

Numerical Matrix Analysis

Notes #22 — Eigenvalues Computing the Singular Value Decomposition

Peter Blomgren
(blomgren@sdsu.edu)

Department of Mathematics and Statistics
Dynamical Systems Group
Computational Sciences Research Center
San Diego State University
San Diego, CA 92182-7720
<http://terminus.sdsu.edu/>

Spring 2024
(Revised: April 16, 2024)



Outline

- 1 Flashback
 - The QR-Algorithm with Shifts
- 2 The Big Prize — Computing the SVD
 - $\sigma(A)$ and $\lambda(A^*A)$
 - Stable Approach for $\sigma(A)$
- 3 The Computation, Phase 1
 - Golub-Kahan Bidiagonalization
 - Lawson-Hanson-Chan Bidiagonalization, when $m \gg n$
 - Hybrid LHC/GK 3-Step Bidiagonalization
- 4 The Computation, Phase 2
 - OLD: QR-Like Algorithm, NEW: Divide-and-Conquer
 - Divide-and-Conquer
 - Implementations



The QR-Algorithm with Shifts

Last Time: The QR-Algorithm with Shifts

Starting from the pure QR-Algorithm, which converges linearly, we made a number of critical connections with three other algorithms:

1. Inverse Iteration
2. Shifted Inverse Iteration
3. Rayleigh Quotient Iteration

Adding the tie-breaking Wilkinson shift, we were able to define an algorithm which diagonalizes a real symmetric matrix with **cubic convergence** in general, and **quadratic convergence** in the worst case.

We describe the algorithm to the point where we can quickly identify **one** eigenvalue/eigenvector pair. **Deflation**, i.e. further sub-division of the problem is necessary to identify the full diagonalization.

Ref: Watkins, D.S., 2008. *The QR Algorithm Revisited*. SIAM review, 50(1), pp.133-145.



The QR-Algorithm with Shifts

The Core QR-Algorithm with Wilkinson Shift

Algorithm (The QR-Algorithm with Wilkinson Shifts)

```

A(0) = hessenberg_form(A)
for k = 1:...
    Select  $\mu_w^{(k)} = a_m - \frac{\text{sign}(\delta)b_{m-1}^2}{|\delta| + \sqrt{\delta^2 + b_{m-1}^2}}$ ,  $\delta = \frac{a_{m-1} - a_m}{2}$ 
    [Q(k), R(k)] = qr(A(k-1) -  $\mu_w^{(k)}$ I)
    A(k) = R(k)Q(k) +  $\mu_w^{(k)}$ I
endfor
    
```

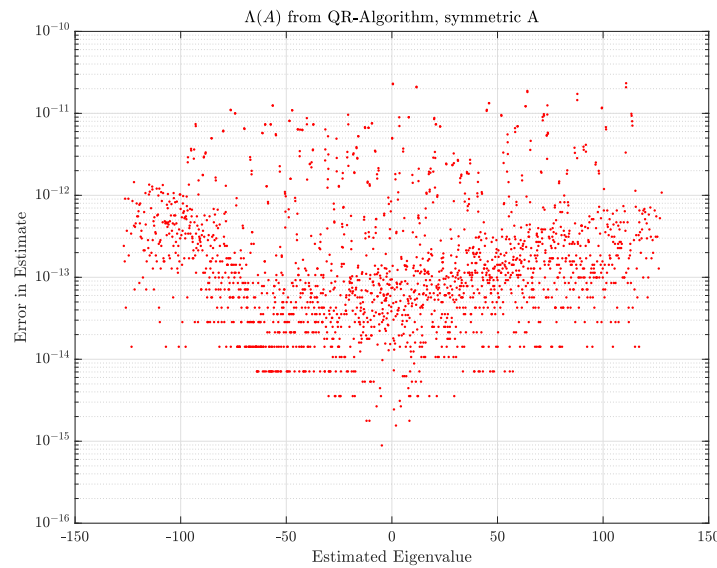
Where,

$$\begin{bmatrix} a_{m-1} & b_{m-1} \\ b_{m-1} & a_m \end{bmatrix} \stackrel{\text{def}}{=} A_{(m-1):m, (m-1):m}$$



Result for Symmetric $A \in \mathbb{R}^{2048 \times 2048}$

“Peel-Off Deflation”



Computing the SVD

Computing the SVD in a **stable** way is non-trivial.

Formally, computation of the SVD can be reduced to an eigenvalue decomposition of a Hermitian square matrix, but the most obvious approach is unstable. (*Which is not stopping some people from using it...*)

Better informed individuals base their SVD computations on a different form of reduction to Hermitian form. As with diagonalizations, **for maximum efficiency** SVD computations are usually done in two phases.



Singular Values of A and Eigenvalues of A^*A

1 of 5

We know that every matrix $A \in \mathbb{C}^{m \times n}$ has a singular value decomposition $A = U\Sigma V^*$, and hence

$$A^*A = V\Sigma^*\Sigma V^* = V \text{diag}(\sigma_1^2, \dots, \sigma_n^2) V^*.$$

Since A^*A and $\text{diag}(\sigma_1^2, \dots, \sigma_n^2)$ are related by a similarity transformation, we must have that $\lambda_i(A^*A) = \sigma_i^2$. Thus, in **infinite** precision the algorithm is clear:

Do-Not-Use-Algorithm (SVD in Infinite Precision)

1. Form A^*A .
2. Compute the eigenvalue decomposition $A^*A = V\Lambda V^*$.
3. Let $\Sigma = \sqrt{\Lambda}$, zero-padded to $(m \times n)$.
4. Solve $U\Sigma = AV$ for unitary U , via QR-factorization.



Singular Values of A and Eigenvalues of A^*A

2 of 5

The algorithm described is unstable since it reduces the SVD to an eigenvalue problem which may be **extremely sensitive to perturbations** — due to ill-conditioning; here $\kappa(A^*A) = (\sigma_1/\sigma_n)^2$.

However, this algorithm is used quite frequently; usually by someone who has “rediscovered” the SVD; — even though it has many names: *the Proper Orthogonal Decomposition, the Karhunen-Loève (KL-) Decomposition, Principal Component Analysis, Empirical Orthogonal Functions, etc...*, the SVD keeps getting “rediscovered.”



Rewind — [NOTES#4]

Hits on scholar.google.com

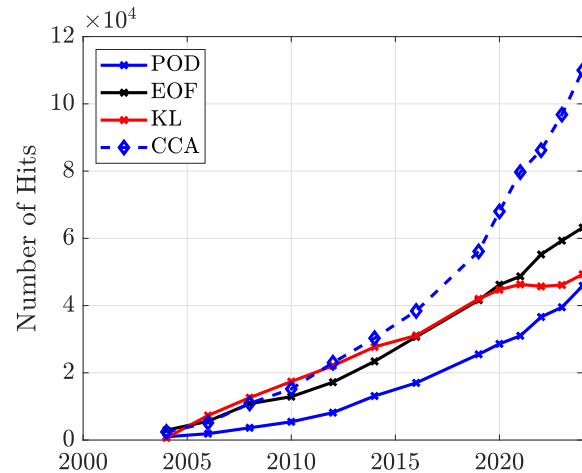


Figure: The many names, faces, and close relatives of the Singular Value Decomposition... Number of hits for "Proper.Orthogonal.Decomposition", "Empirical.Orthogonal.(Function|Functions)", "Karhunen.Loeve", "Canonical.Correlation.Analysis"



Rewind — [NOTES#4]

Hits on scholar.google.com

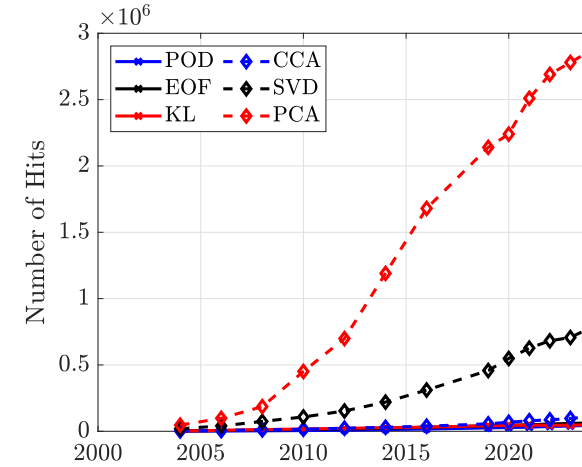


Figure: The many names, faces, and close relatives of the Singular Value Decomposition... Number of hits for "Proper.Orthogonal.Decomposition", "Empirical.Orthogonal.(Function|Functions)", "Karhunen.Loeve", "Canonical.Correlation.Analysis", "Singular.Value.Decomposition", "Principal.Component.Analysis"



Rewind — [NOTES#4]

Hits on scholar.google.com

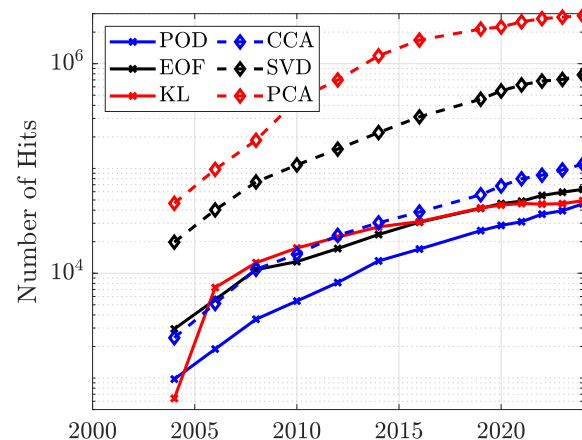


Figure: The many names, faces, and close relatives of the Singular Value Decomposition... Number of hits for "Proper.Orthogonal.Decomposition", "Empirical.Orthogonal.(Function|Functions)", "Karhunen.Loeve", "Canonical.Correlation.Analysis", "Singular.Value.Decomposition", "Principal.Component.Analysis"



Singular Values of A and Eigenvalues of A^*A

3 of 5

The matrix A^*A has familiar and useful interpretations in many fields.

It shows up in linear least squares, as the *normal equations*, and also in the *general orthogonal projector*, $P = A(A^*A)^{-1}A^*$ built from a non-orthogonal matrix. Further, in statistics and other fields, it (or something very much like it) is known as the **co-variance matrix**.

Bottom Line

There are many tempting reasons to form A^*A ...

Don't!!!



Singular Values of A and Eigenvalues of A^*A

4 of 5

We can quantify the instability.

When the Hermitian matrix A^*A is perturbed by δB , the following holds for the perturbation of the eigenvalues

$$|\lambda_k(A^*A + \delta B) - \lambda_k(A^*A)| \leq \|\delta B\|_2$$

A similar bound holds for the perturbation of the singular values

$$|\sigma_k(A + \delta A) - \sigma_k(A)| \leq \|\delta A\|_2.$$

A backward stable SVD algorithm must give $\tilde{\sigma}_k$ satisfying

$$\tilde{\sigma}_k = \sigma_k(A + \delta A), \quad \frac{\|\delta A\|}{\|A\|} = \mathcal{O}(\varepsilon_{\text{mach}}),$$

which implies

$$|\tilde{\sigma}_k - \sigma_k| = \mathcal{O}(\|A\|\varepsilon_{\text{mach}}).$$



Toward a Correct, Stable, Approach...

Given $A \in \mathbb{C}^{m \times m}$, consider (intellectually) the Hermitian matrix

$$H = \begin{bmatrix} 0 & A^* \\ A & 0 \end{bmatrix} = \begin{bmatrix} 0 & V\Sigma U^* \\ U\Sigma V^* & 0 \end{bmatrix}.$$

We can now write the eigenvalue decomposition of H

$$\begin{bmatrix} 0 & A^* \\ A & 0 \end{bmatrix} \begin{bmatrix} V & V \\ U & -U \end{bmatrix} = \begin{bmatrix} V & V \\ U & -U \end{bmatrix} \begin{bmatrix} \Sigma & 0 \\ 0 & -\Sigma \end{bmatrix}.$$

It is clear that from the eigenvalue decomposition of H , we can identify the singular values and singular vectors of A .

Many SVD computations are (implicitly) based on / derived from this observation. We never explicitly form H , and are thus not constrained by the requirement that A is square.



Singular Values of A and Eigenvalues of A^*A

5 of 5

Now, consider $\tilde{\lambda}_k(A^*A)$... If computed using a backward stable algorithm, we expect

$$|\tilde{\lambda}_k - \lambda_k| = \mathcal{O}(\|A^*A\|\varepsilon_{\text{mach}}) = \mathcal{O}(\|A\|^2\varepsilon_{\text{mach}}).$$

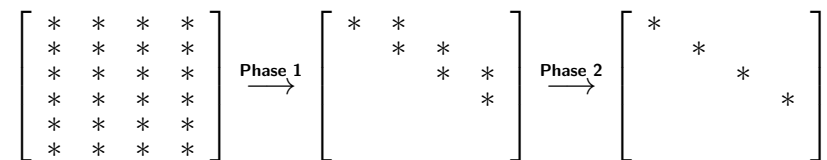
Since $\sigma_k = \sqrt{\lambda_k}$ we get

$$|\tilde{\sigma}_k - \sigma_k| = \mathcal{O}\left(\frac{|\tilde{\lambda}_k - \lambda_k|}{\sqrt{\lambda_k}}\right) = \mathcal{O}\left(\frac{\|A\|^2\varepsilon_{\text{mach}}}{\sigma_k}\right).$$

This result is off by a factor of $\frac{\|A\|}{\sigma_k}$, which is OK for the dominant singular values, but a disaster for small singular values $\sigma_k \ll \|A\|$, in this case we expect a loss of accuracy of order $\kappa(A)$. In a sense we are “squaring the condition number,” much like in the least squares case.



The Two Phases of SVD Computation



The **Bi-Diagonalization** in **Phase 1** requires a finite number of operations $\sim \mathcal{O}(mn^2)$.

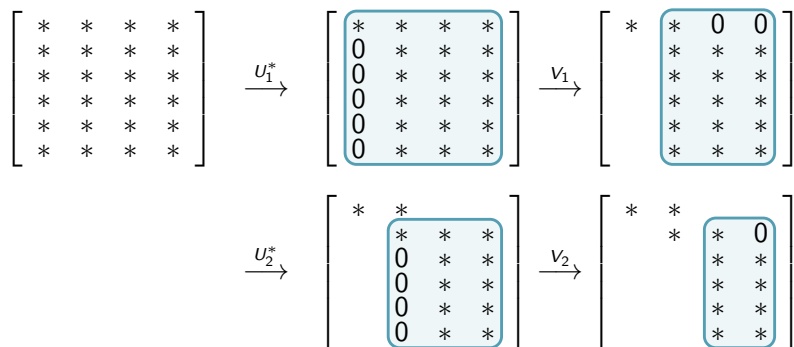
The **Diagonalization** in **Phase 2** is done iteratively, and requires “infinitely many” operations. In practice $\mathcal{O}(n^2)$ operations are sufficient to identify the singular values.



Phase 1: Golub-Kahan Bidiagonalization

1 of 2

Phase-1-Bidiagonalization (for the SVD) is very similar to Phase-1-Hessenberg-transformation (for the QR-algorithm); the main difference here is that we are **not** constrained to a similarity transform, and hence we can apply a different sequence of unitary transforms from the left and right.



Phase 1: Golub-Kahan Bidiagonalization

2 of 2

The unitary matrices U_i are built from full Householder reflectors, and V_i are built from “one-short” reflectors (like in the Hessenberg transformation algorithm)

$$U^*AV = U_m^* \cdots U_1^* A V_1 \cdots V_{n-2} = \begin{bmatrix} * & * & & & \\ & * & * & & \\ & & * & * & \\ & & & * & * \\ & & & & * \end{bmatrix}$$

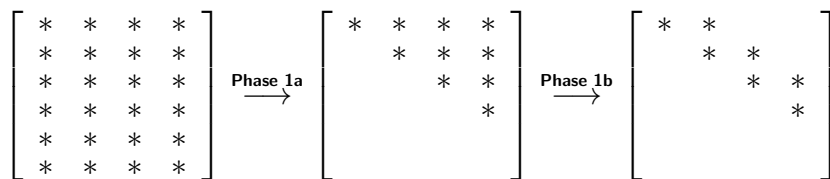
Essentially, this is a QR-factorization from the right and the left, so the total work ends up being

$$\text{Work} \sim \left(4mn^2 - \frac{4}{3}n^3 \right).$$



Faster Methods for Phase 1

When $A \in \mathbb{R}^{m \times n}$, $m \gg n$, Golub-Kahan bidiagonalization is wasteful. In this case, a QR-factorization of A , followed by a the Golub-Kahan bidiagonalization of R is better



i.e. $A \rightarrow Q^*A \rightarrow U^*Q^*AV$. This is known as the Lawson-Hanson-Chan bidiagonalization, and it requires

$$\text{Work} \sim (2mn^2 + 2n^3).$$



Golub-Kahan vs. Lawson-Hanson-Chan Bidiagonalization

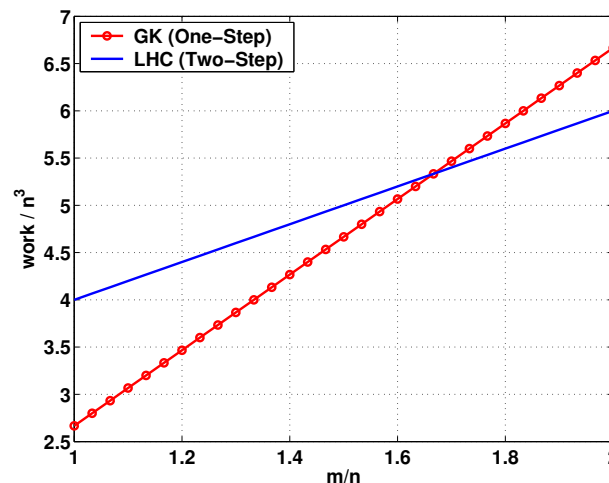
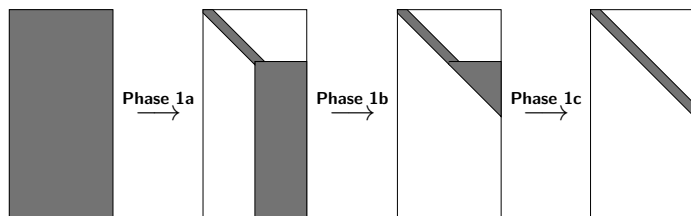


Figure: Comparing the work for Golub-Kahan and Lawson-Hanson-Chan bidiagonalization. The break-even point is $\frac{m}{n} = \frac{5}{3}$.



A Hybrid 3-Step Method

It is possible to define a hybrid algorithm, which switches from Golub-Kahan to Lawson-Hanson-Chan bidiagonalization at the optimal point. We end up with a 3-step method, pictorially defined by



We perform Golub-Kahan bidiagonalization for k steps, until $\frac{m-k}{n-k} = 2$, and then perform Lawson-Hanson-Chan bidiagonalization to the remaining, non-diagonalized part of the matrix.



Computing the SVD: Phase 2

Until recently (1990's), the standard approach to Phase 2 was a variant of the QR-algorithm, applied to the bidiagonal matrix generated during phase 1. E.g. Lapack's `sgesvd`, `cgesvd`, `dgesvd`, and `zgesvd`.

More recently, **divide-and-conquer** algorithms, based on subdivision into smaller subproblems have gained favor in the computational community.

For instance Lapack's `sgesdd`, `cgesdd`, `dgesdd`, and `zgesdd` algorithms are based on this paradigm.

One main advantage of this approach is that it can be parallelized, and thus phase 2 can be computed very rapidly in a multi-core environment. Implementations in e.g. ScaLAPACK, cuSOLVER.

Old-school computing: Variable names in Fortran-77 consist of 1-6 characters chosen from the letters a-z and the digits 0-9. The first character must be a letter. Fortran-77 does not distinguish between upper and lower case, in fact, it assumes all input is upper case.



Hybrid Golub-Kahan / Lawson-Hanson-Chan Bidiagonalization

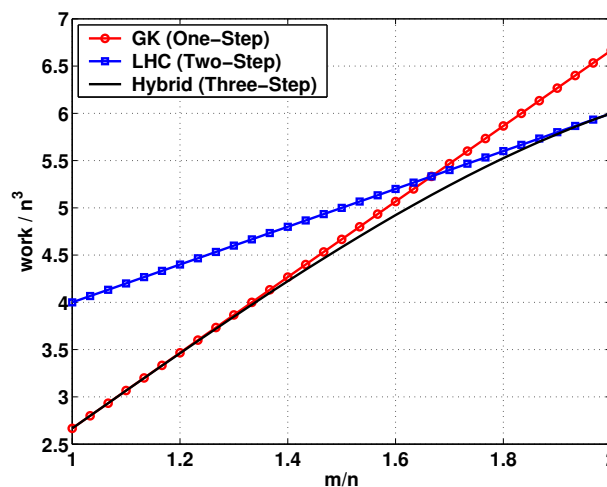


Figure: The work for the hybrid method is $\sim (4mn^2 - \frac{4}{3}n^3 - \frac{2}{3}(m-n)^3)$, and provides a small improvement in the range $n < m < 2n$.



Divide-and-Conquer: Vigorous Hand-waving

In essence divide-and-conquer works like this: We want to compute the diagonalization of B , which we decompose into three parts $B = B_1 + B_2 + \delta B$, where $\text{rank}(\delta B) = 1$:

$$\left[\begin{array}{ccc|ccc} * & * & * & & & \\ & * & * & & & \\ \hline & & & * & & \\ & & & * & * & * \\ & & & & * & * \end{array} \right] = \left[\begin{array}{ccc|ccc} & & & & & \\ & & & & & \\ \hline & & & & & \\ & & & & & \\ & & & & & \end{array} \right] + \left[\begin{array}{ccc|ccc} & & & & & \\ & & & & & \\ \hline & & & & & \\ & & & & & \\ & & & & & \end{array} \right]$$

Now, the diagonalization of the B_1 and B_2 blocks are computed (using the same strategy), then we (iteratively) correct for the rank-1 perturbation

$$\left[\begin{array}{ccc|ccc} \Sigma(B_1) & & & & & \\ & & & * & & \\ \hline & & & & & \\ & & & & & \\ & & & & & \end{array} \right] \rightarrow \left[\begin{array}{ccc|ccc} & & & & & \\ & & & & & \\ \hline & & & & & \\ & & & & & \\ & & & & & \end{array} \right]$$



Phase 2 Implementations

We leave phase 2 implementations as suggested projects.

- Phase 2 implementation based on the QR-algorithm is quite straight-forward.
- Phase 2 implementation based on the divide-and-conquer paradigm requires careful consideration of all the “book-keeping” details. While not necessarily more difficult in a mathematical sense, the practical implementation of this approach is more challenging.
- The implementations in the referenced libraries: Lapack, ScaLAPACK, and cuSOLVER are thousands of lines long.



Phase 2 Implementations in the “Wild”

- The GNU Scientific Library (GSL) also implements an alternative approach: a one-sided Jacobi orthogonalization; the SVD of the bidiagonal matrix is obtained by solving a sequence of (2×2) SVD problems, similar to how the Jacobi eigenvalue algorithm solves a sequence of (2×2) eigenvalue methods
 - “*Matrix Computations*” 4th edition, Gene H. Golub and Charles F. Van Loan. Johns Hopkins University Press (2013). §8.6.3—“The SVD Algorithm”; §8.6.4—“Jacobi SVD Procedures”

SOURCE: https://en.wikipedia.org/wiki/Singular_value_decomposition#Numerical_approach



Phase 2 Implementations in the “Wild”

- LAPACK’s `dbdsqr/zbdsqr` implements an iterative variant of the QR algorithm
 - “*Calculating the Singular Values and Pseudo-Inverse of a Matrix*”, G. Golub and W. Kahan, Journal of the Society for Industrial and Applied Mathematics Series B Numerical Analysis, Volume 2, Issue 2, pp.205–224. (1964). <https://doi.org/10.1137/0702016>
 - “*Accurate Singular Values of Bidiagonal Matrices*”, James Demmel and W. Kahan, SIAM Journal on Scientific and Statistical Computing, Volume 11, Issue 5, pp.873–912. (1990). <https://doi.org/10.1137/0911052>

SOURCE: https://en.wikipedia.org/wiki/Singular_value_decomposition#Numerical_approach
REFERENCE: http://www.netlib.org/lapack/explore-html/d0/da6/group__complex16_o_t_h_e_rcomputational_gae7f455622680c22921ba25be440a726f.html

