Fast Polar Decomposition of an Arbitrary Matrix

Nicholas Higham

88-942
October 1988

Department of Computer Science
Cornell University
Ithaca, NY 14853-7501

Fast Polar Decomposition of an Arbitrary Matrix

Nicholas J. Higham *

Department of Computer Science
Upson Hall
Cornell University
Ithaca, New York 14853

Robert S. Schreiber **

RIACS
NASA Ames Research Center
Moffett Field
CA 94035

1980 Mathematics Subject Classification: Primary 65F05.
Computing Reviews: G.1.3.

Key words: Polar decomposition, complete orthogonal decomposition, matrix square root, matrix multiplication, Schulz iteration, condition estimator.

---

* On leave from the University of Manchester.
** Partially supported by ONR under contract no. N00014-86-K-0610.

Abstract

The polar decomposition of an $m \times n$ matrix A of full rank, where $m \geq n$, can be computed using a quadratically convergent algorithm of Higham [SIAM J. Sci. Stat. Comput., 7 (1986), pp.1160–1174]. The algorithm is based on a Newton iteration involving a matrix inverse. We show how with the use of a preliminary complete orthogonal decomposition the algorithm can be extended to arbitrary A. We also describe how to use the algorithm to compute the positive semi-definite square root of a Hermitian positive semi-definite matrix. We formulate a hybrid algorithm which adaptively switches from the matrix inversion based iteration to a matrix multiplication based iteration due to Kovarik, and to Björck and Bowie. The decision when to switch is made using a condition estimator. This "matrix multiplication rich" algorithm is shown to be more efficient on machines for which matrix multiplication can be executed 1.5 times faster than matrix inversion.

1. Introduction

A polar decomposition of a matrix $A \in \mathbf{C}^{m\times n}$ is a factorization $A = UH$, where $H \in \mathbf{C}^{n\times n}$ is Hermitian positive semi-definite and $U \in \mathbf{C}^{m\times n}$ is unitary; here we define unitary to mean that U has orthonormal rows or columns according as $m \le n$ or $m \ge n$. The decomposition always exists, H is the unique Hermitian positive semi-definite square root of A^*A (i.e. $H = (A^*A)^{1/2}$), and U is unique if and only if A has full rank (these properties are proved in section 2).

The polar decomposition is well-known in the case $m \ge n$; see [7, 10] for example. We have followed Horn and Johnson [13] in extending the definition to $m \le n$. The consistency of the definition can be seen in the result that for any m and n the unitary polar factor U is a nearest unitary matrix to A in the Frobenius norm (this is a straightforward extension of a result from [6]). Because of the role it plays in solving this and other nearness problems, computation of the polar decomposition is required in several applications [12]. A recent application, which motivated the work here, is the computation of block reflectors (generalizations of Householder matrices) [17]. Here, the polar decomposition of an arbitrary matrix must be computed, and it is desirable to do this efficiently on vector and parallel computers.

The polar decomposition can be obtained directly from the singular value decomposition (SVD). Higham [10] describes an alternative approach based on a Newton iteration involving a matrix inverse. The iteration is defined for square, nonsingular matrices only, but in [10] it is pointed out how a preliminary QR decomposition enables the treatment of $A \in \mathbf{C}^{m\times n}$ with $m \ge n$ and $\operatorname{rank}(A) = n$. It is also shown in [10] how the iteration can be used to compute the square root of a Hermitian positive definite matrix. According to the traditional model of computational cost based on operation counts, the iterative algorithm is generally of similar expense to the SVD approach, but is much more efficient when the matrix is nearly unitary. In an attempt to improve the performance of the iterative algorithm on machines which execute matrix multiplication at high efficiency, Schreiber and Parlett [17] propose the use of an inner Schulz iteration to compute most of the matrix inverses; they show that this leads to an increase in efficiency if matrix multiplication can be done at a rate 6.8 times faster than matrix inversion.

The purpose of this work is two-fold. First, we extend the algorithm of [10] so that it is applicable to arbitrary A. Our technique is to use an initial complete orthogonal decomposition so as to extract an appropriate square, nonsingular matrix. One might say that the complete orthogonal decomposition is to the polar decomposition what Chan's preliminary QR factorization is to the SVD! We also show how to use the algorithm of [10] to compute the square root of a (singular) Hermitian positive semi-definite matrix. Second, we introduce a modification of the Schulz inner iteration idea of [17] which reduces the cutoff ratio of multiplication speed to inversion speed from 6.8 to 2, or to 1.5 if advantage is taken of a symmetric matrix product.

2. Iterative Polar Decomposition of an Arbitrary Matrix

The basic algorithm of [10] is as follows. It converges quadratically for any square, nonsingular A. We use a MATLAB-like algorithmic notation, and denote by A^{-*} the conjugate transpose of A^{-1}.

Algorithm 2.1: $[U, H] = \text{polar.square}(A, \delta)$

% Input arguments: square, nonsingular A; convergence tolerance δ.

% Output arguments: U, H.

$X_0 = A$; $k = -1$

repeat

$k = k + 1$

$\gamma_k = \left(\|X_k^{-1}\|_1 \|X_k^{-1}\|_\infty / (\|X_k\|_1 \|X_k\|_\infty)\right)^{1/4}$

$X_{k+1} = \frac{1}{2}\left(\gamma_k X_k + \frac{1}{\gamma_k} X_k^{-*}\right)$

until $\|X_{k+1} - X_k\|_1 \leq \delta \|X_{k+1}\|_1$

$U = X_{k+1}$

$H = \frac{1}{2}(U^* A + A^* U)$ ∎

To adapt the algorithm to arbitrary $A \in \mathbf{C}^{m\times n}$ we begin by computing a complete orthogonal decomposition (COD)

$$A = P \begin{bmatrix} R & 0 \\ 0 & 0 \end{bmatrix} Q^*,$$

where $P \in \mathbf{C}^{m\times m}$ and $Q \in \mathbf{C}^{n\times n}$ are unitary, and $R \in \mathbf{C}^{r\times r}$ is nonsingular and upper triangular (we exclude the trivial case $A = 0$, for which R is empty). This decomposition may be computed using a QR factorization with column pivoting followed by a further Householder reduction step; see [8, p.169] for the details. Now we apply Algorithm 2.1 to R, obtaining $R = U_R H_R$, and we "piece together" the polar factors of A. We have

$$\begin{aligned} A &= P \begin{bmatrix} U_R H_R & 0 \\ 0 & 0 \end{bmatrix} Q^* \\ &= P \begin{bmatrix} U_R & 0 \\ 0 & I_{m-r,n-r} \end{bmatrix} \begin{bmatrix} H_R & 0 \\ 0 & 0 \end{bmatrix} Q^* \\ &= P \begin{bmatrix} U_R & 0 \\ 0 & I_{m-r,n-r} \end{bmatrix} Q^* \cdot Q \begin{bmatrix} H_R & 0 \\ 0 & 0 \end{bmatrix} Q^* \\ &\equiv UH, \end{aligned}$$

where $I_{m-r,n-r}$ denotes the $(m-r)\times(n-r)$ identity matrix. Note that $I_{m-r,n-r}$ could be replaced by any unitary matrix of the same dimensions; this shows the non-uniqueness of the unitary polar factor when $r = \mathrm{rank}(A) < \min(m,n)$. Note also that even though $U^*U \neq I$ when $m < n$, $H = U^*UH$ for all m and n; thus $A^*A = HU^*UH = H^2$, so that $H = (A^*A)^{1/2}$.

In evaluating U and H advantage can be taken of the zero blocks in the products. Denoting by Q_1 the first r columns of Q we have

$$H = Q_1 H_R Q_1^*. \tag{2.1}$$

For U we partition

$$P = \begin{array}{cc} r & m-r \\ (P_1, & P_2\,) \end{array}$$

and we distinguish the two cases

$$\text{(2.2}a\text{)}\quad m \geq n \quad \Rightarrow \quad U = [P_1,\ P_2]\begin{bmatrix} U_R & 0 \\ 0 & \begin{bmatrix} I_{n-r} \\ 0 \end{bmatrix} \end{bmatrix} Q^* = [P_1U_R,\ \ P_2\begin{bmatrix} I_{n-r} \\ 0 \end{bmatrix}]Q^*,$$

$$\text{(2.2}b\text{)}\quad m \leq n \quad \Rightarrow \quad U = [P_1,\ P_2]\begin{bmatrix} U_R & 0 \\ 0 & [\,I_{m-r},\ \ 0\,] \end{bmatrix} Q^* = [P_1U_R,\ \ P_2\,[\,I_{m-r},\ \ 0]]Q^*,$$

in which, respectively, the last $m-n$ columns of P_2 and the last $n-m$ rows of Q^* need not participate in the multiplication.

To summarise, we have the following algorithm:

Algorithm 2.2: $[U,H] = \mathrm{polar}\,(A,\epsilon,\delta)$
% $A \neq 0$ is arbitrary.
$[P,R,Q] = \mathrm{COD}(A,\epsilon)$
$[U_R,H_R] = \mathrm{polar.square}\,(R,\delta)$
Form U, H according to (2.1) and (2.2). ∎

As the notation indicates, in floating point arithmetic a tolerance ϵ is required for the complete orthogonal decomposition in order to determine a numerical rank (i.e. the dimension of R). The natural approach is to set to zero all rows of the trapezoidal QR factor of A which are negligible (in some measure) relative to ϵ (see [8, p.166]).

The choice of ϵ is important, since a small change in ϵ can produce a large change in the computed U when A is rank-deficient. However, a redeeming feature is that whatever the choice of ϵ, and irrespective of how well the QR factorization reveals rank, Algorithm 2.2 is stable, that is, the computed polar factors $\widehat{U}$, $\widehat{H}$ satisfy

$$\widehat{U}\widehat{H} = A + E,$$

where $\|E\|_{\mathrm{F}} \approx \max\{\epsilon, \delta\}\|A\|_{\mathrm{F}}$; this follows from the empirical stability of Algorithm 2.1 (see [10]) together with the stability of the additional orthogonal transformations in Algorithm 2.2.

The operation count of Algorithm 2.2 breaks down as follows, using the "flop" notation [8, p.32]. The complete orthogonal decomposition requires $2mnr - r^2(m+n) + 2r^3/3 + r^2(n-r)$ flops [8, pp.165,170]. Algorithm 2.1 requires, typically, 8 iterations (assuming $\delta \approx 10^{-16}$), and hence $(7+\frac{2}{3})r^3$ flops (taking into account the triangularity of R). And formation of H and U requires at most $nr^2 + n^2r/2$ and $mr^2 + \max\{nm^2, mn^2\}$ flops respectively. By comparison, computing a polar decomposition via the Golub-Reinsch SVD algorithm requires approximately $8mn^2 + 25n^3/6$ flops when $m \geq n$. The Golub-Reinsch SVD algorithm does not take advantage of rank-deficiency, although it could be modified to do so by using an initial complete orthogonal decomposition as above.

Of course, operation counts are not always a reliable guide to the actual computational cost on modern vector and parallel computers. An alternative performance indicator is the amount of matrix multiplication in an algorithm, since matrix multiplication can be performed very efficiently on many modern computers [1, 2, 16, 18]. As we will see in the next section, Algorithm 2.1 can be modified so that it is rich in matrix multiplication. In the complete orthogonal decomposition in Algorithm 2.2 the second Householder reduction step can be accomplished using the matrix multiplication rich WY representation of [2, 18]. In the initial QR factorization effective use of the WY representation is precluded by the column pivoting. One alternative is to use Bischof's local pivoting and incremental condition estimation technique [3], which does not hinder exploitation of the WY form. Another alternative is to compute a QR factorization *without* pivoting, and then to apply Chan's post-processing algorithm [5] for obtaining a rank-revealing QR factorization.

Finally we show how to use Algorithm 2.1 to compute the Hermitian positive semi-definite square root of a Hermitian positive semi-definite $A \in \mathbf{C}^{n\times n}$. First, we compute a Cholesky decomposition with pivoting,

$$\Pi^T A \Pi = R^* R, \qquad R = \begin{bmatrix} R_{11} & R_{12} \\ 0 & 0 \end{bmatrix},$$

where $R_{11} \in \mathbf{C}^{r\times r}$ is nonsingular and upper triangular. Then Householder transformations are used to zero R_{12} (as in the complete orthogonal decomposition):

$$U^* \Pi^T A \Pi U = \begin{bmatrix} T_{11}^* \\ 0 \end{bmatrix} \begin{bmatrix} T_{11} & 0 \end{bmatrix}, \qquad T_{11} \in \mathbf{C}^{r\times r} \text{ upper triangular.}$$

Next, Algorithm 2.1 applied to T_{11} yields $T_{11} = U_T H_T$, whence, with $Q = \Pi U$,

$$A = Q \begin{bmatrix} H_T^2 & 0 \\ 0 & 0 \end{bmatrix} Q^* = \left(Q \begin{bmatrix} H_T & 0 \\ 0 & 0 \end{bmatrix} Q^* \right)^2 \equiv X^2.$$

Square roots of semi-definite matrices are required in some statistical applications [9]. An alternative to this polar decomposition approach is to make use of an eigendecomposition; the relative merits are similar to those discussed above for the SVD.

3. A Hybrid Iteration

To make Algorithm 2.1 rich in matrix multiplication rather than matrix inversion Schreiber and Parlett [17] use an inner Schulz iteration

$$Z_{j+1} = Z_j + \left(I - Z_j X_k\right) Z_j, \qquad Z_0 = X_{k-1}^{-1}, \tag{3.1}$$

to compute X_k^{-1} on all iterations after the first. This approach takes advantage of the fact that since the X_k are converging quadratically, X_{k-1}^{-1} is an increasingly good approximation to X_k^{-1}. The Schulz iteration (3.1) is a Newton iteration and so also converges quadratically. Schreiber and Parlett observe that for the matrices in their application (which are often well-conditioned) the typical number of inner iterations required for convergence is 6, 5, 3, 2, 1, leading to 17 iterations in total, or 34 matrix multiplications. If the matrix inverses were computed directly, five inverses would be needed. This suggests that the modified algorithm will be faster than Algorithm 2.1 if matrix multiplication can be done at a rate 34/5 times faster than matrix inversion.

Further experimentation with the inner Schulz iteration led us to feel that it is unnecessary to run the inner iteration to convergence, and we considered employing just *one* Schulz iteration, with the starting matrix $Z_0 = X_k^*$ ($\approx X_k^{-1}$ since X_k converges to a unitary matrix). Thus the basic iteration

$$X_{k+1} = \tfrac{1}{2}\left(\gamma_k X_k + \frac{1}{\gamma_k} X_k^{-*}\right) \tag{3.2}$$

is replaced by (setting $\gamma_k = 1$)

$$\begin{aligned} X_{k+1} &= \tfrac{1}{2}\Big(X_k + \left(Z_0^* + Z_0^*(I - Z_0 X_k)^*\right)\Big) \\ &= X_k\Big(I + \tfrac{1}{2}(I - X_k^* X_k)\Big). \end{aligned} \tag{3.3}$$

This is precisely the quadratically convergent iteration of Kovarik [14] and Björck and Bowie [4] for computing the unitary polar factor! Hence, just a single inner Schulz iteration is enough to maintain quadratic convergence.

For (3.3) it holds in any norm for which $\|I\| = 1$ that

$$\|X_{k+1}^* X_{k+1} - I\| \le \|X_k^* X_k - I\|^2 \qquad \text{if} \qquad \|X_0^* X_0 - I\| \le 1.$$

(As this suggests, the asymptotic error constant is 1 for (3.3) compared with 1/2 for (3.2) [10]). To maximise the number of matrix multiplications we therefore need to switch from iteration (3.2) to iteration (3.3) as soon as the convergence condition

$$\|X_k^* X_k - I\| \leq \theta < 1, \tag{3.4}$$

is satisfied; to ensure fast convergence θ should not be too close to 1. As explained below, typically (3.4) is satisfied for $k = 3$ with $\theta = 0.6$ (and obviously for $k = 0$ if $X_0 = A$ happens to be nearly unitary). Rather than expend a matrix multiplication testing (3.4) we can use the matrix norm estimator CONEST from [11]. This computes a lower bound for $\|C\|_1$ by sampling several matrix-vector products Cx and C^*x; thus we can estimate $\|X_k^* X_k - I\|_1$, without forming $X_k^* X_k$, in $O(r^2)$ flops (for $r \times r$ X_k). A suitable way to use the estimate is to test whether it is less than $\lambda\theta$, where $\lambda < 1$. If so, $X_k^* X_k - I$ is formed, in preparation for (3.3), and its norm is taken. If (3.4) is satisfied then (3.3) is used—otherwise we revert to iteration (3.2). The optimum choice of λ depends on the desired bias between wasting a matrix multiplication in an abortive switch of iteration, and not switching soon enough. The estimate from CONEST is almost always correct to within a factor 3, so $\lambda \geq 1/3$ is appropriate. In practice we have found that the performance of the algorithm is fairly insensitive to the choices of θ and λ.

To summarise, our hybrid inversion/multiplication algorithm is as follows.

Algorithm 3.1: $[U, H] = \text{polar.mult}\,(A, \delta, \lambda, \theta)$

% A must be a square, nonsingular matrix.

$X_0 = A$; $k = -1$; $\mu = 1$; $\texttt{switched} = \texttt{false}$

repeat

  $k = k + 1$

  if $\texttt{switched}$

    $R = I - X_k^* X_k$; $\mu = \|R\|_1$

    evaluate (3.3)

  else

    $\mu = \text{CONEST}(I - X_k^* X_k)$

    if $\mu > \lambda\theta$

      evaluate (3.2)

    else

      $R = I - X_k^* X_k$; $\mu = \|R\|_1$

      if $\mu > \theta$, evaluate (3.2), else evaluate (3.3), $\texttt{switched} = \texttt{true}$; end

    end

  end

until $\mu \le \delta$

$U = X_{k+1}$

$H = \frac{1}{2}(U^* A + A^* U)$ ∎

Since iteration (3.3) requires two matrix multiplications, and iteration (3.2) requires one inversion, Algorithm 3.1 will be more efficient than Algorithm 2.1 if matrix multiplication can be done at twice the rate of matrix inversion; thus, compared with using the full inner Schulz iteration, the "cutoff ratio" is 2 instead of 6.8. Moreover, if advantage is taken of the symmetry of the second matrix product in (3.3) the cutoff ratio is reduced to 1.5. The overall speedup depends on the ratio of inversions to multiplications, which in turn depends on the conditioning of the matrix, as discussed below.

All the algorithms mentioned here have been coded and tested in PC-MATLAB [15], running on an IBM PC-AT. For this machine the unit roundoff $u \approx 2.22 \times 10^{-16}$. We used $\theta = .6$, $\lambda = .75$, $\delta = \sqrt{r}u$, where r is the dimension of the matrix A in Algorithms 2.1 and 3.1, and $\epsilon = \max(m, n)|t_{11}|u$ in the complete orthogonal decomposition, where T is the triangular factor from the QR factorization with complete pivoting.

The following comments summarise our numerical experience, based on a wide variety of test matrices.

• Algorithms 2.1 and 3.1 usually require the same number of iterations. Occasionally Algorithm 3.1 requires one more iteration due to the larger error constant for iteration (3.3).

• In general, the typical number of iterations for Algorithm 3.1 is 7–9, with the switch of iteration on iteration 3 or 4.

• For well-conditioned matrices ($\kappa_2(A) \le 10$, say), as are common in certain applications (see [12]), Algorithm 3.1 tends to require at most 7 iterations and to switch on iteration 1–3.

We present the results for one representative matrix in detail: A is MATLAB's "gallery(5)", a 5×5 nilpotent matrix. Using Algorithm 3.1 within Algorithm 2.2, the numerical rank is diagnosed as 4, and Algorithm 3.1 is presented with a triangular matrix having one singular value of order 10^5 and three of order 1. Table 3.1 summarises the iteration. The backward error $\|A - \widehat{U}\widehat{H}\|_1 = 4.7u\|A\|_1$.

Table 3.1

k	$\kappa_F(X_{k+1})$	$\Vert X_k^*X_k - I\Vert_1$†	Iteration	γ_k
0	2.6265E7	1.1380E10	(3.2)	3.1546E-3
1	5.3197E2	2.6233E4	(3.2)	8.0931E-3
2	4.0886E0	8.0962E-2*	(3.3)	
3	3.9959E0	4.4915E-3	(3.3)	
4	4.0000E0	1.3686E-5	(3.3)	
5	4.0000E0	1.2607E-10	(3.3)	
6	4.0000E0	1.5765E-17	(3.3)	

† Norm estimate while (3.2) used; exact quantity while (3.3) used.

* Norm estimate exact to 5 digits.

Acknowledgement

We thank Des Higham for suggesting several improvements to the manuscript.

REFERENCES

[1] D.H. Bailey, Extra high speed matrix multiplication on the Cray-2, SIAM J. Sci. Stat. Comput., 9 (1988), pp. 603–607.

[2] C. Bischof and C.F. Van Loan, The WY representation for products of Householder matrices, SIAM J. Sci. Stat. Comput., 8 (1987), pp. s2–s13.

[3] C.H. Bischof, *QR Factorization Algorithms for Course-grained Distributed Systems*, Ph.D. Thesis, Cornell University, 1988.

[4] Å. Björck and C. Bowie, An iterative algorithm for computing the best estimate of an orthogonal matrix, SIAM J. Numer. Anal., 8 (1971), pp. 358–364.

[5] T.F. Chan, Rank revealing QR factorizations, Linear Algebra and Appl., 88/89 (1987), pp. 67–82.

[6] K. Fan and A.J. Hoffman, Some metric inequalities in the space of matrices, Proc. Amer. Math. Soc., 6 (1955), pp. 111–116.

[7] F.R. Gantmacher, *The Theory of Matrices*, Volume One, Chelsea, New York, 1959.

[8] G.H. Golub and C.F. Van Loan, *Matrix Computations*, Johns Hopkins University Press, Baltimore, Maryland, 1983.

[9] J.C. Gower, Multivariate analysis: ordination, multidimensional scaling and allied topics, in *Handbook of Applicable Mathematics, Vol. VI: Statistics*, E.H. Lloyd, ed., John Wiley, Chichester, 1984, pp. 727–781.

[10] N.J. Higham, Computing the polar decomposition—with applications, SIAM J. Sci. Stat. Comput., 7 (1986), pp. 1160–1174.

[11] N.J. Higham, Fortran codes for estimating the one-norm of a real or complex matrix, with applications to condition estimation, Numerical Analysis Report No. 135, University of Manchester, England, 1987; to appear in ACM Trans. Math. Soft.

[12] N.J. Higham, Matrix nearness problems and applications, Numerical Analysis Report No. 161, University of Manchester, England, 1988; to appear in the Proceedings of the IMA Conference on Applications of Matrix Theory, S. Barnett

and M.J.C. Gover, eds, Oxford University Press.

[13] R.A. Horn and C.R. Johnson, *Matrix Analysis*, Cambridge University Press, 1985.

[14] Z. Kovarik, Some iterative methods for improving orthonormality, SIAM J. Numer. Anal., 7 (1970), pp. 386–389.

[15] C.B. Moler, J.N. Little and S. Bangert, *PC-Matlab User's Guide*, The MathWorks, Inc., 20 North Main St., Sherborn, Massachusetts 01770, 1987.

[16] R.S. Schreiber, Block algorithms for parallel machines, in *Numerical Algorithms for Modern Parallel Computer Architectures*, M.H. Schultz, ed., IMA Volumes In Mathematics and Its Applications 13, Springer-Verlag, Berlin, 1988, pp. 197–207.

[17] R.S. Schreiber and B.N. Parlett, Block reflectors: theory and computation, SIAM J. Numer. Anal., 25 (1988), pp. 189–205.

[18] R.S. Schreiber and C.F. Van Loan, A storage efficient WY representation for products of Householder tranformations, Technical Report TR 87-864, Department of Computer Science, Cornell University, 1987; to appear in SIAM J. Sci. Stat. Comput.