Applied Mathematical Sciences, Vol. 2, 2008, no. 49, 2401 - 2424

Specialized Hybrid Newton Schemes for Matrix pth Roots

Braulio De Abreu[1]

Departamento de Matemáticas, Facultad de Ingeniería
Universidad de Carabobo, Valencia, Venezuela
bdeabreu@thor.uc.edu.ve

Marlliny Monsalve[2]

Departamento de Computación, Facultad de Ciencias
Universidad Central de Venezuela, Ap. 47002, Caracas 1041-A, Venezuela
mmonsalv@kuaimare.ciens.ucv.ve

Marcos Raydan[3]

Departamento de Computación, Facultad de Ciencias
Universidad Central de Venezuela, Ap. 47002, Caracas 1041-A, Venezuela
mraydan@kuaimare.ciens.ucv.ve

Abstract

We discuss different variants of Newton's method for computing a pth root of a given matrix. A suitable implementation is presented for solving the Sylvester equation, that appears at every Newton's iteration, via Kronecker products. This approach is quadratically convergent and stable, but too expensive in computational cost. In contrast we propose and analyze some specialized versions that exploit the commutation of the iterates with the given matrix. We establish convergence of all the new versions under mild assumptions on the initial guess, and stability for one of them. These versions are relatively inexpensive and can easily be combined, at the final stages, with the Newton-Kronecker implementation to avoid possible stability problems when high precision is required. Preliminary and encouraging numerical results of the hybrid schemes are presented for $p = 3$ and $p = 5$.

[1]Supported by the CNU Alma Mater Scholarship program.
[2]Supported by CDCH-UCV project 03.00.6640.2007.
[3]Supported by the Scientific Computing Center at UCV.

1 Introduction

Consider the nonlinear matrix equation

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

where $A \in \mathbb{C}^{n\times n}$ and p is a positive integer. A solution X of (1) is called a matrix p-th root of A. This problem appears for instance as a useful tool in the calculation of matrix logarithms [3, 8], and also for computing the matrix sector function [12, 16]. The problem of computing the square root ($p = 2$) has received significant attention [2, 4, 5, 7, 9, 13], and the general case has also been considered [1, 14, 15, 17]. In this work we pay special attention to the computational issues associated with Newton's method for computing (1).

2 Classical Newton's method

Newton's method for finding the roots of $F : \mathbb{C}^{n\times n} \to \mathbb{C}^{n\times n}$ is given by the following iterative scheme

$$X_{k+1} = X_k - F'(X_k)^{-1}F(X_k), \qquad k = 0, 1, \ldots$$

where F' denotes the Fréchet derivative of F, and X_0 is given. In order to obtain F', consider the expression for $F(X + H)$, where H is an arbitrary matrix

$$\begin{aligned} F(X+H) &= (X+H)^p - A \\ &= (X^p - A) + \sum_{i=0}^{p-1} X^{p-1-i} H X^i + O(H^2). \end{aligned}$$

Now, recalling the Taylor series for F about X, $F(X+H) = F(X)+F'(X)H+R(H)$, where $R(X)$ is such that

$$\lim_{\|H\|\to 0} \frac{\|R(H)\|}{\|H\|} = 0,$$

we observe that $F'(X)$ is the linear operator given by

$$F'(X)H = \sum_{i=0}^{p-1} X^{p-1-i} H X^i.$$

Therefore, the classical Newton's method, starting at X_0, can be written as

Algorithm 1 *(Newton's method for a matrix p-th root)*
For $k = 0, 1, 2, \ldots$

Solve $\sum_{i=0}^{p-1} X_k^{p-1-i} H_k X_k^i = A - X_k^p \quad$ *(for* H_k*)*

Set $X_{k+1} = X_k + H_k$

End

The classical local convergence analysis for Newton's method guarantees that, under standard assumptions, if X_0 is sufficiently close to a p-th root of A, then the sequence generated by Algorithm 1 converges q-quadratically to that p-th root of A. On the negative side, we need to solve the linear matrix equation for H_k at every iteration of algorithm 1. For that we can use the Kronecker product and solve the following standard linear system instead

$$\left[(I \otimes X_k^{p-1}) + \sum_{q=1}^{p-2} \left((X_k^q)^T \otimes X_k^{p-q-1}\right) + \left((X_k^{p-1})^T \otimes I\right)\right] vec(H_k) = vec(A - X_k^p), \tag{2}$$

where I represents the $n \times n$ identity matrix, and $vec(X) : \mathbb{C}^{n\times n} \to \mathbb{C}^{n^2}$ is given by $vec(X) = (x_1^t \quad x_2^t \quad \cdots \quad x_n^t)^t$ where $x_j \in \mathbb{C}^n$ represents the $j-$th column of X. In here, we are using the well-known properties of the Kronecker product (see [6]), $vec(XBX) = (X^T \otimes X)vec(B)$ and $vec(XB + BX) = (I \otimes X + X^T \otimes I)vec(B)$.

The linear system (2) can be solved by means of many different well-known iterative or direct method. However, it involves n^2 equations and n^2 unknowns, and so the computational cost is very expensive. It can be simplified and the computational cost reduced by using the Schur factorization of the matrix X_k as described below. Instead of solving (2) we propose to solve at every k, the following linear system

$$\left[(I \otimes R_k^{p-1}) + \sum_{q=1}^{p-2} \left((R_k^q)^T \otimes R_k^{p-q-1}\right) + \left((R_k^{p-1})^T \otimes I\right)\right] vec(Y_k) = vec(\widetilde{C}_k), \tag{3}$$

where $X_k = Q_k R_k Q_k^T$ (Schur factorization), $Y_k = Q_k^T H_k Q_k$, and $\tilde{C}_k = Q_k^T (A - X_k^p) Q_k$.

The advantage of this new and equivalent formulation is that the coefficient matrix in (3) is block lower triangular, and each block, denoted by S_{ij}^k, is upper triangular. Hence, (3) can be solved using the next block forward substitution algorithm.

Algorithm 2 *(Block forward substitution)*

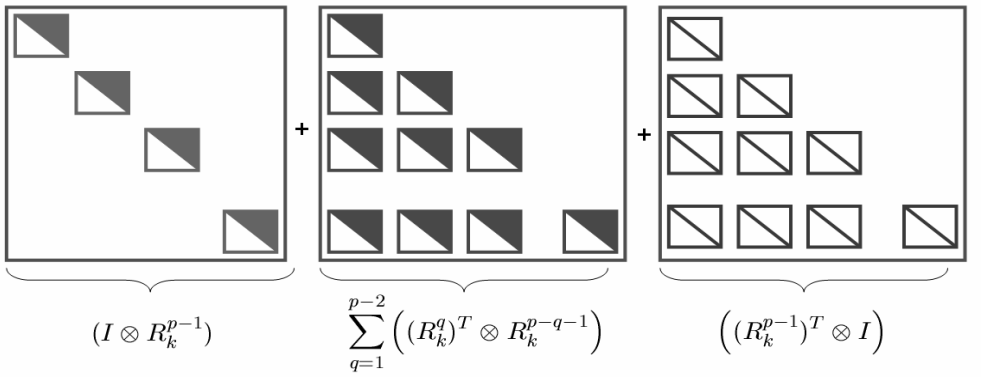

Figure 1: Structure of the coefficient matrix in (3)

Solve $S_{11}^k y_1^k = c_1^k$

For $m = 2, 3, \cdots, n$ **do**

$b_m = c_m^k - \sum_{j=1}^{m-1} S_{mj}^k y_j^k$

Solve $S_{mm}^k y_m^k = b_m$ *by back substitution*

End

where y_j^k *and* c_j^k *are the* $j-th$ *columns of* Y_k *and* $\tilde{C}_k$ *respectively.*

Consequently, using the Schur factorization of X_k, Newton's method can be written as

Algorithm 3 *(Newton-Kronecker-Schur method)*

Given X_0

For *k=0,1,2...*

$[Q_k, R_k] =$*Schur*(X_k)

Set $\tilde{C}_k = Q_k^T(A - X_k^p)Q_k$

Solve (3) for Y_k *using Algorithm 2*

Set $H_k = Q_k Y_k Q_k^T$

$X_{k+1} = X_k + H_k$

End

The structure of the coefficient matrix in (3) is shown in Figure 1.

Notice that solving (3) for Y_k using Algorithm 2, as required in Algorithm 3, does not need the explicit construction of the coefficient matrix, and this represents a significant reduction in storage. It is clear that this new formulation has a lot of potential for parallel implementations. It can also be observed,

in Figure 1, that the algorithm only needs to build the S_{mj}^k blocks, $1 \leq j \leq m$, one at a time for sequential implementations. Indeed, the S_{ij}^k blocks can be dynamically built as follows

$$S_{ij}^k = \begin{cases} \displaystyle\sum_{q=1}^{p-2} (r_{ji}^k)^q R_k^{p-q-1} + diag(t_{ji}^k) & \text{if} \quad i \neq j \\ \displaystyle\sum_{q=1}^{p-2} (r_{ii}^k)^q R_k^{p-q-1} + diag(t_{ii}^k) + R_k^{p-1} & \text{if} \quad i = j \end{cases}$$

where $(r_{ji}^k)^q$ represents the $ij-$th position R_k^q and t_{ij}^k represents the $ij-$th position of $R_k^{(p-1)}$ for $j \geq i$.

3 Specialized versions

If X_k commutes with A, then H_k commutes with X_k, and X_{k+1} will also commute with A. As we will see later, these events can be guaranteed for some special initial choices X_0. Under these circumstances the system to be solved at every iteration, of the classical Newton's method, can be simplified as follows

$$p(X_k^{p-1} H_k) = A - X_k^p,$$

and so,

$$H_k = (AX_k^{1-p} - X_k)/p.$$

After some simple manipulations, where the commutativity between A and X_k^{-1} is also exploited, we obtain three different specialized (or simplified) versions:

$$(I): \quad Y_{k+1} = \frac{1}{p}\left[Y_k^{(1-p)} A + (p-1) Y_k\right] \tag{4}$$

$$(II): \quad Z_{k+1} = \begin{cases} \frac{1}{p}\left[Z_k^{(1-p)/2} A Z_k^{(1-p)/2} + (p-1) Z_k\right] & \text{for } p \text{ odd} \\ \frac{1}{p}\left[Z_k^{-p/2} A Z_k^{-p/2} + (p-1) I\right] Z_k & \text{for } p \text{ even} \end{cases} \tag{5}$$

$$(III): \quad W_{k+1} = \frac{1}{p}\left[A W_k^{(1-p)} + (p-1) W_k\right]. \tag{6}$$

We now establish an important result concerning the commutativity of the involved matrices when the iterates are well-defined, i.e., when the Fréchet derivative, $F'(X_k)$, is nonsingular at each k. It is motivated by Theorem 1 in [4].

Theorem 1 *Consider iterations (I),(II), and (III), and also the scheme described in Algorithm 1. Suppose that $X_0 = Y_0 = Z_0 = W_0$ commute with A, and that all the Newton iterates X_k are well-defined. Then*

1. *$X_k A = A X_k$ for all k,*
2. *$X_k S_k = S_k X_k$ for all k,*
3. *$X_k = Y_k = Z_k = W_k$ for all k.*

Proof. We will prove the theorem for p odd. For p even, the arguments can be followed almost verbatim. The first part will be established by induction. If $k = 0$, $AX_0 = X_0 A$ by assumption. Suppose that $AX_k = X_k A$, which implies that $X_k^{-1} A = A X_k^{-1}$. Let

$$G_k = \frac{1}{p}\left[X_k^{(1-p)/2} A X_k^{(1-p)/2} - X_k\right].$$

We have that

$$\begin{aligned}
F'(X_k)G_k &= \sum_{q=0}^{p-1} X_k^{(p-q-1)} G_k X_k^q \\
&= \frac{1}{p}\sum_{q=0}^{p-1} X_k^{(p-q-1)}\left[X_k^{(1-p)/2} A X_k^{(1-p)/2} - X_k\right] X_k^q = \frac{1}{p}\sum_{q=0}^{p-1}\left[X_k^{(p-2q-1)/2} A X_k^{(1-p)/2} - X_k^{p-q}\right] X_k^q \\
&= \frac{1}{p}\sum_{q=0}^{p-1}\left[X_k^{(p-2q-1)/2} A X_k^{(-p+2q+1)/2} - X_k^p\right] = \frac{1}{p}\sum_{q=0}^{p-1}\left[X_k^{(p-2q-1)/2} X_k^{-(p-2q-1)/2} A - X_k^p\right] \\
&= \frac{1}{p}\sum_{q=0}^{p-1}[A - X_k^p] = A - X_k^p = -F(X_k).
\end{aligned}$$

Hence, since all Newton iterates are well-defined, then $G_k = S_k$. Moreover, $X_{k+1} = X_k + S_k = X_k + G_k$, and so

$$X_{k+1} = X_k + \frac{1}{p}\left[X_k^{(1-p)/2} A X_k^{(1-p)/2} - X_k\right].$$

Consequently

$$\begin{aligned}
AX_{k+1} &= AX_k + \tfrac{1}{p}\left[A X_k^{(1-p)/2} A X_k^{(1-p)/2} - AX_k\right] \\
&= X_k A + \tfrac{1}{p}\left[X_k^{(1-p)/2} A X_k^{(1-p)/2} A - X_k A\right] \\
&= \left[X_k + \tfrac{1}{p}\left[X_k^{(1-p)/2} A X_k^{(1-p)/2} - X_k\right]\right] A \\
&= X_{k+1} A.
\end{aligned}$$

For the second part, since $S_k = G_k$ then showing that $X_k S_k = S_k X_k$ is equivalent to showing that $X_k G_k = G_k X_k$. Indeed,

$$\begin{aligned} X_k G_k &= X_k \left[\tfrac{1}{p} \left[X_k^{(1-p)/2} A X_k^{(1-p)/2} - X_k \right] \right] \\ &= \tfrac{1}{p} \left[X_k X_k^{(1-p)/2} A X_k^{(1-p)/2} - X_k^2 \right] = \tfrac{1}{p} \left[X_k^{(1-p)/2} X_k A X_k^{(1-p)/2} - X_k^2 \right] \\ &= \tfrac{1}{p} \left[X_k^{(1-p)/2} A X_k X_k^{(1-p)/2} - X_k^2 \right] = \tfrac{1}{p} \left[X_k^{(1-p)/2} A X_k^{(1-p)/2} X_k - X_k^2 \right] \\ &= \tfrac{1}{p} \left[X_k^{(1-p)/2} A X_k^{(1-p)/2} - X_k \right] X_k \\ &= G_k X_k. \end{aligned}$$

To establish the third part it is enough to commute, in a suitable way, X_k and S_k in the Newton iteration. □

Another specialized version (let us call it $\{V_k\}$), that we would like to present, is based on a recent Newton type scheme discussed by Iannazzo [7] for computing matrix square roots. It is based on the fact that H_k commutes with X_{k+1} in the three simplified Newton schemes presented above. In that case $V_{k+1} = V_k + H_k$, and

$$H_k = \frac{1}{p}(A V_k^{1-p} - V_k) = \frac{1}{p}(A - V_k^p) V_k^{1-p}. \tag{7}$$

Hence $A - V_k^p - p V_k^{p-1} H_k = 0$, and so

$$\begin{aligned} H_{k+1} &= \tfrac{1}{p} \left[A - (V_k + H_k)^p \right] V_{k+1}^{1-p} \\ &= \frac{1}{p} \left[A - \sum_{q=0}^{p} \binom{p}{q} V_k^q H_k^{p-q} \right] V_{k+1}^{1-p} \\ &= \frac{1}{p} \left[A - p V_k^{p-1} H_k - V_k^p - \sum_{q=0}^{p-2} \binom{p}{q} V_k^q H_k^{p-q} \right] V_{k+1}^{1-p} \\ &= \frac{1}{p} \left[- \sum_{q=0}^{p-2} \binom{p}{q} V_k^q H_k^{p-q} \right] V_{k+1}^{1-p}. \end{aligned}$$

This identity yields the following general scheme for computing $\{V_{k+1}\}$

$$(IV): \begin{cases} V_{k+1} &= V_k + H_k \\ H_{k+1} &= \dfrac{1}{p} \left[- \displaystyle\sum_{q=0}^{p-2} \binom{p}{q} V_k^q H_k^{p-q} \right] V_{k+1}^{1-p}, \end{cases}$$

where V_0 is given and $H_0 = \frac{1}{p}\left[A - V_0^p\right] V_0^{1-p}$.

Based on the previous results, it follows that the Newton iteration $\{X_k\}$ and the four simplified versions $\{Y_k\}$, $\{Z_k\}$, $\{W_k\}$, and $\{V_k\}$ produce the same iterates provided the initial guess $X_0 = Y_0 = Z_0 = W_0 = V_0$ commutes with A, and $F'(X_k)$ is nonsingular at each k. We now discuss briefly the convergence of these sequences when the matrix A is diagonalizable. Following the arguments in Smith [15] (see also Higham [4]), we can in theory diagonalize the iterates, and uncouple the process into n scalar Newton iterations for the pth root of the eigenvalues, λ_i, of A. According to the standard analysis for the scalar Newton's method, these scalar sequences converge to $\lambda_i^{1/p}$ provided the eigenvalues of the initial guess are close enough to the eigenvalues at the solution. Therefore, in particular, if the matrix A is symmetric and positive definite, and the initial guess is chosen as a symmetric and positive definite matrix that commutes with A (e. g., A or I), then the four simplified versions converge to the unique symmetric and positive definite pth root of A. Notice that in this case we are guaranteeing convergence to the so-called principal pth root of A [16].

Concerning the stability issue, it is well-known that under standard assumptions, the classical Newton's method is a stable algorithm for finding roots of nonlinear maps. Since the Schur factorization of a given matrix is also a stable process, then Algorithm 3 for computing a p-th root of a matrix is stable. On the other hand, as discussed by Smith [15], the simplified versions that generate the sequences $\{Y_k\}$ and $\{W_k\}$ are unstable. The version that generates $\{Z_k\}$ belongs to the same family and, as we will see in practice, it is also unstable although in general it reduces the loss of numerical commutativity, and as a consequence it has a better behavior than the other two simplified versions. On the other hand, as we will establish later, the sequence $\{V_k\}$ has a stable behavior.

To illustrate the discussed behavior of the sequences $\{Y_k\}$, $\{Z_k\}$, $\{W_k\}$, and $\{V_k\}$ we apply them to the specific problem of finding the cubic root ($p = 3$) of the 5×5 Hilbert matrix. In that case, the three simplified versions $\{Y_k\}$, $\{Z_k\}$, $\{W_k\}$ can be written as

$$\begin{aligned} Y_{k+1} &= \frac{1}{3}\left[Y_k^{-2}A + 2Y_k\right] \\ Z_{k+1} &= \frac{1}{3}\left[Z_k^{-1}AZ_k^{-1} + 2Z_k\right] \\ W_{k+1} &= \frac{1}{3}\left[AW_k^{-2} + 2W_k\right]. \end{aligned}$$

For the sequence $\{V_k\}$, when $p = 3$, we present the following expression to compute H_{k+1} obtained from Algorithm (IV). Computing the cubic root of matrices has been recently associated with the calculation of fractional derivatives that appear in high energy physics [10, 11].

$$\begin{aligned} H_{k+1} &= \tfrac{1}{3}(-H_k^3 - 3V_k H_k^2)V_{k+1}^{-2} \\ &= \tfrac{1}{3}(-3V_k - H_k)H_k^2 V_{k+1}^{-2} \\ &= \tfrac{1}{3}(-3V_{k+1} + 2H_k)H_k^2 V_{k+1}^{-2} \\ [2mm] &= \tfrac{1}{3}H_k V_{k+1}^{-1} H_k (2V_{k+1}^{-1}H_k - 3I). \end{aligned}$$

This identity produces the following simplified version of Algorithm (IV) for computing the cubic root of A.

Algorithm 4 *Given V_0 such that $AV_0 = V_0 A$*

Set $H_0 = \frac{1}{3}(V_0^{-1}AV_0^{-1} - V_0)$

For *k=0,1,2...*

$V_{k+1} = V_k + H_k$

$T_{k+1} = V_{k+1}^{-1}H_k$

$H_{k+1} = \frac{1}{3}H_k T_{k+1}(2T_{k+1} - 3I)$

End

Repeating the same arguments for $p = 5$, it follows from Algorithm (IV) that

$$H_{k+1} = \frac{1}{5}H_k V_{k+1}^{-1} H_k (4V_{k+1}^{-3}H_k^3 - 15V_{k+1}^{-2}H_k^2 + 20V_{k+1}^{-1}H_k - 10I),$$

and the following algorithm for computing the fifth root of A is obtained

Algorithm 5 *Given V_0 such that $AV_0 = V_0 A$*

Set $H_0 = \frac{1}{5}(V_0^{-2}AV_0^{-2} - V_0)$

For *k=0,1,2...*

$V_{k+1} = V_k + H_k$

$T_{k+1} = V_{k+1}^{-1}H_k$

$H_{k+1} = \frac{1}{5}H_k T_{k+1}(4T_{k+1}^3 - 15T_{k+1}^2 + 20T_{k+1} - 10I)$

End

In Figure 2 we can see the behavior of the four simplified versions of Newton's method for finding the cubic root of the 5×5 Hilbert matrix. As expected, the versions that produce the sequences $\{Z_k\}$, $\{W_k\}$ and $\{Y_k\}$ are unstable, whereas the one that produces the sequence $\{V_k\}$ is not. However, the sequence $\{V_k\}$ shows stagnation below the required accuracy. Moreover, it is

worth noticing that the sequence $\{Z_k\}$ achieves a better approximation before blowing up. This behavior has been observed in several experiments for different test matrices with different dimensions. This is precisely the characteristic that motivates us to introduce, later in this paper, hybrid techniques that combine the sequences $\{Z_k\}$ and $\{V_k\}$ with Algorithm 3 and a suitable alternating projection scheme. First, in our next section we study the stability of the sequence $\{V_k\}$.

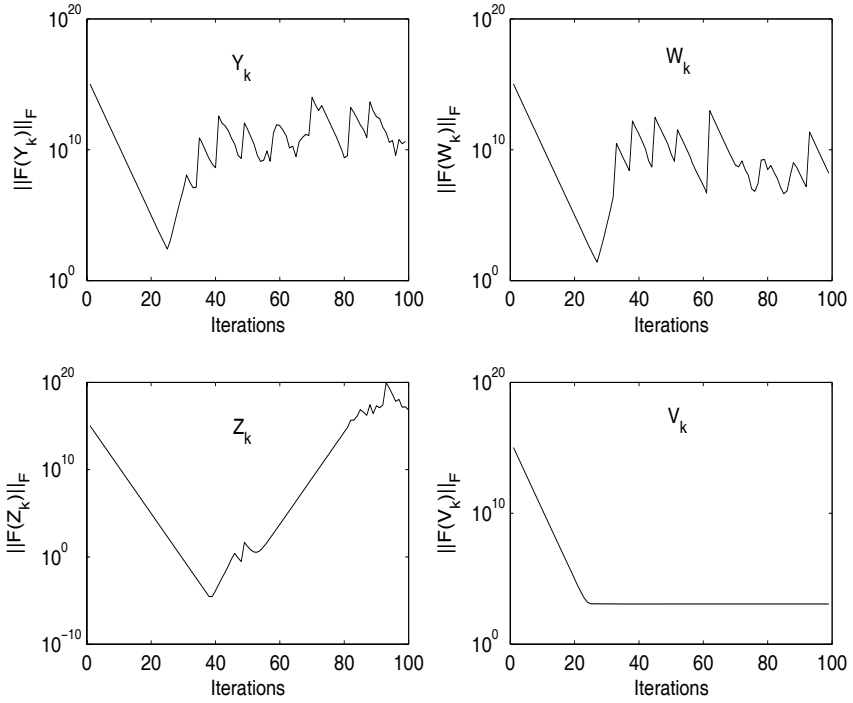

Figure 2: Behavior of simplified versions of Newton's method for finding the cubic root of the 5×5 Hilbert matrix.

4 Stability of the sequence $\{V_k\}$

In this section we establish that the sequence $\{V_k\}$ is stable for any p. As described before for $p = 3$ and $p = 5$, H_{k+1} can be written as a polynomial expression involving $T_{k+1} = V_{k+1}^{-1} H_k$ multiplied by H_k, and so H_{k+1} depends only on H_k and V_{k+1}. For that we need to establish the following useful result.

Proposition 1 *Let X, Y be any two matrices in $\mathbb{R}^{n\times n}$ such that $XY = YX$. Then*

$$\forall \beta_q \in I\!R : \exists \lambda_q \in I\!R : \sum_{q=0}^{m} \beta_q X^q Y^{m-q} = \sum_{q=0}^{m} \lambda_q (X+Y)^q Y^{m-q} \tag{8}$$

where $m \geq 0$.

Proof. We proceed by induction on m. For $m = 0$, if we take $\lambda_0 = \beta_0$ the result clearly holds. Let us assume that the result is true for $m = k$. We need to show that it holds for $m = k+1$. For that let $\widehat{\beta}_q \in I\!R$ be any real scalars for each $0 \leq q \leq k+1$, and let us consider the matrix $\sum_{q=0}^{k+1} \widehat{\beta}_q X^q Y^{k+1-q}$, that can be written as

$$\sum_{q=0}^{k+1} \widehat{\beta}_q X^q Y^{k+1-q} = \sum_{q=0}^{k} \widehat{\beta}_q X^q Y^{k+1-q} + \widehat{\beta}_{k+1} X^{k+1}. \tag{9}$$

Using now Newton's binomial formula we obtain

$$(X+Y)^{k+1} = \sum_{q=0}^{k+1} \left(\begin{array}{c} k+1 \\ q \end{array} \right) X^q Y^{k+1-q} = X^{k+1} + \sum_{q=0}^{k} \left(\begin{array}{c} k+1 \\ q \end{array} \right) X^q Y^{k+1-q},$$

which implies in our setting

$$\widehat{\beta}_{k+1} X^{k+1} = \widehat{\beta}_{k+1} (X+Y)^{k+1} - \sum_{q=0}^{k} \widehat{\beta}_{k+1} \left(\begin{array}{c} k+1 \\ q \end{array} \right) X^q Y^{k+1-q}. \tag{10}$$

Substituting (10) in (9) it follows that

$$\begin{aligned} \sum_{q=0}^{k+1} \widehat{\beta}_q X^q Y^{k+1-q} &= \sum_{q=0}^{k} \widehat{\beta}_q X^q Y^{k+1-q} - \sum_{q=0}^{k} \widehat{\beta}_{k+1} \left(\begin{array}{c} k+1 \\ q \end{array} \right) X^q Y^{k+1-q} + \widehat{\beta}_{k+1} (X+Y)^{k+1} \\ &= \left(\sum_{q=0}^{k} \left[\widehat{\beta}_q - \widehat{\beta}_{k+1} \left(\begin{array}{c} k+1 \\ q \end{array} \right) \right] X^q Y^{k-q} \right) Y - \widehat{\beta}_{k+1} (X+Y)^{k+1}. \end{aligned} \tag{11}$$

Using the inductive hypothesis in (11) we have that there exist $\widehat{\lambda}_q$ for $0 \leq q \leq k$ such that

$$\begin{aligned} \sum_{q=0}^{k+1} \widehat{\beta}_q X^q Y^{k+1-q} &= \left(\sum_{q=0}^{k} \widehat{\lambda}_q (X+Y)^q Y^{k-q} \right) Y - \widehat{\beta}_{k+1} (X+Y)^{k+1} \\ &= \sum_{q=0}^{k} \widehat{\lambda}_q (X+Y)^q Y^{k+1-q} - \widehat{\beta}_{k+1} (X+Y)^{k+1} Y^0 \\ &= \sum_{q=0}^{k+1} \widehat{\lambda}_q (X+Y)^q Y^{k+1-q}, \end{aligned} \tag{12}$$

where $\widehat{\lambda}_{k+1} = -\widehat{\beta}_{k+1}$, and the result is established. □

We are now ready to write H_{k+1} as a function of H_k and V_{k+1}. Indeed, from Algorithm (IV), H_{k+1} can be written as

$$H_{k+1} = \frac{1}{p}\left[-\sum_{q=0}^{p-2}\left(\begin{array}{c} p \\ q \end{array}\right) V_k^q H_k^{p-q-2}\right] H_k^2 V_{k+1}^{1-p}.$$

Using Proposition 1 and recalling that H_k commutes with V_k we obtain that

$$\begin{aligned} H_{k+1} &= \frac{1}{p}\left[\sum_{q=0}^{p-2}\lambda_q (V_k + H_k)^q H_k^{p-q-2}\right] H_k^2 V_{k+1}^{1-p} = \frac{1}{p} H_k \left[\sum_{q=0}^{p-2}\lambda_q V_{k+1}^{-p+q+1} H_k^{p-q-1}\right] \\ &= \frac{1}{p} H_k \left[\sum_{q=0}^{p-2}\lambda_q (V_{k+1}^{-1} H_k)^{p-q-1}\right] = \frac{1}{p} H_k \left[\sum_{q=0}^{p-2}\lambda_q T_{k+1}^{p-q-1}\right] \end{aligned} \tag{13}$$

where $T_{k+1} = V_{k+1}^{-1} H_k$.

Concerning equation (13), it is worth noticing that in order to have a practical expression for a chosen p, we must know explicitly λ_q for each q. Nevertheless, to establish the stability of the sequence $\{V_k\}$, we only need to know that λ_q is a real number for $0 \leq q \leq p-2$.

Theorem 2 (Stability of the sequence $\{V_k\}$) *Let us assume that $A \in \mathbb{C}^{n\times n}$. If V_0 is given such that $V_0 A = A V_0$, and $V_k \to A^{1/p}$ when $k \to \infty$, then the sequence $\{V_k\}$ generated by Algorithm (IV) is numerically stable.*

Proof. Emulating the arguments used in [7, pp. 277-278], we will analyze the effect of small perturbations at a given iteration over the forthcoming iterations.

Let ΔV_k and ΔH_k be the numerical perturbations introduced at the k-th iteration of Algorithm (IV), and let

$$\begin{aligned} \widehat{V}_k &= V_k + \Delta V_k, \\ \widehat{H}_k &= H_k + \Delta H_k. \end{aligned}$$

In our analysis we assume exact arithmetic, and we also assume that the second order terms $(\Delta V_k)^2$, $(\Delta H_k)^2$, $\Delta V_k \Delta H_k$, and $\Delta H_k \Delta V_k$ are insignificant and so they are not taken into account. For $\widehat{V}_{k+1}$ it follows that

$$\widehat{V}_{k+1} = \widehat{V}_k + \widehat{H}_k = V_{k+1} + \Delta V_k + \Delta H_k,$$

and hence

$$\Delta V_{k+1} = \widehat{V}_{k+1} - V_{k+1} = \Delta V_k + \Delta H_k. \tag{14}$$

For $\widehat{H}_{k+1}$ we have using (13) that

$$\widehat{H}_{k+1} = \frac{1}{p}\widehat{H}_k \left[\sum_{q=0}^{p-2} \lambda_q \widehat{T}_{k+1}^{p-q-1}\right]. \tag{15}$$

Since $\widehat{T}_{k+1}$ can be written as

$$\widehat{T}_{k+1} = \widehat{V}_{k+1}^{-1}\widehat{H}_k = (V_{k+1} + \Delta V_{k+1})^{-1}(H_k + \Delta H_k),$$

and using the well-known fact that for any nonsingular matrix B and any matrix C, $(B+C)^{-1} \approx B^{-1} - B^{-1}CB^{-1}$, up to second order terms, we obtain that

$$\begin{aligned} \widehat{T}_{k+1} &\approx (V_{k+1}^{-1} - V_{k+1}^{-1}\Delta V_{k+1}V_{k+1}^{-1})(H_k + \Delta H_k) \\ &\approx V_{k+1}^{-1}H_k + V_{k+1}^{-1}\Delta H_k - V_{k+1}^{-1}\Delta V_{k+1}V_{k+1}^{-1}H_k. \end{aligned} \tag{16}$$

From equation (14) we know that $\Delta V_{k+1} = \Delta V_k + \Delta H_k$, and so (16) can be written as

$$\begin{aligned} \widehat{T}_{k+1} &\approx V_{k+1}^{-1}H_k + V_{k+1}^{-1}\Delta H_k - V_{k+1}^{-1}(\Delta V_k + \Delta H_k)V_{k+1}^{-1}H_k \\ &= V_{k+1}^{-1}H_k + V_{k+1}^{-1}\Delta H_k - V_{k+1}^{-1}\Delta V_k V_{k+1}^{-1}H_k - V_{k+1}^{-1}\Delta H_k V_{k+1}^{-1}H_k \\ &= T_{k+1} + M_{k+1}, \end{aligned} \tag{17}$$

where $M_{k+1} = V_{k+1}^{-1}\Delta H_k - V_{k+1}^{-1}\Delta V_k T_{k+1} - V_{k+1}^{-1}\Delta H_k T_{k+1}$.
Substituting (17) in (15) we have that

$$\widehat{H}_{k+1} = \frac{1}{p}\widehat{H}_k \left[\sum_{q=0}^{p-2} \lambda_q (T_{k+1} + M_{k+1})^{p-q-1}\right]. \tag{18}$$

Since second order terms are not taken into account, then for the polynomial $(T_{k+1} + M_{k+1})^{p-q-1}$, in (18), we only need to consider the terms for which M_{k+1} is of degree zero or one:

$$(T_{k+1} + M_{k+1})^{p-q-1} \approx T_{k+1}^{p-q-1} + \sum_{j=0}^{p-q-2} T_{k+1}^{j} M_{k+1} T_{k+1}^{p-q-2-j}. \tag{19}$$

Substituting (19) in (18), and using $\widehat{H}_k = H_k + \Delta H_k$ and equation (13), it follows that

$$\begin{aligned}\widehat{H}_{k+1} &\approx \frac{1}{p}(H_k + \Delta H_k)\left[\sum_{q=0}^{p-2}\lambda_q T_{k+1}^{p-q-1} + \sum_{q=0}^{p-2}\lambda_q \sum_{j=0}^{p-q-2} T_{k+1}^{j} M_{k+1} T_{k+1}^{p-q-2-j}\right] \\ &\approx H_{k+1} + \frac{1}{p}H_k \sum_{q=0}^{p-2}\lambda_q \sum_{j=0}^{p-q-2} T_{k+1}^{j} M_{k+1} T_{k+1}^{p-q-2-j} + \frac{1}{p}\Delta H_k \sum_{q=0}^{p-2}\lambda_q T_{k+1}^{p-q-1}.\end{aligned}$$

Recalling now that $\Delta H_{k+1} = \widehat{H}_{k+1} - H_{k+1}$ and that H_k commutes with V_k, we conclude that H_k also commutes with $T_{k+1} = V_{k+1}^{-1} H_k$, and so

$$\Delta H_{k+1} \approx \frac{1}{p}\sum_{q=0}^{p-2}\lambda_q \sum_{j=0}^{p-q-2} T_{k+1}^{j} H_k M_{k+1} T_{k+1}^{p-q-2-j} + \frac{1}{p}\Delta H_k \sum_{q=0}^{p-2}\lambda_q T_{k+1}^{p-q-1}. \quad (20)$$

Using now the definition of M_{k+1} we obtain that

$$\begin{aligned}H_k M_{k+1} &= H_k[V_{k+1}^{-1}\Delta H_k - V_{k+1}^{-1}\Delta V_k T_{k+1} - V_{k+1}^{-1}\Delta H_k T_{k+1}] \\ &= T_{k+1}[\Delta H_k - \Delta V_k T_{k+1} - \Delta H_k T_{k+1}]. \quad (21)\end{aligned}$$

Finally, substituting (21) in (20), we have the desired expression for ΔH_{k+1}

$$\begin{aligned}\Delta H_{k+1} &\approx \frac{1}{p}\sum_{q=0}^{p-2}\lambda_q \sum_{j=0}^{p-q-2} T_{k+1}^{j+1}\Delta H_k T_{k+1}^{p-q-2-j} - \frac{1}{p}\sum_{q=0}^{p-2}\lambda_q \sum_{j=0}^{p-q-2} T_{k+1}^{j+1}\Delta V_k T_{k+1}^{p-q-1-j} \\ &\quad -\frac{1}{p}\sum_{q=0}^{p-2}\lambda_q \sum_{j=0}^{p-q-2} T_{k+1}^{j+1}\Delta H_k T_{k+1}^{p-q-1-j} - \frac{1}{p}\Delta H_k \sum_{q=0}^{p-2}\lambda_q T_{k+1}^{p-q-1}. \quad (22)\end{aligned}$$

Applying any subordinate norm to (22), it follows that

$$\begin{aligned}&\|\Delta H_{k+1}\| \leq \\ &\|\Delta H_k\|\left[\frac{1}{p}\sum_{q=0}^{p-2}|\lambda_q|(p-q-1)\|T_{k+1}\|^{p-q-1}\right] + \|\Delta V_k\|\left[\frac{1}{p}\sum_{q=0}^{p-2}|\lambda_q|(p-q-1)\|T_{k+1}\|^{p-q}\right] \\ &+ \|\Delta H_k\|\left[\frac{1}{p}\sum_{q=0}^{p-2}|\lambda_q|(p-q-1)\|T_{k+1}\|^{p-q}\right] + \|\Delta H_k\|\left[\frac{1}{p}\sum_{q=0}^{p-2}|\lambda_q|\|T_{k+1}\|^{p-q-1}\right]. \quad (23)\end{aligned}$$

Setting now, for simplicity, $\alpha_k = \|T_{k+1}\|$, $c_1 = \frac{1}{p}\sum_{q=0}^{p-2}|\lambda_q|(p-q-1)\|T_{k+1}\|^{p-q-1}$ and $c_2 = \frac{1}{p}\sum_{q=0}^{p-2}|\lambda_q|\|T_{k+1}\|^{p-q-1}$, (23) can be written in a compact form as

$$\begin{aligned}\|\Delta H_{k+1}\| &\leq c_1\|\Delta H_k\| + c_1\|\Delta V_k\|\|T_{k+1}\| + c_1\|\Delta H_k\|\|T_{k+1}\| + c_2\|\Delta H_k\| \\ &= [c_1\alpha_k]\|\Delta V_k\| + [c_1 + c_1\alpha_k + c_2]\|\Delta H_k\|. \quad (24)\end{aligned}$$

Moreover, (7) implies that

$$H_k = \frac{1}{p}(AV_k^{1-p} - V_k) = \frac{1}{p}\left[A - V_k^p\right] V_k^{1-p},$$

and hence

$$\alpha_k = \|T_{k+1}\| = \frac{1}{p}\|V_{k+1}^{-1}(AV_k^{1-p} - V_k)\| = \frac{1}{p}\|(AV_k^{-p} - I)V_k V_{k+1}^{-1}\|,$$

which in turn implies that

$$\alpha_k \leq \frac{1}{p}\|AV_k^{-p} - I\| \, \|V_k\| \, \|V_{k+1}\|^{-1}.$$

Consequently, for k large enough, if V_k is close to a p-th root of A, $\|V_{k+1}\| \approx \|V_k\|$ and $\|AV_k^{-p} - I\| \approx 0$. Therefore, $0 < \alpha_k \ll 1$. Furthermore, since c_1 and c_2 are polynomial expressions in the variable $\alpha_k = \|T_{k+1}\|$, they both tend to zero when k goes to infinity. Hence, using (24), for k large enough $\|\Delta H_{k+1}\| \ll \|\Delta H_k\|$ and $\|\Delta V_{k+1}\| \approx \|\Delta V_k\|$, i.e., the error at iteration k does not amplify and the sequence $\{V_k\}$ is numerically stable. □

5 Hybrid versions

Summing up, we have two interesting simplified Newton versions for computing a p-th root of a given matrix A. The sequence $\{Z_k\}$ that tends to achieve a much better approximation of the p-th root of A than most previously-known simplified schemes, but that could show an unstable behavior; and the sequence $\{V_k\}$ for which we have established stability for any p, but that tends to stagnate before reaching a highly accurate approximation. In this section we discuss some simple schemes to combine these two promising schemes with a few Newton-Kronecker-Schur final steps, to improve their performance.

We can combine the algorithm that generates the sequence $\{Z_k\}$ with Algorithm 3 as follows. Generate the sequence $\{Z_k\}$, monitoring the size of the residual, and change to Algorithm 3 at the first iteration $\bar{k}$ for which

$$\|F(Z_{\bar{k}+1})\|_F \geq \delta * \|F(Z_{\bar{k}})\|_F,$$

for $1 < \delta \ll 2$, using $X_0 = Z_{\bar{k}}$ as its initial guess. i. e., when the residual associated with the sequence $\{Z_k\}$ shows an unstable behavior. This hybrid scheme makes sense, since the sequence $\{Z_k\}$ tends to achieve good accuracy before showing an unstable behavior, and so, the number of Newton-Kronecker-Schur steps required should be very small. Consequently, the expensive final steps would be performed very few times.

A similar hybrid scheme can be defined combining the sequence $\{V_k\}$ with Algorithm 3. In this case, the decision to start the Newton-Kronecker-Schur iterations is based on the possible stagnation of the sequence $\{V_k\}$, described in the previous section. Therefore, we proceed as follows. Generate the sequence $\{V_k\}$, monitoring the size of the residual, until

$$\|V_{\bar{k}+1} - V_{\bar{k}}\|_F \leq 1.D - 15,$$

and continue with Algorithm 3 otherwise, using $X_0 = V_{\bar{k}}$ as its initial guess.

Another possible hybrid schemes can be constructed based on the lack of commutativity when using the simplified versions. This fact motivates us to propose the following combinations. Generate the sequence $\{Z_k\}$ (or $\{V_k\}$), monitoring $\rho_k = \|AZ_k - Z_kA\|_F$ (or $\rho_k = \|AV_k - V_kA\|_F$). If $\rho_k \geq 1.D-8$ then project the iterate Z_k (or V_k) onto the subspace of matrices X that satisfy $AX - XA = 0$, and continue with the sequence $\{Z_k\}$ (or $\{V_k\}$) from the projected matrix. There is a straightforward implementation of this projection process in MATLAB using the SVD factorization of Z_k (or V_k) that, unfortunately, is very expensive as we will see in our numerical experiments. Nevertheless, if a suitable and less expensive projection is available, then this combination of ideas is a promising one.

In this work, we are using the following MATLAB code to perform a projection onto the subspace of matrices that commute with A:

```
function [P]=projSCA(A,B,I,n)
  C=kron(I,A)-kron(A',I);
  b=reshape(B,n*n,1);
  invC=pinv(C*C');
  pb=b-C'*invC*C*b;
  P=reshape(pb,n,n);
```

6 Numerical Experiments

In the implementation of our hybrid schemes we proceed as follows:

- We stop all considered algorithms when the Frobenius norm of the residual is less than $0.5D - 12$.
- We consider that an algorithm fails when the number of iterations exceeds 100.
- The initial guess for all algorithms is the matrix A.
- We fix the parameter $\delta = 1.2$

All experiments were run on a Pentium IV, 3.4GHz, using Matlab 7. We report the number of required iterations (Iter), the CPU time in seconds (CPU), and the norm of the residual ($\|F(X_k)\|_F$) when the process is stopped. We use the symbol (**) to report a failure. The algorithms that generate the sequence $\{Z_k\}$ in (5) can be written in a compact form as follows:

Algorithm 6 *For p odd*
Given $Z_0 = I$, $T_0 = A$
For *k=0,1,2...*

$$Z_{k+1} = \tfrac{1}{p}\left[T_k + (p-1)Z_k\right]$$
$$U_{k+1} = Z_{k+1}^{-1} Z_k$$
$$T_{k+1} = U_{k+1}^{(p-1)/2}\, T_k\, U_{k+1}^{(p-1)/2}$$

End

Algorithm 7 *For p even*
Given $Z_0 = I$, $T_0 = A$
For *k=0,1,2...*

$$Z_{k+1} = \tfrac{1}{p}\left[T_k + (p-1)I\right] Z_k$$
$$U_{k+1} = Z_{k+1}^{-1} Z_k$$
$$T_{k+1} = U_{k+1}^{p/2}\, T_k\, U_{k+1}^{p/2}$$

End

We compare the following list of hybrid Newton algorithms with the Newton-Kronecker-Schur method (NKS), based on Algorithm 3, and the simplified versions that generate the sequences Z_k (NZ) and V_k (NV):

- Z_k + Newton-Kronecker-Schur (NKS)
- V_k + Newton-Kronecker-Schur (NKS)
- Z_k + Projections (P) (at most 3)
- V_k + Projections (P) (at most 3)
- Z_k + one Projection (P) + Newton-Kronecker-Schur (NKS).

We test the algorithms with the following matrices from the Matlab gallery:

- `hilb:` Hilbert matrix.
- `kahan:` an upper trapezoidal matrix. We set $\theta = 2.3$.
- `fiedler:` Symmetric. We set $c = \frac{1}{n}(1, 2, \cdots, n)^t$.
- `lehmer:` Symmetric and positive definite.
- `parter:` It possesses complex eigenvalues. We use $X_0 = A + (1.D{-}16) * i$.
- `pei:` Symmetric. We set $\alpha = -3$ such that the matrix is indefinite.

Since the Hilbert and the lehmer matrices are positive definite and our initial guess is the matrix A, then in case of successful convergence, the associated sequence converges to the principal p-th root of them. In our first experiment, we show the performance of the different algorithms for finding

the principal cubic root of the 5×5 Hilbert matrix. For the sequence $\{V_k\}$ we use Algorithm 4. The results are in Table 1 and we can see in this table that the *pure* sequences NZ and NV do not converge, and for this reason we do not report those options in the forthcoming tables.

In Figure 3 we compare the residual norms of *pure* sequences with the residual norms of hybrid versions. We can observe that the norm of the residual, for the sequence Z_k, was approximately 10^{-5} before changing to NKS, while the norm of the residual, for the sequence V_k, was approximately 10^2 before changing to NKS. This significant difference explains why Z_k + NKS requires fewer iterations than V_k + NKS.

Scheme	Iter	CPU	$\|F(X_k)\|_F$
Z_k + NKS	46(40+6)	0.046875	1.7609e-016
V_k + NKS	52(31+21)	0.03125	2.7195e-016
Z_k + P	46(2)	0.03125	2.1302e-013
V_k + P	55(2)	0.03125	2.1394e-013
Z_k + P + NKS	46(40+1+6)	0.0412	1.7609e-016
NKS	45	0.0625	3.8858e-016
NZ	**	**	**
NV	**	**	**

Table 1: Performance of Hybrid versions of Newton's method, simplified versions of Newton's method and Newton's method for finding the principal cubic root of the 5×5 Hilbert matrix.

In Table 2 we show the performance of the different algorithms for finding a cubic root of the 25×25 Kahan matrix and we observe that when n increases the projection onto the subspace of matrices that commute with A takes a significant amount of CPU time. For that reason we do not report those options in the forthcoming tables.

From Table 3 to Table 5 we observe once again that for all values of n, small or large, the hybrid versions required less CPU time than the Newton-Kronecker-Schur method. It is also clear that the advantage of using the hybrid versions increases when n increases.

In Table 6 we observe that the hybrid version Z_k + NKS required more NKS iterations than the version V_k + NKS. This behavior could be explained as follows. The criterion used in the hybrid version Z_k+ NKS, for changing to NKS, does not guarantee that $\|F(Z_{\overline{k+1}})\|$ has reached the smallest possible value, whereas in the hybrid version V_k + NKS we can guarantee that $\|F(V_{\overline{k+1}})\|$ can not decrease any more.

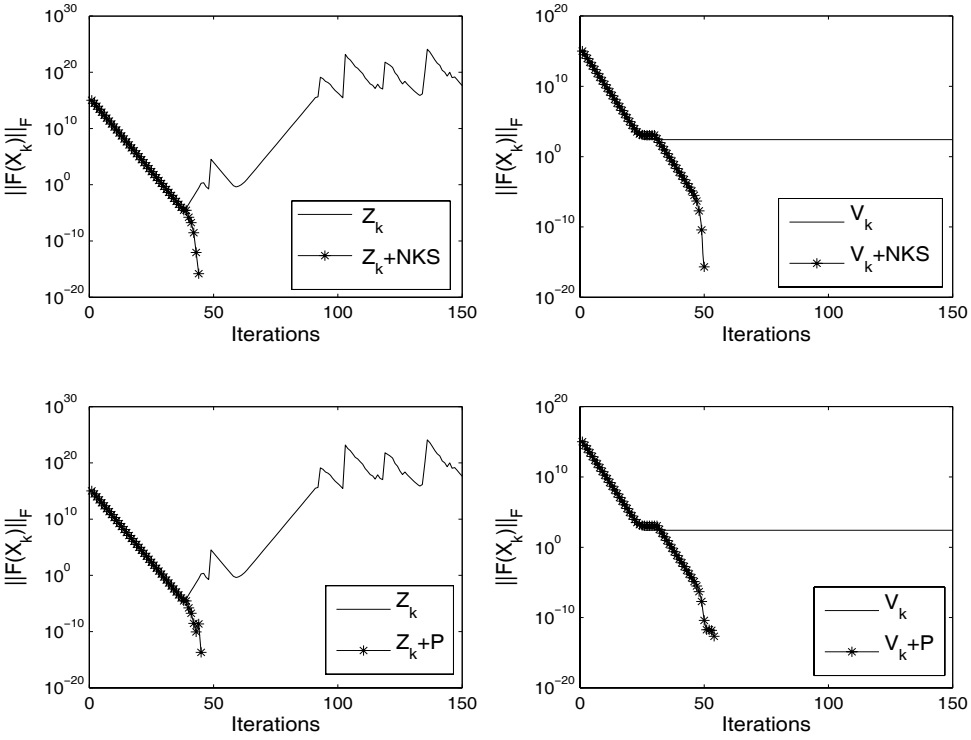

Figure 3: Behavior of simplified and hybrid versions of Newton's method for finding the principal cubic root of the 5×5 Hilbert matrix.

Scheme	Iter	CPU	$\|F(X_k)\|_F$
Z_k + NKS	30(28+2)	0.11938	6.2156e-016
V_k + NKS	29(27+2)	0.09375	8.5677e-016
Z_k + P	30(3)	12.5625	2.6393e-015
V_k + P	29(1)	4.2344	2.5872e-015
Z_k + P + NKS	30(28+1+2)	4.3125	6.2156e-016
NKS	27	0.67188	7.3154e-016

Table 2: Performance of Hybrid versions of Newton's method finding a cubic root of the 25×25 `Kahan` matrix.

In Tables 7, 8 and 9 we show the performance of the different hybrid versions of Newton's method for finding fifth roots of several matrices. For the sequence $\{V_k\}$ we use Algorithm 5. As in the previous experiments, the required CPU time for the NKS option is very high and we do not report it

		Z_k + NKS	V_k + NKS	NKS
$n = 10$	iter	13(13+0)	13(13+0)	13
	CPU	0.03125	0.03125	0.036875
	$\|F(X_k)\|_F$	5.6203e-013	1.05e-013	8.758e-013
$n = 50$	iter	21(19+2)	21(19+2)	19
	CPU	0.48438	0.48438	4.3125
	$\|F(X_k)\|_F$	6.637e-015	6.4122e-015	6.4e-015
$n = 90$	iter	22(20+2)	23(21+2)	21
	CPU	7.1406	7.1719	76.7344
	$\|F(X_k)\|_F$	1.5339e-014	1.4756e-014	1.4855e-014

Table 3: Performance of Hybrid versions of Newton's method for finding a cubic root of the `fiedler` matrix for different values of n.

		Z_k + NKS	V_k + NKS	NKS
$n = 50$	iter	15(13+2)	14(12+2)	12
	CPU	1.9219	1.9219	10.7188
	$\|F(X_k)\|_F$	1.6735e-014	1.7466e-014	1.5328e-014
$n = 150$	iter	16(14+2)	15(13+2)	13
	CPU	48.1875	48.2813	312.7188
	$\|F(X_k)\|_F$	8.3214e-014	9.5548e-014	1.3116e-013

Table 4: Performance of Hybrid versions of Newton's method for finding a cubic root of the `pei` matrix for different values of n.

anymore. We can observe that the hybrid version Z_k + NKS does not require NKS iterations, while V_k + NKS in all cases requires NKS iterations. This happens because the sequence V_k is stabilized when the residual still has a large norm value.

		Z_k + NKS	V_k + NKS	NKS
$n = 60$	iter	22(20+2)	21(19+2)	19
	CPU	1.0313	1.037	9.25
	$\|F(X_k)\|_F$	9.6909e-015	9.9393e-015	1.0116e-014
$n = 80$	iter	22(20+2)	22(20+2)	20
	CPU	3.6094	3.5938	35.1094
	$\|F(X_k)\|_F$	1.5194e-014	1.5074e-014	1.5336e-014
$n = 100$	iter	23(21+2)	23(21+2)	21
	CPU	11.3906	11.3906	118.625
	$\|F(X_k)\|_F$	2.1076e-014	2.0934e-014	2.1402e-014
$n = 120$	iter	23(21+2)	24(22+2)	21
	CPU	26.5781	26.625	277.2656
	$\|F(X_k)\|_F$	2.76e-014	2.7636e-014	2.7719e-014

Table 5: Performance of Hybrid versions of Newton's method for finding the principal cubic root of the `lehmer` matrix for different values of n.

		Z_k + NKS	V_k + NKS	NKS
$n = 10$	iter	12(7+5)	12(11+1)	11
	CPU	0.03125	0.03125	0.046875
	$\|F(X_k)\|_F$	1.4954e-015	1.9252e-015	1.6216e-015
$n = 20$	iter	13(5+8)	13(12+1)	12
	CPU	0.29688	0.0625	0.32813
	$\|F(X_k)\|_F$	4.3693e-015	5.0361e-015	5.1429e-015
$n = 50$	iter	20(5+15)	21(19+2)	19
	CPU	13.7969	2.0156	17.3438
	$\|F(X_k)\|_F$	9.6557e-015	9.9789e-015	9.5033e-015

Table 6: Performance of Hybrid versions of Newton's method for finding a cubic root of the `parter` matrix for different values of n.

Scheme	Iter	CPU	$\|F(X_k)\|_F$
Z_k + NKS	15(15+0)	0	1.7299e-013
V_k + NKS	23(17+6)	0.015625	8.5898e-014

Table 7: Performance of Hybrid versions of Newton's method for finding a fifth ($p = 5$) root of the 5×5 `Kahan` matrix.

Scheme	Iter	CPU	$\|F(X_k)\|_F$
Z_k + NKS	25(25+0)	0.0133	1.1974e-015
V_k + NKS	54(26+28)	0.95313	2.2748e-013

Table 8: Performance of Hybrid versions of Newton's method for finding the principal fifth root of the 5×5 `lehmer` matrix.

		Z_k + NKS	V_k + NKS
$n = 10$	iter	12(12+0)	20(13+7)
	CPU	0.001	0.0625
	$\|F(X_k)\|_F$	1.9817e-014	2.7559e-013
$n = 15$	iter	14(14+0)	15(15+14)
	CPU	0.015625	0.21875
	$\|F(X_k)\|_F$	2.2914e-014	3.4695e-013

Table 9: Performance of Hybrid versions of Newton's method for finding a fifth root of the `pei` matrix for different values of n.

7 Conclusions

We present several specialized (or simplified) versions of Newton's Method for computing p-th roots of a given matrix A. We also discuss an efficient implementation, called the Newton-Kronecker-Schur scheme, for solving the Sylvester equation that appears at every Newton's iteration using the Kronecker product.

All the specialized versions are based on the commutativity of the iterates with the matrix A. Among the new specialized versions, we pay special attention to the one that produces the sequence Z_k. The Z_k scheme is not always stable, but it tends to achieve a much better approximation of the p-th root of A than the previously-known simplified schemes (denoted as Y_k and W_k), before the unstable behavior is observed. The unstable behavior appears when the commutativity between A and Z_k is lost.

Another specialized version that we present is the one denoted as V_k. We establish that the V_k scheme is stable for any p. Unfortunately, in some cases, the V_k scheme tends to stagnate before reaching an accurate approximation of the p-th root of A. This phenomenon is also due to the lack of commutativity between A and the iterates V_k, and appears in most specialized Newton versions known in the literature.

We also present some hybrid schemes that try to avoid the lack of commutativity combining the simplified methods with the Newton-Kronecker-Schur scheme. Although more experiments are needed to assess the effectiveness of

these hybrid schemes, the initial results on several matrices are encouraging.

Current work includes the search of a smoother and automatic way of combining the Z_k scheme with the Newton-Kronecker-Schur method. In this work we change from the Z_k scheme to the Newton-Kronecker-Schur method using a parameter δ, chosen ad-hoc. Future work should also include the study of an inexpensive way of projecting onto the set of matrices that commute with a given one. As observed in our experiments, an inexpensive projection of that kind could lead to very powerful hybrid schemes for the calculation of a p-th root of a given matrix.

References

[1] D. A. Bini, N. J. Higham and B. Meini [2005], Algorithms for the matrix p-th root, *Numerical Algorithms* 39, pp. 349–378.

[2] A. Bjorck and S. Hammarling [1983], A Schur methods for the square root of a matrix, *Linear Algebra and its Applications* 52/53, pp. 127–140.

[3] S. H. Cheng, N. J. Higham, C. S. Kenney and A. J. Laub [2001], Approximating the logarithm of a matrix to specified accuracy, *SIAM J. Matrix Anal. Appl.* 2, pp. 1112–1125.

[4] N. J. Higham [1986], Newton's methods for the matrix square root, *Mathematics of Computation* 46, pp. 537–549.

[5] N. J. Higham [1997], Stable iterations for the matrix square root, *Numer. Algorithms* 15, pp. 227–242.

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

[7] B. Iannazzo [2003], A note on computing the matrix square root, *Calcolo* 40, pp. 273–283.

[8] C. S. Kenney and A. J. Laub [1989], Padé error estimates for the logarithm of a matrix, *nternat. J. Control* 50, pp. 707–730.

[9] B. Meini [2004], The matrix square root from a new functional perspective: theoretical results and computational issues, *SIAM J. Matrix Anal. Appl.* 26, pp. 362–376.

[10] M. S. Plyushchay and M. Rausch de Traubenberg [2000], Cubic root of Klein-Gordon equation, *Phys. Lett. B* 477, pp. 276–284.

[11] A. Raspini [2000], Dirac equation with fractional derivatives of order 2/3, *FIZIKA B (Zagreb)* 9, pp. 49–54.

[12] L. S. Shieh, Y. T. Tsay and C. T. Wang [1984], Matrix sector functions and their application to system theory, *IEEE Proc.* 131, pp. 171–181.

[13] L. S. Shieh, S. R. Lian and B. C. Mcinnis [1987], Fast and stable algorithms for computing the principal square root of a complex matrix, *IEEE Transactions on Automatic Control* AC-32, pp. 820–822.

[14] L. S. Shieh, Y. T. Tsay and R. E. Yates [1985], Computation of the principal nth roots of complex matrices, *IEEE Transactions on Automatic Control* AC-30, pp. 606–608.

[15] M. I. Smith [2003], A Schur algorithm for computing matrix pth roots, *SIAM J. Matrix Anal. Appl.* 24, pp. 971–989.

[16] J. S. H. Tsai, L. S. Shieh, and R. E. Yates [1988], Fast and stable algorithms for computing the principal nth root of a Complex matrix and the matrix sector function, *Comput. Math. Applic.* 15, pp. 903–913.

[17] Y. T. Tsay, L. S. Shieh, and J. S. H. Tsai [1986], A fast method for computing the principal nth roots of complex matrices, *Linear Alg. Appl.* 76, pp. 205–221.

Received: February 6, 2008