Project: Computing the Polar Decomposition

Xiaobo (Bob) Liu[*]

October 25, 2018

Tasks

Q1. First, we review the basic theorem of polar decomposition below, which proves useful for later proofs of questions.

Theorem 1 *(Polar decomposition). Let $A \in \mathbb{C}^{m\times n}$ with $m \geq n$. There exists a matrix $U \in \mathbb{C}^{m\times n}$ with orthonormal columns and a unique Hermitian positive semidefinite matrix $H \in \mathbb{C}^{n\times n}$ such that $A = UH$. The matrix H is given by $(A^*A)^{1/2}$. If rank$(A) = n$ then H is positive definite and U is uniquely determined.*

Proof [1, Chap. 8]. We can write a singular value decomposition (SVD) of A as

$$A = P\begin{bmatrix}\Sigma_r & 0\\ 0 & 0_{m-r,n-r}\end{bmatrix}Q^* = P\begin{bmatrix}I_r & 0\\ 0 & I_{m-r,n-r}\end{bmatrix}Q^*\cdot Q\begin{bmatrix}\Sigma_r & 0\\ 0 & 0_{n-r}\end{bmatrix}Q^*, \tag{1}$$

where $r = \operatorname{rank}(A)$,

$$P\begin{bmatrix}I_r & 0\\ 0 & I_{m-r,n-r}\end{bmatrix}Q^* =: U$$

has orthonormal columns. Also,

$$Q\begin{bmatrix}\Sigma_r & 0\\ 0 & 0_{n-r}\end{bmatrix}Q^* =: H = \widetilde{Q}^{-1}\begin{bmatrix}\Sigma_r & 0\\ 0 & 0_{n-r}\end{bmatrix}\widetilde{Q} \quad (\widetilde{Q} := Q^*)$$

is Hermitian positive semidefinite since H and $\left[\begin{smallmatrix}\Sigma_r & 0\\ 0 & 0_{n-r}\end{smallmatrix}\right]$ are similar, where the later has singular values of A (which are nonnegative) on its diagonal. Then, from $A^*A = (UH)^*UH = HU^*UH = H^2$ it follows that $H = (A^*A)^{1/2}$, which uniquely defines H. If $r = n$ then H is clearly nonsingular and hence $U = AH^{-1}$ is unique. □

Let $A \in \mathbb{C}^{m\times n}$ with $m \geq n$ have the QR factorization $A = QR, Q \in \mathbb{C}^{m\times n}, R \in \mathbb{C}^{n\times n}$. If $R = UH$ is a polar decomposition then $QU \cdot H$ is a polar decomposition of A. This means we can restrict our attention to square matrices since we can always turn the problem of polar decomposition of a rectangular matrix into the one of polar decomposition of a square matrix.

Q2. This is obvious from the proof of Theorem 1 above.

[*]School of Mathematics, University of Manchester, Manchester, M13 9PL, England (`xiaobo.liu@postgrad.manchester.ac.uk`).

Q3. *Proof* [1, Chap. 8]. If $UH = HU$ then, since U is unitary and H is Hermitian, $HU^* = (UH)^* = (HU)^* = U^*H$. Hence, $A^*A = HU^*UH = H^2 = HUU^*H = UHHU^* = AA^*$, So A is normal.

If $A^*A = AA^*$ then $H^2 = UHHU^* = (UHU^*)UHU^*$. Taking the principal square root of both sides gives $H = UHU^*$, or $HU = UH$, as required. $\square$

Q4. Suppose $A = P\Sigma Q^*$ is a SVD of A, $\Sigma = \mathrm{diag}(\sigma_1, \sigma_2, \dots, \sigma_n)$. Then we have

$$\sigma_1 \geq \sigma_2 \geq \cdots \geq \sigma_n > 0$$

since A is full rank. Also, from previous analysis we know $U = PQ^*$ for $A \in \mathbb{C}^{n\times n}$. By substitution the formula that we want to verify becomes

$$PQ^* = \frac{2}{\pi} P\Sigma Q^* \int_0^\infty (t^2 I + Q\Sigma^*\Sigma Q^*)^{-1} \mathrm{d}t.$$

Premultiplying by P^* gives

$$\begin{aligned} & Q^* = \frac{2}{\pi}\Sigma Q^* \int_0^\infty (t^2 I + Q\Sigma^*\Sigma Q^*)^{-1} \mathrm{d}t \\ \Leftrightarrow \quad & Q^* = \frac{2}{\pi} \int_0^\infty (Q\Sigma^{-1})^{-1}(t^2 I + Q\Sigma^*\Sigma Q^*)^{-1} \mathrm{d}t \\ \Leftrightarrow \quad & Q^* = \frac{2}{\pi} \int_0^\infty (t^2 Q\Sigma^{-1} + Q\Sigma)^{-1} \mathrm{d}t \\ \Leftrightarrow \quad & Q^* = \frac{2}{\pi} \int_0^\infty (t^2 \Sigma^{-1} + \Sigma)^{-1} \mathrm{d}t \cdot Q^{-1}. \end{aligned}$$

Postmultiplying by Q gives

$$I = \frac{2}{\pi} \int_0^\infty (t^2\Sigma^{-1} + \Sigma)^{-1} \mathrm{d}t,$$

where the matrix $t^2\Sigma^{-1} + \Sigma$ is diagonal. Therefore, the formula that we want to verify reduces to

$$1 = \frac{2}{\pi} \int_0^\infty \frac{\sigma_i}{t^2 + \sigma_i^2} \mathrm{d}t, \quad i = 1:n. \tag{2}$$

Without loss of generality, we only prove (2) with σ_i. Let $t = \sigma_i x$, it follows the right hand side of (2) becomes

$$\frac{2}{\pi} \int_0^\infty \frac{\sigma_i^2}{\sigma_i^2 x^2 + \sigma_i^2} \mathrm{d}x = \frac{2}{\pi} \int_0^\infty \frac{1}{x^2+1} \mathrm{d}x = \frac{2}{\pi} \cdot \arctan x \Big|_{x=0}^{+\infty} = \frac{2}{\pi} \times \frac{\pi}{2} = 1.$$

Q5. Let X be an approximation to U, the unitary polar factor of A and write $U = X + E$. Then we have

$$I = U^*U = (X+E)^*(X+E) = X^*X + X^*E + E^*X + E^*E \approx X^*X + X^*E + E^*X \tag{3}$$

since E^*E is a second order term in the small perturbation E. Then we can rewrite (3) as

$$X^*E + E^*X = I - X^*X, \tag{4}$$

where both sides are Hermitian. We know that any matrix A can be written as the sum of a Hermitian matrix and a skew-Hermitian matrix:

$$A = \frac{1}{2}(A + A^*) + \frac{1}{2}(A - A^*).$$

In (4), we can calculate the Hermitian part of X^*E and E^*X as $\frac{1}{2}(X^*E + E^*X) = \frac{1}{2}(I - X^*X)$. So, we can assume that X^*E is Hermitian to obtain a feasible solution to equation (4). By doing this, (4) becomes

$$2X^*E = I - X^*X.$$

Premultiplying by $\frac{1}{2}X^{-*}$ gives $E = \frac{1}{2}(X^{-*} - X)$, and then $U = X + E = \frac{1}{2}(X^{-*} + X)$, which leads to the Newton's method for computing U if we take A as the initial approximation:

$$X_{k+1} = \frac{1}{2}(X_k^{-*} + X_k), \quad X_0 = A. \tag{5}$$

Q6. *Proof.* Firstly, we note that to use Newton's method, $X_0 = A$ needs to be of full rank, so here we assume A is nonsingular. Let $A = P\Sigma Q^*$ be an SVD, so we know the unitary polar factor $U = PQ^*$. Next we show by induction that

$$X_n = PD_nQ^*, \tag{6}$$

where

$$D_{n+1} = \frac{1}{2}(D_n + D_n^{-1}), \quad D_0 = \Sigma. \tag{7}$$

Base case: When $n = 1$, $X_1 = \frac{1}{2}(X_0^{-*} + X_0) = \frac{1}{2}(A^{-*} + A)$. Substituting $A = P\Sigma Q^*$ into it gives

$$X_1 = \frac{1}{2}((Q\Sigma^*P^*)^{-1} + P\Sigma Q^*) = \frac{1}{2}P(\Sigma + \Sigma^{-1})Q^* = \frac{1}{2}P(D_0 + D_0^{-1})Q^* = PD_1Q^*,$$

so (6) is true for $n = 1$.

Induction step: Let $k \in \mathbb{N}^+$ be given and suppose (6) is true for $n = k$. Then

$$\begin{aligned} X_{k+1} =& \frac{1}{2}(X_k^{-*} + X_k) \\ =& \frac{1}{2}((PD_kQ^*)^{-*} + PD_kQ^*) \\ =& \frac{1}{2}P(D_k^{-1} + D_k)Q^* \\ =& PD_{k+1}Q^*. \end{aligned}$$

Thus, (6) holds for $n = k + 1$, and the proof of the induction step is complete.

Conclusion: By the principle of induction, (6) is true for all $n \in \mathbb{N}^+$.

Then we want to prove that the Newton iterates D_n in (7) with nonsingular $D_0 = A$ converge quadratically to the matrix sign function $S = \text{sign}(A)$.

We know if A has eigenvalues $\lambda_1, \lambda_2, \ldots$, then eigenvalues of A^{-1} are $\lambda_1^{-1}, \lambda_2^{-1}, \ldots$. For $\lambda = re^{i\theta}$ we have $\lambda + \lambda^{-1} = (r + r^{-1})\cos\theta + i(r - r^{-1}\sin\theta)$, and hence eigenvalues of D_n remain in their open half-plane under the mapping (7). Hence, D_n is well-defined and nonsingular for all n. Then according to the definition of matrix sign function by Jordan canonical form, it is easy to see $\operatorname{sign}(D_n) = \operatorname{sign}(D_0) = \operatorname{sign}(A) = S$. Therefore, $D_n + S = D_n + \operatorname{sign}(D_n)$ is also nonsingular from the view of their eigenvalues.

Also, we see D_n are rational functions of $D_0 = A$ and hence, like A, commute with $\operatorname{Sign}(A) = S$, then we have

$$\begin{aligned} D_{n+1} \pm S &= \frac{1}{2}(D_n + D_n^{-1} \pm 2S) \\ &= \frac{1}{2}D_n^{-1}(D_n^2 \pm 2D_nS + I) \\ &= \frac{1}{2}D_n^{-1}(D_n \pm S)^2, \end{aligned} \tag{8}$$

and hence

$$\begin{aligned} (D_{n+1} - S)(D_{n+1} + S)^{-1} &= \frac{1}{2}D_n^{-1}(D_n - S)^2 \cdot 2(D_n + S)^{-2}D_n \\ &= (D_n - S)^2(D_n + S)^{-2} \\ &= ((D_n - S)(D_n + S)^{-1})^2. \end{aligned}$$

If we define

$$G_n = (D_n - S)(D_n + S)^{-1}, \tag{9}$$

we have $G_{n+1} = G_n^2 = \cdots = G_0^{2^{n+1}}$. Now $G_0 = (D_0 - S)(D_0 + S)^{-1} = (A - S)(A + S)^{-1}$ has eigenvalues $(\lambda - \operatorname{sign}(\lambda))/(\lambda + \operatorname{sign}(\lambda))$, where $\lambda \in \Lambda(A)$. It follows the eigenvalues of G_0 lie inside the unit circle since λ is not pure imaginary, which means $\rho(G_0) < 1$, then $G_n = G_0^{2^n} \to 0$ as $n \to \infty$. Hence from (9) we have

$$D_n = (I - G_n)^{-1}(I + G_n)S \to S \quad \text{as } n \to \infty.$$

Finally, taking norms in (8) with the minus sign gives

$$\|D_{n+1} - S\| = \frac{1}{2}\|D_n^{-1}(D_n - S)^2\| \le \frac{1}{2}\|D_n^{-1}\|\|D_n - S\|^2,$$

which displays the quadratic rate of convergence.

Above all, we have proved that the Newton iterates X_n for computing U are equivalently $X_n = PD_nQ^*$, where P and Q are given, and D_n with $D_0 = \Sigma$ converge quadratically to the matrix sign function $\operatorname{sign}(\Sigma) = I$. Hence we can deduce that X_n converge quadratically to $PIQ^* = PQ^* = U$. $\square$

Q7. The Newton–Schulz iteration for computing U is written as

$$X_{k+1} = \frac{1}{2}X_k(3I - X_k^*X_k), \quad X_0 = A. \tag{10}$$

In Newton's method, there is a need for computing the inverse of matrices, which can make the cost of this method exorbitant. One way to avoid this (and hence use only matrix multiplication) is to approximate it by one step of Newton's method for matrix inverse, say, for computing A^{-1} [1, Chap. 5]:

$$X_{k+1} = X_k(2I - AX_k).$$

Then using this formula we can remove the matrix inverse from the Newton iteration (5) to obtain the Newton–Schulz iteration (10) for computing U.

First, we note that to use Newton–Schulz method, $X_0 = A$ needs to be of full rank, so here we assume A is nonsingular. Let $A = P\Sigma Q^*$ be an SVD, so we know the unitary polar factor $U = PQ^*$. Next we show by induction that

$$X_k = P\Sigma_k Q^*, \tag{11}$$

where

$$\Sigma_{k+1} = \frac{1}{2}(3\Sigma_k - \Sigma_k^3), \quad \Sigma_0 = \Sigma. \tag{12}$$

Base case: When $k = 1$, $X_1 = \frac{1}{2}X_0(3I - X_0^* X_0) = \frac{1}{2}A(3I - A^*A)$. Substituting $A = P\Sigma Q^* = P\Sigma_0 Q^*$ into it gives

$$\begin{aligned} X_1 &= \frac{1}{2}P\Sigma_0 Q^*(3I - (P\Sigma_0 Q^*)^* P\Sigma_0 Q^*) \\ &= \frac{1}{2}(3P\Sigma_0 Q^* - P\Sigma_0^3 Q^*) \\ &= P \cdot \frac{1}{2}(3\Sigma_0 - \Sigma_0^3) \cdot Q^* \\ &= P\Sigma_1 Q^*, \end{aligned}$$

so (11) is true for $k = 1$.

Induction step: Let $s \in \mathbb{N}^+$ be given and suppose (11) is true for $k = s$. Then

$$\begin{aligned} X_{s+1} &= \frac{1}{2}X_s(3I - X_s^* X_s) \\ &= \frac{1}{2}P\Sigma_s Q^*(3I - (P\Sigma_s Q^*)^* P\Sigma_s Q^*) \\ &= \frac{1}{2}(3P\Sigma_s Q^* - P\Sigma_s^3 Q^*) \\ &= P \cdot \frac{1}{2}(3\Sigma_s - \Sigma_s^3) \cdot Q^* \\ &= P\Sigma_{s+1} Q^* \end{aligned}$$

Thus, (11) holds for $k = s + 1$, and the proof of the induction step is complete.

Conclusion: By the principle of induction, (11) is true for all $n \in \mathbb{N}^+$.

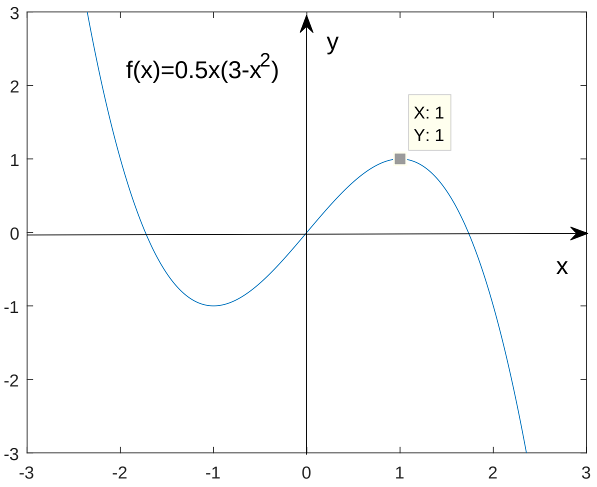
Now we want to find the conditions for $\Sigma_0 = \Sigma$ such that the map (12) converges to the identity matrix, then X_k converges to $PQ^* = U$. It is obvious that Σ_k are diagonal. According to the assumption A is nonsingular, we know the diagonal elements of Σ: $\sigma_1 = \|A\|_2 > \sigma_2 > \cdots > \sigma_n$ are strictly positive. Therefore, we consider the scalar map with $k = 0, 1, \ldots$:

$$x_{k+1} = f(x_k) = \frac{1}{2}x_k(3 - x_k^2), \quad x_0 > 0. \tag{13}$$

By solving $x = \frac{1}{2}x(3 - x^2)$ we obtain the only fixed point of this map in $(0, +\infty)$ is $x = 1$. Then we plot the graph of the function (map) as below.

It is easy to compute that $f(0) = 0$, $f(\sqrt{3}) = 0$, and $f(1) = 1$, which is the maximum of f in $(0, +\infty)$. Hence, $0 < f(x) < 1$ for $x \in (0, \sqrt{3})$. If we write

$$x_{k+1} = x_k(1 + \frac{1}{2}(1 - x_k^2)),$$



we see immediately $x_{k+1} > x_k$ for $x_k \in (0,1)$. It follows that $x_k \to f(1) = 1$ as $k \to \infty$ for $x_0 \in (0, \sqrt{3})$. For $x_0 > \sqrt{3}$, even though the monotonic decrease of f can simplify the problem, the cases become much more complicated. For example, it is can be shown in a similar way that $f(f(2)) = f(-1) = -1$, and thus $x_k \to f(-1) = -1$ as $k \to \infty$ for $x_0 \in (\sqrt{3}, 2)$. Furthermore, let

$$f(f(m_1)) = f(-\sqrt{3}) = 0, \quad f(f(m_2)) = \sqrt{3} \tag{14}$$

for $m_2 > m_1 > 2$, then it is not hard to see that for $x_0 \in (m_1, m_2)$, $x_k \to 1$ as $k \to \infty$. Numerically solving (14) by MATLAB, we obtain $m_1 \approx 2.1478, m_2 \approx 2.2212$, so we can assert for $x_0 \in (2.15, 2.22)$, $x_k \to 1$ as $k \to \infty$. By solving $-x = f(x) = \frac{1}{2}x(3-x^2)$ we obtain $x = \pm\sqrt{5}$. Actually, if we consider more iterations of the map f, there are infinitely many such intervals $(a,b) \subset (\sqrt{3}, \sqrt{5})$ on which if the starting point x_0 lies then $x_k \to 1$ as $k \to \infty$. However, the later results are hard to be practically exploited since the requirements are on all singular values of the matrix A. Besides, if we write (13) as

$$x_{k+1} - 1 = -\frac{1}{2}(x_k^3 - 2x_k^2 + x_k + 2x_k^2 - 4x_k + 2) = -\frac{1}{2}(x_k - 1)^2(x_k + 2),$$

which leads to

$$\|x_{k+1} - U\| \le \frac{1}{2}\|x_k - U\|^2 \|x_k + 2U\|$$

by taking norms on both sides, which displays the quadratic rate of convergence. Hence, we conclude that in the Newton–Schulz iteration (10), $X_k \to U$ quadratically as $k \to \infty$ if $\|A\|_2 < 3$.

Q8. For the Newton's method: $X_{k+1} = \frac{1}{2}(X_k^{-*} + X_k)$, $X_0 = A$, the computational cost mainly comes from the matrix inversion since it only requires one matrix inversion and one matrix addition in matrix level. To compute X_k^{-1} for a given X_k, we form $X_k X_k^{-1} = I$ and solve the linear systems $X_k X_{ki}^{-1} = e_i$, $i = 1:n$ using the $PA = LU$, the LU factorization with permutation that requires $\frac{2}{3}n^3 + \mathcal{O}(n^2)$ flops. Then we have n upper triangular systems and n lower triangular systems to solve, where the right-hand side vector e_i has $i-1$ leading zeros. Exploiting this, we can reduce the flops to $\frac{1}{3}n^3 + n^3 + \mathcal{O}(n^2) = \frac{4}{3}n^3 + \mathcal{O}(n^2)$ from $2n^3 + \mathcal{O}(n^2)$ that would be required for solving

$2n$ general triangular systems. Hence, the total cost of one step of Newton's method is essentially $\frac{2}{3}n^3 + \frac{4}{3}n^3 = 2n^3$ flops.

The Newton–Schulz iteration can be written as: $X_{k+1} = \frac{1}{2}(3X_k - X_k X_k^* X_k),\ X_0 = A$, the computational cost mainly comes from the two matrix multiplications of forming $X_k X_k^* X_k$. We can exploit the symmetry of the product of $X_k X_k^*$ to reduce the cost of computing it to $n^3 + \mathcal{O}(n^2)$ flops. Then we compute the second matrix multiplication $(X_k X_k^*)X_k$ in the general way, which requires $2n^3 + \mathcal{O}(n^2)$ flops. Therefore, the total cost of one step of Newton–Schulz iteration is essentially $n^3 + 2n^3 = 3n^3$ flops.

Ignoring operation counts, from the results above we see immediately that matrix multiplication has to be at least $3n^3/2n^3 = 1.5$ times faster than matrix inversion for Newton–Schulz to be faster than Newton.

Q9. The link to my GitHub repository: `https://github.com/Xiaobo-Liu/polar_decom`.

The algorithm computes the polar decomposition $A = UH$ of $A \in \mathbb{C}^{n\times n}$ (ideally, A should be nonsingular), which uses the methods we have discussed before to compute the unitary polar factor U, and then it easily computes the Hermitian factor H by just one matrix multiplication $H = U^*A$.

Let us assume A is nonsingular. First, from our previous proofs, we have known that the Newton's method is bound to quadratically converge to the unique polar factor U, while the quadratic convergence of Newton–Schulz method is subject to the condition that $\|A\|_2 < 3$. Because matrix multiplication is very fast on high-performance computers compared with matrix inversion, the algorithm will start with Newton iteration and switch to the Newton–Schulz iteration when the later is certain to converge, to ensure the convergence and to some degree optimize the speed. Based on this, we could use the criteria

$$\|X_k\|_2 < \sqrt{3}$$

to judge when the algorithm switch to the Newton–Schulz iteration (we have tried using this criteria, in which case it makes no big difference to the iterations required). Alternatively, we chose to use

$$\|I - X_k^* X_k\| \le \theta \tag{15}$$

for some $\theta < 1$, where we think $\|I - X_k^* X_k\|$ can be viewed as a measure of unitarity of X_k. It is not hard to show (as shown in [2]) that if $\theta < 1$ for certain X_k, then $\|I - X_k^* X_k\|$ is strictly decreasing if the Newton–Schulz iteration is employed. In the algorithm, we used $\theta = 0.6$ as suggested in [2]. Also, it is shown in [1, Lem. 8.17] that

$$\frac{\|A^*A - I\|}{1 + \sigma_1(A)} \le \|A - U\| \le \frac{\|A^*A - I\|}{1 + \sigma_n(A)}, \tag{16}$$

for any unitarily invariant norm. This inequality (16) shows the measure of the distance $\|X_k^* X_k - I\|$ is essentially equivalent to the measure of the forward error $\|X_k - U\|$, even if we are using the infinity norm to measure $\|X_k^* X_k - I\|$ due to the norm equivalence of matrices between the 2-norm and the infinity norm

$$\frac{1}{\sqrt{n}}\|A\|_\infty \le \|A\|_2 \le \sqrt{n}\|A\|_\infty.$$

The discussion above validates the convergence of the algorithm which combines the Newton's iteration and the Newton–Schulz iteration. On each iteration we will also print out $\|I - X_k^* X_k\|_\infty$ and $\|X_k - X_{k-1}\|_\infty / \|X_k\|_\infty$, the normwise relative distance between two iterates, which reveals the 'movement' we make during each iteration. Finally, we will terminate the algorithm only when $\|X_k - X_{k-1}\|_\infty / \|X_k\|_\infty$ becomes less than the tolerance that we choose as $10u$ (here u is the unit roundoff), that is to say, when further progress that we can make becomes rather limited. Below the Table 1 lists the results on five kinds of test matrices that we used. In Appendix A, we plot RD_k and EFE_k on each iteration for the matrices.

Table 1: Results on test matrices. We set $RD_k := \|X_k - X_{k-1}\|_\infty / \|X_k\|_\infty$, $EFE_k := \|I - X_k^* X_k\|_\infty$, and k to be the total iterations. IS denotes the first iteration that the algorithm switch to the Newton–Schulz method.

A	$\kappa_\infty(A)$	k	IS	RD_k	EFE_k	$\|U - X_k\|_\infty$
`randn(5)`	38.63	9	5	8.56×10^{-16}	5.17×10^{-16}	9.44×10^{-16}
`randn(20)`	330.32	9	5	2.31×10^{-16}	1.26×10^{-15}	6.13×10^{-15}
`randn(100)`	2.60×10^{3}	10	6	6.04×10^{-16}	4.14×10^{-15}	2.65×10^{-14}
`eye(8)`	1	1	NA	NA	0	0
`hilb(6)`	2.91×10^{7}	29	25	1.12×10^{-16}	1.34×10^{-27}	9.37×10^{-12}
`magic(6)`	5.00×10^{17}	58	54	1.46×10^{-16}	4.10×10^{-16}	2.36
`hadamard(8)`	8	8	3	1.77×10^{-16}	6.39×10^{-16}	1.78×10^{-15}

Some observations on the results:

1. For random matrices `randn(n)`, the performance of our algorithm is generally good. When the condition number grows as the size of the matrix increases, it tends to require more iterations. Also, it shows that $\|I - X_k^* X_k\|_\infty$ is a good indicator of the forward error $\|U - X_k\|_\infty$ in these cases.

2. For the identity matrix, the algorithm exactly converges to the U we want with one iteration. From the view of polar decomposition of the identity $I = I \cdot I$, this is not surprising because the initial matrix that we starts with in the Newton method is the U we want to compute.

3. For `hilb(6)`, it takes distinctly more iterations than previous matrices for the algorithm to reach its stopping criteria. From its profiles of $\|X_k - X_{k-1}\|_\infty / \|X_k\|_\infty$ and $\|I - X_k^* X_k\|_\infty$ in Appendix A, we see the convergence of the Newton method is much slower than that of the Newton–Schulz method. Also, we note that the result is still accurate given that the condition number is of order 10^7.

4. `magic(6)` is a singular matrix, which becomes numerically invertible in MATLAB due to the occurrence of rounding errors. In this case it takes the algorithm 58 iterations to converge. Again, from its profiles of $\|X_k - X_{k-1}\|_\infty / \|X_k\|_\infty$ and $\|I - X_k^* X_k\|_\infty$, we see the convergence of the Newton method is very slow because matrix inversion is required in its iterations. We also note that the forward error of the result is considerably big and much larger than $\|I - X_k^* X_k\|_\infty$.

5. For the well-conditioned matrix `hadamard(8)`, it only takes 8 iterations to converge.

6. To summary, it tends to require more iterations for the algorithm to converge when the condition number of the computed matrix grows. Generally, $\|I - X_k^* X_k\|_\infty$ is a good indicator of the forward error $\|U - X_k\|_\infty$ for nonsingular matrices, while for singular matrices it may not be. Finally, our algorithm is not a good choice for singular matrices in terms of accuracy and speed.

Q10. For a symmetric positive definite matrix A, if we firstly factorise A using a Cholesky decomposition: $A = R^* R$, and then we compute a polar decomposition of the Cholesky factor $R = UH$, then we will obtain

$$A = R^* R = (UH)^*(UH) = H^* U^* U H = H^* H = H^2,$$

where by definition H is the square root of A. Using this idea, we write a function `plosqrt(A)` that computes the square root of a symmetric positive definite matrix A.

References

[1] Nicholas J. Higham. *Functions of Matrices: Theory and Computation.* Society for Industrial and Applied Mathematics, Philadelphia, PA, USA, 2008.

[2] Nicholas J. Higham and Robert S. Schreiber. Fast polar decomposition of an arbitrary matrix. *SIAM J. Sci. Statist. Comput.*, 11(4):648–655, July 1990.

Appendix A: Figures

Figure 1: RD_k and EFE_k on each iteration for `randn(5)`

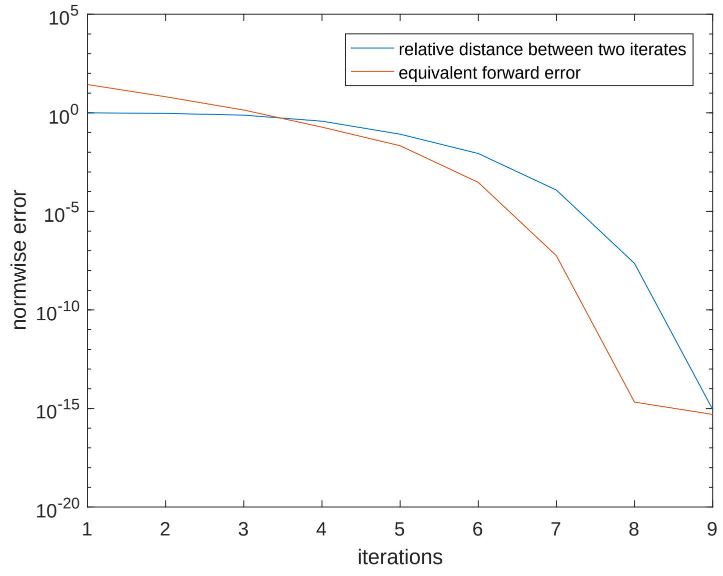


Figure 2: RD_k and EFE_k on each iteration for `randn(20)`

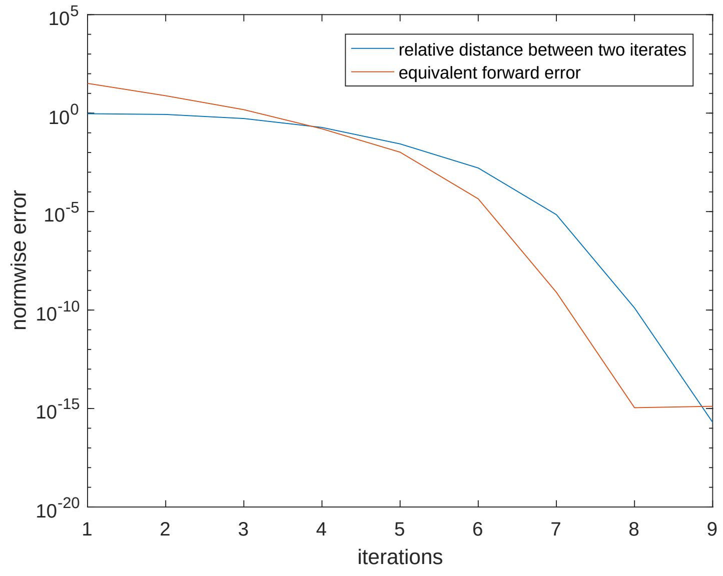


Figure 3: RD_k and EFE_k on each iteration for `randn(100)`

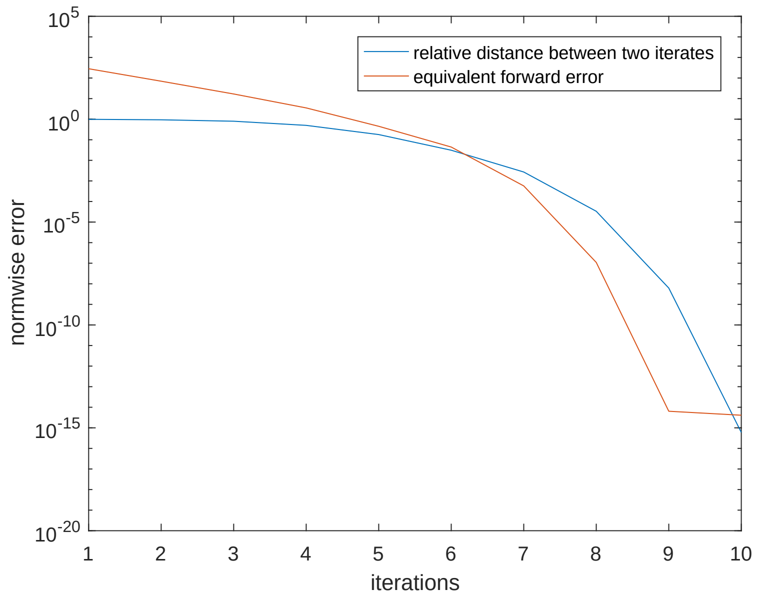


Figure 4: RD_k and EFE_k on each iteration for `hilb(6)`

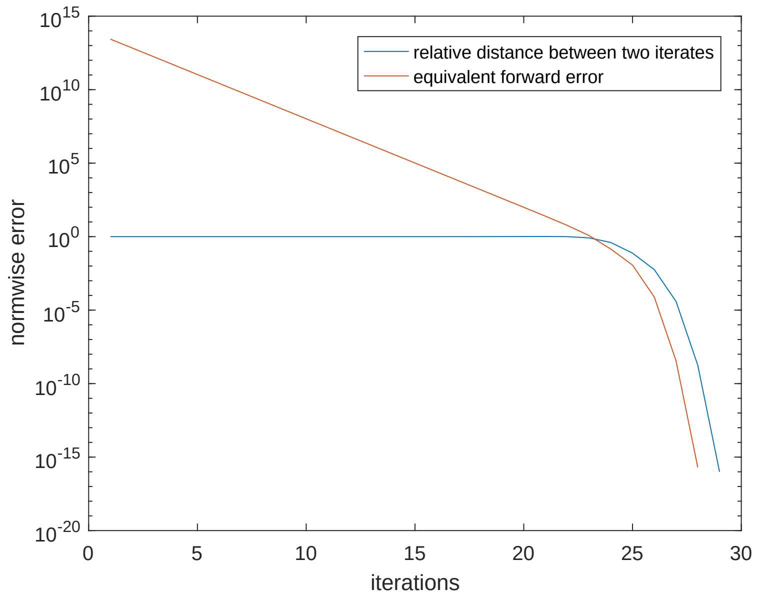


Figure 5: RD_k and EFE_k on each iteration for `magic(6)`

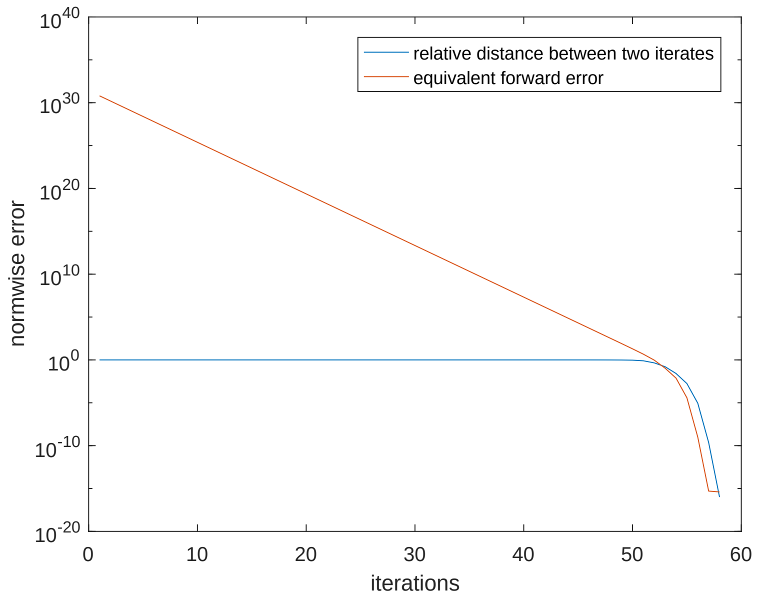
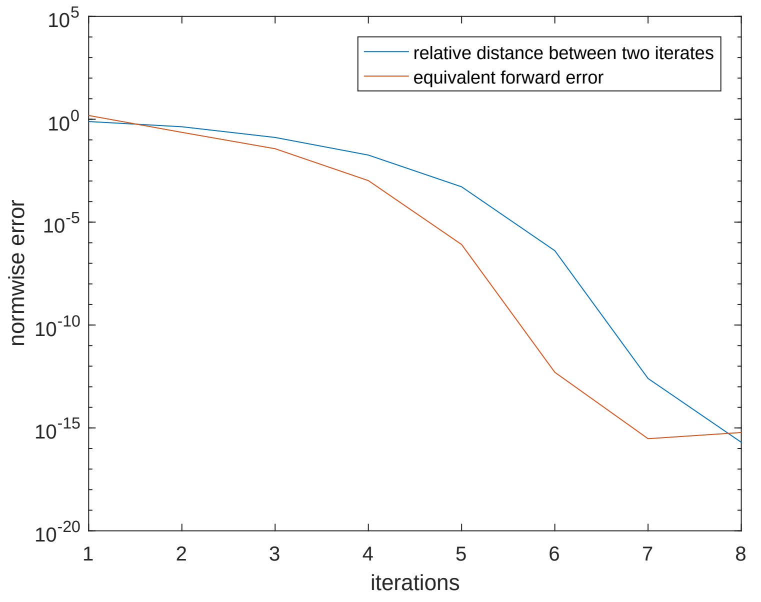


Figure 6: RD_k and EFE_k on each iteration for `hadamard(8)`



Appendix B: MATLAB Code

```
function [U, H, its] = poldec(A)
%POLDEC Polar decomposition
%   [U, H, its] = poldec(A) computes the polar decomposition A = U*H of the
%   square, nonsingular matrix A. ITS is the number of iterations for
%   convergence.

% Setting the parameters which can be changed if necessary
tol = 10*eps; % convergence tolerance: of order of the unit roundoff
theta = 0.6; % switch tolerance
plots = true; % plot the errors against iterations

validateattributes(A, {'double', 'single'}, {'square'});
n = length(A);
X_prev = A;
I = eye(n);
its = 0;
terminates = false;
switches = false;
notice = false; % give a notice when switches
if plots
   fid = fopen('data.txt','w'); % store data into data.txt for plotting
end
while terminates == false
   its = its + 1;
   if ~switches % Newton method
      X_next = (X_prev + (X_prev\I)')/2;
      % compute the normwise relative distance (RD) between two
      % iterates and the equivalent forward error (FE)
      RD = norm(X_next - X_prev, Inf)/norm(X_next, Inf);
      FE = norm(I - X_next'*X_next, Inf);
      if FE <= theta
         switches = true;
         notice = true;
      end
   else % switch to Newton-Schulz method
      X_next = (3*X_prev - X_prev*(X_prev')*X_prev)/2;
      RD = norm(X_next - X_prev, Inf)/norm(X_next, Inf);
      FE = norm(I - X_next'*X_next, Inf);
   end
   if plots
      fprintf(fid, '%.2d %.16f %.16f \n', its, RD, FE);
   end
   fprintf('%.2dth iters: Relative Distance %.4e, Forward Error %.4e \n',
       its, RD, FE);
   if notice
      fprintf(' Now switch from the Newton iteration to the Newton-Schulz
         iteration.\n');
      notice = false; % only one notice is needed
   end
   if RD < tol
      terminates = true;
      if plots
         fclose(fid);
      end
   end
   X_prev = X_next;
end
U = X_prev;
H = U'*A;
H = (H' + H)/2; % Force Hermitian by taking nearest Hermitian matrix

% load data and plot the errors
if plots
   data = load('data.txt');
   semilogy(data(:,1),data(:,2),data(:,1),data(:,3));
   legend('relative distance between two iterates','equivalent forward
       error');
   xlabel('iterations');
   ylabel('normwise error');
end
end
```

```
function S = plosqrt(A)
%POLSQRT Matrix square root
%   [S] = plosqrt(A) Computes the square root S = A^(1/2) of the symmetric
%   positive definite matrix A by doing a Cholesky decomposition and
%   calling poldec.

[R, p] = chol(A);
if p~= 0
   disp('sorry, I only work for a sym.pos.def matrix.');
else
   [~, S, ~] = poldec(R);
   clc; % clear the displays that result from calling poldec()
   disp('>> plosqrt()');
end
end
```