A New Scaling and Squaring Algorithm for the Matrix Exponential

Awad H. Al-Mohy and Nicholas J. Higham

August 2009

MIMS EPrint: **2009.9**

Manchester Institute for Mathematical Sciences
School of Mathematics

The University of Manchester

Reports available from: http://www.manchester.ac.uk/mims/eprints
And by contacting: The MIMS Secretary
School of Mathematics
The University of Manchester
Manchester, M13 9PL, UK

ISSN 1749-9097

SIAM J. MATRIX ANAL. APPL.
Vol. 31, No. 3, pp. 970–989

A NEW SCALING AND SQUARING ALGORITHM FOR THE MATRIX EXPONENTIAL*

AWAD H. AL-MOHY† AND NICHOLAS J. HIGHAM†

Abstract. The scaling and squaring method for the matrix exponential is based on the approximation $e^A \approx (r_m(2^{-s}A))^{2^s}$, where $r_m(x)$ is the $[m/m]$ Padé approximant to e^x and the integers m and s are to be chosen. Several authors have identified a weakness of existing scaling and squaring algorithms termed overscaling, in which a value of s much larger than necessary is chosen, causing a loss of accuracy in floating point arithmetic. Building on the scaling and squaring algorithm of Higham [*SIAM J. Matrix Anal. Appl.*, 26 (2005), pp. 1179–1193], which is used by the MATLAB function `expm`, we derive a new algorithm that alleviates the overscaling problem. Two key ideas are employed. The first, specific to triangular matrices, is to compute the diagonal elements in the squaring phase as exponentials instead of from powers of r_m. The second idea is to base the backward error analysis that underlies the algorithm on members of the sequence $\{\|A^k\|^{1/k}\}$ instead of $\|A\|$, since for nonnormal matrices it is possible that $\|A^k\|^{1/k}$ is much smaller than $\|A\|$, and indeed this is likely when overscaling occurs in existing algorithms. The terms $\|A^k\|^{1/k}$ are estimated without computing powers of A by using a matrix 1-norm estimator in conjunction with a bound of the form $\|A^k\|^{1/k} \le \max\bigl(\|A^p\|^{1/p}, \|A^q\|^{1/q}\bigr)$ that holds for certain fixed p and q less than k. The improvements to the truncation error bounds have to be balanced by the potential for a large $\|A\|$ to cause inaccurate evaluation of r_m in floating point arithmetic. We employ rigorous error bounds along with some heuristics to ensure that rounding errors are kept under control. Our numerical experiments show that the new algorithm generally provides accuracy at least as good as the existing algorithm of Higham at no higher cost, while for matrices that are triangular or cause overscaling it usually yields significant improvements in accuracy, cost, or both.

Key words. matrix exponential, matrix function, scaling and squaring method, Padé approximation, backward error analysis, matrix norm estimation, overscaling, MATLAB, `expm`

AMS subject classifications. 15A60, 65F30

DOI. 10.1137/09074721X

1. Introduction.

The scaling and squaring method is the most popular method for computing the matrix exponential. It is used, for example, in Mathematica (function `MatrixExp`), MATLAB (function `expm`), SLICOT (subroutine MB05OD) [17], and the Expokit package [16]. It is also used in more general contexts, such as for computing the group exponential of a diffeomorphism [2]. The method is based on the approximation

$$e^A = (e^{2^{-s}A})^{2^s} \approx r_m(2^{-s}A)^{2^s}, \tag{1.1}$$

where r_m is the $[m/m]$ Padé approximant to e^x and the nonnegative integers m and s are chosen in a prescribed way that aims to achieve full machine accuracy at minimal cost. In practice the method behaves reliably in floating point arithmetic across a wide range of matrices. The method does, however, have a weakness manifested in a subtle phenomenon known as *overscaling*, in which a large $\|A\|$ causes a larger

*Received by the editors January 20, 2009; accepted for publication (in revised form) by D. P. O'Leary May 12, 2009; published electronically August 12, 2009.
http://www.siam.org/journals/simax/31-3/74721.html

†School of Mathematics, The University of Manchester, Manchester, M13 9PL, UK (almohy@maths.manchester.ac.uk, http://www.maths.manchester.ac.uk/~almohy, higham@ma.man.ac.uk, http://www.ma.man.ac.uk/~higham). The work of the second author was supported by a Royal Society-Wolfson Research Merit Award and by Engineering and Physical Sciences Research Council grant EP/D079403.

TABLE 1.1

Errors and condition numbers for A in (1.2) and $B = Q^*AQ$. The columns headed "s" show the values of s used by `expm` *to produce the results in the previous column. The superscripts † and ‡ denote that a particular choice of s was forced: $s = 0$ for † and the $s \in [0, 25]$ giving the most accurate result for ‡.*

b	`expm`(A)	s	`expm`$(A)^\dagger$	`funm`(A)	`expm`(B)	s	`expm`$(B)^\ddagger$	s	`funm`(B)	$\kappa_{\exp}(A)$
10^3	1.7e-15	8	1.9e-16	1.9e-16	2.8e-12	8	2.6e-13	4	2.9e-14	1.6e5
10^4	1.8e-13	11	7.6e-20	3.8e-20	4.0e-8	12	1.9e-10	1	4.1e-10	1.6e7
10^5	7.5e-13	15	1.2e-16	1.2e-16	2.2e-5	15	5.0e-7	4	1.3e-8	1.6e9
10^6	1.3e-11	18	2.0e-16	2.0e-16	8.3e-4	18	7.5e-6	13	7.5e-8	1.6e11
10^7	7.2e-11	21	1.6e-16	1.6e-16	1.2e2	22	6.9e-1	14	6.2e-4	1.6e13
10^8	3.0e-12	25	1.3e-16	1.3e-16	4.4e37	25	1.0e0	3	6.3e-2	1.6e15

than necessary s to be chosen, with a harmful effect on accuracy. We illustrate the phenomenon with the matrix

$$A = \begin{bmatrix} 1 & b \\ 0 & -1 \end{bmatrix}, \qquad e^A = \begin{bmatrix} e & \frac{b}{2}(e - e^{-1}) \\ 0 & e^{-1} \end{bmatrix}. \tag{1.2}$$

Our computations are carried out in MATLAB 7.6 (R2008a), which uses IEEE double precision arithmetic with unit roundoff $u = 2^{-53} \approx 1.1 \times 10^{-16}$. We computed the exponential of A using `expm`, which implements the algorithm of Higham [10], and `funm`, which is applicable to general matrix functions and implements the Schur–Parlett method of Davies and Higham [3], [11, Sec. 10.4.3]. For b ranging from 10^3 to 10^8, the normwise relative errors in the Frobenius norm are shown in the columns of Table 1.1 headed "`expm`(A)" and "`funm`(A)". We see that while `funm` provides full accuracy in every case, the accuracy for `expm` deteriorates with increasing b. As b increases so does the chosen s in (1.1), with m equal to 13 in each case, which is the maximum value that `expm` allows. For $b = 10^8$ the diagonal elements of e^A are approximated by $r_m(x)^{2^{25}} \approx \big((1 + x/2)/(1 - x/2)\big)^{2^{25}}$ with $x = \pm 2^{-25} \approx \pm 10^{-8}$. Approximately half the digits of x are lost in forming $1 \pm x$ and the repeated squarings can only amplify the loss. The essential problem is loss of significance due to too large a choice of s. If we force `expm` to take smaller values of s (still with $m = 13$) we find that the accuracy of the computed exponential increases as s decreases, until a result correct to full machine precision is obtained for $s = 0$, as shown in the column of the table headed "`expm`$(A)^\dagger$". Note that $s = 0$ corresponds to disregarding the large values of a_{12} completely.

To gain some more insight we note the following expression for the exponential of a block 2×2 block triangular matrix (see, e.g., [11, Problem 10.12], [18]):

$$\exp\left(\begin{bmatrix} A_{11} & A_{12} \\ 0 & A_{22} \end{bmatrix}\right) = \begin{bmatrix} e^{A_{11}} & \displaystyle\int_0^1 e^{A_{11}(1-s)} A_{12} e^{A_{22}s}\, ds \\ 0 & e^{A_{22}} \end{bmatrix}. \tag{1.3}$$

Note that A_{12} appears only in the (1,2) block of e^A, where it enters *linearly*. This suggests that the approximation procedure for e^A should be unaffected by $\|A_{12}\|$ and should depend only on $\|A_{11}\|$ and $\|A_{22}\|$. And if A_{11} and A_{22} are upper triangular this argument can be recurred to reason that only the diagonal elements of A should influence the parameters m and s. The need for exponentials of triangular matrices does arise in practice, for example in the solution of radioactive decay equations [15], [20].

In practice A is most often full, rather than (block) triangular. Of course a full matrix A can be reduced by unitary similarities to a triangular matrix T via a Schur decomposition, but applying the scaling and squaring method to T is not numerically equivalent to applying it to A. To investigate possible overscaling for full matrices we repeated the experiment above using $B = Q^*AQ$, where A is as in (1.2) and Q is a random orthogonal matrix, different for each b. The relative normwise errors are shown in the second group of columns of Table 1.1. We see that both `expm` and `funm` produce errors increasing with b, those from `expm` being somewhat larger. The column headed "`expm`$(B)^{\ddagger}$" shows that by forcing an optimal choice of s, `expm` can be made significantly more accurate, so `expm` is again suffering from overscaling.

To determine whether the computed results from `expm` are acceptable we need to know the condition number of the problem, which is

$$\kappa_{\exp}(A) := \lim_{\epsilon \to 0} \sup_{\|E\| \le \epsilon \|A\|} \frac{\|e^{A+E} - e^A\|}{\epsilon \|e^A\|}. \tag{1.4}$$

We evaluated this condition number in the Frobenius norm (for which $\kappa_{\exp}(A) = \kappa_{\exp}(B)$) using a combination of [11, Alg. 3.17] and [1, Alg. 6.4], implemented in a modified version of `expm_cond` from the Matrix Function Toolbox [8]. The results are shown in the final column of Table 1.1. For a stable method we expect an error bounded by a modest multiple of $\kappa_{\exp}(A)u$. Thus `funm` is performing stably but `expm` is behaving unstably, especially for $b = 10^7, 10^8$.

For the original A, the errors for `expm` are all substantially less than $\kappa_{\exp}(A)u$, but of course this condition number allows arbitrary perturbations and so is not appropriate for this triangular A. For structured condition numbers for (block) triangular matrices see Dieci and Papini [5].

Our simple 2×2 example reveals two things. First, that for triangular matrices overscaling can happen because of large off-diagonal elements. Second, that for full matrices overscaling is also possible and may cause unstable behavior of the scaling and squaring method.

The goal of this work is to modify the scaling and squaring method in order to overcome the overscaling problem. To this end we employ two novel ideas, one specific to triangular matrices and one applying to general matrices.

Triangular matrices. For the triangular matrix (1.2) we noted that the diagonal elements are calculated inaccurately by `expm` for large $|b|$. A simple solution is to replace the diagonal elements of the computed exponential by $e^{a_{ii}}$. To benefit the off-diagonal elements as well, we can replace the values $r_m(2^{-s}a_{ii})^{2^j}$ in the squaring phase by $e^{2^{j-s}a_{ii}}$, thereby attenuating the propagation of errors.

Full matrices. For full matrices we introduce a new way of sharpening the truncation error bounds that are used in the derivation of the method. This allows the method to take a potentially smaller s, and hence evaluate the Padé approximant at a larger-normed matrix and require fewer squarings. We will argue that the sharpening is likely to have a particularly beneficial effect when overscaling is possible.

Our key idea is to exploit the sequence $\{\|A^k\|^{1/k}\}$. It is easy to see that

$$\rho(A) \le \|A^k\|^{1/k} \le \|A\|, \qquad k = 1\colon \infty, \tag{1.5}$$

where ρ is the spectral radius, and moreover $\|A^k\|^{1/k} \to \rho(A)$ as $k \to \infty$ [13, Cor. 5.6.14]. Figure 1.1 plots the sequence $\{\|A^k\|_2^{1/k}\}_{k=1}^{20}$ for 54 nonnormal 16×16 matrices A, normalized (without loss of generality) so that $\|A\|_2 = 1$, drawn from the

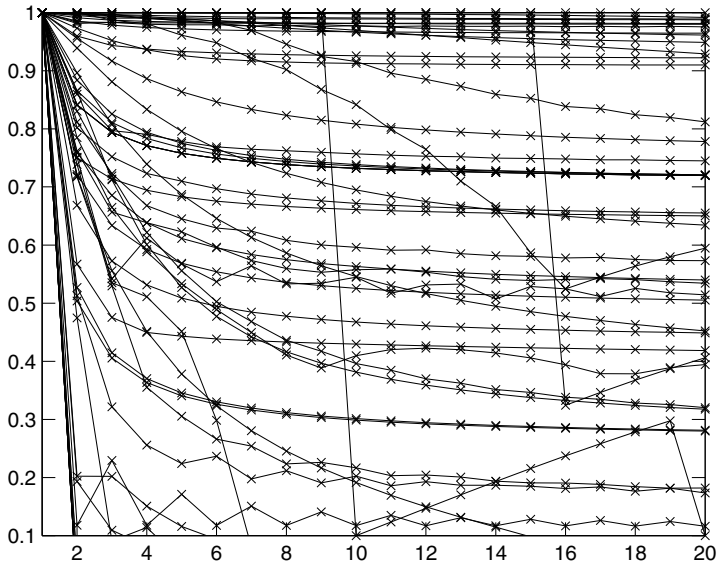

FIG. 1.1. $\{\|A^k\|_2^{1/k}\}_{k=1}^{20}$ *for* 54 16×16 *matrices* A *with* $\|A\|_2 = 1$.

MATLAB function `gallery`, from the Matrix Computation Toolbox [7], and from the e^A literature. We see that typically the sequence is decreasing, although very nonmonotonic behavior is possible. It is this decrease that we will exploit.

In the derivation of the scaling and squaring algorithm of [10] a power series

$$h_\ell(x) = \sum_{k=\ell}^{\infty} c_k\, x^k$$

has to be bounded at the matrix argument A (or, more precisely, $2^{-s}A$, but we drop the scale factor for now), where $\ell = 2m + 1$. The bound used previously is [10], [11, Sec. 10.3]

$$\|h_\ell(A)\| \le \sum_{k=\ell}^{\infty} |c_k|\, \|A\|^k. \tag{1.6}$$

The following theorem provides a sharper bound.

THEOREM 1.1. *Let* $h_\ell(x) = \sum_{k=\ell}^{\infty} c_k\, x^k$ *be a power series with radius of convergence* ω*, and let* $\widetilde{h}_\ell(x) = \sum_{k=\ell}^{\infty} |c_k|\, x^k$*. For any* $A \in \mathbb{C}^{n\times n}$ *with* $\rho(A) < \omega$ *we have*

$$\|h_\ell(A)\| \le \widetilde{h}_\ell(\|A^t\|^{1/t}), \tag{1.7}$$

where $\|A^t\|^{1/t} = \max\{\|A^k\|^{1/k} : k \ge \ell \text{ and } c_k \ne 0\}$.

Proof. The existence of t is guaranteed since the sequence $\{\|A^k\|^{1/k}\}$ is bounded above and convergent, as noted above. We have

$$\begin{aligned}\|h_\ell(A)\| \le \sum_{k=\ell}^{\infty} |c_k|\, \|A^k\| &= \sum_{k=\ell}^{\infty} |c_k| \left(\|A^k\|^{1/k}\right)^k \\ &\le \sum_{k=\ell}^{\infty} |c_k| \left(\|A^t\|^{1/t}\right)^k = \widetilde{h}_\ell(\|A^t\|^{1/t}). \qquad \square\end{aligned}$$

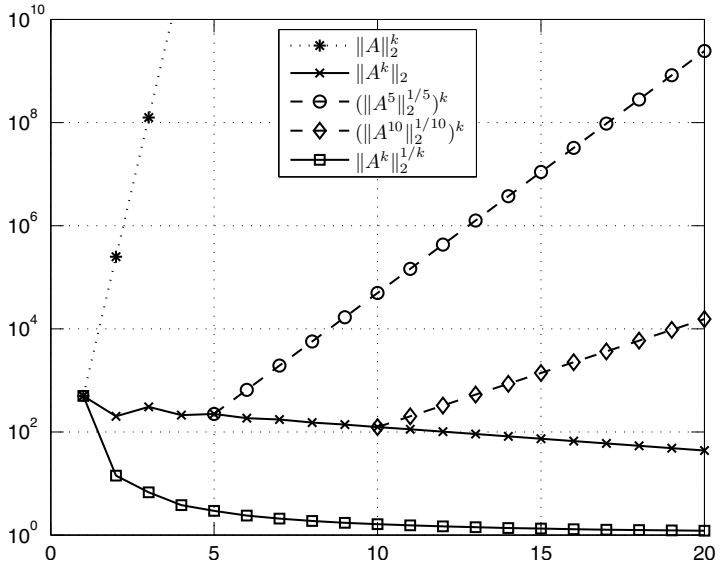

FIG. 1.2. *For the* 2×2 *matrix* A *in* (1.9), $\|A^k\|_2$ *and various bounds for* $k = 1\colon 20$.

The bound (1.7) is clearly sharper than (1.6) since $\|A^t\|^{1/t} \le \|A\|$, and it can be arbitrarily smaller. In particular, if $A \ne 0$ is nilpotent and $\ell \ge n$ then the bound (1.7) is zero while (1.6) is nonzero unless $h_\ell(x) \equiv 0$. Note that as essentially a special case of Theorem 1.1, if the sequence $\{\|A^k\|\}_{k \ge i}$ is nonincreasing then

$$\|A^k\| \le \left(\|A^i\|^{1/i}\right)^k, \qquad k \ge i. \tag{1.8}$$

To see why (1.7) may help to avoid overscaling, consider the matrix

$$A = \begin{bmatrix} 0.9 & 500 \\ 0 & -0.5 \end{bmatrix}, \tag{1.9}$$

for which the 2-norms of the powers of A decay monotonically for $k \ge 5$, despite the large (1,2) element. Figure 1.2 plots $\|A^k\|_2$, the crude bound $\|A\|_2^k$, and the more refined bounds $(\|A^5\|_2^{1/5})^k$ valid for $k \ge 5$ and $(\|A^{10}\|_2^{1/10})^k$ valid for $k \ge 10$, by (1.8). The crude bound is an extreme overestimate and the refined bounds are a significant improvement. The reason for the improvement is that when A is powered the large (1,2) element multiplies the diagonal elements and there is both multiplicative and subtractive cancellation, resulting in little or no growth. The refined bounds take advantage of this. For the power series h_ℓ, such a reduction in the bound for $\|A^k\|$ translates into a reduction in the bound for $h_\ell(A)$, and this in turn can lead to a much smaller s being chosen in the scaling and squaring method.

In essence, what we are doing is using the behavior of the first few powers of A to extract information about the nonnormality of A. In the scaling and squaring method we will exploit the powers that must be computed anyway during the course of the algorithm, thereby gaining potentially improved accuracy and reduced cost. This idea has already been used in an ad hoc way by Hargreaves and Higham [6], who in the context of computing the matrix cosine express error bounds in terms of $\|A^2\|^{1/2}$ instead of $\|A\|$, but here we are exploiting the idea more systematically.

This paper is organized as follows. We begin, in section 2, by showing how to improve the squaring phase of the scaling and squaring method for triangular matrices. In section 3 we summarize the scaling and squaring algorithm of Higham. Section 4 presents new bounds for norms of matrix powers that are then exploited in section 5 to produce a new algorithm that is often more accurate, more efficient, or both. Numerical experiments that illustrate the benefits of the new algorithm are given in section 6, and section 7 presents conclusions.

Finally, we make connections with earlier work. The overscaling phenomenon was first identified by Kenney and Laub [14]. It was later analyzed by Dieci and Papini [4] for the case of block 2×2 block upper triangular matrices $(A_{ij})_{i,j=1}^{2}$. The latter analysis suggests that if the scaling and squaring method is used with s determined so that $2^{-s}\|A_{11}\|$ and $2^{-s}\|A_{22}\|$ are appropriately bounded, without consideration of $\|A_{12}\|$, then an accurate approximation to e^A will still be obtained. However, no algorithm for general A was proposed in [4].

2. Squaring phase for triangular matrices. Our new approach for triangular matrices was inspired by the practical observation that the scaling and squaring method seems to be immune to overscaling for nilpotent triangular matrices T—those with zero diagonal elements. Indeed, the diagonal entries of $r_m(2^{-s}T)$ are correctly computed as ones and remain as ones through the squaring process. Now for a general upper triangular matrix T, the scaling and squaring method computes $r_m(2^{-s}T) =: D_s + F_s$, where D_s is diagonal and F_s is strictly upper triangular, and then forms

$$D_{i-1} + F_{i-1} = (D_i + F_i)^2, \qquad i = s : -1 : 1,$$

after which $D_0 + F_0 \approx e^T$. Hence we have the recurrence

$$\left.\begin{aligned} D_{i-1} &= D_i^2 \\ F_{i-1} &= D_i F_i + F_i D_i + F_i^2 \end{aligned}\right\} \quad i = s : -1 : 1. \tag{2.1}$$

Clearly, errors in the computation of the D_i propagate into the off-diagonals contained in F_{i-1}. Indeed a single error ϵI (say) in D_i transmits into F_{i-1} as $2\epsilon F_i$ and into D_{i-1} as $2\epsilon D_i$, so there is potential exponential error growth. We can virtually remove errors in the diagonal and thereby attenuate the overall error growth by computing $D_i = \exp(2^{-i}\mathrm{diag}(T))$ at each stage instead of computing $D_s = r_m(2^{-s}\mathrm{diag}(T))$ and then repeatedly squaring. Thus the final steps of the scaling and squaring method are rewritten as follows.

Code Fragment 2.1.

```
1  Form X = r_m(2^{-s}T). % First phase of method (unchanged).
2  Replace diag(X) by exp(2^{-s}diag(T)).
3  for i = s - 1: -1: 0
4      X ← X^2
5      Replace diag(X) by exp(2^{-i}diag(T)).
6      Replace (first) superdiagonal of X by explicit formula
       for superdiagonal of exp(2^{-i}T) from [11, eq. (10.42)].
7  end
```

Note that we have gone further in line 6 by computing the correct superdiagonal as well, since it is available from an accurate formula at negligible cost.

We give a numerical example to illustrate the benefits of this approach. We take the matrix formed by the MATLAB code, with $n = 8$,

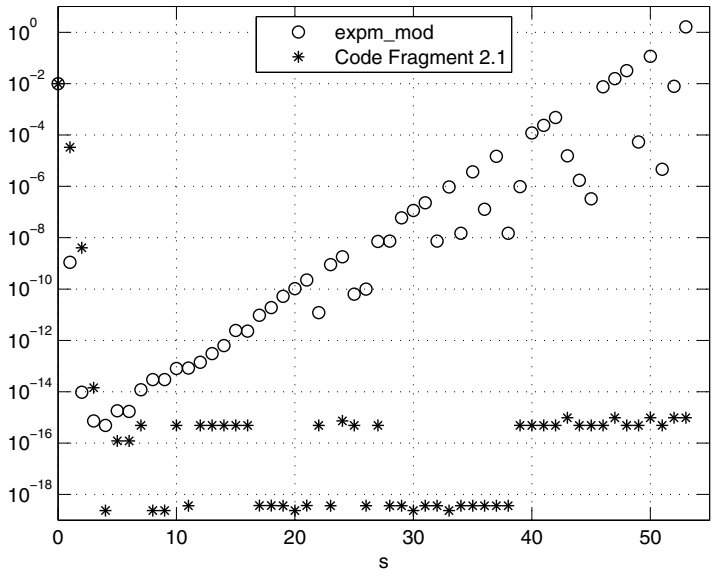

FIG. 2.1. *Relative errors from Code Fragment* 2.1 *and* `expm_mod` *for a single* 8×8 *matrix with* $s = 0\colon 53$.

```
T = gallery('triw',n,-1);  T(1,n) = 1e4; T(1:n+1:n^2) = -(1:n).^2
```

The MATLAB function `expm` chooses $s = 11$ and $m = 13$ for this matrix and produces a relative error 8.4×10^{-14}. For s from 0 to 53 we compare Code Fragment 2.1, using $m = 13$, with a modified version `expm_mod` of `expm` that accepts a user-specified choice of s. The normwise relative errors for both methods are plotted in Figure 2.1. The optimum value of s for `expm_mod` is 4, for which a relative error 4.9×10^{-16} is achieved; as s increases the relative error deteriorates rapidly until it reaches 1 at $s = 53$. However, Code Fragment 2.1 remains fully accurate for all $s \geq 4$, showing the effectiveness of the strategy of injecting the correct diagonal into the recurrence.

This approach can be extended to quasi-triangular matrices T, which are block triangular matrices whose diagonal blocks T_{ii} are 1×1 or 2×2. Such T arise in the real Schur decomposition of a real matrix, in which case the 2×2 blocks T_{ii} have distinct eigenvalues that are nonreal complex conjugates. We need to compute the exponentials of the diagonal blocks

$$A = \begin{bmatrix} a & b \\ c & d \end{bmatrix},$$

which we assume have distinct eigenvalues λ_1, λ_2. From the general formula $f(A) = f(\lambda_1)I + f[\lambda_1, \lambda_2](A - \lambda_2 I)$, where $f[\lambda_1, \lambda_2]$ is a divided difference [11, Prob. 1.9], we obtain

$$e^A = \frac{e^{\lambda_1} - e^{\lambda_2}}{\lambda_1 - \lambda_2} A + \left(e^{\lambda_1} - \lambda_2 \frac{e^{\lambda_1} - e^{\lambda_2}}{\lambda_1 - \lambda_2} \right) I.$$

The eigenvalues of A are $(a+d\pm\mu)/2$, where $\mu=\sqrt{(a-d)^2+4bc}$. After some manipulation we obtain

(2.2)
$$e^A = e^{(a+d)/2}\begin{bmatrix} \cosh(\mu/2)+\frac{1}{2}(a-d)\,\mathrm{sinch}(\mu/2) & b\,\mathrm{sinch}(\mu/2) \\ c\,\mathrm{sinch}(\mu/2) & \cosh(\mu/2)-\frac{1}{2}(a-d)\,\mathrm{sinch}(\mu/2)\end{bmatrix},$$

where

$$\mathrm{sinch}(x)=\begin{cases}\sinh(x)/x, & x\neq 0,\\ 1, & x=0.\end{cases}$$

This formula is not always evaluated to high relative accuracy, so the use of extra precision in its evaluation might be justified.

Combining this formula with an initial real Schur decomposition we have the following outline applicable to any $A\in\mathbb{R}^{n\times n}$.

Code Fragment 2.2.

```
1  Compute the real Schur decomposition, A = QTQ*,
   with block q × q upper quasi-triangular T = (T_ij).
2  Form X = r_m(2^{-s}T).
3  Replace diag(X_ii) by exp(2^{-s}diag(T_ii)), i = 1:q, using (2.2).
4  for i = s-1:-1:0
5      X ← X^2
6      Replace diag(X_ii) by exp(2^{-i}diag(T_ii)), i = 1:q, using (2.2).
7  end
8  X ← QXQ*
```

Note that this approach has the advantage that it works entirely in real arithmetic, in contrast to the Schur–Parlett method specialized to the exponential, which necessarily uses the complex Schur form [3], [11, Sec. 10.4.3].

Our main interest is in improving the scaling and squaring method for full matrices without using a Schur decomposition. In the next section we summarize the derivation of the existing algorithm on which we will work.

3. The existing scaling and squaring algorithm. In this section we review the scaling and squaring algorithm of Higham [10]. Write the $[m/m]$ Padé approximant of e^x as $r_m(x)=p_m(x)/q_m(x)$. We will later need the properties that p_m has positive coefficients and $q_m(x)=p_m(-x)$.

Let $\nu_m=\min\{\,|t|: q_m(t)=0\,\}$ and

$$\Omega_m=\{\,X\in\mathbb{C}^{n\times n}: \rho(e^{-X}r_m(X)-I)<1 \text{ and } \rho(X)<\nu_m\,\}.$$

The functions

$$h_{2m+1}(X)=\log(e^{-X}r_m(X)) \tag{3.1}$$

are defined for $X\in\Omega_m$, where log denotes the principal logarithm, and so for $X\in\Omega_m$ we have $r_m(X)=e^{X+h_{2m+1}(X)}$. Now choose s so that $2^{-s}A\in\Omega_m$. Then $r_m(2^{-s}A)^{2^s}=e^{A+2^s h_{2m+1}(2^{-s}A)}=:e^{A+\Delta A}$ and the matrix $\Delta A=2^s h_{2m+1}(2^{-s}A)$ represents the backward error resulting from the approximation of e^A by the scaling and squaring method. Over Ω_m, the functions h_{2m+1} have a power series expansion

$$h_{2m+1}(X)=\sum_{k=2m+1}^{\infty} c_k X^k.$$

TABLE 3.1
Parameters θ_m needed in Algorithms 3.1 *and* 5.1 *and upper bounds for $\kappa_A(q_m(A))$.*

m	θ_m	$\kappa_A(q_m(A))$
3	1.495585217958292e-2	1.0e0
5	2.539398330063230e-1	1.3e0
7	9.504178996162932e-1	2.6e0
9	2.097847961257068e0	8.2e0
13 (Alg. 3.1)	5.371920351148152e0	2.2e2
13 (Alg. 5.1)	4.25	7.1e1

Higham [10] employs the bound[1]

$$\frac{\|\Delta A\|}{\|A\|} = \frac{\|h_{2m+1}(2^{-s}A)\|}{\|2^{-s}A\|} \le \frac{\widetilde{h}_{2m+1}(\|2^{-s}A\|)}{\|2^{-s}A\|}, \tag{3.2}$$

where $\widetilde{h}_{2m+1}(x) = \sum_{k=2m+1}^{\infty} |c_k|\, x^k$. For $m = 1\colon 21$, he uses high precision arithmetic to compute the values

$$\theta_m = \max\{\, \theta : \widetilde{h}_{2m+1}(\theta)/\theta \le u \,\}, \tag{3.3}$$

some of which are listed in Table 3.1. He finds that $\theta_m < \nu_m$. Thus in *exact* arithmetic, $\|\Delta A\| \le u\|A\|$ if s is chosen so that $\|2^{-s}A\| \le \theta_m$ and $r_m(2^{-s}A)^{2^s}$ is used to approximate e^A. Higham's cost analysis eliminates even degrees of Padé approximants and reveals that $m = 13$ is the optimal degree to use when scaling is required. When $\|A\| \le \theta_{13}$, his algorithm chooses the first $m \in \{3, 5, 7, 9, 13\}$ such that $\|A\| \le \theta_m$.

For $m = 3, 5, 7, 9$, Higham uses the evaluation scheme

$$p_m(A) = A \sum_{k=0}^{(m-1)/2} b_{2k+1} A_{2k} + \sum_{k=0}^{(m-1)/2} b_{2k} A_{2k} =: (u_m + v_m)(A), \tag{3.4}$$

where $A_{2k} = A^{2k}$. For $m = 13$, he uses the scheme

$$\begin{aligned} p_{13}(A) \;=\; & A\big[A_6(b_{13}A_6 + b_{11}A_4 + b_9A_2) + b_7A_6 + b_5A_4 + b_3A_2 + b_1I\big] \\ & + A_6(b_{12}A_6 + b_{10}A_4 + b_8A_2) + b_6A_6 + b_4A_4 + b_2A_2 + b_0I \\ =: & (u_{13} + v_{13})(A), \end{aligned} \tag{3.5}$$

where $A_2 = A^2$, $A_4 = A_2^2$, and $A_6 = A_2A_4$. Since $q_m(A) = p_m(-A)$, we have $q_m(A) = (-u_m + v_m)(A)$ and $r_m(A)$ is obtained from the equation

$$(-u_m + v_m)(A)r_m(A) = (u_m + v_m)(A). \tag{3.6}$$

The algorithm of [10] can be summarized as follows.

ALGORITHM 3.1 (scaling and squaring algorithm for the matrix exponential). *This algorithm evaluates the matrix exponential $X = e^A$ of $A \in \mathbb{C}^{n\times n}$ by the scaling and squaring method. It uses the parameters θ_m given in Table* 3.1*. The algorithm is intended for IEEE double precision arithmetic.*

[1] In fact, (3.2) is a slightly sharper variant of the bound used by Higham [10], but it leads to the same parameters.

```
for m = [3 5 7 9]
    if ‖A‖_1 ≤ θ_m, evaluate X = r_m(A) using (3.4) and (3.6), quit, end
end
A ← 2^{-s}A with s = ⌈log_2(‖A‖_1/θ_13)⌉
Evaluate r_13(A) using (3.5) and (3.6).
X = r_13(A)^{2^s} by repeated squaring.
```

Cost: $(\pi_m + s)M + D$, where m is the degree of Padé approximant used and π_m (tabulated in [10, Table 2.2]) is the cost of evaluating p_m and q_m.

4. Practical bounds for norm of matrix power series. The bound for $\|h_\ell(A)\|$ in Theorem 1.1 is not readily usable in the form stated since it employs $\|A^t\|^{1/t}$ and we will rarely know the value of t. We now develop a more convenient form of bound that will be used in the next section to improve Algorithm 3.1. We denote by $\mathbb{N}$ the set of positive integers.

LEMMA 4.1. *For any $k \geq 1$ such that $k = pm_1 + qm_2$ with $p, q \in \mathbb{N}$ and $m_1, m_2 \in \mathbb{N} \cup \{0\}$,*

$$\|A^k\|^{1/k} \leq \max\bigl(\|A^p\|^{1/p}, \|A^q\|^{1/q}\bigr).$$

Proof. Let $\delta = \max(\|A^p\|^{1/p}, \|A^q\|^{1/q})$. The bound follows from the inequality

$$\begin{aligned}\|A^k\| &\leq \|A^{pm_1}\|\,\|A^{qm_2}\| \\ &\leq \left(\|A^p\|^{1/p}\right)^{pm_1}\left(\|A^q\|^{1/q}\right)^{qm_2} \\ &\leq \delta^{pm_1}\delta^{qm_2} = \delta^k. \qquad \square\end{aligned}$$

THEOREM 4.2. *Define h_ℓ and $\widetilde{h}_\ell$ as in Theorem 1.1 and suppose $\rho(A) < \omega$ and $p \in \mathbb{N}$. Then*

(a) $\|h_\ell(A)\| \leq \widetilde{h}_\ell\bigl(\max(\|A^p\|^{1/p}, \|A^{p+1}\|^{1/(p+1)})\bigr)$ *if $\ell \geq p(p-1)$.*

(b) $\|h_\ell(A)\| \leq \widetilde{h}_\ell\bigl(\max(\|A^{2p}\|^{1/(2p)}, \|A^{2p+2}\|^{1/(2p+2)})\bigr)$ *if $\ell \geq 2p(p-1)$ and h_ℓ is even.*

Proof. For the first part, let $X_p = \{\, m \in \mathbb{N} : m \geq p(p-1) \,\}$ and $Y_p = \{\, (p+1)m_1 + pm_2 : m_1, m_2 \in \mathbb{N} \cup \{0\} \,\}$. We have $X_p \subset Y_p$ since any element $k \in X_p$ can be written as $k = pq + r$ with $q \geq p-1$ and $0 \leq r < p$. Then $k = (p+1)r + p(q-r) \in Y_p$, since $q - r \geq p - r - 1 \geq 0$. Hence by Lemma 4.1 we have

$$\|A^k\|^{1/k} \leq \max\{\, \|A^{p+1}\|^{1/(p+1)}, \|A^p\|^{1/p} \,\}, \quad k \geq p(p-1),$$

and the result follows from Theorem 1.1. Part (b) follows similarly from the relations

$$\begin{aligned}\{\, \ell : \ell \text{ even}, \ell \geq 2p(p-1) \,\} &= 2X_p \subset 2Y_p \\ &= \{\, (2p+2)m_1 + 2pm_2 : m_1, m_2 \in \mathbb{N} \cup \{0\} \,\}. \qquad \square\end{aligned}$$

To illustrate Theorem 4.2, suppose $\ell = 12$. In view of the inequality $\|A^{2k}\|^{1/(2k)} \leq \|A^k\|^{1/k}$, the p that minimizes $\max(\|A^p\|^{1/p}, \|A^{p+1}\|^{1/(p+1)})$ subject to $\ell \geq p(p-1)$ is either $p = 3$ or $p = 4$. So the first part of the theorem gives

$$\|h_{12}(A)\| \leq \widetilde{h}_{12}\bigl(\min\bigl(\max(\|A^3\|^{1/3}, \|A^4\|^{1/4}), \max(\|A^4\|^{1/4}, \|A^5\|^{1/5})\bigr)\bigr). \tag{4.1}$$

If h_{12} is even we can apply the second part of the theorem, for which the optimal choice of p is 3, which gives

$$\|h_{12}(A)\| \leq \widetilde{h}_{12}\bigl(\max(\|A^6\|^{1/6}, \|A^8\|^{1/8})\bigr). \tag{4.2}$$

For a normal matrix and the 2-norm these upper bounds are both identical to the bound $h_{12}(\|A\|_2)$, but for nonnormal matrices they can be significantly smaller. For A in (1.9), we have $\|h_{12}(A)\|_2 \le \widetilde{h}_{12}(\|A\|_2) \approx \widetilde{h}_{12}(500)$, but $\|h_{12}(A)\|_2 \le \widetilde{h}_{12}(3.82)$ from (4.1) and $\|h_{12}(A)\|_2 \le \widetilde{h}_{12}(2.39)$ from (4.2), demonstrating the benefit of exploiting the structure of h_{12} as an even function. Figure 1.1 suggests that it will typically be the case that (4.2) is sharper than (4.1).

5. New algorithm. We now derive a new algorithm that builds on Algorithm 3.1. Our development focuses on increasing the sharpness of the inequality in (3.2) by using the bounds of the previous section.

The functions h_{2m+1} in (3.1) are odd, which follows from the fact that Padé approximants to the exponential function satisfy $r_m(-x) = (r_m(x))^{-1}$ [11, Sec. 10.3]:

$$\begin{aligned} h_{2m+1}(-x) &= \log(e^x r_m(-x)) \\ &= \log((e^{-x} r_m(x))^{-1}) \\ &= -\log(e^{-x} r_m(x)) = -h_{2m+1}(x). \end{aligned}$$

Therefore for $X \in \Omega_m$ we can write

$$h_{2m+1}(X) = X \sum_{k=2m}^{\infty} c_{k+1} X^k =: X\phi_{2m}(X),$$

where the ϕ_{2m} are even functions. Let $\widetilde{\phi}_{2m}(x) = \sum_{k=2m}^{\infty} |c_{k+1}|\, x^k$. We can now refine the bound in (3.2) using Theorem 4.2(b):

$$\begin{aligned} \frac{\|\Delta A\|}{\|A\|} = \frac{\|h_{2m+1}(2^{-s}A)\|}{\|2^{-s}A\|} &= \frac{\|2^{-s}A\phi_{2m}(2^{-s}A)\|}{\|2^{-s}A\|} \\ &\le \|\phi_{2m}(2^{-s}A)\| \le \widetilde{\phi}_{2m}(\alpha_p), \end{aligned} \tag{5.1a}$$

where

$$\alpha_p = 2^{-s} \max\bigl(\|A^{2p}\|^{1/(2p)}, \|A^{2p+2}\|^{1/(2p+2)}\bigr) \tag{5.1b}$$

and we choose p to minimize α_p subject to $2m \ge 2p(p-1)$. As we have $\widetilde{\phi}_{2m}(\theta) = \widetilde{h}_{2m+1}(\theta)/\theta$, clearly this analysis does not affect the calculation of the values θ_m in (3.3), but it does affect the choice of scaling parameter s. Whereas before, the pair (s, m) could be used if $2^{-s}\|A\| \le \theta_m$, now the requirement is only

$$2^{-s} \max\bigl(\|A^{2p}\|^{1/(2p)}, \|A^{2p+2}\|^{1/(2p+2)}\bigr) \le \theta_m, \tag{5.2}$$

and for a given m this is potentially satisfied with a much smaller s when A is nonnormal. A significant computational saving could therefore accrue.

At this point, we have an improved way to choose the parameters m and s, but we have not yet considered the effect of rounding errors. The analysis in [10] shows that Algorithm 3.1 is not unduly affected by rounding errors except, possibly, in the squaring phase. But with our more liberal choice of parameters numerical stability needs further analysis. We will combine rigorous error bounds with some heuristics in order to arrive at our final algorithm.

The main aim is to check that the evaluation of r_m is accurate in floating point arithmetic. Let $A \leftarrow 2^{-s}A$, so that A denotes the scaled matrix, and consider the

evaluation of $p_m(A)$ by the schemes described in the previous section. It is shown in [10] that the computed matrix $\widehat{p}_m(A)$ satisfies

$$\|p_m(A) - \widehat{p}_m(A)\|_1 \le \widetilde{\gamma}_{mn} p_m(\|A\|_1) \lesssim \widetilde{\gamma}_{mn} \|p_m(A)\|_1 \, e^{\|A\|_1}, \tag{5.3}$$

where $\widetilde{\gamma}_k = cku/(1-cku)$ with c a small integer constant. While this is a satisfactory bound for the values of $\|A\|$ allowed by Algorithm 3.1, it is not so for an algorithm based on (5.2), because $\|A\|$ can be arbitrarily large. Therefore we will use the sharper error bound [11, Thm. 4.5]

$$\max(\|p_m - \widehat{p}_m\|_1, \|q_m - \widehat{q}_m\|_1) \le \widetilde{\gamma}_{mn} \|p_m(|A|)\|_1 = \widetilde{\gamma}_{mn} \|p_m(|A|)^T e\|_\infty, \tag{5.4}$$

where $e = [1, 1, \ldots, 1]^T$ and we have used the properties of p_m and q_m mentioned at the start of section 3 together with the relations

$$\|B\|_1 = \| \, |B| \, \|_1 = \| \, |B|^T e \, \|_\infty. \tag{5.5}$$

The bound (5.4) can be computed in just $O(n^2)$ operations.

The a priori bound (5.4) applies to several different evaluation schemes and does not exploit the particular properties of our scheme. In particular, it contains $|A|^m$, which is clearly pessimistic since our schemes for p_m and q_m do not explicitly evaluate the mth power of A. However, it turns out that the bound is surprisingly sharp when used within our algorithm in practice. We have found that if the inequality

$$\|\widehat{p}_m(|A|)^T e\|_\infty / \min(\|\widehat{p}_m\|_1, \|\widehat{q}_m\|_1) \le c e^{\theta_m} \tag{5.6}$$

is satisfied for a suitable integer constant c then this is a reliable indicator that $\widehat{p}_m$ and $\widehat{q}_m$ have been computed to close to full precision (in other words, we can replace $\widetilde{\gamma}_{mn}$ by $\widetilde{\gamma}_1$ in (5.4)). We have tried the alternative of computing a running error bound (an a posteriori bound that is the smallest possible) [9, Sec. 3.3], but found that it brings no benefits.

Suppose that (5.6) is not satisfied, which suggests that the evaluation of p_m or q_m may have been insufficiently accurate. Since the weaker bound (5.3) is satisfactory for Algorithm 3.1, this means that $s < s_{\max}$, where $s_{\max}$ is the scaling parameter selected by Algorithm 3.1. We could simply revert to $s_{\max}$ and execute Algorithm 3.1, but instead we will use a strategy that in certain cases increases s based on the available information. One approach is to increase s so that (5.6) is satisfied. However, we have found a more heuristic approach to perform better in practice. Let A denote the original, unscaled matrix. Returning to the bound (5.1a), using $|h_{2m+1}(A)| \le \widetilde{h}_{2m+1}(|A|)$ we have

$$\begin{aligned} \frac{\|\Delta A\|_1}{\|A\|_1} &= \frac{\|h_{2m+1}(2^{-s}A)\|_1}{\|2^{-s}A\|_1} \le \frac{\|\widetilde{h}_{2m+1}(2^{-s}|A|)\|_1}{\|2^{-s}A\|_1} \\ &\le |c_{2m+1}| \frac{\| \, |2^{-s}A|^{2m+1} \, \|_1}{\|2^{-s}A\|_1} + \sum_{k=2m+2}^{\infty} |c_k| \frac{\| \, |2^{-s}A|^k \, \|_1}{\|2^{-s}A\|_1} \\ &\le \widetilde{\phi}_{2m}(\|2^{-s}A\|_1). \end{aligned} \tag{5.7}$$

We select the smallest integer $\ell_m \ge 0$ so that $|c_{2m+1}| \, \||2^{-s-\ell_m}A|^{2m+1}\|_1 / \|2^{-s-\ell_m}A\|_1 \le u$, that is,

$$\ell_m = \max\left(\left\lceil \log_2 \left(|c_{2m+1}| \frac{\||2^{-s}A|^{2m+1}\|_1}{u\|2^{-s}A\|_1} \right) / (2m) \right\rceil, 0 \right). \tag{5.8}$$

If $\ell_m > 0$ then we increase s to $s + \ell_m$. The value $s + \ell_m$ cannot exceed $s_{\max}$ as long as $s \le s_{\max}$. To see why, write $s_{\max} = s + t$ and note that by the definition of $s_{\max}$ we have $\widetilde{\phi}_{2m}(\|2^{-s_{\max}}A\|_1) \le u$, which yields from (5.7) that

$$|c_{2m+1}| \frac{\| \, |2^{-s-t}A|^{2m+1} \, \|_1}{\|2^{-s-t}A\|_1} \le u.$$

As ℓ_m is chosen to be the smallest nonnegative integer such that this relation holds, we have $t \ge \ell_m$, that is, $s + \ell_m \le s_{\max}$. Note that we can evaluate $\| \, |2^{-s}A|^{2m+1} \, \|_1$ in $O(n^2)$ operations by repeated matrix–vector products, as $\| \, |2^{-s}A|^{2m+1}e \, \|_\infty$. Also, while $\| \, |2^{-s}A|^{2m+1} \, \|_1$ can be large, we have $|c_{2m+1}| \ll 1$, so ℓ_m should not typically be large. Experiments show that this heuristic choice of ℓ_m has the ability usually to increase s just when needed. If (5.6) is satisfied we proceed with this s; otherwise we revert to Algorithm 3.1, reusing as many of the computed quantities as possible.

To obtain $r_m(A)$, with A once again denoting the scaled matrix $2^{-s}A$, we solve the multiple right-hand side linear system (3.6) with the coefficient matrix $q_m(A) = -U + V$, where $U = u_m(A)$, $V = v_m(A)$. Since $\rho(A) \le \alpha_p \le \theta_m < \nu_m$ by (1.5) and (5.2), the matrix $q_m(A)$ is nonsingular and the series $q_m(A)^{-1} = \sum_{k=0}^{\infty} a_k A^k$ converges absolutely. But in addition we want $q_m(A)$ to be well conditioned, so that the system (3.6) can be solved accurately in floating point arithmetic. For any $\epsilon > 0$, there exists a consistent matrix norm $\|\cdot\|_A$ such that $\|A\|_A \le \rho(A) + \epsilon \le \alpha_p + \epsilon$. The corresponding condition number is

$$\begin{aligned}\kappa_A(q_m(A)) &= \|q_m(A)\|_A \|q_m(A)^{-1}\|_A \\ &\le p_m(\alpha_p + \epsilon) \sum_{k=0}^{\infty} |a_k| \, (\alpha_p + \epsilon)^k \le p_m(\theta_m + \epsilon) \sum_{k=0}^{\infty} |a_k| \, (\theta_m + \epsilon)^k,\end{aligned}$$

where we have used the properties of p_m and q_m mentioned at the start of this section. We choose $\epsilon = u$ and list these upper bounds for $\kappa_A(q_m(A))$ in Table 3.1. Since the norm $\|\cdot\|_A$ can be very badly scaled the practical value of these bounds for a particular A is difficult to judge. However, we can compute an a posteriori forward error bound for (3.6). This bound is, with X denoting r_m and the residual matrix $\widehat{R} = fl(U + V - (-U + V)\widehat{X})$ for the computed U and V,

$$\frac{\|X - \widehat{X}\|_M}{\|\widehat{X}\|_M} \le \frac{\| \, |(-U+V)^{-1}| (|\widehat{R}| + \gamma_{n+1}(|-U+V||\widehat{X}| + |U+V|)) \, \|_M}{\|\widehat{X}\|_M}, \tag{5.9}$$

where $\|X\|_M = \max_{i,j} |x_{ij}|$ and $\gamma_k = ku/(1-ku)$. This bound is from [9, eq. (7.31)] and is essentially the best possible forward error bound. Given that we already have an LU factorization of $-U + V$ from solving the linear system, this bound can be cheaply estimated without computing $(-U+V)^{-1}$, as described in [9, Sec. 15.1]. The cost of forming the bound (5.9) is therefore essentially one matrix multiplication—that needed for R.

We are now in a position to design the new algorithm. Higham's cost analysis and evaluation schemes stated above remain applicable. From (5.1b) it is easily seen that $\alpha_3 \le \alpha_2 \le \alpha_1$. Thus, for $m = 3, 5, 7, 9$ the optimal values of p subject to the constraint $2m \ge 2p(p-1)$ are $p = 2, 2, 3, 3$, respectively, and for $m = 13$ the optimal value of p is 3 or 4 (either of α_3 and α_4 can be the smaller). Thus, using the 1-norm, we need to compute the quantities

$$\alpha_p = \max\bigl(d_{2p}, d_{2p+2}\bigr), \quad p = 2\colon 4, \qquad d_{2j} := \|A^{2j}\|_1^{1/(2j)}, \quad j = 2\colon 5.$$

However, computing the d_{2j} requires computing powers of A that are not needed to evaluate r_m, for which the highest explicitly computed power is, depending on A, the eighth or lower. We will use the powers of A that are evaluated in the schemes (3.4) and (3.5), and for other powers compute norm *estimates* using the block 1-norm estimation algorithm of Higham and Tisseur [12], which for an $n \times n$ matrix carries out a 1-norm power iteration whose iterates are $n \times t$ matrices, where t is a parameter that we take to be 2. This algorithm typically requires the evaluation of about $4t$ matrix–vector products and almost invariably produces a norm estimate (which is, in fact, a lower bound on the norm) correct to within a factor 3.

Now we describe the details of how to choose m and s, with s minimal, so that the bound in (5.1a) is no larger than u.

1. Compute $A_2 = A^2$ and set $s = 0$ and $m = 3$, so $p = 2$ is the optimal value such that $2m \geq 2p(p-1)$, as explained above. We need $\eta_1 = \max(d_4, d_6)$. Since A^4 and A^6 are not needed by r_3, use estimates of d_4 and d_6 obtained by applying the norm estimator to A_2^2 and A_2^3 (a product $A_2^3 x$ required by the estimator is computed by three matrix–vector multiplications with A_2, for example). If $\eta_1 \leq \theta_3$ quit, otherwise continue to step 2.
2. Compute $A_4 = A_2^2$ and set $m = 5$, for which $p = 2$ is again the optimal value such that $2m \geq 2p(p-1)$. Now we have d_4 and can reuse the estimate of d_6, setting $\eta_2 = \max(d_4, d_6)$. If $\eta_2 \leq \theta_5$ quit, otherwise continue to step 3.
3. Compute $A_6 = A_4 A_2$. For $m \in \{7, 9\}$, $p = 3$ is the optimal value such that $2m \geq 2p(p-1)$. We compute $\eta_3 = \max(d_6, d_8)$, in which we estimate d_8 by applying the norm estimator to A_4^2. If $\eta_3 \leq \theta_7$ set $m = 7$, else if $\eta_3 \leq \theta_9$ set $m = 9$, else continue to step 4.
4. Set $m = 13$, for which either $p = 3$ or $p = 4$ is the optimal value such that $2m \geq 2p(p-1)$. The highest power of A that we compute to evaluate r_{13} by (3.5) is A^6, so we use the norm estimator to estimate d_{10} and set $\eta_4 = \max(d_8, d_{10})$. Choose the smallest $s \geq 0$ such that $2^{-s}\eta_5 \leq \theta_{13}$, where $\eta_5 = \min(\eta_3, \eta_4)$.

We introduce two more algorithmic refinements. First, we use $\theta_{13} = 4.25$ in place of the value $\theta_{13} = 5.37$ used in Algorithm 3.1 (see Table 3.1). The reason is that this produces a slightly better conditioned denominator polynomial q_{13} and our experiments show that in the context of our more liberal choice of s this is beneficial to the accuracy of the computed exponential. This refinement can lead to s exceeding $s_{\max}$, but only by 1, and in this case $\ell_m = 0$ as can be seen from (5.7). The second refinement is that for each putative m we compute the correction (5.8) *before* evaluating p_m and q_m and checking the inequality (5.6) and, if the correction is nonzero, we proceed to the next larger choice of m. This is simply another means for trying to avoid an inaccurate evaluation (or, put another way, wasted computation).

ALGORITHM 5.1 (new scaling and squaring algorithm for the matrix exponential). *This algorithm evaluates the matrix exponential $X = e^A$ of $A \in \mathbb{C}^{n\times n}$ by the scaling and squaring method. It is intended for IEEE double precision arithmetic. It uses the parameters θ_m given in Table* 3.1 *and the following functions:*

- `normest`*, which when invoked as* `normest`$(A_1, A_2, \ldots, A_k)$ *produces an estimate of $\|A_1 A_2 \ldots A_k\|_1$ and when invoked as* `normest`(A, m) *produces an estimate of $\|A^m\|_1$;*
- `ell`(A, m)*, which returns the integer* $\max(\lceil(\log_2(\alpha/u)/(2m)\rceil), 0)$*, where* $\alpha = |c_{2m+1}|$`normest`$(|A|, 2m+1)/\|A\|_1$.

1 $A_2 = A^2$
2 $d_6 = \texttt{normest}(A_2, 3)^{1/6}$, $\eta_1 = \max\bigl(\texttt{normest}(A_2, 2)^{1/4}, d_6\bigr)$
3 if $\eta_1 \le \theta_3$ and $\texttt{ell}(A, 3) = 0$
4 Evaluate $p_3(A)$ and $q_3(A)$ using (3.4).
5 if $\|p_3(|A|)^T e\|_\infty / \min(\|p_3\|_1, \|q_3\|_1) \le 10e^{\theta_3}$
6 Evaluate r_3 using (3.6), quit.
7 end
8 end
9 $A_4 = A_2^2$, $d_4 = \|A_4\|_1^{1/4}$
10 $\eta_2 = \max\bigl(d_4, d_6\bigr)$
11 if $\eta_2 \le \theta_5$ and $\texttt{ell}(A, 5) = 0$
12 Evaluate $p_5(A)$ and $q_5(A)$ using (3.4).
13 if $\|p_5(|A|)^T e\|_\infty / \min(\|p_5\|_1, \|q_5\|_1) \le 10e^{\theta_5}$
14 Evaluate r_5 using (3.6), quit.
15 end
16 end
17 $A_6 = A_2 A_4$, $d_6 = \|A_6\|_1^{1/6}$
18 $d_8 = \texttt{normest}(A_4, 2)^{1/8}$, $\eta_3 = \max(d_6, d_8)$
19 for $m = [7, 9]$
20 if $\eta_3 \le \theta_m$ and $\texttt{ell}(A, m) = 0$
21 Evaluate $p_m(A)$ and $q_m(A)$ using (3.4).
22 if $\|p_m(|A|)^T e\|_\infty / \min(\|p_m\|_1, \|q_m\|_1) \le 10e^{\theta_m}$
23 Evaluate r_m using (3.6), quit.
24 end
25 end
26 end
27 $\eta_4 = \max\bigl(d_8, \texttt{normest}(A_4, A_6)^{1/10}\bigr)$
28 $\eta_5 = \min(\eta_3, \eta_4)$
29 $s = \max\bigl(\lceil \log_2(\eta_5/\theta_{13}) \rceil, 0\bigr)$
30 $s = s + \texttt{ell}(2^{-s}A, 13)$
31 $A \leftarrow 2^{-s}A$, $A_2 \leftarrow 2^{-2s}A_2$, $A_4 \leftarrow 2^{-4s}A_4$, $A_6 \leftarrow 2^{-6s}A_6$
32 Evaluate $p_{13}(A)$ and $q_{13}(A)$ using (3.5).
33 if $\|p_{13}(|A|)^T e\|_\infty / \min(\|p_{13}\|_1, \|q_{13}\|_1) \le (10 + s_{\max})e^{\theta_{13}}$
34 Evaluate r_{13} using (3.6), quit.
35 else
36 $s_1 = s_{\max} - s$, $s = s_{\max}$
37 $A \leftarrow 2^{-s_1}A$, $A_2 \leftarrow 2^{-2s_1}A_2$, $A_4 \leftarrow 2^{-4s_1}A_4$, $A_6 \leftarrow 2^{-6s_1}A_6$
38 Evaluate r_{13} using (3.5) and (3.6).
39 end
40 if A is triangular
41 Invoke Code Fragment 2.1.
42 else
43 $X = r_{13}(A)^{2^s}$ by repeated squaring.
44 end

Cost: $(\pi_m + s)M + D$, where m is the degree of Padé approximant used and π_m (tabulated in [10, Table 2.2]) is the cost of evaluating p_m and q_m. If line 36 is executed then the cost is $(\pi_{13} + s + 3)M + D$. If any of the tests at lines 5, 13, and 22 fail then there is some wasted work in evaluating lower degree polynomials p_m and q_m that are not used.

Note that we have not included the bound (5.9) because it would require an extra matrix multiplication and the algorithm performs well in practice without the use of it. It is easy to check that if line 33 is reached then Algorithm 3.1 would choose $m = 13$, so at line 36 the algorithm is reverting to Algorithm 3.1.

6. Numerical experiments. We now compare Algorithm 3.1, as implemented in `expm`, and Algorithm 5.1 experimentally. We will use four sets of test matrices.

Set 1 Matrices from the literature on developing methods for e^A (including (1.2) with $b = 10^7$), mostly intended to be difficult for the scaling and squaring method. All are of dimension 10 or less.

Set 2 10×10 matrices from MATLAB (in particular, from the `gallery` function), and from the Matrix Computation Toolbox [7].

Set 3 The upper triangular Schur factors of the matrices from Set 2.

Set 4 Test matrices provided with EigTool [19], which are mainly discretizations of operators. The matrices are of variable dimension, which we have taken to be as close as possible to $n = 50$.

The tests in [10] and [11] used Sets 1 and 2 together.

Our tests were done in MATLAB 7.6 (R2008a). We compute normwise relative errors $\|\widehat{X} - e^A\|_F/\|e^A\|_F$ of the computed $\widehat{X}$ by approximating e^A by the result computed at 100 digit precision using the Symbolic Math Toolbox. For Sets 1 and 3, Algorithm 5.1 produces many errors of zero, but to facilitate the plots we replace a zero error for this algorithm by 10^{-18}.

For each set we present the results as four plots in a 2×2 grid; see Figures 6.1–6.4. The (1,1) plot shows the relative errors for the two algorithms, where the matrices are sorted by decreasing value of the condition number $\kappa_{\exp}(A)$ in (1.4), and $\kappa_{\exp}(A)u$ is shown as a solid line. The (1,2) plot shows the $\log_{10}$ of the ratio of relative errors, sorted in increasing order, and this same ordering is also used by the (2,1) and (2,2) plots. The (2,1) plot shows the values of s chosen by each method. The (2,2) plot shows the ratio of the costs of the algorithms, where cost is measured as described after the statement of each algorithm and we regard M and D as equal. Note that this measure of cost is appropriate only for $n \gg 10$, but since the choice of s and m depends only on $\|A\|_1$ for Algorithm 3.1 and on $\|A^k\|_1^{1/k}$ for certain k for Algorithm 5.1, these results are indicative of the relative costs for much larger matrices. We note that the cost ratio cannot exceed $8/7 \approx 1.14$, and can be greater than 1 only because of the differing values for θ_{13} for the two algorithms (see Table 3.1).

The main points to note are as follows.

1. Algorithm 5.1 did not revert to Algorithm 3.1 (on line 36) for any of the test matrices. Moreover, the tests at lines 5, 13, and 22 never failed to be satisfied. The correction (5.8) was nonzero at line 30 on 6, 10, 0, and 2 occasions on the four test sets, respectively. If we remove line 30 then there are 6 reversions in Test 1, 1 in Test 2, and none in Tests 3 and 4. The value of `ell` at lines 3, 11, and 20 was nonzero once for Test 1, 5 times for Test 2, and not at all for Tests 3 and 4.

2. For Set 1, Algorithm 5.1 has a cost up to about 5 times smaller than Algorithm 3.1 while achieving error barely any larger and sometimes orders of magnitude smaller. This is due to Algorithm 5.1 frequently choosing a smaller s.

3. For Set 2 there is no significant difference in the accuracy of the two algorithms. But in 22% of the cases Algorithm 5.1 is less expensive than Algorithm 3.1, by up to 17%, while in just two cases it is more expensive, by 12%.

4. For Set 3, Algorithm 5.1 is more accurate than Algorithm 3.1 in almost every case, often by orders of magnitude. This is mainly due to exploiting triangularity in the squaring phase.

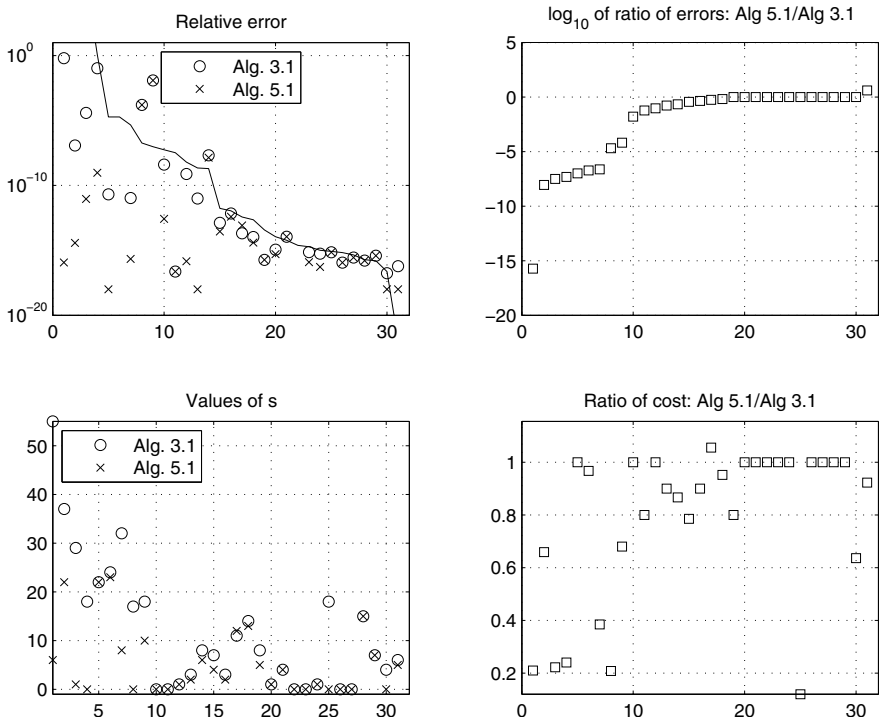

FIG. 6.1. *Results for test matrix Set* 1.

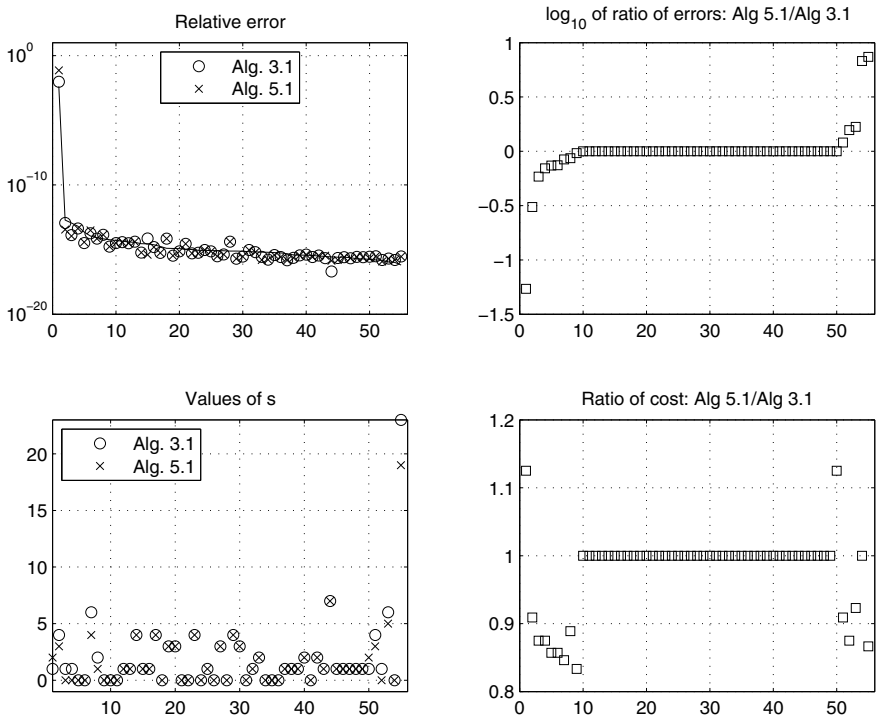

FIG. 6.2. *Results for test matrix Set* 2.

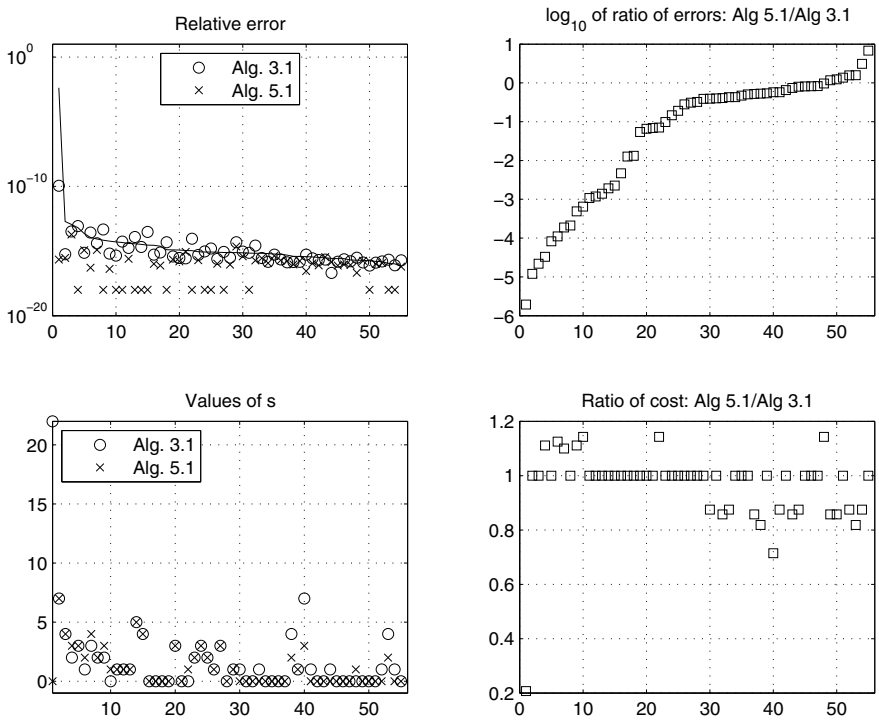

FIG. 6.3. *Results for test matrix Set* 3.

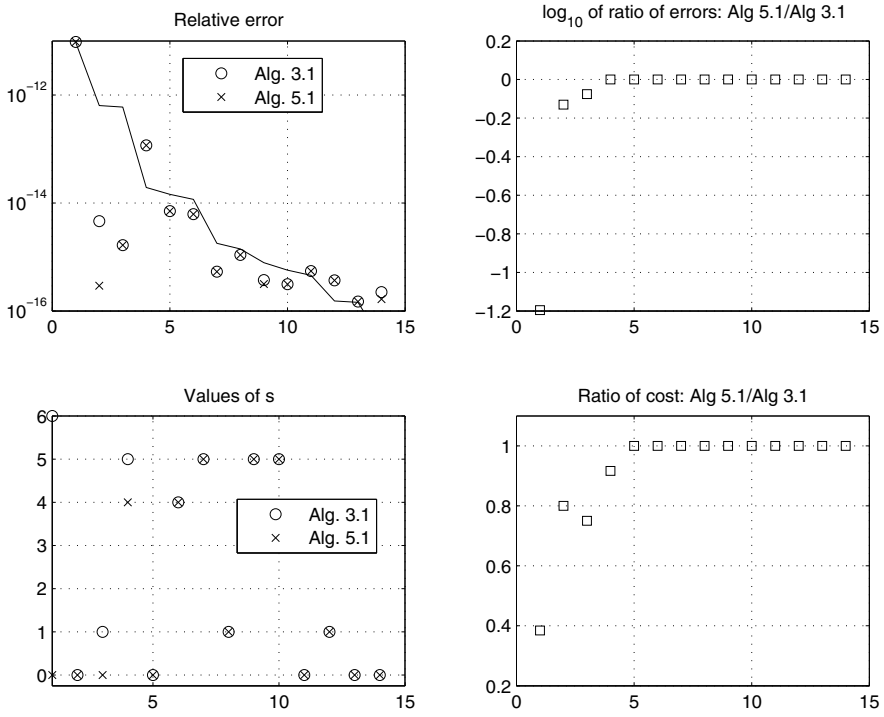

FIG. 6.4. *Results for test matrix Set* 4.

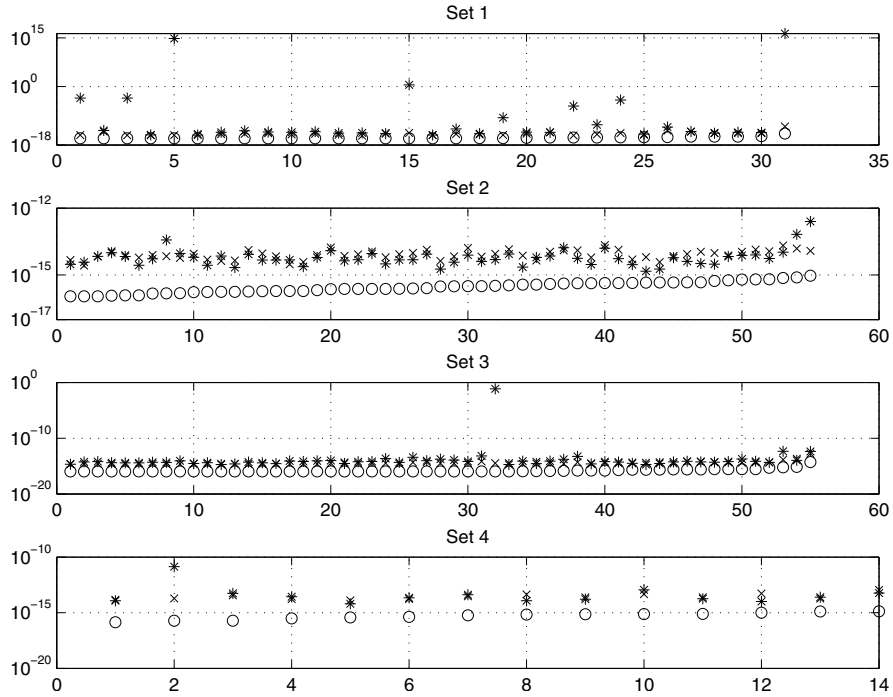

FIG. 6.5. *Quantities associated with the computed* $\widehat{r}_m \approx r_m(2^{-s}A)$ *for Algorithm* 5.1: *relative error in* $\widehat{r}_m$ ("$\circ$"), *a posteriori forward error bound* (5.9) ("$\times$"), *and* $n\kappa_1(q_m)u$ ("$*$")—*an approximate a priori bound for the error.*

5. For Set 4, Algorithm 5.1 is superior to Algorithm 3.1 in speed and accuracy for four of the matrices and performs equivalently to it for the rest.

6. Figure 6.5 provides information about the linear systems (3.6) that are solved to obtain $r_m(2^{-s}A)$. It shows the relative error $\|r_m - \widehat{r}_m\|_1/\|r_m\|_1$ along with the a posteriori bound (5.9) and the approximate a priori bound $n\kappa_1(q_m)u$ for this error. The results show that (a) the relative error is reasonably small in every case, (b) the system is sometimes solved to much better accuracy than the condition number $\kappa_1(q_m)$ would suggest (see Set 1), and (c) the a posteriori bound is a surprisingly good predictor of the actual error.

These experiments and our additional investigations lead us to conclude that no benefit is gained from using the test (5.6) to gauge whether the evaluation of p_m and q_m has been sufficiently accurate. Therefore we recommend the following simplification of Algorithm 5.1, which we emphasize performs identically to Algorithm 5.1 on the tests reported here.

ALGORITHM 6.1 (new scaling and squaring algorithm for the matrix exponential). *This algorithm is identical to Algorithm* 5.1 *except that lines* 5, 7, 13, 15, 22, 24, 33, *and* 35–39 *are removed.*

7. Conclusions. The propensity of the scaling and squaring method for overscaling has been known for over a decade. The new algorithm developed here, Algorithm 6.1, remedies this weakness in two different ways. First, it exploits triangularity, when present, to ensure that the diagonal and first off-diagonal are computed accurately during the squaring phase, benefitting all elements of the computed exponential. Second, it employs more refined truncation error bounds, based

on the quantities $\|A^k\|^{1/k}$, which can be much smaller than the bounds based on $\|A\|$ that were used previously. These refined bounds enable the algorithm to produce a result that often has one or both of the advantages of being more accurate and having a lower computational cost than the original algorithm from [10] (Algorithm 3.1).

A general conclusion of this work is that although matrix norms are sometimes described as a blunt instrument, they can extract much more information than might be thought about the behavior of a matrix power series, through the use of (estimates of) the norms of a small number of matrix powers.

We are currently adapting the ideas developed here to the algorithm derived in [1] for computing the Fréchet derivative of the matrix exponential and to algorithms for computing other transcendental matrix functions.

A remaining open question is to understand, and ideally improve, the numerical stability of the squaring phase of the scaling and squaring method. Our treatment of the triangular case is a first step in this direction.

REFERENCES

[1] A. H. Al-Mohy and N. J. Higham, *Computing the Fréchet derivative of the matrix exponential, with an application to condition number estimation*, SIAM J. Matrix Anal. Appl., 30 (2009), pp. 1639–1657.

[2] M. Bossa, E. Zacur, and S. Olmos, *Algorithms for computing the group exponential of diffeomorphisms: Performance evaluation*, in Proceedings of the Computer Vision and Pattern Recognition Workshops, 2008 (CVPRW '08), IEEE Computer Society, Washington, DC, 2008, pp. 1–8.

[3] 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.

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

[5] L. Dieci and A. Papini, *Conditioning of the exponential of a block triangular matrix*, Numer. Algorithms, 28 (2001), pp. 137–150.

[6] G. I. Hargreaves and N. J. Higham, *Efficient algorithms for the matrix cosine and sine*, Numer. Algorithms, 40 (2005), pp. 383–400.

[7] N. J. Higham, *The Matrix Computation Toolbox*, available online at http://www.ma.man.ac.uk/~higham/mctoolbox.

[8] N. J. Higham, *The Matrix Function Toolbox*, available online at http://www.ma.man.ac.uk/~higham/mftoolbox.

[9] N. J. Higham, *Accuracy and Stability of Numerical Algorithms*, 2nd ed., SIAM, Philadelphia, 2002.

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

[11] N. J. Higham, *Functions of Matrices: Theory and Computation*, SIAM, Philadelphia, 2008.

[12] N. J. Higham and F. Tisseur, *A block algorithm for matrix* 1*-norm estimation, with an application to* 1*-norm pseudospectra*, SIAM J. Matrix Anal. Appl., 21 (2000), pp. 1185–1201.

[13] R. A. Horn and C. R. Johnson, *Matrix Analysis*, Cambridge University Press, Cambridge, UK, 1985.

[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] L. Morai and A. F. Pacheco, *Algebraic approach to the radioactive decay equations*, Amer. J. Phys., 71 (2003), pp. 684–686.

[16] R. B. Sidje, *Expokit: A software package for computing matrix exponentials*, ACM Trans. Math. Software, 24 (1998), pp. 130–156.

[17] *The Control and Systems Library SLICOT*, available online at http://www.slicot.org/.

[18] C. F. Van Loan, *Computing integrals involving the matrix exponential*, IEEE Trans. Automat. Control, 23 (1978), pp. 395–404.

[19] T. G. Wright, *Eigtool*, 2002. Available online at http://www.comlab.ox.ac.uk/pseudospectra/eigtool/.

[20] D. Yuan and W. Kernan, *Explicit solutions for exit-only radioactive decay chains*, J. Appl. Phys., 101 (2007), 094907.