SIAM J. MATRIX ANAL. APPL.
Vol. 34, No. 2, pp. 794–813

A SCHUR LOGARITHMIC ALGORITHM FOR FRACTIONAL POWERS OF MATRICES*

BRUNO IANNAZZO† AND CARLO MANASSE†

Abstract. We describe a recurrence method for computing primary pth roots of a matrix A with a cost, in terms of elementary arithmetic operations and memory, which is logarithmic with respect to p. When A is real and the primary root is real as well, the algorithm is based on the real Schur form of A and uses real arithmetic. The numerical experiments confirm the good behavior of the new algorithm in finite arithmetic. The case of arbitrary fractional powers of A is also considered.

Key words. matrix pth root, primary matrix function, Schur recurrence method, binary powering technique, fractional power of a matrix

AMS subject classifications. 65F30, 15A15

DOI. 10.1137/120877398

1. Introduction.

A pth root function in the set $\Omega \subset \mathbb{C}$ is a function $\varphi : \Omega \to \mathbb{C}$ such that $\varphi(z)^p = z$. If Ω is the union of t connected components none of which contain either 0 or a closed path around 0, then there exist exactly p^t analytic pth root functions in Ω.

Let $A \in \mathbb{C}^{n\times n}$ be a nonsingular matrix whose spectrum is $\Omega = \{\lambda_1, \dots, \lambda_t\}$. There are exactly p^t pth root functions in Ω determined by the scalar pth roots of any of the eigenvalues. Any such pth root function $\varphi(z)$ can be uniquely extended to a function analytic in an open neighborhood $\mathcal{U}$ of Ω and we can define the matrix function

$$\varphi(A) := \frac{1}{2\pi \mathbf{i}} \oint_\gamma \varphi(z)(zI - A)^{-1} dz,$$

where γ is a closed contour contained in $\mathcal{U}$ and surrounding the spectrum of A, while I is the identity matrix.

Any such function $\varphi(A)$ is said to be a primary matrix pth root of A and is a solution of the equation $X^p = A$. The adjective primary is necessary since not every solution of $X^p = A$ arises as a function of A in the sense given above (consider the equation $X^2 = I$), and one likes to call the pth root of A any solution of $X^p = A$. A solution of the equation $X^p = A$ which is not primary is called the nonprimary matrix pth root. We will discuss how to efficiently compute primary matrix roots, letting alone nonprimary matrix roots, which are not suited for numerical computation.

If $\Omega = \mathbb{C}\backslash(-\infty, 0]$, then one of the pth root functions deserves particular attention, that is, the principal pth root function $\varphi(z) = z^{1/p}$, which for any complex number z chooses the unique pth root lying in the sector $\mathcal{S}_p = \{z \in \mathbb{C} \setminus \{0\} : |\arg(z)| < \pi/p\}$. If the matrix A has no nonpositive real eigenvalues, then we can define the principal pth root $A^{1/p}$ of A, which in turn has eigenvalues in $\mathcal{S}_p$. The principal pth root is the one usually required in the applications.

*Received by the editors May 15, 2012; accepted for publication (in revised form) by Chun-hua Guo May 7, 2013; published electronically June 20, 2013.
http://www.siam.org/journals/simax/34-2/87739.html

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

When $A \in \mathbb{R}^{n\times n}$ is nonsingular, with spectrum $\Omega = \{\lambda_1, \ldots, \lambda_t\}$, some among the primary roots are real, precisely those corresponding to the functions $\varphi(z)$ such that $\varphi(\overline{z}) = \overline{\varphi(z)}$ for any $z \in \Omega$ [8]. In other words, the condition for a primary root of a real matrix to be real is that the root of the real eigenvalues is real and that the same determination is chosen for any couple of complex conjugate eigenvalues. We call these functions *real primary pth root functions*. The principal pth root of a real matrix, if it exists, is real [6, 7].

To compute the primary roots of a normal matrix, it is sufficient to compute the eigendecomposition of A, say, $A = QDQ^*$, where Q is unitary and D is diagonal, and get $\varphi(A) = Q\varphi(D)Q^*$. However, this approach may give poor numerical results in the nonnormal case, due to the possible ill-conditioning of the eigenvectors. Nevertheless, there are certain relevant applications, where the matrix A is not necessarily normal (see [8] and the references therein); moreover, one would like to have a general purpose algorithm for the problem.

The existing efficient algorithms for computing primary roots of a generic matrix can be divided into two classes: Schur recurrence algorithms, which get the roots by a direct formula once the Schur form of A is computed [13, 2], and rational iteration/approximation algorithms, whose core is a rational iteration converging to the matrix roots [10, 4, 11, 3, 12] or one or more than one rational approximation such as the Padé approximation [9].

A natural request from a numerical method is that its asymptotic computational cost has a dependence on p less than or equal to $\log_2 p$, in terms of both memory and number of arithmetic operations (ops) necessary to perform the computation. This is because, for instance, computing a 2^kth root of A should not cost much more than computing k times a square root. Another desire is that a numerical method working on real data performs the computation using only real arithmetic.

Rational iterations fulfill both conditions, while Padé algorithms use sometimes complex arithmetics, but their cost does not depend on p. Concerning existing Schur recurrence algorithms, their curse is the dependence on p: for the original method of Smith [13] the dependence is quadratic, while for the update given by Greco and Iannazzo [2] the dependence on p is linear.

The contribution of the paper is a new algorithm which updates the existing Schur recurrence algorithms reducing the computational cost to be logarithmic in p leaving unchanged its excellent numerical stability features. Moreover, the overall cost, for moderate values of p, makes the algorithm cheaper than most existing algorithms.

Besides the computation of primary matrix roots, we consider also the computation of fractional powers of a matrix, that is, compute $A^{q/p}$, for p, q positive integers with $q < p$. This is a special issue of the more general problem of computing A^α for $\alpha \in (0, 1)$. (See [9] for an introduction to the numerical issues related to A^α.)

Let $z = \rho e^{\mathbf{i}\theta} \in \mathbb{C} \setminus \{0\}$ and $\alpha \in (0, 1)$; for any integer k, the function $\varphi_k(z) := \rho^\alpha e^{\mathbf{i}\alpha(\theta+2k\pi)}$ with $\rho^\alpha = e^{\alpha \log \rho}$ is a determination of the function z to the power α. Two determinations $\varphi_k(z)$ and $\varphi_h(z)$ coincide, namely, $\rho^\alpha e^{\mathbf{i}\alpha(\theta+2k\pi)} = \rho^\alpha e^{\mathbf{i}\alpha(\theta+2h\pi)}$, for any ρ and θ, if and only if $\alpha(k-h)$ is an integer. Thus, if $\alpha = \frac{q}{p}$ with p and q relatively prime positive integers with $q \leqslant p$, then there are just p different determinations corresponding to the pth roots of z raised to the power q, for instance, $\varphi_0(z), \ldots, \varphi_{p-1}(z)$. If α is irrational, then there are infinitely many determinations.

Let $A \in \mathbb{C}^{n\times n}$ be a nonsingular matrix with spectrum $\Omega = \{\lambda_1, \ldots, \lambda_t\}$ and α be as above; then one can define the primary powers to α of A as $\varphi(A)$, where $\varphi(z)$ is a function which for each distinct eigenvalue λ_i of A coincides with a certain $\varphi_{k_i}(\lambda_i)$. If

α is rational, namely, $\alpha = \frac{q}{p}$, with p and q relatively prime, then there are t^p primary powers of A, corresponding to any choice of $\varphi_k(z)$, while if α is irrational, then there are infinitely many primary powers.

Let p, q be positive integers and $A \in \mathbb{C}^{n\times n}$ be nonsingular. The primary powers of A to $\frac{q}{p}$ are solutions of the matrix equation $X^p = A^q$; in particular, they are the primary pth roots of the matrix A^q. The primary powers of A to $\frac{q}{p}$ can be characterized also as the primary pth roots of the matrix A, raised to q. Notice that there may exist solutions of the equation $X^p = A^q$ which are nonprimary functions of A. With the notation introduced above, we define the principal $\frac{q}{p}$th power of a matrix A with no nonpositive real eigenvalues as $A^{q/p} = (A^{1/p})^q$.

We will describe how the proposed Schur algorithm for primary matrix roots can be used for computing primary fractional powers of a matrix.

The paper is organized as follows. In section 2 we discuss the Schur methods reviewing the existing methods, namely, the method of Smith and the method of Greco and Iannazzo, reviewed in sections 2.1 and 2.2, respectively. Then we present the new algorithm in section 2.3 and we discuss the case of arbitrary powers of a matrix in section 2.4. Some numerical tests are presented in section 3 and the conclusions are drawn in the final section.

In the following, to simplify the statements and the proofs, we use the convention that the sum $\sum_{i=k}^{h} a_i$ for $k > h$ is 0 and the product $\prod_{i=k}^{h} a_i$ for $k > h$ is 1. For $x \in \mathbb{R}$, we denote by $\lfloor x \rfloor$ the largest integer less than or equal to x.

Given two matrices M and N we denote by $M \otimes N$ their Kronecker (tensor) product and by $\operatorname{vec}(M)$ the vector obtained by stacking the columns of M. The link between the Kronecker product and the vec operator is through the formula $\operatorname{vec}(AXB) = (B^T \otimes A)\operatorname{vec}(X)$, which yields the product of matrices as a matrix-vector product; in this case we say that the product is written in Kronecker notation.

In the rest of the paper we use some properties of matrix functions of the type $f(A)$, namely, those obtained extending a scalar function $f(z)$ to a square matrix A. (See [7] for a treatise on the topic.) First, we use the fact that $f(A)$ is a polynomial of A. Second, if T is block upper triangular, then so is $f(T)$, and the diagonal blocks of $f(T)$ are $f(T_{11}), \dots, f(T_{\sigma\sigma})$, where $T_{11}, \dots, T_{\sigma\sigma}$ are the diagonal blocks of T. Finally, for any invertible matrix M, it holds that $f(MAM^{-1}) = Mf(A)M^{-1}$; we call this property the *similarity invariance* of matrix functions. The proof of these properties can be found in [7].

2. Recurrence methods for primary matrix roots. Let $A \in \mathbb{C}^{n\times n}$ be a nonsingular matrix whose distinct eigenvalues are $\lambda_1, \dots, \lambda_t$, and let $\varphi(z)$ be a primary pth root function defined on the spectrum of A. We describe and update the class of recurrence methods for computing $\varphi(A)$.

Recurrence methods are based on the fact that any primary root $X = \varphi(A)$ of A is a polynomial of A. Thus, if Q is a unitary matrix such that $T = Q^*AQ$ is block upper triangular, then $Y = Q^*XQ = Q^*\varphi(A)Q = \varphi(Q^*AQ) = \varphi(T)$ is a primary root of T with the same block upper triangular structure as T and the same eigenvalues as X. The problem of computing $\varphi(A)$ is thus reduced to the computation of $\varphi(T)$, which is the unique solution of the equation $Y^p = T$ whose eigenvalues are $\varphi(\lambda_1), \dots, \varphi(\lambda_t)$. The equation $Y^p = T$ is solved by a suitable recurrence.

The algorithms can be given for a block upper triangular matrix T with no restriction on the block sizes, but two cases are of major interest:

- T is upper triangular, obtained using the (complex) Schur normal form of A, say, $T = Q^*AQ$, with Q unitary;

- T is upper quasi-triangular, namely, is a real and block upper triangular matrix with diagonal blocks of size at most 2, obtained using the real Schur normal form of A, when the latter has real entries, say, $T = Q^T AQ$, with Q orthogonal; moreover, the required pth root function $\varphi(T)$ is real.

When the Schur form is used the method is called the Schur recurrence method or just the Schur method.

In sections 2.1, 2.2, and 2.3 we will describe the main ideas of recurrence methods in the real case for an upper quasi-triangular matrix T. In section 2.3.2 we will discuss the generalizations to the complex and upper triangular case, together with some specific instances.

The idea of computing a matrix function using the Schur form has been fruitfully applied to the matrix square root by Björck and Hammarling [1] and specialized to the real case by Higham [6]. The case of roots with indices greater than 2 was sketched in the complex case by Björck and Hammarling [1], given in full detail by Smith [13], and later updated by Greco and Iannazzo [2].

2.1. Smith's method. Let $T \in \mathbb{R}^{n\times n}$ be a $\sigma\times\sigma$ upper quasi-triangular matrix, whose blocks are denoted by T_{ij}, for $i, j = 1, \dots, \sigma$, and let $\varphi(z)$ be a real pth root function so that $\varphi(T)$ is real.

The method of Smith is based on the construction of the set $\{R^{(0)}, \dots, R^{(p-1)}\}$ of matrices with the same block structure as T, such that $R^{(k)} = Y^{k+1}$, for $k = 0, \dots, p-1$, so that $R^{(0)} = Y$ and $R^{(p-1)} = T$. Using the recurrence $R^{(k+1)} = YR^{(k)}$ for $k = 0, \dots, p-2$, Smith deduces for $i < j$ the formula

$$T_{ij} = \sum_{\ell=0}^{p-1} Y_{ii}^{p-\ell-1} Y_{ij} Y_{jj}^{\ell} + \sum_{\ell=0}^{p-2} Y_{ii}^{p-\ell-2} \widehat{B}_{ij}^{(\ell)}, \quad \widehat{B}_{ij}^{(k)} = \sum_{\ell=i+1}^{j-1} Y_{i\ell} R_{\ell j}^{(k)}, \tag{2.1}$$

where $R_{ij}^{(k)}$ is the block of $R^{(k)}$ with indices (i, j).

The blocks appearing in formula (2.1) besides Y_{ij} are blocks of the matrices $R^{(k)}$ with indices (i, ℓ) for $i \leqslant \ell < j$ or (ℓ, j) for $i < \ell \leqslant j$; in other words they are on the left of and below the position (i, j) in the matrices $R^{(k)}$. Assuming that all blocks $R_{i\ell}^{(k)}$ and $R_{\ell j}^{(k)}$, $\ell = i+1, \dots, j-1$, are known, formula (2.1) becomes a linear matrix equation from which we derive Y_{ij}.

This suggests the following strategy of computation: compute the blocks of Y a block column at a time, from the diagonal block up to the first block row. For example, for a matrix with three diagonal blocks the flow of computation is $Y_{11} \to Y_{22} \to Y_{12} \to Y_{33} \to Y_{23} \to Y_{13}$. More precisely, for each block column j

- compute Y_{jj} as the desired primary root of T_{jj} (using, for instance, a direct formula) and then $R_{jj}^{(k)} = Y_{jj} R_{jj}^{(k-1)}$ for $k = 1, \dots, p-2$;
- for each row $i < j$, compute Y_{ij} using (2.1) and compute $R_{ij}^{(k)}$ for $k \geqslant 1$ using $R_{ij}^{(k)} = \sum_{\ell=0}^{k} Y_{ii}^{k-\ell} Y_{ij} Y_{jj}^{\ell} + \sum_{\ell=0}^{k-1} Y_{ii}^{k-1-\ell} \widehat{B}_{ij}^{(\ell)}$.

The strategy used in the paper by Smith is to compute a block superdiagonal at a time. It is equivalent to the one described here.

First, the diagonal blocks of Y must be computed. Recall that $Y_{jj}^p = T_{jj}$ for $j = 1, \dots, \sigma$, and then Y_{jj} is a pth root of T_{jj} and precisely $\varphi(T_{jj})$. If Y_{jj} has size one, then it is enough to apply $\varphi(z)$ to the scalar T_{jj}; if Y_{jj} has size two and eigenvalues $\theta \pm \mathbf{i}\mu$, then one can use the direct formula

$$Y_{jj} = \varphi(T_{jj}) = \alpha I + \frac{\beta}{\mu}(T_{jj} - \theta I), \tag{2.2}$$

where $\alpha + \mathbf{i}\beta = \varphi(\theta + \mathbf{i}\mu)$. The blocks $R_{jj}^{(k)}$ are computed using the recurrence $R_{jj}^{(k)} = Y_{jj}R_{jj}^{(k-1)}$ for $k = 1, \dots, p-2$.

Second, for $i < j$, the matrix equation (2.1) is solved for Y_{ij}. Using Kronecker notation, (2.1) can be transformed into a linear system

$$\left(\sum_{\ell=0}^{p-1}(Y_{jj}^T)^\ell \otimes Y_{ii}^{p-\ell-1}\right)\operatorname{vec}(Y_{ij}) = \operatorname{vec}\big(T_{ij} - \sum_{\ell=0}^{p-2} Y_{ii}^{p-2-\ell}\widehat{B}_{ij}^{(\ell)}\big), \tag{2.3}$$

whose matrix coefficient has size at most 4. It can be proved that the linear system in (2.3) has a unique solution for each $i < j$ (see [13] for a proof), so that we can determine the matrix Y_{ij}.

If the principal pth root of T is required, then it is sufficient to choose the principal pth root when the root of a diagonal block is computed, that is, to choose $\varphi(T_{jj}) = T_{jj}^{1/p}$ for each j.

The computational cost of the algorithm is quadratic in p; in fact the most costly part, with respect to p, is the computation of $R_{ij}^{(k)}$, which, for each i, j, requires about $\sum_{k=1}^{p-2} 3k = O(p^2)$ multiplications of matrices of size at most 2. The total cost of Smith's algorithm is of $O(n^3p + n^2p^2)$ ops, and some considerations on the backward error suggest that it is backward stable [13].

With a minor modification, that is, computing $R_{ij}^{(k)}$ using the formula $R_{ij}^{(k+1)} = Y_{ii}R_{ij}^{(k)} + Y_{ij}R_{jj}^{(k)} + \widehat{B}_{ij}^{(k)}$, the cost of Smith's algorithm is easily lowered to $O(n^3p)$ ops. However, the analysis of Smith would say nothing about the numerical stability of this variant.

2.2. The method of Greco and Iannazzo. Greco and Iannazzo proposed in [2] a modification of the Smith algorithm, observing that in order to get Y from T, it is not necessary to consider all powers Y^k for $k = 1, \dots, p$ but just a part of them using the binary powering technique. As in section 2.1 we assume that T is upper quasi-triangular and that the required pth root function is real.

Let p have $t = \lfloor \log_2 p \rfloor + 1$ digits in its binary expansion; then we can write

$$p = b_1 2^{t-1} + b_2 2^{t-2} + \cdots + b_{t-1}2 + b_t, \quad b_1 = 1, \tag{2.4}$$

with $b_1 = 1$ and $b_i \in \{0, 1\}$ for $i > 1$, and

$$p = 2^{c_1} + 2^{c_2} + \cdots + 2^{c_m}, \qquad c_1 > c_2 > \cdots > c_m \geqslant 0, \tag{2.5}$$

where m is the number of nonzero binary digits and the set $\{c_1, \dots, c_m\}$ contains the positions of these nonzero digits in the binary expansion of p. Note that $c_1 = t - 1 = \lfloor \log_2 p \rfloor$.

For instance, with $p = 23$ we have $b_1 = 1$, $b_2 = 0$, $b_3 = 1$, $b_4 = 1$, $b_5 = 1$, while $c_1 = 4$, $c_2 = 2$, $c_3 = 1$, $c_4 = 0$. In order to get $Y^{23} = T$, the algorithm of Smith uses all the matrices Y^k for $k = 1, \dots, 22$, and the idea of Greco and Iannazzo is to consider a smaller number of powers of k, in fact, $Y^{23} = Y^{16}Y^4Y^2Y$, and thus in order to get Y^{23}, it is sufficient to form the six matrices Y^2, Y^4, Y^8, Y^{16}, Y^{20}, Y^{22}.

In the algorithm of Greco and Iannazzo the recursion is constructed through the sequences

$$\begin{cases} V^{(1)} = I, \quad V^{(2)} = Y, \\ V^{(k)} = V^{(k-1)}V^{(k-1)} = Y^{2^{k-2}}, \quad k = 3, \dots, t+1, \end{cases} \tag{2.6}$$

and

$$\begin{cases} W^{(1)} = I, \\ W^{(k)} = W^{(k-1)} V^{(c_{k-1}+2)} = Y^{2^{c_1}+2^{c_2}+\cdots+2^{c_{k-1}}}, \quad k = 2, \ldots, m+1. \end{cases} \tag{2.7}$$

It holds that $W^{(2)} = V^{(c_1+2)} = V^{(t+1)} = Y^{2^{c_1}}$ and $W^{(m+1)} = T$. (Notice that the relations (2.6) and (2.7) are slightly different from the ones appearing in the paper of Greco and Iannazzo [2]; since the indices have been shifted by one, this choice makes the implementation easier.)

Since T is upper quasi-triangular, all matrices $V^{(k)}$ and $W^{(k)}$ have the same upper quasi-triangular structure as T. As in section 2.1, the diagonal blocks of Y are computed directly, and (2.6) and (2.7) give the following recursions for the blocks with indices (i, j) such that $1 \leqslant i < j \leqslant \sigma$:

$$V_{ij}^{(k+1)} = \sum_{\xi=i}^{j} V_{i\xi}^{(k)} V_{\xi j}^{(k)} = V_{ii}^{(k)} V_{ij}^{(k)} + V_{ij}^{(k)} V_{jj}^{(k)} + B_{ij}^{(k)}, \quad k = 2, \ldots, t, \tag{2.8}$$

$$W_{ij}^{(k+1)} = \sum_{\xi=i}^{j} W_{i\xi}^{(k)} V_{\xi j}^{(c_k+2)} = W_{ii}^{(k)} V_{ij}^{(c_k+2)} + W_{ij}^{(k)} V_{jj}^{(c_k+2)} + C_{ij}^{(k)}, \quad k = 2, \ldots, m, \tag{2.9}$$

where

$$B_{ij}^{(k)} = \sum_{\xi=i+1}^{j-1} V_{i\xi}^{(k)} V_{\xi j}^{(k)}, \quad C_{ij}^{(k)} = \sum_{\xi=i+1}^{j-1} W_{i\xi}^{(k)} V_{\xi j}^{(c_k+2)}, \tag{2.10}$$

and the expression of $B_{ij}^{(k)}$ involves just blocks of $V^{(k)}$ which lie on the left of and below the block $V_{ij}^{(k)}$, while the expression of $C_{ij}^{(k)}$ involves just blocks of $W^{(k)}$ and $V^{(c_k+2)}$ which lie on the left of and below the block $W_{ij}^{(k)}$. We recall that all void sums should be understood as zero.

Putting together relations (2.8) and (2.9), Greco and Iannazzo deduce an algorithm for computing primary matrix roots whose asymptotic computational cost is $O(n^3 \log_2 p + n^2 p)$ ops, which is smaller than the asymptotic cost of the algorithm of Smith. However, the linear dependence on p, even if multiplied by n^2, is bothering.

In the next section we explain how to use (2.8) and (2.9) in order to obtain an algorithm for computing matrix roots whose computational cost is of $O(n^3 \log_2 p)$ ops.

2.3. The new algorithm. Following the idea of Smith and the construction of Greco and Iannazzo, we derive for each $1 \leqslant i < j \leqslant \sigma$ formula (2.12), which relates the block Y_{ij} to T_{ij} and to the blocks of the matrices $V^{(k)}$ and $W^{(k)}$ lying on the left of and below the block on position (i, j) together with the diagonal blocks of Y^k for $k = 1, \ldots, p$. In this way, the formula can be used as an equation in the unknown Y_{ij}. The novelty with respect to the formulae given by Smith and Greco and Iannazzo is that it can be evaluated with just $O(\log_2 p)$ ops, leading to an algorithm which computes any primary matrix pth root of a $n \times n$ matrix with $O(n^3 \log_2 p)$ ops. As in section 2.1 we assume that T is upper quasi-triangular and that the required pth root function is real.

The diagonal blocks of Y and of the matrices $V^{(k)}$ and $W^{(k)}$ can be obtained directly (using the argument of sections 2.1 and 2.2), so we focus on the off-diagonal blocks.

Consider first the case $p = 2^s$ and the sequence (2.6) of matrices $V^{(k)}$, where $V^{(1)} = I$, $V^{(2)} = Y$ is the sought pth root and $V^{(s+2)} = T$ is the given matrix. The recurrence $V^{(k)} = V^{(k-1)}V^{(k-1)}$ can be written for the blocks obtaining the recurrence (2.8) for $V_{ij}^{(k)}$ with $B_{ij}^{(k)}$ defined in (2.10).

We consider the block $T_{ij} = V_{ij}^{(s+2)}$. Using (2.8) with $k = s + 1$, one obtains an expression for T_{ij} involving $V_{ij}^{(s+1)}$ and other blocks on the left of and below the position (i, j). Then, using (2.8) again, but with $k = s$, that is, for $V_{ij}^{(s+1)}$, one obtains an expression involving $V_{ij}^{(s)}$. The substitution can be iterated until the only matrices of the sequence $V_{ij}^{(k)}$ are $Y_{ij} = V_{ij}^{(2)}$ and $T_{ij} = V_{ij}^{(s+2)}$. The precise formula is given in the following result.

THEOREM 2.1. *Let $p = 2^s \geqslant 1$, T be an upper quasi-triangular matrix with blocks T_{ij}, $1 \leqslant i, j \leqslant \sigma$. Let Y be a primary pth root of T and $V^{(k)} = Y^{2^{k-2}}$ for $k = 2, \dots, s + 2$; then for $i < j$,*

$$T_{ij} = \sum_{h=0}^{s} \sum_{\ell=0}^{2^{s-h}-1} Y_{ii}^{2^h \ell} B_{ij}^{(h+1)} Y_{jj}^{2^s - 2^h(\ell+1)}, \tag{2.11}$$

where $B_{ij}^{(1)} = Y_{ij}$ and $B_{ij}^{(h)} = \sum_{\xi=i+1}^{j-1} V_{i\xi}^{(h)} V_{\xi j}^{(h)}$ for $h = 2, \dots, s + 1$.

Proof. We prove the formula by induction on s. For $s = 0$, the formula reduces to $T_{ij} = B_{ij}^{(1)} = Y_{ij}$, which is true since $Y = T$. We assume that formula (2.11) is true for $p = 2^s$ and we prove that it is true for $p = 2^{s+1}$:

$$\begin{aligned}
T_{ij} &= V_{ij}^{(s+3)} = V_{ii}^{(s+2)} V_{ij}^{(s+2)} + V_{ij}^{(s+2)} V_{jj}^{(s+2)} + B_{ij}^{(s+2)} \\
&= \sum_{h=0}^{s} \sum_{\ell=0}^{2^{s-h}-1} (Y_{ii}^{2^s + 2^h \ell} B_{ij}^{(h+1)} Y_{jj}^{2^s - 2^h(\ell+1)} + Y_{ii}^{2^h \ell} B_{ij}^{(h+1)} Y_{jj}^{2^s + 2^s - 2^h(\ell+1)}) + B_{ij}^{(s+2)} \\
&= \sum_{h=0}^{s} \left(\sum_{\ell=2^{s-h}}^{2^{s-h+1}-1} Y_{ii}^{2^h \ell} B_{ij}^{(h+1)} Y_{jj}^{2^{s+1} - 2^h(\ell+1)} + \sum_{\ell=0}^{2^{s-h}-1} Y_{ii}^{2^h \ell} B_{ij}^{(h+1)} Y_{jj}^{2^{s+1} - 2^h(\ell+1)} \right) \\
&\quad + \sum_{\ell=0}^{2^{s-(s+1)+1}-1} Y_{ii}^{2^{s+1} \ell} B_{ij}^{(s+2)} Y_{jj}^{2^{s+1} - 2^{s+1}(\ell+1)} \\
&= \sum_{h=0}^{s+1} \sum_{\ell=0}^{2^{s-h+1}-1} Y_{ii}^{2^h \ell} B_{ij}^{(h+1)} Y_{jj}^{2^{s+1} - 2^h(\ell+1)},
\end{aligned}$$

and the proof is achieved. ∎

Now we turn to the general case. Let $p = 2^{c_1} + \cdots + 2^{c_m}$ as in section 2.2. In this case, one should consider both the recurrences $V^{(k)}$ for (2.6) and $W^{(k)}$ for (2.7), where $W^{(m+1)} = T$ is the given matrix. The recurrences for $V^{(k)}$ and $W^{(k)}$ can be written for the blocks obtaining the recurrences (2.8) and (2.9), respectively, where the matrices $B_{ij}^{(k)}$ and $C_{ij}^{(k)}$ are defined in (2.10).

As before, we consider the block $T_{ij} = W_{ij}^{(m+1)}$. Using (2.9) with $k = m$, one obtains an expression involving $V_{ij}^{(c_m+2)}$ and $W_{ij}^{(m)}$ and other blocks on the left and

below the position (i,j). Then, using (2.9) again, but with $k = m-1$, one obtains an expression involving $V_{ij}^{(c_m+2)}$, $V_{ij}^{(c_{m-1}+2)}$, and $W_{ij}^{(m-1)}$. The substitution can be iterated using (2.9) until the only block of the type $W_{ij}^{(k)}$ is $W_{ij}^{(2)}$, which is $V_{ij}^{(t+1)}$. Then, repeatedly using (2.8) one obtains a relation in which the only blocks in position (i,j) are Y_{ij} and T_{ij}.

The procedure seems to be very complicated, since a lot of terms arise during the substitutions, but fortunately they can be grouped in a very nice way, as in the following result.

THEOREM 2.2. *Let $p \geqslant 1$, with dyadic expansion $p = 2^{c_1} + 2^{c_2} + \cdots + 2^{c_m}$, where $c_1 > c_2 > \cdots > c_m \geqslant 0$, and T be an upper quasi-triangular matrix, let Y be a primary pth root of T, $V^{(k)} = Y^{2^{k-2}}$ for $k = 1, \ldots, s+2$, where $s = c_1 = \lfloor \log_2 p \rfloor$, and $W^{(k)} = Y^{2^{c_1}+2^{c_2}+\cdots+2^{c_{k-1}}}$ for $k = 2, \ldots, m+1$, so that $W^{(m+1)} = T$; then for $i < j$,*

$$T_{ij} = \sum_{h=0}^{\lfloor \log_2 p \rfloor} \sum_{\ell=0}^{\lfloor \frac{p}{2^h} \rfloor - 1} Y_{ii}^{2^h \ell} B_{ij}^{(h+1)} Y_{jj}^{p-2^h(\ell+1)} + \sum_{h=2}^{m} C_{ij}^{(h)} Y_{jj}^{p-2^{c_h} \lfloor \frac{p}{2^{c_h}} \rfloor}, \tag{2.12}$$

where $B_{ij}^{(1)} = Y_{ij}$, $B_{ij}^{(h)} = \sum_{\xi=i+1}^{j-1} V_{i\xi}^{(h)} V_{\xi j}^{(h)}$ for $h = 2, \ldots, s+1$ and $C_{ij}^{(h)} = \sum_{\xi=i+1}^{j-1} W_{i\xi}^{(h)} V_{\xi j}^{(c_h+2)}$ for $h = 2, \ldots, m$.

Proof. We prove the theorem by induction on m, the number of nonzero digits in the binary expansion of p. The case $m = 1$ is Theorem 2.1. We assume that formula (2.12) is true for $1 \leqslant k \leqslant m$ and we prove that it is true for $k = m+1$. Let $p = 2^{c_1} + \cdots + 2^{c_m} + 2^{c_{m+1}}$; then

$$T_{ij} = W_{ij}^{(m+2)} = W_{ii}^{(m+1)} V_{ij}^{(c_{m+1}+2)} + W_{ij}^{(m+1)} V_{jj}^{(c_{m+1}+2)} + C_{ij}^{(m+1)}. \tag{2.13}$$

The theorem is true for $p' = p - 2^{c_{m+1}} = 2^{c_1} + \cdots + 2^{c_m}$ and for $p'' = 2^{c_{m+1}}$. Observing that $Y_{jj}^{p-2^{c_h} \lfloor \frac{p}{2^{c_h}} \rfloor} = \prod_{\ell=h+1}^{m+1} Y_{jj}^{2^{c_\ell}}$, we have

(2.14)

$$W_{ij}^{(m+1)} = \sum_{h=0}^{c_1} \sum_{\ell=0}^{\lfloor \frac{p-2^{c_{m+1}}}{2^h} \rfloor - 1} Y_{ii}^{2^h \ell} B_{ij}^{(h+1)} Y_{jj}^{p-2^{c_{m+1}}-2^h(\ell+1)} + \sum_{h=2}^{m} C_{ij}^{(h)} \prod_{\ell=h+1}^{m} Y_{jj}^{2^{c_\ell}},$$

$$V_{ij}^{(c_{m+1}+2)} = \sum_{h=0}^{c_{m+1}} \sum_{\ell=0}^{2^{c_{m+1}-h}-1} Y_{ii}^{2^h \ell} B_{ij}^{(h+1)} Y_{jj}^{2^{c_{m+1}}-2^h(\ell+1)}.$$

Putting the formulae (2.14) into (2.13), we get

$$\begin{aligned} T_{ij} = & \sum_{h=0}^{c_{m+1}} \sum_{\ell=0}^{2^{c_{m+1}-h}-1} Y_{ii}^{p-2^{c_{m+1}}+2^h \ell} B_{ij}^{(h+1)} Y_{jj}^{2^{c_{m+1}}-2^h(\ell+1)} \\ & + \sum_{h=0}^{c_{m+1}} \sum_{\ell=0}^{\lfloor \frac{p}{2^h} \rfloor - 2^{c_{m+1}-h}-1} Y_{ii}^{2^h \ell} B_{ij}^{(h+1)} Y_{jj}^{p-2^h(\ell+1)} \\ & + \sum_{h=c_{m+1}+1}^{c_1} \sum_{\ell=0}^{\lfloor \frac{p-2^{c_{m+1}}}{2^h} \rfloor - 1} Y_{ii}^{2^h \ell} B_{ij}^{(h+1)} Y_{jj}^{p-2^h(\ell+1)} + \sum_{h=2}^{m+1} C_{ij}^{(h)} \prod_{\ell=h+1}^{m+1} Y_{jj}^{2^{c_\ell}}, \end{aligned}$$

where we have split the sum involving $W_{ij}^{(m+1)}$. A change of variable in the first summand of the right-hand side yields

$$\begin{aligned}
T_{ij} &= \sum_{h=0}^{c_{m+1}} \sum_{\ell=\lfloor \frac{p}{2^h}\rfloor - 2^{c_{m+1}-h}}^{\lfloor \frac{p}{2^h}\rfloor - 1} Y_{ii}^{p-2^h\lfloor \frac{p}{2^h}\rfloor + 2^h\ell} B_{ij}^{(h+1)} Y_{jj}^{2^h\lfloor \frac{p}{2^h}\rfloor - 2^h(\ell+1)} \\
&\quad + \sum_{h=0}^{c_{m+1}} \sum_{\ell=0}^{\lfloor \frac{p}{2^h}\rfloor - 2^{c_{m+1}-h}-1} Y_{ii}^{2^h\ell} B_{ij}^{(h+1)} Y_{jj}^{p-2^h(\ell+1)} \\
&\quad + \sum_{h=c_{m+1}+1}^{c_1} \sum_{\ell=0}^{\lfloor \frac{p}{2^h}\rfloor - 1} Y_{ii}^{2^h\ell} B_{ij}^{(h+1)} Y_{jj}^{p-2^h(\ell+1)} + \sum_{h=2}^{m+1} C_{ij}^{(h)} \prod_{\ell=h+1}^{m+1} Y_{jj}^{2^{c_\ell}} \\
&= \sum_{h=0}^{c_1} \sum_{\ell=0}^{\lfloor \frac{p}{2^h}\rfloor - 1} Y_{ii}^{2^h\ell} B_{ij}^{(h+1)} Y_{jj}^{p-2^h(\ell+1)} + \sum_{h=2}^{m+1} C_{ij}^{(h)} \prod_{\ell=h+1}^{m+1} Y_{jj}^{2^{c_\ell}},
\end{aligned}$$

which is the desired result. We have used the fact that $\lfloor \frac{p-2^{c_{m+1}}}{2^h}\rfloor = \lfloor \frac{p}{2^h}\rfloor$ for $h = c_{m+1}+1, \dots, c_1$ and $2^h\lfloor \frac{p}{2^h}\rfloor = p$ for $h = 0, \dots, c_{m+1}$. □

Equation (2.12) can be used to compute Y from T; in fact, isolating the terms containing Y_{ij} (which is $B_{ij}^{(1)}$) one has

$$\sum_{\ell=0}^{p-1} Y_{ii}^{\ell} Y_{ij} Y_{jj}^{p-\ell-1} = T_{ij} - \sum_{h=1}^{c_1} \sum_{\ell=0}^{\lfloor \frac{p}{2^h}\rfloor - 1} Y_{ii}^{2^h\ell} B_{ij}^{(h+1)} Y_{jj}^{p-2^h(\ell+1)} - \sum_{h=2}^{m} C_{ij}^{(h)} \prod_{\ell=h+1}^{m} Y_{jj}^{2^{c_\ell}},$$

which, using Kronecker notation, can be rewritten as the linear system

(2.15)
$$\begin{aligned}
\left(\sum_{\ell=0}^{p-1} (Y_{jj}^T)^{p-\ell-1} \otimes Y_{ii}^{\ell}\right) \operatorname{vec}(Y_{ij}) = \operatorname{vec}(T_{ij}) - \sum_{h=1}^{\lfloor \log_2 p\rfloor} \left(\sum_{\ell=0}^{\lfloor \frac{p}{2^h}\rfloor - 1} (Y_{jj}^T)^{p-2^h(\ell+1)} \otimes Y_{ii}^{2^h\ell}\right) \\
\times \operatorname{vec}(B_{ij}^{(h+1)}) - \operatorname{vec}\left(\sum_{h=2}^{m} C_{ij}^{(h)} \prod_{\ell=h+1}^{m} Y_{jj}^{2^{c_\ell}}\right),
\end{aligned}$$

where, besides Y_{ij} and T_{ij}, all quantities are blocks of the matrices $V^{(k)}$ and $W^{(h)}$ lying below or left of the position (i, j) or are diagonal blocks of Y^k for $k = 0, \dots, p$. Thus, the system (2.15) is useful for computing the matrix Y, a column at a time, from the diagonal up to the first line, as in the customary Schur recurrence method of Smith described in section 2.1.

The matrix coefficient is a sum of p terms; moreover, the right-hand side is a sum of about p terms. Nevertheless, the special form in which the linear system is given allows one to devise an algorithm which computes the coefficient of the linear system and the right-hand side with a number of arithmetic operations whose dependence on p is $\log_2 p$.

With this objective in mind, we rewrite the linear system (2.15) in a more compact form defining

$$M_{ij}^{(h)} := \sum_{\ell=0}^{\lfloor \frac{p}{2^h}\rfloor - 1} (Y_{jj}^T)^{p-2^h(\ell+1)} \otimes Y_{ii}^{2^h\ell}, \qquad U_{jj}^{(h)} = \prod_{\ell=h}^{m} Y_{jj}^{2^{c_\ell}}, \tag{2.16}$$

so that the system to be solved, for $i < j$, becomes

$$M_{ij}^{(0)} \operatorname{vec}(Y_{ij}) = \operatorname{vec}(T_{ij}) - \sum_{h=1}^{\lfloor \log_2 p \rfloor} M_{ij}^{(h)} \operatorname{vec}(B_{ij}^{(h+1)}) - \operatorname{vec}\Big(\sum_{h=2}^{m} C_{ij}^{(h)} U_{jj}^{(h+1)}\Big). \tag{2.17}$$

We provide the new algorithm, leaving a small debt which will be paid in section 2.3.1, which shows how to obtain together all the values $M_{ij}^{(h)}$ for $h = 0, \dots, \lfloor \log_2 p \rfloor$ with $O(\log_2 p)$ ops. The values t, m, and c_h are defined in (2.4) and (2.5).

ALGORITHM 1. Given an upper quasi-triangular matrix T with blocks T_{ij}, $1 \leqslant i, j \leqslant \sigma$, and a real primary pth root function $\varphi(z)$, compute $\varphi(T)$ using only real arithmetic:

1. for $j = 1, \dots, \sigma$,
2. set $V_{jj}^{(1)} = I$, compute $V_{jj}^{(2)} = Y_{jj} = \varphi(T_{jj})$, and compute $V_{jj}^{(h)} = Y_{jj}^{2^{h-2}}$, for $h = 3, \dots, t+1$, using $V_{jj}^{(h)} = (V_{jj}^{(h-1)})^2$;
3. set $W_{jj}^{(1)} = I$, $W_{jj}^{(2)} = V_{jj}^{(t+1)}$, and compute $W_{jj}^{(h)}$, for $h = 3, \dots, m$, using $W_{jj}^{(h)} = W_{jj}^{(h-1)} V_{jj}^{(c_{h-1}+2)}$, set $W_{jj}^{(m+1)} = T_{jj}$;
4. set $U_{jj}^{(m+1)} = I$, $U_{jj}^{(m)} = V_{jj}^{(c_m+2)}$, and compute $U_{jj}^{(h)}$, for $h = m-1, \dots, 2$, using $U_{jj}^{(h)} = U_{jj}^{(h+1)} V_{jj}^{(c_h+2)}$, set $U_{jj}^{(1)} = T_{jj}$;
5. for $i = j-1, j-2, \dots, 1$
6. compute $B_{ij}^{(h)}$, for $h = 2, \dots, t$, and $C_{ij}^{(h)}$, for $h = 2, \dots, m$, using (2.10);
7. compute $M_{ij}^{(h)}$ for $h = 0, \dots, t-1$, using Algorithm 3 of section 2.3.1;
8. compute the right-hand side of (2.17) and solve the linear system for Y_{ij};
9. compute $V_{ij}^{(h)}$ and $W_{ij}^{(h)}$ using (2.8) and (2.9).

To determine the computational cost of Algorithm 1, we analyze the cost of the single steps. We consider first the case in which the matrix T is real upper triangular.

Steps 2, 3, and 4 require about $t + 2m$ ops for each j; step 6 requires $2(j - i - 1)(t + m - 2)$ ops for each $i < j$; steps 7 and 8 require about $2(t + m)$ ops each for each $i < j$; finally, step 9 requires about $4(t + m)$ ops for each $i < j$. Since $\sum_{i<j}(j - i - 1) \approx \sum_{j=2}^{m}(j^2 - j^2/2) \approx n^3/6$, the principal part of the total cost as n and p tend to infinity is $\mathcal{C}(p, n) = \frac{1}{3}n^3(t + m - 2)$, and we have the bounds $\frac{1}{3}n^3 \lfloor \log_2 p \rfloor \leqslant \mathcal{C}(p, n) \leqslant \frac{2}{3}n^3 \log_2 p$.

When some of the diagonal blocks of T are 2×2 matrices, the number of blocks ranges from $\lceil \frac{n}{2} \rceil$ to $n - 1$. With respect to an upper triangular matrix T, a smaller number of block operations is performed, but on blocks of sizes at most 2×2, and the linear systems to be solved are of size at most 4. For this reason the computational cost of Algorithm 1 for a nontriangular matrix T of size n is a small multiple of the cost of the same algorithm for an upper triangular matrix T. In the extremal case where n is even and all diagonal blocks are 2×2, the indices j and i are summed from 1 to $n/2$, while the cost of a block operation in step 6 is of 12 ops. In summary, in the block case the asymptotic cost is no larger than about $n^3 \log_2 p$ ops.

2.3.1. Computation of $M_{ij}^{(h)}$. Given $i < j$, we present an algorithm for computing together all the quantities $M_{ij}^{(h)}$ of (2.16) for $h = 0, \dots, \lfloor \log_2 p \rfloor$ with a total cost of $O(\log_2 p)$ ops.

We consider first the scalar case, that is, compute

$$m_h := \sum_{\ell=0}^{\lfloor \frac{p}{2^h} \rfloor - 1} x^{p-2^h(\ell+1)} y^{2^h \ell}, \quad h = 0, \ldots, \lfloor \log_2 p \rfloor, \tag{2.18}$$

for $x, y \in \mathbb{C}$. The matrix case follows easily and will be discussed later. In order to design an efficient algorithm we need the following result.

THEOREM 2.3. *Let $x, y \in \mathbb{C}$ and $p = b_1 2^{t-1} + b_2 2^{t-2} + \cdots + b_{t-1} 2 + b_t$ be the binary expansion of $p \geqslant 1$ with $b_1 = 1$ and $b_i \in \{0, 1\}$ for $i > 1$, so that $t = \lfloor \log_2 p \rfloor + 1$. Define $\sigma_k = x^{2^{k-1}} + y^{2^{k-1}}$ for $k = 1, \ldots, t$ and $\widetilde{u}_k = x^{p-2^k \lfloor \frac{p}{2^k} \rfloor}$, $\widetilde{w}_k = y^{2^{k+1} \lfloor \frac{p}{2^{k+1}} \rfloor}$ for $k = 0, \ldots, t-1$. Then*

$$m_h = m_{h+1} \sigma_{h+1} + b_{t-h} \widetilde{u}_h \widetilde{w}_h, \quad h = 0, \ldots, t-2, \tag{2.19}$$

while $m_{t-1} = b_1 \widetilde{u}_{t-1} \widetilde{w}_{t-1} = \widetilde{u}_{t-1} = x^{p-2^{t-1}}$.

Proof. The formula for m_{t-1} can be verified directly. To prove the inductive step, observe that for any sequence $\{a_\ell\}_\ell$ and $0 \leqslant h \leqslant t-1$, we have

$$\sum_{\ell=0}^{\lfloor \frac{p}{2^h} \rfloor - 1} a_\ell = \sum_{\ell=0}^{\lfloor \frac{p}{2^{h+1}} \rfloor - 1} (a_{2\ell} + a_{2\ell+1}) + b_{t-h} a_{\lfloor \frac{p}{2^h} \rfloor - 1};$$

in fact $\lfloor \frac{p}{2^h} \rfloor = 2 \lfloor \frac{p}{2^{h+1}} \rfloor + b_{t-h}$. Using the aforementioned decomposition, for $h < t-1$ we have

$$\begin{aligned}
m_h &= \sum_{\ell=0}^{\lfloor \frac{p}{2^h} \rfloor - 1} x^{p-2^h(\ell+1)} y^{2^h \ell} \\
&= \sum_{\ell=0}^{\lfloor \frac{p}{2^{h+1}} \rfloor - 1} \left(x^{p-2^h(2\ell+1)} y^{2^h(2\ell)} + x^{p-2^h(2\ell+2)} y^{2^h(2\ell+1)} \right) \\
&\quad + b_{t-h} x^{p-2^h \lfloor \frac{p}{2^h} \rfloor} y^{2^h \lfloor \frac{p}{2^h} \rfloor - 2^h} \\
&= (x^{2^h} + y^{2^h}) \sum_{\ell=0}^{\lfloor \frac{p}{2^{h+1}} \rfloor - 1} x^{p-2^{h+1}(\ell+1)} y^{2^{h+1}\ell} + b_{t-h} \widetilde{u}_h \widetilde{w}_h = m_{h+1} \sigma_{h+1} + b_{t-h} \widetilde{u}_h \widetilde{w}_h,
\end{aligned}$$

and the proof is completed, where we have used $2^h \lfloor \frac{p}{2^h} \rfloor - 2^h b_{t-h} = 2^{h+1} \lfloor \frac{p}{2^{h+1}} \rfloor$. ∎

Theorem 2.3 allows one to compute all the values of m_h for $h = 0, \ldots, t-1$ with an algorithm which performs $O(\log_2 p)$ ops.

ALGORITHM 2. Given $x, y \in \mathbb{C}$ and $p \geqslant 1$, compute m_h as in (2.18) with $O(\log_2 p)$ ops:

1. compute $x_h := x^{2^{h-1}}$, $y_h := y^{2^{h-1}}$ and $\sigma_h := x^{2^{h-1}} + y^{2^{h-1}}$, using $x_1 = 1$, $y_1 = 1$, and $x_h = x_{h-1}^2$, $y_h = y_{h-1}^2$, for $h = 2, \ldots, t$, $\sigma_h = x_h + y_h$, for $h = 2, \ldots, t-1$;
2. set $\widetilde{u}_0 = 1$ and compute $\widetilde{u}_k = x^{p-2^k \lfloor \frac{p}{2^k} \rfloor}$, as

$$\begin{aligned}
\widetilde{u}_k &= x^{b_t + b_{t-1} 2 + \cdots + b_{t-k+1} 2^{k-1}} \\
&= x^{b_t + \cdots + b_{t-k+2} 2^{k-2}} x^{b_{t-k+1} 2^{k-1}} = \widetilde{u}_{k-1} x_k^{b_{t-k+1}}
\end{aligned}$$

for $k = 1, \ldots, t-1$;

3. set $\widetilde{w}_{t-1} = 1$ and compute $\widetilde{w}_k = y^{2^{k+1}\lfloor \frac{p}{2^{k+1}} \rfloor}$, as

$$\begin{aligned}\widetilde{w}_k &= y^{b_{t-k-1}2^{k+1}+\cdots+b_2 2^{t-2}+2^{t-1}} \\ &= y^{b_{t-k-1}2^{k+1}} y^{b_{t-k-2}2^{k+2}+\cdots+b_1 2^{t-1}} = \widetilde{w}_{k+1} y_{k+2}^{b_{t-k-1}}.\end{aligned}$$

for $k = t-2, \ldots, 1, 0$;

4. set $m_{t-1} = \widetilde{u}_{t-1}$ and compute m_h using formula (2.19) for $h = t-2, \ldots, 1, 0$.

The computational cost of the method is bounded by $8 \log_2 p$ ops.

In Theorem 2.3 the commutativity between x and y has not been used; thus the statement is true also when x and y are square matrices and (2.18) is written as

$$m_h := \sum_{\ell=0}^{\lfloor \frac{p}{2^h} \rfloor - 1} x^{p-2^h(\ell+1)} \otimes y^{2^h \ell}, \quad h = 0, \ldots, \lfloor \log_2 p \rfloor. \tag{2.20}$$

Equation (2.16) is a special case of (2.20). We have the following corollary of Theorem 2.3.

COROLLARY 2.4. *Let $x \in \mathbb{C}^{n\times n}$ and $y \in \mathbb{C}^{m\times m}$ and let p be as in Theorem* 2.3. *Define $\sigma_k = x^{2^{k-1}} \otimes I_m + I_n \otimes y^{2^{k-1}}$, $\widetilde{u}_k = x^{p-2^k\lfloor \frac{p}{2^k} \rfloor}$, $\widetilde{w}_k = y^{2^{k+1}\lfloor \frac{p}{2^{k+1}} \rfloor}$ for $k = 1, \ldots t$. Then for m_h in* (2.20)*, the following recurrence holds:*

$$m_h = m_{h+1}\sigma_{h+1} + b_{t-h}\widetilde{u}_h \otimes \widetilde{w}_h, \quad h = 0, \ldots, t-2, \tag{2.21}$$

with $m_{t-1} = \widetilde{u}_{t-1} \otimes I_m = x^{p-2^{t-1}} \otimes I_m$.

Algorithm 2 for computing m_h is easily generalized to matrices. It is enough to initialize the sequences with identity matrices, namely, $x_1 = I_n$, $y_1 = I_m$, $\sigma_1 = 2I_n \otimes I_m$, $\widetilde{w}_{t-1} = I_n$, $\widetilde{u}_0 = I_m$, and to set $\sigma_h = x_h \otimes I_m + I_n \otimes y_h$, where m_h is obtained through formula (2.21) with $m_{t-1} = \widetilde{u}_{t-1} \otimes I_n$.

Algorithm 2 can be used to compute the values of $M_{ij}^{(h)}$ which are needed by step 7 of Algorithm 1 of section 2.3. However, in this case we can save computation since most of the quantities are already known from the previous steps. We restate a specialized Algorithm 2 for this problem, recalling that in our case $x = Y_{jj}^T$ and $y = Y_{ii}$.

ALGORITHM 3. Compute $M_{ij}^{(h)}$ in Algorithm 1, where s_i and s_j are the sizes of the blocks V_{ii} and V_{jj}, respectively:

1. set $x_h = (V_{jj}^{(h+1)})^T$, $y_h = V_{ii}^{(h+1)}$, for $h = 0, \ldots, t$;
2. compute $\sigma_h = x_h \otimes I_{s_i} + I_{s_j} \otimes y_h$, for $h = 0, \ldots, t$;
3. set $\widetilde{u}_0 = I_{s_j}$, $\widetilde{u}_h = (U_{jj}^{(\theta)})^T$, where $\theta = 1 + b_1 + b_2 + \cdots + b_{t-h}$, for $h = 1, \ldots, t-1$;
4. set $\widetilde{w}_{t-1} = I_{s_i}$, $\widetilde{w}_h = W_{ii}^{(\theta)}$, where $\theta = 1 + b_1 + b_2 + \cdots + b_{t-h-1}$, for $h = 0, \ldots, t-2$;
5. compute $M_{ij}^{(t-1)} = \widetilde{u}_{t-1} \otimes I_{s_i}$, and $M_{ij}^{(h)}$, for $h = t-2, \ldots, 1, 0$, using formula (2.21).

Algorithm 3 requires a computation just in step 2, where in the scalar case about t sums are required, and step 5, where in the scalar case about $t + \mu$ products and μ sums are required, where μ is the number of nonzero digits in the binary expansion of p.

2.3.2. Generalizations and special cases. The Schur logarithmic algorithm can be used also for a generic nonsingular complex matrix A and for a generic primary pth root function $\varphi(z)$. The formulae given in the previous section still hold and are much more simple, since in the complex Schur form all blocks are 1×1 and thus the linear system is in fact a linear equation. In this case complex arithmetic is used, but the complexity is roughly the same.

Now we discuss how the complex version of our algorithm compares with the other Schur algorithms for small values of p. We slightly change the notation with respect to the previous sections, using lower case letters for scalar complex quantities.

In the complex case, for instance, the linear system to be solved to get y_{ij} is always a scalar equation which can be written as

$$y_{ij} = \frac{t_{ij} - \sum_{h=1}^{\lfloor \log_2 p \rfloor} m_{ij}^{(h)} b_{ij}^{(h+1)} - \sum_{h=2}^{m} c_{ij}^{(h)} u_{jj}^{(h+1)}}{m_{ij}^{(0)}}$$

with $m_{ij}^{(h)} = \sum_{\ell=0}^{\lfloor \frac{p}{2^h} \rfloor - 1} y_{ii}^{2^h \ell} y_{jj}^{p-2^h(\ell+1)}$.

We describe Algorithm 1 for triangular matrices and for small values of p, considering for simplicity just the principal pth root. We recall that in finite arithmetic a different order in the evaluation of the same quantity may lead to different results.

- For $p = 2$ we have $t = 2$, $b_2 = 0$, $m = 1$. For each j we must compute $v_{jj}^{(2)} = t_{jj}^{1/2}$ and for each $i < j$ we must compute $b_{ij}^{(2)} = \sum_{k=i+1}^{j-1} y_{ik} y_{kj}$ and thus $m_{ij}^{(1)} = 1$, $m_{ij}^{(0)} = y_{ii} + y_{jj}$ and

  $$y_{ij} = \frac{t_{ij} - \sum_{k=i+1}^{j-1} y_{ik} y_{kj}}{y_{ii} + y_{jj}}.$$

  This is the same relation used by the Björck and Hammarling algorithm [1]. Thus, our method generalizes the Björck and Hammarling algorithm (and the algorithm of Higham [6] in the real case).
- For $p = 3$ we have $t = 2$, $b_2 = 1$, $m = 2$. For each j we must compute $y_{jj} = v_{jj}^{(2)} = t_{jj}^{1/3}$, $v_{jj}^{(3)} = (v_{jj}^{(2)})^2$, and for each $i < j$ we must compute $b_{ij}^{(2)} = \sum_{k=i+1}^{j-1} y_{ik} y_{kj}$, and $c_{ij}^{(2)} = \sum_{k=i+1}^{j-1} v_{ik}^{(3)} y_{kj}$. Thus, $m_{ij}^{(1)} = y_{jj}$, $m_{ij}^{(0)} = y_{jj}(y_{ii} + y_{jj}) + y_{ii}^2$ and

  $$y_{ij} = \frac{t_{ij} - y_{jj} \sum_{k=i+1}^{j-1} y_{ik} y_{kj} - \sum_{k=i+1}^{j-1} v_{ik}^{(3)} y_{kj}}{y_{jj}(y_{ii} + y_{jj}) + y_{ii}^2}.$$

  This is the recurrence obtained by the Smith algorithm [13] modified in such a way that $V^{(k+1)} = V^{(k)} Y$ (instead of $V^{(k+1)} = Y V^{(k)}$). Thus, for $p = 3$ our algorithm essentially coincides with the one of Smith.
- For $p = 4$ we have $t = 3$, $b_2 = 0$, $b_3 = 0$, $m = 1$. For each j we must compute $y_{jj} = v_{jj}^{(2)} = t_{jj}^{1/4}$, $v_{jj}^{(3)} = (v_{jj}^{(2)})^2$, and for each $i < j$ we must compute $b_{ij}^{(2)} = \sum_{k=i+1}^{j-1} y_{ik} y_{kj}$, $b_{ij}^{(3)} = \sum_{k=i+1}^{j-1} v_{ik}^{(3)} v_{kj}^{(3)}$ and thus $m_{ij}^{(2)} = 1$, $m_{ij}^{(1)} = v_{ii}^{(2)} + v_{jj}^{(2)}$, $m_{ij}^{(0)} = m_{ij}^{(1)}(y_{ii} + y_{jj})$. We have

  $$y_{ij} = \frac{t_{ij} - m_{ij}^{(1)} b_{ij}^{(2)} - b_{ij}^{(3)}}{m_{ij}^{(0)}}.$$

We discuss how this algorithm for the fourth root of a matrix compares with two recalls of the Björck and Hammarling algorithm. The two algorithms are similar, since the quantities $b_{ij}^{(2)}$ and $b_{ij}^{(3)}$ of our algorithm must be computed also by the Björck and Hammarling algorithm and so the most expensive part of the computation is the same for both algorithms. However, they are different, in fact, in the preliminary step; our algorithm computes y_{jj}, the fourth root of t_{jj}, and then the square of y_{jj}, namely, $v_{jj}^{(2)}$, for each j, while two Björck and Hammarling executions compute $v_{jj}^{(2)}$ and then its square root, that is, y_{jj}. Moreover, the formulae for y_{ij} in the two algorithms correspond essentially to the two evaluations

$$y_{ij} = \frac{t_{ij} - (y_{ii}^2 + y_{jj}^2)b_{ij}^{(2)} - b_{ij}^{(3)}}{(y_{ii}^2 + y_{jj}^2)(y_{ii} + y_{jj})}, \quad y_{ij} = \frac{\frac{t_{ij} - b_{ij}^{(3)}}{y_{ii}^2 + y_{jj}^2} - b_{ij}^{(2)}}{y_{ii} + y_{jj}},$$

where the former formula is the one used in our algorithm and is computed with 7 ops (we assume y_{ii}^2 and y_{jj}^2 are known) and the second is the one obtained by two steps of the Björck and Hammarling algorithm and is computed with 6 ops. Finally, our algorithm requires the computation of $v_{ij}^{(2)}$, which needs 5 more ops.

In summary, our algorithm requires about $3n^2$ ops more than two steps of the Björck and Hammarling algorithm. This extra cost, for a sufficiently large n, is negligible with respect to n^3 and thus we can conclude that the performances of the two algorithms are essentially the same.

- For $p = 5$ the new algorithm is different from the Smith algorithm since it requires less intermediate matrices and gives a good advantage. It also takes advantage with respect to the algorithm of Greco and Iannazzo [2], since for each i, it does not require the powers y_{ii}^2, y_{ii}^3, and y_{ii}^4 to derive y_{ij} but just y_{ii}^2, y_{ii}^4.

For $p = 2^k$ the same argument as for $p = 4$ allows one to conclude that our algorithm and k consecutive steps of the Björck and Hammarling algorithm require about $\frac{n^3}{3} \log_2 p$, with a slightly larger total count in $O(n^3)$ terms for our algorithm. It is worth pointing out that for the memory required, the Björck and Hammarling algorithm is preferable since it does not require the storage of all 2^sth powers of Y for $s = 2, \ldots, k$. On the other hand, if a nonprincipal root is required, it may be more difficult to choose the right determination using the Björck and Hammarling algorithm.

Another generalization of the Schur logarithmic method is to block upper triangular matrices. In fact, in the construction of the formulae and in the algorithm, the hypothesis that T is upper quasi-triangular can be substituted with the hypothesis that T is block upper triangular. Nevertheless, block upper triangular is more complicated since there is no such simple formula as (2.2) to compute $\varphi(T_{jj})$ for a generic block of size greater than 1.

2.3.3. To use or not to use the factorization of *p*. If $p = p_1 p_2$, then the relation $Y^{1/p} = (Y^{1/p_1})^{1/p_2}$ implies that the principal pth root of Y can be computed in two steps: first, compute the p_1th root of Y; then, compute the p_2th root of Y^{1/p_1}. If a factorization of p is known and we use this trick, that is, we choose to compute the pth root of Y by computing a sequence of roots of smaller indices corresponding to the factors of p, then we say that we *use the factorization* of p. For Smith's method,

using the factorization gives a great advantage, since the computational cost depends polynomially on p.

In our logarithmic method, if $p = p_1 p_2$, since $\log p = \log p_1 + \log p_2$, we do not expect a great advantage in terms of the computational cost when the factorization is used. In fact, if p has t_p binary digits, m_p of which are nonzero, the computational cost is $\frac{1}{3}(t_p + m_p - 2)n^3 + o(n^3)$ ops and this leads to three possibilities:

- using the factorization is indifferent, e.g., $p = 2^k$, where $t_p + m_p - 2 = t_p - 1 = k = k(t_2 + m_2 - 2)$;
- using the factorization is convenient, e.g., $p = 15$, where $t_p + m_p - 2 = 6 > 5 = (t_3 + m_3 - 2) + (t_5 + m_5 - 2)$;
- using the factorization is not convenient, e.g., $p = 33$, where $t_p + m_p - 2 = 6 < 7 = (t_3 + m_3 - 2) + (t_{11} + m_{11} - 2)$.

Analyzing the positive integers from 2 to 10^5, we have counted $35,964$ cases in which using the factorization is indifferent, $42,698$ cases where using the factorization is convenient, and $22,237$ cases where using the factorization is not convenient.

So there is no clear advantage in using the factorization of p, and the strategy can be decided case by case. However, an aspect that could suggest the use of factorization is the gain in terms of memory required, since fewer intermediate matrices need to be stored if the factorization is used.

2.4. Arbitrary powers of a matrix. Consider a primary $\frac{q}{p}$th power Z of A. The matrix Z can be written as $Z = X^q$, where X is the corresponding primary pth root of A. The trivial algorithm for computing the desired primary $\frac{q}{p}$th power of A is to compute the pth root X and then raise it to the power q.

A more interesting implementation is obtained using a Schur form of A, namely, $A = QTQ^*$, and computing using Algorithm 1 the desired primary pth root of A, say, $Y = Q^*XQ$. Computing the qth power of Y yields the desired power of A.

The step of raising Y to the power q is implemented using the matrices $V^{(k)} = Y^{2^{k-2}}$ of Algorithm 1 and the dyadic expansion of q, say,

$$q = 2^{d_1} + 2^{d_2} + \cdots + 2^{d_g}, \quad d_1 > d_2 > \cdots > d_g \geqslant 0.$$

Let $Z^{(1)} = V^{(d_1+2)} = Y^{2^{d_1}}$ and define $Z^{(k)} = Z^{(k-1)}V^{(d_k+2)} = Z^{(k-1)}Y^{2^{d_k}}$ for $k = 2, \ldots, g$. We have $Y^q = Z^{(g)}$. The computation of $Z^{(g)}$ could be made using the known powers of Y and requires, in the worst case, $g - 1 = O(\log_2 p)$ multiplications of upper quasi-triangular matrices.

For an irrational $\alpha \in (0, 1)$, it is possible to approximate α by a suitable rational number $\frac{q}{p}$ (using, for instance, the continued fraction expansion of α) and then use Algorithm 1 of section 2.3 to compute $A^{\frac{q}{p}}$ as an approximation of A^α.

Observe that given a rational approximation $\frac{q}{p}$ of an irrational number α, just p primary $\frac{q}{p}$th powers exist, while there is an infinite number of primary powers of A to α. We can conclude that when we approximate a primary power, we are approximating at the same time infinitely many other primary powers.

However, this approach seems to be of limited use, since it introduces a new source of ill-conditioning (the pth root computation) which for large p and q can be much greater than the conditioning of the power. When A has no nonpositive real eigenvalues and the principal power A^α is required, then unless α is approximated with a fraction q/p with small values of p and q, it is convenient to use another approach as, for instance, the algorithm of Higham and Lin [9].

3. Numerical tests. We present some numerical tests to illustrate the behavior of Algorithm 1 in finite arithmetic. The tests have been performed using MATLAB R2011b with unit roundoff $2^{-53} \approx 1.1 \times 10^{-16}$.

Comparisons have been made between our algorithm (`pthroot_log`) and some existing algorithms for matrix powers. For the Smith method [13], we have used the implementation `rootpm_real` of Higham's matrix function toolbox [5], while for the Schur–Padé [9] method we have used the authors' script `powerm_pade`. For the other algorithms, namely, the Greco–Iannazzo method [2], the Schur–Halley method [11], and the explog method, that is, using the formula $A^\alpha = e^{\alpha \log A}$, we have used our implementations. The scripts for MATLAB and Octave are available at the first author's personal webpage.

We compare the performance of the algorithms on some test matrices. In particular the accuracy of the pth root computation is estimated in terms of the quantity

$$\rho_A(\widetilde{X}) := \frac{\|A - \widetilde{X}^p\|}{\|\widetilde{X}\| \left\| \sum_{i=0}^{p-1} (\widetilde{X}^{p-1-i})^T \otimes \widetilde{X}^i \right\|}, \tag{3.1}$$

where $\widetilde{X}$ is the computed pth root of A and $\|\cdot\|$ is any matrix norm. (In our tests we use the spectral norm.) In [7], the quantity ρ_A is proved to be a measure of accuracy more realistic than the norm of the relative residual, say, $\|\widetilde{X}^p - A\|/\|A\|$.

Test 1. In section 2.3 we have shown that the computational cost of Algorithm 1 for an upper quasi-triangular matrix is $\theta n^3 \log_2 p$ with $\theta \leqslant 1$, where the precise value of θ depends on the number of 2×2 blocks and on the number of nonzero digits in the binary expansion of p.

To highlight the logarithmic dependence of Algorithm 1 on p we compute the principal pth root for $10 \leqslant p \leqslant 300$ of an upper quasi-triangular matrix with no nonpositive real eigenvalues. (The matrix has been randomly chosen once and for all.) The CPU time for any value of p is shown in Figure 3.1. The curve has a logarithmic trend with a lot of oscillations due to the fact that the computational cost of Algorithm 1 depends on the number of ones in the binary expansion of p with low peaks for $p = 2^k$ and high peaks for $p = 2^k - 1$.

We have also computed the principal pth root with Algorithm 1 but using the factorization of p, as discussed in section 2.3.3. The timing is generally greater than or equal to the one obtained without using the factorization.

Test 2. As in [9, Experiment 1], we consider the matrix

$$A(\varepsilon) = \begin{bmatrix} 1 & 1 \\ 0 & 1+\varepsilon \end{bmatrix},$$

with $\varepsilon = 10^{-t}$, whose eigenvectors become very ill-conditioned as ε tends to 0. We compute the power $A(\varepsilon)^{q/p}$ for $(q,p) \in \{(1,10),(1,2),(9,10)\}$ and with 65 equally spaced values of $t \in [0,16]$. For all these values, the conditioning of the matrix power $\kappa_{x^p}(A(\varepsilon))$ (as defined in [9]) is of the order 1. Using the eigendecomposition of $A(\varepsilon)$, say, $A(\varepsilon) = M \operatorname{diag}\big(\lambda_1(\varepsilon), \lambda_2(\varepsilon)\big) M^{-1}$, that is, computing $A(\varepsilon)^{q/p} = M \operatorname{diag}\big(\lambda_1(\varepsilon)^{q/p}, \lambda_2(\varepsilon)^{q/p}\big) M^{-1}$, yields poor results as ε tends to 0. As a measure of the error we consider the quantity $\|\widetilde{X} - A(\varepsilon)^{q/p}\|/\|A(\varepsilon)^{q/p}\|$, where $\widetilde{X}$ is the computed value of $A(\varepsilon)^{q/p}$ and the "exact" value $M(\operatorname{diag}(\lambda_1(\varepsilon)^{q/p}, \lambda_2(\varepsilon)^{q/p}) M^{-1}$ is computed using VPA in MATLAB.

Using Algorithm 1 gives an error bounded by $4u$, showing that it is insensitive to the eigenconditioning.

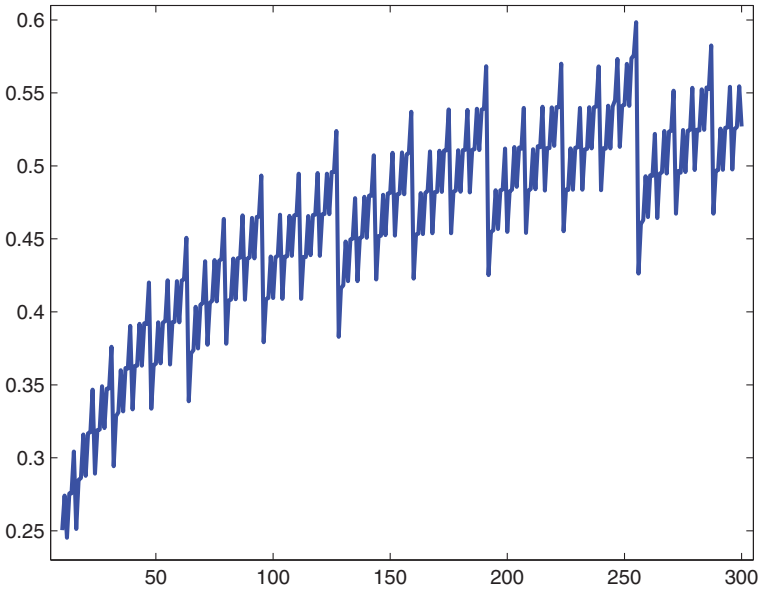

FIG. 3.1. *The CPU time (in seconds) required by Algorithm* 1 *to compute the principal* p*th root of a random* 50×50 *upper quasi-triangular matrix for certain values of* p.

Test 3. As in [9], we select among the classical test matrices from the MATLAB `gallery` function and the matrix computation toolbox a number of 10×10 matrices with no nonpositive real eigenvalues and we compute their principal pth root for $p = 2, 3, 12, 52$. We compare the residuals obtained using (3.1). As one can see in Figure 3.2, the numerical results of Algorithm 1 are very good and often coincide with the ones of the customary Schur algorithm. They compare very well with the other algorithms.

Test 4. Algorithm 1 can be used to compute all primary pth roots of a given matrix A. We consider the matrix

$$A = \begin{bmatrix} a & 1 \\ 0 & b \end{bmatrix} \tag{3.2}$$

for various $a, b \notin (-\infty, 0]$ and $a \neq b$. All p^2 primary pth roots of A are given symbolically by

$$X_{k,h} = \begin{bmatrix} \omega^{k-1} a^{1/p} & \dfrac{\omega^{h-1} b^{1/p} - \omega^{k-1} a^{1/p}}{b - a} \\ 0 & \omega^{h-1} b^{1/p} \end{bmatrix}, \quad k, h = 1, \dots, p,$$

where $\omega = e^{2\mathbf{i}\pi/p}$ and $a^{1/p}$ and $b^{1/p}$ are the principal pth roots of a and b, respectively.

For each computed pth root $\widetilde{X}_{k,h}$ we compute the relative residual $\operatorname{res}(\widetilde{X}_{k,h}) = \|\widetilde{X}_{k,h}^p - A\|/\|A\|$, the value $\rho_A(\widetilde{X}_{k,h})$ of (3.1), and the relative error $\operatorname{err}(\widetilde{X}_{k,h}) = \|\widetilde{X}_{k,h} - X_{k,h}\|/\|X_{k,h}\|$, where $X_{k,h}$ is computed using VPA in MATLAB with a sufficient number of digits such that it is exact (up to machine precision); finally we compute the quantity $\beta(\widetilde{X}_{k,h}) = \|\widetilde{X}_{k,h}\|^p/\|A\|$, which has been introduced in [6, 13] as a measure of stability.

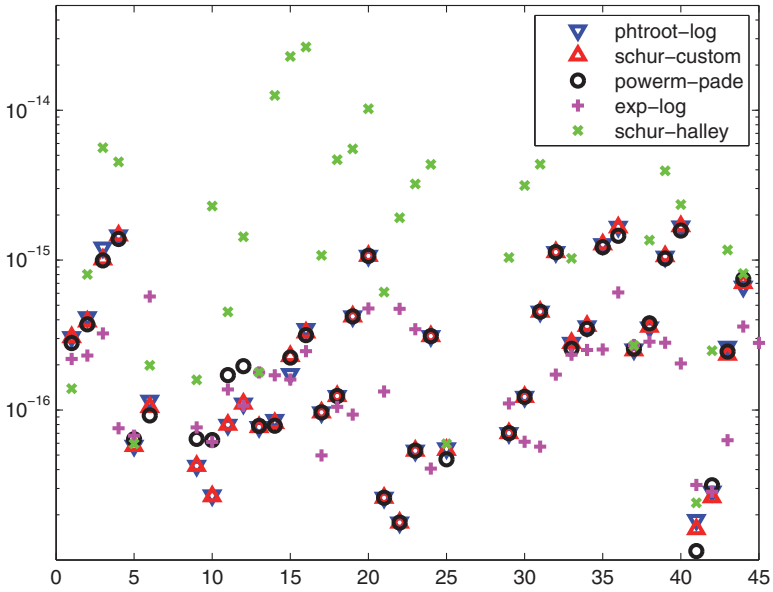

FIG. 3.2. *Comparison, in terms of relative residual, of the performance of various algorithms for computing the principal pth root for* $p = 2, 3, 12, 52$ *of certain test matrices.*

For a and b not too close, Algorithm 1 computes with good accuracy all primary roots. For instance, we run the algorithm for $(a, b) \in \{(1, 2), (\varepsilon, 1/\varepsilon), (1 + \mathbf{i}, 1 - \mathbf{i})\}$ (with $\varepsilon = 10^{-8}$) and for $p \in \{3, 5, 11\}$ and we get an error always smaller than $23u$ and a value of $\rho_A(\widetilde{X})$ always smaller than $2u$. For larger values of p these errors grow: for $p = 53$ we get a maximum error of about $350u$.

For $a \approx b$ and choosing different determinations for the roots of a and b, that is, choose $k \neq h$, the value of $\beta(\widetilde{X}_{k,h})$ becomes very large, highlighting ill-conditioning of the computed root; this captures the fact that in the limit case, for $a = b$, choosing different determinations yields a nonprimary pth root. The numerical results for $a = 1$ and $b = 1 - \varepsilon$, with $\varepsilon = 10^{-8}$, are given in Table 3.1. As one can see, for $k \neq h$, the value $\rho(\widetilde{X})$ is not a good measure of accuracy anymore, since in this case one of the matrices appearing in the denominator of (3.1) has very small norm. Moreover, the error in computing the pth root degrades, according to $\beta(\widetilde{X}_{k,h})^{1/p}$. Unfortunately, this is the best that can be done numerically in the generic case, as discussed in [6, 13]. Along the diagonal of Table 3.1, that is, computing the same determination for near eigenvalues, one can see that the computation gives good results.

Test 5. We try to understand how Algorithm 1, provided with the modifications described in section 2.4, works with fractional powers of a matrix. We consider the matrix $A = MDM^{-1}$ with $D = \mathrm{diag}([1 \;\; 2 \;\; 3])$ and

$$M = \begin{bmatrix} 1 & 1 & 1 \\ 1 & 2 & 3 \\ -3 & -2 & 1 \end{bmatrix}.$$

For $p = 5, 11, 31, 101$, we use Algorithm 1 and the Schur–Padé algorithm to compute the powers $A_i^{q/p}$ for $1 \leqslant q \leqslant p - 1$ and evaluate the relative errors $\|\widetilde{X} - A_i^{q/p}\|/\|A_i^{q/p}\|$, where $\widetilde{X}$ is the computed value of $A_i^{q/p}$ and the "exact" value

TABLE 3.1
Various measures of accuracy in computing all third roots of the matrix A of (3.2) with $a = 1$ and $b = 1 - 10^{-8}$; k and h are the determination chosen for the third root of a and b, respectively.

$h\backslash k$	1	2	3
1	$\mathrm{res}(\widetilde{X}_{1,1}) = 9.7 \cdot 10^{-17}$ $\mathrm{err}(\widetilde{X}_{1,1}) = 0$ $\rho_A(\widetilde{X}_{1,1}) = 3.2 \cdot 10^{-17}$ $\beta(\widetilde{X}_{1,1}) = 1.0$	$\mathrm{res}(\widetilde{X}_{1,2}) = 2.1 \cdot 10^{-8}$ $\mathrm{err}(\widetilde{X}_{1,2}) = 6.4 \cdot 10^{-8}$ $\rho_A(\widetilde{X}_{1,2}) = 6.4 \cdot 10^{-33}$ $\beta(\widetilde{X}_{1,2}) = 3.2 \cdot 10^{24}$	$\mathrm{res}(\widetilde{X}_{1,3}) = 1.3 \cdot 10^{-8}$ $\mathrm{err}(\widetilde{X}_{1,3}) = 1.2 \cdot 10^{-7}$ $\rho_A(\widetilde{X}_{1,3}) = 4.0 \cdot 10^{-33}$ $\beta(\widetilde{X}_{1,3}) = 3.2 \cdot 10^{24}$
2	$\mathrm{res}(\widetilde{X}_{2,1}) = 9.2 \cdot 10^{-9}$ $\mathrm{err}(\widetilde{X}_{2,1}) = 6.0 \cdot 10^{-8}$ $\rho_A(\widetilde{X}_{2,1}) = 2.9 \cdot 10^{-33}$ $\beta(\widetilde{X}_{2,1}) = 3.2 \cdot 10^{24}$	$\mathrm{res}(\widetilde{X}_{2,2}) = 4.2 \cdot 10^{-16}$ $\mathrm{err}(\widetilde{X}_{2,2}) = 2.7 \cdot 10^{-16}$ $\rho_A(\widetilde{X}_{2,2}) = 1.4 \cdot 10^{-16}$ $\beta(\widetilde{X}_{2,2}) = 1.0$	$\mathrm{res}(\widetilde{X}_{2,3}) = 1.8 \cdot 10^{-8}$ $\mathrm{err}(\widetilde{X}_{2,3}) = 6.5 \cdot 10^{-8}$ $\rho_A(\widetilde{X}_{2,3}) = 5.7 \cdot 10^{-33}$ $\beta(\widetilde{X}_{2,3}) = 3.2 \cdot 10^{24}$
3	$\mathrm{res}(\widetilde{X}_{3,1}) = 2.8 \cdot 10^{-8}$ $\mathrm{err}(\widetilde{X}_{3,1}) = 1.2 \cdot 10^{-7}$ $\rho_A(\widetilde{X}_{3,1}) = 8.6 \cdot 10^{-33}$ $\beta(\widetilde{X}_{3,1}) = 3.2 \cdot 10^{24}$	$\mathrm{res}(\widetilde{X}_{3,2}) = 9.2 \cdot 10^{-9}$ $\mathrm{err}(\widetilde{X}_{3,2}) = 5.9 \cdot 10^{-8}$ $\rho_A(\widetilde{X}_{3,2}) = 2.9 \cdot 10^{-33}$ $\beta(\widetilde{X}_{3,2}) = 3.2 \cdot 10^{24}$	$\mathrm{res}(\widetilde{X}_{3,3}) = 8.1 \cdot 10^{-16}$ $\mathrm{err}(\widetilde{X}_{3,3}) = 5.1 \cdot 10^{-16}$ $\rho_A(\widetilde{X}_{3,3}) = 2.7 \cdot 10^{-16}$ $\beta(\widetilde{X}_{3,3}) = 1.0$

$MD^{q/p}M^{-1}$ is computed using VPA in MATLAB. The results are drawn in Figure 3.3, where the dashed line represents the error for Algorithm 1, while the continuous line represents the error for the Schur–Padé algorithm.

As one can see, for moderate values of q, our algorithm shows a good accuracy, which unfortunately degrades as q becomes large. For this reason we recommend the use of our algorithm just for moderate values of q.

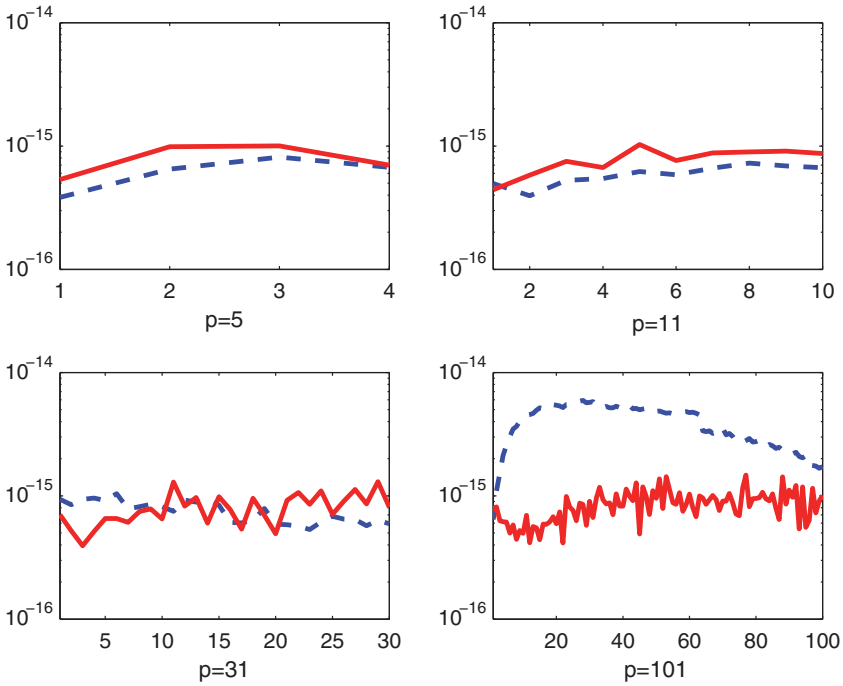

FIG. 3.3. *The relative errors in the computation of $A^{q/p}$, with A as in Test 5, using Algorithm 1 (dashed line) and the Schur–Padé algorithm (continuous line) for $1 \leqslant q \leqslant p - 1$.*

4. Conclusions. We have presented a new Schur algorithm for computing matrix roots. A crude comparison of the computational costs shows that for an upper triangular matrix, Algorithm 1 is faster than the other existing algorithms in most of the interesting cases. For instance, for matrices with real eigenvalues the cost of our algorithm is $\theta n^3 \log_2 p$ with $\theta \leqslant \frac{2}{3}$, while the cost of the Schur–Padé algorithm [9] is kn^3, where k is usually larger than 7; this makes our algorithm cheaper, at least for moderate values of p.

Concerning numerical stability, the new algorithm has shown in all our experiments the same stability properties as the customary Schur method. Nevertheless, an analysis of the numerical stability is yet to be given.

All these properties make the method reliable for the computation of matrix roots. It can be also considered for the computation of fractional powers of the kind A^α with $\alpha \in (0, 1)$ well approximated by a fraction p/q with small p and q.

Acknowledgments. We wish to thank the anonymous referees for their remarks, which enabled us to improve the presentation of the paper.

REFERENCES

[1] Å. Björck and S. Hammarling, *A Schur method for the square root of a matrix*, Linear Algebra Appl., 52/53 (1983), pp. 127–140.
[2] F. Greco and B. Iannazzo, *A binary powering Schur algorithm for computing primary matrix roots*, Numer. Algorithms, 55 (2010), pp. 59–78.
[3] C.-H. Guo, *On Newton's method and Halley's method for the principal pth root of a matrix*, Linear Algebra Appl., 432 (2010), pp. 1905–1922.
[4] C.-H. Guo and N. J. Higham, *A Schur-Newton method for the matrix pth root and its inverse*, SIAM J. Matrix Anal. Appl., 28 (2006), pp. 788–804.
[5] N. J. Higham, *The Matrix Functions Toolbox*. http://www.ma.man.ac.uk/ higham/mctoolbox, (accessed date: Feb. 2012).
[6] N. J. Higham, *Computing real square roots of a real matrix*, Linear Algebra Appl., 88/89 (1987), pp. 405–430.
[7] N. J. Higham, *Functions of Matrices: Theory and Computation*, SIAM Philadelphia, 2008.
[8] N. J. Higham and L. Lin, *On pth roots of stochastic matrices*, Linear Algebra Appl., 435 (2011), pp. 448–463.
[9] N. J. Higham and L. Lin, *A Schur-Padé algorithm for fractional powers of a matrix*, SIAM J. Matrix Anal. Appl., 32 (2011), pp. 1056–1078.
[10] B. Iannazzo, *On the Newton method for the matrix pth root*, SIAM J. Matrix Anal. Appl., 28 (2006), pp. 503–523.
[11] B. Iannazzo, *A family of rational iterations and its application to the computation of the matrix pth root*, SIAM J. Matrix Anal. Appl., 30 (2008), pp. 1445–1462.
[12] B. Laszkiewicz and K. Ziętak, *A Padé family of iterations for the matrix sector function and the matrix pth root*, Numer. Linear Algebra Appl., 16 (2009), pp. 951–970.
[13] M. I. Smith, *A Schur algorithm for computing matrix pth roots*, SIAM J. Matrix Anal. Appl., 24 (2003), pp. 971–989.