

***Computing the Frechet Derivative of the Matrix
Logarithm and Estimating the Condition Number***

Al-Mohy, Awad H. and Higham, Nicholas J. and Relton,
Samuel D.

2012

MIMS EPrint: **2012.72**

Manchester Institute for Mathematical Sciences
School of Mathematics

The University of Manchester

Reports available from: <http://eprints.maths.manchester.ac.uk/>

And by contacting: The MIMS Secretary
School of Mathematics
The University of Manchester
Manchester, M13 9PL, UK

ISSN 1749-9097

COMPUTING THE FRÉCHET DERIVATIVE OF THE MATRIX LOGARITHM AND ESTIMATING THE CONDITION NUMBER*

AWAD H. AL-MOHY[†], NICHOLAS J. HIGHAM[‡], AND SAMUEL D. RELTON[‡]

Abstract. The most popular method for computing the matrix logarithm is the inverse scaling and squaring method, which is the basis of the recent algorithm of [A. H. Al-Mohy and N. J. Higham, *Improved inverse scaling and squaring algorithms for the matrix logarithm*, SIAM J. Sci. Comput., 34 (2012), pp. C152–C169]. We show that by differentiating the latter algorithm a backward stable algorithm for computing the Fréchet derivative of the matrix logarithm is obtained. This algorithm requires complex arithmetic, but we also develop a version that uses only real arithmetic when A is real; as a special case we obtain a new algorithm for computing the logarithm of a real matrix in real arithmetic. We show experimentally that our two algorithms are more accurate and efficient than existing algorithms for computing the Fréchet derivative. We also show how the algorithms can be used to produce reliable estimates of the condition number of the matrix logarithm.

Key words. matrix logarithm, principal logarithm, inverse scaling and squaring method, Fréchet derivative, condition number, Padé approximation, backward error analysis, matrix exponential, matrix square root, MATLAB, logm.

AMS subject classifications. 65F60, 65F30

1. Introduction. A logarithm of $A \in \mathbb{C}^{n \times n}$ is a matrix X such that $e^X = A$. When A has no eigenvalues on \mathbb{R}^- , the closed negative real line, there is a unique logarithm X whose eigenvalues lie in the strip $\{z : -\pi < \text{Im}(z) < \pi\}$ [24, Thm. 1.31]. This is the principal logarithm denoted by $\log(A)$. Under the same assumptions on A , there is a unique matrix X satisfying $X^2 = A$ that has all its eigenvalues in the open right half-plane [24, Thm. 1.29]. This is the principal square root of A , denoted by $A^{1/2}$.

The matrix logarithm appears in a wide variety of applications, of which some recent examples include reduced-order models [6], image registration [8], patch modeling-based skin detection [28], and aesthetically pleasing computer animations [33]. The Fréchet derivative of the matrix logarithm has recently been used in nonlinear optimization techniques for model reduction [32].

An excellent method for evaluating the matrix logarithm is the inverse scaling and squaring method proposed by Kenney and Laub [30], which uses the relationship $\log(A) = 2^s \log(A^{1/2^s})$ together with a Padé approximant of $\log(A^{1/2^s})$. This method has been developed by several authors, including Dieci, Morini, and Papini [15], Cardoso and Silva Leite [11], Cheng, Higham, Kenney, and Laub [12], and Higham [24, sect. 11.5]. Most recently, Al-Mohy and Higham [5] developed backward error analysis for the method and obtained an algorithm that is faster and more accurate than previous inverse scaling and squaring algorithms.

This work has three main aims: to develop a version of the algorithm of Al-Mohy and Higham [5] that works with the real Schur decomposition when A is real, to extend both versions of the algorithm to compute the Fréchet derivative, and to

* Version of July 25, 2012. This work was supported by European Research Council Advanced Grant MATFUN (267526).

[†]Department of Mathematics, King Khalid University, PO Box 9004 Abha, Saudi Arabia (aal-mohy@hotmail.com, <http://www.ma.man.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>, samuel.relton@ma.man.ac.uk, <http://www.ma.man.ac.uk/~srelton>).

develop an integrated algorithm that first computes $\log(A)$ and then computes one or more Fréchet derivatives with appropriate reuse of information—in particular, to allow efficient estimation of the condition number of the logarithm.

We recall that the Fréchet derivative of a function $f: \mathbb{C}^{n \times n} \rightarrow \mathbb{C}^{n \times n}$ at a point $A \in \mathbb{C}^{n \times n}$ is a linear function mapping $E \in \mathbb{C}^{n \times n}$ to $L_f(A, E) \in \mathbb{C}^{n \times n}$ such that

$$f(A + E) - f(A) - L_f(A, E) = o(\|E\|), \quad (1.1)$$

and if $L_f(A, E)$ exists then it is unique. As well as providing information about the sensitivity of f in a given direction E , the Fréchet derivative has an intimate connection with the relative condition number of f ,

$$\text{cond}(f, A) = \lim_{\epsilon \rightarrow 0} \sup_{\|E\| \leq \epsilon \|A\|} \frac{\|f(A + E) - f(A)\|}{\epsilon \|f(A)\|}. \quad (1.2)$$

Throughout this paper the norm is any subordinate matrix norm. The condition number has the explicit representation [24, Thm. 3.1]

$$\text{cond}(f, A) = \frac{\|L_f(A)\| \|A\|}{\|f(A)\|}, \quad (1.3)$$

where

$$\|L_f(A)\| = \max_{E \neq 0} \frac{\|L_f(A, E)\|}{\|E\|}. \quad (1.4)$$

Hence if we can compute $L_f(A, E)$ for any given E then we can estimate $\text{cond}(f, A)$ using a norm estimator. Important in this respect is the representation

$$\text{vec}(L_f(A, E)) = K_f(A) \text{vec}(E), \quad (1.5)$$

where $K_f(A) \in \mathbb{C}^{n^2 \times n^2}$ is the Kronecker form of the Fréchet derivative and vec is the operator that stacks the columns of a matrix on top of each other [24, sect. 3.2].

The rest of this paper is organized as follows. In section 2 we describe the outline of our algorithm for computing the Fréchet derivative of the matrix logarithm. We derive backward error bounds in section 3 and use them to choose the algorithmic parameters. Estimation of the condition number is discussed in section 4 and detailed algorithms are then given in section 5 for the complex case and section 6 for the real case. We compare our algorithms to current alternatives theoretically in section 7 before performing numerical experiments in section 8. Our conclusions are presented in section 9.

2. Basic algorithm. We begin by deriving the basic structure of an algorithm for approximating the Fréchet derivative of the logarithm. The idea is to Fréchet-differentiate the inverse scaling and squaring approximation $\log(A) \approx 2^s r_m(A^{1/2^s} - I)$, where $r_m(x)$ is the $[m/m]$ Padé approximant to $\log(1+x)$, following the computational framework suggested in [2], [25, sect. 7.4].

Applying a result on the Fréchet derivative of an inverse function [24, Thm. 3.5] and using $L_{x^2}(A, E) = AE + EA$ gives us that $L = L_{x^{1/2}}(A, E)$ satisfies $A^{1/2}L + LA^{1/2} = E$. Applying the chain rule for Fréchet derivatives [24, Thm. 3.4] to $\log(A) = 2 \log(A^{1/2})$ gives $L_{\log}(A, E) = 2L_{\log}(A^{1/2}, E_1)$ where $E_1 = L_{x^{1/2}}(A, E)$. Repeated use of these two results yields $L_{\log}(A, E) = 2^s L_{\log}(A^{1/2^s}, E_s)$ where $E_0 = E$ and

$$A^{1/2^i} E_i + E_i A^{1/2^i} = E_{i-1}, \quad i = 1: s. \quad (2.1)$$

For suitably chosen s and m we then approximate $L_{\log}(A, E) \approx 2^s L_{r_m}(A^{1/2^s} - I, E_s)$.

The following outline algorithm will be refined in the subsequent sections.

ALGORITHM 2.1. *Given $A \in \mathbb{C}^{n \times n}$ with no eigenvalues on \mathbb{R}^- , $E \in \mathbb{C}^{n \times n}$, and nonnegative integers s and m , this algorithm approximates $\log(A)$ and the Fréchet derivative $L_{\log}(A, E)$.*

- 1 $E_0 = E$
- 2 for $i = 1: s$
- 3 Compute $A^{1/2^i}$.
- 4 Solve the Sylvester equation $A^{1/2^i} E_i + E_i A^{1/2^i} = E_{i-1}$ for E_i .
- 5 end
- 6 $\log(A) \approx 2^s r_m(A^{1/2^s} - I)$
- 7 $L_{\log}(A, E) \approx 2^s L_{r_m}(A^{1/2^s} - I, E_s)$

Higham [22] shows that among the various alternative representations of r_m , the partial fraction form given by

$$r_m(X) = \sum_{j=1}^m \alpha_j^{(m)} (I + \beta_j^{(m)} X)^{-1} X, \quad (2.2)$$

where $\alpha_j^{(m)}, \beta_j^{(m)} \in (0, 1)$ are the weights and nodes of the m -point Gauss–Legendre quadrature rule on $[0, 1]$ respectively, provides the best balance between efficiency and numerical stability. To calculate the Fréchet derivative L_{r_m} , we differentiate (2.2) using the product rule. Recalling that $L_{x^{-1}}(X, E) = -X^{-1} E X^{-1}$ we obtain

$$\begin{aligned} L_{r_m}(X, E) &= \sum_{j=1}^m \alpha_j^{(m)} (I + \beta_j^{(m)} X)^{-1} E - \alpha_j^{(m)} \beta_j^{(m)} (I + \beta_j^{(m)} X)^{-1} E (I + \beta_j^{(m)} X)^{-1} X \\ &= \sum_{j=1}^m \left(\alpha_j^{(m)} (I + \beta_j^{(m)} X)^{-1} E \right) \left(I - \beta_j^{(m)} (I + \beta_j^{(m)} X)^{-1} X \right) \\ &= \sum_{j=1}^m \alpha_j^{(m)} (I + \beta_j^{(m)} X)^{-1} E (I + \beta_j^{(m)} X)^{-1}. \end{aligned} \quad (2.3)$$

3. Backward error analysis. We now develop a backward error result for the approximation errors in lines 6 and 7 of Algorithm 2.1. Define the function $h_{2m+1}(X): \mathbb{C}^{n \times n} \mapsto \mathbb{C}^{n \times n}$ by $h_{2m+1}(X) = e^{r_m(X)} - X - I$, which has the power series expansion

$$h_{2m+1}(X) = \sum_{k=2m+1}^{\infty} c_k X^k. \quad (3.1)$$

We need the backward error result from [5] for the approximation of the logarithm. In the following, ρ denotes the spectral radius.

THEOREM 3.1. *If $X \in \mathbb{C}^{n \times n}$ satisfies $\rho(r_m(X)) < \pi$ then $r_m(X) = \log(I + X + \Delta X)$, where, for any $p \geq 1$ satisfying $2m + 1 \geq p(p - 1)$,*

$$\frac{\|\Delta X\|}{\|X\|} \leq \sum_{k=2m+1}^{\infty} |c_k| \alpha_p(X)^{k-1}, \quad (3.2)$$

for $\alpha_p(X) = \max(\|X^p\|^{1/p}, \|X^{p+1}\|^{1/(p+1)})$. Furthermore, $\Delta X = h_{2m+1}(X)$.

TABLE 3.1

Maximal values θ_m of $\alpha_p(X)$ such that the bound in (3.2) for $\|\Delta X\|/\|X\|$ does not exceed u .

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

The $\alpha_p(X)$ values were first introduced and exploited in [3]. We recall that $\alpha_p(X) \leq \|X\|$ and that $\alpha_p(X) \ll \|X\|$ is possible for very nonnormal X , so that bounds based upon $\alpha_p(X)$ are potentially much sharper than bounds based solely on $\|X\|$.

Maximal values of θ_m such that $\alpha_p(X) \leq \theta_m$ implies $\|\Delta X\|/\|X\| \leq u$, where $u = 2^{-53} \approx 1.1 \times 10^{-16}$ is the unit roundoff for IEEE double precision arithmetic, were determined in [5] for $m = 1: 16$ and are shown in Table 3.1. It is shown in [5] that $\rho(X) < 0.91$ implies $\rho(r_m(X)) < \pi$. Since we will need $\alpha_p(X) \leq \theta_{16} = 0.749$ and as $\rho(X) \leq \alpha_p(X)$ for all p , the condition $\rho(r_m(X)) < \pi$ in Theorem 3.1 is not a practical restriction. Now we give a backward error result for the Fréchet derivative computed via Algorithm 2.1.

THEOREM 3.2. *If $X \in \mathbb{C}^{n \times n}$ satisfies $\rho(r_m(X)) < \pi$ then*

$$L_{r_m}(X, E) = L_{\log}(I + X + \Delta X, E + \Delta E), \quad (3.3)$$

where $\Delta X = h_{2m+1}(X)$ and $\Delta E = L_{h_{2m+1}}(X, E)$.

Proof. From Theorem 3.1 we know that $r_m(X) = \log(I + X + \Delta X)$ with $\Delta X = h_{2m+1}(X)$. Using the chain rule we obtain

$$\begin{aligned} L_{r_m}(X, E) &= L_{\log}(I + X + h_{2m+1}(X), E + L_{h_{2m+1}}(X, E)) \\ &=: L_{\log}(I + X + \Delta X, E + \Delta E). \quad \square \end{aligned}$$

Note that Theorems 3.1 and 3.2 show that $r_m(X) = \log(I + X + \Delta X)$ and $L_{r_m}(X, E) = L_{\log}(I + X + \Delta X, E + \Delta E)$ with the *same* ΔX , so we have a single backward error result encompassing both r_m and L_{r_m} . It remains for us to bound $\|\Delta E\|$, which can be done using the following result [24, Prob. 3.6], [30].

LEMMA 3.3. *Suppose f has the power series expansion $f(x) = \sum_{k=1}^{\infty} a_k x^k$ with radius of convergence r . Then for $X, E \in \mathbb{C}^{n \times n}$ with $\|X\| < r$,*

$$L_f(X, E) = \sum_{k=1}^{\infty} a_k \sum_{j=1}^k X^{j-1} E X^{k-j}. \quad (3.4)$$

We would like to use Lemma 3.3 with $f = h_{2m+1}$ to obtain a bound similar to (3.2) in terms of $\alpha_p(X)$. Unfortunately, this is impossible when X and E do not commute (which must be assumed, since we do not wish to restrict E). To see why, note that the sum in (3.4) contains terms $a_k X^{j-1} E X^{k-j}$, with j ranging from 1 to k . The next lemma implies that $\|X\|$, for example, is always a factor in the norm of one of these terms for some E .

LEMMA 3.4. *For $A, B, E \in \mathbb{C}^{n \times n}$, $\|AEB\| \leq \|A\| \|E\| \|B\|$ and the bound is attained for a rank-1 matrix E .*

TABLE 3.2

Maximal values β_m of $\|X\|$ such that the bound in (3.5) for $\|\Delta E\|/\|E\|$ does not exceed u , and the values of μ_m .

m	1	2	3	4	5	6	7	8
β_m	2.11e-8	2.51e-4	5.93e-3	2.89e-2	7.39e-2	1.36e-1	2.08e-1	2.81e-1
μ_m	4.00e0	6.00e0	8.06e0	1.03e1	1.27e1	1.54e1	1.85e1	2.20e1
m	9	10	11	12	13	14	15	16
β_m	3.52e-1	4.17e-1	4.77e-1	5.30e-1	5.77e-1	6.18e-1	6.54e-1	6.86e-1
μ_m	2.59e1	3.02e1	3.49e1	4.00e1	4.56e1	5.15e1	5.79e1	6.48e1

Proof. Only the attainability of the bound is in question. Using properties of the norm $\|\cdot\|_D$ dual to $\|\cdot\|$ [23, sect. 6.1, Probs. 6.2, 6.3] it is straightforward to show that the bound is attained for $E = xy^*$, where $\|Ax\| = \|A\|\|x\|$ and $\|B^*y\|_D = \|B^*\|_D\|y\|_D$. \square

It follows that we cannot obtain a bound for $\|\Delta E\|/\|E\|$ depending only on $\alpha_p(X)$. We therefore simply take norms in (3.4) with $f = h_{2m+1}$ (see (3.1)), to obtain

$$\frac{\|\Delta E\|}{\|E\|} \leq \sum_{k=2m+1}^{\infty} k|c_k|\|X\|^{k-1}. \quad (3.5)$$

Define $\beta_m = \max\{\theta : \sum_{k=2m+1}^{\infty} k|c_k|\theta^{k-1} \leq u\}$, so that $\|X\| \leq \beta_m$ implies that $\|\Delta E\|/\|E\| \leq u$. Table 3.2 shows the values of β_m , calculated using the Symbolic Math Toolbox for MATLAB. In each case we have $\beta_m < \theta_m$, as is immediate from the definitions of β_m and θ_m .

There is a way to obtain unified bounds for $\|\Delta X\|/\|X\|$ and $\|\Delta E\|/\|E\|$ in terms of $\alpha_p(X)$. For $\alpha_p(X) < 1$ there exists $\epsilon > 0$ and a matrix norm $\|\cdot\|_{\epsilon}$ such that

$$\|X\|_{\epsilon} \leq \rho(X) + \epsilon \leq \alpha_p(X) + \epsilon < 1.$$

Taking norms in (3.1) using $\|\cdot\|_{\epsilon}$ and taking $\|\cdot\| = \|\cdot\|_{\epsilon}$ in (3.5) we obtain

$$\begin{aligned} \frac{\|\Delta X\|_{\epsilon}}{\|X\|_{\epsilon}} &\leq \sum_{k=2m+1}^{\infty} |c_k|(\alpha_p(X) + \epsilon)^{k-1}, \\ \frac{\|\Delta E\|_{\epsilon}}{\|E\|_{\epsilon}} &\leq \sum_{k=2m+1}^{\infty} k|c_k|(\alpha_p(X) + \epsilon)^{k-1}. \end{aligned}$$

Unfortunately the norm $\|\cdot\|_{\epsilon}$ is badly scaled if ϵ is small, so these bounds are difficult to interpret.

We will build our algorithm for computing the logarithm and its derivative on the condition $\alpha_p(X) \leq \theta_m$ that ensures that $\|\Delta X\|/\|X\| \leq u$. The bound (3.5) for $\|\Delta E\|/\|E\|$ is generally larger than u due to (a) $\|X\|$ exceeding $\alpha_p(X)$ by a factor that can be arbitrarily large and (b) the extra factor k in (3.5) compared with (3.2). The effect of (b) can be bounded as follows. Suppose we take $\|X\| \leq \theta_m$, and define μ_m by

$$\mu_m = \frac{1}{u} \sum_{k=2m+1}^{\infty} k|c_k|\theta_m^{k-1}. \quad (3.6)$$

Then $\|X\| \leq \theta_m$ implies that $\|\Delta E\|/\|E\| \leq \mu_m u$. Table 3.2 gives the values of μ_m , which we see are of modest size and increase slowly with m . The algorithm of [5] normally chooses a Padé approximant of degree $m = 6$ or 7 , for which we have the reasonable bound $\|\Delta E\|/\|E\| \leq 18.5u$.

Since our main use of the Fréchet derivative is for condition number estimation, for which only order of magnitude estimates are required, it is reasonable to allow this more liberal bounding on the backward error ΔE . We will see in the numerical experiments of section 8 that in fact our algorithm gives very accurate estimates of the Fréchet derivative in practice.

4. Condition number estimation. In the Frobenius norm the condition number (1.3) can be computed by explicitly computing the Kronecker form of the Fréchet derivative and taking its 2-norm [24, sects. 3.2, 3.4]. However, this is an expensive computation that requires $O(n^5)$ flops. We will therefore estimate the condition number, and for this we will use the block 1-norm estimation algorithm of Higham and Tisseur [27], which is available in MATLAB as function `normest1`; this algorithm carries out iterations with $n \times t$ matrices, where t is a parameter whose significance is explained below. In the 1-norm we have [24, Lem. 3.18]

$$\frac{\|L_f(A)\|_1}{n} \leq \|K_f(A)\|_1 \leq n\|L_f(A)\|_1. \quad (4.1)$$

Using the block 1-norm estimation algorithm, we now employ the same idea as [2, Alg. 7.3] and approximate $\|L_{\log}(A)\|_1 \approx \|K_{\log}(A)\|_1$.

ALGORITHM 4.1. *This algorithm estimates the 1-norm of $K_{\log}(A)$ using Fréchet derivative evaluations of the logarithm at A .*

- 1 Apply [27, Alg. 2.4] with parameter $t = 2$ to the Kronecker matrix $B = K_{\log}(A)$, noting that $By = \text{vec}(L_{\log}(A, E))$ and $B^*y = \text{vec}(L_{\log}^*(A, E))$, where $\text{vec}(E) = y$.

Here, \star denotes the adjoint, and as the logarithm has a real power series expansion, $L_{\log}^*(A, E) = L_{\log}(A^*, E) = L_{\log}(A, E^*)^*$ and so it is straightforward to compute $L_{\log}^*(A, E)$. Known properties of Algorithm 4.1 are that it requires $4t$ Fréchet derivative evaluations on average and is rarely more than a factor of 3 away from $\|K_{\log}(A)\|_1$ [24, p. 67]. Higher values of t give greater accuracy at the cost of more derivative evaluations.

In the next two sections we give algorithms that compute $\log(A)$ and one or more Fréchet derivatives $L_{\log}(A, E)$ and $L_{\log}^*(A, E)$, making appropriate reuse of information from the logarithm computation in the calculation of the Fréchet derivatives.

5. Complex algorithm. We first give an algorithm that computes the matrix logarithm and one or more Fréchet derivatives and adjoints of Fréchet derivatives using complex arithmetic, building on Algorithm 2.1. As in [5] and [24, Alg. 11.9] we begin with a reduction to Schur form $A = QTQ^*$ (Q unitary, T upper triangular), because working with a triangular matrix leads to a generally smaller operation count and better accuracy. We will use the relation $L_{\log}(A, E) = QL_{\log}(T, Q^*EQ)Q^*$ [24, Prob 3.2]. Our algorithm employs the inverse scaling and squaring algorithm in [5, Alg. 4.1] and reduces to it if all the lines associated with the Fréchet derivative are removed.

ALGORITHM 5.1. *Given $A \in \mathbb{C}^{n \times n}$ with no eigenvalues on \mathbb{R}^- this algorithm computes the principal logarithm $X = \log(A)$ and one or more Fréchet derivatives $L_{\log}(A, E)$ and $L_{\log}^*(A, E)$.*

- 1 Compute a complex Schur decomposition $A = QTQ^*$.
- 2 $T_0 = T$
- 3 Determine the integers s and $m \in [1, 7]$ as in [5, Alg. 4.1], at the same time computing (and storing, if Fréchet derivatives are required) the matrices $T_{k+1} = T_k^{1/2}$, $k = 0: s - 1$ using the recurrence of [9], [24, Alg. 6.3].
- 4 $R = T_s - I$
- 5 Replace the diagonal and first superdiagonal of R by the diagonal and first superdiagonal of $T_0^{1/2^s} - I$ computed via [1, Alg. 2] and [26, (5.6)], respectively.
- 6 $X = 0$
- 7 for $i = 1:m$
 - 8 Solve $(I + \beta_j^{(m)} R)U = \alpha_j^{(m)} R$ for U by substitution.
 - 9 $X \leftarrow X + U$
- 10 end
- 11 $X \leftarrow 2^s X$
- 12 Replace $\text{diag}(X)$ by $\log(\text{diag}(T_0))$ and the elements of the first superdiagonal of X with those given by [24, (11.28)] taking $T = T_0$.
- 13 $X \leftarrow QXQ^*$
- 14 ... To compute $L_{\log}(A, E)$ for a given E :
 - 15 $E_0 = Q^*EQ$
 - 16 for $i = 1:s$
 - 17 Solve the Sylvester equation $T_i E_i + E_i T_i = E_{i-1}$ for E_i by substitution.
 - 18 end
 - 19 $L = 0$
 - 20 for $j = 1:m$
 - 21 Compute $Y = \alpha_j^{(m)}(I + \beta_j^{(m)} R)^{-1} E_s (I + \beta_j^{(m)} R)^{-1}$ by substitution.
 - 22 $L \leftarrow L + Y$
 - 23 end
 - 24 $L \leftarrow 2^s QLQ^*$
- 25 ... To compute $L_{\log}^*(A, E)$ for a given E :
 - 26 Execute lines 15–24 with E replaced by E^* and take the conjugate transpose of the result.

Cost: $25n^3$ flops for the Schur decomposition plus $(3 + (s + m)/3)n^3$ flops for X and $(8 + 2(s + m))n^3$ flops for each Fréchet derivative evaluation.

6. Real algorithm. If A and E are real and A has no eigenvalues on \mathbb{R}^- then both $\log(A)$ and $L_{\log}(A, E)$ will be real. To avoid complex arithmetic we can modify Algorithm 5.1 to use a real Schur decomposition $A = QTQ^T$, where Q is orthogonal and T is upper quasi-triangular, that is, block upper triangular with diagonal blocks of dimension 1 or 2. The use of real arithmetic increases the efficiency of the algorithm, since a complex elementary operation has the same cost as two or more real elementary operations. It also avoids the result being contaminated by small imaginary parts due to rounding error and halves the required intermediate storage.

The main difference from Algorithm 5.1 is that since T is now upper quasi-triangular it is more complicated to replace the diagonal and superdiagonal elements of the shifted square root and final $\log(T)$ with more accurately computed ones.

Consider first the computation of $T^{1/2^s} - I$. To avoid cancellation we recompute the 1×1 diagonal blocks using [1, Alg. 2] and the 2×2 blocks using [5, Alg. 5.1],

modified to use the recurrence of [21] for the square roots. We also recompute every superdiagonal $(i, i + 1)$ element for which the (i, i) and $(i + 1, i + 1)$ elements are in 1×1 blocks using [26, (5.6)].

As in the complex version of the algorithm, it is desirable to replace the diagonal blocks of the computed $\log(T)$ with more accurately computed ones. When the real Schur decomposition is computed using `dgees` from LAPACK [7], as in MATLAB, the 2×2 diagonal blocks are of the form

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

where $bc < 0$ and B has eigenvalues $\lambda_{\pm} = a \pm i(-bc)^{1/2}$. Let $\theta = \arg(\lambda_+) \in (-\pi, \pi)$. Using the polynomial interpolation definition of a matrix function [24, Def. 1.4] we can derive the formula

$$\log(B) = \begin{bmatrix} \log(a^2 - bc)/2 & \theta b(-bc)^{-1/2} \\ \theta c(-bc)^{-1/2} & \log(a^2 - bc)/2 \end{bmatrix}, \quad (6.1)$$

which can also be obtained by specializing a formula in [15, Lem. 3.3]. Since $bc < 0$, (6.1) involves no subtractive cancellation; so long as we can compute the scalar logarithm and the argument θ accurately we will obtain $\log(B)$ to high componentwise accuracy.

ALGORITHM 6.1. *Given $A \in \mathbb{R}^{n \times n}$ with no eigenvalues on \mathbb{R}^- this algorithm computes the principal matrix logarithm $X = \log(A)$ and one or more Fréchet derivatives $L_{\log}(A, E)$ and $L_{\log}^*(A, E)$, where $E \in \mathbb{R}^{n \times n}$, using only real arithmetic.*

- 1 Compute a real Schur decomposition $A = QTQ^T$.
- 2 $T_0 = T$
- 3 Determine the integers s and $m \in [1, 7]$ as in [5, Alg. 4.1], at the same time computing (and storing, if Fréchet derivatives are required) the matrices $T_{k+1} = T_k^{1/2}$, $k = 0: s - 1$ using the recurrence of [21], [24, Alg. 6.7].
- 4 $R = T_s - I$
- 5 Replace the diagonal blocks of R by the diagonal blocks of $T_0^{1/2^s} - I$ computed by [1, Alg. 2] for the 1×1 blocks and [5, Alg. 5.1] (with square roots computed by [24, (6.9)]) for the 2×2 blocks.
- 6 For every i for which t_{ii} and $t_{i+1, i+1}$ are in 1×1 diagonal blocks, recompute $r_{i, i+1}$ using [26, (5.6)].
- 7 Evaluate $X = 2^s r_m(R)$ as in lines 6–11 of Algorithm 5.1.
- 8 Recompute the block diagonal of X using (6.1) for the 2×2 blocks of T_0 and as $\log((T_0)_{ii})$ for the 1×1 blocks.
- 9 For every i for which x_{ii} and $x_{i+1, i+1}$ are in 1×1 diagonal blocks, recompute $x_{i, i+1}$ using [24, (11.28)].
- 10 $X \leftarrow QXQ^T$
- 11 ... To compute $L_{\log}(A, E)$ or $L_{\log}^*(A, E)$ for a given E , execute lines 15–24 or line 26 of Algorithm 5.1.

The cost of this algorithm is essentially the same as Algorithm 5.1, except that the flops are real flops instead of complex ones.

7. Comparison with existing methods. In this section we give a comparison between our algorithms and those currently in the literature for computing the matrix logarithm and its Fréchet derivatives, concentrating mainly on computational cost. Section 8 contains a variety of numerical experiments comparing the accuracy of the algorithms empirically.

7.1. Methods to compute the logarithm. We have obtained a new algorithm for computing the logarithm of a real matrix (Algorithm 6.1). The relevant comparison is between the following three methods.

- `iss_schur_complex`: the complex Schur decomposition-based Algorithm 5.1, which is equivalent to [5, Alg. 4.1] when just the logarithm is required.
- `iss_schur_real`: our real Schur decomposition-based Algorithm 6.1, which is the real analogue of `iss_schur_complex`.
- `iss_noschur`: the transformation-free inverse scaling and squaring algorithm [5, Alg. 5.2] that requires only matrix multiplications and the solution of multiple-right-hand side linear systems. This algorithm uses only real arithmetic when A is real.

In general, `iss_noschur` is much more expensive than `iss_schur_real`. For example, if $s = 3$ and $m = 7$ in both `iss_schur_real` and `iss_noschur` and if `iss_noschur` requires five iterations to compute each square root (for which it uses a Newton iteration) then the operation counts are approximately $31n^3$ flops for `iss_schur_real` and $79n^3$ flops for `iss_noschur`.

7.2. Methods to compute the Fréchet derivative. We now compare our algorithms `iss_schur_complex` and `iss_schur_real` to existing methods for computing the Fréchet derivative.

Kenney and Laub give an algorithm based on a special Kronecker representation of L_{\log} that solves Sylvester equations and employs a Padé approximant to the function $\tanh(x)/x$ [31], [24, Alg. 11.12]. We will refer to this as the Kronecker–Sylvester algorithm. The minimum cost of this algorithm per Fréchet derivative, assuming a Schur decomposition is used, is the cost of solving $s + 9$ triangular Sylvester equations and computing 16 products of a triangular matrix with a full matrix. This is to be compared with a smaller *maximum* cost for `iss_schur_complex` and `iss_schur_real` of s triangular Sylvester equations and $2m$ (≤ 14) multiple-right-hand side triangular substitutions. Moreover, the value of s for `iss_schur_complex` and `iss_schur_real` is potentially much smaller than for the Kronecker–Sylvester algorithm, and the latter always requires complex arithmetic, even when A and E are real.

Another method, which we denote by `dbl_size`, evaluates the left-hand side of the formula

$$\log \left(\begin{bmatrix} A & E \\ 0 & A \end{bmatrix} \right) = \begin{bmatrix} \log(A) & L_{\log}(A, E) \\ 0 & \log(A) \end{bmatrix}, \quad (7.1)$$

from [24, (3.16)], by `iss_schur_real` (or `iss_schur_complex` when A or E is complex). This method has the disadvantages that it doubles the problem size and that the entire logarithm of the block matrix must be re-evaluated if we require further Fréchet derivatives, greatly increasing the cost of this method. On the other hand, the backward error analysis of [5] fully applies, giving a sharper backward error bound than (3.5) for the Fréchet derivative. However, backward error is now measured with respect to the matrix $\begin{bmatrix} A & E \\ 0 & A \end{bmatrix}$ rather than A and E separately. Moreover, it is unclear how to scale E : its norm is arbitrary because L_{\log} is linear in its second argument, but the size of $\|E\|$ affects both the accuracy and the cost of the inverse scaling and squaring method. In the absence of a better approach, we have left E unscaled in our tests.

We mention two other methods. Dieci, Morini, and Papini [15] propose using the inverse scaling and squaring approach with adaptive Simpson’s rule applied to the integral $L_{\log}(A, E) = \int_0^1 ((A - I)t + I)^{-1} E ((A - I)t + I)^{-1} dt$. Since we are interested

in computing L_{\log} to full precision the tolerance for the quadrature rule needs to be set to order u , and this makes the method prohibitively expensive, as each function evaluation requires two multiple-right-hand side solves. As noted in [15], this method is more appropriate when only low accuracy is required, so we will test it only for condition number estimation.

We also mention the complex step approximation $L_{\log}(A, E) \approx \text{Im} \log(A + ihE)/h$ suggested by Al-Mohy and Higham [4], which is valid for real A and E and has error $O(h^2)$; unlike finite difference approximations it does not suffer from inherent cancellation, so h can be taken very small. Since the argument $A + ihE$ of the logarithm is complex we must use Algorithm 5.1 for the evaluation. As noted in [4], this approximation is likely to suffer from numerical instability if used with an algorithm that intrinsically employs complex arithmetic such as `iss_schur_complex`. This is indeed what we observed, with relative errors of order at best 10^{-7} , so we will not consider this approach further.

8. Numerical experiments. Our numerical experiments are performed in MATLAB R2012a in IEEE double precision arithmetic. Throughout the experiments we use a set of 66 (mostly 10×10) test matrices, extending those used in [5] and [24, sect. 11.7] which include matrices from the literature, the MATLAB `gallery` function, and the Matrix Computation Toolbox [19]. All experiments begin with an initial reduction of A to real Schur form if A is real and complex Schur form otherwise. This is to minimize the impact that errors in forming the real or complex Schur form may have on the computation, and hence to reveal more clearly the differences between the methods.

Our tests are concerned with accuracy, not speed, but we comment on two implementation issues. Square roots of (quasi-) triangular matrices are required in the algorithms, and since this operation is not one of the BLAS [10] it must be coded specially. A straightforward implementation using nested loops does not provide good performance, but a blocked algorithm described by Deadman, Higham, and Ralha [14] yields a much more efficient implementation rich in matrix multiplication. Similarly, triangular Sylvester equations can be solved efficiently using the recursive algorithm of Jonsson and Kågström [29].

8.1. Real versus complex arithmetic for evaluating the logarithm. Our first experiment compares, on the 60 real matrices in the test set, the three algorithms defined in section 7.1. The matrices are used in real Schur form. We compute all relative errors in the 1-norm and for our “exact” logarithm we diagonalize A in 250-digit precision, using the Symbolic Math Toolbox, and use the relationship $\log(A) = V \log(D) V^{-1}$ where $A = V D V^{-1}$, rounding the result to double precision. If A is not diagonalizable then we add a small random perturbation of order 10^{-125} so that with high probability we can diagonalize it without affecting the accuracy of the final rounded solution [13].

Figure 8.1 shows the normwise relative errors, with the problems ordered by decreasing condition number. The solid line denotes $\text{cond}(\log, A)u$ (with $\|L_{\log}(A)\|_1$ approximated by $\|K_{\log}(A)\|_1$). Figure 8.2 shows the same data in the form of a performance profile [17], [18, sect. 22.4], for which we use the transformation in [16] to lessen the influence of tiny relative errors.

The results show that `iss_schur_complex` and `iss_schur_real` are both significantly more accurate than `iss_noschur` (as also shown in [5] for `iss_schur_complex`) and that `iss_noschur` is often a little unstable (by which we mean errors exceed

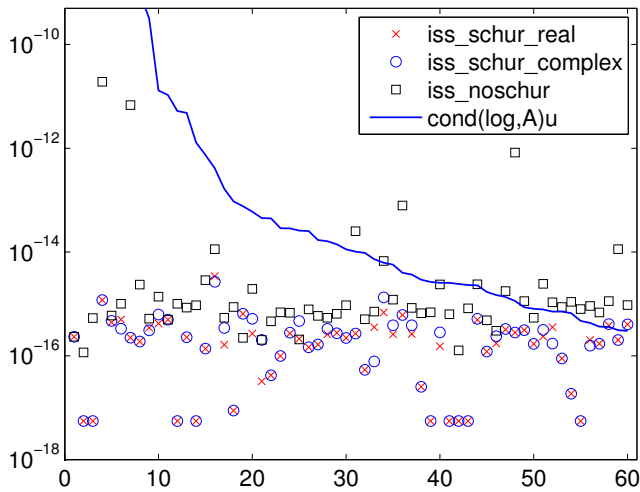


FIG. 8.1. Normwise relative errors in computing the logarithm.

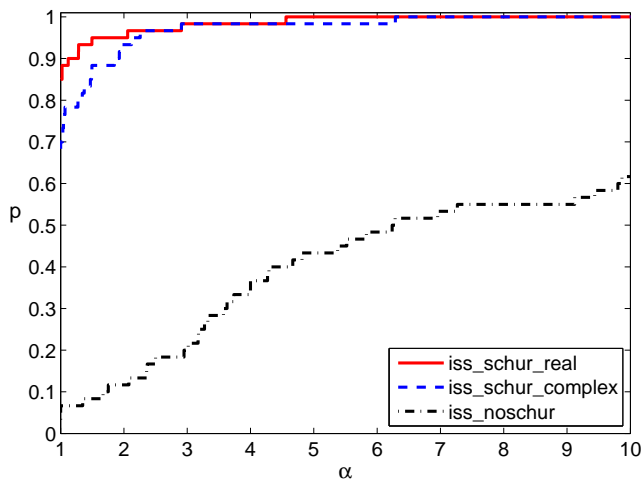


FIG. 8.2. Performance profile for the data in Figure 8.1.

$n \text{cond}(\log, A)u$, say). Moreover, `iss_schur_real` outperforms `iss_schur_complex`, showing that treating real matrices in real arithmetic benefits accuracy.

We repeated the experiments on full matrices and found that `iss_schur_real` is again the most accurate algorithm overall but that `iss_noschur` is now much more competitive with `iss_schur_real` and `iss_schur_complex`. The performance profile is given in Figure 8.3. The reason for the improved relative performance of `iss_noschur` is that rounding errors in the reduction to Schur form in `iss_schur_complex` and `iss_schur_real` tend to lead to larger errors for these algorithms than in the quasitriangular case above.

8.2. Fréchet derivative evaluation. We now test our algorithms against the alternatives mentioned in section 7.2 for computing Fréchet derivatives. We define `iss_schur` to be the algorithm that invokes Algorithm 6.1 when A is real and otherwise

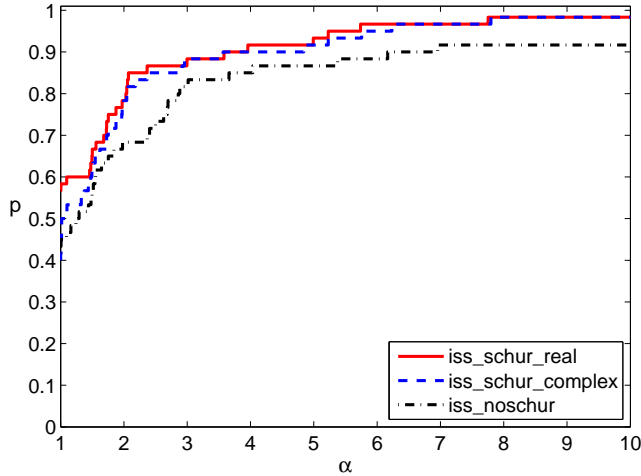


FIG. 8.3. Performance profile for the accuracy of the logarithm algorithms on full matrices.

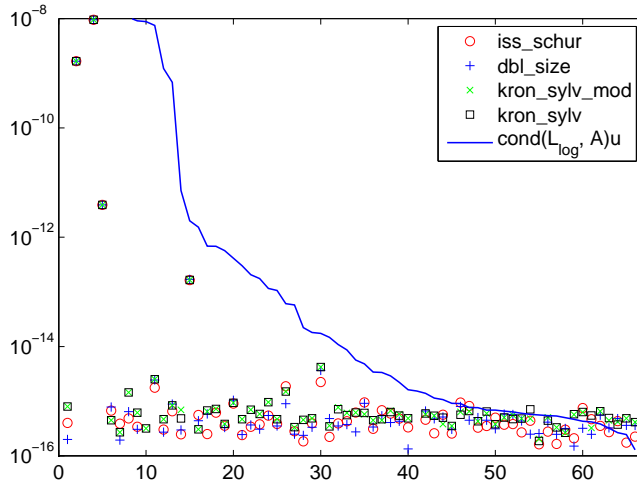


FIG. 8.4. Normwise relative errors in computing $L_{\log}(A, E)$.

invokes Algorithm 5.1. In each test we take for E , the direction in which to calculate the Fréchet derivative, a random real matrix with normal $(0, 1)$ distributed elements.

In other experiments not reported here we took E to be a matrix such that $\|A^{2m+1}E\|_1 = \|A^{2m+1}\|_1\|E\|_1$ (cf. Lemma 3.4) in an attempt to minimize the sharpness of the upper bound in (3.5). We obtained similar results to those presented.

Figure 8.4 shows the relative errors of the Fréchet derivatives computed with the methods described in section 7.2. Here, `kron_sylv` denotes the implementation of the algorithm of Kenney and Laub [31], [24, Alg. 11.12] in the Matrix Function Toolbox [20] (named `logm_frechet_pade` there), and `kron_sylv_mod` denotes a modification of it in which calls to `logm` are replaced by calls to the more accurate `iss_schur_complex`. The values are ordered by descending condition number $\text{cond}(L_{\log}, A)$, where, analogously to [2], we estimate $\text{cond}(L_{\log}, A)$ with finite differences, by taking (1.2) with $f \leftarrow L_{\log}$ and a random direction E of norm 10^{-8} , employing the Kronecker form of

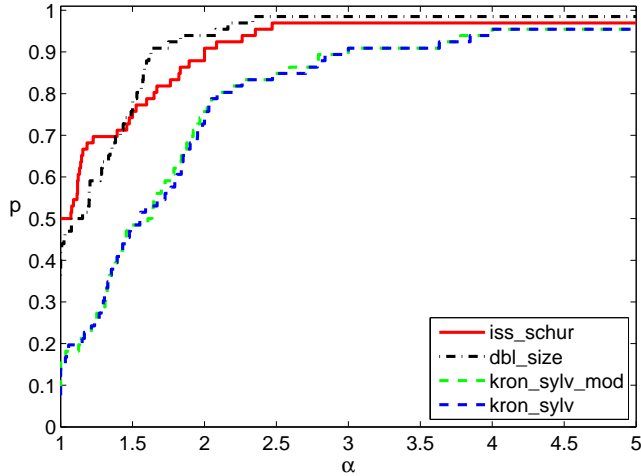


FIG. 8.5. Performance profile for the data in Figure 8.4.

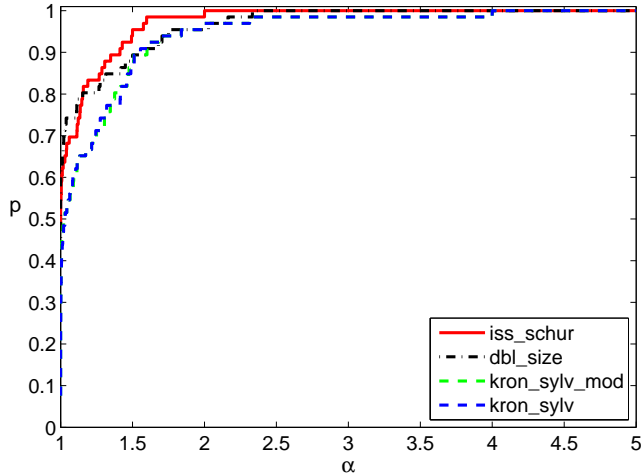


FIG. 8.6. Performance profile for the accuracy of the $L_{\log}(A, E)$ algorithms for full matrices.

the derivative. In Figure 8.5 we present the same data as a performance profile.

The relative errors in Figure 8.4 show that the algorithms all performed very stably. We see from Figure 8.5 that the two most accurate methods for Fréchet derivative computation are `iss_schur` and `dbl_size`. Figure 8.5 also shows that using the more accurate logarithm evaluation in `kron_sylv_mod` produces a slight improvement in accuracy over `kron_sylv`.

Testing with full matrices we obtain similar results, although most relative errors are now within an order of magnitude of $\text{cond}(L_{\log}, A)$. The associated performance profile is shown in Figure 8.6. The much tighter grouping of the algorithms is again due to the relative errors introduced by the Schur reduction.

8.3. Condition number estimation. We now consider the estimation of $\|K_{\log}(A)\|_1$, which is the key quantity needed to estimate $\text{cond}(\log, A)$ (see (4.1)). We obtain the exact value by explicitly computing the $n^2 \times n^2$ matrix $K_{\log}(A)$ via

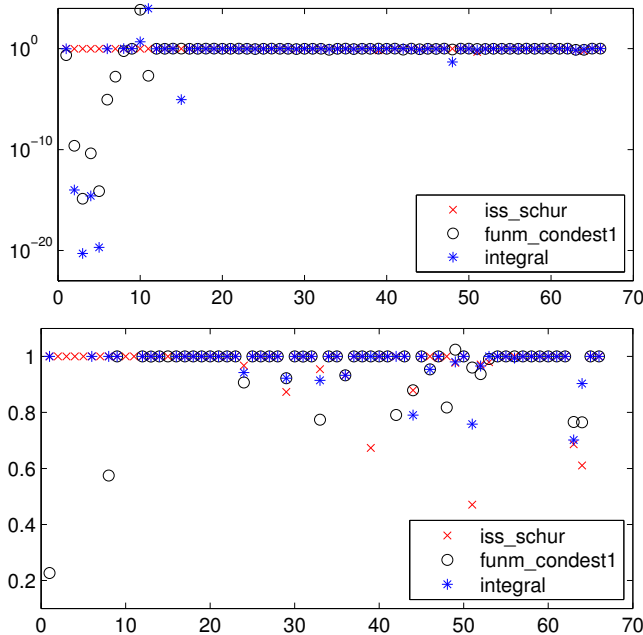


FIG. 8.7. Underestimation ratios $\eta/\|K(A)\|_1$ where η is the estimate of $\|K(A)\|_1$; lower plot is zoomed version of upper plot.

[24, Alg. 3.17] and taking its 1-norm.

We use Algorithm 4.1 with three different methods for computing the Fréchet derivatives. The first is our new algorithm, `iss_schur` (again denoting the use of Algorithm 5.1 or Algorithm 6.1 as appropriate). The second is a general purpose method based on finite differences, as implemented in the code `funm_condest1` from the Matrix Function Toolbox [20]. The last method, denoted `integral`, is the quadrature method of [15] described in section 7.2, used with quadrature tolerance 10^{-10} . We do not test `dbl_size` or `kron_sylv` within Algorithm 4.1, as these algorithms are substantially more expensive than `iss_schur` (see section 7) and no more accurate (as shown in the previous subsection).

In Figure 8.7 we plot the ratios of the estimated condition numbers to the accurately computed ones, sorted by decreasing condition number. These underestimation ratios should be less than or equal to 1 as we use the 1-norm power method to estimate $\|K(A)\|_1$. From these ratios we see that while the `funm_condest1` and `integral` methods provide good estimates for most of the well conditioned problems, they produce estimates many orders of magnitude too small for some of the more ill conditioned problems. On the other hand, `iss_schur` gives excellent estimates that are at worst a factor 0.47 smaller than the true value, making it the clear winner.

The unreliability of `funm_condest1` can be attributed to the presence of the potentially very large term $\|\log(A)\|_1$ in the steplength formula used for the finite differences [24, (3.23)]. It is not clear whether a different choice of finite different steplength leading to better results can be derived.

Estimating the condition number using Algorithm 4.1 with $t = 2$, as in our algorithms, requires around 8 Fréchet derivative evaluations. With the mean s and m found to be 4 and 6 respectively in our tests, the cost of computing the logarithm and

its condition number via our algorithms is around 8 times that of the logarithm alone. By reducing t in the block 1-norm estimation algorithm to 1, this can be lowered to 4 times the cost at the risk of lower reliability.

9. Conclusions. We have extended the complex Schur form-based inverse scaling and squaring algorithm of Al-Mohy and Higham [5] for computing the matrix logarithm in two ways. First, Algorithm 6.1 extends the algorithm to work entirely in real arithmetic for real matrices. It has the obvious advantages of reduced arithmetic cost and reduced intermediate storage, whilst also yielding generally more accurate results.

Second, Algorithm 5.1 extends the algorithm of [5] to compute one or more Fréchet derivatives after computing $\log(A)$, with reuse of information, while Algorithm 6.1 does the same but working in real arithmetic for real data. We have shown that the new algorithms for $L_{\log}(A, E)$ are significantly less expensive than existing algorithms (see section 7) and are also more accurate in practice (see section 8).

The fact that our choice of the algorithmic parameters m and s is based on $\alpha_p(A)$, while our backward error bounds for the Fréchet derivative involve the potentially much larger quantity $\|A\|$, does not appear to affect the accuracy of the Fréchet derivative computations: in our experiments the Fréchet derivatives were computed in a forward stable way throughout.

By combining the new algorithms with the block 1-norm estimation algorithm of Higham and Tisseur [27] reliable condition estimates are obtained, whereas we have shown that a general purpose $\text{cond}(A, f)$ estimate based on finite differences can greatly underestimate the condition number.

REFERENCES

- [1] AWAD H. AL-MOHY, *A more accurate Briggs method for the logarithm*, Numer. Algorithms, 59 (2012), pp. 393–402.
- [2] AWAD H. AL-MOHY AND NICHOLAS 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.
- [3] ———, *A new scaling and squaring algorithm for the matrix exponential*, SIAM J. Matrix Anal. Appl., 31 (2009), pp. 970–989.
- [4] ———, *The complex step approximation to the Fréchet derivative of a matrix function*, Numer. Algorithms, 53 (2010), pp. 133–148.
- [5] ———, *Improved inverse scaling and squaring algorithms for the matrix logarithm*, SIAM J. Sci. Comput., 34 (2012), pp. C152–C169.
- [6] DAVID AMSALLEM AND CHARBEL FARHAT, *An online method for interpolating linear parametric reduced-order models*, SIAM J. Sci. Comput., 33 (2011), pp. 2169–2198.
- [7] E. ANDERSON, Z. BAI, C. H. BISCHOF, S. BLACKFORD, J. W. DEMMEL, J. J. DONGARRA, J. J. DU CROZ, A. GREENBAUM, S. J. HAMMARLING, A. MCKENNEY, AND D. C. SORENSEN, *LAPACK Users' Guide*, Society for Industrial and Applied Mathematics, Philadelphia, PA, USA, third ed., 1999.
- [8] VINCENT ARSIGNY, OLIVIER COMMOWICK, NICHOLAS AYACHE, AND XAVIER PENNEC, *A fast and log-Euclidean polyaffine framework for locally linear registration*, J. Math. Imaging Vis., 33 (2009), pp. 222–238.
- [9] ÅKE BJÖRCK AND SVEN HAMMARLING, *A Schur method for the square root of a matrix*, Linear Algebra Appl., 52/53 (1983), pp. 127–140.
- [10] L. SUSAN BLACKFORD, JAMES DEMMEL, JACK DONGARRA, IAIN DUFF, SVEN HAMMARLING, GREG HENRY, MICHAEL HEROUX, LINDA KAUFMAN, ANDREW LUMSDAINE, ANTOINE PETITET, ROLDAN POZO, KARIN REMINGTON, AND R. CLINT WHALEY, *An updated set of Basic Linear Algebra Subprograms (BLAS)*, ACM Trans. Math. Software, 28 (2002), pp. 135–151.
- [11] JOÃO R. CARDOSO AND F. SILVA LEITE, *Theoretical and numerical considerations about Padé approximants for the matrix logarithm*, Linear Algebra Appl., 330 (2001), pp. 31–42.
- [12] 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 (2001), pp. 1112–1125.
- [13] E. B. DAVIES, *Approximate diagonalization*, SIAM J. Matrix Anal. Appl., 29 (2007), pp. 1051–1064.
- [14] EDVIN DEADMAN, NICHOLAS J. HIGHAM, AND RUI RALHA, *A recursive blocked Schur algorithm for computing the matrix square root*, MIMS EPrint 2012.26, Manchester Institute for Mathematical Sciences, The University of Manchester, UK, Jan. 2012.
- [15] LUCA DIECI, BENEDETTA MORINI, AND ALESSANDRA PAPINI, *Computational techniques for real logarithms of matrices*, SIAM J. Matrix Anal. Appl., 17 (1996), pp. 570–593.
- [16] 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, Nov. 2011.
- [17] ELIZABETH D. DOLAN AND JORGE J. MORÉ, *Benchmarking optimization software with performance profiles*, Math. Programming, 91 (2002), pp. 201–213.
- [18] DESMOND J. HIGHAM AND NICHOLAS J. HIGHAM, *MATLAB Guide*, Society for Industrial and Applied Mathematics, Philadelphia, PA, USA, second ed., 2005.
- [19] NICHOLAS J. HIGHAM, *The Matrix Computation Toolbox*. <http://www.ma.man.ac.uk/~higham/mctoolbox>.
- [20] ———, *The Matrix Function Toolbox*. <http://www.ma.man.ac.uk/~higham/mftoolbox>.
- [21] ———, *Computing real square roots of a real matrix*, Linear Algebra Appl., 88/89 (1987), pp. 405–430.
- [22] ———, *Evaluating Padé approximants of the matrix logarithm*, SIAM J. Matrix Anal. Appl., 22 (2001), pp. 1126–1135.
- [23] ———, *Accuracy and Stability of Numerical Algorithms*, Society for Industrial and Applied Mathematics, Philadelphia, PA, USA, second ed., 2002.
- [24] ———, *Functions of Matrices: Theory and Computation*, Society for Industrial and Applied Mathematics, Philadelphia, PA, USA, 2008.
- [25] NICHOLAS J. HIGHAM AND AWAD H. AL-MOHY, *Computing matrix functions*, Acta Numerica, 19 (2010), pp. 159–208.
- [26] NICHOLAS J. HIGHAM AND LIJING LIN, *A Schur-Padé algorithm for fractional powers of a matrix*, SIAM J. Matrix Anal. Appl., 32 (2011), pp. 1056–1078.

- [27] 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 (2000), pp. 1185–1201.
- [28] WEIMING HU, HAIQIANG ZUO, OU WU, YUNFEI CHEN, ZHONGFEI ZHANG, AND DAVID SUTER, *Recognition of adult images, videos, and web page bags*, ACM Trans. Multimedia Comput. Commun. Appl., 78 (2011), pp. 28:1–28:24.
- [29] ISAK JONSSON AND BO KÄGSTRÖM, *Recursive blocked algorithms for solving triangular systems—Part I: One-sided and coupled Sylvester-type matrix equations*, ACM Trans. Math. Software, 28 (2002).
- [30] CHARLES S. KENNEY AND ALAN J. LAUB, *Condition estimates for matrix functions*, SIAM J. Matrix Anal. Appl., 10 (1989), pp. 191–209.
- [31] ———, *A Schur–Fréchet algorithm for computing the logarithm and exponential of a matrix*, SIAM J. Matrix Anal. Appl., 19 (1998), pp. 640–663.
- [32] DANIEL PETERSSON AND JOHAN LÖFBERG, *Model reduction using a frequency-limited \mathcal{H}_2 -cost*, tech. report, Department of Electrical Engineering, Linköpings Universitet, SE-581 83, Linköping, Sweden, Mar. 2012.
- [33] JAREK ROSSIGNAC AND ÁLVAR VINACUA, *Steady affine motions and morphs*, ACM Trans. Graphics, 30 (2011), pp. 116:1–116:16.