Elsevier Editorial System(tm) for Journal of Computational and Applied Mathematics
Manuscript Draft

Manuscript Number: CAM-D-18-00207R1

Title: On-the-fly backward error estimate for matrix exponential approximation by Taylor algorithm

Article Type: Research Paper

Section/Category: 41-xx (Approximations and expansions)

Keywords: Backward error estimate for the matrix exponential; overscaling; multiple precision floating point computation; scaling and squaring algorith; Paterson--Stockmeyer evaluation scheme; truncated Taylor series; Krylov methods

Corresponding Author: Professor Marco Caliari, PhD

Corresponding Author's Institution: University of Verona

First Author: Marco Caliari, PhD

Order of Authors: Marco Caliari, PhD; Franco Zivcovich

Abstract: In this paper we show that it is possible to estimate the backward error for the approximation of the matrix exponential \emph{on-the-fly}, without the need to precompute in high precision quantities related to specific accuracies. In this way, the scaling parameter s and the degree m of the truncated Taylor series (the underlying method) are adapted to the input matrix and not to a class of matrices sharing some spectral properties, such as with the classical backward error analysis.
The result is a very flexible method which can approximate the matrix exponential at any desired accuracy. Moreover, the risk of overscaling, that is the possibility to select a value s larger than necessary, is mitigated by a new choice as the sum of two powers of two. Finally, several numerical experiments in \textsc{MATLAB} with data in double and variable precision and with different
tolerances confirm that the method is accurate and
often faster than available good alternatives.

On-the-fly backward error estimate for matrix exponential approximation by Taylor algorithm

M. Caliari[a,*], F. Zivcovich[b]

[a] *Department of Computer Science, University of Verona, Italy*
[b] *Department of Mathematics, University of Trento, Italy*

Abstract

In this paper we show that it is possible to estimate the backward error for the approximation of the matrix exponential *on-the-fly*, without the need to precompute in high precision quantities related to specific accuracies. In this way, the scaling parameter s and the degree m of the truncated Taylor series (the underlying method) are adapted to the input matrix and not to a class of matrices sharing some spectral properties, such as with the classical backward error analysis. The result is a very flexible method which can approximate the matrix exponential at any desired accuracy. Moreover, the risk of overscaling, that is the possibility to select a value s larger than necessary, is mitigated by a new choice as the sum of two powers of two. Finally, several numerical experiments in MATLAB with data in double and variable precision and with different tolerances confirm that the method is accurate and often faster than available good alternatives.

Keywords: Backward error estimate for the matrix exponential, overscaling, multiple precision floating point computation, scaling and squaring algorithm, Paterson–Stockmeyer evaluation scheme, truncated Taylor series, Krylov methods

1. Introduction

Among all the ways to approximate the matrix exponential (see [1]), the scaling and squaring algorithm applied to a basic method such as Padé rational approximation is the most popular. An effective choice for the order of approximation and the scaling parameter is possible thanks to the backward error analysis introduced in [2] and successively refined in [3, 4]. The backward error analysis needs some precomputation in high precision which have to be performed once and for all for each kind of approximation and desired

*Corresponding author
Email addresses: marco.caliari@univr.it (M. Caliari), franco.zivcovich@unitn.it (F. Zivcovich)

accuracy (usually only double precision). This precomputation can be done in advanced since it overestimates the relative backward error in the worst case scenario. As a result, one gets some general constraints for the magnitude of the input matrices which guarantee a satisfying approximation. A SageMath code[1] for performing the backward error analysis for a wide class of approximations have been developed by the authors of the present manuscript together with P. Kandolf.

The Padé rational approximation is not the only possible basic method. For instance, in [5–9] the basic approximation has been replaced by a truncated Taylor series, computed in an efficient way through the Paterson–Stockmeyer scheme. In particular, the Taylor algorithms in [5, 6, 9] use a combination of forward and backward relative error analysis, and [7] is based on a combination of Taylor and Hermite matrix polynomials. Truncated Taylor series and other polynomial approximations (see [10–12]) have been proposed for the action of the exponential matrix to a vector, too.

In this paper, we use as a basic method the truncated Taylor series T_m about 0. It is generally good practice to shift the matrix A by

$$\mu I = \frac{\operatorname{trace}(A)}{n} I, \quad A \in \mathbb{C}^{n\times n}$$

where I is the identity matrix. In this way, the eigenvalues of the resulting matrix $B = A - \mu I$ have zero sum and their convex hull includes the origin. For the same reason, the quality of this approximation degrades when the norm of B grows bigger, hence a scaling algorithm has to be taken into account. We have to select a scaling parameter s such that the matrix power

$$\mathrm{e}^{\mu} \left(T_m(s^{-1}B)\right)^s \tag{1}$$

accurately approximates the matrix exponential of A for a given m.

The main novelty of this paper consists in determing the degree of approximation m and the scaling parameter s by an on-the-fly estimate of the backward error ΔB in the identity

$$\mathrm{e}^{\mu} \left(T_m(s^{-1}B)\right)^s = \exp(A + \Delta B)$$

without any precomputation. The neat advantage of this approach is that we get in general a sharper overestimate of the backward error, which is based on the input matrix itself and cannot be larger than the worst case. This approach is by far more flexible, since it allows the user to ask for any accuracy and not only precomputed ones.

In Section 2 the choice of m is narrowed down to values with specific properties that make them particularly cheap to use when it comes to evaluating the approximation. These values stem from a cost analysis of the Paterson–Stockmeyer evaluation scheme for polynomials. In Section 3 we take care of the

[1] Available at `https://bitbucket.org/expleja/expbea`.

choice of the candidate scaling parameter s. We employ few values $\|B^j\|^{1/j}$ in order to estimate the spectral radius of B and thus determine s in an original manner. Furthermore, we explain how s can be chosen not only classically as a power of two but also as the sum of two powers of two in order to reduce the risk of *overscaling*. In Section 4 we show how to estimate the backward error for the given input matrix A. Moreover, it is explained in details the algorithm of selection of the parameters m and s. The objective is to grant the required accuracy minimizing the cost in term of matrix products. A possible reduction of the scaling parameter s is also discussed. In Section 6 we report the numerical experiments we conducted and, finally, we draw the conclusions in Section 7.

2. Paterson–Stockmeyer evaluation scheme

The Taylor series for the exponential function is a convergent series, hence the more the degree m at which we truncate is large the more accurate will be the approximation (at least in exact arithmetic). For the matrix exponential, the cost of evaluating powers of matrices has to be taken into account, too. Therefore, our goal is to maximize the reachable interpolation degree m with a given number MP of matrix products. The Horner evaluation scheme would lead us to reach m with $\mathrm{MP} = m - 1$ matrix products and so, for a given MP, we would merely use $m = \mathrm{MP} + 1$. By means of a regrouping of the terms in the series, we can aim to better performances. For a given square matrix $X \in \mathbb{C}^{n\times n}$, and z positive integer, we consider the rewriting

$$T_m(X) = \sum_{i=0}^{m} \frac{X^i}{i!} = I + \sum_{k=0}^{r} (X^z)^k P_k, \quad r = \left\lfloor \frac{m}{z} \right\rfloor$$

where

$$P_k = \begin{cases} \displaystyle\sum_{i=1}^{z} \frac{X^i}{(zk+i)!}, & k = 0, 1, \ldots, r-1, \\ \displaystyle\sum_{i=1}^{m-zk} \frac{X^i}{(zk+i)!}, & k = r. \end{cases}$$

This is the Paterson–Stockmeyer evaluation scheme applied to Taylor truncated series. It was used for degree $m \leq 30$ in [6]. The first $z-1$ powers $X^2, \ldots, X^z$ of X have to be computed and stored. A variant requiring only $3n^2$ storage, independently of z, is reported in [13]. Then, the P_k matrices are iteratively computed for $k = r, r-1, \ldots, 0$ and used in the Horner scheme in order to assemble the final matrix polynomial

$$T_m(X) = (((X^z P_r + P_{r-1})X^z + P_{r-2})X^z + \ldots + P_1)X^z + P_0 + I.$$

Taking into account that P_r is the null matrix if z divides m, the total number MP of matrix products needed to evaluate $T_m(X)$ with this regrouping turns out to be

$$\mathrm{MP} = z + \left\lceil \frac{m}{z} \right\rceil - 2.$$

	$z=1$	$z=2$	$z=3$	$z=4$	$z=5$	$z=6$
$m=1$	0					
$m=2$	1	1				
$m=3$	2	2	2			
$m=4$	3	2	3	3		
$m=5$	4	3	3	4	4	
$m=6$	5	3	3	4	5	5
$m=7$	6	4	4	4	5	6
$m=8$	7	4	4	4	5	6
$m=9$	8	5	4	5	5	6
$m=10$	9	5	5	5	5	6
$m=11$	10	6	5	5	6	6
$m=12$	11	6	5	5	6	6
$m=13$	12	7	6	6	6	7
$m=14$	13	7	6	6	6	7
$m=15$	14	8	6	6	6	7
$m=16$	15	8	7	6	7	7
$m=17$	16	9	7	7	7	7
$m=18$	17	9	7	7	7	7
$m=19$	18	10	8	7	7	8
$m=20$	19	10	8	7	7	8
$m=21$	20	11	8	8	8	8
$m=22$	21	11	9	8	8	8
$m=23$	22	12	9	8	8	8
$m=24$	23	12	9	8	8	8
$m=25$	24	13	10	9	8	9

Table 1: Paterson–Stockmeyer costs for combinations of m and z. The entry on the m-th row and z-th column equals $\mathrm{MP} = z + \lceil \frac{m}{z} \rceil - 2$.

As an example, the costs relative to all the combinations of m and z for $1 \leq m \leq 25$ and $1 \leq z \leq \min(m, 6)$ are listed in Table 1. As it can be seen in the table and already done in [6], it is possible to detect those values of m that are maximal for a given number of products MP. These maxima are unique but not uniquely determined: in case of two different values of z leading to the same interpolation degree m at the same evaluation cost, we pick the larger for reasons that will be clear in the next section. Finally, we notice that the values of z of interest always correspond to $\lceil \sqrt{m} \rceil$. Costs leading to maximal m and z are highlighted in gray inside Table 1. The corresponding values for z and m are given by

$$z = \left\lceil \frac{\mathrm{MP}}{2} \right\rceil + 1, \quad m = (\mathrm{MP} - z + 2)z. \tag{2}$$

Notice indeed that this choice of m and z implies that m/z is an integer number. It goes with it that these combinations of (m, z) are the ones that we set to be

admissible. We will denote by $(m_{\mathrm{MP}}, z_{\mathrm{MP}})$ the pair requiring MP $= z_{\mathrm{MP}} + m_{\mathrm{MP}}/z_{\mathrm{MP}} - 2$ matrix products. In the next sections we will introduce a greedy algorithm for the selection of a proper pair among the admissible ones which keeps MP as small as possible and guarantees a prescribed accuracy. Starting with MP $= 2$, the algorithm will increase MP until it finds the wanted pair. Only then, the corresponding Paterson–Stockmeyer scheme will be evaluated.

3. Scaling and shifting algorithm

As already mentioned, it is good practice (see [4, 10]) to shift the original matrix A by

$$\mu I = \frac{\mathrm{trace}(A)}{n} I. \tag{3}$$

On the other hand, the computation in finite arithmetic of

$$\mathrm{e}^{\mu} \left(T_m(s^{-1}B)\right)^s, \quad B = A - \mu I$$

can be problematic if A has large negative eigenvalues, since it requires the explicit approximation of $\exp(A-\mu I)$. Such a quantity could overflow. Therefore, in the practical implementation we apply the shifting strategy in the following way

$$\exp(A) \approx \begin{cases} \mathrm{e}^{\mu} \left(T_m(s^{-1}B)\right)^s, & \text{if } \Re(\mu) \geq 0 \\ \left(\mathrm{e}^{\mu/s} T_m(s^{-1}B)\right)^s, & \text{otherwise.} \end{cases}$$

The truncated Taylor series cannot be applied, in general, to the shifted matrix B, whose norm can be arbitrarily large. In fact, the matrix has to be scaled by s^{-1}, s a positive integer. In this section we are going to describe how to make a first selection of the scaling parameter s. Our strategy consists in fixing a positive value $\rho_{\max}$ such that we scale B if its spectral radius $\rho(B)$ is not smaller or equal to $\rho_{\max}$. Since $\rho(B)$ is in general not available, we work with an overestimate which we can update as soon as we have powers of B available. The value $\rho_{\max}$ depends uniquely on considerations of the user. The default choice is 3.5 for data in double or higher precision and 6.3 for data in single precision (see Remark 1). On the other hand, we have to be careful not to pick s too large, or a phenomenon called *overscaling* will destructively kick in, making the accuracy to drop. The overscaling phenomenon, first identified in [14], can be simply illustrated by considering the two approximations $1 + x$ and $(1 + x/100)^{100}$ to e^x for $x = 2^{-27}$: in double precision, we have $\mathrm{e}^x = 1 + x$, while $\mathrm{e}^x - (1 + x/100)^{100} \approx 7.1 \cdot 10^{-15}$. Last but not least, we also desire the scaling strategy to require few matrix products when it comes to recover the wanted approximation from the scaled one.

Since the parameter z_{MP} does not decrease while MP increases, at each step of the greedy algorithm we compute, if needed, the new power of B, keeping the previous ones. By means of them, we can derive information on the spectral

radius of B. In fact, we can easily compute the vector $\vec{\rho} = (\rho_1, \rho_2, \ldots, \rho_{z_{\mathrm{MP}}})$ whose components are $\rho_j = \|B^j\|^{1/j}$. We know that

$$\rho_j \geq \rho(B)$$

and, thanks to Gelfand's formula,

$$\lim_{j\to\infty} \rho_j = \rho(B).$$

We choose the 1-norm in order to compute the values ρ_j. Therefore, the smallest component of $\vec{\rho}$, denoted by $\underline{\rho}$ is the best known overestimate of the spectral radius of B. Then we have to pick s large enough to grant

$$\rho(s^{-1}B) \leq s^{-1}\underline{\rho} = s^{-1} \min_{j=1,\ldots,z_{\mathrm{MP}}} \{\|B^j\|^{\frac{1}{j}}\} \leq \rho_{\max}. \tag{4}$$

Naturally, the larger is z_{MP}, the smaller s may be chosen. This is the reason why we picked z as the largest integer minimizing the cost of Paterson–Stockmeyer scheme. We could hence select s as the smallest power of two for which the inequality holds true. Doing so, we would face a minimal cost when it comes to the powering part in (1). In fact, using the identity

$$\exp\left(\frac{2A}{s}\right) = \exp\left(\frac{A}{s}\right)\exp\left(\frac{A}{s}\right),$$

the recovering of the wanted approximation of $\exp(A)$ costs merely $\log_2(s)$ matrix products. This is the so called *scaling and squaring* algorithm. A weakness of this approach is that the scaling parameter could still be larger than necessary. Let us suppose in fact that, e.g., $\underline{\rho} = (2^l + 1)\rho_{\max}$. The choice of taking s as a power of 2 would lead us to pick $s = 2^{l+1}$, a number about double than the necessary when l is large. Hence, the new idea is to determine the smallest scaling parameter s as a sum of powers of two in such a way that

$$s^{-1}\underline{\rho} = (2^p + 2^q)^{-1}\,\underline{\rho} \leq \rho_{\max}, \tag{5}$$

with $p \geq 0$ and $q \in \{-\infty\} \cup [0, p)$, where we mean $2^{-\infty} = 0$. This possibly leads to sensibly smaller scaling parameters in general: in the above example the scaling parameter chosen in such a way would have been exactly $2^l + 2^0$. For what concerns the powering part, it is possible to recover the wanted approximation of $\exp(A)$ using exactly the same number of matrix products required by the single power of two choice. In fact, let 2^{l+1} be the smallest power of two such that $2^{-(l+1)}\underline{\rho} \leq \rho_{\max}$ and $2^p + 2^q$, with q as above, the smallest number in this form such that $(2^p + 2^q)^{-1}\underline{\rho} \leq \rho_{\max}$. If $q = -\infty$, then $p = l + 1$ and the squaring procedure requires exactly $p = l + 1$ matrix products. Otherwise, $0 \leq q < p$, $p = l$, and

$$\exp\left(\frac{A}{2^p + 2^q}\right)^{2^p+2^q} = \exp\left(\frac{A}{2^p + 2^q}\right)^{2^q} \exp\left(\frac{A}{2^p + 2^q}\right)^{2^p} = E_q E_p.$$

The evaluation of the powering phase of E_q requires q matrix products and the evaluation of E_p, taking into account that E_q has been already evaluated, costs $p-q$ matrix products. With the additional product between E_q and E_p, we have $p+1 = l+1 = \lceil \log_2(2^p + 2^q) \rceil$ matrix products. Once the parameter s has been determined, we will indicate by $(m_{\mathrm{MP}}, z_{\mathrm{MP}}, s_{\mathrm{MP}})$ the candidate triple of parameters to be used to approximate the matrix exponential.

4. Backward error analysis

We need now to equip with a tool that determines if the approximation stemming from the choice of the scaling parameter s and the interpolation degree m is accurate up to a certain tolerance tol. To do so, we employ the backward error analysis tool introduced by [10] on $T_m(s^{-1}B)^s$. The idea is to suppose that we are computing exactly the matrix exponential of a slightly perturbed matrix. Now, since

$$\mathrm{e}^{\mu}\left(T_m(s^{-1}B)\right)^s = \mathrm{e}^{\mu}\exp(B+\Delta B) = \exp(A+\Delta B),$$

we consider ourselves satisfied whenever the relative backward error does not exceed tol, that is

$$\|\Delta B\| \leq \mathrm{tol}\cdot\|A\|. \tag{6}$$

To this aim we can represent ΔB as a function of B by exploiting the relation

$$\begin{aligned}\exp(B+\Delta B) &= T_m(s^{-1}B)^s \\ &= \exp(B)\exp(-B)T_m(s^{-1}B)^s \\ &= \exp\left(B + s\log\left(\exp\left(-s^{-1}B\right)T_m(s^{-1}B)\right)\right)\end{aligned}$$

and by setting

$$h_{m+1}(s^{-1}B) := \log(\exp\left(-s^{-1}B\right)T_m(s^{-1}B)) = s^{-1}\Delta B. \tag{7}$$

Let's now investigate the relation between B and ΔB in more depth: for every X in the set

$$\Omega = \{X \in \mathbb{C}^{n\times n}\colon \rho(\exp(-X)T_m(X) - I) < 1\})$$

the function $h_{m+1}(X)$ has the power series expansion

$$h_{m+1}(X) = \sum_{k=0}^{\infty} c_{k,m}X^k. \tag{8}$$

In order to derive the coefficients $c_{k,m}$, we exploit the equivalence

$$\exp(-X)T_m(X) = \exp(-X)(\exp(X) - R_m(X)) = I - \exp(-X)R_m(X)$$

where

$$R_m(X) = \sum_{i=m+1}^{\infty}\frac{X^i}{i!}.$$

We are able now to represent the backward error by means of the series

$$h_{m+1}(X) = \log(\exp(-X)T_m(X)) =$$
$$= \log(I - \exp(-X)R_m(X)) = -\sum_{j=1}^{\infty} \frac{(\exp(-X)R_m(X))^j}{j}.$$

In the same way as it has been done in [9], it is possible now to obtain the series representation of $\exp(-X)R_m(X)$ by convolving the coefficients of its factors.

In fact, one can check that

$$\exp(-X)R_m(X) = \sum_{p=1}^{\infty} \left(\sum_{i=0}^{p-1} \frac{(-1)^i}{(m+p-i)!i!} \right) X^{m+p} =$$
$$= \sum_{p=1}^{\infty} \frac{(-1)^{p-1}}{(p-1)!m!(m+p)} X^{m+p} = \sum_{k=0}^{\infty} b_{k,m} X^k$$

where the coefficients $b_{k,m}$ are given by

$$b_{k,m} = \begin{cases} 0, & \text{if } 0 \le k \le m, \\ \dfrac{(-1)^{k-m-1}}{(k-m-1)!m!k!}, & \text{if } k \ge m+1 \end{cases} \tag{9}$$

(see [9, formula (22)])[2]. Therefore, any positive power j of $\exp(-X)R_m(X)$ can be expanded into a power series starting at degree $j(m+1)$. Combining these power series in order to expand the logarithm, we can obtain the coefficients $c_{k,m}$ of (8) up to any degree. Let us stress that the first $m+1$ coefficients of the series expansion of h_{m+1} are hence equal to 0 and $b_{k,m} = c_{k,m}$ for $k = 0, 1, \ldots, 2m+1$, as already noticed in [9, formula (15)]. For those values of s such that

$$s^{-1}B \in \Omega, \tag{10a}$$

by applying the triangular inequality we obtain

$$s^{-1}\|\Delta B\| = \left\| \sum_{k=m+1}^{\infty} c_{k,m} s^{-k} B^k \right\| \le \hat{h}_{m+1,z}(s^{-1}B)$$

where

$$\hat{h}_{m+1,z}(s^{-1}B) := \sum_{j=m/z}^{\infty} \epsilon_{j-m/z}, \quad \epsilon_{j-m/z} := \left\| \left(s^{-z}B^z\right)^j \sum_{i=1}^{z} c_{jz+i,m} s^{-i} B^i \right\|.$$

Such a representation for $\hat{h}_{m+1,z}(s^{-1}B)$ holds true for any (m, z) such that z divides m, hence in particular for any admissible pair. Therefore, requirement (6) holds true as long as m is a positive integer large enough to achieve

$$\hat{h}_{m+1,z}(s^{-1}B) \le \mathrm{tol} \cdot s^{-1}\|A\|. \tag{10b}$$

[2]The closed form for the coefficients has been independently derived also in [15].

Notice that due to triangular inequality we have the following chain of inequalities

$$\|h_{m+1}(s^{-1}B)\| \leq \hat{h}_{m+1,z}(s^{-1}B) \leq \hat{h}_{m+1,1}(s^{-1}B) \leq \tilde{h}_{m+1}(\|s^{-1}B\|) \tag{11}$$

where

$$\tilde{h}_{m+1}(x) := \sum_{k=m+1}^{\infty} |c_{k,m}|x^k$$

is the function to estimate the backward error commonly employed in the literature [2, 5, 7, 8, 10].

Remark 1. The classical backward error analysis [2] proceeds here by considering

$$\frac{\|\Delta B\|}{\|B\|} = \frac{\|h_{m+1}(s^{-1}B)\|}{\|s^{-1}B\|} \leq \frac{\tilde{h}_{m+1}(s^{-1}\|B\|)}{s^{-1}\|B\|} \tag{12}$$

and thus looking for the largest scalar parameter θ_m such that

$$\frac{\tilde{h}_{m+1}(\theta_m)}{\theta_m} \leq \text{tol}.$$

This computation is usually done in high precision arithmetic once and for all (for each degree m and selected tolerances). For instance, for $m = 30$ and $\text{tol} = 2^{-53}, 2^{-24}$ we have $\theta_{30} \approx 3.54, 6.32$, respectively. We took an approximation of these two values as default values $\rho_{\max}$, and our numerical experience indicates that these are appropriate choices. Once the values θ_m have been computed, the given matrix $B = A - \mu I$ is properly scaled in such a way that $\|s^{-1}B\| = s^{-1}\|B\| \leq \theta_m$. The choice of the actual degree of approximation m and the related value θ_m is usually based on the minimization of the computational effort. The result is a scaling parameter s such that

$$\|\Delta B\| \leq \text{tol} \cdot \|B\|.$$

Since the shift strategy is applied in order to have $\|B\| \leq \|A\|$, such a result is more stringent (and possibly requires more computational work) than (6).

As a final consideration, we remark that instead of $\|B\|$ in the right hand side of (12), it has been shown possible to use smaller values related to $\rho_j = \|B^j\|^{1/j}$, see, for instance, [3, Thm. 4.2], [6, Thm. 1], and [5, Thm. 1].

4.1. Backward error estimate

In order to verify both (10a) and (10b), we proceed in the following way. We can rewrite Ω as

$$\Omega = \{X \in \mathbb{C}^{n\times n} \colon \rho(\exp(-X)R_m(X)) < 1\}.$$

Due to the following chain of inequalities

$$\rho(\exp(-X)R_m(X)) \leq \|\exp(-X)R_m(X)\| \leq \sum_{j=m/z}^{\infty} \left\| (X^z)^j \sum_{i=1}^{z} b_{jz+i,m} X^i \right\|,$$

if the right hand side is smaller than one for $X = s^{-1}B$, then $s^{-1}B \in \Omega$. In order to simplify the notation, we set

$$\delta_{j-m/z} := \left\| (s^{-z}B^z)^j \sum_{i=1}^{z} b_{jz+i,m} s^{-i} B^i \right\|, \quad j = m/z, m/z+1, \ldots, \infty$$

in such a way that

$$\sum_{l=0}^{\infty} \delta_l < 1 \Rightarrow s^{-1}B \in \Omega.$$

We can explicitly compute at a small cost the matrix summation inside δ_l, since we stored the matrix powers $B^2, \ldots, B^z$. Then, by means of matrix-vector products we can estimate $\delta_0, \delta_1, \ldots$ following the algorithm for 1-norm estimate given in [16].

In order to rapidly decide whether or not the series on the right hand side satisfies our requirement, we are going to make the following unique conjecture. Namely, we assume that if the sequence $\{\delta_l\}_l$ starts to decrease at a certain point, then it decreases quite fast: if $\delta_{\bar{l}+1} \leq \delta_{\bar{l}}$, then $\delta_{\bar{l}+i+1} \leq \frac{\delta_{\bar{l}+i}}{2}$ for any $i \geq 0$. As a matter of fact, this is a mild assumption. In fact, in our numerical experiments, the decay rate of the sequence $\{\delta_l\}_l$ is much faster, as it is shown in Figure 1. This implies the sequence $\{\delta_l\}_l$ to enjoy the property of having each

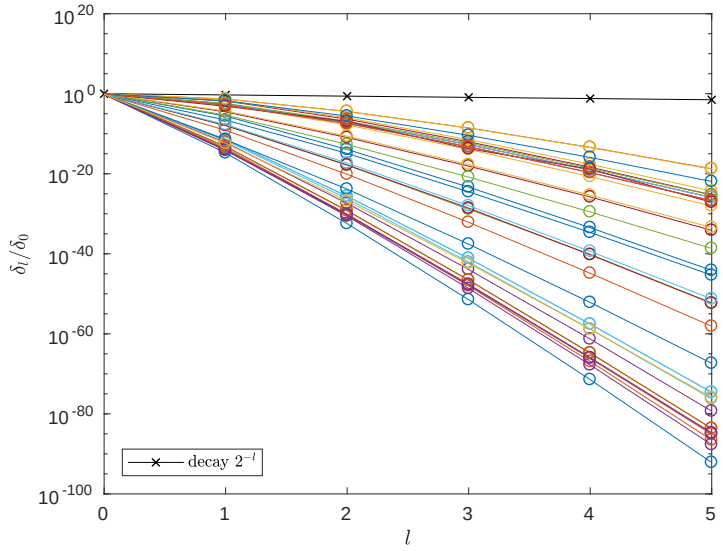

Figure 1: Comparison of the decay rate 2^{-l} (crossed black line) and the normalized sequences $\{\delta_l/\delta_0\}_l$ for the matrices of size $n = 512$ from the test set described in Section 6.2.

term bigger than the sum of all the following whenever it starts to decrease. In other words if $\delta_{\bar{l}} \leq \delta_{\bar{l}-1}$ then

$$\delta_{\bar{l}} \geq \sum_{l=\bar{l}+1}^{\infty} \delta_l.$$

Therefore, in order to check whether

$$\sum_{l=0}^{\infty} \delta_l < 1$$

we proceed as follows: we sequentially evaluate δ_l until we encounter an integer $l^\star$ which falls in one of the following cases:

$$\begin{cases} \delta_{l^\star} \leq \delta_{l^\star-1} \text{ and } \displaystyle\sum_{l=0}^{l^\star} \delta_l + \delta_{l^\star} < 1 & \Rightarrow \text{accept } (m,z,s) \\ \displaystyle\sum_{l=0}^{l^\star} \delta_l \geq 1 \text{ or } l^\star > m/z - 1 & \Rightarrow \text{reject } (m,z,s) \end{cases}$$

Notice that we decided to produce a possibly false negative response whenever $l^\star > m/z-1$. This is not likely to happen, in fact in our numerical experiments the registered values of $l^\star$ were most times 1, very rarely 2.

On the other hand, setting $m/z-1$ as a maximum value after which we return a false negative allows us to simultaneously check (10a) and (10b). In fact, as already noticed, $b_{k,m} = c_{k,m}$ for $k = 0, 1, \ldots, 2m+1$, hence it follows that $\epsilon_l = \delta_l$ for $l = 0, 1, \ldots, m/z-1$. Therefore, if we set as early termination cases

$$\begin{cases} \delta_{l^\star} \leq \delta_{l^\star-1} \text{ and } \displaystyle\sum_{l=0}^{l^\star} \delta_l + \delta_{l^\star} < \min\{1, \mathrm{tol} \cdot s^{-1}\|A\|\} & \Rightarrow \text{accept } (m,z,s) \\ \displaystyle\sum_{l=0}^{l^\star} \delta_l \geq \min\{1, \mathrm{tol} \cdot s^{-1}\|A\|\} \text{ or } l^\star > m/z - 1 & \Rightarrow \text{reject } (m,z,s) \end{cases} \tag{13}$$

we verify at once both (10a) and (10b). As an additional confirm that the assumption made is safe we refer to the accuracy tests in the numerical examples of Section 6.

5. Approximation selection

Exploiting the above algorithm to decide whether or not a triple of parameters (m,z,s) is *acceptable* (that is, the approximation stemming from (m,z,s) satisfies the first condition in (13)), we can greedily determine the optimal combination for our purposes.

As previously stated, the starting point is the pair (m_2, z_2). Right after B^2 is computed, we can obtain the scaling parameter $s = s_2$ just as we described in Section 3. Now we check whether the triple (m_2, z_2, s_2) satisfies the accuracy criteria as described in Section 4. If not, we increase MP by 1. From that, it will follow a new pair (m_3, z_3) from which it may follow a new, smaller, scaling parameter s_3. This is possible since, by increasing MP to 3, the optimization index j in (4) can reach the value $z_3 = 3$ and $\vec{\rho}$ can be smaller. We continue

analogously until we find a MP such that the triple $(m_{\mathrm{MP}}, z_{\mathrm{MP}}, s_{\mathrm{MP}})$ is acceptable. The computational cost of the algorithm, in terms of matrix products, is

$$\underbrace{z_{\mathrm{MP}} + \frac{m_{\mathrm{MP}}}{z_{\mathrm{MP}}} - 2}_{\mathrm{MP}} + \lceil \log_2(s_{\mathrm{MP}}) \rceil. \tag{14}$$

At this point, we would be ready to actually evaluate the approximation of the matrix exponential by means of the Paterson–Stockmeyer evaluation scheme. But before doing that, there is a possible last refinement to do in order to increase the accuracy. As we mentioned in Section 3, the phenomenon called overscaling can seriously compromises the overall accuracy. In order to reduce the risk of incurring in such a phenomenon, one should always pick s as small as possible. So, if $s^0_{\mathrm{MP}} := s_{\mathrm{MP}}$ is larger than one, we elaborated the following strategy: once we obtain a satisfying triple of parameters $(m_{\mathrm{MP}}, z_{\mathrm{MP}}, s_{\mathrm{MP}})$, we set i to 1 and check whether the triple $(m_{\mathrm{MP}}, z_{\mathrm{MP}}, s^i_{\mathrm{MP}})$ with

$$s^i_{\mathrm{MP}} := 2^{\lceil \log_2(s^{i-1}_{\mathrm{MP}}) \rceil - 1} < s^{i-1}_{\mathrm{MP}} \tag{15}$$

still grants our accuracy criteria. This is not an expensive check: in fact, all the required powers of B are still available and their coefficients have simply to be scaled by the powers of a different scaling parameter. Moreover, such a refinement can only reduce the cost of the squaring phase of the algorithm, without changing the cost MP. If the prescribed accuracy is still reached and $s^i_{\mathrm{MP}} > 1$, then we try again to reduce the scaling parameter. By proceeding in this way, we generate and test a sequence of triples $\{(m_{\mathrm{MP}}, z_{\mathrm{MP}}, s^j_{\mathrm{MP}})\}_j$ with $s^j_{\mathrm{MP}} < s^{j-1}_{\mathrm{MP}}$. If at a certain point $s^i_{\mathrm{MP}} = 1$ and the triple $(m_{\mathrm{MP}}, z_{\mathrm{MP}}, 1)$ is acceptable, we proceed with the corresponding polynomial evaluation. Otherwise, we stop as soon as we find a not acceptable triple $(m_{\mathrm{MP}}, z_{\mathrm{MP}}, s^i_{\mathrm{MP}})$. In this case, we estimate the backward error corresponding to the interpolation with parameters $(m_{\mathrm{MP}+1}, z_{\mathrm{MP}}, s^i_{\mathrm{MP}})$. Such an evaluation does not require any additional matrix power, but only different 1-norm estimates. In fact, we can estimate the error as described in Section 4.1 since z_{MP} divides $m_{\mathrm{MP}+1}$. If this new triple is still not acceptable, we proceed with the polynomial evaluation corresponding to $(m_{\mathrm{MP}}, z_{\mathrm{MP}}, s^{i-1}_{\mathrm{MP}})$. Otherwise, the triple $(m_{\mathrm{MP}+1}, z_{\mathrm{MP}}, s^i_{\mathrm{MP}})$ is acceptable and we compute $B^{z_{\mathrm{MP}+1}}$ and $\underline{\vec{\rho}} = \min\{\rho_1, \rho_2, \ldots, \rho_{z_{\mathrm{MP}+1}}\}$, if $z_{\mathrm{MP}+1} > z_{\mathrm{MP}}$ (see Table 1). Now, we could proceed with the polynomial evaluation associated to the acceptable triple $(m_{\mathrm{MP}+1}, z_{\mathrm{MP}+1}, s^0_{\mathrm{MP}+1})$, where $s^0_{\mathrm{MP}+1} := s^i_{\mathrm{MP}}$, but we restart instead the scaling refinement process, taking into account the new $\rho_{z_{\mathrm{MP}+1}}$. Notice that the overall cost associated to the triple $(m_{\mathrm{MP}}, z_{\mathrm{MP}}, s^{i-1}_{\mathrm{MP}})$ is equivalent to the one corresponding to $(m_{\mathrm{MP}+1}, z_{\mathrm{MP}+1}, s^0_{\mathrm{MP}+1})$. In fact, the additional cost associated to the pair $(m_{\mathrm{MP}+1}, z_{\mathrm{MP}+1})$ with respect to $(m_{\mathrm{MP}}, z_{\mathrm{MP}})$ is compensated by the smaller cost due to the scaling $s^0_{\mathrm{MP}+1} < s^{i-1}_{\mathrm{MP}}$. We finally sketch in Algorithm 1 the steps for the selection of the final triple (m, z, s) associated to the polynomial evaluation.

Algorithm 1 Scheme of the selection of an acceptable triple (m, z, s)

Compute $B := A - \mu I$, see formula (3), and $\rho_1 := \|B\|_1$
Set MP := 1
while true **do**
    Compute $m_{\mathrm{MP}+1}$ and $z_{\mathrm{MP}+1}$ according to formula (2)
    if $z_{\mathrm{MP}+1} > z_{\mathrm{MP}}$ **then**
        Compute $B^{z_{\mathrm{MP}+1}}$ and $\rho_{z_{\mathrm{MP}+1}} := \|B^{z_{\mathrm{MP}+1}}\|_1^{1/z_{\mathrm{MP}+1}}$
    end if
    Choose $s^0_{\mathrm{MP}+1}$ according to formula (4)
    Set MP := MP + 1
    if $(m_{\mathrm{MP}}, z_{\mathrm{MP}}, s^0_{\mathrm{MP}})$ is acceptable (see formula (13)) **then**
        Go to 14:
    end if
end while
Set $i := 1$
while $s^{i-1}_{\mathrm{MP}} > 1$ **do**
    Compute s^i_{MP} by taking the minimum arising from formulas (4) and (15)
    if $(m_{\mathrm{MP}}, z_{\mathrm{MP}}, s^i_{\mathrm{MP}})$ is acceptable (see formula (13)) **then**
        Set $i := i + 1$
        Go to 33:
    end if
    Compute $m_{\mathrm{MP}+1}$ according to formula (2)
    if $(m_{\mathrm{MP}+1}, z_{\mathrm{MP}}, s^i_{\mathrm{MP}})$ is acceptable (see formula (13)) **then**
        Set $s^0_{\mathrm{MP}+1} := s^i_{\mathrm{MP}}$
        Compute $z_{\mathrm{MP}+1}$ according to formula (2)
        if $z_{\mathrm{MP}+1} > z_{\mathrm{MP}}$ **then**
            Compute $B^{z_{\mathrm{MP}+1}}$ and $\rho_{z_{\mathrm{MP}+1}} := \|B^{z_{\mathrm{MP}+1}}\|_1^{1/z_{\mathrm{MP}+1}}$
        end if
        Set $i := 0$ and MP := MP + 1
        Go to 33:
    else
        Go to 35:
    end if
    Continue
end while
Evaluate the polynomial according to $(m, z, s) := (m_{\mathrm{MP}}, z_{\mathrm{MP}}, s^i_{\mathrm{MP}})$.

6. Numerical experiments

In this section we are going to test the validity of our algorithm. We will first test the possibility to achieve an accuracy smaller than 2^{-53} (corresponding to double precision), with no need for any precomputation. Moreover, we will show that our new method is competitive with state of the art algorithms in terms of computational performance.

6.1. Accuracy analysis

We start by showing the accuracy of the method (implemented in MATLAB as `exptayotf`[3], EXPonential TAYlor On-The-Fly) with respect to trusted reference solutions. Hence we select, similarly to what done in [5],

- fifty matrices of type $V^T DV$, where V is a Hadamard matrix of dimension n (as generated by the MATLAB command `hadamard`) normalized in order to be orthogonal and D is a diagonal matrix with entries uniformly randomly distributed in $[-50, 50] + \mathrm{i}[-50, 50]$,
- fifty matrices of type $V^T JV$, where J is a block diagonal matrix made of Jordan blocks. The $(i+1)$-th Jordan block J_{i+1} has dimension $|J_{i+1}|$ randomly chosen in $\{1, 2, \ldots, n - |J_1| - |J_2| - \ldots - |J_i|\}$. The diagonal elements of different Jordan blocks are uniformly randomly distributed in $[-50, 50] + \mathrm{i}[-50, 50]$.

The chosen dimensions are $n = 16$, $n = 64$ and $n = 256$, respectively. The computations were carried out in Matlab R2017a using variable precision arithmetic with 34 decimal digits (roughly corresponding to quadruple precision) and requiring tolerance $\mathrm{tol} = 2^{-83} \approx 10^{-25}$ to our algorithm. This is a sort of intermediate accuracy between double and quadruple precision. This choice allows to show the possibility of using the algorithm at any desired accuracy and not only at the usual single or double precision (see [3, 5]). The reference solutions were computed with the same number of decimal digits by using $\exp(V^T DV) = V^T \exp(D)V$ and $\exp(V^T JV) = V^T \exp(J)V$. In Figure 2 we report the achieved relative error in 1-norm for the three different dimensions. The filled circles indicate the average errors, while the endpoints of the vertical lines indicate the maximum errors among the set of 100 matrices for each chosen dimension. From this experiment, we see that the method is able to approximate the matrix exponential at high precision, without relying on any precomputed thresholds. As such, this method could be employed as general routine for the matrix exponential in any environment for multiple precision floating point computations.

[3] Available at `https://bitbucket.org/francoemarco/exptayotf`

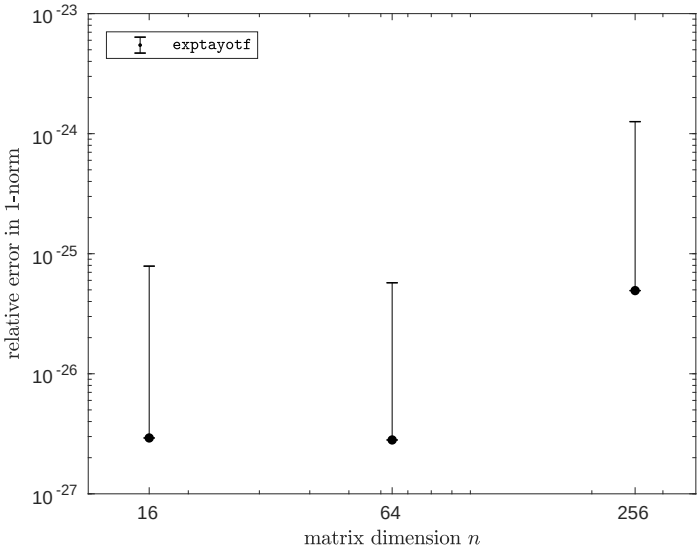

Figure 2: Average and largest relative errors for three sets of matrices orthogonally similar to diagonal or Jordan normal forms.

6.2. Performance and accuracy comparisons

As a second step to validate our algorithm, we proceed with a performance comparison with its two main competitors based on the analysis of the backward or forward error. They are `expm` (as implemented in Matlab R2017a), which exploits a scaling and squaring Padé approximation [3] and `exptaynsv3`[4], which is based on a scaling and squaring Taylor approximation [5]. Both routines are the result of several successive refinements of the original algorithms. In fact, the Padé approximation is used in `expm` at least since Matlab R14 (2004) and was based on [17]. Then, it was improved by considering the backward error analysis developed in [2] and refined some years later following the new ideas introduced in [3]. On the other hand, `exptaynsv3` is the result of successive revisions [5, 9] of the original ideas in [6], in which the backward or forward error analysis were used.

We run this comparison over the matrices coming from the Matrix Computation Toolbox [18], plus few random complex matrices. We decided to scale the test matrices so that they have 1-norm not exceeding 512 for the double data input and 64 for the single data input. This because some matrices from the test set were having a 1-norm too big to have a representable exponential matrix. In addition, by doing so, we kept the cost of the scaling and squaring phase (that may be more or less common to all the tested algorithms for the majority of matrices) from shadowing the differences in cost of the approximations. We

[4]Revised version: 2017/10/23, available at `http://hipersc.blogs.upv.es/software/`.

notice that the function `expm` treats differently particular instances of the input. In fact, diagonal and Hermitian matrices are exponentiated following a Schur algorithm. We decided then to discard those cases from our set of tests matrices, in order to compare algorithms purely based on the backward or forward error analysis. The final set comprises 38 matrices. We run all our tests on a quite old machine with six AMD Phenom II X6 cores at 2.80 GHz and 8 GB of RAM. Although the main operations are multi-threaded (matrix products for all the algorithms and one matrix division only for `expm`), we limited Matlab to use a single computational thread with the starting option `-singleCompThread`. In this way, we obtained more reliable results in the measure of the computational times. The tests were run for double data input asking for double precision 2^{-53} and comparing accuracy and elapsed computational times. For the comparison to be fair, there are some details that we needed to sort out in advance.

- Both our competitor algorithms do not present a shifting strategy (although, for instance, in [5, § 2] it is explicitly mentioned that steps for preprocessing and postprocessing of the matrix to reduce the norm of the matrix can be considered). One could think that the shift is intended to be performed in advance, i.e., computing the exponential of A as $\mathrm{e}^r \exp(A - rI)$ for some scalar r. But, such a simple strategy can be harmful. For example, consider the matrix A given by `matrix(51,32)*2^30` from the Matrix Computational Toolbox. This matrix has eigenvalues with large negative real part. The Matlab function `expm` applied to the shifted matrix $B = A - \mu I$, $\mu = \mathrm{trace}(A)/n$, returns a matrix of `NaN`s. This makes it impossible to recover the exact solution which, in double precision arithmetic, is a matrix of zeros. Hence we are not performing any external shifting strategy for our competitors.

  On the other hand we are not going to transform every matrix from the test set in a zero traced matrix just to avoid this issue. In fact, that would be unfair toward our special backward error inequality (10b) that on the right hand side takes in account not $B = A - \mu I$ but A. As a trivial example, consider any random valued matrix with small norm and big trace, for instance `randn(1024)*1e-10+eye(1024)*1e2`. To compute the exponential of this matrix we take averagely just a hundredth of the time taken by our competitors. Anyway, we did not include such an extreme example in our test set.

- We excluded from the test set matrix number 2, the Chebyshev spectral differentiation matrix corresponding to `gallery('chebspec')`. In fact, the relative condition number $\kappa_{\exp}(A)$, see [4, § 3.1], for such a matrix A is so large (about $1.4 \cdot 10^9$ at dimension $n = 32$) that it turns out to be impossible for any routine to compute an accurate matrix exponential in double precision.

We moreover tested `exptayotf` on double precision data asking for different precisions ranging from 2^{-202} to 2^{-10}. The possibility for `exptayotf` to profitably work with different data types and precisions is not shared by the competitors

which were developed for double precision arithmetic and accuracy (although `expm` works without any problem with single precision data).

6.2.1. Double precision data with tolerance set to 2^{-53}

In these experiments, the reference solutions were computed with the `expm` method using variable precision arithmetic with 34 decimal digits. All the three methods take tol $= 2^{-53} \approx 10^{-16}$ as default tolerance. For each matrix in the test set, the three algorithms were run 10 times and, after discarding the best and the worst performance, the mean elapsed computational time $\bar{t}$ and the standard deviation σ were computed. In Figure 3 we reported the average and largest forward relative errors in 1-norm (top) and the average and largest computational times (bottom) of the three methods. The average relative error committed by `exptayotf` is always smaller than the competitors. The `expm` function is clearly the fastest up to dimension $n = 128$. From dimension $n = 256$, the three methods behave in the same way from the computational point of view. As done in [4, § 10.5], we computed for the test sets of dimension $n = 16$ and $n = 512$ the normwise relative errors in the Frobenius norm. The results are summarized in Figure 4. The condition number of the problems $\kappa_{\exp}(A)$ was computed by the Matrix Function Toolbox [19] function `expm_cond`. The same data are displayed in Figure 5 as a performance profile: a point (α, p) on a curve related to a method is the relative number of matrices in the test set for which the corresponding error in the Frobenius norm is bounded by $\alpha \cdot$ tol. These figures confirm the previous observations. The new scaling parameter possibly chosen as the sum of two powers of two (see Section 3) showed slight improvements. For instance, when we force `exptayotf` to take the scaling parameter as a single power of two at dimension $n = 512$, the average relative error is $9.30 \cdot 10^{-15}$ versus the value $8.58 \cdot 10^{-15}$ obtained with our new choice. The average computational cost was the same, about 0.7 seconds. We summarize in Table 2 the computational efforts of the three methods measured both in terms of the average number of matrix products MP for each test set (now up to dimension $n = 4096$) and in terms of computational times. Besides the average of the mean computational times $\bar{t}$ for each test set, we report the maximum relative standard deviation $\sigma/\bar{t}$. A part from the smallest matrix dimensions, for which the average $\bar{t}$ is of the order or smaller than 0.01 seconds, we see that the average $\bar{t}$ is a reliable measure. For the largest matrix dimensions ($n \geq 1024$) we do not compute a reference solution, but can see in Table 2 that the on-the-fly backward error analysis of `exptayotf` is competitive with the standard approaches. In fact, the method turns out to be faster than the competitors. We observe that the two polynomial methods take about the same number of matrix products MP. Despite for `exptaynsv3` the average MP is slightly smaller, `exptayotf` tends to be faster. This can be explained because the on-the-fly error estimate usually requires a particularly small number of matrix vector products. In fact, thanks to the typical behavior of the sequences $\{\delta_l\}_l$ (see Figure 1), we generally need very few calls to the 1-norm estimator that we purposely implemented according to [16]. Finally we notice that for the `expm` method the amount of matrix products is smaller than the polynomial

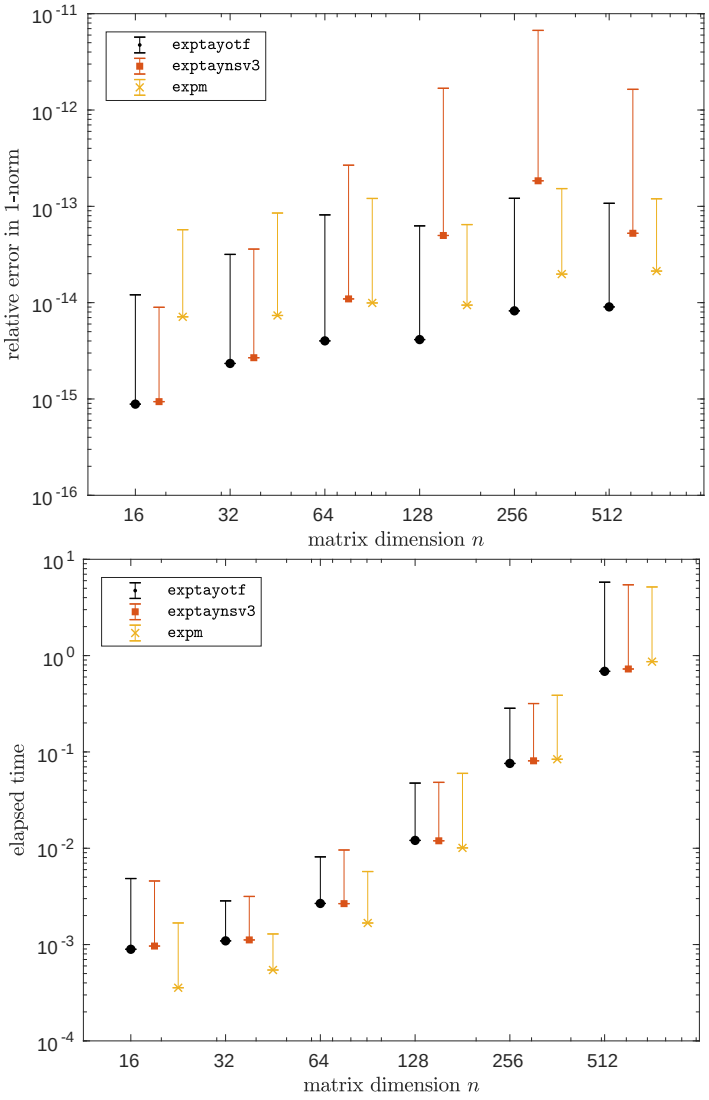

Figure 3: Average and largest relative errors for the test cases described in Section 6.2, tolerance tol set to 2^{-53} (top) and average and largest computational times (bottom).

methods. On the other hand, this method requires to perform a matrix division MD which is slower than a matrix product, although it has the same complexity with respect to the matrix size n.

6.2.2. Double precision data with tolerances different from 2^{-53}

In this experiment we consider the same test set as above, but require an accuracy corresponding to single precision. It is not possible to change the

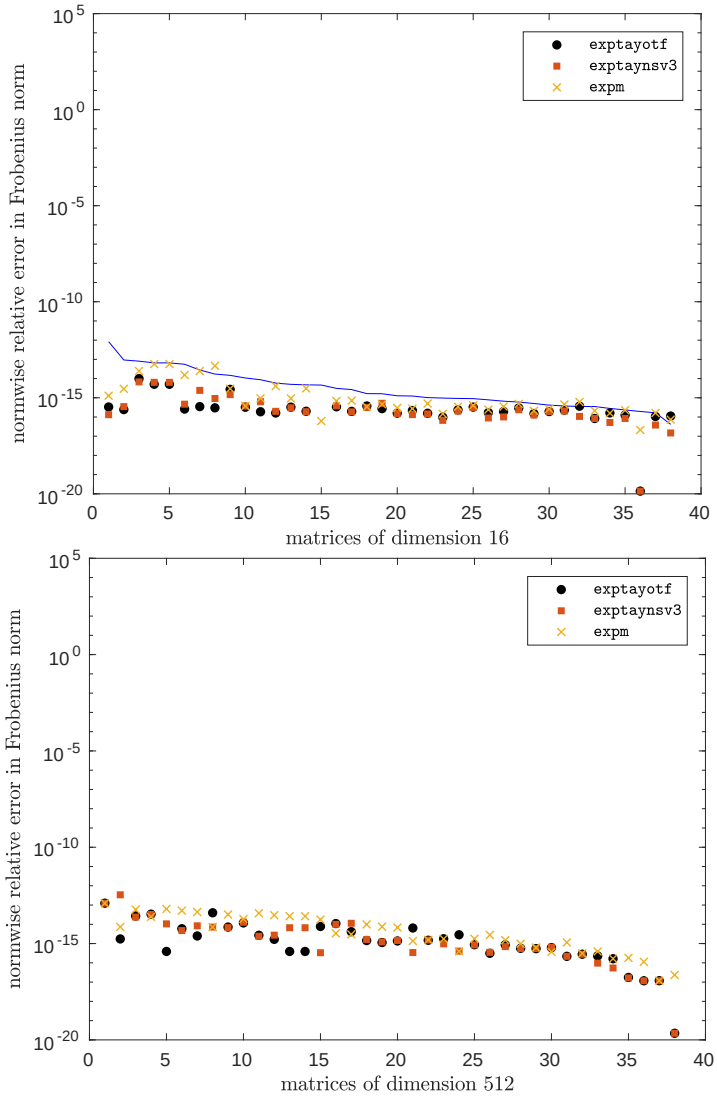

Figure 4: Normwise relative errors in Frobenius norm for the test cases described in Section 6.2, tolerance tol set to 2^{-53}. The solid line is $\kappa_{\exp}(A) \cdot \mathrm{tol}$.

precision in the codes `exptaysnv3` and `expm`, since the error analysis was done in advance only for the precision 2^{-53}. Figures 6 and 7 report the normwise relative errors in the Frobenius norm for the test set of matrices of dimension $n = 16$. We see that `exptayotf` performs in a stable way, with errors well below the corresponding $\kappa_{\exp}(A) \cdot \mathrm{tol}$. In Table 3 we see that the average relative error in the 1-norm is always smaller than the prescribed tolerance for all the test sets up to dimension $n = 512$. In Table 4 we reported the average number of

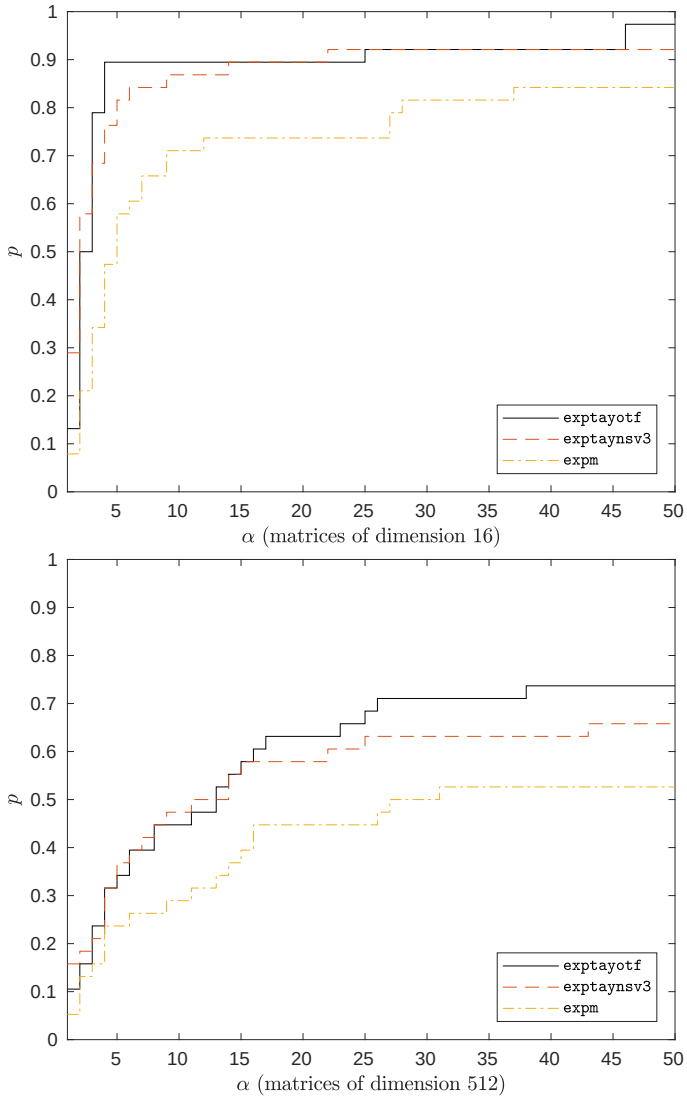

Figure 5: Performance profiles for the test cases described in Section 6.2, tolerance tol set to 2^{-53}.

matrix products and the average elapsed computational times, for the test sets up to dimension $n = 4096$. We see that, if less accuracy is required, it is possible to save up to 43% of the computational time (see the comparison for $n = 512$ between double and half precision).

In `exptayotf` it is possible to ask for a tolerance smaller than 2^{-53}, even with data in double precision. For instance, it is well known that the first column of

n	comput. effort	`exptayotf`	`exptaynsv3`	`expm`
16	average MP	9.47	9.71	7.63+MD
	average $\bar{t}$	8.94e-04	9.65e-04	3.56e-04
	max $\sigma/\bar{t}$	1.01e+00	1.19e+00	1.13e+00
32	average MP	10.26	10.24	8.13+MD
	average $\bar{t}$	1.09e-03	1.12e-03	5.44e-04
	max $\sigma/\bar{t}$	1.21e-01	5.09e-01	9.32e-02
64	average MP	10.55	10.47	8.53+MD
	average $\bar{t}$	2.67e-03	2.66e-03	1.67e-03
	max $\sigma/\bar{t}$	6.81e-02	2.64e-01	1.96e-01
128	average MP	10.87	10.82	8.89+MD
	average $\bar{t}$	1.21e-02	1.19e-02	1.01e-02
	max $\sigma/\bar{t}$	1.05e-01	8.53e-02	2.60e-02
256	average MP	11.29	11.18	9.45+MD
	average $\bar{t}$	7.60e-02	8.09e-02	8.40e-02
	max $\sigma/\bar{t}$	2.48e-02	2.35e-02	9.51e-02
512	average MP	11.84	11.76	9.92+MD
	average $\bar{t}$	6.87e-01	7.26e-01	8.65e-01
	max $\sigma/\bar{t}$	2.64e-02	2.34e-02	2.32e-02
1024	average MP	11.69	11.50	9.89+MD
	average $\bar{t}$	4.83e+00	4.92e+00	5.92e+00
	max $\sigma/\bar{t}$	1.07e-02	2.00e-02	1.54e-02
2048	average MP	11.42	11.36	9.83+MD
	average $\bar{t}$	2.81e+01	3.00e+01	3.54e+01
	max $\sigma/\bar{t}$	1.31e-02	1.19e-02	1.76e-02
4096	average MP	11.31	11.22	9.86+MD
	average $\bar{t}$	2.17e+02	2.32e+02	2.68e+02
	max $\sigma/\bar{t}$	4.21e-03	1.25e-03	1.12e-02

Table 2: Average number of matrix products MP, average mean elapsed time $\bar{t}$ in seconds and maximum relative standard deviation $\sigma/\bar{t}$ for the test cases described in Section 6.2, tolerance tol set to 2^{-53}. MD indicates the matrix division in the Padé algorithm.

the matrix E defined by

$$E = (e_{ij}) = \exp(Z), \quad Z = \begin{bmatrix} \xi_1 & & & & \\ 1 & \xi_2 & & & \\ & 1 & \xi_3 & & \\ & & \ddots & \ddots & \\ & & & 1 & \xi_n \end{bmatrix}$$

are the divided differences of the interpolation polynomial of the exponential function at the sequence of points $\{\xi_j\}_{j=1}^n$ (see [20]). Their accurate approximation is a key point for the polynomial methods for the approximation of the action of the matrix exponential (see [11]). In the trivial case in which $\xi_1 = \xi_2 = \ldots = \xi_n = 0$, the divided differences coincide with the inverse of the

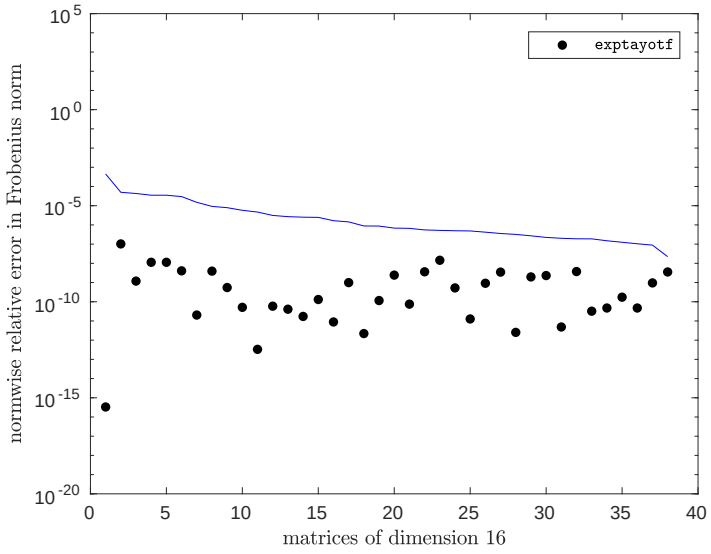

Figure 6: Normwise relative errors in Frobenius norm of the algorithm `exptayotf` for the test cases described in Section 6.2, tolerance tol set to 2^{-24}. The solid line is $\kappa_{\exp}(A) \cdot \mathrm{tol}$.

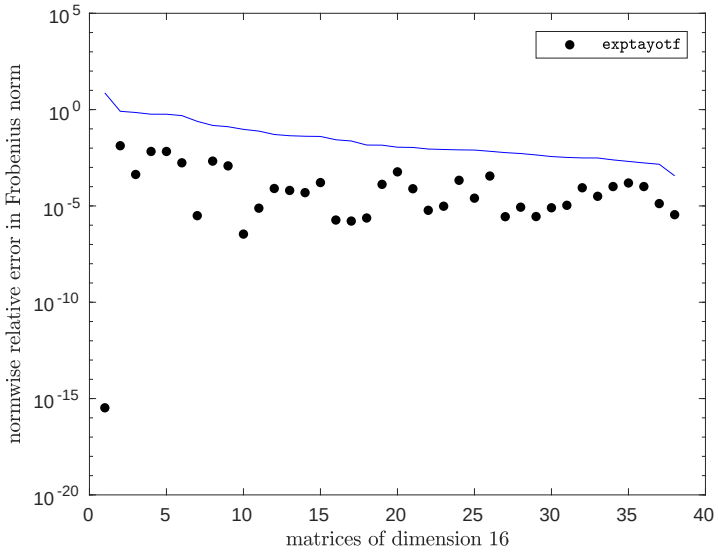

Figure 7: Normwise relative errors in Frobenius norm of the algorithm `exptayotf` for the test cases described in Section 6.2, tolerance tol set to 2^{-10}. The solid line is $\kappa_{\exp}(A) \cdot \mathrm{tol}$.

factorials of the first nonnegative integers (i.e., they are the Taylor coefficients). In Table 5, for $n = 31$, we can see that the algorithm `exptayotf` is able to compute in a accurate way those coefficients, provided that a sufficiently small tolerance is prescribed. This is in general not possible with algorithms (such as

n	relative error 1-norm	`exptayotf` tol $= 2^{-53}$	`exptayotf` tol $= 2^{-24}$	`exptayotf` tol $= 2^{-10}$
16	average	8.84e-16	4.88e-09	9.19e-04
	max	1.21e-14	1.02e-07	1.33e-02
32	average	2.34e-15	2.40e-08	2.47e-04
	max	3.17e-14	5.64e-07	4.88e-03
64	average	4.01e-15	1.49e-08	2.25e-04
	max	8.15e-14	1.60e-07	4.97e-03
128	average	4.14e-15	1.16e-08	5.37e-04
	max	6.27e-14	2.40e-07	4.97e-03
256	average	8.22e-15	1.80e-08	5.26e-04
	max	1.21e-13	4.56e-07	4.97e-03
512	average	9.03e-15	2.90e-08	6.41e-04
	max	1.08e-13	6.09e-07	4.97e-03

Table 3: Average and largest relative errors of the algorithm `exptayotf` for the test cases described in Section 6.2, tolerances tol set to 2^{-53}, 2^{-24}, and 2^{-10}.

`expm` and `exptaynsv3`) developed for the double precision.

Another interesting case is the computation of the exponential matrix for the Hessenberg matrices arising in the Krylov method for the approximation of $\exp(\tau A)v$. In fact, we have

$$\exp(\tau A)v \approx \beta V_n \exp(\tau H_n)e_1$$

(see, for instance, [21, formula (4)]), where $H_n = V_n^T A V_n$ is an Arnoldi factorization with $V_n^T V_n = I$ and H_n a Hessenberg matrix of dimension n much smaller than the dimension of A and $\beta = \|v\|_2$. The elements of the vector $\exp(\tau H_n)e_1$ can be seen as the coefficients of a linear combination of the orthonormal columns of V_n. If we consider the matrix A of dimension 256 which discretizes the one-dimensional advection-diffusion operator $\partial_{xx} + \partial_x$ with homogeneous Dirichlet boundary conditions (in MATLAB syntax)

```
m = 256;
h = 1 / (m + 1);
A = toeplitz (sparse ([1, 1], [1, 2], [-2, 1] / h ^ 2, 1, m)) + ...
    toeplitz (sparse (1, 2, -1 / (2 * h), 1, m), ...
    sparse (1, 2, 1 / (2 * h), 1, m));
```

and initial vector v

```
v = sin (pi * linspace (0, 1, m + 2)');
v = v(2:m + 1);
```

and produce the Hessenberg matrix H_{41} by the standard Arnoldi process and compute $E = \exp(\tau H_{41})$ for $\tau = 10^{-5}$, we obtain the results in Table 6. Once again, it is possible to recover accurate coefficients only by asking for higher

n	comput. effort	`exptayotf` tol $= 2^{-53}$	`exptayotf` tol $= 2^{-24}$	`exptayotf` tol $= 2^{-10}$
16	average MP	9.47	7.63	5.95
	average $\bar{t}$	8.94e-04	8.33e-04	7.65e-04
	max $\sigma/\bar{t}$	1.01e+00	1.02e+00	1.11e+00
32	average MP	10.26	8.16	6.58
	average $\bar{t}$	1.09e-03	8.88e-04	7.86e-04
	max $\sigma/\bar{t}$	1.21e-01	1.20e-01	2.80e-01
64	average MP	10.55	8.45	6.95
	average $\bar{t}$	2.67e-03	2.11e-03	1.70e-03
	max $\sigma/\bar{t}$	6.81e-02	5.73e-02	6.33e-02
128	average MP	10.87	8.84	7.08
	average $\bar{t}$	1.21e-02	9.35e-03	7.59e-03
	max $\sigma/\bar{t}$	1.05e-01	3.28e-02	4.09e-02
256	average MP	11.29	9.24	7.53
	average $\bar{t}$	7.60e-02	5.85e-02	4.82e-02
	max $\sigma/\bar{t}$	2.48e-02	2.73e-02	4.71e-02
512	average MP	11.84	9.76	8.05
	average $\bar{t}$	6.87e-01	5.11e-01	3.91e-01
	max $\sigma/\bar{t}$	2.64e-02	1.97e-02	1.65e-01
1024	average MP	11.69	9.53	7.72
	average $\bar{t}$	4.83e+00	3.69e+00	2.81e+00
	max $\sigma/\bar{t}$	1.07e-02	1.12e-02	1.68e-01
2048	average MP	11.42	9.33	7.69
	average $\bar{t}$	2.81e+01	2.21e+01	1.81e+01
	max $\sigma/\bar{t}$	1.31e-02	1.58e-02	1.81e-02
4096	average MP	11.31	9.28	7.61
	average $\bar{t}$	2.17e+02	1.76e+02	1.40e+02
	max $\sigma/\bar{t}$	4.21e-03	1.75e-02	6.69e-02

Table 4: Average number of matrix products MP, average mean elapsed time $\bar{t}$ in seconds and maximum relative standard deviation $\sigma/\bar{t}$ of the algorithm `exptayotf` for the test cases described in Section 6.2, tolerances tol set to 2^{-53}, 2^{-24}, and to 2^{-10}.

precision than 2^{-53}. Moreover, the computational cost was negligible with respect to the computation of the reference solution by `expm` in variable precision arithmetic with 34 decimal digits.

7. Conclusions

We have shown that it is possible to estimate on-the-fly the backward error for the approximation of exponential of a given matrix A. By means of this estimate, we can select the scaling parameter s and Taylor polynomial degree m in order to reach any desired accuracy. This is confirmed by the results made in variable precision arithmetic and reported in Figure 2. Moreover, for data

n	$e_{n,1}$, tol $= 2^{-53}$	$e_{n,1}$, tol $= 2^{-106}$
1	1.000000000000000e+00	1.000000000000000e+00
2	1.000000000000000e+00	1.000000000000000e+00
6	8.333333333333333e-03	8.333333333333333e-03
11	2.755731922398589e-07	2.755731922398589e-07
16	7.647163731819814e-13	7.647163731819814e-13
21	4.110317623312165e-19	4.110317623312165e-19
26	0.000000000000000e+00	6.446950284384474e-26
31	0.000000000000000e+00	3.769987628815907e-33

Table 5: Some divided differences of the exponential function at the points $\xi_1 = \xi_2 = \ldots = \xi_n = 0$ (first column of $\exp(Z)$) computed by `exptayotf`.

n	$e_{n,1}$, reference	$e_{n,1}$, tol $= 2^{-53}$	$e_{n,1}$, tol $= 2^{-202}$
1	9.999013095670584e-01	9.999013095670580e-01	9.999013095670580e-01
2	3.115436398118472e-05	3.115436398118472e-05	3.115436398118472e-05
6	2.723279966591328e-09	2.723279966591318e-09	2.723279966591328e-09
11	9.034777099109725e-15	9.034777098536935e-15	9.034777099109728e-15
16	2.888709002783644e-21	2.888696660347384e-21	2.888709002783644e-21
21	1.864181126800604e-28	1.716726863031155e-28	1.864181126800605e-28
26	3.573156001812017e-36	0.000000000000000e+00	3.573156001812017e-36
31	2.577050509143562e-44	0.000000000000000e+00	2.577050509143563e-44
36	8.203346368054099e-53	0.000000000000000e+00	8.203346368054094e-53
41	1.292460535272054e-61	0.000000000000000e+00	1.292460535272055e-61

Table 6: Some entries in the first column of $\exp(\tau H_{41})$ computed by `exptayotf`.

in double precision, the resulting algorithm, named `exptayotf`, has proven to be competitive both in terms of accuracy and computational time with two state-of-the-art routines such as `exptaynsv3` and the default `expm` available in Matlab R2017a. In fact, from Figures 3–5 it appears clear that we are able to always achieve the best average forward error on heterogeneous sets of 38 matrices. Such a result was possible thanks to our careful shifting strategy and the new selection of the scaling parameter s among sums of powers of two, instead of the classical choice of a single power of two. In terms of computational times, for small matrix dimensions ($n < 64$), `expm` turned out to be by far the fastest method. But, for higher dimensions, as clear from Figure 3 and Table 2, `exptayotf` is revealed to have the smallest cost growth. For instance, as reported in Table 2 for tolerance set to 2^{-53}, `exptayotf` is slightly faster than `exptaynsv3`, while reaches a speedup of 1.23 over `expm` for matrix dimension $n = 1024$. If instead less accuracy is required, for instance a tolerance set to 2^{-10}, the speedup over the double precision accuracy 2^{-53} for the same matrix dimension is 1.76. The final result is an algorithm which is proven to be reliable at any desired accuracy and faster than the competitors for medium to large sized matrices.

References

[1] C. B. Moler, C. F. Van Loan, Nineteen dubious ways to compute the exponential of a matrix, twenty-five years later, SIAM Rev. 45 (1) (2003) 3–49.

[2] N. J. Higham, The scaling and squaring method for the matrix exponential revisited, SIAM J. Matrix Anal. Appl. 26 (4) (2005) 1179–1193.

[3] 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.

[4] N. J. Higham, Functions of Matrices, SIAM, Philadelphia, 2008.

[5] R. Ruiz, J. Sastre, J. Ibáñez, E. Defez, High performance computing of the matrix exponential, J. Comput. Appl. Math. 291 (1) (2016) 370–379.

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

[7] J. Sastre, J. Ibáñez, E. Defez, R. Ruiz, Efficient orthogonal matrix polynomial based method for computing matrix exponential, Appl. Math. Comp. 217 (2011) 6451–6463.

[8] J. Sastre, J. Ibáñez, E. Defez, R. Ruiz, New scaling-squaring Taylor algorithms for computing the matrix exponential, SIAM J. Sci. Comput. 37 (1) (2015) A439–A455.

[9] J. Sastre, J. Ibáñez, R. Ruiz, E. Defez, Accurate and efficient matrix exponential computation, Int. J. Comp. Math. 91 (1) (2014) 97–112.

[10] 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.

[11] M. Caliari, P. Kandolf, A. Ostermann, S. Rainer, The Leja method revisited: backward error analysis for the matrix exponential, SIAM J. Sci. Comput. 38 (3) (2016) A1639–A1661.

[12] M. Caliari, P. Kandolf, F. Zivcovich, Backward error analysis of polynomial approximations for computing the action of the matrix exponential (2018). Submitted.

[13] C. van Loan, A note on the evaluation of matrix polynomials, IEEE Trans. Autom. Control 24 (2) (1979) 320–321.

[14] C. S. Kenney, A. J. Laub, A Schur–Fréchet algorithm for computing the logarithm and exponential of a matrix, SIAM J. Matrix Anal. Appl. 19 (1998) 640–663.

[15] T. M. Fischer, On the algorithm by Al-Mohy and Higham for computing the action of the matrix exponential: A posteriori roundoff error estimation, Linear Alg. Appl. 531 (2017) 141–168.

[16] N. 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 (4) (2000) 1185–1201.

[17] G. H. Golub, C. F. Van Loan, Matrix computations, 4th Edition, Johns Hopkins Studies in the Mathematical Sciences, Johns Hopkins University Press, 2012.

[18] N. J. Higham, The Matrix Computation Toolbox, version 1.2 (2002). URL `http://www.ma.man.ac.uk/~higham/mctoolbox`

[19] N. J. Higham, The Matrix Function Toolbox, version 1.0 (2008). URL `http://www.ma.man.ac.uk/~higham/mftoolbox`

[20] A. McCurdy, K. C. Ng, B. N. Parlett, Accurate computation of divided differences of the exponential function, Math. Comp. 43 (168) (1984) 501–528.

[21] Y. Saad, Analysis of some Krylov subspace approximations to the matrix exponential operator, SIAM J. Numer. Anal. 29 (1) (1992) 209–228.

Revision of manuscript CAM-D-18-00207

On-the-fly backward error estimate for matrix exponential approximation by Taylor algorithm

M. Caliari F. Zivcovich

Dear Prof. Wuytack,

Thank you very much for the reviewers' reports and your help in the refereeing process. We also want to thank the three anonymous reviewers for their very valuable suggestions to improve the presentation of our research. We have prepared a revision of our manuscript, marking in blue the changes. Since Reviewer #3 asked to removed some numerical experiments, we did it and added at the end of section 6.2.2 a couple of new examples about the possibility to use `exptayotf` with different precisions. We hope that the current version is suitable for publication on the Journal of Computational and Applied Mathematics.

Answer to Reviewer #2

Suggestions

- The code is now available at `https://bitbucket.org/francoemarco/exptayotf/`

Minor Comments

- We made clear the definition of δ.
- As written in the revised version in section 6.2, although the main operations are multi-threaded (matrix products for all the algorithms and one matrix division only for `expm`), we limited Matlab to use a single computational thread with the starting option `-singleCompThread`. In this way, we obtained more reliable results in the measure of the computational times. Without such an option, we simply observed a larger variance in the measure of the computational times (see Tables 2 and 4).
- The third reviewer made us notice that the comparisons reported in the old Section 6.3 were not fair, since our competitor algorithms were not designed for single precision data type. We now agree with this position and we have decided to remove that section. Anyway, in the revised version we used the algorithm `expm` (as it was in the old Section 6.3) in variable precision arithmetic with 34 decimal digits in Section 6.2.1.

Major Comments

- We made publicly available the full algorithm, which is quite long (about 350 lines). The kernel of our algorithm are the well known Paterson–Stockmeyer scheme for the Taylor truncated series of the exponential function and the scaling and squaring algorithm. We believe that it is not necessary to give in the paper the pseudocode of these parts. Therefore, we reported at the end of Section 5 the algorithm for the selection of the triple (m, z, s) which determines the polynomial evaluation.

- We incorporated the normwise relative errors and the performance profiles for our test sets of matrices of dimension 16 and 512. When the tolerance for the `exptayotf` method was set to 2^{-24} or 2^{-10}, we incorporated the normwise relative errors for the test set of matrices of dimension 16. In fact, being impossible to compute the condition number $\kappa_{\exp}(A)$ for matrices of dimension $n = 512$ (because of memory requirements) and being not fair the comparison with the other methods, the normwise relative error plot would have been a meaningless sequence of black circles.

Answer to Reviewer #3

Major comments

- The code is now available at `https://bitbucket.org/francoemarco/exptayotf`

- For the sake of clarity, we did not shorten this already short paragraph. But clearly the coefficients were first derived in [9] and hence we made sure that this merit is highlighted by means of explicit citations to the corresponding formulas in [9].

- We were not able to theoretically justify the assumption on the decay of the sequence $\{\delta_l\}_l$. Therefore, in the revised version of the manuscript we computed the values of the sequence for the test set of matrices of dimension $n = 512$ and displayed the results in Figure 1. Our numerical tests seem to fully confirm the reasonableness of our assumption.

- The values $\rho_j = ||B^j||_1^{1/j}$ have been computed "exactly" with the function `norm`, since the matrix powers are available and necessary for the final polynomial approximation. The values δ_l (beginning of page 10 of the revised version), instead, are approximated by a function that we implemented by following the algorithm in [16]. It is important for our

purposes only to recover the magnitude of these values and not the full precision.

- We can assure the reviewer that it was not our intention to perform unfair comparisons. In fact, we even highlighted (end of page 16 of the original manuscript) that the gain obtained by lowering the accuracy requirements was not much. Anyway, we removed all comparisons in precision different than double precision arithmetic. We emphasized that our competitors were developed for double precision. We removed the tests with data in single precision since quite uninformative without terms of comparison. We added the normwise relative error figures for `exptayotf` run with single and half precision, for the test set of matrices of dimension $n = 16$ (Figures 6 and 7).

- We added the normwise relative error and performance profile figures for the double precision results, for the lowest and the biggest matrix sizes (Figures 4 and 5). For the latter, it was not possible to compute the conditions numbers due to the excessive memory requirements.

- We run three methods over six test sets with about 40 matrices each. We do not know how to report in a smart way the standard deviation "for each matrix". We did instead the following: for each method and matrix, after discarding the best and the worst elapsed time measures, we computed the standard deviation σ and the mean $\bar{t}$. Then, we reported in Tables 2 and 4 for each method the *average* of the mean $\bar{t}$ among all matrices of a given test set and the *maximum* relative standard deviation $\sigma/\bar{t}$ among all matrices of a given test set.

- We computed the number of matrix products and reported them in Tables 2 and 4.

Minor comments

- We clarified that Taylors algorithms in [5,6,9] use a combination of forward and backward error analysis.

- Here, and in other parts of the original manuscript, we wrongly referenced to [7] instead of [6]. We fixed this.

- We made the choice of z clear below formula (3).

- The maximum m is not 25, it is written below Table 1 that we reported "as an example" the costs up to $m = 25$.

- The default values of $\rho_{\max}$ are 3.5 for data in double or higher precision and 6.3 for data in single precision. We clarified this choice.
- We fixed the notation about ρ_j.
- We defined $\epsilon_{j-m/z}$.
- It is true that different values of $\alpha_p(B)$ are used in the literature. Since in our approach we do not make use of any $\alpha_p(B)$, we decided to give a more general overview of the classical backward analysis in Remark 1 and give precise references to the reader.
- We defined $\delta_{j-m/z}$.
- We clarified the choice of $\rho_{\max}$.
- We tried to better explain the scaling refinement and added an algorithm.
- We clarified the sentence about the equivalent costs.
- We repeated the experiment using variable precision arithmetic with 34 decimal digits and keeping the tolerance to $2^{-83} \approx 10^{-25}$. We removed from the plot the smallest errors, since confusing. In fact, it can happen the some particular cases may result particularly accurate, but they do not represent the general behavior. The average error is smaller than the required tolerance, too. This phenomenon can be explained by the overestimate of the actual backward error. This was true also in the previous version the manuscript. In fact, even if we used only 25 decimal digits (corresponding to the required tolerance), Matlab uses a rather large number of guard digits when working in variable precision arithmetic.
- We described the specifics of the used machine in section 6.2. With the option `-singleCompThread` we obtained more reliable measures of computational times, without penalizing any method.
- We specified that [5,6,9] use the backward or the forward error analysis.
- We accepted the suggested sentence about the scaling of the matrices.
- We mentioned that in [5, § 2] the preprocessing and the postprocessing were taken into account.

- The reference results in Section 6.2.1 were recomputed by using `expm` in variable precision arithmetic with 34 decimal digits, which roughly corresponds to quadruple precision.
- We modified the sentence about the scaling parameter based on the sum of two powers.
- Reference [12] has been submitted again to the same journal after a major revision and it is still under revision.