The dual inverse scaling and squaring algorithm for the matrix logarithm

Fasi, Massimiliano and Iannazzo, Bruno

2020

MIMS EPrint: **2020.14**

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

The dual inverse scaling and squaring algorithm for the matrix logarithm*

Massimiliano Fasi† Bruno Iannazzo‡

The inverse scaling and squaring algorithm computes the logarithm of a square matrix A by evaluating a rational approximant to the logarithm at the matrix $B := A^{2^{-s}}$ for a suitable choice of s. We introduce a dual approach and approximate the logarithm of B by solving the rational equation $r(X) = B$, where r is a diagonal Padé approximant to the matrix exponential at 0. This equation is solved by a substitution technique in the style of those developed in (Fasi & Iannazzo, *Elect. Trans. Num. Anal.*, 53 (2020), pp. 500–521). The new method is tailored to the special structure of the diagonal Padé approximants to the exponential, and in terms of computational cost is more efficient than the state-of-the-art inverse scaling and squaring algorithm. Primary matrix function, matrix logarithm, inverse scaling and squaring algorithm, Schur form, Padé approximant.

1 Introduction

Any matrix $X \in \mathbb{C}^{N\times N}$ satisfying the equation

$$\exp(X) = A \tag{1}$$

is a logarithm of $A \in \mathbb{C}^{N\times N}$. It can be shown that (1) has infinitely many solutions if A is nonsingular and no solution otherwise. When A has no nonpositive real eigenvalues, however, there exists a unique matrix that satisfies (1) and has spectrum in the

*Version of 28/04/2021. The work of the first author was supported by MathWorks, the Royal Society, and the Wenner-Gren Foundations grant UPD2019-0067. The work of the second author was supported by the Istituto Nazionale di Alta Matematica, INdAM–GNCS Project 2019. The opinions and views expressed in this publication are those of the authors, and not necessarily those of the funding bodies.

†School of Science and Technology, Örebro University, 702 81 Örebro, Sweden (massimiliano.fasi@oru.se).

‡Dipartimento di Matematica e Informatica, Università di Perugia, Via Vanvitelli 1, 06123 Perugia, Italy (bruno.iannazzo@unipg.it).

complex strip $\{z \in \mathbb{C} : |\mathrm{Im}(z)| < \pi\}$. This solution, denoted by $\log(A)$, is the *principal logarithm* of A (also called *standard branch of the logarithm* of A by Tao [2015]). With some abuse of notation, we can define the principal logarithm of nonsingular matrices with eigenvalues on the open negative real axis as the unique solution to (1) whose eigenvalues lie in the strip $\{z \in \mathbb{C} : -\pi < \mathrm{Im}(z) \leq \pi\}$.

The principal logarithm has applications in a wide variety of domains. In engineering, it can be employed to recover the coefficient matrix of a system governed by the linear differential equation $dy/dt = Xy$ from observations of the state vector y [Kenney and Laub, 1998]. In control theory, it is used to convert discrete-time linear dynamical systems to continuous-time state-space systems [Lastman and Sinha, 1991] and to compute the time-invariant component of the state transition matrix of ordinary differential equations with periodic time-varying coefficients [Grant Kirkland and Sinha, 2016]. More recent applications include the evaluation of the Lambert W function [Fasi et al., 2015] and of the matrix geometric mean [Iannazzo and Porcelli, 2018], and the computation of generators of Markov chains in finance [Israel et al., 2001]. The matrix logarithm is also needed in optics [Ossikovski and De Martino, 2015], mechanics [Ramézani and Jeong, 2015], and computer graphics [Rossignac and Vinacua, 2011].

The principal logarithm can be defined as a primary matrix function [Higham, 2008, Chap. 11]. If A is diagonalizable, i.e., there exists a nonsingular matrix M such that $M^{-1}AM = \mathrm{diag}(\lambda_1, \ldots, \lambda_N)$, then

$$\log(A) = M \,\mathrm{diag}(\log(\lambda_1), \ldots, \log(\lambda_N))M^{-1}. \tag{2}$$

If A is nondiagonalizable, $\log(A)$ can be obtained from (2) by continuity [Horn and Johnson, 1991, Chap. 6], but extra care is necessary for matrices with real negative eigenvalues.

The formula (2) readily translates into an algorithm for computing the principal logarithm of a diagonalizable matrix. Such an algorithm turns out to be unstable, as an ill-conditioned M may lead to a forward error far larger than that predicted by the conditioning of the matrix logarithm itself. In view of this limitation, in practice (2) is the algorithm of choice only when A is normal, as in this case the matrix M can be chosen to be unitary and thus perfectly conditioned.

Several diverse techniques for approximating numerically the principal logarithm of nonnormal matrices have been proposed in the literature, see [Higham, 2008, Chap. 11] for a survey and [Cardoso and Ralha, 2016] and [Tatsuoka et al., 2020] for more recent examples. A well-established approach is the inverse scaling and squaring algorithm, a method initially proposed by Kenney and Laub [1989] and further developed by several authors over the course of the last thirty years [Al-Mohy and Higham, 2012, Al-Mohy et al., 2013, Cheng et al., 2001, Dieci et al., 1996, Fasi and Higham, 2018, **?**]. This technique exploits the matrix identity $\log(A) = 2^s \log(A)^{2^{-s}}$, and evaluates $\log(A)$ by combining argument reduction and rational approximation. By taking a certain number of square roots of A, the problem is reduced to the approximation of the logarithm of a matrix with eigenvalues close to 1. The latter task is accomplished by evaluating the rational function t_m at the matrix argument $B := A^{2^{-s}}$, where t_m is a

diagonal Padé approximant to the scalar function $\log(z)$ at $z = 1$. Scaling the result to take into account the effect of the initial square roots leads to the approximation $\log(A) \approx 2^s t_m(A^{2^{-s}})$.

Here we follow a dual approach, and rather than evaluating a rational function we approximate the logarithm of $A^{2^{-s}}$ by solving the rational equation

$$r(X) = A^{2^{-s}},$$

where $r(z)$ is a rational approximant to e^z at $z = 0$. This technique may seem unnecessarily convoluted, as in most cases evaluating a function f at a matrix A requires less effort than solving the matrix equation $f(X) = A$. As we shall see, this is not the case for the matrix function at hand. In previous work [Fasi and Iannazzo, 2019, 2020] we have shown that if T is an upper triangular matrix and r is a rational function, then computing an upper triangular solution Y to $r(Y) = T$ has the same asymptotic computational cost as evaluating $r(T)$. The focus on triangular matrices is not a restriction: if a triangular matrix Y satisfies $r(Y) = T$, where $T = U^*AU$ is upper triangular and U is unitary, then the matrix $X = UYU^*$ is a solution to the matrix equation $r(X) = A$, and conversely, any solution to $r(X) = A$ that is a polynomial of A can be obtained in this way. A similar argument applies when T is real and upper quasi-triangular, that is, block upper triangular with diagonal blocks of size 1, corresponding to real eigenvalues, or 2, corresponding to a pair of complex conjugate eigenvalues.

The advantage of the dual approach for the matrix logarithm lies in the fact that the $[2\ell + 1, 2\ell + 1]$ Padé approximant to e^z at $z = 0$ can be written as [Gautschi, 2011, Thm. 5.9.1]

$$r(z) = \frac{g(z^2) + zh(z^2)}{g(z^2) - zh(z^2)}, \tag{3}$$

where g and h are polynomials of degree ℓ. The diagonal Padé approximants to $\log(z)$ at $z = 1$, on which the inverse scaling and squaring algorithm is based, do not have such a special form.

In the next section we recall some key definitions and results that will be used later on. In Section 3 we briefly review existing substitution algorithms for rational equations and introduce our new algorithm for rational functions that can be written in the form (3). In Section 6 we discuss how this technique can be used to compute the matrix logarithm in an inverse scaling and squaring fashion, and evaluate the performance of the resulting algorithm in Section 8. The last section summarizes our contribution and outlines possible directions for future work.

2 Background

2.1 Characterizing primary and isolated solutions to matrix equations

Let us consider the matrix equation $f(X) = A$ where $X, A \in \mathbb{C}^{N\times N}$ and f is a primary matrix function in the sense of [Higham, 2008, Chap. 1]. We say that $\widehat{X}$ such that $f(\widehat{X}) = A$ is a *primary* solution if there exists a polynomial p such that $\widehat{X} = p(A)$, and

that it is *isolated* if there exists a neighborhood $\mathcal{U} \subset \mathbb{C}^{N\times N}$ of $\widehat{X}$ where $\widehat{X}$ is the only matrix that satisfies $f(X) = A$.

Primary solutions can be characterized in terms of their spectrum. We recall that an eigenvalue is semisimple if it appears only in Jordan blocks of size 1, and that a semisimple eigenvalue is simple if it has algebraic multiplicity 1. It can be shown [Evard and Uhlig, 1992, Thm. 6.1] that a solution X is primary if and only if the following two conditions are satisfied:

1. $f(\xi_i) \neq f(\xi_j)$ for any two distinct eigenvalues ξ_i, ξ_j of X;
2. if an eigenvalue ξ of X is critical ($f'(\xi) = 0$), then it is semisimple.

Furthermore, a primary solution is isolated if and only if all its critical eigenvalues are simple [Fasi and Iannazzo, 2019, Thm. 6].

2.2 Padé approximation of the exponential function

Rational approximation is a powerful tool for the computation of matrix functions. Here we recall the definition of Padé approximants and state some of their fundamental properties. Readers interested in theoretical aspects of Padé approximation are referred to the encyclopedic work by Baker and Graves-Morris [1996].

Let $f : \Omega \to \mathbb{C}$ be analytic on $U \subset \Omega$ and let $z_0 \in U$. The rational function $r_{mn}(z) := p_{mn}(z)q_{mn}(z)^{-1}$, where

$$p_{mn}(z) := \sum_{k=0}^{m} c_k^{[m/n]} z^k, \qquad q_{mn}(z) := \sum_{k=0}^{n} d_k^{[m/n]} z^k,$$

is an $[m/n]$ Padé approximant to $f(z)$ at $z = z_0$ if the denominator is normalized so that $q_{mn}(z_0) = 1$ and $f(z) - r_{mn}(z) = O\big((z - z_0)^{m+n+1}\big)$ as $z \to z_0$. Given f, m, and n, an $[m/n]$ Padé approximation might not exist, but if it does then it is unique. If we further require that p_{mn} and q_{mn} are coprime, then p_{mn} and q_{mn} are also unique.

In order to develop a new algorithm for the matrix logarithm, we consider the Padé approximants to e^z at $z = 0$, which we denote by $\widetilde{r}_{mn}(z) := \widetilde{p}_{mn}(z)\widetilde{q}_{mn}(z)^{-1}$. These approximants exist for any choice of m and n, and the coefficients of numerator and denominator of $\widetilde{r}_{mn}$ are [Gautschi, 2011, Thm. 5.9.1]

$$\begin{aligned} \widetilde{c}_k^{[m/n]} &= \binom{m}{k} \frac{(m+n-k)!}{(m+n)!}, && \text{for } k = 0, \dots, m, \\ \widetilde{d}_k^{[m/n]} &= (-1)^k \binom{n}{k} \frac{(m+n-k)!}{(m+n)!}, && \text{for } k = 0, \dots, n, \end{aligned} \tag{4}$$

respectively. Here we focus in particular on diagonal approximants, for which $m = n$, as our algorithm exploits the property $c_k^{[m/m]} = (-1)^k d_k^{[m/m]}$, which in turn implies that $\widetilde{p}_{mm}(z) = \widetilde{q}_{mm}(-z)$ and that $\widetilde{r}_m(z) := \widetilde{r}_{mm}(z)$ has the special structure (3).

As is customary in the literature of matrix functions, we restrict our attention to approximants that, from a computational point of view, are optimal, in the sense that

they are the approximants of highest degree that can be evaluated with a fixed number of matrix multiplications. Since in our algorithms we do not use $\widetilde{r}_2$, which is the only optimal Padé approximant to the matrix exponential of even degree [Fasi, 2019, Prop. 4], in Section 3 we will consider only odd values of m.

3 Specialized substitution algorithms

In prior work, we have introduced a class of algorithms, based on a substitution procedure, for the solution of the matrix equation

$$p(X)q^{-1}(X) = A, \tag{5}$$

where p and q are coprime polynomials. Section 3.1 is devoted to a review of these methods. Our new contribution is outlined in Section 3.2, where we introduce a specialized substitution algorithm that takes full advantage of the special structure (3) of the diagonal Padé approximants to the exponential.

The algorithms in Section 3.1 and Section 3.2 are designed to solve equation (6) below rather than (5). The former equation may appear simpler, but in fact the two formulations are equivalent: equation (5) can be reduced to a polynomial equation of the form $p(X) = Aq(X)$, and the matrices A and X can be assumed to be block upper triangular without loss of generality. As shown in [Fasi and Iannazzo, 2019, Prop. 9], if p and q are coprime then X satisfies (5) if and only if it satisfies $p(X) = Aq(X)$, and it is beneficial to consider only the latter equation, which does not feature matrix inversions. By noting that $p(U^{-1}XU) = U^{-1}AUq(U^{-1}XU)$, we can further reduce (5) to

$$p(Y) = Tq(Y), \tag{6}$$

where

$$T = U^{-1}AU \tag{7}$$

is block upper triangular with ν blocks of size $\tau_1, \dots, \tau_\nu$ along the main diagonal. Note that if Y is a primary solution to (6), then it has the same block structure as T.

Even though our algorithm would in principle work with any block upper triangular structure, in practice it suffices to consider two cases: upper triangular matrices, which have diagonal blocks of order 1, and upper quasi-triangular matrices, which can have diagonal blocks of order 1 or 2. More specifically, we focus on two (block) triangularizations: the Schur decomposition $A =: UTU^*$, where $T, U \in \mathbb{C}^{N\times N}$ are upper triangular and unitary, respectively; and the real Schur decomposition $A := QSQ^T$, where $S, Q \in \mathbb{R}^{N\times N}$ are upper quasi-triangular and orthogonal, respectively. We stress that this choice is not restrictive, as all square complex matrices have a Schur decomposition, and all square real matrices have also a real Schur decomposition.

In order to exploit the structure of the diagonal Padé approximants to the exponential

function, it is convenient to rewrite (6) as

$$g(Y^2)+Yh(Y^2)-Tg(Y^2)+TYh(Y^2) = 0, \qquad g(x) = \sum_{k=0}^{\ell} \gamma_k x^k, \qquad h(x) = \sum_{k=0}^{\ell} \delta_k x^k. \quad (8)$$

Note that since g and h are polynomials of degree ℓ, working with (8) might significantly reduce the computational cost of solving (5).

3.1 Substitution algorithms for (6)

Now we recall a family of methods [Fasi and Iannazzo, 2019, 2020] for solving general equations of the form (6), where

$$T = (T_{ij})_{i,j=1,\dots,\nu} \quad \text{and} \quad Y = (Y_{ij})_{i,j=1,\dots,\nu}$$

are block upper triangular matrices with the same block structure.

The general idea behind these algorithms is to write the equation at block level, and then solve the block equations in a specific order. By rewriting (6), we obtain the equivalent nonlinear system of equations

$$p(Y_{ii}) = T_{ii}q(Y_{ii}), \qquad i = 1, \dots, \nu, \quad (9)$$
$$L_{ij}(Y_{ij}) = b_{ij}, \qquad 1 \le i < j \le \nu, \quad (10)$$

where L_{ij} is a linear operator depending only on Y_{ii} and Y_{jj}, and b_{ij} is a nonlinear polynomial in the blocks $T_{i'j'}$ and $Y_{i'j'}$ such that either $i' = i$ and $j' < j$, or $i' > i$ and $j' = j$.

As an example, consider the matrix equation $Y^2 = T$, whose solutions are square roots of T. This equation can be rewritten as

$$Y_{ii}^2 = T_{ii}, \qquad i = 1, \dots, \nu,$$
$$L_{ij}(Y_{ij}) = b_{ij}, \quad 1 \le i < j \le \nu,$$

where

$$L_{ij}(Y_{ij}) = Y_{ii}Y_{ij} + Y_{ij}Y_{jj} \quad \text{and} \quad b_{ij} = \sum_{\ell=i+1}^{j-1} Y_{i\ell}Y_{\ell j}.$$

Equations (9) and (10) naturally lead to an algorithm for the solution of (6). The diagonal blocks $Y_{11}, \dots, Y_{\nu\nu}$ of the solution can be computed by solving the equations in (9), whereas the blocks above the diagonal can be obtained by solving those in (10), one superdiagonal at a time, with a substitution procedure. In the case of the equation defining the square root, this procedure yields the well-known method of Björck and Hammarling [1983].

In order for this algorithm to be effective, it is necessary to find efficient techniques to compute Y_{ii} and to construct L_{ij} and b_{ij}. When T is upper (quasi-)triangular, the equations in (9) contain only matrices of size 1 or 2, and can be reduced to a scalar

polynomial equation [Fasi and Iannazzo, 2019, Prop. 15]. The matrices in (10), on the other hand, can be computed in a number of ways which, being mathematically but not numerically equivalent, lead to different numerical methods. The algorithms that construct L_{ij} and b_{ij} are related to the techniques used to evaluate the rational function r at a matrix argument. For several evaluation schemes, it has been shown that there exists a substitution algorithm for computing Y that has the same computational cost as the evaluation of r at an upper quasi-triangular matrix.

Existing substitution methods are derived from schemes that evaluate r at the matrix argument A by computing $P := p(A)$ and $Q := q(A)$ separately and then solving the multiple right-hand side linear system $QX = P$. These algorithms differ in the way the numerator and denominator are evaluated: [Fasi and Iannazzo, 2019, Alg. 1] uses Horner's scheme, whereas [Fasi and Iannazzo, 2020, Alg. 1 and Alg. 2] rely on the explicit computation of the powers of A and on the more powerful Paterson–Stockmeyer scheme, respectively.

The applicability of the algorithms depends on the existence of solutions to the equations in (9) and on the nonsingularity of the operators L_{ij} in (10). It has been shown in [Fasi and Iannazzo, 2020, Thm. 3.4] that once the diagonal blocks are computed, the three algorithms are applicable if and only if for $1 \le i < j \le \nu$ one has that $r[\lambda_i, \lambda_j] \neq 0$, where λ_i and λ_j are an eigenvalue of Y_{ii} and Y_{jj}, respectively, and $r[\cdot, \cdot]$ denotes the divided differences of r.

In the case of the equation $Y^2 = T$ which defines the matrix square root, for example, the equation $L_{ij}(Y_{ij}) = b_{ij}$ is a Sylvester equation. Therefore L_{ij} is nonsingular if and only if Y_{ii} and $-Y_{jj}$ have no common eigenvalues, which is guaranteed if in the equations $Y_{ii}^2 = T_{ii}$ and $Y_{jj}^2 = T_{jj}$ the same branch of the square root is chosen. Note that this is equivalent to requiring that $r[\lambda_i, \lambda_j] = \lambda_i + \lambda_j \neq 0$.

In principle, equation (8) could be solved with any of the aforementioned algorithms, but exploiting the special structure of the numerator and denominator of this rational equation can reduce the computational cost of the approach. The idea is to develop a tailored algorithm that uses only the even powers of Y. This approach roughly halves the computational cost of solving (5) with respect to [Fasi and Iannazzo, 2020, Alg. 1], and for $\ell = 1, \dots, 5$ requires fewer matrix multiplications than [Fasi and Iannazzo, 2020, Alg. 2].

For ℓ greater than 5, one could use evaluation schemes that do not require all the powers of Y^2. For any such evaluation scheme, in principle, one could derive a "more complicated" structured algorithm for the solution of $r(Y) = T$ that has roughly the same computational cost as the evaluation of $r(T)$. Such "optimal schemes" are not necessary if one intends to compute the logarithm of a matrix in double precision floating-point arithmetic, as neither accuracy nor performance would improve if approximants of higher degree were used. Larger values may be of interest, or even necessary, in a multiprecision setting, but this aspect will not be discussed here.

3.2 Structured algorithm based on even powers

The rational function

$$r(z) = \frac{g(z^2) + zh(z^2)}{g(z^2) - zh(z^2)},$$

with g and h polynomials of degree ℓ, has a special structure that can be easily exploited. Indeed, $r(z)$ can be evaluated by forming only the even powers of z, saving about a half of the operations with respect to the evaluation of the quotient of two generic polynomials of degree $2\ell + 1$.

The key observation here is that a scheme for evaluating $r(z)$ leads to a substitution algorithm for $r(Y) = T$ that for a (quasi-)triangular matrix T has the same cost as the evaluation of $r(T)$. This allows us to develop a new substitution algorithm for equation (8) which is based only on the even powers of Y and roughly halves the computational cost of solving (8) if compared with [Fasi and Iannazzo, 2020, Alg. 1].

We define thus the sequence of even powers

$$Y^{[k]} = \begin{cases} I, & k = 0, \\ Y^2, & k = 1, \\ Y^{[1]}Y^{[k-1]}, & k = 2, \dots, \ell, \end{cases} \tag{11}$$

where $Y^{[k]} = Y^{2k}$. As before, the diagonal blocks of Y can be computed by solving the equation $r(Y) = T_{ii}$, for $i = 1, \dots, \nu$, while for the off-diagonal blocks we rewrite (8) as an equation involving only Y_{ij} and blocks $g(Y)_{i'j'}$, $h(Y)_{i'j'}$, and $Y^{[k]}_{i'j'}$ such that $i'+j' < i+j$. These blocks are located below and to the left of that in position (i, j), and together with the blocks of T they can be regarded as *known quantities*, implicitly referring to an algorithms that computes the blocks of Y one diagonal at a time from the main diagonal to the top right corner. Unlike the techniques in Section 3.1, such a method does not require constructing odd powers of Y, which reduces the number of operations required to solve (8) by about a half.

From the last of the three equations in (11) we get that the block in position (i, j) of $Y^{[k]}$ can be written as

$$Y^{[k]}_{ij} = Y^{[1]}_{ii}Y^{[k-1]}_{ij} + Y^{[1]}_{ij}Y^{[k-1]}_{jj} + F^{[k]}_{ij}, \qquad F^{[k]}_{ij} := \sum_{t=i+1}^{j-1} Y^{[1]}_{it}Y^{[k-1]}_{tj}. \tag{12}$$

For $k > 2$, by substituting the formula for $Y^{[k-1]}_{ij}$ into that for $Y^{[k]}_{ij}$ one obtains

$$Y^{[k]}_{ij} = \left(Y^{[1]}_{ii}\right)^2 Y^{[k-2]}_{ij} + Y^{[1]}_{ii}Y^{[1]}_{ij}Y^{[k-2]}_{jj} + Y^{[1]}_{ij}Y^{[k-1]}_{jj} + Y^{[1]}_{ii}F^{[k-2]}_{ij} + F^{[k-1]}_{ij},$$

which involves only known quantities and the two blocks $Y^{[k-2]}_{ij}$ and $Y^{[1]}_{ij}$. Repeating

the procedure we obtain, for $k = 2, \dots, \ell$,

$$Y_{ij}^{[k]} = \sum_{u=0}^{k-1} Y_{ii}^{[u]} Y_{ij}^{[1]} Y_{jj}^{[k-u-1]} + \Phi_{ij}^{[k]}, \qquad \Phi_{ij}^{[k]} := \sum_{u=0}^{k-2} Y_{ii}^{[u]} F_{ij}^{[k-u]}, \tag{13}$$

where $Y_{ij}^{[k]}$ is given in terms of $Y_{ij}^{[1]}$ and known quantities. Using (13) and setting $\Phi_{ij}^{[1]} = 0$, we can write, for $k = 1, \dots, \ell$, in a more compact form

$$Y_{ij}^{[k]} = E_{ij}^{[k]}(Y_{ij}^{[1]}) + \Phi_{ij}^{[k]}, \qquad V \xrightarrow{E_{ij}^{[k]}} \sum_{u=0}^{k-1} Y_{ii}^{[u]} V Y_{jj}^{[k-u-1]}. \tag{14}$$

In order to produce a substitution algorithm, we need to obtain a formula in which $Y_{ij}^{[1]}$ does not appear. First, we write the equation for the block in position (i, j) of (8) and rearrange it so that the terms on the right-hand side contain only known quantities, obtaining

$$(I - T_{ii}) g(Y^2)_{ij} + (I + T_{ii}) \big(Y_{ii} h(Y^2)_{ij} + Y_{ij} h(Y^2)_{jj} \big) = -(I + T_{ii}) \sum_{t=i+1}^{j-1} Y_{it} h(Y^2)_{tj} + \sum_{t=i+1}^{j} T_{it} Q_{tj}, \tag{15}$$

where $Q = g(Y^2) - Y h(Y^2)$. In order to isolate the terms containing Y_{ij} in the left-hand side, we first separate the terms containing $Y_{ij}^{[1]}$ and then apply (14), thereby obtaining

$$\begin{aligned} g(Y^2)_{ij} &= \sum_{k=0}^{\ell} \gamma_k Y_{ij}^{[k]} = \sum_{k=1}^{\ell} \gamma_k E_{ij}^{[k]}\left(Y_{ij}^{[1]}\right) + \sum_{k=1}^{\ell} \gamma_k \Phi_{ij}^{[k]}, \\ h(Y^2)_{ij} &= \sum_{k=0}^{\ell} \delta_k Y_{ij}^{[k]} = \sum_{k=1}^{\ell} \delta_k E_{ij}^{[k]}\left(Y_{ij}^{[1]}\right) + \sum_{k=1}^{\ell} \delta_k \Phi_{ij}^{[k]}, \end{aligned} \tag{16}$$

where all the terms other than $Y_{ij}^{[1]}$ are known. In order to obtain an expression where only Y_{ij} and known quantities appear, we note that

$$Y_{ij}^{[1]} = N_{ij}(Y_{ij}) + \widetilde{\Phi}_{ij}, \qquad V \xrightarrow{N_{ij}} Y_{ii} V + V Y_{jj}, \qquad \widetilde{\Phi}_{ij} := \sum_{t=i+1}^{j-1} Y_{it} Y_{tj}, \tag{17}$$

which plugged into (16) and then into (15) yields, after some simple manipulations, the linear equation

$$L_{ij}(Y_{ij}) = b_{ij}, \tag{18}$$

where

$$L_{ij}(Y_{ij}) := \sum_{k=1}^{\ell} \mu_k E_{ij}^{[k]}(N_{ij}(Y_{ij})) + (I + T_{ii})Y_{ij}h(Y^2)_{jj},$$
$$b_{ij} := \sum_{t=i+1}^{j} T_{it}q(Y)_{tj} - (I + T_{ii}) \sum_{t=i+1}^{j-1} Y_{it}h(Y^2)_{tj} - \sum_{k=1}^{\ell} \mu_k \left(E_{ij}^{[k]}(\widetilde{\Phi}_{ij}) + \Phi_{ij}^{[k]}\right),$$

with $\mu_k = \gamma_k(I - T_{ii}) + \delta_k(I - T_{ii})Y_{ii}$.

In order to get the matrix coefficient of the linear system $L_{ij}(Y_{ij}) = b_{ij}$, we use the operator vec, which stacks the columns of an $M \times N$ matrix into a long vector of length MN. The vec operator features in the identity $\mathrm{vec}(A_1A_2A_3) = (A_3^T \otimes A_1)\mathrm{vec}(A_2)$, where $A \otimes B$ denotes the Kronecker product of the two matrices A and B [Horn and Johnson, 1991, Lemma 4.3.1].

By taking the vec of both sides of (18), we get

$$M_{ij}\mathrm{vec}(Y_{ij}) = \varphi_{ij}, \tag{19}$$

where

$$M_{ij} := \sum_{k=1}^{\ell} \mu_k \widehat{E}_{ij}^{[k]} \widehat{N}_{ij} + h(Y^2)_{jj}^T \otimes (I + T_{ii}),$$
$$\varphi_{ij} := \mathrm{vec}\left(\sum_{t=i+1}^{j} T_{it}q(Y)_{tj} - (I + T_{ii}) \sum_{t=i+1}^{j-1} Y_{it}h(Y^2)_{tj} \right) - \sum_{k=1}^{\ell} \mu_k \left(\widehat{E}_{ij}^{[k]} \mathrm{vec}\left(\widetilde{\Phi}_{ij}\right) + \mathrm{vec}\left(\Phi_{ij}^{[k]}\right) \right), \tag{20}$$

and $\widehat{E}_{ij}^{[k]}$ and $\widehat{N}_{ij}$ are the matrices representing the operators $E_{ij}^{[k]}$ and N_{ij}, respectively.

We can use (20) to design an algorithm for the solution of (8): after computing the diagonal blocks of $Y^{[k]}$, $g(Y^2)$ and $h(Y^2)$ directly, the off-diagonal blocks in the upper triangular half of Y can be obtained, one super-diagonal at a time, by solving the linear system (19). The pseudocode of this procedure is given in detail in Algorithm 1.

Some computation can be saved by observing that $\Phi_{ij}^{[k]}$ and $\widehat{E}_{ij}^{[k]}$ follow the recursions

$$\Phi_{ij}^{[2]} = F_{ij}^{[2]}, \qquad \Phi_{ij}^{[k]} = F_{ij}^{[k]} + Y_{ii}^{[1]}\Phi_{ij}^{[k-1]}, \qquad k = 3, \ldots, \ell,$$
$$\widehat{E}_{ij}^{[1]} = I, \qquad \widehat{E}_{ij}^{[k]} = \left((Y_{jj}^{[1]})^T \otimes I\right)\widehat{E}_{ij}^{[k-1]} + I \otimes Y_{ii}^{[k-1]}, \qquad k = 2, \ldots, \ell.$$

3.2.1 Applicability

The algorithm may have a breakdown if M_{ij} is singular. We show that this depends on what solutions to (9) are chosen, but does not happen when computing isolated solutions. Before stating the applicability condition, we prove a technical lemma.

Algorithm 1: Algorithm for $r(X) = A$, with r as in (3), based on even powers.

Input : $A \in \mathbb{C}^{N\times N}$, $\gamma_0, \dots, \gamma_\ell$ coefficients of g, $\delta_0, \dots, \delta_\ell$ coefficients of h.
Output: $X \in \mathbb{C}^{N\times N}$ such that $p(X)q^{-1}(X) \approx A$.

Compute a block upper triangular decomposition $A := UTU^{-1}$ as in (7).
for $i = 1$ **to** ν **do**
  $Y_{ii} \leftarrow$ a solution to $g(X^2) + Xh(X^2) - T_{ii}(g(X^2) - Xh(X^2)) = 0$
  $Y_{ii}^{[1]} \leftarrow Y_{ii}^2$
  for $k = 2$ **to** ℓ **do**
    $Y_{ii}^{[k]} \leftarrow Y_{ii}^{[1]} Y_{ii}^{[k-1]}$
  $H_{ii} \leftarrow \sum_{k=0}^{\ell} \delta_k Y_{ii}^{[k]}$
  $Q_{ii} \leftarrow \sum_{k=0}^{\ell} \gamma_k Y_{ii}^{[k]} - Y_{ii} H_{ii}$
for $u = 1$ **to** $\nu - 1$ **do**
  for $i = 1$ **to** $\nu - u$ **do**
    $j \leftarrow i + u$
    for $k = 2$ **to** ℓ **do**
      $F_{ij}^{[k]} \leftarrow \sum_{t=i+1}^{j-1} Y_{it}^{[1]} Y_{tj}^{[k-1]}$
    $\widetilde{\Phi}_{ij} \leftarrow \sum_{t=i+1}^{j-1} Y_{it} Y_{tj}$
    $\Phi_{ij}^{[2]} \leftarrow F_{ij}^{[2]}$
    for $k = 3$ **to** ℓ **do**
      $\Phi_{ij}^{[k]} \leftarrow F_{ij}^{[k]} + Y_{ii}^{[1]} \Phi_{ij}^{[k-1]}$
    $\widehat{E}_{ij}^{[1]} \leftarrow I_{\tau_i \tau_j}$
    for $k = 2$ **to** ℓ **do**
      $\widehat{E}_{ij}^{[k]} \leftarrow \left(\left(Y_{jj}^{[1]} \right)^T \otimes I_{\tau_i} \right) \widehat{E}_{ij}^{[k-1]} + \left(I_{\tau_j} \otimes Y_{ii}^{[k-1]} \right)$
    for $k = 1$ **to** ℓ **do**
      $\mu_k \leftarrow \gamma_k I_{\tau_i} + \delta_k Y_{jj} + T_{ii}(\delta_k Y_{ii} - \gamma_k I_{\tau_i})$
    $M_{ij} = \left(\sum_{k=1}^{\ell} (I_{\tau_j} \otimes \mu_k) \widehat{E}_{ij}^{[k]} \right) (Y_{jj}^T \otimes I_{\tau_i} + I_{\tau_j} \otimes Y_{ii}) + H_{jj}^T \otimes (I_{\tau_i} + T_{ii})$
    $K \leftarrow \sum_{t=i+1}^{j-1} Y_{it} H_{tj}$
    $\varphi_{ij} \leftarrow \operatorname{vec}\left(\sum_{t=i+1}^{j} T_{it} Q_{tj} - (I_{\tau_i} + T_{ii}) K \right)$
    $\varphi_{ij} \leftarrow \varphi_{ij} - \sum_{k=2}^{\ell} \mu_k \left(\widehat{E}_{ij}^{[k]} \operatorname{vec}\left(\widetilde{\Phi}_{ij} \right) + \operatorname{vec}\left(\Phi_{ij}^{[k]} \right) \right) - \mu_1 \operatorname{vec}\left(\widetilde{\Phi}_{ij} \right)$
    $Y_{ij} \leftarrow \operatorname{vec}^{-1}(M_{ij}^{-1} \varphi_{ij})$
    $Y_{ij}^{[1]} \leftarrow Y_{ii} Y_{ij} + Y_{ij} Y_{jj} + \widetilde{\Phi}_{ij}$
    for $k = 2$ **to** ℓ **do**
      $Y_{ij}^{[k]} \leftarrow Y_{ii}^{[1]} Y_{ij}^{[k-1]} + Y_{ij}^{[1]} Y_{jj}^{[k-1]} + F_{ij}^{[k]}$
    $H_{ij} \leftarrow \sum_{k=1}^{\ell} \delta_k Y_{ij}^{[k]}$
    $Q_{ij} \leftarrow \sum_{k=1}^{\ell} \gamma_k Y_{ij}^{[k]} - K - Y_{ii} H_{ij} - Y_{ij} H_{jj}$
$X \leftarrow UYU^{-1}$

Lemma 4. *For the matrix M_{ij} in (20), one has that*

$$M_{ij} = \sum_{k=1}^{2\ell+1} c_k \widehat{B}_{ij}^{[k]} - (I \otimes T_{ii}) \sum_{k=1}^{2\ell+1} d_k \widehat{B}_{ij}^{[k]}, \qquad \widehat{B}_{ij}^{[k]} := \sum_{u=0}^{k-1} (Y_{jj}^{k-1-u})^T \otimes Y_{ii}^u. \tag{21}$$

where c_k and d_k, for $k = 1, \ldots, 2\ell + 1$, are the coefficients of the polynomials $p(z) := g(z^2) + zh(z^2)$ and $q(z) := g(z^2) - zh(z^2)$, respectively.

Proof. First, observe that

$$\begin{aligned}
\widehat{E}_{ij}^{[k]} \widehat{N}_{ij} &= \left(\sum_{u=0}^{k-1} (Y_{jj}^{[k-1-u]})^T \otimes Y_{ii}^{[u]} \right) (Y_{jj}^T \otimes I + I \otimes Y_{ii}) \\
&= \sum_{u=0}^{k-1} \left((Y_{jj}^{2(k-1-u)+1})^T \otimes Y_{ii}^{2u} + (Y_{jj}^{2(k-1-u)})^T \otimes Y_{ii}^{2u+1} \right) \\
&= \sum_{u=0}^{2k-1} (Y_{jj}^{2k-1-u})^T \otimes Y_{ii}^u = \widehat{B}_{ij}^{[2k]}.
\end{aligned}$$

Define

$$\begin{aligned}
\widetilde{P}_{ij} &:= \sum_{k=1}^{\ell} \big(I \otimes (\gamma_k I + \delta_k Y_{ii}) \big) \widehat{B}_{ij}^{[2k]} + h(Y^2)_{jj}^T \otimes I, \\
\widehat{P}_{ij} &:= \sum_{k=1}^{\ell} \big(I \otimes (\gamma_k I - \delta_k Y_{ii}) \big) \widehat{B}_{ij}^{[2k]} - h(Y^2)_{jj}^T \otimes I.
\end{aligned} \tag{22}$$

In view of (20), it is necessary to prove only that $\widetilde{P}_{ij} = \sum_{k=1}^{2\ell+1} c_k \widehat{B}_{ij}^{[k]}$ and that $\widehat{P}_{ij} = \sum_{k=1}^{2\ell+1} d_k \widehat{B}_{ij}^{[k]}$. Note that, since $\gamma_k = c_{2k}$ and $\delta_k = c_{2k+1}$, we have

$$(I \otimes \gamma_k I) \widehat{B}_{ij}^{[2k]} = c_{2k} \widehat{B}_{ij}^{[2k]}, \qquad k = 1, \ldots, \ell, \tag{23}$$

and, since $h(Y^2)_{jj} = \sum_{k=0}^{\ell} \delta_k Y_{jj}^{2k}$, we have

$$\begin{aligned}
&\sum_{k=1}^{\ell} \big(I \otimes \delta_k Y_{ii} \big) \widehat{B}_{ij}^{[2k]} + h(Y^2)_{jj}^T \otimes I \\
&\qquad = \sum_{k=1}^{\ell} c_{2k+1} \big((I \otimes Y_{ii}) \widehat{B}_{ij}^{[2k]} + (Y_{jj}^{2k})^T \otimes I \big) + c_1 I = \sum_{k=0}^{\ell} c_{2k+1} \widehat{B}_{ij}^{[2k+1]},
\end{aligned} \tag{24}$$

where the latter equality follows from the definition of $\widehat{B}_{ij}^{[k]}$, since

$$(I \otimes Y_{ii})\widehat{B}_{ij}^{[2k]} + (Y_{jj}^{2k})^T \otimes I = \sum_{u=0}^{2k-1} (Y_{jj}^{2k-1-u})^T \otimes Y_{ii}^u Y_{ii} + (Y_{jj}^{2k})^T \otimes I$$
$$= \sum_{u=0}^{2k} (Y_{jj}^{2k-u})^T \otimes Y_{ii}^u = \widehat{B}_{ij}^{[k+1]}.$$

Equations (23) and (24) together prove the first equality in (22). The corresponding equality for $\widehat{P}_{ij}$ can be proved analogously, and this concludes the proof. □

Now we can state the applicability theorem.

Theorem 5. *Let g, h be polynomials of degree ℓ, let $p(z) := g(z^2) + zh(z^2)$ and $q(z) := g(z^2) - zh(z^2)$ be coprime, let $r(z) := p(z)q(z)^{-1}$, let $T = (T_{ij}) \in \mathbb{C}^{N\times N}$ be block upper triangular with ν diagonal blocks of size $\tau_1, \dots, \tau_\nu$, and let $\Xi_i \in \mathbb{C}^{\tau_i \times \tau_i}$, for $i = 1, \dots, \nu$, be a solution to $r(\Xi) = T_{ii}$. Then the following two conditions are equivalent:*

1. *Algorithm 1 with the choice $Y_{ii} = \Xi_i$ is applicable, i.e., equation (19) has a unique solution Y_{ij} for $1 \le i < j \le \nu$;*
2. *for all $1 \le i < j \le \nu$, if ξ_i and ξ_j are eigenvalues of Ξ_i and Ξ_j, respectively, then $r[\xi_i, \xi_j] \neq 0$.*

Under these conditions, if Ξ_i is an isolated solution of the equation $r(\Xi) = T_{ii}$ for $i = 1, \dots, \nu$ then Algorithm 1 computes an isolated solution to (8).

Proof. The proof is analogous to that of [Fasi and Iannazzo, 2020, Thm. 3.4] once one observes that the matrix M_{ij} in (21), which determines the applicability of Algorithm 1, is the same as that in [Fasi and Iannazzo, 2020, Eq. (3.15)], which determines the applicability of Algorithm 1 therein. □

5.0.1 Implementation details

Cancellation can occur in the computation of the diagonal of the matrix $I - T_{ii}$. This issue can be addressed by defining $g_{jj}^{[u]} := \sum_{v=u}^{\ell} \gamma_v Y_{jj}^{2(v-u)}$ and $h_{jj}^{[u]} := \sum_{v=u}^{\ell} \delta_v Y_{jj}^{2(v-u)}$, and rewriting the matrix coefficient in (21) as

$$M_{ij} = (I \otimes (I - T_{ii})) \sum_{u=1}^{\ell} ((g_{jj}^{[u]})^T \otimes Y_{ii}^{2u-1}) + ((g_{jj}^{[u]} Y_{jj})^T \otimes Y_{ii}^{2u-2})$$
$$+ (I \otimes (I + T_{ii}) Y_{ii}) \sum_{u=1}^{\ell} ((h_{jj}^{[u]})^T \otimes Y_{ii}^{2u-1}) + ((h_{jj}^{[u]} Y_{jj})^T \otimes Y_{ii}^{2u-2}) + (h_{jj}^{[0]})^T \otimes (I + T_{ii}),$$

which, given $g_{jj}^{[u]}$, $h_{jj}^{[u]}$, and $Y_{ii}^{[u]} = Y_{ii}^{2u}$ for $u = 1, \dots, \ell$, can be evaluated with the following algorithm:

1. compute $\Gamma_{ij} = (I \otimes Y_{ii} + Y_{jj}^T \otimes I)$;
2. compute $(g_{jj}^{[u]})^T \otimes Y_{ii}^{[u-1]}$ and $(h_{jj}^{[u]})^T \otimes Y_{ii}^{[u-1]}$, for $u = 1, \ldots, \ell$;
3. compute $C_1 = \sum_{u=1}^{\ell} ((g_{jj}^{[u]})^T \otimes Y_{ii}^{[u-1]})\Gamma_{ij}$;
4. compute $C_2 = (I \otimes Y_{ii}) \sum_{u=1}^{\ell} ((h_{jj}^{[u]})^T \otimes Y_{ii}^{[u-1]})\Gamma_{ij}$;
5. compute $M_{ij} = C_2 + (h_{jj}^{[0]})^T \otimes I + C_1 + (I \otimes T_{ii})(C_2 + (h_{jj}^{[0]})^T \otimes I - C_1)$.

The last step replaces the cheaper and more obvious:

*5. compute $M_{ij} = (I \otimes (I - T_{ii}))C_1 + (I \otimes (I + T_{ii}))(C_2 + (h_{jj}^{[0]})^T \otimes I)$;

which may, however, be prone to numerical cancellation.

5.0.2 Computational complexity

As discussed above, the computational cost of Algorithm 1 is related to the number of matrix multiplications needed by the evaluation scheme on which the method is based. The most expensive steps of the procedure with respect to the size $v = N$ are the computation of $F_{ij}^{[k]}$ in (12), for $k = 1, \ldots, \ell$, and the two sums on the right-hand side of (15). Therefore, Algorithm 1 overall requires $(\ell + 2)v^3/3 + o(v^3)$ operations.

For comparison, [Fasi and Iannazzo, 2020, Alg. 1] requires $(2\ell + 1)v^3/3 + o(v^3)$ operations, while [Fasi and Iannazzo, 2020, Alg. 2] requires, for a given positive integer s, $(2r + s)v^3/3 + o(v^3)$ operations, where $r := \lceil (2\ell + 1)/s - 1 \rceil$. Therefore Algorithm 1 is always asymptotically cheaper than [Fasi and Iannazzo, 2020, Alg. 1] and is not more expensive than [Fasi and Iannazzo, 2020, Alg. 2] for $\ell \leq 9$, which, as discussed in Section 6, includes all values of ℓ needed for computing the matrix logarithm.

Our implementation of Algorithm 1 works on one super-diagonal at a time from the main diagonal to the top right corner, and in addition to Y it requires the storage of $\ell + 2$ intermediate (block) triangular matrices, that is, $h(Y^2)$, $q(Y^2)$, and $Y^{[k]}$ for $k = 1, \ldots, \ell$. Alternatively, he matrix Y could be computed one column at a time from left to right, with each column being computed from the element just above the diagonal to the first row. This alternative implementation can significantly reduce the amount of additional storage needed by the algorithm without affecting the asymptotic computational cost in terms of v: only Y_{ij} and $Y_{ij}^{[1]}$ would have to be kept for the whole duration of the algorithm, while for all the other stages it would suffice to store only the current column and the diagonal elements. This technique is analogous to the that suggested by Van Loan [1979] for reducing the storage requirements of the Paterson–Stockmeyer method for polynomial evaluation.

6 Computing the matrix logarithm

The inverse scaling and squaring algorithm is one of the most effective techniques for computing the principal logarithm of a matrix. The built-in MATLAB function `logm` and the Julia function `Base.log(A{T}::StridedMatrix{T})` are based on the inverse scaling and squaring method as described in [Al-Mohy and Higham, 2012, Alg. 4.1] and [Al-Mohy et al., 2013, Alg. 6.1], whereas the Octave core function `logm` implements the variant in [Higham, 2008, Alg. 11.9].

The algorithm finds the (real) Schur decomposition $A = UTU^*$ and sets $B = T^{2^{-s}}$ where s is an integer such that the spectrum of B lies within the open disc $\{z \in \mathbb{C} : |z-1| < 1\}$. The matrix $\log(B)$ is then approximated as $r_m(B-I)$, where r_m is the $[m/m]$ diagonal Padé approximant to $\log(z)$ at $z = 1$. The logarithm of A is then recovered by exploiting the relation

$$\log(A) = U \cdot \log(T) \cdot U^* = U \cdot 2^s \log(B) \cdot U^* \approx U \cdot 2^s r_m(B-I) \cdot U^*.$$

The degree m is chosen so to guarantee that the relative backward truncation error

$$\frac{\|\Delta_m\|}{\|B-I\|}, \tag{25}$$

where $\Delta_m := \exp(r_m(B-I)) - B$, is smaller than the unit roundoff u in exact arithmetic. Computing Δ_m directly would be too expensive, and a possible approach is to expand Δ_m in a power series and bound (25) by

$$\frac{\|\Delta_m\|}{\|B-I\|} \le \sum_{k=2m+1}^{\infty} |\widehat{\delta}_k| \alpha_p(B-I)^k =: F_m(\alpha_p(B-I)), \tag{26}$$

where

$$\alpha_p(X) = \max\{\|X^p\|^{1/p}, \|X^{p+1}\|^{1/(p+1)}\} \tag{27}$$

and p is any positive integer that satisfies $p(p-1) \le 2m+1$.

The coefficients of the series (26) can be computed symbolically, and by combining symbolic and high precision computation, one can estimate accurately, for i between 1 and some positive integer $m_{\max}$, the quantity

$$\theta_i^F = \max_{\theta \in \mathbb{R}^+} \{F_i(\theta) \le u\}. \tag{28}$$

Note that $\alpha_p(B-I) \le \theta_i^F$ guarantees that $\|\Delta_i\|/\|B-I\| \le u$. It is convenient to set m to the smallest i such that $r_i(B-I)$ delivers an approximation to $\log(B)$ with a backward error below the unit roundoff.

We now discuss how to choose the number s of square roots and the degree m of the approximant. On the one hand, a large s leads to a matrix B with eigenvalues near 1, for which a small m is sufficient to compute an approximation whose truncation error is bounded by u. On the other hand, a small s requires a larger m to get an approximation of the same quality. The algorithm attempts to minimize the computational cost by

Table 1: The values of θ_m in [Al-Mohy and Higham, 2012, Table 2.1] are compared with those of θ_m^F for m between 1 and 8.

m	θ_m	θ_m^F
1	$1.586970738772063 \times 10^{-5}$	$3.650024116682167 \times 10^{-8}$
2	$2.313807884242979 \times 10^{-3}$	$3.759321363926338 \times 10^{-4}$
3	$1.938179313533253 \times 10^{-2}$	$8.202379304954202 \times 10^{-3}$
4	$6.209171588994762 \times 10^{-2}$	$3.792548581321354 \times 10^{-2}$
5	$1.276404810806775 \times 10^{-1}$	$9.334652296460314 \times 10^{-2}$
6	$2.060962623452836 \times 10^{-1}$	$1.668083440029836 \times 10^{-1}$
7	$2.879093714241195 \times 10^{-1}$	$2.479601520292692 \times 10^{-1}$
8	$3.666532675959788 \times 10^{-1}$	$3.287599317808182 \times 10^{-1}$

finding a trade-off between s and m. Since for a triangular B evaluating $r_m(B - I)$ and computing the square root of B require $mN^3/3$ and $N^3/3$ flops, respectively, it may be worth taking an additional square root if doing so is expected to decrease the degree of the approximant to be used by more than 1. Therefore, in view of the estimate

$$\alpha_p(A^{1/2} - I) \approx \frac{\alpha_p(A - I)}{2}, \tag{29}$$

the algorithm takes an additional square root when $\alpha_p(B - I) \le 2\theta_{m-2}^F$. In fact, this condition has to be tested only if $\theta_{m-1}^F < 2\theta_{m-2}^F$, as the degree m is selected as a candidate only if $\theta_{m-1}^F < \alpha_p(B - I) \le \theta_m^F$.

Once a pair (s, m) is chosen, the algorithm evaluates the $[m/m]$ Padé approximant at $B - I$, then reverts the square roots by exploiting the matrix identity $\log(T) = 2^s \log(T^{2^{-s}}) = 2^s \log(B - I)$, and returns the approximation $2^s U r_m(B - I)U^*$. In order to improve the accuracy of the solution, the diagonal and first upper diagonal of $B - I$ and $r_m(B - I)$ can be recomputed by using direct formulae for the blocks along the diagonal.

As noted in [Higham, 2008, Chap. 11], it is not necessary to check the values of m larger that $m_{\max} = 7$, since $m_{\max}$ is the largest integer i that satisfies $\theta_i^F \le 2\theta_{i-2}^F$, and taking an additional square root is expected to reduce the cost of the evaluation of the Padé approximant by at least $2N^3/3$ flops for larger values of m.

Remark 7. *The values of θ_m in [Al-Mohy and Higham, 2012, Table 2.1] for double precision are computed using an expansion different from that in (26). These constants and their correct value for double precision are reported in the first and second column of Table 1, respectively.*

Finally, we discuss how the algorithm above can be expressed in terms of BLAS and LAPACK operations. For convenience, we divide the algorithm into three phases: a pre-processing step that computes the Schur decomposition and performs the initial scaling, a second stage in which the rational approximant is evaluated at an upper (quasi-)triangular matrix, and a final post-processing step that recovers the solution.

P1) **Pre-processing phase.** The Schur decomposition can be computed using the LAPACK routines `xGEHRD`, which reduces the matrix A to upper-Hessenberg form, and `xHSEQR`, which returns the (quasi-)triangular Schur factor T and the corresponding Schur vectors Q. The square roots of T can be computed using a divide-and-conquer method of Deadman et al. [2013], which requires only the solution of Sylvester equations, an operation that can be performed using the LAPACK routine `xLASY2`.

P2) **Evaluation of rational function.** The evaluation of the numerator and denominator of the Padé approximant requires the repeated use of the BLAS routines `xAXPBY` (level 1) and `xTRMM` (level 3). The solution of the ensuing triangular linear system can be achieved by relying on the BLAS routine `xTRSM` (level 3).

P3) **Post-processing phase.** The solution is recovered by performing two matrix multiplications, one between the matrix Q and an upper (quasi-)triangular matrix, and one between two dense matrices. As the BLAS do not provide a routine for computing the product between a general and a triangular matrix, this final computation is performed by a double invocation of the general routine `xGEMM` (level 3).

7.1 A new analysis

In order to exploit the algorithm in Section 3 to compute $\log(A)$, we explore a different approach to the approximation of $\log(B)$. The new technique relies on the solution of the rational equation $\widetilde{r}_{mn}(X) = B$, where $\widetilde{r}_{mn}(z) = \widetilde{p}_{mn}(z)\widetilde{q}_{mn}(z)^{-1}$ is the $[m/n]$ Padé approximant to e^z at $z = 0$. As the poles of $\widetilde{r}_{mn}(z)$ lie in the annulus [Saff and Varga, 1977, Thm. 2.2]

$$\left\{ z \in \mathbb{C} : (m+n)W_0(\mathrm{e}^{-1}) < |z| < m+n+\frac{4}{3} \right\},$$

where W_0 is the principal branch of the Lambert W function, there exists a neighborhood of 0 where $\widetilde{r}_{mn}$ is analytic. By applying the quotient rule to $\widetilde{r}_{mn}$ and using (4), one can easily see that $\widetilde{r}'_{mn}(0) \neq 0$. Thus, there exists an inverse of $\widetilde{r}_{mn}$, say $\widetilde{r}_{mn}^{-1}$, analytic in a neighborhood of 1 and such that $\widetilde{r}_{mn}^{-1}(1) = 0$.

For a sufficiently small $\varepsilon > 0$, there exists a ball $\mathcal{B}_{mn}$ with center 1 where the branch $\widetilde{r}_{mn}^{-1}$ is analytic, $|\mathrm{e}^{\widetilde{r}_{mn}^{-1}(z)} - z|/|z| \leq \varepsilon$ for all $z \in \mathcal{B}_{mn}$, and $\widetilde{r}'_{mn}(z) \neq 0$ in $\widetilde{r}_{mn}^{-1}(\mathcal{B}_{mn})$. This branch of $\widetilde{r}_{mn}^{-1}$ approximates the principal logarithm in a neighborhood of 0, where the quantity $\varepsilon_{mn}(z) := \mathrm{e}^{\widetilde{r}_{mn}^{-1}(z)}/z - 1$ can be seen as a relative backward truncation error, and in a neighborhood of 1 we have the series expansion

$$\varepsilon_{mn}(z) = \sum_{k=m+n+1}^{\infty} \widehat{\delta}_k (z-1)^k. \tag{30}$$

Table 2: First few optimal degrees for the algorithm that computes the logarithm by solving a matrix equation with the diagonal approximants to the exponential. For each degree m_j we report the asymptotic cost of the algorithm $C(m_j)$, the values of p that satisfy $p(p-1) \leq 2m_j + 1$, and the value $\theta^G_{m_j}$ in (33).

j	m_j	$C(m_j)$	p	$\theta^G_{m_j}$
1	1	$N^3/3$	2	$1.1003511163692342 \times 10^{-5}$
2	2	$2N^3/3$	2	$2.4012849957497128 \times 10^{-3}$
3	3	$3N^3/3$	2, 3	$2.7099573188927441 \times 10^{-2}$
4	5	$4N^3/3$	2, 3	$2.6059916466908718 \times 10^{-1}$
5	7	$5N^3/3$	2, 3, 4	$6.5282885430846634 \times 10^{-1}$
6	9	$6N^3/3$	2, 3, 4	$9.0572865457020838 \times 10^{-1}$

The coefficients in (30) can be easily computed from those of the series expansion

$$\exp(\widetilde{r}_{mn}^{-1}(z)) - z = \sum_{k=0}^{\infty} \beta_k (z-1)^k, \tag{31}$$

which can be obtained by composing the series expansion for e^z at $z = \widetilde{r}_{mn}^{-1}(1) = 0$ with the series expansion for $\widetilde{r}_{mn}^{-1}(z)$ at 1 determined using Lagrange's expansion formula [Abramowitz and Stegun, 1972, Fact 3.6.7]. Since $\widetilde{r}'_{mn}(z)$ approximates e^z up to the $(m+n)$th derivative, we have that $\beta_k = 0$ for $k \leq m+n$, and we can conclude that (30) holds with $\widehat{\delta}_{m+n+1} = \beta_{m+n+1}$ and $\widehat{\delta}_k = \beta_k - \widehat{\delta}_{k-1}$ for $k > m+n+1$.

Turning to matrices, if the eigenvalues $\lambda_1, \ldots, \lambda_N$ of B lie in $\mathcal{B}_{mn}$, then the equation $\widetilde{r}_{mn}(X) = B$, has a unique solution, say $X := \widetilde{r}_{mn}^{-1}(B)$, with eigenvalues $\widetilde{r}_{mn}^{-1}(\lambda_1), \ldots, \widetilde{r}_{mn}^{-1}(\lambda_N)$, which is primary and isolated, and can thus be computed using Algorithm 1. The existence of such a solution follows from [Fasi and Iannazzo, 2020, Thm. 3.3], since $\widetilde{r}'_{mn}(\lambda_i) \neq 0$ for $i = 1, \ldots, N$ and no pair of distinct eigenvalues of X is mapped to the same complex value. The applicability of Algorithm 1 follows from Theorem 5.

The backward truncation error in the approximation of $\log(B)$ by means of the inverse of $\widetilde{r}_{mn}$ is given by the matrix $\Delta_{mn} = \mathrm{e}^X - B = \exp(\widetilde{r}_{mn}^{-1}(B)) - B \in \mathbb{C}^{N\times N}$, which satisfies $X = \log(B + \Delta_{mn})$. If the eigenvalues of B are within the radius of convergence of the series (30), we can write

$$\begin{aligned} \frac{\|\Delta_{mn}\|}{\|B\|} &\leq \left\| \sum_{k=m+n+1}^{\infty} \widehat{\delta}_k (B-I)^k \right\| \\ &\leq \sum_{k=m+n+1}^{\infty} |\widehat{\delta}_k| \alpha_p(B-I)^k \\ &=: G_{mn}(\alpha_p(B-I)), \end{aligned} \tag{32}$$

where the function α_p is defined in (27).

In analogy with the algorithm of Al-Mohy and Higham [2012], we compute

$$\theta_i^G = \max_{\theta \in \mathbb{R}^+} \{G_{ii}(\theta) \leq u\}, \tag{33}$$

to aid with the choice of the diagonal Padé approximant that will deliver full accuracy. We report the values of θ_i^G for optimal degrees between 1 and 9 in Table 2. These values were determined by computing symbolically the first 600 terms of the series expansion G_{ii} and performing all subsequent computation using 250 digits of accuracy. As implicitly assumed in the analysis by Al-Mohy and Higham [2012], we conjecture that the radius of convergence of G_{ii} is indeed larger than θ_i^G, which seems to be the case numerically.

A variant of this algorithm is obtained by considering the $[m/n]$ Padé approximant to $\mathrm{e}^z - 1$ at $z = 0$, say $\widehat{r}_{mn}(z) := \widehat{p}_{mn}(z)\widehat{q}_{mn}(z)^{-1}$, and approximating $\log(B)$ as a solution to $\widehat{r}_{mn}(X) = B - I$. Note that $\widetilde{r}_{mn}(z) = \widehat{r}_{mn}(z) - 1$, and in exact arithmetic this variant computes the same solution as Algorithm 2. Hence the backward error in (32), can equivalently be written as

$$\Delta_{mn} = \exp\big(\widehat{r}_{mn}^{-1}(B - I)\big) - B.$$

and bounded by expanding the function $\mathrm{e}^{\widehat{r}_{mn}^{-1}(z)} - 1 - z$ at 0. We note that this method would require the same values of θ_i^G, but in finite arithmetic may produce results which differ from those of the method above.

7.2 The dual inverse scaling and squaring algorithm

In order to develop a new algorithm for computing the matrix logarithm, we begin by determining what degrees the new method should use. A closer look at Table 2 reveals two important points. First, since θ_m^G must be below 1 and $\theta_{m_5}^G = \theta_7^G > 0.5$, we are guaranteed that $\theta_{m_j}^G < 2\theta_{m_{j-2}}^G$ for $j > 6$, and the largest degree to consider is $m_6 = 9$, since for larger values of m taking an additional square root is expected to reduce the computational cost by at least $N^3/3$.

On the other hand, $\theta_{m_{j-1}}^G > 2\theta_{m_{j-2}}^G$ for $j \leq 6$, thus if $\alpha_p(B - I)$ is smaller than $\theta_{m_j}^G$, then in view of the estimate (29) taking an additional square root will never reduce the cost of the approximant to be used by more than $N^3/3$, and we conclude that taking an additional square root is not likely to reduce the computational cost once an approximant has been found. Finally, we do not consider the approximants of degree 1 or 2, whose evaluation is prone to loss of accuracy in floating-point arithmetic [Higham, 2008, p. 245].

The pseudocode of our strategy for computing the matrix logarithm using the analysis in Section 7.1 is given in Algorithm 2. The algorithm begins by computing the Schur decomposition $A =: UTU^*$, and then uses the algorithm by Björck and Hammarling [1983] to take square roots of T, s of them say, until the spectral radius of $B - I$, where $B = T^{2^{-s}}$ as before, becomes smaller than θ_9^G. Since $\|X^k\|^{1/k} \geq \rho(X)$ for all $k \in \mathbb{N}$ and $X \in \mathbb{C}^{N \times N}$, this is a sufficient condition for $\alpha_p(B - I)$ to be smaller than

θ_9^G. We prefer to rely on the spectral radius here as $\rho(X)$ is much cheaper than $\alpha_p(X)$ to compute if $X \in \mathbb{C}^{N\times N}$ is quasi-triangular.

Then, the algorithm tries to determine the smallest $m \in \{3, 5, 7, 9\}$ such that the backward error in the evaluation of $\widetilde{r}_m^{-1}(B - I)$ is smaller than u, using the error bound (32) and the values in Table 2. Since only an estimate of $\alpha_p(B - I)$ is needed, we estimate the 1-norm of powers of $X \in \mathbb{C}^{N\times N}$ with the algorithm of Higham and Tisseur [2000], which performs matrix-(block) vector multiplications without explicitly computing any powers of X, and thus requires only $O(N^2)$ floating-point operations. In our pseudocode, `normest`(X, k) denotes the function that estimates $\|X^k\|_1$ using this algorithm.

Note that $\|X^4\|_1^{1/4} \leq \|X^2\|_1^{1/2}$ implies $\alpha_3(X) \leq \alpha_2(X)$, thus for $m > 2$ there is no need to compute $\alpha_2(B - I)$. Therefore, our algorithm computes a_3, an estimate of $\alpha_3(B - I)$, and then checks whether $a_3 \leq \theta_m^G$ for one of the values of m of interest. If that is the case, then it selects the lowest m that satisfies the inequality, and exits the parameter selection loop. Otherwise, the algorithm computes a_4, an estimate of $\alpha_4(B - I)$, and uses $\zeta := \min(a_3, a_4)$ to determine whether an approximant of degree 7 or 9 is expected to deliver full accuracy. The algorithm sets $m = 9$ if $\theta_7^G < \zeta \leq \theta_9^G$ and $m = 7$ if $\zeta \leq \theta_7^G$, and in both cases it exits the parameter selection loop. Finally, if $\zeta > \theta_9^G$, another square root is taken, and the selection process is attempted again.

Once a pair (s, m) such that $G_{mm}\big(\alpha_p(B - I)\big) < u$ is found, the algorithm solves the matrix equation $\widetilde{r}_m(X) = B$. The ith diagonal block of X is chosen as the solutions to the matrix equation $\widetilde{r}_m(X_{ii}) = B_{ii}$ that is closer to $\log(B_{ii})$ in norm. Finally the approximation $2^s U X U^*$ is returned.

We conclude by noting that, compared to the algorithm discussed at the beginning of this section, the new algorithm can harness the extremely optimized BLAS and LAPACK routines only during the pre- and post-processing steps, which are essentially the same as P1) and P3) above. The evaluation of the rational function P2) is here replaced by Algorithm 1, which cannot be recast in terms of high-level BLAS operations, and should therefore be implemented in a BLAS-like fashion [Goto and Van De Geijn, 2008] for Algorithm 2 to reach a comparable level of performance.

7.3 Alternative choice for the diagonal blocks of X

The solution to $\widetilde{r}_m(X) = B$ on line 27 of Algorithm 2 has diagonal elements $\widetilde{r}_m^{-1}(B_{11})$, $\ldots$, $\widetilde{r}_m^{-1}(B_{\nu\nu})$, where $B_{11}, \ldots, B_{\nu\nu}$ are the ν diagonal blocks of B. These values are approximations to the corresponding diagonal blocks of $\log(B)$.

Since X aims to approximate $\log(B)$, a variant of Algorithm 1 can be obtained by setting, on line 3 of Algorithm 1, the diagonal blocks of X to $\log(B_{11}), \ldots, \log(B_{\nu\nu})$ and using these values in the subsequent substitution step. The whole procedure can then be interpreted as applying Algorithm 2 to the matrix equation

$$\widetilde{r}_m(X) = \widetilde{B}, \qquad \widetilde{B} = B + \operatorname{diag}\left(\widetilde{r}_m(\log(B_{11})) - B_{11}, \ldots, \widetilde{r}_m(\log(B_{\nu\nu})) - B_{\nu\nu}\right).$$

Algorithm 2: Matrix logarithm via inverse Padé approximation.

Input : $A \in \mathbb{C}^{N\times N}$ such that $\sigma(A) \subset \mathbb{C} \setminus \mathbb{R}_0^-$.
Output: $X \approx \log(A)$.

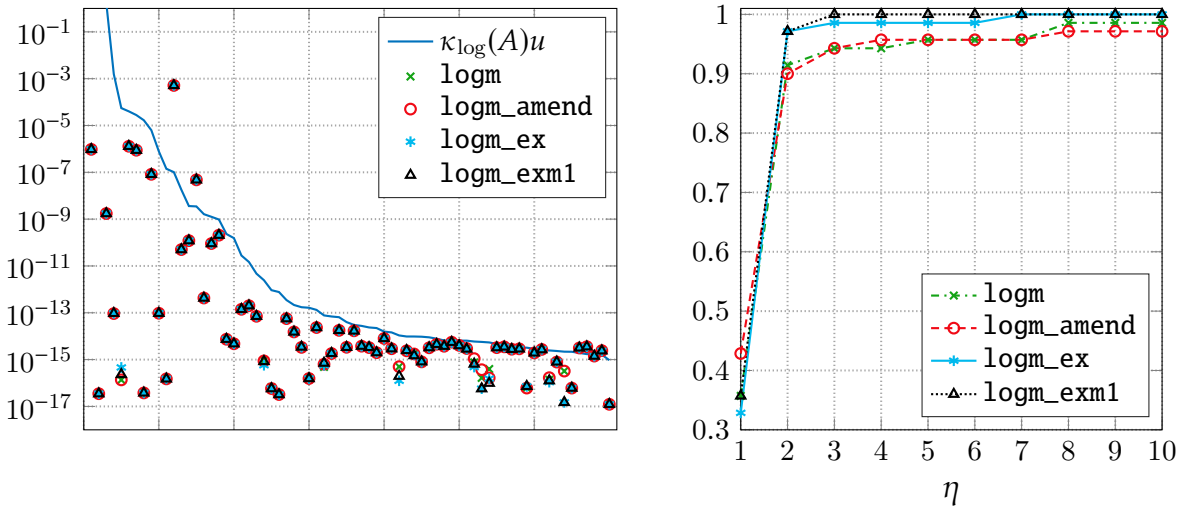
Compute the (real) Schur decomposition $A =: UTU^*$.
$s \leftarrow 0$
$m \leftarrow 0$
$B \leftarrow T$
while $\rho(B - I) > \theta_9^G$ **do**
    $B \leftarrow B^{1/2}$
    $s \leftarrow s + 1$
while $m = 0$ **do**
    $\mu_4 \leftarrow \texttt{normest}(B - I, 4)^{1/4}$
    $a_3 \leftarrow \max\{\texttt{normest}(B - I, 3)^{1/3}, \mu_4\}$
    $i \leftarrow 9$
    while $i \geq 3$ **do**
        if $a_3 \leq \theta_i^G$ **then**
            $m \leftarrow i$
        $i \leftarrow i - 2$
    if $m = 0$ **then**
        $a_4 \leftarrow \max\{\mu_4, \texttt{normest}(B - I, 5)^{1/5}\}$
        $\zeta \leftarrow \min(a_3, a_4)$
        if $\zeta \leq \theta_9^G$ **then**
            if $\zeta \leq \theta_7^G$ **then**
                $m \leftarrow 7$
            else
                $m \leftarrow 9$
        else
            $B \leftarrow B^{1/2}$
            $s \leftarrow s + 1$
$Y \leftarrow 2^s U\widetilde{r}_m^{-1}(B)U^*$

8 Numerical experiments

In this section we assess experimentally the accuracy and performance of the algorithms discussed in Section 6 and compare them with the the MATLAB function `logm`. The experiments were run on the 64-bit version of MATLAB 9.8.0 (2020a) on a machine equipped with an Intel Xeon Gold 6130 processor, running at 2.10GHz. We use a test set of 70 nonnormal matrices from the literature of the matrix logarithm [Al-Mohy and Higham, 2012, Fasi and Higham, 2018, 2019]. These matrices have size ranging between 2 and 400, and their spectra do not contain any real nonpositive eigenvalues.

In the tests, we consider the following implementations.

- `logm`: the built-in MATLAB function that combines the inverse scaling and squaring algorithms for complex and real matrices from [Al-Mohy and Higham, 2012,



(a) Forward error.

(b) Performance profile for (a).

Figure 1: Left: forward error of `logm`, `logm_amend`, `logm_ex`, and `logm_ex1` on the matrices in the test set sorted by descending condition number $\kappa_{\log}(A)$. Right: corresponding performance profile.

Al-Mohy et al., 2013], respectively, and uses the constants θ_m in Table 1.

- `logm_amend`: same as `logm`, using the constants θ_m^F in Table 1.
- `logm_ex`: an implementation of Algorithm 2 that uses the approximants to e^z at T and the variant in Section 7.3 for the choice of the diagonal blocks.
- `logm_exm1`: an implementation of Algorithm 2 that uses the approximants to $e^z - 1$ at $T - I$ and the variant in Section 7.3 for the choice of the diagonal blocks.

In order to assess the accuracy of the computed solution $\widetilde{X}$, we consider the relative forward error $\|\widetilde{X} - X\|_1 / \|X\|_1$, where X is a reference solution computed using the `logm` function in the Advanpix Multiprecision Computation Toolbox with precision set with the command `mp.Digits(64)`. In order to gauge the stability of these algorithms, we compare the forward error with the quantity $\kappa_{\log}(A)u$, where $\kappa_{\log}(A)$ is the 1-norm condition number of the logarithm of A and $u = 2^{-53}$ is the unit roundoff of IEEE double precision arithmetic.

8.1 Accuracy of the matrix logarithm

Figure 1 illustrates the accuracy of `logm`, `logm_amend`, `logm_ex`, and `logm_ex1`. In Figure 1(a) we compare the forward error of the four implementations on the matrices in our test set, sorted by decreasing value of $\kappa_{\log}(A)$. Figure 1(b) reports the same data on a by-algorithm rather than by-matrix basis by means of performance profiles [Dolan and Moré, 2002]: for a given method, the height of the line at $\eta = \eta_0$ represents the

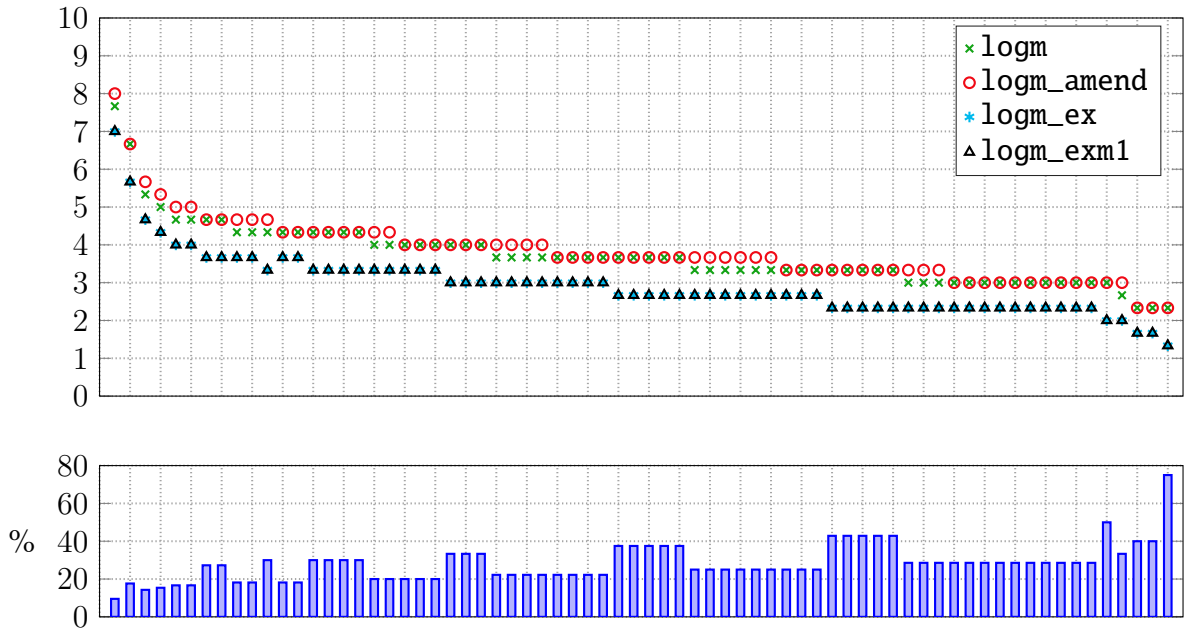


Figure 2: Top: computational cost of `logm`, `logm_amend`, `logm_ex`, and `logm_ex1` on the matrices in the test set. Bottom: relative gain of `logm_ex` with respect to `logm`, according to the formula in (34).

fraction of matrices in the test set for which the relative forward error is at most η_0 times that of the algorithm that delivers the most accurate result for that matrix.

Our new algorithms appear to be as forward stable as the MATLAB function `logm`, as the forward error is essentially the same. Occasionally minor disagreements occur, but only when the forward error of all algorithms is well below $\kappa_{\log}(A)u$. The performance of `logm_ex` and `logm_exm1` is remarkably similar, and these two new algorithms appear to be marginally more accurate than `logm` and `logm_amend` on just a few of the matrices in our test set. The difference between the accuracy of `logm` and `logm_amend` is negligible, with results indistinguishable for all but one matrix, where the latter algorithm is slightly less accurate.

8.2 Computational cost of the matrix logarithm

Figure 2 reports the computational cost of the four algorithms on the matrices in our test set. The top plot shows, for each of the matrices in the test set, the coefficient in front of the leading term N^3 in the expression for the computational cost. The bottom plot reports the relative improvement, measured in percent as

$$\frac{C_{\texttt{logm}} - C_{\texttt{logm\_ex}}}{C_{\texttt{logm\_ex}}} \cdot 100, \tag{34}$$

where $C_{\texttt{logm}}$ and $C_{\texttt{logm\_ex}}$ are the coefficient in front of N^3 in the experimental computational cost of `logm` and `logm_ex`, respectively. The matrices are sorted in descending

order by computational cost of `logm_amend`, `logm`, and `logm_ex`.

The algorithm `logm_amend` reaches the highest computational cost on all matrices in the test set. The algorithm `logm` has mostly the same cost as `logm_amend`, but requires an approximant of lower degree on one third of the test matrices.

The two new algorithms `logm_ex` and `logm_exm1` always have same computational cost, as they use the same strategy to select the number of square roots to take and the degree of the Padé approximant to use. According to the metric we consider, the new algorithms are always more efficient than both `logm` and `logm_amend`. The computational cost of `logm` is typically 20% to 40% (and up to 75%) higher than that of `logm_ex` and `logm_exm1`.

It is important to stress that this theoretical advantage does not translate into a performance benefit for our MATLAB codes. In fact, while `logm` evaluates the Padé approximant at a matrix argument using LAPACK and BLAS routines, the new algorithms rely on a MATLAB implementation of Algorithm 1, which is much slower than a BLAS-like implementation of the same algorithm would be.

9 Conclusions

We presented a new algorithm for solving the matrix equation $\widetilde{r}_m(X) = A$, where $\widetilde{r}_m$ is the diagonal approximant to the exponential at 0. This algorithm exploits the special structure of the coefficients of the rational function $\widetilde{r}_m$ to accelerate the solution of the matrix equation $\widetilde{r}_m(Y) = T$, where T is a block upper (quasi-)triangular matrix. We discussed how this new algorithm can be combined with a suitable adaptation of the strategy developed by Al-Mohy and Higham [2012] in order to develop two new variants of a dual inverse scaling and squaring algorithm for computing the matrix logarithm.

According to our experimental results, the new algorithms are essentially as accurate as the MATLAB function `logm`, occasionally even delivering solutions with a slightly smaller forward error, and are more efficient in terms of asymptotic computational cost.

This does not lead to a faster algorithm for our MATLAB implementations. The missing step to obtaining a truly competitive algorithm is a low-level implementation of Algorithm 1 in the style of the level 3 BLAS. This highly nontrivial task will be the subject of future work.

Acknowledgements

The authors thank the anonymous referees and Nicholas J. Higham for providing feedback and suggestions on early drafts of this manuscript.

References

Milton Abramowitz and Irene A. Stegun. *Handbook of Mathematical Functions with Formulas, Graphs, and Mathematical Tables*. Dover, New York, 10th edition, 1972. ISBN 0486612724.

Awad H. Al-Mohy and Nicholas J. Higham. Improved inverse scaling and squaring algorithms for the matrix logarithm. *SIAM J. Sci. Comput.*, 34(4):C153–C169, 2012. doi:10.1137/110852553.

Awad H. Al-Mohy, Nicholas J. Higham, and Samuel D. Relton. Computing the Fréchet derivative of the matrix logarithm and estimating the condition number. *SIAM J. Sci. Comput.*, 35(4):C394–C410, 2013. doi:10.1137/120885991.

George A. Baker, Jr. and Peter Graves-Morris. *Padé Approximants*, volume 59 of *Encyclopedia of Mathematics and Its Applications*. Cambridge University Press, Cambridge, UK, 2nd edition, 1996.

Åke Björck and Sven Hammarling. A Schur method for the square root of a matrix. *Linear Algebra Appl.*, 52/53:127–140, 1983. doi:10.1016/0024-3795(83)80010-X.

João R. Cardoso and Rui Ralha. Matrix arithmetic-geometric mean and the computation of the logarithm. *SIAM J. Matrix Anal. Appl.*, 37(2):719–743, 2016. doi:10.1137/140998226.

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. doi:10.1137/S0895479899364015.

Edvin Deadman, Nicholas J. Higham, and Rui Ralha. Blocked Schur algorithms for computing the matrix square root. In P. Manninen and P. Öster, editors, *Applied Parallel and Scientific Computing: 11th International Conference, PARA 2012, Helsinki, Finland*, volume 7782 of *Lecture Notes in Computer Science*, pages 171–182. „, 2013. doi:10.1007/978-3-642-36803-5_12.

Luca Dieci, Benedetta Morini, and Alessandra Papini. Computational techniques for real logarithms of matrices. *SIAM J. Matrix Anal. Appl.*, 17(3):570–593, 1996. doi:10.1137/S0895479894273614.

Elizabeth D. Dolan and Jorge J. Moré. Benchmarking optimization software with performance profiles. *Math. Prog.*, 91:201–213, 2002. doi:10.1007/s101070100263.

Jean-Claude Evard and Frank Uhlig. On the matrix equation $f(X) = A$. *Linear Algebra Appl.*, 162-164:447–519, 1992. doi:10.1016/0024-3795(92)90390-V.

Massimiliano Fasi. Optimality of the Paterson–Stockmeyer method for evaluating matrix polynomials and rational matrix functions. *Linear Algebra Appl.*, 574:182–200, 2019. ISSN 0024-3795. doi:10.1016/j.laa.2019.04.001.

Massimiliano Fasi and Nicholas J. Higham. Multiprecision algorithms for computing the matrix logarithm. *SIAM J. Matrix Anal. Appl.*, 39(1):472–491, 2018. doi:10.1137/17m1129866.

Massimiliano Fasi and Nicholas J. Higham. An arbitrary precision scaling and squaring algorithm for the matrix exponential. *SIAM J. Matrix Anal. Appl.*, 40(4):1233–1256, 2019. ISSN 1095-7162. doi:10.1137/18m1228876.

Massimiliano Fasi and Bruno Iannazzo. Computing primary solutions of equations involving primary matrix functions. *Linear Algebra Appl.*, 560:17–42, 2019. doi:10.1016/j.laa.2018.09.010.

Massimiliano Fasi and Bruno Iannazzo. Substitution algorithms for rational matrix equations. *Electron. Trans. Numer. Anal.*, 53:500–521, 2020. doi:10.1553/etna_vol53s500.

Massimiliano Fasi, Nicholas J. Higham, and Bruno Iannazzo. An algorithm for the matrix Lambert *W* function. *SIAM J. Matrix Anal. Appl.*, 36(2):669–685, 2015. doi:10.1137/140997610.

Walter Gautschi. *Numerical Analysis*. Birkhäuser, New York, NY, USA, 2nd edition, 2011. ISBN 0817682589, 9780817682583.

Kazushige Goto and Robert Van De Geijn. High-performance implementation of the level-3 BLAS. *ACM Trans. Math. Software*, 35(1):1–14, July 2008. ISSN 1557-7295. doi:10.1145/1377603.1377607.

William Grant Kirkland and Subhash C. Sinha. Symbolic computation of quantities associated with time-periodic dynamical systems. *J. Comput. Nonlinear Dynam.*, 11 (4), May 2016. ISSN 1555-1423. doi:10.1115/1.4033382.

Nicholas J. Higham. *Functions of Matrices: Theory and Computation*. Society for Industrial and Applied Mathematics, Philadelphia, PA, USA, 2008. ISBN 978-0-898716-46-7. doi:10.1137/1.9780898717778.

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. doi:10.1137/S0895479899356080.

Roger A. Horn and Charles R. Johnson. *Topics in Matrix Analysis*. Cambridge University Press, Cambridge, UK, 1991. doi:10.1017/CBO9780511840371.

Bruno Iannazzo and Margherita Porcelli. The Riemannian Barzilai–Borwein method with nonmonotone line search and the matrix geometric mean computation. *IMA J. Numer. Anal.*, 38:495–517, 2018. doi:10.1093/imanum/drx015.

Robert B. Israel, Jeffrey S. Rosenthal, and Jason Z. Wei. Finding generators for Markov chains via empirical transition matrices, with applications to credit ratings. *Math. Finance*, 11(2):245–265, 2001. doi:10.1111/1467-9965.00114.

Charles Kenney and Alan J. Laub. A Schur–Fréchet algorithm for computing the logarithm and exponential of a matrix. *SIAM J. Matrix Anal. Appl.*, 19(3):640–663, July 1998. ISSN 1095-7162. doi:10.1137/s0895479896300334.

Charles S. Kenney and Alan J. Laub. Condition estimates for matrix functions. *SIAM J. Matrix Anal. Appl.*, 10(2):191–209, 1989. doi:10.1137/0610014.

G.J. Lastman and Naresh K. Sinha. Infinite series for logarithm of matrix, applied to identification of linear continuous-time multivariable systems from discrete-time models. *Electron. Lett.*, 27(16):1468, 1991. ISSN 0013-5194. doi:10.1049/el:19910919.

Razvigor Ossikovski and Antonello De Martino. Differential Mueller matrix of a depolarizing homogeneous medium and its relation to the Mueller matrix logarithm. *J. Opt. Soc. Am. A*, 32:343–348, 2015. doi:10.1364/JOSAA.32.000343.

Hamidréza Ramézani and Jena Jeong. Non-linear elastic micro-dilatation theory: Matrix exponential function paradigm. *Int. J. Solids Struct.*, 67-68:1–26, 2015. doi:10.1016/j.ijsolstr.2015.02.008.

Jarek Rossignac and Álvar Vinacua. Steady affine motions and morphs. *ACM Trans. Graph.*, 30(5):1–16, October 2011. ISSN 1557-7368. doi:10.1145/2019627.2019635.

Edwarwd B. Saff and Richard S. Varga. On the zeros and poles of Padé approximants to e^z. II. In Edwarwd B. Saff and Richard S. Varga, editors, *Padé and Rational Approximation*, pages 195–213. Academic Press, 1977. ISBN 978-0-12-614150-4. doi:10.1016/B978-0-12-614150-4.50022-8.

Terence Tao. The standard branch of the matrix logarithm, May 2015. What's new, Terence Tao blog, https://terrytao.wordpress.com/2015/05/03/the-standard-branch-of-the-matrix-logarithm/.

Fuminori Tatsuoka, Tomohiro Sogabe, Yuto Miyatake, and Shao-Liang Zhang. Algorithms for the computation of the matrix logarithm based on the double exponential formula. *J. Comput. Appl. Math*, 373:112396, August 2020. ISSN 0377-0427. doi:10.1016/j.cam.2019.112396.

Charles Van Loan. A note on the evaluation of matrix polynomials. *IEEE Trans. Automat. Control*, 24(2):320–321, April 1979. ISSN 0018-9286. doi:10.1109/tac.1979.1102005.