On computing complex square roots of real matrices

Zhongyun Liu[a], Yulin Zhang[b],* Jorge Santos[c] and Rui Ralha[b]

[a] School of Math., Changsha University of Science & Technology, Hunan, 410076, China
[b] Centro de Matemática, Universidade do Minho, 4710-057 Braga, Portugal
[c] Departamento de Matemática, Universidade de Coimbra, 3001-454 Coimbra, Portugal

Abstract: We present an idea for computing complex square roots of matrices using only real arithmetic.

Keywords: Matrix square root, Schur algorithm, real arithmetic.
AMS Classifications: 65F30

1 Introduction

The computation of square roots of a real matrix A has been studied for many years by several authors, see [4, 8, 3, 9, 10] and references therein. An important class of algorithms rely on the Schur form of A and this is the case of the routine sqrtm in Matlab. This routine, which implements the method proposed by A. Björck and S. Hammarling's [3], uses the complex Schur form. N. Higham [8] has shown how to compute, in a stable way, a real square root using only real arithmetic. This algorithm is implemented in Higham's Matrix Function Toolbox routine sqrtm_real. However, for complex square roots this routine still requires complex arithmetic. In this paper, we present an idea for computing complex square roots using only real arithmetic. Throughout this paper, we consider only those square roots of A which are functions of A (as defined in [6], [12] or [11]).

2 Computing complex square roots of a real matrix

Assume that a function f is defined on the spectrum of A. Let

$$\hat{T} = \begin{bmatrix} \hat{T}_{11} & \hat{T}_{12} & \cdots & \hat{T}_{1m} \\ 0 & \hat{T}_{22} & \cdots & \hat{T}_{2m} \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \hat{T}_{mm} \end{bmatrix} \in \mathbb{R}^{n\times n}$$

be the real Schur form of A, where $\hat{T}_{ii}$ is either a real eigenvalue of A or a 2×2 block with complex conjugate eigenvalues of A. Then we have

$$f(A) = U f(\hat{T}) U^T. \tag{2.1}$$

*Corresponding author. Email: zhang@math.uminho.pt

For $f(\lambda) = \lambda^{1/2}$, the problem of computing a square root of A is reduced, due to (2.1), to that of computing a square root of $\hat{T}$. It is shown in [8] that $S = (\hat{T})^{1/2}$ is quasi-upper triangular and

$$S_{ii} = (\hat{T}_{ii})^{1/2}, \quad 1 \le i \le m \tag{2.2}$$

and

$$S_{ii}S_{ij} + S_{ij}S_{jj} = \hat{T}_{ij} - \sum_{k=i+1}^{j-1} S_{ik}S_{kj}, \quad j > i. \tag{2.3}$$

Moreover, it is shown in [8] that square roots of a 2×2 real matrix B with complex conjugate eigenvalues have specific structures which are depicted as follows.

Lemma 1 *[8, Lemma 2] Assume that $B \in \mathbb{R}^{2\times 2}$ has complex conjugate eigenvalues $\lambda, \bar{\lambda} = \mu \pm i\nu$ with $\nu \neq 0$. Then B has four square roots, each of which is a function of B. In particular, two of them are real, and the other two are pure imaginary.*

We note from (2.2) and (2.3) that once S_{ii}, $i = 1, \ldots, m$, are determined, then S_{ij}, for $j > i$, are uniquely determined. Furthermore, by Lemma 1, we have that S_{ii}, $i = 1, \ldots, m$, are either real or pure imaginary.

We will now describe our strategy to get any complex square using only real arithmetic. An important tool is a reordering of the diagonal blocks. In fact, from $\hat{T}$ one may get a similar Schur form T' with real eigenvalues and 2×2 blocks T'_{ii} in any prescribed order by an orthogonal similarity transformation. For example, if

$$\hat{T} = \begin{bmatrix} \hat{T}_{11} & \hat{T}_{12} & \hat{T}_{13} \\ & \hat{T}_{22} & \hat{T}_{23} \\ & & \hat{T}_{33} \end{bmatrix}$$

then there is an orthogonal similarity transformation Q such that

$$T' = Q\hat{T}Q^T = \begin{bmatrix} T'_{22} & T'_{12} & T'_{13} \\ & T'_{33} & T'_{23} \\ & & T'_{11} \end{bmatrix}$$

where $\hat{T}_{ii}$ is similar to $T'_{ii}, i = 1, 2, 3$. This can be achieved with an algorithm (see [1]) that is implemented in the LAPACK library and also in the Matlab's function ordschur.

Now, let us express the Schur form of A as in

$$T = \begin{bmatrix} T_1 & T_3 \\ & T_2 \end{bmatrix} \tag{2.4}$$

where T_1 glues those blocks for which real square roots will be produced using Higham's algorithm and T_2 stands for the blocks for which pure imaginary square roots are to be computed. Obviously, all those $\hat{T}_{ii}$ that are real negative

eigenvalues will belong to T_2 but we may also glue in T_2 some 2×2 blocks (if for some reason, the pure imaginary square roots of such blocks are needed). Let S_1 be the real square root of T_1 and iS_2 the pure imaginary square root of T_2. Then

$$S = \begin{bmatrix} S_1 & E + iF \\ & iS_2 \end{bmatrix}$$

is a square root of T, where S_2, E and F are real matrices. We have

$$\begin{bmatrix} T_1 & T_3 \\ & T_2 \end{bmatrix} = \begin{bmatrix} S_1 & E + iF \\ & iS_2 \end{bmatrix} \begin{bmatrix} S_1 & E + iF \\ & iS_2 \end{bmatrix}$$

and

$$T_3 = S_1E - FS_2 + i(S_1F + ES_2).$$

Since T_3 is real, we write

$$S_1F + ES_2 = 0 \tag{2.5}$$

and

$$S_1E - FS_2 = T_3 \tag{2.6}$$

Multiply (2.6) by S_1 and replace $-S_1F$ with ES_2, as it results from (2.5), to get

$$S_1^2E + ES_2^2 = S_1T_3 \tag{2.7}$$

and finally

$$T_1E - ET_2 = S_1T_3. \tag{2.8}$$

The solution of the Sylvester equation (2.8) gives E, then F comes from (2.6). Note that we may assume, without loss of generality, that the spectra of T_1 and T_2 are disjoint, as a result of arranging $\hat{T}_{ii}$ in a suitable manner. For reasons of stability of the numerical procedure, we may also gather in the same T_1 or T_2 those eigenvalues that are very close.

As in Higham's algorithm, given an arbitrary square real matrix, the initial step is to get the real Schur form (and this is in fact the most expensive part of the total process, see [11, p.136]). From this point, our algorithm can be presented as

Algorithm For a given nonsingular real matrix $\hat{T}$ in Schur form, this algorithm uses real arithmetic to compute a complex square root of $\hat{T}$.
Step 1. reorder the diagonal blocks of $\hat{T}$ to produce $T = P^T\hat{T}P = \begin{bmatrix} T_1 & T_3 \\ & T_2 \end{bmatrix}$.
Step 2. compute the square roots S_1 and S_2 using Higham's real algorithm for the quasi-upper triangular matrices T_1 and $-T_2$.
Step 3. solve Sylvester equation (2.8) to get E, and use (2.6) to produce F.
Step 4. compute $P \begin{bmatrix} S_1 & E \\ 0 & 0 \end{bmatrix} P^T + iP \begin{bmatrix} 0 & F \\ 0 & S_2 \end{bmatrix} P^T$ to get the square root of $\hat{T}$.

Now, we count the flops required in our complex arithmetic free algorithm. Let $r \leq n$ be the order of T_1.

- The cost of step 1 and 4 is comparatively small* and we may neglect it.
- The cost of step 2 is about $\frac{1}{3}(r^3 + (n-r)^3)$ real flops [11, p.136].
- In step 3, the computation of E involves the product $S_1 T_3$ and solving the Sylvester equation (2.8). The product takes $2r^2(n-r)$ flops and solving the equation requires about $r(n-r)n$ flops (see the Bartels–Stewart algorithm in [7, p.367] and [2] for triangular matrices; adapting this algorithm to our quasi-triangular matrices T_1 and T_2 causes only a slight increase in the number of flops). For F, computing $S_1 E - T_3$ and solving the equation take $r^2(n-r) + r(n-r)$ and $r(n-r)^2$, respectively. The total number of flops for step 3 is about $(n-r)r(3r+2n+1)$.

Therefore the total cost is about

$$C(r) = \frac{1}{3}(r^3 + (n-r)^3) + (n-r)r(3r+2n+1)$$

which amounts to $\frac{1}{3}n^3$ when $r = 0$, i.e., when the square is real (this is Higham's real Schur method). Also, we have $C(n) = \frac{1}{3}n^3$ which corresponds to the case of the square root being pure imaginary. The maximum of the cost C is attained for $r \approx \frac{n}{2}$ and $C\left(\frac{n}{2}\right) \approx 0.9n^3$, that is, about three times the number of operations required by Higham's algorithm that, in this case, requires complex arithmetic. The advantage of our proposal steams from the fact that it uses only real arithmetic.

3 A numerical example

The following results have been produced, in "format short", with Matlab 7.2.0.232 (R2006a) on Pentium M 1.6 GHz machine under Windows XP.

Let

$$A = \begin{bmatrix} -3.0003 & -2.9668 & -4.2832 & -8.4829 & -7.4019 \\ -6.0681 & 6.6166 & 5.1440 & -8.9210 & 1.3765 \\ -4.9783 & 1.7053 & 5.0746 & 0.6159 & -0.6122 \\ 2.3209 & 0.9945 & -2.3911 & 5.5833 & -9.7620 \\ -0.5342 & 8.3439 & 1.3564 & 8.6802 & -3.2575 \end{bmatrix}.$$

which has the following real Schur form

$$T = \begin{bmatrix} 13.208 & -5.9066 & -2.9196 & 0.16782 & 3.1015 \\ 0 & 5.1487 & 0.79683 & 3.5912 & 3.0922 \\ 0 & 0 & -3.8716 & 10.79 & 4.7058 \\ 0 & 0 & 0 & -1.734 & 14.707 \\ 0 & 0 & 0 & -6.9954 & -1.734 \end{bmatrix}.$$

*Swapping adjacent diagonal elements of T requires $20n$ flops, plus another $20n$ flops to update the Schur vectors, so the cost of the swapping is $40n$ times the number of swaps, see [5]. In case the number of swaps is greater than $O(n)$, then our Algorithm can be easily extended to one involving a multi-block structure. Therefore the total cost is usually small compared with the overall cost of the algorithm.

No real square root exists in this case. To compute a complex square root, our Algorithm now comes into play:
Step 1. reordering the the diagonal blocks of T gives

$$\begin{bmatrix} T_1 & T_3 \\ 0 & T_2 \end{bmatrix} = \left[\begin{array}{ccccc} 13.208 & -5.9066 & -1.2002 & 4.0267 & 0.71892 \\ 0 & 5.1487 & 3.553 & 2.6922 & 1.7947 \\ 0 & 0 & 2.231 & 12.079 & 9.7856 \\ 0 & 0 & -9.8185 & -5.6991 & 5.8545 \\ \hline 0 & 0 & 0 & 0 & -3.8716 \end{array}\right],$$

with

$$P = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0.46206 & -0.68738 & 0.56036 \\ 0 & 0 & 0.88685 & 0.35814 & -0.29196 \\ 0 & 0 & 0 & 0.63185 & 0.77509 \end{bmatrix}.$$

Step 2. Using Higham's algorithm, we get

$$S_1 = (T_1)^{\frac{1}{2}} = \begin{bmatrix} 3.6342 & -1.0006 & 0.20119 & 0.75793 \\ 0 & 2.2691 & 0.74099 & 0.1564 \\ 0 & 0 & 3.0269 & 2.9201 \\ 0 & 0 & -2.3735 & 1.1098 \end{bmatrix}$$

$$S_2 = T_2^{\frac{1}{2}} = \begin{bmatrix} 1.9676 \end{bmatrix}.$$

Step 3. solve Sylvester equation to get

$$E = \begin{bmatrix} -0.30442 \\ -0.061449 \\ 1.0861 \\ 3.3186 \end{bmatrix} \text{ and } F = \begin{bmatrix} 0.49298 \\ -0.31019 \\ 1.6225 \\ -2.4137 \end{bmatrix}.$$

Step 4. compute $P \begin{bmatrix} S_1 & E \\ 0 & 0 \end{bmatrix} P^T + iP \begin{bmatrix} 0 & F \\ 0 & S_2 \end{bmatrix} P^T$ to get the square root of T which is

$$\begin{bmatrix} 3.6342 & -1.0006 & -0.59861 & 0.53875 & 0.24295 \\ 0 & 2.2691 & 0.20044 & 0.7311 & 0.0511 \\ 0 & 0 & 0 & 3.4167 & -1.0086 \\ 0 & 0 & 0 & 2.0684 & 3.5552 \\ 0 & 0 & 0 & -1.6911 & 2.0683 \end{bmatrix} + i \begin{bmatrix} 0 & 0 & 0.27625 & -0.14393 & 0.3821 \\ 0 & 0 & -0.17382 & 0.0905 & -0.24043 \\ 0 & 0 & 1.9676 & -1.0252 & 2.7216 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

4 Acknowledgements

This research was financed by the National Natural Science Foundation of China under Grant No. 10771022 and DMS-03-53510, and Major Foundation of Educational Committee of Hunan Province under Grant No. 09A002 [2009], Also, supported by FEDER Funds through "Programa Operacional Factores de Competitividade - COMPETE" and by Portuguese Funds through FCT - "Fundação para a Ciência e a Tecnologia", within the Project PEst-C/MAT/UI0013/2011 and PTDC/MAT/112273/2009, Portugal.

References

[1] Zhaojun Bai and Lames W. Demmel *On swapping diagonal blocks in real schur form*, Linear Algebra Appl., 186: (1983), pp. 73–95.

[2] R. H. Bartels and G. W. Stewart, *Solution of equation* $AX + XB = C$, Comm. ACM 15 (1972), pp. 820-826.

[3] A. Björck and S. Hammarling, *A Schur method for the square root of a matrix*, Linear Algebra Appl., 52/53 (1983), pp. 127–140.

[4] G.W. Cross and P. Lancaster, *Square roots of complex matrices*, Linear and Multilinear Algebra, 1 (1974), pp. 289–293.

[5] P. I. Davis and N. J. Higham, *A Parlett-Schur algorithm for computing matrix functions*, SIAM J. Matrix Anal. Appl. Vol. 25 (2003), pp. 464-485.

[6] F. R. Gantmacher, *The Theory of Matrices*, Vol. 1, Chelsea, New York, 1959.

[7] G. H. Golub and C.F. Van Loan, *Matrix Computations*, Johns Hopkins University Press, Baltimore and London, 3rd edition, 1996.

[8] N. J. Higham, *Computing real square roots of a real matrix*, Linear Algebra Appl., 88/89 (1987), pp. 405–430.

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

[10] N.J. Higham, D.S. Mackey, N. Mackey, and F. Tisseur, *Function preserving matrix groups and iterations for the matrix square root*, SIAM J. Matrix Anal. Appl., 26(3) (2005): pp. 849–877.

[11] N.J. Higham, *Functions of matrices-theory and computation*, SIAM, 2008.

[12] P. Lancaster, *Theory of Matices*, Academic, New York, 1969.