On Bernoulli matrix polynomials and matrix exponential approximation

Article *in* Journal of Computational and Applied Mathematics · September 2020

DOI: 10.1016/j.cam.2020.113207

CITATION
1

READS
25

5 authors, including:

Emilio Defez
Universitat Politècnica de València
84 PUBLICATIONS **899** CITATIONS
SEE PROFILE

Javier Ibáñez
Universitat Politècnica de València
6 PUBLICATIONS **11** CITATIONS
SEE PROFILE

Pedro Alonso
Universitat Politècnica de València
114 PUBLICATIONS **523** CITATIONS
SEE PROFILE

José M. Alonso
Universitat Politècnica de València
77 PUBLICATIONS **579** CITATIONS
SEE PROFILE

On Bernoulli matrix polynomials and matrix exponential approximation

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

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

[a] *Instituto de Matemática Multidisciplinar*
[b] *Instituto de Instrumentación para Imagen Molecular*
[c] *Department of Information Systems and Computation*

Abstract

We present in this paper a new method based on Bernoulli matrix polynomials to approximate the exponential of a matrix. The developed method has given rise to two new algorithms whose efficiency and precision are compared to the most efficient implementations that currently exist. For that, a state-of-the-art test matrix battery, that allows deeply exploring the highlights and downsides of each method, has been used. Since the new algorithms proposed here do make an intensive use of matrix products, we also provide a GPUs-based implementation that allows to achieve a high performance thanks to the optimal implementation of matrix multiplication available on these devices.

Keywords:
Bernoulli matrix approximation, Matrix exponential function, GPU computing

1. Introduction

The computation of matrix functions has received attention in the last years because of its several applications in different areas of science and technology. Of all the matrix functions, the exponential matrix e^A, $A \in \mathbb{C}^{r\times r}$, stands out, due both to its applications in the resolution of systems of differential equations, or in the graph theory, and to the difficulties involved in its computation, see [1–5].

Among the proposed methods for the approximate computation of the exponential matrix, two fundamental ones stand out: those based on rational Padé approximations [6–10], and those based on polynomial approximations, which are either Taylor series developments [11–13] or serial developments of Hermite matrix polynomials [14, 15]. In general, polynomial approximations showed to

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

be more efficient than the Padé algorithm in tests because they are more accurate, despite a slightly higher cost in some cases. All these methods use the basic property of *Scaling-Squaring*, based on the relationship

$$e^A = \left(e^{A/2^s}\right)^{2^s}.$$

Thus, if $P_m(A)$ is a matrix polynomial approximation of e^A, then given a matrix A and a scaling factor s, $P_m(A/2^s)$ is an approximation to $e^{A/2^s}$ and

$$e^A \approx \left(P_m(A/2^s)\right)^{2^s}. \tag{1}$$

Bernoulli polynomials and Bernoulli numbers have been extensively used in several areas of mathematics, as number theory, and they appear in many mathematical formulas, such as the residual term of the Euler-Maclaurian quadrature rule [16, p. 63], the Taylor series expansion of the trigonometric functions $\tan(x), \csc(x)$ and $\cot(x)$ [16, p. 116-117] and the Taylor series expansion of the hyperbolic function $\tanh(x)$, [16, p. 125]. They are also employed in the well known exact expression of the even values of the Riemann zeta function:

$$\xi(2k) = \sum_{n\geq 1} \frac{1}{n^{2k}} = \frac{(-1)^{k-1}(2\pi)^{2k}\mathcal{B}_{2k}}{2(2k)!}, k \geq 1.$$

Moreover, they are even used for solving initial value problems [17], boundary value problems [18, 19], high-order linear and nonlinear Fredholm and Volterra integro-differential equations [20, 21], complex differential equations [22] and partial differential equations [23–25]. An excellent survey about Bernoulli polynomials and its applications can be found in [26]. The development of series functions of Bernoulli polynomials has been studied in [27, 28].

In this paper, we present a new series development of the exponential matrix in terms of *Bernoulli matrix polynomials* which demonstrates that the polynomial approximations of the exponential matrix are more accurate and less computationally expensive in most cases than those based on Padé approximants. We also verify that this new method based on Bernoulli matrix polynomials is a competitive method for the approximation of the exponential matrix, with a similar computational cost than the Taylor one but, generally, more accurate.

The organization of the paper is as follows. Section 2 is devoted to Bernoulli polynomials. We show how to obtain a series of a matrix exponential in terms of Bernoulli matrix polynomials and how to approach that exponential of a matrix. The following section describes the algorithms proposed. Tests and comparisons are presented in Section 4. We close the document with some conclusion remarks.

Notation

Throughout this paper, we denote by $\mathbb{C}^{r\times r}$ the set of all the complex square matrices of size r and by I the identity matrix. A polynomial of degree m means an expression of the form $P_m(t) = a_m t^m + a_{m-1}t^{m-1} + \cdots + a_1 t + a_0$,

where t is a real variable and a_j, for $0 \leq j \leq m$, are complex numbers. In this way, we can define the matrix polynomial $P_m(B)$ for $B \in \mathbb{C}^{r\times r}$ as $P_m(B) = a_m B^m + a_{m-1}B^{m-1} + \cdots + a_1 B + a_0 I$. Throughout this paper, we denote I_n (or I) and $0_{n\times n}$ the matrix identity and the null matrix of order n, respectively. With $\lceil x \rceil$, we denote the result of rounding x to the nearest integer greater than or equal to x and $\lfloor x \rfloor$ denotes the result of rounding x to the nearest integer less than or equal to x. As usual, the matrix norm $||\cdot||$ denotes any subordinate matrix norm; in particular $||\cdot||_1$ is the usual $1-$norm. Finally, if $\mathcal{A}(k,m)$ are matrices in $\mathbb{C}^{n\times n}$ for $m \geq 0, k \geq 0$, from [29] it follows that

$$\sum_{m\geq 0}\sum_{k\geq 0}\mathcal{A}(k,m) = \sum_{m\geq 0}\sum_{k=0}^{m}\mathcal{A}(k,m-k). \tag{2}$$

2. On Bernoulli matrix polynomials

The Bernoulli polynomials $B_n(x)$ are defined in [16, p.588] as the coefficients of the generating function

$$g(x,t) = \frac{te^{tx}}{e^t - 1} = \sum_{n\geq 0}\frac{B_n(x)}{n!}t^n \ , \ |t| < 2\pi, \tag{3}$$

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

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

where the Bernoulli numbers are defined by $\mathcal{B}_n = B_n(0)$. Therefore, it follows that the Bernoulli numbers satisfy

$$\mathcal{B}_0 = 1, \ \mathcal{B}_k = -\sum_{i=0}^{k-1}\binom{k}{i}\frac{\mathcal{B}_i}{k+1-i}, k \geq 1. \tag{5}$$

Note that $\mathcal{B}_3 = \mathcal{B}_5 = \cdots = \mathcal{B}_{2k+1} = 0$, for $k \geq 1$. For a matrix $A \in \mathbb{C}^{r\times r}$ we define the $m-th$ Bernoulli matrix polynomial by the expression

$$B_m(A) = \sum_{k=0}^{m}\binom{m}{k}\mathcal{B}_k A^{m-k}. \tag{6}$$

Thus, we can now calculate the exact value of $e^{At}\left(\frac{t}{e^t-1}\right)$, where $A \in \mathbb{C}^{r\times r}$. By using (3) and (6) one gets

$$e^{At}\left(\frac{t}{e^t-1}\right) = \left(\sum_{n\geq 0}\frac{A^n}{n!}t^n\right)\left(\sum_{k\geq 0}\frac{\mathcal{B}_k}{k!}t^k\right) = \sum_{n\geq 0}\sum_{k\geq 0}\frac{A^n\mathcal{B}_k}{n!k!}t^n t^k.$$

Taking into account that $\mathcal{A}(k,n) = \dfrac{A^n \mathcal{B}_k t^n t^k}{n!k!}$ from (2), we have

$$e^{At}\left(\frac{t}{e^t-1}\right) = \sum_{n\geq 0}\sum_{k=0}^{n} \frac{\mathcal{B}_k}{k!} t^k \frac{A^{n-k}}{(n-k)!} t^{n-k}$$

$$= \sum_{n\geq 0}\left(\sum_{k=0}^{n} \binom{n}{k} \mathcal{B}_k A^{n-k}\right)\frac{t^n}{n!} = \sum_{n\geq 0}\frac{B_n(A)t^n}{n!},$$

where $B_n(A)$ is the n-th Bernoulli matrix polynomial defined in (6). In this way, we can use the series expansion

$$e^{At} = \left(\frac{e^t-1}{t}\right)\sum_{n\geq 0}\frac{B_n(A)t^n}{n!}\ ,\ |t|<2\pi, \tag{7}$$

to obtain approximations of the matrix exponential. To do this, let's take s the scaling (to be determined) of the matrix A and take $t=1$ in (7). We use the matrix exponential approximation

$$P_m(A/2^s) = (e-1)\sum_{n=0}^{m}\frac{B_n\left(A/2^s\right)}{n!}. \tag{8}$$

Approximation (8) has the problem that it is not expressed in explicit terms of powers of the matrix $A/2^s$. This explicit relationship is provided below.

Lemma 1. *Given expression (8), we get*

$$\frac{1}{(e-1)}P_m(A/2^s) \quad = \quad \sum_{n=0}^{m}\frac{B_n\left(A/2^s\right)}{n!} = \sum_{i=0}^{m}\alpha_i^{(m)}\left(A/2^s\right)^i, \tag{9}$$

where $\alpha_i^{(m)} = \displaystyle\sum_{k=i}^{m}\binom{k}{k-i}\frac{\mathcal{B}_{k-i}}{k!}$.

Proof: For $m=0$ and $m=1$ formula (9) trivially holds. From (8) we have that, for $m=2$ and using (6), one gets

$$\begin{aligned}
&\frac{1}{(e-1)}P_2(A/2^s) = \sum_{n=0}^{2}\frac{B_n\left(A/2^s\right)}{n!}\\
= \ & \frac{1}{0!}B_0\left(A/2^s\right) + \frac{1}{1!}B_1\left(A/2^s\right) + \frac{1}{2!}B_2\left(A/2^s\right)\\
= \ & \frac{1}{0!}\left(\sum_{k=0}^{0}\binom{0}{k}\mathcal{B}_k\left(A/2^s\right)^{0-k}\right) + \frac{1}{1!}\left(\sum_{k=0}^{1}\binom{1}{k}\mathcal{B}_k\left(A/2^s\right)^{1-k}\right)\\
+ \ & \frac{1}{2!}\left(\sum_{k=0}^{2}\binom{2}{k}\mathcal{B}_k\left(A/2^s\right)^{2-k}\right)
\end{aligned}$$

$$
\begin{aligned}
&= \left(\sum_{k=0}^{2}\frac{1}{k!}\binom{k}{k}\mathcal{B}_k\right)(A/2^s)^0 + \left(\sum_{k=1}^{2}\frac{1}{k!}\binom{k}{k-1}\mathcal{B}_{k-1}\right)(A/2^s)^1 \\
&+ \left(\sum_{k=2}^{2}\frac{1}{k!}\binom{k}{k-2}\mathcal{B}_{k-2}\right)(A/2^s)^2 \\
&= \alpha_0^{(2)}(A/2^s)^0 + \alpha_1^{(2)}(A/2^s)^1 + \alpha_2^{(2)}(A/2^s)^2 \\
&= \sum_{i=0}^{2}\alpha_i^{(2)}(A/2^s)^i,
\end{aligned}
$$

thus formula (9) is true for $m = 2$. Similarly, for $m = 3$ one gets:

$$
\begin{aligned}
&\frac{1}{(e-1)}P_3(A/2^s) = \sum_{n=0}^{3}\frac{B_n(A/2^s)}{n!} \\
&= \frac{1}{0!}B_0(A/2^s) + \frac{1}{1!}B_1(A/2^s) + \frac{1}{2!}B_2(A/2^s) + \frac{1}{3!}B_3(A/2^s) \\
&= \frac{1}{0!}\left(\sum_{k=0}^{0}\binom{0}{k}\mathcal{B}_k(A/2^s)^{0-k}\right) + \frac{1}{1!}\left(\sum_{k=0}^{1}\binom{1}{k}\mathcal{B}_k(A/2^s)^{1-k}\right) \\
&+ \frac{1}{2!}\left(\sum_{k=0}^{2}\binom{2}{k}\mathcal{B}_k(A/2^s)^{2-k}\right) + \frac{1}{3!}\left(\sum_{k=0}^{3}\binom{3}{k}\mathcal{B}_k(A/2^s)^{3-k}\right) \\
&= \left(\sum_{k=0}^{3}\frac{1}{k!}\binom{k}{k}\mathcal{B}_k\right)(A/2^s)^0 + \left(\sum_{k=1}^{3}\frac{1}{k!}\binom{k}{k-1}\mathcal{B}_{k-1}\right)(A/2^s)^1 \\
&+ \left(\sum_{k=2}^{3}\frac{1}{k!}\binom{k}{k-2}\mathcal{B}_{k-2}\right)(A/2^s)^2 + \left(\sum_{k=3}^{3}\frac{1}{k!}\binom{k}{k-3}\mathcal{B}_{k-3}\right)(A/2^s)^3 \\
&= \alpha_0^{(3)}(A/2^s)^0 + \alpha_1^{(3)}(A/2^s)^1 + \alpha_2^{(3)}(A/2^s)^2 + \alpha_3^{(3)}(A/2^s)^3 \\
&= \sum_{i=0}^{3}\alpha_i^{(3)}(A/2^s)^i,
\end{aligned}
$$

and formula (9) is also true for $m = 3$. We will use now an induction argument. Suppose that formula (9) it is valid for m and let's see for $m + 1$. Using (6) together with the induction hypothesis, one gets

$$
\begin{aligned}
&\frac{1}{(e-1)}P_{m+1}(A/2^s) = \sum_{n=0}^{m+1}\frac{B_n(A/2^s)}{n!} \\
&= \sum_{n=0}^{m}\frac{B_n(A/2^s)}{n!} + \frac{1}{(m+1)!}B_{m+1}(A/2^s) \\
&= \sum_{i=0}^{m}\alpha_i^{(m)}(A/2^s)^i + \frac{1}{(m+1)!}\left(\sum_{k=0}^{m+1}\binom{m+1}{k}\mathcal{B}_k(A/2^s)^{m+1-k}\right)
\end{aligned}
$$

$$
\begin{aligned}
&= \left(\alpha_0^{(m)} + \binom{m+1}{m+1}\frac{\mathcal{B}_{m+1}}{(m+1)!}\right)(A/2^s)^0 + \cdots + \left(\alpha_m^{(m)} + \binom{m+1}{1}\frac{\mathcal{B}_1}{(m+1)!}\right)(A/2^s)^m \\
&+ \left(\binom{m+1}{0}\frac{\mathcal{B}_0}{(m+1)!}\right)(A/2^s)^{m+1} \\
&= \left(\sum_{k=0}^{m+1}\binom{k}{k}\frac{\mathcal{B}_k}{k!}\right)(A/2^s)^0 + \left(\sum_{k=1}^{m+1}\binom{k}{k-1}\frac{\mathcal{B}_{k-1}}{k!}\right)(A/2^s)^1 + \cdots \\
&\cdots + \left(\sum_{k=m+1}^{m+1}\binom{k}{k-(m+1)}\frac{\mathcal{B}_{k-(m+1)}}{k!}\right)(A/2^s)^{m+1} \\
&= \sum_{i=0}^{m+1}\alpha_i^{(m+1)}(A/2^s)^i .\square
\end{aligned}
$$

3. The proposed algorithms

The matrix polynomial $P_m(A/2^s)$ from (8) can be computed efficiently in terms of matrix products using values for m in the set

$$m_k \in \{2, 4, 6, 9, 12, 16, 20, 25, 30, \ldots\},$$

with $k = 1, 2, \ldots$, by using the Paterson–Stockmeyer's method [30]. If we consider $P_{m_k}(A)$, then:

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

where $q = \lceil\sqrt{m_k}\rceil$ or $q = \lfloor\sqrt{m_k}\rfloor$. Taking into account Table 4.1 from [4, pp. 74], the computational cost in terms of matrix products of (10) is k. To obtain the exponential of matrix A with enough precision and efficiency, it is necessary to determine the values of m and s in expression (8). Once these values have been determined, the approximation (1) is used to compute e^A by means of Bernoulli matrix polynomials.

For obtaining the optimal order of the series expansion m_k and the scaling parameter s, we have used a similar error analysis as in [12]:

Let be

$$A \in \Omega_m = \left\{X \in {}^{n\times n} : \rho\left(e^{-X}T_m(X) - I\right) < 1\right\},$$

where $\rho(\cdot)$ is the spectral radius of a matrix and $T_m(X)$ the Taylor approximation of matrix exponential of order m.

The backward error for computing e^A can be defined as the matrix ΔA such that $e^{A+\Delta A} = T_m(A)$. It can be verified that

$$\Delta A \backsimeq \sum_{k \geq m+1} c_k^{(m)} A^k,$$

where $\sum_{k \geq m+1} c_k^{(m)} A^k$ is the backward error of matrix exponential for the Taylor approximation. The absolute backward error can be bound as follows:

$$\begin{aligned} E_{ab}(A) = \|\Delta A\| &\simeq \left\| \sum_{k \geq m+1} c_k^{(m)} A^k \right\| \leq \sum_{k \geq m+1} |c_k^{(m)}| \Big(||A^k||^{1/k} \Big)^k \\ &\leq \sum_{k \geq m+1} |c_k^{(m)}| \beta_m^k, \end{aligned} \tag{11}$$

where $\beta_m = \max\left\{ ||A^k||^{1/k},\ k \geq m+1 \right\}$. Theorem 2 from [12] shows that $\beta_m = ||A^{k_0}||^{1/k_0}$, where $m+1 \leq k_0 \leq 2m+1$, and that value can be approximated by means of

$$\beta_m \simeq \max\left\{ a_{m+1}^{1/(m+1)}, a_{m+2}^{1/(m+2)} \right\},$$

where a_{m+1} and a_{m+2} are the 1–norm estimation of $||A^{m+1}||$ and $||A^{m+2}||$, respectively, using the block 1–norm estimation algorithm of Higham and Tisseur [31]. Let be

$$\Theta_m^{(ab)} = \max\left\{ \theta \geq 0 : \sum_{k \geq m+1} \left| c_k^{(m)} \right| \theta^k \leq u \right\}, \tag{12}$$

where $u = 2^{-53}$ is the unit roundoff in IEEE double precision floating-point arithmetic. If an integer $s \geq 0$ verifies $2^{-s}\beta_m < \Theta_m^{(ab)}$, then

$$E_{ab}(A/2^s) \leq \sum_{k \geq m+1} \left| c_k^{(m)} \right| (\Theta_m^{(ab)})^k < u$$

and m and s will be taken, respectively, as the adequate order of the polynomial and the scaling parameter.

Using the above reasoning, the relative backward error can be bound in the following way:

$$\begin{aligned} E_{rb}(A) = \frac{\|\Delta A\|}{\|A\|}| &\simeq \frac{\left\| \sum_{k \geq m+1} c_k^{(m)} A^k \right\|}{\|A\|} \leq \frac{\sum_{k \geq m+1} |c_k^{(m)}| \left\| A^k \right\|}{\|A\|} \\ &= \sum_{k \geq m} |c_{k+1}^{(m)}| \Big(||A^k||^{1/k} \Big)^k \leq \sum_{k \geq m} |c_{k+1}^{(m)}| \beta_m^k. \end{aligned} \tag{13}$$

Let be

$$\Theta_m^{(rb)} = \max\left\{\theta \geq 0 : \sum_{k\geq m}\left|c_{k+1}^{(m)}\right|\theta^k \leq u\right\}. \tag{14}$$

Therefore, if an integer $s \geq 0$ satisfies $2^{-s}\beta_m < \Theta_m^{(rb)}$, then

$$E_{rb}(A/2^s) \leq \sum_{k\geq m}\left|c_{k+1}^{(m)}\right|(\Theta_m^{(rb)})^k < u$$

and the polynomial order m and the scaling parameter s will have been obtained.

The $\Theta_m^{(ab)}$ and $\Theta_m^{(rb)}$ parameters were worked out, with the required precision by using symbolic computations, from $m = 2$ to $m = 30$. Then, the maximum value Θ_m between $\Theta_m^{(ab)}$ and $\Theta_m^{(rb)}$ was computed for each m, i.e. $\Theta_m = max(\Theta_m^{(ab)}, \Theta_m^{(rb)})$, giving place to the Θ_m parameter included in the second column of Table 2 from [12]. As a result, Θ_m equals $\Theta_m^{(ab)}$ when $m \leq 16$ and Θ_m matches $\Theta_m^{(rb)}$ otherwise. On other words, the absolute backward error was considered when the polynomial order is less or equal than 16 and the relative backward error was taken into account for greater values.

Algorithm 1 computes the matrix exponential function based on Bernoulli series and the Paterson–Stockmeyer's method. Step 1 of this algorithm uses the procedure previously described to obtain the values of m and s (as known, a more detailed description can be found in [12]). In steps 2, the Bernoulli approximation is employed, depending of value m calculated in step 1. Finally, in steps 3-5, the matrix exponential is recovered.

Algorithm 1 Scaling and squaring Bernoulli algorithm for computing $B = e^A$, where $A \in \mathbb{C}^{r\times r}$ and m_M is the maximum approximation order allowed.

1: Choose adequate order $m_k \leqslant m_M$ and scaling parameter $s \in \mathbb{N} \cup \{0\}$
2: $B = P_{m_k}(A/2^s)$ by using (10) ($P_{m_k}(\cdot)$ Bernoulli matrix polynomial)
3: **for** $i = 1 : s$ **do** ▷ Recovering the matrix exponential
4: $\quad B = B^2$
5: **end for**

Bearing in mind (9), if $P_m(A/2^s)$ represents the matrix polynomial of order m corresponding to the Bernoulli approximation of exponential of matrix $A/2^s$ and $T_m(A/2^s)$ denotes the same matrix polynomial but according to the Taylor approach, then:

$$P_m(A/2^s) = (e-1)\sum_{i=0}^{m}\alpha_i^{(m)}\left(A/2^s\right)^i = \sum_{i=0}^{m} b_i^{(m)}\left(A/2^s\right)^i, \tag{15}$$

$$T_m(A/2^s) = \sum_{i=0}^{m}\frac{1}{i!}\left(A/2^s\right)^i = \sum_{i=0}^{m} t_i\left(A/2^s\right)^i.$$

Table 1: Approximation polynomial coefficient vector differences between Bernoulli (b) and Taylor (t) methods.

m	‖b-t‖	‖b-t‖/‖t‖
2	5.023311e-01	2.009324e-01
4	5.695696e-02	2.103026e-02
6	2.741618e-03	1.008669e-03
9	1.293850e-05	4.759808e-06
12	4.657888e-08	1.713541e-08
16	2.819122e-11	1.037097e-11
20	1.445479e-14	5.317621e-15
25	2.735502e-19	1.006335e-19
30	4.901565e-22	1.803185e-22

The coefficients $b_i^{(m)}$ of the Bernoulli approximation polynomial differ significantly from those of the Taylor one, t_i, with $i = 0, \ldots, m$, when $m \in \{2, 4, 6, 9, 12, 16, 20\}$. However, they are practically identical for $m \in \{25, 30, \ldots\}$. This is so because, as the degree m of the Bernoulli polynomial increases, all its coefficients $b_i^{(m)}$ vary, approaching the corresponding Taylor ones. Table 1 collects the 1-norm of the absolute and relative differences between the coefficient vectors of Bernoulli and Taylor polynomial approximations for different values of m. As it can be seen, the 1-norm of these differences is less than the unit roundoff ($u = 2^{-53} \backsimeq 1.11 \times 10^{-16}$) when $m = 25$ or $m = 30$. As an example, the 1-norm of the relative difference between these coefficients when $m = 25$ is equal to 1.006335×10^{-19}. As expected, the differences are smaller when m increases.

Therefore, these experimental results show that backward errors expressed in (11) and (13) are only fulfilled for Bernoulli approximation for values of m greater than or equal to 25, but not for lower values. As a consequence of this analysis carried out, the Algorithm 2 has been developed. It computes the matrix exponential by means of Taylor series, when m is less than or equal to 20, or by means of Bernoulli series, when m is equal to 25 or 30.

Algorithm 2 Scaling and squaring Bernoulli algorithm for computing $B = e^A$, where $A \in \mathbb{C}^{r \times r}$ and m_M is the maximum approximation order allowed.

Choose adequate order $m_k \leqslant m_M$ and scaling parameter $s \in \mathbb{N} \cup \{0\}$
if $m_k \leq 20$ **then**
    $B = P_{m_k}(A/2^s)$ using (10) ($P_{m_k}(\cdot)$ Taylor matrix polynomial)
else
    $B = P_{m_k}(A/2^s)$ using (10) ($P_{m_k}(\cdot)$ Bernoulli matrix polynomial)
end if
for $i = 1 : s$ **do** ▷ Recovering the matrix exponential
    $B = B^2$
end for

4. Numerical experiments

In this section, we will firstly compare `expmber`, the MATLAB implementation corresponding to Algorithm 1, based on Bernoulli approximation, with the functions `exptaynsv3` [12], that computes the matrix exponential using Taylor matrix polynomials, and `expm_new` [8], which implements a scaling and squaring Padé-based algorithm to work out the mentioned matrix function. Next, we will compare `expmbertay`, the function that combines Taylor and Bernoulli approximations in accordance with Algorithm 2, with `expmber`, `exptaynsv3` and `expm_new`.

Algorithm 3 computes the *"exact"* matrix exponential function thanks to MATLAB's Symbolic Math Toolbox with 256 digits of precision. This algorithm provides the exact solution when it finds that the relative error between $T_{m_k}^{(j)}(n)$ and $T_{m_{k-1}}^{(i)}(n)$ is less than $u = 2^{-53}$ (see (18)), where $T_m^{(s)}(n)$ is the Taylor matrix approximation of order m of the scaled matrix $A/2^s$, computed with n digits of precision by using the `vpa` (variable-precision arithmetic) MATLAB function in the Algorithm 4. Previously, $T_{m_k}^{(j)}(n)$ and $T_{m_{k-1}}^{(i)}$ have been calculated so that (16) and (17) are fulfilled.

Algorithm 3 Computes the *"exact"* matrix exponential $T = e^A$, where $A \in \mathbb{C}^{r\times r}$, by means of Taylor expansion.

1: **if** there exist two consecutive orders $m_{k-1},\ m_k \in \{30, 36, 42, 49, 56, 64\}$ and integers $1 \leq i, j \leq 15$ for s such that

$$\frac{\left\|T_{m_{k-1}}^{(i)}(n) - T_{m_{k-1}}^{(i-1)}(n)\right\|_1}{\left\|T_{m_{k-1}}^{(i)}(n)\right\|_1} < u, \tag{16}$$

and

$$\frac{\left\|T_{m_k}^{(j)}(n) - T_{m_k}^{(j-1)}(n)\right\|_1}{\left\|T_{m_k}^{(j)}(n)\right\|_1} < u, \tag{17}$$

and

$$\frac{\left\|T_{m_k}^{(j)}(n) - T_{m_{k-1}}^{(i)}(n)\right\|_1}{\left\|T_{m_k}^{(j)}(n)\right\|_1} < u \tag{18}$$

by using Algorithm 4, **then**
return $T = T_{m_k}^{(j)}(n)$

2: **else**
return error

3: **end if**

Algorithm 4 Computes $T_m^{(s)}(n) = e^A$, where $A \in \mathbb{C}^{r\times r}$, by Taylor expansion of order m and scaling parameter s using vpa MATLAB function with n digits of precision.

1: Compute $T_m^{(s)}(n) = P_m(A/2^s)$ using Taylor expansion of order m with n digits of precision
2: **for** $i = 1 : s$ **do**
3: $\quad T_m^{(s)}(n) = [T_m^{(s)}(n)]^2$
4: **end for**

4.1. Experiments description

The following test battery, composed of three types of different and representative matrices, has been chosen to compare the numerical performance of the above described codes:

a) One hundred diagonalizable 128×128 real matrices with 1-norms varying from 2.18 to 207.52. These matrices have the form $A = VDV^T$, where D is a diagonal matrix with real and complex eigenvalues and V is an orthogonal matrix obtained as $V = H/\sqrt{128}$, being H the Hadamard matrix. The *"exact"* matrix exponential was computed as $\exp(A) = V\exp(D)V^T$ (see [4, pp. 10]).

b) One hundred non-diagonalizable 128×128 complex matrices with 1-norms ranging from 84 to 98. These matrices have the form $A = VJV^T$, where J is a Jordan matrix with complex eigenvalues of modulus less than 10 and random algebraic multiplicity varying from 1 to 5. V is an orthogonal matrix obtained as $V = H/\sqrt{128}$, where H is the Hadamard matrix. The *"exact"* matrix exponential was worked out as $\exp(A) = V\exp(J)V^T$.

c) State-of-the-art matrices:

- Forty 128×128 matrices from the Matrix Computation Toolbox (MCT) [32].
- Sixteen matrices from the Eigtool MATLAB package (EMP) [33] with sizes 127×127 and 128×128.

  The *"exact"* matrix exponential for these matrices was computed by using Taylor approximations of orders 30, 36, 42, 49, 56 and 64, changing their scaling parameter (see Algorithm 3).

  Although the MCT and the EMP are initially composed of fifty-two and twenty matrices, respectively, twelve from the MCT and four from the EMP matrices were discarded for different reasons. For example, matrices numbers 5, 10, 16, 17, 21, 25, 26, 42, 43, 44 and 49 belonging to the MCT and matrices numbers 5 and 6 appertaining to the EMP were not taken into account since the exact exponential solution could not be computed. Besides, matrix number 2 from the MCT and matrices numbers 3 and 10 from the EMP were not

Table 2: Matrix products (P) for Tests 1, 2 and 3 using `expmber`, `exptaynsv3` and `expm_new` MATLAB codes.

	`P(expmber)`	`P(exptaynsv3)`	`P(expm_new)`
Test 1	1131	1131	1178.33
Test 2	1100	1100	1227.33
Test 3	617	617	654.67

considered because the excessively high relative error provided by all the methods to be compared.

An experiment, called Test, is performed for each of the three sets of matrices described, respectively, that evaluates the computational cost and the numerical accuracy of the methods under comparison. The three tests have been executed using 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.

4.2. Experimental results

Table 2 shows the computational costs of each method represented in terms of the number of matrix products (`P`), taking into account that the cost of the rest of the operations is negligible compared to it for big enough matrices. As it can be seen, `expmber` and `exptaynsv3` achieved an identical number of matrix multiplications, since the same algorithm was used by both of them to calculate the degree of the polynomial (m) and the value of the scaling (s). This number of products was lower than that required by `expm_new`, which gave rise to the highest computational cost. In addition to the matrix products, `expm_new` solves a system of linear equations with n right-hand side vectors where n represents the size of the square coefficient matrix, whose computational cost was approximated as 4/3 matrix products.

Table 3, on the other hand, shows the percentage of cases in which the relative errors of `expmber` are lower, greater or equal than those of `exptaynsv3` and `expm_new`. More in detail, the relative error was computed as

$$\mathrm{E} = \frac{\| \exp(A) - e\tilde{x}p(A)\|_1}{\|\exp(A)\|_1}$$

where $e\tilde{x}p(A)$ is the approximate solution and $\exp(A)$ is the exact one.

With the exception of Test 3, the Bernoulli approach resulted in relative errors lower than those of Taylor one. With regard to Padé, the Bernoulli algorithm always offered considerably more accurate results, which reached up to 100% of the matrices for Test 2.

For the three tests, respectively, the normwise relative errors (a), the performance profiles (b), the ratio of the relative errors (c) and the ratio of the matrix products (d) among the distinct compared methods have been plotted in Figures 1, 2, and 3.

Table 3: Relative error comparison among `expmber` vs `exptaynsv3` and `expmber` vs `expm_new` for the three tests.

	Test 1	Test 2	Test 3
E(`expmber`)<E(`exptaynsv3`)	56%	91%	30.36%
E(`expmber`)>E(`exptaynsv3`)	43%	9%	62.5%
E(`expmber`)=E(`exptaynsv3`)	1%	0%	7.14%
E(`expmber`)<E(`expm_new`)	97%	100%	69.64%
E(`expmber`)>E(`expm_new`)	3%	0%	30.36%
E(`expmber`)=E(`expm_new`)	0%	0%	0%

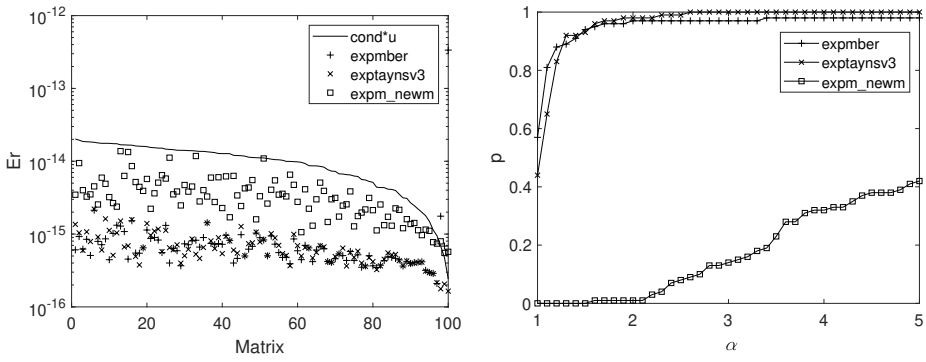
Regarding the normwise relative errors presented in Figures 1a, 2a and 3a, their solid line represents the function $k_{\exp}u$, where $k_{\exp}$ (or *cond*) is the condition number of the matrix exponential function [4, Chapter 3] and u is the unit roundoff. In general, `expmber` exhibited a very good numerical stability. This can be appreciated seeing the distance from each matrix normwise relative error to the $cond * u$ line. In Figures 1a and 2a, the numerical stability is even much better because these errors are below this line. Because $k_{\exp}$ was infinite or enormously high for the matrices 6, 7, 12, 15, 23, 36, 39, 50 and 51 from the MCT and for the matrices 1, 4, 8 and 15 from the EMP, all of them were rejected in the Figure 3a visualisation but considered in the other ones.

In the performance profile Figures (1b, 2b and 3b), the α coordinate, on the x-axis, varies from 1 to 5 in steps equal to 0.1. For a concrete α value, the p coordinate, on the y-axis, means the probability that the considered algorithm has a relative error lower than or equal to α-times the smallest relative error over all the methods on the given test. For the first two tests (Figures 1b, 2b), the performance profile shows that Bernoulli and Taylor methods accuracy was similar. Both of them had much better correctness than the Padé method. Notwithstanding, Figure 3b reveals that `exptaynsv3` code improved the result accuracy against the `expmber` function, for the Test 3.

In Figures 1c, 2c and 3c, the ratios of relative errors have been presented in decreasing order with respect to E(`expmber`)/E(`exptaynsv3`). They confirm the data exposed in Table 3, where it was shown that `expmber` provides more accurate results than `exptaynsv3` for Tests 1 and 2, but not for Test 3. It is obvious to note that Padé offered the worst performance in most cases.

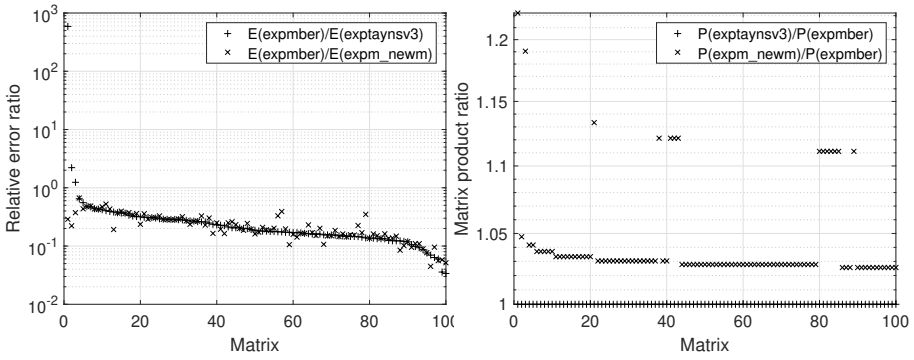
In our opinion, this is clearly due to the distinctive numerical characteristics of the 3 sets of matrices analysed and the degree of the polynomial (m) required to be used. According to our experience, `expmber` provides results with a very appropriate accuracy for values of m equal to 25 or 30. However, for significantly lower values, `expmber` will be less competitive than other codes, such as `exptaynsv3`. Minimum, maximum and average values of m required for Tests 1, 2 and 3 are collected in Table 4. In more detail, Figure 4 shows the approximation polynomial order employed in the calculation of the exponential function by means of `expmber` (or `exptaynsv3`) for each of the matrices that are part of the test battery.

As it was presented in Table 2, `expmber` and `exptaynsv3` functions per-



(a) Normwise relative errors.

(b) Performance profile.



(c) Ratio of relative errors.

(d) Ratio of matrix products.

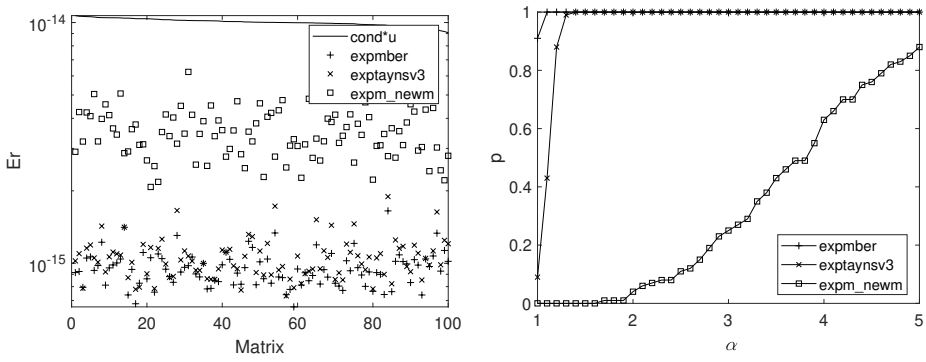
Figure 1: Experimental results for Test 1.

Table 4: Minimum, maximum and average polynomial degree (m) required for Tests 1, 2 and 3 using `expmber` or `exptaynsv3` functions.

	Minimum	Maximum	Average
Test 1	16	30	27.51
Test 2	30	30	30
Test 3	12	30	25.70

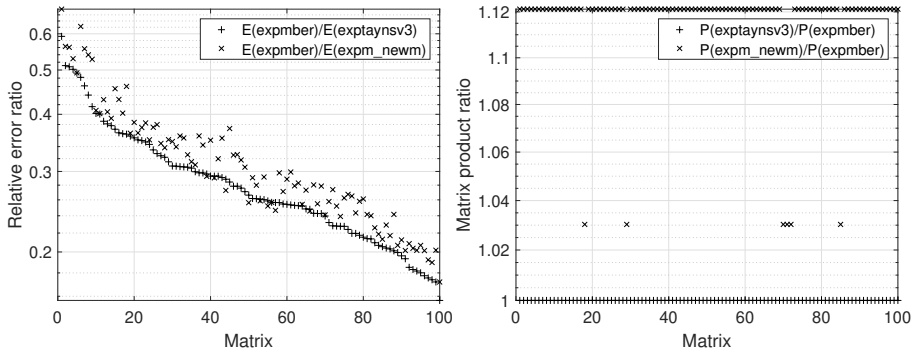
formed a lower number of matrix operations than `expm_new` one. This statement can be also corroborated from the results displayed in Figures 1d, 2d and 3d, where the ratio between the number of `expm_new` and `expmber` matrix products ranged from 1.03 to 1.22 for Test 1, from 1.03 to 1.12 for Test 2 and from 0.67 to 2.87 for Test 3.

Next, we will analyse the idea of using Bernoulli and Taylor methods together, giving place to a novel approach to compute the matrix exponential function. For that, we will start getting the benefits of the `exptaynsv3` function against `expm_new` one. As Table 5 shows, the percentage of cases in which



(a) Normwise relative errors.

(b) Performance profile.



(c) Ratio of relative errors.

(d) Ratio of matrix products.

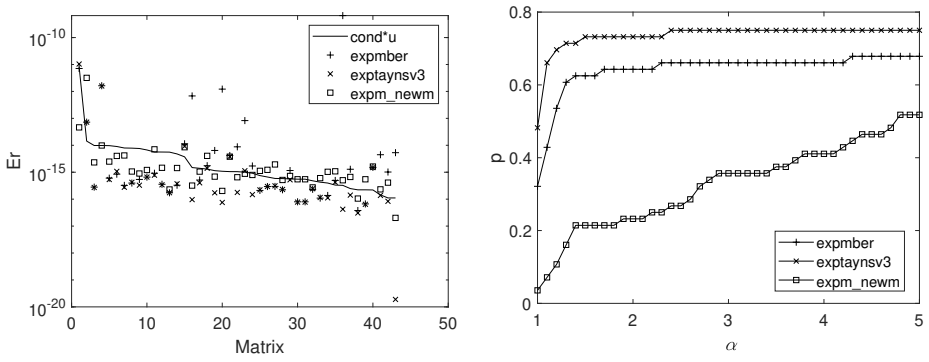
Figure 2: Experimental results for Test 2.

Table 5: Relative error comparison between `exptaynsv3` and `expm_new` for the three tests.

	Test 1	Test 2	Test 3
E(`exptaynsv3`)<E(`expm_new`)	100%	100%	89.29%
E(`exptaynsv3`)>E(`expm_new`)	0%	0%	10.71%
E(`exptaynsv3`)=E(`expm_new`)	0%	0%	0%

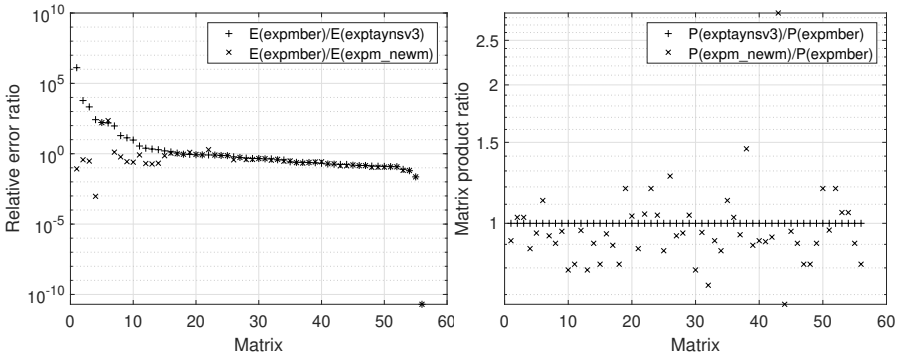
Taylor relative error is lower than Padé reaches 100% for Test 1 and 2, and 89.29% for Test 3. Evidently, these error percentages improve those offered by Bernoulli approximation with respect to `expm_new`, as described in Table 3.

From these excellent results, we therefore considered the possibility of combining Bernoulli and Taylor methods, giving rise to the `expmbertay` code. In this new function, and according to the comparison between the coefficients of their polynomials carried out previously, we will use the Taylor approach (`exptaynsv3`) for values of m below 25 and the Bernoulli approximation (`expmber`) when m equals 25 or 30. In this way, the number of matrix products needed by `expmbertay` will obviously be identical to that of `expmber` or



(a) Normwise relative errors.

(b) Performance profile.



(c) Ratio of relative errors.

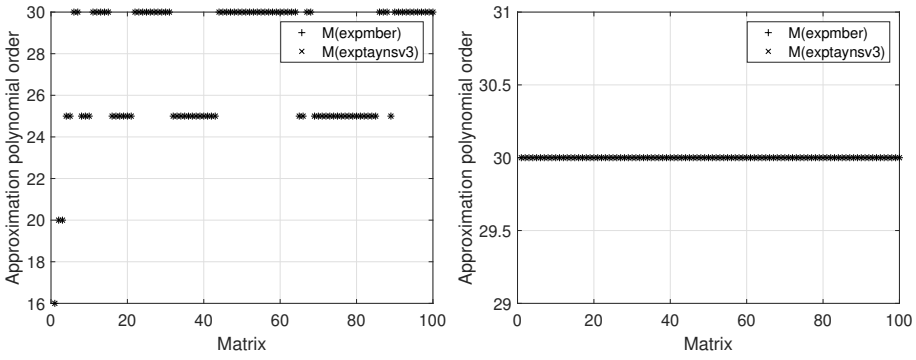
(d) Ratio of matrix products.

Figure 3: Experimental results for Test 3.

`exptaynsv3`.

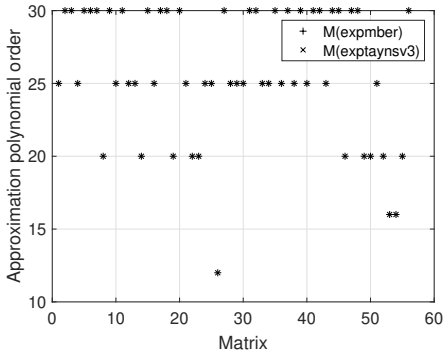
Table 6 collects thus the percentage of matrices in which the relative errors of `expmbertay` are lower, greater or equal than those of `exptaynsv3`, `expmber` and `expm_new`. For the vast majority of matrices, `expmbertay` provided an accuracy in the results practically identical to that of `expmber`, but improving even the latter by 23.21% for the matrices of Test 3. With respect to `exptaynsv3`, `expmbertay` also enhanced the results achieved by `expmber`, so that this combined method is now better or equal than `exptaynsv3` in 55.36% of cases for Test 3. Moreover, `expmbertay` became better than `expm_new` in 100% of the matrices for Test 1 and 2, and 91.07% for Test 3, which is higher than the percentages individually offered by `expmber` (69.64%) and `exptaynsv3` (89.29%).

Numerical features of `expmbertay` are finally exposed in Figures 5, 6 and 7 for the three tests by means of the normwise relative errors (a), the performance profiles (b) and the ratio of the relative errors (c). As it can be seen, the method presents an excellent precision in the results, with very low relative errors and a very high probability in the performance profile pictures.



(a) Test 1.

(b) Test 2.



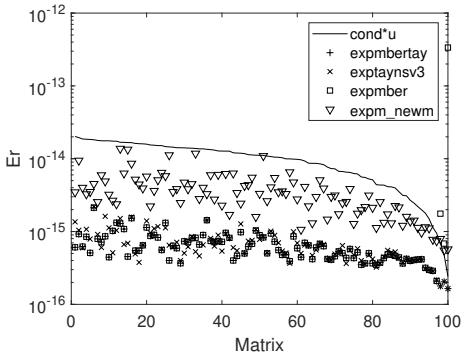
(c) Test 3.

Figure 4: Polynomial order (m) for Test 1, 2 and 3.

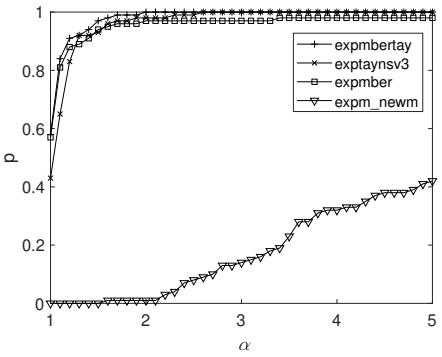
Table 6: Relative error comparison among `expmbertay` vs `exptaynsv3`, `expmbertay` vs `expmber`, and `expmbertay` vs `expm_new` for the three tests.

	Test 1	Test 2	Test 3
E(`expmbertay`)<E(`exptaynsv3`)	57%	91%	44.64%
E(`expmbertay`)>E(`exptaynsv3`)	42%	9%	44.64%
E(`expmbertay`)=E(`exptaynsv3`)	1%	0%	10.72%
E(`expmbertay`)<E(`expmber`)	3%	0%	23.21%
E(`expmbertay`)>E(`expmber`)	0%	0%	0%
E(`expmbertay`)=E(`expmber`)	97%	100%	76.79%
E(`expmbertay`)<E(`expm_new`)	100%	100%	91.07%
E(`expmbertay`)>E(`expm_new`)	0%	0%	8.93%
E(`expmbertay`)=E(`expm_new`)	0%	0%	0%

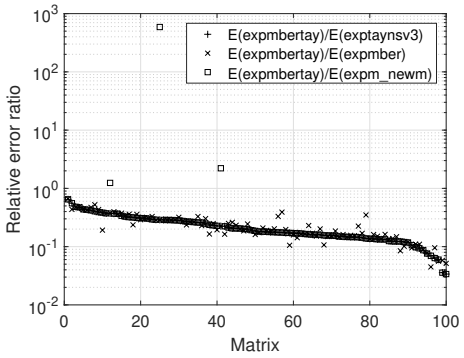
We have also included into the developed software an "accelerated" version of the `expmber` function that computes the matrix exponential on an NVIDIA GPU. Matrix multiplication is an operation very rich in intrinsic parallelism that



(a) Normwise relative errors.



(b) Performance profile.

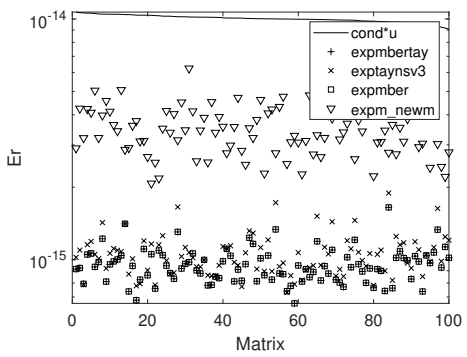


(c) Ratio of relative errors.

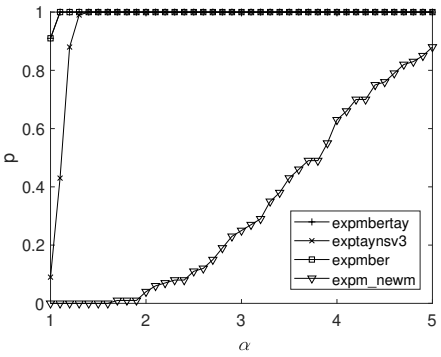
Figure 5: Experimental results for Test 1.

can be optimized for GPUs. Algorithms that rely on many matrix multiplications, like the one proposed, can take full advantage of these devices through the use of the cuBLAS [34] package. Our "GPU version" uses the regular MATLAB scripting language in the same way as in the other algorithms used so far but, at some points in the code, a function implemented into a MEX file is called. This function is implemented in CUDA [35] and dispatches the operation described in the function to the GPU. This way, all the matrix products are computed by the GPU presented in the computing platform. The exact implementation details about how these MEX files are implemented can be found in [36].

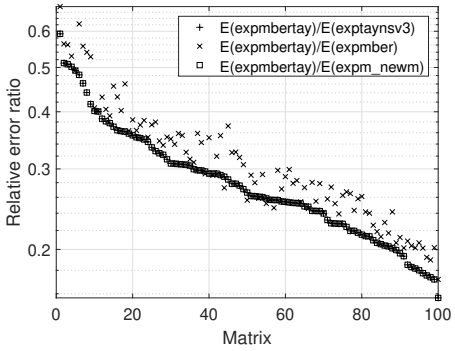
The experimental results corresponding to this part of the work were carried out on a computer equipped with two processors Intel Xeon CPU E5-2698 v4 @2.20GHz (*Broadwell* architecture) featuring 20 cores each. The regular MATLAB files, i.e. all those that do not make use of the GPU through a MEX file,



(a) Normwise relative errors.



(b) Performance profile.



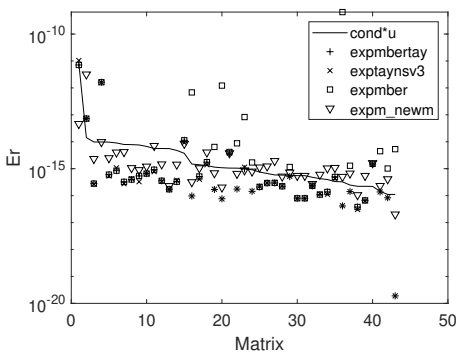
(c) Ratio of relative errors.

Figure 6: Experimental results for Test 2.

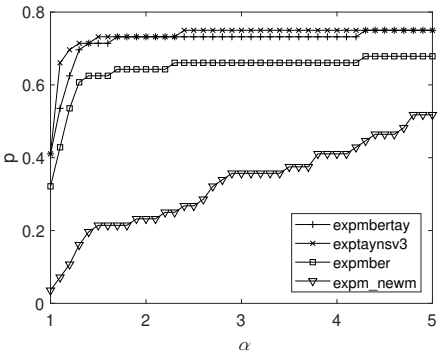
use the 40 cores available in the target computer by default[1]. We denote this implementation as the "CPU version" when compared with the "GPU version" described above. To get the algorithm performance on the GPU, we used one NVIDIA Tesla P100-SXM2 (*Kepler* architecture) that counts on 3584 CUDA cores and 16 GB of memory.

Figure 8 shows the execution time in seconds on the left and the speed up achieved with the GPU version with regard to its CPU counterpart on the right. The plots also compare the performance of the former algorithm based on Taylor series (`exptaynsv3`) with the new one based on Bernoulli series (`expmber`) presented here. At the light of the figure, it can be concluded that both algo-

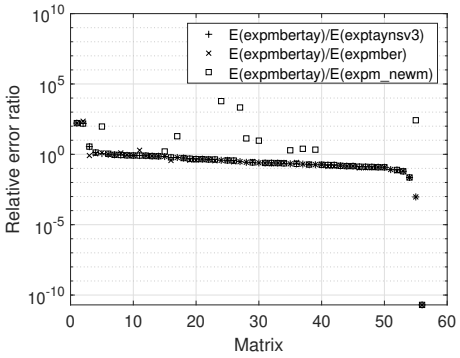
[1] "Linear algebra and numerical functions such as `fft`, `\` `(mldivide)`, `eig`, `svd`, and `sort` are multithreaded in MATLAB. Multithreaded computations have been on by default in MATLAB since Release 2008a." In particular, MATLAB uses the Intel MKL, where the matrix multiplication is *threaded*, i.e. is a parallel implementation with OpenMP.



(a) Normwise relative errors.



(b) Performance profile.



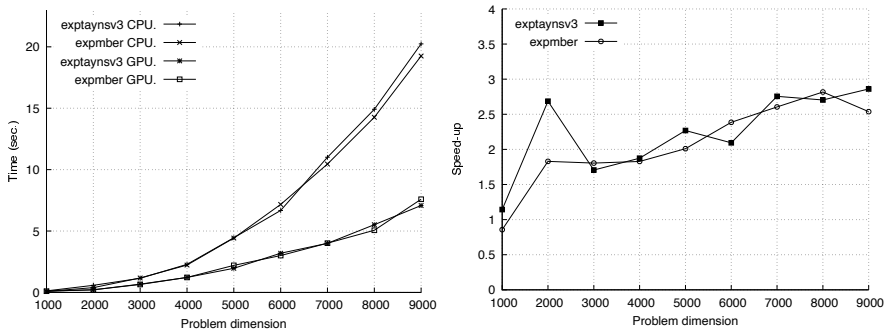
(c) Ratio of relative errors.

Figure 7: Experimental results for Test 3.

rithms, `exptaynsv3` and `expmber`, behave very similarly. The reduction in time obtained with the GPU with respect to the CPU starts approximately with matrices of size $n = 1000$ and increases with the problem size. The weight of both algorithms falls on the same basic computational kernel (matrix multiplications) and both of them require an identical number of them. The computational performance of routine `expmbertay` would be very similar to `expmber`, since it uses once again the same number of matrix products.

5. Conclusions

The starting point of this work is a new expression of the matrix exponential function cast in terms of Bernoulli matrix polynomials. Using this series expansion, a new method for calculating the exponential of a matrix (implemented as `expmber` code) has been developed. The proposed algorithm has been tested using a state-of-the-art matrix test battery with different features (diagonalizable and non diagonalizable, with particular eigenvalue spectrum) that covers a wide



(a) Execution time.

(b) Speed up (GPU time / CPU time).

Figure 8: Execution time (a) and speed up (b) of the algorithm to compute the matrix exponential using the Taylor series (`exptaynsv3`) and the Bernoulli (`expmber`) series on CPU and on GPU for large randomly generated matrices.

range of cases. The developed code has been compared with the best implementations available, i.e. Padé-based algorithm (`expm_new`) and Taylor-based one (`exptaynsv3`), outperforming Padé-based algorithm and giving results at the level of Taylor-based solutions in both accuracy and computational cost.

Preliminary results with the Bernoulli version for the matrix exponential function motivated us to develop a hybrid code (called `expmbertay`) that combines the best of both the Taylor and Bernoulli solutions and resulting in excellent results. Therefore, `expmbertay` code is clearly competitive and highly recommended for the matrix exponential calculation, regardless of the type of matrix to be computed. Finally, we showed that the two algorithms developed in this contribution keep the advantages of other ones based on matrix polynomial expansion. Since they all are based on matrix multiplications, the GPU version implemented has turned out to be a strong tool to compute the matrix exponential approximation when the numerical methods employed are stressed with large dimension matrices.

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.

References

[1] C. F. V. Loan, A study of the matrix exponential, numerical analysis report, Tech. rep., Manchester Institute for Mathematical Sciences, The University

(2006).

[2] C. B. Moler, C. V. Loan, Nineteen dubious ways to compute the exponential of a matrix, SIAM Rev. 20 (4) (1978) 801–836.

[3] C. B. Moler, C. V. Loan, Nineteen dubious ways to compute the exponential of a matrix, twenty-five years later*, SIAM Rev. 45 (2003) 3–49.

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

[5] M. Benzi, E. Estrada, C. Klymko, Ranking hubs and authorities using matrix functions, Linear Algebra and its Applications 438 (2013) 2447–2474.

[6] G. A. Baker, P. Graves-Morris, Padé Approximants, Encyclopedia of Mathematics and its Applications, Cambridge University Press, 1996.

[7] L. Dieci, A. Papini, Padé approximation for the exponential of a block triangular matrix, Linear Algebra Appl. 308 (2000) 183–202.

[8] A. H. Al-Mohy, N. J. Higham, A new scaling and squaring algorithm for the matrix exponential, SIAM J. Matrix Anal. Appl. 31 (3) (2009) 970–989.

[9] N. J. Higham, The scaling and squaring method for the matrix exponential revisited, Tech. Rep. 452, Manchester Centre for Computational Mathematics (2004).

[10] R. B. Sidje, Expokit: A software package for computing matrix exponentials, ACM Trans. Math. Softw. 24 (1) (1998) 130–156.

[11] J. Sastre, J. Ibáñez, E. Defez, P. Ruiz, New scaling-squaring Taylor algorithms for computing the matrix exponential, SIAM Journal on Scientific Computing 37 (1) (2015) A439–A455.

[12] P. Ruiz, J. Sastre, J. Ibáñez, E. Defez, High performance computing of the matrix exponential, Journal of Computational and Applied Mathematics 291 (2016) 370–379.

[13] J. Sastre, J. Ibáñez, E. Defez, Boosting the computation of the matrix exponential, Applied Mathematics and Computation 340 (2019) 206–220.

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

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

[16] F. W. Olver, D. W. Lozier, R. F. Boisvert, C. W. Clark, NIST handbook of mathematical functions hardback and CD-ROM, Cambridge University Press, 2010.

[17] E. Tohidi, K. Erfani, M. Gachpazan, S. Shateyi, A new Tau method for solving nonlinear Lane-Emden type equations via Bernoulli operational matrix of differentiation, Journal of Applied Mathematics 2013 (2013).

[18] A. W. Islam, M. A. Sharif, E. S. Carlson, Numerical investigation of double diffusive natural convection of co_2 in a brine saturated geothermal reservoir, Geothermics 48 (2013) 101–111.

[19] E. Tohidi, A. Bhrawy, K. Erfani, A collocation method based on Bernoulli operational matrix for numerical solution of generalized pantograph equation, Applied Mathematical Modelling 37 (6) (2013) 4283–4294.

[20] A. Bhrawy, E. Tohidi, F. Soleymani, A new Bernoulli matrix method for solving high-order linear and nonlinear Fredholm integro-differential equations with piecewise intervals, Applied Mathematics and Computation 219 (2) (2012) 482–497.

[21] E. Tohidi, M. Ezadkhah, S. Shateyi, Numerical solution of nonlinear fractional Volterra integro-differential equations via Bernoulli polynomials, Abstract and Applied Analysis 2014 (2014).

[22] F. Toutounian, E. Tohidi, S. Shateyi, A collocation method based on the Bernoulli operational matrix for solving high-order linear complex differential equations in a rectangular domain, Abstract and Applied Analysis 2013 (2013).

[23] E. Tohidi, F. Toutounian, Convergence analysis of Bernoulli matrix approach for one-dimensional matrix hyperbolic equations of the first order, Computers & Mathematics with Applications 68 (1-2) (2014) 1–12.

[24] E. Tohidi, M. K. Zak, A new matrix approach for solving second-order linear matrix partial differential equations, Mediterranean Journal of Mathematics 13 (3) (2016) 1353–1376.

[25] F. Toutounian, E. Tohidi, A new Bernoulli matrix method for solving second order linear partial differential equations with the convergence analysis, Applied Mathematics and Computation 223 (2013) 298–310.

[26] O. Kouba, Lecture Notes, Bernoulli Polynomials and Applications, arXiv preprint arXiv:1309.7560 (2013).

[27] F. Costabile, F. Dell'Accio, Expansion over a rectangle of real functions in Bernoulli polynomials and applications, BIT Numerical Mathematics 41 (3) (2001) 451–464.

[28] F. Costabile, F. Dell'Accio, Expansions over a simplex of real functions by means of Bernoulli polynomials, Numerical Algorithms 28 (1-4) (2001) 63–86.

[29] E. D. Rainville, Special functions, Vol. 442, New York, 1960.

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

[31] J. Higham, F. Tisseur, A block algorithm for matrix 1-norm estimation, with an application to 1-norm pseudospectra, SIAM J. Matrix Anal. Appl. 21 (2000) 1185–1201.

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

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

[34] NVIDIA, cuBLAS (2020).
URL `https://docs.nvidia.com/cuda/cublas`

[35] NVIDIA, CUDA Toolkit Documentation v11.0.3 (2020).
URL `https://docs.nvidia.com/cuda`

[36] P. Alonso, J. Peinado, J. Ibáñez, J. Sastre, E. Defez, Computing matrix trigonometric functions with GPUs through Matlab, The Journal of Supercomputing 75 (3) (2019) 1227–1240.