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

In this paper, a new method to approximate the exponential of a matrix, based on Bernoulli matrix polynomials, is presented. 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, GPUs-based implementations have been carried out, achieving a great computational advantage.

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 and to the difficulties involved in its computation, see [1–4].

Among the proposed methods for the approximate computation of the exponential matrix, two fundamental ones stand out: those based on rational Padé approximations [5–8], and those based on polynomial approximations, which are either Taylor series developments [9–12] or serial developments of Hermite matrix polynomials [13, 14]. In general, polynomial approximations showed to 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

*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)

basic property of *Scaling-Squaring*, based on the relationship

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

Thus, if $P_m(X)$ 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 [15, p. 63], the Taylor series expansion of the trigonometric functions $\tan(x), \csc(x)$ and $\cot(x)$ [15, p. 116-117] and the Taylor series expansion of the hyperbolic function $\tanh(x)$, [15, p. 125]. They are also employed in the well known exact expression of the even values of the Riemann zeta function:

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

Morever, they are even used for solving initial value problems [16], boundary value problems [17, 18], high-order linear and nonlinear Fredholm and Volterra integro-differential equations [19, 20], complex differential equations [21] and partial differential equations [22–24]. An excellent survey about Bernoulli polynomials and its applications can be found in [25]. The development of series functions of Bernoulli polynomials has been studied in [26, 27].

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 algorithm 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 $P_m(B) =$

$a_m B^m + a_{m-1}B^{m-1} + \cdots + a_1 B + a_0 I$. As usual, the matrix norm $\|\cdots\|$ denotes any subordinate matrix norm; in particular $\|\cdots\|_1$ 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 [28] 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 [15, 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

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

where the Bernoulli numbers are defined by $B_n = B_n(0)$. Therefore, it follows that the Bernoulli numbers satisfy

$$B_0 = 1, \ B_k = -\sum_{i=0}^{k-1}\binom{k}{i}\frac{B_i}{k+1-i}, k \geq 1. \tag{5}$$

Note that $B_3 = B_5 = \cdots = B_{2k+1} = 0$, 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

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

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 (2) and (6) one gets

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

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

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

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

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

$$e^{A2^{-s}} \approx P_m(A2^{-s}) = (e-1)\sum_{n=0}^{m}\frac{B_n\left(A2^{-s}\right)}{n!}. \tag{8}$$

Approximation (8) has the problem that it is not expressed in explicit terms of powers of the matrix $A2^{-s}$. We are going to find that explicit relationship. From (8) we have that, for $m=2$ and using (6), one gets

$$\begin{aligned}
&\frac{1}{(e-1)}e^{A2^{-s}}\\
\approx\ & \sum_{n=0}^{2}\frac{B_n\left(A2^{-s}\right)}{n!}\\
=\ & \frac{1}{0!}B_0\left(A2^{-s}\right)+\frac{1}{1!}B_1\left(A2^{-s}\right)+\frac{1}{2!}B_2\left(A2^{-s}\right)\\
=\ & \frac{1}{0!}\left(\sum_{k=0}^{0}\binom{0}{k}B_k\left(A2^{-s}\right)^{0-k}\right)+\frac{1}{1!}\left(\sum_{k=0}^{1}\binom{1}{k}B_k\left(A2^{-s}\right)^{1-k}\right)\\
+\ & \frac{1}{2!}\left(\sum_{k=0}^{2}\binom{2}{k}B_k\left(A2^{-s}\right)^{2-k}\right)\\
=\ & \left(\sum_{k=0}^{2}\frac{1}{k!}\binom{k}{k}B_k\right)\left(A2^{-s}\right)^0+\left(\sum_{k=1}^{2}\frac{1}{k!}\binom{k}{k-1}B_{k-1}\right)\left(A2^{-s}\right)^1\\
+\ & \left(\sum_{k=2}^{2}\frac{1}{k!}\binom{k}{k-2}B_{k-2}\right)\left(A2^{-s}\right)^2.
\end{aligned}$$

Similarly, for $m=3$:

$$\begin{aligned}
&\frac{1}{(e-1)}e^{A2^{-s}}\\
\approx\ & \sum_{n=0}^{3}\frac{B_n\left(A2^{-s}\right)}{n!}
\end{aligned}$$

$$
\begin{aligned}
&= \frac{1}{0!}B_0\left(A2^{-s}\right)+\frac{1}{1!}B_1\left(A2^{-s}\right)+\frac{1}{2!}B_2\left(A2^{-s}\right)+\frac{1}{3!}B_3\left(A2^{-s}\right)\\
&= \frac{1}{0!}\left(\sum_{k=0}^{0}\binom{0}{k}B_k\left(A2^{-s}\right)^{0-k}\right)+\frac{1}{1!}\left(\sum_{k=0}^{1}\binom{1}{k}B_k\left(A2^{-s}\right)^{1-k}\right)\\
&+ \frac{1}{2!}\left(\sum_{k=0}^{2}\binom{2}{k}B_k\left(A2^{-s}\right)^{2-k}\right)+\frac{1}{3!}\left(\sum_{k=0}^{3}\binom{3}{k}B_k\left(A2^{-s}\right)^{3-k}\right)\\
&= \left(\sum_{k=0}^{3}\frac{1}{k!}\binom{k}{k}B_k\right)\left(A2^{-s}\right)^0+\left(\sum_{k=1}^{3}\frac{1}{k!}\binom{k}{k-1}B_{k-1}\right)\left(A2^{-s}\right)^1\\
&+ \left(\sum_{k=2}^{3}\frac{1}{k!}\binom{k}{k-2}B_{k-2}\right)\left(A2^{-s}\right)^2+\left(\sum_{k=3}^{3}\frac{1}{k!}\binom{k}{k-3}B_{k-3}\right)\left(A2^{-s}\right)^3.
\end{aligned}
$$

Taking into account the particular cases $m=2$ and $m=3$, it seems that it is satisfied that

$$
\frac{1}{(e-1)}e^{A2^{-s}}
$$

$$
\begin{aligned}
&\approx \sum_{n=0}^{m}\frac{B_n\left(A2^{-s}\right)}{n!}\\
&= \left(\sum_{k=0}^{m}\frac{1}{k!}\binom{k}{k}B_k\right)\left(A2^{-s}\right)^0+\left(\sum_{k=1}^{m}\frac{1}{k!}\binom{k}{k-1}B_{k-1}\right)\left(A2^{-s}\right)^1+\cdots\\
&\cdots + \left(\sum_{k=m}^{m}\frac{1}{k!}\binom{k}{k-m}B_{k-m}\right)\left(A2^{-s}\right)^m. \qquad (9)
\end{aligned}
$$

Let's test the result using induction. Obviously the formula (9) is valid for $m=0,1,2,3$. Suppose it is valid for m and let's see for $m+1$. We have

$$
\begin{aligned}
&\frac{1}{(e-1)}e^{A2^{-s}}\\
&\approx \sum_{n=0}^{m+1}\frac{B_n\left(A2^{-s}\right)}{n!}\\
&= \sum_{n=0}^{m}\frac{B_n\left(A2^{-s}\right)}{n!}+\frac{1}{(m+1)!}B_{m+1}\left(A2^{-s}\right)\\
&= \sum_{n=0}^{m}\frac{B_n\left(A2^{-s}\right)}{n!}+\frac{1}{(m+1)!}\left(\sum_{k=0}^{m+1}\binom{m+1}{k}B_k\left(A2^{-s}\right)^{m+1-k}\right)\\
&= \left(\sum_{k=0}^{m}\frac{1}{k!}\binom{k}{k}B_k\right)\left(A2^{-s}\right)^0+\cdots+\left(\sum_{k=m}^{m}\frac{1}{k!}\binom{k}{k-m}B_{k-m}\right)\left(A2^{-s}\right)^m
\end{aligned}
$$

$$
\begin{aligned}
+ \quad & \frac{1}{(m+1)!}\left(\sum_{k=0}^{m+1}\binom{m+1}{k}B_k\left(A2^{-s}\right)^{m+1-k}\right) \\
= \quad & \left(\sum_{k=0}^{m+1}\frac{1}{k!}\binom{k}{k}B_k\right)\left(A2^{-s}\right)^0+\left(\sum_{k=1}^{m+1}\frac{1}{k!}\binom{k}{k-1}B_{k-1}\right)\left(A2^{-s}\right)^1+\cdots \\
\cdots \quad & +\left(\sum_{k=m+1}^{m+1}\frac{1}{k!}\binom{k}{k-(m+1)}B_{k-(m+1)}\right)\left(A2^{-s}\right)^{m+1}.
\end{aligned}
$$

Summarizing we establish the following result:

Theorem 2.1. *Given expresion (8), we get*

$$
\frac{1}{(e-1)}e^{\left(A2^{-s}\right)} \approx \sum_{n=0}^{m}\frac{B_n\left(A2^{-s}\right)}{n!}=\sum_{i=0}^{m}\alpha_i\left(A2^{-s}\right)^i, \tag{10}
$$

where $\alpha_i=\sum_{k=i}^{m}\binom{k}{k-i}\frac{B_{k-i}}{k!}$.

3. The proposed algorithm

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. Algorithm 1 computes e^A by using Bernoulli matrix polynomials.

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

1: Choose adequate order $m_k \leqslant m_M$ and scaling parameter $s\in\mathbb{N}\cup\{0\}$ for the Bernouilli approximation with scaling.
2: Compute $B=P_{m_k}(A/2^s)$ using (8)
3: **for** $i=1:s$ **do**
4: $\quad B=B^2$
5: **end for**

As mentioned, in Step 1, the optimal order of the series expansion $m_k \leqslant m_M$ and the scaling parameter s are chosen. Matrix polynomial $P_m(2^sA)$ can be computed optimally in terms of matrix products using values for m in the set $m_k=\{2,4,6,9,12,16,20,25,30,\ldots\}$, $k=0,1,\ldots$, respectively, see [4, p. 72–74].

The Bernoulli polynomial coefficients $B_m(A)$, given in (6), differ significantly from those of the Taylor approximation when $m\in\{2,4,6,9,12,16,20\}$. However, they are practically identical for $m\in\{25,30\}$. This is because, as the degree of the Bernoulli polynomial increases, all its coefficients vary, approaching the corresponding Taylor ones.

More in detail, the differences in absolute value between the Bernoulli and Taylor series coefficients, when $m = 25$ or $m = 30$, are approximately equal or less than the unit roundoff in IEEE double precision $u = 2^{-54} \backsimeq 1.11 \times 10^{-16}$). For example, for $m = 25$ the difference between the coefficients of the terms of degree 25 has been approximately equal to 1.7431×10^{-16}. For the rest of the coefficients, they have been lower than 8.72751×10^{-17}, decreasing those values until reaching the lowest value for the coefficient of degree 0 (4.63073×10^{-26}). As expected, for $m = 30$, the differences are smaller than in the previous case, becoming equal to 2.70791×10^{-33} for the term of degree 0. It should be remembered that, in power series of a function, the most significant terms are the first ones.

The algorithm applied in step 1 to calculate m and s is the Algorithm 2 from [11]. In Step 2, we compute the matrix exponential approximation of the scaled matrix by using the modified Paterson–Stockmeyer's method proposed in [29, p. 1836-1837]:

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

Taking into account Table 4.1 from [4], the computational cost in terms of matrix products of (11) is k. Finally, in steps $3-5$, the approximation of e^A is recovered by using squaring matrix products.

4. Numerical experiments

In this section, we firstly compare `expmber`, the new MATLAB implementation developed in this paper based on Bernoulli approximation, with the functions `exptaynsv3` [11], that computes the matrix exponential using Taylor matrix polynomials, and `expm_new` [7] which implements a scaling squaring Padé algorithm to work out the mentioned matrix function.

Next, we compare `expmbertay`, the novel function that combines Taylor and Bernoulli approximations, with `expmber`, `exptaynsv3` and `expm_new`. In `expmbertay`, the matrix exponential is carried out by means of Taylor series, when m is less than or equal to 20, or Bernoulli series, when m is equal to 25 or 30.

All the computations to obtain the *"exact"* matrix exponential function were carried out thanks to MATLAB's Symbolic Math Toolbox with 256 digits of precision, using the `vpa` (variable-precision arithmetic) function and Algorithm 2.

Algorithm 2 Computes the *"exact"* matrix exponential $B = e^A$, where $A \in \mathbb{C}^{r\times r}$, by means of Taylor expansion using vpa MATLAB function with n digits of precision.

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

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

and

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

and

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

by using Algorithm 3, **then**
    return $B = B^{(j)}_{m_k}(n)$
2: **else**
    return error
3: **end if**

Algorithm 3 Computes $B^{(s)}_m(n) = e^A$, where $A \in \mathbb{C}^{r\times r}$, by Taylor expansion of order m and parameter scaling s with n digits of precision.

1: Compute $B^{(s)}_m(n) = P_m(A/2^s)$ using Taylor expansion of order m with n digits of precision
2: **for** $i = 1 : s$ **do**
3: $\quad B^{(s)}_m(n) = [B^{(s)}_m(n)]^2$
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) [30].
   - Sixteen matrices from the Eigtool MATLAB package (EMP) [31] 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 2).

     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 5, 10, 16, 17, 21, 25, 26, 42, 43, 44 and 49 belonging to the MCT and matrices 5 and 6 appertaining to the EMP were not taken into account since the exact exponential solution could not be computed. Besides, matrix 2 from the MCT and matrices 3 and 10 from the EMP were not 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 1 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` achieve 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 1: 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

Table 2: 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%

Table 2, 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

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

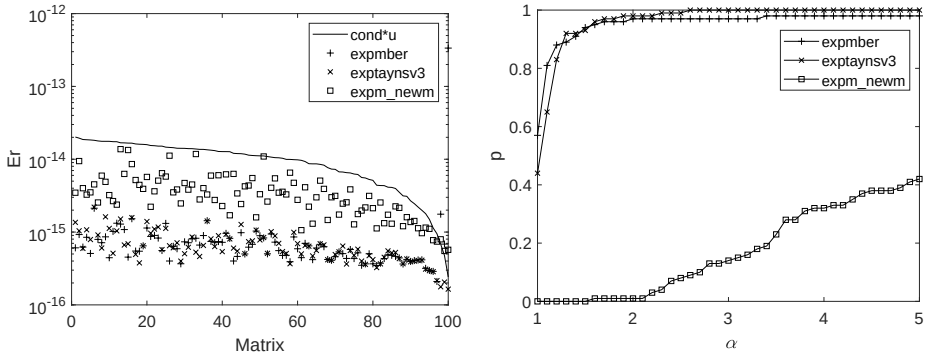
where $e\tilde{x}p(A)$ is the approximate solution and $\exp(A)$ is the exact one.

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.

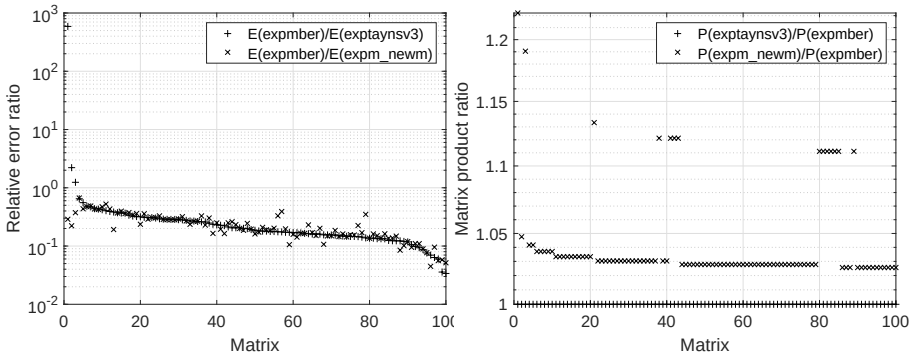
Regarding the normwise relative errors presented in Figures 1a, 2a and 3a, their solid line represents the function $k_{\exp}u$, where $k_{\exp}$ (or *cond*) is the condition number of the matrix exponential function [4, Chapter 3] and $u = 2^{-53}$ is the unit roundoff in the IEEE double precision floating-point arithmetic. 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



(a) Normwise relative errors.

(b) Performance profile.



(c) Ratio of relative errors.

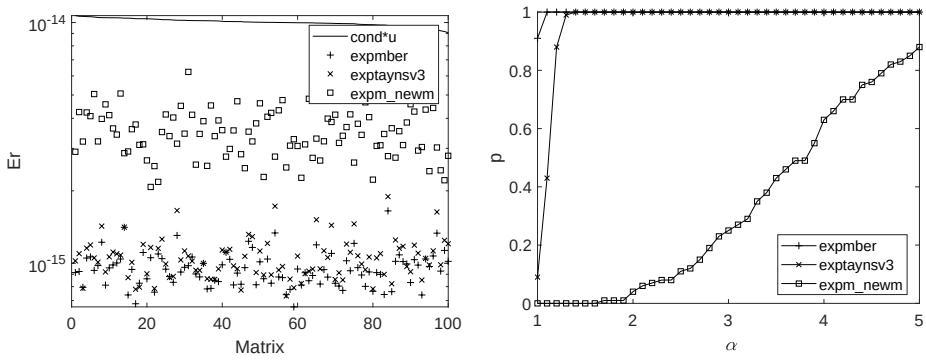
(d) Ratio of matrix products.

Figure 1: Experimental results for Test 1.

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. Nothwithstanding, Figure 3b reveals that `exptaynsv3` code improved the result accuracy against the `expmber` function, for the Test 3.

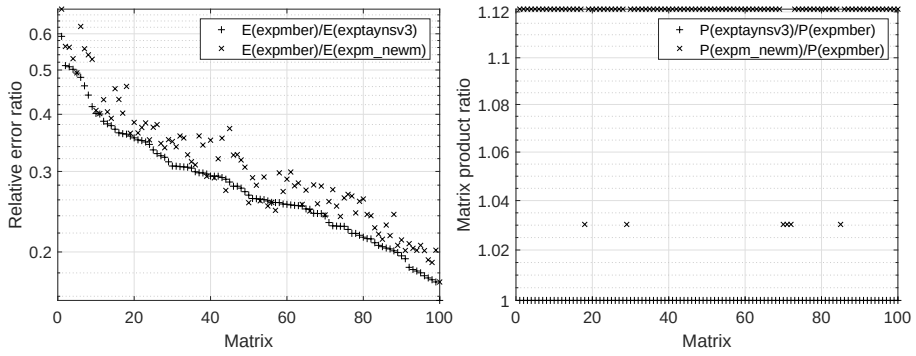
In Figures 1c, 2c and 3c, the ratios of relative errors have been presented in decreasing order with respect to E(`expmber`)/E(`exptaynsv3`). They confirm the data exposed in Table 2, where it was shown that `expmber` provides more accurate results than `exptaynsv3` for Tests 1 and 2, but not for Test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

(b) Performance profile.



(c) Ratio of relative errors.

(d) Ratio of matrix products.

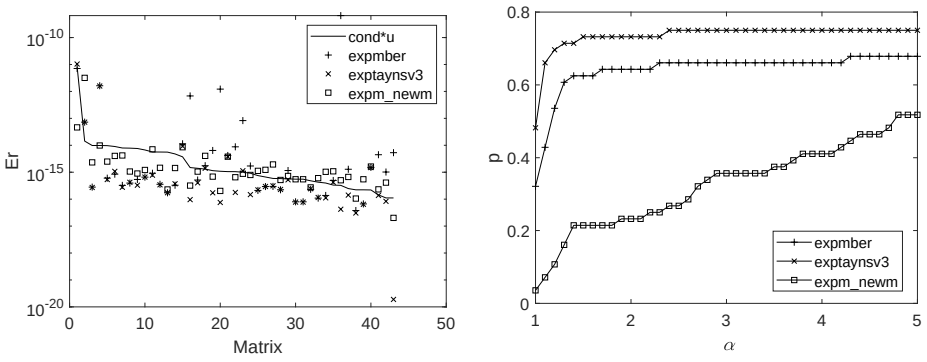
Figure 2: Experimental results for Test 2.

Table 3: Mininum, 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

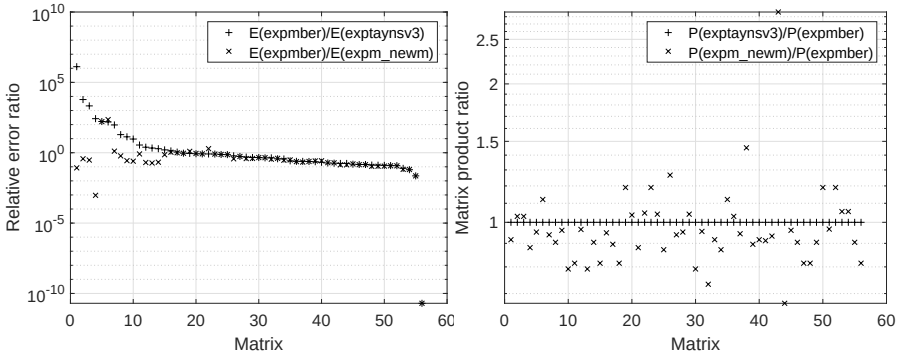
a very appropriate accuracy for values of m equal to 25 or 30. However, for significantly lower values, `expmber` will be less competitive than other codes, such as `exptaynsv3`. Minimum, maximum and average values of m required for Tests 1, 2 and 3 are collected in Table 3. In more detail, Figure 4 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 1, `expmber` and `exptaynsv3` functions performed a lower number of matrix operations than `expm_new` one. This statement



(a) Normwise relative errors.

(b) Performance profile.



(c) Ratio of relative errors.

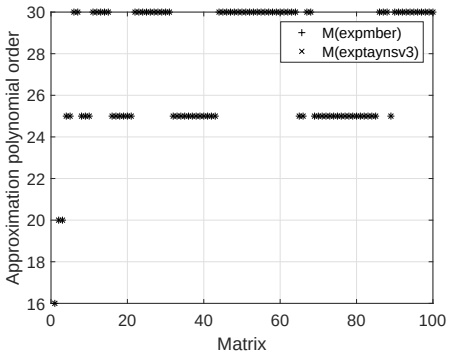
(d) Ratio of matrix products.

Figure 3: Experimental results for Test 3.

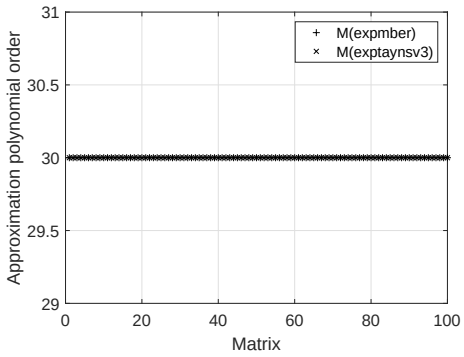
can be also corroborated from the results displayed in Figures 1d, 2d and 3d, where the ratio between the number of `expm_new` and `expmber` matrix products ranged from 1.03 to 1.22 for Test 1, from 1.03 to 1.12 for Test 2 and from 0.67 to 2.87 for Test 3.

Next, we will analyse the possibility 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 4 shows, the percentage of cases in which 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 2.

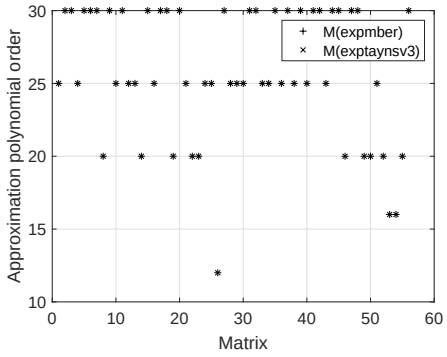
From these excellent results, we therefore considered the possibility of combining Bernoulli and Taylor methods, giving rise to the `expmbertay` code. In this new function, and according to the comparison between the coefficients of their polynomials carried out previously, we will use the Taylor approach (`exptaynsv3`) for values of m below 25 and the Bernoulli approximation



(a) Test 1.



(b) Test 2.



(c) Test 3.

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

Table 4: 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`)	3%	0%	10.71%
E(`exptaynsv3`)=E(`expm_new`)	0%	0%	0%

(`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 `exptaynsv3`.

Table 5 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 com-

Table 5: 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%

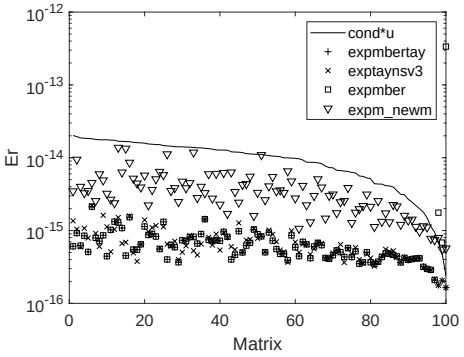
bined method is now better or equal than `exptaynsv3` in 55.36% of cases for Test 3. Moreover, `expmbertay` became better than `expm_new` in 100% of the matrices for Test 1 and 2, and 91.07% for Test 3, which is higher than the percentages individually offered by `expmber` (69.64%) and `exptaynsv3` (89.29%).

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.

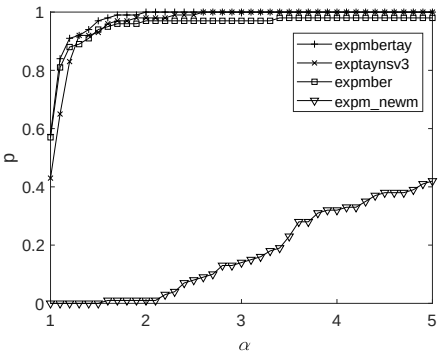
We have also included in the software developed an "accelerated" version of the algorithm proposed in this paper to compute the matrix exponential on an NVIDIA GPU. Matrix multiplication is an operation very rich in intrinsic parallelism that can be optimized for GPUs. Algorithms, like the one proposed, that rely on many matrix multiplications can be highly get the most of this devices through the use of the cuBLAS [32] package.

Experimental results, shown in Fig. 8, 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 CPU version execution shown in the figure was obtained using all the 40 cores available in the target computer. 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.

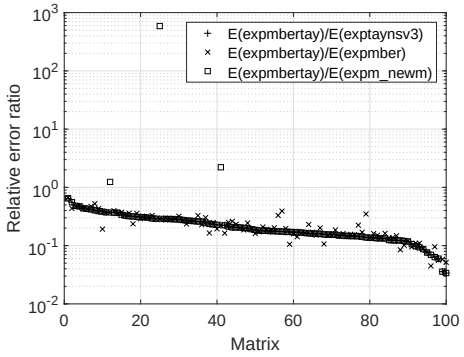
At the light of Figure 8, it can be concluded that both the former algorithm based on Taylor series (`exptaynsv3`) and the one based on Bernoulli series (`expmber`), studied in this paper, very similarly regarding the execution time, also on the two target architectures tackled: the host CPU and one of the most currently advanced GPUs. The reduction in time obtained with the GPU regarding the CPU starts approximately with matrices of size $n = 1000$. 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` will be very similar to `expmber`, since it uses once again the same number of matrix products.



(a) Normwise relative errors.



(b) Performance profile.



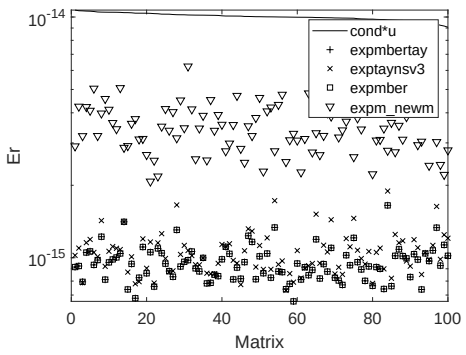
(c) Ratio of relative errors.

Figure 5: Experimental results for Test 1.

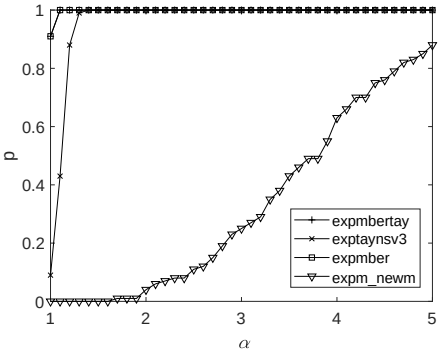
5. Conclusions

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

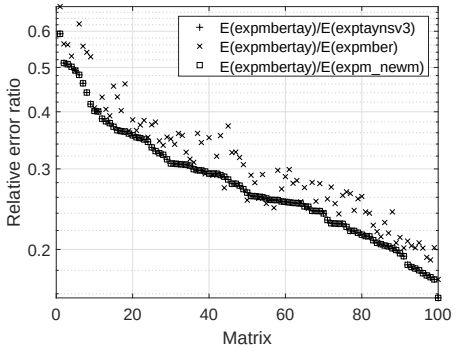
Preliminary results with the Bernoulli version for the exponential matrix function motivated us to develop a hybrid algorithm (called `expmbertay`) that combines the best of both the Taylor and Bernoulli solutions and resulting in excellent results. Finally, we showed that the two algorithms developed in this



(a) Normwise relative errors.



(b) Performance profile.



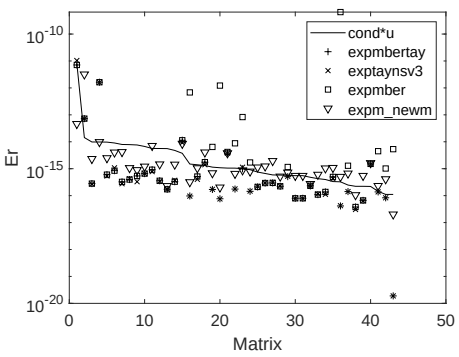
(c) Ratio of relative errors.

Figure 6: Experimental results for Test 2.

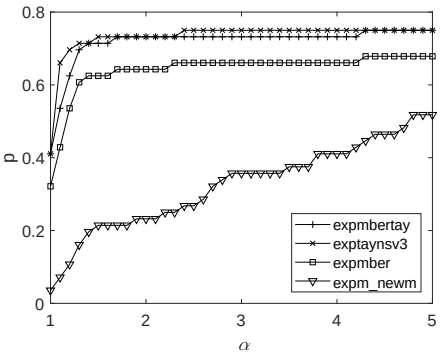
contribution keep the advantages of other ones based on matrix polynomial expansion. Since they all are based on matrix multiplications, the GPU version implemented has turned out to be a strong tool to compute the matrix exponential approximation when the numerical methods employed are stressed with large dimension matrices.

Acknowledgements

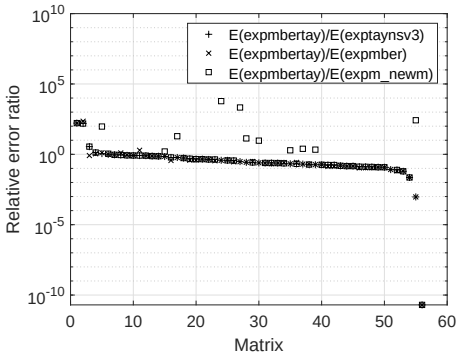
This work has been partially supported by Spanish Ministerio de Economía y Competitividad and European Regional Development Fund (ERDF) grants TIN2017-89314-P and by the Programa de Apoyo a la Investigación y Desarrollo 2018 of the Universitat Politècnica de València (PAID-06-18) grants SP20180016.



(a) Normwise relative errors.



(b) Performance profile.



(c) Ratio of relative errors.

Figure 7: Experimental results for Test 3.

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] G. A. Baker, P. R. Graves-Morris, Padé Approximants, encyclopaedia of mathematics and its applications Edition, Cambridge University Press, 1996.

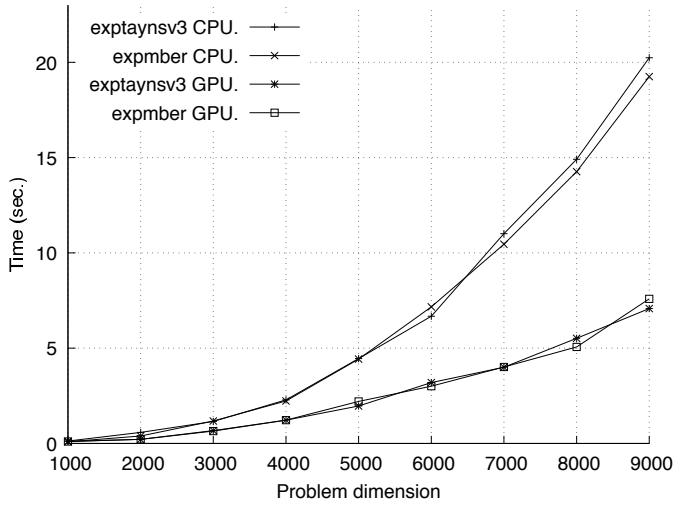


Figure 8: Execution time (sec.) of the algorithm to compute the matrix exponential using the Taylor series (`exptaynsv3`) and the Bernoulli (`expmber`) series on CPU and on GPU for large diagonalizable matrices.

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

[7] 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.

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

[9] 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.

[10] J. Sastre, J. J. Ibáñez, E. Defez, P. A. Ruiz, Efficient scaling-squaring Taylor method for computing matrix exponential, SIAM J. on Scientific Comput.Accepted with modifications.

[11] 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.

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

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

[14] 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.

[15] 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.

[16] 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.

[17] 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.

[18] 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.

[19] 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.

[20] E. Tohidi, M. Ezadkhah, S. Shateyi, Numerical solution of nonlinear fractional volterra integro-differential equations via Bernoulli polynomials, in: Abstract and Applied Analysis, Vol. 2014, Hindawi, 2014.

[21] 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, in: Abstract and Applied Analysis, Vol. 2013, Hindawi, 2013.

[22] 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.

[23] 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.

[24] 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.

[25] O. Kouba, Lecture Notes, Bernoulli Polynomials and Applications, arXiv preprint arXiv:1309.7560.

[26] 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.

[27] 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.

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

[29] J. Sastre, J. J. Ibáñez, E. Defez, P. A. Ruiz, Accurate matrix exponential computation to solve coupled differential models in engineering, Math. Comput. Model. 54 (2011) 1835–1840.

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

[31] T. Wright, Eigtool, version 2.1.
URL `web.comlab.ox.ac.uk/pseudospectra/eigtool.`

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

[33] E. Defez, J. Ibáñez, J. Peinado, J. Sastre, P. Alonso-Jordá, An efficient and accurate algorithm for computing the matrix cosine based on new Hermite approximations, Journal of Computational and Applied Mathematics 348 (2019) 1–13.