Discontinuous Plane Rotations and the Symmetric Eigenvalue Problem

Edward Anderson
Lockheed Martin Services Inc.
Anderson.Edward@epa.gov

Elementary plane rotations are one of the building blocks of numerical linear algebra and are employed in reducing matrices to condensed form for eigenvalue computations and during the QR algorithm. Unfortunately, their implementation in standard packages such as EISPACK, the BLAS and LAPACK lack the continuity of their mathematical formulation, which makes results from software that use them sensitive to perturbations. Test cases illustrating this problem will be presented, and reparations to the standard software proposed.

1.0 Introduction

Unitary transformations are frequently used in numerical linear algebra software to reduce dense matrices to bidiagonal, tridiagonal, or Hessenberg form as a preliminary step towards finding the eigenvalues or singular values. Elementary plane rotation matrices (also called Givens rotations) are used to selectively reduce values to zero, such as when reducing a band matrix to tridiagonal form, or during the QR algorithm when attempting to deflate a tridiagonal matrix. The mathematical properties of rotations were studied by Wilkinson [9] and are generally not questioned.

However, Wilkinson based his analysis on an equation for computing a rotation matrix that is different from those in common use in the BLAS [7] and LAPACK [2]. While the classical formula is continuous in the angle of rotation, the BLAS and LAPACK algorithms have lines of discontinuity that can lead to abrupt changes in the direction of computed eigenvectors. This makes comparing results difficult and proving backward stability impossible. It is easy to restore continuity in the generation of Givens rotations even while scaling to avoid overflow and underflow, and we show how in this report.

Model implementations of the software described in this report have been written and tested with the LAPACK test programs on CRAY PVP and CRAY T3E systems, and are freely available [1]. Current users of LAPACK may observe a rescaling of some of their eigenvectors (usually by -1) with the revised software.

2.0 Givens rotations -- theory and practice

In the real case, a Givens rotation is a rank-2 correction to the identity of the form[1]

$$G(i,j,\theta) = \begin{bmatrix} 1 & & & & & & & & & & \\ & \cdots & & & & & & & & & \\ & & 1 & & & & & & & & \\ & & & c & & & & s & & & \\ & & & & 1 & & & & & & \\ & & & & & \cdots & & & & & \\ & & & & & & 1 & & & & \\ & & & -s & & & & c & & & \\ & & & & & & & & 1 & & \\ & & & & & & & & & \cdots & \\ & & & & & & & & & & 1 \end{bmatrix}$$

where $c = \cos(\theta)$ and $s = \sin(\theta)$ for some angle θ. Premultiplication of a vector x by $G(i,j,\theta)$ amounts to a clockwise rotation of θ radians in the (i, j) plane. If $y = G(i, j, \theta)\, x$, the vector y can be described by

$$y_k = \begin{cases} cx_i + sx_j, & k = i \\ -sx_i + cx_j, & k = j \\ x_k, & k \neq i, j \end{cases}$$

We can force y_j to be zero by choosing θ to be the angle described by the vector $\begin{bmatrix} x_i & x_j \end{bmatrix}^T$ in the (i, j) plane, which leads to the formulas

$$c = \frac{x_i}{\sqrt{x_i^2 + x_j^2}}$$

$$s = \frac{x_j}{\sqrt{x_i^2 + x_j^2}}$$

These are the formulas used by Wilkinson for his error analysis of plane rotations. Note that $r = y_i = \sqrt{x_i^2 + x_j^2}$ is always positive by this definition. In practice, r, like any square root of a sum of squares, must be computed with scaling to avoid underflow when both x_i and x_j are less than the square root of underflow, or when one of x_i or x_j is greater than the square root of overflow.

---

1. This notation is consistent with LAPACK, as well as Wilkinson [9] or Higham [6]. Some other authors define Givens rotations using the transpose of G.

Since a Givens rotation only modifies two elements of a vector, its action can be described by the 2-by-2 linear transformation

$$\begin{bmatrix} c & s \\ -s & c \end{bmatrix} \begin{bmatrix} f \\ g \end{bmatrix} = \begin{bmatrix} r \\ 0 \end{bmatrix}, \qquad c^2 + s^2 = 1 \tag{2.1}$$

Stewart [8] observed that the values of *c* and *s* could be reconstructed from a single stored number *z* if a sign were introduced in the value of *r*, an idea used in the Level 1 BLAS subroutine SROTG [7]. Given *f* and *g*, SROTG computes

$$\sigma = \begin{cases} \operatorname{sgn}(f), & |f| > |g| \\ \operatorname{sgn}(g), & |f| \le |g| \end{cases}$$

$$r = \sigma\sqrt{f^2 + g^2}$$

$$c = \begin{cases} f/r, & r \ne 0 \\ 1, & r = 0 \end{cases}$$

$$s = \begin{cases} g/r, & r \ne 0 \\ 0, & r = 0 \end{cases}$$

The value of *z* from Stewart's compact representation is also computed and returned by SROTG, but it is not of interest here.

In LAPACK, the auxiliary routine SLARTG is used instead of SROTG when generating a Givens rotation. The Cray Scientific Library (libsci) implementation of SLARTG was based on an early LAPACK release and is identical to SROTG in its computation of *c*, *s*, and *r*.

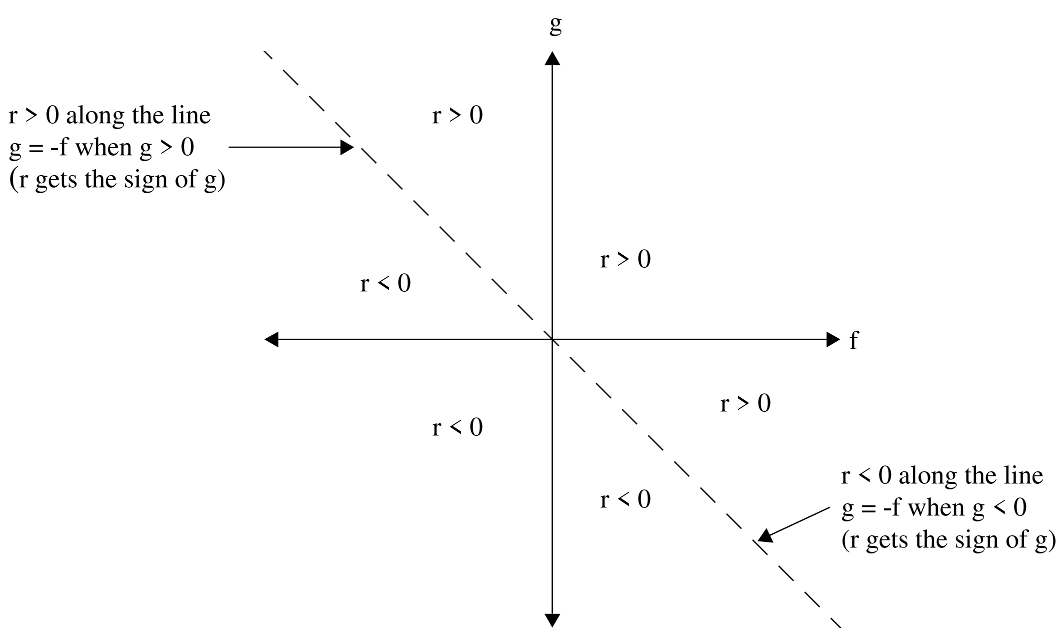
Algorithm 1 (BLAS-like SLARTG): Given f and g, compute c, s, and r satisfying (2.1).

if ($g = 0$) then
    $c = 1;\ s = 0;\ r = f$
else if ($f = 0$) then
    $c = 0;\ s = 1;\ r = g$
else if ($|f| > |g|$) then
    $t = g/f$
    $u = \sqrt{1 + t^2}$
    $c = 1/u$
    $s = t{*}c$
    $r = f{*}u$
else
    $t = f/g$
    $u = \sqrt{1 + t^2}$
    $s = 1/u$
    $c = t{*}s$
    $r = g{*}u$
end if

Note that the value of the scalar u is between 1 and $\sqrt{2}$, which avoids underflow or overflow in the computation of the sum of squares. There is still a slight risk of overflow when computing r as the product of f or g times u if f or g is scaled near overflow, but all algorithms for computing Givens rotations run this risk.

Consider what happens to the value of r in Algorithm 1. When $|f| > |g|$, r takes the sign of f, and when $|f| \le |g|$, r takes the sign of g. The magnitude of r is always $\sqrt{f^2 + g^2}$. In quadrants of the f-g plane where f and g have the same sign, r is continuous, but r changes sign when crossing the line $g = -f$. This line of discontinuity is illustrated in Figure 1. Similarly discontinuities are found in c and s if plotted as functions of f and g.

FIGURE 1. Value of r as a function of f and g from libsci (BLAS-like) SLARTG



In the LAPACK version of SLARTG (from the most recent revision, LAPACK 3), a different algorithm is used which results in r being more often positive than negative. This version is complicated by some iterative scaling to avoid overflow or destructive underflow in the computation of $\sqrt{f^2 + g^2}$, which we will not show. The rest of the algorithm is outlined in Algorithm 2.

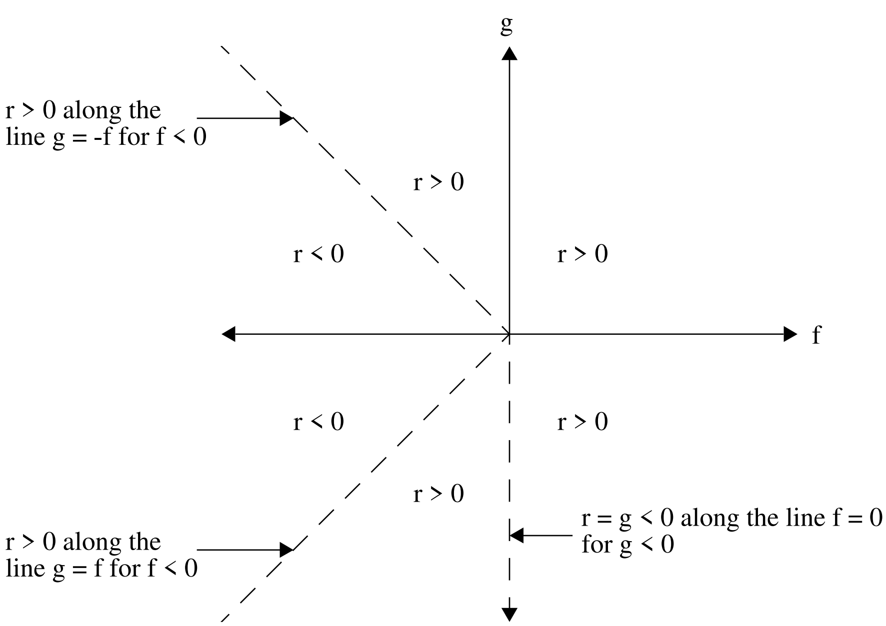
Algorithm 2 (LAPACK SLARTG):

```
if (g = 0) then
        c = 1;  s = 0;  r = f
else if (f = 0) then
        c = 0;  s = 1;  r = g
else
```

$f_1 = f$
$g_1 = g$
{scale f_1 and g_1 to avoid overflow/underflow}
$r = \sqrt{f_1^2 + g_1^2}$
$c = f_1 / r$
$s = g_1 / r$
if ($|f| > |g|$ AND $f < 0$) then
    $c = -c$
    $s = -s$
    $r = -r$
end if
end if

In this version, r is positive except when $|f| > |g|$ and $f < 0$, which consists of one-fourth of the $f-g$ plane bounded by $g = -f$ and $g = f$ for $g < 0$, and when $f = 0$ and $g \le 0$. The negative g axis is thus a line of discontinuity, as are the boundaries of the region where $r < 0$, as shown in Figure 2. This is a peculiar form of plane rotation and its differences from the standard BLAS may have been inadvertent, particularly since other uses of plane rotations in LAPACK, such as the subroutine SLARGV to generate multiple plane rotations, follow the BLAS convention.

FIGURE 2. Value of r as a function of f and g from LAPACK 3.0 SLARTG



Still another algorithm is used by Bindel *et al.* to generate Givens rotations in their justification of the proposed new BLAS standard [4]. Despite the authors' stated requirement that "the mapping from *(f,g)* to *(c,s,r)* should be continuous whenever possible", their proposed algorithm carries over the discontinuity of its predecessors. The mathematical description of this proposed version, which ignores the scaling of $\sqrt{f^2+g^2}$, is reproduced in Algorithm 3. The model implementation in [4] implements the scaling of f and g in an iterative fashion as in the LAPACK SLARTG.

Algorithm 3 (BLAS Technical Forum version):

if ($g = 0$) then
  $c = 1;\ s = 0;\ r = f$
else if ($f = 0$) then
  $c = 0;\ s = \operatorname{sgn}(g);\ r = |g|$
else

$$c = \frac{|f|}{\sqrt{f^2+g^2}}$$

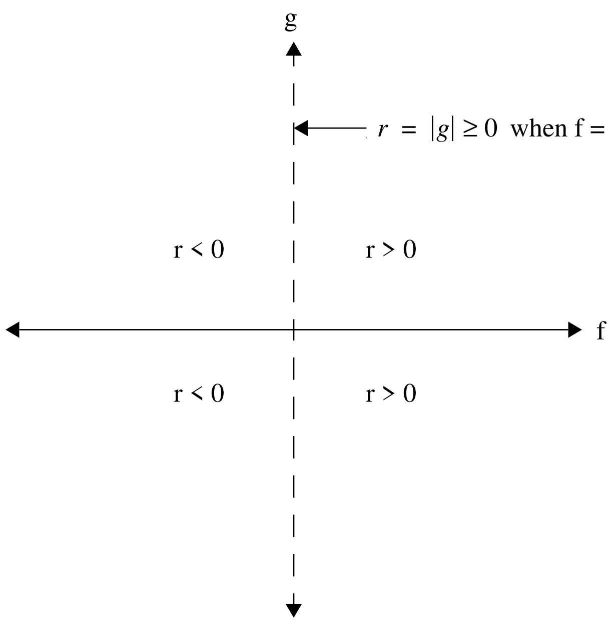
$$s = \frac{\operatorname{sgn}(f)g}{\sqrt{f^2+g^2}}$$

$$r = \operatorname{sgn}(f)\sqrt{f^2+g^2}$$

end if

This version eliminates $g = f$ and $g = -f$ as lines of discontinuity, but it creates a new line of discontinuity on the g axis, as shown in Figure 3.

FIGURE 3. Value of r as a function of f and g from BLAS Technical Forum SLARTG



3.0 Effects of discontinuity

In each of Algorithms 1, 2, and 3, there were lines of discontinuity in the *f*-*g* plane, along which the sign of the computed value of *r* (as well as the signs of *c* and *s*) could be sensitive to perturbations in the values of *f* or *g*. A change of sign of *c*, *s*, and *r* in the computed Givens rotation matrix is equivalent to a multiplication by the negative of the 2-by-2 identity matrix:

$$\begin{bmatrix} -c & -s \\ s & -c \end{bmatrix} = \begin{bmatrix} -1 & 0 \\ 0 & -1 \end{bmatrix} \begin{bmatrix} c & s \\ -s & c \end{bmatrix}$$

In the symmetric eigenvalue problem, Givens rotations are applied to both sides of a symmetric tridiagonal matrix during the QR algorithm. Suppose that a matrix *A* is tridiagonal with a bulge in *A*(3, 1) (and, by symmetry, *A*(1, 3)) after application of the first step of the QR algorithm. The bulge-chasing step generates a rotation matrix to annihilate *A*(3, 1) using *A*(2, 1), that is, $f = A(2, 1)$ and $g = A(3, 1)$ in the previous algorithms. It is easy to see that if the first rotation matrix of the bulge-chasing step returns the negative of c and s, then this sign change will propagate down the matrix, negating all the offdiagonal elements and all the bulges before they are annihilated in turn. The sign of the diagonal elements of the tridiagonal matrix will be the same whether *c* and *s* or $-c$ and $-s$ are used. Since both the offdiagonal element $A(i+1, i)$ and the bulge it must annihilate at $A(i+2, i)$ are negated, the same rotations will be generated at each step except perhaps for the sign of *c* and *s*.

The tests for convergence of the QR algorithm are similarly impervious to the sign of the offdiagonal elements. In IMTQL2 from EISPACK, the test to determine if offdiagonal elements are small enough for the matrix to be split was

$$|E(i)| \leq \varepsilon(|D(i-1)| + |D(i)|)$$

where $E(i) = A(i+1, i)$ and $D(i) = A(i, i)$, while in SSTEQR from LAPACK, the convergence test is

$$|E(i)| \leq \varepsilon\sqrt{|D(i-1)|}\sqrt{|D(i)|}$$

Since both of these tests depend only on the magnitudes of the offdiagonal elements, the eigenvalues are generally computed identically regardless of which algorithm is used to generate the rotation matrices.[2]

2. There are some differences in the computed eigenvalues in the repeated eigenvalue case that can be traced to the algorithm for computing Givens rotations. In this case the differences arise from the sensitivity of the computation $r = \sqrt{f^2 + g^2}$, which is scaled differently in Algorithms 1 and 2. There is no statistical basis for preferring one scaling method over the other.

However, the eigenvectors do depend on the sign of the generated Givens rotations because they are computed by multiplying the saved rotations together. In fact, every variation in the QR algorithm, from the convergence criteria to the order of calculations, can lead to a different sequence of rotations and slightly different computed eigenvectors. Due to the lack of uniformity in algorithms for generating Givens rotations, the most common difference between the eigenvectors of two competing methods is in the sign of the nonzero elements of the vectors. This is to be expected, but users of numerical software packages are still sometimes surprised by it. We will illustrate this difficulty with an example run on a CRAY C90 comparing driver routines for the symmetric eigenvalue problem from five different packages:

- netlib EISPACK -- the version of EISPACK found at www.netlib.org
- libsci EISPACK -- the version of EISPACK from the Cray Scientific Library
- netlib LAPACK -- the latest netlib version of LAPACK (LAPACK 3.0)
- libsci LAPACK -- the version of LAPACK from the Cray Scientific Library
- new LAPACK -- libsci with my LAPACK 3.0 supplement

In EISPACK the symmetric eigenvalue driver routine is called RS and in LAPACK it is called SSYEV. Differences between netlib LAPACK and libsci are described in [3]; the LAPACK 3.0 supplement to libsci is described in [1].

Example 1: Sign changes due to the choice of library

Let $A = \begin{bmatrix} 1 & 2 & 3 \\ 2 & 3 & 4 \\ 4 & 5 & 6 \end{bmatrix}$. Using the "new LAPACK" algorithm, we compute

$$\Lambda = diag(-1.51, -5.74\times 10^{-2}, 11.6), \qquad X = \begin{bmatrix} 0.683 & -0.620 & -0.386 \\ 0.386 & 0.755 & -0.531 \\ -0.621 & -0.213 & -0.754 \end{bmatrix}$$

All five packages compute the same set of well-separated eigenvalues, but the scaling of the columns of the matrix of eigenvectors is different in every case, as shown in Table 1:

Table 1 Scaling of columns of X from Example 1

Method	Eigenvectors		
netlib RS	x1	x2	x3
libsci RS	x1	−x2	x3
netlib SSYEV	x1	−x2	−x3
libsci SSYEV	−x1	−x2	x3
new SSYEV	x1	x2	x3

A separate problem -- one that is at least partly correctable -- is the variability of the signs of components of the computed eigenvectors within the same algorithm due to perturbations in the input data. Any implementation of the QR algorithm using a discontinuous algorithm for generating Givens rotations (unfortunately including all the standard libraries) is sensitive to this type of perturbation. To illustrate, we consider another example, for which we use only the netlib LAPACK version of SSYEV on a CRAY C90:

Example 2: Sign changes due to perturbations

Let $A = \begin{bmatrix} 1 & -1 & 0 \\ -1 & 1 & -1 \\ 0 & -1 & 1 \end{bmatrix}$. Using the netlib LAPACK version of SSYEV, we compute

$$\Lambda = diag(-0.414, 1.00, 2.41), \qquad X = \begin{bmatrix} 0.500 & -0.707 & -0.500 \\ 0.707 & 0.000 & 0.707 \\ 0.500 & 0.707 & -0.500 \end{bmatrix}$$

Now we will introduce perturbations on the order of $\delta = 0.0001$ into selected entries of the matrix *A* and observe the effect on the sign of the computed eigenvectors while still using the netlib LAPACK version of SSYEV. The eigenvalues and the magnitudes of the entries of the eigenvectors remain approximately the same under the perturbation. Since our test code stores only the lower triangle of the symmetric matrix A to interface with the LAPACK routines, we indicate only the element of the lower triangle that was perturbed in each test, but in effect the corresponding element of the upper triangle was perturbed as well. The sign changes, shown in Table 2, are due solely to the discontinuity in the algorithm for computing Givens rotations.

Table 2 Scaling of columns of *X* from Example 2

Perturbation	Eigenvectors		
A(2,1) + δ	x1	x2	x3
A(3,2) + δ	−x1	−x2	x3
A(2,1) - δ	−x1	−x2	x3
A(3,2) - δ	x1	x2	x3

4.0 Continuous algorithms for generating plane rotations

In order to restore continuity to the algorithm for generating Givens rotations, we must bring together the formulas for *r* when $|f| > |g|$ and $|f| \le |g|$. There are different ways this can be done, but the most natural is to return to the original formulas found in Wilkinson [9]:

$$\begin{bmatrix} c & s \\ -s & c \end{bmatrix} \begin{bmatrix} f \\ g \end{bmatrix} = \begin{bmatrix} r \\ 0 \end{bmatrix}, \qquad c^2 + s^2 = 1$$

$$\begin{aligned} r &= \sqrt{f^2 + g^2} \\ c &= f / r \\ s &= g / r \end{aligned}$$

With these formulas, r is always positive, and c and s are just the cosine and sine of the angle between the positive f axis and the line from the origin to the point (f, g) in the f-g plane. Of course, we still need to scale the sum of squares in the computation of r. Doing so leads to Algorithm 4.

Algorithm 4: A continuous algorithm for generating real plane rotations

```
if (g = 0) then
    c = sgn(f)
    s = 0
    r = |f|
else if (f = 0) then
    c = 0
    s = sgn(g)
    r = |g|
else if (|f| > |g|) then
    t = g / f
    u = sgn(f) sqrt(1 + t^2)
    c = 1 / u
    s = t*c
    r = f*u
else
    t = f / g
    u = sgn(g) sqrt(1 + t^2)
    s = 1 / u
    c = t*s
    r = g*u
end if
```

In the most general form of a complex plane rotation, we have a choice of making c real, s real, or r real. There is a certain elegance to making r real because then the formulas are very much like those of the real plane rotation. However, for performance reasons it is much more beneficial to choose either c or s to be real because this reduces the cost of applying the rotation to a larger matrix. In LAPACK, c is chosen to be real, and we would like the algorithm that generates a complex plane rotation to compute c, s, and r identically to the real case if f and g are real. This requirement leads to the following set of constraints for the complex plane rotation, where c is real and f, g, s, and r are complex. We include a definition of sgn(x) that generalizes the sign function to complex numbers.

$$\begin{bmatrix} c & s \\ -\bar{s} & c \end{bmatrix} \begin{bmatrix} f \\ g \end{bmatrix} = \begin{bmatrix} r \\ 0 \end{bmatrix}, \qquad c^2 + s \cdot \bar{s} = 1$$

$$\operatorname{sgn}(x) \equiv \begin{cases} x/|x|, & x \neq 0 \\ 1, & x = 0 \end{cases}$$

$$r = \operatorname{sgn}(Re(f))\operatorname{sgn}(f)\sqrt{|f|^2 + |g|^2}$$

$$c = f/r = \operatorname{sgn}(Re(f))\frac{|f|}{\sqrt{|f|^2 + |g|^2}}$$

$$s = \bar{g}/\bar{r} = \operatorname{sgn}(Re(f))\frac{\operatorname{sgn}(f) \cdot \bar{g}}{\sqrt{|f|^2 + |g|^2}}$$

The leading sign term in each of r, c, and s could be either positive or negative and still satisfy the constraint equation. We have made it the sign of the real part of f in order to force r to be positive when f and g are both real, for consistency with the real algorithm. A straightforward implementation of these equations with some basic scaling of the sum of squares is shown in Algorithm 5.

Algorithm 5: A continuous algorithm for generating complex plane rotations

if ($g = 0$) then
  $c = \operatorname{sgn}(f)$
  $s = 0$
  $r = f*c$
else if ($f = 0$) then
  $c = 0$
  $s = \operatorname{sgn}(\bar{g})$
  $r = |g|$
else
  $f_1 = |Re(f)| + |Im(f)|$
  $g_1 = |Re(g)| + |Im(g)|$
  if ($f_1 \geq g_1$) then
    $f_s = f/f_1$
    $f_2 = Re(f_s)^2 + Im(f_s)^2$
    $g_s = g/f_1$
    $g_2 = Re(g_s)^2 + Im(g_s)^2$
    $u = \operatorname{sgn}(Re(f))\sqrt{1 + g_2/f_2}$
    $c = 1/u$
    $s = \bar{g}_s f_s (c/f_2)$
    $r = f*u$
  else
    $f_s = f/g_1$
    $f_2 = Re(f_s)^2 + Im(f_s)^2$

$$g_s = g / g_1$$
$$g_2 = Re(g_s)^2 + Im(g_s)^2$$
$$u = \operatorname{sgn}(Re(f)) \cdot g_1 \sqrt{f_2 + g_2}$$
$$f_1 = |f|$$
$$f_s = f / f_1$$
$$c = f_1 / u$$
$$s = f_s(\bar{g}/u)$$
$$r = f_s * u$$

end if

end if

There is slightly more risk of overflow in this algorithm than in the real case because the pseudo-norm $f_1 = |Re(f)| + |Im(f)|$ could overflow if the real and imaginary parts were both greater than half the overflow threshold. A more complete scaling strategy is described in Bindel *et al.* [4].

5.0 A note about Jacobi rotations

Jacobi rotations [5] have the same form as Givens rotations but the method used to generate them must satisfy a different set of constraints. In a Jacobi eigenvalue method we seek a cosine-sine pair (c,s) that diagonalizes a symmetric 2-by-2 input matrix, that is,

$$\begin{bmatrix} B_{11} & 0 \\ 0 & B_{22} \end{bmatrix} = \begin{bmatrix} c & s \\ -s & c \end{bmatrix}^T \begin{bmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{bmatrix} \begin{bmatrix} c & s \\ -s & c \end{bmatrix}$$

which implies

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

If $A_{12} = 0$, then the matrix is already diagonal and we can set $(c, s) = (1, 0)$. Otherwise we solve a quadratic equation in the tangent $t = \tan(\theta)$ by setting

$$\tau = \frac{A_{22} - A_{11}}{2A_{12}}, \qquad t = s/c$$

Then

$$t^2 + 2\tau t - 1 = 0$$

which has as its roots $t = -\tau \pm \sqrt{1 + \tau^2}$. Choosing t to be the smaller of these two roots minimizes the difference between B and A and ensures that $|\theta| \le \frac{\pi}{4}$, which is important for the convergence of the iterative Jacobi algorithm. Once t has been determined in this way, the formulas for c and s follow from trigonometry:

$$c = 1/\sqrt{1+t^2}$$
$$s = tc$$

These equations for c and s are the same as those of Algorithm 1 when $|f| > |g|$. However, they are not the same as the equations of Algorithm 4, our recommended alternative, and practitioners who require Jacobi rotations should be aware of this difference.

6.0 Conclusions

In this paper, we have presented continuous algorithms for generating a real or complex plane rotation, one of the building blocks of numerical linear algebra. Previous implementations in the BLAS, EISPACK, and LAPACK have failed to be continuous. An algorithm that is not continuous is not backward stable, and hence its output can be sensitive to perturbations in the input values. We have observed this sensitivity as abrupt sign changes in the computed eigenvectors when solving the symmetric eigenvalue problem, even when the eigenvalues are well-separated.

It is surprising that so many people have quoted Wilkinson's proof of the backward stability of Givens plane rotations, which assumes continuity, while overlooking the discontinuity present in almost every implementation. In hindsight, one can say that more rigorous testing of the backward stability of this basic building block might have detected its shortcomings sooner. No lesser authorities than Wilkinson and Fox suggested this approach in their 1975 forward to the NAG Fortran Library Manual:

> "With regard to sensitivity and accuracy [the NAG Fortran Library] achieves rather less, but this is a problem so far not well treated even by numerical analysts. Information is provided in a fairly economical way for the solution of linear equations, in which the so-called "iterative refinement" involving a little double precision arithmetic gives valuable information on the sensitivity and a more accurate answer when this is meaningful. For many other problems, the user can only obtain this sort of information by his own efforts, for example, by deliberately introducing small perturbations and observing their effects on his solutions. This whole area is one in which one hopes for continual improvements in the library routines when better ways to implement them are discovered."[3]

Acknowledgments

I am grateful to Jack Dongarra of the University of Tennessee and Oak Ridge National Laboratory for allowing me to retain a web page on LAPACK issues there, even though the current maintainers of LAPACK don't necessarily agree with all my suggestions.

3. Professor L. Fox and Dr. J. H. Wilkinson, Foreword to the NAG Fortran Library Manual, 1975. Reprinted in NAG Fortran Library Introductory Guide, Mark 19, 1999.

References

[1] Edward Anderson, *Installing LAPACK 3 on CRAY Machines,* online technical report, Sept. 2000. (`http://www.cs.utk.edu/~eanderso/lapack3.html`)

[2] E. Anderson, Z. Bai, C. Bischof, S. Blackford, J. Demmel, J. Dongarra, J. Du Croz, A. Greenbaum, S. Hammarling, A. McKenney, and D. Sorensen, *LAPACK Users' Guide, Third Edition*, SIAM, Philadelphia, 1999.

[3] Edward Anderson and Mark Fahey, *Performance Improvements to LAPACK for the Cray Scientific Library*, Technical Report CS-97-359, University of Tennessee, April 1997. Also available as LAPACK Working Note 126, `http://www.netlib.org/lapack/lawns/lawn126.ps`.

[4] D. Bindel, J. Demmel, W. Kahan, and O. Marques, *On computing Givens rotations reliably and efficiently*, Technical Report CS-00-448, University of Tennessee, October 2000. Also available as LAPACK Working Note 148, `http://www.netlib.org/lapack/lawns/lawn148.ps`.

[5] Gene H. Golub and Charles F. Van Loan, *Matrix Computations, 3rd edition*, Johns Hopkins University Press, Baltimore, 1996.

[6] Nicholas J. Higham, *Accuracy and Stability of Numerical Algorithms*, SIAM, Philadelphia, 1996.

[7] C. L. Lawson, R. J. Hanson, D. R. Kincaid, and F. T. Krogh, *Basic Linear Algebra Subprograms for Fortran Usage*, ACM Trans. Math. Soft. 5, 3 (Sept. 1979), 308-323.

[8] G. W. Stewart, *The economical storage of plane rotations*, Numer. Math. 25, 2 (1976), 137-139.

[9] J. H. Wilkinson, *The Algebraic Eigenvalue Problem*, Clarendon Press, 1965.