Journal of Computational and Applied Mathematics 425 (2023) 115074

Contents lists available at ScienceDirect

Journal of Computational and Applied Mathematics

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

Euler polynomials for the matrix exponential approximation

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

[a] Instituto de Instrumentación para Imagen Molecular, Universitat Politècnica de València, Camino de Vera s/n, 46022, Valencia, Spain
[b] Instituto de Matemática Multidisciplinar, Universitat Politècnica de València, Camino de Vera s/n, 46022, Valencia, Spain
[c] Department of Computer Systems and Computation, Universitat Politècnica de València, Camino de Vera s/n, 46022, Valencia, Spain

ARTICLE INFO

Article history:
Received 22 September 2022
Received in revised form 3 January 2023

Keywords:
Matrix functions
Matrix exponential
Euler polynomials

ABSTRACT

In this work, a new method to compute the matrix exponential function by using an approximation based on Euler polynomials is proposed. These polynomials are used in combination with the scaling and squaring technique, considering an absolute forward-type theoretical error. Its numerical and computational properties have been evaluated and compared with the most current and competitive codes dedicated to the computation of the matrix exponential. Under a heterogeneous test battery and a set of exhaustive experiments, it has been demonstrated that the new method offers performance in terms of accuracy and stability which is as good as or even better than those of the considered methods, with an intermediate computational cost among all of them. All of the above makes this a very competitive alternative that should be considered in the growing list of available numerical methods and implementations dedicated to the approximation of the matrix exponential.

1. Introduction

Due to its numerous and varied applications in different areas of science and technology, the calculus of matrix functions has been the focus of many researchers in recent years. Among all matrix functions, the matrix exponential e^A, where $A \in \mathbb{C}^{r\times r}$, stands out for its well-known application in solving systems of linear differential equations of first order with constant coefficients, as well as the difficulties involved in its accurate calculation and approximation, which has motivated to the development of papers that today could be considered classics (see the Refs. [1,2]).

Furthermore, the matrix exponential has applicability in other science areas. Numerous and different types of *exponential integrators* are employed to solve diverse and varied problems, see for example [3] or [4]. In graph theory, the matrix exponential, and other matrix functions [5], is used in the study of the connectivity of graphs. Precisely, the *Estrada index* is defined as the trace of the matrix e^A, where A is the adjacency matrix associated with the graph, see for example [6]. Among the recent works that require the computation of the matrix exponential, we can mention the Refs. [7–9], among many others.

Among the different methods proposed for the approximate calculation of the matrix exponential, we can highlight those belonging to two large families, i.e., those based on rational approximations, such as Padé approximants, see [10–14] for example, and those based on polynomial approximations, either those related to Taylor polynomials [15–19] or

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

https://doi.org/10.1016/j.cam.2023.115074

to other types such as Hermite [20,21] or Bernoulli [22] polynomials. From the numerical experiments performed, it can be concluded that, in general, polynomial approximations offer higher accuracy than rational ones, although maybe requiring a higher computational cost. On the other hand, an alternative approach to those ones just described for computing matrix exponentials based on Vandermonde matrices is detailed in [23].

Most current methods use the scaling and squaring technique [12,13], consisting in the relationship

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

This technique scales the matrix A by 2^s to properly reduce its norm, being s the scaling parameter to be determined. Next, once the approximation to the matrix exponential has been computed by means of rational or polynomial approaches, successive s squaring steps must be applied to reverse the scaling effect.

Let $P_m(A)$ be a polynomial of order m to be considered as an approximation to e^A, after this degree m had been properly chosen. Then, given the scaling factor s, from (1) one gets that $P_m(A/2^s)$ is an approximation to $e^{A/2^s}$ and

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

Throughout this paper, we will denote by $\mathbb{C}^{n\times n}$ the set of all the complex square matrices of size n. We will represent as I_n (or I if there is no possible confusion) the identity matrix in $\mathbb{C}^{n\times n}$. A matrix polynomial $P_m(A)$ of degree m, where $A \in \mathbb{C}^{n\times n}$ and p_j, for $0 \le j \le m$, are complex numbers, is defined as

$$P_m(A) = p_m A^m + p_{m-1}A^{m-1} + \cdots + p_1 A + p_0 I.$$

Traditionally, the cost of computing matrix functions by means of polynomial approximations, as is the case of the exponential function according to expression (2), is expressed by means of matrix products. Thus, the number of such multiplications to be performed and their efficient execution becomes a key aspect that determines the effectiveness of any new proposed algorithm, especially when comparing its computational performance with those ones of the methods already present in the literature. A good survey of algorithms devoted to compute matrix products with a complexity lower than $\mathcal{O}(n^3)$ is collected in [24].

As usual, the matrix norm $\|\cdot\|$ stands for any subordinate matrix norm. As an example, $\|\cdot\|_1$ is the well-known 1−norm. Finally, if $\mathcal{A}(k, m)$ are matrices in $\mathbb{C}^{n\times n}$ for $m \ge 0$, $k \ge 0$, from [25] it follows that

$$\sum_{m\ge 0}\sum_{k\ge 0}\mathcal{A}(k,m) = \sum_{m\ge 0}\sum_{k=0}^{m}\mathcal{A}(k,m-k). \tag{3}$$

The paper is organized as follows: In Section 2, a new series development of the matrix exponential in terms of the Euler matrix polynomials is presented. Next, in Section 3, the algorithms designed to approximate the matrix exponential by means of this kind of polynomials are exposed. Note that an absolute forward-type theoretical error analysis has been considered. Section 4 includes a comprehensive numerical and computational comparison of the code derived from the new numerical method with respect to the most relevant codes related to the computation of the matrix exponential. Finally, conclusions are given in Section 5.

2. Euler matrix polynomials

The Euler polynomials $E_n(x)$ are defined as the coefficients of the generating function

$$\frac{2e^{xt}}{e^t+1} = \sum_{n\ge 0}\frac{E_n(x)}{n!}t^n, \quad |t| < \pi, \tag{4}$$

see formula 24.2.6 from [26, p. 588]. These polynomials can be calculated either by using the Bernoulli polynomials $B_n(x)$ [27] through expression

$$E_n(x) = \frac{2}{n+1}\left(B_{n+1}(x) - 2^{n+1}B_{n+1}\left(\frac{x}{2}\right)\right), \tag{5}$$

or by means of the explicit expression

$$E_n(x) = \sum_{k=0}^{n}\binom{n}{k}\frac{\mathcal{E}_k}{2^k}\left(x-\frac{1}{2}\right)^{n-k}, \tag{6}$$

where $\mathcal{E}_k$ are Euler numbers of the form $\mathcal{E}_n = 2^n E_n(1/2)$. The Euler numbers are given by the coefficients of

$$\frac{2e^t}{e^{2t}+1} = \frac{2}{e^t+e^{-t}} = \sum_{n\ge 0}\frac{\mathcal{E}_n}{n!}t^n, \quad |t| < \frac{\pi}{2}, \tag{7}$$

and satisfy both $\mathcal{E}_{2n+1} = 0, n \geq 0$ and

$$\mathcal{E}_{2n} = 1 - \sum_{k=1}^{n} \binom{2n}{2k-1} \frac{2^{2k}(2^{2k}-1)}{2k} \mathcal{B}_{2k}, n \geq 0,$$

where $\mathcal{B}_{2k}$ is the 2kth Bernoulli number. These Bernoulli numbers $\mathcal{B}_n$ are defined by expressions (with the exception of $\mathcal{B}_1$, all other odd Bernoulli numbers are null)

$$\mathcal{B}_0 = 1, \mathcal{B}_1 = -\frac{1}{2}, \mathcal{B}_n = -\sum_{k=0}^{n-1} \binom{n}{k} \frac{\mathcal{B}_k}{n+1-k}, n \geq 2. \tag{8}$$

Euler numbers appear, for example, in the MacLaurin series expansion of the secant and hyperbolic secant functions given, respectively, by (formula (4.19.5) from [26])

$$\begin{aligned} \sec(t) &= \sum_{n\geq 0} \frac{(-1)^n \mathcal{E}_n}{(2n)!} t^{2n} \ , \ |t| < \frac{\pi}{2} \ , \\ \operatorname{sech}(t) &= \sum_{n\geq 0} \frac{\mathcal{E}_n}{(2n)!} t^{2n} \ , \ |t| < \frac{\pi}{2}. \end{aligned}$$

Some interesting properties and relations about Euler and Bernoulli polynomials can be found in [28,29]. A recent application of the Euler polynomials to the study and forecast of air pollutants and to mechanical engineering can be found in Refs. [30,31], respectively.

Computing $\dfrac{2}{e^t+1} e^{At}$, where $A \in \mathbb{C}^{n\times n}$, by using (3) and (7), we find that:

$$\begin{aligned} \left(\frac{2}{e^t+1}\right) e^{At} &= e^{t(A-I/2)} \left(\frac{2e^{\frac{t}{2}}}{e^t+1}\right) \\ &= \left(\sum_{n\geq 0} \frac{(A-I/2)^n}{n!} t^n\right) \left(\sum_{k\geq 0} \frac{\mathcal{E}_k}{2^k k!} t^k\right) \\ &= \sum_{n\geq 0} \sum_{k\geq 0} \frac{(A-I/2)^n}{n!} t^n \frac{\mathcal{E}_k}{2^k k!} t^k \\ &= \sum_{n\geq 0} \sum_{k=0}^{n} \frac{(A-I/2)^{(n-k)}}{(n-k)!} t^{(n-k)} \frac{\mathcal{E}_k}{2^k k!} t^k \\ &= \sum_{n\geq 0} \left(\sum_{k=0}^{n} \binom{n}{k} \frac{\mathcal{E}_k}{2^k} (A-I/2)^{n-k}\right) \frac{t^n}{n!} \\ &= \sum_{n\geq 0} \frac{E_n(A) t^n}{n!}, \end{aligned} \tag{9}$$

where $E_n(A)$ is the nth Euler matrix polynomial (6) evaluated on matrix A. Thus, we can use expression (9) in order to obtain the approximations to the matrix exponential as

$$e^{At} = \frac{e^t+1}{2} \sum_{n\geq 0} \frac{E_n(A) t^n}{n!}, |t| < \pi. \tag{10}$$

Moreover, if we take $t = 1$ and s represents the scaling factor of matrix A, then we will use the following expression as an approximation to $e^{A/2^s}$:

$$e^{A/2^s} \approx P_m(A/2^s) = \frac{e+1}{2} \sum_{n=0}^{m} \frac{E_n(A/2^s)}{n!}, \tag{11}$$

In order to employ this formula in an effective way for the computation of the matrix exponential, it is necessary to determine, for each matrix A, the scaling factor s and the degree m of the approximation.

3. Algorithms

3.1. Algorithm based on the Euler series of the matrix exponential

Taking $t = 1$ from (10) again, polynomial $P_m(A)$ can be expressed as an Euler approximation of order m to the matrix exponential e^A in the way

$$e^A \approx P_m(A) = \frac{e+1}{2}\sum_{n=0}^{m}\frac{E_n(A)}{n!} = \sum_{i=0}^{m} p_i^{(m)} A^i, \tag{12}$$

where the coefficients $p_i^{(m)}$ depend on the order m of the approximation. These coefficients are more and more similar to the Taylor series coefficients as the polynomial degree m increases.

One of the most used procedures to efficiently compute the matrix polynomial $P_m(A)$ is the Paterson–Stockmeyer method [32]. In this method, the most efficient polynomial orders m belong to the following set:

$$\mathbb{M} = \{1, 2, 4, 6, 9, 12, 16, 20, 25, 30, 36, 42, 49, 56, 64, 72, \dots\}.$$

If we denote with m_k the kth element of $\mathbb{M}$, the powers A^i, $2 \le i \le q$, must be calculated, with $q = \lceil\sqrt{m_k}\rceil$ or $q = \lfloor\sqrt{m_k}\rfloor$ being an integer divisor of m_k. With these matrix powers A^i, we can efficiently compute $P_{m_k}(A)$ with a computational cost of the k matrix products as

$$\begin{aligned}
&P_{m_k}(A) = \\
&(((p_{m_k}A^q + p_{m_k-1}A^{q-1} + p_{m_k-2}A^{q-2} + \cdots + p_{m_k-q+1}A + p_{m_k-q}I)A^q \\
&\qquad + p_{m_k-q-1}A^{q-1} + p_{m_k-q-2}A^{q-2} + \cdots + p_{m_k-2q+1}A + p_{m_k-2q}I)A^q \\
&\qquad + p_{m_k-2q-1}A^{q-1} + p_{m_k-2q-2}A^{q-2} + \cdots + p_{m_k-3q+1}A + p_{m_k-3q}I)A^q \\
&\qquad \dots \\
&\qquad + p_{q-1}A^{q-1} + p_{q-2}A^{q-2} + \cdots + p_1A + p_0I.
\end{aligned} \tag{13}$$

The reason for such a choice of polynomial orders in the above set $\mathbb{M}$ is that the number of matrix products required to evaluate polynomials of degree m with $m_k < m \le m_{k+1}$ is the same. However, the approximation of degree m_{k+1} will be theoretically more accurate than the ones of the other lower degrees.

Absolute forward error will be considered for the determination of the order $m \in \mathbb{M}$ of the Euler approximation and the scaling factor s, as exposed next. Let $m \in \mathbb{M}$ be a sufficiently large value so that the coefficients $p_i^{(m)}$ of the Euler approximation $P_m(A)$ from (12) to e^A are almost equal to Taylor approximation coefficients of the same order m. Then, the absolute forward error when computing $P_m(A)$ can be bounded as

$$E_{af}(P_m(A)) = \left\|e^A - P_m(A)\right\| \approx \left\|\sum_{i>m} a_i A^i\right\|,$$

where $\sum_{i>m} a_i A^i$ is the absolute forward error of the Taylor approximation of order m to the matrix exponential e^A.

Let $h_m(x) = \sum_{i>m} a_i x^i$ and $\tilde{h}_m(x) = \sum_{i>m} |a_i|\, x^i$. If Theorem 1.1 from [33] is applied, then

$$E_{af}(P_m(A)) \approx \|h_m(A)\| \le \tilde{h}_m(\alpha_m),$$

where

$$\alpha_m = \max\left\{\left\|A^i\right\|^{1/i} : i = m+1, m+2, \dots, 2m+1\right\}.$$

Let Θ_m be

$$\Theta_m = \max\left\{\theta \ge 0 : \sum_{i>m} |a_i|\,\theta^i \le u\right\}, \tag{14}$$

where $u = 2^{-53}$ is the unit roundoff in IEEE double precision arithmetic. Table 1 shows for each $m \in \mathbb{M}$ the corresponding value Θ_m.

If it is satisfies that

$$\alpha_m < \Theta_m, \tag{15}$$

then, we have

$$E_{af}(P_m(A)) \approx \|h_m(A)\| \leqslant \tilde{h}_m(\alpha_m) \leqslant \tilde{h}_m(\Theta_m) \leqslant u. \tag{16}$$

and the scaling parameter s will be 0.

Table 1
Values Θ_m.

m	Θ_m	m	Θ_m
1	$1.4901161156840223 \cdot 10^{-8}$	25	$2.5585766884181380 \cdot 10^{0}$
2	$8.7334702258487179 \cdot 10^{-6}$	30	$3.7810696269831392 \cdot 10^{0}$
4	$1.6783942982781048 \cdot 10^{-3}$	36	$5.4064650937902918 \cdot 10^{0}$
6	$1.7764527083684662 \cdot 10^{-2}$	42	$7.1556200904384877 \cdot 10^{0}$
9	$1.1483174747739708 \cdot 10^{-1}$	49	$9.3073843996022152 \cdot 10^{0}$
12	$3.3521368782861483 \cdot 10^{-1}$	56	$1.15453483152121912 \cdot 10^{1}$
16	$8.2460319163860885 \cdot 10^{-1}$	64	$1.41791073371113185 \cdot 10^{1}$
20	$1.5041473223951629 \cdot 10^{0}$	72	$1.68723620122422204 \cdot 10^{1}$

Algorithm 1: Given a matrix $A \in \mathbb{C}^{n\times n}$, a minimum order $m_{lower} \in \mathbb{M}$, and a maximum order $m_{upper} \in \mathbb{M}$, this algorithm computes $E \approx e^A$ by an Euler approximation (12) of order m, $m_{lower} \leq m \leq m_{upper}$.

```
1 Obtain the order $m$, $m_{lower} \leq m \leq m_{upper}$, and the scaling parameter $s \in \mathbb{N} \cup \{0\}$ by using Algorithm 2;
2 Evaluate the matrix polynomial $E = P_m(A/2^s)$ by Formula (13);
3 for $i = 1 : s$ do
4     $E = E^2$;
5 end
```

However, if (15) is not fulfilled, then the smallest value s such that $2^{-s}\alpha_m < \Theta_m$ must be determined. In this case, we obtain

$$E_{af}(P_m(A/2^s)) \approx \left\| h_m(A/2^s) \right\| \leqslant \tilde{h}_m(2^{-s}\alpha_m) \leqslant \tilde{h}_m(\Theta_m) \leqslant u. \tag{17}$$

Further information about how computing forward and backward errors is available at [34].

Algorithm 1 calculates the approximation to the matrix exponential, where the scaling and squaring technique and the Paterson–Stockmeyer method have been considered. In Line 1, the order m of Euler approximation and the scaling parameter s are calculated. For that, Algorithm 2 is run. In Line 2, the Paterson–Stockmeyer method is used for evaluating $P_m(A/2^s)$. Finally, in Lines 3–5, e^A is recovered by using Formula (1).

3.2. Algorithm for providing the polynomial order m and the factor scaling s

Algorithm 2 is responsible for supplying the appropriate values of the order m of the approximation polynomial $P_m(A)$ to the matrix exponential and the scaling factor s. The algorithm also returns all the powers of matrix A needed to efficiently evaluate the mentioned polynomial by the Paterson–Stockmeyer method. From Lines 1 to 12, the algorithm tries to find the minimum degree $m \in [m_{lower}, m_{upper}]$ of the polynomial for which expression (15) is satisfied. Meanwhile, the necessary powers of A are also calculated. If this condition is satisfied for any of the allowed values of m, the scaling value s will be equal to 0, as indicated in Line 13. Otherwise, Lines 15 and 16 will determine that the order m of the polynomial will be equal to the maximum value allowed and the scaling factor s will be set after evaluating the expression $s = \log_2(\alpha_m/\theta_m)$.

4. Numerical experiments

Under the premise of evaluating and comparing the performance of the proposed algorithms in the computation of the matrix exponential function, a series of numerical experiments have been carried out. In all of them, the following state-of-the-art codes were used:

- `expm_euler`: this is the MATLAB implementation of Algorithm 1. The selectable values of m as orders of the approximation polynomial $P_m(A)$ to the exponential for each matrix will belong to the set $\{42, 49, 56\}$. The code is available at the website http://personales.upv.es/joalab/software/expm_euler.m.
- `exptaynsv3`: it computes the matrix exponential using a scaling and squaring algorithm and Taylor polynomial approximation [16].
- `expm_mp`: it is based on a multiprecision algorithm that can employ diagonal Padé or Taylor approximants after the corresponding Schur decomposition or in transformation-free form [35]. This code requires the Advanpix Multiprecision Computing Toolbox [36]. In our case, all executions were performed using the Taylor-based transformation-free variant in double numeric precision (16 decimal digits).
- `exptayotf`: it estimates on-the-fly the backward error for the approximation of the matrix exponential and selects the scaling parameter s and the Taylor polynomial degree m in order to reach any desired accuracy [18]. Default values for the code input parameters were considered.

Algorithm 2: Given a matrix $A \in \mathbb{C}^{n\times n}$, a minimum order $m_{lower} \in \mathbb{M}$, and a maximum order $m_{upper} \in \mathbb{M}$, this algorithm provides an order m, $m_{lower} \le m \le m_{upper}$, a scaling factor s, and the necessary powers of A to compute e^A.

```
$A_1 = A$; $i = lower$; $f = 0$;
for $j = 2$ to $\lceil \sqrt{m_i} \rceil$ do
    $A_j = A_{j-1}A$;
end
while $f = 0$ and $i \le upper$ do
    $v = \sqrt{m_i}$; $j = \lceil v \rceil$;
    if $j > v$ then $A_j = A_{j-1}A$ ;
    Compute $a_i \approx \left\| A^{m_i+1} \right\|$ from $A_j$ and $A$;
    $\alpha_i = \sqrt[m_i+1]{a_i}$;
    if $\alpha_i < \Theta_i$ then $f = 1$ ;
    else $i = i + 1$ ;
end
if $f = 1$ then $s = 0$ ;
else
    $i = upper$;
    $s = \max(0, \lceil \log_2(\alpha_i/\Theta_i) \rceil)$;
end
$m = m_i$;
```

- expm3: it combines a scaling and squaring algorithm with a Taylor approximation which reduces the number of matrix multiplications, in comparison with the Paterson–Stockmeyer method, for polynomial evaluation. The code incorporates additional adjustments to avoid overscaling [19].
- expm: it is the MATLAB built-in function that computes the matrix exponential by using a scaling and squaring algorithm with a Padé approximation [13,37].
- sexpm: it is based on a variant of the scaling and squaring algorithm and subdiagonal Padé approximants of low degree, with the aim of reducing the overall cost and avoiding the potential instability caused by overscaling [14].

All the numerical experiments were launched on a Microsoft Windows 11 x64 PC equipped with an 11th Generation Intel Core i5-11500 @2.70 GHz processor and 16 GB of RAM, using MATLAB R2022a.

The following three groups of matrices were considered to form a useful testbed for comparing the performance of the different codes with each other. In the three mentioned groups, the matrix exponential function was "*exactly*" computed thanks to the MATLAB Symbolic Math Toolbox and the function vpa (variable-precision floating-point arithmetic) with 256 decimal digits:

- Group 1: one hundred diagonalizable complex matrices of size 128×128 whose 2-norm ranged from 0.1 to 300. All these matrices were created as $A = V \cdot D \cdot V^{-1}$, where V is an orthogonal matrix such as $V = H/\sqrt{128}$, with H being the Hadamard matrix of dimension 128, and D is a random diagonal matrix with complex eigenvalues. The "*exact*" matrix exponential was computed as $\exp(A) = V \cdot \exp(D) \cdot V^T$, in combination with the function vpa.
- Group 2: one hundred non-diagonalizable complex matrices of size 128×128 and with 2-norm varying from 3.76 to 337.72. They were generated as $A = V \cdot J \cdot V^{-1}$. Meanwhile V was again obtained in the same way as for the previous group, J is a Jordan matrix with complex eigenvalues whose modules are less than 5 and with a randomly determined algebraic multiplicity between 1 and 3. The matrix exponential was "*exactly*" computed as $\exp(A) = V \cdot \exp(J) \cdot V^{-1}$, together with the function vpa.
- Group 3: fifty-two matrices of dimension 128×128 belonging to the Matrix Computation Toolbox (MCT) [38] and twenty matrices of the same size coming from the Eigtool MATLAB Package (EMP) [39]. Of all these, forty-five matrices could be utilized in the corresponding numerical experiments, while the remaining twenty-seven matrices (sixteen from the MCT and eleven from the EMP) were excluded for these reasons:
  - Their "*exact*" solution could not be computed: matrices 4, 5, 10, 16, 17, 18, 21, 25, 26, 35, 40, 42, 43, 44, and 49 from the MCT, and matrices 5, and 6, 7, and 9 from the EMP.
  - The relative error produced by any of the above codes was excessively large, due to their ill-conditioning: matrix 2 from the MCT, and matrices 3, 4, 5 and 10 from the EMP.
  - They are repetitive (already present in MCT): matrices 8, 11, 13, and 16 from the EMP.

  The matrix exponential function was "*exactly*" computed as:

Table 2
Improvement percentage in the normwise relative error committed by `expm_euler` and the rest of the codes, for the 3 groups of matrices.

	Group 1	Group 2	Group 3
Er(`expm_euler`)<Er(`exptaynsv3`)	71.00%	68.00%	64.44%
Er(`expm_euler`)>Er(`exptaynsv3`)	29.00%	32.00%	35.56%
Er(`expm_euler`)<Er(`expm_mp`)	54.00%	53.00%	66.67%
Er(`expm_euler`)>Er(`expm_mp`)	46.00%	47.00%	33.33%
Er(`expm_euler`)<Er(`exptayotf`)	75.00%	68.00%	62.22%
Er(`expm_euler`)>Er(`exptayotf`)	25.00%	32.00%	37.78%
Er(`expm_euler`)<Er(`expm3`)	100.00%	99.00%	95.56%
Er(`expm_euler`)>Er(`expm3`)	0.00%	1.00%	4.44%
Er(`expm_euler`)<Er(`expm`)	96.00%	93.00%	91.11%
Er(`expm_euler`)>Er(`expm`)	4.00%	7.00%	8.89%
Er(`expm_euler`)<Er(`sexpm`)	–	–	100.0%
Er(`expm_euler`)>Er(`sexpm`)	–	–	0%

- Thanks to the application of the MATLAB function `eig` to the input matrix A, matrices D and V will be obtained such that $A = V \cdot D \cdot V^{-1}$ and the matrix $E_1 = V \cdot \exp(D) \cdot V^{-1}$ will be computed.
- The matrix $E_2 = \exp(A)$ will be worked out by means of the MATLAB function `expm`, using the function `vpa` as well.
- Matrix E_1 will be considered as the "*exact*" exponential of A if it is fulfilled that:

  $$\frac{\|E_1 - E_2\|_2}{\|E_1\|_2} \leq u.$$

  Otherwise, matrix A will be excluded from Group 3.

The normwise relative error incurred by each of the codes described above in the calculation of the matrix exponential will be quantified as follows:

$$\mathrm{Er(A)} = \frac{\| \exp(A) - \widetilde{\exp}(A)\|_2}{\|\exp(A)\|_2}.$$

While $\exp(A)$ represents the exact solution, $\widetilde{\exp}(A)$ is equivalent to the approximate one computed by each code. In fact, Table 2 summarizes the percentage of matrices for which the relative error committed by `expm_euler` was lower or higher than that attributable to the rest of the codes. As it can be seen, `expm_euler` always obtained a more accurate result for more than 50% of the cases than the other functions, to a greater or lesser extent. As an example, `expm_euler` was more accurate than `exptaynsv3` in 71%, 68%, or 64.44% of the matrices belonging to Groups 1, 2, and 3. In general, `expm_euler` was far superior to `expm3`, `expm` or `sexpm`. Nevertheless, its superiority was much more modest over `expm_mp` in Groups 1 and 2, and somewhat more notable in Group 3. Intermediate improvement percentages compared to those mentioned above, but quite similar to those of `exptaynsv3`, were obtained against `exptayotf`.

It should be clarified that the function `sexpm` was not used in the experiments performed on the first 2 sets of matrices. This was because the excessively high relative errors involved in its execution would have implied rejecting about 50% of the matrices in these groups.

Moreover, Tables 3, 4, and 5 provide more detailed information on the improvement percentage of `expm_euler` versus the remaining codes, and vice versa, distributed in four error intervals and respectively for the matrices conforming the three groups. While E_1 represents the relative error corresponding to `expm_euler`, E_2 is equivalent to the error incurred by any of the other functions. As can be appreciated, the highest improvement percentages of `expm_euler` versus `exptaynsv3`, `expm_mp` and `exptayotf` occur mainly in the first two ranges. Conversely, these percentages are more equitably distributed in the four intervals for `expm3` or `expm`, or mainly centred on the last interval for `sexpm`. Obviously, this means that the improvement in the accuracy of the results provided by `expm_euler` is, in quantifiable terms, clearly more remarkable in the case of the latter (`expm3`, `expm` and `sexpm`) and less noticeable in the case of the former (`exptaynsv3`, `expm_mp` and `exptayotf`). In general, the enhancement of all the functions against `expm_euler` also seems to be slight, except for `expm` in the second group of matrices, and for `expm3` and `expm` in the third group.

Table 6 stores different statistical metrics corresponding to the relative error of the distinct methods. These include the maximum or minimum values reached, the mean, the median, the standard deviation and the sum of all errors. In the particular case of the median, `expm_euler` offered the smallest value for the 3 sets of matrices. Regarding the sum of the errors, `expm_euler` provided the smallest value in the second group of matrices, or the smallest second value for the matrices in the first and third groups. In any case, a careful analysis of any of the parameters listed in the table will give an idea of the outstanding goodness of the results provided by `expm_euler`.

Directly correlated to the above-mentioned error values is the number of significant digits that at least guarantee each of the methods in the computations, as reported in Table 7. In the worst case, `expm_euler` delivers at least 13 correct

Table 3
In detail, improvement percentage in the normwise relative error committed by `expm_euler` (E_1) and the rest of the codes (E_2) for Group 1.

	$E_2 < 2E_1$ $E_1 < 2E_2$	$2E_1 \leq E_2 < 5E_1$ $2E_2 \leq E_1 < 5E_2$	$5E_1 \leq E_2 < 10E_1$ $5E2 \leq E_1 < 10E_2$	$10E_1 \leq E_2$ $10E2 \leq E1$
Er(expm_euler)<Er(exptaynsv3)	69.01%	23.94%	5.63%	1.41%
Er(expm_euler)>Er(exptaynsv3)	82.76%	17.24%	0.00%	0.00%
Er(expm_euler)<Er(expm_mp)	85.19%	11.11%	3.70%	0.00%
Er(expm_euler)>Er(expm_mp)	89.13%	8.70%	2.17%	0.00%
Er(expm_euler)<Er(exptayotf)	58.67%	33.33%	6.67%	1.33%
Er(expm_euler)>Er(exptayotf)	88.00%	12.00%	0.00%	0.00%
Er(expm_euler)<Er(expm3)	3.00%	18.00%	23.00%	56.00%
Er(expm_euler)>Er(expm3)	0.00%	0.00%	0.00%	0.00%
Er(expm_euler)<Er(expm)	8.33%	23.96%	47.92%	19.79%
Er(expm_euler)>Er(expm)	100.00%	8.89%	0.00%	0.00%
Er(expm_euler)<Er(sexpm)	–	–	–	–
Er(expm_euler)>Er(sexpm)	–	–	–	–

Table 4
In detail, improvement percentage in the normwise relative error committed by `expm_euler` (E_1) and the rest of the codes (E_2) for Group 2.

	$E_2 < 2E_1$ $E_1 < 2E_2$	$2E_1 \leq E_2 < 5E_1$ $2E_2 \leq E_1 < 5E_2$	$5E_1 \leq E_2 < 10E_1$ $5E2 \leq E_1 < 10E_2$	$10E_1 \leq E_2$ $10E2 \leq E1$
Er(expm_euler)<Er(exptaynsv3)	91.18%	8.82%	0.00%	0.00%
Er(expm_euler)>Er(exptaynsv3)	93.75%	6.25%	0.00%	0.00%
Er(expm_euler)<Er(expm_mp)	98.11%	1.89%	0.00%	0.00%
Er(expm_euler)>Er(expm_mp)	95.74%	4.26%	0.00%	0.00%
Er(expm_euler)<Er(exptayotf)	82.35%	17.65%	0.00%	0.00%
Er(expm_euler)>Er(exptayotf)	87.50%	12.50%	0.00%	0.00%
Er(expm_euler)<Er(expm3)	15.15%	44.44%	28.28%	12.12%
Er(expm_euler)>Er(expm3)	100.0%	0.00%	0.00%	0.00%
Er(expm_euler)<Er(expm)	51.61%	46.24%	2.15%	0.00%
Er(expm_euler)>Er(expm)	42.86%	42.86%	14.29%	0.00%
Er(expm_euler)<Er(sexpm)	–	–	–	–
Er(expm_euler)>Er(sexpm)	–	–	–	–

Table 5
In detail, improvement percentage in the normwise relative error committed by `expm_euler` (E_1) and the rest of the codes (E_2) for Group 3.

	$E_2 < 2E_1$ $E_1 < 2E_2$	$2E_1 \leq E_2 < 5E_1$ $2E_2 \leq E_1 < 5E_2$	$5E_1 \leq E_2 < 10E_1$ $5E2 \leq E_1 < 10E_2$	$10E_1 \leq E_2$ $10E2 \leq E1$
Er(expm_euler)<Er(exptaynsv3)	79.31%	10.34%	6.90%	3.45%
Er(expm_euler)>Er(exptaynsv3)	75.00%	12.50%	0.00%	12.50%
Er(expm_euler)<Er(expm_mp)	56.67%	10.00%	10.00%	23.33%
Er(expm_euler)>Er(expm_mp)	73.33%	13.33%	0.00%	13.33%
Er(expm_euler)<Er(exptayotf)	60.71%	25.00%	7.14%	7.14%
Er(expm_euler)>Er(exptayotf)	76.47%	11.76%	0.00%	11.76%
Er(expm_euler)<Er(expm3)	4.65%	16.28%	37.21%	41.86%
Er(expm_euler)>Er(expm3)	50.00%	0.00%	0.00%	50.00%
Er(expm_euler)<Er(expm)	19.51%	29.27%	24.39%	26.83%
Er(expm_euler)>Er(expm)	25.00%	0.00%	0.00%	75.00%
Er(expm_euler)<Er(sexpm)	0.00%	2.22%	2.22%	95.56%
Er(expm_euler)>Er(sexpm)	0.00%	0.00%	0.00%	0.00%

digits in the exponential calculation for each matrix in Groups 1 and 2, or 7 digits for matrices in Group 3. Identical values were also achieved by `exptaynsv3`, `expm_mp`, or `exptayotf`, although `exptaynsv3` ensures 8 decimal digits for matrices in Group 3. These values were equal or slightly lower for `expm3` or `expm`, and excessively reduced for `sexpm`.

Figs. 1, 2, and 3 graphically show the results of the different experiments performed in the approximation of the exponential of the matrices comprising Groups 1, 2 and 3, respectively. More specifically, each Figure is in turn constituted by the following 8 subfigures: the normwise relative errors (a), the performance profiles (b), the ratio of normwise relative errors between `expm_euler` and the other codes (c), the lowest and highest normwise relative error rates (d), the orders

Table 6
Minimum, maximum, mean, median, standard deviation, and sum values for the normwise relative errors incurred by the distinct codes for Groups 1, 2, and 3, respectively.

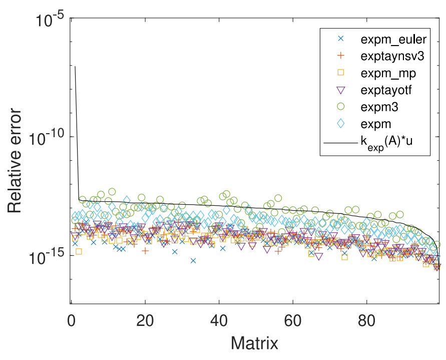
	Min.	Max.	Mean	Median	Std. Dev.	Sum
`expm_euler`	9.23e−18	2.62e−14	5.53e−15	3.75e−15	5.10e−15	5.53e−13
`exptaynsv3`	3.25e−17	2.51e−14	7.36e−15	6.30e−15	5.66e−15	7.36e−13
`expm_mp`	9.20e−18	1.98e−14	5.36e−15	4.40e−15	4.02e−15	5.36e−13
`exptayotf`	2.27e−16	2.29e−14	7.99e−15	6.14e−15	6.11e−15	7.99e−13
`expm3`	8.30e−16	4.87e−13	8.27e−14	4.96e−14	9.21e−14	8.27e−12
`expm`	6.17e−17	2.08e−13	3.22e−14	2.50e−14	3.29e−14	3.22e−12
`sexpm`	–	–	–	–	–	–
`expm_euler`	2.68e−16	1.88e−14	5.56e−15	4.19e−15	4.28e−15	5.56e−13
`exptaynsv3`	2.59e−16	2.69e−14	6.66e−15	5.65e−15	5.30e−15	6.66e−13
`expm_mp`	2.42e−16	2.21e−14	5.77e−15	4.75e−15	4.35e−15	5.77e−13
`exptayotf`	2.89e−16	2.81e−14	6.64e−15	4.78e−15	5.34e−15	6.64e−13
`expm3`	1.48e−15	1.87e−13	2.72e−14	1.82e−14	2.79e−14	2.72e−12
`expm`	5.82e−16	3.28e−14	1.04e−14	8.66e−15	7.60e−15	1.04e−12
`sexpm`	–	–	–	–	–	–
`expm_euler`	2.04e−23	1.21e−08	3.44e−10	1.93e−16	1.86e−09	1.55e−08
`exptaynsv3`	1.86e−20	2.20e−09	5.82e−11	3.12e−16	3.32e−10	2.62e−09
`expm_mp`	1.86e−20	2.41e−08	6.63e−10	4.48e−16	3.68e−09	2.98e−08
`exptayotf`	1.86e−20	5.02e−08	1.12e−09	2.43e−16	7.48e−09	5.03e−08
`expm3`	5.31e−17	1.83e−08	4.19e−10	1.85e−15	2.73e−09	1.89e−08
`expm`	1.77e−17	4.53e−08	1.02e−09	1.35e−15	6.76e−09	4.59e−08
`sexpm`	1.97e−14	9.04e−05	2.01e−06	2.18e−13	1.35e−05	9.06e−05

Table 7
Minimum, maximum, mean, median, and standard deviation values for the number of significant digits in the computed solution by the distinct codes for Groups 1, 2, and 3, respectively.

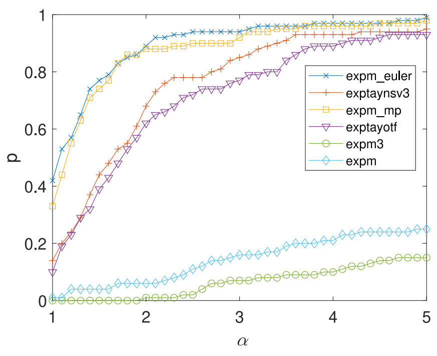
	Min	Max.	Mean	Median	Std. Dev.
`expm_euler`	13	17	13.95	14	0.58
`exptaynsv3`	13	16	13.80	14	0.59
`expm_mp`	13	17	13.97	14	0.56
`exptayotf`	13	15	13.73	14	0.58
`expm3`	12	15	12.84	13	0.65
`expm`	12	16	13.18	13	0.59
`sexpm`	–	–	–	–	–
`expm_euler`	13	15	13.94	14	0.47
`exptaynsv3`	13	15	13.88	14	0.52
`expm_mp`	13	15	13.96	14	0.49
`exptayotf`	13	15	13.87	14	0.51
`expm3`	12	14	13.26	13	0.46
`expm`	13	15	13.62	14	0.53
`sexpm`	–	–	–	–	–
`expm_euler`	7	22	14.49	15	2.20
`exptaynsv3`	8	19	14.51	15	1.84
`expm_mp`	7	19	14.31	15	2.00
`exptayotf`	7	19	14.40	15	1.89
`expm3`	7	16	13.56	14	1.71
`expm`	7	16	13.93	14	1.72
`sexpm`	4	13	11.64	12	1.71

m of the approximation polynomials or the Padé approximants (e), the values of the scaling parameter s (f), the execution times expressed in seconds (g), and the ratio of the execution times between `expm_euler` and the remaining codes in comparison (h).

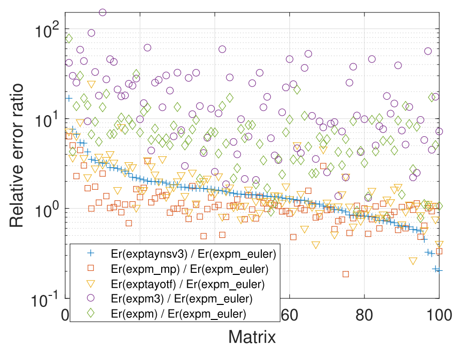
Firstly, the Figs. 1(a), 2(a), and 3(a) depict the normwise relative error caused by each method in the computation of matrices in our testbed with respect to the exact solution. In them, the solid black line represents the $k_{exp}(A)u$ function, where $k_{exp}(A)$ is the condition number of the exponential function for each matrix A and where u is the unit roundoff. This line indicates the expected error in the computation of each matrix. Although slightly above it is also allowed, the stability of the methods is manifested if their errors are, preferably, below this line. Clearly, all the methods have their values below or slightly above this continuous line, with the exception of `sexpm`, which presents its values above in a more differentiated way. It is convenient to clarify that the following matrices were not incorporated in the data shown in Fig. 3(a), due to the high condition numbers they present for the exponential function: 6, 7, 12, 15, 23, 23, 36, 39, 50, and 51 from the MCT, and 1, and 15 from the EMP. Nevertheless, they were taken into account and included in equal conditions in any of the other results exposed.



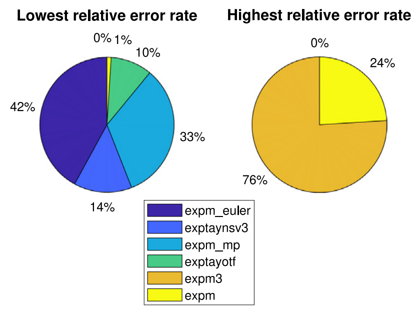
(a) Normwise relative error.



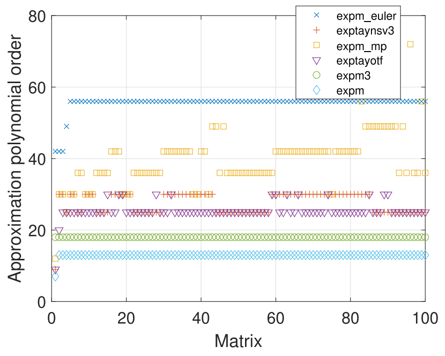
(b) Performance profile.



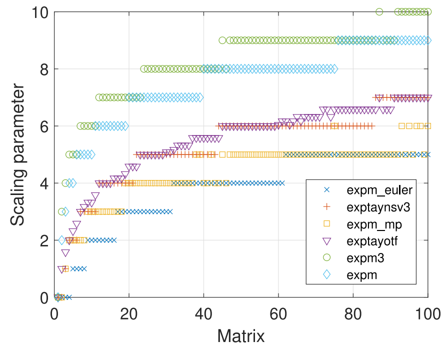
(c) Ratio of relative error.



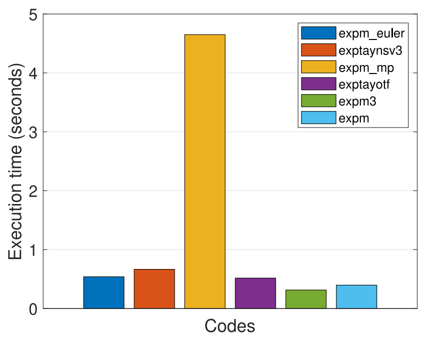
(d) Lowest and highest relative error rate.



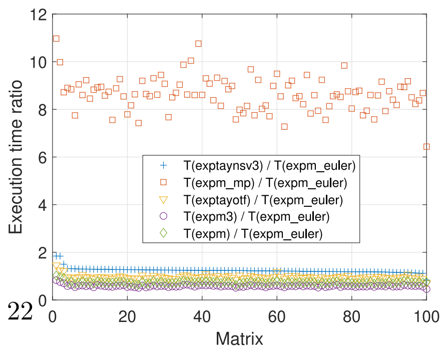
(e) Polynomial order.



(f) Scaling parameter.

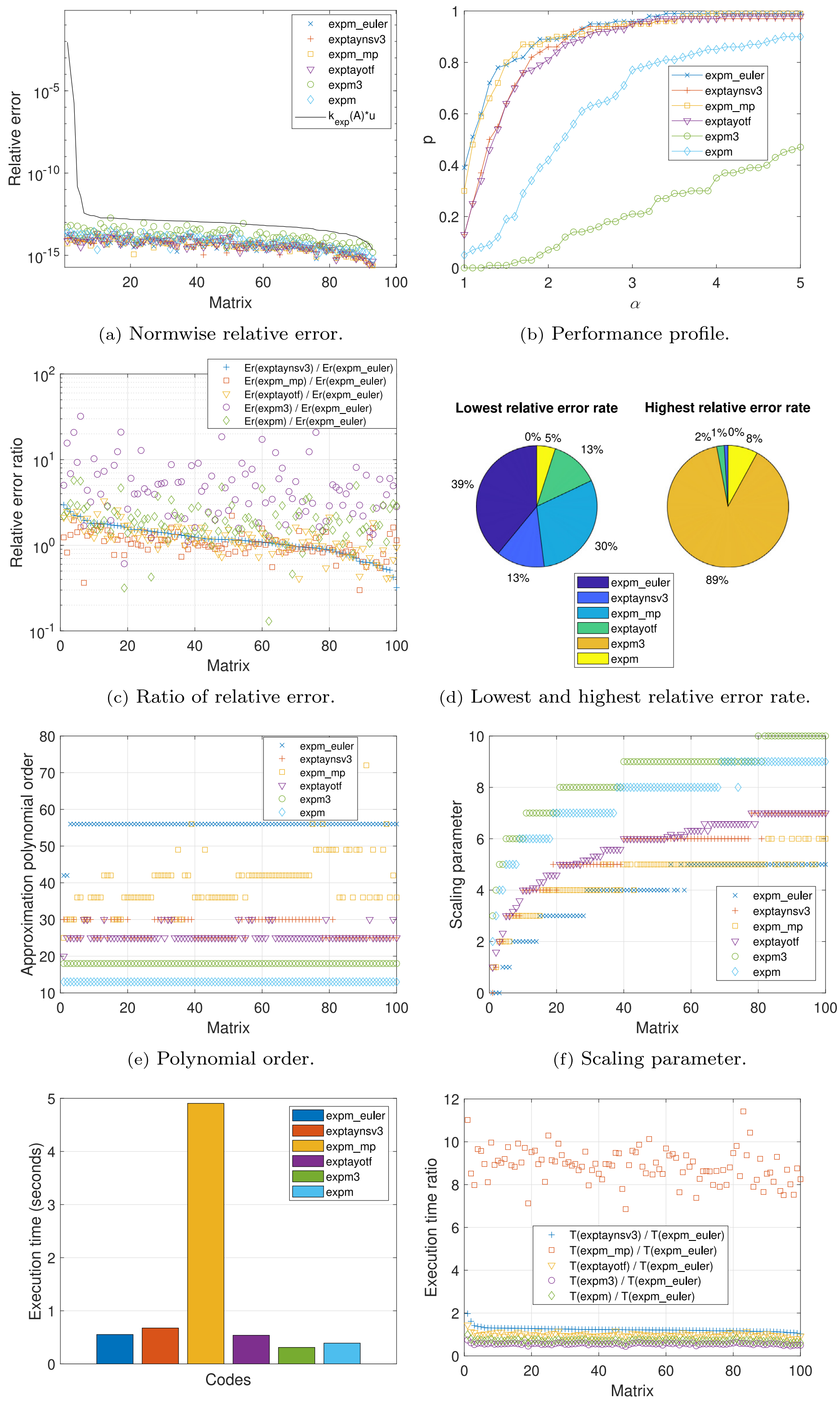


(g) Execution time.



(h) Ratio of execution time.

Fig. 1. Experimental results for Group 1.



(a) Normwise relative error.

(b) Performance profile.

(c) Ratio of relative error.

(d) Lowest and highest relative error rate.

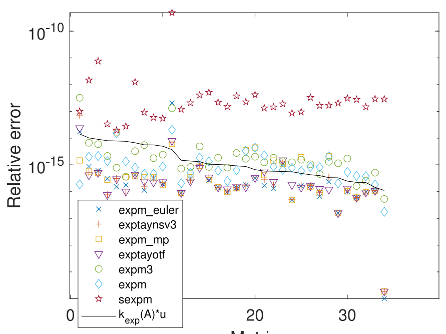
(e) Polynomial order.

(f) Scaling parameter.

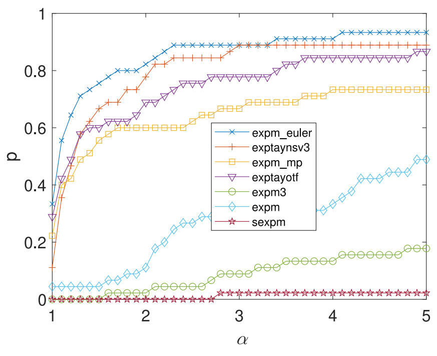
(g) Execution time.

(h) Ratio of execution time.

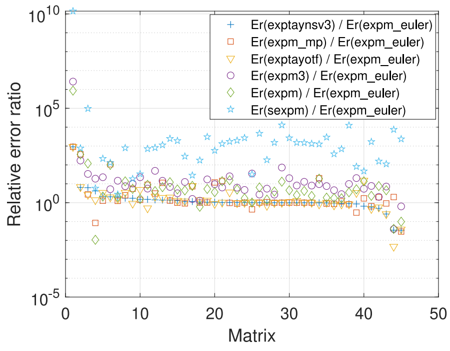
Fig. 2. Experimental results for Group 2.



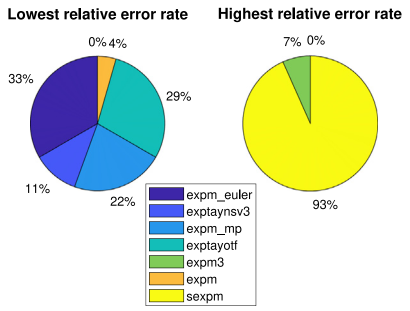
(a) Normwise relative error.



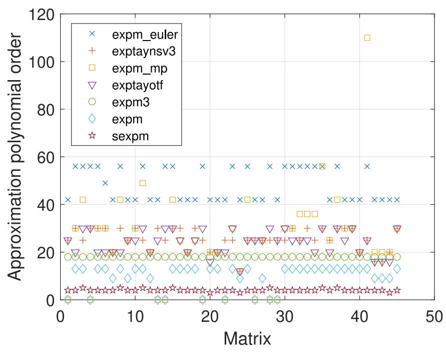
(b) Performance profile.



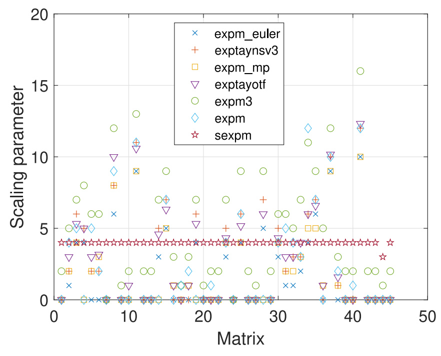
(c) Ratio of relative error.



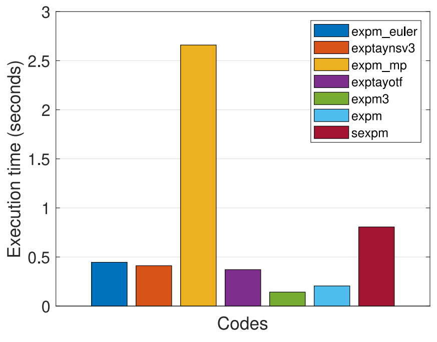
(d) Lowest and highest relative error rate.



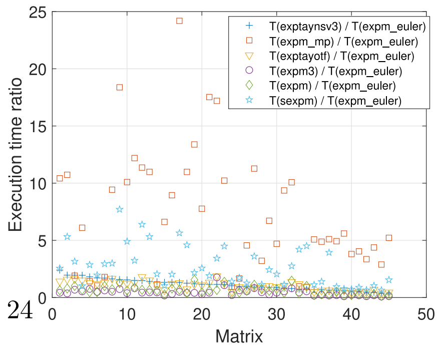
(e) Polynomial order.



(f) Scaling parameter.



(g) Execution time.



(h) Ratio of execution time.

Fig. 3. Experimental results for Group 3.

Table 8
Minimum, maximum, mean and median values reached by the parameters m and s for Groups 1, 2, and 3, respectively.

	m				s			
	Min.	Max.	Mean	Median	Min.	Max.	Mean	Median
`expm_euler`	42	56	55.51	56	0	5.00	3.80	4
`exptaynsv3`	9	30	27.54	30	0	7.00	5.29	6
`expm_mp`	12	72	39.36	42	0	6.00	4.33	5
`exptayotf`	9	30	25.44	25	0	7.00	5.49	6
`expm3`	18	18	18.00	18	0	10.00	8.17	9
`expm`	7	13	12.94	13	0	9.00	7.45	8
`sexpm`	–	–	–	–	–	–	–	–
`expm_euler`	42	56	55.72	56	0	5.00	3.93	4
`exptaynsv3`	25	30	27.45	25	0	7.00	5.48	6
`expm_mp`	25	72	40.25	42	1	6.00	4.49	5
`exptayotf`	20	30	25.65	25	1	7.01	5.69	6
`expm3`	18	18	18.00	18	3	10.00	8.44	9
`expm`	13	13	13.00	13	2	9.00	7.61	8
`sexpm`	–	–	–	–	–	–	–	–
`expm_euler`	42	56	48.38	42	0	10.00	1.87	0
`exptaynsv3`	12	30	25.42	25	0	12.00	2.69	0
`expm_mp`	0	110	25.56	25	0	10.00	1.89	0
`exptayotf`	12	30	24.02	25	0	12.32	2.82	1
`expm3`	18	18	18.00	18	0	16.00	4.93	3
`expm`	0	13	9.69	13	0	12.00	2.60	0
`sexpm`	3	5	4.09	4	3	4.00	3.98	4

Figs. 1(b), 2(b), and 3(b) plot the performance profiles. Each point (α, p) on the picture represents the percentage p of matrices, expressed as a percentage of one, for which the error that takes places in calculating their exponential by each method in particular is less than or equal to α times the smallest relative error made by all the codes to be compared. Regarding the first 2 groups of matrices, `expm_euler` and `expm_mp` present the highest values in the graph, very similar to each other, confirming that they are the best option and provide the most accurate results. They are followed by `exptaynsv3` and `exptayotf`, while `expm` and `expm` 3 occupy lower positions, offering more inaccurate results. In the case of Group 3, `expm_euler` now stands out with a greater difference against its competitors, matched only in a small strip of the graph by `exptaynsv3`. Below them we find `exptayotf` and `expm_mp`, where the latter now offers a more modest performance in terms of accuracy. Once again, `expm`, `expm3`, and especially `sexpm`, are clearly surpassed, exposing than they are the least reliable codes.

The ratio of the relative errors between the other codes and `expm_euler` is shown in Figs. 1(c), 2(c), and 3(c), all of them arranged in descending order according to the quotient Er(`exptaynsv3`)/Er(`expm_euler`). Logically, values of this ratios above unity for each matrix indicate the highest accuracy in the exponential function obtained by `expm_euler`, while values below reflect the opposite. As we can observe, a large part of the values in the graph are greater than unity, thus being favourable results for `expm_euler`.

Figs. 1(d), 2(d), and 3(d) illustrate percentage of matrices for which each method has obtained either the most accurate or the most inaccurate result in calculating the exponential function. Respectively for the matrices composing the test battery, `expm_euler` provided the smallest relative errors in 42%, 39%, or 33% of the cases. These values were always higher than those supplied by any of the other methods considered, followed by those of `expm_mp` for Groups 1 (33%) and 2 (30%), or `exptayotf` for Group 3 (29%). As expected, the percentages of cases in which the highest relative error occurred corresponded to `sexpm`, `expm3`, and `expm`.

Table 8 contains the minimum and maximum values attained by the order m of the approximation polynomial to the matrix exponential and the scaling factor s. The mean and median of these parameters are reported as well. As we mentioned in its description at the beginning of this section, `expm_euler` used values of m between 42 and 56, where the tendency was to employ a degree equal to 56 for the first two sets of matrices and more diversified values for the third. With respect to the methods based on Taylor polynomials (`exptaynsv3`, `expm_mp`, `exptayotf`, and `expm3`), `expm_mp` required the highest orders for the matrices of Groups 1 and 2. In the case of the matrices of Group 3, the value of its median was similar to that of `exptaynsv3` and `exptayotf`, although the maximum value reached was equal to 110, while the latter did not exceed the value of 30. It is worth noting that `expm3` always required a fixed value of m equal to 18, which would explain their behaviour in terms of accuracy of results, since it is a low value. The values of m in the case of `expm` are referred to the order $[m/m]$ of the diagonal Padé approximant. In contrast, the value of m given for `sexpm` is the denominator of the Padé approximant of order $[k/m]$.

Precisely, Figs. 1(e), 2(e), and 3(e) allow us to analyse in more detail the different values of m required by each code. Thus, we can notice that `expm_euler` needed values of m equal to 56 for most of the matrices in the first two groups, while these values were distributed almost 50/50 between 42 and 56 for the third group.

Simultaneously, Table 8 also compiles the values of the scaling parameter s. In the light of these data, we can indicate that `expm_euler` usually demanded smaller values of s than the other codes, while the highest values were on the side

Table 9
Elapsed time (T), in seconds, required by all the codes for the three groups.

	Group 1	Group 2	Group 3
T(expm_euler)	0.54	0.55	0.43
T(exptaynsv3)	0.66	0.68	0.41
T(expm_mp)	4.65	4.91	2.66
T(exptayotf)	0.52	0.54	0.37
T(expm3)	0.31	0.31	0.14
T(expm)	0.40	0.39	0.21
T(sexpm)	–	–	0.81

Table 10
Ratio of the whole elapsed time between the remaining codes and `expm_euler`, for the 3 groups of matrices.

	Group 1	Group 2	Group 3
T(exptaynsv3)/T(expm_euler)	1.23	1.22	0.95
T(expm_mp)/T(expm_euler)	8.63	8.86	6.19
T(exptayotf)/T(expm_euler)	0.96	0.97	0.86
T(expm3)/T(expm_euler)	0.58	0.56	0.33
T(expm)/T(expm_euler)	0.74	0.71	0.49
T(sexpm)/T(expm_euler)	–	–	1.88

of `expm3`. Figs. 1(f), 2(f), and 3(f) represent these same results graphically, although individually for each matrix. For the matrices of Groups 1 and 2, their 2-norms are growing as the index of the matrix increases from 1 to 100. Consequently, it is very straightforward to appreciate how the value of s necessary becomes higher and higher according to the increment of the matrix identifier, regardless of the code to which we pay attention. This ordering according to its 2-norm is no longer respected for matrices from Group 3. Consequently, these values of s are now much more freely dispersed in the picture. It is noticeable the value of s required by `sexpm`, which in most cases is 4.

Regarding the computational cost of the codes taken into account, Table 9 details the execution times of all of them, for the 3 groups of matrices. They have been accounted for by launching 11 distinct runs, discarding the first one and obtaining the average value of the remaining ones. These values indicate that `expm_mp` was always the code that spent the most time on the exponential computation, followed by `exptaynsv3` 3 for matrices of Groups 1 and 2, or by `sexpm` for Group 3. Inversely, `expm3`, followed by `expm`, were always the fastest codes. However, it should be noted that the comparison with `expm` is not entirely fair, since it could run pre-compiled code as part of the MATLAB distribution, which is much more efficient than the interpreted code of which the other implementations are composed. If we take a look at the `expm_euler` response times, we can conclude that they can be categorized as intermediate between those invested by the different methods. These same results are depicted in Figs. 1(g), 2(g), and 3(g) in the form of a bar graph.

The quotient between the total computation time devoted by each of the methods and `expm_euler` is given in Table 10. Thus, for each of the three groups of matrices, `expm_euler` is 8.63, 8.86 and 6.19 times faster than `expm_mp`, one of its main competitors especially for the matrices of the first two sets. For Groups 1 and 2, `expm_euler` exhibited a speedup of 1.23 and of 1.22 versus `exptaynsv3`. Nevertheless, for Group 3, `exptaynsv3` improved in time to `expm_euler`, although very slightly. Compared to `exptayotf`, `expm_euler` was always outperformed, with velocity increments for the former of 1.04, 1.03 and 1.16.

Finally, the ratio of the time spent by each method and `expm_euler` on the calculation of the exponential of each of the matrices forming our testbed is plotted in detail in Figs. 1(h), 2(h), and 3(h), all of them ordered according to the factor T(`exptaynsv3`)/T(`expm_euler`). With the exception of `expm_mp` and `sexpm`, whose values stand out in the upper part of the pictures, it can be stated that the rest of the methods gave place to values that were always, or in the vast majority of cases, below the above-mentioned factor. Accordingly, the ratio between `expm_mp` and `expm_euler` took values belonging to the intervals [6.43, 10.97], [6.86, 11.42] and [0.93, 24.19], respectively for each group of matrices. Furthermore, these intervals were equal to [0.84, 1.84], [1.03, 1.97] and [0.31, 2.40] in the case of `exptaynsv3`. Not surprisingly, the values for `exptayotf` were somewhat more modest, corresponding to the ranges [0.71, 1.46], [0.77, 1.44], and [0.36, 1.88].

5. Conclusions

In this paper, the novel application of Euler polynomials in the computation of the matrix exponential function has been presented. Besides, the well-known scaling and squaring technique has been also taken into consideration to reduce the norm of the calculation matrices. The paper incorporates the appropriate deductions and the mathematical formulations corresponding to the series development of this function in terms of the Euler matrix polynomials. From this formulation, together with the one described for determining the order of the approximation polynomial and the scaling parameter by means of an absolute forward-type theoretical error, the pertinent algorithms have been designed. They have been implemented and have given rise to a code programmed in the MATLAB language whose numerical and computational performance has been compared with those of the best known codes in the literature devoted to the calculation of the exponential function. For this purpose, a test battery composed of matrices belonging to three well differentiated categories has been used. The results reveals that the new numerical method is as accurate or even more accurate than practically all the methods used in the comparison, with computation times halfway among all of them.

Data availability

I have shared the link to the code at the article.

Acknowledgements

This work has been partially supported by Spanish Ministerio de Ciencia e Innovación (Project PID2020-113656RB-C22) supported by MCIN/AEI/10.13039/501100011033 and by the Vicerrectorado de Investigación de la Universitat Politècnica de València (PAID-11-21).

References

[1] C. Moler, C. Van Loan, Nineteen dubious ways to compute the exponential of a matrix, SIAM Rev. 20 (4) (1978) 801–836.
[2] C. Moler, C. Van Loan, Nineteen dubious ways to compute the exponential of a matrix, twenty-five years later, SIAM Rev. 45 (1) (2003) 3–49.
[3] B. Hammoud, L. Olivieri, L. Righetti, J. Carpentier, A. Del Prete, Exponential integration for efficient and accurate multibody simulation with stiff viscoelastic contacts, Multibody Syst. Dyn. 54 (4) (2022) 443–460.
[4] E. Ponsoda, S. Blanes, P. Bader, New efficient numerical methods to describe the heat transfer in a solid medium, Math. Comput. Modelling 54 (7–8) (2011) 1858–1862.
[5] M. Benzi, E. Estrada, C. Klymko, Ranking hubs and authorities using matrix functions, Linear Algebra Appl. 438 (2013) 2447–2474.
[6] E. Estrada, J.A. Rodriguez-Velazquez, Subgraph centrality in complex networks, Phys. Rev. E 71 (5) (2005) 056103.
[7] H. Zhang, Y. Xu, A Chebyshev collocation based sequential matrix exponential method for the generalized density evolution equation, Probab. Eng. Mech. 63 (2021) 103118.
[8] M. Caliari, L. Einkemmer, A. Moriggl, A. Ostermann, An accurate and time-parallel rational exponential integrator for hyperbolic and oscillatory PDEs, J. Comput. Phys. 437 (2021) 110289.
[9] J. Nianga, F. Mejni, T. Kanit, J. Li, A. Imad, Analysis of crack parameters under pure mode II loading by modified exponential matrix method, Theor. Appl. Fract. Mech. 111 (2021) 102820.
[10] G.A. Baker, P. Graves-Morris, Padé Approximants, in: Encyclopedia of Mathematics and its Applications, Cambridge University Press, 1996.
[11] L. Dieci, A. Papini, Padé approximation for the exponential of a block triangular matrix, Linear Algebra Appl. 308 (2000) 183–202.
[12] N.J. Higham, The Scaling and Squaring Method for the Matrix Exponential Revisited, Tech. Rep. 452, Manchester Centre for Computational Mathematics, 2004.
[13] 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.
[14] S. Güttel, Y. Nakatsukasa, Scaled and squared subdiagonal Padé approximation for the matrix exponential, SIAM J. Matrix Anal. Appl. 37 (1) (2016) 145–170.
[15] J. Sastre, J. Ibáñez, E. Defez, P. Ruiz, New scaling-squaring Taylor algorithms for computing the matrix exponential, SIAM J. Sci. Comput. 37 (1) (2015) A439–A455.
[16] P. Ruiz, J. Sastre, J. Ibáñez, E. Defez, High performance computing of the matrix exponential, J. Comput. Appl. Math. 291 (2016) 370–379.
[17] J. Sastre, J. Ibáñez, E. Defez, Boosting the computation of the matrix exponential, Appl. Math. Comput. 340 (2019) 206–220.
[18] M. Caliari, F. Zivcovich, On-the-fly backward error estimate for matrix exponential approximation by Taylor algorithm, J. Comput. Appl. Math. 346 (2019) 532–548.
[19] P. Bader, S. Blanes, F. Casas, Computing the matrix exponential with an optimized Taylor polynomial approximation, Mathematics 7 (12) (2019) 1174.
[20] E. Defez, L. Jódar, Some applications of the Hermite matrix polynomials series expansions, J. Comput. Appl. Math. 99 (1) (1998) 105–117.
[21] J. Sastre, J. Ibáñez, E. Defez, P. Ruiz, Efficient orthogonal matrix polynomial based method for computing matrix exponential, Appl. Math. Comput. 217 (14) (2011) 6451–6463.
[22] E. Defez, J. Ibáñez, P. Alonso-Jordá, J.M. Alonso, J. Peinado, On Bernoulli matrix polynomials and matrix exponential approximation, J. Comput. Appl. Math. 404 (2022) 113207.
[23] J. Respondek, Another dubious way to compute the exponential of a matrix, in: Computational Science and Its Applications, ICCSA 2021, in: Lecture Notes in Computer Science, vol. 12949, Springer International Publishing, Springer, 2021, pp. 465–478.
[24] J. Respondek, Matrix black box algorithms – A survey, Bull. Pol. Acad. Sci. Tech. Sci. 70 (2) (2022) e140535.
[25] E.D. Rainville, Special Functions, Vol. 442, New York, 1960.
[26] 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.
[27] W. Wang, Some results on sums of products of Bernoulli polynomials and Euler polynomials, Ramanujan J. 32 (2) (2013) 159–184.
[28] H. Srivastava, A. Pinter, Remarks on some relationships between the Bernoulli and Euler polynomials, Appl. Math. Lett. 17 (4) (2004) 375–380.
[29] G.-S. Cheon, A note on the Bernoulli and Euler polynomials, Appl. Math. Lett. 16 (3) (2003) 365–368.
[30] X. Xiang, X. Ma, M. Ma, W. Wu, L. Yu, Research and application of novel Euler polynomial-driven grey model for short-term PM10 forecasting, Grey Syst.: Theory Appl. (2020).
[31] N.A. Khan, O.I. Khalaf, C.A.T. Romero, M. Sulaiman, M.A. Bakar, Application of Euler neural networks with soft computing paradigm to solve nonlinear problems arising in heat transfer, Entropy 23 (8) (2021) 1053.
[32] M.S. Paterson, L.J. Stockmeyer, On the number of nonscalar multiplications necessary to evaluate polynomials, SIAM J. Comput. 2 (1) (1973) 60–66.
[33] J. Sastre, J. Ibáñez, P. Ruiz, E. Defez, Accurate and efficient matrix exponential computation, Int. J. Comput. Math. 91 (1) (2013) 97–112.
[34] J. Sastre, J. Ibáñez, On the backward and forward error of approximations of analytic functions and applications to the computation of matrix functions, J. Comput. Appl. Math. (2022) 114706.
[35] M. Fasi, N.J. Higham, An arbitrary precision scaling and squaring algorithm for the matrix exponential, SIAM J. Matrix Anal. Appl. 40 (4) (2019) 1233–1256.
[36] Advanpix, Multiprecision computing toolbox, 2022, URL: http://www.advanpix.com.
[37] N.J. Higham, The scaling and squaring method for the matrix exponential revisited, SIAM J. Matrix Anal. Appl. 26 (4) (2005) 1179–1193.
[38] N.J. Higham, The matrix computation toolbox, 2002, URL: http://www.ma.man.ac.uk/~higham/mctoolbox.
[39] T.G. Wright, Eigtool, version 2.1, 2009, URL: http://www.comlab.ox.ac.uk/pseudospectra/eigtool.