EFFICIENT SCALING-SQUARING TAYLOR METHOD FOR COMPUTING THE MATRIX EXPONENTIAL[*]

J. SASTRE[†], J. IBÁÑEZ[‡], E. DEFEZ[§], AND P. RUIZ[¶]

Abstract. The matrix exponential plays a fundamental role in linear systems arising in engineering, mechanics and control theory. In this paper, an efficient Taylor method for computing matrix exponentials is presented. Taylor series truncation together with a modification of the Paterson-Stockmeyer method avoiding factorial evaluations, and the scaling-squaring technique, allow efficient computation of the matrix exponential approximation. A careful backward-error analysis of the approximation is given and a theoretical estimate for the optimal scaling of matrices is obtained. The modified Paterson-Stockmeyer implementation was compared with the classical implementation and other efficient state of the art methods on dense matrices for different dimensions from 2×2 to 100×100. Numerical tests show that it obtains higher precision than all compared methods in the majority of cases. We show that it presents lower computational cost in terms of matrix products than efficient Padé methods for general dense matrices satisfying $2.53e-1 < ||A|| \leq 2.99e-1$, $1.49e-2 < ||A|| \leq 8.95e-2$ and $||A|| \leq 9.06e-3$, exceeding asymptotically Padé cost in between $1 + 2/3$ matrix products in most of the rest of cases, or a maximum of $2 + 2/3$ in few of them.

Key words. Matrix exponential, Taylor series, Paterson-Stockmeyer method, backward-error analysis .

AMS subject classifications. 65F30

1. Introduction.

Many engineering processes are described by systems of linear first-order ordinary differential equations. Solving this kind of systems involves the evaluation of the exponential of square matrices, and the same occurs with the partial differential case when using the semi-discretization method [FDC46, HS74, Kai80, ML78, Smi85]. These facts have motivated numerous studies on the computation of the matrix exponential. Many of the proposed methods have been reviewed by Moler and Van Loan [ML78] and recently studied exhaustively by Higham [Hig08, chap. 10]. Probably the more promising methods are those based on argument reduction, approximation and recovery and in particular on the scaling and squaring technique [Hig05, NH95], which is the most widely used.

Most exponential approximations with a scaled argument are based on truncated Taylor or Padé approximations. The truncation method for the Taylor series poses difficulties, specially in the classical implementation, as shown in [GL96, pp. 567] and [ML03, pp. 10]. Here the example is the computation of the matrix exponential of

$$A = \begin{bmatrix} -49 & 24 \\ -64 & 31 \end{bmatrix}, \tag{1.1}$$

[*]This work has been supported by the *Generalitat Valenciana* project GVPRE/2008/340.

[†]Instituto de Telecomunicaciones y Aplicaciones Multimedia, Universidad Politécniva Valencia, Camino de Vera s/n, 46022-Valencia (Spain) (jorsasma@iteam.upv.es).

[‡]Instituto de Aplicaciones de las Tecnologías de la Información y de las Comunicaciones Avanzadas, Camino de Vera s/n, 46022-Valencia (Spain) (jjibanez@dsic.upv.es).

[§]Instituto de Matemática Multidisciplinar, Universidad Politécnica de Valencia, Camino de Vera s/n, 46022-Valencia (Spain) (edefez@imm.upv.es).

[¶]Instituto de Aplicaciones de las Tecnologías de la Información y de las Comunicaciones Avanzadas, Camino de Vera s/n, 46022-Valencia (Spain) (pruiz@dsic.upv.es).

using only six decimal digits in the computations, which gives

$$\begin{bmatrix} -22.25880 & -1.4322766 \\ -61.49931 & -3.474280 \end{bmatrix}, \tag{1.2}$$

needing 59 terms for convergence. However, the approximate value of e^A is

$$e^A = \begin{bmatrix} -0.73584 & 0.55189 \\ -1.47169 & 1.10377 \end{bmatrix}.$$

The large difference between both results is due to the cancellation of rounding errors [Hig02, pp. 19], which is produced when matrices of similar magnitude but different sign are added. For example in this case it happens with the terms $A^{16}/16!$ and $A^{17}/17!$, showing that this Taylor approach is only useful near the origin.

In this paper, a backward-error analysis of the Taylor method with scaling and squaring has been provided to find the optimum matrix scaling. It uses exact arithmetic and depends on the matrix norm, following the analysis made in [Hig05]. We also use a similar approach to the Paterson-Stockmeyer method [PS73] in order to evaluate the partial Taylor sums which avoids calculation and division by factorials. Numerical tests show that this modification of the usual Paterson-Stockmeyer scheme improves the accuracy of the method in floating point arithmetics.
In addition, tests have been carried out comparing our method with the MATLAB functions `expm` [Hig05] and `funm` [DH03]. These tests show that the new scaling-squaring Taylor method is more accurate than the other methods in a high percentage of the cases.
Throughout this paper $\mathbb{R}^{n\times n}$ denotes the set of real matrices of size $n \times n$, and I denotes the identity matrix for these sets. The matrix norm $\|\cdot\|$ denotes any subordinate matrix norm, in particular $\|\cdot\|_1$ is the 1-norm, and $\lceil x \rceil$ denotes the least integer not less than x.
This paper is organized as follows. Section 2 presents the scaling and squaring Taylor error analysis. Section 3 deals with the algorithms and numerical experiments. Conclusions are given in Section 4.

2. Error analysis and algorithm.

If we denote the truncated matrix exponential Taylor series by

$$T_m(A) = \sum_{i=0}^{m} \frac{A^i}{i!} , \tag{2.1}$$

in an analogous way to the demonstration of Theorem 2.1 of [Hig05, pp. 1182] it follows the next result:

THEOREM 2.1. *Let the matrix exponential Taylor approximation $T_m(A)$ of (2.1) satisfy*

$$e^{-2^{-s}A} T_m(2^{-s}A) = I + G , \tag{2.2}$$

where $\|G\| < 1$. Then

$$\left[T_m(2^{-s}A)\right]^{2s} = e^{A+E} , \tag{2.3}$$

where E commutes with A and

$$\frac{\|E\|}{\|A\|} \leq \frac{-\log\left(1-\|G\|\right)}{\|2^{-s}A\|} \ . \qquad \square \tag{2.4}$$

We are bounding the norm of G in (2.2) in terms of $\|2^{-s}A\|$. Define the function

$$\rho\left(A\right) = e^{-A}\ T_m(A) - I \ , \tag{2.5}$$

and note that

$$\rho\left(A\right) = -e^{-A}\left(e^{A} - T_m(A)\right) = -e^{-A}\sum_{i\geq m+1}\frac{A^i}{i!} \ . \tag{2.6}$$

Hence

$$\rho\left(A\right) = \left(\sum_{j\geq 0}\frac{(-1)^j}{j!}A^j\right)\left(\sum_{i\geq m+1}\frac{A^i}{i!}\right) = \sum_{k\geq m+1} c_k A^k \ , \tag{2.7}$$

and then from (2.2) one obtains

$$\|G\| = \left\|\rho\left(2^{-s}A\right)\right\| \leq \sum_{k\geq m+1}\left|c_k\right|\theta^k := f(\theta) \ , \tag{2.8}$$

where $\theta := \|2^{-s}A\|$.

Using (2.4) and (2.8) it follows that

$$\frac{\|E\|}{\|A\|} \leq \frac{-\log\left(1-f(\theta)\right)}{\theta} \ . \tag{2.9}$$

Using MATLAB Symbolic Math Toolbox we have evaluated $f(\theta)$ in 250 decimal digit arithmetic, summing the first 200 terms of the series where the c_k in (2.8) are obtained symbolically. Analogously to [Hig05, pp. 1184] for $m = 3, 4, \ldots, 42$ we have used a zero-finder to determine the largest value of θ, denoted by θ_m, such that the backward error bound (2.9) does not exceed the unit roundoff in IEEE double precision arithmetic, i.e. $u = 2^{-53}$. Table 2.1 shows that $\theta_m = ||2^{-s}A||$ increases with m, therefore the matrix scaling needed decreases with m, saving matrix multiplications in the squaring phase.

Next we determine the cost of evaluating $T_m(A)$ in terms of matrix products. The method developed by Paterson-Stockmeyer [Loa79, PS73] allows the efficient computation of polynomial matrix functions. This method generalizes the Horner method by making groups of two or more terms in matrix polynomial expressions. Applying this method it is easy to prove that we can evaluate the matrix polynomial $T_m(A)$ minimizing the number of matrix products and maximizing the order of the approximation for $m = 2, 4, 6, 9, 12, 16, 20, 25, 30, 36, 42, \cdots$, i.e. for $m = k^2$ and $m = k^2 - k$, $k = 2, 3, \ldots$, calculating the even matrix powers as $A^2 = AA$, $A^4 = A^2A^2$, $A^6 = A^4A^2, \ldots$, and the odd matrix powers as $A^{2k+1} = AA^{2k}$, $k = 1, 2, \cdots$. For computing the Taylor polynomial we have considered two approaches: A classical Paterson-Stockmeyer version consisting of evaluating Taylor polynomial coefficients as $p_k = \frac{1}{k!}$, $k = 0, 1, 2, \cdots$ (see Table 2.2), and the new approach which avoids

m	3	4	5	6	7	8	9
θ_m	$1.39e-5$	$3.40e-4$	$2.40e-3$	$9.07e-3$	$2.38e-2$	$4.99e-2$	$8.96e-2$
m	10	11	12	13	14	15	16
θ_m	$1.44e-1$	$2.14e-1$	$3.00e-1$	$4.00e-1$	$5.14e-1$	$6.41e-1$	$7.80e-1$
m	17	18	19	20	21	22	23
θ_m	$9.31e-1$	1.09	1.26	1.44	1.62	1.82	2.01
m	24	25	26	27	28	29	30
θ_m	2.22	2.43	2.64	2.86	3.08	3.31	3.54
m	31	32	33	34	35	36	37
θ_m	3.77	4.01	4.25	4.49	4.73	4.97	5.22
m	38	39	40	41	42		
θ_m	5.47	5.72	5.97	6.22	6.48		

TABLE 2.1
Maximal values θ_m of $\|2^{-s}A\|$ such that the backward error bound (2.9) does not exceed $u = 2^{-53}$.

m	$T_m(A)$	matrices in memory
2	$\left(\frac{A}{2!}+I\right)A+I$	A
4	$\left(\frac{A^2}{4!}+\frac{A^2}{3!}+\frac{I}{2!}\right)A^2+A+I$	A^2, A
6	$\left(\left(\frac{A^2}{6!}+\frac{A^2}{5!}+\frac{I}{4!}\right)A^2+\frac{A}{3!}+\frac{I}{2!}\right)A^2+A+I$	A^2, A
9	$\left(\left(\frac{A^3}{9!}+\frac{A^2}{8!}+\frac{A}{7!}+\frac{I}{6!}\right)A^3+\frac{A^2}{5!}+\frac{A}{4!}+\frac{I}{3!}\right)A^3+\frac{A^2}{2!}+A+I$	A^3, A^2, A
12	$\left(\left(\left(\frac{A^3}{12!}+\frac{A^2}{11!}+\frac{A}{10!}+\frac{I}{9!}\right)A^3+\frac{A^2}{8!}+\frac{A}{7!}+\frac{I}{6!}\right)A^3+\right.$ $\left.+\frac{A^2}{5!}+\frac{A}{4!}+\frac{I}{3!}\right)A^3+\frac{A^2}{2!}+A+I$	A^3, A^2, A

TABLE 2.2
Evaluation of $T_m(A)$ for $m = 2, 4, 6, 9, 12$.

computing factorials as shown in Table 2.3. Both methods have the same cost in terms of matrix products, and have the same accuracy in exact arithmetic. However numerical tests in next section show that the precision of the second method is higher in floating point arithmetic in the majority of cases. We have evaluated $T_m(A)$, $m =$ 16, 20, 25, 30, 36, 42 in a similar way as in Tables 2.2 and 2.3. Note that increasing m from one of these values to the next requires an extra matrix multiplication in the evaluation of $T_m(A)$, but the greater corresponding value of θ_m compensates this if it increases by more than a factor of 2, saving one matrix multiplication in the squaring phase. Hence, taking into account the values of θ_m in Table 2.1, in terms of computational cost the optimum choice of m is $m = 16$.

Following [Hig05, pp. 1184], we present another way to obtain this optimum value. Noting that $s = \lceil \log_2 ||A||/\theta_m \rceil$ if $||A|| \geq \theta_m$ and $s = 0$ otherwise, the cost of the algorithm in matrix multiplications is

$$\pi_m + s = \pi_m + \max(\lceil \log_2 ||A|| - \log_2 \theta_m \rceil, 0)\,, \tag{2.10}$$

where π_m denotes the number of necessary matrix products. Considering $||A|| \geq \theta_m$ and ignoring the constant shift $||A||$ we have to minimize

$$C_m = \pi_m - \log_2 \theta_m\,, \tag{2.11}$$

in order to obtain the best choice for m in the approximation $T_m(A)$. Table 2.4

m	$T_m(A)$	matrices in memory
2	$\left(\frac{A}{2}+I\right)A+I$	A
4	$\left(\left(\frac{A^2}{4}+A\right)/3+I\right)\frac{A^2}{2}+A+I$	A^2, A
6	$\left(\left(\left(\left(\frac{A^2}{6}+A\right)/5+I\right)\frac{A^2}{4}+A\right)/3+I\right)\frac{A^2}{2}+A+I$	A^2, A
9	$(((((((\frac{A^3}{9}+A^2)/8+A)/7+I)\frac{A^3}{6}+$ $A^2)/5+A)/4+I)\frac{A^3}{3}+A^2)/2+A+I$	A^3, A^2, A
12	$((((((((((\frac{A^3}{12}+A^2)/11+A)/10+I)\frac{A^3}{9}+A^2)/8+$ $+A)/7+I)\frac{A^3}{6}+A^2)/5+A)/4+I)\frac{A^3}{3}+A^2)/2+A+I$	A^3, A^2, A

TABLE 2.3
Evaluation of $T_m(A)$ for $m = 2, 4, 6, 9, 12$.

presents the C_m values for $m = 3, 4, \ldots, 42$ and shows that C_m has an absolute minimum for $m = 16$ and local minimums for the optimal values of m in terms of number of matrix products: $m = 4$, 6, 9, 12, 20, 25, 30, 36, 42. Table 2.5 presents the corresponding values of θ_m in IEEE double precision arithmetic and a comparison with the same values for Padé method proposed in [Hig05]. It is important to note that C_{20}, C_{25} or even C_{30} are not much larger than C_{16}. Therefore the best compromise between precision and efficiency might be obtained by increasing the maximum degree to $m = 20$, 25 or 30.

With respect to rounding errors we rule out $m = 0, 1, 2$ as $T_m(A)$ can suffer from loss of significance in floating point arithmetic [Hig05, pp. 1184]. Also, there is no point in using $m = 3$ since the cost of evaluating the more accurate $T_4(A)$ is the same as the cost of evaluating $T_3(A)$ in terms of matrix products.

The effect of rounding errors on the evaluation of the matrix polynomial $T_m(A)$ can be bounded analogously to the numerator of Padé approximants in [Hig05, pp. 1185]. Using Theorem 2.2 of [Hig05, pp. 1184], taking into account that $\|A\|_1 \leq \theta_m$ and $e^{-||A||} \leq ||e^A||$, and noting that $T_m(A)$ has all positive coefficients, it follows that

$$\begin{aligned}
\left\|T_m(A) - \hat{T}_m(A)\right\|_1 &\leq \tilde{\gamma}_{mn} T_m\left(\|A\|_1\right)\\
&\approx \tilde{\gamma}_{mn} e^{\|A\|_1}\\
&\leq \tilde{\gamma}_{mn}\left\|e^A\right\|_1 e^{2\|A\|_1}\\
&\simeq \tilde{\gamma}_{mn}\left\|T_m(A)\right\|_1 e^{2\|A\|_1}\\
&\leq \tilde{\gamma}_{mn}\left\|T_m(A)\right\|_1 e^{2\theta_m},
\end{aligned}$$

where $A \in \mathbb{C}^{n\times n}$, $\hat{T}_m(A)$ is the computed Taylor approximation using explicit formation of matrix powers as in Tables 2.2 and 2.3 and $\tilde{\gamma}_{mn}$ can be derived from (1.1) of [Hig05, pp. 1180] using techniques of [Hig02].

Hence, the relative error is bounded approximately by $\tilde{\gamma}_{mn}e^{2\theta_m}$, which is a satisfactory bound taking into account the values of θ_m given in Table 2.5.

Algorithm 1 (`dgeexftay`) computes, for $\underline{\text{d}}$ouble precision $\underline{\text{ge}}$neral matrices, the $\underline{\text{ex}}$ponential $\underline{\text{f}}$unction by scaling-squaring $\underline{\text{Tay}}$lor approximation using the classical Paterson-Stockmeyer method. Algorithm 2 (`dgeexftaw`) computes, for $\underline{\text{d}}$ouble preci-

m	π_m	C_m	Matrices in memory
3	2	18.14	A
4	2	13.52	A^2, A
5	3	11.70	A^2, A
6	3	9.79	$\cdots$
7	4	9.39	$\cdots$
8	4	8.32	A^2, A
9	4	7.48	A^3, A^2, A
10	5	7.79	A^3, A^2, A
11	5	7.22	A^3, A^2, A
12	5	6.74	$\cdots$
13	6	7.32	$\cdots$
14	6	6.96	$\cdots$
15	6	6.64	A^3, A^2, A
16	6	6.36	$A^4, ..., A$
17	7	7.1039	$\cdots$
18	7	6.8745	$A^4, ..., A$
19	7	6.67	$A^4, ..., A$
20	7	6.48	$\cdots$
21	8	7.30	$\cdots$
22	8	7.14	$\cdots$
23	8	6.99	$\cdots$
24	8	6.85	$A^4, ..., A$
25	8	6.72	$A^5, ..., A$
26	9	7.60	$\cdots$
27	9	7.48	$A^5, ..., A$
28	9	7.38	$\cdots$
29	9	7.27	$\cdots$
30	9	7.18	$A^5, ..., A$
31	10	8.08	$A^6, ..., A$
32	10	8.00	$\cdots$
33	10	7.91	$\cdots$
34	10	7.83	$\cdots$
35	10	7.76	$A^6, ..., A$
36	10	7.69	$\cdots$
37	11	8.62	$\cdots$
38	11	8.55	$\cdots$
39	11	8.48	$\cdots$
40	11	8.42	$\cdots$
41	11	8.36	$\cdots$
42	11	8.31	$A^6, ..., A$

TABLE 2.4
Number of matrix multiplications, π_m, measure of overall cost C_m and matrices in memory.

TABLE 2.5
Theoretical optimal values of θ_m and parameter comparison between Taylor method and Padé `expm` *method (m = order of approximation, π_m = number of matrix products).*

Taylor			Padé		
m	θ_m	π_m	m	θ_m	π_m
4	3.397168839976 86e-4	2	3	1.495585217958292e-2	2
6	9.065656407595102e-3	3	5	2.539398330063230e-1	3
9	8.957760203223344e-2	4	7	9.504178996162932e-1	4
12	0.2996158913811581	5	9	2.097847961257068	5
16	0.7802874256626574	6	13	5.371920351148152	6
20	1.4382525968043369	7			
25	2.4285825244428265	8			
30	3.5396663487436890	9			

sion general matrices, the exponential function by scaling-squaring Taylor approximation without computing factorials. These algorithms can be divided into the following stages:

1. The truncation and rounding errors caused by the Taylor series approach increase with the norm of matrix A. Hence it is desirable to reduce the matrix norm by using the techniques proposed by Ward in [War77]:

Algorithm 1 computes the exponential of a matrix by means of Taylor series with scaling and squaring using the classical Paterson-Stockmeyer method.

```
    Function $F = \texttt{dgeexftay}(A)$
    Inputs: Matrix $A \in \mathbb{C}^{n\times n}$
    Output: Matrix $F \cong e^A \in \mathbb{C}^{n\times n}$
 1: $\mu = \text{trace}(A)/n$
 2: $A \leftarrow A - \mu I$
 3: Determine a diagonal matrix $D$ and a permutation matrix $P$ such that
    $D^{-1}P^TAPD$ is balanced
 4: $A \leftarrow D^{-1}P^TAPD$                                ▷ Preprocessing of $A$
 5: Initialize $\theta$ with Taylor $\theta_m$ values of Table 2.5, and coefficients $p_k = 1/k!$, $k =$
    $2, 3, \cdots, 30$
 6: $normA = ||A||_1$
 7: for $m = [4\ 6\ 9\ 12\ 16\ 20\ 25\ 30]$ do
 8:     if $normA \leq \theta_m$ then
 9:         break
10:     end if
11: end for
12: if $m == 30$ then
13:     $[w, s] = \log_2(normA/\theta_m)$
14:     $s = s - (w == 0.5)$                    ▷ adjust $s$ if $normA/\theta_m$ is a power of 2
15:     if $normA/2^s \leq \theta_{25}$ then     ▷ This condition can occur because $\theta_{30}/2 < \theta_{25}$
16:         m==25
17:     end if
18:     $A \leftarrow A/2^s$                                    ▷ Scaling matrix $A$
19: else
20:     $s = 0$
21: end if
22: $A_2 = A^2$
23: if $m == 4$ then
24:     $F = (A_2p_4 + Ap_3 + Ip_2)\,A_2 + A + I$
25: else if $m == 6$ then
26:     $F = ((A_2p_6 + Ap_5 + Ip_4)A_2 + Ap_3 + Ip_2)A_2 + A + I$
27: else if $m == 9$ then
28:     $A_3 = A \cdot A^2$
29:     $F = (((A_3p_9 + A_2p_8 + Ap_7 + Ip_6)A_3 + A_2p_5 + Ap_4 + Ip_3)A_3 + A_2p_2 + A + I$
30: else if $m == 12$ then
31:     ..............................
32: end if
33: for $k = 1 : s$ do                               ▷ Repeated squaring matrix $F$
34:     $F = F^2$
35: end for
36: $F \leftarrow e^{\mu}PDFD^{-1}P^T$                         ▷ Postprocessing of $F$
```

- The first technique attempts to minimize the matrix norm over all possible translations $A - \mu I$ such that $\|A - \mu I\|$ is minimized. Because it is difficult to compute the optimal value μ, the following approximation

Algorithm 2 computes the exponential of a matrix by scaling and squaring Taylor approach without computing factorials.

```
    Function $F =$ dgeexftaw$(A)$
    Inputs: Matrix $A \in \mathbb{C}^{n\times n}$
    Output: Matrix $F \cong e^A \in \mathbb{C}^{n\times n}$
 1: $\mu = \mathrm{trace}(A)/n$
 2: $A \leftarrow A - \mu I$
 3: Determine a diagonal matrix $D$ and a permutation matrix $P$ such that
    $D^{-1}P^TAPD$ is balanced
 4: $A \leftarrow D^{-1}P^TAPD$                                ▷ Preprocessing of $A$
 5: Initialize $\theta$ with values of Table 2.5
 6: $normA = ||A||_1$
 7: for $m = [4\ 6\ 9\ 12\ 16\ 20\ 25\ 30]$ do
 8:     if $normA \le \theta_m$ then
 9:         break
10:     end if
11: end for
12: if $m == 30$ then
13:     $[w, s] = \log_2(normA/\theta_m)$
14:     $s = s - (w == 0.5)$                ▷ adjust $s$ if $normA/\theta_m$ is a power of 2
15:     if $normA/2^s \le \theta_{25}$ then   ▷ This condition can occur because $\theta_{30}/2 < \theta_{25}$
16:         m==25
17:     end if
18:     $A \leftarrow A/2^s$                                   ▷ Scaling matrix $A$
19: else
20:     $s = 0$
21: end if
22: $A_2 = A^2$
23: if $m == 4$ then
24:     $F = ((A_2/4 + A)/3 + I)A_2/2 + A + I$
25: else if $m == 6$ then
26:     $F = ((((A_2/6 + A)/5 + I)A_2/4 + A)/3 + I)A_2/2 + A + I$
27: else if $m == 9$ then
28:     $A_3 = A \cdot A^2$
29:     $F = (((((((A_3/9+A_2)/8+A)/7+I)A_3/6+A_2)/5+A)/4+I)A_3/3+A_2)/2+A+I$
30: else if $m == 12$ then
31:     ..............................
32: end if
33: for $k = 1 : s$ do                            ▷ Repeated squaring matrix $F$
34:     $F = F^2$
35: end for
36: $F \leftarrow e^{\mu}PDFD^{-1}P^T$                        ▷ Postprocessing of $F$
```

can be used: $\mu = \mathrm{trace}(A)/n$, where $\mathrm{trace}(A) = \sum_{i=1}^{n} a_{ii}$.

- The second technique attempts to minimize $\|A\|_1$ over all possible diagonal similarity transformations. This may be done by balancing the above matrix, finding a permutation matrix P and a diagonal matrix D

so that the 1-norm of $D^{-1}P^TAPD$ is minimal.

2. After applying the balancing transformation 1, the optimal value s (scaling) is calculated (steps 5-17).
3. In steps 18 and 33-35 the scaling and squaring of Taylor approximation is done, respectively.
4. Finally in step 36 the postprocessing is applied.

It is important to note that both algorithms have the same cost in terms of matrix multiplications.

3. Numerical experiments. Numerous tests have been carried out in order to show the accuracy of Taylor method developed in the previous section. Two MATLAB implementations of this method were tested on an Intel Core 2 Duo T5600 with 2 GB main memory, using MATLAB 7.7. This implementations were compared to MATLAB functions `expm` and `funm`.

- `Expm` is a MATLAB 7.7 function that uses the Padé approximation of exponential function with scaling and squaring proposed by Higham in [Hig05].
- `Funm` is a built-in MATLAB 7.7 function that enables computation of the exponential, logarithm, cosine, sine, hyperbolic sine and hyperbolic cosine of a matrix. This function implements the Schur-Parlett method of Davies and Higham [DH03].

`Funm` was shown to be less effective than `expm` in [Hig05] with examples of matrices of sizes $n \times n$ from $n = 2$ to $n = 10$. However next comparison is made with matrices of dimensions from $n = 2$ to $n = 100$. Algorithm accuracy was tested by computing the relative error

$$\mathrm{Er} = \frac{\left\| e^A - \tilde{X} \right\|_1}{\|e^A\|_1},$$

where $\tilde{X}$ is the computed solution and e^A the exact solution.

In the tests we did not use any preprocessing/postprocessing in Algorithms 1 and 2. Analogously to the experiments in [Hig05], we found that turning on preprocessing in this algorithm provides similar results to those presented in this section.

Regarding memory issues, it is important to note that Algorithms 1 and 2 need the same matrices in memory as `expm` when both methods use their maximum orders, $m = 30$ and $m = 13$ respectively: A^5, A^4, A^3, A^2, A plus one to perform the calculation for both Taylor methods, and A^6, A^4, A^2, A plus two for the numerator and denominator for Padé method, taking into account that the final rational approximation can be performed re-using the memory allocated for the power of A involved in the numerator and denominator computation.

Regarding computational cost, from Tables 2.4 and [Hig05], Table 2.5 presents the orders of the approximation, the θ_m values and the number of matrix products required for both Taylor `dgeexftaw` and `dgeexftay` methods, and Padé `expm` method. Using (2.10), for matrices with $||A|| > 5.371920351148152$, the cost of both Taylor methods in terms of matrix products, denoted by C_m^T, is

$$C_{30}^T = 9 + s_T = 9 + \lceil \log_2 ||A|| - \log_2(3.5396663487436890) \rceil, \text{if } \frac{||A||}{2^{s_T}} > \theta_{25}, \quad (3.1)$$

$$C_{25}^T = 8 + s_T = 8 + \lceil \log_2 ||A|| - \log_2(3.5396663487436890) \rceil, \text{if } \frac{||A||}{2^{s_T}} \le \theta_{25}, \quad (3.2)$$

where s_T denotes the scaling in Taylor methods, and the cost of `expm` Padé method, denoted by C_m^P, is

$$C_{13}^P = 6 + C_{LS} + s_P = 6 + C_{LS} + \lceil \log_2 ||A|| - \log_2(5.371920351148152) \rceil, \qquad (3.3)$$

where

$$s_P = \lceil \log_2 ||A|| - \log_2(5.371920351148152) \rceil, \qquad (3.4)$$

denotes the scaling in Padé `expm` method and C_{LS} denotes the cost of solving the multiple right-hand sides linear system in Padé method, in terms of matrix products. From [BD99] the costs of the matrix product in $\mathbb{R}^{n\times n}$ and the solution of the multiple right-hand sides of the same size with Padé approximants is $2n^3 - n^2$ and $\frac{8n^3}{3} - \frac{n^2}{2} + \frac{5n}{6}$ flops, respectively. Therefore, asymptotically

$$C_{LS} \approx 4/3\,. \qquad (3.5)$$

Taking into account that θ_{30} for Taylor method is greater than $\theta_{13}/2$ for Padé method [Hig05, p. 1186], taking `expm`'s matrix scaling by 2^{s_P}, for matrices with

$$5.371920351148152/2 = 2.685960175574076 < \frac{||A||}{2^{s_P}} \leq 3.5396663487436890, \qquad (3.6)$$

Taylor methods use $m = 30$ and $s_T = s_P$, therefore they need $3 - C_{LS}$ more matrix products than `expm`, which results in a relative higher cost of $(1 + 2/3)/(7 + 1/3 + s_P) \times 100\% \leq 20\%$ in matrix products. For matrices such that

$$3.5396663487436890 < \frac{||A||}{2^{s_P}} \leq 2 \times 2.4285825244428265 = 4.8571650488856530, \qquad (3.7)$$

Taylor methods use $m = 25$ and $s_T = s_P + 1$, therefore they need $3 - C_{LS}$ more matrix products than `expm` again, resulting in the same relative higher cost $(1 + 2/3)/(7 + 1/3 + s_P) \times 100\% \leq 20\%$ in matrix products. Finally, for matrices such that

$$4.8571650488856530 < \frac{||A||}{2^{s_P}} \leq 5.371920351148152, \qquad (3.8)$$

Taylor methods use $m = 30$ and $s_T = s_P + 1$, therefore they need $4 - C_{LS}$ more matrix products than `expm`, resulting in a relative higher cost of $(2 + 2/3)/(7 + 1/3 + s_P) \times 100\% \leq 32\%$ in matrix products. Note that this norm interval represents only the 19.10% of the total interval considered in the three cases (3.6), (3.7) and (3.8).

Table 3.1 presents the cost comparison for matrices with $||A|| \leq 5.371920351148152$, where the θ_m values are presented with three truncated significant figures. Note that there are some cases where Taylor method cost is lower than Padé method, i.e. $2.53e-1 < ||A|| \leq 2.99e-1$, $1.49e-2 < ||A|| \leq 8.95e-2$ and $||A|| \leq 9.06e-3$, reaching relative efficiency gains from 6.25% up to 40%. For matrices satisfying $4.85 \leq ||A|| \leq 5.37$ the cost of Taylor methods is 36.36% higher and in the rest of cases Taylor method cost exceeds `expm` cost in $2/3$ or $1+2/3$ matrix products. Another time the interval where the cost is higher in 36.36%, i.e. $2 + 2/3$ matrix multiplications, is a small part of all the interval considered, representing only the 9.58% of the total. One more final squaring step than in `expm` is required for matrices satisfying $3.53 < ||A||/2^{s_P} \leq 5.37$ with $||A|| > 5.37$, and $3.53 < ||A|| \leq 5.37$. This could be a possible source of error especially if A is ill-conditioned. However,

TABLE 3.1
Cost comparison between Taylor methods with maximum order $m = 30$ and Padé `expm` *method for $||A|| \leq 5.371920351148152$.*

	Taylor			Padé			
Norm limits	m	s_T	C_m^T	m	s_P	C_m^P	$\frac{C_m^T - C_m^P}{C_m^P}$ %
$2 \times 2.42 < \|\|A\|\| \leq 5.37$	30	1	10	13	0	6+C_{LS}	36.36
$3.53 < \|\|A\|\| \leq 2 \times 2.42$	25	1	9	13	0	6+C_{LS}	22.73
$2.42 < \|\|A\|\| \leq 3.53$	30	0	9	13	0	6+C_{LS}	22.73
$2.09 < \|\|A\|\| \leq 2.42$	25	0	8	13	0	6+C_{LS}	9.09
$1.43 < \|\|A\|\| \leq 2.09$	25	0	8	9	0	5+C_{LS}	26.32
$9.50e-1 < \|\|A\|\| \leq 1.43$	20	0	7	9	0	5+C_{LS}	10.53
$7.80e-1 < \|\|A\|\| \leq 9.50e-1$	20	0	7	7	0	4+C_{LS}	31.25
$2.99e-1 < \|\|A\|\| \leq 7.80e-1$	16	0	6	7	0	4+C_{LS}	12.50
$2.53e-1 < \|\|A\|\| \leq 2.99e-1$	12	0	5	7	0	4+C_{LS}	-6.25
$8.95e-2 < \|\|A\|\| \leq 2.53e-1$	12	0	5	5	0	3+C_{LS}	15.38
$1.49e-2 < \|\|A\|\| \leq 8.95e-2$	9	0	4	5	0	3+C_{LS}	-7.69
$9.06e-3 < \|\|A\|\| \leq 1.49e-2$	9	0	4	3	0	2+C_{LS}	20
$3.39e-4 < \|\|A\|\| \leq 9.06e-3$	6	0	3	3	0	2+C_{LS}	-10
$\|\|A\|\| \leq 3.39e-4$	4	0	2	3	0	2+C_{LS}	-40

Taylor method with maximum order $m = 30$ has obtained better precision than Padé method in a high percentage of cases in numerical tests, see sections 3.2 and 3.3.

In an analogous way, it is easy to show that if we use Taylor methods with maximum order $m = 25$ then the cost of Taylor approximation for $||A|| > 2.4285825244428265$ is

$$C_{25}^T = 8 + \lceil \log_2 ||A|| - \log_2(2.4285825244428265) \rceil, \text{if } ||A||/2^{s_T} > \theta_{20}, \quad (3.9)$$
$$C_{20}^T = 7 + \lceil \log_2 ||A|| - \log_2(2.4285825244428265) \rceil, \text{if } ||A||/2^{s_T} \leq \theta_{20}. \quad (3.10)$$

For that kind of matrices `expm` cost is $C_{13}^P = 6 + C_{LS} + s_P = 7 + 1/3 + s_P$, where $s_P = 0$ if $||A|| \leq 5.371920351148152$, and for $||A|| > 5.371920351148152$ it follows that

$$\begin{aligned} C_{20}^T &= 7 + s_T = 7 + 2 + s_p, \ 2 \times 2.42 < ||A||/2^{s_P} \leq 5.37, \\ C_{25}^T &= 8 + s_T = 8 + 1 + s_p, \ 2 \times 1.43 < ||A||/2^{s_P} \leq 2 \times 2.42, \\ C_{20}^T &= 7 + s_T = 7 + 1 + s_p, \ 5.37/2 < ||A||/2^{s_P} \leq 2 \times 1.43, \end{aligned}$$

giving a maximum higher relative cost of 20% for the first and second cases, and 8% for the third. On the other hand, for $2.4285825244428265 < ||A|| \leq 5.371920351148152$ one gets

$$\begin{aligned} C_{20}^T &= 7 + s_T = 7 + 2, \ 2 \times 2.42 < ||A|| \leq 5.37, \\ C_{25}^T &= 8 + s_T = 8 + 1, \ 2 \times 1.43 < ||A|| \leq 2 \times 2.42, \\ C_{20}^T &= 7 + s_T = 7 + 1, \ 2.42 < ||A|| \leq 1.43 \times 2, \end{aligned}$$

giving a higher relative cost of 22.73% for the first and second cases, and 9.09 for the third. For $||A|| \leq 2.4285825244428265$ the cost comparison of Taylor approximation with maximum order $m = 25$ is obviously the same as in Table 3.1. Hence, Taylor approximation with maximum order $m = 25$ presents a maximum higher cost than `expm` of 1+2/3 matrix multiplications, and lower cost than `expm` for matrices satisfying $2.53e-1 < ||A|| \leq 2.99e-1$, $1.49e-2 < ||A|| \leq 8.95e-2$ and $||A|| \leq 9.06e-3$, reaching the same savings as Taylor method with $m = 30$.

3.1. Case study 1. In this case study MATLAB implementation `dgeexftaw` (Algorithm 2) and the classical Taylor implementation are compared when six decimal digits precision are used (see MATLAB functions `vpa` and `digits`). In [ML03, pp. 10] the exponential of matrix A in (1.1) is computed giving the inaccurate result of (1.2). However, if we use `dgeexftaw` with maximum Taylor order $m = 30$, the approximate value of e^A is

$$E_2 = \begin{bmatrix} -0.73584 & 0.55189 \\ -1.47169 & 1.10377 \end{bmatrix}.$$

The relative error associated to the 1-norm of classical Taylor implementation is greater than 36, nevertheless the relative error associated to the 1-norm of `dgeexftaw` is lower than $1.14 \cdot 10^{-4}$.

3.2. Case study 2. In this case study the MATLAB implementation of Algorithm 2 (`dgeexftaw`) is compared to the implementation of Algorithm 1 (`dgeexftay`), and MATLAB functions `expm` and `funm`. The following matrices were used in tests:

- Matrices 1-48: Forty-eight 8×8 real matrices obtained from the function `matrix` of the Matrix Computation Toolbox [Hig93]. Function `matrix` enables the generation of fifty-two matrices. Three of these matrices were excluded because their exponentials cannot be represented in double precision due to overflow errors. Also, matrix 38 was excluded because it is not a real matrix for that dimension.
- Matrix 49: 3×3 real diagonal matrix from [NH95].
- Matrix 50: 2×2 real matrix from [DH03, example 2].
- Matrix 51: 5×5 real matrix from [KL98].
- Matrices 52-55: Three 3×3 and one 10×10 real matrices from [War77].
- Matrices 56-62: Seven 10×10 real matrices from [DP00].

When it was not possible to calculate analytically the matrix exponential, its "exact" value was computed using MATLAB's Symbolic Math Toolbox and a [33/33] diagonal Padé method with scaling and squaring at 1000-digit precision in an iterative way: different increasing scalings higher than that provided in [Hig05] for `expm` have been used until the norm of the relative difference between the approximations converted to IEEE double precision arithmetic has been zero. The [33/33] diagonal Padé approximation has been evaluated with matrix power aggregation similar to that proposed in [Hig05, p. 1183].

In Table 3.2 the relative errors of `dgeexftaw` and `dgeexftay` are compared using the matrices of this case study. The values that appear in this table are the percentages of times that the relative error of `dgeexftaw`(m) is lower, equal or greater than the relative error using `dgeexftay`(m), $m = 16, 20, 25, 30$. Results show that `dgeexftaw` is more accurate than `dgeexftay` for all orders m in the majority of cases.

Table 3.3 presents the same relative error comparative between `dgeexftaw` and functions `expm` and `funm`. It shows that the scaling-squaring Taylor implementation `dgeexftaw` with maximum order $m = 20$, $m = 25$ and $m = 30$ is generally more precise than `expm` ($83.87\% - 87.10\%$) and it is generally more precise than `funm` from order $m = 16$ ($72.58\% - 80.65\%$).

Table 3.2
Relative error comparison between `dgeexftaw` *and* `dgeexftay` *(case study 2): percentage of times that relative error of* `dgeexftaw`*(m) is lower, equal or greater than relative error of* `dgeexftay`*(m).*

	m=16	m=20	m=25	m=30
Lower	54.84	66.13	70.97	58.06
Equal	20.97	12.90	9.68	14.52
Greater	24.19	20.97	19.35	27.42

Table 3.3
Relative error comparison between `dgeexftaw` *and the MATLAB functions* `expm` *and* `funm` *(case study 2): percentage of times that the relative error of* `dgeexftaw`*(m) is lower than* `expm` *and* `funm` *relative errors.*

	m=16	m=20	m=25	m=30
`expm`	48.39	83.87	87.10	87.10
`funm`	72.58	80.65	80.65	80.65

Figure 3.1 shows the normwise relative errors of `dgeexftaw-expm-funm`. This figure shows the relative errors of all implementations, and a solid line that represents the unit roundoff multiplied by the relative condition number of the exponential function at X [Hig08, pp. 56]

$$\mathrm{cond}_{\exp}(X) = \lim_{\varepsilon\to 0} \sup_{\|E\|\le\varepsilon} \frac{\left\|e^{X+E} - e^{X}\right\|}{\varepsilon} \frac{\|X\|}{\|e^{X}\|}.$$

Relative condition number has been computed using the MATLAB function `expm_cond`. This function is incorporated into the Matrix Function Toolbox developed by Higham [Hig08, Appendix D] and available at

http://www.ma.man.ac.uk/~higham/mftoolbox.

For a method to perform in a backward and forward stable manner, its error should lie not far above this line on the graph [Hig05, p. 1188]. In figure 3.1 the matrices used in case study 2 are sorted by decreasing condition number. Figure 3.1 shows that all functions perform in a numerically stable way on this test, even for matrices 52-55 where overscaling problems are produced.

The performance profile [DM02] is presented in figure 3.2. This figure shows the performances of the compared functions, where α coordinate varies between 1 and 5 in steps equal to 0.1, and p coordinate is the probability that the considered method has a relative error lower than or equal to α-times the smallest error over all the methods, where probabilities are defined over all matrices. Figure 3.2 shows that for this case study `dgeexftaw` has better precision performance than the other MATLAB functions.

3.3. Case study 3. The main objective of this subsection is to compare the MATLAB implementations of Algorithm 2 (`dgeexftaw`) with the implementation of Algorithm 1 (`dgeexftay`) and MATLAB functions `expm` and `funm` when the matrix dimension is varied. In this case study forty-three matrices of dimension 25×25 and thirty-nine matrices of dimensions $n = 50, 75, 100$ from MATLAB function `matrix` of the Matrix Computation Toolbox were used as test battery (matrices whose exponential cannot be represented in double precision due to overflow errors were excluded). For computing the "exact" matrix exponentials MATLAB's Symbolic Math Toolbox with the same scaling-squaring [33/33] diagonal Padé method as in last section was

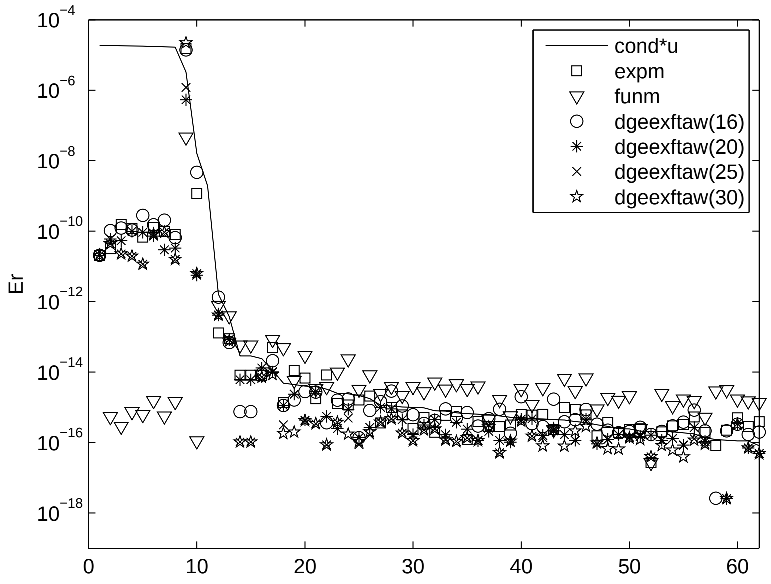

Fig. 3.1. *Normwise relative errors for functions* `dgeexftaw`*,* `expm` *and* `funm` *(case study 2).*

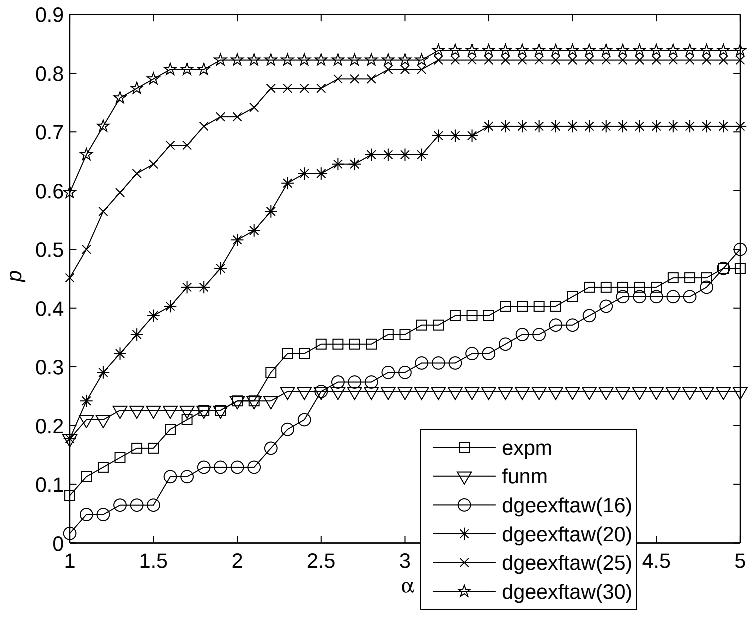

Fig. 3.2. *Performance profile for functions* `dgeexftaw`*,* `expm` *and* `funm` *(case study 2).*

used at 250-digit precision. Here a small relative error tolerance much lower than the IEEE double precision arithmetic unit roundoff $u = 2^{-53}$ was used to make the

Table 3.4
Percentage of times that the relative error of `dgeexftaw` is lower than the relative error of `dgeexftay` (case study 3).

	m=16	m=20	m=25	m=30
$n = 25$	74.42	74.42	81.40	74.42
$n = 50$	79.49	66.67	64.10	61.54
$n = 75$	74.36	66.67	66.67	64.10
$n = 100$	71.80	66.67	61.54	58.97

Table 3.5
Percentage of times that the relative error of `dgeexftaw` is lower than the relative error of `expm` (case study 3).

	m=16	m=20	m=25	m=30
n=25	23.26	69.77	86.05	93.02
n=50	28.21	46.15	74.36	92.31
n=75	25.64	41.03	71.79	79.49
n=100	20.51	35.90	56.41	76.92

Table 3.6
Percentage of times that the relative error of `dgeexftaw` is lower than the relative error of `funm` (case study 3).

	m=16	m=20	m=25	m=30
n=25	86.05	93.02	95.35	100.00
n=50	74.36	76.92	89.74	92.31
n=75	74.36	76.92	89.74	94.87
n=100	71.79	76.92	82.05	89.74

iterative algorithm converge faster.

Tables 3.4, 3.5 and 3.6 compare the relative errors of `dgeexftaw` with `dgeexftay`, `expm` and `funm`. The values that appear in each table have the same meaning as the pairs in Tables 3.2 and 3.3, corresponding to the percentage of times that the error committed by `dgeexftaw` is lower than that from the compared function. Those tables show that `dgeexftaw` is generally more accurate than `dgeexftay` and `funm` for all matrix dimensions and orders m reaching 81.40% and 100%, respectively. The same happens with `expm` for maximum orders $m = 25$ and $m = 30$, reaching a maximum of 93.02%, and for maximum order $m = 20$ with matrix dimension $n = 25$.

Figures 3.3 and 3.4 show the normwise relative errors of `dgeexftaw-expm-funm` for matrices of size $n \times n$ with $n = 25$ and $n = 50$. Due to memory problems of `expm_cond` the condition numbers of 75×75 and 100×100 matrices have not been computed. The figures show that all compared functions perform in a numerically stable way on this test.

Figures 3.5, 3.6, 3.7 and 3.8 show the performance profile when n varies between 25, 50, 75 and 100. `Dgeexftaw` with maximum order $m = 30$ is the most accurate function on the set of considered matrices, independently of the matrix dimension. The next more accurate function is `dgeexftaw` with maximum order $m = 25$. Hence, from obtained results we can conclude that `dgeexftaw` with maximum order $m = 30$ is the best choice in terms of accuracy.

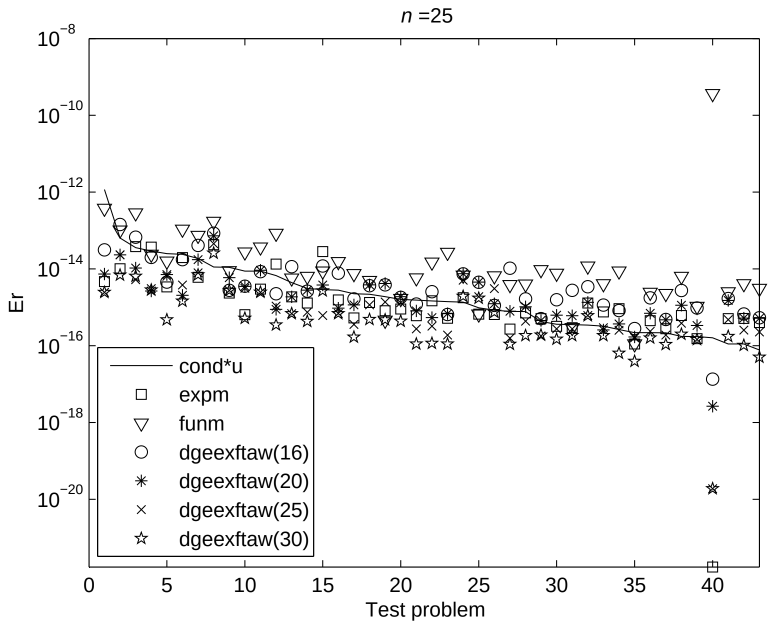

FIG. 3.3. *Normwise relative errors for matrices* 25 × 25 *of* **dgeexftaw**, **expm** *and* **funm** *(case study 3).*

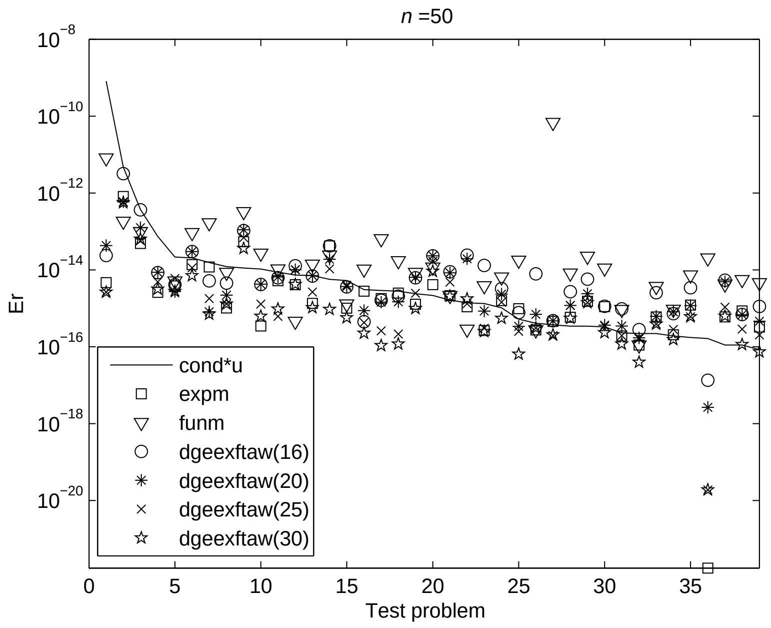

FIG. 3.4. *Normwise relative errors for matrices* 50 × 50 *of* **dgeexftaw**, **expm** *and* **funm**.

4. Conclusions. In this work an efficient and accurate method based on Taylor expansion has been developed. A modified Paterson-Stockmeyer algorithm (`dgeexftaw`)

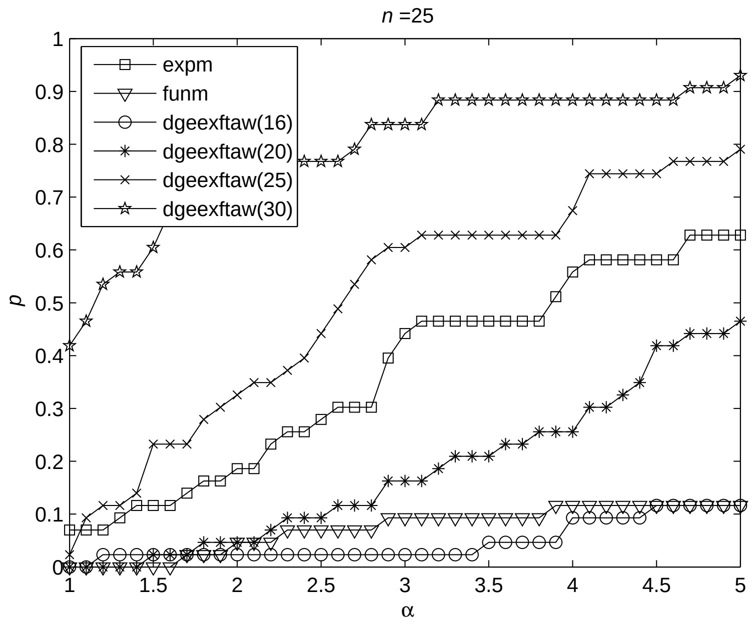

FIG. 3.5. *Performance profile for matrices* 25 × 25 *of* `dgeexftaw`*,* `expm` *and* `funm` *(case study 3).*

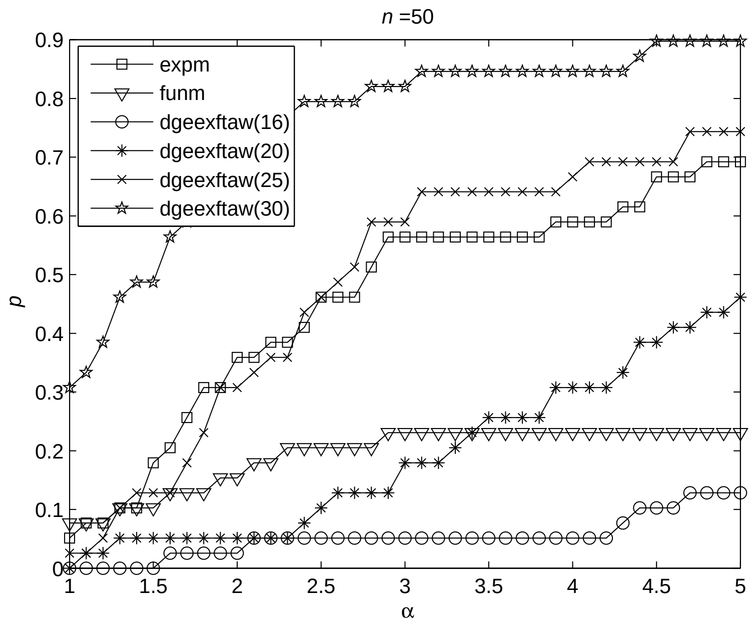

FIG. 3.6. *Performance profile for matrices* 50 × 50 *of* `dgeexftaw`*,* `expm` *and* `funm` *(case study 3).*

to evaluate different order matrix polynomial approximations, avoiding the calculation and division by factorials, together with the theoretical optimal scaling and squaring

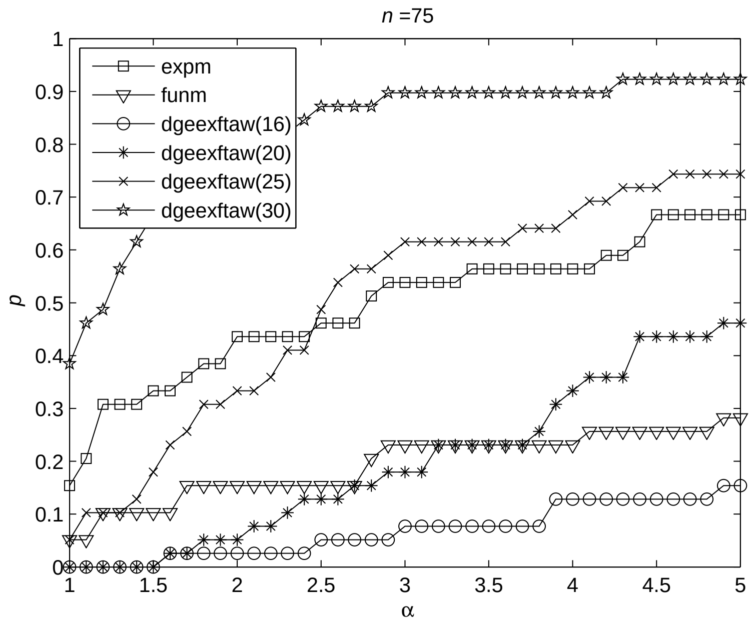

FIG. 3.7. *Performance profile for matrices* 75×75 *of* **dgeexftaw**, **expm**, *and* **funm** *(case study 3).*

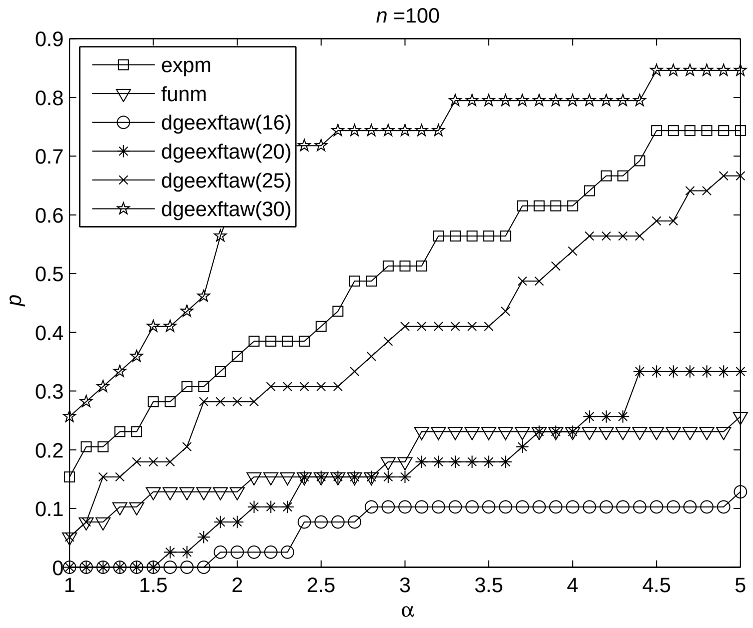

FIG. 3.8. *Performance profile for matrices* 100×100 *of* **dgeexftaw**, **expm** *and* **funm** *(case study 3).*

of the matrices, enable the implementation of a competitive method in comparison to the existing algorithms. Numerical tests show that the modified Paterson-Stockmeyer method is more effective than traditional Paterson-Stockmeyer schemes, and following the ideas of [Hig05] an optimal backward error bound for the scaling and squaring Taylor method in exact arithmetic has been developed, depending on A only through $\|A\|$. From experimental results we have identified the most efficient choice of degree m of Taylor approximation in terms of accuracy: $m = 30$. The backward error results provide an optimal scaling ensuring $\|A\| \leq 3.53$ for this Taylor order.

The algorithm stores the same number of matrices in memory as `expm` when both methods use their maximum orders, $m = 30$ and $m = 13$ respectively. It has lower cost than `expm` in terms of matrix products for matrices satisfying $2.53e-1 < ||A|| \leq 2.99e-1$, $1.49e-2 < ||A|| \leq 8.95e-2$ and $||A|| \leq 9.06e-3$, reaching relative computational cost gains from 6.25% up to 40%, and higher cost not exceeding 20% for matrices satisfying (3.6) and (3.7), 32% for matrices satisfying (3.8), where s_P is given by (3.4), and 36.36% if $4.85 \leq ||A|| \leq 5.37$. Both last matrix norm intervals where the cost is 32% and 36.36% represent only the 19.10% and the 9.58% of the total matrix norm intervals considered. Taylor method does not need the solution of multiple linear systems. [Hig05] shows that it does not introduce large errors in `expm`. However, `dgeexftaw` has obtained higher accuracy than both `funm` and `expm` in the majority of case study matrices, reaching values between 80.65% and 100% comparing to `funm`, and between 76.92% and 93.02% comparing to `expm`.

REFERENCES

[BD99] S. Blackford and J. Dongarra. Installation guide for LAPACK. LAPACK Working Note 411, Department of Computer Science University of Tenessee, 1999.

[DH03] P. I. Davies and N. J. Higham. A Schur–Parlett algorithm for computing matrix functions. *SIAM J. Matrix Anal. Appl.*, 25(2):464–485, 2003.

[DM02] E. D. Dolan and J. J. Moré. Benchmarking optimization software with performance profiles. *Math. Programming*, 91:201–213, 2002.

[DP00] L. Dieci and A. Papini. Padé approximation for the exponential of a block triangular matrix. *Linear Algebra Appl.*, 308:183–202, 2000.

[FDC46] R. A. Frazer, W. J. Duncan, and A. R. Collar. *Elementary Matrix and Some Applications to Dynamics and Differential Equations.* Macmillan, New York, 1946.

[GL96] G. H. Golub and C. Van Loan. *Matrix Computations.* Johns Hopkins Studies in Mathematical Sciences. The Johns Hopkins University Press, third edition, 1996.

[Hig93] N. J. Higham. The Test Matrix Toolbox for MATLAB. Numerical Analysis Report No. 237, Manchester, England, December 1993.

[Hig02] N. J. Higham. *Accuracy and Stability of Numerical Algorithms.* Society for Industrial and Applied Mathematics, Philadelphia, PA, USA, second edition, 2002.

[Hig05] N. J. Higham. The scaling and squaring method for the matrix exponential revisited. *SIAM J. Matrix Anal. Appl.*, 26(4):1179–1193, 2005.

[Hig08] N. J. Higham. *Functions of Matrices: Theory and Computation.* Society for Industrial and Applied Mathematics, Philadelphia, PA, USA, 2008.

[HS74] M. W. Hirsch and S. Smale. *Differential Equations, Dynamical Systems and Linear Algebra.* Academic Press, New York, 1974.

[Kai80] T. Kailath. *Linear Systems.* Prentice-Hall, Englewood Cliffs, NJ, 1980.

[KL98] C. S. Kenney and A. J. Laub. A Schur–Fréchet algorithm for computing the logarithm and exponential of a matrix. *SIAM J. Matrix Anal. Appl.*, 19(3):640–663, 1998.

[Loa79] C. Van Loan. A note on the evaluation of matrix polynomials. *IEEE Transactions on Automatic Control*, 24(2):320–321, 1979.

[ML78] C. B. Moler and C. Van Loan. Nineteen dubious ways to compute the exponential of a matrix. *SIAM Review*, 20(4):801–836, 1978.

[ML03] C. B. Moler and C. Van Loan. Nineteen dubious ways to compute the exponential of a matrix, twenty-five years later*. *SIAM Review*, 45:3–49, 2003.

[NH95] I. Najfeld and T. F. Havel. Derivatives of the matrix exponential and their computation. *Advances in Applied Mathematics*, 16:321–375, 1995.

[PS73] M. S. Paterson and L. J. Stockmeyer. On the number of nonscalar multiplications necessary to evaluate polynomials. *SIAM J. Comput.*, 2(1):60–66, 1973.

[Smi85] G. D. Smith. *Numerical Solution of Partial Differential Equations: Finite Difference Methods.* Oxford University Press, 3^{rd} edition, 1985.

[War77] R. C. Ward. Numerical computation of the matrix exponential with accuracy estimate. *SIAM J. Numer. Anal.*, 14(4):600–610, 1977.