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.

Computing the SVD

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.”
The Big Prize — Computing the SVD
The Computation, Phase 1
The Computation, Phase 2

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 Values of $A$ and Eigenvalues of $A^*A$

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 $\delta B$, 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

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

which implies

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

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 "squearing the condition number," much like in the least squares case.

Toward a Correct, Stable, Approach...

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

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

We can now write the eigenvalue decomposition of $H$

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

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.

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

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.

\[
\begin{pmatrix}
* & * & * & * \\
* & * & * & * \\
* & * & * & * \\
* & * & * & *
\end{pmatrix} \xrightarrow{U^*} \\
\begin{pmatrix}
* & * & * & * \\
0 & * & * & * \\
0 & 0 & * & * \\
0 & 0 & 0 & *
\end{pmatrix} \xrightarrow{V_1} \\
\begin{pmatrix}
* & * & * & * \\
0 & * & * & * \\
0 & 0 & * & * \\
0 & 0 & 0 & *
\end{pmatrix} \xrightarrow{U^*} \\
\begin{pmatrix}
* & 0 & 0 & 0 \\
* & * & * & * \\
* & * & * & * \\
* & * & * & *
\end{pmatrix}
\]

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

\[
\begin{pmatrix}
* & * & * & * \\
* & * & * & * \\
* & * & * & * \\
* & * & * & *
\end{pmatrix} \quad \xrightarrow{\text{Phase 1a}} \\
\begin{pmatrix}
* & * & * & * \\
0 & * & * & * \\
0 & 0 & * & * \\
0 & 0 & 0 & *
\end{pmatrix} \quad \xrightarrow{\text{Phase 1b}} \\
\begin{pmatrix}
* & * & * & * \\
0 & * & * & * \\
0 & 0 & * & * \\
0 & 0 & 0 & *
\end{pmatrix}
\]

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

\[
\text{Work} \sim \left( 2mn^2 + 2n^3 \right).
\]

Phase 1b

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)

\[
U^*AV = U_m^* \cdots U_1^* AV_1 \cdots V_{n-2} =
\]

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

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

Golub-Kahan vs. Lawson-Hanson-Chan Bidiagonalization

\[
\begin{array}{c|c|c}
\text{work / n}^3 & \text{m/n} & \text{work / n}^3 \\
\hline
2.5 & 1 & 3.0 \\
3 & 1.2 & 3.5 \\
3.5 & 1.4 & 4.0 \\
4 & 1.6 & 4.5 \\
4.5 & 1.8 & 5.0 \\
5 & 2 & 5.5 \\
\end{array}
\]

\text{Figure: Comparing the work for Golub-Kahan and Lawson-Hanson-Chan bidiagonalization. The break-even point is } 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.

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

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.

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 \( \text{rank}(\delta B) = 1 \):

\[
\begin{bmatrix}
* & * & \cdots & * \\
\vdots & \ddots & \ddots & \vdots \\
\end{bmatrix} =
\begin{bmatrix}
\delta_1 & \cdot \\
\cdot & \cdot \\
\end{bmatrix} +
\begin{bmatrix}
\delta_2 & \cdot \\
\cdot & \cdot \\
\end{bmatrix}
\]

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

\[
\begin{bmatrix}
\Sigma(\delta_1) & * \\
\cdot & \Sigma(\delta_2) \\
\end{bmatrix} \rightarrow
\begin{bmatrix}
\Sigma(\delta) \\
\cdot \\
\end{bmatrix}.
\]
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”

- LAPACK’s dbdsqr/zbdqsqr implements an iterative variant of the QR algorithm

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.html

- 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 $(2 \times 2)$ SVD problems, similar to how the Jacobi eigenvalue algorithm solves a sequence of $(2 \times 2)$ eigenvalue methods

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