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*‡

Abstract: 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 algorithm in the style of [M. Fasi and B. Iannazzo, *MIMS EPrint 2019.8*, 2019] which is tailored to the special structure of the approximants to the exponential. In terms of floating-point operations, the resulting method is cheaper than the state-of-the-art inverse scaling and squaring algorithm.

Keywords: Primary matrix function, matrix logarithm, inverse scaling and squaring algorithm, Schur form, Padé approximant

AMS Classification: 65F60, 15A16

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 complex strip $\{z \in \mathbb{C} : |\mathrm{Im}(z)| < \pi\}$. This solution, denoted by $\log A$, is the *principal logarithm* or the *standard branch of the logarithm* [10] of A. 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 $\{z \in \mathbb{C} : -\pi < \mathrm{Im}(z) \le \pi\}$.

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 [23]. In control theory, it is used to convert discrete-time linear dynamical systems to continuous-time state-space systems [28] and to compute the time-invariant component of the state transition matrix of ordinary differential equations with periodic time-varying coefficients [8]. Other recent applications include

*Version of 25th May 2020. **Funding:** The work of the first author was supported by MathWorks and the Royal Society. 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.

†Department of Mathematics, The University of Manchester, Oxford Road, Manchester M13 9PL, UK (massimiliano.fasi@manchester.ac.uk).

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

computing the Lambert W function [11] and the matrix geometric mean [7], and finding generators of Markov chains in finance [21] and sociology [33]. The matrix logarithm also appears in optics [12], mechanics [13], and computer graphics [16].

The principal logarithm of matrix can be defined as a primary matrix function [19, Chap. 11]. If A is diagonalizable, that is, there exists a nonsingular matrix M such that $M^{-1}AM = \operatorname{diag}(\lambda_1, \dots, \lambda_N)$, then

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

If A is nondiagonalizable, $\log A$ can be obtained from (2) by continuity [27], 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 the conditioning of M may potentially spoil the computation, leading to a forward error far larger than that predicted by the conditioning of the matrix logarithm itself. Due to this restriction, 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 [19, Chap. 11] for a survey and [9] and [2] for more recent examples. A well-established approach is the inverse scaling and squaring algorithm, a method initially proposed by Kenney and Laub [29] and further developed by several authors over the course of the last thirty years [15], [14], [20], [25], [6], [19, Chap. 11], [30]. 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 the $[m/m]$ diagonal 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 [5], [3], it has been shown that if T is a quasi-triangular matrix and r is a rational function, then computing a quasi-triangular solution Y to $r(Y) = T$ has the same asymptotic computational cost as evaluating $r(T)$. The use of quasi-triangular matrices is not a restriction: if a quasi-triangular matrix Y satisfies $r(Y) = T$, where $T = U^*AU$ is upper quasi-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 [5].

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 [17, Thm. 5.9.1]

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

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 of used in later sections. 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 4 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 5. The final section summarizes our contribution and outlines possible directions for future work.

2 Background

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 [19, 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 [26, 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 [5, Thm. 6].

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 [24].

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 [17, 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 function, 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 [4, Prop. 4], in section 3 we will consider only odd values of m.

3 Specialized substitution algorithms

We discuss numerical algorithms for the solution of the matrix equation

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

where p and q are coprime polynomials of degree $2\ell + 1$ for some positive integer ℓ. Then we will move to the special case $p(z) = q(-z)$, in order to take full advantage of the special structure (3) of the diagonal Padé approximants to the exponential.

As shown in [5, Prop. 9], if p and q are coprime then X satisfies (5) if and only if it satisfies $p(X) = Aq(X)$, thus we will 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 $\tau_1, \ldots, \tau_\nu$ along the main diagonal. Note that if Y is a primary solution to (6), then it has the same block structure as T.

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

$$g(Y^2) + Yh(Y^2) - Tg(Y^2) + TYh(Y^2) = 0, \tag{8}$$

where $g(x) = \sum_{k=0}^{\ell} \gamma_k x^k$ and $h(x) = \sum_{k=0}^{\ell} \delta_k x^k$. 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 (8)

Now we describe a technique for solving general equations of the type (6), where $T = (T_{ij})_{i,j=1,\ldots,\nu}$ and $Y = (Y_{ij})_{i,j=1,\ldots,\nu}$ are block upper triangular matrices with the same block structure. By

writing the equation at block level, we obtain the nonlinear system of equations

$$p(Y_{ii}) = T_{ii}q(Y_{ii}), \qquad i = 1, \dots, \nu, \tag{9}$$

$$L_{ij}(Y_{ij}) = b_{ij}, \qquad 1 \le i < j \le \nu, \tag{10}$$

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 $T_{\bar{i}\bar{j}}$ and $Y_{\bar{i}\bar{j}}$ such that either $\bar{i} = i$ and $\bar{j} < j$, or $\bar{i} > i$ and $\bar{j} = j$.

Equations (9) and (10) naturally lead to an algorithm for the solution of (6), as we now explain. 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 order for this algorithm to be effective, it is necessary to find efficient techniques to compute Y_{ii}, L_{ij}, and b_{ij}. When T is upper (quasi-)triangular, (9) is made of scalar equations or matrix equations with 2×2 unknowns and coefficients, whose solutions can be found by solving only scalar polynomial equations [5, Prop. 15]. The computation of L_{ij} and b_{ij} can be performed in a number of ways that are equivalent mathematically, but not numerically.

For several evaluation schemes it has been shown that the computational cost of computing Y by constructing L_{ij} and b_{ij} for all i and j is asymptotically the same as the cost of evaluating the rational function $r(z) := p(z)q(z)^{-1}$ 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: [5, Alg. 1] uses Horner's scheme, [3, Alg. 1] explicitly computes powers of A and combines them, whereas [3, Alg. 2] relies on the more powerful Paterson–Stockmeyer scheme for polynomial evaluation.

The applicability of the algorithms depends on the existence of solutions to the equations in (9) and to the nonsingularity of the operators L_{ij} in (10). It has been shown in [3, 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 denote an eigenvalue of Y_{ii} and Y_{jj}, respectively. In principle, any of [5, Alg. 1], [3, Alg. 1], or [3, Alg. 2] can be used to solve (8), but by exploiting the special structure of the numerator and denominator of this rational equation we can develop tailored algorithms that require fewer arithmetic operations than a straightforward application of the algorithms in [5] or [3].

Our algorithm for solving (8) is based on an evaluation scheme that uses only even powers of Y. This approach roughly halves the computational cost of solving (5) with respect to [3, Alg. 1], and for $\ell = 1, \dots, 5$ requires fewer matrix multiplications than [3, Alg. 2] or [5, Alg. 1].

For ℓ greater than 5 it is possible to use evaluation schemes that do not form all the powers of Y^2. In principle, for any such evaluation scheme, one could implement a "more complicated" structured algorithm for the solution of $r(Y) = T$ that roughly requires as many operations as the evaluation of $r(T)$. Such "optimal schemes" are not needed for the computation of the matrix logarithm in double precision floating-point arithmetic, as neither accuracy nor performance would improve if ℓ greater than 5 were used. Larger values may be of interest or even necessary in a multiprecision setting, but we will not discuss this aspect here.

3.2 Structured algorithm based on even powers

In analogy with [3, Eq. (3.6)], we define the sequence

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

where $Y^{[k]} = Y^{2k}$. The diagonal blocks of $Y^{[k]}$ can be computed by solving the equation $r(Y) = T_{ii}$, for $i = 1, \dots, v$. For the off-diagonal block we rewrite (8) as an equation involving only Y_{ij} and blocks of $g(Y)_{\overline{ij}}$, $h(Y)_{\overline{ij}}$, and $Y^{[k]}_{\overline{ij}}$ such that $\bar{\imath}+\bar{\jmath} < i+j$. 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*, assuming an algorithms that computes the blocks of Y one diagonal at a time from the main diagonal to the top right corner.

The block in position (i, j) of the sequence can be written, for $k = 2, \dots, \ell$, as

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

For $k > 2$, by substituting the formula for $Y^{[k-1]}_{ij}$ into that for $Y^{[k]}_{ij}$ one obtains a formula for the latter that involves only known quantities and the two blocks $Y^{[k-2]}_{ij}$ and $Y^{[1]}_{ij}$, and repeating the procedure we obtain, for $k = 2, \dots, \ell$,

$$Y^{[k]}_{ij} = \sum_{u=0}^{k-1} Y^{[u]}_{ii}Y^{[1]}_{ij}Y^{[k-u-1]}_{jj} + \Phi^{[k]}_{ij}, \qquad \Phi^{[k]}_{ij} := \sum_{u=0}^{k-2} Y^{[u]}_{ii}F^{[k-u]}_{ij}, \tag{11}$$

where $Y^{[k]}_{ij}$ is given in terms of $Y^{[1]}_{ij}$ and known quantities. Using (11) and setting $\Phi^{[1]}_{ij} = 0$, we can write, for $k = 1, \dots, \ell$,

$$Y^{[k]}_{ij} = E^{[k]}_{ij}(Y^{[1]}_{ij}) + \Phi^{[k]}_{ij}, \qquad E^{[k]}_{ij}(V) := \sum_{u=0}^{k-1} Y^{[u]}_{ii}VY^{[k-u-1]}_{jj}. \tag{12}$$

From (8), we have that

$$\begin{aligned}(I - T_{ii})g(Y^2)_{ij} + (I + T_{ii})\big(Y_{ii}h(Y^2)_{ij} + Y_{ij}h(Y^2)_{jj}\big) = \\ - (I + T_{ii}) \sum_{t=i+1}^{j-1} Y_{it}h(Y^2)_{tj} + \sum_{t=i+1}^{j} T_{it}Q_{tj},\end{aligned} \tag{13}$$

where $Q = g(Y^2) - Yh(Y^2)$, and the right hand side contains just known quantities. In order to isolate the terms containing Y_{ij} in the left hand side, we exploit (12) to obtain

$$\begin{aligned} g(Y^2)_{ij} &= \sum_{k=0}^{\ell} \gamma_k Y^{[k]}_{ij} = \sum_{k=1}^{\ell} \gamma_k E^{[k]}_{ij}(Y^{[1]}_{ij}) + \sum_{k=1}^{\ell} \gamma_k \Phi^{[k]}_{ij}, \\ h(Y^2)_{ij} &= \sum_{k=0}^{\ell} \delta_k Y^{[k]}_{ij} = \sum_{k=1}^{\ell} \delta_k E^{[k]}_{ij}(Y^{[1]}_{ij}) + \sum_{k=1}^{\ell} \delta_k \Phi^{[k]}_{ij}, \end{aligned}$$

and we note that

$$Y_{ij}^{[1]} = N_{ij}(Y_{ij}) + \widetilde{\Phi}_{ij}, \qquad N_{ij}(Y) := Y_{ii}Y + YY_{jj}, \qquad \widetilde{\Phi}_{ij} := \sum_{t=i+1}^{j-1} Y_{it}Y_{tj}. \tag{14}$$

Therefore, (8) is equivalent to an equation of the form

$$L_{ij}(Y_{ij}) = b_{ij}, \tag{15}$$

where

$$L_{ij}(Y_{ij}) := \sum_{k=1}^{\ell} \mu_k E_{ij}^{[k]}(N_{ij}(Y_{ij})) + (I_{\tau_i} + T_{ii})Y_{ij}h(Y^2)_{jj},$$

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

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

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 $\mathrm{vec}(A_1A_2A_3) = (A_3^T \otimes A_1)\mathrm{vec}(A_2)$ [27, Lemma 4.3.1], where $A \otimes B$ denotes the Kronecker product of the two matrices A and B.

By taking the vec of both sides of (15), we get

$$M_{ij}\mathrm{vec}(Y_{ij}) = \varphi_{ij}, \tag{16}$$

where

$$\begin{aligned}
M_{ij} &:= \sum_{k=1}^{\ell} \mu_k \widehat{E}_{ij}^{[k]} \widehat{N}_{ij} + h(Y^2)_{jj}^T \otimes (I_{\tau_i} + T_{ii}), \\
\varphi_{ij} &:= \chi_{ij} - \psi_{ij}, \\
\chi_{ij} &:= \mathrm{vec}\left(\sum_{t=i+1}^{j} T_{it}q(Y)_{tj} - (I_{\tau_i} + T_{ii}) \sum_{t=i+1}^{j-1} Y_{it}h(Y^2)_{tj} \right), \\
\psi_{ij} &:= \sum_{k=1}^{\ell} \mu_k \left(\widehat{E}_{ij}^{[k]}\, \mathrm{vec}\left(\widetilde{\Phi}_{ij}\right) + \mathrm{vec}\big(\Phi_{ij}^{[k]}\big) \right),
\end{aligned} \tag{17}$$

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

Using the techniques in [3, sect. 3], we can use (17) 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, using (16). 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

$$\Phi_{ij}^{[2]} = F_{ij}^{[2]}, \qquad \Phi_{ij}^{[k]} = F_{ij}^{[k]} + Y_{ii}^{[1]}\Phi_{ij}^{[k-1]}, \qquad k = 3, \dots, \ell,$$

$$\widehat{E}_{ij}^{[1]} = I_{\tau_i\tau_j}, \qquad \widehat{E}_{ij}^{[k]} = \big((Y_{jj}^{[1]})^T \otimes I_{\tau_i}\big)\widehat{E}_{ij}^{[k-1]} + I_{\tau_j} \otimes Y_{ii}^{[k-1]}, \qquad k = 2, \dots, \ell.$$

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**
    $Y_{ii}^{[k]} \leftarrow Y_{ii}^{[1]} Y_{ii}^{[k-1]}$
  $G_{ii} \leftarrow \sum_{k=0}^{\ell} \gamma_k Y_{ii}^{[k]}$
  $H_{ii} \leftarrow \sum_{k=0}^{\ell} \delta_k Y_{ii}^{[k]}$
  $Q_{ii} \leftarrow G_{ii} - Y_{ii} H_{ii}$
for $v = 1$ **to** $\nu - 1$ **do**
  for $i = 1$ **to** $\nu - v$ **do**
    $j \leftarrow i + v$
    for $k = 2$ **to** ℓ **do**
      $F_{ij}^{[k]} \leftarrow \sum_{t=i+1}^{j-1} Y_{it}^{[1]} Y_{tj}^{[k-u-1]}$
    $\widetilde{\Phi}_{ij} \leftarrow \sum_{t=i+1}^{j-1} Y_{it} Y_{tj}$
    $\Phi_{ij}^{[2]} \leftarrow F_{ij}^{[2]}$
    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**
      $Y_{ij}^{[k]} \leftarrow Y_{ii}^{[1]} Y_{ij}^{[k-1]} + Y_{ij}^{[1]} Y_{jj}^{[k-1]} + F_{ij}^{[k]}$
    $H_{ij} \leftarrow \sum_{k=1}^{\ell} \delta_k Y_{ij}^{[k]}$
    $Q_{ij} \leftarrow \sum_{k=1}^{\ell} \gamma_k Y_{ij}^{[k]} - K - Y_{ii} H_{ij} - Y_{ij} H_{jj}$
$X \leftarrow UYU^{-1}$

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 provide a technical lemma.

Lemma 1. *For the matrix M_{ij} in (17), one has that*

$$M_{ij} = \sum_{k=1}^{2\ell+1} c_k \widehat{B}_{ij}^{[k]} - (I \otimes T_{ii}) \sum_{k=1}^{2\ell+1} d_k \widehat{B}_{ij}^{[k]}, \qquad \widehat{B}_{ij}^{[k]} := \sum_{u=0}^{k-1} (Y_{jj}^{k-1-u})^T \otimes Y_{ii}^u. \tag{18}$$

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

Proof. First, observe that

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

Define

$$\begin{aligned}
\widetilde{P}_{ij} &:= \sum_{k=1}^{\ell} \big(I_{\tau_j} \otimes (\gamma_k I_{\tau_i} + \delta_k Y_{ii}) \big) \widehat{B}_{ij}^{[2k]} + h(Y^2)_{jj}^T \otimes I_{\tau_i}, \\
\widehat{P}_{ij} &:= \sum_{k=1}^{\ell} \big(I_{\tau_j} \otimes (\gamma_k I_{\tau_i} - \delta_k Y_{ii}) \big) \widehat{B}_{ij}^{[2k]} - h(Y^2)_{jj}^T \otimes I_{\tau_i}.
\end{aligned} \tag{19}$$

In view of (17), 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

$$(I_{\tau_j} \otimes \gamma_k I_{\tau_i}) \widehat{B}_{ij}^{[2k]} = c_{2k} \widehat{B}_{ij}^{[2k]}, \qquad k = 1, \ldots, \ell, \tag{20}$$

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

$$\begin{aligned}
&\sum_{k=1}^{\ell} \big(I_{\tau_j} \otimes \delta_k Y_{ii} \big) \widehat{B}_{ij}^{[2k]} + h(Y^2)_{jj}^T \otimes I_{\tau_i} \\
&\qquad = \sum_{k=1}^{\ell} c_{2k+1} \big((I_{\tau_j} \otimes Y_{ii}) \widehat{B}_{ij}^{[2k]} + (Y_{jj}^{2k})^T \otimes I_{\tau_i} \big) + c_1 I_{\tau_i \tau_j} = \sum_{k=0}^{\ell} c_{2k+1} \widehat{B}_{ij}^{[2k+1]},
\end{aligned} \tag{21}$$

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

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

Equations (20) and (21) together show the first equality in (19). The corresponding equality for $\widehat{P}_{ij}$ can be proved analogously, and this concludes the proof. □

Now we can state the applicability theorem.

Theorem 2. *Let g and 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 (16) 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 [3, Thm. 3.4] once one observes that the matrix M_{ij} in (18), which determines the applicability of Algorithm 1, is the same as that in [3, Eq. (3.15)], which determines the applicability of [3, Alg. 1]. □

Implementation details Cancellation can occur in the computation of the diagonal elements 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 (18) as

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

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 $u = 1, \dots, \ell$;
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.

Computational cost As in the other substitution algorithms for (quasi-)triangular matrices, the computational cost is related to the number of matrix multiplication needed by the evaluation scheme on which the algorithm is based. For a triangular matrix, the most expensive steps of Algorithm 1 with respect to the size $\nu = N$ are the computation of $F_{ij}^{[k]}$, for $k = 1, \dots, \ell$, and the two sums on the right-hand side of (13). The total cost of the algorithm is $\frac{\ell+2}{3}\nu^3 + o(\nu^3)$ operations. Since $\mu = \max\{m, n\} = 2\ell + 1$, we can write the cost as $\frac{\mu+3}{6}\nu^3 + o(\nu^3)$. Therefore Algorithm 1 is asymptotically cheaper than [3, Alg. 1] for any m and not more expensive than [3, Alg. 2] for $m \leq 15$.

4 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. For instance, the functions `logm` of MATLAB and `Base.log` of Julia are based on the inverse scaling and squaring method as described in [15, Alg. 4.1] and [14, Alg. 6.1], whereas the function `logm` of Octave implements the variant in [19, 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 by $r_m(B - I)$, where r_m is the $[m/m]$ diagonal Padé approximant to $\log z$ at $z = 1$. Finally, the logarithm of A is recovered through $\log A = U(\log T)U^* = U(2^s \log B)U^* \approx U\big(2^s r_m(B - I)\big)U^*$.

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

$$\frac{\|\Delta_m\|}{\|B - I\|}, \tag{22}$$

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 (22) by

$$\frac{\|\Delta_m\|}{\|B - I\|} \leq \sum_{k=2m+1}^{\infty} |\widehat{\delta}_k| \alpha_p(B - I)^k =: F_m(\alpha_p(B - I)), \tag{23}$$

where

$$\alpha_p(X) = \max\{\|X^p\|^{1/p}, \|X^{p+1}\|^{1/(p+1)}\} \tag{24}$$

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

The coefficients of the series (23) 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

$$\theta_i^F = \max_{\theta \in \mathbb{R}^+}\{F_i(\theta) \leq u\}. \tag{25}$$

Note that $\alpha_p(B - I) \leq \theta_i^F$ guarantees that $\|\Delta_i\|/\|B - I\| \leq u$. 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 comment on the choice of the number of square roots s and the degree of the approximation m. 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

Tab. 1: The values of θ_m in [15, 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	$3.666532675959788 \times 10^{-1}$	$3.287599317808182 \times 10^{-1}$

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 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

$$\alpha_p(A^{1/2} - I) \approx \frac{\alpha_p(A-I)}{2}, \tag{26}$$

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 checked 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 the 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 [15].

As noted in [19, 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 for larger values 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.

Remark 1. *The values of θ_m in [15, Table 2.1] for double precision are computed using an expansion different from that in (23). The values in [15, Table 2.1] and the correct value of these constants for double precision are reported in the first and second column of Table 1, respectively.*

4.1 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 $\log B$, which relies on solving 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 [32, Thm. 2.2]

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

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, $|e^{\widetilde{r}_{mn}^{-1}(z)} - z|/|z| \leq \varepsilon$ 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 $\varepsilon_{mn}(z) := e^{\widetilde{r}_{mn}^{-1}(z)}/z - 1$ can be seen as a relative backward truncation error, and in a neighborhood of 1 we have the series expansion

$$\varepsilon_{mn}(z) = \sum_{k=m+n+1}^{\infty} \widehat{\delta}_k (z-1)^k. \tag{27}$$

The coefficients in (27) can be easily computed from those of the series expansion

$$\exp(\widetilde{r}_{mn}^{-1}(z)) - z = \sum_{k=0}^{\infty} \beta_k (z-1)^k, \tag{28}$$

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 [34, 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 (27) 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 [3, 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 2.

The backward truncation error in the approximation of $\log B$ by means of the inverse of $\widetilde{r}_{mn}$ is given by the matrix $\Delta_{mn} = e^X - B = \exp(\widetilde{r}_{mn}^{-1}(B)) - B \in \mathbb{C}^{N\times N}$, which satisfies $X = \log(B + \Delta_{mn})$. If the eigenvalues of B are within the radius of convergence of the series (27), we can write

$$\begin{aligned} \frac{\|\Delta_{mn}\|}{\|B\|} &\leq \left\| \sum_{k=m+n+1}^{\infty} \widehat{\delta}_k (B-I)^k \right\| \\ &\leq \sum_{k=m+n+1}^{\infty} |\widehat{\delta}_k| \alpha_p(B-I)^k \\ &=: G_{mn}(\alpha_p(B-I)), \end{aligned} \tag{29}$$

where the function α_p is defined in (24). Note that unlike [15], we divide by the norm of B instead of $B - I$ (compare equation (23)), as the function we are approximating is in fact $\log B$.

In analogy with the algorithm of Al-Mohy and Higham [15], we compute

$$\theta_i^G = \max_{\theta \in \mathbb{R}^+} \{G_{ii}(\theta) \leq u\}, \tag{30}$$

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

Tab. 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 (30).

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 analysis by Al-Mohy and Higham [15], we conjecture that the radius of convergence of G_{ii} is indeed larger than θ^G_i, which seems to be the case numerically.

A variant of this algorithm is obtained by considering the $[m/n]$ Padé approximant to $e^z - 1$ 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 (29), can equivalently be written as

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

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

4.2 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 be using. A closer look at Table 2 reveals two important points. First, since θ^G_m must be below 1 and $\theta^G_{m_5} = \theta^G_7 > 0.5$, we are guaranteed that $\theta^G_{m_j} < 2\theta^G_{m_{j-2}}$ 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, $\theta^G_{m_{j-1}} > 2\theta^G_{m_{j-2}}$ for $j \leq 6$, thus if $\alpha_p(B - I)$ is smaller than $\theta^G_{m_j}$, in view of the estimate (26) 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 [19, p. 245].

The pseudocode of our strategy for computing the matrix logarithm using the analysis in section 4.1 is given in Algorithm 2. The algorithm begins by computing the Schur decomposition $A =: UTU^*$, and then uses the algorithm by Björck and Hammarling [31] to take square roots of T, s of them say, until the spectral radius of $T^{2^{-s}} - I$ becomes smaller than θ^G_9. 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 $\alpha_p(T^{2^{-s}} - I)$ to be smaller than θ^G_9. 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 $\widetilde{r}_m^{-1}(T^{2^{-s}} - I)$ is smaller than u, using the error bound (29) and the

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**
  $\mu_3 \leftarrow \texttt{normest}(B - I, 3)^{1/3}$
  $\mu_4 \leftarrow \texttt{normest}(B - I, 4)^{1/4}$
  $a_3 \leftarrow \max\{\mu_3, \mu_4\}$
  $i \leftarrow 9$
  while $i \geq 3$ **do**
    if $a_3 \leq \theta_i^G$ **then**
      $m \leftarrow i$
    $i \leftarrow i - 2$
  if $m = 0$ **then**
    $\mu_5 \leftarrow \texttt{normest}(B - I, 5)^{1/5}$
    $a_4 \leftarrow \max\{\mu_4, \mu_5\}$
    $\zeta \leftarrow \min(a_3, a_4)$
    if $\zeta \leq \theta_9^G$ **then**
      if $\zeta \leq \theta_7^G$ **then**
        $m \leftarrow 7$
      else
        $m \leftarrow 9$
    else
      $B \leftarrow B^{1/2}$
      $s \leftarrow s + 1$
$Y \leftarrow 2^s U \widetilde{r}_m^{-1}(B) U^*$

values in Table 2. Since only an estimate of $\alpha_p(T^{2^{-s}} - I)$ is needed, we estimate the 1-norm of powers of $X \in \mathbb{C}^{N\times N}$ with the algorithm of Higham and Tisseur [22], 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, $\texttt{normest}(X, k)$ 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 $\alpha_2(T^{2^{-s}} - I)$. Therefore, our algorithm computes a_3, an estimate of $\alpha_3(T^{2^{-s}} - I)$, 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 $\alpha_4(T^{2^{-s}} - I)$, 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 $G_{mm}(\alpha_p(T^{2^{-s}} - I)) < u$ is found, the algorithm solves the matrix equation $\widetilde{r}_m(X) = T^{2^{-s}}$. The ith diagonal block of X is chosen as the solutions to the matrix equation $\widetilde{r}_m(X_{ii}) = T_{ii}^{2^{-s}}$ that is closer to $\log T_{ii}^{2^{-s}}$ in norm. Finally the approximation $2^s U X U^*$ is returned.

4.3 Alternative choice for the diagonal blocks of X

The solution of the equation $\widetilde{r}_m(X) = B$ on line 29 of Algorithm 2 has diagonal elements $\widetilde{r}_m^{-1}(B_{11}), \dots, \widetilde{r}_m^{-1}(B_{vv})$, where $B_{11}, \dots, B_{vv}$ are the v diagonal blocks of $T^{2^{-s}}$. 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 $\log B_{11}, \dots, \log B_{vv}$ and using these values in the subsequent substitution step. The whole procedure can then be interpreted as applying Algorithm 2 to the matrix equation

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

5 Numerical experiments

In this section we assess experimentally the accuracy and performance of the algorithms discussed in section 4 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 I5-6500 processor, running at 3.20GHz. We use a test set of 62 nonnormal matrices from the literature of the matrix logarithm [15], [6]. These matrices have size ranging between 2 and 13, and their spectra do not contain any real nonpositive eigenvalues.

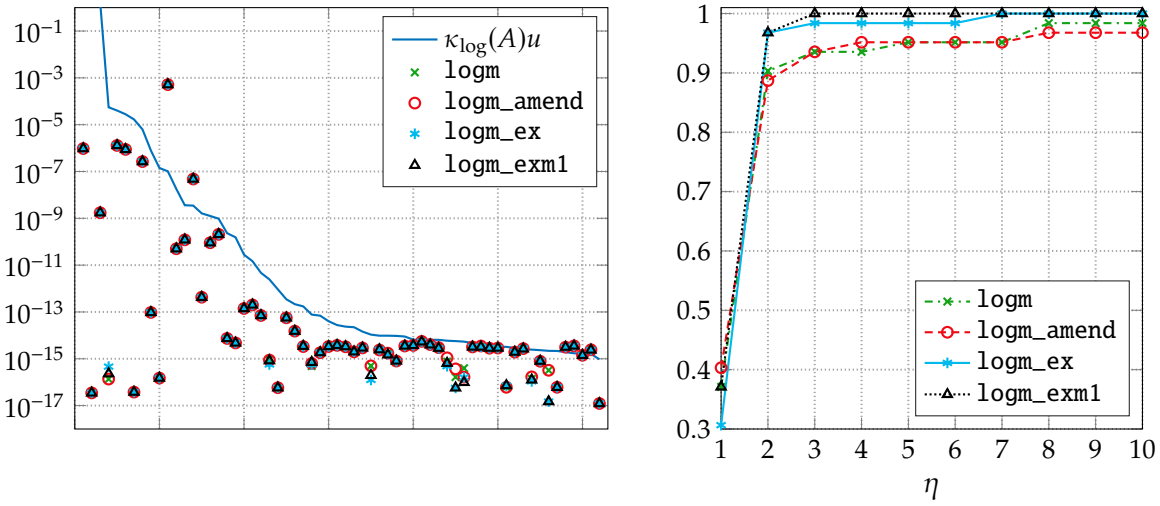
In the tests, we consider four implementations:

- `logm`: the built-in MATLAB function that combines the inverse scaling and squaring algorithms for complex and real matrices from [15] and [14], 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 4.3 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 4.3 for the choice of the diagonal blocks.

The accuracy of a computed solution $\widetilde{X}$ is assessed by means of 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 [1] 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 $\kappa_{\log}(A)$ and $u = 2^{-53}$ are the 1-norm condition number of the logarithm of A and the unit roundoff of IEEE double precision arithmetic, respectively.

5.1 Accuracy of the matrix logarithm

Figure 1 illustrates the accuracy of `logm`, `logm_amend`, `logm_ex`, and `logm_ex1`. In Figure 1a 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 1b reports the same data by means of performance profiles.



(a) Forward error.

(b) Performance profile for (a).

Fig. 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.

Our new algorithms appear to be as forward stable as the MATLAB function `logm`, and in fact the forward error is essentially the same, with occasional differences when the forward error is smaller than $\kappa_{\log}(A)u$.

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.

5.2 Computational cost of 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, for each matrix, the relative improvement, measured in percent as

$$\frac{C_{\texttt{logm}} - C_{\texttt{logm\_ex}}}{C_{\texttt{logm\_ex}}} \cdot 100, \tag{31}$$

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, these are always more efficient

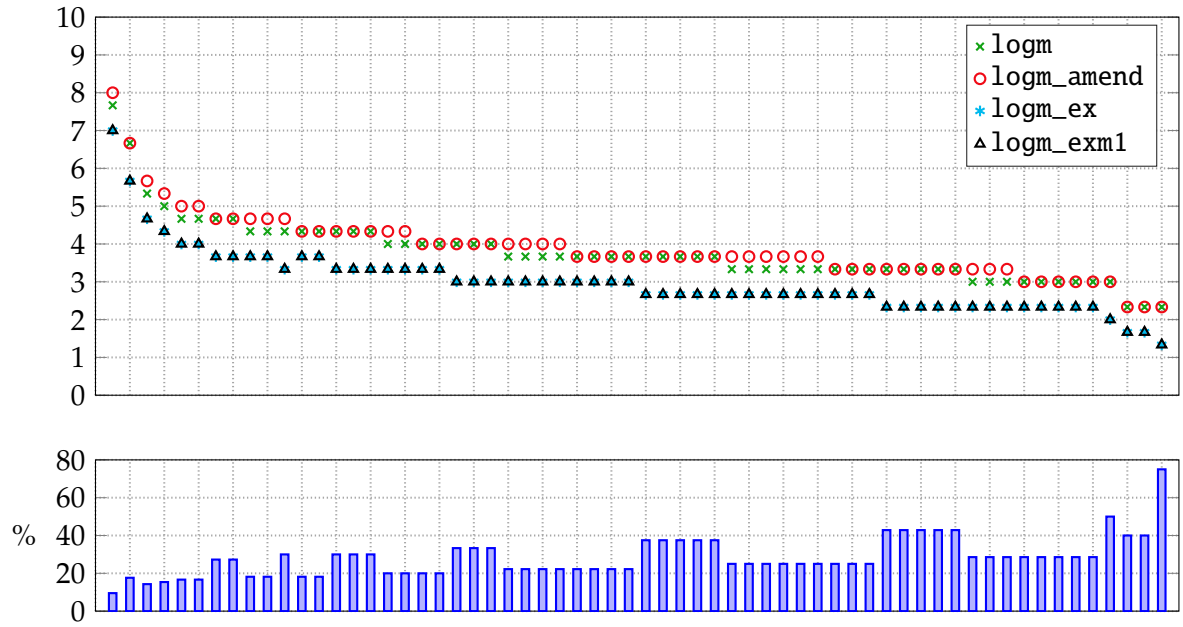


Fig. 2: Top: computational cost of `logm`, `logm_amend`, `logm_ex`, and `logm_ex1` on the matrices in the test set. Bottom: relative gain of `logm_ex` with respect to `logm`, according to the formula in (31).

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.

6 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 functions $\widetilde{r}_m$ 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 [15] 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`, despite occasionally delivering a solution with a slightly smaller forward error, but are more efficient in terms of asymptotic computational cost.

This does not lead to a faster algorithm in the case of 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 [18]. This highly nontrivial task will be the subject of future work.

Acknowledgements

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

References

[1] Multiprecision Computing Toolbox. Advanpix, Tokyo. http://www.advanpix.com.

[2] F. Tatsuoka, T. Sogabe, Y. Miyatake, and S.-L. Zhang. Algorithms for the computation of the matrix logarithm based on the double exponential formula. *J. Comput. Appl. Math*, 373: 112396, 2020.

[3] M. Fasi and B. Iannazzo. Substitution algorithms for rational matrix equations. MIMS EPrint 2019.8, Manchester Institute for Mathematical Sciences, The University of Manchester, UK, Dec 2019. To appear in *Electron. Trans. Numer. Anal.*.

[4] M. Fasi. Optimality of the Paterson–Stockmeyer method for evaluating matrix polynomials and rational matrix functions. *Linear Algebra Appl.*, 574:182–200, 2019.

[5] M. Fasi and B. Iannazzo. Computing primary solutions of equations involving primary matrix functions. *Linear Algebra Appl.*, 560:17–42, 2019.

[6] M. Fasi and N. J. Higham. Multiprecision algorithms for computing the matrix logarithm. *SIAM J. Matrix Anal. Appl.*, 39(1):472–491, 2018.

[7] B. Iannazzo and M. Porcelli. The Riemannian Barzilai–Borwein method with nonmonotone line search and the matrix geometric mean computation. *IMA J. Numer. Anal.*, 38:495–517, 2018.

[8] W. Grant Kirkland and S. C. Sinha. Symbolic computation of quantities associated with time-periodic dynamical systems. *J. Comput. Nonlinear Dynam.*, 11(4), 2016.

[9] J. R. Cardoso and R. Ralha. Matrix arithmetic-geometric mean and the computation of the logarithm. *SIAM J. Matrix Anal. Appl.*, 37(2):719–743, 2016.

[10] T. 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/.

[11] M. Fasi, N. J. Higham, and B. Iannazzo. An algorithm for the matrix Lambert *W* function. *SIAM J. Matrix Anal. Appl.*, 36(2):669–685, 2015.

[12] R. Ossikovski and A. 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.

[13] H. Ramézani and J. Jeong. Non-linear elastic micro-dilatation theory: Matrix exponential function paradigm. *Int. J. Solids Struct.*, 67-68:1–26, 2015.

[14] A. H. Al-Mohy, N. J. Higham, and S. D. Relton. Computing the Fréchet derivative of the matrix logarithm and estimating the condition number. *SIAM J. Sci. Comput.*, 35(4): C394–C410, 2013.

[15] A. H. Al-Mohy and N. J. Higham. Improved inverse scaling and squaring algorithms for the matrix logarithm. *SIAM J. Sci. Comput.*, 34(4):C153–C169, 2012.

[16] J. Rossignac and Á. Vinacua. Steady affine motions and morphs. *ACM Trans. Graph.*, 30 (5):1–16, 2011.

[17] W. Gautschi. *Numerical Analysis*. 2nd edition, Birkhäuser, New York, NY, USA, 2011. ISBN 0817682589, 9780817682583.

[18] K. Goto and R. Van De Geijn. High-performance implementation of the level-3 BLAS. *ACM Trans. Math. Software*, 35(1):1–14, 2008.

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

[20] S. H. Cheng, N. J. Higham, C. S. Kenney, and A. J. Laub. Approximating the logarithm of a matrix to specified accuracy. *SIAM J. Matrix Anal. Appl.*, 22(4):1112–1125, 2001.

[21] R. B. Israel, J. S. Rosenthal, and J. Z. Wei. Finding generators for Markov chains via empirical transition matrices, with applications to credit ratings. *Math. Finance*, 11(2): 245–265, 2001.

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

[23] C. S. Kenney and A. J. Laub. A Schur–Fréchet algorithm for computing the logarithm and exponential of a matrix. *SIAM J. Matrix Anal. Appl.*, 19(3):640–663, 1998.

[24] G. A. Baker, Jr. and P. Graves-Morris. *Padé Approximants*, volume 59 of *Encyclopedia of Mathematics and Its Applications*. 2nd edition, Cambridge University Press, Cambridge, UK, 1996. xiv+746 pp.

[25] L. Dieci, B. Morini, and A. Papini. Computational techniques for real logarithms of matrices. *SIAM J. Matrix Anal. Appl.*, 17(3):570–593, 1996.

[26] J.-C. Evard and F. Uhlig. On the matrix equation $f(X) = A$. *Linear Algebra Appl.*, 162-164: 447–519, 1992.

[27] R. A. Horn and C. R. Johnson. *Topics in Matrix Analysis*. Cambridge University Press, Cambridge, UK, 1991.

[28] G. Lastman and N. 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.

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

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

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

[32] E. B. Saff and R. S. Varga. On the zeros and poles of Padé approximants to e^z. II. In *Padé and Rational Approximation*, E. B. Saff and R. S. Varga, editors, Academic Press, 1977, pages 195–213.

[33] B. Singer and S. Spilerman. The representation of social processes by Markov models. *Am. J. Sociol.*, 82(1):1–54, 1976.

[34] M. Abramowitz and I. A. Stegun. *Handbook of Mathematical Functions with Formulas, Graphs, and Mathematical Tables*. 10th edition, Dover, New York, 1972. ISBN 0486612724.