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 block triangular matrices, the new methods yield the same computational cost as the evaluation schemes from which they are obtained. A general equation is reduced to block upper triangular form by exploiting the Schur decomposition of the given matrix. For real data, using the real Schur decomposition, the algorithms compute the real solutions using only real arithmetic, and numerical experiments show that our implementations are superior, in terms of both accuracy and speed, to existing alternatives. Finally, we discuss how these techniques can be used as building blocks in algorithms for computing functions of matrices defined through matrix equation of the type $f(X) = A$, where f is a primary matrix function.

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

AMS subject classifications. 15A24, 65F60

1. Introduction. Many functions of matrices that arise in applications are defined implicitly as solutions to functional matrix equations of the form

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

where f is a complex function applied to a matrix (in the sense of primary matrix functions as defined in [20, Chap. 1]) and A and X are square matrices of the same size. This equation has been extensively studied in the literature, and a number of theoretical results, such as existence, uniqueness, and classification of real and complex solutions, are discussed by Evard and Uhlig [11].

The most prominent instance of (1.1) is probably the equation $X^\alpha = A$, which defines the matrix αth root [21], [22], [30], [4], a function that finds applications in sociology [29], economy [5], [24], healthcare [7], and ecology [32]. Other notable examples include the matrix logarithm [1], [26], which is a solution to the matrix equation $\exp X = A$, and the matrix Lambert W function [13], a lesser-known special function defined implicitly by the matrix equation $X \exp X = A$, which is of interest in the analysis of the stability of delay differential equations [2], [6], [16], [25].

Here we focus on a class of matrix equations that sits between the simple monomial equations that define the matrix roots and the transcendental ones that give rise to the logarithm and the Lambert W function. In particular, we examine rational matrix equations of the form

$$r(X) = A, \tag{1.2}$$

[*]Version of 12th May 2019.

Funding: The work of the first author was supported by MathWorks. 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.

[†]School 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).

where $r = p/q$, with p and q coprime polynomials of degree m and n, respectively. Many state-of-the-art algorithms for computing matrix functions rely on rational approximation, and compute $f(A)$ by evaluating at A a rational function of suitable degree. Here we show that a similar approach can be followed to compute solutions to (1.1), which can be reduced to an equation of the form (1.2) after replacing the function f by a rational approximant $r \approx f$.

A reliable algorithm for the numerical solution of rational matrix equations has been recently proposed [14]. This method, based on a substitution (or recursion) procedure, is the culmination of a line of algorithms for the computation of matrix roots [15], [22], [30]. The common idea of these techniques is to reduce the problem to an equation of the type

$$r(Y) = T, \tag{1.3}$$

where Y and T are block upper triangular matrices with the same block structure, and then operate at the block level. Indeed, 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 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 instance, for the αth root, Smith [30] obtains a substitution algorithm by symbolically constructing, with $\alpha - 1$ matrix multiplications, the first α powers of the solution Y, namely

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

and deducing, recursively, 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 $\frac{1}{3}(\alpha - 1)N^3 + o(N^3)$ operations for a triangular $N \times N$ matrix. This is the case, for example, if the complex Schur form is used. A variant, proposed by Iannazzo and Manasse [22] (as an update of prior work by Greco and Iannazzo [15]), 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 [14] obtain an explicit expression for the off-diagonal blocks of Y in (1.3). Their algorithm is based on 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, for equation (1.2), requires $\frac{1}{3}(m + n - 1)N^3 + o(N^3)$ operations, which asymptotically is the same as the evaluation scheme itself.

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 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.

Our contribution is threefold: we develop new algorithms for the solution of (1.2), extend some of the theoretical results in [14] regarding the existence of solutions, and show how these algorithms can be exploited to solve a wide range of practical problems.

From a numerical point of view, we propose two substitution algorithms for computing primary solutions to (1.2), one based on the explicit powering technique and the other on the Paterson–Stockmeyer scheme. As for the approach based on Horner's method [14], the asymptotic computational cost of these new algorithms turns out to be the same as that of the evaluation scheme itself, and thus lower than that of the state-of-the-art algorithm for the solution of this computational problem [14].

From a theoretical point of view, it has be shown [14, Cor. 14] that if a solution to (1.2) 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, the only one in a neighborhood. Here we extend this result and show that requiring the existence of the solution is not necessary: if $\lambda_1, \dots, \lambda_n$ 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 unique a solution to (1.2) with eigenvalues $\xi_1, \dots, \xi_n$, which is isolated and can be computed by any of our substitution algorithms, including [14, Alg. 1]. The precise statement can be found in Theorem 3.3.

Turning to applications, we discuss how these techniques can be used to evaluate more general matrix functions defined implicitly by equations of the form (1.1). In particular, we develop an algorithm for computing the Lambert W function which, according to our experiments, is faster and more accurate than the state-of-the-art technique [13, 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 [11], [14]. 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 reminder, we always refer to the $[m/n]$ rational function

$$r(z) = q(z)^{-1} p(z), \tag{2.1}$$

whose numerator and denominator are the polynomials

$$p(z) := \sum_{k=0}^{m} c_k z^k \in \mathbb{C}_m[z], \qquad \text{and} \qquad q(z) := \sum_{k=0}^{n} d_k z^k \in \mathbb{C}_n[z], \tag{2.2}$$

respectively. Without restriction, we assume that p and q are coprime, that is, that 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 implicitly 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, 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 the powers $A^2, \dots, A^m$ 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, 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

$$\left\{ \begin{array}{ll} p^{[m]}(A) = c_m I, & \\ p^{[j]}(A) = Ap^{[j+1]}(A) + c_j I, & \qquad \text{for } j = 0, \dots, m-1, \end{array} \right. \tag{2.3}$$

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

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

$$p(A) = \sum_{k=0}^{\widetilde{r}} (A^s)^k C_k(A), \qquad \widetilde{r} = \left\lceil \frac{m}{s} - 1 \right\rceil \tag{2.4}$$

and

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

An analogous rewriting for $q(A)$ yields

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

where D_k 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 simplicity's sake, we adopt a primed sum notation for C_k and D_k, 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 $m - s\widehat{r}$, respectively.

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

$$L_s(m,n) = s - 1 + \widetilde{r} + \widehat{r}. \tag{2.5}$$

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 (2.4) should be evaluated à la Horner by constructing the sequence

$$\begin{cases} P^{[\widetilde{r}]}(A) = C_{\widetilde{r}}(A), \\ P^{[j]}(A) = A^s P^{[j+1]}(A) + C_j(A), \qquad j = 0, \ldots, \widetilde{r} - 1, \end{cases} \tag{2.6}$$

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 identities

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

which can be proved by a direct computation.

Classifying the solutions to matrix equations. Let $f : \mathbb{C}^{N\times N} \to \mathbb{C}^{N\times N}$ be a primary matrix function, in the sense of [20, Def. 1.2, Def. 1.4, Def. 1.11], for some f 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 (1.1) such that f is defined on the spectrum of X [20, Def. 1.1]. X is a *primary solution* if there exists a polynomial χ such that $X = \chi(A)$, and is *isolated* if there exists a neighborhood $\mathcal{U}$ of X where X is the unique solution to (1.1).

An eigenvalue ξ of X is said to be critical if $f'(\xi) = 0$. Evard and Uhlig [11, 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. On the other hand, Fasi and Iannazzo [14, 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 consider also 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 (1.2). In fact, in the discussion below we consider the seemingly simpler equation

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

where

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

and $Y \in \mathbb{C}^{N\times N}$ has the same structure as T. This is not a restriction if only primary solutions are sought, since when p and q are coprime any matrix equation of the form (1.2) can be reduced to an equation of the form (3.1), as we now explain.

On the one hand, it has been shown [14, Prop. 9] that if p and q are coprime, then X satisfies (1.2) 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, primary solutions have the same block upper triangular structure (being polynomials of T) and we can conclude that (3.1) is equivalent to (1.2).

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, then $\chi(Y)_{ii} = \chi(Y_{ii})$ for $i = 1, \ldots, \nu$.

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

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

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 $i = 1, \ldots, \nu$;
2. Compute L_{ij} and b_{ij} and solve the linear matrix equation $L_{ij}(Y) = b_{ij}$ for Y_{ij}, for $1 \le i < j \le \nu$.

The first step is easy to perform when T is a Schur normal form: for 1×1 blocks, it suffices to solve the corresponding scalar equation, whereas for 2×2 real blocks with complex conjugate eigenvalues, [14, 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, the equation for Y_{ij} is obtained by substituting into the expression for b_{ij} the blocks of Y that have been already 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 of 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 [4], while in the real case, with blocks of size at most 2×2, we recover the algorithm of Higham [19]. In the former case, the equation on 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 substitution algorithms 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 [15], [22]. 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 algorithm.

Our previous algorithm [14] 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 a rational matrix functions [20, Chap. 4].

3.2. Algorithm based on explicit powers. Let $\mu = \max\{m,n\}$, and 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 $p(Y)_{\overline{\imath}\overline{\jmath}}$, $q(Y)_{\overline{\imath}\overline{\jmath}}$, and $Y^{[k]}_{\overline{\imath}\overline{\jmath}}$, with $k = 1, \dots, \mu$, such that $\overline{\jmath} - \overline{\imath} < j - i$. 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 [14, Alg. 1], the key difference being the sequence of stages that is used to derive L_{ij} and b_{ij} in the equations in (3.4). In fact, while [14, 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

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

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

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

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

$$B^{[k]}_{ij}(V) := \sum_{u=0}^{k-1} Y^{[u]}_{ii} V Y^{[k-1-u]}_{jj}, \qquad k = 1, \dots, \mu, \tag{3.9}$$

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

$$Y^{[k]}_{ij} = B^{[k]}_{ij}(Y_{ij}) + \sum_{u=0}^{k-2} Y^{[u]}_{ii} F^{[k-u]}_{ij}, \qquad k = 1, \dots, \mu. \tag{3.10}$$

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

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

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

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

where $p^{[k]}(z)$ and $q^{[k]}(z)$ are the stages of Horner's scheme (see (2.3)) applied to p and q, respectively, and we have used the identity

$$\sum_{k=1}^{m} c_k \sum_{u=0}^{k-2} Y^{[u]}_{ii} F^{[k-u]}_{ij} = \sum_{k=1}^{m} c_k \sum_{u=2}^{k} Y^{[k-u]}_{ii} F^{[u]}_{ij} = \sum_{u=2}^{m} \sum_{k=u}^{m} c_k Y^{k-u}_{ii} F^{[u]}_{ij} = \sum_{u=2}^{m} p^{[u]}(Y_{ii}) F^{[u]}_{ij}.$$

Algorithm 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  $\mu \leftarrow \max\{m, n\}$.
2  for $i = 1$ to $\nu$ do
3      $Y_{ii}^{[1]} \leftarrow$ a solution to $p(Y) - T_{ii}q(Y) = 0$
4      for $k = 2$ to $\mu$ do
5          $Y_{ii}^{[k]} \leftarrow Y_{ii}^{[1]} Y_{ii}^{[k-1]}$
6      $P_{ii}^{[m-1]} \leftarrow c_{m-1} I_{\tau_i} + c_m Y_{ii}$
7      for $k = m - 2$ downto $0$ do
8          $P_{ii}^{[k]} \leftarrow c_k I_{\tau_i} + Y_{ii} P_{ii}^{[k+1]}$
9      $Q_{ii}^{[n-1]} \leftarrow d_{n-1} I_{\tau_i} + d_n Y_{ii}$
10     for $k = n - 2$ downto $0$ do
11         $Q_{ii}^{[k]} \leftarrow d_k I_{\tau_i} + Y_{ii} Q_{ii}^{[k+1]}$
12 for $v = 1$ to $\nu - 1$ do
13     for $i = 1$ to $\nu - v$ do
14         $j \leftarrow i + v$
15         for $k = 2$ to $\mu$ do
16             $F_{ij}^{[k]} \leftarrow \sum_{t=i+1}^{j-1} Y_{it}^{[1]} Y_{tj}^{[k-1]}$
17         $M_{ij} \leftarrow \sum_{k=1}^{m} (P_{jj}^{[k]})^T \otimes Y_{ii}^{[k-1]} - (I_{\tau_j} \otimes T_{ii}) \sum_{k=1}^{n} (Q_{jj}^{[k]})^T \otimes Y_{ii}^{[k-1]}$
18         $b_{ij} \leftarrow \operatorname{vec}\Big( \sum_{t=i+1}^{j} T_{it} Q_{tj} - \sum_{k=2}^{m} P_{ii}^{[k]} F_{ij}^{[k]} + T_{ii} \sum_{k=2}^{n} Q_{ii}^{[k]} F_{ij}^{[k]} \Big)$
19         $Y_{ij}^{[1]} \leftarrow \operatorname{vec}^{-1}(M_{ij}^{-1} b_{ij})$
20         for $k = 2$ to $\mu$ do
21             $Y_{ij}^{[k]} \leftarrow Y_{ii}^{[1]} Y_{ij}^{[k-1]} + Y_{ij}^{[1]} Y_{jj}^{[k-1]} + F_{ij}^{[k]}$
22         $Q_{ij} \leftarrow \sum_{k=1}^{n} d_k Y_{ij}^{[k]}$
```

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 taking the vec of 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 are determined, 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 1.

The algorithm has a breakdown if some of the matrices M_{ij} is singular, in which case the linear system $M_{ij}Y = \varphi_{ij}$ does not have unique solution. In the next lemma, we show that these matrices are the same as those appearing in [14, Eq. (21)], which allows us to relate the applicability of Algorithm 1 to that of the algorithms in [14].

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*

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

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

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

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

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

and similarly that

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

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 [14, Prop. 15].

For the complex Schur form, M_{ij} is the same as ψ_{ij} in [14, Eq. (12)], and the applicability of the algorithm depends on what solutions of the scalar equation $p(Y) = T_{ii}q(Y)$ are chosen, for $i = 1, \dots, \nu$.

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 $\xi_1, \dots, \xi_N \in \mathbb{C}$ be such that $r(\xi_i) = t_{ii}$ for $i = 1, \dots, N$. Then the two conditions are equivalent:*

(*a*) *Algorithm 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 $1 \leq i < j \leq N$.*
If either condition is satisfied, then the solution Y is primary and isolated.

Proof. The identity [14, Lemma 11] allows us to express the divided differences of p on $a, b \in \mathbb{C}$, 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 [14, 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 of (1.2).

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 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 $1 \leq i < j \leq N$. 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 $r(\xi_1), \dots, r(\xi_N)$ [20]. From [14, 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 $U^* A U = T$ 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 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 [14, 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 [14, 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, 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 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 $1 \leq i < j \leq \nu$;*

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

If Ξ_i is an isolated solution of the equation $r(X) = T_{ii}$ for $i = 1, \dots, \nu$, and either of the conditions above is satisfied, then Algorithm 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 $U_j^* Y_{jj}^T U_j$ 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 $1 \leq a \leq \tau_i$ and $1 \leq b \leq \tau_i$ with $a \neq b$. This condition, together with Theorem 3.4(*b*), implies that $r[\zeta_i, \zeta_j] \neq 0$ for $1 \leq i < j \leq N$, where $\zeta_1, \dots, \zeta_N$ are the eigenvalues of the computed solution Y, which is isolated by [14, 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 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 $1 \leq i < j \leq N$, a 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 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 (1.2), using Algorithm 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 $1 \leq i < j \leq \nu$, an 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 whose 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 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 of the left hand side depend on Y_{ij}, while the third can be treated as a known quantity, we need to deduce an expression for $p(Y)_{ij}$ and $q(Y)_{ij}$ that involves 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 (2.6) 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[1]

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

with

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

In order to get an equation in terms of Y_{ij} and known quantities only, we note the third summand of 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

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

with

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

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

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

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

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

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

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

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

where

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

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 $Y_{ij} = \operatorname{vec}^{-1}(M_{ij}^{-1} \operatorname{vec}(b_{ij}))$.

The pseudocode of the algorithm based on the Paterson–Stockmeyer approach is given in Algorithm 2. In order to reduce the overall computational cost of that method, note that $\widehat{B}_{ij}^{[k]}$ 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

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

Indeed, the sums on 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 the Algorithm 2 is applicable if and only if Algorithm 1 is.

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

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

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

Algorithm 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 $p(Y^{[1]})q^{-1}(Y^{[1]}) = T$.
$\widetilde{r} \leftarrow \lfloor m/s \rfloor$
$\widehat{r} \leftarrow \lfloor n/s \rfloor$
for $i = 1$ to $\nu$ do
    $Y_{ii}^{[0]} \leftarrow I_{\tau_i}$
    $Y_{ii}^{[1]} \leftarrow$ a solution to $p(Y) - T_{ii}q(Y) = 0$
    for $k = 2$ to $s$ do $Y_{ii}^{[k]} \leftarrow Y_{ii}^{[1]} Y_{ii}^{[k-1]}$
    $P_{ii}^{[\widetilde{r}]} \leftarrow \sum_{u=0}^{m-\widetilde{r}s} c_{s\widetilde{r}+u} Y_{ii}^{[u]}$
    for $k = \widetilde{r} - 1$ downto $1$ do $P_{ii}^{[k]} \leftarrow Y_{ii}^{[s]} P_{ii}^{[k+1]} + \sum_{u=0}^{s-1} c_{sk+u} Y_{ii}^{[u]}$
    $Q_{ii}^{[\widehat{r}]} \leftarrow \sum_{u=0}^{n-\widehat{r}s} d_{s\widehat{r}+u} Y_{ii}^{[u]}$
    for $k = \widehat{r} - 1$ downto $0$ do $Q_{ii}^{[k]} \leftarrow Y_{ii}^{[s]} Q_{ii}^{[k+1]} + \sum_{u=0}^{s-1} d_{sk+u} Y_{ii}^{[u]}$
for $v = 1$ to $\nu - 1$ do
    for $i = 1$ to $\nu - v$ do
        $j \leftarrow i + v$
        $\widehat{B}_{ij}^{[1]} \leftarrow I_{\tau_i \tau_j}$
        for $k = 2$ to $s$ do
            $F_{ij}^{[k]} \leftarrow \sum_{t=i+1}^{j-1} Y_{it}^{[1]} Y_{tj}^{[k-1]}$
            $\widehat{B}_{ij}^{[k]} \leftarrow \left( \big(Y_{jj}^{[1]}\big)^T \otimes I_{\tau_i} \right) \widehat{B}_{ij}^{[k-1]} + I_{\tau_j} \otimes Y_{ii}^{[k-1]}$
        $\Phi_{ij}^{[2]} \leftarrow F_{ij}^{[2]}$
        for $k = 3$ to $s$ do
            $\Phi_{ij}^{[k]} \leftarrow F_{ij}^{[k]} + Y_{ii} \Phi_{ij}^{[k-1]}$
        for $k = 0$ to $\widetilde{r} - 1$ do
            $\widetilde{\Psi}_{ij}^{[k]} = \sum_{t=i+1}^{j-1} Y_{it}^{[s]} P_{tj}^{[k+1]}$
        for $k = 0$ to $\widehat{r} - 1$ do
            $\widehat{\Psi}_{ij}^{[k]} = \sum_{t=i+1}^{j-1} Y_{it}^{[s]} Q_{tj}^{[k+1]}$
        $\widetilde{K} \leftarrow \sum_{k=0}^{\widetilde{r}-1} \Big( \big(P_{jj}^{[k+1]}\big)^T \otimes \big(Y_{ii}^{[s]}\big)^k \Big)$
        $\widehat{K} \leftarrow \sum_{k=0}^{\widehat{r}-1} \Big( \big(Q_{jj}^{[k+1]}\big)^T \otimes \big(Y_{ii}^{[s]}\big)^k \Big)$
        $\varphi_p \leftarrow \sum_{k=0}^{\widetilde{r}} \big(Y_{ii}^{[s]}\big)^k {\sum_{u=2}^{s-1}}' c_{sk+u} \Phi_{ij}^{[u]} + \mathrm{vec}^{-1}(\widetilde{K}\mathrm{vec}(\Phi_{ij}^{[s]})) + \sum_{k=0}^{\widetilde{r}-1} \big(Y_{ii}^{[s]}\big)^k \widetilde{\Psi}_{ij}^{[k]}$
        $\varphi_q \leftarrow \sum_{k=0}^{\widehat{r}} \big(Y_{ii}^{[s]}\big)^k {\sum_{u=2}^{s-1}}' d_{sk+u} \Phi_{ij}^{[u]} + \mathrm{vec}^{-1}(\widehat{K}\mathrm{vec}(\Phi_{ij}^{[s]})) + \sum_{k=0}^{\widehat{r}-1} \big(Y_{ii}^{[s]}\big)^k \widehat{\Psi}_{ij}^{[k]}$
        $\varphi_{ij} \leftarrow \mathrm{vec} \left( \sum_{t=i+1}^{j} T_{it} Q_{tj}^{[0]} - \varphi_p + T_{ii} \varphi_q \right)$
        $M_p \leftarrow \sum_{k=0}^{\widetilde{r}} \Big( I_{\tau_j} \otimes \big(Y_{ii}^{[s]}\big)^k \Big) {\sum_{u=1}^{s-1}}' c_{sk+u} \widehat{B}_{ij}^{[u]} + \widetilde{K} \widehat{B}_{ij}^{[s]}$
        $M_q \leftarrow \sum_{k=0}^{\widehat{r}} \Big( I_{\tau_j} \otimes \big(Y_{ii}^{[s]}\big)^k \Big) {\sum_{u=1}^{s-1}}' d_{sk+u} \widehat{B}_{ij}^{[u]} + \widehat{K} \widehat{B}_{ij}^{[s]}$
        $Y_{ij}^{[1]} \leftarrow \mathrm{vec}^{-1} \left( \big( M_p - (I_{\tau_j} \otimes T_{ii}) M_q \big)^{-1} \varphi_{ij} \right)$
        for $k = 2$ to $s$ do
            $Y_{ij}^{[k]} \leftarrow Y_{ii}^{[1]} Y_{ij}^{[k-1]} + Y_{ij}^{[1]} Y_{jj}^{[k-1]} + F_{ij}^{[k]}$
        $P_{ij}^{[\widetilde{r}]} \leftarrow \sum_{u=1}^{m-\widetilde{r}s} c_{s\widetilde{r}+u} Y_{ij}^{[u]}$
        for $k = \widetilde{r} - 1$ downto $1$ do
            $P_{ij}^{[k]} \leftarrow \widetilde{\Psi}_{ij}^{[k]} + Y_{ij}^{[s]} P_{jj}^{[k+1]} + Y_{ii}^{[s]} P_{ij}^{[k+1]} + \sum_{u=1}^{s-1} c_{sk+u} Y_{ij}^{[u]}$
        $Q_{ij}^{[\widehat{r}]} \leftarrow \sum_{u=1}^{n-\widehat{r}s} d_{s\widehat{r}+u} Y_{ij}^{[u]}$
        for $k = \widehat{r} - 1$ downto $0$ do
            $Q_{ij}^{[k]} \leftarrow \widehat{\Psi}_{ij}^{[k]} + Y_{ij}^{[s]} Q_{jj}^{[k+1]} + Y_{ii}^{[s]} Q_{ij}^{[k+1]} + \sum_{u=1}^{s-1} d_{sk+u} Y_{ij}^{[u]}$
```

Proof. We prove that $M'_{ij} = M_{ij}$ by showing, for the matrices in (3.23), that $\widetilde{P}_{ij} = \sum_{k=0}^{m} c_k \widehat{B}_{ij}^{[k]}$ and $\widehat{P}_{ij} = \sum_{k=0}^{m} d_k \widehat{B}_{ij}^{[k]}$.

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

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

while for the second we can obtain

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

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

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

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

$$(I \otimes Y_{ii}^{a})\widehat{B}_{ij}^{[b]} + ((Y_{jj}^{b})^T \otimes I)\widehat{B}_{ij}^{[a]} = \widehat{B}_{ij}^{[a+b]}, \qquad a, b > 0, \tag{3.25}$$

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 2 is applicable if and only if Algorithm 1 is, thus Theorem 3.3 and Theorem 3.4 hold for Algorithm 2 with the same hypotheses.

Computational cost. As done in the previous section, we discuss the cost of Algorithm 2 for the case $\nu = N$ and $\tau_i = 1$ for $i = 1, \ldots, N$.

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 matrix multiplications and inversions needed in the Paterson–Stockmeyer evaluation scheme, that is, $\widetilde{r} + \widehat{r} + s$. Indeed the most expensive operations the algorithm performs 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 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 (1.2), using Algorithm 2, is $\big(28 + \frac{\widetilde{r}+\widehat{r}+s}{3}\big)N^3 + o(N^3)$ flops.

4. Numerical experiments. We compare experimentally the performance of Algorithm 1, Algorithm 2, and [14, Alg. 1], and give an example showing how these can be used to approximate matrix functions defined via matrix equations of the form $f(X) = A$, by a solution of $r(X) = A$, where r is a rational approximant to f.

The experiments were run in MATLAB 2019a (version 9.6) on a machine equipped with an Intel I5-5287 processor running at 2.90GHz. The three algorithms for the solution of rational matrix functions we consider are:

- `invrat_horn`, an implementation of [14, Alg. 1];
- `invrat_pow`, an implementation of Algorithm 1 based on the complex Schur decomposition;
- `invrat_ps`, an implementation of Algorithm 2 based on the complex Schur decomposition.

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 [8], defined implicitly as any solution to the equation $X\mathrm{e}^X = A$. The inverse of the real function $[-1/e, \infty] \to \mathbb{R}$ that maps 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:

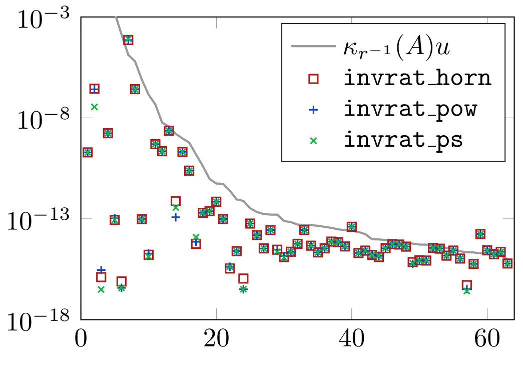
- `lambertwm`, an implementation of [13, Alg. 1];
- `lambertwm_rat`, an algorithm that uses `invrat_ps` to solve $r(X) = A$, where r is a diagonal Padé approximant to xe^x. In our experiments, we found that 28 is the lowest optimal degree for the Paterson–Stockmeyer method [12] that provides sufficient accuracy for all the matrices in the test set we consider. However, the algorithm is not used exactly as described in previous sections. If we were to follow the approach in Algorithms 1 and 2 exactly, after reducing the problem to the triangular form (1.3) we would obtain the diagonal blocks of Y by choosing the solution to $r(X) = T_{ii}$ that best approximates $W_0(T_{ii})$. In practice, we found that better accuracy can be obtained by setting $Y_{ii} = W_0(T_{ii})$ and continuing the recursion using this value in lieu of of $r^{-1}(T_{ii})$. The off-diagonal blocks are still computed by substitution.

We evaluate the stability of these 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 [18], and the forward error by computing in double precision the quantity

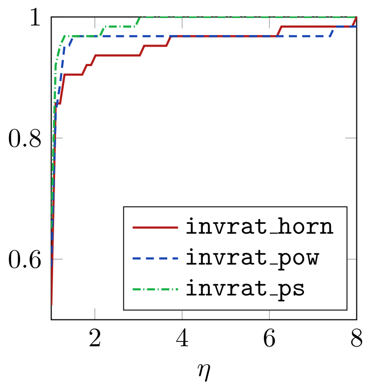
$$e_{\mathcal{A}} = \frac{\|X_{\mathcal{A}} - X\|_1}{\|X\|_1},$$

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 `invrat_horn`, `invrat_pow`, and `invrat_ps` by performing an experiments similar to [14, 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 [23, Table 3.2]. In order to work with precision higher than double, we rely on the overloaded methods



(a) Forward error.



(b) Performance profile for (a).

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

of the Advanpix Multiprecision Computing Toolbox [27] (version 4.4.7.12739).

In Figure 4.1a, we compare the 1-norm forward error of the three algorithms on the same test as in [14, 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 the $\kappa_{r^{-1}}(A)u$, suggests that the three implementations behave in a forward stable fashion. Note that the algorithms attain a remarkably similar accuracy on 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 [10]. 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 conduct a second experiment. For each pair of distinct implementations $\mathcal{A}_1$ and $\mathcal{A}_2$, we tried to find the 5×5 matrix A and, for different values of m, the $[m/m]$ rational function r that maximize the ratios $e_{\mathcal{A}_1}/e_{\mathcal{A}_2}$ and $e_{\mathcal{A}_2}/e_{\mathcal{A}_1}$ when the two algorithms $\mathcal{A}_1$ and $\mathcal{A}_2$ are used to solve (1.2). As optimization method, we used the multidirectional search method of Dennis and Torczon [9], implemented in the `mdsmax` function of the Matrix Computation Toolbox [17].

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,

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 numerator and denominator of a $[m/m]$ rational function r that maximize $e_{\mathcal{A}_1}/e_{\mathcal{A}_2}$. 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).

$\mathcal{A}_1$ \ $\mathcal{A}_2$	`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	–	–

and that their good stability properties make them reliable enough to be of use in practice.

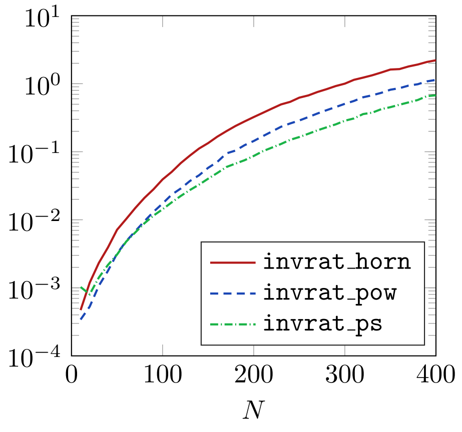
4.2. Computational time. Figure 4.2 shows how the execution time required by `invrat_horn`, `invrat_pow`, and `invrat_ps` to solve the matrix equation $r(X) = A$ depends on the size of the matrix A and on the order of the rational function r.

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 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`.

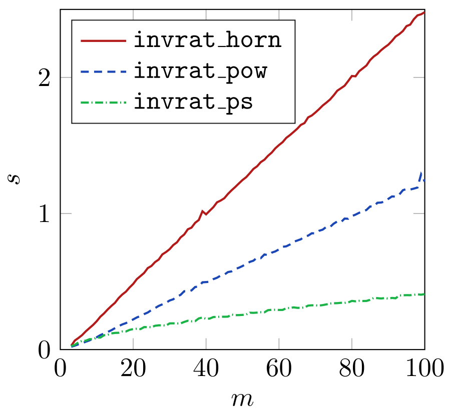
In Figure 4.2a, we fix the order of the rational function to 25, and let the size of the matrix increase from 10 to 400. As predicted by the analysis of the computational cost, the time required by the three algorithms rises more than linearly. For matrices of any size, `invrat_horn` is the slowest algorithm,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

In Figure 4.2b the algorithms are run on matrices of size 250 using rational functions of order 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 this section we compare the performance of `lambertwm` and `lambertwm_rat` on the test set used in [13, 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 [13, Exp. 2], which diagonalizes the matrix in higher precision by using the `eig` function provided by the



(a) $m = 25$.



(b) $N = 250$.

Figure 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).

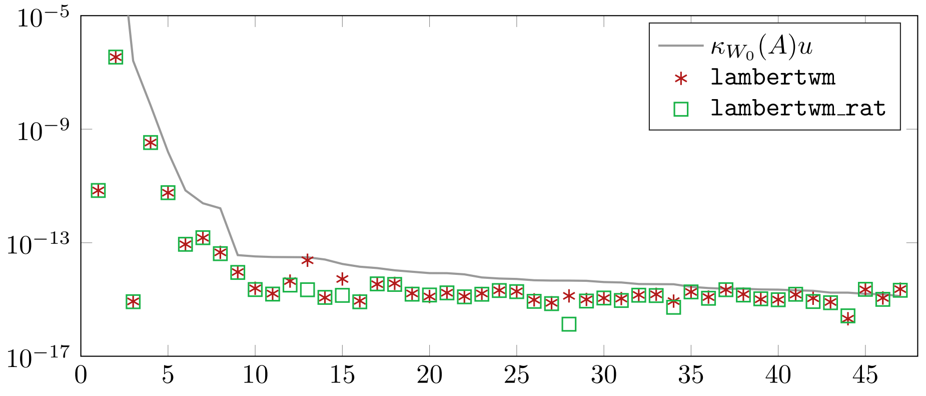
Symbolic Math Toolbox [31].

Figure 4.3b 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 on most matrices in the data set. The algorithm `lambertwm_rat` achieves slightly higher accuracy on about a quarter of the matrices in the data set, which justifies the more favorable curves of this algorithm in the performance profile.

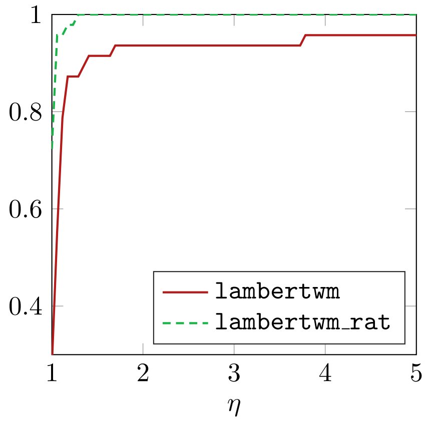
In Figure 4.3c, we compare the leading terms of the computational cost of the two implementations on the matrices in the test set. In both cases, $28N^3$ flops are required to compute the Schur decomposition and recovering the result at the end of the computation. The additional cost of `lambertwm_rat` 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 on 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 [3]. 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 the new algorithm is always much cheaper than the state-of-the-art algorithm for computing the Lambert W function, as `lambertwm_rat` is up to eleven times faster than `lambertwm`, and the speed up reaches a factor seven on more than half of the matrices in the test set.

5. Conclusions.

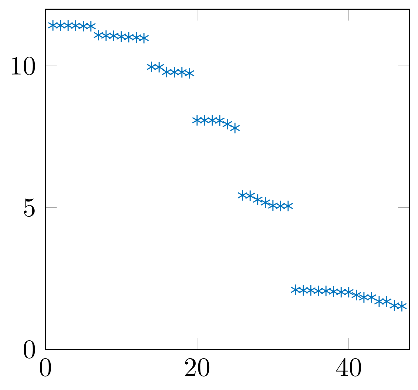
Algorithms for the solution of rational matrix equations can be a powerful tool in the evaluation of matrix functions defined implicitly as solutions to functional matrix equations. We developed two new such algorithms based on



(a) Forward error.



(b) Performance profile for (a).



(c) Cost ratio.

Figure 4.3: Top: forward error of `lambertwm` and `lambertwm_rat` on 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`.

schemes for the evaluation of rational matrix functions, and showed, both theoretically and experimentally, that they are more efficient than existing alternatives without sacrificing any accuracy. We characterized the applicability of substitution algorithms for rational matrix equations in a way that feels more natural than that previously attempted in the literature [14].

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 discussed how this strategy can be exploited to develop an algorithm for computing the Lambert W function that is more accurate and efficient that the state-of-the-art algorithm [13], in a number of test problems. 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, while the rational approximants to $X \exp X$ do not present any particular structure that can be exploited to reduce the computational cost of our substitution algorithms, 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] A. H. Al-Mohy and N. J. Higham, *Improved inverse scaling and squaring algorithms for the matrix logarithm*, SIAM J. Sci. Comput., 34 (2012), pp. C153–C169, https://doi.org/10.1137/110852553.

[2] F. M. Asl and A. G. Ulsoy, *Analysis of a system of linear delay differential equations*, Transactions-American Society of Mechanical Engineers Journal of Dynamic Systems Measurement and Control, 125 (2003), pp. 215–223.

[3] R. H. Bartels and G. W. Stewart, *Algorithm 432: Solution of the matrix equation $AX + XB = C$*, Comm. ACM, 15 (1972), pp. 820–826, https://doi.org/10.1145/361573.361582.

[4] Å. Björck and S. Hammarling, *A Schur method for the square root of a matrix*, Linear Algebra Appl., 52/53 (1983), pp. 127–140, https://doi.org/10.1016/0024-3795(83)80010-X.

[5] M. Bladt and M. Sørensen, *Efficient estimation of transition rates between credit ratings from observations at discrete time points*, Quant. Finance, 9 (2009), p. 147–160, https://doi.org/10.1080/14697680802624948.

[6] 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, https://doi.org/10.1016/j.automatica.2015.01.016.

[7] 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), p. 905–921, https://doi.org/10.1002/sim.2970.

[8] R. M. Corless, G. H. Gonnet, D. E. G. Hare, D. J. Jeffrey, and D. E. Knuth, *On the Lambert W function*, Adv. in Comput. Math., 5 (1996), pp. 329–359, https://doi.org/10.1007/BF02124750.

[9] J. E. Dennis, Jr. and V. Torczon, *Direct search methods on parallel machines*, SIAM J. Optim., 1 (1991), pp. 448–474, https://doi.org/10.1137/0801027.

[10] E. D. Dolan and J. J. Moré, *Benchmarking optimization software with performance profiles*, Math. Programming, 91 (2002), pp. 201–213, https://doi.org/10.1007/s101070100263.

[11] J.-C. Evard and F. Uhlig, *On the matrix equation $f(X) = A$*, Linear Algebra Appl., 162-164 (1992), pp. 447–519, https://doi.org/10.1016/0024-3795(92)90390-V.

[12] M. Fasi, *Optimality of the Paterson–Stockmeyer method for evaluating matrix polynomials and rational matrix functions*, Linear Algebra Appl., 574 (2019), p. 182–200, https://doi.org/10.1016/j.laa.2019.04.001.

[13] 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, https://doi.org/10.1137/140997610.

[14] M. Fasi and B. Iannazzo, *Computing primary solutions of equations involving primary matrix functions*, Linear Algebra Appl., 560 (2019), p. 17–42, https://doi.org/10.1016/j.laa.2018.09.010.

[15] F. Greco and B. Iannazzo, *A binary powering Schur algorithm for computing primary matrix roots*, Numer. Algorithms, 55 (2010), pp. 59–78, https://doi.org/10.1007/s11075-009-9357-1.

[16] J. M. Hefernan and R. M. Corless, *Solving some delay differential equations with computer algebra*, Math. Sci., 31 (2006), pp. 21–34.

[17] N. J. Higham, *The Matrix Computation Toolbox*. http://www.maths.manchester.ac.uk/~higham/mctoolbox.

[18] N. J. Higham, *The Matrix Function Toolbox*. http://www.maths.manchester.ac.uk/~higham/mftoolbox.

[19] N. J. Higham, *Computing real square roots of a real matrix*, Linear Algebra Appl., 88/89 (1987), pp. 405–430, https://doi.org/10.1016/0024-3795(87)90118-2.

[20] N. J. Higham, *Functions of Matrices: Theory and Computation*, Society for Industrial and Applied Mathematics, Philadelphia, PA, USA, 2008, https://doi.org/10.1137/1.9780898717778.

[21] 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, https://doi.org/10.1137/130906118.

[22] B. Iannazzo and C. Manasse, *A Schur logarithmic algorithm for fractional powers of matrices*, SIAM J. Matrix Anal. Appl., 34 (2013), pp. 794–813, https://doi.org/10.1137/120877398.

[23] *IEEE Standard for Floating-Point Arithmetic, IEEE Std 754-2008 (revision of IEEE Std 754-1985)*, Institute of Electrical and Electronics Engineers, 2008, https://doi.org/10.1109/IEEESTD.2008.4610935.

[24] 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), p. 245–265, https://doi.org/10.1111/1467-9965.00114.

[25] E. Jarlebring and T. Damm, *The Lambert W function and the spectrum of some multidimensional time-delay systems*, Automatica, 43 (2007), pp. 2124–2128, https://doi.org/10.1016/j.automatica.2007.04.001.

[26] C. S. Kenney and A. J. Laub, *Condition estimates for matrix functions*, SIAM J. Matrix Anal. Appl., 10 (1989), pp. 191–209, https://doi.org/10.1137/0610014.

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

[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, https://doi.org/10.1137/0202007.

[29] B. Singer and S. Spilerman, *The representation of social processes by Markov models*, Am. J. Sociol., 82 (1976), pp. 1–54, http://www.jstor.org/stable/2777460.

[30] M. I. Smith, *A Schur algorithm for computing matrix pth roots*, SIAM J. Matrix Anal. Appl., 24 (2003), pp. 971–989, https://doi.org/10.1137/S0895479801392697.

[31] *Symbolic Math Toolbox*. The MathWorks, Inc., Natick, MA, USA. http://www.mathworks.co.uk/products/symbolic/.

[32] 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), p. 561–572, https://doi.org/10.1007/s10980-009-9433-x.