Journal of Computational and Applied Mathematics 291 (2016) 370–379

Contents lists available at ScienceDirect

Journal of Computational and Applied Mathematics

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

# High performance computing of the matrix exponential

P. Ruiz [a,*], J. Sastre [b], J. Ibáñez [a], E. Defez [c]

[a] Instituto de Instrumentación para Imagen Molecular, Universitat Politècnica de València, Spain
[b] Instituto de Telecomunicaciones y Aplicaciones Multimedia, Universitat Politècnica de València, Spain
[c] Instituto de Matemática Multidisciplinar, Universitat Politècnica de València, Spain

ARTICLE INFO

*Article history:*
Received 15 October 2014
Received in revised form 30 March 2015

*Keywords:*
Matrix exponential
Scaling and squaring
Taylor series

ABSTRACT

This work presents a new algorithm for matrix exponential computation that significantly simplifies a Taylor scaling and squaring algorithm presented previously by the authors, preserving accuracy. A Matlab version of the new simplified algorithm has been compared with the original algorithm, providing similar results in terms of accuracy, but reducing processing time. It has also been compared with two state-of-the-art implementations based on Padé approximations, one commercial and the other implemented in Matlab, getting better accuracy and processing time results in the majority of cases.



## 1. Introduction

Matrix function computation has received remarkable attention in the last decades due to its usefulness in a great variety of engineering problems. Especially noteworthy is the matrix exponential, which emerge in the solution of systems of linear differential equations in numerous applications and a large number of methods for its computation have been proposed [1,2]. Moreover, in many cases, the resolution of these systems involve large matrices, so, not only accurate, but also efficient methods are needed. In this sense, the authors presented in [3] two modifications of a Taylor-based scaling and squaring algorithm to reduce computational costs while preserving accuracy.

In [4] the authors presented a scaling and squaring Taylor algorithm based on an improved mixed backward and forward error analysis, which was more accurate than existing state-of-the-art algorithms for matrix exponential such as that in [5], in the majority of test matrices with a lower or similar cost. Subsequently, in [6], the authors gave a new formula for the forward relative error of matrix exponential Taylor approximation and proposed to increase the allowed forward error bound depending on the matrix size and the Taylor approximation order. This algorithm reduces the computational cost in exchange for a small impact in accuracy. In this work, we present a new algorithm that significantly simplifies the one presented in [4] providing a competitive scaling and squaring algorithm for matrix exponential computation in comparison with both previous algorithms and the state-of-the-art implementations based on Padé approximations from [5,7].

Throughout this paper $\mathbb{C}^{n\times n}$ denotes the set of complex matrices of size $n \times n$, $I$ denotes the identity matrix for this set, $\rho(A)$ is the spectral radius of matrix $A$, and $\mathbb{N}$ denotes the set of positive integers. The matrix norm $\|\cdot\|$ denotes any subordinate matrix norm; in particular $\|\cdot\|_1$ and $\|\cdot\|_2$ are the 1-norm and the 2-norm, respectively. The symbols $\lceil\cdot\rceil$ and $\lfloor\cdot\rfloor$ denote the smallest following and the largest previous integer, respectively. This paper is organized as follows: Section 2 presents a general scaling and squaring Taylor algorithm; Section 3 presents the scaling and squaring error analysis; the

* Corresponding author.
*E-mail address:* pruiz@dsic.upv.es (P. Ruiz).

http://dx.doi.org/10.1016/j.cam.2015.04.001
0377-0427/

new algorithm is given in Section 4; finally, Section 5 shows numerical results and Section 6 gives some conclusions. Next Theorem 1 from [6] and the new Theorem 2 will be used in Section 3 to bound the norm of matrix power series.

**Theorem 1.** *Let $h_l(x) = \sum_{k\geq l} b_k x^k$ be a power series with radius of convergence R, and let $\tilde{h}_l(x) = \sum_{k\geq l} |b_k| x^k$. For any matrix $A \in \mathbb{C}^{n\times n}$ with $\rho(A) < R$, if $a_k$ is an upper bound for $\|A^k\|$ ($\|A^k\| \leq a_k$), $p \in \mathbb{N}$, $1 \leq p \leq l$, $p_0 \in \mathbb{N}$ is the multiple of $p$ with $l \leq p_0 \leq l + p - 1$, and*

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

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

**Theorem 2.** *Let $l \in \mathbb{N}$, $l \geq 1$, and let $q \in \mathbb{N}$ be the minimum value with $1 \leq q \leq l$ such that*

$$\|A^q\|^{\frac{1}{q}} \leq \max\{\|A^k\|^{\frac{1}{k}} : k = l, l+1, \ldots, q_0 - 1, q_0 + 1, q_0 + 2, \ldots, l + q - 1\}, \tag{2}$$

*where $q_0 \in \mathbb{N}$ is the multiple of $q$ with $l \leq q_0 \leq l + q - 1$. Then if*

$$\|A^{k_0}\|^{\frac{1}{k_0}} = \max\{\|A^k\|^{\frac{1}{k}} : k = l, l+1, \ldots, q_0 - 1, q_0 + 1, q_0 + 2, \ldots, l + q - 1\} \tag{3}$$

*then*

$$\max\{\|A^k\|^{\frac{1}{k}} : k \geq l\} = \|A^{k_0}\|^{\frac{1}{k_0}} \tag{4}$$

**Proof.** Since $q_0$ is a multiple of $q$, then $q_0/q \in \mathbb{N}$ and using (2) and (3) one gets

$$\|A^{q_0}\|^{1/q_0} = \|A^{qq_0/q}\|^{1/q_0} \leq \|A^q\|^{q_0/(qq_0)} = \|A^q\|^{1/q} \leq \|A^{k_0}\|^{1/k_0}. \tag{5}$$

For any integer $k \geq l + q$ we can write $k = l + i + jq$ for positive integers $i$ and $j$ with $0 \leq i \leq q - 1$ and $j = [k - (l + i)]/q$, and then using (2), (3) and (5) it follows that

$$\|A^k\|^{\frac{1}{k}} \leq \left[\|A^{l+i}\| \|A^q\|^j\right]^{\frac{1}{k}} \leq \left[\|A^{k_0}\|^{\frac{l+i}{k_0}} \|A^{k_0}\|^{\frac{jq}{k_0}}\right]^{\frac{1}{k}} = \|A^{k_0}\|^{\frac{k}{k_0 k}} = \|A^{k_0}\|^{\frac{1}{k_0}}. \quad \square \tag{6}$$

## 2. Taylor algorithm

Taylor approximation of order $m$ of exponential of matrix $A \in \mathbb{C}^{n\times n}$ can be expressed as the matrix polynomial $T_m(A) = \sum_{k=0}^{m} A^k/k!$. The scaling and squaring algorithms with Taylor approximations are based on the approximation $e^A = \left(e^{2^{-s}A}\right)^{2^s} \approx \left(T_m(2^{-s}A)\right)^{2^s}$ [1], where the nonnegative integers $m$ and $s$ are chosen to achieve full machine accuracy at a minimum cost.

A general scaling and squaring Taylor algorithm for computing the matrix exponential is presented in Algorithm 1, where $m_M$ is the maximum allowed value of $m$.

**Algorithm 1** General scaling and squaring Taylor algorithm for computing $B = e^A$, where $A \in \mathbb{C}^{n\times n}$ and $m_M$ is the maximum approximation order allowed.

1: Preprocessing of matrix $A$.
2: Choose $m_k \leqslant m_M$, and an adequate scaling parameter $s \in \mathbb{N} \cup \{0\}$ for the Taylor approximation with scaling.
3: Compute $B = T_{m_k}(A/2^s)$ using (7)
4: **for** $i = 1 : s$ **do**
5: $\quad B = B^2$
6: **end for**
7: Postprocessing of matrix $B$.

The preprocessing and postprocessing steps (1 and 7) are based on applying transformations to reduce the norm of matrix $A$, see [2,8], and will not be discussed in this paper.

In Step 2, the optimal order of Taylor approximation $m_k \leqslant m_M$ and the scaling parameter $s$ are chosen. Matrix polynomial $T_m(2^s A)$ can be computed optimally in terms of matrix products using values for $m$ in the set $m_k = \{1, 2, 4, 6, 9, 12, 16, 20, 25, 30, \ldots\}$, $k = 0, 1, \ldots$, respectively, see [2, p. 72–74]. The choice of $s$ is fully described in Section 3.

After that, in Step 3, we compute the matrix exponential approximation of the scaled matrix by using the modified Horner and Paterson–Stockmeyer's method proposed in [4, p. 1836–1837]. Note that this modified method has the same optimal

**Table 1**
Values of $q_k$ depending on the selection of $m_M$.

| $k$ | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
|---|---|---|---|---|---|---|---|---|---|---|
| $m_M$ | $m_k$ | | | | | | | | | |
| | 1 | 2 | 4 | 6 | 9 | 12 | 16 | 20 | 25 | 30 |
| 20 | 1 | 2 | 2 | 3 | 3 | 4 | 4 | 4 | | |
| 25 | 1 | 2 | 2 | 3 | 3 | 4 | 4 | 5 | 5 | |
| 30 | 1 | 2 | 2 | 3 | 3 | 4 | 4 | 5 | 5 | 5 |

values for $m$ as the original one:

$$\begin{aligned} T_m\left(2^{-s}A\right) = &\left\{\left\{\cdots\left\{\frac{A_q}{2^s m}+A_{q-1}\right\}/[2^s(m-1)]+A_{q-2}\right\}/[2^s(m-2)]+\cdots+A_2\right\}/[2^s(m-q+2)]+A \\ &+2^s(m-q+1)I\Bigg\}\frac{A_q}{2^{2s}(m-q+1)(m-q)}+A_{q-1}\Bigg\}/[2^s(m-q-1)]+A_{q-2}\Bigg\} \\ &/\left[2^s(m-q-2)\right]+\cdots+A_2\Bigg\}/[2^s(m-2q+2)]+A+2^s(m-2q+1)I\Bigg\} \\ &\times\frac{A_q}{2^{2s}(m-2q+1)(m-2q)}+\cdots+A_2\Bigg\}/[2^s(q+2)]+A+2^s(q+1)I\Bigg\} \\ &\times\frac{A_q}{2^{2s}(q+1)q}+A_{q-1}\Bigg\}/[2^s(q-1)]+\cdots+A_2\Bigg\}/[2^s 2]+A\Bigg\}/2^s+I, \end{aligned} \tag{7}$$

where matrix powers $A_i = A^i$, $i = 1, 2, \ldots, q$ are computed, with $q = \lceil\sqrt{m_k}\rceil$ or $\lfloor\sqrt{m_k}\rfloor$, both values dividing $m_k$ and giving the same cost [2, p. 74]. Table 1 shows some optimal values of $q$ for orders $m_k$, $k = 0,\ 1,\ 2, \ldots, M$, with $m_M = 20, 25, 30$, denoted by $q_k$.

Finally, after the evaluation of $T_m\left(2^{-s}A\right)$, $s$ repeated squarings are applied in Steps 4–6 and the postprocessing is applied in Step 7 to obtain the matrix exponential approximation of the original matrix $A$. The computational cost of Algorithm 1 in terms of matrix products is $\mathrm{Cost}(m_k, s) = k + s$.

## 3. Error analysis

Following [4,6], denoting the remainder of the Taylor series as $R_m(A) = \sum_{k\geq m+1} A^k/k!$, for a scaled matrix $2^{-s}A$, $s \in \mathbb{N}\cup\{0\}$, we can write

$$\left(T_m(2^{-s}A)\right)^{2^s} = e^A\left(I+g_{m+1}(2^{-s}A)\right)^{2^s} = e^{A+2^s h_{m+1}\left(2^{-s}A\right)}, \tag{8}$$

$$g_{m+1}(2^{-s}A) = -e^{-2^{-s}A}R_m(2^{-s}A), \qquad h_{m+1}\left(2^{-s}A\right) = \log\left(I+g_{m+1}(2^{-s}A)\right), \tag{9}$$

where log denotes the principal logarithm, $h_{m+1}(X)$ is defined in the set $\Omega_m = \left\{X\in\mathbb{C}^{n\times n} : \rho\left(e^{-X}T_m(X)-I\right)<1\right\}$, and both $g_{m+1}(2^{-s}A)$ and $h_{m+1}\left(2^{-s}A\right)$ are holomorphic functions of $A$ in $\Omega_m$ and then commute with $A$. As showed in [4], $h_{m+1}(2^{-s}A)$ and $g_{m+1}(2^{-s}A)$ are related with the backward and forward errors in exact arithmetic from the approximation of $e^A$ by the Taylor series with scaling and squaring, respectively. Choosing $s$ so that

$$\left\|h_{m+1}\left(2^{-s}A\right)\right\| \leq \max\left\{1, \left\|2^{-s}A\right\|\right\}u, \tag{10}$$

where $u = 2^{-53}$ is the unit roundoff in IEEE double precision arithmetic, then: if $2^{-s}\|A\| \geq 1$, then the backward error $\Delta A \leq \|A\|\,u$ and using (8) one gets $\left(T_m\left(2^{-s}A\right)\right)^{2^s} = e^{A+\Delta A} \approx e^A$, and if $2^{-s}\|A\| < 1$, using (8)–(10) and the Taylor series one gets

$$\left\|R_m\left(2^{-s}A\right)\right\| \approx \left\|T_m\left(2^{-s}A\right)\right\|u. \tag{11}$$

Hence, as Algorithm 1 evaluates explicitly $T_m\left(2^{-s}A\right)$, by (11) one gets $e^{2^{-s}A} = T_m\left(2^{-s}A\right)+R_m\left(2^{-s}A\right) \approx T_m\left(2^{-s}A\right)$, and there is no need to increase $m$ or the scaling parameter $s$ to try to get a higher accuracy. Using scalar Taylor series in (9) one gets

$$g_{m+1}(x) = \sum_{k\geq m+1} b_k^{(m)}x^k, \qquad h_{m+1}(x) = \sum_{k\geq 1}\frac{(-1)^{k+1}(g_{m+1}(x))^k}{k} = \sum_{k\geq m+1} c_k^{(m)}x^k, \tag{12}$$

where $b_k^{(m)}$ and $c_k^{(m)}$ depend on the order $m$. Moreover, $b_k^{(m)} = c_k^{(m)}$, $k = m+1, m+2, \ldots, 2m+1$ and if $\|h_{m+1}(2^{-s}A)\| \ll 1$ or if $\|g_{m+1}(2^{-s}A)\| \ll 1$, then $h_{m+1}(2^{-s}A) \approx g_{m+1}(2^{-s}A)$, see [6]. Using MATLAB symbolic Math Toolbox, high precision arithmetic, 200 series terms and a zero finder we obtained the maximal values $\Theta_m$ of $\Theta = \left\|2^{-s}A\right\|$, shown in Table 2, such

**Table 2**
Maximal values $\Theta_m = \left\|2^{-s}A\right\|$ such that $\tilde{h}_{m+1}(\Theta_m) \leq \max\{1, \Theta_m\}u$, coefficient ratios $c_{m+1}^{(m)}/c_{m+2}^{(m)}$, values $u/|c_{m+2}^{(m)}|$, maximal values $\hat{\Theta}_m$ using only the first 2 terms in series $\tilde{h}_{m+1}(\hat{\Theta}_m)$, and values $\tilde{h}_{m+1}(\hat{\Theta}_m)/(\max\{1, \hat{\Theta}_m\}u)$ considering 200 series terms.

| $m$ | $\Theta_m$ | $c_{m+1}^{(m)}/c_{m+2}^{(m)}$ | $u/\lvert c_{m+2}^{(m)}\rvert$ | $\hat{\Theta}_m$ | $\frac{\tilde{h}_{m+1}(\hat{\Theta}_m)}{\max\{1,\hat{\Theta}_m\}u}$ |
|---|---|---|---|---|---|
| 1 | 1.490116111983279e−8 | −3/2 | 3.3e−16 | 1.490e−8 | 1.00 |
| 2 | 8.733457513635361e−6 | −4/3 | 8.9e−16 | 8.733e−6 | 1.00 |
| 4 | 1.678018844321752e−3 | −6/5 | 1.6e−14 | 1.678e−3 | 1.00 |
| 6 | 1.773082199654024e−2 | −8/7 | 6.4e−13 | 1.777e−2 | 1.02 |
| 9 | 1.137689245787824e−1 | −11/10 | 4.4e−10 | 1.150e−1 | 1.11 |
| 12 | 3.280542018037257e−1 | −14/13 | 7.5e−7 | 3.358e−1 | 1.37 |
| 16 | 7.912740176600240e−1 | −18/17 | 4.2e−2 | 8.269e−1 | 2.19 |
| 20 | 1.438252596804337 | −22/21 | 5.9e03 | 1.474 | 1.70 |
| 25 | 2.428582524442827 | −27/26 | 4.7e10 | 2.538 | 3.36 |
| 30 | 3.539666348743690 | −32/31 | 9.4e17 | 3.771 | 8.34 |

that, using the notation of Theorem 1

$$\|h_{m+1}\left(2^{-s}A\right)\| \leq \tilde{h}_{m+1}(\Theta) = \sum_{k\geq m+1} |c_k^{(m)}|\Theta^k \leq \max\{1, \Theta\}\, u. \tag{13}$$

Hence, if $\|2^{-s}A\| \leq \Theta_m$ then (10) holds. For the cases where $\Theta_m > 1$, note that $f(\Theta) = \tilde{h}_{m+1}(\Theta) - \Theta u$ is a continuous function in $[0, \Theta_m]$ and $f(\Theta_m) = 0, f(0) = 0$. For $m = 20, 25, 30$ we have checked that there are no other zeros in $[0, \Theta_m]$, and $f(\Theta) < 0$, $\Theta \in ]0, \Theta_m[$. Thus, for those orders the next bound holds

$$\|h_{m+1}\left(2^{-s}A\right)\| \leq \tilde{h}_{m+1}\left(\|2^{-s}A\|\right) = \tilde{h}_{m+1}(\Theta) \leq \Theta u, \quad 0 \leq \Theta \leq \Theta_m. \tag{14}$$

## 4. New Taylor algorithm

### 4.1. New scaling algorithm

In this section a new scaling algorithm is proposed, being a simplification of that presented in [4, p. 1837–1838]. For all norms appearing in the scaling algorithm we will use the 1-norm, and $m_M$ will be the maximum allowed Taylor order. Using the bounds and a similar process that we describe below, we will first check if any of the Taylor optimal orders $m_k = 1, 2, 4, \ldots, m_{M-1}$ satisfy (10) without scaling, i.e. with $s = 0$. If not, we will calculate the optimal scaling $s$ for order $m_M$ in two phases: first, we will calculate an initial value of the scaling parameter, $s_0$, and then we will try to refine it, testing if it can be reduced. In this paper we have simplified both phases with respect to the algorithms from [4,6], avoiding costly and complex checks that rarely allow to reduce the scaling parameter.

We begin estimating the 1-norm of $\|A^{m+1}\|$ using the block 1-norm estimation algorithm of Higham and Tisseur [9]. For a $n \times n$ matrix this algorithm carries out a 1-norm power iteration whose iterates are $n \times t$ matrices, where $t$ is a parameter that has been taken to be 2, see [5, p. 983]. Hence, the estimation algorithm has $O(n^2)$ computational cost, negligible compared to the cost of a matrix product, i.e. $O(n^3)$.

In [4, p. 1837], the upper bounds $a_k$ for $\|A^k\|$ needed to apply Theorem 1 in (13) were obtained using products of norms of matrix powers estimated for current and previous tested orders, i.e. $\|A^{m_k+1}\|$, $k = 0, 1, 2, \ldots, M$, and the powers of $A$ computed for evaluation of $T_{m_M}\left(2^{-s}A\right), A^i$, $i = 1, 2, \ldots, q$, as

$$\begin{aligned}\left\|A^k\right\| \leq a_k = \min \Big\{ &\|A\|^{i_1} \left\|A^2\right\|^{i_2} \cdots \left\|A^q\right\|^{i_q} \left\|A^{m_1+1}\right\|^{i_{m_1+1}} \left\|A^{m_2+1}\right\|^{i_{m_2+1}} \cdots \\ &\times \left\|A^{m_M+1}\right\|^{i_{m_M+1}} : i_1 + 2i_2 + \cdots + qi_q + (m_1+1)i_{m_1+1} \\ &+ (m_2+1)i_{m_2+1} + \cdots + (m_M+1)i_{m_M+1} = k \Big\},\end{aligned} \tag{15}$$

where the minimum was desirable, but not necessary. Then, in [4], $\alpha_p$ values from Theorem 1 with $l = m_M + 1$, see (1), were obtained successively for $p = 2, 3, \ldots, q, m_1 + 1, m_2 + 1, \ldots, m_M + 1$, stopping the process when $(a_p)^{1/p} \leq \max\{(a_k)^{1/k} : k = m+1, m+2, \ldots, m+p\}$, since if this condition holds, then for $p' > p$ it follows that $\alpha_{p'} \geq \alpha_p$, see [6, p. 105] for a demonstration. Then, the minimum value among all values $\alpha_p$, denoted by $\alpha_{\min}$, was selected. Finally, the initial minimum scaling parameter $s_0 \geq 0$ so that $2^{-s_0}\alpha_{\min} \leq \Theta_{m_M}$ was computed as follows: if $\alpha_{\min} \leq \Theta_{m_M}$ then $s_0 = 0$, and otherwise $s_0 = \lceil \log_2(\alpha_{\min}/\Theta_{m_M}) \rceil$. In [4] it was also shown that taking $s = s_0$ then (10) holds.

In this work, we have simplified all that process by directly approximating

$$\alpha_{\min} \approx \max\{a_{m+1}^{1/(m+1)}, a_{m+2}^{1/(m+2)}\}, \tag{16}$$

where $a_{m+1}$ and $a_{m+2}$ are the 1-norm estimation of $\|A^{m+1}\|$ and $\|A^{m+2}\|$, respectively, using the block 1-norm estimation algorithm of Higham and Tisseur [9]. Theorem 2 establishes that $\max\{\|A^k\|^{\frac{1}{k}} : k \geq l\}$ is obtained for $l \leq k \leq l+q-1$, where $1 \leq q \leq l$. If we use maximum Taylor order $m_M = 30$ for the computation of a matrix exponential then from Theorems 1 and 2 it follows that $l = m_M + 1 = 31$ and $l + q - 1 \leq 61$. For normal matrices, since $\|A^k\|_2 = \|A\|_2^k$, then $\max\{\|A^k\|_2^{\frac{1}{k}} : k \geq l\} = \max\{\|A\|_2^{\frac{k}{k}} : k \geq l\} = \|A\|_2$ and (16) will be a good approximation (it would be exact if we used the 2-norm and the exact values of $\|A^{m+1}\|_2$ and $\|A^{m+2}\|_2$). For the case of nonnormal matrices Fig. 1 shows the values $\{\|A^k\|_2^{1/k}\}_{k=1}^{61}$ for 103 matrices $A$ from the matrix exponential literature with $\|A\|_2 = 1$ and sizes from $2 \times 2$ to $100 \times 100$, see [5, p. 973]. Since $\|A^k\|^{1/k} \to \rho(A)$ as $k \to \infty$ [5, p. 972], in the majority of matrices the values $\|A^k\|^{1/k}$ tend to be decreasing and tend to have less variations for higher matrix powers, so (16) will be a good approximation for $\max\{\|A^k\|^{\frac{1}{k}} : k \geq m_M + 1\}$. In fact, for those nonnormal matrices with monotonically decreasing sequence $\|A^k\|^{\frac{1}{k}}$, $k = 1, 2, \ldots$ it follows that $\max\{\|A^k\|^{\frac{1}{k}} : k \geq m_M + 1\} = \|A^{m_M+1}\|^{\frac{1}{m_M+1}}$ and then the value $\alpha_{\min}$ from (16) gives the best possible selection. The fact of taking at least two matrix powers in (16) comes from the example matrix (3.8) from [8]

$$A = \begin{bmatrix} 1 & a \\ 0 & -1 \end{bmatrix}, \quad |a| \gg 1, \qquad \|A^{2k}\|_1 = 1, \qquad \|A^{2k+1}\|_1 = 1 + |a|, \tag{17}$$

where the norm of the odd powers is much greater than the norm of the even powers. Note that in this case

$$\max\{\|A^k\|^{\frac{1}{k}} : k \geq m+1\} = \max\{\|A^{m_M+1}\|^{\frac{1}{m_M+1}}, \|A^{m_M+2}\|^{\frac{1}{m_M+2}}\}, \tag{18}$$

and it is necessary to check at least two matrix powers to obtain the maximum. Numerical tests confirmed that using only two terms in (16) made no noticeable difference in the accuracy results in the majority of test matrices.

Once obtained $s_0$, if $s_0 \geq 1$ we check if (10) holds reducing the scaling $s = s_0 - 1$, and using the bounds for $\|A^k\| \leq a_k$ to test if bound

$$\frac{\|h_{m+1}(2^{-s}A)\|}{|c_{m+2}^{(m)}|} \leq \sum_{k=m+1}^{m+2} \left|\frac{c_k^{(m)}}{c_{m+2}^{(m)}}\right| \frac{a_k}{2^{sk}} \leq \max\{1, \|2^{-s}A\|\} \frac{u}{|c_{m+2}^{(m)}|}, \tag{19}$$

holds, truncating the series. Note that we will stop the series summation if after summing the first term, the sum is greater than $\max\{1, \|2^{-s}A\|\}u/|c_{m+2}^{(m)}|$. Table 2 presents some values of $c_k^{(m)}/c_{m+2}^{(m)}$, and the values $u/|c_{m+2}^{(m)}|$. In (19) we have simplified the process of determining the error bound with respect to those in [4,6], using only the two first terms of the error series, instead of the $q$ terms used in algorithms from [4,6]. Table 2 shows the maximal values $\hat{\Theta}_m$ such that $\tilde{h}_{m+1}(\hat{\Theta}_m) \leq \max\{1, \hat{\Theta}_m\}u$ using only the first 2 terms in series $\tilde{h}_{m+1}(\hat{\Theta}_m)$, and the values $\tilde{h}_{m+1}(\hat{\Theta}_m)/(\max\{1, \hat{\Theta}_m\}u)$ considering then 200 series terms. These values show that for normal matrices, considering only two series terms makes little relative difference, ($\leq$8.34) in the worst case, with respect to considering all the series terms.

For nonnormal matrices the value of the remaining series terms will depend on the values $\|A^k\|^{1/k}$, $k \geq m+3$ and their ratio with $\max\{\|A^{m+1}\|^{\frac{1}{m+1}}, \|A^{m+2}\|^{\frac{1}{m+2}}\}$. Given the results from Fig. 1, typically the terms of the power series of function $\|h_{m+1}(2^{-s}A)\|$ will be decreasing, so the first terms tend to determine the error bound.

Taking all the previous into account we decided to reduce the number of terms of the error series to be used and we checked empirically that using more terms rarely modified the final result. Again, taking into account matrix (17) we use the first two terms of the power series of $\|h_{m+1}(2^{-s}A)\|$.

In this step, we have also removed a complex and costly test that previous versions of the new Taylor algorithm do when expression (19) does not hold with $s = s_0 - 1$, see (15) from [4] and (45) from [6]. We have found empirically that when (19) does not hold, then very rarely those tests are satisfied.

In the next subsection, a detailed description of the new algorithm is given.

### 4.2. Taylor algorithm

The new proposed algorithm is presented in Algorithm 2. For all norms appearing in the scaling algorithm we use the 1-norm. The maximum allowed Taylor order, $m_M$, is an input parameter. In our experience, the optimal values for $m_M$ are 20, 25 or 30, increasing slightly the cost in tests with increasing $m_M$ but also improving the accuracy, see Fig. 1 and Table 2 of [4]. For clarity in the algorithm, we have considered that the maximum value for $m_M$ is 30. However, extending the algorithm to higher values for $m_M$ is straightforward.

In Steps 3–27, Algorithm 2 checks if any of the Taylor optimal orders $m_k = 1, 2, 4, \ldots, m_M$ satisfies (10) without scaling ($s = 0$), using the bounds provided in the previous section. As mentioned above, we compute the 1-norm estimate of $\|A^{m+1}\|$ and $\|A^{m+2}\|$ using the block 1-norm estimation algorithm of [9].

If no value of $m_k \leq m_M$ satisfies (10), the algorithm computes $\alpha_{\min}$ using only the 1-norm estimate of the matrix powers $\|A^{m+1}\|$ and $\|A^{m+2}\|$, and determines the initial scaling parameter $s_0$ in Steps 29–30. Then, if $s_0 > 0$, the algorithm checks in Steps 31–37 if the initial scaling parameter can be reduced, testing if (10) holds with $s = s_0 - 1$.

Then, in Steps 38–42, similarly to the algorithms proposed in [4,6], Algorithm 2 tests if (10) holds with $s$ and $m_{M-1}$; Taylor order $m = m_{M-1}$ will be used if (10) holds, or $m = m_M$ otherwise.

Finally, in Step 43 we use (7) to compute the exponential approximation of the scaled matrix, and in Steps 44–46 $s$ squaring steps are done to obtain the matrix exponential approximation of the original matrix $A$.

**Algorithm 2** Given a matrix $A \in \mathbb{C}^{n\times n}$ and a maximum order $m_M$, this algorithm computes $B = e^A$ by a Taylor approximation of order $m \leqslant m_M$.

**Inputs:** $A \in \mathbb{C}^{n\times n}$ and maximum order approximation $m_M$
**Output:** $B = e^A$

1: Set $\Theta_m$, $c_{m+1}^{(m)}/c_{m+2}^{(m)}$ and $u/|c_{m+2}^{(m)}|$ values from Table 2
2: $s \leftarrow 0$
3: **if** $||A|| < \Theta_1$ **then**
4: $m \leftarrow 1$
5: $B \leftarrow A + I_n$
6: quit
7: **end if**
8: **for** $m \in [2, 4, 6, \ldots, m_M]$ **do** ▷ Optimal values for $m$, from 2 to $m_M$
9: **if** $m = 2$ **then**
10: Compute and save $A^2$
11: **else if** $m = 6$ **then**
12: Compute and save $A^3$
13: **else if** $m = 12$ **then**
14: Compute and save $A^4$
15: **else if** $m = 20$ **then**
16: Compute and save $A^5$
17: **end if**
18: $b \leftarrow \max\{1, ||A||\} \cdot u/|c_{m+2}^{(m)}|$
19: $a(m+1) \leftarrow ||A^{m+1}||$ ▷ Estimate value of $||A^{m+1}||$
20: **if** $|c_{m+1}^{(m)}/c_{m+2}^{(m)}| \cdot a(m+1) \leq b$ **then**
21: $a(m+2) \leftarrow ||A^{m+2}||$ ▷ Estimate value of $||A^{m+2}||$
22: **if** $|c_{m+1}^{(m)}/c_{m+2}^{(m)}| \cdot a(m+1) + a(m+2) \leq b$ **then**
23: $B \leftarrow T_m(A)$ ▷ Evaluate $T_m(A)$ using (7) with $s = 0$
24: quit
25: **end if**
26: **end if**
27: **end for**
28: $m \leftarrow m_M$ ▷ Maximum order selected
29: $\alpha_{min} \leftarrow \max\{a(m+1)^{1/(m+1)}, a(m+2)^{1/(m+2)}\}$
30: $s_0 \leftarrow \lceil \log_2(\alpha_{min}/\Theta_m) \rceil$
31: **if** $s_0 > 0$ **then** ▷ Check if (10) holds reducing the scaling $s = s_0 - 1$
32: $s \leftarrow s_0 - 1$
33: $b \leftarrow \max\{1, ||A||/2^s)\}u/|c_{m+2}^{(m)}|$
34: **if** $|c_{m+1}^{(m)}/c_{m+2}^{(m)}| \cdot a(m+1)/2^{(m+1)s} + a(m+2)/2^{(m+2)s} > b$ **then**
35: $s \leftarrow s_0$ ▷ (10) does not hold, then $s = s_0$
36: **end if**
37: **end if**
38: $m \leftarrow m_{M-1}$ ▷ Test if scaled matrix allows using $m_{M-1}$
39: $b \leftarrow \max\{1, ||A||/2^s\} \cdot u/|c_{m+2}^{(m)}|$
40: **if** $|c_{m+1}^{(m)}/c_{m+2}^{(m)}| \cdot a(m+1)/2^{(m+1)s} + a(m+2)/2^{(m+2)s} > b$ **then**
41: $m \leftarrow m_M$ ▷ Scaled matrix does not allow using $m_{M-1}$, then $m = m_M$
42: **end if**
43: Compute $B = T_m(A/2^s)$ ▷ Evaluate $T_m(2^{-s}A)$ using (7)
44: **for** $i = 1 : s$ **do** ▷ Squaring phase
45: $B = B^2$
46: **end for**

## 5. Numerical experiments and conclusions

In this section we compare a Matlab implementation of the new algorithm, denoted by **exptaynsv3**, with the functions **exptayns** and **exptaynsv2** from [4,6], respectively, and also a Fortran version of **exptaynsv3** with one of the main commercial software available for computing matrix exponentials: the NAG library [7]. The four functions are available at:

http://personales.upv.es/jorsasma/Software/exptayns.m
http://personales.upv.es/jorsasma/Software/exptaynsv2.m
http://personales.upv.es/jorsasma/Software/exptaynsv3.m
http://personales.upv.es/jorsasma/Software/exptaynsv3fortran.zip.

We have also included a comparison of **exptaynsv3** with MATLAB function **expm_new** from [5], that implements a scaling squaring Padé algorithm to compute matrix exponential. The accuracy was tested by computing the relative error

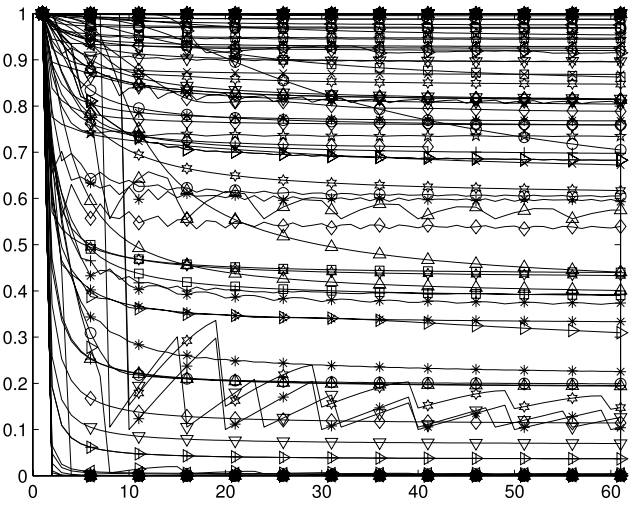


**Fig. 1.** $\{\|A^k\|_2^{1/k}\}_{k=1}^{61}$ for 103 matrices $A$ with $\|A\|_2 = 1$ and sizes from $2 \times 2$ to $100 \times 100$.

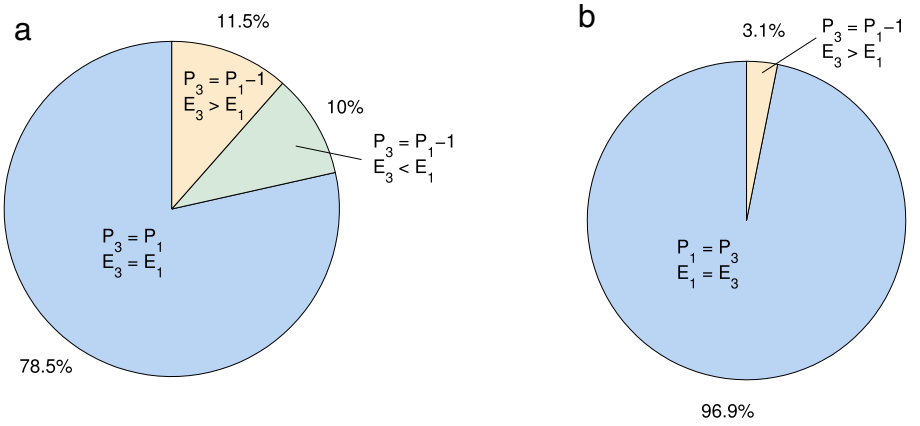


**Fig. 2.** Comparison of the cost in terms of matrix products ($P$) and the relative error ($E$) between `expta ynsv3` ($P_3$ and $E_3$) and `exptayns` ($P_1$ and $E_1$) with matrix sets 1 and 2 (a), and matrix set 3 (b).

$\mathrm{E} = \|e^A - \tilde{X}\|_1 / \left\|e^A\right\|_1$, where $\tilde{X}$ is the computed approximation and the cost is given in terms of matrix products. We used the following sets of matrices for testing:

1. One hundred diagonalizable matrices of size 1024. These matrices have the form $V^T D V$, where $D$ is a diagonal matrix whose diagonal elements are random values between $-k$ and $k$ with different integer values of $k$, and $V$ is an orthogonal matrix obtained as $V = H/16$, where $H$ is the Hadamard matrix.
2. One hundred matrices with multiple eigenvalues of size 1000. These matrices have the form $V^T D V$, where $D$ is a block diagonal matrix whose diagonal blocks are Jordan blocks with random dimension and random eigenvalues between $-50$ and 50, and $V$ is an invertible matrix with random values in $[-0.5, 0.5]$.
3. 32 matrices $1000 \times 1000$ from the function `matrix` from the Matrix Computation Toolbox [10]. Matrices whose exponential cannot be represented in double precision due to overflow were excluded from all the matrices given by function `matrix`. These matrices appear in the state of the art in the exponential matrix computation [5,11].

The "exact" value of matrix exponential for matrix sets 1 and 2 was computed by using transformations $e^A = V^T e^D V$, where $V^T e^D V$ was computed using `vpa` function from Matlab's Symbolic Math Toolbox with 32 decimal digit precision. For matrix set 3, we used quadruple precision Taylor algorithm in Fortran with different orders and scaling parameters for each matrix to check the result correctness. The maximum order used for Taylor approximation in all cases was $m_M = 30$.

Fig. 2 presents the comparison of functions `exptaynsv3` and `exptayns` in terms of matrix products and relative errors. The new algorithm saved one matrix product in 21.5% of cases from the matrix sets 1 and 2, and one matrix product in 3.1% of cases from the matrix set 3. Both algorithms achieved very similar accuracy results, with $\max(|E_1 - E_3|/|E_1|) = 0.0194$ for matrix sets 1 and 2, and $\max(|E_1 - E_3|/|E_1|) = 0.0038$ for matrix set 3.

Similarly, Fig. 3 shows the results of the comparison between the functions `exptaynsv3` and `exptaynsv2`. In this case considering matrix sets 1 and 2, the new algorithm saved one matrix product in 18.5% of cases. With respect to matrix set

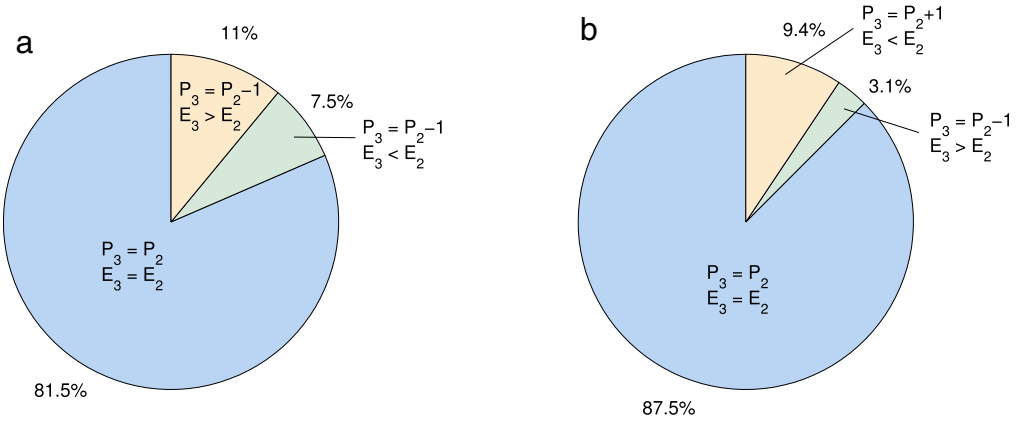


**Fig. 3.** Comparison of the cost in terms of matrix products ($P$) and the relative error ($E$) between `exptaynsv3` ($P_3$ and $E_3$) and `exptaynsv2` ($P_1$ and $E_1$) with matrix sets 1 and 2 (a), and matrix set 3 (b).

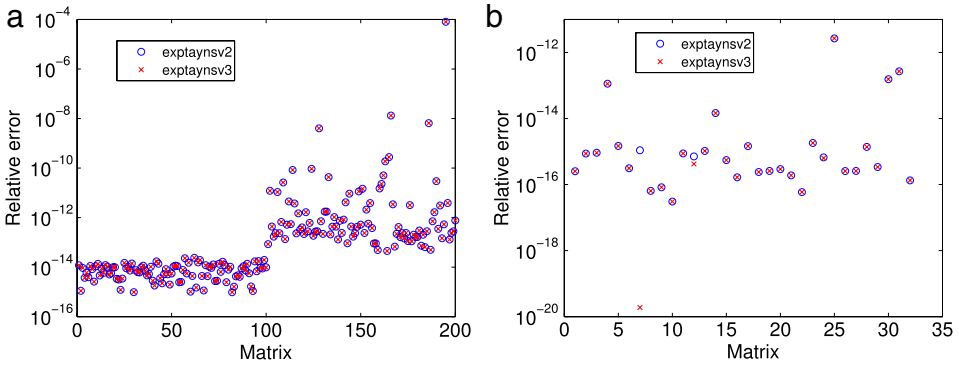


**Fig. 4.** Relative error comparison between `exptaynsv3` and `exptaynsv2` with matrix sets 1 and 2 (a) and matrix set 3 (b). Taylor maximum order $m_M = 30$.

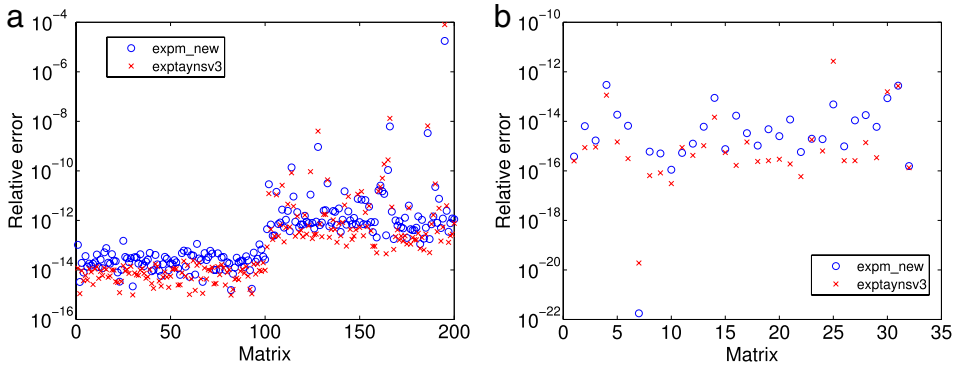


**Fig. 5.** Relative error comparison between `exptaynsv3` and `expm_new` with matrix sets 1 and 2 (a) and matrix set 3 (b). Taylor maximum order $m_M = 30$.

3, it saved one matrix product in 3.1% of cases, whereas it performed one more matrix product in 9.4% of cases. In terms of accuracy, see Fig. 4, results are very similar to those obtained in the previous case when considering matrix sets 1 and 2, obtaining $\max(|E_2 - E_3|/|E_2|) = 0.0194$. Considering matrix set 3, see Fig. 4(b), the only case where the error difference is significant is for matrix 7, where `exptaynsv2` gives a $10^{-15}$ order accuracy, while `exptaynsv3` obtains a higher accuracy of order $10^{-20}$. For the remaining cases $\max(|E_2 - E_3|/|E_2|) = 0.41$.

The accuracy comparison between the functions `exptaynsv3` and `expm_new` is presented in Fig. 5. Function `exptaynsv3` achieved a higher accuracy in 88.5% of the matrices from matrix sets 1 and 2, and in 87.5% of the matrices from matrix set 3. The number of matrix products performed by `exptaynsv3` to compute all the matrix exponentials of matrix sets 1, 2 and 3 were 1115, 700 and 351, respectively, whilst `expm_new` carried out 1338, 1596 and 345.6 matrix products, respectively.

**Table 3**
Execution time comparison in seconds between the functions `exptayns`, `exptaynsv2` and `exptaynsv3` in Matlab.

| | `exptayns` | `exptaynsv2` | `exptaynsv3` |
|---|---|---|---|
| Matrix set 1 | 262.83 | 266.94 | 227.69 |
| Matrix set 2 | 178.25 | 179.54 | 136.37 |
| Matrix set 3 | 171.93 | 170.81 | 165.47 |

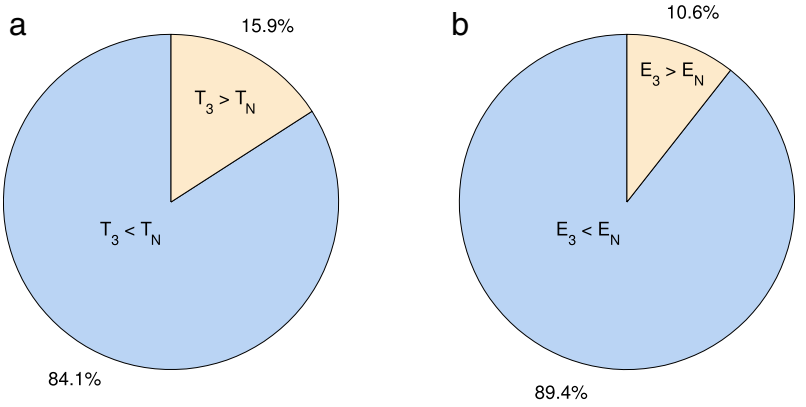


**Fig. 6.** Cost in seconds ($T$) (a) and relative error comparison ($E$) (b) between `exptaynsv3` ($T_3$ and $E_3$) and `f01ecc` from NAG Library ($T_N$ and $E_N$) with matrix sets 2 and 3.

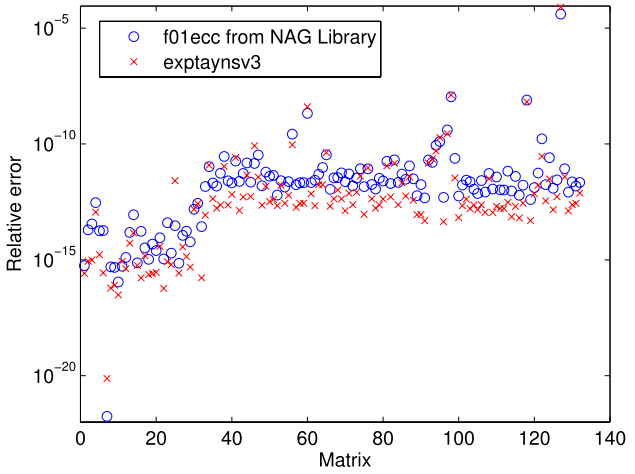


**Fig. 7.** Relative error comparison between `exptaynsv3` and function `f01ecc` from NAG Software.

Although matrix set 3 includes some ill-conditioned matrices, we found interesting to check the behaviour of the functions with a very nonnormal and ill-conditioned matrix, such as the one generated by the command `gallery('triw', 20,4.1)` of Matlab [8]. We computed the "exact" exponential of this matrix using `vpa` Matlab's function with 500 decimal digit precision and the relative error obtained was $1.8163 \cdot 10^{-16}$ for the three Taylor functions and $1.1408 \cdot 10^{-15}$ for `expm_new`. The number of matrix products performed were 10 for the Taylor functions and 7.3 for `expm_new`.

We have also included an execution time comparative of the three Taylor functions in Matlab. Although execution time in Matlab is not always reliable, since the three functions are similar the results can be useful. Table 3 shows the total time in seconds taken by each function to compute all matrix exponentials of each matrix set. As shown, `exptaynsv3` computes the exponential faster than the other two previous versions in all cases.

Finally, we have compared a Fortran version of the new algorithm with one of the main commercial software packages that allows the computation of matrix exponentials. The function `f01ecc` from NAG Library, see [7], computes the matrix exponential of a real square matrix, using the algorithm based on Padé approximants and the scaling and squaring method described in [11,2].

The tests have been done in a Linux system with an Intel processor, using the Math Kernel Libraries from Intel. We have used the 32 matrices from Matlab Toolbox and the 100 random Jordan matrices from previous tests. Results are shown in Figs. 6 and 7. In this case, execution time instead of matrix products was used to evaluate the cost of both functions, and

function `exptaynsv3` obtained better results than the NAG routine in both accuracy, 89.4% of cases, and execution time, 84.1% of cases. The total time taken by `exptaynsv3` to compute all matrix exponentials was 521 s, versus 866 s taken by `f01ecc` function. Moreover, `exptaynsv3` was significantly more accurate in the majority of cases, see Fig. 7.

## 6. Conclusions

A competitive modification of the Taylor algorithm from [4] has been proposed based on a simplification of the calculation of the error bounds used to select the order of the approximation and the scaling parameter. These modifications were based on theoretical results for normal matrices and empirical results for nonnormal matrices, leading to a new simplified algorithm that obtained similar accuracy as previous algorithms from the authors in the majority of test matrices with a lower processing time, and also better results than state-of-the-art algorithms based on Padé approximations.

## References

[1] 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.
[2] N.J. Higham, Functions of Matrices: Theory and Computation, Society for Industrial and Applied Mathematics, Philadelphia, PA, USA, 2008.
[3] 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, http://dx.doi.org/10.1137/090763202.
[4] J. Sastre, J. Ibáñez, E. Defez, P. Ruiz, Accurate matrix exponential computation to solve coupled differential models in engineering, Math. Comput. Model. 54 (2011) 1835–1840.
[5] 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.
[6] J. Sastre, J. Ibáñez, E. Defez, P. Ruiz, Accurate and efficient matrix exponential computation, Int. J. Comput. Math. 91 (1) (2014) 97–112.
[7] NAG Library Function Document, http://www.nag.co.uk/numeric/cl/nagdoc_cl23/html/F01/f01ecc.html.
[8] A.H. Al-Mohy, N.J. Higham, Computing the action of the matrix exponential, with an application to exponential integrators, SIAM J. Sci. Comput. 33 (2) (2011) 488–511.
[9] J. Higham, F. Tisseur, A block algorithm for matrix 1-norm estimation, with an application to 1-norm pseudospectra, SIAM J. Matrix Anal. Appl. 21 (2000) 1185–1201.
[10] N.J. Higham, The Matrix Computation Toolbox, http://www.ma.man.ac.uk/~higham/mctoolbox/.
[11] N.J. Higham, The scaling and squaring method for the matrix exponential revisited, SIAM J. Matrix Anal. Appl. 26 (4) (2005) 1179–1193.