Electronic Transactions on Numerical Analysis.
Volume 53, pp. 500–521, 2020.
Copyright © 2020, Kent State University.
ISSN 1068–9613.
DOI: 10.1553/etna_vol53s500

ETNA
Kent State University and
Johann Radon Institute (RICAM)

SUBSTITUTION ALGORITHMS FOR RATIONAL MATRIX EQUATIONS*

MASSIMILIANO FASI† AND BRUNO IANNAZZO‡

Abstract. We study equations of the form $r(X) = A$, where r is a rational function and A and X are square matrices of the same size. We develop two techniques for solving these equations by inverting (through a substitution strategy) two schemes for the evaluation of rational functions of matrices. For triangular matrices, the new methods yield the same computational cost as the evaluation schemes from which they are obtained. A general equation can be reduced to upper triangular form by exploiting the Schur decomposition of the given matrix. For real data, the algorithms rely on the real Schur decomposition in order to compute real solutions using only real arithmetic. Numerical experiments show that our implementations are faster than existing alternatives without sacrificing accuracy.

Key words. rational matrix equation, Paterson–Stockmeyer scheme, powering technique, rational function evaluation, primary matrix function

AMS subject classifications. 15A24, 65F60

1. Introduction.

We consider rational matrix equations of the form

$$r(X) = A, \tag{1.1}$$

where $r = p/q$ with p and q coprime polynomials of degree m and n, respectively, and A and X are square matrices of the same size.

The most prominent instance of (1.1) is arguably the equation $X^\alpha = A$, defining the matrix αth root [6, 23, 24, 30], which finds applications in, for example, sociology [29], economy [7, 26], healthcare [9], and ecology [31].

This equation belongs to the more general class of functional matrix equations of the form $f(X) = A$, where f is a complex analytic function applied to a matrix (in the sense of primary matrix functions as defined in [22, Chap. 1]). The latter class has been extensively studied in the literature, and a number of theoretical results, such as existence, uniqueness, and a classification of real and complex solutions, are discussed by Evard and Uhlig [13]. To the best of our knowledge, no general algorithm for dealing with the equation $f(X) = A$ exists, even though algorithms tailored to most special cases of interest are available.

A reliable algorithm for the numerical solution of rational matrix equations of the type (1.1) has been recently proposed [16]. This method based on a substitution (or recursion) procedure is the culmination of a line of algorithms for the computation of matrix roots [17, 24, 30]. The common idea of these techniques is to reduce the problem to an equation of the type

$$r(Y) = T, \tag{1.2}$$

where Y and T are block upper triangular matrices with the same block structure, and then operate at the block level. Once the problem is turned to block triangular form, one discovers that the diagonal blocks can be computed by solving smaller equations of the same type, while

*Received September 14, 2019. Accepted March 3, 2020. Published online on August 18, 2020. Recommended by Marco Donatelli. 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 2018. 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).

those in the upper triangular part, if computed in a certain order, are determined by an explicit formula involving only blocks that have already been calculated.

Because of the way the explicit formulae for the strictly upper triangular blocks are derived, these algorithms can also be interpreted as being based on a recursion. For the αth root, for instance, Smith [30] obtains a substitution algorithm by symbolically constructing, with $\alpha - 1$ matrix multiplications, the first α powers of the solution Y, namely

$$Y^2, Y^3, \ldots, Y^{\alpha-1}, Y^\alpha = T,$$

and deducing an explicit expression for the blocks of Y. We call these $\alpha - 1$ powers the *stages* of the algorithm. The number of stages is related to the asymptotic computational cost of the method, which is of the order of $\frac{1}{3}(\alpha - 1)N^3 + o(N^3)$ operations for a triangular matrix of order N when the complex Schur form is used to reduce (1.1) to (1.2). A variant, proposed by Iannazzo and Manasse [24] (as an update of prior work by Greco and Iannazzo [17]), relies on a binary powering technique to form Y^α with no more than $2\lfloor \log_2 \alpha \rfloor$ matrix multiplications, thereby reducing the cost of computing the αth root to $\frac{2}{3}\lfloor \log_2 \alpha \rfloor N^3 + o(N^3)$ operations.

Similarly, Fasi and Iannazzo [16] obtain an explicit expression for the off-diagonal blocks of Y in (1.2). Their algorithm is obtained by reverting a two-step scheme that first evaluates $p(x)$ and $q(x)$ by using Horner's rule and then $r(x)$ as $p(x)/q(x)$. In the matrix case, this amounts to evaluating the two polynomials p and q at a matrix argument X and then solving the multiple right-hand side linear system $q(X)^{-1}p(X)$. The resulting algorithm solves equation (1.1), for a triangular matrix of order N, with $\frac{1}{3}(m + n - 1)N^3 + o(N^3)$ operations and is thus as expensive as Horner's algorithm for evaluating $r(X)$.

Rational functions can, however, be evaluated in several ways, and at a matrix argument some alternatives may require fewer matrix multiplications than applying Horner's method twice. This is appealing as asymptotically these are the most expensive operations that an evaluation scheme is expected to perform. A trivial alternative, for instance, is to evaluate all the powers of x that are needed and then combine them to get $p(x)$ and $q(x)$. In the scalar case, this would offer no benefit over using Horner's rule twice, but in the matrix case it typically leads to performing fewer matrix multiplications at the price of more matrix additions. An ever better example is the Paterson–Stockmeyer method [28], which for polynomials of high degree can require considerably fewer matrix multiplications than both Horner's method and the explicit powering strategy.

We have observed that both evaluation schemes, for upper (quasi-)triangular matrices, can be reverted yielding algorithms for the solution of $r(X) = A$ with the same cost as the evaluation itself and thus requiring fewer operations than the algorithms in [16].

The main contribution of the paper are two substitution algorithms for computing primary solutions to (1.1), one based on the explicit powering technique and the other on the Paterson–Stockmeyer scheme. As it is the case for the approach based on Horner's method [16], for upper (quasi-)triangular matrices, the asymptotic computational cost of these substitution algorithms is the same as that of the corresponding evaluation scheme. Therefore, the new algorithms require considerably fewer operations than those in [16] to solve (1.1).

The paper also provides a theoretical contribution. It has been shown [16, Cor. 14] that if a solution to (1.1) with a chosen set of eigenvalues exists, then the algorithm based on Horner's method can compute it if and only if it is *isolated*, that is, there exists a neighborhood containing only X. Here we extend this result and show that requiring the existence of that solution is not necessary: if $\lambda_1, \ldots, \lambda_n$ are the eigenvalues of A and we select, for each i, a solution ξ_i to the scalar equation $r(\xi_i) = \lambda_i$ such that the divided differences $r[\xi_i, \xi_j]$ are nonzero for each $i \neq j$, then there exists a unique solution to (1.1) with eigenvalues $\xi_1, \ldots, \xi_n$, which is isolated and can be computed by any of our substitution algorithms, including [16, Alg. 1]. The precise statement of this result is given in Theorem 3.3.

A possible application of these techniques is the computation of more general matrix functions defined implicitly by equations of the form $f(X) = A$, where f is an analytic function. As an example, we develop an algorithm for computing the Lambert W function which, in a number of cases, is faster and more accurate than the reference algorithm [15, Alg. 1].

We begin the paper by recalling, in Section 2, schemes for the evaluation of matrix rational functions and some of the results from the literature of matrix equations [13, 16]. We present our new algorithms in Section 3 and evaluate them experimentally in Section 4. Finally, in Section 5 we summarize our contribution and discuss possible directions for future work.

2. Background and notation. We denote by $\mathbb{C}_k[z]$ the vector space of complex polynomials of the complex variable z of degree at most k. In the remainder, we always refer to the $[m/n]$ rational function

$$r(z) = q(z)^{-1}p(z),$$

whose numerator and denominator are the polynomials

$$p(z) := \sum_{k=0}^{m} c_k z^k \in \mathbb{C}_m[z], \qquad \text{and} \qquad q(z) := \sum_{k=0}^{n} d_k z^k \in \mathbb{C}_n[z],$$

respectively. Without restriction, we assume that p and q are coprime, that is, they have no roots in common, and that c_m and d_n are not zero.

Let $f : \Omega \to \mathbb{C}$, where $\Omega \subset \mathbb{C}$, and let $x, y \in \Omega$. We denote by $f[x, y]$ the divided difference operator defined by

$$f[x,y] = \begin{cases} f'(x), & x = y, \\ \dfrac{f(x) - f(y)}{x - y}, & x \neq y, \end{cases}$$

which requires f to be differentiable at x, when $x = y$.

Evaluation of rational functions of matrices. The obvious strategy to evaluate the rational function $r(A) = q(A)^{-1}p(A)$ is to first compute $P := p(A)$ and $Q := q(A)$ reusing as much computation as possible and then solve the multiple right-hand side linear system $Qr(A) = P$. If this technique is used, then the scheme to evaluate $r(A)$ is entirely determined by the strategy chosen to evaluate the two polynomials.

The most straightforward way of evaluating $p(A)$ is to explicitly compute $I, A, \dots, A^m$ and then take their linear combination. This algorithm requires $m - 1$ matrix multiplication and the storage of one additional matrix, thus if $p(A)$ and $q(A)$ are evaluated together, then computing $r(A)$ requires the solution of one multiple right-hand side linear system and $\max\{m, n\} - 1$ matrix multiplications.

A more expensive algorithm is obtained if $p(A)$ and $q(A)$ are evaluated by using Horner's method, which we now briefly recall. Let us define, for $j = 0, \dots, m$, the polynomials $p^{[j]}(A) = \sum_{i=0}^{m-j} c_{i+j}A^i = \sum_{i=j}^{m} c_i A^{i-j}$. By observing that $p^{[0]}(A) = p(A)$, we can evaluate $p(A)$ by using the recursion

$$\begin{cases} p^{[m]}(A) = c_m I, \\ p^{[j]}(A) = Ap^{[j+1]}(A) + c_j I, \qquad \text{for } j = m-1, m-2, \dots, 0, \end{cases} \tag{2.1}$$

and in a similar manner we can evaluate $q(A)$ by defining $q^{[j]}(A)$, for $j = 0, \dots, n$. In this case, the evaluation of $r(A)$ requires the solution of one multiple right-hand side linear system

ETNA
Kent State University and
Johann Radon Institute (RICAM)

and $m+n-2$ matrix multiplications, which for $m=n$ is exactly twice as much as the algorithm based on explicit powers.

Finally, we summarize the Paterson–Stockmeyer scheme for polynomial evaluation [28]. For any positive integer s, the polynomial $p(A)$ can be rewritten as

$$p(A) = \sum_{k=0}^{\widetilde{r}} (A^s)^k C_k(A), \qquad \widetilde{r} = \left\lceil \frac{m}{s} - 1 \right\rceil, \tag{2.2}$$

where

$$C_k(A) = \sum_{u=0}^{\eta_k} c_{sk+u} A^u, \qquad \eta_k = \begin{cases} s-1, & 0 \le k < \widetilde{r}, \\ m - s\widetilde{r}, & k = \widetilde{r}. \end{cases}$$

An analogous rewriting for $q(A)$ yields

$$q(A) = \sum_{h=0}^{\widehat{r}} (A^s)^h D_h(A), \qquad \widehat{r} = \left\lceil \frac{n}{s} - 1 \right\rceil,$$

where D_h is defined as C_k but using the coefficients d_k of q and replacing m and $\widetilde{r}$ with n and $\widehat{r}$, respectively. For the sake of simplicity, we adopt a primed sum notation for C_k and D_h and write

$$C_k(A) =: {\sum_{u=0}^{s-1}}' c_{sk+u} A^u, \qquad k = 0, \dots, \widetilde{r},$$

$$D_h(A) =: {\sum_{u=0}^{s-1}}' d_{sh+u} A^u, \qquad h = 0, \dots, \widehat{r}.$$

In other words, the primed sum coincides with the usual sum when $k < \widetilde{r}$ and $h < \widehat{r}$, whereas for $k = \widetilde{r}$ and $h = \widehat{r}$ it denotes the sum up to $m - s\widetilde{r}$ and $n - s\widehat{r}$, respectively.

The popularity of this nonobvious scheme stems from the fact that it allows an efficient evaluation of matrix polynomials and, in our case, matrix rational functions. In particular, if $A^2, \dots, A^s$ are evaluated by successive multiplications by A and stored in memory, then evaluating $r(A)$ requires one matrix inversion and $L_s(m,n)$ matrix multiplications, where

$$L_s(m,n) = s - 1 + \widetilde{r} + \widehat{r}.$$

The function $L_s(m,n)$ is approximately minimized by taking either $s = \lfloor \sqrt{m+n} \rfloor$ or $s = \lceil \sqrt{m+n} \rceil$.

In order to achieve this computational cost, the expression in (2.2) should be evaluated à la Horner by constructing the sequence

$$\begin{cases} P^{[\widetilde{r}]}(A) = C_{\widetilde{r}}(A), \\ P^{[j]}(A) = A^s P^{[j+1]}(A) + C_j(A), \qquad j = \widetilde{r}-1, \widetilde{r}-2, \dots, 0, \end{cases} \tag{2.3}$$

which allows one to compute $p(A) = P^{[0]}(A)$ with $\widetilde{r}$ matrix multiplications once the powers of A have been formed. A similar argument shows that $q(A)$ can be evaluated with $\widehat{r}$ matrix multiplications yielding a total of $L_s(m,n)$ matrix multiplications for the entire procedure. In the following, for theoretical purposes, we will use the identities

$$P^{[t]}(z) = \sum_{k=t}^{\widetilde{r}} z^{s(k-t)} C_k(z), \qquad Q^{[t]}(z) = \sum_{h=t}^{\widehat{r}} z^{s(h-t)} D_h(z),$$

which can be proved by a direct computation.

ETNA
Kent State University and
Johann Radon Institute (RICAM)

Regarding the numerical stability of these evaluation schemes, we recall that the Paterson–Stockmeyer algorithm, Horner's method, and the method based on the explicit evaluation of matrix powers have similar stability as shown in [22, Thm. 4.5].

Classifying the solutions to matrix equations. Let f be analytic on a subset of $\mathbb{C}$, let $A \in \mathbb{C}^{N\times N}$, and let $X \in \mathbb{C}^{N\times N}$ be a solution to $f(X) = A$, where f is a primary matrix function in the sense of [22, Def. 1.2, Def. 1.4, Def. 1.11] and is defined on the spectrum of X [22, Def. 1.1]. The matrix X is a *primary solution* if there exists a polynomial χ such that $X = \chi(A)$, and it is *isolated* if there exists a neighborhood $\mathcal{U}$ of X where X is the unique solution to the equation $f(X) = A$.

An eigenvalue ξ of X is said to be critical if $f'(\xi) = 0$. Evard and Uhlig [13, Thm. 6.1] show that a solution is primary if and only if for any two distinct eigenvalues ξ_1 and ξ_2 of X, we have that $f(\xi_1) \neq f(\xi_2)$, and all critical eigenvalues of X are semisimple, that is, belong to Jordan blocks of size exactly 1. Moreover, Fasi and Iannazzo [16, Thm. 6] show that a solution is isolated if and only if for any two distinct eigenvalues ξ_1 and ξ_2 of X, we have that $f(\xi_1) \neq f(\xi_2)$, and all critical eigenvalues of X are simple, that is, have algebraic multiplicity one.

We will also consider equations of the type $p(X) = Aq(X)$ with p, q polynomials. In this case, a primary solution is one that can be written as a polynomial of A.

3. Substitution algorithms for matrix equations. We now present algorithms for computing primary solutions to the matrix equation (1.1). In fact, in the discussion below we consider the seemingly simpler equation[1]

$$p(Y) = Tq(Y), \tag{3.1}$$

where

$$T = \begin{bmatrix} T_{11} & \dots & T_{1\nu} \\ & \ddots & \vdots \\ & & T_{\nu\nu} \end{bmatrix} \in \mathbb{C}^{N\times N}, \qquad T_{11} \in \mathbb{C}^{\tau_1\times\tau_1}, \dots, T_{\nu\nu} \in \mathbb{C}^{\tau_\nu\times\tau_\nu}, \tag{3.2}$$

and $Y \in \mathbb{C}^{N\times N}$ has the same nonzero block structure as T (the empty blocks below the block diagonal should be understood as zero blocks). This is not a restriction if only primary solutions are sought, since when p and q are coprime, any matrix equation of the form (1.1) can be reduced to an equation of the form (3.1) as we now explain.

On the one hand, it has been shown [16, Prop. 9] that if p and q are coprime, then X satisfies (1.1) if and only if it satisfies

$$p(X) = Aq(X). \tag{3.3}$$

On the other hand, if $T = UAU^{-1}$, then X satisfies (3.3) if and only if $Y := UXU^{-1}$ satisfies $p(Y) = Tq(Y)$, and X is a primary solution if and only if Y is. Furthermore, if T is block upper triangular, then primary solutions have the same block upper triangular structure (being polynomials of T), and we can conclude that (3.1) is equivalent to (1.1).

A similarity transformation that exists for all $A \in \mathbb{C}^{N\times N}$ is the Schur decomposition $A =: UTU^*$, where $T, U \in \mathbb{C}^{N\times N}$ are upper triangular and unitary, respectively. If A has real entries, it is customary to consider the real Schur decomposition $A =: QSQ^T$, where $S, Q \in \mathbb{R}^{N\times N}$ are upper quasi-triangular and orthogonal, respectively.

In the following, we will use the fact that if χ is a polynomial and Y is block upper triangular, then $\chi(Y)_{ii} = \chi(Y_{ii})$, for $i = 1, \dots, \nu$.

[1] The choice of post-multiplying T by $q(Y)$ was made only to fix the notation. Since $T = r(Y)$ commutes with $q(Y)$, one can consider the equation $p(Y) = q(Y)T$ instead and perform the analysis with essentially the same results.

3.1. Idea of the algorithms. In order to derive our substitution algorithms, we rewrite equation (3.1) at the block level and then try to find an explicit expression for the block relations

$$\begin{cases} p(Y_{ii}) = T_{ii}q(Y_{ii}), & i = 1,\dots,\nu, \\ L_{ij}(Y_{ij}) = b_{ij}, & 1 \le i < j \le \nu, \end{cases} \tag{3.4}$$

where L_{ij} is a linear function depending uniquely on Y_{ii} and Y_{jj}, and b_{ij} is a nonlinear function of T_{ij} and the blocks of Y lying to the left and below the block Y_{ij}. The key point of these algorithms is a careful construction of L_{ij} and b_{ij}, as (3.4) readily translates into the two-step algorithm:

1. Choose a solution Y_{ii} to the equation $p(Y) = T_{ii}q(Y)$, for $i = 1,\dots,\nu$;
2. Compute L_{ij} and b_{ij} and solve for Y_{ij} the linear matrix equation $L_{ij}(Y) = b_{ij}$, for $1 \le i < j \le \nu$.

The first step is easy to perform when T is a triangular Schur factor: for 1×1 blocks it suffices to solve the corresponding scalar equation, whereas for 2×2 real blocks with complex conjugate eigenvalues, [16, Prop. 15] provides a direct formula for computing real solutions. This step is delicate since, in general, there are several solutions for each diagonal block: this captures the fact that (3.1) may have several solutions. When the Schur form is used, choosing a solution corresponds to selecting a branch of the inverse of $r(z)$ for each eigenvalue of T (and A). The choice of the diagonal block solutions (or the branch of the inverse of r) is assumed as an input value of our algorithms and should be dictated by the application under consideration.

The second equation in (3.4) justifies our use of the term "substitution": if one computes the block entries of Y one superdiagonal at a time from the main diagonal to the top right corner, then the equation for Y_{ij} is obtained by substituting into the expression for b_{ij} the blocks of Y that have already been computed. If L_{ij} is singular for some i and j, then the algorithm has a breakdown. If, on the contrary, all the operators are nonsingular, then all the off-diagonal blocks of Y are uniquely determined, and we say that the algorithm is applicable. As we shall see, the applicability of a substitution algorithm depends on what solution to the equation $p(Y) = T_{ii}q(Y)$ is chosen for $i = 1,\dots,\nu$.

As an example, the matrix equation $Y^2 = T$, which defines the matrix square root, produces the simplest substitution algorithm, namely

$$\begin{cases} Y_{ii}^2 = T_{ii}, & i = 1,\dots,\nu, \\ Y_{ii}Y_{ij} + Y_{ij}Y_{jj} = T_{ij} - \sum_{t=i+1}^{j-1} Y_{it}Y_{tj}, & 1 \le i < j \le \nu. \end{cases} \tag{3.5}$$

When all blocks are of size 1×1, these equations yield the algorithm of Björck and Hammarling [6], while in the real case, with blocks of size at most 2×2, we recover the algorithm of Higham [21]. In the former case, the equation in the first line of (3.5) has two solutions for $T_{ii} \neq 0$, and the algorithm has a breakdown if $Y_{ii} + Y_{jj} = 0$ for any $1 \le i < j \le N$. This happens when T has two zero diagonal entries or when T has two equal diagonal entries, say $T_{11} = T_{22}$, and the square roots are chosen so that $Y_{11} = -Y_{22}$. In all the other cases, the algorithm produces a unique solution.

Another instance of a substitution algorithm is the method developed by Smith [30] to compute the αth root, which produces

$$\begin{cases} Y_{ii}^{\alpha} = T_{ii}, & i = 1,\dots,\nu, \\ \sum_{u=0}^{\alpha-1} Y_{ii}^{\alpha-1-u} Y_{ij} Y_{jj}^{u} = T_{ij} - \sum_{u=1}^{\alpha-1} Y_{ii}^{\alpha-1-u} \sum_{t=i+1}^{j-1} Y_{it} Y_{tj}^{[u]}, & 1 \le i < j \le \nu, \end{cases}$$

where $Y_{tj}^{[u]}$ is the entry in position (t,j) of Y^u, for $u = 1,\dots,\alpha-1$. In this case, calculating the b_{ij} requires computing and storing $\alpha - 1$ additional matrices, which we call the *stages* of the method.

Since the asymptotic cost of the algorithm is given by the number of stages it requires, techniques that involve fewer stages have been proposed [17, 24]. These methods do not form all the powers of Y but only those needed to compute Y^α by means of the binary powering technique, which leads to more efficient algorithms.

Our previous algorithm [16] for the solution of (3.1) uses the stages of Horner's scheme for p and q and $Tq(Y)$. Here we propose two different techniques for solving the same problem more efficiently: one, described in Section 3.2, uses as stages the powers of Y, $p(Y)$, $q(Y)$, and $Tq(Y)$, whereas the other, described in Section 3.3, uses the stages of the Paterson–Stockmeyer scheme for the evaluation of p and q and $Tq(Y)$. The latter is the counterpart to the state-of-the-art algorithm for the evaluation of rational matrix functions [22, Chap. 4].

3.2. Algorithm based on explicit powers. Let $\mu = \max\{m,n\}$, and let Y be a solution to (3.1) that is block upper triangular and has the same block structure as T. We construct the sequence

$$\begin{aligned} Y^{[0]} &= I, \\ Y^{[1]} &= Y, \\ Y^{[2]} &= YY^{[1]}, \\ &\vdots \\ Y^{[\mu]} &= YY^{[\mu-1]}, \end{aligned} \tag{3.6}$$

where $Y^{[k]} = Y^k$, for $k = 0,\dots,\mu$. For $1 \le i < j \le \nu$, we can write the block in position (i,j) of (3.1) as $L_{ij}(Y_{ij}) = b_{ij}$, where the formulae for the linear operator L_{ij} and for the matrix b_{ij} involve only blocks $p(Y)_{\bar{\imath}\bar{\jmath}}$, $q(Y)_{\bar{\imath}\bar{\jmath}}$, and $Y_{\bar{\imath}\bar{\jmath}}^{[k]}$, with $k = 1,\dots,\mu$, such that $\bar{\jmath} - \bar{\imath} < j - i$. We consider these blocks and the blocks of T to be *known quantities*, having in mind an algorithm that computes the elements of Y from the main diagonal to the top right corner.

The approach of this technique is similar to that of [16, Alg. 1]. The key difference is the sequence of stages that is used to derive L_{ij} and b_{ij} in the equations in (3.4). In fact, while [16, Alg. 1] exploits the stages of Horner's rule applied to p and q, the new algorithm relies on the recursion (3.6). As we will see, this change allows us to halve the number of stages and thus the resulting computational cost of the algorithm.

The block (i,j) of $Y^{[k]}$, for $1 \le i < j \le \nu$, can be written, for $k = 2,\dots,\mu$, as

$$Y_{ij}^{[k]} = Y_{ii}Y_{ij}^{[k-1]} + Y_{ij}Y_{jj}^{[k-1]} + F_{ij}^{[k]}, \qquad F_{ij}^{[k]} := \sum_{t=i+1}^{j-1} Y_{it}Y_{tj}^{[k-1]}, \tag{3.7}$$

where Y_{ij} appears (implicitly for $k > 2$) in the first summand on the right-hand side and (explicitly) in the second. When $k > 2$, by substituting the formula for $Y_{ij}^{[k-1]}$ into that for $Y_{ij}^{[k]}$, one obtains a formula for $Y_{ij}^{[k]}$ involving only $Y_{ij}^{[k-2]}$, Y_{ij}, and known quantities. Repeating this procedure we deduce, for $k = 2,\dots,\mu$, the formula

$$Y_{ij}^{[k]} = \sum_{u=0}^{k-1} Y_{ii}^{[u]} Y_{ij} Y_{jj}^{[k-1-u]} + \sum_{u=0}^{k-2} Y_{ii}^{[u]} F_{ij}^{[k-u]}, \tag{3.8}$$

ETNA
Kent State University and
Johann Radon Institute (RICAM)

where $Y_{ij}^{[k]}$ is given in terms of Y_{ij} and known quantities. By introducing the operator

$$B_{ij}^{[k]}(V) := \sum_{u=0}^{k-1} Y_{ii}^{[u]} V Y_{jj}^{[k-1-u]}, \qquad k = 1, \ldots, \mu, \tag{3.9}$$

where V has the size of Y_{ij}, we can rewrite (3.8) in the more compact form

$$Y_{ij}^{[k]} = B_{ij}^{[k]}(Y_{ij}) + \sum_{u=0}^{k-2} Y_{ii}^{[u]} F_{ij}^{[k-u]}, \qquad k = 1, \ldots, \mu. \tag{3.10}$$

The block in position (i, j) of the equation $p(Y) - Tq(Y) = 0$ reads

$$\sum_{k=0}^{m} c_k Y_{ij}^{[k]} - \sum_{t=i}^{j} T_{it} \sum_{k=0}^{n} d_k Y_{tj}^{[k]} = 0,$$

and substituting for $Y_{ij}^{[k]}$ the expression in (3.10) gives, for $1 \leq i < j \leq \nu$, the equation $L_{ij}(Y_{ij}) = b_{ij}$, where

$$\begin{aligned} L_{ij}(Y_{ij}) &:= \sum_{k=1}^{m} c_k B_{ij}^{[k]}(Y_{ij}) - T_{ii} \sum_{k=1}^{n} d_k B_{ij}^{[k]}(Y_{ij}), \\ b_{ij} &:= \sum_{t=i+1}^{j} T_{it} q(Y)_{tj} - \sum_{k=2}^{m} p^{[k]}(Y_{ii}) F_{ij}^{[k]} + T_{ii} \sum_{k=2}^{n} q^{[k]}(Y_{ii}) F_{ij}^{[k]}, \end{aligned} \tag{3.11}$$

where $p^{[k]}(z)$ and $q^{[k]}(z)$ are the stages of Horner's scheme (see (2.1)) applied to p and q, respectively, and we have used the identity

$$\begin{aligned} \sum_{k=1}^{m} c_k \sum_{u=0}^{k-2} Y_{ii}^{[u]} F_{ij}^{[k-u]} &= \sum_{k=1}^{m} c_k \sum_{u=2}^{k} Y_{ii}^{[k-u]} F_{ij}^{[u]} = \sum_{u=2}^{m} \sum_{k=u}^{m} c_k Y_{ii}^{k-u} F_{ij}^{[u]} \\ &= \sum_{u=2}^{m} p^{[u]}(Y_{ii}) F_{ij}^{[u]} \end{aligned}$$

and the analogous relation for $\sum_{k=1}^{n} d_k \sum_{u=0}^{k-2} Y_{ii}^{[u]} F_{ij}^{[k-u]}$. Finally, by applying the vec operator that stacks the columns of a matrix into one vector on both sides, equation (3.11) can be rewritten as

$$M_{ij} \operatorname{vec}(Y_{ij}) = \varphi_{ij}, \tag{3.12}$$

where M_{ij} is the matrix form of the operator L_{ij}, and $\varphi_{ij} = \operatorname{vec}(b_{ij})$.

These equations readily translate into a substitution algorithm for solving the rational matrix equation (3.1): once the diagonal blocks of Y have been chosen, the remaining blocks can be computed one super-diagonal at a time by solving the linear system $M_{ij} Y = \varphi_{ij}$, with M_{ij} and φ_{ij} as in (3.12), in order to obtain the block $Y_{ij} = \operatorname{vec}^{-1}(M_{ij}^{-1} \varphi_{ij})$. The detailed pseudocode of this approach is given in Algorithm 3.1.

The algorithm has a breakdown if any of the matrices M_{ij} is singular, in which case the linear system $M_{ij} Y = \varphi_{ij}$ does not have a unique solution. In the next lemma, we show that

ETNA
Kent State University and
Johann Radon Institute (RICAM)

Algorithm 3.1: Algorithm for $r(Y) = T$, based on explicit powers.

Input : T as in (3.2), $c_0, \dots, c_m$ coefficients of p, $d_0, \dots, d_n$ coefficients of q.
Output : $Y^{[1]} \in \mathbb{C}^{N \times N}$ such that $p(Y^{[1]})q^{-1}(Y^{[1]}) \approx T$.

```
1  μ ← max{m, n}.
2  for i = 1 to ν do
3  |   Y_ii^[1] ← a solution to p(Y) − T_ii q(Y) = 0
4  |   for k = 2 to μ do
5  |   |   Y_ii^[k] ← Y_ii^[1] Y_ii^[k−1]
6  |   P_ii^[m] ← c_m I_τi
7  |   for k = m − 1 downto 1 do
8  |   |   P_ii^[k] ← c_k I_τi + Y_ii P_ii^[k+1]
9  |   Q_ii^[n] ← d_n I_τi
10 |   for k = n − 1 downto 0 do
11 |   |   Q_ii^[k] ← d_k I_τi + Y_ii Q_ii^[k+1]
12 for v = 1 to ν − 1 do
13 |   for i = 1 to ν − v do
14 |   |   j ← i + v
15 |   |   for k = 2 to μ do
16 |   |   |   F_ij^[k] ← Σ_{t=i+1}^{j−1} Y_it^[1] Y_tj^[k−1]
17 |   |   M_ij ← Σ_{k=1}^{m} (P_jj^[k])^T ⊗ Y_ii^[k−1] − (I_τj ⊗ T_ii) Σ_{k=1}^{n} (Q_jj^[k])^T ⊗ Y_ii^[k−1]
18 |   |   b_ij ← vec( Σ_{t=i+1}^{j} T_it Q_tj^[0] − Σ_{k=2}^{m} P_ii^[k] F_ij^[k] + T_ii Σ_{k=2}^{n} Q_ii^[k] F_ij^[k] )
19 |   |   Y_ij^[1] ← vec^{−1}(M_ij^{−1} b_ij)
20 |   |   for k = 2 to μ do
21 |   |   |   Y_ij^[k] ← Y_ii^[1] Y_ij^[k−1] + Y_ij^[1] Y_jj^[k−1] + F_ij^[k]
22 |   |   Q_ij^[0] ← Σ_{k=1}^{n} d_k Y_ij^[k]
```

these matrices are the same as those appearing in [16, Eq. (21)], which allows us to relate the applicability of Algorithm 3.1 to that of the algorithms in [16].

LEMMA 3.1. *With the notation of Section 3.2, let M_{ij}, for $1 \le i < j \le \nu$, be the matrix associated with L_{ij} appearing in (3.11). We have*

$$M_{ij} = \sum_{u=1}^{m} \left(p^{[u]}(Y_{jj})\right)^T \otimes Y_{ii}^{[u-1]} - (I \otimes T_{ii}) \sum_{u=1}^{n} \left(q^{[u]}(Y_{jj})\right)^T \otimes Y_{ii}^{[u-1]}, \tag{3.13}$$

where $p^{[u]}(z)$, for $u = 1, \dots, m$, and $q^{[u]}(z)$, for $u = 1, \dots, n$, are the stages of Horner's scheme for the evaluation of $p(z)$ and $q(z)$, respectively.

Proof. If we denote the matrix form of the operator $B_{ij}^{[k]}$ in (3.9) by

$$\widehat{B}_{ij}^{[k]} = \sum_{u=0}^{k-1} (Y_{jj}^{[k-1-u]})^T \otimes Y_{ii}^{[u]}, \tag{3.14}$$

ETNA
Kent State University and
Johann Radon Institute (RICAM)

then we can rewrite $L_{ij}(Y_{ij})$ in matrix form as $M_{ij}\mathrm{vec}(Y_{ij})$, where

$$M_{ij} := \sum_{k=1}^{m} c_k \widehat{B}_{ij}^{[k]} - T_{ii} \sum_{k=1}^{n} d_k \widehat{B}_{ij}^{[k]}. \tag{3.15}$$

On the other hand, we have that

$$\begin{aligned} \sum_{k=1}^{m} c_k \widehat{B}_{ij}^{[k]} &= \sum_{k=1}^{m} c_k \sum_{u=0}^{k-1} (Y_{jj}^{[k-1-u]})^T \otimes Y_{ii}^{[u]} \\ &= \sum_{u=0}^{m-1} \left(\sum_{k=u+1}^{m} c_k Y_{jj}^{[k-1-u]} \right)^T \otimes Y_{ii}^{[u]} \\ &= \sum_{u=1}^{m} \big(p^{[u]}(Y_{jj})\big)^T \otimes Y_{ii}^{[u-1]}, \end{aligned} \tag{3.16}$$

and similarly that

$$\sum_{k=1}^{n} d_k \widehat{B}_{ij}^{[k]} = \sum_{u=1}^{n} \big(q^{[u]}(Y_{jj})\big)^T \otimes Y_{ii}^{[u-1]}. \tag{3.17}$$

Plugging (3.16) and (3.17) into (3.15) concludes the proof. □

Applicability. If T is the triangular factor of a complex Schur decomposition, then all diagonal blocks have size 1, and the diagonal elements of Y can be computed by solving a scalar equation. If T is in real Schur form, then the diagonal blocks have size at most 2, and the 2×2 diagonal blocks of Y can be determined by using, for example, a direct formula as in [16, Prop. 15].

For the complex Schur form, M_{ij} is the same as ψ_{ij} in [16, Eq. (12)], and the applicability of the algorithm depends on what solutions to the scalar equation $p(Y) = T_{ii}q(Y)$ are chosen for $i = 1, \ldots, \nu$.

THEOREM 3.2. *Let $r = p/q$ be a rational function with $p \in \mathbb{C}_m[z]$ and $q \in \mathbb{C}_n[z]$ coprime. Let $T \in \mathbb{C}^{N\times N}$ be upper triangular, and let $\xi_1, \ldots, \xi_N \in \mathbb{C}$ be such that $r(\xi_i) = t_{ii}$ for $i = 1, \ldots, N$. Then the two conditions are equivalent:*

(*a*) *Algorithm 3.1 with the choice $Y_{ii} = \xi_i$ is applicable to the equation $r(Y) = T$, that is, equation (3.12) has a unique solution Y_{ij} for $1 \le i < j \le N$;*

(*b*) *$r[\xi_i, \xi_j] \neq 0$, for $1 \le i < j \le N$.*

If either condition is satisfied, then the solution Y is primary and isolated.

Proof. By [16, Lemma 11] we can express the divided differences of p for $a, b \in \mathbb{C}$ in terms of the stages of Horner's method as $p[a, b] = \sum_{j=1}^{m} a^{j-1}p^{[j]}(b)$. This can be used in the scalar version of (3.13) to show that

$$M_{ij} = p[Y_{ii}, Y_{jj}] - T_{ii}q[Y_{ii}, Y_{jj}] = r[\xi_i, \xi_j]q(\xi_j).$$

Since $q(\xi_j) \neq 0$ by hypothesis, we have that $M_{ij} \neq 0$ if and only if $r[\xi_i, \xi_j] \neq 0$.

When either of these conditions is fulfilled, we are in the hypotheses of [16, Thm. 6], and the solution is primary and isolated. □

A consequence of Lemma 3.1 is the following result on the existence of isolated solutions to (1.1).

THEOREM 3.3. *Let $r = p/q$ be a rational function with $p \in \mathbb{C}_m[z]$ and $q \in \mathbb{C}_n[z]$ coprime, and let $A \in \mathbb{C}^{N\times N}$. There exists a unique solution $X \in \mathbb{C}^{N\times N}$ to $r(X) = A$ with*

ETNA
Kent State University and
Johann Radon Institute (RICAM)

eigenvalues $\xi_1,\dots,\xi_N$ if and only if $r(\xi_1),\dots,r(\xi_N)$ are the eigenvalues of A (counted with multiplicities) and $r[\xi_i,\xi_j]\neq 0$ for $1\le i<j\le N$. Moreover, X is primary and isolated.

Proof. Let X be a solution with eigenvalues $\xi_1,\dots,\xi_N$, then the eigenvalues of $r(X)$ are $r(\xi_1),\dots,r(\xi_N)$ [22]. From [16, Thm. 6], we know that if there exists a unique solution with a given set of eigenvalues, then $r[\xi_i,\xi_j]\neq 0$ for $i\neq j$ and X is primary and isolated.

Let $U^*AU=T$ be the triangular factor of the complex Schur form of A ordered so that $t_{ii}=r(\xi_i)$, for $i=1,\dots,N$. Since $r[\xi_i,\xi_j]\neq 0$, we have that Algorithm 3.1 is applicable to $r(Y)=T$ and gives a solution Y, which in turn provides $X=UYU^*$ as solution to $r(X)=A$. By [16, Thm. 6], X is primary, isolated, and is the unique solution with eigenvalues $\xi_1,\dots,\xi_N$. □

We stress that the result in Theorem 3.3 is stronger than that in [16, Cor. 14] as the latter requires, as a hypothesis, the existence of a solution with the given eigenvalues.

The case of blocks of arbitrary size can be addressed analogously with the difference that if a non-isolated solution is chosen for any of the block equations $p(Y)-T_{ii}q(Y)=0$, then the solution produced by the algorithm, when applicable, may be non-isolated or even non-primary.

THEOREM 3.4. *Let $r=p/q$ be a rational function with $p\in\mathbb{C}_m[z]$ and $q\in\mathbb{C}_n[z]$ coprime, and let $T=(T_{ij})\in\mathbb{C}^{N\times N}$ be block upper triangular with ν diagonal blocks of size $\tau_1,\dots,\tau_\nu$. Let $\Xi_i\in\mathbb{C}^{\tau_i\times\tau_i}$, for $i=1,\dots,\nu$, be a solution to $r(\Xi)=T_{ii}$ with eigenvalues $\xi_{i1},\dots,\xi_{i\tau_i}$. Then the two conditions are equivalent:*

(a) *Algorithm 3.1 with the choice $Y_{ii}=\Xi_i$ is applicable to the equation $r(Y)=T$, that is, equation (3.12) has a unique solution Y_{ij} for $1\le i<j\le\nu$;*

(b) *$r[\xi_{ik_i},\xi_{jk_j}]\neq 0$, for ξ_{ik_i} an eigenvalue of Ξ_i and ξ_{jk_j} an eigenvalue of Ξ_j, for $1\le i<j\le\nu$, $1\le k_i\le\tau_i$, $1\le k_j\le\tau_j$.*

If Ξ_i is an isolated solution to the equation $r(X)=T_{ii}$ for $i=1,\dots,\nu$, and either of the conditions above is satisfied, then Algorithm 3.1 computes an isolated solution.

Proof. Let $\xi_{i1},\dots,\xi_{i\tau_i}$ and $\xi_{j1},\dots,\xi_{j\tau_j}$ be the eigenvalues of Y_{ii} and Y_{jj}, respectively. If U_i and U_j are unitary matrices such that $U_i^*Y_{ii}U_i$ and $U_j^*Y_{jj}^TU_j$ are upper triangular, then $\widetilde{T}_{ii}:=U_i^*T_{ii}U_i=r(U_i^*Y_{ii}U_i)$ is upper triangular as well, and so is $(U_j^*\otimes U_i^*)M_{ij}(U_j\otimes U_i)$ whose diagonal entries (eigenvalues) are

$$\sum_{k=1}^{m}p^{[k]}(\xi_{jk_j})\xi_{ik_i}^{k-1}-r(\xi_{ik_i})\sum_{k=1}^{n}q^{[k]}(\xi_{jk_j})\xi_{ik_i}^{k-1}=r[\xi_{ik_i},\xi_{jk_j}]q(\xi_{jk_j}).$$

By hypothesis we have that $q(\xi_{jk_j})\neq 0$, thus M_{ij} is nonsingular if and only if $r[\xi_{ik_i},\xi_{jk_j}]\neq 0$ for any k_i and k_j. If Ξ_i is isolated, then $r[\xi_{ia},\xi_{ib}]\neq 0$ for $1\le a\le\tau_i$ and $1\le b\le\tau_i$ with $a\neq b$. This condition together with Theorem 3.4(b) implies that $r[\zeta_i,\zeta_j]\neq 0$ for $1\le i<j\le N$, where $\zeta_1,\dots,\zeta_N$ are the eigenvalues of the computed solution Y, which is isolated by [16, Thm. 6]. □

The case in which A is real and the real Schur form of A is used can be seen as a particular case of Theorem 3.4, where the diagonal blocks are of size 1×1 or 2×2.

Computational cost. We now discuss the cost of Algorithm 3.1 for the triangular case $\nu=N$ and $\tau_i=1$, for $i=1,\dots,N$. When T presents nontrivial diagonal blocks, the results are similar with operations counted at the block instead of the scalar level.

Asymptotically, the most expensive quantities to compute are the $\mu-1$ sums on line 16, which are related to the stages of the recursion (3.6), and the first sum on line 18, which corresponds to the final inversion $q(Y)^{-1}p(Y)$. Each of these stages entails, for $1\le i<j\le N$, a

ETNA
Kent State University and
Johann Radon Institute (RICAM)

sum of the type

$$\sigma_{ij} := \sum_{t=i+1}^{j-\varepsilon} a_{it} b_{tj}, \tag{3.18}$$

where ε is either zero or one and a_{it} and b_{tj} are scalars for $t = i+1, \dots, j-\epsilon$. Evaluating σ_{ij} requires $(j-i-\varepsilon)$ multiplications and $(j-i-\varepsilon-1)$ sums, thus $\frac{1}{3}N^3 + o(N^3)$ operations are needed to compute all the σ_{ij}, for $1 \leq i < j \leq N$. As these are the only expressions whose cost is cubic in N, Algorithm 3.1 requires $\frac{\mu}{3}N^3 + o(N^3)$ operations.

Note that evaluating a rational function of order $[m/n]$ at a triangular matrix argument of size N via explicit powering has cost $\frac{\mu}{3}N^3 + o(N^3)$, which is exactly the same as that of our substitution algorithm.

Since computing the Schur decomposition and recovering the result require $25N^3$ and $3N^3$ flops, respectively, the asymptotic cost of solving the general equation (1.1) using Algorithm 3.1 is $\left(28 + \frac{\mu}{3}\right)N^3 + o(N^3)$ flops.

3.3. Algorithm based on the Paterson–Stockmeyer method. Let Y be a block upper triangular solution to (3.1) that has the same block structure as T. Using the Paterson–Stockmeyer evaluation scheme, we can construct, for $1 \leq i < j \leq \nu$, the matrix equation $L_{ij}(Y_{ij}) = b_{ij}$, where L_{ij} is linear with respect to the block Y_{ij} and both L_{ij} and b_{ij} can be computed by using blocks of Y and T such that the difference between the column and row index is greater than $j-i$, which again we treat as *known quantities*. We will use these equations to deduce an algorithm to solve (3.1) that is cheaper than Algorithm 3.1.

The first step is to construct a recursion with one stage for each matrix multiplication in the Paterson–Stockmeyer scheme. We start from the block (i,j) of the equation $p(Y) - Tq(Y) = 0$, which reads

$$p(Y)_{ij} - T_{ii}q(Y)_{ij} - \sum_{t=i+1}^{j} T_{it}q(Y)_{tj} = 0.$$

Since only the first two summands on the left-hand side depend on Y_{ij} while the third can be treated as a known quantity, we need to deduce expressions for $p(Y)_{ij}$ and $q(Y)_{ij}$ that involve only Y_{ij} and known quantities. We will give a detailed derivation of the expression for $p(Y)_{ij}$, which is based on the sequence $P^{[k]}(Y)$ defined in (2.3) and the sequence $Y^{[k]}$ defined in (3.6). The corresponding expression for $q(Y)_{ij}$ can be deduced in a similar manner. It can be shown that[2]

$$p(Y)_{ij} = \sum_{k=0}^{\widetilde{r}} Y_{ii}^{ks} C_k(Y)_{ij} + \sum_{k=0}^{\widetilde{r}-1} Y_{ii}^{ks} Y_{ij}^{[s]} P^{[k+1]}(Y_{jj}) + \sum_{k=0}^{\widetilde{r}-1} Y_{ii}^{ks} \widetilde{\Psi}_{ij}^{[k]}, \tag{3.19}$$

with

$$\widetilde{\Psi}_{ij}^{[k]} := \sum_{t=i+1}^{j-1} Y_{it}^{[s]} P^{[k+1]}(Y)_{tj}, \qquad k = 0, \dots, \widetilde{r}-1.$$

[2] In fact, by an induction argument one can prove that the formula

$$p(Y)_{ij} = Y_{ii}^{ds} P^{[d]}(Y)_{ij} + \sum_{k=0}^{d-1} Y_{ii}^{ks} \left(C_k(Y)_{ij} + Y_{ij}^{[s]} P^{[k+1]}(Y_{jj}) + \widetilde{\Psi}_{ij}^{[k]}\right),$$

which coincides with (3.19) for $d = \widetilde{r}$, holds for $d = 0, \dots, \widetilde{r}-1$ as well.

ETNA
Kent State University and
Johann Radon Institute (RICAM)

In order to get an equation in terms of Y_{ij} and known quantities only, we note the third summand on the right-hand side of (3.19) contains only known quantities, while $Y_{ij}^{[s]}$ and $C_k(Y)_{ij}$ can be further reduced as we now explain. From (3.10) we obtain

(3.20)
$$\begin{aligned}\sum_{k=0}^{\widetilde{r}} Y_{ii}^{ks} C_k(Y)_{ij} &= \sum_{k=0}^{\widetilde{r}} Y_{ii}^{ks} \sum_{\ell=1}^{s-1}{}' c_{\ell+sk} B_{ij}^{[\ell]}(Y_{ij}) + \sum_{k=0}^{\widetilde{r}} Y_{ii}^{ks} \sum_{\ell=2}^{s-1}{}' c_{\ell+sk} \Phi_{ij}^{[\ell]},\\ \sum_{k=0}^{\widetilde{r}-1} Y_{ii}^{ks} Y_{ij}^{[s]} P^{[k+1]}(Y_{jj}) &= \sum_{k=0}^{\widetilde{r}-1} Y_{ii}^{ks} B_{ij}^{[s]}(Y_{ij}) P^{[k+1]}(Y_{jj}) + \sum_{k=0}^{\widetilde{r}-1} Y_{ii}^{ks} \Phi_{ij}^{[s]} P^{[k+1]}(Y_{jj}),\end{aligned}$$

with

$$\Phi_{ij}^{[k]} := \sum_{u=0}^{k-2} Y_{ii}^{u} F_{ij}^{[k-u]}, \qquad k = 1, \dots, s, \tag{3.21}$$

where $F_{ij}^{[k]}$ is defined in (3.7). Therefore, the last sum on the right-hand side of both equations in (3.20) contains only known quantities.

A similar reduction holds for $q(Y)_{ij}$, and we get the equation $L'_{ij}(Y_{ij}) = b'_{ij}$, where

$$\begin{aligned} b'_{ij} := &\sum_{t=i+1}^{j} T_{it} q(Y)_{tj} - \sum_{k=0}^{\widetilde{r}} Y_{ii}^{ks} \sum_{\ell=2}^{s-1}{}' c_{\ell+sk} \Phi_{ij}^{[\ell]} - \sum_{k=0}^{\widetilde{r}-1} Y_{ii}^{ks} \left(\Phi_{ij}^{[s]} P^{[k+1]}(Y_{jj}) + \widetilde{\Psi}_{ij}^{[k]} \right)\\ &+ T_{ii} \sum_{k=0}^{\widehat{r}} Y_{ii}^{ks} \sum_{\ell=2}^{s-1}{}' d_{\ell+sk} \Phi_{ij}^{[\ell]} + T_{ii} \sum_{k=0}^{\widehat{r}-1} Y_{ii}^{ks} \left(\Phi_{ij}^{[s]} Q^{[k+1]}(Y_{jj}) + \widehat{\Psi}_{ij}^{[k]} \right),\end{aligned}$$

with

$$\widehat{\Psi}_{ij}^{[k]} = \sum_{t=i+1}^{j-1} Y_{it}^{[s]} Q^{[k+1]}(Y)_{tj}, \qquad k = 0, \dots, \widehat{r} - 1,$$

and the matrix representing L'_{ij} in the vec-basis is

$$M'_{ij} = \widetilde{P}_{ij} - (I_{\tau_j} \otimes T_{ii}) \widehat{P}_{ij}, \tag{3.22}$$

where

$$\begin{aligned}\widetilde{P}_{ij} &:= \sum_{k=0}^{\widetilde{r}} (I_{\tau_j} \otimes Y_{ii}^{ks}) \sum_{\ell=1}^{s-1}{}' c_{\ell+sk} \widehat{B}_{ij}^{[\ell]} + \sum_{k=0}^{\widetilde{r}-1} \big((P^{[k+1]}(Y)_{jj})^T \otimes Y_{ii}^{ks} \big) \widehat{B}_{ij}^{[s]},\\ \widehat{P}_{ij} &:= \sum_{k=0}^{\widehat{r}} (I_{\tau_j} \otimes Y_{ii}^{ks}) \sum_{\ell=1}^{s-1}{}' d_{\ell+sk} \widehat{B}_{ij}^{[\ell]} + \sum_{k=0}^{\widehat{r}-1} \big((Q^{[k+1]}(Y)_{jj})^T \otimes Y_{ii}^{ks} \big) \widehat{B}_{ij}^{[s]}.\end{aligned} \tag{3.23}$$

As before, these relations lead to an algorithm for computing Y: we can first compute the diagonal blocks of Y (either recursively or by a direct method) and then obtain the others, one super-diagonal at a time, by exploiting the relation $Y_{ij} = \mathrm{vec}^{-1}(M_{ij}^{-1}\,\mathrm{vec}(b_{ij}))$.

The pseudocode of the algorithm based on the Paterson–Stockmeyer approach is given in Algorithm 3.2. In order to reduce the overall computational cost of that method, note that $\widehat{B}_{ij}^{[k]}$

Algorithm 3.2: Solve $r(Y) = T$ inverting the Paterson–Stockmeyer scheme.

Input : T as in (3.2), $c_0, \dots, c_m$ coefficients of p, $d_0, \dots, d_n$ coefficients of q, $s \in \mathbb{N}$.
Output : $Y^{[1]} \in \mathbb{C}^{N \times N}$ such that $p(Y^{[1]})q^{-1}(Y^{[1]}) = T$.

1. $\widetilde{r} \leftarrow \lfloor m/s \rfloor$
2. $\widehat{r} \leftarrow \lfloor n/s \rfloor$
3. **for** $i = 1$ **to** ν **do**
4. $\quad Y_{ii}^{[0]} \leftarrow I_{\tau_i}$
5. $\quad Y_{ii}^{[1]} \leftarrow$ a solution to $p(Y) - T_{ii}q(Y) = 0$
6. $\quad$ **for** $k = 2$ **to** s **do** $Y_{ii}^{[k]} \leftarrow Y_{ii}^{[1]} Y_{ii}^{[k-1]}$
7. $\quad P_{ii}^{[\widetilde{r}]} \leftarrow \sum_{u=0}^{m-\widetilde{r}s} c_{s\widetilde{r}+u} Y_{ii}^{[u]}$
8. $\quad$ **for** $k = \widetilde{r} - 1$ **downto** 1 **do** $P_{ii}^{[k]} \leftarrow Y_{ii}^{[s]} P_{ii}^{[k+1]} + \sum_{u=0}^{s-1} c_{sk+u} Y_{ii}^{[u]}$
9. $\quad Q_{ii}^{[\widehat{r}]} \leftarrow \sum_{u=0}^{n-\widehat{r}s} d_{s\widehat{r}+u} Y_{ii}^{[u]}$
10. $\quad$ **for** $k = \widehat{r} - 1$ **downto** 0 **do** $Q_{ii}^{[k]} \leftarrow Y_{ii}^{[s]} Q_{ii}^{[k+1]} + \sum_{u=0}^{s-1} d_{sk+u} Y_{ii}^{[u]}$
11. **for** $v = 1$ **to** $\nu - 1$ **do**
12. $\quad$ **for** $i = 1$ **to** $\nu - v$ **do**
13. $\quad\quad j \leftarrow i + v$
14. $\quad\quad \widehat{B}_{ij}^{[1]} \leftarrow I_{\tau_i \tau_j}$
15. $\quad\quad$ **for** $k = 2$ **to** s **do**
16. $\quad\quad\quad F_{ij}^{[k]} \leftarrow \sum_{t=i+1}^{j-1} Y_{it}^{[1]} Y_{tj}^{[k-1]}$
17. $\quad\quad\quad \widehat{B}_{ij}^{[k]} \leftarrow \left((Y_{jj}^{[1]})^T \otimes I_{\tau_i} \right) \widehat{B}_{ij}^{[k-1]} + I_{\tau_j} \otimes Y_{ii}^{[k-1]}$
18. $\quad\quad \Phi_{ij}^{[1]} \leftarrow 0$
19. $\quad\quad$ **for** $k = 2$ **to** s **do**
20. $\quad\quad\quad \Phi_{ij}^{[k]} \leftarrow F_{ij}^{[k]} + Y_{ii} \Phi_{ij}^{[k-1]}$
21. $\quad\quad$ **for** $k = 0$ **to** $\widetilde{r} - 1$ **do**
22. $\quad\quad\quad \widetilde{\Psi}_{ij}^{[k]} = \sum_{t=i+1}^{j-1} Y_{it}^{[s]} P_{tj}^{[k+1]}$
23. $\quad\quad$ **for** $k = 0$ **to** $\widehat{r} - 1$ **do**
24. $\quad\quad\quad \widehat{\Psi}_{ij}^{[k]} = \sum_{t=i+1}^{j-1} Y_{it}^{[s]} Q_{tj}^{[k+1]}$
25. $\quad\quad \widetilde{K} \leftarrow \sum_{k=0}^{\widetilde{r}-1} \left((P_{jj}^{[k+1]})^T \otimes (Y_{ii}^{[s]})^k \right)$
26. $\quad\quad \widehat{K} \leftarrow \sum_{k=0}^{\widehat{r}-1} \left((Q_{jj}^{[k+1]})^T \otimes (Y_{ii}^{[s]})^k \right)$
27. $\quad\quad \varphi_p \leftarrow \sum_{k=0}^{\widetilde{r}} (Y_{ii}^{[s]})^k \sum_{u=2}^{\prime s-1} c_{sk+u} \Phi_{ij}^{[u]} + \operatorname{vec}^{-1}(\widetilde{K} \operatorname{vec}(\Phi_{ij}^{[s]})) + \sum_{k=0}^{\widetilde{r}-1} (Y_{ii}^{[s]})^k \widetilde{\Psi}_{ij}^{[k]}$
28. $\quad\quad \varphi_q \leftarrow \sum_{k=0}^{\widehat{r}} (Y_{ii}^{[s]})^k \sum_{u=2}^{\prime s-1} d_{sk+u} \Phi_{ij}^{[u]} + \operatorname{vec}^{-1}(\widehat{K} \operatorname{vec}(\Phi_{ij}^{[s]})) + \sum_{k=0}^{\widehat{r}-1} (Y_{ii}^{[s]})^k \widehat{\Psi}_{ij}^{[k]}$
29. $\quad\quad \varphi_{ij} \leftarrow \operatorname{vec} \left(\sum_{t=i+1}^{j} T_{it} Q_{tj}^{[0]} - \varphi_p + T_{ii} \varphi_q \right)$
30. $\quad\quad M_p \leftarrow \sum_{k=0}^{\widetilde{r}} \left(I_{\tau_j} \otimes (Y_{ii}^{[s]})^k \right) \sum_{u=1}^{\prime s-1} c_{sk+u} \widehat{B}_{ij}^{[u]} + \widetilde{K} \widehat{B}_{ij}^{[s]}$
31. $\quad\quad M_q \leftarrow \sum_{k=0}^{\widehat{r}} \left(I_{\tau_j} \otimes (Y_{ii}^{[s]})^k \right) \sum_{u=1}^{\prime s-1} d_{sk+u} \widehat{B}_{ij}^{[u]} + \widehat{K} \widehat{B}_{ij}^{[s]}$
32. $\quad\quad Y_{ij}^{[1]} \leftarrow \operatorname{vec}^{-1} \left(\left(M_p - (I_{\tau_j} \otimes T_{ii}) M_q \right)^{-1} \varphi_{ij} \right)$
33. $\quad\quad$ **for** $k = 2$ **to** s **do**
34. $\quad\quad\quad Y_{ij}^{[k]} \leftarrow Y_{ii}^{[1]} Y_{ij}^{[k-1]} + Y_{ij}^{[1]} Y_{jj}^{[k-1]} + F_{ij}^{[k]}$
35. $\quad\quad P_{ij}^{[\widetilde{r}]} \leftarrow \sum_{u=1}^{m-\widetilde{r}s} c_{s\widetilde{r}+u} Y_{ij}^{[u]}$
36. $\quad\quad$ **for** $k = \widetilde{r} - 1$ **downto** 1 **do**
37. $\quad\quad\quad P_{ij}^{[k]} \leftarrow \widetilde{\Psi}_{ij}^{[k]} + Y_{ij}^{[s]} P_{jj}^{[k+1]} + Y_{ii}^{[s]} P_{ij}^{[k+1]} + \sum_{u=1}^{s-1} c_{sk+u} Y_{ij}^{[u]}$
38. $\quad\quad Q_{ij}^{[\widehat{r}]} \leftarrow \sum_{u=1}^{n-\widehat{r}s} d_{s\widehat{r}+u} Y_{ij}^{[u]}$
39. $\quad\quad$ **for** $k = \widehat{r} - 1$ **downto** 0 **do**
40. $\quad\quad\quad Q_{ij}^{[k]} \leftarrow \widehat{\Psi}_{ij}^{[k]} + Y_{ij}^{[s]} Q_{jj}^{[k+1]} + Y_{ii}^{[s]} Q_{ij}^{[k+1]} + \sum_{u=1}^{s-1} d_{sk+u} Y_{ij}^{[u]}$

in (3.14) and $\Phi_{ij}^{[k]}$ in (3.21) can be computed recursively by exploiting the identities

$$\Phi_{ij}^{[k]} = \begin{cases} F_{ij}^{[2]}, & k=2, \\ F_{ij}^{[k]} + Y_{ii}\Phi_{ij}^{[k-1]}, & k=3,\dots,s, \end{cases}$$

$$\widehat{B}_{ij}^{[k]} = \begin{cases} I_{\tau_i\tau_j}, & k=1, \\ \left((Y_{jj}^{[1]})^T \otimes I_{\tau_i}\right)\widehat{B}_{ij}^{[k-1]} + I_{\tau_j} \otimes Y_{ii}^{[k-1]}, & k=2,\dots,s. \end{cases}$$

The computational cost can be further reduced by using the identities

$$\operatorname{vec}\Big(\sum_{k=0}^{\widetilde{r}-1} Y_{ii}^{ks}\Phi_{ij}^{[s]}P^{[k+1]}(Y_{jj})\Big) = \sum_{k=0}^{\widetilde{r}-1}\big((P^{[k+1]}(Y)_{jj})^T \otimes Y_{ii}^{ks}\big)\operatorname{vec}\big(\Phi_{ij}^{[s]}\big),$$

$$\operatorname{vec}\Big(\sum_{k=0}^{\widehat{r}-1} Y_{ii}^{ks}\Phi_{ij}^{[s]}Q^{[k+1]}(Y_{jj})\Big) = \sum_{k=0}^{\widehat{r}-1}\big((Q^{[k+1]}(Y)_{jj})^T \otimes Y_{ii}^{ks}\big)\operatorname{vec}\big(\Phi_{ij}^{[s]}\big).$$

Indeed, the sums on the left-hand side appear in the expression for b'_{ij} while those on the right-hand side appear in (3.22). Thus, it is more convenient to compute them only once at the beginning of the iteration and reuse them when needed later on.

Applicability. We show that M'_{ij} in (3.22) is the same as M_{ij} in (3.13), which implies that Algorithm 3.2 is applicable if and only if Algorithm 3.1 is.

LEMMA 3.5. *For the matrix M'_{ij} in (3.22), with $1 \le i < j \le \nu$, we have that*

$$M'_{ij} = \sum_{k=1}^{m} c_k \widehat{B}_{ij}^{[k]} - (I_{\tau_j} \otimes T_{ii}) \sum_{k=1}^{n} d_k \widehat{B}_{ij}^{[k]},$$

that is, M'_{ij} is the same as M_{ij} in (3.13).

Proof. We prove that $M'_{ij} = M_{ij}$ by showing for the matrices in (3.23) that

$$\widetilde{P}_{ij} = \sum_{k=0}^{m} c_k \widehat{B}_{ij}^{[k]} \qquad \text{and} \qquad \widehat{P}_{ij} = \sum_{k=0}^{m} d_k \widehat{B}_{ij}^{[k]}.$$

With $\widehat{B}_{ij}^{[0]} = 0$, the first summand of $\widetilde{P}_{ij}$ can be written as

$$\sum_{k=0}^{\widetilde{r}} {\sum_{\ell=0}^{s-1}}' c_{\ell+sk}(I_{\tau_j} \otimes Y_{ii}^{ks})\widehat{B}_{ij}^{[\ell]},$$

while for the second we have

$$\begin{aligned}
&\sum_{t=0}^{\widetilde{r}-1} (P^{[t+1]}(Y_{jj}))^T \otimes Y_{ii}^{ts})\widehat{B}_{ij}^{[s]} \\
&\quad= \sum_{t=0}^{\widetilde{r}-1}\Bigg(\Big(\sum_{k=t+1}^{\widetilde{r}} {\sum_{\ell=0}^{s-1}}' c_{\ell+sk} Y_{jj}^{\ell+s(k-t-1)}\Big)^T \otimes Y_{ii}^{ts}\Bigg) \sum_{v=0}^{s-1} (Y_{jj}^{s-1-v})^T \otimes Y_{jj}^{v} \\
&\quad= \sum_{k=1}^{\widetilde{r}} {\sum_{\ell=0}^{s-1}}' c_{\ell+sk} \sum_{t=0}^{k-1}\sum_{v=0}^{s-1}\big((Y_{jj}^{\ell+sk-1-st-v})^T \otimes Y_{ii}^{st+v}\big) \\
&\quad= \sum_{k=1}^{\widetilde{r}} {\sum_{\ell=0}^{s-1}}' c_{\ell+sk} \sum_{u=0}^{sk-1}\big((Y_{jj}^{\ell+sk-1-u})^T \otimes Y_{ii}^{u}\big) = \sum_{k=1}^{\widetilde{r}} {\sum_{\ell=0}^{s-1}}' c_{\ell+sk}\big((Y_{jj}^{\ell})^T \otimes I_{\tau_i}\big)\widehat{B}_{ij}^{[sk]}.
\end{aligned}$$

ETNA
Kent State University and
Johann Radon Institute (RICAM)

The fact that $\widetilde{P}_{ij} = \sum_{k=0}^{m} c_k \widehat{B}_{ij}^{[k]}$ follows from the identity

$$(I_{\tau_j} \otimes Y_{ii}^{ks})\widehat{B}_{ij}^{[\ell]} + ((Y_{jj}^{\ell})^T \otimes I_{\tau_i})\widehat{B}_{ij}^{[sk]} = \widehat{B}_{ij}^{[\ell+sk]}, \tag{3.24}$$

which holds for $\ell = 0, \dots, s-1$ when $k < \widetilde{r}$ and for $\ell = 0, \dots, m - \widetilde{r}s$ when $k = \widetilde{r}$. Equation (3.24) is a special case of the more general identity

$$(I_{\tau_j} \otimes Y_{ii}^{a})\widehat{B}_{ij}^{[b]} + ((Y_{jj}^{b})^T \otimes I_{\tau_i})\widehat{B}_{ij}^{[a]} = \widehat{B}_{ij}^{[a+b]}, \qquad a, b > 0,$$

whose proof is immediate.

A similar argument shows that $\widehat{P}_{ij} = \sum_{k=0}^{n} d_k \widehat{B}_{ij}^{[k]}$, and this concludes the proof of the lemma. □

In view of Lemma 3.5, Algorithm 3.2 is applicable if and only if Algorithm 3.1 is, thus Theorem 3.3 and Theorem 3.4 hold for Algorithm 3.2 with the same hypotheses.

Computational cost. As done in the previous section, we discuss the cost of Algorithm 3.2 for the case $\nu = N$ and $\tau_i = 1$ for $i = 1, \dots, N$.

Note that the coefficient of N^3 in the computational cost is obtained by counting, for all $1 \le i < j \le N$, the number of sums of the type (3.18) appearing in the pseudocode. Evaluating each of these sums requires $\frac{1}{3}N^3 + o(N^3)$ operations, and their number is exactly the same as that of the matrix multiplications and inversions needed in the Paterson–Stockmeyer evaluation scheme, that is, $\widetilde{r} + \widehat{r} + s$. Indeed the most expensive operations in the algorithm are: the sum on line 16, which is repeated $s-1$ times and is related to the recursion (3.4); the two sums on lines 22 and 24, which correspond to the evaluation of $C_k(A)$ and $D_k(A)$ and are performed $\widetilde{r}$ and $\widehat{r}$ times, respectively; and the sum on line 29, which is the counterpart of the final inversion in $q(Y)^{-1}p(Y)$. Therefore, the total cost of Algorithm 3.2 is $\frac{\widetilde{r}+\widehat{r}+s}{3}N^3 + o(N^3)$ flops, and the asymptotic cost of of solving the general equation (1.1) using Algorithm 3.2 is $\left(28 + \frac{\widetilde{r}+\widehat{r}+s}{3}\right)N^3 + o(N^3)$ flops.

4. Numerical experiments.

We compare experimentally the performance of Algorithm 3.1, Algorithm 3.2, and [16, Alg. 1] and give an example showing how these can be used to approximate matrix functions defined implicitly via matrix equations.

The experiments were run in MATLAB 2019a (version 9.6) on a machine equipped with an Intel I5-5287 processor running at 2.90GHz. We compare the following codes for the solution of rational matrix equations.

- `invrat_horn`, an implementation of [16, Alg. 1];
- `invrat_pow`, an implementation of Algorithm 3.1 based on the complex Schur decomposition;
- `invrat_ps`, an implementation of Algorithm 3.2 based on the complex Schur decomposition.

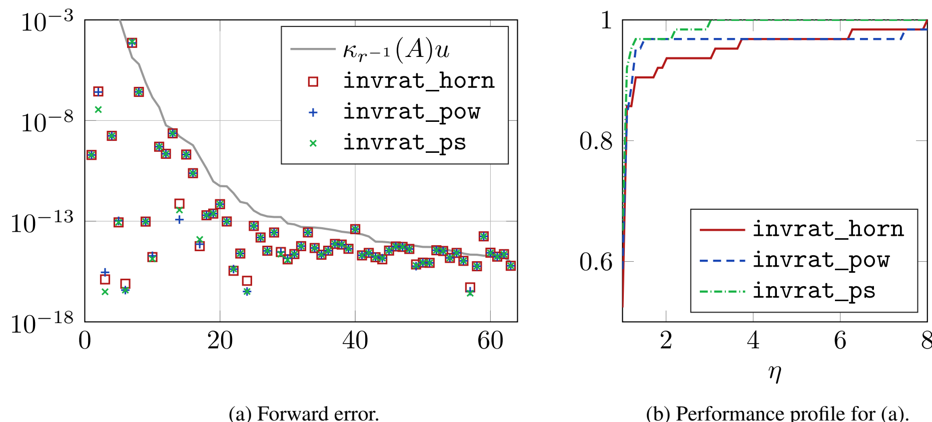
The implementations we used to perform the experiments in this section are available on the MATLAB Central File Exchange.[3]

We evaluate the stability of our algorithms by comparing the 1-norm forward error of the computed solutions with the quantity $\kappa_{f^{-1}}(A)u$, where $u = 2^{-53}$ is the unit roundoff of IEEE double precision arithmetic, and $\kappa_{f^{-1}}(A)$ is the 1-norm condition number of the solution to $f(X) = A$ for a specific choice of f^{-1}. Numerically, we estimate $\kappa_{f^{-1}}(A)$ by means of the function `funm_condest1` from the Matrix Function Toolbox [20], and the forward error by computing in double precision the quantity

$$e_{\mathcal{A}} = \frac{\|X_{\mathcal{A}} - X\|_1}{\|X\|_1},$$

[3] https://mathworks.com/matlabcentral/fileexchange/74317.

ETNA
Kent State University and
Johann Radon Institute (RICAM)



(a) Forward error. (b) Performance profile for (a).

FIG. 4.1. *Left: relative forward error of* `invrat_horn`, `invrat_pow`, *and* `invrat_ps` *for the matrices in the test set sorted by descending condition number* $\kappa_{r^{-1}}(A)$. *Right: corresponding performance profile.*

where $X_{\mathcal{A}}$ is the solution computed by algorithm $\mathcal{A}$ and X is a reference solution.

4.1. Numerical stability. In this first experiment, we assess the stability of the procedures `invrat_horn`, `invrat_pow`, and `invrat_ps` by performing an experiment similar to [16, Test 1]. As the order of the rational function used there is too low to show any remarkable difference among the three algorithms, we turn to a rational function of higher order and consider the $[5/5]$ Padé approximant to the exponential. The reference solution is obtained by running `invrat_pow` with 113 significant binary digits of accuracy, which corresponds to IEEE `binary128` floating point arithmetic [25, Table 3.2]. In order to work with precision higher than double, we rely on the overloaded methods of the Advanpix Multiprecision Computing Toolbox [1] (version 4.4.7.12739).

In Figure 4.1a, we compare the 1-norm forward error of the three algorithms for the same test as in [16, Test 1], which contains 63 nonnormal matrices of size 10 with no nonpositive real eigenvalues. We do not consider normal matrices, for which the triangular Schur factor is in fact diagonal, as `invrat_horn`, `invrat_pow`, and `invrat_ps` all reduce to diagonalization in that case. The fact that the forward error is always approximately bounded by $\kappa_{r^{-1}}(A)u$ suggests that the three implementations behave in a forward stable fashion. Note that the algorithms attain a remarkably similar accuracy for most matrices, and when that is not the case, the difference among the performance of the three methods is marginal.

Figure 4.1b presents the same data by means of a performance profile [12]. In the plot, the height of the line corresponding to algorithm $\mathcal{A}$ at $\eta = \eta_0$ represents the fraction of matrices in the test set on which the 1-norm relative forward error of $\mathcal{A}$ is at most η_0 times that of the algorithm that gives the most accurate result. The figure confirms that the accuracy of the three algorithms on our test set is very similar, but `invrat_ps` appears to be, overall, slightly more accurate than the two alternative approaches.

In order to draw more general results, we conducted a second experiment. For each pair of distinct implementations $\mathcal{A}$ and $\mathcal{B}$, we tried to find the 5×5 matrix A and, for different values of m, the $[m/m]$ rational function r that maximize the ratio $e_{\mathcal{A}}/e_{\mathcal{B}}$ when the two algorithms $\mathcal{A}$ and $\mathcal{B}$ are used to solve (1.1). As optimization method, we used the multidirectional search method of Dennis and Torczon [11], implemented in the `mdsmax` function of the Matrix Computation Toolbox [19].

ETNA
Kent State University and
Johann Radon Institute (RICAM)

TABLE 4.1
Maximum ratio of forward errors for all possible pairs of algorithms. The optimization procedure tries to determine a 5×5 matrix and the coefficients of the numerator and denominator of a $[m/m]$ rational function r that maximize $e_{\mathcal{A}}/e_{\mathcal{B}}$. Each cell contains four values corresponding to the cases $m = 3$ (top left), $m = 5$ (top right), $m = 7$ (bottom left), and $m = 9$ (bottom right).

$\mathcal{A}$	$\mathcal{B}$					
	`invrat_horn`		`invrat_pow`		`invrat_ps`	
`invrat_horn`	–	–	3.20	1.37	1.95	2.32
	–	–	1.53	1.73	1.42	1.61
`invrat_pow`	1.20	1.77	–	–	71.53	2.37
	1.37	2.92	–	–	2.74	1.29
`invrat_ps`	2.24	1.33	35.37	2.29	–	–
	1.59	2.16	1.51	2.25	–	–

In Table 4.1, we report these ratios for the four cases $m = 3$, 5, 7, and 9, which appear in the top left, top right, bottom left, and bottom right corners, respectively, of every cell. The results show that the three algorithms tend to behave similarly, as the forward error of one algorithm does not exceed that of any other by more than one order of magnitude in most cases.

We conclude that the three algorithms do not differ much in terms of accuracy, and that their good stability properties make them reliable enough to be of practical use.

4.2. Computational time. Figure 4.2 shows how the execution time required by the algorithms `invrat_horn`, `invrat_pow`, and `invrat_ps` to solve the matrix equation $r(X) = A$ depends on the size of the matrix A and on the order of the rational function r.

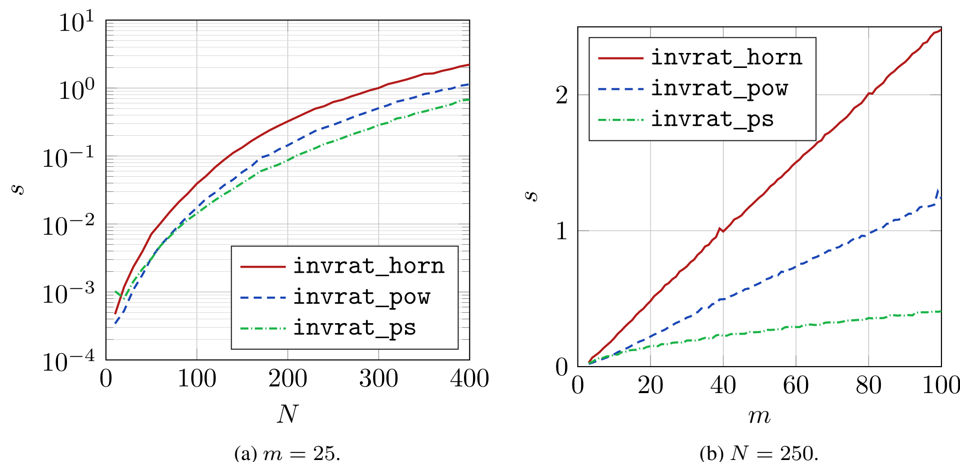
As the analysis of the computational cost shows, the time required to compute the Schur decomposition of the input matrix tends to be preponderant for rational functions of low order, thus we prefer not to take it into account and to feed the algorithm matrices that are already in upper triangular form. As the number of operations which the algorithms carry out depends on the number of nonzeros in the matrix but not on their numerical value, we use the 0-1 matrix $A \in \mathbb{C}^{N \times N}$ with $a_{ij} = 1$ for $1 \leq i \leq j \leq N$. A similar observation can be made for the coefficients of the rational functions, which we draw from a Gaussian distribution. The execution time is estimated by means of the MATLAB function `timeit`.

In Figure 4.2a, we fix the order of the rational function to 25 and let the size of the matrix increase from 10 to 400. The time required by the three algorithms rises more than linearly. For matrices of order 20 or more, `invrat_horn` is the slowest algorithm, `invrat_pow` is the fastest for matrices of size smaller than 50, whereas `invrat_ps` is the fastest for larger matrices. This is in line with the analysis of the computational cost.

In Figure 4.2b the algorithms are run for matrices of size 250 using rational functions of orders between 3 and 100. As expected, the execution time of `invrat_horn` and `invrat_pow` grows linearly with the order of the approximant, and the latter is always the fastest of the two. The execution time of `invrat_ps`, on the other hand, grows sublinearly, again following the analysis of the computational cost. The results for complex matrices with complex coefficients are qualitatively analogous.

4.3. Computing the Lambert *W* function. In order to illustrate how these algorithms can be employed to solve more general matrix equations of the form $f(X) = A$ where f is a primary matrix function, we consider the computation of the matrix Lambert W function [10] defined implicitly as any solution to the equation $X\mathrm{e}^X = A$, which is of interest

ETNA
Kent State University and
Johann Radon Institute (RICAM)



(a) $m = 25$.

(b) $N = 250$.

FIG. 4.2. *Execution time (in seconds) of* `invrat_horn`, `invrat_pow`, *and* `invrat_ps` *on matrices of increasing size N (left) and rational functions of increasing order m (right).*

in the analysis of the stability of delay differential equations [4, 8, 18, 27]. The inverse of the real function $[-1/e, \infty] \to \mathbb{R} : x \to xe^x$ can be extended analytically to a function $W_0(z) : \mathbb{C} \setminus (-\infty, 1/e) \to \mathbb{C}$, which is said to be the 0th branch of the Lambert W function.

For computing $W_0(A)$, we compare:

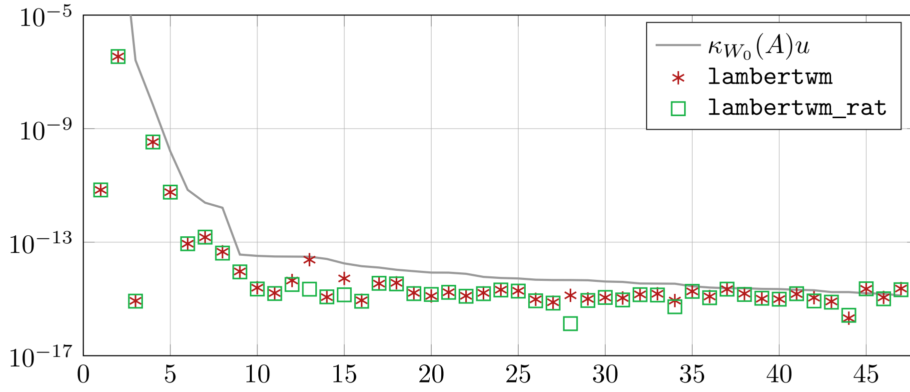
- `lambertwm`, an implementation of [15, Alg. 1];
- `lambertwm_rat`, an algorithm that uses `invrat_ps` to solve $r(X) = A$, where r is a diagonal Padé approximant to xe^x. In our experiments, we found that 28 is the lowest optimal degree for the Paterson–Stockmeyer method [14] that provides sufficient accuracy for all the matrices in the test set we consider.

Following the well-established paradigm of recomputing the diagonal of functions of upper triangular matrices in a direct way [2, 3], in `lambertwm_rat` we recompute the diagonal elements of Y using a direct formula. More precisely, if we were to follow the approach in Algorithms 3.1 and 3.2 exactly, after reducing the problem to the triangular form (1.2), we would obtain the diagonal blocks of Y by choosing the solution to $r(X) = T_{ii}$ that best approximates $W_0(T_{ii})$. Instead, we set $Y_{ii} = W_0(T_{ii})$ and continue the recursion using this value in lieu of $r^{-1}(T_{ii})$. The off-diagonal blocks are still computed by substitution. The main advantage of this technique is that we do not need to solve smaller matrix equation for non-trivial diagonal blocks. Moreover, our results show that this strategy leads to more accurate solutions in some cases.

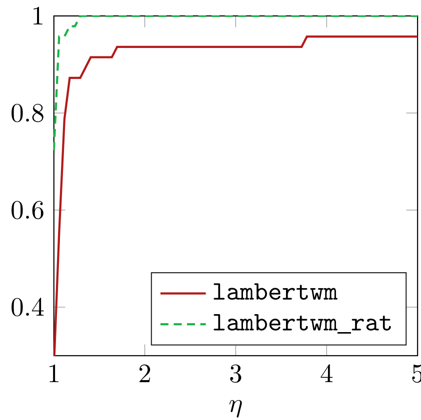
This modification can be seen as applying Algorithms 3.1 and 3.2 to the matrix equation $r(Y) = \widetilde{T}$, where $\widetilde{T}$ coincides with T except for the diagonal blocks, since T_{ii} is replaced by the block $\widetilde{T}_{ii}$ such that $r^{-1}(\widetilde{T}_{ii}) = W_0(T_{ii})$. When $W_0(T_{ii})$ is a 2×2 block arising from the real Schur decomposition, we use [16, Prop. 15].

We compare the performance of `lambertwm` and `lambertwm_rat` for the test set used in [15, Exp. 2], which contains 47 matrices of size 10 taken from the MATLAB `gallery`. As `lambertwm` is an iterative method, it cannot be easily extended to multiprecision, and to compute our reference solution we adopt the same algorithm used in [15, Exp. 2], which diagonalizes the matrix in higher precision by using the `eig` function provided by the Symbolic Math Toolbox [32].

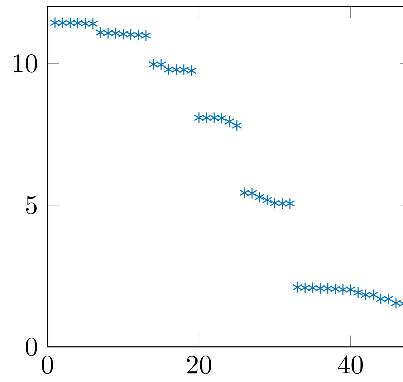
ETNA
Kent State University and
Johann Radon Institute (RICAM)



(a) Forward error.



(b) Performance profile for (a).



(c) Cost ratio.

FIG. 4.3. *Top: forward error of* `lambertwm` *and* `lambertwm_rat` *for the matrices in the test set sorted by descending condition number* $\kappa_{W_0}(A)$*. Bottom left: corresponding performance profile. Bottom right: ratio of the estimated computational costs of* `lambertwm` *and* `lambertwm_rat`.

Figure 4.3a compares the 1-norm forward error of `lambertwm` and `lambertwm_rat` with the quantity $\kappa_{W_0}(A)u$, and Figure 4.3b presents the same data by means of performance profiles. The results suggest that both algorithms behave in a forward stable way and achieve remarkably similar accuracy for most matrices in the test set. The algorithm `lambertwm_rat` achieves slightly higher accuracy for about a quarter of the matrices in the data set, which justifies the more favorable curves of this algorithm in the performance profile.

In Figure 4.3c, we compare the leading terms of the computational cost of the two implementations for the matrices in the test set. In both cases, $28N^3$ flops are required to compute the Schur decomposition and recovering the result at the end of the computation. The additional cost of `lambertwm_rat` depends only on the size of the matrix and on the order of the rational approximant and can be easily seen to be $13N^3/3$. On the other hand, `lambertwm` splits the matrix into two blocks and performs a few steps of the Newton method for each block. In assessing the flop count, we took into account the number of matrix multiplications, inversions, and square roots needed to compute the starting value and perform the required steps of the Newton method and the cost for the solution of the ensuing Sylvester equation

by means of the Bartels–Stewart algorithm [5]. We ignored the cost of ordering the Schur decomposition so that the eigenvalues appear along the diagonal of the triangular Schur factor in two clusters. The results show that, in the test set that we considered, the new algorithm is up to eleven times faster than `lambertwm`, and the speedup reaches a factor seven for more than half of the matrices in the test set.

These differences are not as apparent in our MATLAB implementations, where the gain of `lambertwm_rat` over `lambertwm` is limited. This is not surprising: the former method mostly comprises element-wise operations that are interpreted by MATLAB at runtime, whereas the latter is rich in matrix multiplications and inversions, kernels for which MATLAB relies on highly optimized C and Fortran libraries.

As we use the Padé approximant centered at 0, the accuracy of this algorithm deteriorates for matrices with large eigenvalues. In order to make the new algorithm a reliable alternative to `lambertwm`, this case has to be addressed. This will be the subject of future work.

5. Conclusions. We developed two new algorithms for solving rational matrix equations that are more efficient than existing algorithms for the same problem. These new techniques invert two customary methods for the evaluation of rational matrix functions, one based on the explicit powering technique and the other on the Paterson–Stockmeyer algorithm. Our experiments suggest that the new techniques are as accurate as existing alternatives: this is consistent with the error analysis of similar algorithms for the square root and pth root, as well as with the stability of the corresponding methods for the evaluation of rational matrix functions, which are all equivalently stable [22, Thm. 4.5]. We characterized the applicability of substitution algorithms for rational matrix equations in a way that feels more natural than previous attempts in the literature [16].

General functional matrix equations such as those that define the matrix logarithm or the matrix Lambert W function can be reduced to rational form by means of rational approximation. We briefly discussed how this strategy can be exploited to develop a naïve method for computing the Lambert W function that, in a number of cases, is more accurate and efficient than the reference algorithm [15]. An analogous strategy for the matrix logarithm or the inverse sine and cosine functions would not provide an advantage over the current state-of-the-art algorithms.

Nevertheless, the diagonal Padé approximants to the exponential, sine, and cosine all show symmetries in their coefficients. If these patterns were exploited, one could in principle deliver faster substitution algorithms tailored to the solution of specific problems. We intend to investigate this in future work.

REFERENCES

[1] ADVANPIX, *Multiprecision Computing Toolbox*, Advanpix, Tokyo. http://www.advanpix.com
[2] A. H. AL-MOHY AND N. J. HIGHAM, *A new scaling and squaring algorithm for the matrix exponential*, SIAM J. Matrix Anal. Appl., 31 (2009), pp. 970–989.
[3] ———, *Improved inverse scaling and squaring algorithms for the matrix logarithm*, SIAM J. Sci. Comput., 34 (2012), pp. C153–C169.
[4] F. M. ASL AND A. G. ULHSOY, *Analysis of a system of linear delay differential equations*, J. Dyn. Sys. Meas. Control., 125 (2003), pp. 215–223.
[5] R. H. BARTELS AND G. W. STEWART, *Algorithm 432: Solution of the matrix equation $AX + XB = C$*, Comm. ACM, 15 (1972), pp. 820–826.
[6] A. BJÖRCK AND S. HAMMARLING, *A Schur method for the square root of a matrix*, Linear Algebra Appl., 52/53 (1983), pp. 127–140.
[7] M. BLADT AND M. SØRENSEN, *Efficient estimation of transition rates between credit ratings from observations at discrete time points*, Quant. Finance, 9 (2009), pp. 147–160.

[8] R. CEPEDA-GOMEZ AND W. MICHIELS, *Some special cases in the stability analysis of multi-dimensional time-delay systems using the matrix Lambert W function*, Automatica J. IFAC, 53 (2015), pp. 339–345.
[9] T. CHARITOS, P. R. DE WAAL, AND L. C. VAN DER GAAG, *Computing short-interval transition matrices of a discrete-time Markov chain from partially observed data*, Stat. Med., 27 (2008), pp. 905–921.
[10] R. M. CORLESS, G. H. GONNET, D. E. G. HARE, D. J. JEFFREY, AND D. E. KNUTH, *On the Lambert W function*, Adv. Comput. Math., 5 (1996), pp. 329–359.
[11] J. E. DENNIS, JR. AND V. TORCZON, *Direct search methods on parallel machines*, SIAM J. Optim., 1 (1991), pp. 448–474.
[12] E. D. DOLAN AND J. J. MORÉ, *Benchmarking optimization software with performance profiles*, Math. Program., 91 (2002), pp. 201–213.
[13] J.-C. EVARD AND F. UHLIG, *On the matrix equation $f(X) = A$*, Linear Algebra Appl., 162/164 (1992), pp. 447–519.
[14] M. FASI, *Optimality of the Paterson-Stockmeyer method for evaluating matrix polynomials and rational matrix functions*, Linear Algebra Appl., 574 (2019), pp. 182–200.
[15] M. FASI, N. J. HIGHAM, AND B. IANNAZZO, *An algorithm for the matrix Lambert W function*, SIAM J. Matrix Anal. Appl., 36 (2015), pp. 669–685.
[16] M. FASI AND B. IANNAZZO, *Computing primary solutions of equations involving primary matrix functions*, Linear Algebra Appl., 560 (2019), pp. 17–42.
[17] F. GRECO AND B. IANNAZZO, *A binary powering Schur algorithm for computing primary matrix roots*, Numer. Algorithms, 55 (2010), pp. 59–78.
[18] J. M. HEFFERNAN AND R. M. CORLESS, *Solving some delay differential equations with computer algebra*, Math. Sci., 31 (2006), pp. 21–34.
[19] N. J. HIGHAM, *The Matrix Computation Toolbox*, Website.
http://www.maths.manchester.ac.uk/~higham/mctoolbox
[20] ———, *The Matrix Function Toolbox*, Website.
http://www.maths.manchester.ac.uk/~higham/mftoolbox
[21] ———, *Computing real square roots of a real matrix*, Linear Algebra Appl., 88/89 (1987), pp. 405–430.
[22] ———, *Functions of Matrices: Theory and Computation*, SIAM, Philadelphia, 2008.
[23] N. J. HIGHAM AND L. LIN, *An improved Schur-Padé algorithm for fractional powers of a matrix and their Fréchet derivatives*, SIAM J. Matrix Anal. Appl., 34 (2013), pp. 1341–1360.
[24] B. IANNAZZO AND C. MANASSE, *A Schur logarithmic algorithm for fractional powers of matrices*, SIAM J. Matrix Anal. Appl., 34 (2013), pp. 794–813.
[25] IEEE, *IEEE Standard for Floating-Point Arithmetic. IEEE Std. 754-2008 (revision of IEEE Std. 754-1985)*, The Institute of Electrical and Electronics Engineers, New York, August 2008.
[26] 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 (2001), pp. 245–265.
[27] E. JARLEBRING AND T. DAMM, *The Lambert W function and the spectrum of some multidimensional time-delay systems*, Automatica J. IFAC, 43 (2007), pp. 2124–2128.
[28] M. S. PATERSON AND L. J. STOCKMEYER, *On the number of nonscalar multiplications necessary to evaluate polynomials*, SIAM J. Comput., 2 (1973), pp. 60–66.
[29] B. SINGER AND S. SPILERMAN, *The representation of social processes by Markov models*, Am. J. Sociol., 82 (1976), pp. 1–54.
[30] M. I. SMITH, *A Schur algorithm for computing matrix pth roots*, SIAM J. Matrix Anal. Appl., 24 (2003), pp. 971–989.
[31] T. TAKADA, A. MIYAMOTO, AND S. F. HASEGAWA, *Derivation of a yearly transition probability matrix for land-use dynamics and its applications*, Landsc. Ecol., 25 (2009), pp. 561–572.
[32] THE MATHWORKS, *Symbolic Math Toolbox*, Software, The MathWorks, Natick.
http://www.mathworks.co.uk/products/symbolic/