

*A Recursive Blocked Schur Algorithm for
Computing the Matrix Square Root*

Deadman, Edvin and Higham, Nicholas J. and Ralha,
Rui

2012

MIMS EPrint: **2012.26**

Manchester Institute for Mathematical Sciences
School of Mathematics

The University of Manchester

Reports available from: <http://eprints.maths.manchester.ac.uk/>

And by contacting: The MIMS Secretary
School of Mathematics
The University of Manchester
Manchester, M13 9PL, UK

ISSN 1749-9097

A Recursive Blocked Schur Algorithm for Computing the Matrix Square Root

Edvin Deadman
University of Manchester
Numerical Algorithms Group
edvin.deadman@nag.co.uk

Nicholas J. Higham
University of Manchester
higham@maths.manchester.ac.uk

Rui Ralha
University of Minho, Portugal
r_ralha@math.uminho.pt

Abstract—The Schur method for computing a matrix square root reduces the matrix to the Schur triangular form and then computes a square root of the triangular matrix. We show that by using a recursive blocking technique the computation of the square root of the triangular matrix can be made rich in matrix multiplication. Numerical experiments making appropriate use of level 3 BLAS show significant speedups over the point algorithm, both in the square root phase and in the algorithm as a whole. The excellent numerical stability of the point algorithm is shown to be preserved by recursive blocking. These results are extended to the real Schur method. Recursive blocking is also shown to be effective for multiplying triangular matrices.

I. INTRODUCTION

A square root of matrix $A \in \mathbb{C}^{n \times n}$ is any matrix satisfying $X^2 = A$. Matrix square roots have many applications, including in Markov models of finance, the solution of differential equations and the computation of the polar decomposition and the matrix sign function [9].

A square root of a matrix (if one exists) is not unique. However, if A has no eigenvalues on the closed negative real line then there is a unique *principal square root* $A^{1/2}$ whose eigenvalues all lie in the open right half-plane. If A is real, then so is $A^{1/2}$. For proofs of these facts and more on the theory of matrix square roots see [9].

The most numerically stable way of computing matrix square roots is via the Schur method of Björck and Hammarling [3]. The matrix A is reduced to upper triangular form and a recurrence relation enables the square root of the triangular matrix to be computed a column or superdiagonal at a time. In §II we show that the recurrence can be reorganized using a standard blocking scheme or recursive blocking in order to make it rich in matrix multiplications. We show experimentally that significant speedups result when level 3 BLAS are exploited in the implementation. In §III we show that the blocked methods maintain the excellent backward stability of the non-blocked method. In §IV we discuss the use of the new approach within the Schur method and explain how it can be extended to the real Schur method of Higham [7]. We compare our implementations written for the NAG Library with existing MATLAB functions. Finally, in §V we discuss some further applications of recursive blocking for triangular matrices.

II. THE USE OF BLOCKING IN THE SCHUR METHOD

To compute $A^{1/2}$, a Schur decomposition $A = QTQ^*$ is obtained, where T is upper triangular and Q is unitary. Then $A^{1/2} = QT^{1/2}Q^*$. For the remainder of this section we will focus on upper triangular matrices only. The equation

$$U^2 = T \quad (1)$$

can be solved by noting that U is also upper triangular, so that by equating elements,

$$U_{ii}^2 = T_{ii}, \quad (2)$$

$$U_{ii}U_{ij} + U_{ij}U_{jj} = T_{ij} - \sum_{k=i+1}^{j-1} U_{ik}U_{kj}. \quad (3)$$

These equations can be solved either a column or a superdiagonal at a time. Different choices of sign in the scalar square roots of (2) lead to different matrix square roots. This method will be referred to hereafter as the “point” method.

The algorithm can be blocked by letting the U_{ij} and T_{ij} in (2) and (3) refer to $m \times m$ blocks, where $m \ll n$. The diagonal blocks U_{ii} are then obtained using the point method and the off-diagonal blocks are obtained by solving the Sylvester equations (3) using LAPACK routine xTRSYL (where ‘x’ denotes D or Z according to whether real or complex arithmetic is used) [2]. Level 3 BLAS can be used in computing the right-hand side of (3) so significant improvements in efficiency are expected. This approach is referred to as the block method.

To test this approach, a Fortran implementation was written and compiled with gfortran on a 64 bit Intel i3 dual-core machine, using the ACML Library for LAPACK and BLAS calls. Complex upper triangular matrices were generated, with random elements whose real and imaginary parts were chosen from the uniform distribution on $[0, 1)$. Figure 1 shows the run times for the methods, for various different sized matrices. A block size of 32 was chosen, although the speed did not appear to be particularly sensitive to the block size—similar results were obtained with blocks of size 16, 64 and 128. The block method was found to be up to 17 times faster than the point method. The residuals $\|\widehat{U}^2 - T\|/\|T\|$, where \widehat{U} is the computed value of U , were similar for both methods. Table II shows that, for $n = 4096$, over 90% of the run time is spent in ZGEMM calls.

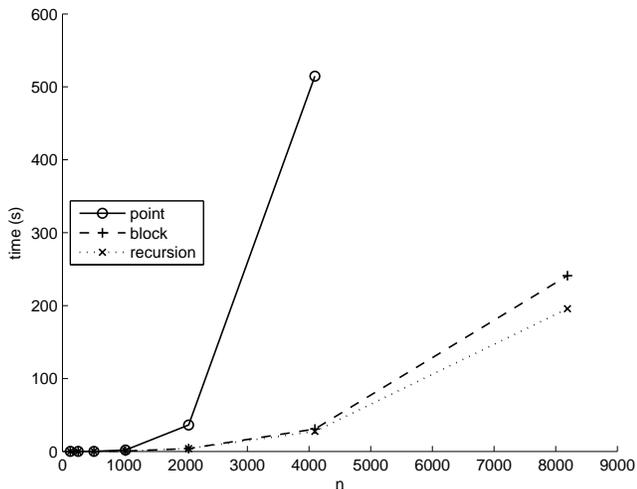


Fig. 1. Run times for the point, block and recursion methods for computing the square root of a complex $n \times n$ triangular matrix.

A larger block size enables larger GEMM calls to be made. However, it leads to larger calls to the point algorithm and to xTRSYL (which only uses level 2 BLAS). A recursive approach may allow increased use of level 3 BLAS.

Equation (1) can be rewritten as

$$\begin{pmatrix} U_{11} & U_{12} \\ 0 & U_{22} \end{pmatrix}^2 = \begin{pmatrix} T_{11} & T_{12} \\ 0 & T_{22} \end{pmatrix}, \quad (4)$$

where the submatrices are of size $n/2$ or $(n \pm 1)/2$ depending on the parity of n . Then $U_{11}^2 = T_{11}$ and $U_{22}^2 = T_{22}$ can be solved recursively, until some base level is reached, at which point the point algorithm is used. The Sylvester equation $U_{11}U_{12} + U_{12}U_{22} = T_{12}$ can then be solved using a recursive algorithm devised by Jonsson and Kågström [11]. In this algorithm, the Sylvester equation $AX + XB = C$, with A and B triangular, is written as

$$\begin{pmatrix} A_{11} & A_{12} \\ 0 & A_{22} \end{pmatrix} \begin{pmatrix} X_{11} & X_{12} \\ X_{21} & X_{22} \end{pmatrix} + \begin{pmatrix} X_{11} & X_{12} \\ X_{21} & X_{22} \end{pmatrix} \begin{pmatrix} B_{11} & B_{12} \\ 0 & B_{22} \end{pmatrix} = \begin{pmatrix} C_{11} & C_{12} \\ C_{21} & C_{22} \end{pmatrix},$$

where each submatrix is of size $n/2$ or $(n \pm 1)/2$. Then

$$A_{11}X_{11} + X_{11}B_{11} = C_{11} - A_{12}X_{21}, \quad (5)$$

$$A_{11}X_{12} + X_{12}B_{22} = C_{12} - A_{12}X_{22} - X_{11}B_{12}, \quad (6)$$

$$A_{22}X_{21} + X_{21}B_{11} = C_{21}, \quad (7)$$

$$A_{22}X_{22} + X_{22}B_{22} = C_{22} - X_{21}B_{12}. \quad (8)$$

Equation (7) is solved recursively, followed by (5) and (8), and finally (6). At the base level a routine such as xTRSYL is used.

The run times for a Fortran implementation of the recursion method in complex arithmetic, with a base level of size 32, are shown in Figure 1. The approach was found to be faster than the block method, and up to 19 times faster than the

TABLE I
Profiling of the block method for computing the square root of a triangular matrix, with $n = 4096$. Format: *time in seconds (number of calls)*.

Total time taken:	30.74
Calls to looping method:	0.009 (128)
Calls to ZTRSYL	2.40 (8128)
Calls to ZGEMM with $n = 32$:	28.16 (341376)

TABLE II
Profiling of the recursive method for computing the square root of a triangular matrix, with $n = 4096$. Format: *time in seconds (number of calls)*.

Total time taken:	27.40
Calls to looping method:	0.006 (128)
Calls to ZTRSYL	2.22 (8128)
Calls to ZGEMM total:	22.06 (10668)
Calls to ZGEMM with $n = 1024$	8.01 (4)
Calls to ZGEMM with $n = 512$	6.19 (24)
Calls to ZGEMM with $n = 256$	3.82 (112)
Calls to ZGEMM with $n = 128$	2.13 (480)
Calls to ZGEMM with $n = 64$	1.18 (1984)
Calls to ZGEMM with $n = 32$	0.72 (8064)

point method, with similar residuals in each case. The precise choice of base level made little difference to the run time.

Table II shows that the run time is dominated by GEMM calls and that the time spent in ZTRSYL and the point algorithm is similar to the block method. The largest GEMM call uses a submatrix of size $n/4$.

III. STABILITY OF THE BLOCKED ALGORITHMS

We use the standard model of floating point arithmetic [8, §2.2] in which the result of a floating point operation, op , on two scalars x and y is written as

$$fl(x \text{ op } y) = (x \text{ op } y)(1 + \delta), \quad |\delta| \leq u,$$

where u is the unit roundoff. In analyzing a sequence of floating point operations it is useful to write [8, §3.4]

$$\prod_{i=1}^n (1 + \delta_i)^{\rho_i} = 1 + \theta_n, \quad \rho_i = \pm 1,$$

where

$$|\theta_n| \leq \frac{nu}{1 - nu} =: \gamma_n.$$

It is also convenient to define $\tilde{\gamma}_n = \gamma_{cn}$ for some small integer c whose precise value is unimportant. We use a hat to denote a computed quantity and write $|A|$ for the matrix whose elements are the absolute values of the elements of A .

Björck and Hammarling [3] obtained a normwise backward error bound for the Schur method. The computed square root \hat{X} of the full matrix A satisfies $\hat{X}^2 = A + \Delta A$, where

$$\|\Delta A\|_F \leq \tilde{\gamma}_n \|\hat{X}\|_F^2. \quad (9)$$

Higham [9, §6.2] obtained a componentwise bound for the triangular phase of the algorithm. The computed square root \hat{U} of the triangular matrix T satisfies $\hat{U}^2 = T + \Delta T$, where

$$|\Delta T| \leq \tilde{\gamma}_n |\hat{U}|^2. \quad (10)$$

This bound implies (9). We now investigate whether the bound (10) still holds when the triangular phase of the algorithm is blocked.

Consider the Sylvester equation $AX + XB = C$ in $n \times n$ matrices with triangular A and B . When it is solved in the standard way by the solution of n triangular systems the residual of the computed \hat{X} satisfies [8, §16.1]

$$|C - (A\hat{X} + \hat{X}B)| \leq \tilde{\gamma}_n(|A||\hat{X}| + |\hat{X}||B|). \quad (11)$$

In the (non-recursive) block method, to bound ΔT_{ij} we must account for the error in performing the matrix multiplications on the right-hand side of (3). Standard error analysis for matrix multiplication yields, for blocks of size m ,

$$\left| fl \left(\sum_{k=i+1}^{j-1} \hat{U}_{ik} \hat{U}_{kj} \right) - \sum_{k=i+1}^{j-1} \hat{U}_{ik} \hat{U}_{kj} \right| \leq \tilde{\gamma}_n |\hat{U}|_{ij}^2.$$

Substituting this into the residual for the Sylvester equation in the off-diagonal blocks, we obtain the componentwise bound (10).

To obtain a bound for the recursive blocked method we must first check if (11) holds when the Sylvester equation is solved using Jonsson and Kågström’s recursive algorithm. This can be done by induction, assuming that (11) holds at the base level. For the inductive step, it suffices to incorporate the error estimates for the matrix multiplications in the right hand sides of (5)–(8) into the residual bound.

Induction can then be applied to the recursive blocked method for the square root. The bounds (10) and (11) are assumed to hold at the base level. The inductive step is similar to the analysis for the block method. Overall, (10) is obtained.

We conclude that both our blocked algorithms for computing the matrix square root satisfy backward error bounds of the same forms (9) and (10) as the point algorithm.

IV. IMPLEMENTATION

When used with full (non-triangular) matrices, more modest speedups are expected because of the significant overhead in computing the Schur decomposition. Figure 2 compares run times of the MATLAB function `sqrtm` (which does not use any blocking) with Fortran implementations of the recursive blocked method (`fort_recurse`) and the point algorithm (`fort_point`), called from within MATLAB using a mex interface. The recursive routine is found to be up to 2.5 times faster than `sqrtm` and 2 times faster than `fort_point`.

An extension of the Schur method due to Higham [7] enables the square root of a real matrix to be computed without using complex arithmetic. A real Schur decomposition of A is computed. Square roots of the 2×2 diagonal blocks of the upper quasi-triangular factor are computed using an explicit formula. The recursion (3) now proceeds either a block column or a block superdiagonal at a time, where the blocks are of size 1×1 , 1×2 , 2×1 or 2×2 depending on the diagonal block structure. A MATLAB implementation of this algorithm `sqrtm_real` is available in the Matrix Function Toolbox [6]. The algorithm can also be implemented in a recursive

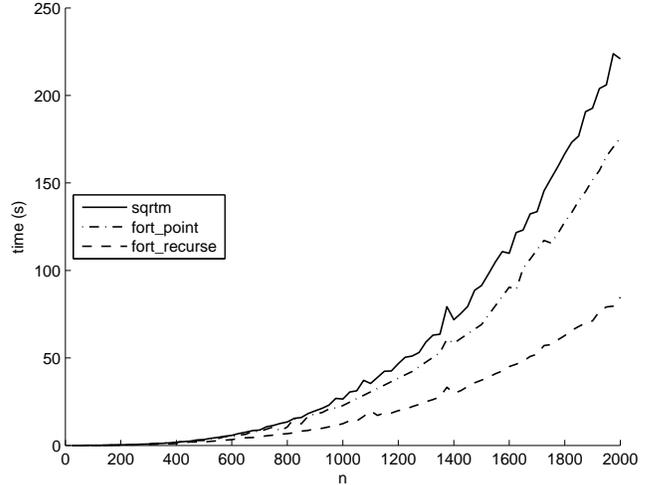


Fig. 2. Run times for `sqrtm`, `fort_recurse` and `fort_point` for computing the square root of a full $n \times n$ matrix with elements whose real and imaginary parts are chosen from the uniform random distribution on the interval $[0, 1)$.

manner, the only subtlety being that the “splitting point” for the recursion must be chosen to avoid splitting any 2×2 diagonal blocks. A similar error analysis to §III applies to the real recursive method, though since only a normwise bound is available for the point algorithm applied to the quasi-triangular matrix the backward error bound (10) holds in the Frobenius norm rather than elementwise.

Figure 3 compares the run times of `sqrtm` and `sqrtm_real` with Fortran implementations of the real recursive method (`fort_recurse_real`) and the real point method (`fort_point_real`), also called from within MATLAB. The recursive routine is found to be up to 6 times faster than `sqrtm` and `sqrtm_real` and 2 times faster than `fort_point_real`.

Both the real and complex recursive blocked routines spend over 90% of their run time in computing the Schur decomposition, compared with 25% for `sqrtm`, 44% for `fort_point`, 16% for `sqrtm_real` and 46% for `fort_point_real`. The latter four percentages reflect the overhead of the MATLAB interpreter in executing the recurrences for the (quasi-)triangular square root phase.

V. FURTHER APPLICATIONS OF RECURSIVE BLOCKING

We briefly mention two further applications of recursive blocking schemes.

Currently there are no LAPACK or BLAS routines designed specifically for multiplying two triangular matrices, $T = UV$ (the closest is the BLAS routine `xTRMM` which multiplies a triangular matrix by a full matrix). However, a block algorithm is easily derived by partitioning the matrices into blocks. The product of two off-diagonal blocks is computed using `xGEMM`. The product of an off-diagonal block and a diagonal block is computed using `xTRMM`. Finally the point method is used when multiplying two diagonal blocks.

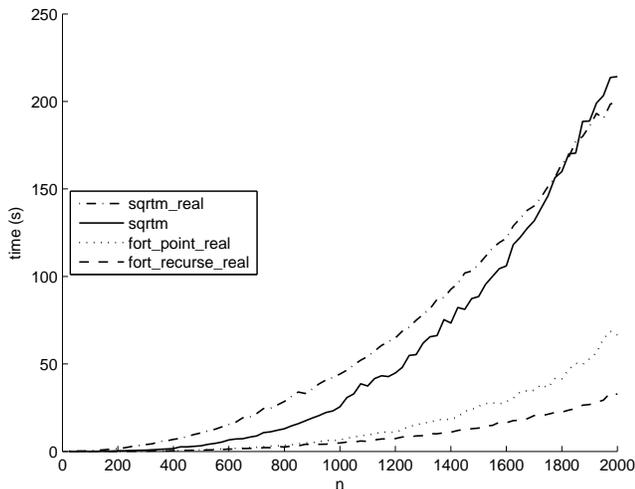


Fig. 3. Run times for `sqrtm`, `sqrtm_real`, `fort_recurse_real` and `fort_point_real` for computing the square root of a full $n \times n$ matrix with elements chosen from the uniform random distribution on $[0, 1)$.

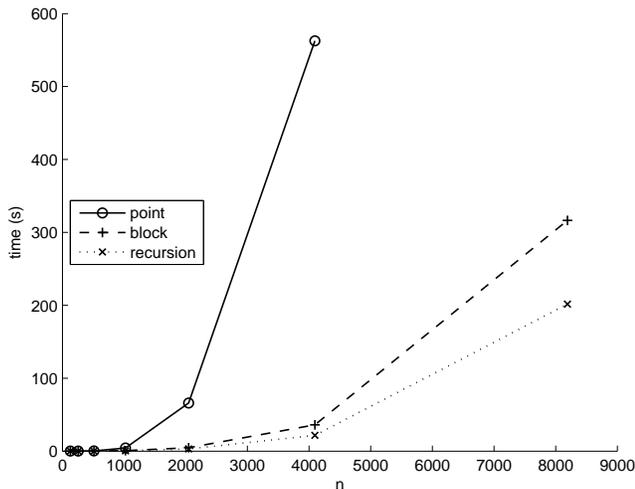


Fig. 4. Run times for the point, block and recursion methods for multiplying randomly generated triangular matrices.

In the recursive approach, $T = UV$ is rewritten as

$$\begin{pmatrix} T_{11} & T_{12} \\ 0 & T_{22} \end{pmatrix} = \begin{pmatrix} U_{11} & U_{12} \\ 0 & U_{22} \end{pmatrix} \begin{pmatrix} V_{11} & V_{12} \\ 0 & V_{22} \end{pmatrix}.$$

Then $T_{11} = U_{11}V_{11}$ and $T_{22} = U_{22}V_{22}$ are computed recursively and $T_{12} = U_{11}V_{12} + U_{12}V_{22}$ is computed using two calls to `xTRMM`.

Figure 4 shows run times for some triangular matrix multiplications using Fortran implementations of the point method, standard blocking and recursive blocking (the block size and base levels were both 32 in this case, although the results were not too sensitive to the precise choice of these parameters). As for the matrix square root, the block algorithms significantly outperform the point algorithm, with the recursive approach performing the best.

The inverse of a triangular matrix can be computed recursively, by expanding $UU^{-1} = I$ as

$$\begin{pmatrix} U_{11} & U_{12} \\ 0 & U_{22} \end{pmatrix} \begin{pmatrix} (U^{-1})_{11} & (U^{-1})_{12} \\ 0 & (U^{-1})_{22} \end{pmatrix} = \begin{pmatrix} I & 0 \\ 0 & I \end{pmatrix}.$$

Then $(\hat{U}^{-1})_{11}$ and $(\hat{U}^{-1})_{22}$ are computed recursively and $(\hat{U}^{-1})_{12}$ is obtained by solving $U_{11}(\hat{U}^{-1})_{12} + U_{12}(\hat{U}^{-1})_{22} = 0$. Provided that forward substitution is used, the right (or left) recursive inversion method can be shown inductively to satisfy the same right (or left) elementwise residual bound as the point method [4]. A Fortran implementation of this idea was found to perform similarly to LAPACK code `xTRTRI`, so no real benefit was derived from recursive blocking.

VI. CONCLUSIONS

We investigated two different blocking techniques within Björck and Hammarling's recurrence for computing a square root of a triangular matrix, finding that recursive blocking gives the best performance. Neither approach entails any loss of backward stability. We implemented the recursive blocking with both the Schur method and the real Schur method (which works entirely in real arithmetic) and found the new codes to be significantly faster than corresponding point codes, which include the MATLAB functions `sqrtm` (built-in) and `sqrtm_real` (from [6]). The new codes will appear in the next mark of the NAG Library [12]. Recursive blocking is also fruitful for multiplying triangular matrices.

Because of the importance of the (quasi-) triangular square root, which arises in algorithms for computing the matrix logarithm [1], matrix p th roots [5], and arbitrary matrix powers [10], this computational kernel is a strong contender for inclusion in any future extensions of the BLAS.

REFERENCES

- [1] A. H. Al-Mohy and N. J. Higham. Improved inverse scaling and squaring algorithms for the matrix logarithm. MIMS EPrint 2011.83, The University of Manchester, Oct. 2011.
- [2] E. Anderson, Z. Bai, C. Bischof, S. Blackford, J. Demmel, J. Dongarra, J. Du Croz, A. Greenbaum, S. Hammarling, A. McKenney, and D. Sorensen. *LAPACK Users' Guide*. Society for Industrial and Applied Mathematics, Philadelphia, PA, third edition, 1999.
- [3] Å. Björck and S. Hammarling. A Schur method for the square root of a matrix. *Linear Algebra Appl.*, 52/53:127–140, 1983.
- [4] J. J. Du Croz and N. J. Higham. Stability methods for matrix inversion. *IMA J. Numer. Anal.*, 12(1):1–19, 1992.
- [5] C.-H. Guo and N. J. Higham. A Schur–Newton method for the matrix p th root and its inverse. *SIAM J. Matrix Anal. Appl.*, 28(3):788–804, 2006.
- [6] N. J. Higham. The Matrix Function Toolbox. <http://www.ma.man.ac.uk/~higham/mfttoolbox>.
- [7] N. J. Higham. Computing real square roots of a real matrix. *Linear Algebra Appl.*, 88/89:405–430, 1987.
- [8] N. J. Higham. *Accuracy and Stability of Numerical Algorithms*. SIAM, 2nd edition, 2002.
- [9] N. J. Higham. *Functions of Matrices: Theory and Computation*. SIAM, 2008.
- [10] N. J. Higham and L. Lin. A Schur–Padé algorithm for fractional powers of a matrix. *SIAM J. Matrix Anal. Appl.*, 32(3):1056–1078, 2011.
- [11] I. Jonsson and B. Kågström. Recursive blocked algorithms for solving triangular systems - part I: One-sided and coupled Sylvester-type matrix equations. *ACM Trans. Math. Software*, 28(4):392–415, 2002.
- [12] Numerical Algorithms Group. The NAG Fortran Library. <http://www.nag.co.uk>.