Software for Roundoff Analysis

WEBB MILLER

The Pennsylvania State University

Fortran programs for locating numerical instabilities in algebraic processes are given. They easily diagnose known instabilities in certain versions of the QR algorithm and the Gram-Schmidt method.

To analyze a given numerical algorithm we proceed as follows. A number which measures the effect of roundoff error is assigned to each set of data. "Hill-climbing" procedures are then applied to search for values large enough to signal instability.

Key Words and Phrases: automatic roundoff analysis, numerical stability, QR algorithm, Gram-Schmidt orthogonalization, streamlined polynomial forms
CR Categories: 5.10, 5.11, 5.14

1. INTRODUCTION

Among the dangers faced by the designer of mathematical software is the specter of a product which is numerically unstable. The major weapons currently at the designer's disposal are (i) testing the product upon sets of data for which a solution can be found by alternative means and (ii) formal roundoff analysis. Each of these has a defect. The first is sometimes insufficient, and the second requires an expenditure of effort that cannot always be afforded.

There is convincing evidence that weapons (i) and (ii) need to be augmented where possible. A square-root-free QR algorithm for computing the eigenvalues of a symmetric tridiagonal matrix was published by Ortega and Kaiser [17] in 1963. The algorithm appears (in slightly modified form) in Wilkinson's treatise *The Algebraic Eigenvalue Problem* [25, p. 567], surely a definitive work on rounding errors. An Algol 60 version was published in the *Communications of the ACM* [4]. Apparently the algorithm was subjected to extensive testing.

However, it is numerically unstable [9, 23]. The error, once committed, propagates persistently. The algorithm is advocated by more than one numerical analysis text published since the announcement of its instability. It is impossible to judge the cost of the oversight.

Perhaps the most unfortunate aspect of this lapse is the ease with which it could have been avoided. Discovery of the instability is almost automatic given (1) an

This work was supported in part by the National Science Foundation under NSF Grant GJ-42968.
Author's address: Computer Science Department, The Pennsylvania State University, University Park, PA 16802.

appropriate general approach to rounding errors, (2) a few simple Fortran programs, (3) a minimal understanding of the eigenvalue problem, (4) a statement of the Ortega-Kaiser algorithm, and (5) a few hours of rather routine work.

The goal of this paper is to supply ingredients (1) and (2) (all five are combined in Section 8). The basic idea is to use the computer to search for a set of data for which rounding errors cause a given numerical method to produce inaccurate results. First, a number $\rho(\mathbf{d})$ is assigned to each set $\mathbf{d}$ of data in such a way that large values of ρ correspond to $\mathbf{d}$ for which the effect of rounding error is excessive. In addition, the function ρ should be "smooth" and "easy to evaluate." The second step is to apply a numerical "hill-climbing" routine to ρ.

Proper understanding of this technique requires the realization that a number of idealizations and concessions have been made. In Section 2 we will lay this groundwork. The basic approach is outlined in Sections 3 and 4, with actual programs given in Section 5. Sections 7 through 10 contain case studies designed to illustrate the scope of these methods. The reader interested in more rigorous justification of our techniques is invited to consult Miller [13, 15].

2. THE GOAL; THE PRICE

Some of the most striking successes in the field of roundoff analysis have come in the understanding of error propagation in algebraic processes such as Gaussian elimination, orthonormalization, and the QR algorithm (see Wilkinson [24–26]). Such analyses are typically very tedious, though not conceptually difficult. This suggests that the computer be employed to do at least part of the work.

To delineate the current status of automatic roundoff analysis we need to draw the distinction between *local* and *global* techniques. We do this to make explicit a major difference between our approach, which is "global," and previous efforts, virtually all of which are "local."

By *local* computer error analysis we mean use of the computer to bound (or to estimate) the error incurred in *a single computation.* The basic idea is to use special systems of computer arithmetic (e.g. interval arithmetic [16] or unnormalized arithmetic) to monitor the error in each computed value. In general these schemes are designed to provide the user with information about the total error (including, for example, the effects of data uncertainty) in his results.

By *global* roundoff analysis we mean the determination of how rounding error propagates in a given numerical method *for many or all permissible sets of data.* Perhaps the best known fact of this type is Wilkinson's result that, roughly, the *computed* solution of the $n \times n$ system of linear equations $Ax = b$ found by Gaussian elimination with row interchanges satisfies $(A + \delta A)x = b$, where $\| \delta A \| \leq u \cdot f(n) \cdot \| A \|$. Here $\| \cdot \|$ is a certain matrix norm, f is a specified function of n, and u is a bound on the local rounding error (for more details see Stewart [21, pp. 148–159]). These analyses are generally performed only once for a given algorithm and they are usually not intended to provide the user with realistic information about the error in his results. Furthermore, rounding errors are often treated separately (however, see Case Study IV, Section 10). In fact, to get meaningful results it may be necessary to compensate for inherent sensitivity to data error (see Stewart [21, pp. 69–80]).

Only very recently have attempts been made to find nontrivial uses for the computer in global roundoff analyses. Preliminary work has been done on the use of theorem-proving techniques (Hull et al. [8]), symbol-manipulation programs (Kahan [10], Stoutemyer [22]), and certain ad hoc methods (Miller [11, 12]). The motivation for these attempts resembles that for automatic program verification (Elspas [5]) more than it resembles the motivation for interval arithmetic.

The goal of this paper is to explain a computer technique for global roundoff analysis which is of definite practical value. We have no intention of trying to automate analyses as precise as, say, those depending on the idiosyncrasies of a particular machine. Nor shall we consider numerical methods for processes from real analysis, like differential equations. Rather, we will take aim at roundoff analyses for "algebraic processes." Our techniques can successfully analyze algorithms from areas other than linear algebra (see Brent [3] and Case Study IV, Section 10), but the exact domain must be left imprecise since we are far from a complete understanding of the range of applicability of these methods.

Even the limited goal of automating roundoff analyses of algebraic processes seems impossible. Further concessions must be made. For this paper we make the following sacrifices.

(1) Our methods are diagnostic and not prescriptive. The most they can do is to locate numerical instability. No hint is given for correcting the problem.

(2) No attempt is made to deal with an arbitrary numerical algorithm complete with looping and branching. Instead, small dimensions are fixed to limit the program to approximately 100 (or fewer) arithmetic operations. Furthermore, possible paths through any comparisons are treated separately.

(3) We consider only "worst case" results, never statements about (i) the "probable" error for a fixed set of data or (ii) about the error for an "average" set of data. For algorithms of the type we are considering there are important probabilistic statements of the second kind, like Wilkinson's assertion (e.g. [24, 105–107]) that the computed solution of triangular equations usually has small relative error (even given the worst possible combination of rounding errors). Wilkinson draws many interesting conclusions from this fact. On the other hand, probabilistic statements of the first type (see [24, pp. 25–26]) are of less interest when analyzing programs with only, say, 100 operations.

(4) A "heuristic" approach is used instead of seeking a method which always produces full information. Thus the technique advocated here will sometimes yield misleading results (see Example 2, Section 4, and Case Study I, Section 7).

Limited experimentation indicates that in spite of concession (4) we can get surprisingly accurate information at a small cost. The first three sacrifices seem to be more important. For example, consider the following informal statements (for more details see Stewart [21, pp. 148–159]).

(i) Without any form of pivoting, Gaussian elimination is unstable.

(ii) With partial pivoting, the effect of rounding error can grow exponentially with the dimension n. With complete pivoting it is bounded by a small polynomial $p(n)$.

(iii) The possible exponential growth with partial pivoting is extremely unlikely.

Our techniques are designed to verify statements like (i). For (ii) they are of doubtful value, and for (iii) they are probably worthless.

There is yet another concession to be made, namely, that we will use an idealized model of machine arithmetic. First we make the standard simplifications of ignoring overflow and underflow and of using only one of the properties of rounding error, i.e. a uniform bound (3.0) on the local error. Second is the (not so standard) idealization of considering only the first order effects of errors (i.e. we use derivatives). By considering bogus rounding errors we open the door for pessimistic results; by neglecting products of rounding errors we introduce a tendency for optimistic results and rule out the possibility of showing, for example, that a computation "loses half of the significant digits."

3. ROUNDING ERRORS AND OTHER PERTURBATIONS

We now consider ways of defining a number $\rho(\mathbf{d})$ which measures the maximum effect of rounding errors upon the computation by a fixed algorithm with data $\mathbf{d}$. The specific results of this paper concern the sensitivity of a single number $R(\mathbf{d}, \boldsymbol{\delta})$ to rounding errors $\boldsymbol{\delta}$. The more general case of vector-valued $R(\mathbf{d}, \boldsymbol{\delta})$ is considered in Miller [13, 15].

The functions $R(\mathbf{d}, \boldsymbol{\delta})$ arise as follows. Consider a fixed sequence of m binary operations, $+$, $-$, $\times$, or $/$, applied to a set $\mathbf{d} = (d_1, \ldots, d_n)$ of data (we will later allow the unary operation $\sqrt{}$). A relative rounding error of δ_j is associated with the jth operation. Hence the computed value of any intermediate result V is a function $V(\mathbf{d}, \boldsymbol{\delta})$, $\boldsymbol{\delta} = (\delta_1, \ldots, \delta_m)$.

For example, let $\mathbf{d} = (a, b)$ and consider the algorithm

$$X \leftarrow a \times b$$
$$Y \leftarrow X + b$$
$$Z \leftarrow a \times Y$$

We have

$$X(\mathbf{d}, \boldsymbol{\delta}) = ab(1 + \delta_1)$$
$$Y(\mathbf{d}, \boldsymbol{\delta}) = [ab(1 + \delta_1) + b](1 + \delta_2)$$
$$Z(\mathbf{d}, \boldsymbol{\delta}) = \{a[ab(1 + \delta_1) + b](1 + \delta_2)\}(1 + \delta_3).$$

Thus the rounding errors enter in the manner of floating-point arithmetic with a guard digit (see Wilkinson [24, pp. 7–11]). Other modes, for example, no guard digit or fixed-point arithmetic, can be handled with slight modifications.

Often the choice $R(\mathbf{d}, \boldsymbol{\delta}) = V_m(\mathbf{d}, \boldsymbol{\delta})$ = "the last computed value" is made (for example, see Case Study I, Section 7). More generally we will take $R(\mathbf{d}, \boldsymbol{\delta})$ to be some function of $\mathbf{d}$ and of the m computed intermediate values $V_j(\mathbf{d}, \boldsymbol{\delta})$.

Intuitively, our only constraint upon $\boldsymbol{\delta}$ is that the δ_j are uniformly bounded by some miniscule constant u. Using the uniform norm $\| \boldsymbol{\delta} \| = \max_{1 \le j \le m} | \delta_j |$ we can express this requirement as

$$\| \boldsymbol{\delta} \| \le u \quad \text{where } u > 0 \text{ is fixed.} \tag{3.0}$$

Our interest often centers on the problem of determining the maximum sensitivity of $R(\mathbf{d}, \boldsymbol{\delta})$ to the errors $\boldsymbol{\delta}$. The number

$$\sigma(\mathbf{d}) = \sum_{j=1}^{m} | (\partial R / \partial \delta_j)(\mathbf{d}, \mathbf{0}) | \tag{3.1}$$

measures this sensitivity in the sense that we have the *approximate* bound

$$| R(\mathbf{d}, \boldsymbol{\delta}) - R(\mathbf{d}, \mathbf{0}) | \lesssim \sigma(d) \cdot u \tag{3.2}$$

whenever $| \boldsymbol{\delta} | \leq u$. Notice that (3.2) is merely a natural generalization of the single-variable approximation, $| r(h) - r(0) | \lesssim | r'(0) | \cdot u$, whenever $| h | \leq u$. Intuitively, σ is roughly the maximum factor by which the error in the computed value of $R(\mathbf{d}) = R(\mathbf{d}, \mathbf{0})$ can exceed the roundoff level u.

More formally we have the following easy result, whose proof we omit.

Proposition 1. *Let* $\mathbf{d}$ *be such that* $R(\mathbf{d}, \boldsymbol{\delta})$ *is a differentiable function of* $\boldsymbol{\delta}$ *at* $\boldsymbol{\delta} = \mathbf{0}$. *Then* $\sigma(\mathbf{d}) = lim_{u \to 0+} \sigma(\mathbf{d}, u)$, *where* $\sigma(\mathbf{d}, u) = u^{-1} \cdot sup_{|\delta| \leq u} | R(\mathbf{d}, \boldsymbol{\delta}) - R(\mathbf{d}, \mathbf{0}) |$.

Notice in Proposition 1 that $| R(\mathbf{d}, \boldsymbol{\delta}) - R(\mathbf{d}, \mathbf{0}) | \leq \sigma(\mathbf{d}, u) \cdot u$ holds exactly. We work with $\sigma(\mathbf{d})$ instead of $\sigma(\mathbf{d}, u)$ since the latter is harder to evaluate and has a "machine dependent" flavor.

It is often the case that for certain sets $\mathbf{d}$ of data we have reason to expect $\sigma(\mathbf{d})$ to be large even for "stable" algorithms. These $\mathbf{d}$ which are inherently hypersensitive to small errors are called *ill-conditioned* (with respect to R). We can often determine a reasonable "condition number" $K(\mathbf{d}) > 0$ which measures the extent of this ill-condition: $\mathbf{d}$ is ill-conditioned if $K(\mathbf{d})$ is large, well-conditioned if $K(\mathbf{d})$ is small.

We then define

$$\rho(\mathbf{d}) = \sigma(\mathbf{d}) / K(\mathbf{d}) = \left(\sum_{j=1}^{m} | (\partial R / \partial \delta_j)(\mathbf{d}, \mathbf{0}) | \right) \Big/ K(\mathbf{d}). \tag{3.3}$$

Then we have the approximate bound

$$| R(\mathbf{d}, \boldsymbol{\delta}) - R(\mathbf{d}, \mathbf{0}) | \lesssim \rho(d) \cdot K(\mathbf{d}) \cdot u \tag{3.4}$$

whenever $| \boldsymbol{\delta} | \leq u$. Intuitively, if $\rho(\mathbf{d})$ is always of moderate size, then the effect of rounding errors is never much worse than can be explained by ill-condition.

One "condition number" which is often useful measures the sensitivity of the "exact value" $R(\mathbf{d}) = R(\mathbf{d}, \mathbf{0})$ to rounding errors in the data $\mathbf{d}$. It is easily seen that the largest possible effect of changing d_i to $d_i^1 = d_i(1 + \pi_i)$, where $| \pi_i | \leq u$ for $i = 1, 2, \ldots, n$, is $| R(\mathbf{d}^1, \mathbf{0}) - R(\mathbf{d}, \mathbf{0}) | \approx u \cdot \sum_{i=1}^{n} | d_i \cdot (\partial R / \partial d_i)(\mathbf{d}, \mathbf{0}) |$, a natural generalization of the single variable approximation $| r(x \cdot (1+\pi)) - r(x) | \approx | \pi | \cdot | x \cdot r'(x) |$. Using $K(\mathbf{d}) = \sum_{i=1}^{n} | d_i \cdot (\partial R / \partial d_i)(\mathbf{d}, \mathbf{0}) |$ in (3.3) we have the intuitive interpretation that for

$$\rho = \rho(\mathbf{d}) = \sum_{j=1}^{m} | (\partial R / \partial \delta_j)(\mathbf{d}, \mathbf{0}) | \Big/ \sum_{i=1}^{n} | d_i \cdot (\partial R / \partial d_i)(\mathbf{d}, \mathbf{0}) | \tag{3.5}$$

the effect of rounding error at $\mathbf{d}$ is at most ρ times worse than the possible effect of merely rounding the data. (For a formal interpretation see [15, theorem 2.1]). Functionals ρ of the form (3.5) are employed in Sections 4 and 7.

4. SOME SIMPLE EXAMPLES

To illustrate the notation of Section 3 we will consider three elementary examples of functionals

$$\rho(\mathbf{d}) = \sum_{j=1}^{m} | (\partial R / \partial \delta_j)(\mathbf{d}, \mathbf{0}) | \Big/ \sum_{i=1}^{n} | d_i \cdot (\partial R / \partial d_i)(\mathbf{d}, \mathbf{0}) |. \tag{4.1}$$

In each case $n = 1$ (i.e. the program has a single input) and $R(d, \boldsymbol{\delta}) = V_m(d, \boldsymbol{\delta})$ (i.e. we consider the effect of rounding errors upon the final computed value).

Example 1 (see Figure 1). For

$$x \leftarrow d \times d, \qquad y \leftarrow d + x, \qquad z \leftarrow y - x,$$

we have

$$R(d, \mathbf{0}) = d, \qquad R(d, \boldsymbol{\delta}) = \{[d + d^2(1 + \delta_1)](1 + \delta_2) - d^2(1 + \delta_1)\}(1 + \delta_3),$$

and using (4.1),

$$\rho(d) = (|\, d + d^2 \,| + |\, d \,|)/|\, d \,|.$$

When $\rho(d)$ is very large we run the risk that roundoff in $R(d, \boldsymbol{\delta})$ will yield a much poorer result than can be explained by the sensitivity of R to small variations in d. Hence we seek those values of d for which $\rho(d)$ is huge.

Clearly, hill-climbing techniques applied to $\rho(d)$ anywhere except at $d = -1$ or $d = 0$ will reveal that the given process is numerically unstable for d of large magnitude.

Example 2 (see Figure 2). For

$$x \leftarrow d \times d, \qquad y \leftarrow d + x, \qquad z \leftarrow y - d,$$

we find

$$\rho(d) = (2d^2 + |\, d + d^2 \,|)/2d^2.$$

Hill-climbing started at $d < -1$ gives the mistaken appearance that ρ is bounded by $\frac{3}{2}$. From $d > -1$, $d \neq 0$, one finds the numerical instability around $d = 0$.

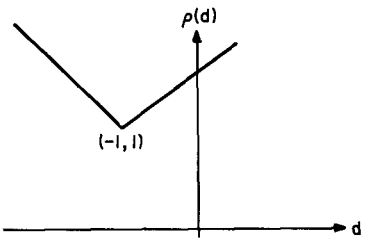

Fig. 1

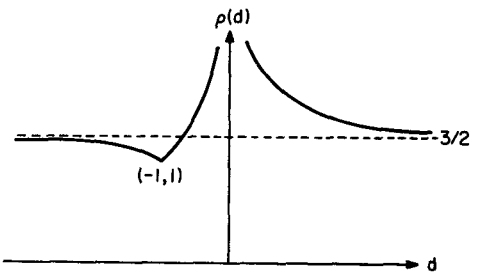

Fig. 2

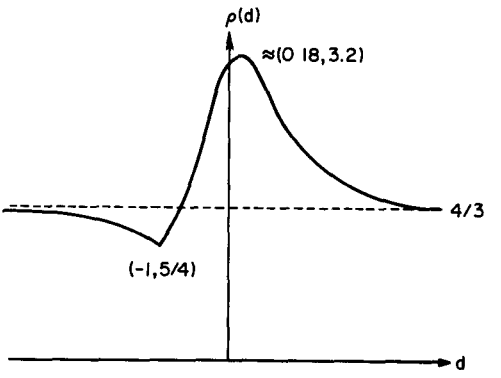

Fig. 3

Example 3 (see Figure 3). For

$$v \leftarrow d \times d, \quad w \leftarrow d + v, \quad x \leftarrow d \times v, \quad y \leftarrow w + x, \quad z \leftarrow y - v,$$

we find

$$\rho(d) = (2 \mid d^3 \mid + \mid d + d^2 \mid + \mid d + d^2 + d^3 \mid + \mid d + d^3 \mid)/\mid d + 3d^3 \mid.$$

From $d > -1$, $d \neq 0$, one converges to the actual bound of just under 3.2 for ρ. Beginning at $d < -1$, hill-climbing does not pass $\frac{4}{3}$.

5. THE PROGRAMS

Listing 1 shows the following three Fortran routines to aid roundoff analyses of the type described above:

(1) a calling program which performs input and output duties;

(2) a primitive numerical maximizer called UP;

(3) a subroutine ROUND which generates each value computed by a given straight-line program and which simultaneously computes the partial derivatives of the value with respect to the given set of data and to the rounding errors.

These programs are to be used in conjunction with a user-supplied subroutine called USER, which performs the actual evaluation of $\rho(d)$. In the case studies we give examples of USER subroutines and of typical sets of data.

6. SOME COMMENTS ON THE PROGRAMS

Our experience with these programs suggests that computer costs are often negligible (see Sections 7 through 10). The programs reflect the decision to consider portability, modifiability, and simplicity over efficiency. In cases where efficiency is a dominant criterion, for instance, if a straight-line program with hundreds of instructions is to be tested, two changes immediately suggest themselves.

(1) A machine language subroutine could replace ROUND for the evaluation of partial derivatives. In fact, one can easily imagine a "precompiler" which accepts, say, a Fortran-like specification of the algorithm (perhaps written with DO-loops) and which produces such a subroutine.

(2) More sophisticated maximization techniques could be employed. However, this modification is not as natural as it may seem. For our purposes the performance near finite local maxima is relatively unimportant. It is the ability to locate singularities that really counts. The usual efficient maximization routines often perform very poorly in this respect.

Modifiability of the maximization routine is especially important. For instance, one might want to test an algorithm which uses conditional branching. It is quite easy to modify our subroutine UP so that the search is limited to **d** for which, say, the tenth computed value is positive. In extreme cases stability of the algorithm is needed only on a domain so restricted that the maximizer can be removed and a few representative samples of ρ taken (see Case Study IV, Section 10).

Other modifications are also useful. Perhaps most natural are changes which allow analysis of straight-line programs involving operations other than $+$, $-$, $\times$, $/$, and $\sqrt{}$. It is easy to accommodate unary minus, unary inverse, trigonometric

LISTING 1

```
      COMMON VALUE(100),DELTAX(100,20),DELTAE(100,100),D(20)
     1,RHO,STOPX,NLOP(100),NOPER(100),NROP(100),NOP,NDIM
      DOUBLE PRECISION VALUE,DELTAX,DELTAE,D,RHO,STOPX,BIG
C THIS MAIN PROGRAM PERFORMS INPUT/OUTPUT FUNCTIONS (ERROR
C MESSAGES MAY ALSO BE GENERATED BY THE SUBROUTINE ROUND).
C THE FOLLOWING ARE READ.
C   NOP - THE NUMBER OF OPERATIONS IN THE STRAIGHT-LINE
C     PROGRAM TO BE ANALYZED.  1.LE.NOP.LE.100.
C   NLOP(I) - THE LEFT OPERAND IN THE I-TH INSTRUCTION.  THE
C     CONSTANT 1 IS ENCODED AS 0, THE K-TH ENTRY OF THE DATA
C     AS K, AND THE J-TH COMPUTED VALUE AS J+100.
C   NOPER(I) - THE OPERATOR IN THE I-TH INSTRUCTION.  +,-,*,
C     / AND SQRT ARE ENCODED AS 1,2,3,4 AND 5, RESPECTIVELY.
C   NROP(I) - THE RIGHT OPERAND IN THE I-TH INSTRUCTION,
C     ENCODED LIKE NLOP.  FOR SQRT USE NROP(I) = 0.
C   NDIM - THE NUMBER OF DATA ENTRIES.  1.LE.NDIM.LE.20.
C   D(I) - THE I-TH ENTRY IN THE INITIAL SET OF DATA.
C   STOPX - EXECUTION STOPS IF RHO ATTAINS THIS VALUE.
C   ITMAX - THE NUMBER OF TIMES THE MAXIMIZER IS TO BE
C     APPLIED.  TYPICAL VALUES ARE 5-10.
C THE SUBROUTINE UP IS CALLED TO ALTER D SO AS TO INCREASE
C RHO.  RHO IS EVALUATED BY THE SUBROUTINE ROUND.  FOR EACH K,
C 1.LE.K.LE.NOP, ROUND COMPUTES THE FOLLOWING.
C   VALUE(K) - THE RESULT OF THE K-TH OPERATION GIVEN THE
C     CURRENT D.
C   DELTAX(K,I) - THE VALUE AT (D,0) OF THE PARTIAL
C     DERIVATIVE OF THE K-TH COMPUTED VALUE WITH RESPECT TO
C     D(I), 1.LE.I.LE.NDIM.
C   DELTAE(K,J) - THE VALUE AT (D,0) OF THE PARTIAL
C     DERIVATIVE OF THE K-TH COMPUTED VALUE WITH RESPECT TO
C     THE J-TH ROUNDING ERROR.  1.LE.J.LE.NOP.
C ROUND THEN CALLS THE USER-SUPPLIED SUBROUTINE USER TO
C COMPLETE THE EVALUATION OF RHO.  NORMAL OUTPUT ENDS WITH
C A LIST OF INCREASING VALUES OF RHO AND THE FINAL VALUE OF
C D.
      NIN = 5
C NIN = STANDARD INPUT UNIT
      NOUT = 6
C NOUT = STANDARD OUTPUT UNIT
      READ(NIN,1) NOP
    1 FORMAT(I2)
      IF((NOP.LE.0).OR.(NOP.GT.100)) GO TO 99
      DO 4 I = 1,NOP
        READ(NIN,2) MLOP,MOPER,MROP
    2   FORMAT(I3,I1,I3)
        WRITE(NOUT,3) I,MLOP,MOPER,MROP
    3   FORMAT(2I4,I2,I4)
        II = I + 100
        IF((MLOP.LT.0).OR.(MLOP.GE.II)) GO TO 99
        IF((MROP.LT.0).OR.(MROP.GE.II)) GO TO 99
        IF((MOPER.LE.0).OR.(MOPER.GT.5)) GO TO 99
        NLOP(I) = MLOP
        NOPER(I) = MOPER
        NROP(I) = MROP
    4 CONTINUE
      READ(NIN,1) NDIM
      IF((NDIM.LE.0).OR.(NDIM.GT.20)) GO TO 99
      DO 7 I = 1,NDIM
        READ(NIN,5) D(I)
    5   FORMAT(D28.16)
        WRITE(NOUT,6) I,D(I)
    6   FORMAT(3H D(,I2,4H) = ,G16.8)
    7 CONTINUE
      READ(NIN,5) STOPX
      WRITE(NOUT,8) STOPX
    8 FORMAT(19H STOPPING VALUE IS ,G10.4)
      READ(NIN,1) ITMAX
      WRITE(NOUT,9) ITMAX
```

LISTING 1 (continued)

```
    9 FORMAT(23H USE MAXIMIZER AT MOST ,I2,6H TIMES)
      CALL ROUND
      WRITE(NOUT,10) RHO
   10 FORMAT(4X,G10.4)
      IF((RHO.GE.STOPX).OR.(ITMAX.LE.0)) GO TO 14
      BIG = RHO
      DO 11 ITIMES = 1,ITMAX
        CALL UP
        WRITE(NOUT,10) RHO
        IF((RHO.GE.STOPX).OR.(RHO.LE.BIG)) GO TO 12
        BIG = RHO
   11 CONTINUE
   12 DO 13 I = 1,NDIM
        WRITE(NOUT,6) I,D(I)
   13 CONTINUE
   14 STOP
   99 WRITE(NOUT,100)
  100 FORMAT(15H INCORRECT DATA)
      GO TO 14
      END



      SUBROUTINE UP
      COMMON VALUE(100),DELTAX(100,20),DELTAE(100,100),D(20)
     1,RHO,STOPX,NLOP(100),NOPER(100),NROP(100),NCP,NDIM
      DOUBLE PRECISION VALUE,DELTAX,DELTAE,D,RHO,STOPX,DI,H
     1,BEST
C TO FIND A LARGER VALUE OF RHO THIS SUBROUTINE SEARCHES BY
C SUCCESSIVELY CHANGING EACH ENTRY OF D.  INITIALLY THE
C I-TH ENTRY IS MODIFIED BY 1/100 PER CENT.  THIS AMOUNT IS
C DOUBLED AT MOST 13 TIMES.  SINCE 0.0001*(2**13) IS ABOUT
C 0.8, THE ENTRY WILL NEITHER CHANGE SIGN NOR DOUBLE IN
C MAGNITUDE.  AS SOON AS RHO DECREASES THE NEXT COORDINATE
C OF D IS SEARCHED.  CONTROL IS RETURNED AFTER ALL NDIM
C ENTRIES ARE TRIED.
      BEST = RHO
      DO 5 I = 1,NDIM
        DI = D(I)
        IF(DI.EQ.0.0D0) GO TO 5
        H = DI*1.0D-4
        D(I) = DI + H
        CALL ROUND
C WE CAN NOW TELL WHICH DIRECTION TO SEARCH.
        IF(RHO.GE.STOPX) GO TO 6
        IF(RHO.LE.BEST) GO TO 1
        BEST = RHO
        H = H + H
        LIM = 13
        GO TO 2
    1   H = -H
        LIM = 14
C SEARCH.  SUCCESSIVELY DOUBLE THE INCREMENT H IN D(I).
    2   DO 3 ITIMES = 1,LIM
          D(I) = DI + H
          CALL ROUND
          IF(RHO.GE.STOPX) GO TO 6
          IF(RHO.LE.BEST) GO TO 4
          BEST = RHO
    3   H = H + H
C GO BACK ONE STEP
    4   D(I) = DI + 0.5D0*H
C PERHAPS RHO COULD NOT BE INCREASED
        IF((LIM.EQ.14).AND.(ITIMES.EQ.1)) D(I) = DI
      RHO = BEST
    5 CONTINUE
    6 RETURN
      END
```

LISTING 1 (continued)

```
      SUBROUTINE ROUND
      COMMON VALUE(100),DELTAX(100,20),DELTAE(100,100),D(20)
     1,RHO,STOPX,NLOP(100),NOPER(100),NROP(100),NOP,NDIM
      DOUBLE PRECISION VALUE,DELTAX,DELTAE,D,RHO,STOPX,ARGL
     1,ARGR,DERIVL,DERIVR,DSQRT
C GIVEN DATA D THIS SUBROUTINE EVALUATES ALL INTERMEDIATE
C VALUES OF THE STRAIGHT-LINE PROGRAM AND THEIR PARTIAL
C DERIVATIVES WITH RESPECT TO THE DATA AND TO THE ROUNDING
C ERRORS.  RHO = 0 IS RETURNED IF SOME VALUE IS UNDEFINED,
C THEREBY CAUSING THE MAXIMIZER TO AVOID D.  OTHERWISE, THE
C USER-SUPPLIED SUBROUTINE USER IS CALLED TO EVALUATE RHO.
      NOUT = 6
C NOUT = STANDARD OUTPUT UNIT
C BEGIN THE LOOP TREATING THE K-TH INSTRUCTION.
      DO 400 K = 1,NOP
C GET THE OPERANDS AND THE OPERATOR
        ILOP = NLOP(K)
        IOPER = NOPER(K)
        IROP = NROP(K)
C IF(ILOP.EQ.0), THEN THE LEFT ARGUMENT IS THE CONSTANT 1.
C IF NOT, BUT (ILOP.LE.100), THEN IT IS THE ILOP ENTRY OF D.
C OTHERWISE, IT IS THE ILOP-100 COMPUTED VALUE.
        IF(ILOP.LE.100) GO TO 1
        ARGL = VALUE(ILOP-100)
        GO TO 3
    1   IF(ILOP.EQ.0) GO TO 2
        ARGL = D(ILOP)
        GO TO 3
    2   ARGL = 1.0D0
    3   IF(IROP.LE.100) GO TO 4
        ARGR = VALUE(IROP-100)
        GO TO 6
    4   IF(IROP.EQ.0) GO TO 5
        ARGR = D(IROP)
        GO TO 6
    5   ARGR = 1.0D0
C BRANCH ACCORDING TO THE OPERATOR.  EXECUTE THE OPERATION.
    6   GO TO (7,8,9,10,11), IOPER
    7   VALUE(K) = ARGL + ARGR
        GO TO 100
    8   VALUE(K) = ARGL - ARGR
        GO TO 100
    9   VALUE(K) = ARGL*ARGR
        GO TO 100
   10   IF(ARGR.EQ.0.0D0) GO TO 12
        VALUE(K) = ARGL/ARGR
        GO TO 100
   11   IF(ARGL.LT.0.0D0) GO TO 14
        VALUE(K) = DSQRT(ARGL)
        GO TO 100
C IF THE OPERATION IS IMPOSSIBLE, THEN PRINT A MESSAGE, SET
C RHO = 0 AND RETURN.
   12   WRITE(NOUT,13)
   13   FORMAT(25H GIVEN PROCEDURE TRIED /0)
        RHO = 0.0D0
        GO TO 1000
   14   WRITE(NOUT,15)
   15   FORMAT(38H GIVEN PROCEDURE TRIED SQRT(V), V.LT.0)
        RHO = 0.0D0
        GO TO 1000
C EVALUATE THE PARTIAL DERIVATIVE OF THE K-TH VARIABLE
C WITH RESPECT TO THE I-TH DATA ENTRY.  IF(ILOP.LE.100),
C THEN THE DERIVATIVE OF THE LEFT ARGUMENT IS EITHER 1 OR 0.
C OTHERWISE, IT HAS ALREADY BEEN COMPUTED
  100   DO 200 I = 1,NDIM
          IF(ILOP.LE.100) GO TO 101
          DERIVL = DELTAX(ILOP-100,I)
          GO TO 102
```

LISTING 1 (continued)

```
  101      DERIVL = C.0DC
           IF(ILOP.EQ.I) DERIVL = 1.0D0
  102      IF(IROP.LE.100) GO TC 103
           DERIVR = DELTAX(IROP-100,I)
           GO TO 104
  103      DERIVR = C.0DO
           IF(IROP.EQ.I) DERIVR = 1.0D0
  104      GO TO (105,106,107,108,109), ICPER
  105      DELTAX(K,I) = DERIVL + DERIVR
           GO TO 200
  106      DELTAX(K,I) = DERIVL - DERIVR
           GO TO 200
  107      DELTAX(K,I) = (ARGL*DERIVR) + (ARGR*DERIVL)
           GO TO 200
  108      DELTAX(K,I) = ((ARGR*DERIVL) - (ARGL*CERIVR))/
     1                                   (ARGR*ARGR)
           GO TO 200
  109      DELTAX(K,I) = 0.5D0*(DERIVL/VALUE(K))
  200    CONTINUE
C EVALUATE THE PARTIAL DERIVATIVE OF THE K-TH VARIABLE WITH
C RESPECT TO THE J-TH ROUNDING ERRCR.  IF(J.GT.K) IT IS 0.
C IF(J.EQ.K) IT IS VALUE(K). OTHERWISE IT MUST BE COMPUTED.
         IF(K.EQ.1) GO TO 301
         KK = K-1
         DO 300 J = 1,KK
           IF(ILOP.LE.100) GO TO 201
           DERIVL = DELTAE(ILOP-100,J)
           GO TO 202
  201      DERIVL = 0.0DO
  202      IF(IROP.LE.100) GO TO 203
           DERIVR = DELTAE(IROP-100,J)
           GO TO 204
  203      DERIVR = C.0DO
  204      GO TO (205,206,207,208,209), ICPER
  205      DELTAE(K,J) = DERIVL + DERIVR
           GO TO 300
  206      DELTAE(K,J) = DERIVL - DERIVR
           GO TO 300
  207      DELTAE(K,J) = (ARGL*DERIVR) + (ARGR*DERIVL)
           GO TO 300
  208      DELTAE(K,J) = ((ARGR*DERIVL) - (ARGL*CERIVR))/
     1                                   (ARGR*ARGR)
           GO TO 300
  209      DELTAE(K,J) = 0.5D0*(DERIVL/VALUE(K))
  300    CONTINUE
  301    DELTAE(K,K) = VALUE(K)
         IF(K.EQ.NOP) GO TO 400
         KP = K + 1
         DO 302 J = KP,NOP
  302    DELTAE(K,J) = 0.0DO
  400  CONTINUE
       CALL USER
 1000  RETURN
       END
```

functions, etc. With more extensive modifications one can locate instabilities which essentially have nothing to do with rounding errors [14].

In cases where ρ assumes values of ∞ it often happens that numerical instability is diagnosed very quickly by even the most primitive maximization routine. This can happen if the singularity of $\rho(\mathbf{d})$ occurs on an $(n-1)$-dimensional manifold,

where n is the dimension of $\mathbf{d}$. Convergence by the maximizer is very rapid, since the situation is not appreciably more complicated than a one-dimensional search. This phenomenon occurs in Case Study II, Section 8.

7. CASE STUDY I. TRIDIAGONAL LINEAR SYSTEMS

Consider this problem: Given a 10-tuple $\mathbf{d} = (a_1, a_2, a_3, b_1, b_2, c_1, c_2, f_1, f_2, f_3)$ with nonsingular "left-hand side" $a_1, \ldots, c_2$, solve

$$\begin{aligned} a_1x + b_1y &= f_1 \\ c_1x + a_2y + b_2z &= f_2 \\ c_2y + a_3z &= f_3 . \end{aligned}$$

Algorithm A computes y by Gaussian elimination. Algorithm B uses "two-sided" elimination from Babuska [1].

Algorithm A	Algorithm B
$m_1 \leftarrow c_1/a_1$	$m_1 \leftarrow c_1/a_1$
$s_1 \leftarrow m_1 \times b_1$	$s_1 \leftarrow m_1 \times b_1$
$d_2 \leftarrow a_2 - s_1$	$d_2 \leftarrow a_2 - s_1$
$t_1 \leftarrow m_1 \times f_1$	$t_1 \leftarrow m_1 \times f_1$
$g_2 \leftarrow f_2 - t_1$	$g_2 \leftarrow f_2 - t_1$
$m_2 \leftarrow c_2/d_2$	$m_3 \leftarrow b_2/a_3$
$s_2 \leftarrow m_2 \times b_2$	$s_3 \leftarrow m_3 \times c_2$
$d_3 \leftarrow a_3 - s_2$	$d_2^* \leftarrow a_2 - s_3$
$t_2 \leftarrow m_2 \times g_2$	$t_3 \leftarrow m_3 \times f_3$
$g_3 \leftarrow f_3 - t_2$	$g_2^* \leftarrow f_2 - t_3$
$z \leftarrow g_3/d_3$	$u \leftarrow g_2 + g_2^*$
$u \leftarrow b_2 \times z$	$v \leftarrow u - f_2$
$v \leftarrow g_2 - u$	$w \leftarrow d_2 + d_2^*$
$y \leftarrow v/d_2$	$r \leftarrow w - a_2$
	$y \leftarrow v/r$

For each method of computation we ask: How much more is y changed by rounding errors than is it changed by merely rounding the data? The appropriate functional ρ is (3.5)

$$\rho = \rho(\mathbf{d}) = \sum_{j=1}^{m} \left| (\partial R/\partial \delta_j)(\mathbf{d}, \mathbf{0}) \right| \Big/ \sum_{i=1}^{10} \left| d_i \cdot (\partial R/\partial d_i)(\mathbf{d}, \mathbf{0}) \right|$$

where m equals 14 for algorithm A and 15 for algorithm B and where $R = V_m$, the *computed* value of y. Theorem 5.1 of Babuska [1] implies that ρ is unbounded for algorithm A, but bounded by 9 for algorithm B.

We tested our program by verifying Babuska's results. For $\mathbf{d} = (1, 1, 1, 1, \ldots, 1)$ or $\mathbf{d} = (1, 2, 1, 1, \ldots, 1)$, algorithm A calls for division by zero. This does not

LISTING 2

```
      SUBROUTINE USER
      COMMON VALUE(100),DELTAX(100,20),DELTAE(100,100),C(20)
     1,RHO,STOPX,NLOP(100),NOPER(100),NROP(100),NCP,NDIM
      DOUBLE PRECISION VALUE,DELTAX,DELTAE,D,RHO,STOPX,DABS
     1,TOP,BOTTOM
C THIS SUBROUTINE EVALUATES THE FUNCTIONAL RHO USED IN CASE
C STUDY I.
      NOUT = 5
      TOP = 0.0D0
      DO 1 J = 1,NOP
    1 TOP = TOP + DABS(DELTAE(NOP,J))
      BOTTOM = 0.0D0
      DO 2 I = 1,NDIM
    2 BOTTOM = BOTTOM + DABS(D(I)*DELTAX(NCP,I))
      IF(BOTTOM.EQ.0.0D0) GO TO 4
      RHO = TOP/BOTTOM
    3 RETURN
    4 RHO = 0.0D0
      WRITE(NOUT,5) TOP
    5 FORMAT(7H RHO = ,G10.4,4H / 0)
      GO TO 3
      END
```

cause our program any difficulty, but we chose to avoid these sets of data. Both algorithms A and B were tested using the initial values

$$\mathbf{d}_1 = (1, 1.1, 1, 1, \ldots, 1),$$

$$\mathbf{d}_2 = (1, 2.1, 1, 1, \ldots, 1),$$

$$\mathbf{d}_3 = (-1, 1, 1, 1, \ldots, 1),$$

$$\mathbf{d}_4 = (1, -1, 1, 1, \ldots, 1).$$

As in Case Studies II and III, Sections 8 and 9, we set the parameter STOPX to 10^4 and ITMAX to 10 compiled under Fortran G and executed on the IBM 370/168 at Penn State.

First we analyzed algorithm A. Starting at $\mathbf{d}_1$ or $\mathbf{d}_4$ the maximizer quickly pushed ρ past STOPX. But from $\mathbf{d}_2$ or $\mathbf{d}_3$ it did not get past 1.75 or 3.0, respectively, reminding us that unstable algorithms may appear stable to our heuristic techniques (this problem did not arise in the other case studies). The largest $\rho(\mathbf{d})$ found for algorithm B (from any initial $\mathbf{d}_i$) was about 5.5. Execution times ranged from 2 to 6 seconds (at 10 cents per second).

Listing 2 shows the USER subroutine employed in this case study. Listing 3 shows the set of data read by our program when it tested algorithm A beginning at $\mathbf{d}_1$. We have added annotations.

8. CASE STUDY II. THE ORTEGA-KAISER METHOD

Ortega and Kaiser [17] give a rational (i.e. square-root-free) QR method for symmetric tridiagonal matrices. In the 3×3 case this process works with 5-tuples $\mathbf{d} = (a_1, a_2, a_3, b_1^2, b_2^2)$ representing matrices

$$A(\mathbf{d}) = \begin{pmatrix} a_1 & b_1 & 0 \\ b_1 & a_2 & b_2 \\ 0 & b_2 & a_3 \end{pmatrix}. \tag{8.1}$$

LISTING 3

```
14          NOP = number of operations in the following program
0064001     6 = sixth data entry, 4 = /, 1 = first data entry
1013004     101 = first computed value, 3 = ×, 4 = fourth data entry
0022102                  .
1013008                  .
0092104                  .
0074103     an encoding of algorithm A
1063005
0032107
1063105
0102109
1104108
0053111
1052112
1134103     NDIM = number of data entries
10              a1 = 1
1.0             a2 = 1.1
1.1             a3 = 1
1.0             b1 = 1
1.0               ⋮
1.0         d1
1.0
1.0
1.0
1.0
1.0
10000.      STOPX = stopping value for the maximizer
10          ITMAX = maximum number of iterations of the maximizer.
```

The algorithm (neglecting special cases which avoid division by zero)

$$
\begin{aligned}
p_1^2 &\leftarrow a_1 \times a_1 & s_2^2 &\leftarrow b_2^2/(p_2^2 + b_2^2)\\
s_1^2 &\leftarrow b_1^2/(p_1^2 + b_1^2) & u_2 &\leftarrow s_2^2(g_2 + a_3)\\
u_1 &\leftarrow s_1^2(a_1 + a_2) & \bar{a}_2 &\leftarrow g_2 + u_2\\
\bar{a}_1 &\leftarrow a_1 + u_1 & \bar{a}_3 &\leftarrow a_3 - u_2\\
g_2 &\leftarrow a_2 - u_1 & p_3^2 &\leftarrow \bar{a}_3^2/(1 - s_2^2)\\
p_2^2 &\leftarrow g_2^2/(1 - s_1^2) & \bar{b}_2^2 &\leftarrow s_2^2 \times p_3^2\\
\bar{b}_1^2 &\leftarrow s_1^2(p_2^2 + b_2^2)
\end{aligned}
$$

produces $Q(\mathbf{d}, \boldsymbol{\delta}) = (\bar{a}_1, \bar{a}_2, \bar{a}_3, \bar{b}_1^2, \bar{b}_2^2)$, where $A(\mathbf{d})$ and $A(Q(\mathbf{d}, \mathbf{0}))$ have the same eigenvalues (the eigenvalues can be easily located after several iterations).

For any $Q = Q(\mathbf{d}, \boldsymbol{\delta})$, let $\mathrm{EIG}(Q)$ be the absolute value of the largest eigenvalue of $A(Q)$. We wish to analyze the behavior of $R(\mathbf{d}, \boldsymbol{\delta}) = \mathrm{EIG}(Q(\mathbf{d}, \boldsymbol{\delta}))$, so we want to evaluate the derivatives $(\partial R/\partial \delta_j)(\mathbf{d}, \mathbf{0})$. Notice that the subroutine ROUND evaluates the 5×21 matrix $(\partial Q/\partial \boldsymbol{\delta})(\mathbf{d}, \mathbf{0})$. Thus we need only evaluate the derivative of EIG with respect to Q at $Q(\mathbf{d}, \mathbf{0})$ and then apply the chain rule.

It is possible to compute the derivative of EIG with respect to Q using standard perturbation results for eigenvalues ([24, pp. 137–138] or [25, ch. 2]). However, we prefer to use divided difference approximations, an approach which is both simpler and more general. For instance, given $Q = Q(\mathbf{d}, \mathbf{0}) = (\bar{a}_1, \bar{a}_2, \bar{a}_3, \bar{b}_1^2, \bar{b}_2^2)$ we can use an eigenvalue routine to compute $\mathrm{EIG} = \mathrm{EIG}(Q)$, change $\bar{a}_1$ to $\bar{a}_1 + H$ (getting Q_1), compute $\mathrm{BIG} = \mathrm{EIG}(Q_1)$, and then use the approximation $\partial \mathrm{EIG}/\partial \bar{a}_1 \approx (\mathrm{BIG} - \mathrm{EIG})/H$.

If $\mathbf{d}$ has large entries, then $Q = Q(\mathbf{d}, \mathbf{0})$ also has large entries so that just rounding the entries of Q will produce large absolute (though small relative) errors.

This, in turn, generally produces large absolute errors in EIG(Q). Let us compensate for this fact by working with $\rho(\mathbf{d}) = (\sum_{j=1}^{21} |\, (\partial R/\partial \delta_j)(\mathbf{d}, \mathbf{0})\,|)/|\, d\,|$, where $|\,\mathbf{d}\,| = |(a_1, a_2, a_3, b_1^2, b_2^2)| = \max\{|\,a_1\,|, |\,a_2\,|, |\,a_3\,|, b_1^2, b_2^2\}$. Thus $|\,\mathbf{d}\,|$ plays the role of $K(\mathbf{d})$ in (3.3).

With a USER subroutine based upon these ideas we chose the parameters $\mathbf{d}_0 = (.9, .8, .7, .6, .5)$, STOPX $= 10^4$, and IMAX $= 10$ (for $\mathbf{d} = (1, 1, 1, 1, 1)$ the Ortega-Kaiser method performs a rare branch to avoid division by zero). Almost immediately (i.e. after three seconds of CPU time) a $\mathbf{d}$ near (.73, .82, .7, .6, .5) was located for which $\rho(\mathbf{d}) > 10^4$, indicating instability. We tried all eight possible combinations of signs for the first three entries of $\mathbf{d}_0$ (the last two must be $+$). Instability was diagnosed each time, though not always so quickly as for the original $\mathbf{d}_0$. However, the cost was always under two dollars.

We also applied our program to a rational QR algorithm of Reinsch [18] (for other references see Stewart [21, p. 381]). Only minor changes in USER were required, e.g. this method uses 23 instructions in the 3×3 case and it produces the values in a different order. With the same eight starting values we could not locate a value of ρ exceeding 12.

The Ortega-Kaiser algorithm provides a clear illustration of the power of our techniques for locating numerical instability. It is not hard to see that the entries of the computed matrix $A(Q(\mathbf{d}, \boldsymbol{\delta}))$ can be extremely sensitive to rounding errors $\boldsymbol{\delta}$. However, the same can be said of many stable algorithms for eigenvalues (for example, see Wilkinson [25, p. 288]). To determine analytically the dependence of the eigenvalues upon the rounding errors seems to be a formidable problem. On the other hand, as we have just seen, it yields readily to numerical sampling.

Listing 4 shows the USER routine and Listing 5 a set of data for the analysis of the Ortega-Kaiser method.

9. CASE STUDY III. THE GRAM-SCHMIDT METHOD

Given linearly independent n-vectors $\mathbf{a}$, $\mathbf{b}$, and $\mathbf{c}$, either of the following versions of the Gram-Schmidt (GS) method produces orthonormal $\mathbf{x}$, $\mathbf{y}$, and $\mathbf{z}$ (assuming exact arithmetic).

GS*

$$\mathbf{x} \leftarrow \mathbf{a}/||\,\mathbf{a}\,||$$
$$\mathbf{u} \leftarrow \mathbf{b} - (\mathbf{b}^T\mathbf{x})\mathbf{x}$$
$$\mathbf{y} \leftarrow \mathbf{u}/||\,\mathbf{u}\,||$$
$$\mathbf{v} \leftarrow \mathbf{c} - (\mathbf{c}^T\mathbf{x})\mathbf{x}$$
$$\mathbf{w} \leftarrow \mathbf{v} - (\mathbf{c}^T\mathbf{y})\mathbf{y}$$
$$\mathbf{z} \leftarrow \mathbf{w}/||\,\mathbf{w}\,||$$

The modified Gram-Schmidt (MGS) is the same as GS* except that $\mathbf{w}$ is defined by

$$\mathbf{w} \leftarrow \mathbf{v} - (\mathbf{v}^T\mathbf{y})\mathbf{y}.$$

LISTING 4

```
      SUBROUTINE USER
      COMMON VALUE(100),DELTAX(100,20),DELTAE(100,100),D(20)
     1,RHO,STOPX,NLOP(100),NOPER(100),NPOP(100),NCP,NDIM
      DOUBLE PRECISION VALUE,DELTAX,DELTAE,D,RHO,STOPX,SAVE
     1,H,AQ(3,3),DSQRT,DUMMY,RCOT(3),BIG,AB,DABS,EIG
     1,DEIGDQ(5),S,SUM,DNORM
C THIS SUBROUTINE EVALUATES THE FUNCTIONAL RHO USED IN CASE
C STUDY II.
C THE I-TH ENTRY OF THE OUTPUT Q IS THE IOUT(I)-TH COMPUTED
C VALUE.
      INTEGER IOUT(5)
      DATA IOUT/6,16,17,12,21/
C THIS LOOP COMPUTES DIVIDED DIFFERENCE APPROXIMATIONS TO
C THE VECTOR DEIGDQ OF DERIVATIVES OF THE LARGEST EIGENVALUE
C WITH RESPECT TO THE OUTPUT Q.
      DO 4 II = 1,6
        I = II-1
C THE FIRST TIME THROUGH GET THE EXACT LARGEST EIGENVALUE.
        IF(I.EQ.0) GO TO 1
C OTHERWISE PERTURB THE I-TH OUTPUT.
        SAVE = VALUE(IOUT(I))
        H = SAVE*1.0D-7
        VALUE(IOUT(I)) = SAVE + H
C COMPUTE THE LARGEST EIGENVALUE OF A(Q), WHERE THE OUTPUT
C Q IS POSSIBLY PERTURBED.
    1   AQ(1,1) = VALUE(IOUT(1))
        AQ(2,2) = VALUE(IOUT(2))
        AQ(3,3) = VALUE(IOUT(3))
        AQ(1,2) = DSQRT(VALUE(IOUT(4)))
        AQ(2,3) = DSQRT(VALUE(IOUT(5)))
        AQ(1,3) = 0.0D0
C USE A LOCAL SYMMETRIC EIGENVALUE ROUTINE. THE THREE
C EIGENVALUES ARE RETURNED IN ROOT.
            CALL DMXDGI(3,-3,1,AQ,3,DUMMY,RCOT)
C GET THE LARGEST EIGENVALUE.
        BIG = 0.0D0
        DO 2 K = 1,3
          AB = DABS(ROOT(K))
          IF(AB.GT.BIG) BIG = AB
    2   CONTINUE
        IF(I.GT.0) GO TO 3
C THE FIRST TIME THROUGH GET THE EXACT LARGEST EIGENVALUE.
        EIG = BIG
        GO TO 4
C FIND THE DIVIDED DIFFERENCES.
    3   DEIGDQ(I) = (BIG - EIG)/H
        VALUE(IOUT(I)) = SAVE
    4 CONTINUE
C MULTIPLY DEIGDQ BY THE DERIVATIVE OF THE OUTPUT C WITH
C RESPECT TO THE ROUNDING ERRORS.  THIS YIELDS THE DERIVATIVE
C OF THE LARGEST EIGENVALUE WITH RESPECT TO THE ROUNDING
C ERRORS.  ADD THE ABSOLUTE VALUES OF THE RESULTS.
      S = 0.0D0
      DO 6 J = 1,NOP
        SUM = 0.0D0
        DO 5 I = 1,5
    5   SUM = SUM + DEIGDQ(I)*DELTAE(IOUT(I),J)
    6 S = S + DABS(SUM)
C DIVIDE BY THE NORM OF THE DATA.
      DNORM = 0.0D0
      DO 7 I = 1,NDIM
        AB = DABS(D(I))
        IF(AB.GT.DNORM) DNORM = AB
    7 CONTINUE
      RHO = S/DNORM
      RETURN
      END
```

LISTING 5

```
21
0013001
1011004
0044102
0011002
1033104
0011105
0022105
1073107
0002103
1084109
1101005
1033111
0054111
1071003
1133114
1071115
0032115
1173117
0002113
1184119
1133120
05
0.9
0.8
0.7
0.6
0.5
10000.
10
```

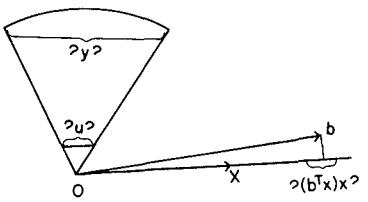

Fig. 4

The procedure GS* is intermediate between the standard GS method and the MGS procedure (see Stewart [21, pp. 216–217]). For the case $n = 3$ either procedure requires 60 arithmetic operations, including three square roots. The only difference between them lies in one operand of each of n multiplications.

In either case the **x**, **y**, **z** computed with rounding errors can be sharply nonorthogonal (though it is not hard to show that they must have nearly unit length). To see how this can happen suppose that the only rounding error in the computation of **y** occurs in the evaluation of $\mathbf{b}^T\mathbf{x}$. If **a** and **b** are nearly linearly dependent in the sense that **u** is small compared with **b**, then $\mathbf{x}^T\mathbf{y}$ may be large (see Figure 4).

Can the departure from orthogonality of the computed **x**, **y**, **z** be large without **a**, **b**, **c** being nearly linearly dependent? To make this question precise we need a measurement of dependence among **a**, **b**, **c**. Let **d** be the 3×3 matrix with columns **a**, **b**, **c** and define

$$K(\mathbf{d}) = \| \mathbf{d} \|_F \cdot \| \mathbf{d}^{-1} \|_F \tag{9.1}$$

where, e.g.

$$\| \mathbf{d} \|_F^2 = \sum_{i=1}^{9} d_i^2. \tag{9.2}$$

A large value of $K(\mathbf{d})$ means that **a**, **b**, **c** are nearly linearly dependent (see Stewart [21, ch. 4]; also see [21, p. 221] for a definition of $K(\mathbf{d})$ which can be used when $n > 3$).

Define $R(\mathbf{d}, \boldsymbol{\delta}) = \mathbf{y}^T\mathbf{z}$ and $\rho(\mathbf{d}) = (\sum_{j=1}^{60} | (\partial R/\partial \delta_j)(\mathbf{d}, \mathbf{0}) |)/K(\mathbf{d})$. Notice that

$$\rho(\mathbf{d}) = \left(\sum_{j=1}^{60} \left| \sum_{i=1}^{3} [y_i \cdot (\partial z_i/\partial \delta_j)(\mathbf{d}, \mathbf{0}) + z_i \cdot (\partial y_i/\partial \delta_j)(\mathbf{d}, \mathbf{0})] \right| \right) \Big/ K(\mathbf{d}) \tag{9.3}$$

LISTING 6

```
      SUBROUTINE USER
      COMMON VALUE(100),DELTAX(100,20),DELTAE(100,100),D(20)
     1,RHO,STOPX,NLOP(100),NOPER(100),NRCP(100),NCP,NDIM
      DOUBLE PRECISION VALUE,DELTAX,DELTAE,D,RHO,STOPX,A(3,3)
     1,SIGMA,PARTJ,DABS,T,FNORM,FNORM1,K,DSQRT
C THIS SUBROUTINE EVALUATES THE FUNCTIONAL RHO USED IN CASE
C STUDY III.
C EVALUATE THE NUMERATOR OF (9-3). THE ENTRIES OF Y ARE
C COMPUTED VALUES 27,28,29.  THE ENTRIES OF Z ARE COMPUTED
C VALUES 58,59,60.
      SIGMA = 0.0D0
      DO 2 J = 1,60
        PARTJ = 0.0D0
        DO 1 I = 1,3
    1   PARTJ = PARTJ + VALUE(26+I)*DELTAE(57+I,J)
     1                + VALUE(57+I)*DELTAE(26+I,J)
    2 SIGMA = SIGMA + DABS(PARTJ)
C EVALUATE (9-2) AND PREPARE FOR MATRIX INVERSION.
      FNORM = 0.0D0
      DO 3 I = 1,3
        DO 3 J = 1,3
        T = D(3*(J-1) + I)
        FNORM = FNORM + T*T
    3 A(I,J) = T
C USE A LOCAL MATRIX INVERSION ROUTINE
            CALL DMXINV(A,3,3)
C EVALUATE (9-1).
      FNORM1 = 0.0D0
      DO 4 I = 1,3
        DO 4 J = 1,3
          T = A(I,J)
    4 FNORM1 = FNORM1 + T*T
      K = DSQRT(FNORM*FNORM1)
C EVALUATE (9-3).
      RHO = SIGMA/K
      RETURN
      END
```

since $\mathbf{y}^T\mathbf{z} = \sum_{i=1}^{3} (y_i \cdot z_i)$. All components for the evaluation of ρ are available to the subroutine USER.

If ρ is uniformly bounded by some number B, then by (3.4) we can (approximately) guarantee that the computed $\mathbf{y}$ and $\mathbf{z}$ satisfy $| \mathbf{y}^T\mathbf{z} | \leq Bu \cdot K(\mathbf{d})$, since $R(\mathbf{d}, \boldsymbol{\delta}) = \mathbf{y}^T\mathbf{z}$ and $R(\mathbf{d}, \mathbf{0}) = 0$. For MGS such a bound is known to exist (see [2, p. 15]; for a less formal discussion of the numerical properties of GS procedures see Rice [20]).

To test GS* we chose the initial value

$$\mathbf{d}_0 = \begin{pmatrix} 1 & 2 & 1 \\ 1 & 1 & 2 \\ 1 & 1 & 1 \end{pmatrix}$$

and set STOPX = 10^4 and ITMAX = 10. After a few iterations of the maximizer, ρ exceeded 10^4. Similarly, ρ appeared unbound in each of several more tests where we began with some altered signs in $\mathbf{d}_0$. The largest value of ρ found for MGS was about 5.5. Thus, unlike MGS, GS* is unstable in the sense that the departure from orthogonality of the computed vectors is not merely proportional to the product of the unit rounding error and the condition number of the data.

These computations were somewhat more expensive than those in the other case

studies (though each run cost under five dollars). It is not hard to see why. The cost of evaluating the partial derivative is $O(m^2)$, where m is the number of operations in the straight-line program being analyzed. For reasons of economy we list only the USER routine, as shown in Listing 6.

10. CASE STUDY IV. STREAMLINED POLYNOMIAL FORMS

According to Fike [6, ex. 4] the following polynomial of degree 6 has been used in an IBM System/360 library routine as an approximation to 2^x for $-\frac{1}{16} \leq x \leq 0$.

$$\begin{aligned} P(x) = 1 &+ .6931471805599346x + .2402265069563678x^2 \\ &+ .05550410840231345x^3 + .009618117095313700x^4 \\ &+ .001333073417706260x^5 + .0001507368551403575x^6. \end{aligned}$$

In this form evaluation requires six multiplications and six additions. However, $P(x)$ can be evaluated in four multiplications and seven additions using the streamlined form

$$\begin{aligned} q_1 &\leftarrow a_1 x \\ q_2 &\leftarrow (q_1 + a_2)^2 \\ q_3 &\leftarrow (q_2 + a_3)(q_1 + a_4) \\ P(x) &\leftarrow (q_2 + q_3 + a_5)(q_3 + a_6) + a_7 \end{aligned}$$

where

$a_1 = .23069414645549892$ $a_4 = .10750058720428135$ $a_6 = 1.67018802067794794$

$a_2 = .20629788601177007$ $a_5 = .13060869898889258$ $a_7 = .48953177059399256.$

$a_3 = 1.00939719129704594$

We would like to assess the total relative error in the result caused by both the rounding errors and the fact that the a_i cannot be represented exactly. Let $\mathbf{d} = (d_1, \ldots, d_8) = (a_1, \ldots, a_7, x)$ and let $P(\mathbf{d}, \boldsymbol{\pi}, \boldsymbol{\delta})$ be the value computed by the streamlined scheme with rounding errors $\boldsymbol{\delta} = (\delta_1, \ldots, \delta_{11})$ given data $\mathbf{d}^1$, where $d_i^1 = d_i(1 + \pi_i)$ for $1 \leq i \leq 8$. Thus $P(\mathbf{d}, \mathbf{0}, \mathbf{0}) = P(x)$. Define

$$\rho(x) = \left(\sum_{i=1}^{8} \left| (\partial P/\partial \pi_i)(\mathbf{d}, \mathbf{0}, \mathbf{0}) \right| + \sum_{j=1}^{11} \left| (\partial P/\partial \delta_j)(\mathbf{d}, \mathbf{0}, \mathbf{0}) \right| \right) \Big/ |P(x)|$$

(think of the a_i as fixed, and $-\frac{1}{16} \leq x \leq 0$). By the obvious variation on the theme of Section 3 we see that if $|\boldsymbol{\pi}| \leq u$ and $|\boldsymbol{\delta}| \leq u$, then we have the approximate bound

$$(|P(d, \boldsymbol{\pi}, \boldsymbol{\delta}) - P(x)| / |P(x)|) \leq \rho(x) \cdot u.$$

Thus the computed value of $P(x)$ has a maximum relative error of at most $\rho(x) \cdot u$.

We modified the programs of Section 5 to evaluate $\rho(x)$ at 100 evenly spaced points between $-\frac{1}{16}$ and 0. The maximum value of ρ was about 5.6 (a complete listing shows $\rho(-\frac{1}{16}) \approx 5.5$, $\rho(0) \approx 5.6$, and ρ monotone increasing for $-\frac{1}{16} \leq x \leq 0$). The maximum for the corresponding ρ gotten by applying nested

multiplication (i e. Horner's rule) to the original form of $P(x)$ is about 2.2. If we take into account the fact that the constant coefficient 1.0 can be represented exactly (this can be done by removing the term $\partial P/\partial \pi_1$ from ρ), then ρ decreases to nearly 1. (Actually, we may also want to consider x as exact for both methods) Thus nested multiplication nearly produces a correctly rounded value of $P(x)$, while the error in the streamlined form may be about five times larger.

While we do not propose that methods like ours be used as the only experimental technique for certifying function evaluation routines, they may be useful for studies like that conducted by Rice [19] on the conditioning of polynomial and rational forms These techniques have the advantage of testing, for example, double precision routines using only double precision arithmetic instead of troublesome multiple precision. (Of course it is possible to compute what are essentially upper bounds for ρ by using interval arithmetic or simple inequalities like Hart et al. [7, p. 68, eq. 4.5.4]. However, such methods which use only $O(m)$ operations, corresponding to the m operations being analyzed, often yield pessimistic results)

The interested reader will have no problem seeing how to modify the first program of Section 5 to perform the above computation. The proper USER routine is only slightly different from that of Case Study I.

ACKNOWLEDGMENTS

Some preliminary work for this paper was done at the IBM Watson Research Center It could not have been written without support from H. R. Strong and help from W. Kahan, who, among other things, suggested the use of numerical maximization methods in computer-aided roundoff analysis. David Stoutemyer and the referee made helpful suggestions.

REFERENCES

1 Babuska, I. Numerical stability in problems of linear algebra. *SIAM J. Numer. Anal. 9* (1972), 53–77
2. Bjorck, A. Solving linear least square problems by Gram-Schmidt orthogonalization. *BIT 7* (1967), 1–21.
3 Brent, R. P. The parallel evaluation of general arithmetic expressions. *J. ACM 21*, 2 (April 1974), 201–206.
4 Businger, P A Eigenvalues of a real symmetric matrix by the QR method, Algorithm 253. *Comm ACM 8*, 4 (April 1965), 217–218.
5 Elspas, B , et al An assessment of techniques for proving program correctness. *Computing Surveys 4*, 2 (June 1972), 97–147.
6 Fike, C Methods of evaluating polynomial approximations in function evaluation routines. *Comm ACM 10*, 3 (March 1967), 175–178.
7. Hart, J., et al. *Computer Approximations.* Wiley, New York, 1968.
8 Hull, T , et al The correctness of numerical algorithms. Proc ACM Conf on Proving Assertions About Programs, New Mexico State U., University Park, N Mex , Jan 6–7, 1972
9 Kahan, W , and Varah, J Two working algorithms for the eigenvalues of symmetric tridiagonal matrices Stanford Tech Rep CS43, 1966
10 Kahan W One numerical analyst's experience with one symbol manipulator *SIAM Rev 16* (1974), 129
11. Miller, W Automatic a priori roundoff analysis *Computing 10* (1972), 213–219.
12 Miller W Remarks on the complexity to roundoff analysis *Computing 12* (1974), 149–161.
13 Miller, W Numerical heuristics in computer-aided roundoff analysis IBM Tech. Rep. RC4332, May 2, 1973

14 MILLER, W Numerical search for ill-condition IBM Tech Rep RC4516, 1973.
15 MILLER, W Computer search for numerical instability *J ACM*, to appear
16 MOORE, R *Interval Analysis.* Prentice-Hall, Englewood Cliffs, N J , 1966
17 ORTEGA, J , AND KAISER, H The LL^T and QR methods for symmetric tridiagonal matrices *Computer J. 6* (1963), 99–101
18 REINSCH, C A stable, rational QR algorithm for the computation of the eigenvalues of an Hermitian, tridiagonal matrix *Math Computation 25* (1971), 591–597
19 RICE, J On the conditioning of polynomial and rational forms *Numer. Math 7* (1965), 426–435
20 RICE, J Experiments on Gram-Schmidt orthogonalization *Math Computation 20* (1966), 325–328
21 STEWART, G *Introduction to Matrix Computations.* Academic Press, New York, 1973
22 STOUTEMYER, D Automatic error analysis using the computer symbolic manipulation language, REDUCE. Submitted to a technical journal
23 WELSCH, J. Certification of Algorithm 253. *Comm. ACM 10* (1967), 367
24. WILKINSON, J *Rounding Errors in Algebraic Processes.* Prentice-Hall, Englewood Cliffs, N. J., 1963.
25 WILKINSON, J *The Algebraic Eigenvalue Problem* Clarendon Press, Oxford, England, 1965
26 WILKINSON, J Modern error analysis *SIAM Rev 13* (1971), 548–568

Received July 1974