Computer Sciences Department

Computing the Singular Value Decomposition of 3×3 matrices with minimal branching and elementary floating point operations

Aleka McAdams
Andrew Selle
Rasmus Tamstorf
Joseph Teran
Eftychios Sifakis

Technical Report #1690

May 2011

UNIVERSITY OF WISCONSIN MADISON

Computing the Singular Value Decomposition of 3×3 matrices with minimal branching and elementary floating point operations

Aleka McAdams[1,2] Andrew Selle[1] Rasmus Tamstorf[1] Joseph Teran[2,1] Eftychios Sifakis[3,1]

[1] Walt Disney Animation Studios [2] University of California, Los Angeles
[3] University of Wisconsin, Madison

Abstract

A numerical method for the computation of the Singular Value Decomposition of 3×3 matrices is presented. The proposed methodology robustly handles rank-deficient matrices and guarantees orthonormality of the computed rotational factors. The algorithm is tailored to the characteristics of SIMD or vector processors. In particular, it does not require any explicit branching beyond simple conditional assignments (as in the C++ ternary operator `?:`, or the SSE4.1 instruction `VBLENDPS`), enabling trivial data-level parallelism for any number of operations. Furthermore, no trigonometric or other expensive operations are required; the only floating point operations utilized are addition, multiplication, and an inexact (yet fast) reciprocal square root which is broadly available on current SIMD/vector architectures. The performance observed approaches the limit of making the 3×3 SVD a memory-bound (as opposed to CPU-bound) operation on current SMP platforms.

Keywords: singular value decomposition, Jacobi eigenvalue algorithm

1 Method overview

Let $\mathbf{A}$ be a real-valued, 3×3 matrix. A factorization of $\mathbf{A}$ as

$$\mathbf{A} = \mathbf{U}\mathbf{\Sigma}\mathbf{V}^T$$

is guaranteed to exist, where $\mathbf{U}$ and $\mathbf{V}$ are 3×3 real orthogonal matrices and $\mathbf{\Sigma}$ is a 3×3 diagonal matrix with real and nonnegative diagonal entries. Since the matrix product $\mathbf{U}\mathbf{\Sigma}\mathbf{V}^T$ remains invariant if the same permutation is applied to the columns of $\mathbf{U}$,$\mathbf{V}$ and to the diagonal entries of $\mathbf{\Sigma}$, a common convention is to choose $\mathbf{\Sigma}$ so that its diagonal entries appear in non-increasing order. The exact convention followed in our method is slightly different, specifically:

- The orthogonal factors $\mathbf{U}$ and $\mathbf{V}$ will be true rotation matrices by construction (i.e. $\det(\mathbf{U}) = \det(\mathbf{V}) = 1$). This is contrasted to the possibility of $\mathbf{U}$ or $\mathbf{V}$ having a determinant of -1, which corresponds to a rotation combined with a reflection.
- The diagonal entries of $\mathbf{\Sigma}$ will be sorted in decreasing order of magnitude, but will not necessarily be non-negative (relaxing the non-negativity constraint is necessary to allow $\mathbf{U}$ and $\mathbf{V}$ to be true rotations, since the determinant of $\mathbf{A}$ could be either positive or negative). More specifically, the singular value with the smallest magnitude (σ_3, or $\mathbf{\Sigma}_{33}$) will have the same sign as $\det(\mathbf{A})$, while the two larger singular values σ_1, σ_2 will be non-negative.

These conventions are motivated by applications in graphics which require the orthogonal matrices $\mathbf{U}$ and $\mathbf{V}$ to correspond to real 3D spatial rotations. In any case, different conventions can be enforced as a post process with simple manipulations such as negating and/or permuting singular values and vectors.

The algorithm first determines the factor $\mathbf{V}$ by computing the eigenanalysis of the matrix $\mathbf{A}^T\mathbf{A} = \mathbf{V}\mathbf{\Sigma}^2\mathbf{V}^T$ which is symmetric and positive semi-definite. This is accomplished via a modified Jacobi iteration where the Jacobi factors are approximated using inexpensive, elementary arithmetic operations as described in section 2. Since the symmetric eigenanalysis also produces $\mathbf{\Sigma}^2$, and consequently $\mathbf{\Sigma}$ itself, the remaining factor $\mathbf{U}$ can theoretically be obtained as $\mathbf{U} = \mathbf{A}\mathbf{V}\mathbf{\Sigma}^{-1}$; however this process is not applicable when $\mathbf{A}$ (and as a result, $\mathbf{\Sigma}$) is singular, and can also lead to substantial loss of orthogonality in $\mathbf{U}$ for an ill-conditioned, yet nonsingular, matrix $\mathbf{A}$. Another possibility is to form $\mathbf{A}\mathbf{V} = \mathbf{U}\mathbf{\Sigma}$ and observe that this matrix contains the columns of $\mathbf{U}$, each scaled by the respective diagonal entry of $\mathbf{\Sigma}$. The Gram-Schmidt process could generate $\mathbf{U}$ as the orthonormal basis for $\mathbf{A}\mathbf{V}$, yet this method would still suffer from instability in the case of near-zero singular values. Our approach is based on the $\mathbf{QR}$ factorization of the matrix $\mathbf{A}\mathbf{V}$, using Givens rotations. With proper attention to some special cases, as described in section 4, the Givens $\mathbf{QR}$ procedure is guaranteed to produce a factor $\mathbf{Q}(= \mathbf{U})$ which is exactly orthogonal by construction, while the upper-triangular factor $\mathbf{R}$ will be shown to be in fact *diagonal* and identical to $\mathbf{\Sigma}$ up to sign flips of the diagonal entries.

The novel contribution of our approach lies in the computation of the Jacobi/Givens rotations without the use of expensive arithmetic such as trigonometric functions, square roots or even division; the only necessary operations are addition, multiplication and an *inexact* reciprocal square root function (which is available in many architectures and is often faster not only than square root, but standard division as well). The method is robust for any real matrix, and converges to a given accuracy within a fixed small number of iterations.

2 Symmetric eigenanalysis

The first step in computing the decomposition $\mathbf{A} = \mathbf{U}\mathbf{\Sigma}\mathbf{V}^T$ is to compute the eigenanalysis of the symmetric, positive semidefinite matrix $\mathbf{S} = \mathbf{A}^T\mathbf{A} = \mathbf{V}\mathbf{\Sigma}^2\mathbf{V}^T$. This will be performed by a variant of the Jacobi eigenvalue algorithm [Golub and van Loan 1989].

2.1 Jacobi iteration

We provide a summary presentation of the classical Jacobi eigenvalue algorithm; for a more detailed exposition the reader is referred to [Golub and van Loan 1989]. The Jacobi process constructs a sequence of similarity transforms

$$\mathbf{S}^{(k+1)} = [\mathbf{Q}^{(k)}]^T\mathbf{S}^{(k)}\mathbf{Q}^{(k)}.$$

Each matrix $\mathbf{Q}^{(k)}$ is constructed as a Givens rotation, of the form

$$\mathbf{Q}(p,q,\theta) = \begin{pmatrix} 1 & \cdots & 0 & \cdots & 0 & \cdots & 0 \\ \vdots & \ddots & \vdots & & \vdots & & \vdots \\ 0 & \cdots & c & \cdots & -s & \cdots & 0 \\ \vdots & & \vdots & \ddots & \vdots & & \vdots \\ 0 & \cdots & s & \cdots & c & \cdots & 0 \\ \vdots & & \vdots & & \vdots & \ddots & \vdots \\ 0 & \cdots & 0 & \cdots & 0 & \cdots & 1 \end{pmatrix} \begin{matrix} \\ \\ \leftarrow \texttt{row p} \\ \\ \leftarrow \texttt{row q} \\ \\ \\ \end{matrix}$$

where $c = \cos(\theta)$ and $s = \sin(\theta)$. The objective of every conjugation with a Given matrix $\mathbf{Q}^{(k)} = \mathbf{Q}(p,q,\theta)$ is to bring the next iterate $\mathbf{S}^{(k+1)}$ closer to a diagonal form, by eliminating the off-diagonal entries $s_{pq}^{(k)}$ and $s_{qp}^{(k)}$ (i.e. by enforcing that $s_{pq}^{(k+1)} = s_{qp}^{(k+1)} = 0$). It can be shown that

$$\sum_{i \neq j} [s_{ij}^{(k+1)}]^2 = \sum_{i \neq j} [s_{ij}^{(k)}]^2 - 2[s_{pq}^{(k)}]^2.$$

Therefore, after each Jacobi iteration, the sum of squared off-diagonal entries of $\mathbf{S}$ is reduced by the sum of squares of the two off-diagonal annihilated entries. There is a natural choice of the entries to be eliminated that can easily seen to generate a convergent process: if we annihilate the *maximum* off-diagonal entry of an 3×3 we are guaranteed to annihilate at least $1/3$ of the off-diagonal sum-of-squares (in fact, at each iteration after the first, this reduction will be at least $1/2$ of the previous sum of squares, since previous iterations leave only one other non-zero off-diagonal pair).

The Jacobi iteration rapidly drives the iterates $\mathbf{S}^{(k)}$ to a diagonal form. For 3×3 matrices, $10-15$ iterations are typically sufficient to diagonalize $\mathbf{S}$ to within single-precision roundoff error. Although the previous argument suggests at least linear convergence, the order becomes in fact quadratic once $\mathbf{S}^{(k)}$ has been brought somewhat close to diagonal form. Additionally, it can be shown that the same asymptotic order of convergence is attained even if the off-diagonal elements to be eliminated are selected in a fixed, *cyclic* order, i.e.

$$(p,q) = (1,2), (1,3), (2,3), (1,2), (1,3), \ldots$$

This alleviates the need for conditional execution based on the magnitude of the off-diagonal entries. *Note: Cyclic Jacobi requires that we always pick $|\theta| < \pi/4$ to ensure convergence, an option that is always possible as illustrated next.* The rest of this section addresses the computation of the trigonometric factors $c = \cos(\theta)$ and $s = \sin(\theta)$. Once the indices (p,q) of the off-diagonal pair to be annihilated have been selected, the values of c and s depend only on the 2×2 submatrix at the intersection of the p-th row and q-th column (note that $s_{pq} = s_{qp}$ due to symmetry):

$$\begin{pmatrix} s_{pp} & s_{pq} \\ s_{pq} & s_{qq} \end{pmatrix}$$

Thus, the determination of c and s is equivalent to the problem of diagonalizing a 2×2 symmetric matrix.

2.2 The classical 2×2 Givens diagonalization

Let $\mathbf{A}$ be a symmetric 2×2 matrix. We seek trigonometric factors $c = \cos(\theta)$ and $s = \sin(\theta)$ such that the result of the conjugation:

$$\mathbf{B} = \mathbf{Q}^T \mathbf{A} \mathbf{Q} = \begin{pmatrix} c & s \\ -s & c \end{pmatrix} \begin{pmatrix} a_{11} & a_{12} \\ a_{12} & a_{22} \end{pmatrix} \begin{pmatrix} c & -s \\ s & c \end{pmatrix}$$

$$= \begin{pmatrix} c^2 a_{11} + 2csa_{12} + s^2 a_{22} & cs(a_{22} - a_{11}) + (c^2 - s^2)a_{12} \\ cs(a_{22} - a_{11}) + (c^2 - s^2)a_{12} & s^2 a_{11} - 2csa_{12} + c^2 a_{22} \end{pmatrix}$$

is a diagonal matrix. Therefore, we need to enforce that

$$b_{12} = b_{21} = cs(a_{22} - a_{11}) + (c^2 - s^2)a_{12} = 0$$

If $a_{22} - a_{11} = 0$, we can simply select $\theta = \pi/4$ (or equivalently, $c = s = 1/\sqrt{2}$). Otherwise, the previous condition can be rewritten as:

$$\begin{aligned} \frac{cs}{c^2 - s^2} &= \frac{a_{12}}{a_{11} - a_{22}} \\ \frac{2\cos\theta\sin\theta}{\cos^2\theta - \sin^2\theta} &= \frac{2a_{12}}{a_{11} - a_{22}} \\ \frac{2\tan\theta}{1 - \tan^2\theta} &= \frac{2a_{12}}{a_{11} - a_{22}} \quad (1) \\ \tan(2\theta) &= \frac{2a_{12}}{a_{11} - a_{22}} \quad (2) \end{aligned}$$

From equation (2) we have $\theta = \frac{1}{2}\arctan(2a_{12}/(a_{11} - a_{22}))$. If the arc-tangent function returns value in the interval $(-\pi/2, \pi/2)$, this expression will guarantee $|\theta| < \pi/4$ as required for convergence of Cyclic Jacobi. Naturally, the use of (forward and inverse) trigonometric functions will significantly increase the cost of this computation. An alternative technique is described in [Golub and van Loan 1989] where the quadratic equation (1) is solved to yield $\tan\theta$ directly, from which the sine and cosine are computed algebraically. This approach still requires a minimum of two reciprocals, one square root and one *exact* reciprocal-of-square-root in addition to any multiply/add operations (an extra reciprocal plus a square root will be needed if branching instructions are to be avoided).

Our proposed optimization quickly computes an approximate form of the Givens rotation, where inaccuracy may be introduced both in the angle computation, as well as a constant scaling of the rotation matrix. However, the nature of this approximation is such that the Jacobi procedure is only impacted by a minimal deceleration, while the accuracy of the converged result is not compromised. Our methodology requires only multiply/add operations, plus a single inexpensive, *inexact* reciprocal-of-square-root evaluation (even large relative errors are well acceptable). No branching is required, with the exception of conditional assignments. Additional computational savings arise from the compact representation of rotations as quaternions, instead of explicit matrices.

2.3 Approximating the trigonometric factors

In this section, we introduce a first approximation to the trigonometric factors c, s in the Givens matrices, which does not require evaluating trigonometric functions or solving a quadratic. Our approach stems from an asymptotic approximation when the rotation angle θ is small. Equation (2) suggests that this would be the case for example when the Jacobi iteration is close to convergence and $\mathbf{\Sigma}$ does not have repeated singular values; nevertheless, our process is designed to guarantee reasonable progress regardless of any such conditions.

Let us temporarily assume that θ is small. Under this assumption, we can approximate $\tan(2\theta) \approx 2\tan\theta$. In fact, let us denote with

ϕ the angle that satisfies the equation $\tan(2\theta) = 2\tan\phi$ as an exact identity; we can then equivalently state that when θ is small, we will have $\phi \approx \theta$. We summarize these approximations, in conjunction with equation (2) as follows:

$$\tan(2\theta) = \frac{2a_{12}}{a_{11} - a_{22}} = 2\tan\phi \overset{\theta \ll 1}{\approx} 2\tan(\theta). \tag{3}$$

This expression can be rewritten to provide the following expression for $\cos(\phi)$ and $\sin(\phi)$ (which, in turn, approximate the trigonometric factors $c = \cos(\theta)$ and $s = \sin(\theta)$, respectively):

$$\frac{a_{12}}{a_{11} - a_{22}} = \frac{\sin\phi}{\cos\phi} \Rightarrow \begin{cases} \sin\phi = \omega a_{12} \\ \cos\phi = \omega(a_{11} - a_{22}) \\ \omega = 1/\sqrt{a_{12}^2 + (a_{11} - a_{22})^2} \end{cases} \tag{4}$$

What is the quality of this approximation, however, specifically for our purposes of generating a diagonal matrix $\mathbf{B} = \mathbf{Q}^T\mathbf{A}\mathbf{Q}$? This can be quantified by looking at the off-diagonal element b_{12} generated after the conjugation with these approximate Givens rotations:

$$\begin{aligned} b_{12} &= \cos\phi\sin\phi(a_{22} - a_{11}) + (\cos^2\phi - \sin^2\phi)a_{12} \\ &= \sin(2\phi)\frac{a_{22} - a_{11}}{2} + \cos(2\phi)a_{12} \\ &\overset{Eq.(2)}{=} \sin(2\phi)\frac{a_{12}}{\tan(2\theta)} + \cos(2\phi)a_{12} \\ &= \frac{\sin(2\phi)\cos(2\theta) + \cos(2\phi)\sin(2\theta)}{\sin(2\theta)}a_{12} \\ &= \frac{\sin(2\phi - 2\theta)}{\sin(2\theta)}a_{12} \Rightarrow \\ \overset{Eq.(3)}{\Longrightarrow} \frac{|b_{12}|}{|a_{12}|} &= \left|\frac{\sin(2\arctan(\tan(2\theta)/2) - 2\theta)}{\sin(2\theta)}\right| \end{aligned} \tag{5}$$

Equation (5) provides a concise expression for the reduction of the magnitude of the off-diagonal element, as a function of the optimal Givens rotation angle (which is, in turn, a function of the matrix entries). Figure 1 (solid line) illustrates the magnitude reduction fraction as a function of the angle θ. We observe that for small values of θ the quality of the approximation is excellent, effectively leading to annihilation of the off-diagonal element. However, for larger values of θ, the reduction becomes smaller, and we actually obtain no reduction at all for values $\theta \approx \pi/4$.

The poor performance of the previous approximation when $\theta \approx \pi/4$ will be addressed by considering yet another choice for the Givens angle ϕ. Equation (4) reveals that this approximate ϕ may lie outside the interval $(-\pi/4, \pi/4)$, which in contrast could have been guaranteed for the optimal angle θ. This restriction was important in ensuring convergence of the Cyclic Jacobi method. Thus, we consider the possibility of truncating the approximate value to the value $\phi = \pi/4$, at the very least in the case when the computed value lies outside that interval. For this fixed choice of the Givens angle, the off-diagonal element b_{12} after the conjugation becomes:

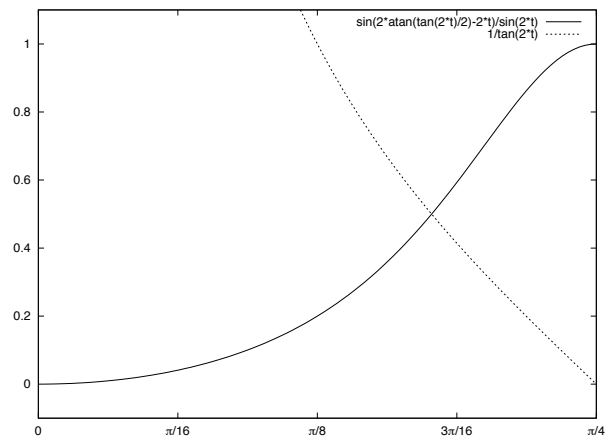

Figure 1: *Approximating the trigonometric Givens factors using $\tan(2\theta) \approx 2\tan\theta$ (or by setting a fixed angle $\theta = \pi/4$).The off-diagonal magnitude reduction fraction $|b_{12}|/|a_{12}|$ is plotted on the vertical axis, as a function of the optimal Givens angle.*

$$\begin{aligned} b_{12} &= \cos\frac{\pi}{4}\sin\frac{\pi}{4}(a_{22} - a_{11}) + (\cos^2\frac{\pi}{4} - \sin^2\frac{\pi}{4})a_{12} \\ &= \frac{a_{22} - a_{11}}{2} \\ &= \frac{a_{12}}{\tan(2\theta)} \\ \Longrightarrow \frac{|b_{12}|}{|a_{12}|} &= \frac{1}{|\tan(2\theta)|} \end{aligned} \tag{6, 7}$$

This reduction fraction is also plotted in figure 1 (dashed line). Note that both magnitude reduction fractions are even, and periodic ($T = \pi/2$) functions, so it is sufficient to study them in the interval $[0, \pi/4]$. Notably, if we were able to pick the *best* of the two proposed approximations (in terms of the magnitude reduction they produce) we see that a reduction fraction significantly smaller than 1 can be guaranteed. In fact, it is possible to formalize this selection between the two approximations; from equations (3,7) we can solve for the intersection point of the two curves in figure 1 as $\theta_0 = \arctan(2)/2 \approx 0.55357$ (we omit the relevant trigonometric manipulations). The magnitude reduction ratio $|b_{12}|/|a_{12}|$ at the intersection point is equal to $1/|\tan(2\theta_0)| = 0.5$. Thus, by selecting the best of the two approximations we are guaranteed at least a 50% reduction in the magnitude of the off-diagonal element. Finally, the choice about which approximation is the best one to use can be made without resorting to the obvious angle criterion ($\theta < \theta_0$), by observing that the fixed angle $\phi = \pi/4$ should be selected only when it yields a magnitude reduction by a factor no larger than 0.5:

$$\begin{aligned} \frac{|b_{12}|}{|a_{12}|} &\leq \frac{1}{2} \\ \frac{|a_{11} - a_{22}|}{2|a_{12}|} &\leq \frac{1}{2} \\ (a_{11} - a_{22})^2 &\leq a_{12}^2 \end{aligned}$$

These results suggest the following algorithm

Algorithm 1 Non-trigonometric approximation of the Givens angle

function APPROXGIVENS(a_{11}, a_{12}, a_{22}) ▷ Returns (c, s)
  $b \leftarrow [a_{12}^2 < (a_{11} - a_{22})^2]$ ▷ b is boolean
  $\omega \leftarrow 1/\sqrt{a_{12}^2 + (a_{11} - a_{22})^2}$
  $s \leftarrow b?\omega a_{12} : \sqrt{.5}$ ▷ $\sqrt{.5} = \sin(\pi/4)$
  $c \leftarrow b?\omega(a_{11} - a_{22}) : \sqrt{.5}$ ▷ $\sqrt{.5} = \cos(\pi/4)$
  return (c, s)
end function

2.4 Approximate Givens rotation using quaternions

A 3×3 rotation matrix can be equivalently encoded as a quaternion $(a, b, c, d) = (\cos(\theta/2), \sin(\theta/2)\boldsymbol{v})$, where θ is the angle of rotation, and $\boldsymbol{v} = (v_x, v_y, v_z)$ is the normalized axis of rotation. In particular, a 3×3 Givens rotation with $(p, q) = (1, 2)$, i.e. a rotation of the top-leftmost 2×2 submatrix, has the matrix form

$$\begin{pmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

and the equivalent quaternion representation

$$(\cos(\theta/2), 0, 0, \sin(\theta/2)).$$

The added benefit of this representation is that the quaternion does not need to be normalized, i.e. the quaternion $(\gamma\cos(\theta/2), 0, 0, \gamma\sin(\theta/2))$, where $\gamma \in \mathbf{R}$, is just as valid as a representation of this rotation. This suggests that we could mimic the asymptotic approximation $\tan(2\theta) \approx 2\tan\theta$ of the previous section to obtain $\tan(2\theta) \approx 4\tan(\theta/2)$. This suggests the following expression for the approximate Givens angle ϕ:

$$\tan(2\theta) = \frac{2a_{12}}{a_{11} - a_{22}} = 4\tan(\phi/2) \overset{\theta \ll 1}{\approx} 2\tan(\theta). \quad (8)$$

Consequently,

$$\frac{a_{12}}{2(a_{11}-a_{22})} = \frac{\sin\frac{\phi}{2}}{\cos\frac{\phi}{2}} \Rightarrow \begin{cases} \sin(\phi/2) = \omega a_{12} \\ \cos(\phi/2) = 2\omega(a_{11}-a_{22}) \\ \omega = 1/\sqrt{a_{12}^2 + [2(a_{11}-a_{22})]^2} \end{cases} \quad (9)$$

Thus, we can represent this rotation with the quaternion

$$(2\omega(a_{11}-a_{22}), 0, 0, \omega a_{12}). \quad (10)$$

We note that this quaternion representation is equally acceptable and accurate, regardless of the value of the scale factor ω. In fact, even without any scaling at all, the quaternion $(a_{11}-a_{22}, 0, 0, a_{12})$ from a theoretical standpoint would be a perfectly accurate representation of this rotation. The quaternions of subsequent Jacobi iterations can be multiplicatively combined without normalization, and even the conjugation $\mathbf{Q}^T\mathbf{A}\mathbf{Q}$ can be computed using un-normalized quaternions, yielding a result that is identical to using explicit orthogonal matrices, *up to a global scaling of the resulting matrix*. This scaling would need to be corrected just once, at the end of the sequence of Jacobi iterations, by normalizing just once the quaternion that combines all Jacobi rotations, and repeating the conjugation one last time. In our case, we would not even perform this last conjugation, since we do not expect to obtain the matrix $\mathbf{\Sigma}^2$ from the solution of the symmetric eigenproblem, but instead compute $\mathbf{\Sigma}$ directly from the Givens $\mathbf{QR}$ factorization of $\mathbf{AV} = \mathbf{U\Sigma}$ (the product $\mathbf{AV}$ can be computed using the once-normalized quaternion corresponding to matrix $\mathbf{V}$).

In practice, we do have a motive to perform *some* normalization of the Givens quaternion, to avoid the risk of overflow or underflow after a large number of Jacobi iterations. We typically never perform more than 15-20 Jacobi iterations, so even if we scale the quaternion to within, say, a factor of 2 away from a normalized quaternion, any risk of overflow or underflow would be eliminated. We found that a natural (and very inexpensive) way to perform this normalization is to compute the scalefactor ω in equation (9) using the *inexact* Reciprocal-Square-Root function that is built-in and very efficient on most modern processors. For example the SSE `RSQRTPS` instruction yields a relative error of at most 0.1% while having a latency comparable to 2x-3x of a standard packed multiply or add (which is much less than an exact x87 square root, or even a reciprocal computation). For reasons already explained, the accuracy of the symmetric eigenanalysis is in no way affected by the inaccuracy of this operation, and even a higher relative error would not matter, as long as overflow and underflow are averted.

For the purposes of evaluating the magnitude reduction factor of the off-diagonal element, it is appropriate to assume *exact* normalization, since any residual scaling would simply affect the entire matrix and would be corrected once at the end of the process. Once again, we have:

$$b_{12} = \frac{\sin(2\phi - 2\theta)}{\sin(2\theta)} a_{12} \Rightarrow$$
$$\overset{Eq.(8)}{\Longrightarrow} \quad \frac{|b_{12}|}{|a_{12}|} = \left| \frac{\sin(4\arctan(\tan(2\theta)/4) - 2\theta)}{\sin(2\theta)} \right| \quad (11)$$

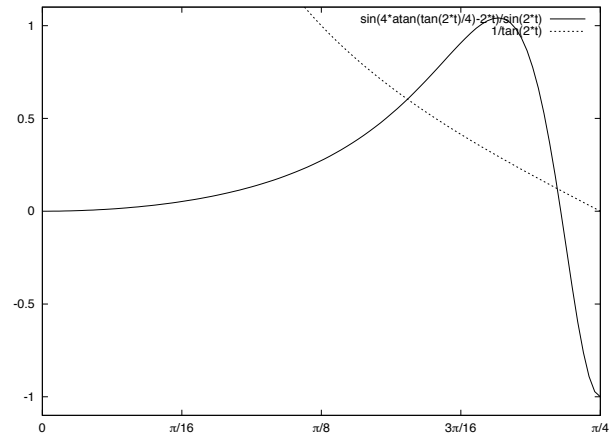

Figure 2: *Approximating the trigonometric Givens factors using* $\tan(2\theta) \approx 4\tan(\theta/2)$ *(or by setting a fixed angle* $\theta = \pi/4$*). The off-diagonal magnitude reduction fraction* $|b_{12}|/|a_{12}|$ *is plotted on the vertical axis, as a function of the optimal Givens angle.*

Figure 2 compares the off-diagonal magnitude reduction fraction obtained by the quaternion approximation of equation 9 with the previously discussed fixed choice of $\phi = \pi/4$. Equating expressions (7) and (11) we obtain that the two curves intersect at $\theta_0 = \arctan(4\tan(\pi/8))/2 \approx 0.51388$ (this is the leftmost intersection in figure 2), while the off-diagonal magnitude reduction

fraction at this point is $\cot(\pi/8)/4 \approx 0.60355$. Therefore, by choosing the best option between the approximate Givens quaternion (10), or the fixed angle $\phi = \pi/4$, we are guaranteed a magnitude reduction of about 60%, slightly less than the approximation of the previous section. Nevertheless, this is strictly the worst-case scenario, and both approximations become much more accurate after just a few Jacobi iterations once the matrix is brought closer to a diagonal form. As in section 2.3 the choice whether to use the asymptotic approximation, or the fixed angle $\phi = \pi/4$ can be made without checking angles or trigonometric quantities; the fixed angle should be used when it achieves a better residual reduction than the maximum value $\cot(\pi/8)/4 \approx 0.60355$:

$$\begin{aligned} \frac{|b_{12}|}{|a_{12}|} = \frac{|a_{11}-a_{22}|}{2|a_{12}|} &\leq \frac{\cot(\pi/8)}{4} = \frac{1+\sqrt{2}}{4} \\ 2|a_{11}-a_{22}| &\leq (1+\sqrt{2})|a_{12}| \\ [2(a_{11}-a_{22})]^2 &\leq (3+2\sqrt{2})a_{12}^2 \end{aligned}$$

The final algorithm becomes:

Algorithm 2 Computation of approximate Givens quaternion.

const $\gamma \leftarrow 3 + 2\sqrt{2}$
const $c_* \leftarrow \cos(\pi/8)$
const $s_* \leftarrow \sin(\pi/8)$
function APPROXGIVENSQUATERNION(a_{11}, a_{12}, a_{22})
  $c_h \leftarrow 2(a_{11}-a_{22})$ ▷ $c_h \approx \cos(\theta/2)$
  $s_h \leftarrow a_{12}$ ▷ $s_h \approx \sin(\theta/2)$
  $b \leftarrow [\gamma s_h^2 < c_h^2]$ ▷ b is boolean
  $\omega \leftarrow \texttt{RSQRT}(c_h^2 + s_h^2)$ ▷ $\texttt{RSQRT}(x) \approx 1/\sqrt{x}$
  $c_h \leftarrow b?\omega c_h : c_*$
  $s_h \leftarrow b?\omega s_h : s_*$
  return $(c_h, 0, 0, s_h)$ ▷ returns a quaternion
end function

Note that Algorithm 2 corresponds to a Jacobi rotation with $(p, q) = (1, 2)$. In order to rotate another pair, the inputs and the ordering of the quaternion elements are adjusted accordingly. We finally address one implementation detail: it may be more efficient (from an implementation standpoint) to compute the elements of the actual rotation matrix $\mathbf{Q}$ before performing the actual conjugation, rather than using the quaternion itself. The (unscaled) corresponding rotation matrix is:

$$\begin{aligned} \mathbf{Q}_{\text{unscaled}} &= \begin{pmatrix} c_h^2-s_h^2 & -2s_hc_h & 0 \\ 2s_hc_h & c_h^2-s_h^2 & 0 \\ 0 & 0 & c_h^2+s_h^2 \end{pmatrix} = \\ &= (c_h^2+s_h^2) \begin{pmatrix} \cos\phi & -\sin\phi & 0 \\ \sin\phi & \cos\phi & 0 \\ 0 & 0 & 1 \end{pmatrix} = (c_h^2+s_h^2)\mathbf{Q} \quad (12) \end{aligned}$$

3 Sorting the singular values

Once the orthogonal factor $\mathbf{V}$ has been computed, we can obtain an expression for the product of $\mathbf{U}$ and $\mathbf{\Sigma}$ as

$$\mathbf{B} := \mathbf{U\Sigma} = \mathbf{U\Sigma V}^T\mathbf{V} = \mathbf{AV}.$$

Note that the last expression is the one actually used to evaluate $\mathbf{B}$. Since $\mathbf{B} = \mathbf{U\Sigma}$, this matrix is simply the result of scaling each column of the orthogonal factor $\mathbf{U}$ with the respective singular value (i.e. the respective diagonal element of $\mathbf{\Sigma}$). Consequently, the magnitude of each singular value in $\mathbf{\Sigma}$ can be computed by simply evaluating the 2-norm of the respective column of $\mathbf{B}$.

We previously stated that our algorithm will be required to produce a diagonal matrix $\mathbf{\Sigma}$ where the singular values along the diagonal are sorted in decreasing order of magnitude. This ordering is not merely an arbitrary convention, but will also benefit the $\mathbf{QR}$ factorization explained later in section 4. We shall enforce this property by reordering the columns of $\mathbf{B}$ in decreasing order of their 2-norm (which will induce the same ordering in the diagonal entries of Σ, as discussed) and also apply the same permutation to the columns of $\mathbf{V}$ at the same time. In order to prove that such a transformation is allowed, consider the individual columns of $\mathbf{B} = [\boldsymbol{b}_1\ \boldsymbol{b}_2\ \boldsymbol{b}_3]$ and $\mathbf{V} = [\boldsymbol{v}_1\ \boldsymbol{v}_2\ \boldsymbol{v}_3]$ respectively. Since $\mathbf{A} = \mathbf{BV}^T$, we have:

$$\mathbf{A} = \sum_{i=1}^{3} \boldsymbol{b}_i \boldsymbol{v}_i^T$$

Thus, if the same permutation is applied to the columns of matrices $\mathbf{B}$ and $\mathbf{V}$, the matrix $\mathbf{A}$ reconstructed as their product remains unaffected. Note that it is also possible to simultaneously *negate* a corresponding pair of columns $\boldsymbol{b}_i$ and $\boldsymbol{v}_i$ without affecting the validity of the decomposition. We can therefore sort the singular values by swapping pairs of columns $(\boldsymbol{b}_i, \boldsymbol{b}_j)$ along with their counterparts $(\boldsymbol{v}_i, \boldsymbol{v}_j)$ in the fashion of a bubblesort method, until the columns of $\mathbf{B}$ appear in decreasing order of their 2-norm. Note that simply swapping two columns of $\mathbf{V}$ will flip the sign of its determinant, violating the property that $\mathbf{V}$ is a true rotation matrix; instead, we also negate one of the two columns being swapped (both for $\mathbf{V}$ and the respective column in $\mathbf{B}$) which will keep $\mathbf{V}$ as a true rotation. The entire process is summarized in the following pseudocode:

Algorithm 3 Singular value sort in decreasing magnitude order

procedure CONDSWAP(c, X, Y) ▷ c is boolean
  $Z \leftarrow X$ ▷ Z is a temporary variable
  $X \leftarrow c?Y : X$
  $Y \leftarrow c?Z : Y$
end procedure
procedure CONDNEGSWAP(c, X, Y) ▷ c is boolean
  $Z \leftarrow -X$ ▷ Z is a temporary variable
  $X \leftarrow c?Y : X$
  $Y \leftarrow c?Z : Y$
end procedure
procedure SORTSINGULARVALUES($\boldsymbol{b}_1, \boldsymbol{b}_2, \boldsymbol{b}_3, \boldsymbol{v}_1, \boldsymbol{v}_2, \boldsymbol{v}_3$)
  $\rho_1 \leftarrow \|\boldsymbol{b}_1\|_2^2,\ \rho_2 \leftarrow \|\boldsymbol{b}_2\|_2^2,\ \rho_3 \leftarrow \|\boldsymbol{b}_3\|_2^2$
  $c \leftarrow [\rho_1 < \rho_2]$ ▷ c is boolean
  CONDNEGSWAP($c, \boldsymbol{b}_1, \boldsymbol{b}_2$); CONDNEGSWAP($c, \boldsymbol{v}_1, \boldsymbol{v}_2$)
  CONDSWAP(c, ρ_1, ρ_2)
  $c \leftarrow [\rho_1 < \rho_3]$
  CONDNEGSWAP($c, \boldsymbol{b}_1, \boldsymbol{b}_3$); CONDNEGSWAP($c, \boldsymbol{v}_1, \boldsymbol{v}_3$)
  CONDSWAP(c, ρ_1, ρ_3)
  $c \leftarrow [\rho_2 < \rho_3]$
  CONDNEGSWAP($c, \boldsymbol{b}_2, \boldsymbol{b}_3$); CONDNEGSWAP($c, \boldsymbol{v}_2, \boldsymbol{v}_3$)
end procedure

Lastly, we recall that in section 2 the rotation matrix $\mathbf{V}$ was in fact constructed as a quaternion $\boldsymbol{q} = (s, x, y, z)$. For the purposes of the current section, we could either convert this representation to an explicit 3×3 matrix, or simply compute the matrix $\mathbf{B} = \mathbf{AV}$ by rotating each row vector of $\mathbf{A}$ with the conjugate quaternion $\bar{\boldsymbol{q}}$. However, if we need to produce $\mathbf{V}$ in quaternion form at the end of the SVD algorithm, it would be inconvenient to convert back and forth between matrix and quaternion representations only so that the previously defined procedure CONDNEGSWAP could be

applied to a matrix representation of $\mathbf{V}$. Fortunately, this operation can also be expressed by a simple quaternion. In particular, note that a function call CONDNEGSWAP($true, \boldsymbol{v}_1, \boldsymbol{v}_2$) is equivalent to replacing $\mathbf{V}$ with $\mathbf{VR}$, where

$$\mathbf{R} = \begin{pmatrix} 0 & -1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

which is a rotation matrix, with a corresponding (un-normalized) quaternion $\boldsymbol{q}_{\mathbf{R}} = (1, 0, 0, 1)$. In the case we want to make the call to CONDNEGSWAP conditional on the variable c, the permutation quaternion is simply $\boldsymbol{q}_{\mathbf{R}} = (1, 0, 0, c)$, assuming that c takes a binary value of either 0 or 1. The quaternion corresponding to the product $\mathbf{VR}$ will then simply be $\boldsymbol{q} \cdot \boldsymbol{q}_{\mathbf{R}}$ (which, notably, requires only 4 additions or subtractions). The same logic can be followed to emulate the action of CONDNEGSWAP on other pairs of columns of $\mathbf{V}$, while operating purely on its quaternion representation.

4 Computation of the factors U and Σ

We previously constructed the matrix $\mathbf{B} = \mathbf{AV}$ and explained that it is equal to the product $\mathbf{U\Sigma}$ of the two remaining unknown components of the SVD. In the last phase of our algorithm we will compute the individual factors $\mathbf{U}$ and $\mathbf{\Sigma}$ from the matrix $\mathbf{B}$.

4.1 Extracting U and Σ via QR decomposition

The matrix $\mathbf{B} = \mathbf{U\Sigma}$ is essentially a column scaling of $\mathbf{U}$ by the respective diagonal entries of $\mathbf{\Sigma}$. Thus, a seemingly straightforward method for computing the orthogonal matrix $\mathbf{U}$ would be to simply rescale each column vector so that it has a unit norm. However, this procedure cannot be used in the case of a zero singular value; moreover, even when a singular value is nonzero yet orders of magnitude smaller than the other values, this normalization may produce a matrix $\mathbf{U}$ that is far from orthogonal. Intuitively, this loss of orthogonality is due in part to the fact that, when a column of $\mathbf{U}$ with very small entries is multiplied with a large number to convert this column to a unit vector, any numerical errors will be greatly amplified. These issues are exacerbated in the case where more than one of the singular values is equal to zero.

Our approach guarantees the orthogonality of the computed matrix $\mathbf{U}$ and is based on the $\mathbf{QR}$ factorization of $\mathbf{B}$ using Givens rotations. We start by showing the following lemma, for a general dimension of the SVD (i.e. potentially larger than the 3×3 case):

Lemma 1. *Let $\mathbf{U}$ be an orthonormal $n \times n$ matrix and $\mathbf{\Sigma}$ a diagonal matrix of the same dimensions. Let $\mathbf{QR} = \mathbf{U\Sigma}$ be the (not necessarily unique) $\mathbf{QR}$-factorization of the product $\mathbf{U\Sigma}$, where $\mathbf{Q}$ is orthogonal and $\mathbf{R}$ is upper triangular.*

If the nonzero diagonal elements of $\mathbf{\Sigma}$ appear before any zero entries (i.e. if $\mathbf{\Sigma}$ has k nonzero entries and $[\Sigma]_{ii} \neq 0, 1 \leq i \leq k$, while $[\Sigma]_{ii} = 0, k+1 \leq i \leq n$), then the following statements hold true:

1. *If $\mathbf{Q} = [\boldsymbol{q}_1\ \boldsymbol{q}_2\ \cdots\ \boldsymbol{q}_n]$, $\mathbf{U} = [\boldsymbol{u}_1\ \boldsymbol{u}_2\ \cdots\ \boldsymbol{u}_n]$, $r_{ij} := [R]_{ij}$, and $\Sigma = diag(\sigma_1, \sigma_2, \ldots, \sigma_n)$, the following statements are true when $i \in [1, k]$:*
   - $\boldsymbol{q}_i = \pm \boldsymbol{u}_i$
   - $r_{ii} = \pm \sigma_i$ *(with the same sign as the identity above)*
   - $r_{ij} = 0$ *for any $j \neq i$.*
2. *The factor $\mathbf{R}$ is in fact $\boldsymbol{diagonal}$.*

Proof.

1. The i-th column of the matrix equation $\mathbf{U\Sigma} = \mathbf{QR}$ is written as follows:
$$\sigma_i \boldsymbol{u}_i = \sum_{k=1}^{i} r_{ki} \boldsymbol{q}_k$$
We will prove the combination of the 3 properties by induction on i.
   - For $i = 1$, we have:
$$\sigma_1 \boldsymbol{u}_1 = r_{11} \boldsymbol{q}_1 \Rightarrow |\sigma_1| \|\boldsymbol{u}_1\|_2 = |r_{11}| \|\boldsymbol{u}_1\|_2 \Rightarrow$$
$$\Rightarrow |\sigma_1| = |r_{11}| \Rightarrow r_{11} = \pm \sigma_1$$
And, from the first equation, we also have $\boldsymbol{q}_1 = \pm \boldsymbol{u}_1$ (with the same sign as in the identity $r_{11} = \pm \sigma_1$). Also, let $j \neq 1$. We have:
$$\begin{aligned} \sigma_j \boldsymbol{u}_j &= \sum_{k=1}^{j} r_{kj} \boldsymbol{q}_k \\ \boldsymbol{u}_1^T (\sigma_j \boldsymbol{u}_j) &= \boldsymbol{u}_1^T \left(\sum_{k=1}^{j} r_{kj} \boldsymbol{q}_k \right) \\ \sigma_j \boldsymbol{u}_1^T \boldsymbol{u}_j &= \sum_{k=1}^{j} r_{kj} \boldsymbol{u}_1^T \boldsymbol{q}_k \end{aligned}$$
Since $j \neq i$, we have $\boldsymbol{u}_1^T \boldsymbol{u}_j$=0. Also, we previously showed that $\boldsymbol{q}_1 = \pm \boldsymbol{u}_1$, thus $\boldsymbol{u}_1^T \boldsymbol{q}_k = \pm \delta_{1k}$ (δ_{ij} is the Kronecker delta). Combining these results with the last equation we get:
$$\begin{aligned} 0 &= \sum_{k=1}^{j} r_{kj} (\pm \delta_{1k}) \\ &= \pm r_{1j} \end{aligned}$$
   - For the induction step $i \rightarrow i+1$ we have:
$$\begin{aligned} \sigma_{i+1} \boldsymbol{u}_{i+1} &= \sum_{k=1}^{i+1} r_{k,i+1} \boldsymbol{q}_k \\ &= \underbrace{\sum_{k=1}^{i} r_{k,i+1} \boldsymbol{q}_k}_{=0 \text{ (induction)}} + r_{i+1,i+1} \boldsymbol{q}_{i+1} \\ &= r_{i+1,i+1} \boldsymbol{q}_{i+1}. \end{aligned}$$
Taking the 2-norm of this equation yields, as before, $r_{i+1,i+1} = \pm \sigma_{i+1}$ and $\boldsymbol{q}_{i+1} = \pm \boldsymbol{u}_{i+1}$. Similarly, for $j \neq i+1$ we have
$$\begin{aligned} \sigma_j \boldsymbol{u}_j &= \sum_{k=1}^{j} r_{kj} \boldsymbol{q}_k \\ \boldsymbol{u}_{i+1}^T (\sigma_j \boldsymbol{u}_j) &= \boldsymbol{u}_{i+1}^T \left(\sum_{k=1}^{j} r_{kj} \boldsymbol{q}_k \right) \\ \sigma_j \boldsymbol{u}_{i+1}^T \boldsymbol{u}_j &= \sum_{k=1}^{j} r_{kj} \boldsymbol{u}_{i+1}^T \boldsymbol{q}_k \\ 0 &= \sum_{k=1}^{j} r_{kj} (\pm \delta_{i+1,k}) \\ &= \pm r_{i+1,j} \end{aligned}$$
which completes our proof.

2. According to the properties proven in Part 1, the matrix $\mathbf{R}$ has the structure

$$\mathbf{R} = \begin{pmatrix} \mathbf{D} & \mathbf{0} \\ \mathbf{0} & \hat{\mathbf{R}} \end{pmatrix}, \text{ where } \mathbf{D} = \begin{pmatrix} \pm\sigma_1 & & \\ & \ddots & \\ & & \pm\sigma_k \end{pmatrix}$$

and $\hat{\mathbf{R}}$ is an upper triangular matrix of size $(n-k)\times(n-k)$. Therefore, the system $\mathbf{U\Sigma} = \mathbf{QR}$ is written as

$$\mathbf{U\Sigma} = \mathbf{Q}\begin{pmatrix} \mathbf{D} & \mathbf{0} \\ \mathbf{0} & \hat{\mathbf{R}} \end{pmatrix}.$$

Since $\sigma_{k+1} = \cdots = \sigma_n = 0$, the last $(n-k)$ columns of this matrix equation are written as:

$$\mathbf{0} = \mathbf{Q}\begin{pmatrix} \mathbf{0} \\ \hat{\mathbf{R}} \end{pmatrix}.$$

The matrix $\mathbf{Q}$ is nonsingular, thus the last equation implies that $\hat{\mathbf{R}} = 0$, suggesting that

$$\mathbf{R} = \begin{pmatrix} \mathbf{D} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} \end{pmatrix}$$

is a purely diagonal matrix.

□

This lemma indicates that the $\mathbf{QR}$ decomposition can be used to factorize $\mathbf{B}$ into an orthogonal matrix (taken as the factor $\mathbf{U}$) and a diagonal matrix which will play the role of Σ. The condition that nonzero singular values need to precede those equal to zero (we achieve this in our case by the sorting process in section 3) is absolutely essential. Consider the counter example of a system $\mathbf{U\Sigma} = \mathbf{QR}$ with the following values :

$$\begin{pmatrix} -.8 & .6 & 0 \\ .6 & .8 & 0 \\ 0 & 0 & 1 \end{pmatrix}\begin{pmatrix} 0 & & \\ & 5 & \\ & & 0 \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}\begin{pmatrix} 0 & 3 & 0 \\ 0 & 4 & 0 \\ 0 & 0 & 0 \end{pmatrix}$$

The factorization on the right is a perfectly valid $\mathbf{QR}$ decomposition, yet $\mathbf{R}$ is neither diagonal, nor does it approximate $\mathbf{\Sigma}$ in any way. By performing a strict sort, rather than a simple separation of zero/nonzero singular values, our methodology is robust to situations where a singular value (respectively, the norm of a column of $\mathbf{B}$) is nonzero, yet much smaller than the magnitude of some other, larger singular value. Finally, we note that this general theory does not guarantee any particular sign for the diagonal elements in the factor $\mathbf{R}$; the convention presented in section 1 will be a consequence of the methodology (Givens rotations) which we employ to compute the $\mathbf{QR}$ decomposition.

4.2 Givens QR factorization

We shall use the method of Givens rotations to compute the $\mathbf{QR}$ factorization, due to the simplicity of its fundamental operations and the fact it guarantees to produce a true rotation matrix $\mathbf{Q}$. In contrast, a Gram-Schmidt procedure would require significant attention to produce a true rotation matrix, especially in the presence of small (or zero) singular values. The Householder scheme would also be an option, albeit one that requires more complex steps, and care needs to be taken due to the fact that it operates by constructing orthogonal *reflections* rather than true rotations.

For a general $n \times n$ matrix $\mathbf{B}$, the method of Givens rotations constructs the triangular factor $\mathbf{R}$ by annihilating the elements below the diagonal one-by-one, in a column-major lexicographical order, i.e. $(2,1),(3,1),\ldots,(n,1),(2,2),(3,2),\ldots,(n,n-1)$. The (i,j) element is annihilated by left-multiplying the result of the previous operations with a Givens matrix $\mathbf{Q}(i,j,\theta_{ij})^T$, as follows:

$$\mathbf{Q}(n,n\text{-}1,\theta_{n,n\text{-}1})^T \cdots \mathbf{Q}(3,1,\theta_{31})^T\mathbf{Q}(2,1,\theta_{21})^T\mathbf{B} = \mathbf{R}$$
$$\Rightarrow \mathbf{Q}^T\mathbf{B} = \mathbf{R} \Rightarrow \mathbf{B} = \mathbf{QR}$$

where $\mathbf{Q} = \mathbf{Q}(2,1,\theta_{21})\mathbf{Q}(3,1,\theta_{31})\cdots\mathbf{Q}(n,n\text{-}1,\theta_{n,n\text{-}1})$.

Due to the specific order in which the elements below the diagonal of $\mathbf{B}$ are being annihilated, every Givens rotation in this sequence will not change any of the zeroes that were introduced by the Givens rotations applied before it. Schematically, when the Givens rotation intended to annihilate element (q,p) is ready to be applied, the following transformation takes place :

$$\mathbf{Q}(p,q,\theta_{pq})^T \begin{pmatrix} a_{11} & \cdots & a_{1q} & a_{1,q+1} & \cdots & a_{1n} \\ & \ddots & \vdots & \vdots & & \vdots \\ & & a_{qq} & a_{q,q+1} & \cdots & a_{qn} \\ & & 0 & a_{q+1,q+1} & \cdots & a_{q+1,n} \\ & & \vdots & \vdots & & \vdots \\ & \mathbf{0} & 0 & a_{p-1,q+1} & \cdots & a_{p-1,n} \\ & & a_{pq} & a_{p,q+1} & \cdots & a_{p,n} \\ & & a_{p+1,q} & a_{p+1,q+1} & \cdots & a_{p+1,n} \\ & & \vdots & \vdots & & \vdots \\ & & a_{nq} & a_{n,q+1} & \cdots & a_{nn} \end{pmatrix} =$$

$$= \begin{pmatrix} a_{11} & \cdots & a_{1q} & a_{1,q+1} & \cdots & a_{1n} \\ & \ddots & \vdots & \vdots & & \vdots \\ & & a'_{qq} & a'_{q,q+1} & \cdots & a'_{qn} \\ & & 0 & a_{q+1,q+1} & \cdots & a_{q+1,n} \\ & & \vdots & \vdots & & \vdots \\ & \mathbf{0} & 0 & a_{p-1,q+1} & \cdots & a_{p-1,n} \\ & & 0 & a'_{p,q+1} & \cdots & a'_{p,n} \\ & & a_{p+1,q} & a_{p+1,q+1} & \cdots & a_{p+1,n} \\ & & \vdots & \vdots & & \vdots \\ & & a_{nq} & a_{n,q+1} & \cdots & a_{nn} \end{pmatrix}.$$

As seen in the last equation, only rows p and q are affected, and only from the q-th column onwards. We can also see that this Givens rotation will succeed in annihilating the element a_{pq} if an only if

$$\begin{pmatrix} \cos\theta_{pq} & \sin\theta_{pq} \\ -\sin\theta_{pq} & \cos\theta_{pq} \end{pmatrix}\begin{pmatrix} a_{qq} \\ a_{pq} \end{pmatrix} = \begin{pmatrix} a'_{qq} \\ 0 \end{pmatrix} \qquad (13)$$

This property can be enforced by simply selecting:

$$\cos\theta_{pq} = \frac{a_{qq}}{\sqrt{a_{qq}^2 + a_{pq}^2}}, \quad \sin\theta_{pq} = \frac{a_{pq}}{\sqrt{a_{qq}^2 + a_{pq}^2}}$$

We also observe that after applying this rotation, the sign of $a'_{qq} = \sqrt{a_{qq}^2 + a_{pq}^2}$ will be non-negative. As a consequence, if the Givens rotations are constructed in this fashion, at the end of the sequence of rotations all diagonal elements of the resulting matrix $\mathbf{R}$, with the exception of the very last one, will be non-negative. This property satisfies the last convention we had adopted in section 1 for the sign of the diagonal elements of $\mathbf{\Sigma}$.

A special case that needs to be addressed occurs when *both* of a_{qq} and a_{pq} are either zero, or extremely small. In this case, the normalization required to obtain the trigonometric factors $\cos\theta_{pq}$ and $\sin\theta_{pq}$ can lead to a division by zero (or significant loss of accuracy, at the very least). We detect this case by checking if $a_{qq}^2 + a_{pq}^2 < \epsilon^2$ for a specified threshold ϵ (in the same order of magnitude as our tolerance for errors in the singular values). When this special case is detected, we set instead:

$$\cos\theta_{pq} = \text{signum}(a_{qq}), \quad \sin\theta_{pq} = 0.$$

These values will still guarantee that $a'_{qq} \geq 0$ and that, ultimately, the first $n-1$ singular values in $\mathbf{\Sigma}$ will be non-negative.

4.3 Quaternion implementation of Givens QR

We conclude by illustrating a methodology that generates the Givens rotations directly in quaternion form; we would utilize this approach if, for the purposes of a given application, it is preferable to compute the rotations $\mathbf{U}$ and $\mathbf{V}$ as quaternions. Although it is certainly possible to convert the 3×3 rotation matrix $\mathbf{U}$ to a quaternion as a post-process, it is preferable to construct the rotations as quaternions in the first place. Doing so will avoid the explicit matrix-to-quaternion conversion, a procedure that needs to consider a number of different cases, and is not optimally structured for aggressive SSE optimizations.

We will describe the methodology in the context of the first matrix $\mathbf{Q}(2,1,\theta_{21})$ from the sequence of Givens rotations used to compute the $\mathbf{QR}$ factorization; the remaining rotations will be constructed in an analogous fashion. The matrix representation of this rotation is :

$$\mathbf{Q}(2,1,\theta) = \begin{pmatrix} \cos\theta & -\sin\theta & \\ \sin\theta & \cos\theta & \\ & & 1 \end{pmatrix}$$

where we dropped the subscripts in the angle θ for simplicity. In order for the operation $\mathbf{Q}(2,1,\theta)^T\mathbf{B}$ to annihilate element b_{21} the following condition must hold, based on equation (13) :

$$-\sin\theta \cdot b_{11} + \cos\theta \cdot b_{21} = 0$$

or, more generally, for the Givens rotation designed to annihilate element b_{pq} we will require :

$$-\sin\theta \cdot a_1 + \cos\theta \cdot a_2 = 0 \tag{14}$$

where a_1 denotes the Pivot element on the diagonal (this is element b_{qq} on the matrix being rotated), and a_2 is the matrix entry to be eliminated (or, b_{pq}).

The same rotation can alternatively be represented by an (unnormalized) quaternion $\boldsymbol{q}$:

$$\boldsymbol{q} = (c_h, 0, 0, s_h) = (\gamma\cos\frac{\theta}{2}, 0, 0, \gamma\sin\frac{\theta}{2})$$

where γ is an arbitrary scaling factor. From equation (14) we get:

$$\frac{a_2}{a_1} = \frac{\sin\theta}{\cos\theta} = \tan\theta = \frac{2\tan\frac{\theta}{2}}{1-\tan^2\frac{\theta}{2}}. \tag{15}$$

Equation (15) is essentially a quadratic equation on $\tan\frac{\theta}{2}$. The two solutions of this quadratic are:

$$\left(\frac{s_h}{c_h} =\right) \quad \tan\frac{\theta}{2} = \frac{-a_1 \pm \sqrt{a_1^2 + a_2^2}}{a_2}. \tag{16}$$

Since the quaternion scale factor γ is irrelevant, we are free to simply choose $c_h = a_2$ and $s_h = -a_a \pm \sqrt{a_1^2 + a_2^2}$ (with either sign). Regardless of the sign chosen in the formula for s_h, in theory both of these values will generate a Givens rotation that successfully annihilates the intended matrix entry. However, we need to pay attention to the following 2 issues:

- One of the 2 choices for s_h may be prone to catastrophic cancellation and loss of accuracy. For example, if $a_1 \gg a_2 > 0$, the operation $-a_1 \pm \sqrt{a_1^2 + a_2^2}$ will lose accuracy, as it is subtracting the finite precision representations of near-identical quantities.
- We need to ensure that after the Givens rotation, the resulting Pivot element a'_{qq} is non-negative, per our convention in section 1.

With a simple case study (which will be omitted here, in the interest of terseness) the best choices for c_h and s_h are determined to be:

- If $a_1 < 0$, then

$$\begin{aligned} c_h &= a_2 \\ s_h &= -a_1 + \sqrt{a_1^2 + a_2^2} \quad \left(= |a_1| + \sqrt{a_1^2 + a_2^2}\right) \end{aligned}$$

- If $a_1 > 0$ then

$$\begin{aligned} c_h &= a_1 + \sqrt{a_1^2 + a_2^2} \quad \left(= |a_1| + \sqrt{a_1^2 + a_2^2}\right) \\ s_h &= a_2 \end{aligned}$$

For the case $a_1 > 0$ it may be initially unclear how these values relate to the solution of equation (15). However, we know that one of the admissible solutions is:

$$\begin{aligned} \frac{s_h}{c_h} &= \frac{-a_1 + \sqrt{a_1^2 + a_2^2}}{a_2} \\ &= \frac{(-a_1 + \sqrt{a_1^2 + a_2^2})(a_1 + \sqrt{a_1^2 + a_2^2})}{a_2(a_1 + \sqrt{a_1^2 + a_2^2})} \\ &= \frac{a_2^2}{a_2(a_1 + \sqrt{a_1^2 + a_2^2})} \\ &= \frac{a_2}{a_1 + \sqrt{a_1^2 + a_2^2}} \end{aligned}$$

from which the formulas for the case $a_1 > 0$ are derived.

Finally, we need to establish that after the constructed Givens rotation has been applied, the value of a'_{qq} will be positive. Noting that $s_h = \gamma\sin(\theta/2)$ and $c_h = \gamma\cos(\theta/2)$, we define:

$$c_* := c_h^2 - s_h^2 = \gamma^2(\cos^2\frac{\theta}{2} - \sin^2\frac{\theta}{2}) = \gamma^2\cos\theta$$

$$s_* := 2s_h c_h^2 = \gamma^2\sin\frac{\theta}{2}\cos\frac{\theta}{2} = \gamma^2\sin\theta$$

Thus, we can obtain the sine and cosine of of the Givens angle theta by normalizing:

$$\cos\theta = \frac{c_*}{\sqrt{c_*^2 + s_*^2}}, \quad \sin\theta = \frac{s_*}{\sqrt{c_*^2 + s_*^2}}$$

With the assistance of these formulas, we can verify that the values chosen for c_h, s_h, either for $a_1 > 0$ or $a_1 < 0$ will ultimately yield (omitting the necessary, yet tedious algebraic reductions):

$$\cos\theta = \frac{a_1}{\sqrt{a_1^2 + a_2^2}}, \quad \sin\theta = \frac{a_2}{\sqrt{a_1^2 + a_2^2}}.$$

As a consequence $a_{qq} = a_1 \cos\theta + a_2 \sin\theta = \sqrt{a_1^2 + a_2^2} \geq 0$. In contrast, some of the roots of equation (15) which were *not* used would have produced $\cos\theta = -a_1/\sqrt{a_1^2 + a_2^2}$, and $\sin\theta = -a_2/\sqrt{a_1^2 + a_2^2}$. These values would have also eliminated element a'_{pq}, but would have produced a *nonpositive* diagonal element a'_{qq} instead. As in the Givens Jacobi procedure of section 2, equation (12) can be used to obtain a (un-normalized) version of the corresponding 3×3 rotation matrix, if such representation of the Givens rotation is desired. The entire procedure is summarized in Algorithm 4; note that the additional checks in lines 3,4 of the pseudocode are designed to safeguard against division by (near-)zero when both elements a_1 and a_2 are extremely small. The threshold value ϵ is set to our tolerance for the magnitude of the elements remaining below the diagonal of $\mathbf{R}$ after the Givens procedure is concluded.

Algorithm 4 Computation Givens quaternion for $\mathbf{QR}$ factorization

1: **function** QRGIVENSQUATERNION(a_1, a_2)
2: $\rho \leftarrow \sqrt{a_1^2 + a_2^2}$
3: $s_h \leftarrow [\rho > \epsilon] ? a_2 : 0$
4: $c_h \leftarrow |a_1| + \max(\rho, \epsilon)$
5: $b \leftarrow [a_1 < 0]$ ▷ b is boolean
6: CONDSWAP(b, s_h, c_h) ▷ CONDSWAP defined in Alg. 3
7: $\omega \leftarrow \texttt{RSQRT}(c_h^2 + s_h^2)$ ▷ $\texttt{RSQRT}(x) \approx 1/\sqrt{x}$
8: $c_h \leftarrow \omega c_h$
9: $s_h \leftarrow \omega s_h$
10: **return** $(c_h, 0, 0, s_h)$ ▷ returns a quaternion
11: **end function**

Note The quaternion representation of the rotational factors $\mathbf{U}$ and $\mathbf{V}$ has limited our need for an exact square root (or reciprocal square root) operation. However, such an exact normalization will be needed at least once, at the end of the SVD algorithm to remove any accumulated scaling. In addition, Algorithm 4 calls for an exact square root in line 2. For these purposes, we found it sufficient to improve the accuracy of `RSQRTPS` by performing one iteration of Newton's method for the equation

$$f(y) = \frac{1}{y^2} - x = 0$$

(the solution of this equation is exactly $1/\sqrt{x}$), as detailed in [Lomont 2003]. The resulting, more accurate versions of the square root function (and its reciprocal) are summarized in pseudocode as follows:

Algorithm 5 Improved accuracy `SQRT` and `RSQRT`

1: **function** ACCURATERSQRT(x)
2: $y \leftarrow \texttt{SQRT}(x)$
3: $y \leftarrow y \cdot (3 - xy^2)/2$
4: **return** y.
5: **end function**
6: **function** ACCURATESQRT(x)
7: **return** $x \cdot$ ACCURATERSQRT(x)
8: **end function**

5 Results and performance

We have implemented and tested a SIMD, multithreaded version of our algorithm, using explicit SSE intrinsics. The following performance measurements were captured on a 12-core/24-thread (hyperthreading enabled) 2.66GHz Intel Xeon X5650 server, using the Intel C++ compiler for Linux, version 12.0.3. We benchmarked our code on the computation of $2^{24} \approx 16.7M$ decompositions of matrices with uniformly random elements, normalized such that the Frobenius norm of each input matrix is equal to one. For the purposes of this benchmark, we fixed the number of Jacobi sweeps (using our approximate, quaternion-based formulation) to a constant number of 4 iterations. Naturally, various degrees of accuracy can be obtained by using a different count of Jacobi iterations; however, for our $16.7M$ uniformly random, unit-normalized matrices, this number of iterations resulted in:

- The maximum magnitude among off-diagonal entries after the symmetric eigenanalysis was 0.004.
- A 99.9% percentile of input matrices achieved a maximum off-diagonal magnitude of less than 0.0005.
- The average maximum off-diagonal magnitude across all input matrices was 3×10^{-6}.

This level of accuracy was deemed well appropriate for the purposes of the accompanying submission [McAdams et al. 2011].

5.1 Performance and scalability

Figure 3 illustrates the total runtime of our SVD algorithm on the sample input of $2^{24} \approx 16.7M$ random matrices (of course, since our algorithm has completely fixed control flow, computation time is input-independent). We generally observed near-linear speedup between single core and 12-core performance. Observed deviations include:

- We observed an additional $\sim 25\%$ performance boost when moving from a 12-core/12-thread to a 12-core/24-thread setup, leveraging the hyperthreading capability of the processor. We attribute this additional acceleration to the hiding of instruction latency of our dense, explicitly vectorized code achieved in the hyperthreading setting.
- Executions with just a single core per socket take advantage of the frequency boost of single-threaded runs, native in the Nehalem architecture.

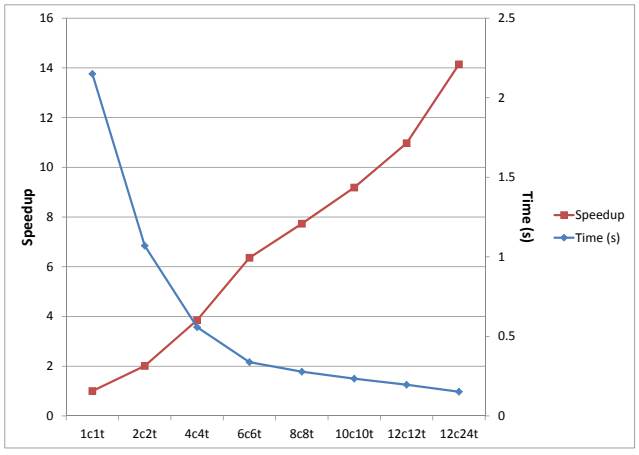

Figure 3: *Execution times, and speedup relative to the single-thread baseline performance of our SVD algorithm, on a dual socket Intel Xeon X5650 server. Benchmark includes a total of 2^{24} decompositions. $McNt$ denotes an M-core, N-thread execution.*

5.2 Comparison with other eigenanalysis methods

Figure 4 provides a comparison between our method, and popular alternatives for solving variants of eigenvalue problems. The methods being compared include:

- Our method, with all the necessary computation overhead required to compute the rotational factors of the SVD in quaternion form. Explicitly SIMD vectorized.
- Our method, with the SVD factors being computed only in matrix (not quaternion) form. (Note that [McAdams et al. 2011] requires a slightly more expensive variant of these two options, requiring both a quaternion representation of the rotation $\mathbf{R} = \mathbf{U}\mathbf{V}^T$, as well as an explicit matrix form of $\mathbf{V}$ itself). Explicitly SIMD vectorized.
- The symmetric eigenanalysis component only of our method. A constant four modified Jacobi sweeps are used. Explicitly SIMD vectorized.
- The symmetric eigenanalysis component only of the Polar Decomposition in [Rivers and James 2007]. No warm starts have been used; the number of Jacobi sweeps is fixed to three, which produces an average accuracy comparable to 4 sweeps of our modified Jacobi procedure. Scalar implementation only (multithreading used without vectorization).
- Computation of eigenvalues of a symmetric 3×3 matrix, using a closed-form solution [Smith 1961]. Scalar implementation only.
- A quaternion-based implementation of the Jacobi procedure for the 3×3 symmetric eigenanalysis (*http://www.melax.com/diag.html?attredirects=0*).

Method	Time per decomposition (ns)	
	1core, 1thread	12core, 24thread
Complete SVD computation with rotations in quaternion form (our method, 4-wide SIMD)	147	9.06
Complete SVD computation with rotations in matrix form (our method, 4-wide SIMD)	121	7.45
Symmetric eigenanalysis only (our method, 4-wide SIMD)	81.4	4.78
Symmetric eigenanalysis only ([Rivers and James 2007], scalar)	460	31.6
Closed-form eigenvalue computation only (scalar)	112	7.03
Computation of diagonalizing quaternion (scalar)	568	34.3

Figure 4: *Comparison of various algorithm for 3×3 eigenanalysis tasks. Single threaded and 12-core/24-thread times are given, normalized to the time required for every individual decomposition. Note that some of the methods may not be directly comparable; refer to the text for a discussion of differences and assumptions.*

It should be noted that these performance numbers cannot be taken as absolute and definitive measures of the superiority of an individual algorithm, since a number of factors have to be considered before accepting these figures as commensurate. Namely:

- Many variants only address the symmetric eigenanalysis problem, instead of the entire SVD (note that for our algorithm we need not only the polar decomposition, but the factor V of the SVD as well). In order to allow for a more fair comparison with these methods, we conducted comparisons with a prefix of our method, that stops when the symmetric eigenanalysis has been computed.
- Instead of relying on published performance figures, we reran the best implementations of these techniques we could find, with the same machine/compiler/optimization settings used for our code. Also, we multithreaded many of these algorithms to give them the same benefit of parallel execution (including the latency-hiding features of hyperthreading, when available).
- For some of these alternative algorithms (or parts thereof) we have reasonable expectation of SIMD potential. When comparing our approach to these methods, one should normalize to the same vector width. Note however that this SIMD potential may often NOT apply to the entire SVD process, but only a fraction of it (e.g. the symmetric eigenanalysis).
- Stopping criteria. Some alternative algorithms iterate until a certain criterion has been satisfied (e.g. the maximum off-diagonal element has been reduced below a certain threshold). Instead, in our approach we chose to implement a *fixed* number of Jacobi sweeps. The reason for this choice is that when using SSE/SIMD the iteration cannot be conveniently stopped for only some out of the decompositions that are packed into an SIMD sequence. We previously explained why our choice of 4 sweeps is a reasonable one.
- Perhaps the most important differentiating factor is the following: When attempting to implement alternative SVD methods as a part of an end-to-end system as [McAdams et al. 2011], we realized that certain alternatives were simply not acceptable for the purposes of specific applications. A simple example is the FastLSM-type decomposition [Rivers and James 2007], which computes the factor $\mathbf{S}$ of $\mathbf{F} = \mathbf{R}\mathbf{S}$ using a Jacobi symmetric eigenanalysis, and then forms $\mathbf{R}$ as $\mathbf{R} = \mathbf{F}\mathbf{S}^{-1}$. The way $\mathbf{S}$ is constructed, it is always a positive definite matrix; thus, in the presence of inversion where often $\det(\mathbf{F}) < 0$, the produced polar decomposition will produce a factor $\mathbf{R}$ that contains a reflection. (i.e. $\det(\mathbf{R}) = -1$). In cases with near-zero singular values, the produced factor $\mathbf{R}$ may severely lack orthogonality as well. See [McAdams et al. 2011] for a further discussion of this issue.

References

GOLUB, G., AND VAN LOAN, C. 1989. *Matrix Computations*. The John Hopkins University Press.

LOMONT, C. 2003. Fast inverse square root. *Purdue University, http://www.lomont.org/Math/Papers/2003/InvSqrt.pdf*.

MCADAMS, A., ZHU, Y., SELLE, A., EMPEY, M., TAMSTORF, R., TERAN, J., AND SIFAKIS, E. 2011. Efficient elasticity for character skinning with contact and collisions. *ACM Trans. Graph.*.

RIVERS, A., AND JAMES, D. 2007. FastLSM: fast lattice shape matching for robust real-time deformation. *ACM Trans. Graph. (SIGGRAPH Proc.) 26*, 3.

SMITH, O. K. 1961. Eigenvalues of a symmetric 3x3 matrix. *Commun. ACM 4* (April), 168–.