On Bernoulli series approximation for the matrix cosine[*]

Emilio Defez[⋆], Javier Ibáñez[†], José M. Alonso[†], Pedro Alonso-Jordá[♮]

⋆ Instituto de Matemática Multidisciplinar.

† Instituto de Instrumentación para Imagen Molecular.

♮ Departamento de Sistemas Informáticos y Computación.

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

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

Abstract

This paper presents a new series expansion based on Bernoulli matrix polynomials to approximate the matrix cosine function. An approximation based on this series is not a straightforward exercise since there exist different options to implement such a solution. We dive into these options and include a thorough comparative of performance and accuracy in the experimental results section that shows benefits and downsides of each one. Also, a comparison with the Padé approximation is included. The algorithms have been implemented in MATLAB and in CUDA for NVIDIA GPUs.

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

[*]**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.

already well-known. Of all matrix functions, it is certainly the matrix exponential which attracts much of the attention because of its connection with systems of first order linear differential equations

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

whose solution is given by $Y(t) = e^{At}Y_0$ and where $\mathbb{C}^{r\times r}$ represents the set of all complex square matrices of size r. 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 differential equations [4]. In particular, the sine and cosine trigonometric matrix functions have been proven to be especially useful for 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 function can be calculated using the same methods as for the matrix cosine one. Usually, research is concentrated on developing efficient state-of-the-art algorithms to compute the matrix cosine function approximately. The main of these methods and algorithms can be found in references [5–8]. Other algorithms, for normal and non-negative matrices, which are based on approximations L_∞ have been presented in [9]. Alternative methods for computing matrix functions using interpolation techniques is given in [6]. Methods based on finite differences, and its application to solve fractional partial differential equations, can be found in references [10–12].

Among the methods proposed to approximate the matrix cosine, two fundamentally stand out: 1) those focused on *polynomial approximations*, thanks to the developments of the matrix cosine in Taylor or Hermite series (see [13–15]); or 2) those based on *rational approximations*, such as Padé approach (see [5,9,16,17]). In general, polynomial methods are more efficient in terms of accuracy than rational ones, although they may be somewhat more computationally expensive.

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

In this paper, Bernoulli matrix polynomials are defined and a new series expansion of the matrix sine and cosine functions in terms of them is presented. Then, this series expansion is evaluated to analyse if, indeed, provides a new and efficient method to approximate the matrix cosine.

The organization of the paper is as follows: in Section 2, two series expansions of the matrix cosine in terms of the Bernoulli matrix polynomials are described. Then, the algorithms in charge of computing the matrix cosine function and those ones responsible for providing the most appropriate polynomial order and the scaling parameter are presented in Section 3. Next, different experiments, that have been performed to compare the numerical performance of the distinct implemented MATLAB codes, are incorporated in Section 4, together with their migration and execution on a GPU-based parallel computing platform. Finally, conclusions are given in Section 5.

Throughout this paper, we denote by $\mathbb{C}^{n\times n}$ the set of all complex square matrices of order n, by I the identity matrix, as mentioned before, and by $\rho(A)$ its spectral radius. A polynomial of degree m is given by an expression of the form $P_m(x) = p_m x^m + p_{m-1}x^{m-1} + \cdots + p_1 x + p_0$, where x is the variable (real or complex) and the coefficients $p_j, 0 \le j \le m$, are complex numbers. Moreover, we can define the matrix polynomial $P_m(A)$, for $A \in \mathbb{C}^{n\times n}$, as the expression $P_m(A) = p_m A^m + p_{m-1}A^{m-1} + \cdots + p_1 A + p_0 I$. With $\lceil r \rceil$, we denote the result of rounding a real number r to the nearest integer greater than or equal to r, and $\lfloor r \rfloor$ refers to the result of rounding r to the nearest integer less than or equal to r. As usual, the matrix norm $\|\cdot\|$ symbolizes any subordinate matrix norm; in particular $\|\cdot\|_1$ is the usual 1-norm.

2 On Bernoulli matrix polynomials

The sequence of Bernoulli polynomials, denoted by $\{B_n(x)\}_{n\geq 0}$, and Bernoulli numbers, $\mathcal{B}_n = B_n(0)$, appear in important applications of different areas of mathematics, from number theory to classical analysis. For example, they are used for representing the remainder term of the composite Euler-McLaurin quadrature rule. They also appear in the Taylor expansion in the neighbourhood of the origin of circular and hyperbolic tangent and co-tangent functions. Moreover, this sequence expresses the exact value of $\zeta(2p)$, where p is a positive integer and $\zeta(z) = \sum_{k\geq 1} \frac{1}{k^z}$ is the well-known Riemann's zeta function.

These polynomials and numbers 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 $B_n(x)$ 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 variable t, that has an avoidable singularity in $t = 0$ [19, p. 588]. Bernoulli polynomials have the explicit expression

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

where the Bernoulli numbers $\mathcal{B}_n$ satisfy the following recurrence relation (formula (24.5.3) from [19, p. 591]):

$$\mathcal{B}_0 = 1, \sum_{k=0}^{n-1} \binom{n}{k} \mathcal{B}_k = 0, \, n \geq 2.$$

From this last relation, the explicit expression for the Bernoulli numbers can be derived:

$$\mathcal{B}_0 = 1, \mathcal{B}_n = -\sum_{k=0}^{n-1} \binom{n}{k} \frac{\mathcal{B}_k}{n+1-k}, n \geq 1. \tag{3}$$

Notice that all Bernoulli numbers with odd index vanish, except $\mathcal{B}_1 = -1/2$.

For a matrix $A \in \mathbb{C}^{r\times r}$, we define the n-th Bernoulli matrix polynomial by the expression

$$B_n(A) = \sum_{k=0}^{n} \binom{n}{k} \mathcal{B}_k A^{n-k}. \tag{4}$$

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 [20]. An efficient method based on (5) to approximate the exponential matrix also was presented and developed in [20].

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

$$\begin{aligned} \cos(A) &= (\cos(1) - 1) \sum_{n\geq 0} \frac{(-1)^n B_{2n+1}(A)}{(2n+1)!} + \sin(1) \sum_{n\geq 0} \frac{(-1)^n B_{2n}(A)}{(2n)!}, \\ \sin(A) &= \sin(1) \sum_{n\geq 0} \frac{(-1)^n B_{2n+1}(A)}{(2n+1)!} - (\cos(1) - 1) \sum_{n\geq 0} \frac{(-1)^n B_{2n}(A)}{(2n)!}. \end{aligned}$$

Nevertheless, we will use the truncated series

$$\cos(A) \approx P_m(A) = (\cos(1) - 1) \sum_{n=0}^{m} \frac{(-1)^n B_{2n+1}(A)}{(2n+1)!} + \sin(1) \sum_{n=0}^{m} \frac{(-1)^n B_{2n}(A)}{(2n)!}, \tag{6}$$

to provide a first approximation to the matrix cosine.

On the other hand, replacing t by it ($i = \sqrt{-1}$), firstly, and by $-it$, secondly, in (5), and then calculating the arithmetic mean of the corresponding results, it is obtained that

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

Now, 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 (8), only the Bernoulli polynomials with even index appear. As previously, we will use the truncated series

$$\cos(A) \approx P_m(A) = \sin(1) \sum_{n=0}^{m} \frac{(-1)^n 2^{2n} B_{2n}\left(\frac{A+I}{2}\right)}{(2n)!}, \tag{9}$$

to obtain a second approximation to the matrix cosine.

3 Algorithms

The most efficient and accurate start-of-the-art algorithms that can be found in the literature, for matrix functions computation, are those based on polynomial or rational approximations. Basically, all of them consist of truncating the appropriate series so that the relative forward or backward errors are smaller than the unit roundoff in double precision floating-point arithmetic, determining the polynomial optimal order m and scaling, if necessary, the matrix. This type of methods can be used as long as the matrix function (exponential, trigonometric, logarithm, p-th root, and so on) can be recovered from its computation to the scaled matrix.

Actually, the two truncated series (6) or (9) can be expressed in explicit terms of powers of matrix A,

$$P_m(A) = \sum_{i=0}^{m} p_i^{(m)} A^i, \tag{10}$$

where coefficients $p_i^{(m)}$ depend on the integer m and the truncated expression employed, and where $P_m(A)$ represents the matrix polynomial of order m corresponding to the Bernoulli approximation of the cosine function of matrix A. These coefficients converge to those of the Taylor series for increasing values of m.

Taking into account the two previous truncated series, three different approximations have been addressed: two of them are based on expressions (6) or (9), respectively, in which all the polynomial terms have been considered; and one more, derived from expression (9), where only the coefficients $p_i^{(m)}$ of the even order terms have been taken into account, similarly to what happens when considering cosine series expansions using Taylor or Hermite polynomials [13,15].

According to the above mentioned approaches we have developed Algorithms 1 and 2. In Phase I (for both algorithms), integers m and s are estimated so that the Bernoulli approximation of the scaled matrix is computed accurately and efficiently. There exist several methods that can be applied to compute efficiently $C = P_{m_k}(A)$ in Phase II [21,22]. In our implementations, we have used those based on the Paterson-Stockmeyer's method [21]. According to it, an integer m_k (order of the Bernoulli approximation polynomial) is chosen from the set

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

Algorithm 1: Given a matrix $A \in \mathbb{C}^{n\times n}$, this algorithm computes $C = \cos(A)$ by Bernoulli series (6) or (9), where all coefficients are considered.

```
1 Select suitable 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 Bernoulli approximation (6) or (9) */
4 for $i = 1$ to $s$ do /* Phase III: Recovering $\cos(A)$ */
5     $C = 2C^2 - I$
6 end
```

Algorithm 2: Given a matrix $A \in \mathbb{C}^{n\times n}$, this algorithm computes $C = \cos(A)$ by Bernoulli series (9), where only the coefficients of the even order terms are considered.

```
1 Select suitable 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 Bernoulli approximation (9) */
4 for $i = 1$ to $s$ do /* Phase III: Recovering $\cos(A)$ */
5     $C = 2C^2 - I$
6 end
```

Then, powers A^i, $2 \le i \le q$, are calculated, being $q = \lceil \sqrt{m_k} \rceil$ or $q = \lfloor \sqrt{m_k} \rfloor$ a divisor of the integer m_k. With these matrix powers, we can efficiently compute $C = P_{m_k}(A)$ as

$$
\begin{aligned}
&P_{m_k}(A) = \qquad (11)\\
&(((p_{m_k}A^q + p_{m_k-1}A^{q-1} + p_{m_k-2}A^{q-2} + \cdots + p_{m_k-q+1}A + p_{m_k-q}I)A^q\\
&\qquad + p_{m_k-q-1}A^{q-1} + p_{m_k-q-2}A^{q-2} + \cdots + p_{m_k-2q+1}A + p_{m_k-2q}I)A^q\\
&\qquad + p_{m_k-2q-1}A^{q-1} + p_{m_k-2q-2}A^{q-2} + \cdots + p_{m_k-3q+1}A + p_{m_k-3q}I)A^q\\
&\qquad \cdots\\
&\qquad + p_{q-1}A^{q-1} + p_{q-2}A^{q-2} + \cdots + p_1A + p_0I.
\end{aligned}
$$

The computational cost of (11), in terms of matrix products, is k.

The calculation of m and s in Phase I of Algorithms 1 and 2 is based on the relative backward error of approximating $\cos(A)$ using (10). This error is defined as a matrix ΔA such that $\cos(A + \Delta A) = P_m(A)$. We bound the relative backward error as

$$
E_{rb} = \frac{\|\Delta A\|}{\|A\|} = \frac{\left\|\sum\limits_{i\ge 0} c_i^{(m)} A^i\right\|}{\|A\|} \backsimeq \frac{\left\|\sum\limits_{i\ge m+1} c_i^{(m)} A^i\right\|}{\|A\|} \le \left\|\sum_{i\ge m} c_{i+1}^{(m)} A^i\right\|.
$$

If we define $h_m(x) = \sum\limits_{i \geqslant m} c_{i+1}^{(m)} x^i$, $\tilde{h}_m(x) = \sum\limits_{i \geq m} \left| c_{i+1}^{(m)} \right| x^i$ and we apply Theorem 1.1 from [23] for $a_k = ||A^k||$, considering $p = m$ and $l = m$, then

$$||h_m(A)|| \leq \tilde{h}_l(\alpha_m),$$

where $\alpha_m = \max\{||A^k||^{\frac{1}{k}} : k = m, m+1, m+2, \ldots, 2m-1\}$. Hence

$$E_{rb} \leq \|h_m(A)\| \leq \tilde{h}_m(\alpha_m). \qquad (12)$$

Let Θ_m be

$$\Theta_m = \max\left\{ \theta \geq 0 : \sum_{i \geq m} \left| c_{i+1}^{(m)} \right| \theta^i \leq u \right\}, \qquad (13)$$

where $u = 2^{-53}$ is the unit roundoff in IEEE double precision arithmetic.

If $\alpha_m < \Theta_m$, then we have

$$E_{rb} \leqslant \|h_m(A)\| \leqslant \tilde{h}_m(\alpha_m) \leqslant \tilde{h}_m(\Theta_m) \leqslant u, \qquad (14)$$

and the polynomial order m will be obtained and the scaling parameter s will be set to 0. Otherwise, we should find values of m and s such that $2^{-s}\alpha_m < \Theta_m$, if $P_{m_k}(2^{-s}A)$ had to be computed in step 3 of Algorithm 1, or values of m and s such as $4^{-s}\alpha_m < \Theta_m$, if $P_{m_k}(4^{-s}A)$ had to be estimated in the same step of Algorithm 2. In our case, MATLAB Symbolic Math Toolbox has been used to compute Θ_m.

In this paper, we propose to use the approximation

$$\alpha_m \approx ||A^m||^{1/m}, \qquad (15)$$

where $\lim\limits_{m \to \infty} ||A^m||^{1/m} = \rho(A)$. Experimental results show that this is a good approach when considering relatively high values for m, such as $m \geq 30$. The norm $||A^m||$ can be computed approximately by using the one-norm estimation algorithm from [24].

Algorithms 3, 4 and 5 have been developed with this theoretical analysis in mind. For the sake of simplicity, the following notation has been used in Algorithms 3, 4 and 5:

$$\alpha_i \equiv \alpha_{m_i}, \ \Theta_i \equiv \Theta_{m_i}.$$

Algorithm 3: Given a matrix $A \in \mathbb{C}^{n\times n}$, a minimum order $m_{lower} \in \mathbb{M}$ and a maximum order $m_{upper} \in \mathbb{M}$, this algorithm provides an order $m \in \mathbb{M}$, $m_{lower} \leq m \leq m_{upper}$, a factor s and several powers of A.

```
$A_1 = A$; $i = lower$; $f = 0$
for $j = 2$ to $\lceil\sqrt{m_i}\rceil$ do
    $A_j = A_{j-1}A$
end
while $f = 0$ and $i \leq upper$ do
    $v = \sqrt{m_i}$
    $j = \lceil v \rceil$
    if $j > v$ then $A_j = A_{j-1}A$
    $\alpha_i \approx \|A^{m_i}\|^{1/m_i}$ from $A_j$ /* based on Algorithm 1 from [24] */
    if $\alpha_i < \Theta_i$ then $f = 1$
    else $i = i + 1$
end
if $f = 1$ then $s = 0$
else
    $i = upper$
    $s = \lceil\max(0, f_s\log_2(\alpha_i/\Theta_i))\rceil$ /* $f_s = 1$ (Alg. 1) or $f_s = 0.5$ (Alg. 2) */
    $j = i$
    while $f = 0$ and $j > lower$ do
        $j = j - 1$
        $s_1 = \lceil\max(0, f_s\log_2(\alpha_j/\Theta_j))\rceil$
        if $s \geq s_1$ then
            $s = s_1$
            $i = j$
        else $f = 1$
    end
end
$m = m_i$
```

Algorithms 3 and 4 try to find out the minimum value m_i, $m_{lower} \leq m_i \leq m_{upper}$, such that $\alpha_i \leq \Theta_i$, computing the necessary powers of matrix A to obtain $P_{m_i}(A)$ from (11) as i increases. Values of polynomial orders m_{lower} and m_{upper} can be varied in the developed implementations. If m_i is found, then s will be set to 0 and the algorithms finish their execution.

Otherwise, we choose $m = m_{upper}$ and

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

where $f_s = 1$ or 0.5, respectively, if Algorithm 1 or Algorithm 2 is used. Ad-

Algorithm 4: Given a matrix $A \in \mathbb{C}^{n\times n}$, a minimum order $m_{lower} \in \mathbb{M}$ and a maximum order $m_{upper} \in \mathbb{M}$, this algorithm provides an order $m \in \mathbb{M}$, $m_{lower} \leq m \leq m_{upper}$, a scaling factor s and the necessary powers of A.

```
1  $A_1 = A$; $i = lower$; $f = 0$
2  for $j = 2$ to $\lceil\sqrt{m_i}\rceil$ do
3      $A_j = A_{j-1}A$
4  end
5  while $f = 0$ and $i \leq upper$ do
6      $v = \sqrt{m_i}$
7      $j = \lceil v \rceil$
8      if $j > v$ then $A_j = A_{j-1}A$
9      Compute $a_i \approx \|A^{m_i}\|$ from $A^j$ /* based on Algorithm 1 from [24] */
10     $\alpha_i = \sqrt[m_i]{a_i}$
11     if $\alpha_i < \Theta_i$ then $f = 1$
12     else $i = i + 1$
13 end
14 if $f = 1$ then $s = 0$
15 else
16     $i = upper$
17     $s = \lceil \max(0, f_s \log_2(\alpha_i/\Theta_i)) \rceil$
18     if $|p_{m_i}| a_i r^{(1-s)m_i} < u$ then /* $r = 2$ (Alg. 1) or $r = 4$ (Alg. 2) */
19         $s = s - 1$
20         if $|p_{m_i}| a_i r^{(1-s)m_i} < u$ then $s = s - 1$
21     end
22 end
23 $m = m_i$
```

ditionally, at lines 17-25, Algorithm 3 tests whether it is possible to decrease the above values s and $m = m_{upper}$, so that inequality

$$\tilde{h}_m(2^{-s}\alpha_m) \leqslant \tilde{h}_m(\Theta_m) \leqslant u \quad \text{or} \quad \tilde{h}_m(4^{-s}\alpha_m) \leqslant \tilde{h}_m(\Theta_m) \leqslant u,$$

is fulfilled for Algorithm 1 or Algorithm 2, respectively. In such cases, these new values are to be considered.

On the other hand, let $p_{m_{upper}}$ be the coefficient of the term of order m_{upper} of the Bernoulli polynomial. Then, at lines 18-21, Algorithm 4 tests whether it is possible to decrease the value of s to satisfy that

$$|p_{m_{upper}}| ||A^{m_{upper}}|| 2^{(1-s)m_{upper}} < u \quad \text{or} \quad |p_{m_{upper}}| ||A^{m_{upper}}|| 4^{(1-s)m_{upper}} < u,$$

depending on the use of Algorithm 1 or Algorithm 2.

Algorithm 5: Given a matrix $A \in \mathbb{C}^{n\times n}$, a minimum order $m_{lower} \in \mathbb{M}$, a maximum order $m_{upper} \in \mathbb{M}$ and a small value *tol*, this algorithm provides an order $m \in \mathbb{M}$, $m_{lower} \leq m \leq m_{upper}$, the scaling factor s and the necessary powers of A.

```
i = lower; f = 0
Compute $\alpha_0 \approx \|A^{m_i}\|^{1/m_i}$ from $A$  /* based on Algorithm 1 from [24] */
while $f = 0$ and $i < upper$ do
    $i = i + 1$
    Compute $\alpha \approx \|A^{m_i}\|^{1/m_i}$ from $A$  /* based on Algorithm 1 from [24] */
    if $|\alpha - \alpha_0| > \alpha \cdot tol$ then  $\alpha_0 = \alpha$
    else  $f = 1$
end
$i = lower$; $f = 0$
while $f = 0$ and $i \leq upper$ do
    if $\alpha < \Theta_i$ then  $f = 1$
    else  $i = i + 1$
end
if $f = 1$ then  $s = 0$
else
    $i = upper$
    $s = \lceil \max(0, f_s \log_2(\alpha/\Theta_i)) \rceil$ /* $f_s = 1$ (Alg. 1) or $f_s = 0.5$ (Alg. 2) */
    $j = i$
    while $f = 0$ and $j > lower$ do
        $j = j - 1$
        $s_1 = \lceil \max(0, f_s \log_2(\alpha/\Theta_j)) \rceil$
        if $s \geq s_1$ then
            $s = s_1$
            $i = j$
        else  $f = 1$
    end
end
$m = m_i$
$A_1 = A$
for $j = 2$ to $\lceil \sqrt{m_i} \rceil$ do
    $A_j = A_{j-1}A$
end
```

Unlike Algorithms 3 and 4, Algorithm 5 does not compute initially the powers of matrix A, so the estimation (15) is obtained only from A. This

Table 1: Bernoulli polynomial coefficients for $m = 25$.

Coefficients	Expression (6)	Expression (9)
p_0	1.000000000000000e+00	9.999999999999776e-01
p_1	4.268425513808927e-17	2.200931543663476e-25
p_2	-5.000000000000000e-01	-4.999999999998889e-01
p_3	-7.103351130950067e-18	9.114115486610701e-26
p_4	4.166666666666666e-02	4.166666666657532e-02
p_5	3.340635907377075e-19	1.642205248036368e-25
p_6	-1.388888888888889e-03	-1.388888888858840e-03
p_7	1.188305192257936e-20	2.406735243784263e-26
p_8	2.480158730158730e-05	2.480158729629132e-05
p_9	-1.104189679966496e-20	-2.741989705063794e-28
p_{10}	-2.755731922398609e-07	-2.755731916590913e-07
p_{11}	4.004071095795792e-21	-2.506940767309528e-29
p_{12}	2.087675698787421e-09	2.087675655363431e-09
p_{13}	-1.013606842281333e-21	-1.827655933239498e-31
p_{14}	-1.147074559786225e-11	-1.147074324303990e-11
p_{15}	1.905862492984388e-22	-3.648057207260014e-33
p_{16}	4.779477334567797e-14	4.779467650734273e-14
p_{17}	-2.768223171137328e-23	2.331854748958815e-35
p_{18}	-1.561920725009726e-16	-1.561889490721346e-16
p_{19}	3.205121320979757e-24	0
p_{20}	4.110320556779777e-19	4.109509217914593e-19
p_{21}	-3.051721100969466e-25	0
p_{22}	-8.897045307814457e-22	-8.879692513470797e-22
p_{23}	2.530017510290091e-26	0
p_{24}	1.613667223591245e-24	1.582268801402494e-24
p_{25}	-2.511873775100074e-27	0

algorithm computes α_i such that

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

where *tol* is a small prefixed value (lines 1-8).

Then, Algorithm 5 tries to find out the smallest value m_i, $m_{lower} \leq m_i \leq m_{upper}$, that satisfies $\alpha_i \leq \Theta_i$. If so, similarly to Algorithm 4, Algorithm 5 will finish and s will be set to 0. Otherwise, we choose m_{upper} and

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

Next, lines 18-26 of Algorithm 5 make sure whether it is possible to cut down the above values of s and m such that the inequality

$$\tilde{h}_m(2^{-s}\alpha_m) \leqslant \tilde{h}_m(\Theta_m) \leqslant u \quad \text{or} \quad \tilde{h}_m(4^{-s}\alpha_m) \leqslant \tilde{h}_m(\Theta_m) \leqslant u,$$

is fulfilled, for Algorithm 1 or Algorithm 2, respectively. If this condition is satisfied, these new values will be used instead.

4 Numerical Experiments

Although theoretically, and according to formulations (9) and (10), all coefficients $p_i^{(m)}$ occupying odd positions in the Bernoulli polynomial $P_m(A) = p_m^{(m)}A^m + p_{m-1}^{(m)}A^{m-1} + \cdots + p_1^{(m)}A + p_0^{(m)}I$ should be equal to 0, in practice, it might not happen with all of them. As an example, Table 1 shows the coefficients of $P_m(A)$ for formulae (6) and (9) when $m = 25$. For the sake of brevity, p_i will be used instead of $p_i^{(25)}, 0 \leq i \leq 25$. 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. This raises the following dilemma: turn these values into zero and take into account only the even terms or keep these values as they are and consider them all.

Therefore, having in mind expression (6), the two different mentioned above alternatives that derived from expression (9) and the three distinct algorithms described in Section 3 to compute the polynomial degree m and the scaling parameter s, a total of nine different approximations are at our disposal to compute the matrix cosine function. To test and compare the numerical performance of all these different approaches, the following algorithms have been implemented in MATLAB:

- `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 (9). 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 parameters $m \in \{30, 36\}$ and s.

- `cosmber_3_3`, `cosmber_3_4`, and `cosmber_3_5`: developments derived from formula (9) where odd position coefficients have been neglected. In this way, only the even terms will be employed, as in the case of Taylor series [13]. 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 values.

- `cosm`: implementation based on the Padé rational approximation for the matrix cosine function [17].

The next test battery, composed of four types of different and representative matrices, has been used to compare the above mentioned algorithms. MATLAB 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:

a) Diagonalizable real matrices: These matrices are obtained as the result of $A = V \cdot D \cdot V^{-1}$, where D is a diagonal matrix (with real and complex eigenvalues) and matrix V is an orthogonal matrix being $V = H/\sqrt{n}$, where H is a Hadamard matrix and n its number of rows or columns. As 1-norm, we have that $2.18 \leq \|A\|_1 \leq 132.62$. The matrix cosine function was calculated *"exactly"* as $\cos(A) = V \cdot \cos(D) \cdot V^T$ thanks to the `vpa` function.

b) Non-diagonalizable complex matrices: These matrices are computed as $A = V \cdot J \cdot V^{-1}$, where J is a Jordan matrix with complex eigenvalues whose modules are less than 10 and the algebraic multiplicity is randomly generated between 1 and 5. V is an orthogonal random matrix with elements in the interval $[-0.5, 0.5]$. As 1-norm, we have obtained that $91.3 \leq \|A\|_1 \leq 92.6$. The *"exact"* matrix cosine function was computed as $\cos(A) = V \cdot \cos(J) \cdot V^{-1}$, by means of `vpa` function.

c) Matrices from the Matrix Computation Toolbox (MCT) [25] and from the **Eigtool MATLAB Package (EMP)** [26]: These matrices 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 have used 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 MCT and the EMP are initially composed of fifty-two and twenty matrices, respectively, thirteen

Table 2: Codes to be compared for each experiment.

	Evaluated codes
Test 1:	`cosmber_1_3`, `cosmber_1_4`, `cosmber_1_5`
Test 2:	`cosmber_2_3`, `cosmber_2_4`, `cosmber_2_5`
Test 3:	`cosmber_3_3`, `cosmber_3_4`, `cosmber_3_5`

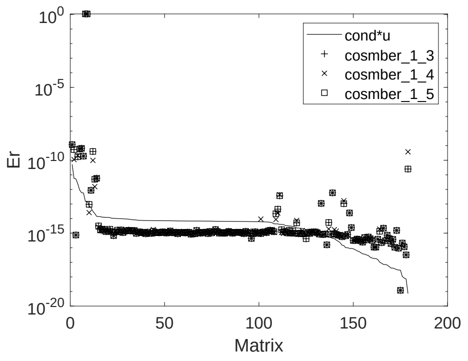
matrices of these two packages were discarded for different reasons. In particular, matrices 5, 15, 16, 17, 21, 42, 43, 44 and 49 belonging to the MCT and matrix 3 from the EMP were not used since the exact cosine solution could not be computed. Matrix 2 of MCT and matrices 4 and 10 of EMP were not considered due to the excessively high relative error provided by all the codes.

Our first analysis is composed by the three tests described in Table 2. These experiments were independently performed to find out the most appropriate combination of the distinct variations of each theoretical formulation with Algorithms 3, 4 and 5, in charge of computing parameters m and s. For each test, the normwise relative errors, the performance profiles, and the number of matrix products required are provided. All these executions were carried out with 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.

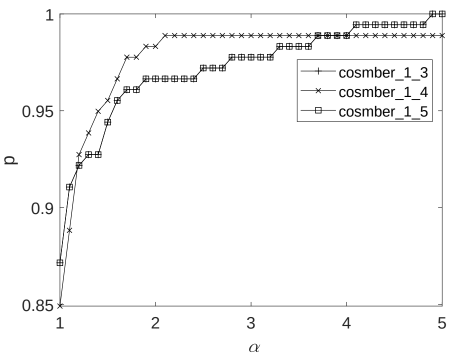
The *normwise relative error* is computed as

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

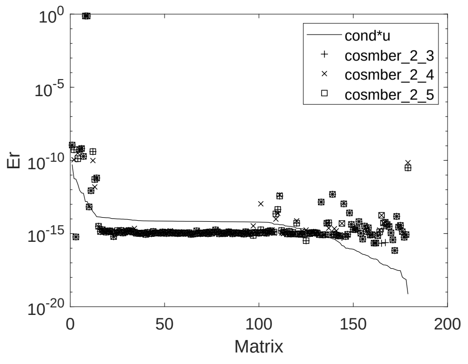
where $\cos(A)$ is the exact solution and $\tilde{cos}(A)$ is the computed approximation to $\cos(A)$. These errors are depicted in Figures 1(a), 1(c), and 1(e), which show the numerical stability of the different compared methods. The solid line represents function $k_{\cos}u$, where $k_{\cos}$ (or *cond*) is the condition number of the matrix cosine function [7, Chapter 3] and u is the unit roundoff in the IEEE double precision floating-point arithmetic ($u = 2^{-53}$). We use the distance from the normwise relative error to the solid line, which illustrates the theoretical expected normwise relative error for each matrix, as a measure of accuracy. For most of the matrices, all the codes exhibit a very good numerical stability according to the small distance that can be appreciated. Furthermore, those methods with a relative error below the $cond * u$ line provide even better accuracy.



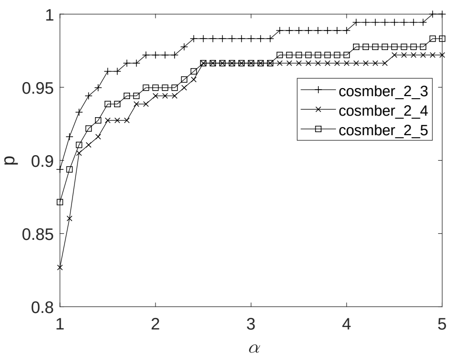
(a) Test 1: Normwise relative errors.



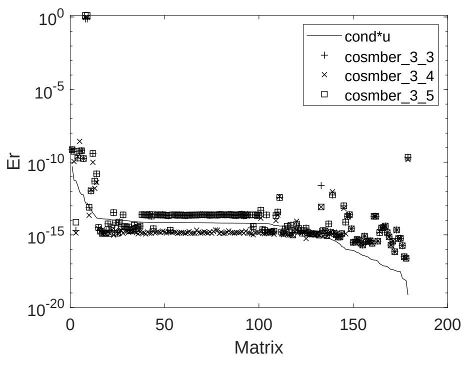
(b) Test 1: Performance profile.



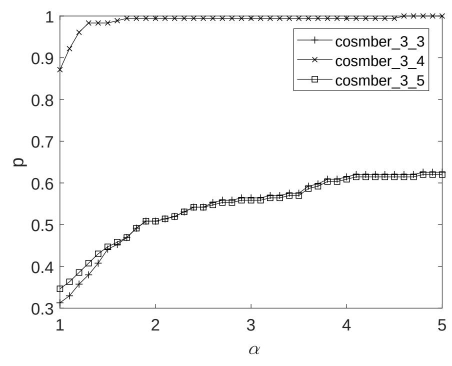
(c) Test 2: Normwise relative errors.



(d) Test 2: Performance profile.



(e) Test 3: Normwise relative errors.



(f) Test 3: Performance profile.

Figure 1: Experimental results for the Test 1, 2 and 3.

Nonetheless, for a few matrices, we can appreciate a remarkable distance between the computed normwise relative error and the expected one. In the calculation of all these matrices it happens that the scaling parameter s reaches high values. It should be clarified that such values lead to a substantial increment in the number of arithmetic operations to be performed in the recovery phase, with the consequent negative effect on results caused by rounding errors.

The *performance profile* is presented in Figures 1(b), 1(d) and 1(f). For a given α value (represented on the x-axis and varying from 1 to 5 in steps equal to 0.1) the p coordinate on the y-axis means the probability that the considered code has a relative error lower than or equal to α-times the smallest relative error over all of them for the given test. This way of measuring the accuracy of an algorithm is quite accepted as it has been defined in [7, p. 252].

For Test 1, the performance profile (Figure 1(b)) shows that `cosmber_1_3` and `cosmber_1_5` code accuracy is identical, while `cosmber_1_4` achieves the highest values for a large portion of the graph and, as a consequence, it outputs the most accurate results in the calculation of the cosine for a large number of matrices. In Test 2 (Figure 1(d)), `cosmber_2_3` always gives rise to the best results, whilst `cosmber_2_4` and `cosmber_2_5` present lower values, though close to it, and quite similar between them. Finally, in the case of Test 3 (Figure 1(f)), the matrix cosine function computed by `cosmber_3_4` approaches the result with a much better numerical precision than that worked out by `cosmber_3_3` and `cosmber_3_5`, the least reliable codes whose results are clearly disappointing. To understand the reasons behind these figures, we need to proceed in our analysis.

Minimum, maximum and average polynomial degree (m) and scaling parameter (s) required by the different codes that compose Tests 1, 2, and 3 are grouped in Table 3. As defined previously, the polynomial order to be provided by Algorithms 3, 4 and 5 varies between 30 and 36, for `cosmber_1_x` and `cosmber_2_x`, and ranges from 16 to 20 for `cosmber_3_x`. It is not difficult to observe a relationship between the parameter m calculated and the results depicted in Figure 1(f). Whereas implementations `cosmber_1_x`, `cosmber_2_x` and `cosmber_3_4` employ an average value of m practically identical to their maximum allowed, this average value is equal or very close to the minimum amount in the case of `cosmber_3_3` and `cosmber_3_5`, hence their poor numerical performance.

Table 4 collects the computational costs of each algorithm in each test cast in terms of number of matrix products (`P`), since the cost of the rest of the op-

Table 3: Minimum, maximum and average computed parameters m and s for Tests 1, 2 and 3.

	m			s		
	Min.	Max.	Average	Min.	Max.	Average
`cosmber_1_3`	30	36	35.13	0	17	2.52
`cosmber_1_4`	30	36	34.99	0	17	2.26
`cosmber_1_5`	30	36	34.99	0	17	2.52
`cosmber_2_3`	30	36	35.13	0	17	2.52
`cosmber_2_4`	30	36	34.99	0	17	2.26
`cosmber_2_5`	30	36	34.99	0	17	2.52
`cosmber_3_3`	16	16	16.00	0	17	2.56
`cosmber_3_4`	16	20	19.26	0	17	1.80
`cosmber_3_5`	16	20	16.22	0	17	2.49

Table 4: Matrix products (P) for implemented codes.

`P(cosmber_1_3)`	`P(cosmber_1_4)`	`P(cosmber_1_5)`
2215	2165	2211
`P(cosmber_2_3)`	`P(cosmber_2_4)`	`P(cosmber_2_5)`
2215	2165	2211
`P(cosmber_3_3)`	`P(cosmber_3_4)`	`P(cosmber_3_5)`
1678	1542	1529

erations is negligible for big enough matrices. The lowest number of products corresponds to implementations obtained from formula (9), after removing the terms from odd positions (`cosmber_3_x`). More in detail, `cosmber_3_5` achieves the lowest computational cost. Both sets of codes, those based on formula (6) (`cosmber_1_x`) and those based on (9) when considering even and odd terms (`cosmber_2_x`), require an identical number of matrix multiplications when the algorithm used to compute m and s (Algorithm 3, 4, or 5) is the same.

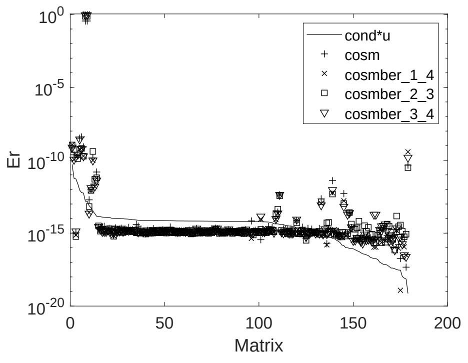
According to the previous analysis, we conclude that the best codes for each one of the three tests are `cosmber_1_4`, `cosmber_2_3` and `cosmber_3_4`, respectively. The choice of `cosmber_2_3` and `cosmber_3_4` for Test 2 and 3 is natural at the light of performance profile figures. More complicated is the selection of `cosmber_1_4` in Test 1, since its numerical performance is very similar to that of the other codes. However, `cosmber_1_4` keeps a very regular behaviour in the most part of the performance profile graph and the number of matrix products required is smaller than that involved when executing `cosmber_1_3` or `cosmber_1_5`. Moreover, `cosmber_1_4` exhibits the smallest average normwise relative error, calculated from all the matrices, among the three codes compared in Test 1. Notwithstanding, the improvement of `cosmber_1_3` and `cosmber_1_5` versus `cosmber_1_4`, when α takes values greater than or equal to 4, can be justified having in mind that the former codes have a slightly lower standard deviation than `cosmber_1_4` in the normwise relative error.

Once performed the above selection, we conducted another experiment, named Test 4, which compares the three chosen codes with `cosm`, i.e. the implementation that computes the matrix cosine function by means of the Padé rational approximation. For this test, Figure 2 depicts the normwise relative errors (a), the performance profiles (b), the ratio of the relative errors (c), the order of the approximation polynomials employed (d) and the ratio of the matrix products (e) among the different implementations now compared.

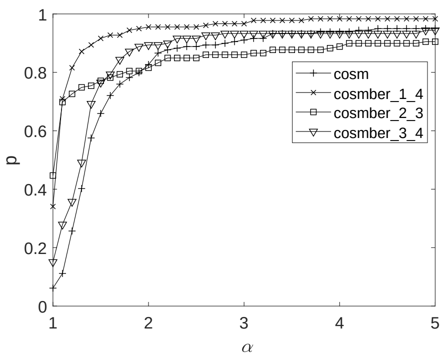
Figure 2(a) shows little differences in the relative errors incurred by the different methods, where the vast majority takes values close to 1E-15 and, in almost all cases, varying from 1E-9 to 1E-17.

Regarding the performance profile, as plotted in Figure 2(b), `cosmber_1_4` always offers the best and highest values, which indicates that its results are the most precise. As it can be seen in the initial part of the graph, `cosmber_2_3` presents very accurate values in the calculation of the cosine function of a matrix, but it also provides more inaccurate results for a large amount of cases than the other codes, giving rise to the lowest probability in a large portion of the picture. The accuracy of the results supplied by `cosmber_3_4` and `cosm` is quite similar, although their computations are not usually the most accurate compared to those offered by the other methods.

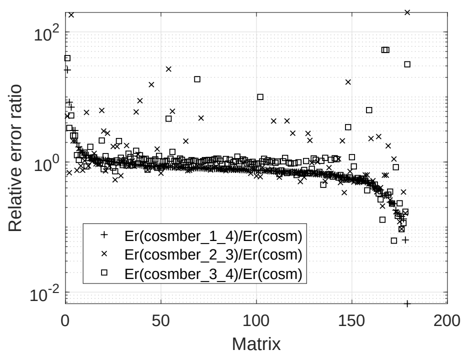
In Figure 2(c), the ratios of normwise relative errors have been presented in decreasing order with respect to Er(`cosmber_1_4`)/Er(`cosm`). As this figure draws, the ratio of relative errors related to `cosmber_1_4` is less than 1 for the vast majority of matrices. Clearly, `cosmber_2_3` displays the largest amount of values furthest from and above the unit, although in most cases the results differ only slightly from those provided by `cosmber_1_4` and are below 1. In terms of `cosmber_3_4`, its ratio of relative errors is usually higher than that of `cosmber_1_4`, with values close to unity. These data can be verified with that exposed in Table 5, where we show the percentage of cases in which the relative error of `cosm` is lower, greater or equal than that of `cosmber_1_4`,



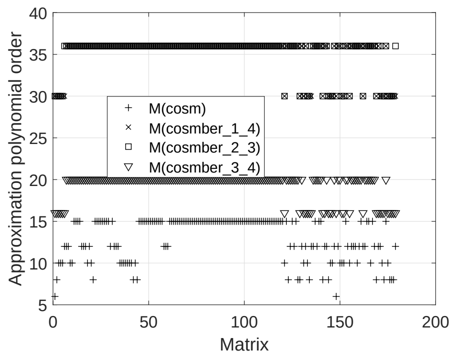
(a) Normwise relative errors.



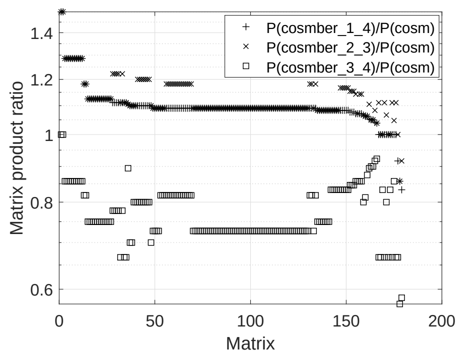
(b) Performance profile.



(c) Ratio of relative errors.



(d) Polynomial orders.



(e) Ratio of matrix products.

Figure 2: Experimental results for Test 4.

Table 5: Relative error comparison among `cosm` versus `cosmber_1_4`, `cosmber_2_3` and `cosmber_3_4` for Test 4.

Er(`cosm`)<Er(`cosmber_1_4`)	13.97%
Er(`cosm`)>Er(`cosmber_1_4`)	86.03%
Er(`cosm`)=Er(`cosmber_1_4`)	0.00%
Er(`cosm`)<Er(`cosmber_2_3`)	24.02%
Er(`cosm`)>Er(`cosmber_2_3`)	75.98%
Er(`cosm`)=Er(`cosmber_2_3`)	0.00%
Er(`cosm`)<Er(`cosmber_3_4`)	53.07%
Er(`cosm`)>Er(`cosmber_3_4`)	46.93%
Er(`cosm`)=Er(`cosmber_3_4`)	0.00%

`cosmber_2_3`, and `cosmber_3_4`, respectively.

As it can be appreciated, `cosmber_1_4` and `cosmber_2_3` both compute the cosine function of most of the matrices included in the testbed in a more accurate way than `cosm`. More in detail, the percentage of matrices in which `cosmber_1_4` and `cosmber_2_3` outperform `cosm` is 86.03% and 75.98%, respectively. As expected, `cosmber_3_4` and `cosm` both offer an analogous behaviour, since `cosmber_3_4` only outputs a better result than `cosm` in 46.93% of cases.

Table 6 compares the minimum, maximum and average order (m) of the involved approximation polynomials and the scaling parameter (s) for `cosm` and the three selected codes. Whereas the order of polynomials used by `cosmber_1_4` and `cosmber_2_3` presents an average around 35, this value is very close to 20 for `cosmber_3_4` or 13 for `cosm`. More specifically, Figure 2(d) depicts the order of the polynomial used in the calculation of each matrix by each code. In the case of the scaling parameter, it reaches a mean value equal to 2.26 for `cosmber_1_4`, 2.52 for `cosmber_2_3`, 1.8 for `cosmber_3_4`, and 1.91 for `cosm`.

Finally, with regard to the computational cost, Table 7 includes the number of matrix products performed. The largest amount of products (2215) was needed by `cosmber_2_3`, followed by `cosmber_1_4` (2165) and `cosm` (1987). Algorithm `cosmber_3_4` gives rise to the lowest computational cost, by requiring the least number of matrix operations (1542). These results are graphically displayed in Figure 2(e), that exposes the ratio of matrix products of the three selected Bernoulli-based methods with respect to `cosm`.

Table 6: Mininum, maximum and average computed parameters m and s for Test 4.

	m			s		
	Min.	Max.	Average	Min.	Max.	Average
`cosm`	6	15	12.93	0	16	1.91
`cosmber_1_4`	30	36	34.99	0	17	2.26
`cosmber_2_3`	30	36	35.13	0	17	2.52
`cosmber_3_4`	16	20	19.26	0	17	1.80

Table 7: Matrix products (P) for Test 4.

`P(cosm)`	`P(cosmber_1_4)`	`P(cosmber_2_3)`	`P(cosmber_3_4)`
1987	2165	2215	1542

This ratio is greater than or equal to 1 in the calculation of almost all matrices for `cosmber_1_4` and `cosmber_2_3`, but less than or equal to 1 for all of them in the case of `cosmber_3_4`.

Tackling large-scale problems is quite common in this context, where, e.g. the number of particles modeled by the physical system under study can be very large. Thus, to conclude our experimental analysis, we show the performance, cast in terms of execution time, stressing `cosmber_1_3`, `cosmber_1_4` and `cosmber_1_5` codes to deal with large problem dimensions ranging from $n = 1000$ to 10000 (Fig. 3).

The nature of the computational core of this type of algorithms, i.e. matrix multiplication, let us to exploit the very highly efficient implementation of this kernel in different computing environments. On the one hand, MathWorks® MATLAB uses the Intel® MKL library and its *threaded* version of this operation when executing on a CPU environment. The user is oblivious to this fact, simply notices that the algorithm runs fast. On the other hand, NVIDIA provides a very optimized implementation of the matrix product for its GPUs, included in the cuBLAS library [27], a development of BLAS (Basic Linear Algebra Subprograms) from CUDA [28].

Our aim is to provide the user with the same capability as in the multicore environment, i.e. in a transparent way, when a GPU is available. Our algorithms, they all implemented in MATLAB, also can make use of the GPU through a MEX file [29] that uploads matrix multiplications to the

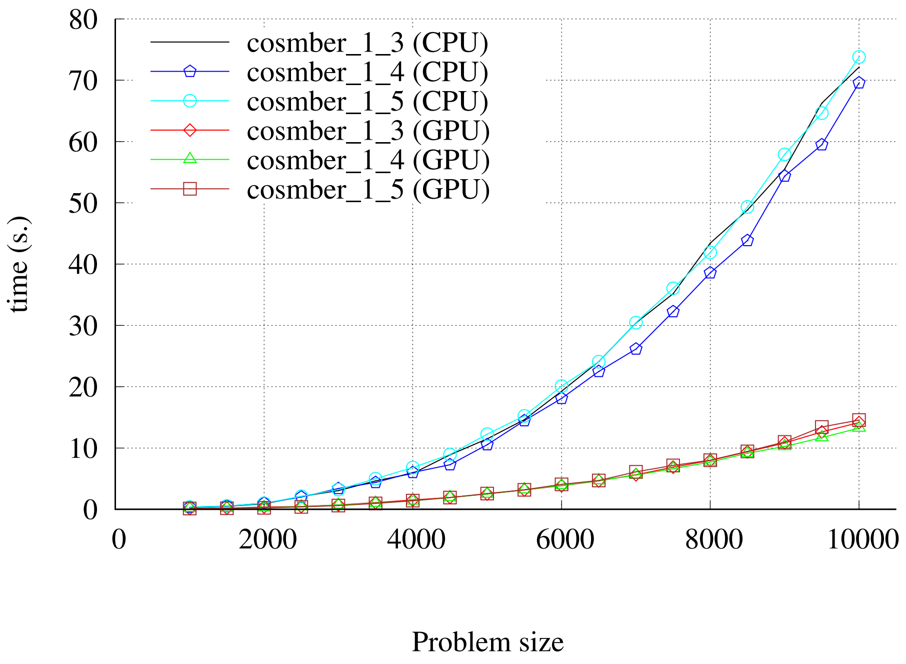


Figure 3: CPU and GPU execution time.

GPU by performing the appropriate calls to the cuBLAS library. To analyse the computational performance of our codes, using either the CPU or the GPU system, we have carried out experiments on a computer equipped with two processors Intel Xeon CPU E5-2698 at 2.20 GHz featuring 20 cores each resulting in 40 cores to operate on matrix multiplications. The computer also features an NVIDIA Tesla P100 SMX2 (Pascal architecture) with 16 GB of memory attached to the PCI of this computer. A GPU of this type features 56 multiprocessors with 64 CUDA cores each, resulting in a total of 3584 CUDA cores.

Apart from the fact that the GPU version performs far more better than its CPU counterpart, we can itemize two other important ideas. Firstly, `cosmber_1_4` is the routine that performs the fastest when the problem size increases and, secondly, this behaviour is the same on the GPU though, due to the reduced execution time, the difference among the three codes is smaller and difficult to appreciate graphically.

5 Conclusions

In this paper, a new efficient method based on Bernoulli matrix polynomials to compute the matrix cosine function has been presented. For that, two series expansions of the matrix cosine, in terms of the Bernoulli matrix polynomials, are described, one of them resulted into two different interpretations. In addition, three different algorithms to compute the polynomial order and the scaling parameter are exposed. With all of this, nine different approaches have been built and implemented in MATLAB.

An experimental analysis, comparing all our Bernoulli-like versions among them and with respect to `cosm`, a code in charge of computing the matrix cosine function by means of Padé rational approximation, has been performed. The different tests carried out have demonstrated that one of these implementations, i.e. `cosmber_1_4`, outputs the best results, with a relative error involved in the computation, with regard to the exact solution, which improved to `cosm` in a 86.03% of cases, although requiring a computational cost slightly higher.

Moreover, a CUDA version of all the algorithms, that allows executing them on an NVIDIA GPU, has been implemented. This version exploits one of the highlights of this approach, being based on matrix multiplications, a very optimized operation in high performance computing environments. Nonetheless, the results in a GPU platform does nothing else than to contribute to the conclusion of the former analysis carried out on the CPU core.

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, A.E. Posso, and M.C. 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, Jesús Peinado, Javier Ibáñez, Jorge Sastre, and Emilio Defez. Computing matrix trigonometric functions with GPUs through Matlab. *The Journal of Supercomputing*, 75(3):1227–1240, 2019.

[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] Mehdi Dehghan and Masoud Hajarian. Determination of a matrix function using the divided difference method of Newton and the interpolation technique of Hermite. *Journal of Computational and Applied Mathematics*, 231(1):67–81, 2009.

[11] Saeed Kazem and Mehdi Dehghan. Semi-analytical solution for time-fractional diffusion equation based on finite difference method of lines (MOL). *Engineering with Computers*, 35(1):229–241, 2019.

[12] Saeed Kazem and Mehdi Dehghan. Application of finite difference method of lines on the heat equation. *Numerical Methods for Partial Differential Equations*, 34(2):626–660, 2018.

[13] 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.

[14] 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.

[15] 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.

[16] 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.

[17] 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.

[18] Omran Kouba. Lecture Notes, Bernoulli polynomials and applications. *arXiv:1309.7560*, 2013.

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

[20] 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. Rota (Cádiz), Spain.

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

[22] J. Sastre. Efficient evaluation of matrix polynomials. *Linear Algebra and its Applications*, 539:229–250, 2018.

[23] J. Sastre, J. Ibáñez, P. Ruiz, and E. Defez. Accurate and efficient matrix exponential computation. *International Journal of Computer Mathematics*, 91(1):97–112, 2013.

[24] Nicholas J. Higham. Fortran codes for estimating the one-norm of a real or complex matrix, with applications to condition estimation. *ACM Transactions on Mathematical Software*, 14(4):381–396, December 1988.

[25] Nicholas J. Higham. The matrix computation toolbox. *URL: http://www.ma.man.ac.uk/ higham/mctoolbox*, 2002.

[26] Thomas G. Wright. Eigtool, version 2.1. *URL: http://www.comlab.ox.ac.uk/pseudospectra/eigtool*, 2009.

[27] NVIDIA. *cuBLAS*, 2020.

[28] NVIDIA. *CUDA Toolkit Documentation v11.0.3*, 2020.

[29] NVIDIA. Accelerating MATLAB with CUDA Using MEX Files. *White Paper*, 2007.