Numerical Algorithms manuscript No.
(will be inserted by the editor)

A binary powering Schur algorithm for computing primary matrix roots

Federico Greco · Bruno Iannazzo

Received: date / Accepted: date

Abstract An algorithm for computing primary roots of a nonsingular matrix A is presented. In particular, it computes the principal root of a real matrix having no nonpositive real eigenvalues, using real arithmetic. The algorithm is based on the Schur decomposition of A and has an order of complexity lower than the customary Schur based algorithm, namely the Smith algorithm.

Keywords matrix pth root · matrix functions · Schur method · binary powering technique

Mathematics Subject Classification (2000) 65F30 · 15A15

1 Introduction

Let p be a positive integer. A primary pth root of a square matrix $A \in \mathbb{C}^{n\times n}$ is a solution of the matrix equation $X^p - A = 0$ that can be written as a polynomial of A. If A has ℓ distinct eigenvalues, say $\lambda_1, \dots, \lambda_\ell$, none of which is zero, then A has exactly p^ℓ primary pth roots. They are obtained as

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

where f is any of the p^ℓ analytic functions defined on the spectrum of A, denoted by $\sigma(A) := \{\lambda_1, \dots, \lambda_\ell\}$, and such that $f(z)^p = z$ and γ is a closed contour which encloses $\sigma(A)$. The reason why $f(A)$ is a polynomial of A is subtle and it is well explained in [10].

If A has no nonpositive real eigenvalues then there exists only one primary pth root whose eigenvalues lie in the sector

$$\mathcal{S}_p = \{z \in \mathbb{C} \setminus \{0\} \,:\, |\arg(z)| < \pi/p\}, \tag{2}$$

Federico Greco, Bruno Iannazzo
Dipartimento di Matematica e Informatica, Università di Perugia
Via Vanvitelli 1, I-06123 Perugia, Italy
E-mail: {greco,bruno.iannazzo}@dipmat.unipg.it

which is called principal pth root.

The main numerical problem is to compute the principal pth root of A, whose applications arise in finance or in the numerical computation of other matrix functions [8,9,15]. In particular if A is real and has no nonpositive real eigenvalues, then the principal pth root is proved to be real [8], and in order to compute it, it is preferable to have an algorithm which works entirely in real arithmetic.

The reliable algorithms are essentially of two kinds:

1. Algorithms based on matrix iterations;
2. Algorithms based on the Schur normal form.

In the first case, one uses a rational matrix iteration which converges to the principal pth root of A. This approach is very complicated since the iterations usually do not depend continuously on the initial data, that is, if a perturbation on some iterate is introduced then it is potentially amplified by the subsequent steps and could result in numerical instability. Moreover, in the case $p > 2$ the convergence properties, also in the scalar case, are hard to describe.

The first rational iteration used for the square root is the so-called Newton method $X_{k+1} = \frac{1}{2}(X_k + X_k^{-1}A)$, which was observed to be unstable by Laasonen [14], but the instability was first analyzed by Higham [6]. Some stable iterations have been proposed, the first of them is the Denman and Beavers iteration [2] and many others have followed [11,17].

The case $p > 2$ is more complicated. In order to have a general algorithm, some kind of preprocessing of the matrix A should be done. The first general and stable algorithm was given by Iannazzo [12] and some others have followed [3,4,13,16]. The computational cost of these algorithms is $O(n^3 \log_2 p)$ arithmetic operations (ops) and the storage required is $O(n^2 \log_2 p)$ real numbers.

The algorithms based on some matrix iteration show good numerical stability in the numerical tests, even if their behavior in the finite arithmetic for any matrix A is practically unpredictable and a thorough analysis is yet to be developed. Moreover, these algorithms compute just the principal pth root, and it is not clear if they can compute any of the primary pth roots of A with the same computational cost.

For the second class of algorithms, in order to compute a solution of $X^p - A = 0$, one computes the Schur normal form of A, say $Q^*AQ = R$, where Q is unitary and R is upper triangular and then solves the equation $Y^p - R = 0$ and deduces $X = QYQ^*$. Since Y is proved to be upper triangular, the equation $Y^p - R = 0$ is solved by a recursion on the elements of Y [8].

In the important case in which A is real, the real Schur form of A is formed, say $Q^TAQ = R$, where Q is orthogonal and R is quasi-upper triangular, that is real and block upper triangular with diagonal blocks of size 1 or 2 according as they correspond to one real or a couple of complex conjugate eigenvalues, respectively. The equation $Y^p - R = 0$ is solved by a recursion on the blocks of Y which is proved to have the same block structure as R. This approach works on the idea of the Schur-Parlett recurrence for computing general matrix functions. The case $p = 2$ was first developed by Björck and Hammarling [1] in the complex case and then by Higham [7] in the real case. Finally, the case $p > 2$ was worked out by Smith [18].

The method developed by Smith has a cost of $O(n^3p)$ ops and requires the storage of $O(n^2p)$ real numbers. If p is composite, say $p = q_1q_2$, it is thus convenient to form first the q_1th root and then the q_2th root of A. However, if p is prime, the cost of the method of Smith can be large and that makes the algorithm uneffective.

A nice feature of the Smith algorithm is that it has been proved to be backward stable, thus it is in some sense optimal in finite arithmetic. Moreover, it can compute any of the primary pth roots of A with the same cost.

We propose a new algorithm based on the Schur normal form of A whose cost is lowered to $O(n^2p + n^3 \log_2 p)$ ops and the storage is lowered to $O(np + n^2 \log_2 p)$ real numbers. The proposed algorithm combines the advantages of being based on the Schur form and the low computational cost of the iterations.

The numerical tests show that the new algorithm reach the same numerical accuracy as the one of Smith.

The paper is organized as follows. In Section 2 we describe and analyze the proposed method; in Section 3 we summarize the resulting algorithm; in Section 4 we discuss how to further reduce the cost of the algorithm; finally, in Section 5 we present some numerical experiments which confirm the reliability of the algorithm.

2 A Schur method based on the binary powering technique

One of the most used methods for computing the principal pth root, for p positive integer, of a real matrix $A \in \mathbb{R}^{n\times n}$ having no nonpositive real eigenvalues has been proposed by Smith in [18]. Since the principal pth root of such a matrix is proved to be real, the method is designed to work entirely in real arithmetics.

The idea of the algorithm is to compute the real Schur normal form of A, say $Q^T AQ = R$, where Q is orthogonal and R is real and quasi-upper triangular, namely the matrix is block $\sigma\times\sigma$, block upper triangular and its σ diagonal blocks are real numbers or 2×2 real matrices corresponding to a couple of complex conjugate eigenvalues. Once the real Schur form is obtained, one applies the transformation to the equation

$$X^p = A, \tag{3}$$

obtaining the new equation

$$U^p = R, \tag{4}$$

where $U = Q^T XQ$. If X is the principal pth root of A then U is the principal pth root of R as well, moreover, the matrix U is quasi-upper triangular with the same block structure as R (that follows from the fact that a primary pth root of a matrix A is a polynomial in A, see also [8]).

From a solution U of (4) one obtains a solution of (3) using $X = QUQ^T$, so if U is chosen to be the principal pth root of R, say $U = R^{1/p}$ then the principal pth root of A is $A^{1/p} = QUQ^T$ (recall that U and QUQ^T have the same eigenvalues).

The key point of the algorithm is the solution of (4) which is done by a clever recursion, employing the same idea as the methods of Björck and Hammarling [1] and Higham [6] for the matrix square root.

The recursion of Smith [18,19] is obtained considering the sequence of matrices

$$\begin{cases} \widetilde{V}^{(0)} &= U, \\ \widetilde{V}^{(k)} &= U\widetilde{V}^{(k-1)} = U^{k+1}, \qquad k = 1,\ldots,p-2, \end{cases} \tag{5}$$

having the same quasi-triangular structure as U and R. The diagonal blocks of U are obtained from the pth roots of the corresponding blocks of R using a simple formula:

if U_{ii} is a 1×1 block, then U_{ii} is the principal pth root of the scalar R_{ii}; if U_{ii} is a 2×2 block corresponding to the complex conjugate eigenvalues $\theta \pm \boldsymbol{i}\mu$, then

$$U_{ii} = \alpha I + \frac{\beta}{\mu}(R_{ii} - \theta I), \tag{6}$$

where I is the 2×2 identity matrix and $\alpha + \boldsymbol{i}\beta$ is the principal pth root of $\theta + \boldsymbol{i}\mu$. The upper triangular part of U is obtained, a block column at a time, equating the (i, j) blocks in equation (5) (more details can be found in [8,18]).

The main drawback of the Smith method is the high cost in terms of arithmetic operations (ops) and storage for large p, in particular it needs $O(n^3 p)$ ops and the storage of $O(n^2 p)$ real numbers. The most expensive part is the computation of the elements of the intermediate matrices $\widetilde{V}^{(k)}$ and their storage. The idea of our algorithm is to use a recursion similar to the one of Smith but with less intermediate matrices obtaining an algorithm with similar features but less expensive. The proposed recursion is based on the binary powering decomposition of the integer p, that is

$$p = \sum_{k=0}^{\lfloor \log_2 p \rfloor} b_k \, 2^k, \qquad \text{for a unique choice of } b_0, \ldots, b_{\lfloor \log_2 p \rfloor} \in \{0, 1\}, \tag{7}$$

where $b_{\lfloor \log_2 p \rfloor} = 1$. Observe that b_i, for $i = 0, \ldots, \lfloor \log_2 p \rfloor$, are the digits in the binary representation of p.

We define also the sets

$$c(p) = \{k \,:\, b_k = 1\}, \quad c(p)^+ = c(p) \setminus \{0\}. \tag{8}$$

The set $c(p)$ has cardinality $m + 1$, for some nonnegative integer m, while the set $c(p)^+$ coincides with $c(p)$ if p is even (that is $0 \notin c(p)$) and has cardinality m if p is odd. Clearly, $m + 1$ denotes the number of 1s comparing in the binary representation of p. Let $c_0, c_1, \ldots, c_m$ be the sequence obtained by sorting the elements of $c(p)$ by decreasing order. Note that if $m > 0$, then

$$\begin{aligned} &c_0 = \lfloor \log_2 p \rfloor, \\ &c_h = \max\{k \,:\, k < c_{h-1},\ b_k = 1\}, \qquad h = 1, \ldots, m, \end{aligned} \tag{9}$$

while if $m = 0$, then the sequence contains just the term $c_0 = \lfloor \log_2 p \rfloor$.

Since $U^p = R$,

$$R = U^p = \prod_{k=0}^{\lfloor \log_2 p \rfloor} U^{b_k 2^k} = \prod_{h=0}^{m} U^{2^{c_h}},$$

and it is possible to devise a method based on a sequence of $c_0 + m = O(\log_2 p)$ intermediate matrices from which construct a recursion for computing U.

We obtain the further c_0 matrices as follows

$$\begin{cases} V^{(0)} \;= U \\ V^{(k)} \;= V^{(k-1)} \cdot V^{(k-1)} \;= U^{2^k}, \qquad k = 1, \ldots, c_0, \end{cases} \tag{10}$$

and the latter m matrices as follows

$$\begin{cases} W^{(0)} \;= V^{(c_0)} \\ W^{(h)} \;= W^{(h-1)} \cdot V^{(c_h)}, \qquad h = 1, \ldots, m, \end{cases} \tag{11}$$

where $W^{(m)} = R$.

The matrices $V^{(k)}$ and $W^{(h)}$ have the same block structure as R, being quasi-upper triangular. We denote the blocks of $V^{(k)}$ and $W^{(h)}$ by $V_{ij}^{(k)}$ and $W_{ij}^{(h)}$, respectively, where the indices i, j go from 1 to σ, where σ^2 is the number of blocks in the partitioning of R. Each choice of i and j could correspond to a 1×1, or to a 1×2, or to a 2×1, or to a 2×2 block, according to the quasi-triangular structure of R.

The idea of the proposed method is to compute, using (10) and (11), the blocks of U, that is $V^{(0)}$, in the following order: first, compute the diagonal blocks of U, then compute the upper part of U, $V^{(k)}$ and $W^{(h)}$ a column at a time from the bottom to the top. During the computation we need the diagonal blocks of U^q for $q = 1, \dots, p$. These blocks can be computed with a cost of $O(n^2 p)$ ops and a storage of $O(np)$ real numbers.

Relations (10) and (11) can be restated in terms of blocks, for each $i, j = 1, \dots, \sigma$ such that $i \leqslant j$, for $k = 1, \dots, c_0$, and for $h = 1, \dots, m$ one has

$$\begin{aligned} V_{ij}^{(k)} &= \sum_{\xi=i}^{j} V_{i\xi}^{(k-1)} V_{\xi j}^{(k-1)}, \\ W_{ij}^{(h)} &= \sum_{\xi=i}^{j} W_{i\xi}^{(h-1)} V_{\xi j}^{(c_h)}, \end{aligned} \tag{12}$$

while for $i > j$ the blocks $V_{ij}^{(k)}$ and $W_{ij}^{(h)}$ are zero for each k.

In order to get useful formulae we isolate the terms containing the indices i and j in the sum, obtaining

$$V_{ii}^{(k-1)} V_{ij}^{(k-1)} + V_{ij}^{(k-1)} V_{jj}^{(k-1)} = V_{ij}^{(k)} - B_{ij}^{(k)}, \tag{13}$$

$$W_{ii}^{(h-1)} V_{ij}^{(c_h)} + V_{ij}^{(c_h)} W_{jj}^{(h-1)} = W_{ij}^{(h)} - C_{ij}^{(h)}. \tag{14}$$

where $B_{ij}^{(k)}$, and $C_{ij}^{(h)}$ denote something which is already known when one is computing the block U_{ij}, in fact, for $j > i + 1$,

$$B_{ij}^{(k)} = \sum_{\xi=i+1}^{j-1} V_{i\xi}^{(k-1)} V_{\xi j}^{(k-1)}, \qquad C_{ij}^{(h)} = \sum_{\xi=i+1}^{j-1} W_{i\xi}^{(h-1)} V_{\xi j}^{(c_h)},$$

and for $j = i + 1$, $B_{ij}^{(k)} = C_{ij}^{(h)} = 0$.

Now we show how to use equations (13) and (14) in order to obtain a single equation from which recover U_{ij} for each i and j. The construction of such equation is quite technical and will be done in the rest of the section.

Let $A_1 = \{(0; 1; 0)\}$, and let

$$A_k = \bigcup_{(r;s;t) \in A_{k-1}} \{(r + 2^{k-1}; s; t),\ (r; s; t + 2^{k-1})\} \cup \{(0; k; 0)\}, \tag{15}$$

for any integer $k > 1$.

The set A_k contains $2^k - 1$ triples; it can be easily shown that the following two properties completely describe A_k:

(i) $(0;k;0) \in A_k$ holds;
(ii) If $(r;s;t) \in A_k$, and $s > 1$, then $(r{+}2^{s-1};s{-}1;t) \in A_k$, and $(r;s{-}1;t{+}2^{s-1}) \in A_k$.

Property (i), and (ii) allow us to represent the elements in A_k by a tree. In Figure 1 the tree in the case $k = 3$ is depicted.

$$\begin{tikzcd}
 & & (6;1;0) \\
 & (4;2;0) \arrow[ur] \arrow[dr] & \\
 & & (4;1;2) \\
(0;3;0) \arrow[uur] \arrow[ddr] & & \\
 & & (2;1;4) \\
 & (0;2;4) \arrow[ur] \arrow[dr] & \\
 & & (0;1;6)
\end{tikzcd}$$

Fig. 1 A tree representation of A_3

We first explain how the blocks of U can be constructed when $p = 2^k$, then the general case is described. We recall that the algorithm will be used essentially only if p is a prime number, the case $p = 2^k$ is presented just for the sake of the clarity.

Let us illustrate what happens in the case $p = 16$ to better understand the case $p = 2^k$. Observe that for $p = 2^k$ one needs only equation (10). Let us suppose that the diagonal blocks U_{ii}^q for any $q = 0, \ldots, 15$ have been already computed. It follows from (13) applied for $k = 4$ that

$$V_{ij}^{(4)} = V_{ii}^{(3)} V_{ij}^{(3)} + V_{ij}^{(3)} V_{jj}^{(3)} + B_{ij}^{(4)}.$$

Note that $B_{ij}^{(4)} = U_{ii}^0 B_{ij}^{(4)} U_{jj}^0$, and it can be associated with the triple $(0;4;0)$ belonging to the first level in the tree corresponding to A_4. Let us consider the term $V_{ii}^{(3)} V_{ij}^{(3)}$, it follows from (13) applied for $k = 3$ that

$$V_{ii}^{(3)} V_{ij}^{(3)} = U_{ii}^8 \left(V_{ii}^{(2)} V_{ij}^{(2)} + V_{ij}^{(2)} V_{jj}^{(2)} \right) + U_{ii}^8 B_{ij}^{(3)}.$$

Note that $U_{ii}^8 B_{ij}^{(3)} = U_{ii}^8 B_{ij}^{(3)} U_{jj}^0$, and that it can be associated with the triple $(8;3;0)$ belonging to a part of the second level of the tree corresponding to A_4. Clearly, the triple $(0;3;8)$ appears when we substitute (13) for $k=3$ in $V_{ij}^{(3)}V_{jj}^{(3)}$. Recalling that $V_{ii}^{(2)}=U_{ii}^4$, and that $V_{jj}^{(2)}=U_{jj}^4$, we obtain by making use of (13) applied for $k=2$ that

$$\begin{aligned} U_{ii}^{12}V_{ij}^{(2)}+U_{ii}^8V_{ij}^{(2)}U_{jj}^4 =& U_{ii}^{12}\left(V_{ii}^{(1)}V_{ij}^{(1)}+V_{ij}^{(1)}V_{jj}^{(1)}\right)+U_{ii}^{12}B_{ij}^{(2)} \\ &+U_{ii}^8\left(V_{ii}^{(1)}V_{ij}^{(1)}+V_{ij}^{(1)}V_{jj}^{(1)}\right)U_{jj}^4+U_{ii}^8B_{ij}^{(2)}U_{jj}^4. \end{aligned}$$

By arguing as above, the two terms $U_{ii}^{12}B_{ij}^{(2)}$ and $U_{ii}^8B_{ij}^{(2)}U_{jj}^4$ are associated with the triples $(12;2;0)$, and $(8;2;4)$, respectively. The other two triples belonging to the third level of the tree corresponding to A_4 appear, for symmetry, when we simplify $V_{ij}^{(3)}V_{jj}^{(3)}$. Remembering that $V_{ii}^{(1)}=U_{ii}^2$, and that $V_{jj}^{(1)}=U_{jj}^2$, we obtain by making use of (13) for $k=1$ that

$$\begin{aligned} U_{ii}^{14}V_{ij}^{(1)} &= U_{ii}^{15}V_{ij}^{(0)}+U_{ii}^{14}V_{ij}^{(0)}U_{jj}+U_{ii}^{14}B_{ij}^{(1)}, \\ U_{ii}^{12}V_{ij}^{(1)}U_{jj}^2 &= U_{ii}^{13}V_{ij}^{(0)}U_{jj}^2+U_{ii}^{12}V_{ij}^{(0)}U_{jj}^3+U_{ii}^{12}B_{ij}^{(1)}U_{jj}^2, \\ U_{ii}^{10}V_{ij}^{(1)}U_{jj}^4 &= U_{ii}^{11}V_{ij}^{(0)}U_{jj}^4+U_{ii}^{10}V_{ij}^{(0)}U_{jj}^5+U_{ii}^{10}B_{ij}^{(1)}U_{jj}^4, \\ U_{ii}^{8}V_{ij}^{(1)}U_{jj}^6 &= U_{ii}^{9}V_{ij}^{(0)}U_{jj}^6+U_{ii}^{8}V_{ij}^{(0)}U_{jj}^7+U_{ii}^{8}B_{ij}^{(1)}U_{jj}^6, \end{aligned}$$

which make appear the missing terms in A_4. In general, it holds the following result.

Lemma 1 *If $p=2^{c_0}$, for some positive integer c_0, then*

$$R_{ij}=\sum_{q=0}^{p-1}U_{ii}^qV_{ij}^{(0)}U_{jj}^{p-1-q}+\sum_{(r;s;t)\in A_{c_0}}U_{ii}^rB_{ij}^{(s)}U_{jj}^t. \tag{16}$$

Proof The claim is proved by induction on c_0. Let us suppose $c_0=1$, from (13) applied for $k=1$ it follows that

$$R_{ij}=V_{ij}^{(1)}=V_{ii}^{(0)}V_{ij}^{(0)}+V_{ij}^{(0)}V_{jj}^{(0)}+B_{ij}^{(1)}.$$

Since $V_{ii}^{(0)}=U_{ii}$, $V_{jj}^{(0)}=U_{jj}$, and $A_1=\{(0;1;0)\}$, the claim trivially follows.

Let us assume the claim for $c_0=c>0$, and let us prove it for $c_0=c+1$. From (13) applied for $k=c+1$ it follows that

$$R_{ij}=V_{ij}^{(c+1)}=V_{ii}^{(c)}V_{ij}^{(c)}+V_{ij}^{(c)}V_{jj}^{(c)}+B_{ij}^{(c+1)}.$$

By making use of inductive hypothesis and observing that $V_{ii}^{(c)}=U_{ii}^{2^c}$, and $V_{jj}^{(c)}=U_{jj}^{2^c}$, the above equation can be written as

$$\begin{aligned} R_{ij} &= U_{ii}^{2^c}\left(\sum_{q=0}^{2^c-1}U_{ii}^qV_{ij}^{(0)}U_{jj}^{2^c-1-q}+\sum_{(r;s;t)\in A_c}U_{ii}^rB_{ij}^{(s)}U_{jj}^t\right) \\ &+\left(\sum_{q=0}^{2^c-1}U_{ii}^qV_{ij}^{(0)}U_{jj}^{2^c-1-q}+\sum_{(r;s;t)\in A_c}U_{ii}^rB_{ij}^{(s)}U_{jj}^t\right)U_{jj}^{2^c}+B_{ij}^{(c+1)} \\ &= \sum_{q=0}^{2^c-1}\left(U_{ii}^{q+2^c}V_{ij}^{(0)}U_{jj}^{2^c-1-q}+U_{ii}^qV_{ij}^{(0)}U_{jj}^{2^c-1-q+2^c}\right) \\ &+\sum_{(r;s;t)\in A_c}\left(U_{ii}^{r+2^c}B_{ij}^{(s)}U_{jj}^t+U_{ii}^rB_{ij}^{(s)}U_{jj}^{t+2^c}\right)+U_{ii}^0B_{ij}^{(c+1)}U_{jj}^0. \end{aligned}$$

The former term of the last expression can be written as $\sum_{q=0}^{2^{c+1}-1} U_{ii}^q V_{ij}^{(0)} U_{jj}^{2^{c+1}-1-q}$, while the latter term of the last expression can be rearranged as $\sum_{(r;s;t)\in A_{c+1}} U_{ii}^r B_{ij}^{(s)} U_{jj}^t$ as a consequence of the definition of A_k for $k = c+1$. The claim thus follows.

Note that the two sums involved in R_{ij} have p, and $2^{c_0} - 1 = p - 1$ terms, respectively.

Lemma 1 provides a basis for an algorithm for the 2^kth root of a matrix. We need the use of the Kronecker notation [10], that is the Kronecker product, the vec operator which stacks the columns of a matrix in a long vector and the well-know relation $\operatorname{vec}(AXB) = (B^T \otimes A)\operatorname{vec}(X)$, for A, X, B matrices of suitable sizes.

Using the Kronecker notation and $V_{ij}^{(0)} = U_{ij}$, equation (16) can be rewritten as

$$\left(\sum_{q=0}^{p-1} \left(U_{jj}^{p-1-q}\right)^T \otimes U_{ii}^q\right) \operatorname{vec}(U_{ij}) = \operatorname{vec}\left(R_{ij} - \sum_{(r;s;t)\in A_{c_0}} U_{ii}^r B_{ij}^{(s)} U_{jj}^t\right), \qquad (17)$$

which is a linear system of size at most 4, whose unknown is $\operatorname{vec}(U_{ij})$, and the matrix coefficient and the right hand side are known quantities since they involve already computed blocks. The solution is unique as the matrix coefficient is the transpose of the one appearing in the Smith algorithm which is proved to be nonsingular [18].

In order to go further to the case in which p is arbitrary, let us illustrate what happens for $p = 23$, where $m = 3$, $c_0 = 4$, $c_1 = 2$, $c_2 = 1$, $c_3 = 0$. By making use of (14) for $k = 3$, we have that

$$W_{ij}^{(3)} = W_{ii}^{(2)} V_{ij}^{(0)} + W_{ij}^{(2)} V_{jj}^{(0)} + C_{ij}^{(3)} = U_{ii}^{22} V_{ij}^{(0)} + W_{ij}^{(2)} U_{jj} + C_{ij}^{(3)},$$

where we have used that $W_{ii}^{(2)} = U_{ii}^{2^{c_0}+2^{c_1}+2^{c_2}} = U_{ii}^{22}$, and that $V_{jj}^{(0)} = U_{jj}$. The only summand which needs to be further reduced is the second one; according to (14), for $k = 2$ we have that

$$\begin{aligned} W_{ij}^{(2)} U_{jj} &= (W_{ii}^{(1)} V_{ij}^{(1)} + W_{ij}^{(1)} V_{jj}^{(1)}) U_{jj} + C_{ij}^{(2)} U_{jj} \\ &= U_{ii}^{20} V_{ij}^{(1)} U_{jj} + W_{ij}^{(1)} U_{jj}^3 + C_{ij}^{(2)} U_{jj}, \end{aligned}$$

where we have used that $W_{ii}^{(1)} = U_{ii}^{2^{c_0}+2^{c_1}} = U_{ii}^{20}$, and $V_{jj}^{(1)} = U_{jj}^2$. Moreover, it follows from Lemma 1 that

$$V_{ij}^{(1)} = \sum_{q=0}^{1} U_{ii}^q V_{ij}^{(0)} U_{jj}^{1-q} + \sum_{(r;s;t)\in A_1} U_{ii}^r B_{ij}^{(s)} U_{jj}^t,$$

hence,

$$U_{ii}^{20} V_{ij}^{(1)} U_{jj} = U_{ii}^{20} \left(\sum_{q=0}^{1} U_{ii}^q V_{ij}^{(0)} U_{jj}^{1-q}\right) U_{jj} + U_{ii}^{20} \left(\sum_{(r;s;t)\in A_1} U_{ii}^r B_{ij}^{(s)} U_{jj}^t\right) U_{jj}.$$

According to (14), for $k = 1$, we have that

$$\begin{aligned} W_{ij}^{(1)} U_{jj}^{3} &= W_{ii}^{(0)} V_{ij}^{(2)} U_{jj}^{3} + W_{ij}^{(0)} V_{jj}^{(2)} U_{jj}^{3} + C_{ij}^{(1)} U_{jj}^{3} \\ &= U_{ii}^{16} V_{ij}^{(2)} U_{jj}^{3} + W_{ij}^{(0)} U_{jj}^{7} + C_{ij}^{(1)} U_{jj}^{3}. \end{aligned}$$

Note that $W_{ii}^{(0)} = U_{ii}^{2^{c_0}} = U_{ii}^{16}$, and that $V_{jj}^{(2)} = U_{jj}^{4}$. Moreover, from Lemma 1 it follows that

$$V_{ij}^{(2)} = \sum_{q=0}^{3} U_{ii}^{q} V_{ij}^{(0)} U_{jj}^{3-q} + \sum_{(r;s;t)\in A_2} U_{ii}^{r} B_{ij}^{(s)} U_{jj}^{t},$$

hence,

$$U_{ii}^{16} V_{ij}^{(2)} U_{jj}^{3} = U_{ii}^{16} \left(\sum_{q=0}^{3} U_{ii}^{q} V_{ij}^{(0)} U_{jj}^{3-q} \right) U_{jj}^{3} + U_{ii}^{16} \left(\sum_{(r;s;t)\in A_2} U_{ii}^{r} B_{ij}^{(s)} U_{jj}^{t} \right) U_{jj}^{3}.$$

On the other hand,

$$W_{ij}^{(0)} U_{jj}^{7} = V_{ij}^{(4)} U_{jj}^{7} = \left(\sum_{q=0}^{15} U_{ii}^{q} V_{ij}^{(0)} U_{jj}^{15-q} \right) U_{jj}^{7} + \left(\sum_{(r;s;t)\in A_4} U_{ii}^{r} B_{ij}^{(s)} U_{jj}^{t} \right) U_{jj}^{7},$$

by making use of Lemma 1 for $c_0 = 4$, and of (14) for $k = 0$.

All terms involving $V_{ij}^{(0)}$ can be grouped as follows

$$\sum_{q=0}^{22} U_{ii}^{q} V_{ij}^{(0)} U_{jj}^{22-q},$$

while the remaining terms can be divided into two summands. The first one containing $C_{ij}^{(h)}$, $i, j = 1, \ldots, \sigma$, can be written as

$$\sum_{h=1}^{3} C_{ij}^{(h)} U_{jj}^{2^{c_{h+1}} + \cdots + 2^{c_3}},$$

where $U_{jj}^{2^{c_{h+1}} + \cdots + 2^{c_3}}$ denotes the identity matrix for $h = 3$.

The second one referring to the $B_{ij}^{(k)}$'s, can be written as

$$\sum_{h\in c(23)^{+}} U_{ii}^{23-2^{c_h}-\cdots-2^{c_3}} \left[\sum_{(r;s;t)\in A_{c_h}} U_{ii}^{r} B_{ij}^{(s)} U_{jj}^{t} \right] U_{jj}^{2^{c_{h+1}} + \cdots + 2^{c_3}},$$

where $c(23)^{+}$ is the set $\{4, 2, 1\}$, according to definition (8).

Now we give the main result of this section.

Theorem 1 *Let* $p = \sum_{h=0}^{m} 2^{c_h}$ *be a positive integer greater than* 1 *and* $c(p)^+$ *as in* (8). *Then*

$$\begin{aligned} R_{ij} &= \sum_{q=0}^{p-1} U_{ii}^{q} V_{ij}^{(0)} U_{jj}^{p-1-q} + \sum_{h=1}^{m} C_{ij}^{(h)} U_{jj}^{2^{c_{h+1}}+\cdots+2^{c_m}} \\ &+ \sum_{h\in c(p)^+} U_{ii}^{p-2^{c_h}-\cdots-2^{c_m}} \left[\sum_{(r;s;t)\in A_{c_h}} U_{ii}^{r} B_{ij}^{(s)} U_{jj}^{t} \right] U_{jj}^{2^{c_{h+1}}+\cdots+2^{c_m}}, \end{aligned} \tag{18}$$

where, for $h = m$, $U_{jj}^{2^{c_{h+1}}+\cdots+2^{c_m}}$ *denotes the identity matrix, for* $m = 0$ *the second summand on the right hand side of* (18) *is the zero matrix and the third summand on the right hand side of* (18) *is the zero matrix when* $c(p)^+$ *is the empty set.*

Proof The claim is done by induction on m. Let us assume $m = 0$, hence, $p = 2^{c_0}$, for some positive integer c_0. The claim thus follows from Lemma 1.

Let us assume the claim for $m = \mu$, and let us prove it for $m = \mu + 1$, note that in this case $p = \sum_{h=0}^{\mu} 2^{c_h} + 2^{c_{\mu+1}}$. Let p' denote the integer $\sum_{h=0}^{\mu} 2^{c_h}$, thus $p = p' + 2^{c_{\mu+1}}$.

It follows from (11) applied for $h = \mu + 1$ that

$$R_{ij} = W_{ij}^{(\mu+1)} = W_{ii}^{(\mu)} V_{ij}^{(c_{\mu+1})} + W_{ij}^{(\mu)} V_{jj}^{(c_{\mu+1})} + C_{ij}^{(\mu+1)}.$$

Note that $W_{ii}^{(\mu)} = U_{ii}^{p'}$, and that $V_{jj}^{(c_{\mu+1})} = U_{jj}^{2^{c_{\mu+1}}}$. By making use of the induction hypothesis for $W_{ij}^{(\mu)}$,

$$\begin{aligned} R_{ij} &= U_{ii}^{p'} V_{ij}^{(c_{\mu+1})} + \Bigg(\sum_{q=0}^{p'-1} U_{ii}^{q} V_{ij}^{(0)} U_{jj}^{p'-1-q} + \sum_{h=1}^{\mu} C_{ij}^{(h)} U_{jj}^{2^{c_{h+1}}+\cdots+2^{c_\mu}} \\ &+ \sum_{h\in c(p')^+} U_{ii}^{p'-2^{c_h}-\cdots-2^{c_\mu}} \left[\sum_{(r;s;t)\in A_{c_h}} U_{ii}^{r} B_{ij}^{(s)} U_{jj}^{t} \right] U_{jj}^{2^{c_{h+1}}+\cdots+2^{c_\mu}} \Bigg) U_{jj}^{2^{c_{\mu+1}}} + C_{ij}^{(\mu+1)}. \end{aligned}$$

As a consequence of the relation $p = p' + 2^{c_{\mu+1}}$, we have that

$$\begin{aligned} R_{ij} &= U_{ii}^{p'} V_{ij}^{(c_{\mu+1})} + \sum_{q=0}^{p'-1} U_{ii}^{q} V_{ij}^{(0)} U_{jj}^{p-1-q} + \sum_{h=1}^{\mu} C_{ij}^{(h)} U_{jj}^{2^{c_{h+1}}+\cdots+2^{c_\mu}+2^{c_{\mu+1}}} + C_{ij}^{(\mu+1)} \\ &+ \sum_{h\in c(p')^+} U_{ii}^{p-2^{c_h}-\cdots-2^{c_\mu}-2^{c_{\mu+1}}} \left[\sum_{(r;s;t)\in A_{c_h}} U_{ii}^{r} B_{ij}^{(s)} U_{jj}^{t} \right] U_{jj}^{2^{c_{h+1}}+\cdots+2^{c_\mu}+2^{c_{\mu+1}}}. \end{aligned}$$

As $\mu + 1 = m$, $U_{jj}^{2^{c_{\mu+2}}+\cdots+2^{c_{\mu+1}}}$ denotes the identity matrix. Hence, the term $C[p] := \sum_{h=1}^{\mu} C_{ij}^{(h)} U_{jj}^{2^{c_{h+1}}+\cdots+2^{c_{\mu+1}}} + C_{ij}^{(\mu+1)}$ is equal to the second sum in the right hand side of equation (18) for $m = \mu + 1$.

In order to complete the proof, we distinguish two cases: $c_{\mu+1} = 0$ and $c_{\mu+1} > 0$ which correspond to p odd and p even, respectively.

If $c_{\mu+1} = 0$, then $V_{ij}^{(c_{\mu+1})} = V_{ij}^{(0)} = U_{ij}$, and $U_{jj}^{2^{c_{\mu+1}}} = U_{jj}$ hold, hence,

$$R_{ij} = U_{ii}^{p'} V_{ij}^{(0)} + \sum_{q=0}^{p'-1} U_{ii}^{q} V_{ij}^{(0)} U_{jj}^{p-1-q} + C[p] \tag{19}$$
$$+ \sum_{h \in c(p')^+} U_{ii}^{p-2^{c_h}-\cdots-2^{c_\mu}-2^{c_{\mu+1}}} \left[\sum_{(r;s;t)\in A_{c_h}} U_{ii}^{r} B_{ij}^{(s)} U_{jj}^{t} \right] U_{jj}^{2^{c_{h+1}}+\cdots+2^{c_\mu}+2^{c_{\mu+1}}}.$$

Since $p = p' + 1$, the first two summands of the right hand side of (19) can be written as $\sum_{q=0}^{p-1} U_{ii}^{q} V_{ij}^{(0)} U_{jj}^{p-1-q}$, which corresponds to the first sum in the the right hand side of equation (18) for $m = \mu + 1$.

Finally, noting that $c(p)^+ = c(p')^+$, the second row in equation (19) corresponds to the third sum in the the right hand side of equation (18) for $m = \mu + 1$.
The claim thus follows in the case $c_{\mu+1} = 0$.

Suppose, now, that $c_{\mu+1} > 0$. Let us compute $V_{ij}^{(c_{\mu+1})}$ using Lemma 1 for $c = c_{\mu+1}$. Hence,

$$R_{ij} = U_{ii}^{p'} \left(\sum_{q=0}^{2^{c_{\mu+1}}-1} U_{ii}^{q} V_{ij}^{(0)} U_{jj}^{2^{c_{\mu+1}}-1-q} + \sum_{(r;s;t)\in A_{c_{\mu+1}}} U_{ii}^{r} B_{ij}^{(s)} U_{jj}^{t} \right)$$
$$+ \sum_{q=0}^{p'-1} U_{ii}^{q} V_{ij}^{(0)} U_{jj}^{p-1-q} + C[p]$$
$$+ \sum_{h \in c(p')^+} U_{ii}^{p-2^{c_h}-\cdots-2^{c_\mu}-2^{c_{\mu+1}}} \left[\sum_{(r;s;t)\in A_{c_h}} U_{ii}^{r} B_{ij}^{(s)} U_{jj}^{t} \right] U_{jj}^{2^{c_{h+1}}+\cdots+2^{c_\mu}+2^{c_{\mu+1}}}.$$

Since $p = p' + 2^{c_{\mu+1}}$,

$$U_{ii}^{p'} \sum_{q=0}^{2^{c_{\mu+1}}-1} U_{ii}^{q} V_{ij}^{(0)} U_{jj}^{2^{c_{\mu+1}}-1-q} + \sum_{q=0}^{p'-1} U_{ii}^{q} V_{ij}^{(0)} U_{jj}^{p-1-q} = \sum_{q=0}^{p-1} U_{ii}^{q} V_{ij}^{(0)} U_{jj}^{p-1-q},$$

thus, the first sum in the the right hand side of equation (18) for $m = \mu + 1$ has been obtained. Moreover,

$$U_{ii}^{p'} \sum_{(r;s;t)\in A_{c_{\mu+1}}} U_{ii}^{r} B_{ij}^{(s)} U_{jj}^{t} = U_{ii}^{p-2^{c_{\mu+1}}} \left(\sum_{(r;s;t)\in A_{c_{\mu+1}}} U_{ii}^{r} B_{ij}^{(s)} U_{jj}^{t} \right) U_{jj}^{0}$$

Noting that $c(p)^+ = c(p')^+ \cup \{\mu + 1\}$, the remaining terms can be written as

$$\sum_{h \in c(p)^+} U_{ii}^{p-2^{c_h}-\cdots-2^{c_\mu}-2^{c_{\mu+1}}} \left[\sum_{(r;s;t)\in A_{c_h}} U_{ii}^{r} B_{ij}^{(s)} U_{jj}^{t} \right] U_{jj}^{2^{c_{h+1}}+\cdots+2^{c_\mu}+2^{c_{\mu+1}}}.$$

The proof is completed.

Note that the first sums involved in R_{ij} according to Theorem 1 has p summands. The second one has m summands, and the third one has $\sum_{h\in c(p)^+}(2^{c_h}-1)$ terms. In particular,

$$m + \sum_{h\in c(p)^+} (2^{c_h}-1) \;=\; p-1$$

in both cases $c_m = 0$, and $c_m > 0$.

3 The algorithm

We summarize the algorithm for computing the principal pth root of a real matrix having no nonpositive real eigenvalues.

Algorithm 1 (Binary powering Schur algorithm for the principal pth root of a real matrix A)

```
1.  compute a real Schur decomposition $A = QRQ^T$, where $R$ is block $\sigma \times \sigma$
2.  compute $b_0, \dots, b_{\lfloor \log_2 p \rfloor}$ and $c_0, \dots, c_m$ in the binary decomposition of $p$ as in (7)
    and (9)
3.  for $j = 1 : \sigma$
4.      compute $U_{jj} = R_{jj}^{1/p}$ (using (6) if the size of $U_{jj}$ is 2)
5.      for $q = 0 : p-1$ compute $D_j^{(q)} = U_{jj}^q$, end
6.      for $k = 0 : c_0$ set $V_{jj}^{(k)} = U_{jj}^{2^k}$, end
7.      $W_{jj}^{(0)} = V_{jj}^{(c_0)}$
8.      for $h = 1 : m$ set $W_{jj}^{(h)} = W_{jj}^{(h-1)} V_{jj}^{(c_h)}$, end
9.      for $i = j-1 : -1 : 1$
10.         for $k = 1 : c_0$
11.             $B_k = \sum_{\ell=i+1}^{j-1} V_{i\xi}^{(k-1)} V_{\xi j}^{(k-1)}$
12.         end
13.         for $h = 1 : m$
14.             $C_h = \sum_{\ell=i+1}^{j-1} W_{i\xi}^{(h)} V_{\xi j}^{(c_h)}$
15.         end
16.         solve $\sum_{q=0}^{p-1} D_i^{(q)} U_{ij} D_j^{(p-q-1)} = R_{ij} - \sum_{h=1}^{m} C_h D_j^{(2^{c_{h+1}}+\cdots+2^{c_m})}$
             $-\sum_{h\in c(p)^+} D_i^{(p-2^{c_h}-\cdots-2^{c_m})} \left[\sum_{(r;s;t)\in A_{c_h}} D_i^{(r)} B_s D_j^{(t)}\right] D_j^{(2^{c_{h+1}}+\cdots+2^{c_m})}$
            with respect to $U_{ij}$
17.         $V_{ij}^{(0)} = U_{ij}$
18.         for $k = 1 : c_0$
19.             $V_{ij}^{(k)} = B_k + V_{ii}^{(k-1)} V_{ij}^{(k-1)} + V_{ij}^{(k-1)} V_{jj}^{(k-1)}$
20.         end
21.         $W_{ij}^{(0)} = V_{ij}^{(c_0)}$
22.         for $h = 1 : m$
23.             $W_{ij}^{(h)} = C_h + W_{ii}^{(h-1)} V_{ij}^{(c_h)} + W_{ij}^{(h-1)} V_{jj}^{(c_h)}$
24.         end
25.     end
26. end
27. compute $A^{1/p} = Q^T U Q$.
```

In Steps 11, 14 and 16, we assume that a void sum is the zero matrix, while in Step 16 we assume that given a matrix M, $M^{2^{c_{h+1}}+\cdots+2^{c_m}}$ is the identity matrix for $h = m$.

Let us analyze the computational cost of Algorithm 1. We can assume that $\sigma = O(n)$, $c_0 = O(\log_2 p)$ and $m = O(\log_2 p)$. Step 5 requires the computation of p powers of s blocks of size at most 2, the cost is $O(np)$ ops. Steps 6–8 are obtained with no more cost. Steps 10–12 require c_0 sums from $i+1$ to $j-1$ for each $i < j-1$, the resulting cost is $O(n^3 \log_2 p)$ ops, the same cost is required for Steps 13–15. Forming the coefficients and solving the equation at Step 16 requires $O(n^2 p)$ ops, since the sum on the right hand side contains no more than $2 \log_2 p$ terms. Finally, the cost of Steps 18–20 and 22–24 is $O(n^2 \log_2 p)$.

In summary the cost of the algorithm is $O(n^2 p + n^3 \log_2 p)$ ops which asymptotically favorably compares to the Smith method whose cost is $O(n^3 p)$ ops. Algorithm 1 requires less operations also for small p or n and this leads to a faster computation as we will show in Section 5. The cost could be further lowered as suggested in Section 4.

Consider now the cost in memory. The main expenses are due: to the storage of $V^{(k)}$, $W^{(h)}$ which are $O(\log_2 p)$ $n \times n$ matrices for a total of $O(n^2 \log_2 p)$ real numbers; to the storage of the block diagonal of U^q, namely, the blocks $D_j^{(q)}$, where $j = 1, \ldots, \sigma$ and $q = 0, \ldots, p-1$ for a total of $O(np)$ real numbers.

In summary the algorithm requires the storage of $O(np + n^2 \log_2 p)$ real numbers.

Algorithm 1 can be slightly modified to work with the complex Schur form as well, in that case one gets the principal pth root of a complex matrix.

More generally the algorithm can be used to compute any primary pth root of a nonsingular matrix A, by choosing for each eigenvalue the desired pth root at Step 4, with the restriction that the same branch of the pth root function must be chosen for repeated eigenvalues.

If two different branches of the pth root are chosen for the same eigenvalue appearing in two different blocks, then the linear matrix equation at Step 18 admits no unique solution, and Algorithm 1 fails. However, in that case the resulting pth root would be nonprimary.

4 Possible further improvements

Algorithm 1 has a computational cost which is $O(n^2 p + n^3 \log_2 p)$ ops and needs the storage of $O(np + n^2 \log_2 p)$ real numbers. The linear dependence on p is bothering since the algorithms for the matrix pth root based on matrix iterations depend only on the logarithm of p. It is possible to reduce further the computational cost of Algorithm 1.

The storage of $O(np)$ real numbers is due to the need of all the powers of U_{ii}, say U_{ii}^q, for $i = 1, \ldots, \sigma$ and $q = 1, \ldots, p$.

The computational cost of $O(n^2 p)$ ops is due to the solution of the matrix equations

$$\begin{aligned}\sum_{q=0}^{p-1} U_{ii}^q U_{ij} U_{jj}^{p-1-q} = R_{ij} &- \sum_{h=1}^{m} C_{ij}^{(h)} U_{jj}^{2^{c_{h+1}}+\cdots+2^{c_m}} \qquad (20)\\ &- \sum_{h \in c(p)^+} U_{ii}^{p-2^{c_h}-\cdots-2^{c_m}} \left[\sum_{(r;s;t) \in A_{c_h}} U_{ii}^r B_{ij}^{(s)} U_{jj}^t \right] U_{jj}^{2^{c_{h+1}}+\cdots+2^{c_m}},\end{aligned}$$

with respect to U_{ij}, for each i and j.

We will explain how to reduce these costs for fixed i and j. First, we compute and store λ^k for any eigenvalue λ of U, and for

$$\begin{aligned} &k = 2, 4, \ldots, 2^{c_0}, \\ &k = p - 2^{c_h} - \cdots - 2^{c_m}, \quad h = 1, \ldots, m, \\ &k = 2^{c_{h+1}} + \cdots + 2^{c_m}, \quad h = 1, \ldots, m-1, \end{aligned}$$

for a total amount of $O(\log_2 p)$ values of k.

Then, observe that if R_{ii} is a scalar μ_i, then $U_{ii} = \mu_i^{1/p} =: \lambda_i$, thus, $U_{ii}^k = \lambda_i^k$; if R_{ii} is a 2×2 real matrix corresponding to the couple of complex eigenvalues $\theta \pm \boldsymbol{i}\mu$, then its principal pth root U_{ii} is obtained from the principal pth root of the scalar $\theta + \boldsymbol{i}\mu$ that is $\alpha + \boldsymbol{i}\beta$ by formula (6). In a similar manner if $\alpha^{(k)} + \boldsymbol{i}\beta^{(k)} := (\alpha + \boldsymbol{i}\beta)^k$, then it is easy to see that

$$U_{ii}^k = \alpha^{(k)} I + \frac{\beta^{(k)}}{\mu}(R_{ii} - \theta I). \tag{21}$$

Now, we can proceed in removing the linear term in p in the asymptotic costs. First, we explain how to construct the matrix coefficient

$$M_{ij} := \sum_{q=0}^{p-1} \left(U_{jj}^{p-q-1}\right)^T \otimes U_{ii}^q$$

with $O(\log_2 p)$ ops.

Let $\lambda_i = \theta_i + \boldsymbol{i}\mu_i$ be one of the two eigenvalues of U_{ii} and let $\lambda_i^q = \alpha_i^{(q)} + \boldsymbol{i}\beta_i^{(q)}$, for $q = 1, \ldots, p-1$, be the corresponding eigenvalue of U_{ii}^q, then using (21) the matrix coefficient becomes

$$\begin{aligned} M_{ij} = &\left(\sum_{q=0}^{p-1} \alpha_j^{(p-q-1)} \alpha_i^{(q)}\right) I + \left(\sum_{q=0}^{p-1} \beta_j^{(p-q-1)} \alpha_i^{(q)}\right) \frac{(R_{jj} - \theta_j I)^T \otimes I}{\mu_j} \\ &+ \left(\sum_{q=0}^{p-1} \alpha_j^{(p-q-1)} \beta_i^{(q)}\right) \frac{I \otimes (R_{ii} - \theta_i I)}{\mu_i} + \left(\sum_{q=0}^{p-1} \beta_j^{(p-q-1)} \beta_i^{(q)}\right) \frac{(R_{jj} - \theta_j I)^T \otimes (R_{ii} - \theta_i I)}{\mu_i \mu_j}, \end{aligned}$$

where, for $\lambda_i \neq \lambda_j$,

$$\begin{aligned} \sum_{q=0}^{p-1} \alpha_j^{(p-q-1)} \alpha_i^{(q)} &= \frac{1}{2}\left(\operatorname{Re}\left(\frac{\lambda_i^p - \lambda_j^p}{\lambda_i - \lambda_j}\right) + \operatorname{Re}\left(\frac{\lambda_i^p - \overline{\lambda}_j^p}{\lambda_i - \overline{\lambda}_j}\right)\right), \\ \sum_{q=0}^{p-1} \beta_j^{(p-q-1)} \alpha_i^{(q)} &= \frac{1}{2}\left(\operatorname{Im}\left(\frac{\lambda_i^p - \lambda_j^p}{\lambda_i - \lambda_j}\right) + \operatorname{Im}\left(\frac{\lambda_i^p - \overline{\lambda}_j^p}{\lambda_i - \overline{\lambda}_j}\right)\right), \\ \sum_{q=0}^{p-1} \alpha_j^{(p-q-1)} \beta_i^{(q)} &= \frac{1}{2}\left(\operatorname{Im}\left(\frac{\lambda_i^p - \lambda_j^p}{\lambda_i - \lambda_j}\right) - \operatorname{Im}\left(\frac{\lambda_i^p - \overline{\lambda}_j^p}{\lambda_i - \overline{\lambda}_j}\right)\right), \\ \sum_{q=0}^{p-1} \beta_j^{(p-q-1)} \beta_i^{(q)} &= \frac{1}{2}\left(\operatorname{Re}\left(\frac{\lambda_i^p - \lambda_j^p}{\lambda_i - \lambda_j}\right) - \operatorname{Re}\left(\frac{\lambda_i^p - \overline{\lambda}_j^p}{\lambda_i - \overline{\lambda}_j}\right)\right), \end{aligned}$$

while for $\lambda_i = \lambda_j$ it holds that $\left(\frac{\lambda_i^p - \lambda_j^p}{\lambda_i - \lambda_j}\right) = p\lambda_i^{p-1}$.

Thus, for computing M_{ij} one needs just the pth power of the eigenvalues of A, which have been already computed, and then performing a fixed number of arithmetic operations.

The second summand on the right hand side of (20) is a sum of $m = O(\log_2 p)$ terms. It can be computed with $O(\log_2 p)$ ops, since $\lambda_j^{2^{c_{h+1}}+\cdots+2^{c_m}}$ and, in view of formula (21), $U_{jj}^{2^{c_{h+1}}+\cdots+2^{c_m}}$ are known for each $h = 1, \ldots, m$.

Finally, we discuss how to compute in $O(\log_2^2 p)$ the last summand on right hand side of equation (20), that is,

$$\sum_{h \in c(p)^+} U_{ii}^{p-2^{c_h}-\cdots-2^{c_m}} \left[\sum_{(r;s;t) \in A_{c_h}} U_{ii}^r B_{ij}^{(s)} U_{jj}^t \right] U_{jj}^{2^{c_{h+1}}+\cdots+2^{c_m}}. \tag{22}$$

The cardinality of $c(p)^+$ is $O(\log_2 p)$, so, in order to obtain a total cost of $O(\log_2^2 p)$ ops we need to compute the sum $\sum_{(r;s;t)\in A_{c_h}} U_{ii}^r B_{ij}^{(s)} U_{jj}^t$ in $O(\log_2 p)$ ops and the premultiplication by $U_{ii}^{p-2^{c_h}-\cdots-2^{c_m}}$ and the post-multiplication by $U_{jj}^{2^{c_{h+1}}+\cdots+2^{c_m}}$ in $O(1)$ ops. The latter two tasks follow from the fact that we already know $\lambda_i^{p-2^{c_h}-\cdots-2^{c_m}}$ and $\lambda_j^{2^{c_{h+1}}+\cdots+2^{c_m}}$ and from the use of (21).

To conclude, we rewrite $\sum_{(r;s;t)\in A_{c_h}} U_{ii}^r B_{ij}^{(s)} U_{jj}^t$ in the equivalent form

$$\sum_{s=1}^{c_h} \Big(\sum_{\substack{r,t \\ (r,s,t)\in A_{c_h}}} (U_{jj}^t)^T \otimes U_{ii}^r \Big) \operatorname{vec}(B_{ij}^{(s)}), \tag{23}$$

where the matrix $\sum (U_{jj}^t)^T \otimes U_{ii}^r$ is computed by a trick similar to the one used for M_{ij}, by using only the known values of λ_i^k and λ_j^k.

For instance for A_3 (compare Figure 1) one must compute the matrices

$$\begin{aligned} (U_{jj}^6)^T \otimes I + (U_{jj}^4)^T \otimes U_{ii}^2 + (U_{jj}^2)^T \otimes U_{ii}^4 + I \otimes U_{ii}^2, \\ (U_{jj}^4)^T \otimes I + I \otimes U_{ii}^4. \end{aligned}$$

If the matrices are 1×1, i.e. $U_{ii} = \lambda_i$, $U_{jj} = \lambda_j$ and $\lambda_i \neq \lambda_j$, then the computation is reduced to

$$\sum_{q=1}^{p/2-1} \lambda_j^{2(p-q-1)} \lambda_i^{2q} = \frac{\lambda_i^p - \lambda_j^p}{\lambda_i^2 - \lambda_j^2} \quad \text{and} \quad \lambda_j^4 + \lambda_i^4.$$

A drawback of this approach is that it is based on the simplification

$$\sum_{q=0}^{p-1} \lambda_i^q \lambda_j^{p-q-1} = \frac{\lambda_i^p - \lambda_j^p}{\lambda_i - \lambda_j},$$

computing the right hand side requires a lower computational cost, but is less numerically stable.

An open problem is the possibility to rearrange these ideas in a way such that the resulting algorithm is stable.

5 Numerical experiments

The analysis of the cost of Algorithm 1 of Section 3 both in terms of arithmetic operations and storage shows that it is asymptotically less expensive than the method proposed by Smith. We show by some numerical tests that in practice Algorithm 1 is faster than the one of Smith also for moderate values of p, moreover, the two algorithms reach the same numerical accuracy. For small p such as 2 or 3 the new algorithm does not give any advantage with respect to the one of Smith, on the contrary the latter in most cases is a bit faster in terms of CPU time.

The tests are performed on MATLAB 6, with unit roundoff $2^{-53} \approx 1.1 \times 10^{-16}$, where for the Smith method the implementation `rootpm_real` of Higham's Matrix Function Toolbox [5] is used and for the new algorithm the implementation can be found at [20].

We compare the performance of the two algorithms on some test matrices. In particular the CPU time required for the execution of the two algorithms is computed and the accuracy is estimated in terms of the quantity

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

where $\widetilde{X}$ is the computed pth root of A and $\|\cdot\|$ is any matrix norm (in our tests we used the Frobenius norm denoted by $\|\cdot\|_F$). In [8], the quantity ρ_A is proved to be a measure of accuracy more realistic than the norm of the relative residual, say $\|\widetilde{X}^p - A\|/\|A\|$. To better describe the numerical properties of the methods we also compute the quantity $\beta(U) = \|U\|_2^p/\|R\|_2$, where U is the computed root of the (quasi) triangular matrix R from the Schur decomposition of A, this quantity has been introduced in [19] as a measure of stability.

The results are summarized in Table 1, where n is the size of the matrices and "time" is the CPU time (in seconds) computed by MATLAB. If not otherwise stated we always compute the principal pth root.

Test 1 We consider the quasi upper triangular matrix

$$A = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 0 & 2 & 1 & 1 \\ 0 & 0 & 1 & -1 \\ 0 & 0 & 1 & 1 \end{bmatrix},$$

and compute its principal pth root for some values of p. Since the difference between Smith's algorithm and Algorithm 1 is the recursion used to compute the pth root of a (quasi) triangular matrix, the test is suitable to compare the accuracy and the CPU time of the two algorithms.

Test 2 We consider a 8×8 random stochastic matrix having no nonpositive real eigenvalues, which may be assumed to be the transition matrix relative to a period of one year in a Markov model [8,9]. If one needs the transition matrix for one day, then a 365th root of A is required. Observe that $365 = 73 \cdot 5$ so it is enough to compute the 73th root followed by the 5th root. The average speedup of computing the 73th root of A with Algorithm 1 with respect to the one of Smith is 9, while the residual ρ_A is essentially the same. For large p, the speedup increases further, for instance, if one computes the 521th root of A, the speedup is 60. The value of $\beta(U)$ is moderate and it is the same for both algorithms.

Test 3 ([19]) We consider the matrix

$$A = \begin{bmatrix} 1.0000 & -1.0000 & -1.0000 & -1.000 \\ 0 & 1.3000 & -1.0000 & -1.0000 \\ 0 & 0 & 1.7000 & -1.000 \\ 0 & 0 & 0 & 2.0000 \end{bmatrix},$$

and compute its non principal 8th root

$$X = \begin{bmatrix} 1.0000 & 6.7778 & 17.091 & 36.469 \\ 0 & -1.0333 & -5.2548 & -17.707 \\ 0 & 0 & 1.0686 & 7.1970 \\ 0 & 0 & 0 & -1.0905 \end{bmatrix},$$

for which β is large. Also in that case the two algorithms give the same numerical results.

Test 4 We consider the 10×10 Frank matrix, from MATLAB gallery function, a matrix with ill-conditioned eigenvalues and for which the value of β and the condition number of the matrix roots are rather large.

Test	n	p	Smith			Algorithm 1		
			$\beta(U)$	$\rho_A(\widetilde{X})$	time	$\beta(U)$	$\rho_A(\widetilde{X})$	time
1	4	11	1.06	$2.78 \cdot 10^{-17}$	< 0.02	1.06	$1.98 \cdot 10^{-17}$	< 0.02
	4	101	1.06	$5.21 \cdot 10^{-17}$	0.58	1.06	$5.21 \cdot 10^{-17}$	0.05
	4	1001	1.06	$4.84 \cdot 10^{-17}$	43	1.06	$4.84 \cdot 10^{-17}$	0.34
2	8	73	60.6	$5.34 \cdot 10^{-16}$	0.91	60.6	$5.36 \cdot 10^{-16}$	0.094
	8	521	61.0	$6.02 \cdot 10^{-16}$	40	61.0	$5.98 \cdot 10^{-16}$	0.70
3	4	8	$6.56 \cdot 10^{12}$	$6.56 \cdot 10^{-19}$	< 0.02	$6.56 \cdot 10^{12}$	$8.34 \cdot 10^{-19}$	< 0.02
4	10	11	$6.18 \cdot 10^{32}$	$4.16 \cdot 10^{-20}$	0.30	$6.18 \cdot 10^{32}$	$4.67 \cdot 10^{-20}$	0.062

Table 1 Comparison between Smith's algorithm and Algorithm 1 of Section 3 for some test matrices.

Acknowledgments

We would like to thank Prof. N. J. Higham and the anonymous referees for their helpful comments which improved the presentation.

References

1. Å. Björck and S. Hammarling. A Schur method for the square root of a matrix. *Linear Algebra Appl.*, 52/53:127–140, 1983.
2. E. D. Denman and A. N. Beavers, Jr. The matrix sign function and computations in systems. *Appl. Math. Comput.*, 2(1):63–94, 1976.
3. C.-H. Guo. On Newton's method and Halley's method for the principal pth root of a matrix. *Linear Algebra Appl.* to appear.

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(3):788–804, 2006.
5. N. J. Higham. The Matrix Function Toolbox. `http://www.ma.man.ac.uk/~higham/mctoolbox` (Retrieved on November 3, 2009).
6. N. J. Higham. Newton's method for the matrix square root. *Math. Comp.*, 46(174):537–549, 1986.
7. N. J. Higham. Computing real square roots of a real matrix. *Linear Algebra Appl.*, 88/89:405–430, 1987.
8. N. J. Higham. *Functions of Matrices: Theory and Computation.* Society for Industrial and Applied Mathematics, Philadelphia, PA, USA, 2008.
9. N. J. Higham and L. Lin. On pth roots of stochastic matrices. MIMS EPrint 2009.21, Manchester Institute for Mathematical Sciences, The University of Manchester, UK, Mar. 2009.
10. R. A. Horn and C. R. Johnson. *Topics in Matrix Analysis.* Cambridge University Press, Cambridge, 1994. Corrected reprint of the 1991 original.
11. B. Iannazzo. A note on computing the matrix square root. *Calcolo*, 40(4):273–283, 2003.
12. B. Iannazzo. On the Newton method for the matrix pth root. *SIAM J. Matrix Anal. Appl.*, 28(2):503–523, 2006.
13. B. Iannazzo. A family of rational iterations and its application to the computation of the matrix pth root. *SIAM J. Matrix Anal. Appl.*, 30(4):1445–1462, 2008.
14. P. Laasonen. On the iterative solution of the matrix equation $AX^2 - I = 0$. *Math. Tables Aids Comput.*, 12:109–116, 1958.
15. B. Laszkiewicz and K. Ziętak. Algorithms for the matrix sector function. *Electron. Trans. Numer. Anal.* To appear.
16. B. Laszkiewicz and K. Ziętak. A Padé family of iterations for the matrix sector function and the matrix pth root. *Numer. Linear Alg. Appl.* DOI: 10.1002/nla.656.
17. B. Meini. The matrix square root from a new functional perspective: theoretical results and computational issues. *SIAM J. Matrix Anal. Appl.*, 26(2):362–376, 2004/05.
18. M. I. Smith. A Schur algorithm for computing matrix pth roots. *SIAM J. Matrix Anal. Appl.*, 24(4):971–989, 2003.
19. M. I. Smith. *Numerical Computation of Matrix Functions.* PhD thesis, University of Manchester, Manchester, England, September 2002.
20. `http:\\bezout.dm.unipi.it\software` (Retrieved on November 3, 2009).