Implicit-shifted Symmetric QR Singular Value Decomposition of $3 \times 3$ Matrices

Theodore Gast\textsuperscript{1}, Chuyuan Fu\textsuperscript{1, 3}, Chenfanfu Jiang\textsuperscript{1, 2}, and Joseph Teran\textsuperscript{1, 1}

\textsuperscript{1}Department of Mathematics, University of California, Los Angeles
\textsuperscript{2}Department of Computer Science, University of California, Los Angeles

Abstract

Computing the Singular Value Decomposition (SVD) of $3 \times 3$ matrices is commonplace in 3D computational mechanics and computer graphics applications. We present a C++ implementation of implicit symmetric QR SVD with Wilkinson shift. The method is fast and robust in both float and double precisions. We also perform a benchmark test to study the performance compared to other popular algorithms.

Keywords: SVD, implicit symmetric QR, Wilkinson shift, Jacobi rotation, eigenvalue, Givens rotation

1 Problem Description

Our goal is finding the SVD of a real $3 \times 3$ matrix $A$ so that

$$A = U \Sigma V^T,$$

where $U$ and $V$ are orthogonal matrices, $\Sigma$ is a diagonal matrix consisting of the singular values of $A$. In computational mechanics, $U$ and $V$ are often enforced to be rotation matrices which better represent geometric transformations. Furthermore, many authors use the conventions as in [Irving et al. 2004], e.g., [Sin et al. 2011; Stomakhin et al. 2012; Hegemann et al. 2013; Stomakhin et al. 2013; Bouaziz et al. 2014; Stomakhin et al. 2014; Saito et al. 2015; Gast et al. 2015; Xu et al. 2015; Klar et al. 2016]. The conventions are

- $U^T U = \mathbf{I}$, $V^T V = \mathbf{I}$;
- $\det(U) = 1$, $\det(V) = 1$;
- $\sigma_1 \geq \sigma_2 \geq |\sigma_3|$.

Note that $\sigma_3 < 0$ if $\det(A) < 0$.

2 Givens Rotation

The QR algorithm largely depends on Givens rotations. Once any $c$ and $s$ with $c^2 + s^2 = 1$ are computed from inputs $x$ and $y$, a 2D Givens rotation is defined as

$$G_2(1, 2, c(x, y), s(x, y)) = \begin{pmatrix} c & s \\ -s & c \end{pmatrix}.$$
Algorithm 3 Polar Decomposition of $2 \times 2$ matrices
1: procedure POLARDecomposition2D(A)
2:     $x \leftarrow A_{11} + A_{22}$
3:     $y \leftarrow A_{21} - A_{12}$
4:     $d \leftarrow \sqrt{x^2 + y^2}$
5:     $R \leftarrow G_2(1, 2, c = 1, s = 0)$ \Comment{R is a Givens rotation}
6:     if $d \neq 0$ then \Comment{no tolerance needed}
7:         $R \leftarrow G_2(1, 2, c = x/d, s = -y/d)$
8:     $S \leftarrow RowRotation(R, A)$ \Comment{R is a rotation, S is symmetric}
9:     return $(R, S)$

and $\hat{G}$. Note that we use a fast inverse square root function (Algorithm 2) from Streaming SIMD Extensions (SSE) intrinsics to accelerate the float case (we use the c++ function _mm_cvtss_f32(_mm_rsqrt_ss(_mm_set_ss(a)))). Similarly to [McAdams et al. 2011], accuracy is improved by performing an additional Newton step. In the case of double precision, we simply use the standard C++ square root function to maintain accuracy.

We further use the definition that
- $B = RowRotation(G, A)$ means $B = G^T A$,
- $B = ColumnRotation(G, A)$ means $B = AG$.

In practice these operations are implemented more efficiently by updating four entries of $A$ in place instead of performing matrix products.

3 SVD of $2 \times 2$ Matrices

As the to-be-presented algorithm proceeds, the problem will eventually degrade into computing the SVD of a $2 \times 2$ matrix. Here we briefly describe how to do so while obeying a similar sign convention ($U, V$ are rotations, $\sigma_1 \geq |\sigma_2|$).

Assuming $A$ is $2 \times 2$, the first step is computing its Polar Decomposition $A = RS$, where $R$ is a rotation and $S$ is symmetric. Assuming

$$R = \begin{pmatrix} c & s \\ -s & c \end{pmatrix},$$

requiring $R^T A$ being symmetric leads to $x s = y c$ where $x = A_{11} + A_{22}, y = A_{21} - A_{12}$. The two solutions are therefore

$$c = \frac{x}{\sqrt{x^2 + y^2}}, \quad s = \frac{y}{\sqrt{x^2 + y^2}}$$

or

$$c = \frac{-x}{\sqrt{x^2 + y^2}}, \quad s = \frac{-y}{\sqrt{x^2 + y^2}}$$

By taking the difference of $||S - I||^2_F$ from two solutions, it can be shown that choosing the first one always minimizes it, therefore guarantees the chosen $R$ is the closest rotation to $A$ (or $S$ is the closest symmetric matrix to $I$).

Once we have the symmetric matrix $S$, diagonalizing it with a Jacobi rotation can be done similarly by solving $c$ and $s$ from

$$\begin{pmatrix} c & s \\ -s & c \end{pmatrix} \begin{pmatrix} S_{11} & S_{12} \\ S_{12} & S_{22} \end{pmatrix} \begin{pmatrix} c & -s \\ s & c \end{pmatrix} = \begin{pmatrix} * & * \\ * & * \end{pmatrix}.$$ 

Attention need to be paid to prevent potential division by zero [Golub and Van Loan 2012]. Finally,

$$V = \begin{pmatrix} c & -s \\ s & c \end{pmatrix},$$

$$U = RV,$$

$$\Sigma = V^T SV.$$ 

After sorting $\sigma_1, \sigma_2$ to obey our sign convention and permuting columns of $U$ and $V$ accordingly, we have $A = U \Sigma V^T$ where

- $U^T U = I, V^T V = I$;
- $\det(U) = 1, \det(V) = 1$;
- $\sigma_1 \geq |\sigma_2|$.

2 $\times$ 2 Polar Decomposition and SVD are shown in Algorithm 3 and 4.

4 Implicit Symmetric QR SVD

The QR algorithm iteratively applies Givens rotations to a tridiagonal symmetric matrix (which in the SVD case corresponds to $T = A^T A$) to solve the symmetric eigenproblem. Instead of constructing $T$, implicit symmetric QR SVD works on an upper bidiagonal $A$ and implicitly does the same thing. This results in a much higher accuracy and improves efficiency [Golub and Van Loan 2012].

4.1 Bidiagonalization and Zerochasing

The implicit symmetric QR algorithm starts with making $A$ upper bidiagonal. For $3 \times 3$ matrices, this can be done with 4 Givens
Algorithm 5 Zerochasing: Assuming input $A_{31} = 0$, $U$, $V$ are rotations, this function makes $A$ upper bidiagonal while maintaining the product $UAV^T$ unchanged.

1: procedure ZEROCHASING($U$, $A$, $V$) \(\triangleright\) update them in place
2: \(G \leftarrow G_3(2, 3, x = A_{12}, y = A_{31})\)
3: \(A \leftarrow AG\), \(U \leftarrow G^TU\)
4: \(G \leftarrow G_3(2, 3, x = A_{12}, y = A_{31})\)
5: \(A \leftarrow G^TA\), \(V \leftarrow G^TV\)
6: \(G \leftarrow G_3(2, 3, x = A_{22}, y = A_{32})\)
7: \(A \leftarrow G^TA\), \(U \leftarrow UG\)
8: return ($U$, $A$, $V$)

Algorithm 6 Upper Bidiagonalizing: Assuming input $U$, $V$ are rotations, this function makes $A$ upper bidiagonal while maintaining the product $UAV^T$ unchanged.

1: procedure BIDIGONALIZE($U$, $A$, $V$) \(\triangleright\) update them in place
2: \(G \leftarrow G_3(2, 3, x = A_{21}, y = A_{31})\)
3: \(A \leftarrow G^TA\), \(U \leftarrow UG\)
4: \((U, A, V) \leftarrow Zerocaching(U, A, V)\)
5: return ($U$, $A$, $V$)


\[
\begin{align*}
A^{(1)} &= \begin{pmatrix} * & * & * \\ * & * & * \\ - & - & - \\ \end{pmatrix} = G_3^{(1)}(2, 3, x = A_{21}, y = A_{31})^TA^{(0)} = \begin{pmatrix} * & * & * \\ * & * & * \\ - & - & - \\ \end{pmatrix}, \\
A^{(2)} &= \begin{pmatrix} * & * & - \\ - & - & - \\ \end{pmatrix} = G_3^{(2)}(1, 2, x = A_{11}, y = A_{21})^TA^{(1)} = \begin{pmatrix} * & * & - \\ - & - & - \\ \end{pmatrix}, \\
A^{(3)} &= \begin{pmatrix} * & * & - \\ - & * & - \\ \end{pmatrix} = A^{(2)}G_3^{(3)}(2, 3, x = A_{12}, y = A_{32}) = \begin{pmatrix} * & * & - \\ - & * & - \\ \end{pmatrix}, \\
A^{(4)} &= \begin{pmatrix} * & * & - \\ - & * & - \\ \end{pmatrix} = G_3^{(4)}(2, 3, x = A_{22}, y = A_{32})^TA^{(3)} = \begin{pmatrix} * & * & - \\ - & * & - \\ \end{pmatrix}.
\end{align*}
\]

In summary, $A^{(4)} = G_3^{(4)}TA_3^{(2)}G_3^{(3)}TA^{(0)}G_3^{(4)}$. The Givens rotations need to be absorbed by $U$ and $V$ accordingly during the process. The later three steps of this process is further called Zerochasing, which takes a matrix of form
\[
\begin{pmatrix}
* & * & * \\
* & * & * \\
\end{pmatrix}
\]

and make it
\[
\begin{pmatrix}
* & * & * \\
* & * & * \\
\end{pmatrix}.
\]

We will be using it again in every implicit symmetric QR iteration (see Section 4.2). We summarize the algorithms for Zerochasing and upper bidiagonalization in Algorithm 5 and 6.

### 4.2 Implicit Symmetric QR SVD with Wilkinson Shift

Our algorithm follows [Golub and Van Loan 2012]. Starting from $U = I$ and $V = I$, we first perform the upper bidiagonalization described in Section 4.1 to matrix $A$ with $U$ and $V$ also updated. We use $B$ to denote the bidiagonal matrix, where we have $UBV^T = A$. The implicit QR iteration operates on $B$ iteratively and update $U$ and $V$ on the fly. Denoting $B$ with
\[
B = \begin{pmatrix}
\alpha_1 & \beta_1 \\
\alpha_2 & \beta_2 \\
\alpha_3 \\
\end{pmatrix},
\]
the corresponding symmetric eigenproblem is on the matrix
\[
T = B^TB = \begin{pmatrix}
\alpha_1^2 & \alpha_1\beta_1 & \alpha_1\beta_2 \\
\alpha_1\beta_1 & \alpha_2^2 + \beta_1^2 & \alpha_2\beta_2 \\
\alpha_1\beta_2 & \alpha_2\beta_2 & \alpha_3^2 + \beta_2^2 \\
\end{pmatrix}.
\]

QR iteration seeks to eliminate the off-diagonal entries of $T$. Equivalently, one or more values of $(\alpha_1, \beta_1, \alpha_2, \beta_2)$ will converge to something close to zero. We will show in Section 4.3 that once any of $(\alpha_1, \beta_1, \alpha_2, \beta_2, \alpha_3)$ becomes smaller than a tolerance $\tau$, we can terminate the QR iterations and degrade the problem to a $2 \times 2$ SVD. The termination tolerance $\tau$ is computed as a relative tolerance via
\[
\tau = \max \left\{ \frac{1}{2} ||B||_F, 1 \right\} \eta
\]
where we choose $\eta = 128\epsilon$ and $\epsilon$ is the floating point machine epsilon.

**QR Factorization** The QR Factorization of a symmetric tridiagonal matrix $T \in \mathbb{R}^{n \times n}$ can be easily done using $n - 1$ Givens rotations with $Q$ being a rotation matrix and $R$ being upper triangular.

**QR Iteration** If $A \in \mathbb{R}^{n \times n}$ is symmetric, $R_0$ is orthogonal and $T_0 = R_0^TAR_0$, then the iteration
\[
T_{k-1} = Q_kR_k, \\
T_k = R_k^TQ_k
\]
implies $T_k = (R_0R_1 \ldots R_k)^T A (R_0R_1 \ldots R_k)$ is symmetric tridiagonal, and converges to a diagonal form [Trefethen and Bau III 1997; Golub and Van Loan 2012].

**Implicit Q Theorem** Given $A \in \mathbb{R}^{n \times n}$ symmetric, $Q^TAQ = T$, $V^TAV = S$, and $Q$ and $V$ are orthogonal, $T$ and $S$ are symmetric tridiagonal. If $A$ is unreduced (meaning it has non-zero sub-diagonal entries) and the first column of $Q$ and $V$ are equal ($q_1 = v_1$), then $q_1 = \pm v_1$ and $|T_{ij}| = |S_{ij}|$ [Golub and Van Loan 2012].

**Explicit Shifted QR Iteration** If $\mu$ is a good approximate eigenvalue of $T$, then $T_{n-1}$ tends to become smaller after a shifted QR step:
\[
T - \mu I = QR, \\
T_{\text{new}} = RQ + \mu I = Q^TQ
\]
and $T$ maintains a symmetric tridiagonal form [Golub and Van Loan 2012].

**Wilkinson Shift** A good choice of the shift $\mu$ is the eigenvalue of $T$’s bottom right $2 \times 2$ block that is closer to $T_{nn}$ [Golub and Van Loan 2012]. This shift gives average cubic convergence rate for reducing $T_{n-1}$ to zero. In the $3 \times 3$ case where
\[
T = \begin{pmatrix}
a_1 & b_1 & b_2 \\
b_1 & a_2 & b_3 \\
b_2 & b_3 & a_3 \\
\end{pmatrix},
\]
the shift is given by \( \mu = a_3 + d - \text{sign}(d) \sqrt{d^2 + b_d^2} \) where \( d = (a_2 - a_3)/2 \) and \( \text{sign}(d) = \pm 1 \) (choose 1 when \( d = 0 \)).

Implicit Shifted QR Iteration The shifted QR iteration can be done without constructing \( T - \mu I \) explicitly. Let’s focus on the \( 3 \times 3 \) case where we have

\[
T - \mu I = \begin{pmatrix}
  a_1 - \mu & b_1 & b_2 \\
  b_1 & a_2 - \mu & a_3 \\
  b_2 & a_3 & a_3 - \mu
\end{pmatrix}.
\]

The QR decomposition \( T - \mu I = QR \) looks like

\[
\begin{pmatrix}
  a_1 - \mu & b_1 & b_2 \\
  b_1 & a_2 - \mu & a_3 \\
  b_2 & a_3 & a_3 - \mu
\end{pmatrix} = \begin{pmatrix}
  q_1 & q_2 & q_3
\end{pmatrix} \begin{pmatrix}
  * & * & * \\
  * & * & * \\
  * & * & *
\end{pmatrix},
\]

this implies \( q_1 = \gamma(a_1 - \mu, b_1, 0)^T \) for some normalization scale \( \gamma \). If we construct a Givens rotation \( G^1 = G_1(1, 2, x = a_1 - \mu, y = \beta_1) \), then it follows \( g_1 = \omega(a_1 - \mu, b_1, 0)^T \) for some normalization scale \( \omega \). Therefore we know \( g_1 = q_1 \), i.e., \( G^1 \) and \( Q \) has the same first column. If we further find \( G^2 \) such that \( Z = G^1 G^2 \) has the same first column with \( G^1 \) and \( S = Z^T TZ \) is symmetric tridiagonal, then by implicit Q Theorem, since \( T^{new} = Q^T TQ \) and \( S = Z^T TZ \), it follows \( q_i = \pm \alpha_i \), and \( |T_{ij}^{new}| = |S_{ij}| \). Therefore, utilizing \( G^1 \) and \( G^2 \) accomplishes the same effect as an explicit shifted QR iteration step for updating \( T \).

Implicit Shifted QR in the SVD Case For SVD, we prefer operating on \( B \) directly to constructing \( T \). Applying \( G^2 \) directly to \( B \) followed by Zerohasing \( B \) back to upper bidiagonal is equivalent to doing implicit QR on \( T^* \) [Golub and Van Loan 2012]. More specifically in our \( 3 \times 3 \) case, after applying \( G_1 \) as a column rotation to \( B \), the column rotation in the Zerohasing (i.e., \( G_2^2 \) in Section 4.1) essentially is the \( G^2 \) we want to find in the implicit QR for \( T \) with the property that \( G^1 G^2 \) has the same first column with \( Q \). Therefore by operating on \( B \) directly, the implicit symmetric QR algorithm is correctly applied.

We summarize the implicit shifted QR SVD in Algorithm 7. The steps after exiting the loop is described in Section 4.3.

### 4.3 Postprocess and Sorting

If any \( \alpha \) or \( \beta \) from Algorithm 7 becomes small, implicit QR iteration is terminated. Here we show how each case is degraded to a \( 2 \times 2 \) easy problem.

#### 4.3.1 Deflation Cases

**Case 1: \( |\beta_1| \leq \tau \).** In this case

\[
B = \begin{pmatrix}
  * & * & * \\
  * & * & * \\
  * & * & *
\end{pmatrix},
\]

we just need to compute the \( 2 \times 2 \) SVD of the bottom right submatrix and assemble back to 3D.

**Case 2: \( |\beta_1| \leq \tau \).** In this case

\[
B = \begin{pmatrix}
  * & * & * \\
  * & * & * \\
  * & * & *
\end{pmatrix},
\]

we just need to compute the \( 2 \times 2 \) SVD of the bottom right submatrix and assemble back to 3D.

**Case 3: \( |\alpha_2| \leq \tau \).** In this case

\[
B = \begin{pmatrix}
  * & * & * \\
  * & * & * \\
  * & * & *
\end{pmatrix},
\]

performing an unconventional Givens rotation \( G = G_3(2, 3, x = B_{23}, y = B_{33}) \) with \( B \leftarrow G^2 B \) reduces \( B \) to the form

\[
B = \begin{pmatrix}
  * & * & * \\
  * & * & *
\end{pmatrix},
\]

where we just need to compute the \( 2 \times 2 \) SVD of the top left submatrix and assemble back to 3D.

**Case 4: \( |\alpha_3| \leq \tau \).** In this case

\[
B = \begin{pmatrix}
  * & * & * \\
  * & * & *
\end{pmatrix},
\]

We can use \( G = G_3(2, 3, x = B_{23}, y = B_{33}) \) with \( B \leftarrow BG \) to reduce \( B \) to the form

\[
B = \begin{pmatrix}
  * & * & * \\
  * & * & * \\
  * & * & *
\end{pmatrix}.
\]
followed by $G = G_3(1,3, x = B_{11}, y = B_{33})$ with $B \leftarrow BG$ to further reduce to 

$$B = \begin{pmatrix} * & * & - \\ * & * \\ + & + \\ \end{pmatrix},$$

where we just need to compute the $2 \times 2$ SVD of the top left sub-matrix and assemble back to 3D.

**Case 5: $|\alpha_1| \leq \tau$.** In this case

$$B = \begin{pmatrix} * & * \\ * & * \\ \end{pmatrix}.$$

Performing an unconventional Givens rotation $\hat{G} = \hat{G}_3(1,2, x = B_{12}, y = B_{22})$ with $B \leftarrow G^T B$ reduces $B$ to the form

$$B = \begin{pmatrix} - & + \\ * & * \\ + & + \\ \end{pmatrix}.$$

Further performing an unconventional Givens rotation $\hat{G} = \hat{G}_3(1,3, x = B_{13}, y = B_{33})$ with $B \leftarrow G^T B$ reduces $B$ to the form

$$B = \begin{pmatrix} * & - \\ + & * \\ \end{pmatrix},$$

where we just need to compute the $2 \times 2$ SVD of the bottom right sub-matrix and assemble back to 3D.
Timing (float)
QR SVD ITF 04 Eigen Jacobi Vega FEM
1 0.3438 0.3637 1.4422 0.6401
2 0.5669 0.6597 2.5292 1.0886
3 ...
Test cases (double)
1 2 3 4 5
QR SVD ITF 04 Eigen Jacobi Vega FEM

Reconstruction Maximum Error (float)
QR SVD ITF 04 Eigen Jacobi Vega FEM
2 ...
4 2.850E-14 3.520E-15 2.887E-15 2.442E-15
5 2.820E-14 3.113E-14 4.219E-15 2.665E-15

In summary, the Implicit QR SVD described in this document provides a nice balance between speed and accuracy. We release our C++ code together with this document and expect it to benefit many applications in computer graphics and computational solid mechanics.

Acknowledgements

We thank Yixin Zhu for the helpful discussions. The authors were partially supported by NSF CCF-1422795, ONR (N000141110719, N000141210834), DOD (W81XWH-15-1-0147), Intel STC-Visual Computing Grant (20112360) as well as a gift from Disney Research.

References


Algorithm 10 Sorting Singular Values

1: procedure SORTWITHTOPLEFTSUB(U, Σ, V) \( \triangleright \sigma_1 \geq |\sigma_2| \)
2: \( \text{if } \sigma_2 \geq |\sigma_3| \) then
3: \( \text{if } \sigma_2 < 0 \) then
4: \( \text{FlipSign}(2, U, Σ) \quad \triangleright \text{sign of } \sigma_2 \) and col 2 of U
5: \( \text{FlipSign}(3, U, Σ) \quad \triangleright \text{sign of } \sigma_3 \) and col 3 of U
6: \( \text{return } (U, Σ, V) \quad \triangleright \sigma_1 \geq \sigma_2 \geq |\sigma_3| \)
7: \( \text{if } \sigma_3 < 0 \) then
8: \( \text{FlipSign}(2, U, Σ) \)
9: \( \text{FlipSign}(3, U, Σ) \)
10: \( \text{swap}(\sigma_2, \sigma_3) \)
11: \( \text{swap}(U.\text{col}(2), U.\text{col}(3)) \)
12: \( \text{swap}(V.\text{col}(2), V.\text{col}(3)) \)
13: \( \text{if } \sigma_2 > \sigma_3 \) then
14: \( \text{swap}(\sigma_1, \sigma_3) \)
15: \( \text{swap}(U.\text{col}(1), U.\text{col}(2)) \)
16: \( \text{swap}(V.\text{col}(1), V.\text{col}(2)) \)
17: \( \text{else} \)
18: \( U.\text{col}(3) \leftarrow -U.\text{col}(3) \)
19: \( V.\text{col}(3) \leftarrow -V.\text{col}(3) \)
20: \( \text{return } (U, Σ, V) \quad \triangleright \sigma_1 \geq \sigma_2 \geq |\sigma_3| \)
21: procedure SORTWITHBOTRIGHTSUB(U, Σ, V) \( \triangleright \sigma_2 \geq |\sigma_3| \)
22: \( \text{if } |\sigma_1| \geq |\sigma_2| \) then
23: \( \text{if } \sigma_1 < 0 \) then
24: \( \text{FlipSign}(1, U, Σ) \)
25: \( \text{FlipSign}(3, U, Σ) \)
26: \( \text{return } (U, Σ, V) \quad \triangleright \sigma_1 \geq \sigma_2 \geq |\sigma_3| \)
27: \( \text{swap}(\sigma_1, \sigma_2) \)
28: \( \text{swap}(U.\text{col}(1), U.\text{col}(2)) \)
29: \( \text{swap}(V.\text{col}(1), V.\text{col}(2)) \)
30: \( \text{if } |\sigma_2| < |\sigma_3| \) then
31: \( \text{swap}(\sigma_2, \sigma_3) \)
32: \( \text{swap}(U.\text{col}(2), U.\text{col}(3)) \)
33: \( \text{swap}(V.\text{col}(2), V.\text{col}(3)) \)
34: \( \text{else} \)
35: \( U.\text{col}(2) \leftarrow -U.\text{col}(2) \)
36: \( V.\text{col}(2) \leftarrow -V.\text{col}(2) \)
37: \( \text{if } \sigma_2 < 0 \) then
38: \( \text{FlipSign}(2, U, Σ) \)
39: \( \text{FlipSign}(3, U, Σ) \)
40: \( \text{return } (U, Σ, V) \quad \triangleright \sigma_1 \geq \sigma_2 \geq |\sigma_3| \)


