The Polar Decomposition

Michael Connolly[*]

October, 2018

1 Basic Properties

We begin by defining the polar decomposition and covering some basic properties. Let $A \in \mathbb{C}^{m\times n}$ with $m \geq n$. There exists $U \in \mathbb{C}^{m\times n}$ with orthonormal columns and a unique Hermitian positive semidefinite $H \in \mathbb{C}^{n\times n}$ such that $A = UH$. All U are given by

$$U = P \begin{bmatrix} I_r & 0 \\ 0 & W \end{bmatrix} Q^*, \tag{1}$$

where

$$A = P \begin{bmatrix} \Sigma_r & 0 \\ 0 & 0_{m-r,\, n-r} \end{bmatrix} Q^* \tag{2}$$

is an SVD, $r = \operatorname{rank}(A)$ and $W \in \mathbb{C}^{(m-r)\times(n-r)}$ is arbitrary subject to having orthonormal columns.

We note some properties of the polar decomposition. From this point forward we consider only square matrices. Let $A \in \mathbb{C}^{n\times n}$ have polar decomposition $A = UH$. Then:

1. $H = (A^*A)^{\frac{1}{2}}$,
2. $\lambda(H) = \sigma(A)$,
3. $\kappa_2(H) = \kappa_2(A)$,
4. A is normal iff $UH = HU$.

We present proofs of the second and fourth properties. To see $\lambda(H) = \sigma(A)$, we first consider the SVD (2) of A, with P, $Q \in \mathbb{C}^{n\times n}$ and both unitary. Σ_r is then a diagonal matrix containing the singular values of A. We now evaluate $A^*A = (P\Sigma Q^*)^*P\Sigma Q^* = Q\Sigma^*P^*P\Sigma Q^* = Q\Sigma^*\Sigma Q^*$. We now note that $H =$

[*]School of Mathematics, University of Manchester, Manchester, M13 9PL, England (`michael.connolly-6@postgrad.manchester.ac.uk`).

$(Q\Sigma^*\Sigma Q^*)^{\frac{1}{2}} = (Q\Sigma^2 Q^*)^{\frac{1}{2}} = Q\Sigma Q^*$, as $\Sigma \in \mathbb{R}^{n\times n}$. We know Σ to be diagonal and Q to be unitary, so this forms a spectral decomposition of H, completing the proof.

We next show that $A^*A = AA^*$ if and only if U and H commute. If $UH = HU$, then $A^*A = (UH)^*UH = H^*U^*UH = H^2 = HUU^*H = AA^*$ and we have that A is normal.

If $A^*A = AA^*$, then $H^2 = UH^2U^*$. Taking the principal square root of both sides gives $H = UHU^*$ and post-multiplying by U gives the required condition.

We verify the formula

$$U = \frac{2}{\pi} A \int_0^\infty (t^2 I + A^*A)^{-1} dt. \tag{3}$$

We first note that we can write $U = PQ^*$ when considering the SVD of A. We can also diagonalize $A^*A = Q\Sigma^2 Q^*$, and so

$$U = \frac{2}{\pi} P\Sigma Q^* \int_0^\infty (t^2 I + Q\Sigma^2 Q^*)^{-1} dt. \tag{4}$$

By writing $I = QQ^*$, we have

$$\begin{aligned} &(t^2 I + Q\Sigma^2 Q^*)^{-1} \\ &= (Q(t^2 I + \Sigma^2)Q^*)^{-1} \\ &= Q(t^2 I + \Sigma^2)^{-1} Q^* \\ &= Q \operatorname{diag}(\frac{1}{\sigma_i^2 + t^2}) Q^*. \end{aligned}$$

We can now integrate the diagonal matrix, moving the unitary matrices with no t dependence outside the integral. The result follows from

$$\int \frac{1}{a^2 + t^2} dt = \frac{tan^{-1}(\frac{t}{a})}{a}, \tag{5}$$

and so we have

$$\int_0^\infty \frac{1}{\sigma_i^2 + t^2} dt = \frac{\pi}{2\sigma_i}, \tag{6}$$

so that the matrix integral reads $(\pi/2)\Sigma^{-1}$. Finally, collecting all terms we have:

$$\begin{aligned} U &= \frac{2}{\pi} P\Sigma Q^* Q \frac{\pi}{2} \Sigma^{-1} Q^* \\ &= PQ^* \text{ as required.} \end{aligned}$$

2 Numerical Methods

2.1 Newton Iteration

Given that $U^*U = I$, we consider the equation $(X+E)^*(X+E) = I$, with E a first order perturbation.

$$\begin{aligned}
(X+E)^*(X+E) &= I, \\
X^*X + X^*E + E^*X &= I, \text{ where we drop the } E^2 \text{ term}, \\
X^*E + E^*X &= I - X^*X.
\end{aligned}$$

Promoting X and E to be members of a sequence, we obtain:

$$X_k^*E_k + E_k^*X_k = I - X_k^*X_k, \tag{7}$$

$$X_{k+1} = X_k + E_k. \tag{8}$$

Writing $E_k = (X_k^{-*} - X_k)/2$, and substituting for E_k in (7), we obtain:

$$\begin{aligned}
&\frac{1}{2}X_k^*(X_k^{-*} - X_k) + \frac{1}{2}(X_k^{-1} - X_k^*)X_k \\
&= \frac{1}{2}I - \frac{1}{2}X_k^*X_k + \frac{1}{2}I - \frac{1}{2}X_k^*X_k \\
&= I - X_k^*X_k.
\end{aligned}$$

E_k of this form thus satisfies (7) and so we arrive at the Newton iteration for the unitary polar factor U:

$$X_{k+1} = \frac{1}{2}(X_k + X_k^{-*}),\ X_0 = A. \tag{9}$$

We analyse the convergence of the Newton iteration by considering the SVD of A. As previously noted, we have that $A = P\Sigma Q^*$ and $U = PQ^*$. We postulate that $X_k = PD_kQ^*$, with $D_{k+1} = (D_k + D_k^{-1})/2$, $D_0 = \Sigma$. The proof is by induction. It is trivially true for X_0 as $D_0 = \Sigma$. For X_1, we have that $X_1 = (A + A^{-*})/2$. The equivalence between the two is immediate:

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

Assuming it to be true that $X_k = PD_kQ^*$, we have

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

which is of course the Newton Iteration. The D_k iteration is the Newton Iteration for $\operatorname{sign}(\Sigma) = I$, which we know to be true as Σ is diagonal with positive entries. These Newton iterates D_k thus converge quadratically to I [2, p. 113].

Moreover, we can consider:

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

We thus have the relation

$$\|X_{k+1} - U\| \leq \frac{1}{2}\|X_k^{-1}\|\|X_k - U\|^2. \tag{10}$$

2.2 Newton-Schulz Iteration

We derive an alternative iterative method to the Newton Iteration. We remove the inverse from the formula by approximating it with one step of Newton's method for the matrix inverse:

$$Y_{k+1} = Y_k(2I - BY_k), \tag{11}$$

for computing B^{-1}. Replacing X_k^{-*} by $X_k(2I - X_k^*X_k)$ in (9), having taken $X_k^* = B$, $Y_k = X_k$, we obtain the Newton-Schulz iteration:

$$X_{k+1} = \frac{1}{2}X_k(3I - X_k^*X_k),\ X_0 = A. \tag{12}$$

The convergence of the Newton-Schulz iteration can be analyzed in a similar fashion to the Newton iteration. By considering the SVD of A we again write $X_k = PD_kQ^*$ with $D_{k+1} = D_k(3I - D_k^2)/2$, with $D_0 = \Sigma$. Again, we show this by induction. It is once again trivial for $X_0 = A$, $D_0 = \Sigma$. For $k = 1$, we have that $D_1 = \Sigma(3I - \Sigma^2)/2$. The required result follows easily:

$$\begin{aligned} X_1 &= \frac{1}{2}P\Sigma(3I - \Sigma^2)Q^* \\ &= \frac{3}{2}P\Sigma Q^* - \frac{1}{2}P\Sigma^3Q^* \\ &= \frac{3}{2}X_0 - \frac{1}{2}X_0X_0^*X_0 \\ &= \frac{1}{2}X_0(3I - X_0^*X_0) \end{aligned}$$

as required. Note in the penultimate step we made use of the fact that $X_0X_0^*X_0 = P\Sigma^3Q^*$, which is trivial to show. We then assume that $X_k = PD_kQ^*$. Writing

$PD_{k+1}Q^* = PD_k(3I - D_k^2)Q^*/2 = 3X_k/2 - (PD_k^3Q^*)/2$, it only remains to be shown that $X_kX_k^*X_k = PD_k^3Q^*$. Exploiting the fact that D_k is diagonal with real entries, we can write $X_k^*X_k = QD_k^2Q^*$. We obtain the required expression by writing $PD_k^3Q^* = PD_kQ^*(QD_k^2Q^*) = X_kX_k^*X_k$.

Given that $D_k = \mathrm{diag}(d_k^i)$, the entries of $\mathrm{diag}(d_k^i)$ are given by $d_{k+1}^i = f(d_k^i)$, $f(x) = x(3-x^2)/2$, $d_0^i = \sigma_i$. So in order for $D_k \to I$, the singular values of A must all lie in the region in which the mapping $x_{n+1} = f(x_n)$ is stable and has a fixed point of 1. The fixed points of an iterative map are those for which $x = f(x)$, given here by $x = 0, \pm 1$. Consider first $x_n \in (0, 1)$. We have $f(0) = 0$, $f(1) = 1$ and $f'(x) > 0$ for $x \in [0, 1)$. As the x_n are always increasing in this region and bounded above by 1, they form a sequence which must converge to the fixed point 1. For $x_n \in (1, \sqrt{3})$, we have $f(1) = 1$, $f(\sqrt{3}) = 0$ and $f'(x) < 0$ for $x > 1$. As a result f maps $(1, \sqrt{3}) \to (0, 1)$ and after the first iteration we can apply the previous argument. The iteration fails to converge for $x_0 = \sqrt{3}$ as $f(\sqrt{3}) = 0$ and 0 is a fixed point of the iteration.

We thus require the diagonal entries of Σ to all be non-zero and less than $\sqrt{3}$ for the Newton-Schulz iteration to converge. We could alternatively phrase this as A being nonsingular and $\|A\|_2 < \sqrt{3}$. This result can be shown more generally for the iterative map

$$y_{k+1} = \frac{1}{p}[(1+p)y_k - y_k^{p+1}] = f(y_k),\ y_0 = a^{1/p}. \tag{13}$$

It can be shown [2, p. 182] that $y_k \to 1$ if $y_0 < (p+1)^{1/p}$, and so the Newton-Schulz iteration reduces to the $p = 2$ case.

Considering the quantity $(X_k - U)(X_k + 2U)^*(X_k - U)$, we diagonalize it using the SVD as usual. We can then write

$$\begin{aligned}&(PD_kQ^* - PQ^*)(QD_kP^* + 2QP^*)(PD_kQ^* - PQ^*)\\ &= (PD_k^2P^* + 2PD_kP^* - PD_kP^* - 2I)(PD_kQ^* - PQ^*)\\ &= PD_k^3Q^* - 3PD_kQ^* + 2PQ^*.\end{aligned}$$

We have that $X_kX_k^*X_k = PD_k^3Q^*$ and so $\frac{1}{2}(X_k - U)(X_k + 2U)^*(X_k - U) = -\frac{1}{2}(X_k(3I - X_k^*X_k) - 2U) = -(X_{k+1} - U)$ so that finally we have the relation:

$$\|X_{k+1} - U\|_2 \le \frac{1}{2}\|X_k + 2U\|_2\|X_k - U\|_2^2. \tag{14}$$

We can also consider the convergence of the Newton-Schulz iteration in terms of the residual $R_k = I - X_k^*X_k$. This satisfies [2, p. 218]

$$R_{k+1} = \frac{3}{4}R_k^2 + \frac{1}{4}R_k^3.$$

Thus for $\|R_k\| < 1$, we have $\|R_{k+1}\| < 3\|R_k\|^2/4 + \|R_k\|^2/4 = \|R_k^2\| < \|R_k\|$, and so the $\|R_k\|$ form a sequence converging to zero. Convergence of the residuals to zero thus implies that the X_k are converging to a unitary matrix as we require. This later provides us with an important test on the convergence of the Newton-Schulz method.

2.3 Computational Cost

We now consider the operation count of the Newton and Netwon-Schulz iterations, where $A \in \mathbb{C}^{n\times n}$. The operation count for the Newton iteration is $2n^3+\mathcal{O}(n^2)$ flops, assuming the matrix inverse is computed by Gaussian elimination with partial pivoting (GEPP), making each iteration $\mathcal{O}(n^3)$ flops. Writing the Newton-Schulz iteration as $X_{k+1} = (3/2)X_k - (1/2)X_kX_k^*X_k$, we see that the cost is $3n^2$ (two matrices multiplied by scalars plus a matrix subtraction, each of cost n^2), plus the cost of forming the matrix product $X_kX_k^*X_k$. We can take advantage of the fact that the product $X_k^*X_k$ forms a symmetric matrix, so that in computing this product we only need to compute either the upper or lower triangular part of the matrix. With each entry in the resultant triangular matrix costing $2n-1$ operations to compute, the cost of this is then $(2n-1)\Sigma_{i=1}^n i = 2n(n-1)(n+1)/2 = n^3 - n$ operations. We have another $2n^3$ operations from the subsequent matrix product, meaning the total cost of the Newton-Schulz iteration is $3n^3+\mathcal{O}(n^2)$, again $\mathcal{O}(n^3)$ flops. Ignoring total operation counts, we see that we perform one matrix inversion in the Newton iteration, compared to two matrix multiplications in the Newton-Schulz iteration. This means we require matrix multiplication to be twice as fast as inversion in order for the Newton-Schulz iteration to be quicker, or 1.5 times as fast if we take advantage of the symmetry in the matrix product.

3 Numerical Experiments and Results

In order to practically compute the polar decomposition, we employ an algorithm whereby we start with the Newton iteration and switch to the Newton-Schulz iteration when it's convergence is guaranteed. The motivation for this is that the Newton-Schulz iteration is matrix multiplication rich, which is very fast on high performance computers. We switch to the Newton-Schulz iteration when $\|I - X_k^*X_k\|_\infty \le \theta$, for some $\theta < 1$. We implement the algorithm in MATLAB.

The termination criteria we employ are:

$$\delta_{k+1} < \eta \text{ or } \delta_{k+1} > \delta_k/2,$$

where $\delta_{k+1} = \|X_{k+1} - X_k\|_\infty / \|X_{k+1}\|_\infty$. η is the desired relative error to which we assign a value of $n^{1/2}u$, with u the unit roundoff. Another point to note is that upon computing $\hat{U}$, we compute $\hat{H} = (H' + (H^*)')/2$ with $H' = \hat{U}^*A$. This guarantees that $\hat{H}$ is Hermitian. This computed $\hat{H}$ is the nearest hermitian matrix to A in the Frobenius norm.

Algorithm 1 Polar Decomposition: Given square A we compute the $A = UH$ using a combination of the Newton and Newton-Schulz iterations. We have tolerance η for determining convergence and θ for switching to the Newton-Schulz iteration.

```
 1: $X_0 \leftarrow A$; switch $\leftarrow$ false
 2: for $k = 0 : \infty$ do
 3:     $Y_k \leftarrow X_k^* X_k$
 4:     if not switch then
 5:         if $\|I - Y_k\|_\infty \leq \theta$ then
 6:             switch $\leftarrow$ true
 7:     if switch then
 8:         $X_{k+1} \leftarrow X_k(3I - Y_k)/2$
 9:     else
10:         $X_{k+1} \leftarrow (X_k + X_k^{-*})/2$
11:     $\delta_{k+1} \leftarrow \|X_{k+1} - X_k\|_\infty / \|X_{k+1}\|_\infty$
12:     if $\delta_{k+1} < \eta$ or $\delta_{k+1} > \delta_k/2$ then
13:         break
14: $U \leftarrow X_{k+1}$
15: $H \leftarrow U^* A$
16: $H \leftarrow \frac{1}{2}(H + H^*)$
```

We present the results of this algorithm on various test matrices. We chose θ to be 0.7, but we note that the choice of any reasonable θ did not have a significant effect on the algorithm.

$\mathcal{N}(0,1)$ matrices: In order to test the algorithm, we computed the polar decomposition of matrices of various sizes, with entries sampled from $\mathcal{N}(0,1)$, the normal distribution with zero mean and unit variance. For a sample matrix with $n = 10$, the algorithm switches to the Newton-Schulz iteration after 3 iterations and terminates after 9 iterations. The algorithm also scales well, requiring just 12 iterations for a sample matrix with $n = 1000$ and $\|A - \hat{U}\hat{H}\|_\infty / \|A\|_\infty = 1.4\text{e}{-14}$.

Identity matrix: For I_8 the iteration terminates immediately as expected,

Table 1: Numerical results on test matrices

A	$k+1$	δ_{k+1}	$\frac{\|A - \hat{U}\hat{H}\|_\infty}{\|A\|_\infty}$	$\|I - X_{k+1}^* X_{k+1}\|_\infty$
`randn(10)`	9	1.5e−16	3.2e−16	5.6e−16
`eye(8)`	1	0	0	0
`hilb(6)`	29	1.1e−16	0	7.7e−34
`magic(6)`	59	1.5e−16	0.052	8.0e−16
`hadamard(8)`	8	1.4e−16	2.5e−16	4.0e−16

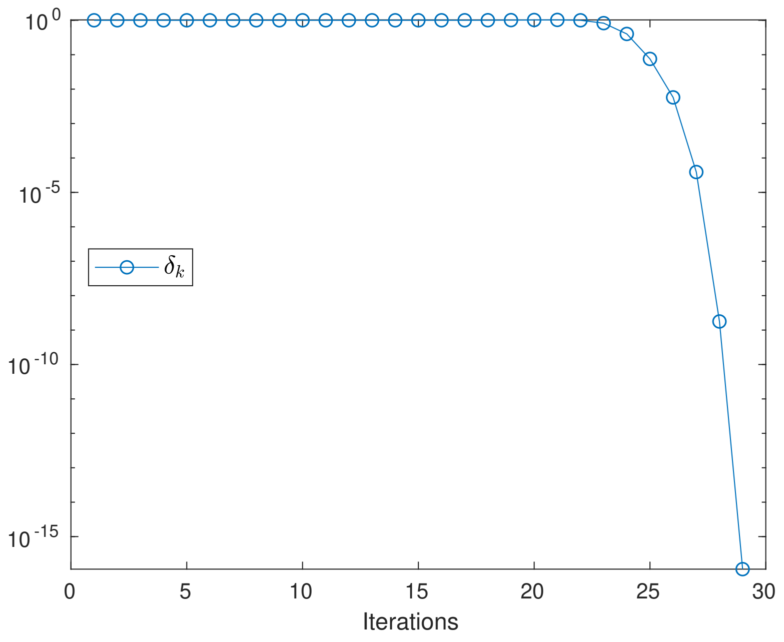

Figure 1: δ_k versus k for the Hilbert matrix.

given that the Newton or Newton-Schulz iteration applied to the identity computes itself. The computed polar factors are again identity matrices, which of course satisfy the required properties of U and H.

Hilbert matrix: For the ill-conditioned Hilbert matrix with $\kappa_2 = 1.4951 \times 10^7$ the algorithm requires 29 iterations for convergence and only switches to the Newton-Schulz iteration after the 24th iteration. Upon terminating, we have $\delta_{k+1} = 1.1\mathrm{e}{-16}$. The computed answer is accurate. We also have $\hat{U} = I$ and so $\hat{H} = A$. This is expected given that A is symmetric positive definite. As such the SVD is equivalent to a spectral decomposition and $U = PQ^* = I$. Figure 1 shows the evolution of δ_k for the Hilbert matrix.

Magic Square: The performance on the rank deficient 6×6 magic square is poor as expected. It requires 59 iterations to converge, yet even upon reforming A from $\hat{U}\hat{H}$, we can see by inspection that the entries are inaccurate by order 10^{-1}. We also have that $\|A - \hat{U}\hat{H}\|_\infty / \|A\|_\infty = 0.0052$ and not of order η as we would hope. This behaviour is of course entirely expected given that `magic(6)` is singular and we attempt to compute it's inverse in the Newton iteration.

Hadamard matrix: For the perfectly conditioned 8×8 Hadamard matrix, we terminate after the 8th iteration, switching to the Newton-Schulz iteration at the 3rd. We note that $\hat{H} = \alpha I$, with $\alpha = 2.8284 = \sqrt{8}$. This is a verification of $\lambda(H) = \sigma(A)$, as can be seen by evaluating `svd(hadamard(8))`. We also expect that $H = \alpha I$ given that the columns of the Hadamard matrix are mutually orthogonal. To thus form U we must simply scale A by an appropriate amount

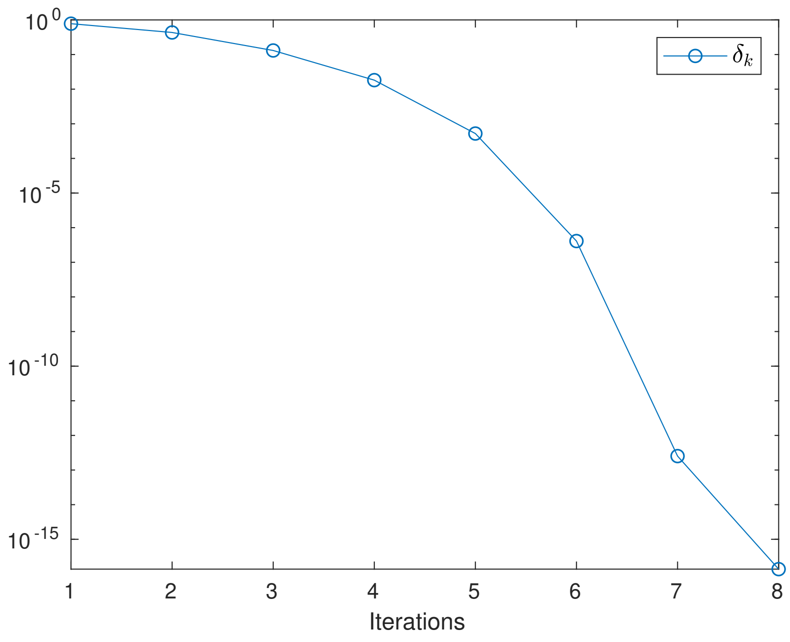

Figure 2: δ_k versus k for the Hadamard matrix.

$1/\alpha$ so that the columns are orthonormal. Figure 2 shows the evolution of δ_k for the Hadamard matrix.

We finally include a routine to compute the square root of a symmetric positive definite matrix by doing a Cholesky decomposition $A = R^*R$ and then performing a polar decomposition. Given $R = UH$, we then have $A = (UH)^*UH = H^*H = H^2$. As seen in the case of the polar decomposition the algorithm is accurate for the ill conditioned Hilbert matrix. We have $\|A - \hat{H}^2\|_\infty / \|A\|_\infty = 5.7\text{e}{-17}$.

Appendix: 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.

plot_display = false; %Set to true to plot relative error vs iterations

if plot_display
    errors = [1];
end
```

```
X = A;
X_old = zeros(size(X));
tol = sqrt(size(X))*eps;    %Tolerance to test convergence

I_size = eye(size(X));  %Identity matrix of appropriate size
its = 0;
theta = 0.7; %parameter to switch to NS

ns_switch = false;  %Boolean to determine switch to NS

delta = 1;
delta_old = inf;
X_product = X'*X;
residual_inf = norm(I_size - X_product, inf);

cond = false; %if true will test delta < delta_old/2 for convergence

while 1
    its = its + 1;
    fprintf('---\nIteration: %2.0f\n', its)
    X_old = X;

    if ~ns_switch
        if residual_inf < theta
            ns_switch = true;
        end
    end

    if ns_switch %If NS will converge do, otherwise Newton
        X = (3/2)*X - X*X_product/2;  %Newton-Schulz
        fprintf("Switched to NS\n");
        X_product = X'*X; %MATLAB should detect symm in this
    else
        X = (X + (inv(X))')/2;    %Newton
        X_product = X'*X;
    end

    delta_old = delta;
    delta = norm(X - X_old, inf)/norm(X, inf);
    fprintf('Relative Error: %g\n', delta)
    if plot_display
        errors(its) = delta;
    end
    residual_inf = norm(I_size - X_product, inf);
    fprintf('Residual: %g\n', residual_inf)

    if delta < 1e-3 %Don't want early termination when delta near 1
        cond = true;
    end

    if (delta < tol) | ((delta > delta_old/2) & cond)
        break
    end

end

if plot_display
    semilogy(1:its, errors, '-o')
    xlabel('Iterations')
    leg = legend('$\delta_k$')
    set(leg, 'Interpreter', 'Latex')
    set(leg, 'FontSize', 12)
    set(leg, 'Location', 'best')
end

U = X;
H = U'*A;
H = 0.5*(H + H'); %H computed naively not Hermitian in general



function B = sqrt_spd(A)
%SQRT_SPD   Matrix Square root of sym pos def
%   [B, its] = sqrt_spd(A) computes the
%   square root of a symmetric positive definite A
%   by doing a Cholesky decomposition and
%   then performing a polar decomposition.
%   The matrix square root of A is given by the
%   hermitian factor of this decomposition

R = chol(A);
[U, H, its] = poldec(R);
B = H;
```

References

[1] Nicholas J. Higham. *Accuracy and Stability of Numerical Algorithms.* Society for Industrial and Applied Mathematics, Philadelphia, PA, USA, second edition, 2002.

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