Computing the Singular Values of 2-by-2 Complex Matrices

Sanzheng Qiao and Xiaohong Wang

Department of Computing and Software
McMaster University
Hamilton, Ontario L8S 4L7

May 2002

Abstract

This paper describes an algorithm for the singular value decomposition of a 2-by-2 complex matrix. It computes accurate singular values.

Keywords Singular value decomposition (SVD), Jacobi method.

1 Introduction

In this note we describe an accurate algorithm for singular values of a 2-by-2 complex matrix

$$B = U\Sigma V^{\mathrm{H}}. \tag{1}$$

The singular value decomposition (SVD) of a 2-by-2 complex matrix does not occur as frequently as that of a 2-by-2 real upper triangular matrix. The commonly used QR method for SVD consists of two stages. In the first stage, a matrix is reduced to bidiagonal form using, say, Householder transformations on both sides. In the second stage, the bidiagonal matrix resulted from the first stage is diagonalized using QR iterations, where we deal with the SVDs of 2-by-2 blocks. Since the first stage, we can assume the 2-by-2 blocks real and upper triangular. However, in Jacobi methods, which are suitable for parallel computing, we have to deal with the SVD of 2-by-2 complex matrix if the original matrix is complex.

Our algorithm consists of two stages. The first stage reduces B to real and upper triangular:

$$B = U_1 R V_1^{\mathrm{H}} \quad \text{where} \quad R = \begin{bmatrix} f & g \\ 0 & h \end{bmatrix} \tag{2}$$

and U_1 and V_1 are unitary. The second stage computes the SVD

$$R = U_2 \Sigma V_2^{\mathrm{T}}, \tag{3}$$

where U_2 and V_2 are orthogonal, using the algorithm by Demmel and Kahan [2]. Then, from (2) and (3), the SVD (1) can be obtained by $U = U_1U_2$ and $V = V_1V_2$.

Demmel and Kahan's algorithm computes singular values of a real and upper triangular 2-by-2 matrix accurate to almost machine precision and is efficient, however, no details were given in [2]. In [1], Bai and Demmel briefly described the algorithm and listed its FORTRAN code SLASV2. In this note, we explain SLASV2 and give a C code.

2 Triangularization

To triangularize $B = [b_{ij}]$, we first find a unitary rotation

$$Q_1 = \begin{bmatrix} c & -\bar{s} \\ s & \bar{c} \end{bmatrix} \quad \text{where} \quad |c|^2 + |s|^2 = 1$$

such that

$$Q_1^{\mathrm{H}} B = \begin{bmatrix} f & z_1 \\ 0 & z_2 \end{bmatrix}.$$

We may assume that f is real. Next, we make the two entries z_1 and z_2 real:

$$Q_2^{\mathrm{H}} \begin{bmatrix} f & z_1 \\ 0 & z_2 \end{bmatrix} V_1 = \begin{bmatrix} f & g \\ 0 & h \end{bmatrix},$$

where

$$Q_2 = \begin{bmatrix} 1 & 0 \\ 0 & \bar{z}_1 z_2/|z_1 z_2| \end{bmatrix} \quad \text{and} \quad V_1 = \begin{bmatrix} 1 & 0 \\ 0 & \bar{z}_1/|z_1| \end{bmatrix}.$$

We can see that $g = |z_1|$ and $h = |z_2|$ are real.

3 Demmel and Kahan's Algorithm

Let

$$A \equiv \begin{bmatrix} a_{11} & a_{12} \\ a_{12} & a_{22} \end{bmatrix} = RR^{\mathrm{T}} = \begin{bmatrix} f^2 + g^2 & gh \\ gh & h^2 \end{bmatrix}. \tag{4}$$

Since (3), $A = U_2 \Sigma^2 U_2^{\mathrm{T}}$ or $AU_2 = U_2 \Sigma^2$. Let $U_2 = [u_1 \; u_2]$ and $\Sigma = \mathrm{diag}(\sigma_1, \sigma_2)$, then

$$\begin{bmatrix} a_{11} & a_{12} \\ a_{12} & a_{22} \end{bmatrix} u_i = \sigma_i^2 u_i, \qquad i = 1, 2.$$

That is

$$\begin{bmatrix} a_{11} - \sigma_i^2 & a_{12} \\ a_{12} & a_{22} - \sigma_i^2 \end{bmatrix} u_i = 0.$$

It then follows that

$$\det \begin{bmatrix} a_{11} - \sigma_i^2 & a_{12} \\ a_{12} & a_{22} - \sigma_i^2 \end{bmatrix} = 0.$$

Equivalently, σ_i^2 are the zeros of the quadratic polynomial

$$\lambda^2 - (a_{11} + a_{22})\lambda + a_{11}a_{22} - a_{12}^2.$$

Thus σ_i are the square roots of

$$\frac{(a_{11} + a_{22}) \pm \sqrt{(a_{11} + a_{22})^2 - 4a_{11}a_{22} + 4a_{12}^2}}{2}. \tag{5}$$

Direct use of the above expression may cause unnecessary under/overflow and catastrophic cancellation. It is reorganized as follows to avoid those numerical problems. Expression (5) equals

$$\frac{4(a_{11}a_{12} - a_{12}^2)}{2\left((a_{11} + a_{22}) \mp \sqrt{(a_{11} - a_{22})^2 + 4a_{12}^2}\right)}.$$

From (4), substituting a_{11}, a_{12}, and a_{22} with f^2+g^2, gh, and h^2 respectively, we get

$$\frac{4((f^2 + g^2)h^2 - (gh)^2)}{2\left(f^2 + g^2 + h^2 \mp \sqrt{(f^2 + g^2 - h^2)^2 + 4(gh)^2}\right)}.$$

The numerator can be reduced to $4(fh)^2$, which is a perfect square. For the denominator, it can be verified that

$$2(f^2 + g^2 + h^2) = (f + h)^2 + g^2 + (f - h)^2 + g^2,$$

and

$$\begin{aligned}
& (f^2 + g^2 - h^2)^2 + 4(gh)^2 \\
= \; & (f^2 - h^2)^2 + g^4 + 2(f^2 - h^2)g^2 + 4(gh)^2 \\
= \; & (f^2 - h^2)^2 + g^4 + 2(f^2 + h^2)g^2 \\
= \; & (f + h)^2(f - h)^2 + g^4 + ((f + h)^2 + (f - h)^2)g^2 \\
= \; & ((f + h)^2 + g^2)((f - h)^2 + g^2).
\end{aligned}$$

So, $2(f^2+g^2+h^2) \mp \sqrt{(f^2+g^2-h^2)^2+4(gh)^2)}$ is also a perfect square $\left(\sqrt{(f+h)^2+g^2} \mp \sqrt{(f-h)^2+g^2}\right)^2$. Thus the singular values

$$\sigma_i = \frac{2|fh|}{\left|\sqrt{(f+h)^2+g^2} \mp \sqrt{(f-h)^2+g^2}\right|}.$$

The smaller singular value is given by

$$\sigma_2 = \frac{2|fh|}{\sqrt{(f+h)^2+g^2} + \sqrt{(f-h)^2+g^2}},$$

and the larger one is

$$\sigma_1 = |fh|/\sigma_2.$$

Now we consider the right singular vectors v_i $(i=1,2)$ in $V_2 = [v_1\ v_2]$. From (3),

$$R^{\mathrm{T}}R = [v_1\ v_2] \begin{bmatrix} \sigma_1^2 & 0 \\ 0 & \sigma_2^2 \end{bmatrix} [v_1\ v_2]^{\mathrm{T}},$$

we have

$$\begin{bmatrix} f^2-\sigma_1^2 & fg \\ fg & g^2+h^2-\sigma_1^2 \end{bmatrix} v_1 = 0.$$

This shows that the vector

$$x = \begin{bmatrix} f^2-\sigma_1^2 \\ fg \end{bmatrix}$$

is orthogonal to v_1. It then follows that v_2 is parallel to x. Let

$$V_2 = \begin{bmatrix} c_v & -s_v \\ s_v & c_v \end{bmatrix} \qquad c_v^2+s_v^2=1,$$

then $[s_v\ c_v]$ is parallel to $[-f^2+\sigma_1^2\ fg]$. Let

$$U_2 = \begin{bmatrix} c_u & -s_u \\ s_u & c_u \end{bmatrix} \qquad c_u^2+s_u^2=1,$$

then from (3) we have

$$\begin{bmatrix} f & g \\ 0 & h \end{bmatrix} \begin{bmatrix} c_v \\ s_v \end{bmatrix} = \sigma_1 \begin{bmatrix} c_u \\ s_u \end{bmatrix},$$

which leads to

$$c_u = (f c_v + g s_v)/\sigma_1 \quad \text{and} \quad s_u = h s_v/\sigma_1.$$

This completes our explanation.

Finally, we list our C code translated from SLASV2 in [1].

```
#include <stdlib.h>
#include <math.h>

#define EPS 2.1e-16                     // double precsion

#define isign(i) ((i)<0 ? (-1) : (+1))  // int sign function
#define sign(x) ((x)<0.0 ? (-1) : (+1)) // float sign function

//------------------------------------------------------------------------
// Lasv2
//      SVD of a 2x2 real upper triangular matrix
//         [f  g] = [cu -su]*[smax   0  ]*[ cv sv]
//         [0  h]   [su  cu] [ 0    smin] [-sv cv]
//      smax is the larger singular value and smin is the smaller
//
//      "smin" and "smax" are singular values
//      "cv" and "sv" are the entries in the right singular vector matrix
//      "cu" and "su" are the entries in the left singular vector matrix
//      "f", "g", and "h" are the entries in the upper triangular matrix
//
// This code is translated from the FORTRAN code SLASV2 listed in
// Z.Bai and J.Demmel,
// "Computing the Generalized Singular Value Decomposition",
// SIAM J. Sci. Comput., Vol. 14, No. 6, pp. 1464-1486, November 1993
//-------------------------------------------------------------------------
void
Lasv2(double *smin, double *smax, double *sv, double *cv,
      double *su, double *cu, double f, double g, double h)
{
    double svt, cvt, sut, cut;          // temporary sv, cv, su, and cu
    double ft = f, gt = g, ht = h;      // temporary f, g, h
    double fa = fabs(f), ga = fabs(g), ha = fabs(h);
                                        // |f|, |g|, and |h|
    int pmax = 1,                       // pointer to max abs entry
```

```
    swap = 0,                         // is swapped
    glarge = 0,                       // is g very large
    tsign;                            // tmp sign
double fmh,                           // |f| - |h|
       d,                             // (|f| - |h|)/|f|
       dd,                            // d*d
       q,                             // g/f
       qq,                            // q*q
       s,                             // (|f| + |h|)/|f|
       ss,                           // s*s
       spq,                           // sqrt(ss + qq)
       dpq,                           // sqrt(dd + qq)
       a;                             // (spq + dpq)/2
double tmp,                           // temporaries
       tt;

// make fa>=ha
if (fa<ha) {
    pmax = 3;
    tmp = ft; ft = ht; ht = tmp;      // swap ft and ht
    tmp = fa; fa = ha; ha = tmp;      // swap fa and ha
    swap = 1;
} // if fa<ha

if (ga==0.0) { // diagonal
    *smin = ha;
    *smax = fa;
    cut = 1.0; sut = 0.0;             // identity
    cvt = 1.0; svt = 0.0;
} else { // not diagonal
    if (ga>fa) { // g is the largest entry
        pmax = 2;
        if ((fa/ga)<EPS) {            // g is very large
            glarge = 1;
            *smax = ga;               // 1 ulp
            if (ha>1.0)
                *smin = fa/(ga/ha);  // 2 ulps
            else
                *smin = (fa/ga)*ha;  // 2 ulps
            cut = 1.0; sut = ht/gt;
```

```
            cvt = 1.0; svt = ft/gt;
        } // if g large
    } // if ga>fa
    if (glarge==0) {                // normal case
        fmh = fa - ha;              // 1 ulp
        if (fmh==fa)                // cope with infinite f or h
            d = 1.0;
        else
            d = fmh/fa;             // note 0<=d<=1.0, 2 ulps
        q = gt/ft;                  // note |q|<1/EPS, 1 ulp
        s = 2.0 - d;                // note s>=1.0, 3 ulps
        qq = q*q; ss = s*s;
        spq = sqrt(ss + qq);        // note 1<=spq<=1+1/EPS, 5 ulps
        if (d==0.0)
            dpq = fabs(q);          // 0 ulp
        else
            dpq = sqrt(d*d + qq);   // note 0<=dpq<=1+1/EPS, 3.5 ulps
        a = 0.5*(spq + dpq);        // note 1<=a<=1 + |q|, 6 ulps
        *smin = ha/a;               // 7 ulps
        *smax = fa*a;               // 7 ulps

        if (qq==0.0) {              // qq underflow
            if (d==0.0)
                tmp = sign(ft)*2*sign(gt);
                                    // 0 ulp
            else
                tmp = gt/(sign(ft)*fmh) + q/s;
                                    // 6 ulps
        } else {
            tmp = (q/(spq + s) + q/(dpq + d))*(1.0 + a);
                                    // 17 ulps
        } // if qq
        tt = sqrt(tmp*tmp + 4.0);
                                    // 18.5 ulps
        cvt = 2.0/tt;               // 19.5 ulps
        svt = tmp/tt;               // 36.5 ulps
        cut = (cvt + svt*q)/a;      // 46.5 ulps
        sut = (ht/ft)*svt/a;        // 45.5 ulps
    } // if g not large
} // if ga
```

```
    if (swap==1) {
        *cu = svt; *su = cvt;
        *cv = sut; *sv = cut;
    } else {
        *cu = cut; *su = sut;
        *cv = cvt; *sv = svt;
    } // if swap

    // correct the signs of smax and smin
    if (pmax==1) tsign = sign(*cv)*sign(*cu)*sign(f);
    if (pmax==2) tsign = sign(*sv)*sign(*cu)*sign(g);
    if (pmax==3) tsign = sign(*sv)*sign(*su)*sign(h);
    *smax = isign(tsign)*(*smax);
    *smin = isign(tsign*sign(f)*sign(h))*(*smin);
} // Lasv2
```

References

[1] Zhaojun Bai and James W. Demmel, Computing the generalized singular value decomposition, *SIAM J. Sci. Comput.*, Vol. 14, No. 6, pp. 1464–1486, November 1993.

[2] James Demmel and W. Kahan, Accurate singular values of bidiagonal matrices, *SIAM J. Sci. Stat. Comput.*, Vol. 11, No. 5, pp. 873–912, September 1990.