

THE SCALING AND SQUARING METHOD FOR THE MATRIX EXPONENTIAL REVISITED*

NICHOLAS J. HIGHAM†

Abstract. The scaling and squaring method is the most widely used method for computing the matrix exponential, not least because it is the method implemented in MATLAB's `expm` function. The method scales the matrix by a power of 2 to reduce the norm to order 1, computes a Padé approximant to the matrix exponential, and then repeatedly squares to undo the effect of the scaling. We give a new backward error analysis of the method (in exact arithmetic) that employs sharp bounds for the truncation errors and leads to an implementation of essentially optimal efficiency. We also give new rounding error analysis that shows the computed Padé approximant of the scaled matrix to be highly accurate. For IEEE double precision arithmetic the best choice of degree of Padé approximant turns out to be 13, rather than the 6 or 8 used by previous authors. Our implementation of the scaling and squaring method always requires at least two fewer matrix multiplications than `expm` when the matrix norm exceeds 1, which can amount to a 37% saving in the number of multiplications, and it is typically more accurate, owing to the fewer required squarings. We also investigate a different scaling and squaring algorithm proposed by Najfeld and Havel that employs a Padé approximation to the function $x \coth(x)$. This method is found to be essentially a variation of the standard one with weaker supporting error analysis.

Key words. matrix function, matrix exponential, Padé approximation, matrix polynomial evaluation, scaling and squaring method, MATLAB, `expm`, backward error analysis, performance profile

AMS subject classification. 65F30

DOI. 10.1137/04061101X

1. Introduction. The matrix exponential is a much-studied matrix function, owing to its key role in the solution of differential equations. Computation of e^A is required in applications such as nuclear magnetic resonance spectroscopy [8], [18], control theory [5], and Markov chain analysis [20]. Motivated by the applications, mathematicians and engineers have produced a large amount of literature on methods for computing e^A .

A wide variety of methods for computing e^A were analyzed in the classic paper of Moler and Van Loan [16], which was reprinted with an update in [17]. The conclusion of the paper was that there are three or four candidates for best method. One of these, the scaling and squaring method, has become by far the most widely used, not least because it is the method implemented in MATLAB.

In this work we take a fresh look at the scaling and squaring method, giving a sharp analysis of truncation errors and a careful treatment of computational cost. We derive a new implementation that has essentially optimal efficiency and show that it requires at least one less matrix multiplication than existing implementations, including that in MATLAB. Our analysis and implementation are presented in section 2. Section 3 contains a comparison with existing implementations and numerical experiments. The new implementation is found to be typically more accurate than

*Received by the editors July 5, 2004; accepted for publication (in revised form) by I. C. F. Ipsen September 30, 2004; published electronically June 3, 2005. This work was supported by Engineering and Physical Sciences Research Council grant GR/T08739 and by a Royal Society-Wolfson Research Merit Award.

<http://www.siam.org/journals/simax/26-4/61101.html>

†School of Mathematics, The University of Manchester, Sackville Street, Manchester, England M60 1QD (higham@ma.man.ac.uk, <http://www.ma.man.ac.uk/~higham/>).

the existing ones, owing to the fact that it usually requires fewer matrix squarings. This work therefore provides another example of the phenomenon, illustrated in the work of Dhillon and Parlett [2], for example, that speed and accuracy are not always conflicting goals in matrix computations.

The standard scaling and squaring method employs Padé approximants to e^x . Najfeld and Havel [18] propose a variation using Padé approximants to the function $x \coth(x)$ instead, and they argue that this approach is more efficient than direct Padé approximation. In section 4 we show that the proposed method is essentially a variation of the standard method, but with weaker supporting error analysis, both in exact arithmetic and in floating point arithmetic.

For other recent work on the scaling and squaring method, concerned particularly with arbitrary precision computations, see Sofroniou and Spaletta [21].

Throughout this paper, $\|\cdot\|$ denotes any subordinate matrix norm. We use the standard model of floating point arithmetic with unit roundoff u [11, sec. 2.2]. Our rounding error bounds are expressed in terms of the constants

$$(1.1) \quad \gamma_k = \frac{ku}{1 - ku}, \quad \tilde{\gamma}_k = \frac{cku}{1 - cku},$$

where c denotes a small integer constant whose exact value is unimportant.

2. The scaling and squaring method. The scaling and squaring method exploits the relation $e^A = (e^{A/\sigma})^\sigma$, for $A \in \mathbb{C}^{n \times n}$ and $\sigma \in \mathbb{C}$, together with the fact that e^A can be well approximated by a Padé approximant near the origin, that is, for small $\|A\|$. The idea is to choose σ an integral power of 2, $\sigma = 2^s$ say, so that A/σ has norm of order 1, approximate $e^{A/2^s} \approx r_{km}(A/2^s)$, where r_{km} is a $[k/m]$ Padé approximant to the exponential, and then take $e^A \approx r_{km}(A/2^s)^{2^s}$, where the approximation is formed by s repeated squarings. Recall that $r_{km}(x) = p_{km}(x)/q_{km}(x)$ is defined by the properties that p and q are polynomials of degrees at most k and m , respectively, and that $e^x - r_{km}(x) = O(x^{k+m+1})$. The scaling and squaring method goes back at least to Lawson [15].

The mathematical elegance of the scaling and squaring method is enhanced by the fact that the $[k/m]$ Padé approximants $r_{km}(x) = p_{km}(x)/q_{km}(x)$ to the exponential function are known explicitly for all k and m :

$$(2.1) \quad p_{km}(x) = \sum_{j=0}^k \frac{(k+m-j)!k!}{(k+m)!(k-j)!} \frac{x^j}{j!}, \quad q_{km}(x) = \sum_{j=0}^m \frac{(k+m-j)!m!}{(k+m)!(m-j)!} \frac{(-x)^j}{j!}.$$

Note that $p_{km}(x) = q_{mk}(-x)$, which reflects the property $1/e^x = e^{-x}$ of the exponential function. Later we will exploit the fact that $p_{mm}(x)$ and $q_{mm}(x)$ approximate $e^{x/2}$ and $e^{-x/2}$, respectively, though they do so *much* less accurately than $r_{mm} = p_{mm}/q_{mm}$ approximates e^x . That r_{km} satisfies the definition of Padé approximant is demonstrated by the error expression [6, Thm. 5.5.1]

$$(2.2) \quad e^x - r_{km}(x) = (-1)^m \frac{k!m!}{(k+m)!(k+m+1)!} x^{k+m+1} + O(x^{k+m+2}).$$

Diagonal approximants ($k = m$) are preferred, since r_{km} with $k \neq m$ is less accurate than r_{jj} , where $j = \max(k, m)$, but r_{jj} can be evaluated at a matrix argument at the same cost. Moreover, the diagonal approximants have the property that if the eigenvalues of A lie in the open left half-plane then the eigenvalues of $r_{mm}(A)$

have modulus less than 1 (that is, the spectral radius $\rho(r_{mm}(A)) < 1$), which is an important property in applications to differential equations [23, Chap. 8]. We will write the diagonal approximants as $r_m(x) = p_m(x)/q_m(x)$.

Our aim is to choose s , in the initial scaling $A \leftarrow A/2^s$, so that the exponential is computed with backward error bounded by the unit roundoff and with minimal cost. In bounding the backward error we assume exact arithmetic and examine solely the effects of the approximation errors in the Padé approximant.

We begin by considering errors. The choice of s will be based on $\|A\|$, where the norm can be any subordinate matrix norm. Our aim is therefore to bound the backward error in terms of $\|2^{-s}A\|$ and then to determine, for each degree m , the maximum $\|2^{-s}A\|$ for which r_m can be guaranteed to deliver the desired backward error. Moler and Van Loan [16] give a very elegant backward error analysis, from which they obtain a criterion for choosing m ; see also Golub and Van Loan [7, sec. 11.3]. Their analysis has two weaknesses. First, it makes an initial assumption that $\|A\| \leq 1/2$, whereas, as we will see, there are good reasons for allowing $\|A\|$ to be much larger. Second, it is designed to provide an explicit and easily computable error bound, and the resulting bound is far from being sharp. We now adapt the ideas of Moler and Van Loan in order to obtain a bound that makes no a priori assumption on $\|A\|$ and is as sharp as possible. The tradeoff is that the bound is hard to evaluate, but this is a minor inconvenience because the evaluation need only be done during the design of the algorithm.

Let

$$(2.3) \quad e^{-A}r_m(A) = I + G = e^H,$$

where we assume that $\|G\| < 1$, so that $H = \log(I + G)$ is guaranteed to exist. (Here, \log denotes the principal logarithm.) From $\log(I + G) = \sum_{j=1}^{\infty} (-1)^{j+1} G^j/j$, we have

$$\|H\| = \|\log(I + G)\| \leq \sum_{j=1}^{\infty} \|G\|^j/j = -\log(1 - \|G\|).$$

Now G is clearly a function of A (in the sense of matrix functions [9], [12, Chap. 6]), hence so is H , and therefore H commutes with A . It follows that

$$r_m(A) = e^A e^H = e^{A+H}.$$

Now we replace A by $A/2^s$, where s is a nonnegative integer, and raise both sides of this equation to the power 2^s to obtain

$$r_m(A/2^s)^{2^s} = e^{A+E},$$

where $E = 2^s H$ satisfies

$$\|E\| \leq -2^s \log(1 - \|G\|)$$

and G satisfies (2.3) with A replaced by $2^{-s}A$. We summarize our findings in the following theorem.

THEOREM 2.1. *Let the diagonal Padé approximant r_m satisfy*

$$(2.4) \quad e^{-2^{-s}A} r_m(2^{-s}A) = I + G,$$

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

$$r_m(2^{-s}A)^{2^s} = e^{A+E},$$

where E commutes with A and

$$(2.5) \quad \frac{\|E\|}{\|A\|} \leq \frac{-\log(1 - \|G\|)}{\|2^{-s}A\|}. \quad \square$$

Theorem 2.1 is a backward error result: it interprets the truncation errors in the Padé approximant as equivalent to a perturbation in the original matrix A . (The result holds, in fact, for any rational approximation r_m , as we have not yet used specific properties of a Padé approximant.) The advantage of the backward error viewpoint is that it automatically takes into account the effect of the squaring phase on the error in the Padé approximant and, compared with a forward error bound, avoids the need to consider the conditioning of the problem.

Our task now is to bound the norm of G in (2.4) in terms of $\|2^{-s}A\|$. Define the function

$$\rho(x) = e^{-x}r_m(x) - 1.$$

In view of the Padé approximation property (2.2), ρ has a power series expansion

$$(2.6) \quad \rho(x) = \sum_{i=2m+1}^{\infty} c_i x^i,$$

and this series will converge absolutely for $|x| < \min\{|t| : q_m(t) = 0\} =: \nu_m$. Hence

$$(2.7) \quad \|G\| = \|\rho(2^{-s}A)\| \leq \sum_{i=2m+1}^{\infty} |c_i| \theta^i =: f(\theta),$$

where $\theta := \|2^{-s}A\| < \nu_m$. It is clear that if A is a general matrix and only $\|A\|$ is known then (2.7) provides the smallest possible bound on $\|G\|$. The corresponding bound of Moler and Van Loan [16, Appx. 1, Lem. 4] is easily seen to be less sharp, and a refined analysis of Dieci and Papini [3, sec. 2], which bounds a different error, is also weaker when adapted to bound $\|G\|$.

Combining (2.7) with (2.5) we have

$$(2.8) \quad \frac{\|E\|}{\|A\|} \leq \frac{-\log(1 - f(\theta))}{\theta}.$$

Evaluation of $f(\theta)$ in (2.7) would be easy if the coefficients c_i were one-signed, for then we would have $f(\theta) = |\rho(\theta)|$. Experimentally, the c_i are one-signed for some, but not all, m . Using MATLAB's Symbolic Math Toolbox, we have evaluated $f(\theta)$, and hence the bound (2.8), in 250 decimal digit arithmetic, summing the first 150 terms of the series, where the c_i in (2.6) are obtained symbolically. For $m = 1: 21$ we have used a zero-finder to determine the largest value of θ , denoted by θ_m , such that the backward error bound (2.8) does not exceed $u = 2^{-53} \approx 1.1 \times 10^{-16}$, the unit roundoff in IEEE double precision arithmetic. The results are shown to two significant figures in Table 2.1.

The second row of the table shows the values of ν_m , and we see that $\theta_m < \nu_m$ in each case, confirming that the bound (2.7) is valid. The inequalities $\theta_m < \nu_m$ also

TABLE 2.1

Maximal values θ_m of $\|2^{-s}A\|$ such that the backward error bound (2.8) does not exceed $u = 2^{-53}$, values of $\nu_m = \min\{|x| : q_m(x) = 0\}$, and upper bound ξ_m for $\|q_m(A)^{-1}\|$.

m	1	2	3	4	5	6	7	8	9	10	
θ_m	3.7e-8	5.3e-4	1.5e-2	8.5e-2	2.5e-1	5.4e-1	9.5e-1	1.5e0	2.1e0	2.8e0	
ν_m	2.0e0	3.5e0	4.6e0	6.0e0	7.3e0	8.7e0	9.9e0	1.1e1	1.3e1	1.4e1	
ξ_m	1.0e0	1.0e0	1.0e0	1.0e0	1.1e0	1.3e0	1.6e0	2.1e0	3.0e0	4.3e0	
m	11	12	13	14	15	16	17	18	19	20	21
θ_m	3.6e0	4.5e0	5.4e0	6.3e0	7.3e0	8.4e0	9.4e0	1.1e1	1.2e1	1.3e1	1.4e1
ν_m	1.5e1	1.7e1	1.8e1	1.9e1	2.1e1	2.2e1	2.3e1	2.5e1	2.6e1	2.7e1	2.8e1
ξ_m	6.6e0	1.0e1	1.7e1	3.0e1	5.3e1	9.8e1	1.9e2	3.8e2	8.3e2	2.0e3	6.2e3

confirm the important fact that $q_m(A)$ is nonsingular for $\|A\| \leq \theta_m$ (which is in any case implicitly enforced by our analysis).

Next we need to determine the cost of evaluating $r_m(A)$. Because of the relation $q_m(x) = p_m(-x)$ between the numerator and denominator polynomials, an efficient scheme can be based on explicitly computing the even powers of A , forming p_m and q_m , and then solving the matrix equation $q_m r_m = p_m$ [22]. If $p_m(x) = \sum_{i=0}^m b_i x^i$, we have, for the even-degree case,

$$(2.9) \quad p_{2m}(A) = b_{2m}A^{2m} + \dots + b_2A^2 + b_0I + A(b_{2m-1}A^{2m-2} + \dots + b_3A^2 + b_1I) =: U + V,$$

which can be evaluated with $m + 1$ matrix multiplications by forming A^2, A^4, \dots, A^{2m} . Then

$$q_{2m}(A) = U - V$$

is available at no extra cost. For odd degrees,

$$(2.10) \quad p_{2m+1}(A) = A(b_{2m+1}A^{2m} + \dots + b_3A^2 + b_1I) + b_{2m}A^{2m} + \dots + b_2A^2 + b_0I =: U + V,$$

and so p_{2m+1} and $q_{2m+1} = -U + V$ can be evaluated at exactly the same cost as p_{2m} and q_{2m} . However, for $m \geq 12$ this scheme can be improved upon. For example, we can write

$$(2.11) \quad p_{12}(A) = A^6(b_{12}A^6 + b_{10}A^4 + b_8A^2 + b_6I) + b_4A^4 + b_2A^2 + b_0I + A[b_{11}A^4 + b_9A^2 + b_7I] + b_5A^4 + b_3A^2 + b_1I =: U + V,$$

and $q_{12}(A) = U - V$. Thus p_{12} and q_{12} can be evaluated in just six matrix multiplications (for A^2, A^4, A^6 , and three additional multiplications). For $m = 13$ an analogous formula holds with the outer multiplication by A transferred to the U term. Similar formulae hold for $m \geq 14$. Table 2.2 summarizes the number of matrix multiplications required to evaluate p_m and q_m , which we denote by π_m , for $m = 1 : 21$.

The information in Tables 2.1 and 2.2 enables us to determine the optimal algorithm when $\|A\| \geq \theta_{21}$. From Table 2.2, we see that the choice is between $m = 1, 2, 3, 5, 7, 9, 13, 17$, and 21 . (There is no reason to use $m = 6$, for example, since the cost of evaluating the more accurate q_7 is the same as the cost of evaluating q_6 .) Increasing

TABLE 2.2

Number of matrix multiplications, π_m , required to evaluate $p_m(A)$ and $q_m(A)$, and the measure of overall cost C_m in (2.12).

m	1	2	3	4	5	6	7	8	9	10	
π_m	0	1	2	3	3	4	4	5	5	6	
C_m	25	12	8.1	6.6	5.0	4.9	4.1	4.4	3.9	4.5	
m	11	12	13	14	15	16	17	18	19	20	21
π_m	6	6	6	7	7	7	7	8	8	8	8
C_m	4.2	3.8	3.6	4.3	4.1	3.9	3.8	4.6	4.5	4.3	4.2

from one of these values of m to the next requires an extra matrix multiplication to evaluate r_m , but this is offset by the larger allowed $\theta_m = \|2^{-s}A\|$ if θ_m jumps by more than a factor 2, since decreasing s by 1 saves one multiplication in the final squaring stage. Table 2.1 therefore shows that $m = 13$ is the best choice. Another way to arrive at this conclusion is to observe that the cost of the algorithm in matrix multiplications is, since $s = \lceil \log_2 \|A\|/\theta_m \rceil$ if $\|A\| \geq \theta_m$ and $s = 0$ otherwise,

$$\pi_m + s = \pi_m + \max(\lceil \log_2 \|A\| - \log_2 \theta_m \rceil, 0).$$

(We ignore the required matrix equation solution, which is common to all m .) We wish to determine which m minimizes this quantity. For $\|A\| \geq \theta_m$ we can remove the max and ignore the $\|A\|$ term, which is essentially a constant shift, and so we minimize

$$(2.12) \quad C_m = \pi_m - \log_2 \theta_m.$$

The C_m values are shown in the second line of Table 2.2. Again, $m = 13$ is clearly the best choice. We repeated the computations with $u = 2^{-24} \approx 6.0 \times 10^{-8}$, which is the unit roundoff in IEEE single precision arithmetic, and $u = 2^{-105} \approx 2.5 \times 10^{-32}$, which corresponds to quadruple precision arithmetic; the optimal m are now $m = 7$ and $m = 17$, respectively.

Now we consider the effects of rounding errors on the evaluation of $r_m(A)$. We immediately rule out $m = 1$ and $m = 2$ because r_1 and r_2 can suffer from loss of significance in floating point arithmetic. For example, r_1 requires $\|A\|$ to be of order 10^{-8} after scaling, and then the expression $r_1(A) = (I + A/2)(I - A/2)^{-1}$ loses about half the significant digits in A in double precision arithmetic; yet if the original A has norm of order at least 1 then all the significant digits of some of the elements of A should contribute to the result.

The effect of rounding errors on the evaluation of the numerator and denominator of $r_m(A)$ is described by the following result, which can be proved using techniques from [11].

THEOREM 2.2. *Let $g_m(x) = \sum_{k=0}^m b_k x^k$. The computed polynomial \hat{g}_m obtained by evaluating g_m at $X \in \mathbb{C}^{n \times n}$ using explicit formation of matrix powers as in the methods above satisfies*

$$|g_m - \hat{g}_m| \leq \tilde{\gamma}_{mn} \tilde{g}_m(|X|),$$

where $\tilde{g}_m(X) = \sum_{i=0}^m |b_i| X^i$. Hence $\|g_m - \hat{g}_m\|_1 \leq \tilde{\gamma}_{mn} \tilde{g}_m(\|X\|_1)$. \square

Applying the theorem to $p_m(A)$, where $\|A\|_1 \leq \theta_m$, and noting that p_m has all

positive coefficients, we deduce that

$$\begin{aligned} \|p_m(A) - \widehat{p}_m(A)\|_1 &\leq \widetilde{\gamma}_{mn} p_m(\|A\|_1) \\ &\approx \widetilde{\gamma}_{mn} e^{\|A\|_1/2} \\ &\leq \widetilde{\gamma}_{mn} \|e^{A/2}\|_1 e^{\|A\|_1} \\ &\approx \widetilde{\gamma}_{mn} \|p_m(A)\|_1 e^{\|A\|_1} \leq \widetilde{\gamma}_{mn} \|p_m(A)\|_1 e^{\theta_m}. \end{aligned}$$

Hence the relative error is bounded approximately by $\widetilde{\gamma}_{mn} e^{\theta_m}$, which is a very satisfactory bound, given the values of θ_m in Table 2.1. Replacing A by $-A$ in the latter bound we obtain

$$\|q_m(A) - \widehat{q}_m(A)\|_1 \lesssim \widetilde{\gamma}_{mn} \|q_m(A)\|_1 e^{\theta_m}.$$

In summary, the errors in the evaluation of p_m and q_m are nicely bounded. This analysis improves that of Ward [24, equation (3.5)], who assumes $\|A\| \leq 1$ and obtains absolute error bounds.

To obtain r_m we solve a multiple right-hand side linear system with $q_m(A)$ as coefficient matrix, so to be sure that this system is solved accurately we need to check that $q_m(A)$ is well conditioned. It is possible to obtain a priori bounds for $\|q_m(A)^{-1}\|$ under assumptions such as $\|A\| \leq 1/2$ [16, Appx. 1, Lem. 2], $\|A\| \leq 1$ [24, Thm. 1], or $q_m(-\|A\|) < 2$ [3, Lem. 2.1], but these assumptions are not satisfied for all the m and $\|A\|$ of interest to us. Therefore we take a similar approach to the way we derived the constants θ_m . With $\|A\| \leq \theta_m$ and by writing

$$q_m(A) = e^{-A/2}(I + e^{A/2}q_m(A) - I) \equiv e^{-A/2}(I + F),$$

we have, if $\|F\| < 1$,

$$\|q_m(A)^{-1}\| \leq \|e^{A/2}\| \|(I + F)^{-1}\| \leq \frac{e^{\theta_m/2}}{1 - \|F\|}.$$

We can expand $e^{x/2}q_m(x) - 1 = \sum_{i=2}^{\infty} d_i x^i$, from which $\|F\| \leq \sum_{i=2}^{\infty} |d_i| \theta_m^i$ follows. Our overall bound is

$$\|q_m(A)^{-1}\| \leq \frac{e^{\theta_m/2}}{1 - \sum_{i=2}^{\infty} |d_i| \theta_m^i}.$$

By determining the d_i symbolically and summing the first 150 terms of the sum in 250 decimal digit arithmetic, we obtained the bounds in the last row of Table 2.1, which confirm that q_m is very well conditioned for m up to about 13 when $\|A\| \leq \theta_m$.

Our algorithm is as follows. It first checks whether $\|A\| \leq \theta_m$ for $m \in \{3, 5, 7, 9, 13\}$ and, if so, evaluates r_m for the smallest such m . Otherwise it uses the scaling and squaring method with $m = 13$.

ALGORITHM 2.3. *This algorithm evaluates the matrix exponential of $A \in \mathbb{C}^{n \times n}$ using the scaling and squaring method. It uses the constants θ_m given in Table 2.3.*

- 1 % Coefficients of degree 13 Padé approximant.
- 2 $b(0:13) = [64764752532480000, 32382376266240000, 7771770303897600,$
- 3 $1187353796428800, 129060195264000, 10559470521600,$
- 4 $670442572800, 33522128640, 1323241920,$
- 5 $40840800, 960960, 16380, 182, 1]$

TABLE 2.3
 Constants θ_m needed in Algorithm 2.3.

m	θ_m
3	1.495585217958292e-2
5	2.539398330063230e-1
7	9.504178996162932e-1
9	2.097847961257068e0
13	5.371920351148152e0

```

6  % Preprocessing to reduce the norm.
7  A ← A − μI, where μ = trace(A)/n.
8  A ← D−1AD, where D is a balancing transformation (or set D = I if
   balancing does not reduce the 1-norm of A).

9  for m = [3 5 7 9 13]
10     if ||A||1 ≤ θm
11         X = rm(A) % rm(A) = [m/m] Padé approximant to A.
12         X = eμDXD−1 % Undo preprocessing.
13     end
14 end
15 A ← A/2s with s a minimal integer such that ||A/2s||1 ≤ θ13
   (i.e., s = ⌈log2(||A||1/θ13)⌉).

16 % Form [13/13] Padé approximant to eA.
17 A2 = A2, A4 = A22, A6 = A2A4
18 U = A[A6(b13A6 + b11A4 + b9A2) + b7A6 + b5A4 + b3A2 + b1I]
19 V = A6(b12A6 + b10A4 + b8A2) + b6A6 + b4A4 + b2A2 + b0I
20 Solve (−U + V)r13 = U + V for r13.

21 X = r132s by repeated squaring.
22 X = eμDXD−1 % Undo preprocessing.

```

The cost of Algorithm 2.3 is $\pi_m + \lceil \log_2(\|A\|_1/\theta_m) \rceil$ matrix multiplications, where m is the degree of Padé approximant used, and π_m is tabulated in Table 2.2, plus the solution of one matrix equation.

It is readily checked that the sequences $\theta_{13}^{2k}b_{2k}$ and $\theta_{13}^{2k+1}b_{2k+1}$ are approximately monotonically decreasing with k , and hence the ordering given in Algorithm 2.3 for evaluating U and V takes the terms in approximately increasing order of norm. This ordering is certainly preferable when A has nonnegative elements, and since there cannot be much cancellation in the sums it cannot be a bad ordering [11, Chap. 4].

The Padé approximant r_m at line 11 is intended to be evaluated using (2.10) for $m \leq 9$, or as in lines 17–19 for $m = 13$.

The preprocessing in Algorithm 2.3 is precisely that suggested by Ward [24] and attempts to reduce the norm by a shift and a similarity transformation.

The use of the [13/13] Padé approximation in Algorithm 2.3 gives optimal efficiency. However, Table 2.1 reports a bound of 3.0 for $\|q_9(A)^{-1}\|$, which is somewhat smaller than the bound of 17 for $\|q_{13}(A)^{-1}\|$, and C_9 is only slightly larger than C_{13} ; therefore the best compromise between numerical stability and efficiency could conceivably be obtained by limiting to maximum degree $m = 9$. We will compare these two degrees experimentally in the next section.

3. Comparison with existing algorithms. We now compare Algorithm 2.3 with existing implementations of the scaling and squaring method that also employ

Padé approximations to e^x .

The function `expm` in MATLAB 7 uses $m = 6$ with $\|2^{-s}A\|_\infty \leq 0.5$ as the scaling criterion and does not employ preprocessing. (`expm` is a built-in function, but `expdemo1` is an M-file implementation of the same algorithm, and in all our tests `expm` and `expdemo1` produced exactly the same results.) Sidje [19] uses the same parameters in his function `padm`. Surprisingly, neither `expm` nor `padm` evaluates r_6 optimally: whereas (2.9) requires just 4 multiplications, `expm` uses 5, because it evaluates all the powers A^2, A^3, \dots, A^6 , while `padm` expends 7 multiplications in using Horner's method with a variant of (2.9). We note that in `padm`, p_m and q_m are evaluated in increasing order of norms of the terms, as in Algorithm 2.3, whereas `expm` uses the reverse ordering.

Ward [24] uses $m = 8$ with $\|2^{-s}A\|_1 \leq 1$ and carries out the same preprocessing as Algorithm 2.3.

In the following discussion we will assume that all the algorithms use the same norm and ignore the preprocessing.

Since Ward's value of θ is twice that used in `expm` and `padm`, and the [8/8] Padé approximant can be evaluated with just one more matrix multiplication than the [6/6] one, Ward's algorithm would have exactly the same cost as `expm` and `padm` for $\|A\| \geq 0.5$, were the latter algorithms to evaluate r_6 efficiently.

It is clear from our analysis that the three algorithms under discussion are not of optimal efficiency. If the [6/6] or [8/8] Padé approximants are to be used, then one can take larger values of θ , as shown by Table 2.1. Moreover, as we have argued, there is no reason to use the degree 6 or 8 approximants because the degree 7 and 9 approximants have the same cost, respectively.

By considering Tables 2.1 and 2.2 it is easy to see the following:

- When $\|A\|_1 > 1$, Algorithm 2.3 requires one or two fewer matrix multiplications than Ward's implementation, two or three fewer than `expm`, and four or five fewer than `padm`. For example, when $\|A\|_1 \in (2, 2.1)$, the number of matrix multiplications reduces from 8 for `expm` and 7 for Ward's implementation to 5 for Algorithm 2.3 (which takes $m = 9$)—a saving of 37% and 29%, respectively.
- When $\|A\|_1 \leq 1$, Algorithm 2.3 requires no more matrix multiplications than `expm`, `padm`, and Ward's algorithm, and up to 3, 5, and 3 fewer, respectively.

Our analysis shows that all these algorithms have a backward error no larger than u , ignoring roundoff. However, it is well known that rounding errors can significantly affect the scaling and squaring method, because the squaring phase can suffer from severe numerical cancellation. The fundamental problem can be seen in the result [11, sec. 3.5]

$$\|A^2 - fl(A^2)\| \leq \gamma_n \|A\|^2,$$

which shows that the errors in the computed squared matrix are small compared with the square of the norm of the original matrix but not necessarily small compared with the matrix being computed. By using standard error analysis techniques it is possible to derive a forward error bound for the scaling and squaring method, as has been done by Ward [24]. However, with our current knowledge of the e^A problem it is not easy to determine whether a large value for the bound signals potential instability of the method or an ill-conditioned problem.

Since the matrix squarings in the scaling and squaring method are potentially dangerous it seems desirable to minimize the number of them. Algorithm 2.3 uses

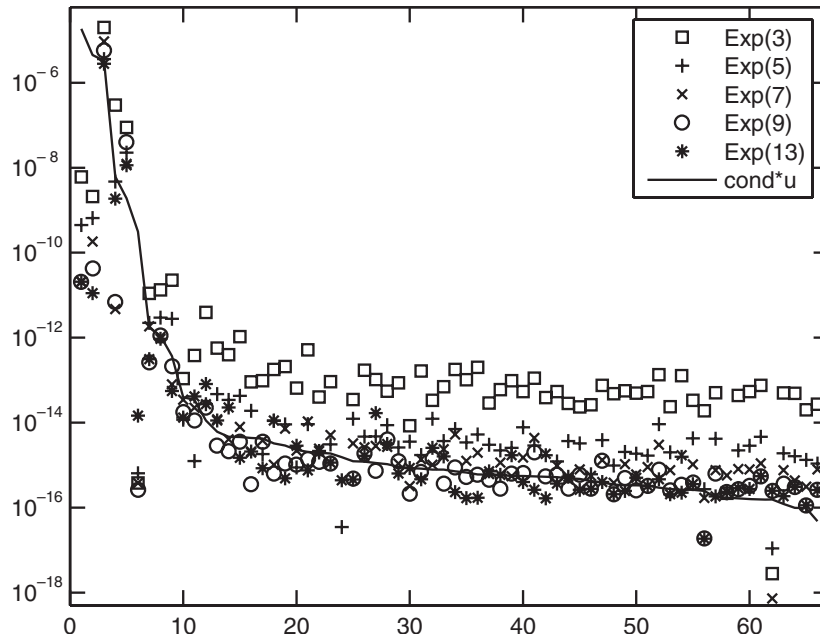


FIG. 3.1. Normwise relative errors for Algorithm 2.3 (Exp(13)) and variants with m_{\max} restricted to 3, 5, 7, and 9.

one to three fewer squarings than the algorithms with which we have compared it, and hence it has a potential advantage in accuracy.

We now present some numerical experiments, carried out in MATLAB 7.0 (R14), that provide some insight into the accuracy of the scaling and squaring method and of Algorithm 2.3. We took 66 8×8 test matrices: 53 obtained from the function `matrix` in the Matrix Computation Toolbox [10] (which include test matrices from MATLAB itself), together with 13 further test matrices of dimension 2–10 from [1, Ex. 3], [3, Ex. 3.10], [14, Ex. 2 and p. 655], [18, p. 370], and [24, Test Cases 1–4]. We evaluated the relative error in the 1-norm of the computed matrices from `expm`, from Algorithm 2.3, and from a modified version of Algorithm 2.3 in which the maximal degree of the Padé approximant is a parameter, m_{\max} . The latter algorithm, denoted by `Exp(m_{\max})`, allows us to study the dependence of the error on m_{\max} . We did not use any preprocessing in this experiment, although we found that turning on preprocessing in Algorithm 2.3 makes essentially no difference to the results. The “exact” e^A is obtained at 100-digit precision using MATLAB’s Symbolic Math Toolbox.

Figure 3.1 compares the errors for the different maximal Padé degrees. It shows a clear trend that the smaller the m_{\max} the larger the error. The solid line is the unit roundoff multiplied by the (relative) condition number

$$\text{cond}(A) = \lim_{\epsilon \rightarrow 0} \max_{\|E\|_2 \leq \epsilon \|A\|_2} \frac{\|e^{A+E} - e^A\|_2}{\epsilon \|e^A\|_2},$$

which we estimate using the finite-difference power method of Kenney and Laub [13], [9]. For a method to perform in a backward stable, and hence forward stable, manner, its error should lie not far above this line on the graph. In all our figures the results are sorted by decreasing condition number $\text{cond}(A)$. We see that Algorithm 2.3

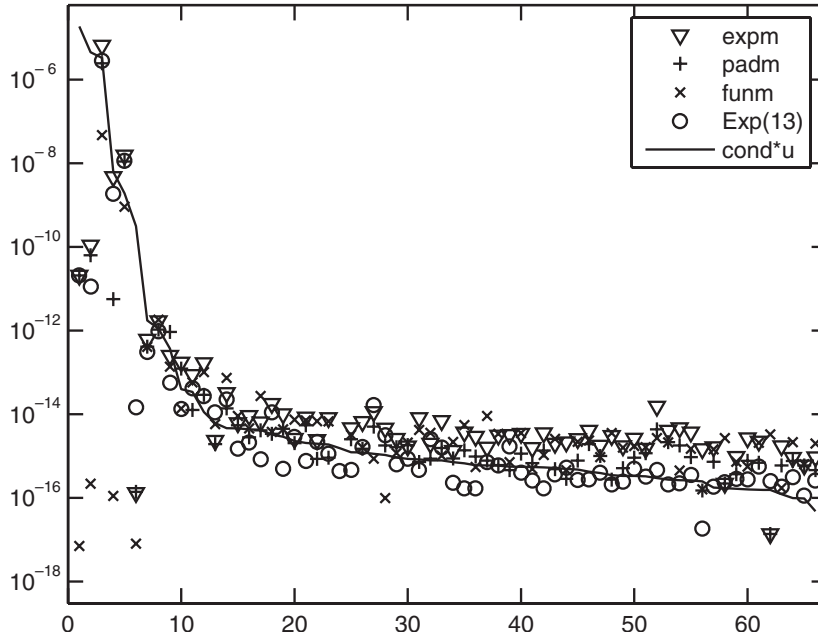


FIG. 3.2. Normwise relative errors for `expm`, `padm` (Sidje), `funm`, and Algorithm 2.3 (Exp(13)).

($m_{\max} = 13$) performs in a numerically stable way on this experiment, even though two of the test matrices were chosen to cause the scaling and squaring method to “overscale”—a phenomenon investigated in [3] and [14]. Some instability is apparent for the smaller m_{\max} . The numerical results therefore concur with the theory in suggesting that the fewer the number of squarings, the smaller the error.

Figure 3.2 compares Algorithm 2.3 with `expm`, Sidje’s function `padm`, and MATLAB 7’s `funm`, which implements the Schur–Parlett method of Davies and Higham [1], which is designed for general f . The figure shows that `expm` exhibits minor instability on many of the test matrices.

Finally, Figure 3.3 plots a performance profile [4] for the experiment. Each of the methods is represented by a curve on the plot. For a given α on the x -axis, the y -coordinate of the corresponding point on the curve is the probability that the method in question has an error within a factor α of the smallest error over all the methods, where probabilities are defined over the set of test problems. For $\alpha = 1$, the `Exp(13)` curve is the highest: it intersects the y -axis at $p = 0.52$, which means that this method has the smallest error in 52% of the examples—more often than any other method. For $\alpha \gtrsim 1.6$, `Exp(9)` is more likely than `Exp(13)` to be within a factor α of the smallest error. Since the curve for `expm` lies below all the other curves, `expm` is the least accurate method on this set of test matrices, as measured by the performance profile. Recall that the functions `expm` and `padm` both use $m = 6$ and differ only in how they evaluate r_6 , as described at the start of this section.

In interpreting the results it is worth noting that the actual errors the methods produce are sensitive to the details of the arithmetic. The version of Figure 3.3 produced by a prerelease version of MATLAB 7.0 was different, though qualitatively similar. (For example, the `Exp(13)` and `Exp(9)` curves touched at $\alpha = 3$, though they did not cross.)

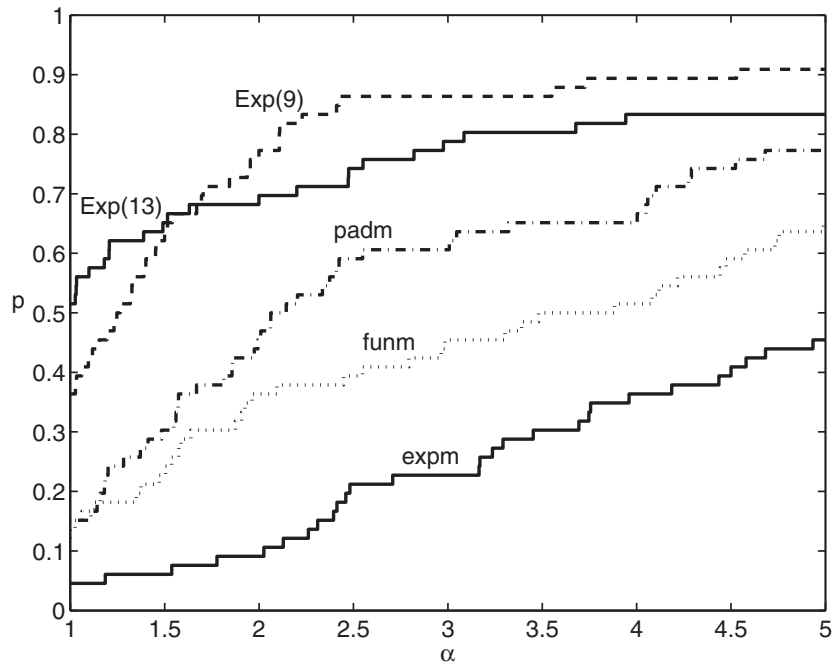


FIG. 3.3. Performance profile: α is plotted against the probability p that a method has error within a factor α of the smallest error over all methods.

This experiment shows that in terms of accuracy in floating point arithmetic there is no clear reason to favor **Exp(13)** over **Exp(9)** or vice versa. Our choice of **Exp(13)** in Algorithm 2.3 on the grounds of its lower cost is therefore justified.

4. Indirect Padé approximation. Najfeld and Havel [18, sec. 2] suggest an interesting variation of the standard scaling and squaring method that they claim is more efficient. Instead of approximating the exponential directly, they use a Padé approximation to the even function

$$\begin{aligned}
 \tau(x) &= x \coth(x) = x(e^{2x} + 1)(e^{2x} - 1)^{-1} \\
 (4.1) \quad &= 1 + \frac{x^2}{3 + \frac{x^2}{5 + \frac{x^2}{7 + \dots}}}
 \end{aligned}$$

in terms of which the exponential can be written

$$(4.2) \quad e^{2x} = \frac{\tau(x) + x}{\tau(x) - x}.$$

The Padé approximants to τ can be obtained by truncating the continued fraction expansion (4.1). For example, using \tilde{r}_{2m} to denote the diagonal $[2m/2m]$ Padé approximant to τ ,

$$\tilde{r}_8(x) = \frac{\frac{1}{765765}x^8 + \frac{4}{9945}x^6 + \frac{7}{255}x^4 + \frac{8}{17}x^2 + 1}{\frac{1}{34459425}x^8 + \frac{2}{69615}x^6 + \frac{1}{255}x^4 + \frac{7}{51}x^2 + 1}.$$

The numerators and denominators of \tilde{r}_{2m} comprise only even powers of x , and so they can be evaluated at a matrix argument A in m matrix multiplications by explicitly forming the required even powers.

The error in r_{2m} has the form

$$(4.3) \quad \tau(x) - \tilde{r}_{2m}(x) = \sum_{k=1}^{\infty} d_k x^{4m+2k} = \sum_{k=1}^{\infty} d_k (x^2)^{2m+k}.$$

(The error is one order in x higher than the definition of Padé approximant requires, due to the fact that τ is even.) Hence the error in the matrix approximation satisfies

$$(4.4) \quad \|\tau(A) - \tilde{r}_{2m}(A)\| \leq \sum_{k=1}^{\infty} d_k \|A^2\|^{2m+k} =: \omega_{2m}(\|A^2\|).$$

Let θ_{2m} be the largest θ such that $\omega_{2m}(\theta) \leq u$. The algorithm of Najfeld and Havel scales $\tilde{A} \leftarrow A/2^{s+1}$ with $s \geq 0$ chosen so that $\|\tilde{A}^2\| = \|A^2\|/2^{2s+2} \leq \theta_{2m}$. Padé approximation is applied to the scaled matrix, \tilde{A} . The final stage consists of s squarings, just as in the standard scaling and squaring method. Note that there are s squarings rather than $s + 1$, because the underlying approximation (4.2) is to e^{2x} and not e^x . Computation of the θ_{2m} and analysis of computational cost in [18] leads Najfeld and Havel to conclude that the choice $m = 8$ of Padé approximant degree leads to the most efficient algorithm.

Detailed study of this algorithm shows that it is competitive in cost with Algorithm 2.3. The following result reveals a close connection with Algorithm 2.3.

THEOREM 4.1. *The $[2m/2m]$ Padé approximant $\tilde{r}_{2m}(x)$ to $x \coth(x)$ is related to the $[2m + 1/2m + 1]$ Padé approximant $r_{2m+1}(x)$ to e^x by*

$$r_{2m+1}(x) = \frac{\tilde{r}_{2m}(x/2) + x/2}{\tilde{r}_{2m}(x/2) - x/2}.$$

Proof. By (4.3),

$$e_{2m}(x) := \tau(x) - \tilde{r}_{2m}(x) = O(x^{4m+2}).$$

Then

$$\begin{aligned} g(x) &:= \frac{\tilde{r}_{2m}(x) + x}{\tilde{r}_{2m}(x) - x} = \frac{\tau(x) + x - e_{2m}(x)}{\tau(x) - x - e_{2m}(x)} \\ &= \frac{\tau(x) + x}{\tau(x) - x} \left[\frac{1 - e_{2m}(x)/(\tau(x) + x)}{1 - e_{2m}(x)/(\tau(x) - x)} \right] \\ &= e^{2x} \left[1 - \frac{e_{2m}(x)}{\tau(x) + x} + \frac{e_{2m}(x)}{\tau(x) - x} + O(e_{2m}(x)^2) \right] \\ &= e^{2x} \left[1 + \frac{2xe_{2m}(x)}{(\tau(x) + x)(\tau(x) - x)} + O(e_{2m}(x)^2) \right] \\ &= e^{2x}(1 + xO(e_{2m}(x))) = e^{2x} + O(x^{4m+3}). \end{aligned}$$

Now $g(x)$ is a rational function with numerator and denominator both of degree at most $2m + 1$, and $g(x/2) = e^x + O(x^{4m+3})$. By the uniqueness of Padé approximants to the exponential, $g(x/2) \equiv r_{2m+1}(x)$. \square

Hence the algorithm of Najfeld and Havel, which takes $m = 8$, is implicitly using the same Padé approximant to e^x as Algorithm 2.3 when the latter takes $m = 9$. The difference is essentially in how A is scaled prior to forming the approximant and in the precise formulae from which the approximant is computed. While the derivation of Najfeld and Havel's algorithm ensures that the error $\|\tau(A) - \tilde{r}_{2m}(A)\|$ is sufficiently small for the scaled A , what this implies about the error $e^{2A} - (\tilde{r}_{2m}(A) + A)(\tilde{r}_{2m}(A) - A)^{-1}$ is unclear, particularly since the matrix $\tilde{r}_{2m}(A) - A$ that is inverted can be arbitrarily ill conditioned. Moreover, it is unclear how to derive an analogue of Theorem 2.1 that expresses the truncation errors in the Padé approximant to τ as backward errors in the original data.

We conclude that the algorithm suggested by Najfeld and Havel is essentially a variation of the standard scaling and squaring method with direct Padé approximation but with weaker guarantees concerning its behavior both in exact arithmetic (since a backward error result is lacking) and in floating point arithmetic (since a possibly ill-conditioned matrix must be inverted). Without stronger supporting analysis the method cannot therefore be recommended.

5. Conclusions. The scaling and squaring method has long been the most popular method for computing the matrix exponential. By analyzing it afresh we have found that existing implementations of Sidje [19] and Ward [24], and in the function `expm` in MATLAB, are not optimal. While they do guarantee a backward error of order the unit roundoff in the absence of roundoff (that is, solely considering truncation errors in the Padé approximation), they use more matrix multiplications than necessary. By developing an essentially optimal backward error bound for the scaling and squaring method in exact arithmetic that depends on A only through $\|A\|$, we have identified the most efficient choice of degree m of Padé approximation and initial scaling for IEEE double precision arithmetic: $m = 13$, as opposed to $m = 6$ for `expm` and Sidje's algorithm and $m = 8$ for Ward's algorithm, with scaling to ensure $\|A\| \leq 5.4$. A welcome side effect has been to reduce the amount of scaling, and hence the number of squarings in the final stage. This reduction, together with a careful evaluation of the Padé approximation, makes the new algorithm typically more accurate than the old ones (see Figures 3.2 and 3.3).

With the aid of some new error analysis we have shown that all but one part of Algorithm 2.3 is numerically stable. The effect of rounding errors on the final squaring phase remains an open question, but in our experiments the overall algorithm has performed in a numerically stable way throughout.

Acknowledgments. I am grateful to Philip Davies for insightful comments on section 4 and Roy Mathias for suggesting evaluation schemes of the form (2.11).

REFERENCES

- [1] P. I. DAVIES AND N. J. HIGHAM, *A Schur–Parlett algorithm for computing matrix functions*, SIAM J. Matrix Anal. Appl., 25 (2003), pp. 464–485.
- [2] I. S. DHILLON AND B. N. PARLETT, *Orthogonal eigenvectors and relative gaps*, SIAM J. Matrix Anal. Appl., 25 (2004), pp. 858–899.
- [3] L. DIECI AND A. PAPINI, *Padé approximation for the exponential of a block triangular matrix*, Linear Algebra Appl., 308 (2000), pp. 183–202.
- [4] E. D. DOLAN AND J. J. MORÉ, *Benchmarking optimization software with performance profiles*, Math. Program., 91 (2002), pp. 201–213.
- [5] G. F. FRANKLIN, J. D. POWELL, AND M. L. WORKMAN, *Digital Control of Dynamic Systems*, 3rd ed., Addison-Wesley, Reading, MA, 1998.
- [6] W. GAUTSCHI, *Numerical Analysis: An Introduction*, Birkhäuser Boston, Boston, MA, 1997.

- [7] G. H. GOLUB AND C. F. VAN LOAN, *Matrix Computations*, 3rd ed., The Johns Hopkins University Press, Baltimore, MD, 1996.
- [8] T. F. HAVEL, I. NAJFELD, AND J. YANG, *Matrix decompositions of two-dimensional nuclear magnetic resonance spectra*, Proc. Natl. Acad. Sci. USA, 91 (1994), pp. 7962–7966.
- [9] N. J. HIGHAM, *Functions of a Matrix: Theory and Computation*; book in preparation.
- [10] N. J. HIGHAM, *The Matrix Computation Toolbox*, <http://www.ma.man.ac.uk/~higham/mctoolbox>.
- [11] N. J. HIGHAM, *Accuracy and Stability of Numerical Algorithms*, 2nd ed., SIAM, Philadelphia, 2002.
- [12] R. A. HORN AND C. R. JOHNSON, *Topics in Matrix Analysis*, Cambridge University Press, Cambridge, UK, 1991.
- [13] C. S. KENNEY AND A. J. LAUB, *Condition estimates for matrix functions*, SIAM J. Matrix Anal. Appl., 10 (1989), pp. 191–209.
- [14] 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 (1998), pp. 640–663.
- [15] J. D. LAWSON, *Generalized Runge-Kutta processes for stable systems with large Lipschitz constants*, SIAM J. Numer. Anal., 4 (1967), pp. 372–380.
- [16] C. B. MOLER AND C. F. VAN LOAN, *Nineteen dubious ways to compute the exponential of a matrix*, SIAM Rev., 20 (1978), pp. 801–836.
- [17] C. B. MOLER AND C. F. VAN LOAN, *Nineteen dubious ways to compute the exponential of a matrix, twenty-five years later*, SIAM Rev., 45 (2003), pp. 3–49.
- [18] I. NAJFELD AND T. F. HAVEL, *Derivatives of the matrix exponential and their computation*, Adv. in Appl. Math., 16 (1995), pp. 321–375.
- [19] R. B. SIDJE, *Expokit: A software package for computing matrix exponentials*, ACM Trans. Math. Software, 24 (1998), pp. 130–156.
- [20] R. B. SIDJE AND W. J. STEWART, *A numerical study of large sparse matrix exponentials arising in Markov chains*, Comput. Statist. Data Anal., 29 (1999), pp. 345–368.
- [21] M. SOFRONIOU AND G. SPALETTA, *Efficient matrix polynomial computation and application to the matrix exponential*, talk given at the workshop on “Dynamical Systems on Matrix Manifolds: Numerical Methods and Applications,” Bari, Italy, 2004.
- [22] C. F. VAN LOAN, *On the limitation and application of Padé approximation to the matrix exponential*, in Padé and Rational Approximation: Theory and Applications, E. B. Saff and R. S. Varga, eds., Academic Press, New York, 1977, pp. 439–448.
- [23] R. S. VARGA, *Matrix Iterative Analysis*, 2nd ed., Springer-Verlag, Berlin, 2000.
- [24] R. C. WARD, *Numerical computation of the matrix exponential with accuracy estimate*, SIAM J. Numer. Anal., 14 (1977), pp. 600–610.