartz, Linear Operators, Part I: General Theory, John Wiley & Songs, Inc., 1988.
[11] G.H. Golub, C.F. Van Loan, Matrix Computations, The Johns Hopkins University Press, Baltimore, Maryland, USA, 2013.
[12] G. Dattoli, C. Cesarano, On a new family of Hermite polynomials associated to parabolic cylinder functions, Appl. Math. Comput. 141 (1) (2003) 143–149.
[13] A. Yari, Numerical solution for fractional optimal control problems by Hermite polynomials, J. Vib. Control 27 (5–6) (2021) 698–716.
[14] D. Masoero, P. Roffelsen, Roots of generalised Hermite polynomials when both parameters are large, Nonlinearity 34 (3) (2021) 1663–1732.
[15] J. Jódar, R. Company, Hermite matrix polynomials and second order matrix differential equations, Approx. Theory Appl. 12 (2) (1996) 20–30.
[16] E. Defez, A. Hervás, L. Jódar, A. Law, Bounding Hermite matrix polynomials, Math. Comput. Modelling 40 (1) (2004) 117–125.
[17] E. Defez, L. Jódar, Some applications of the Hermite matrix polynomials series expansions, J. Comput. Appl. Math. 99 (1) (1998) 105–117.
[18] E. Defez, J. Ibáñez, J. Peinado, J. Sastre, P. Alonso-Jordá, An efficient and accurate algorithm for computing the matrix cosine based on new Hermite approximations, J. Comput. Appl. Math. 348 (2019) 1–13.
[19] J. Sastre, J. Ibáñez, E. Defez, P. Ruiz, New scaling-squaring taylor algorithms for computing the matrix exponential, SIAM J. Sci. Comput. 37 (1) (2015) A439–A455.
[20] P. Alonso, J. Peinado, J. Ibáñez, J. Sastre, E. Defez, Computing matrix trigonometric functions with gpus through matlab, J. Supercomput. (2018) 1–14.
[21] J. Sastre, Efficient evaluation of matrix polynomials, Linear Algebra Appl. 539 (2018) 229–250.
[22] E.D. Rainville, Special functions, 1960, 442, New York.
[23] M.S. Paterson, L.J. Stockmeyer, On the number of nonscalar multiplications necessary to evaluate polynomials, SIAM J. Comput. 2 (1) (1973) 60–66.
[24] J. Sastre, J. Ibáñez, P. Alonso, J. Peinado, E. Defez, Two algorithms for computing the matrix cosine function, Appl. Math. Comput. 312 (2017) 66–77.
[25] J. Sastre, J. Ibáñez, P. Ruiz, E. Defez, Efficient computation of the matrix cosine, Appl. Math. Comput. 219 (14) (2013) 7575–7585.
[26] P. Ruiz, J. Sastre, J. Ibáñez, E. Defez, High perfomance computing of the matrix exponential, J. Comput. Appl. Math. 291 (2016) 370–379.
[27] T.G. Wright, Eigtool, version 2.1, URL http://www.comlab.ox.ac.uk/pseudospectra/eigtool.
[28] N.J. Higham, The matrix computation toolbox, URL: http://www.ma.man.ac.uk/higham/mctoolbox.
[29] M.I. Smith, A schur algorithm for computing matrix pth roots, SIAM J. Matrix Anal. Appl. 24 (4) (2003) 971–989.
[30] NVIDIA, CUDA, CUBLAS library, 2009.