New matrix series expansions for the matrix cosine approximation*

Emilio Defez*, Javier Ibáñez†, José M. Alonso†, Pedro Alonso-Jordá♮

⋆ Instituto de Matemática Multidisciplinar.

† Instituto de Instrumentación para Imagen Molecular.

♮ Grupo Interdisciplinar de Computación y Comunicaciones.

Universitat Politècnica de València, Camino de Vera s/n, 46022, Valencia, España.

edefez@imm.upv.es, {jjibanez, jmalonso, palonso,}@dsic.upv.es

1 Introduction and notation

In recent years, the study of matrix functions has been the subject of increasing focus due to its usefulness in various areas of science and engineering, providing new and interesting problems to those already existing and already well-known. Of all matrix functions, it is certainly the exponential matrix which attracts much of the attention because of its connection with linear first order differential systems

$$\left.\begin{array}{rcl} Y'(t) & = & AY(t) \\ Y(0) & = & Y_0 \end{array}\right\} ,\ A \in \mathbb{C}^{r\times r}$$

whose solution is given by $Y(t) = e^{At}Y_0$. The hyperbolic matrix functions are applied in the study of the communicability analysis in complex networks [1–3] and also in the solution of coupled hyperbolic systems of partial

*Acknowledgements: This work has been partially supported by Spanish Ministerio de Economía 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 (PAID-06-18) grants SP20180016.

differential equations [4]. In particular, the trigonometric matrix functions sine and cosine prove especially useful in solving systems of second-order linear differential equations of the form:

$$\left.\begin{array}{rcl} \dfrac{d^2}{dt^2}Y(t)+A^2Y(t) & = & 0 \\ Y(0) & = & Y_0 \\ Y'(0) & = & Y'_0 \end{array}\right\} ,\ A\in\mathbb{C}^{r\times r}$$

whose solution, if matrix A is non-singular, is given by

$$Y(t)=\cos{(At)}Y_0+A^{-1}\sin{(At)}Y'_0.$$

Due to the relationship $\sin{(A)}=\cos\left(A+\frac{\pi}{2}I\right)$, where I is the identity matrix of $\mathbb{C}^{r\times r}$, the matrix sine can be calculated using the same methods as for the matrix cosine. Usually, research concentrated on approximate calculations of the matrix cosine, developing several efficient state-of-the-art algorithms to approximate it. These methods and algorithms can be found in references [5–8]. Other algorithms, for normal and nonnegative matrices, based on approximations L_∞ also have been presented in Ref. [9].

Among the methods proposed to approximate the matrix cosine, two stand out fundamentally: those based on polynomial approximations (in general based on the developments of the matrix cosine in Taylor or Hermite series, see [10–12]) or those based on rational approximations (i.e. Padé approximation, see [5, 9, 13, 14]). Normally, polynomial methods are more efficient in terms of accuracy (although somewhat more computationally expensive) than rational ones.

On the other hand, Bernoulli polynomials (and numbers) introduced by Jacob Bernoulli (1654–1705) in the 18th century, are widely used in various areas of mathematics, both pure and applied, see for example [15] and references therein.

In this paper, we will define Bernoulli matrix polynomials and present a new series expansion of the matrix sine and cosine functions in terms of these matrix polynomials. Then, we will check that this series expansion will provide a new efficient method to approximate the matrix cosine.

The organization of the paper is as follows: In section 2, we will obtain two serial expansions of the matrix cosine in terms of the Bernoulli matrix polynomials. In section 4, we will perform different numerical tests. Conclusions are given in section XXX.

Throughout this paper, a polynomial of degree m is given by an expression of the form $P_m(x) = a_m x^m + a_{m-1}x^{m-1} + \cdots + a_1 x + a_0$, where x is the variable (real or complex) and the coefficients $a_j, 0 \leq j \leq m$, are complex numbers. Moreover, we can define the matrix polynomial $P_m(B)$ for $B \in \mathbb{C}^{r\times r}$ as the expression $P_m(B) = a_m B^m + a_{m-1}B^{m-1} + \cdots + a_1 B + a_0 I$. As usual, the matrix norm $\|\cdots\|$ denotes any subordinate matrix norm; in particular $\|\cdots\|_1$ is the usual 1-norm.

2 On Bernoulli matrix polynomials

The sequence of Bernoulli polynomials (denoted by $\{B_n(x)\}_{n\geq 0}$) and the Bernoulli numbers, $B_n = B_n(0)$, appear in important applications for different areas of mathematics, from number theory to classical analysis. For example, they are used for representing the remainder term of the composite Euler-MacLaurin quadrature rule. They also appear in the Taylor expansion in the neighborhood of the origin of circular and hyperbolic tangent and co-tangent functions. Moreover, this sequence expresses the exact value of $\zeta(2p)$, with p a positive integer, where $\zeta(z) = \sum_{k\geq 1} \frac{1}{k^z}$ is the well-known Riemann's zeta function. They were first studied by Jacob Bernoulli before 1705 in relation with the computation of sums of powers of m consecutive integers $S_r(m) = \sum_{k=1}^{m} k^r$, where r and m are two given positive integers. The usual way to define these Bernoulli polynomials, see [16, p. 588] , is as the coefficients of the Taylor expansion of the following generating function

$$g(x,t) = \frac{te^{tx}}{e^t - 1} = \sum_{n\geq 0} \frac{B_n(x)}{n!} t^n \ , \ |t| < 2\pi, \tag{1}$$

where $g(x,t)$ is a holomorphic function in $\mathbb{C}$ for the variable t (function $g(x,t)$ has an avoidable singularity in $t = 0$). Polynomials $B_n(x)$ have the explicit expression

$$B_n(x) = \sum_{k=0}^{n} \binom{n}{k} B_k x^{n-k}, \tag{2}$$

where the Bernoulli numbers B_n are defined by the recurrence relation

$$B_0 = 1, B_k = -\sum_{i=0}^{k-1} \binom{k}{i} \frac{B_i}{k+1-i}, k \geq 1. \tag{3}$$

Note that all Bernoulli numbers with impair index vanish, except for $B_1 = -1/2$. For a matrix $A \in \mathbb{C}^{r\times r}$, we define the m-th Bernoulli matrix polynomial by the expression

$$B_m(A) = \sum_{k=0}^{m} \binom{m}{k} B_k A^{m-k}. \tag{4}$$

The series expansion of the exponential matrix function e^{At} given by

$$e^{At} = \left(\frac{e^t - 1}{t}\right) \sum_{n\geq 0} \frac{B_n(A)t^n}{n!} \ , \ 0 < |t| < 2\pi, \tag{5}$$

was demonstrated in Ref. [17]. An efficient method based on formula (5) for approximating the exponential matrix has been presented and developed in Ref. [17].

Taking $t = 1$ in (5) and from the definition of the matrix sine and cosine, it is easy to derive the following expressions

$$\left.\begin{array}{rcl} \cos(A) & = & (\cos(1) - 1) \displaystyle\sum_{n\geq 0} \frac{(-1)^n B_{2n+1}(A)}{(2n+1)!} + \sin(1) \sum_{n\geq 0} \frac{(-1)^n B_{2n}(A)}{(2n)!}, \\ \sin(A) & = & \sin(1) \displaystyle\sum_{n\geq 0} \frac{(-1)^n B_{2n+1}(A)}{(2n+1)!} - (\cos(1) - 1) \sum_{n\geq 0} \frac{(-1)^n B_{2n}(A)}{(2n)!}. \end{array}\right\} \tag{6}$$

Replacing in (5) the value t for it and $-it$ respectively and taking the arithmetic mean, we can also obtain the result

$$\sum_{n\geq 0} \frac{(-1)^n B_{2n}(A)}{(2n)!} t^{2n} = \frac{t}{2\sin\left(\frac{t}{2}\right)} \left(\cos\left(tA - \frac{t}{2}I\right)\right) \ , \ 0 < |t| < 2\pi. \tag{7}$$

Taking $t = 2$ in (7), it follows that

$$\cos(A) = \sin(1) \sum_{n\geq 0} \frac{(-1)^n 2^{2n} B_{2n}\left(\frac{A+I}{2}\right)}{(2n)!}, \tag{8}$$

Note that in formula (8) only Bernoulli's polynomials with even index appear, similar to what happens when considering cosine series expansions using Taylor or Hermite polynomials [10, 12].

3 Algorithms

If we truncate Expressions (6) or (8) of cosine function, then we obtain the following approximation:

$$P_m(A) = \sum_{i=0}^{m} p_i^{(m)} A^i, \tag{9}$$

where the coefficients $p_i^{(m)}$ depend on the integer m. These coefficients converge to the coefficients of the Taylor series when m tends to ∞. In Section 4 three approximations have been considered: two approximations based on Expression (6), an approximation in which all the coefficients are considered and another in which only the coefficients of the even order terms have been considered, and one approximation based on Expression (8), where all the coefficients are considered. According to the approaches considered, two algorithms has been developed.

Algorithm 1 Given a matrix $A \in \mathbb{C}^{n\times n}$, this algorithm computes $C = \cos(A)$ by Bernouilli series (6) or (8), where all coefficients have been considered.

```
1: Select adequate values of $m_k$ and $s \in \mathbb{N} \cup \{0\}$    ▷ Phase I
2: $A = 2^{-s}A$
3: $C = P_{m_k}(A)$    ▷ Phase II: Compute Bernouilli approximation (6) or (8)
4: for $i = 1 : s$ do    ▷ Phase III: Recovering cos($A$)
5:     $C = 2C^2 - I$
6: end for
```

In Phase I the integers m and s are computed so that the Bernouilli approximation of the scaled matrix is computed accurately and efficiently.

Algorithm 2 Given a matrix $A \in \mathbb{C}^{n\times n}$, this algorithm computes $C = \cos(A)$ by Bernouilli series (6), where only the coefficients of the even order terms have been considered.

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

There are several methods can be applied to compute efficiently $C = P_{m_k}(A)$ in Phase II, see [?] and [?]. In our implementations we used the first one based on the Paterson-Stockmeyer's method. In this method, the integer m_k is chosen from the set

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

where only the powers A^i, $2 \le i \le q$, are computed, being $q = \lceil \sqrt{m_k} \rceil$ or $q = \lfloor \sqrt{m_k} \rfloor$ an integer divisor of m_k. Then, $C = P_{m_k}(A)$ can be computed efficiently as

$$\begin{aligned} P_{m_k}(A) = & \\ (((p_{m_k}A^q + p_{m_k-1}A^{q-1} + p_{m_k-2}A^{q-2} + \cdots + p_{m_k-q+1}A + p_{m_k-q}I)A^q & \\ + p_{m_k-q-1}A^{q-1} + p_{m_k-q-2}A^{q-2} + \cdots + p_{m_k-2q+1}A + p_{m_k-2q}I)A^q & \\ + p_{m_k-2q-1}A^{q-1} + p_{m_k-2q-2}A^{q-2} + \cdots + p_{m_k-3q+1}A + p_{m_k-3q}I)A^q & \\ \cdots & \\ + p_{q-1}A^{q-1} + p_{q-2}A^{q-2} + \cdots + p_1A + p_0I. & \end{aligned} \tag{10}$$

The computational cost in terms of matrix products of (10) is $\Pi m_k = k$.

The computation of the scaling factor s and the order of Bernouilli approximation m_k are based on the errors made truncating a series. One first idea is to exploit the sequence $\left\{ \left\| A^k \right\|^{1/k} \right\}$. As $\rho(A) \leqslant \left\| A^k \right\|^{1/k} \leqslant \|A\|$, with $k \geqslant 0$, where ρ is the spectral radius of the matrix A, then

$$\lim_{k\to\infty} ||A^k||^{1/k} = \rho(A).$$

Theorem 1.1 from [?] shows that if $h_l = \sum_{k\geqslant l} c_k x^k$ is a power series with a radius convergence w and $\tilde{h}_l = \sum\limits_{k \geqslant l} |c_k| \, x^k$, then for any $A \in \mathbb{C}^{n\times n}$

$$\|h_l(A)\| \leqslant \tilde{h}_l(||A^t||^{1/t}),$$

where $||A^t||^{1/t} = \max\left\{||A^k||^{1/k} : \ k \geqslant l, \ \ c_l \neq 0\right\}$.

Using the same notation, Theorem 1.1 from [?] shows that if p_0 is the multiple of t such that

$$l \leqslant t_0 \leqslant l + p - 1,$$

if a_k is an upper bound for $||A^k||$ ($||A^k|| \leq a_k$), $p \in \mathbb{N}$, $1 \leq p \leq l$, $p_0 \in \mathbb{N}$ is the multiple of p with $l \leq p_0 \leq l + p - 1$, and

$$\alpha_p = \max\{a_k^{\frac{1}{k}} : k = p, l, l{+}1, l{+}2, \ldots, p_0{-}1, p_0{+}1, p_0{+}2, \ldots, l{+}p{-}1\}, \quad (11)$$

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

If we apply Theorem 1.1 from [?] for $a_k = ||A^k||$ and we consider $p = l$, then

$$||h_l(A)|| \leq \tilde{h}_l(\alpha_l),$$

where $\alpha_l = \max\{||A^k||^{\frac{1}{k}} : k = l, l+1, l+2, \ldots, 2l-1\}$. This result reduces the values of $||A^k||^{\frac{1}{k}}$ given in Theorem 1.1 from [?]. In this paper, we propose use the following approximation:

$$\alpha_m \approx ||A^m||^{1/m}. \quad (12)$$

Using these results, the calculations of m and s from PHASE I of Algorithms 1 and 2 are based on the relative backward of approximating $\cos(A)$ by the approximation (9). This error is defined as the matrix ΔA such that $\cos(A + \Delta A) = P_m(A)$. Below, we bound the relative backward error as follows:

$$E_b = \frac{\|\Delta A\|}{\|A\|} = \frac{\left\| \sum\limits_{i\geq 0} c_i^{(m)} A^{i+1} \right\|}{\|A\|} \backsimeq \frac{\left\| \sum\limits_{i\geq m+1} c_i^{(m)} A^{i+1} \right\|}{\|A\|} \leq \left\| \sum_{i\geq m} c_i^{(m)} A^i \right\|.$$

If we define $h_m(x) = \sum\limits_{i\geqslant m} c_i^{(m)} x^i$ and $\tilde{h}_m(x) = \sum\limits_{i\geq m} \left| c_i^{(m)} \right| x^i$, then

$$E_b \leq \|h_m(A)\| \leq \tilde{h}_m(\alpha_m). \quad (13)$$

Let Θ_m be

$$\Theta_m = \max\left\{\theta \geq 0 :\ \sum_{i\geq m}\left|c_i^{(m)}\right|\theta^i \leq u\right\}, \tag{14}$$

where u is the unit roundoff in IEEE double precision arithmetic ($u = 2^{-53}$). For computing Θ_m, the MATLAB Symbolic Math Toolbox has been used.

On the other hand, if $\alpha_m < \Theta_m$, then

$$E_b \leqslant \|h_m(A)\| \leqslant \tilde{h}_m(\alpha_m) \leqslant \tilde{h}_m(\Theta_m) \leqslant u. \tag{15}$$

Hence, if $\alpha_m < \Theta_m$, then $E_b \leq u$; if not, we would find a value of s such that $2^{-s}\alpha_m < \Theta_m$ (in this case we compute $P_{m_k}(2^{-s}A)$ in step 3 of Algorithm 1) or a value of s such that $4^{-s}\alpha_m < \Theta_m$ (in this case we compute $P_{m_k}(4^{-s}A)$ in step 3 of Algorithm 2).

The norms $||A^m||^{1/m}$ can be computed approximately using matrix powers previously computed based on the estimation of norms of matrix powers block 1-norm estimation algorithm from [**?**]. Taking into account that analysis, Algorithms 3, 4 and 5 have been developed.

Algorithms 3 and 4 initially check whether there is a value m_i, $m_{\min} \leq m_i \leq m_{\max}$, such that $\alpha_{m_i} \leq \Theta_{m_i}$, computing the necessary powers of matrix A to obtain $P_i(A)$ from (10) as i increases. The values of $m_{\min}$ and $m_{\max}$ can be varied in the developed implementations. If there is such value m_i, then we choose the lower order $m = m_i$ such that $\alpha_{m_i} \leq \Theta_{m_i}$ and $s = 0$. Otherwise, we choose $m = m_{\max}$ and

$$s = \max\left\{0, \left\lceil f_s \log\left(\frac{\alpha_{m_{\max}}}{\Theta_{m_{\max}})}\right)\right\rceil\right\},$$

where $fs = 1$, if Algorithm 1 is used, or $fs = 0.5$, if Algorithm 2 is used. In Algorithm 3 the lines 20-29 test if it is possible to decrease the above values s and m (the initial value of m is $m_{\max}$), so that

$$\tilde{h}_m(2^{-s}\alpha_m) \leqslant \tilde{h}_m(\Theta_m) \leqslant u$$

is fulfilled, when Algorithm 1 is used, or

$$\tilde{h}_m(4^{-s}\alpha_m) \leqslant \tilde{h}_m(\Theta_m) \leqslant u$$

it is fulfilled, when Algorithm 2. In those cases, those new values are considered.

In Algorithm 4 the lines 19-24 test if it is possible to decrease the value s so that

$$|p_{m_{\max}}|a_{m_{\max}}2^{(1-s)m_{\max}} < u,$$

if Algorithm 1 is used, or

$$|p_{m_{\max}}|a_{m_{\max}}4^{(1-s)m_{\max}} < u,$$

if Algorithm 2 is used, where $p_{m_{\max}}$ is the coefficient of the term of order $m_{\max}$ of Bernouilli series.

Unlike algorithms 3 and 4, Algorithm 5 does not compute the powers of matrix A, so the estimate (12) is obtained only from A. Algorithm 5 computes α_m such that

$$\left|\frac{\alpha_{m_i} - \alpha_{m_{i-1}}}{\alpha_{m_i}}\right| < tol,$$

where tol is a small prefixed value (see Lines 3-11). Then Algorithm 5 check if $m_i \leq m_{\max}$ such that $\alpha_{_{m_i}} \leq \Theta_{m_i}$. In this case, we choose the lower order $m = m_i$ such that $\alpha_{m_i} \leq \Theta_{m_i}$ and $s = 0$. Otherwise, we choose $m = m_{\max}$ and

$$s = \max\left\{0, \left\lceil f_s \log\left(\frac{\alpha_{m_{\max}}}{\Theta_{m_{\max}})}\right)\right\rceil\right\}.$$

Next, in the lines 23-32 of Algorithm 5 is checked if it is possible to decrease the above values s and m so that

$$\tilde{h}_m(2^{-s}\alpha_m) \leqslant \tilde{h}_m(\Theta_m) \leqslant u$$

is fulfilled, when Algorithm 1 is used, or

$$\tilde{h}_m(4^{-s}\alpha_m) \leqslant \tilde{h}_m(\Theta_m) \leqslant u$$

it is fulfilled, when Algorithm 2. In those cases, those new values are considered.

4 Numerical Experiments

Although theoretically, and according to formulation (8), all terms occupying odd positions should be equal to 0, in practice it might not happen with all of them. As an example, table 1 shows the coefficients of the Bernoulli

Algorithm 3 Given a matrix $A \in \mathbb{C}^{n\times n}$, a minimum order $m_{\min} \in \mathbb{M}$ and a maximum order $m_{\max} \in \mathbb{M}$, this algorithm calculates an order $m_i \in \mathbb{M}$, $m_{\min} \leq m \leq m_{\max}$, a factor s and several powers of A for computing C in PHASE II.

```
1:  $A_1 = A$; $i = \min$; $f$=0
2:  while $f = 0$ and $i \leq \max$ do
3:      $v = \sqrt{m_i}$
4:      $j = \lceil v \rceil$
5:      if $v > j$ then
6:          $A_j = A_{j-1}A$
7:      end if
8:      $\alpha_{m_i} \approx \|A^{m_i}\|^{1/m_i}$ from $A_j$          ▷ based on Algorithm 1 from [?]
9:      if $\alpha_{m_i} < \Theta_{m_i}$ then
10:         $f = 1$
11:     else
12:         $i = i + 1$
13:     end if
14: end while
15: if $f = 1$ then
16:     $s = 0$
17: else
18:     $s = \lceil \max(0, f_s \log_2(\alpha_{m_{\max}}/\Theta_{m_{\max}})) \rceil$ or
19:     $j = i_{\min}$
20:     $f = 0$
21:     while $f = 0$ do
22:         $j = j - 1$
23:         $s_1 = \lceil \max\left(0, f_s \log_2(\alpha_{m_j}/\Theta_{m_j})\right) \rceil$
24:         if $s \geq s_1$ and $j \geq i_{\min}$ then
25:             $s = s_1$
26:             $i = j$
27:         else
28:             $f = 1$
29:         end if
30:     end while
31: end if
32: $m = m_i$
```

Algorithm 4 Given a matrix $A \in \mathbb{C}^{n\times n}$, a minimum order $m_{\min} \in \mathbb{M}$ and a maximum order $m_{\max} \in \mathbb{M}$, this algorithm calculates an order $m \in \mathbb{M}$, $m_{\min} \le m \le m_{\max}$, a scaling factor s and the necessary powers of A for computing C in PHASE II.

```
1: $A_1 = A$; $i = \min$; $f$=0
2: while $f = 0$ and $i \le \max$ do
3:     $v = \sqrt{m_i}$
4:     $j = \lceil v \rceil$
5:     if $v > j$ then
6:         $A_j = A_{j-1}A$
7:     end if
8:     Compute $a_{m_i} \approx \|A^{m_i}\|$ from $A^j$          ▷ based on Algorithm 1 from [?]
9:     $\alpha_{m_i} = \sqrt[m_i]{a_{m_i}}$
10:    if $\alpha_{mi} < \Theta_{m_i}$ then
11:        $f = 1$
12:    else
13:        $i = i + 1$
14:    end if
15: end while
16: if $f = 1$ then
17:     $s = 0$
18: else
19:     $i = i_{\max}$
20:     $s = \lceil \max(0, f_s \log_2(\alpha_{m_i}/\Theta_{mi})) \rceil$
21:     if $|p_{m_i}| a_{m_i} r^{(-s+1)m_i} < u$ then          ▷ $r = 2$ (Algorithm 1)
22:                                                          ▷ $r = 4$ (Algorithm 2)
23:         $s = s - 1$
24:         if $|p_{m_i}| a_{m_i} r^{(-s+1)m_i} < u$ then
25:             $s = s - 1$
26:         end if
27:     end if
28: end if
29: $m = m_i$
```

polynomial $B_m(A) = a_m A^m + a_{m-1}A^{m-1} + \cdots + a_1 A + a_0 I$ for formulae (6) and (8) when m=25. As it can be seen in the third column of the table, most of the odd terms are not equal to 0, although they are close to it. This

Table 1: Bernoulli polynomial coefficientes for m=25.

Coefficients	Expression (6)	Expression (8)
a_0	1.000000000000000e+00	9.999999999999776e-01
a_1	4.268425513808927e-17	2.200931543663476e-25
a_2	-5.000000000000000e-01	-4.999999999998889e-01
a_3	-7.103351130950067e-18	9.114115486610701e-26
a_4	4.166666666666666e-02	4.166666666657532e-02
a_5	3.340635907377075e-19	1.642205248036368e-25
a_6	-1.388888888888889e-03	-1.388888888858840e-03
a_7	1.188305192257936e-20	2.406735243784263e-26
a_8	2.480158730158730e-05	2.480158729629132e-05
a_9	-1.104189679966496e-20	-2.741989705063794e-28
a_{10}	-2.755731922398609e-07	-2.755731916590913e-07
a_{11}	4.004071095795792e-21	-2.506940767309528e-29
a_{12}	2.087675698787421e-09	2.087675655363431e-09
a_{13}	-1.013606842281333e-21	-1.827655933239498e-31
a_{14}	-1.147074559786225e-11	-1.147074324303990e-11
a_{15}	1.905862492984388e-22	-3.648057207260014e-33
a_{16}	4.779477334567797e-14	4.779467650734273e-14
a_{17}	-2.768223171137328e-23	2.331854748958815e-35
a_{18}	-1.561920725009726e-16	-1.561889490721346e-16
a_{19}	3.205121320979757e-24	0
a_{20}	4.110320556779777e-19	4.109509217914593e-19
a_{21}	-3.051721100969466e-25	0
a_{22}	-8.897045307814457e-22	-8.879692513470797e-22
a_{23}	2.530017510290091e-26	0
a_{24}	1.613667223591245e-24	1.582268801402494e-24
a_{25}	-2.511873775100074e-27	0

raises the dilemma of converting these values directly into zero and taking into account only the even terms or keeping them as they are and considering all of them.

Therefore, having in mind the expression (6), the two different mentioned above alternatives that derive from the expression (8) and the three distinct algorithms, described in section 3, to compute the polynomial degree m and the scaling parameter s, nine different approximations are given in this paper to compute cosine matrix function. To test and compare the numerical performance of all these different approaches, the following algorithms have been implemented on MATLAB R2018b:

- *cosmber_1_3*, *cosmber_1_4* and *cosmber_1_5*: codes based on formula (6), where all the polynomial terms must be considered. Algorithms 3, 4 and 5 will be used in each code, respectively, to compute $m \in \{30, 36\}$ and s values.

- *cosmber_2_3*, *cosmber_2_4* and *cosmber_2_5*: implementations belonging to formula (8). As in the previous case, even and odd terms will be taken into account. Algorithms 3, 4 and 5 will be also respectively considered to calculate $m \in \{30, 36\}$ and s parameters.

- *cosmber_3_3*, *cosmber_3_4* and *cosmber_3_5*: developments derived from formula (8) where odd position terms have been neglected. In this way, only the even terms will be employed, as in the case of Taylor series occur [10]. One more time, the same algorithms 3, 4 and 5 will be employed to work out $m \in \{16, 20\}$ (it would be equivalent to $m = 32$ or 40 using the even and odd terms) and s.

- *cosm*: implementation based on the Padé rational approximation for the cosine matrix function [14].

MATLAB's Symbolic Math Toolbox with 256 digits of precision was run to compute the exact matrix cosine function, using the vpa (variable-precision floating-point arithmetic) function.

The next test battery, composed of three types of different and representative matrices, have been used to compare the above mentioned algorithms:

a) Diagonalizable real matrices: they have been obtained as $A = V \cdot D \cdot V^{-1}$, where D is a diagonal matrix (with real and complex eigenvalues) and matrix V is an orthogonal matrix, $V = H/\sqrt{n}$, being H a Hadamard matrix and n its number of rows or columns. We have $2.18 \leq \|A\|_1 \leq 132.62$. The matrix cosine was exactly calculated as $\cos(A) = V \cdot \cos(D) \cdot V^T$.

b) Non-diagonalizables complex matrices: they have been computed as $A = V \cdot J \cdot V^{-1}$, where J is a Jordan matrix with complex eigenvalues with module less than 10 and random algebraic multiplicity between 1 and 5. V is an orthogonal random matrix with elements in the interval $[-0.5, 0.5]$. We have $91.3 \leq \|A\|_1 \leq 92.6$. The *"exact"* matrix cosine was computed as $\cos(A) = V \cdot \cos(J) \cdot V^{-1}$.

c) **Matrices from the Matrix Computation Toolbox (MCT)** [18] and from the **Eigtool MATLAB Package (EMP)** [19]: they have been chosen because they have highly different and significant characteristics from each other. The *"exact"* matrix cosine for these matrices was computed by using Taylor approximations of different orders, changing their scaling parameter.

In the numerical experiments, we used successfully 179 matrices of size 128×128: 60 from the diagonalizable set, 60 from the non-diagonalizable group, 42 from the MCT and 17 from the EMP. Although the Matrix Computation Toolbox and the Eigtool Matlab Package are initially composed of fifty-two and twenty matrices, respectively, ten from the MCT and three from the EMP matrices were discarded for different reasons. For example, matrices 5, 15, 16, 17, 21, 42, 43, 44 and 49 belonging to the MCT and matrix 3 from the EMP were not taken into account since the exact cosine solution could not be computed. Besides, matrix 2 from the MCT and matrices 4 and 10 from the EMP were not considered because the excessively high relative error provided by all the methods to be compared.

Initially, three experiments, called Tests, have been independently performed to evaluate the distinct variations of each theoretical formulation, taking into account the corresponding approaches related to algorithms 3, 4 and 5 to compute m and s parameters. All these executions have been carried out by means of MATLAB (R2018b) running on an HP Pavilion dv8 Notebook PC with an Intel Core i7 CPU Q720 @1.60Ghz processor and 6 GB of RAM.

In the first test, *cosmber_1_3*, *cosmber_2_3* and *cosmber_3_3* codes have been compared to obtain the most appropiate theoretical approach when the algorithm 3 is used.

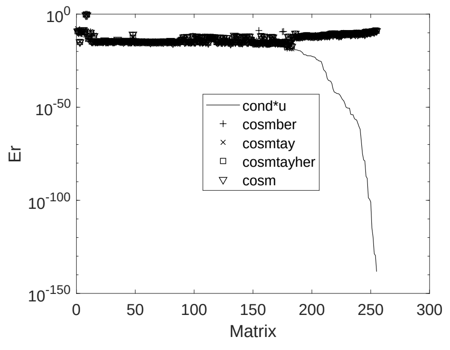
Results are given in Tables 2 and 3. The rows of each table show the percentage of cases in which the relative errors of `cosmber` (Bernoulli) is lower, greater or equal than the relative errors of `cosmtay` (Taylor), `cosmtayher` (Hermite) and `cosm` (Padé). Graphics of the Normwise relative errors and the Performance Profile are given in Figures 1 and 2. The total number of matrix products was: 3202 (*cosmber*), 2391 (*cosmtay*), 1782 (*cosmtayher*) and 3016 (*cosm*). Recall that in the Bernoulli implementation, the maximum value of m to be used was $m = 36$ considering all the terms and, in the rest of algorithms, was $m = 32$ but just having into account the even terms.

Table 2: Using approximation (6)

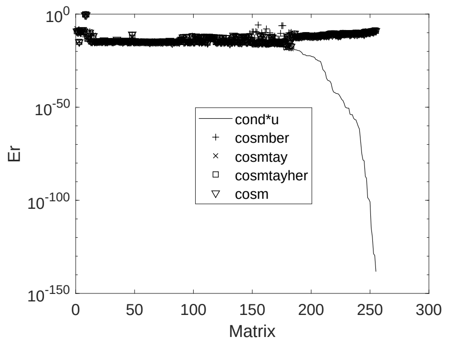
$E(cosmber) < E(cosmtay)$	55.60%
$E(cosmber) > E(cosmtay)$	44.40%
$E(cosmber) = E(cosmtay)$	0%
$E(cosmber) < E(cosmtayher)$	50.97%
$E(cosmber) > E(cosmtayher)$	49.03%
$E(cosmber) = E(cosmtayher)$	0%
$E(cosmber) < E(cosm)$	76.83%
$E(cosmber) > E(cosm)$	23.17%
$E(cosmber) = E(cosm)$	0%

Table 3: Using approximation (8)

$E(cosmber) < E(cosmtay)$	65.64%
$E(cosmber) > E(cosmtay)$	34.36%
$E(cosmber) = E(cosmtay)$	0%
$E(cosmber) < E(cosmtayher)$	60.62%
$E(cosmber) > E(cosmtayher)$	39.38%
$E(cosmber) = E(cosmtayher)$	0%
$E(cosmber) < E(cosm)$	73.75%
$E(cosmber) > E(cosm)$	26.25%
$E(cosmber) = E(cosm)$	0%



(a) Using formula (6)

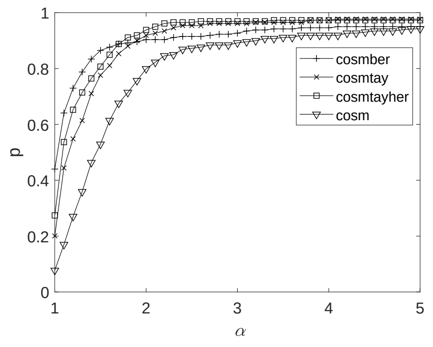


(b) Using formula (8)

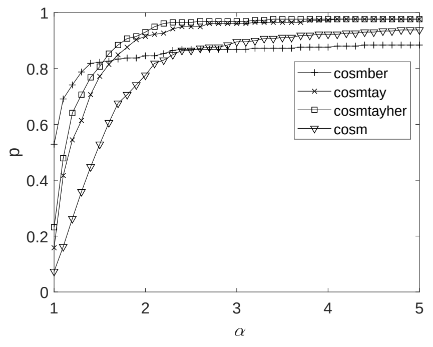
Figure 1: Normwise relative errors.

5 Conclusions

In general, the implementation based on the new Bernoulli series (6) is more accurate than (8), comparing it with the one based on the Taylor series, algorithm (`cosmtay`) and Hermite series, algorithm (`cosmtayher`), and the one based in Padé rational approximation, algorithm (`cosm`).



(a) Using formula (6)



(b) Using formula (8)

Figure 2: Performance Profile.

References

[1] Ernesto Estrada, Desmond J Higham, and Naomichi Hatano. Communicability and multipartite structures in complex networks at negative absolute temperatures. *Physical Review E*, 78(2):026102, 2008.

[2] Ernesto Estrada and Juan A Rodríguez-Velázquez. Spectral measures of bipartivity in complex networks. *Physical Review E*, 72(4):046105, 2005.

[3] Ernesto Estrada and Jesús Gómez-Gardeñes. Network bipartivity and the transportation efficiency of european passenger airlines. *Physica D: Nonlinear Phenomena*, 323:57–63, 2016.

[4] L. Jódar, E. Navarro, AE. Posso, and MC. Casabán. Constructive solution of strongly coupled continuous hyperbolic mixed problems. *Applied Numerical Mathematics*, 47(3–4):477–492, 2003.

[5] Steven M. Serbin and Sybil A. Blalock. An algorithm for computing the matrix cosine. *SIAM Journal on Scientific Computing*, 1(2):198–204, 1980.

[6] Mehdi Dehghan and Masoud Hajarian. Computing matrix functions using mixed interpolation methods. *Mathematical and Computer Modelling*, 52(5-6):826–836, 2010.

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

[8] Pedro Alonso-Jordá, Jesús Peinado, Javier Ibáñez, Jorge Sastre, and Emilio Defez. Computing matrix trigonometric functions with GPUs through Matlab. *The Journal of Supercomputing*, pages 1–14, 2018.

[9] Ch Tsitouras and Vasilios N Katsikis. Bounds for variable degree rational L_∞ approximations to the matrix cosine. *Computer Physics Communications*, 185(11):2834–2840, 2014.

[10] J. Sastre, J. Ibáñez, P. Alonso-Jordá, J. Peinado, and E. Defez. Two algorithms for computing the matrix cosine function. *Applied Mathematics and Computation*, 312:66–77, 2017.

[11] Jorge Sastre, Javier Ibáñez, Pedro Alonso-Jordá, Jesús Peinado, and Emilio Defez. Fast Taylor polynomial evaluation for the computation of the matrix cosine. *Journal of Computational and Applied Mathematics*, 354:641–650, 2019.

[12] Emilio Defez, Javier Ibáñez, Jesús Peinado, Jorge Sastre, and Pedro Alonso-Jordá. An efficient and accurate algorithm for computing the matrix cosine based on new Hermite approximations. *Journal of Computational and Applied Mathematics*, 348:1–13, 2019.

[13] S.M. Serbin. Rational approximations of trigonometric matrices with application to second-order systems of differential equations. *Applied Mathematics and Computation*, 5(1):75–92, 1979.

[14] Awad H. Al-Mohy, Nicholas J. Higham, and Samuel D. Relton. New algorithms for computing the matrix sine and cosine separately or simultaneously. *SIAM Journal on Scientific Computing*, 37(1):A456–A487, 2015.

[15] Omran Kouba. Lecture Notes, Bernoulli Polynomials and Applications. *arXiv preprint arXiv:1309.7560*, 2013.

[16] Frank WJ Olver, Daniel W Lozier, Ronald F Boisvert, and Charles W Clark. *NIST handbook of mathematical functions hardback and CD-ROM.* Cambridge University Press, 2010.

[17] Emilio Defez, Javier Ibáñez, Jesús Peinado, Pedro Alonso-Jordá, and José M. Alonso. Computing matrix functions by matrix Bernoulli series. In *19th International Conference on Computational and Mathematical Methods in Science and Engineering (CMMSE-2019)*, From 30th of Juny to 6th of July 2019. Poster presented at some conference, Rota (Cádiz), Spain.

[18] N. J. Higham. *The test matrix toolbox for MATLAB (Version 3.0)*. University of Manchester Manchester, 1995.

[19] TG Wright. Eigtool, version 2.1. *URL: web. comlab. ox. ac. uk/pseudospectra/eigtool*, 2009.

Algorithm 5 Given a matrix $A \in \mathbb{C}^{n\times n}$ and a small value *tol*, this algorithm calculates calculates an order $m \in \mathbb{M}$, $m_{\min} \le m \le m_{\max}$, and the scaling factor s.

```
1: $i = \min$; $f = 0$;
2: Compute $\alpha_0 \approx \|A^{m_i}\|^{1/m_i}$ from $A$          ▷ based on Algorithm 1 from [?]
3: **while** $f = 0$ and $i < \max$ **do**
4:     $i = i + 1$
5:     Compute $\alpha \approx \|A^{m_i}\|^{1/m_i}$ from $A$      ▷ based on Algorithm 1 from [?]
6:     **if** $|\alpha - \alpha_0| > \alpha \cdot tol$ **then**
7:         $\alpha_0 = \alpha$
8:     **else**
9:         $f = 1$
10:    **end if**
11: **end while**
12: $i = \min$; $f = 0$
13: **while** $f = 0$ and $i \le \max$ **do**
14:     **if** $\alpha < \Theta_{m_i}$ **then**
15:         $f = 1$
16:     **else**
17:         $i = i + 1$
18:     **end if**
19: **end while**
20: **if** $f = 1$ **then**
21:     $s = 0$
22: **else**
23:     $i = i_{\max}$
24:     $s = \lceil \max(0, f_s \log_2(\alpha/\Theta_{m_{\max}})) \rceil$
25:     $f = 0$
26:     $j = i$
27:     **while** $f = 0$ **do**
28:         $j = j - 1$
29:         $s_1 = \left\lceil \max\left(0, f_s \log_2(\alpha/\Theta_{m_j})\right) \right\rceil$
30:         **if** $s \ge s_1$ **then**
31:             $s = s_1$
32:             $i = j$
33:         **else**
34:             $f = 1$
35:         **end if**
36:     **end while**
37: **end if**
38: $m = m_i$
```