Accurate Singular Values and Vectors of an Upper Triangular 2-by-2 Matrix

W. Kahan and J. W. Demmel

Given the elements f, g, h of the real matrix $U = \begin{pmatrix} f & g \\ 0 & h \end{pmatrix}$, we seek its singular values v and w and singular vectors which, when suitably assembled into matrices, will satisfy

$$\begin{pmatrix} c_L & s_L \\ -s_L & c_L \end{pmatrix} \cdot \begin{pmatrix} f & g \\ 0 & h \end{pmatrix} \cdot \begin{pmatrix} c_R & -s_R \\ s_R & c_R \end{pmatrix} = \begin{pmatrix} \pm w & 0 \\ 0 & \pm v \end{pmatrix},$$

$$c_L^2 + s_L^2 = 1, \quad c_R^2 + s_R^2 = 1 \quad \text{and} \quad w \geq v \geq 0.$$

The challenge here is to compute each of v, w, c_L, s_L, c_R and s_R accurately to within a few ulps (Units in its Last Place) unless it deserves to over/underflow beyond the range of the computer's floating-point arithmetic.

The singular values v and w are the nonnegative square roots of the eigenvalues of U^TU ; they are the values of the unobvious expression $| \sqrt{((f+h)^2 + g^2)} \pm \sqrt{((f-h)^2 + g^2)} |/2$, of which the bigger is w and the smaller is $v = |fh|/w$. But computing v and w directly from these expressions is unwise because they can suffer from over/underflow in the squared subexpressions even when v and w are far from the over/underflow thresholds. And explicit formulas for the elements c_L, s_L, c_R and s_R of the singular vectors are fraught with further hazards, as we shall see, that reflect their hypersensitivity to small perturbations when the singular values v and w are nearly coincident. That is why the algorithm developed below is so complicated. The reader who thinks that some simple expedient like scaling would render our complexities unnecessary is invited to consider the situation when the given data f, g and h span almost the full exponent range of his machine. (Our program works if one of f, g or h is ∞ .)

Despite our efforts, we cannot guarantee freedom from anomalies on all computers; some computer arithmetics are too idiosyncratic to be encompassed by a single algorithm that is designed to work well on almost all commercially significant computers. Therefore, our algorithm is designed to be impeccable only on machines that conform to IEEE standard 754 (1985) for floating-point arithmetic. On machines that lack its gradual underflow, but flush underflows to zero instead, accuracy will deteriorate if all the data f, g and h are too close to the underflow threshold, as if the data had been perturbed by amounts comparable with that threshold. On CDC CYBER 1xx machines that suffer from "partial underflow," a test for zero like " If x = 0.0 then ... " must be replaced by a more cumbersome " If 1.0*x = 0.0 then ... " to function correctly. On a CRAY, with its propensity for partial overflow, results that lie between the overflow threshold and half of it may overflow anyway. On some machines with an inaccurate SQRT, its inaccuracy may be amplified surprisingly by our algorithm. Since more machines conform well enough to IEEE 754 than conform to any single other specification, it seems unreasonable to penalize the majority of computers by further complicating our algorithm to allow for idiosyncracies found in only a small minority of today's machines; let the users of those machines allow for them.

Signs and Symmetries and Singular Vectors

The singular values v and w depend upon only |f|, |g| and |h|, but the singular vectors (the c's and s's) depend upon all the signs of the data too. The dependence is a little ambiguous, as indicated by the $\pm$ signs in the defining equation above. Taking determinants on both sides of that equation will reveal both that $fh = \pm wv$ and $|fh| = wv$, so the independent $\pm$ signs in front of w and v are not entirely arbitrary. (see Footnote 1.)

To simplify our algorithm, we shall assume $|f| \geq |h|$. We can impose this condition upon the data by swapping f and h whenever necessary; and then we must also swap c_L with s_R and s_L with c_R at the end, as can be justified by deducing from the defining relation the following equivalent equation:

$$\begin{pmatrix} s_R & c_R \\ -c_R & s_R \end{pmatrix} \cdot \begin{pmatrix} h & g \\ 0 & f \end{pmatrix} \cdot \begin{pmatrix} s_L & -c_L \\ c_L & s_L \end{pmatrix} = \begin{pmatrix} \pm w & 0 \\ 0 & \pm v \end{pmatrix} .$$

Therefore, assume henceforth that $|f| \geq |h|$ and $w \geq v \geq 0$.

We turn now to the determination of the right singular vectors. Except in the special case when $g = 0$ and $|f| = |h| = v = w$, in which case the vectors are indeterminate, the row $(-s_R \quad c_R)$ turns out to be parallel to the rows of $U^TU - w^2I$, and hence to its first row $(f^2-w^2 \quad fg)$. We shall not compute this row lest it over/underflow, but it will serve as a starting point for the derivation of the formula that we shall compute.

After computing a right singular vector we can multiply it by U to obtain a multiple (by a singular value) of the corresponding left singular vector. That is why

$$\begin{pmatrix} c_L & s_L \end{pmatrix} = \begin{pmatrix} fc_R+gs_R & hs_R \end{pmatrix}/(\pm w) ,$$

but we shall not compute it this way lest underflow spoil it; instead this formula will be used as the starting point for the derivation of a better one.

When g is Gargantuan

Consider the special case when the computed value of |g| + |f| rounds to |g|.

What does this mean? The expected meaning is that, to within the limits upon accuracy imposed by roundoff, |f| must be negligible compared with |g|. Of course, another possibility is that both |f| and |g| are zero. Unfortunately another perverse possibility arises on computers that round either towards infinity or by von Neumann's "jamming". *Rounding toward infinity* is one of the *directed rounding* modes afforded by the IEEE standard 754, but it is not the default mode and therefore would be in force only by accident; therefore we shall ignore it. Von Neumann's *jamming* sets the last bit retained to 1 whenever any subsequent nonzero bits are discarded; this kind of rounding must be extremely rare nowadays. In either case, the perverse possibility is that |f| must be zero whenever |g|+|f| rounds to |g|; this is far too perverse to be worth any further consideration. Therefore let ε be the biggest positive number such that |g| + |f| will round to |g| whenever $|f| \leq \varepsilon|g|$. Typically ε is comparable with a rounding error in numbers close to 1.0. Note however that the precise determination of ε could depend delicately upon how the computer rounds sums; and care must be taken not to confuse

two directions of implication:
if $|f| \le \varepsilon|g|$ then $|g| + |f|$ rounds to $|g|$, but
if $|g| + |f|$ rounds to $|f|$ then $|f| \le \beta\varepsilon|g|$, where
β = the arithmetic radix (typically 2, 8, 10 or 16).
Fortunately, only for the proof of our algorithm's correctness must ε be known; it does not have to appear in our program.

A complication arises on machines that evaluate subexpressions in registers with *extended* precision, more than has been stored in the variables f and g. On such machines, the relevant value of ε pertains to extended precision rather than the precision of the variables. This complicates the analysis of our algorithm but does not invalidate it.

Finally, we ignore the possibility that $|g| + |f|$ may overflow, because its value will figure only in a comparison with $|g|$. On a machine that conforms to IEEE 754, the overflow would go to ∞ and the subsequent comparison would discard it after drawing the correct conclusion. On a machine that is stopped by overflow, or on a machine that forces overflows to the biggest finite number, a statement like " if $|g| + |f| = |g|$ then ... " must be replaced by a statement like " if $|f| \le \varepsilon|g|$ then ... " with a value ε chosen to suit the machine as described above.

If $g = f = h = 0$, then $v = w = 0$ too and the singular vectors are indeterminate; setting $c_L = c_R = 1$ and $s_L = s_R = 0$ will satisfy the defining relations well enough, so this is just what we shall do whenever $g = 0$.

At last we come to the case when $g \neq 0$ and is so gargantuan compared with $|f|$ that $|f|/|g| \le \beta\varepsilon$. In this case $w = |g|$ and $v = |fh|/w$ with negligible relative errors smaller than $(\beta\varepsilon)^2$; but the last formula is too vulnerable to spurious over/-underflow, so it has to be re-evaluated carefully thus:
if $|h| > 1$ then $v = |f|/(w/|h|)$ else $v = (|f|/w)|h|$ endif.
Similarly simple formulas suffice in this case to approximate the singular vectors too: $c_R = f/g$, $s_R = c_L = 1$ and $s_L = h/g$.

Now that the case of gargantuan g has been settled, we shall assume henceforth that g is not gargantuan, and consequently that $|g/f| < 1/\varepsilon$.

The Normal Cases
Here is a summary of the hypotheses in force now:
$$|h| \le |f| \quad \text{and} \quad |g/f| < 1/\varepsilon .$$
Now it is safe to compute certain intermediate quantities; first
$$\lambda = (|f| - |h|)/|f| \quad \text{and} \quad \mu = g/f .$$
They satisfy $0 \le \lambda \le 1$ and $|\mu| \le 1/\varepsilon$. Underflow in μ is harmless if handled in the usual way, either gradually or flushed to zero. Otherwise the accuracy of λ and μ is crucial to the accuracy of the singular vectors, especially when both are tiny. On a machine that conforms to IEEE 754 or 854 that accuracy is assured; but extremely rare bad things can happen on some other machines which we digress to discuss now.

On machines that flush underflows to zero, when f is very tiny underflow of $|f| - |h|$ can ruin the accuracy of λ; detect it

by observing when $\lambda = 0$ though $|f| > |h|$, and correct it by scaling. Cancellation in $|f| - |h|$ can ruin its accuracy only on machines that lack a guard digit for subtraction; among such machines are the CDC Cybers, CRAYs, UNIVAC 11xx's and a few others that mimic their arithmetics. On these machines λ might as well be computed from the formula $\lambda = 1 - |h/f|$ to avoid trouble with underflow to zero; then inaccuracy when λ is tiny will be no worse than if f had first been perturbed by an ulp or two. On machines like the CRAY whose division is not atomic, division by f is actually accomplished by multiplying by $1/f$, which can overflow if f is tiny enough; on the CDC Cyber 1xx series, division by f can signal DIVISION BY ZERO when f is extremely tiny though not zero; solve both problems by computing the singular values of U+U and then halving them.

Next compute σ and ρ and α thus:

$\sigma = \sqrt{(2-\lambda)^2 + \mu^2}$;
if $\lambda = 0$ then $\rho = |\mu|$ else $\rho = \sqrt{\lambda^2 + \mu^2}$ endif ;
$\alpha = (\sigma + \rho)/2$.

Nothing bad can happen to them because none exceeds $1+1/\varepsilon$, and $\sigma \geq \alpha \geq 1$. And if $\lambda \neq 0$ then $\lambda \geq \varepsilon$, so μ^2 can underflow if it must without causing harm. Now we can compute the singular values

$$v = |h|/\alpha \quad \text{and} \quad w = |f|\alpha$$

knowing that they will not over/underflow unless they deserve to (except perhaps on a CRAY or a CDC Cyber).

Now it is time to compute the right singular vectors. Recall that $(-s_R \;\; c_R)$ is parallel to $(f^2-w^2 \;\; fg)$; but to avoid trouble from cancellation or underflow we shall divide this by fg before computing it to get, after some algebra, (see Footnote 2.)

$$\tau = 2\,s_R/c_R = (\alpha+1)\left(\mu/(\sigma+2-\lambda) + \mu/(\rho+\lambda)\right) ,$$

from which it follows that $2 \leq \tau/\mu < 2+2/|\mu|$. However, if μ^2 has underflowed we must avoid inaccuracy in $\mu/(\rho+\lambda)$ by computing it directly from the data or, if $\lambda = 0$ too, by setting τ to its limit as $\mu \rightarrow 0$, namely CopySign(2,μ) with IEEE 754/854, Sign(2,g) Sign(1,f) without. Then compute $c_R = 2/\sqrt{(\tau^2 + 4)}$ and $s_R = \tau/\sqrt{(\tau^2 + 4)}$.

Similar reasoning produces a left singular vector:

$$c_L = (c_R + s_R\mu)/\alpha \quad \text{and} \quad s_L = (h/f)s_R/\alpha .$$

Finally, if we swapped f with h at the start, we must remember to swap c_L with s_R and s_L with c_R .

Our Program

Considering how complicated it was to figure out, our program is surprisingly short. It is presented here in a syntax like that of Fortran 77, but with two innovations. One is the invocation of an intrinsic procedure SWAP(x, y) that swaps the values of its arguments. On a machine that contains a SWAP instruction in its hardware, this should be preferable to the three MOVES that would be needed instead. The second innovation is the use of three consecutive dots (...) to introduce a comment at the end of a line rather than have to add a line beginning with "C" for every short annotation.

```
      SUBROUTINE SVD2x2( f, g, h,   cL, sL,  w, v,  cR, sR )
c     Accurate singular value decomposition of a given  2x2 real
c     matrix:   ( cL  sL ).( f  g ).( cR -sR )  =  ( +/- w   0 )
c               (-sL  cL ) ( 0  h ) ( sR  cR )     ( 0   +/- v )
c     with  cL*cL+sL*sL = cR*cR+sR*sR = 1  and  w .GE. v .GE 0 .
      REAL  f, g, h,      cL, sL, w, v, cR, sR
c        -- Input --    ------- Output -------  Aliasing is OK.
c     w and v  are the singular values;  the c's and s's  define
c     the singular vectors of the given matrix.  In the special
c     case  g = 0 ,  we get  cL = cR = 1  and  sL = sR = 0 .  In
c     the special case  h = 0 ,  we get  cL = 1  and  sL = 0 .
      LOGICAL L
      REAL  ft,gt,ht, cLt,sLt, cRt,sRt     ... Copied and scratch values
      REAL  fa,ga,ha                       ... may be kept in registers
      REAL  α, δ, λ, μ, μμ, ρ, σ, τ        ... to improve speed & accuracy.
      REAL                              Zero, Half, One, Two, Four
      DATA  Zero,Half,One,Two,Four / 0.0,  0.5, 1.0, 2.0, 4.0 /
      ft = f
      fa = ABS(ft)
      ht = h
      ha = ABS(ht)
      L = ( ha .GT. fa )
      IF (L)  THEN
                  SWAP( ft, ht )
                  SWAP( fa, ha )
              ENDIF                        ... now fa ≥ ha .
c
      gt = g
      ga = ABS(gt)
      IF ( ga .EQ. Zero )  THEN
              v = ha                       ... the trivial case.
              w = fa
              cLt = One
              cRt = One
              sLt = Zero
              sRt = Zero
c             - - - - - - - - - - - - - - - -
          ELSE IF ( ga+fa .EQ. ga )  THEN
              w = ga                       ... the case of gargantuan g .
              IF ( ha .GT. One )  THEN
                                    v = fa/(ga/ha)
                                ELSE
                                    v = (fa/ga)*ha
                                ENDIF
              cLt = One
              sLt = ht/gt
              cRt = ft/gt
              sRt = One
c             - - - - - - - - - - - - - - - -
          ELSE
              δ = fa - ha                  ... the normal cases.
              IF (δ .EQ. fa) THEN
                                    λ = One  ... copes with infinite f or h .
                             ELSE
                                    λ = δ/fa
                             ENDIF           ... 0 ≤ λ ≤ 1 .
              μ = gt/ft                      ... |μ| ≤ 1/ε .
              τ = Two - λ                    ... τ ≥ 1 .
```

```
                  μμ = μ*μ
                  σ = SQRT( τ*τ + μμ )          ... 1 ≤ σ ( 1+1/ε .
                  IF (λ .EQ. Zero)  THEN
                                       ρ = ABS(μ)
                                    ELSE
                                       ρ = SQRT( λ*λ + μμ )
                                    ENDIF
                  α = Half*(σ + ρ)              ... 1 ≤ α ( 1 + |ε| .
                  v = ha/α
                  w = fa*α
                  IF ( μμ .EQ. Zero )  THEN     ... ε must be very tiny.
                       IF (λ .EQ. Zero) THEN    ... with IEEE 754/854
                                       τ = CopySign(Two, μ)
c                           i. e.,     τ = SIGN(Two,ft)*SIGN(One,gt)
                                    ELSE
                                       τ = gt/SIGN(δ,f) + μ/τ
                                    ENDIF
                     ELSE
                       τ = ( μ/(σ+τ) + μ/(ρ+λ) )*(One+α)
                     ENDIF                      ... see Footnote 2.
                  λ = SQRT( τ*τ + Four )
                  cRt = Two/λ
                  sRt = τ/λ
                  cLt = ( cRt + sRt*μ )/α
                  sLt = ( ht/ft )*sRt/α
               ENDIF
c              - - - - - - - - - - - - - - - - - - -
          IF (L)  THEN
                       SWAP( cLt, sRt )
                       SWAP( sLt, cRt )
                  ENDIF
          cL = cLt
          sL = sLt
          cR = cRt
          sR = sRt
          RETURN
c         Cost: 15 Add/Subtract/Compares, 9 Multiplies, 10 Divides, 3 SQRTs
          END.
c         = = =   End of  SVD2x2  = = =   W. Kahan    April 27, 1988
```

Footnote 1.
An alternative definition of v and w hides the ambiguity of sign thus:

$$\begin{pmatrix} c_L & s_L \\ -s_L & c_L \end{pmatrix}\begin{pmatrix} f & g \\ 0 & h \end{pmatrix}\begin{pmatrix} c_R & -s_R \\ s_R & c_R \end{pmatrix} = \begin{pmatrix} w & 0 \\ 0 & v \end{pmatrix} ;$$

the singular values are now $|v|$ and $|w|$ ordered so $|v| \le |w|$. To conform to this redefinition, change our program by replacing the letter "a" by "t" in every one of the six statements that say either " v = ... " or " w = ... " ; for instance replace "w = fa*α" by "w = ft*α", "v = fa/(ga/ha)" by "v = ft/(gt/ht)".

Footnote 2.
The alternative formula $\lambda = 1 - |h/f|$ ought to ensure $\lambda \ge 0$; but since nobody knows for sure whether this must hold on a CRAY, evaluating $\mu/(\rho+\lambda)$ on it could conceivably hit DIVIDE-BY-ZERO!