Improved Inverse Scaling and Squaring Algorithms for the Matrix Logarithm

Al-Mohy, Awad H. and Higham, Nicholas J.

2011

MIMS EPrint: **2011.83**

ISSN 1749-9097

IMPROVED INVERSE SCALING AND SQUARING ALGORITHMS FOR THE MATRIX LOGARITHM*

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

Abstract. A popular method for computing the matrix logarithm is the inverse scaling and squaring method, which essentially carries out the steps of the scaling and squaring method for the matrix exponential in reverse order. Here we make several improvements to the method, putting its development on a par with our recent version [*SIAM J. Matrix Anal. Appl.*, 31 (2009), pp. 970–989] of the scaling and squaring method for the exponential. In particular, we introduce backward error analysis to replace the previous forward error analysis; obtain backward error bounds in terms of the quantities $\|A^p\|^{1/p}$, for several small integer p, instead of $\|A\|$; and use special techniques to compute the argument of the Padé approximant more accurately. We derive one algorithm that employs a Schur decomposition, and thereby works with triangular matrices, and another that requires only matrix multiplications and the solution of multiple right-hand side linear systems. Numerical experiments show the new algorithms to be generally faster and more accurate than their existing counterparts and suggest that the Schur-based method is the method of choice for computing the matrix logarithm.

Key words. matrix logarithm, inverse scaling and squaring method, matrix exponential, backward error analysis, Padé approximation, matrix square root, MATLAB, `logm`

AMS subject classifications. 15A60, 65F30

1. Introduction.

A matrix $X \in \mathbb{C}^{n\times n}$ is a logarithm of $A \in \mathbb{C}^{n\times n}$ if $e^X = A$. Any nonsingular matrix has infinitely many logarithms, but the one that is most useful in practice is the principal logarithm, denoted by $\log(A)$. For $A \in \mathbb{C}^{n\times n}$ with no eigenvalues on $\mathbb{R}^-$, the closed negative real axis, the principal logarithm is the unique logarithm whose eigenvalues have imaginary parts lying in the interval $(-\pi, \pi)$ [16, Thm. 1.31]. Throughout this paper we assume that A has no eigenvalues on $\mathbb{R}^-$.

While there are many methods for computing the matrix exponential, relatively few methods exist for the matrix logarithm [16], [17]. The most widely used is the inverse scaling and squaring method, proposed by Kenney and Laub [20], which is an extension to matrices of the technique that Briggs used in the 17th century to compute his table of logarithms [11], [22]. The inverse scaling and squaring method first computes $A^{1/2^s}$, for an integer s large enough so that $A^{1/2^s}$ is close to the identity, then approximates $\log(A^{1/2^s})$ by $r_m(A^{1/2^s} - I)$, where r_m is an $[m/m]$ Padé approximant to the function $\log(1+x)$ (we call m the Padé degree), and finally forms the approximation $\log(A) \approx 2^s r_m(A^{1/2^s} - I)$. This approximation exploits the identity [16, Thm. 11.2]

$$\log(A) = 2^s \log(A^{1/2^s}).$$

The inverse scaling and squaring method can be applied to A directly, as in [6], [16, Alg. 11.10], without the use of transformations, or the Schur decomposition $A =$

*Version of April 16, 2012. This work was supported by European Research Council Advanced Grant MATFUN (267526).

†Department of Mathematics, King Khalid University, Abha, Saudi Arabia (aal-mohy@hotmail.com, http://www.maths.manchester.ac.uk/~almohy).

‡School of Mathematics, The University of Manchester, Manchester, M13 9PL, UK (higham@ma.man.ac.uk, http://www.ma.man.ac.uk/~higham). The work of this author was also supported by Engineering and Physical Sciences Research Council grant EP/E050441/1 (CICADA: Centre for Interdisciplinary Computational and Dynamical Analysis).

QTQ^* (Q unitary, T upper triangular) can be computed and the method used to compute $\log(T)$ and thence $\log(A) = Q\log(T)Q^*$ [8], [16, Alg. 11.9], [20]. Generally, it is preferable to employ the Schur form, since the resulting algorithm typically requires fewer flops and is more accurate [16, Sec. 11.5, 11.7].

Early inverse scaling and squaring algorithms used a fixed Padé degree, m, and a fixed condition $\|A^{1/2^s} - I\| \le \theta$ for determining how many square roots to take. Kenney and Laub [20] take $m = 8$ and $\theta = 0.25$, while Dieci, Morini, and Papini [8] take $m = 9$ and $\theta = 0.35$, aiming for double precision accuracy in each case. The algorithm of Cheng, Higham, Kenney, and Laub [6] determines m at run time in a way that aims to minimize the overall cost subject to achieving a user-specified accuracy, making use of a forward error bound of Kenney and Laub [21]. Higham [16, Algs 11.9, 11.10] takes a similar approach but precomputes the necessary parameters and derives algorithms for both full and triangular matrices.

In this work the performance of the inverse scaling and squaring method is considerably improved by

- introducing new backward error analysis for Padé approximation of the matrix logarithm upon which to base the choice of m and s,
- obtaining sharp bounds for the backward error in terms of the quantities $\|A^p\|^{1/p}$ ($p = 2, 3, \ldots$), which can be substantially smaller than $\|A\|$ for nonnormal A,
- accurately computing the diagonal and first superdiagonal of $T^{1/2^s} - I$ in a way that avoids cancellation, and also computing $A^{1/2^s} - I$ more accurately for full A,
- replacing the elements on the diagonal and first superdiagonal of the approximation $2^s r_m(T^{1/2^s} - I)$ by quantities computed accurately from explicit formulae.

Incorporating these features brings the inverse scaling and squaring method into line with recent improvements to the scaling and squaring method for the matrix exponential [2], although the details are quite different than those for the exponential.

We will use the partial fraction form of the $[m/m]$ Padé approximant $r_m(x)$ to $\log(1 + x)$, given by [14]

$$r_m(x) = \sum_{j=1}^{m} \frac{\alpha_j^{(m)} x}{1 + \beta_j^{(m)} x}, \tag{1.1}$$

where the $\alpha_j^{(m)} \in (0, 1)$ and $\beta_j^{(m)} \in (0, 1)$ are the weights and the nodes, respectively, of the m-point Gauss–Legendre quadrature rule on $[0, 1]$. Several different ways are available to evaluate r_m at a matrix argument, but the partial fraction representation (1.1) was found by Higham [14] to provide the best balance between accuracy and efficiency.

In the next section we develop our backward error analysis for the Padé approximant. In Section 3 we explain the danger of subtractive cancellation in the inverse scaling and squaring method. A Schur decomposition-based algorithm for computing the logarithm is designed in Section 4, and a transformation-free algorithm is developed in Section 5. Numerical experiments demonstrating significant improvements in accuracy and efficiency over existing algorithms are reported in Section 6 and conclusions are given in Section 7.

2. Backward error analysis. Previous work on Padé approximation of the matrix logarithm has focused on the use of forward error bounds. The bound

$$\|r_m(X) - \log(I + X)\| \le \big|r_m(-\|X\|) - \log(1 - \|X\|)\big| =: f_m(\|X\|) \tag{2.1}$$

of Kenney and Laub [21], valid for $\|X\| < 1$ and any subordinate matrix norm, has been used by several authors to select the Padé degree m [4], [5], [6], [16, Chap. 11], [20]. It is generally preferable to work with backward error bounds, as these permit an interpretation that is independent of the conditioning of the problem. In this section we derive an explicit expression for the backward error of a Padé approximant and a bound for its norm. Note that backward errors here are with respect to truncation errors; rounding errors are not considered in this section.

Let $\rho(A)$ denote the spectral radius of $A \in \mathbb{C}^{n\times n}$. We will need the bound on $\rho(r_m(A))$ given in the following lemma.

LEMMA 2.1. *Let $A \in \mathbb{C}^{n\times n}$ have no eigenvalues on $\mathbb{R}^-$ and let $\rho(A) < 1$. Then for r_m in* (1.1) *we have*

$$\rho(r_m(A)) \le \sum_{j=1}^{m} \frac{\alpha_j^{(m)}\rho(A)}{1 - \beta_j^{(m)}\rho(A)}. \tag{2.2}$$

Proof. The eigenvalues of $r_m(A)$ are of the form

$$\widetilde{\lambda} = \sum_{j=1}^{m} \frac{\alpha_j^{(m)}\lambda}{1 + \beta_j^{(m)}\lambda},$$

where λ is an eigenvalue of A. Hence

$$|\widetilde{\lambda}| \le \sum_{j=1}^{m} \frac{\alpha_j^{(m)}|\lambda|}{1 - \beta_j^{(m)}|\lambda|}.$$

The functions $f_j(x) = \alpha_j^{(m)}x/(1 - \beta_j^{(m)}x)$ are increasing on $(0, \rho(A)]$ since $f_j'(x) = \alpha_j^{(m)}/(1 - \beta_j^{(m)}x)^2 > 0$. The maximal value of the bound is therefore attained when $|\lambda| = \rho(A)$, and the result follows. □

We now define the matrix function $h_{2m+1} : \mathbb{C}^{n\times n} \to \mathbb{C}^{n\times n}$ by $h_{2m+1}(X) = e^{r_m(X)} - X - I$. It is clear from (1.1) that $r_m(x)$ has no poles in the disc $\{\, x : |x| \le 1 \,\}$ and so it has a power series expansion there. Since $r_m(x) = \log(1 + x) + O(x^{2m+1})$ it follows that h_{2m+1} has a power series expansion of the form

$$h_{2m+1}(X) = \sum_{k=2m+1}^{\infty} c_k X^k. \tag{2.3}$$

We will assume that $\rho(r_m(X)) < \pi$, which ensures that $\log(e^{r_m(X)}) = r_m(X)$ [16, Prob. 1.39]. This turns out not to be a restriction, as we find using Lemma 2.1 that if $\rho(X) < 0.91$ then $\rho(r_m(X)) < \pi$, $m = 1{:}100$, and such a restriction on $\rho(X)$ is harmless provided that $m \le 16$, as Table 2.1 (explained below) shows. By rearranging the definition of h_{2m+1} and taking logarithms we obtain

$$r_m(X) = \log(I + X + h_{2m+1}(X)) =: \log(I + X + \Delta X). \tag{2.4}$$

Hence $\mathit{\Delta} X = h_{2m+1}(X)$ is the backward error resulting from the approximation of $\log(I+X)$ by $r_m(X)$. We bound the backward error by applying [2, Thm. 4.2(a)] to (2.3), to obtain

$$\|\mathit{\Delta} X\| \le \sum_{k=2m+1}^{\infty} |c_k| \alpha_p(X)^k, \tag{2.5}$$

where

$$\alpha_p(X) = \max\bigl(d_p, d_{p+1}\bigr), \qquad d_p = \|X^p\|^{1/p}, \tag{2.6}$$

and the integer $p \ge 1$ must satisfy

$$2m+1 \ge p(p-1). \tag{2.7}$$

Here, the norm is any consistent matrix norm. Since $\alpha_p(X) \le \|X\|$, and indeed $\alpha_p(X)$ can be substantially smaller than $\|X\|$ for nonnormal X [2], the use of the $\alpha_p(X)$ in place of $\|X\|$ leads to a bound sharper than the more obvious one involving terms $|c_k|\|X\|^k$.

We summarize our findings in the following result.

THEOREM 2.2. *If $X \in \mathbb{C}^{n\times n}$ satisfies $\rho(r_m(X)) < \pi$ then $r_m(X) = \log(I + X + \mathit{\Delta} X)$, where, for any $p \ge 1$ satisfying* (2.7),

$$\frac{\|\mathit{\Delta} X\|}{\|X\|} \le \sum_{k=2m+1}^{\infty} |c_k| \alpha_p(X)^{k-1}. \tag{2.8}$$

Let $\theta_m = \max\{\,\theta : \sum_{k=2m+1}^{\infty} |c_k|\theta^{k-1} \le u\,\}$, where $u = 2^{-53} \approx 1.1\times 10^{-16}$ is the unit roundoff for IEEE double precision arithmetic. We used the Symbolic Math Toolbox to evaluate θ_m, $m = 1{:}\,16$, by summing the first 250 terms of the series in 250 decimal digit arithmetic. The values of θ_m are listed to three significant figures in Table 2.1. Thus, if X satisfies $\alpha_p(X) \le \theta_m$ for p and m satisfying (2.7) then the approximation of $\log(I+X)$ by the Padé approximant $r_m(X)$ produces a backward error $\mathit{\Delta} X$ such that $\|\mathit{\Delta} X\| \le u\|X\|$. Our strategy will therefore be to choose the parameters s and m so that

$$\min\bigl\{\,\alpha_p(A^{1/2^s} - I) : p \text{ satisfies } p(p-1) \le 2m+1\,\bigr\} \le \theta_m, \tag{2.9}$$

and in such a way that the computational cost is minimized.

It is important to note that this choice of parameters ensures that the matrix $I + \beta_j^{(m)} X$ arising in (1.1), with $X = A^{1/2^s} - I$, is nonsingular. Indeed, for p achieving the minimum in (2.9),

$$\rho(\beta_j^{(m)} X) \le \alpha_p(\beta_j^{(m)} X) = \beta_j^{(m)} \alpha_p(X) \le \beta_j^{(m)} \theta_m < \theta_m < 1.$$

However, the bound

$$\max_j \kappa(I + \beta_j^{(m)} X) \le \max_j (1 + \beta_j^{(m)} \|X\|)/(1 - \beta_j^{(m)} \|X\|), \tag{2.10}$$

where $\kappa(A) = \|A^{-1}\|\|A\|$, which is used in [14] to show that the evaluation of $r_m(X)$ in floating point arithmetic will be accurate, is no longer valid as $\|X\|$ is not bounded

Table 2.1
Value of θ_m for selected m.

m	1	2	3	4	5	6	7	8
θ_m	1.59e-5	2.31e-3	1.94e-2	6.21e-2	1.28e-1	2.06e-1	2.88e-1	3.67e-1

m	9	10	11	12	13	14	15	16
θ_m	4.39e-1	5.03e-1	5.60e-1	6.09e-1	6.52e-1	6.89e-1	7.21e-1	7.49e-1

(and certainly not bounded by 1). One way around this is to choose $\epsilon > 0$ so that $\alpha_p(X) + \epsilon < 1$ and recall that there exists a norm $\|\cdot\|_\epsilon$ such that $\|X\|_\epsilon \le \rho(X) + \epsilon \le \alpha_p(X) + \epsilon < 1$. Then (2.10) holds in this norm and provides a satisfactory bound, but the norm will be poorly scaled if ϵ is small, so the practical relevance of this bound is unclear in general. When X is triangular, the relevant error bounds are more refined, involving componentwise condition numbers [15, Chap. 8] and are less sensitive to large-normed X. Some loss of accuracy when $r_m(X)$ is computed in floating point arithmetic does not necessarily degrade the accuracy of the computed logarithm, which may nevertheless reflect the conditioning of the problem. But, as with the scaling and squaring method for the matrix exponential, relating the effect of rounding errors incurred within the algorithm to the condition number of $\log(A)$ is an open problem.

Finally, it is interesting to compare the parameters θ_m in Table 2.1 with the corresponding parameters θ'_m in [16, Sec. 11.5], which are derived using the forward error bound (2.1) by requiring $f_m(\theta) \le u$. We have $\theta_m \ge \theta'_m$, $m = 1\colon 16$; in particular, $\theta_8 = 0.367 > 0.340 = \theta'_8$ and $\theta_{16} = 0.749 > 0.724 = \theta'_{16}$. Thus basing the algorithm on backward error bounds rather than forward error bounds leads to a slightly more efficient algorithm (and our use of $\alpha_p(X)$ instead of $\|X\|$ brings further savings that are potentially much larger).

3. Avoiding cancellation.

3. Avoiding cancellation. The inverse scaling and squaring method has a weakness: subtractive cancellation can occur in forming the matrix $A^{1/2^s} - I$, thus bringing into prominence errors committed in computing the square roots. Although we will not require $\|A^{1/2^s} - I\|$ to be orders of magnitude smaller than 1, if A is (for example) triangular with diagonal elements of widely varying size then for some elements there can be significant cancellation.

For scalar $a \in \mathbb{C}$, Al-Mohy [1] writes

$$a^{1/2^s} - 1 = \frac{a-1}{\prod_{i=1}^{s}(1 + a^{1/2^i})} \tag{3.1}$$

and shows that this formula (applied to $a^{1/2}$ if a lies in the left half-plane) avoids subtractive cancellation. We will use this idea in Section 4 and an extension of it for matrices in Section 5.

For triangular matrices A we also employ another approach, similar to that we used for the matrix exponential in [2]. Instead of approximating the diagonal and first superdiagonal of $\log(A)$ by the corresponding elements of $2^s r_m(A^{1/2^s} - I)$, we compute these elements directly using explicit formulas. For the diagonal the formula is simply $\log(a_{ii})$. For the elements on the (first) superdiagonal, which are divided differences [16, p. 84], we use a rather complicated formula given in [16, (11.28)] that is immune to cancellation.

4. A Schur-based algorithm.

4. A Schur-based algorithm. We now use the backward error analysis of Section 2 to design an algorithm that begins with a transformation to Schur form, $A = QTQ^*$, and thereafter works with the triangular matrix T.

The s square roots in $T^{1/2^s}$ are computed by the algorithm of Björck and Hammarling [3], [16, Alg. 6.3]. One square root costs $n^3/3$ flops and the evaluation of $r_m(T^{1/2^s} - I)$ in (1.1) costs $mn^3/3$ flops. The parameters s and m are chosen to minimize the total cost of $(s+m)n^3/3$ flops, using the following reasoning.

For any putative s and m, which must satisfy $\alpha_p(T^{1/2^s} - I) \le \theta_m$ for some p by (2.9), the computational cost can be reduced if taking an extra square root leads to a reduction of the degree of Padé approximant by more than one, that is, if $\alpha_p(T^{1/2^{s+1}} - I) \le \theta_{m-2}$. Irrespective of the triangularity of T, and since $(T^{1/2^{s+1}} - I)(T^{1/2^{s+1}} + I) = T^{1/2^s} - I$, we have asymptotically that

$$\alpha_p(T^{1/2^{s+1}} - I) \approx \frac{1}{2}\alpha_p(T^{1/2^s} - I), \tag{4.1}$$

for suitably large s, and so we will deem that an extra square root is worth taking if $\frac{1}{2}\alpha_p(T^{1/2^s} - I) \le \theta_{m-2}$. Table 2.1 shows that the inequality $\frac{1}{2}\theta_m \le \theta_{m-2}$ holds for $m > 7$. Thus we should take s at least as large as the first value for which $\alpha_p(T^{1/2^s} - I) \le \theta_7$. We use the 1-norm, and instead of computing the quantities $\alpha_p(T^{1/2^s} - I)$ we estimate them, by using the block 1-norm estimation algorithm of Higham and Tisseur [19] to approximate $d_p^p = \|(T^{1/2^s} - I)^p\|_1$; the cost of estimating d_p is just $O(n^2)$ flops, given $T^{1/2^s}$. We can save some work by noting that, on writing $T = D + F$ with $D = \mathrm{diag}(T)$,

$$\rho(D - I) = \rho(T - I) \le \alpha_p(T - I),$$

and so there is no need to estimate $\alpha_p(T^{1/2^s} - I)$ until $\rho(D^{1/2^s} - I) \le \theta_7$, as the latter inequality is necessary for $\alpha_p(T^{1/2^s} - I) \le \theta_7$; denote the smallest such s by s_0.

The inequality $2m+1 \ge p(p-1)$ holds for $m \ge 1$ for $p = 1$, for $m \ge 3$ for $p = 2, 3$, and for $m \ge 6$ for $p = 4$, and by the analysis above m does not exceed 7. From (2.6), we have $\alpha_3(X) \le \alpha_2(X)$ for any X, so after obtaining s_0, computing $T \leftarrow T^{1/2^{s_0}}$, and checking if $m = 1$ or $m = 2$ can be used, we check whether $\alpha_3(T - I) \le \theta_7$. Suppose, first, that $\alpha_3(T - I) \in (\theta_6, \theta_7]$. We have $\frac{1}{2}\alpha_3(T - I) \in (\frac{1}{2}\theta_6, \theta_5] \cup (\theta_5, \frac{1}{2}\theta_7]$. Thus if $\frac{1}{2}\alpha_3(T - I) \le \theta_5$ the algorithm predicts that one more square root should be taken to reduce the cost. Since it is not guaranteed that $\alpha_3(T^{1/2} - I) \le \theta_5$, as this depends on the approximation (4.1), we will allow at most two extra square roots to be taken to avoid unnecessary square roots. If $\frac{1}{2}\alpha_3(T - I) \in (\theta_5, \frac{1}{2}\theta_7]$ we do not immediately choose $m = 7$, as by considering α_4 we may be able to take $m = 6$. Consider now the case where $\alpha_3(T - I) \le \theta_6$. Since $\theta_m \notin (\frac{1}{2}\theta_{m+1}, \frac{1}{2}\theta_{m+2}]$ for $m = 3, 4$, an extra square root is not necessary; we find the smallest $m \in \{3, 4, 5, 6\}$ such that $\alpha_3(T - I) \le \theta_m$ and evaluate r_m. Finally, we test whether $\min(\alpha_3(T-I), \alpha_4(T-I)) \le \theta_m$ for $m = 6, 7$, which provides our last chance to avoid another square root. If none of these tests is satisfied we repeat the process with $T \leftarrow T^{1/2}$.

As pointed out in Section 3, the subtraction $T - I$ can suffer cancellation in the diagonal elements. Thus before evaluating r_m at $T - I$ we replace the diagonal elements by more accurate computed quantities obtained by applying (3.1) (more precisely, [1, Alg. 2]) to the diagonal entries of the original T. We also replace the first superdiagonal of $T-I$ by quantities computed accurately from an explicit formula [18, (5.6)], applied also to the original T.

We are now in a position to state our algorithm.

ALGORITHM 4.1 (inverse scaling and squaring algorithm with Schur decomposition). *Given $A \in \mathbb{C}^{n\times n}$ with no eigenvalues on $\mathbb{R}^-$ this algorithm computes $X = \log(A)$ via the Schur decomposition and inverse scaling and squaring. It uses the constants θ_m listed in Table* 2.1 *and the function* `normest`(A, m)*, which produces an estimate of $\|A^m\|_1$. The algorithm is intended for IEEE double precision arithmetic.*

```
1  Compute a (complex) Schur decomposition $A = QTQ^*$.
2  $T_0 = T$
3  Find $s_0$, the smallest $s$ such that $\rho(D^{1/2^s} - I) \le \theta_7$, where $D = \mathrm{diag}(T)$.
4  for $i = 1\colon s_0$
5      $T \leftarrow T^{1/2}$ using [16, Alg. 6.3].
6  end
7  $s = s_0$, $k = 0$
8  $d_2 = \mathtt{normest}(T - I, 2)^{1/2}$, $d_3 = \mathtt{normest}(T - I, 3)^{1/3}$
9  $\alpha_2 = \max(d_2, d_3)$
10 for $i = 1\colon 2$
11     if $\alpha_2 \le \theta_i$, $m = i$, goto line 35, end
12 end
13 while true
14     if $s > s_0$, $d_3 = \mathtt{normest}(T - I, 3)^{1/3}$, end
15     $d_4 = \mathtt{normest}(T - I, 4)^{1/4}$, $\alpha_3 = \max(d_3, d_4)$
16     if $\alpha_3 \le \theta_7$
17        $j_1 = \min\{\, i\colon \alpha_3 \le \theta_i,\ i = 3\colon 7 \,\}$
18        if $j_1 \le 6$
19           $m = j_1$, goto line 35
20        else
21          if $\frac{1}{2}\alpha_3 \le \theta_5$ and $k < 2$
               % Extra square root predicted worthwhile.
22             $k = k + 1$
23             goto line 32
24          end
25        end
26     end
27     $d_5 = \mathtt{normest}(T - I, 5)^{1/5}$, $\alpha_4 = \max(d_4, d_5)$
28     $\eta = \min(\alpha_3, \alpha_4)$    % Min taken as $\theta_6 < \alpha_3 < \alpha_4 \le \theta_7$ is possible.
29     for $i = 6\colon 7$
30        if $\eta \le \theta_i$, $m = i$, goto line 35, end
31     end
32     $T \leftarrow T^{1/2}$ using [16, Alg. 6.3]
33     $s = s + 1$
34 end
35 Replace $\mathrm{diag}(T - I)$ by $\mathrm{diag}(T_0)^{1/2^s} - 1$ using [1, Alg. 2] and recompute
   the first superdiagonal of $T$ using [18, (5.6)] applied to $T_0$.
36 Evaluate $U = 2^s r_m(T - I)$, using the partial fraction expansion (1.1).
37 Replace $\mathrm{diag}(U)$ by $\log(\mathrm{diag}(T_0))$ and the elements of the first
   superdiagonal of $U$ by those given by [16, (11.28)] with $T = T_0$.
38 $X = QUQ^*$
```

Cost: $25n^3$ flops for the Schur decomposition plus $(s + m)n^3/3$ flops for U and $3n^3$ flops to form X.

5. Transformation-free algorithm.

5. Transformation-free algorithm. Now we develop an algorithm that works on the original matrix A without the use of a Schur decomposition. This algorithm requires only matrix multiplications and the solution of multiple right-hand side linear systems, so is potentially more efficient on a parallel computer; it may also be attractive for higher precision computation (with a recomputation of the θ_i for the relevant value of u).

To compute matrix square roots we use the scaled product form of the Denman–Beavers (DB) iteration [6], [16, (6.29)]. As in the previous section we base the choice of the algorithm parameters on the backward error bound (2.8).

For improved accuracy we compute $X_s = A^{1/2^s} - I$ by solving the equation

$$X_s \prod_{i=1}^{s} \Bigl(I + A^{1/2^i} \Bigr) = A - I, \tag{5.1}$$

which generalizes (3.1). We actually apply the formula to $A^{1/2}$, as an initial square root moves the spectrum to the right half-plane; in the scalar case this ensures that no subtractive cancellation occurs, as shown by Al-Mohy [1]. If A is symmetric positive definite then so are all the roots $A^{1/2^i}$ in (5.1) and hence all their diagonal elements are positive and there is no cancellation in the sums $I + A^{1/2^i}$. There is possible cancellation in forming the right-hand side $A - I$ if some a_{ii} is close to 1, but since any such subtractions are done exactly [15, Thm. 2.5] and involve only original data they are harmless. A complete justification for (5.1) for general matrices is lacking, so we will test its effectiveness in the numerical experiments of Section 6.

The formula (5.1) is implemented as follows.

ALGORITHM 5.1. *For $A \in \mathbb{C}^{n\times n}$ with no eigenvalues on $\mathbb{R}^-$ this algorithm computes $X = A^{1/2^s} - I$ using the formula* (5.1) *applied to $A^{1/2}$. All matrix square roots are computed using the scaled product DB iteration* [16, (6.29)].

```
1  $A \leftarrow A^{1/2}$
2  $Z_0 = A - I$
3  if $s = 1$, $X = Z_0$, quit, end
4  $A \leftarrow A^{1/2}$
5  $P = I + A$
6  for $i = 1\colon s-2$
7      $A \leftarrow A^{1/2}$, $P \leftarrow P(I + A)$
8  end
9  Solve $XP = Z_0$ for $X$.
```

The extra computational cost of Algorithm 5.1 compared with the direct evaluation of $A^{1/2^s} - I$ is $s-2$ matrix multiplications and one multiple-right-hand side solve. This is a small overhead compared with the cost of computing the square roots.

To illustrate the numerical accuracy of this algorithm, consider the matrix

$$A = \begin{bmatrix} \cos\lambda & -\sin\lambda \\ \sin\lambda & \cos\lambda \end{bmatrix} \qquad \lambda \neq (2k+1)\pi,$$

which has the principal logarithm

$$\log(A) = \begin{bmatrix} 0 & -\lambda \\ \lambda & 0 \end{bmatrix}.$$

Suppose that for $\lambda = 1$ we approximate $\log(A)$ via Briggs' formula

$$\log(A) \approx 2^s(A^{1/2^s} - I),$$

which uses the first order approximation of the logarithm function $\log(1+x) \approx x$. The backward error bound (2.8) is trivially adapted for this approximation and shows that we need to choose s so that $\|A^{1/2^s} - I\| \le 2.3 \times 10^{-16}$ in order to achieve a backward error no larger than the unit roundoff. We find that the smallest s is 53. The computed approximations to $2^s(A^{1/2^s} - I)$ from Algorithm 5.1 and by direct evaluation using the scaled product DB iteration are, respectively, to two significant figures,

$$\begin{bmatrix} -1.7 \times 10^{-16} & -1.0 \\ 1.0 & -1.6 \times 10^{-16} \end{bmatrix}, \qquad \begin{bmatrix} 0 & -1.0 \\ 1.0 & -1.0 \end{bmatrix}.$$

Algorithm 5.1 gives a result as accurate as we can expect, but the direct evaluation yields a completely inaccurate (2,2) element.

The following algorithm incorporates Algorithm 5.1. We state the algorithm and then comment on its underlying logic.

Algorithm 5.2. *Given $A \in \mathbb{C}^{n\times n}$ with no eigenvalues on $\mathbb{R}^-$ this algorithm computes $X = \log(A)$ via inverse scaling and squaring. It uses the constants θ_m listed in Table 2.1 and the function* `normest`(A, m)*, which produces an estimate of $\|A^m\|_1$. This algorithm is intended for IEEE double precision arithmetic.*

```
1   $s = 0$, $k = 0$, $\mathrm{it}_0 = 5$
2   $d_2 = \mathtt{normest}(A - I, 2)^{1/2}$, $d_3 = \mathtt{normest}(A - I, 3)^{1/3}$
3   $\alpha_2 = \max(d_2, d_3)$
4   for $i = 1{:}\,2$
5       if $\alpha_2 \le \theta_i$, $m = i$, goto line 52, end
6   end
7   while true
8       if $s > 0$, $d_3 = \mathtt{normest}(A - I, 3)^{1/3}$, end
9       $d_4 = \mathtt{normest}(A - I, 4)^{1/4}$, $\alpha_3 = \max(d_3, d_4)$   % $p = 3$, $m \ge 3$ (see (2.7))
10      if $\alpha_3 \le \theta_{16}$
11          $j_1 = \min\{\, i{:}\, \alpha_3 \le \theta_i, i = 3{:}\,16 \,\}$
12          $j_2 = \min\{\, i{:}\, \frac{1}{2}\alpha_3 \le \theta_i, i = 3{:}\,16 \,\}$
13          if $2(j_1 - j_2)/3 < \mathrm{it}_s$ and $j_1 \le 6$
14              $m = j_1$, goto line 52
15          else
16              if $2(j_1 - j_2)/3 \ge \mathrm{it}_s$ and $k < 2$, $k = k + 1$, goto line 46, end
17          end
18      end
19      $d_5 = \mathtt{normest}(A - I, 5)^{1/5}$
20      $\alpha_4 = \max(d_4, d_5)$, $\eta_4 = \min(\alpha_3, \alpha_4)$   % $p = 4$, $m \ge 6$
21      if $\eta_4 \le \theta_{16}$
22          $j_1 = \min\{\, i{:}\, \eta_4 \le \theta_i, i = 6{:}\,16 \,\}$
23          $j_2 = \min\{\, i{:}\, \frac{1}{2}\eta_4 \le \theta_i, i = 6{:}\,16 \,\}$
24          if $2(j_1 - j_2)/3 < \mathrm{it}_s$ and $j_1 \le 10$
25              $m = j_1$, goto line 52
26          else
27              if $2(j_1 - j_2)/3 \ge \mathrm{it}_s$ and $k < 2$, $k = k + 1$, goto line 46, end
28          end
29      end
30      $d_6 = \mathtt{normest}(A - I, 6)^{1/6}$
31      $\alpha_5 = \max(d_5, d_6)$, $\eta_5 = \min(\eta_4, \alpha_5)$   % $p = 5$, $m \ge 10$
32      if $\eta_5 \le \theta_{16}$
```

```
33          $j_1 = \min\{\, i\colon \eta_5 \le \theta_i, i = 10\colon 16 \,\}$
34          $j_2 = \min\{\, i\colon \frac{1}{2}\eta_5 \le \theta_i, i = 10\colon 16 \,\}$
35          if $2(j_1 - j_2)/3 < \mathrm{it}_s$ and $j_1 \le 15$
36             $m = j_1$, goto line 52
37          else
38             if $2(j_1 - j_2)/3 \ge \mathrm{it}_s$ and $k < 2$, $k = k + 1$, goto line 46, end
39          end
40       end
41       $d_7 = \mathtt{normest}(A - I, 7)^{1/7}$
42       $\alpha_6 = \max(d_6, d_7)$, $\eta_6 = \min(\eta_5, \alpha_6)$     % $p = 6$, $m \ge 15$
43       for $i = 15\colon 16$
44          if $\eta_6 \le \theta_i$, $m = i$, goto line 52, end
45       end
46       $A \leftarrow A^{1/2}$, using the scaled product DB iteration [16, (6.29)];
         let $\mathrm{it}_{s+1}$ be the number of iterations required.
47       $s = s + 1$
48       if $s = 1$, $Z_0 = A - I$, end
49       if $s = 2$, $P = I + A$, end
50       if $s > 2$, $P \leftarrow P(I + A)$, end
51    end
52    if $s < 2$
53       $Y = A - I$
54    else
55       Solve $YP = Z_0$ for $Y$.
56    end
57    Evaluate $X = 2^s r_m(Y)$, using the partial fraction expansion (1.1).
58    end
```

Cost: $\left(\sum_{i=1}^{s} \mathrm{it}_i\right)(4n^3) + 8mn^3/3$ flops, plus an additional $2(s - 2/3)n^3$ flops if $s \ge 2$.

The logic of the algorithm is similar to that of Algorithm 4.1; the limitation $m \le 16$ and the test $2(j_1 - j_2)/3 < \mathrm{it}_s$ on line 13 follow from consideration of the θ_m in Table 2.1 and the cost of the DB iteration, and are explained in [16, Sec. 11.5.2]. The algorithm begins by taking repeated square roots of A until $\alpha_p(A - I) \le \theta_{16}$ for some $p \in \{3, 4, 5, 6\}$. If the condition $2(j_1 - j_2)/3 \ge \mathrm{it}_s$ is satisfied then it is predicted to be worth taking extra square roots, but a limit of two extra square roots is enforced. When $j_1 > 6$, line 13 forces the algorithm to evaluate α_p for $p = 4$, and possibly $p = 5$ and $p = 6$, in an attempt to use a smaller m; note that the sequence $\{d_p\}$ is generally (although not always) decreasing. However, each phase of the algorithm is subject to the constraint $2m + 1 \ge p(p - 1)$ in (2.7), as noted in comments within the algorithm. Importantly, it can be shown from the θ_m values in Table 2.1 and the definition of j_1 and j_2 that $j_1 - j_2$ is nonincreasing as the algorithm proceeds, which avoids the algorithm taking unnecessary square roots resulting from failure of the condition $2(j_1 - j_2)/3 < \mathrm{it}_s$ for one value of p when it had been satisfied for a previous value of p.

Lines 8–40 of Algorithm 5.2 can be replaced by the following equivalent, but less easily understandable, code.

```
1    if $s > 0$, $d_3 = \mathtt{normest}(A - I, 3)^{1/3}$, end, $\alpha_2 = \infty$
2    for $p = 3\colon 5$
3       $d_{p+1} = \mathtt{normest}(A - I, p + 1)^{1/(p+1)}$
4       $\alpha_p = \max(d_p, d_{p+1})$, $\eta_p = \min(\alpha_{p-1}, \alpha_p)$
```

```
5       if $\eta_p \le \theta_{16}$
6          $j_1 = \min\{\, i\colon \eta_p \le \theta_i,\ i = \lceil \frac{p(p-1)-1}{2} \rceil : 16 \,\}$
7          $j_2 = \min\{\, i\colon \frac{1}{2}\eta_p \le \theta_i,\ i = \lceil \frac{p(p-1)-1}{2} \rceil : 16 \,\}$
8          if $2(j_1 - j_2)/3 < \mathrm{it}_s$ and $j_1 \le \lceil \frac{p(p+1)-1}{2} \rceil$
9             $m = j_1$, goto line 52
10         else
11            if $2(j_1 - j_2)/3 \ge \mathrm{it}_s$ and $k < 2$, $k = k + 1$, goto line 46, end
12         end
13      end
14   end
```

6. Numerical experiments. We now compare our new algorithms with existing algorithms empirically. Our experiments were carried out in MATLAB R2011b, and for most of the experiments we use the same set of 67 (mostly 10×10) test matrices as in [16, Sec. 11.7]. We compute normwise relative errors $\|\widetilde{X} - \widehat{X}\|_F / \|\widetilde{X}\|_F$, where $\widehat{X}$ is a computed logarithm and $\widetilde{X}$ is the result of evaluating $\log(A)$ at 100 decimal digit precision using the Symbolic Math Toolbox and rounding the result to double precision, as well as normwise backward errors $\|e^{\widehat{X}} - A\|_F / \|A\|_F$, with $e^{\widehat{X}}$ computed at 100 decimal digit precision and then rounded to double precision. (Here we use the fact that $\widehat{X} = \log(A + \Delta A)$ implies $e^{\widehat{X}} = A + \Delta A$.) The relative errors in our plots have been transformed as suggested in [9] so as to lessen the influence of abnormally tiny errors on the performance profiles; the transformation simply applies a linear scaling that maps $[0, u]$ to $[5 \times 10^{-2}u, u]$.

The MATLAB codes and the algorithms they implement are as follows.

1. `iss_schur_new`: the Schur-based inverse scaling and squaring algorithm, Algorithm 4.1.
2. `iss_schur_old`: the Schur-based inverse scaling and squaring algorithm from [16, Alg. 11.9], which derives its parameters from the forward error bound (2.1).
3. The (standard) MATLAB function `funm`, called as `funm(A,@log)`, which is equivalent to `logm(A)`. This function implements a Schur–Parlett algorithm [7], [16, Alg. 11.11] that uses `iss_schur_old` on diagonal blocks of dimension 3 or larger in the partitioned and reordered triangular Schur factor.
4. A modified version of `funm`, denoted `funm_mod`, in which `iss_schur_new` is used in place of `iss_schur_old`.
5. `iss_new`: Algorithm 5.2.
6. `iss_old`: the transformation-free inverse scaling and squaring algorithm from [16, Alg. 11.10]. Like `iss_schur_old`, this algorithm derives its parameters using the forward error bound (2.1).

Experiment 1. First we compare the codes on the upper triangular matrix A given by

```
  3.2346e-001  3.0000e+004  3.0000e+004  3.0000e+004
            0  3.0089e-001  3.0000e+004  3.0000e+004
            0            0  3.2210e-001  3.0000e+004
            0            0            0  3.0744e-001
```

The true logarithm is, to five significant figures,

```
 -1.1287e+000  9.6142e+004 -4.5248e+009  2.9249e+014
            0 -1.2010e+000  9.6346e+004 -4.6810e+009
            0            0 -1.1329e+000  9.5324e+004
            0             0             0 -1.1795e+000
```
The result from iss_schur_new is correct to five significant figures, whereas iss_schur_old and funm produce the same matrix, which to five significant figures is
```
 -1.2500e+000  9.6142e+004 -4.5248e+009  2.9249e+014
            0 -1.2500e+000  9.6346e+004 -4.6810e+009
            0             0 -1.2500e+000  9.5324e+004
            0             0             0 -1.2500e+000
```
Note that the diagonal elements have only one or two correct significant figures. Nevertheless, all the codes produce a normwise relative error less than $9u$ due to the (1,4) element, which makes $\|\log(A)\|_F$ very large. However, the backward errors $\|e^{\widehat{X}} - A\|_F/\|A\|_F$ are 2.5×10^{-7} for iss_schur_new and 4.0×10^6 for iss_schur_old and funm.

For this matrix, iss_schur_new takes $s = 16$ and $m = 6$, while iss_schur_old takes $s = 50$ and $m = 7$. The much greater efficiency of iss_schur_new is due to it exploiting the nonnormality of A through the use of the α_p: we have $\{\|(A-I)^p\|_F^{1/p} : p = 1, 2, \ldots\} = \{7.3 \times 10^4, 4.7 \times 10^4, 3.0 \times 10^4, 3.0 \times 10^3, 6.6 \times 10^2, \ldots\}$, showing that $\alpha_p(A-I)^k \ll \|A-I\|^k$ for $p = 4$ (and similarly after square roots of A have been taken) and hence that our backward error bound is much sharper than it would be if we had expressed it solely in terms of $\|A - I\|$.

Experiment 2. In this experiment we compare iss_schur_new, iss_schur_old, funm, and funm_mod on the test set. Figure 6.1 plots the relative errors, with the solid line showing $\mathrm{cond}(\log, A)u$, where $\mathrm{cond}(\log, A)$ is the condition number of the matrix logarithm function at A, computed by logm_cond from the Matrix Function Toolbox [13], and the matrices are ordered by decreasing value of $\mathrm{cond}(\log, A)$. Figure 6.2 presents the same data in the form of a performance profile [9], [10], [12, Sec. 22.4]; here, the curve for a given method has height $p(\alpha)$ at α if that method had error within a factor α of the smallest error over the other three methods on a fraction $p(\alpha)$ of all the test problems. Figure 6.3 displays the ratio of the computational cost measured in flops for iss_schur_new and iss_schur_old excluding the cost of the transformation to and from Schur form; the cost is proportional to $s+m$. These results show that all the methods perform in a generally numerically forward stable way (Figure 6.1), and that iss_schur_new is a clear improvement over iss_schur_old and is even superior to funm (Figure 6.2); closer inspection of the errors reveals that iss_schur_new has errors up to factors of order 10^8 and 10^5 smaller than, and never more than 10% larger than, those for iss_schur_old and funm, respectively, Moreover, funm_mod shows some small improvements in accuracy over funm, indicating the benefit of using iss_schur_new instead of iss_schur_old within funm. We also computed backward errors; the resulting performance profile (not shown) is very similar to that for the forward errors, but with a less pronounced advantage for iss_schur_new. Figure 6.3 shows that iss_schur_new never requires more flops than iss_schur_old and can need up to a factor four fewer. Thus iss_schur_new improves in both speed and accuracy over iss_schur_old.

Experiment 3. In this experiment we use the upper triangular QR factors R of each matrix in the test set, replacing any negative diagonal element of R by its absolute value. The errors and performance profile for the same methods as in Experiment 2 are shown in Figures 6.4 and 6.5. The performance profile for the backward errors (not shown) is similar, except that the curve for iss_schur_old stays entirely below that for funm_mod.

This experiment shows the superior accuracy of iss_schur_new over the other

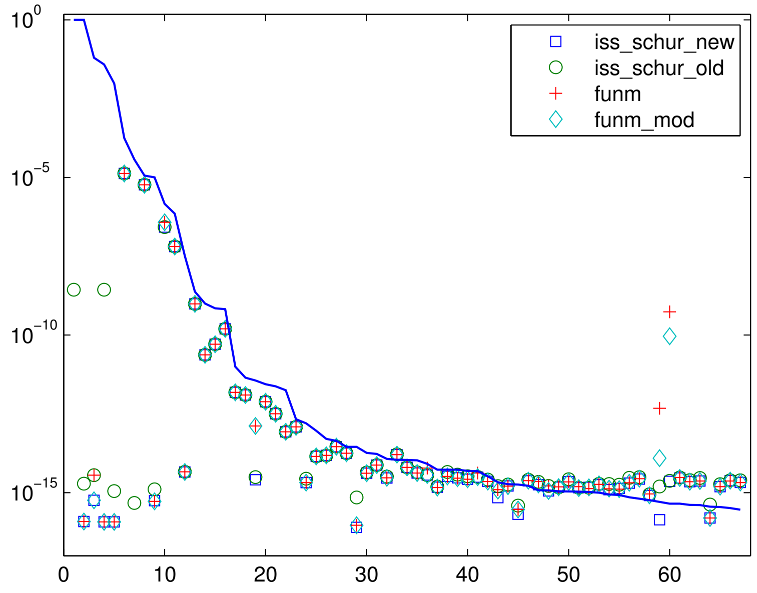

FIG. 6.1. *Experiment 2: normwise relative errors in* $\log(A)$ *computed by* `iss_schur_new`*,* `iss_schur_old`*,* `funm`*, and* `funm_mod`*. The solid line is* $\mathrm{cond}(\log, A)u$*.*

codes for triangular matrices (in Experiment 2 the advantage tends to be reduced by the errors introduced by the Schur transformation). Indeed we can see from Figure 6.4 several matrices for which `funm` and `funm_mod` produce much less accurate results than `iss_schur_new` and behave in a forward unstable way. We also see that `funm_mod` delivers better accuracy than `funm`.

Experiment 4. In this experiment we compare the transformation-free codes `iss_new` and `iss_old` on the test set. We also try `iss_new`*, which denotes `iss_new` without the use of Algorithm 5.1, so that lines 48–50 are deleted and lines 52–56 are replaced by $Y = A - I$. Figure 6.6 plots the relative errors for these codes along with `iss_schur_new` and Figure 6.7 shows the corresponding performance profile. Again, the performance profile for the backward errors (not shown) is very similar to that for the forward errors. Figure 6.7 shows a clear improvement in accuracy of `iss_new` over `iss_old`, and the curve for `iss_new`* shows that some of this improvement is due to the use of Algorithm 5.1. Figure 6.8 compares the computational cost measured in flops of `iss_new` with `iss_old`; `iss_new` is usually the faster, by up to a factor 18, and is at most a factor 1.1 slower. As for the Schur-based algorithms, our new algorithm `iss_new` brings benefits in both speed and accuracy over `iss_old`.

7. Conclusions.

Our new algorithms, Algorithms 4.1 and 5.2, improve significantly in speed and accuracy on those of Higham [16, Algs 11.9, 11.10], which in turn are refinements of those of Cheng, Higham, Kenney, and Laub [6] and Kenney and Laub [20]. The principal improvements are (a) the use of backward error (instead of forward error) bounds and the use of estimates of norms of matrix powers in order to incorporate information about nonnormality and obtain sharper error bounds—both of which lead to better choices of s (the number of square roots) and m (the degree of the Padé approximant), and (b) the steps taken to avoid cancellation in the argument of the Padé approximant and the exploitation of triangular structure in Algorithm 4.1

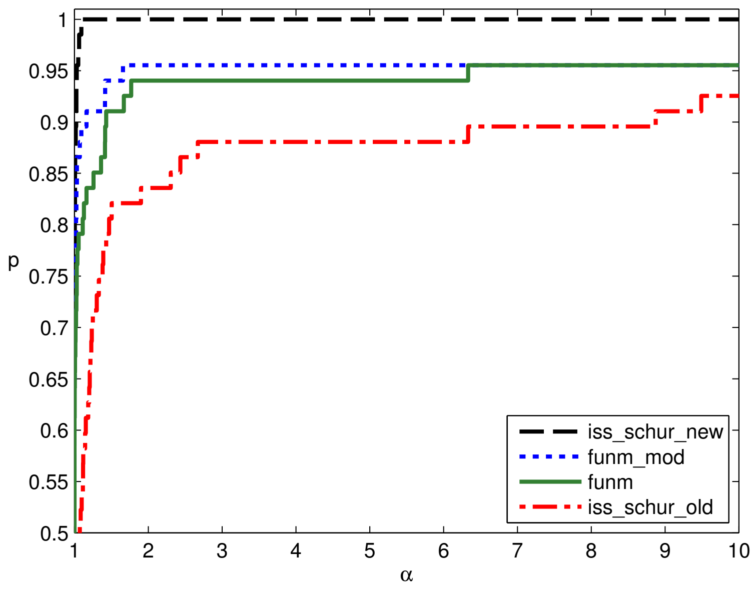

Fig. 6.2. *Experiment* 2*: performance profile for the data presented in Figure* 6.1*. The intercepts with $\alpha = 1$ are at $p =$* 0.55*,* 0.51*,* 0.39*, and* 0.22 *for* `iss_schur_new`*,* `funm_mod`*,* `funm`*, and* `iss_old`*, respectively.*

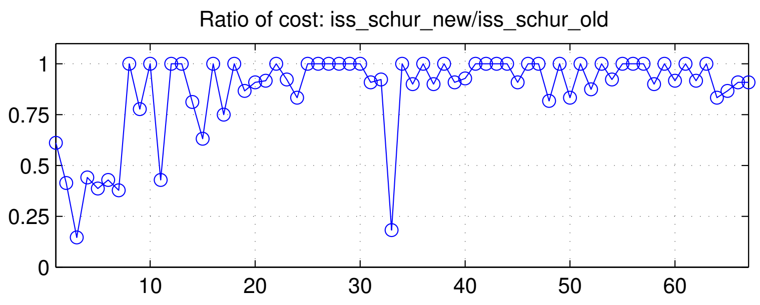

Fig. 6.3. *Experiment* 2*: ratios of the cost of* `iss_schur_new` *divided by the cost of* `iss_schur_old`.

to directly compute certain elements of $\log(T)$. Algorithm 4.1 emerges as the method of choice for computing $\log(A)$, and is a natural partner to our scaling and squaring algorithm for e^A in [2], which—although not based on the Schur form—uses similar techniques to maximize speed and accuracy.

Acknowledgments. The first author thanks the Manchester Institute for Mathematical Sciences for hosting a visit during which much of this work was done.

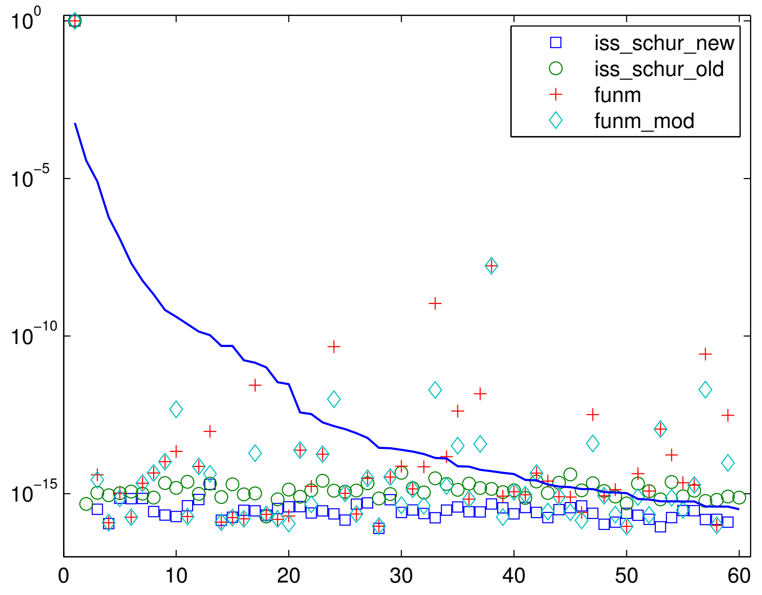

FIG. 6.4. *Experiment 3: normwise relative errors in* $\log(A)$ *for triangular* A *computed by* `iss_schur_new`, `iss_schur_old`, `funm`, *and* `funm_mod`. *The solid line is* $\operatorname{cond}(\log, A)u$.

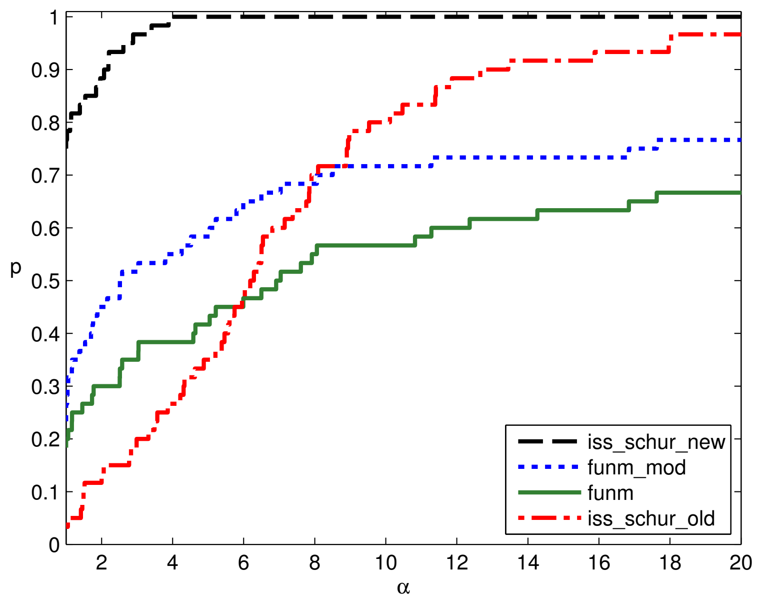

FIG. 6.5. *Experiment 3: performance profile for the data presented in Figure* 6.4.

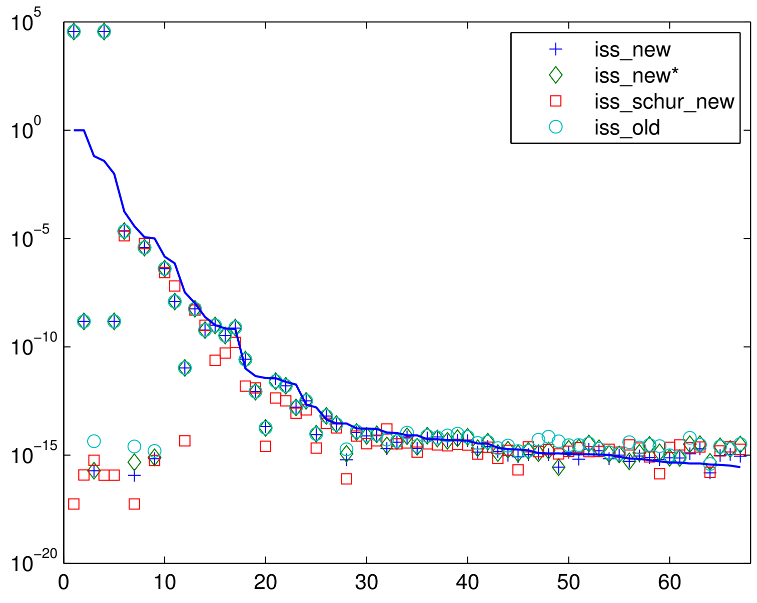

FIG. 6.6. *Experiment 4: normwise relative errors in* $\log(A)$ *computed by* `iss_schur_new`*,* `iss_new`*,* `iss_new`**, and* `iss_old`*. The solid line is* $\operatorname{cond}(\log, A)u$*.*

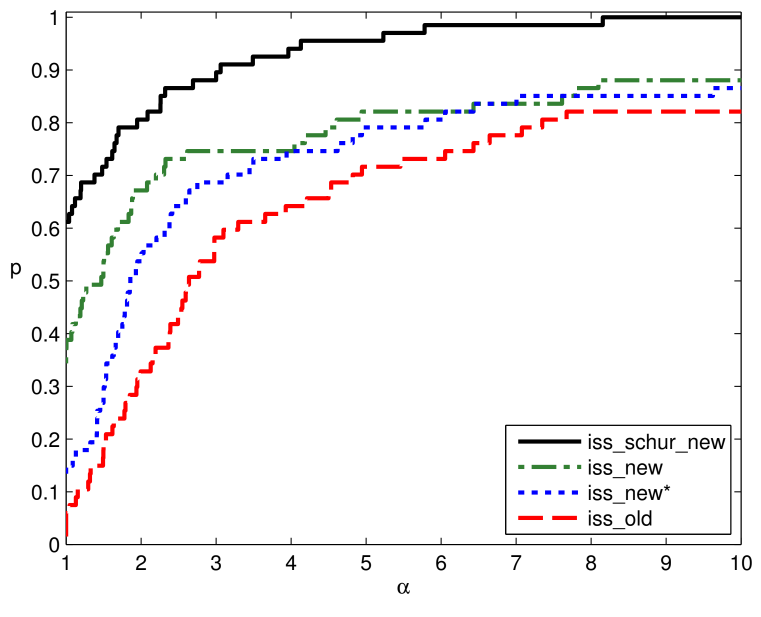

FIG. 6.7. *Experiment 4: performance profiles for the data presented in Figure* 6.6.

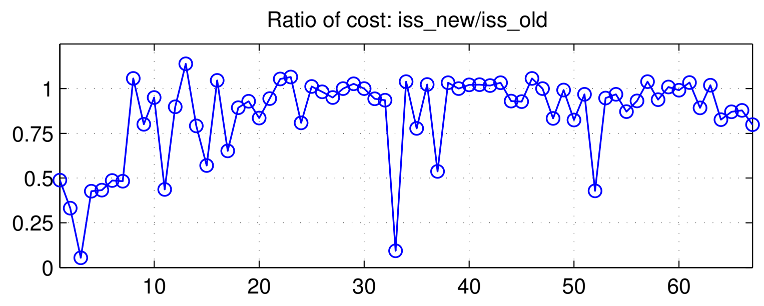

Fig. 6.8. *Experiment 4: ratios of the cost of* `iss_new` *divided by the cost of* `iss_old`*.*

REFERENCES

[1] Awad H. Al-Mohy. A more accurate Briggs method for the logarithm. *Numer. Algorithms*, 59 (3):393–402, 2012.

[2] Awad H. Al-Mohy and Nicholas J. Higham. A new scaling and squaring algorithm for the matrix exponential. *SIAM J. Matrix Anal. Appl.*, 31(3):970–989, 2009.

[3] Åke Björck and Sven Hammarling. A Schur method for the square root of a matrix. *Linear Algebra Appl.*, 52/53:127–140, 1983.

[4] João R. Cardoso and F. Silva Leite. Theoretical and numerical considerations about Padé approximants for the matrix logarithm. *Linear Algebra Appl.*, 330:31–42, 2001.

[5] João R. Cardoso and F. Silva Leite. Padé and Gregory error estimates for the logarithm of block triangular matrices. *Appl. Numer. Math.*, 56:253–267, 2006.

[6] Sheung Hun Cheng, Nicholas J. Higham, Charles S. Kenney, and Alan J. Laub. Approximating the logarithm of a matrix to specified accuracy. *SIAM J. Matrix Anal. Appl.*, 22(4):1112–1125, 2001.

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

[8] Luca Dieci, Benedetta Morini, and Alessandra Papini. Computational techniques for real logarithms of matrices. *SIAM J. Matrix Anal. Appl.*, 17(3):570–593, 1996.

[9] Nicholas J. Dingle and Nicholas J. Higham. Reducing the influence of tiny normwise relative errors on performance profiles. MIMS EPrint 2011.90, Manchester Institute for Mathematical Sciences, The University of Manchester, UK, November 2011. 11 pp.

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

[11] Herman H. Goldstine. *A History of Numerical Analysis from the 16th through the 19th Century.* Springer-Verlag, New York, 1977. xiv+348 pp. ISBN 0-387-90277-5.

[12] Desmond J. Higham and Nicholas J. Higham. *MATLAB Guide.* Second edition, Society for Industrial and Applied Mathematics, Philadelphia, PA, USA, 2005. xxiii+382 pp. ISBN 0-89871-578-4.

[13] Nicholas J. Higham. The Matrix Function Toolbox. `http://www.ma.man.ac.uk/~higham/mftoolbox`.

[14] Nicholas J. Higham. Evaluating Padé approximants of the matrix logarithm. *SIAM J. Matrix Anal. Appl.*, 22(4):1126–1135, 2001.

[15] Nicholas J. Higham. *Accuracy and Stability of Numerical Algorithms.* Second edition, Society for Industrial and Applied Mathematics, Philadelphia, PA, USA, 2002. xxx+680 pp. ISBN 0-89871-521-0.

[16] Nicholas J. Higham. *Functions of Matrices: Theory and Computation.* Society for Industrial and Applied Mathematics, Philadelphia, PA, USA, 2008. xx+425 pp. ISBN 978-0-898716-46-7.

[17] Nicholas J. Higham and Awad H. Al-Mohy. Computing matrix functions. *Acta Numerica*, 19: 159–208, 2010.

[18] Nicholas J. Higham and Lijing Lin. A Schur–Padé algorithm for fractional powers of a matrix. *SIAM J. Matrix Anal. Appl.*, 32(3):1056–1078, 2011.

[19] Nicholas J. Higham and Françoise Tisseur. A block algorithm for matrix 1-norm estimation, with an application to 1-norm pseudospectra. *SIAM J. Matrix Anal. Appl.*, 21(4):1185–1201, 2000.

[20] Charles S. Kenney and Alan J. Laub. Condition estimates for matrix functions. *SIAM J. Matrix Anal. Appl.*, 10(2):191–209, 1989.

[21] Charles S. Kenney and Alan J. Laub. Padé error estimates for the logarithm of a matrix. *Internat. J. Control*, 50(3):707–730, 1989.

[22] George M. Phillips. *Two Millennia of Mathematics: From Archimedes to Gauss.* Springer-Verlag, New York, 2000. xii+223 pp. ISBN 0-387-95022-2.