Testing Linear Algebra Software

Nicholas J. Higham, Department of Mathematics, University of Manchester, Manchester, M13 9PL, England
higham@ma.man.ac.uk, http://www.ma.man.ac.uk/~higham/

Abstract
How can we test the correctness of a computer implementation of an algorithm such as Gaussian elimination, or the QR algorithm for the eigenproblem? This is an important question for program libraries such as LAPACK, that are designed to run on a wide range of systems. We discuss testing based on verifying known backward or forward error properties of the algorithms, with particular reference to the test software in LAPACK. Issues considered include the choice of bound to verify, computation of the backward error, and choice of test matrices. Some examples of bugs in widely used linear algebra software are described.

Keywords
Mathematical software, testing, linear algebra, backward error, LAPACK.

1 INTRODUCTION

How can we test the correctness of a computer implementation of an algorithm in numerical linear algebra? This is an important question for the authors of library software such as LAPACK. A closely related, but different, question is how to measure empirically the stability of an algorithm, independently of any particular implementation in a high-level language. In the former case, which is our interest here, the stability properties of the algorithm are assumed to be at least partly understood, through error analysis; in the latter case, the aim is to supplement or replace a formal error analysis by suitable numerical experiments, and it is assumed that the implementation of the algorithm used in the experiments is correct. In practice, the two questions merge together, because very little numerical software can be guaranteed to be bug-free.

To give a precise definition of "correctness" it would be necessary first to define precisely how an algorithm is represented. An algorithm could, for example, be represented as a pseudocode description to be translated into a target language such as C, Fortran, or MATLAB. Suppose the pseudocode contains the line

$$y = Ax \ (A \in \mathbb{R}^{n\times n},\ x \in \mathbb{R}^n).$$

SRC	353,393
TESTING	308,770
TIMING	104,790
Total	766,953

Table 1 Lines of code in main subdirectories of LAPACK 3.0 distribution.

There are many different ways in which the assignment $y_i = \sum_{j=1}^{n} a_{ij}x_j$ can be evaluated, corresponding to different orderings of the sum; these ways are mathematically equivalent but will, in general, give different answers in finite precision arithmetic. If the pseudocode was, instead,

```
$C = AB$ $(A, B \in \mathbb{R}^{n\times n})$
```

then there is more ambiguity than simply the orderings of sums, because fast matrix multiplication schemes, which use algebraically different formulae from the usual definition of matrix multiplication, could be used. As these observations suggest, a precise definition of the correctness of an implementation of an algorithm is not easy to formulate, and we will not attempt to provide such a definition here.

The magnitude of the task of thoroughly testing linear algebra software is illustrated by a statistic and a quote. Table 1 shows that 40 percent of the source code in the LAPACK 3.0 distribution* is concerned with testing. Dongarra & Stewart (1984, p. 43), two of the LINPACK authors, comment that

> In the development of LINPACK, considerable time and effort were spent in designing and implementing a test package. In some cases, the test programs were harder to design than the programs they tested.

A general approach to testing numerical codes is to define an easily computable function that gives an a posteriori measure of the success of a computation, and to evaluate the function on a selection of problems. The function value is compared with a tolerance that corresponds to some notion of stability. Our main interest here is in testing implementations of well understood algorithms for standard linear algebra computations such as solving linear equation, least squares and eigenproblems. Here the function and tolerance can be chosen to reflect the known stability or accuracy properties of the algorithm. There are two useful ways to apply this approach. One is to evaluate the function on a battery of test problems. The other is to attempt to maximize the function in terms of the problem data, using an optimization routine. This latter technique is investigated in Higham (1993) (see also Higham (1996, Ch. 24)), where direct search methods are shown to be appropriate. The optimization technique is, in general, much more likely to reveal bugs than random tests, but it can be very computationally expensive and, in view of the unpredictable performance of direct search methods, is not easy to automate.

*At the time of writing the current version of LAPACK is LAPACK 2.0 (September 30, 1994) and the release of LAPACK 3.0 is imminent. Unless stated otherwise, all comments about LAPACK in this paper refer to LAPACK 2.0.

We mention briefly some other approaches to testing. Miller and his coauthors (Miller 1975), (Miller & Spooner 1978b), (Miller & Spooner 1978a), (Miller & Wrathall 1980) define a quantity $\sigma(d)$ that bounds, to first order, the sensitivity of an algorithm to perturbations in the data d and in the intermediate quantities that the algorithm generates. They then define the forward stability measure $\rho(d) = \sigma(d)/\kappa(d)$, where $\kappa(d)$ is a condition number for the problem under consideration. The algorithms to be analysed are presented to Miller's Fortran software expressed in a Fortran-like language that allows for-loops but not logical tests (thus Miller's software is unable to test genuine Fortran programs). The software automatically computes the partial derivatives needed to evaluate $\rho(d)$, and attempts to maximize ρ over the data d using the method of alternating directions. This approach is designed for searching for instability in an algorithm rather than testing the correctness of a particular implementation, since the condition number $\kappa(d)$ will be correctly computed only if the encoding of the algorithm is correct.

Rowan (1990) develops another way to search for numerical instability. For an algorithm with data d he maximizes $S(d) = e(d)/\kappa(d)$ using a new direct search maximizer called the subplex method (which is based on the Nelder-Mead simplex method (Nelder & Mead 1965)). Here, $e(d) = y_{\text{acc}} - \widehat{y}$ is an approximation to the forward error in the computed solution $\widehat{y}$, where y_{acc} is a more accurate estimate of the true solution than $\widehat{y}$, and the condition number $\kappa(d)$ is estimated using finite difference approximations. The quantity $S(d)$ is a lower bound on the backward error of the algorithm at d. Fortran software given in Rowan (1990) implements this "functional stability analysis". The software takes as input two user-supplied Fortran subprograms; one implements the algorithm to be tested in single precision, and the other provides a more accurate solution, typically by executing the same algorithm in double precision. The examples in Rowan (1990) show that Rowan's software is capable of detecting numerical instability in a wide variety of (Fortran implementations of) numerical algorithms.

Our concern in this work is with basic issues common to all computing environments. We do not discuss problems arising in parallel implementation of algorithms, in particular, those related to the use of heterogeneous networks of processors (Demmel, Dongarra, Hammarling, Ostrouchov & Stanley 1996).

2 TESTING STABILITY AND ACCURACY

2.1 Backward and Forward Errors

If we have a backward error analysis for an algorithm then a natural way to test a corresponding code is to check that the theoretical error bounds are satisfied. An apparent complication is that most rounding error analyses are derived for one particular sequence of arithmetic operations, without explicit consideration of all the possible orderings of sums, for example. However, many published error bounds are valid for any ordering of the operations, because of majorizations used during the derivation. For those bounds that are ordering dependent, it is usually only the constant terms that are affected and the constants are of limited importance for testing purposes.

The approach we consider, then, requires comparison of the backward error with an a priori backward error bound. Obviously, we must be able to compute the backward error of an arbitrary approximate solution to the problem. For a matrix factorization $A = XYZ \in \mathbb{R}^{n \times n}$ there is a unique matrix ΔA such that

$$A + \Delta A = \widehat{X}\widehat{Y}\widehat{Z}, \qquad (1)$$

where $\widehat{X}$, $\widehat{Y}$ and $\widehat{Z}$ are the computed factors: ΔA is precisely the residual $A - \widehat{X}\widehat{Y}\widehat{Z}$, and its norm is the natural definition of backward error. It may be necessary to verify certain properties of the computed factors, such as closeness to orthogonality. For some problems, however, computing the backward error is a nontrivial task. Perhaps the best example is the least squares problem, as explained below.

A traditional backward error analysis for a factorization of the form (1) typically shows that

$$\|\Delta A\| \le p(n)g(n)u\|A\|, \qquad (2)$$

where u is the unit roundoff, p is a polynomial of third or lower degree, and g is a growth factor (which is present only in factorizations involving non-orthogonal transformations). The norm will depend on the analysis, and is usually the 1-, 2- or ∞-norm. The polynomial p inevitably is the result of many applications of the triangle and submultiplicative inequalities; it causes the error bound to be unattainable and usually to be several orders of magnitude larger than the quantity it is bounding. An often-quoted rule of thumb, which can be supported by statistical arguments, is that a more realistic error bound is obtained by replacing p by its square root (see, for example, Wilkinson (1963, pp. 26, 102)). Even if we take the square root of p, the bound can be pessimistic. For example, in experiments on random matrices reported in the *LINPACK Users' Guide* (Dongarra, Bunch, Moler & Stewart 1979, p. 1.21) it was found that for LU factorization with partial pivoting $\|A - \widehat{L}\widehat{U}\|_1/\|A\|_1$ behaved like a linear function of n. It is important to realize that the lowest available degree of p depends on both the method and the norm (Higham 1996, pp. 177, 206, Problem 10.5):

- For LU factorization, p is cubic for the ∞-norm but only quadratic for the M-norm, $\|A\|_M := \max_{i,j} |a_{ij}|$.
- For Cholesky factorization, p is quadratic for the 2-norm but only linear for the M-norm.

The growth factor term $g(n)$ in (2) is rarely computed by software as a matter of course, and for LU-type factorizations is usually small in practice.

For an a posteriori stability test for a factorization of $A \in \mathbb{R}^{n \times n}$, then, we can use a test of the form

$$\|\Delta A\| \le f(n)u\|A\|,$$

for some "modest" function f. The LAPACK test routines use the 1-norm and take $f(n) = 30n$ for LU factorization with partial pivoting (for both real and complex matrices); the LINPACK test routines also use the 1-norm and take the more conservative choice $f(n) = n$.

For LU-type factorizations the more modern, componentwise style of analysis typically gives, instead of (2), a bound of the form

$$|\Delta A| \le p(n)u|\widehat{X}||\widehat{Y}||\widehat{Z}|, \tag{3}$$

where the absolute value of a matrix, and the matrix inequality, are defined componentwise. In fact, (3) yields a bound of the form (2) on taking norms, the growth factor term $g(n)$ emerging from bounding $|\widehat{X}||\widehat{Y}||\widehat{Z}|$. By their derivation, componentwise bounds involve less triangle and submultiplicative inequalities, so are sharper than normwise bounds, with less pessimistic constant terms. Indeed, the polynomial p in (3) is linear for the LU, Cholesky and symmetric indefinite factorizations (Higham 1996, pp. 175, 206, 223). The componentwise bound (3) has the drawback that it requires a significant amount of extra computation and storage to evaluate.

For linear equation problems $Ax = b$, possibly over- or underdetermined, there are many perturbed systems $(A + \Delta A)x = b + \Delta b$ for which an approximate solution y is an exact solution, and some analysis is required to determine the smallest perturbations, which define the backward error. For $A \in \mathbb{R}^{n\times n}$, the following normwise backward error result is well known (Rigal & Gaches 1967): for any subordinate matrix norm,

$$\begin{aligned}\eta(y) &:= \min\{\, \epsilon : (A+\Delta A)y = b+\Delta b,\ \ \|\Delta A\| \le \epsilon\|A\|,\ \ \|\Delta b\| \le \epsilon\|b\|\,\}\\ &= \frac{\|r\|}{\|A\|\,\|y\| + \|b\|},\end{aligned}$$

where $r = b - Ay$. Similarly, we have the componentwise backward error result (Oettli & Prager 1964)

$$\begin{aligned}\omega(y) &:= \min\{\, \epsilon : (A+\Delta A)y = b+\Delta b,\ \ |\Delta A| \le \epsilon|A|,\ \ |\Delta b| \le \epsilon|b|\,\}\\ &= \max_i \frac{|r_i|}{(|A||y| + |b|)_i}.\end{aligned}$$

Therefore the normwise and componentwise backward errors for a square linear system are easily computed, at the cost of forming a residual. Most standard linear equation solvers are normwise backward stable but not always componentwise backward stable, so it is $\eta(\widehat{x})$ that is the appropriate quantity to test against a multiple of the unit roundoff.

When $A \in \mathbb{R}^{m\times n}$ and $m \ne n$, appropriate definitions of backward error require that y is the *least squares* solution to $(A + \Delta A)y = b + \Delta b$ when $m > n$, or is the *minimum* 2-*norm* solution to $(A + \Delta A)y = b + \Delta b$ when $m < n$. For a long time it was an open problem to find computable expressions for these two backward errors (see, e.g., Stewart (1977)). Recent solutions are given by Waldén, Karlson & Sun (1995) and Sun & Sun (1997) for the case of normwise backward errors; the expressions involve the solution of singular value problems, and hence are relatively expensive to evaluate. It is ironic that while we have known since the mid 1960s that Householder QR factorization provides a backward stable way to solve the linear least squares problem, we have only recently found a way to verify numerically that the computed solution does indeed have a small normwise backward error.

Instead of evaluating the backward error of a computed solution, we could evaluate its forward error: thus for a linear system $Ax = b$, for example, we could compute $\|x - \widehat{x}\|/\|x\|$. This approach requires the computation of an exact, or "sufficiently accurate", solution x, together with the computation or estimation of a condition number, so that the acceptable size of the forward error can be determined. Whether the backward error or forward error should be computed depends on the underlying algorithm. All the linear equation solvers in LAPACK are backward stable (unless large element growth occurs), and so for them it is enough to test the backward error, since a small backward error implies an acceptably small forward error. There are certain algorithms, however, that are forward stable (that is, they have a forward error bounded in the same way as for a backward stable method), but not backward stable, and for these it may be preferable to test the forward error (if a sharp backward error bound, albeit possibly large, is not available, then there is no choice). A simple example of such an algorithm is Cramer's rule for solving a 2×2 system (Higham 1996, §1.10.1).

In connection with testing the forward error, we recall that an old technique for testing linear algebra codes is to generate a problem with known solution and compare the computed and known solutions. For example, a matrix $A \in \mathbb{R}^{n\times n}$ and vector $x \in \mathbb{R}^n$ are chosen and a right-hand side is generated as $b = Ax$; the system is solved to yield a computed $\widehat{x}$ and the relative error $\|x - \widehat{x}\|_\infty/\|x\|_\infty$ is computed. What is often overlooked is that x is not the true solution to the system, because of rounding errors in forming b: we actually compute

$$\widehat{b} = fl(Ax) = Ax + \Delta b, \qquad |\Delta b| \le \gamma_n |A||x|,$$

where $\gamma_n = nu/(1 - nu)$ (Higham 1996, §3.5). Thus

$$|x - A^{-1}\widehat{b}| \le \gamma_n |A^{-1}||A||x|. \tag{4}$$

Now $x - \widehat{x} = (x - A^{-1}\widehat{b}) + (A^{-1}\widehat{b} - \widehat{x})$, so the quantity that we can compute, $x - \widehat{x}$, differs from the true, but unknown, error $A^{-1}\widehat{b} - \widehat{x}$ by at most the bound in (4). If we are testing a normwise backward stable method then we will be testing whether the normwise relative error $\|x - \widehat{x}\|/\|x\|$ is of order $\kappa(A)u$, where the condition number $\kappa(A) = \|A\|\|A^{-1}\|$. In view of the bound (4), we can safely conduct this test using $x - \widehat{x}$. However, in situations where we need several correct significant figures in the computed relative error (for example, when testing a mixed-precision iterative refinement routine), the error in forming b is significant and a sufficiently accurate solution must be obtained by some other means.

For the eigenproblem, definitions of backward error are less clearcut. Consider a Schur decomposition of $A = QTQ^* \in \mathbb{R}^{n\times n}$, where Q is unitary and T is upper triangular, both complex in general. For the computed factors $\widehat{Q}$ and $\widehat{T}$, natural tests are that $\|\widehat{Q}^*\widehat{Q} - I\|$ and

$$\|A - \widehat{Q}\widehat{T}\widehat{Q}^*\|/\|A\| \tag{5}$$

are bounded by appropriate multiples of u. We note the result from Higham (1994) that

$$\frac{\|A^TA - I\|}{\|A\|_2 + 1} \le \min\{\, \|A - U\| : U^TU = I \,\} \le \|A^TA - I\|,$$

where $\|\cdot\|$ denotes any unitarily invariant norm; this result shows that the quantity $\|\widehat{Q}^*\widehat{Q}-I\|$ is an appropriate measure of orthogonality, being within a factor about 2 of the distance to the nearest orthogonal matrix to $\widehat{Q}$ as long as $\|\widehat{Q}\|_2 \approx 1$. Interestingly, although (5) does have a bound of the form $p(n)u$, this is not what the standard error analysis shows. Error analysis for the Schur decomposition is usually presented in the form that says there exists a unitary Q (not necessarily close to $\widehat{Q}$) such that $\|A - Q\widehat{T}Q^*\|/\|A\| \le p(n)u$ (see, e.g., Golub & Van Loan (1989, p. 381)). Correspondingly, an appropriate definition of backward error for $\widehat{T}$ alone is

$$\min\{\, \|A - Q\widehat{T}Q^*\| : Q^*Q = I \,\}.$$

Finding a closed-form expression for this quantity, or an efficient algorithm for computing it, is an open problem.

Testing implementations of iterative algorithms is more difficult than for direct algorithms. These algorithms involve convergence tolerances, which, naturally, must be taken into account by the error tests. The choice of the smallest tolerance to be used must be guided by known stability or accuracy properties of the algorithm—we cannot necessarily expect an iterative linear equation solver to produce a backward error of order u, for example, even for a stationary iterative method (Higham 1996, Ch. 16). A further complicating factor is that an iterative solver could fail to converge in floating point arithmetic on a problem for which it converges in exact arithmetic (for example, the Gauss-Seidel method fails to converge on a certain problem where the iteration matrix has spectral radius 1/2 (Higham 1996, Ch. 16)). For discussions of the choice and influence of stopping criteria for iterative linear equation solvers, see Barrett, Berry, Chan, Demmel, Donato, Dongarra, Eijkhout, Pozo, Romine & van der Vorst (1994, Ch. 4) and Higham (1996, §16.5).

2.2 Test Matrices

The construction of test matrices has occupied numerical analysts since the early days of digital computers, as many papers about test matrices from the 1950s and 1960s testify. Much effort has been spent devising matrices that satisfy one or both of the properties of being "difficult" for linear algebra algorithms to work with (for example, being ill conditioned, or having close or repeated eigenvalues) and of having known inverse, eigenvalues, Jordan form, etc. The classic test matrix, which seems to have an endless fascination for mathematicians (Choi 1983), (Todd 1954), is the Hilbert matrix, with (i,j) element $1/(i+j-1)$. It is well known that the Hilbert matrix is very ill conditioned for even moderate values of n, that the elements of the inverse are integers and are known explicitly, and that the matrix arises in a practical problem: least squares fitting by a polynomial expressed in the monomial basis. What is less well appreciated is that the Hilbert matrix is a poor test matrix, because it is too special: it is symmetric positive definite and totally positive (every submatrix has positive determinant). This means, for example, that Gaussian elimination without pivoting is guaranteed to produce a small componentwise relative backward error. Furthermore, since the Hilbert matrix cannot be exactly represented in floating point arithmetic, the fact that the inverse is known is of less use than it appears.

Specific matrices, such as the Hilbert matrix, are certainly useful to the researcher during the development of theory and algorithms. MATLAB contains a few such matrices and a substantial collection is provided in the Test Matrix Toolbox (Higham 1995b), (Higham 1996, App. E). However, of more general use for the testing of software are random matrices and, particularly for large sparse problems, matrices from applications. A standard technique for constructing random matrices with given eigenvalue or singular value properties is to choose a diagonal or triangular matrix containing the desired eigenvalues or singular values and to pre- and postmultiply by random orthogonal or unitary matrices, or to perform a random similarity transformation. This technique is sometimes used with a random orthogonal matrix constructed by multiplying together a small number of random Householder transformations, but such matrices are low rank corrections to the identity and hence are not completely random. A truly random orthogonal matrix, in an appropriate technical sense, is obtained from the QR factorization of a random matrix from the normal $(0,1)$ distribution (using `qr(randn(n))` in MATLAB, for example), which can be computed more efficiently as a product of random Householder matrices of increasing effective dimension (Stewart 1980). The LAPACK test software constructs test matrices in the way just outlined, as does the routine `randsvd.m` in the Test Matrix Toolbox.

It is important to realise that random matrices have their limitations for testing purposes, in two main respects. First, they tend to reveal average behaviour rather than worst case behaviour. For example, practical experience shows that large growth factors are extremely unlikely to be observed for random matrices. Similarly, poor performance of condition estimators is extremely rare for random matrices. Second, random matrices can sometimes have undesirable structure. For example, random matrices from the uniform $[0,1]$ distribution (`randu(n)` in MATLAB) tend to have one large dominant positive eigenvalue and the other eigenvalues small and of either sign (Andrew 1990).

Large, nonrandom matrices drawn from practical problems are available from a number of sources. Bai (1994) describes a collection of test matrices for large-scale nonsymmetric eigenvalue problems. The Harwell–Boeing collection of sparse matrices is described by Duff, Grimes & Lewis (1989) and Duff, Grimes & Lewis (1992). This collection, together with other test matrices from practical problems, is available over the World Wide Web from the Matrix Market page at URL `http://math.nist.gov/MatrixMarket/`.

3 LAPACK'S TEST SOFTWARE

LAPACK contains a set of programs that thoroughly test all the LAPACK routines. The test programs reside on the path `LAPACK/TESTING` of the LAPACK distribution and are described in detail in Anderson, Dongarra & Ostrouchov (1992).

Consider, first, the linear equation routines. LAPACK tests these routines on a variety of different $n \times n$ matrices chosen, according to the matrix type, from the following list:

- diagonal, upper and lower triangular;
- random with three different 2-norm condition numbers (2, $\sqrt{0.1/u}$ and $0.1/u$);
- first, last, middle or last $n/2$ columns zero; scaled near underflow and overflow;
- random with unspecified condition number;
- block diagonal.

The matrices with zero columns are used to test the error return codes. For each matrix a number of error statistics are computed. Each statistic is tested against a single threshold, whose value, set in a data file, is 30. Warning messages are written out when a ratio exceeds the threshold. In each case $\kappa_1(A)$ denotes a computed condition number, not a condition number estimate, and hats denote computed quantities.

- The matrix is factored using `xyyTRF`[†] and the ratio

  $$\frac{\|A - \widehat{L}\widehat{U}\|_1}{n\|A\|_1 u} \tag{6}$$

  is computed.
- The matrix is inverted using `xyyTRI`, producing $\widehat{X}$, and the ratio[‡]

  $$\frac{\|\widehat{X}A - I\|_1}{n\kappa_1(A)u}$$

  is computed.
- A linear system $Ax = b$ is solved using `xyyTRS` and the ratios

  $$\alpha = \frac{\|b - A\widehat{x}\|_1}{\|A\|_1\|\widehat{x}\|_1 u}, \qquad \beta = \frac{\|x - \widehat{x}\|_1}{\|\widehat{x}\|_1\kappa_1(A)u}, \tag{7}$$

  are computed where x is the "exact solution", that is, the (random) vector used to generate the right-hand side. The reason for testing the forward error is not clear, since, mathematically, if α is less than the tolerance, then β must also be less than the tolerance.
- Iterative refinement (`xyyRFS`) is used to improve the solution computed by `xyyTRS`. For the refined solution $\widehat{x}$, the ratios

  $$\frac{\|x - \widehat{x}\|_1}{\|x\|_1\kappa_1(A)u}, \quad \frac{\eta}{u}, \quad \frac{\|x - \widehat{x}\|_1}{\|x\|_1\xi}, \tag{8}$$

  are computed, where η is the componentwise relative backward error returned by `xyyRFS` and ξ is the forward error bound returned by `xyyRFS` (which is computed with the aid of a condition estimator and hence could, potentially, fail to bound the error).
- The 1-norm condition number κ_1 is computed and the maximum of the ratios $\kappa_1/\widehat{\kappa}_1$ and $\widehat{\kappa}_1/\kappa_1$ is computed, where $\widehat{\kappa}_1$ is the condition estimate computed by `xyyCON`.

The condition number test raises the issue mentioned in §1 of testing the implementation versus testing the underlying algorithm. The condition estimator used in LAPACK, from Higham (1988), provides a lower bound that is nearly always within a factor 3 of the true condition number; however, it can underestimate the true condition number by an

[†]In the LAPACK naming convention `x` denotes the data type, which is one of `S`, `D`, `C`, and `Z`, and `yy` denotes the type of matrix, of which there are 27 in all.

[‡]See §4.3

arbitrary factor. Therefore the threshold in the LAPACK test could be exceeded due to failure of the underlying condition estimation algorithm, though this event is extremely unlikely. One might expect that a poor condition estimate could be produced for another reason: the estimator makes use of an LU (or related) factorization, and if the computed factorization is a poor one because of large element growth, the condition estimate will necessarily be poor. However, the "exact" condition number κ_1 is computed from the same factorization, so both condition numbers always relate to the same, possibly incorrect, matrix and large growth does not necessarily adversely affect the test. Note that the second ratio in (7) and the third ratio in (8) could exceed the threshold when $\widehat{x}$ is computed from an unstable factorization resulting from large element growth; the LAPACK test routines do not take any special precautions for this unlikely event, which would in any case almost certainly be signalled by the ratio (6).

For the least squares solvers for overdetermined systems, LAPACK 2.0 does not compute the backward error of the computed solution. Instead, it checks that the residual is nearly orthogonal to the column space of the coefficient matrix and that, for systems specially constructed to be consistent, the relative residual is small. Theoretically, these tests could both be passed by a routine that is not backward stable. The actual backward error could be computed using the formula from Waldén et al. (1995) in order to produce a more stringent test of the solvers.

For the eigenvalue and singular values routines, LAPACK tests a variety of residuals, such as (5) for the Schur decomposition, and checks that matrices X that would be orthogonal or unitary in exact arithmetic have a sufficiently small value of $\|X^*X - I\|_1$. Again, a variety of test matrices is used, in particular, corresponding to different eigenvalue or singular value distributions.

The LAPACK routines that form random matrices with given singular values, eigenvalues, band structure, and other properties, are located in the directory `LAPACK/TESTING/MATGEN` of the LAPACK distribution. These routines (whose names are of the form `xLAzzz`) are not described in the *LAPACK Users' Guide* (Anderson, Bai, Bischof, Demmel, Dongarra, Du Croz, Greenbaum, Hammarling, McKenney, Ostrouchov & Sorensen 1995), but earlier versions of the routines are described in Demmel & McKenney (1989).

4 CASE HISTORIES

We briefly describe some interesting examples of bugs in widely used linear algebra software.

4.1 Matlab's `rcond`

Matrix condition number estimators attempt to approximate the condition number $\kappa(A) = \|A\|\|A^{-1}\|$ of a nonsingular $A \in \mathbb{R}^{n\times n}$ cheaply, given some factorization of the matrix. They break into two classes: statistically based estimators that use random numbers, and the more frequently used non-statistical estimators such as are employed in LINPACK and LAPACK. For statistically based estimators there may exist results on the distribution of the estimates that can be checked numerically after a sufficiently large number of trials. The non-statistical estimators, however, are based on heuristics that provide no guarantees about the quality of the estimate (indeed, on specially chosen examples both the LINPACK and LAPACK estimators provide estimates that are too small by a factor that can be chosen arbitrarily); this makes these estimators difficult to test, because a poor estimate could be due to an implementation error or a failure in the algorithm itself. As noted in §3, LAPACK tests that the computed and estimated condition numbers do not differ by more than the threshold of 30. This test has apparently never failed for the test matrices used by the software.

MATLAB includes a function `rcond` that produces an upper bound for $\kappa_1(A)^{-1}$; it is a direct translation of the LINPACK condition number estimator. The Release Notes for MATLAB 4.1 state that "This release of MATLAB fixes a bug in the `rcond` function. Previously, `rcond` returned a larger than expected estimate for some matrices ... `rcond` now returns an estimate that matches the value returned by the Fortran LINPACK library." In some experiments on direct search in Higham (1993) we used MATLAB 3.5, and we found it much easier to generate counterexamples to `rcond` (examples in which `rcond` provides an estimate that is much too large) than we do now with MATLAB 4.2. It seems that the maximizations in Higham (1993) were not only defeating the algorithm underlying `rcond`, but also, unbeknown to us, exploiting a bug in the implementation of the function. The conclusions of Higham (1993) are unaffected, however.

4.2 LAPACK's Symmetric Indefinite Factorization

The most popular methods for solving a dense symmetric indefinite linear system $Ax = b$, $A \in \mathbb{R}^{n\times n}$, compute a symmetric indefinite factorization

$$PAP^T = LDL^T,$$

where P is a permutation matrix, L is unit lower triangular and D is block diagonal with diagonal blocks of dimension 1 or 2. LAPACK includes an implementation of the symmetric indefinite factorization with the partial pivoting strategy of Bunch & Kaufman (1977) (routine `xSYTRF`). An unusual feature of the partial pivoting strategy, whose implications for stability were first pointed out by Ashcraft, Grimes & Lewis (1995) and Higham (1995a), is that $\|L\|_\infty/\|A\|_\infty$ can be arbitrarily large, even though the factorization itself is backward stable.

The implementation of the symmetric indefinite factorization with partial pivoting in LAPACK 2.0 can be unstable when $\|L\|_\infty$ is large, as pointed out and explained by Ashcraft et al. (1995). The potential instability stems from replacing a symmetric rank-2 update by two rank-1 updates, via the use of an eigendecomposition, a change that is enough to upset the rather delicate stability of the partial pivoting strategy. This instability has not been detected by the LAPACK test software and so far has been observed only on examples specially constructed by Ashcraft et al. (1995). The problem is corrected in LAPACK 3.0.

4.3 LAPACK's Matrix Inversion Tests

The test software in LAPACK 2.0 tests the general matrix inversion routine by checking that the ratio

$$\frac{\|A\widehat{X} - I\|_1}{n\,u\,\kappa_1(A)} \tag{9}$$

is less than a threshold. However, for the particular algorithm used in LAPACK, this ratio is not bounded independently of A, and it is the ratio

$$\frac{\|\widehat{X}A - I\|_1}{n\,u\,\kappa_1(A)}$$

that is so bounded, as explained in Du Croz & Higham (1992) and Higham (1996, §13.3). This error apparently has not lead to the test being failed in any of the instances when the test software was run, indicating that for the random matrices used the ratio (9) has always been smaller than the threshhold.

Interestingly, precisely the same error is present in the LINPACK test software. Indeed, the LINPACK manual states that the computed inverse $\widehat{X}$ of A from routine `xGEDI` satisfies $\|A\widehat{X} - I\|_1 \le d_n u \|A\|_1 \, \|\widehat{X}\|_1$ (Dongarra et al. 1979, p. 1.20), whereas it is the *left* residual $\|\widehat{X}A - I\|_1$ that is bounded this way, since LINPACK uses the same inversion algorithm as LAPACK.

ACKNOWLEDGEMENTS

This work was supported by Engineering and Physical Sciences Research Council grant GR/H/94528.

REFERENCES

Anderson, E., Bai, Z., Bischof, C. H., Demmel, J. W., Dongarra, J. J., Du Croz, J. J., Greenbaum, A., Hammarling, S. J., McKenney, A., Ostrouchov, S. & Sorensen, D. C. (1995). *LAPACK Users' Guide, Release 2.0*, second edn, Society for Industrial and Applied Mathematics, Philadelphia, PA, USA.

Anderson, E., Dongarra, J. J. & Ostrouchov, S. (1992). Installation guide for LAPACK, *Technical Report CS-92-151*, Department of Computer Science, University of Tennessee, Knoxville, TN, USA. LAPACK Working Note 41.

Andrew, A. L. (1990). Eigenvalues and singular values of certain random matrices, *J. Comp. Appl. Math.* **30**: 165–171.

Ashcraft, C., Grimes, R. G. & Lewis, J. G. (1995). Accurate symmetric indefinite linear equation solvers. Manuscript.

Bai, Z. (1994). A collection of test matrices for large scale nonsymmetric eigenvalue problems (version 1.0). Manuscript.

Barrett, R., Berry, M., Chan, T. F., Demmel, J., Donato, J., Dongarra, J., Eijkhout, V., Pozo, R., Romine, C. & van der Vorst, H. (1994). *Templates for the Solution of Linear Systems: Building Blocks for Iterative Methods*, Society for Industrial and Applied Mathematics, Philadelphia, PA, USA.

Bunch, J. R. & Kaufman, L. (1977). Some stable methods for calculating inertia and solving symmetric linear systems, *Math. Comp.* **31**(137): 163–179.

Choi, M.-D. (1983). Tricks or treats with the Hilbert matrix, *Amer. Math. Monthly* **90**: 301–312.

Demmel, J. W., Dongarra, J. J., Hammarling, S. J., Ostrouchov, S. & Stanley, K. (1996). The dangers of heterogeneous network computing: Heterogeneous networks considered harmful. Manuscript.

Demmel, J. W. & McKenney, A. (1989). A test matrix generation suite, *Preprint MCS-P69-0389*, Mathematics and Computer Science Division, Argonne National Laboratory, Argonne, IL, USA. LAPACK Working Note 9.

Dongarra, J. J., Bunch, J. R., Moler, C. B. & Stewart, G. W. (1979). *LINPACK Users' Guide*, Society for Industrial and Applied Mathematics, Philadelphia, PA, USA.

Dongarra, J. J. & Stewart, G. W. (1984). LINPACK—A package for solving linear systems, *in* W. R. Cowell (ed.), *Sources and Development of Mathematical Software*, Prentice-Hall, Englewood Cliffs, NJ, USA, pp. 20–48.

Du Croz, J. J. & Higham, N. J. (1992). Stability of methods for matrix inversion, *IMA J. Numer. Anal.* **12**: 1–19.

Duff, I. S., Grimes, R. G. & Lewis, J. G. (1989). Sparse matrix test problems, *ACM Trans. Math. Software* **15**(1): 1–14.

Duff, I. S., Grimes, R. G. & Lewis, J. G. (1992). Users' guide for the Harwell–Boeing sparse matrix collection (release 1), *Report RAL-92-086*, Atlas Centre, Rutherford Appleton Laboratory, Didcot, Oxon, UK.

Golub, G. H. & Van Loan, C. F. (1989). *Matrix Computations*, second edn, Johns Hopkins University Press, Baltimore, MD, USA.

Higham, N. J. (1988). FORTRAN codes for estimating the one-norm of a real or complex matrix, with applications to condition estimation (Algorithm 674), *ACM Trans. Math. Software* **14**(4): 381–396.

Higham, N. J. (1993). Optimization by direct search in matrix computations, *SIAM J. Matrix Anal. Appl.* **14**(2): 317–333.

Higham, N. J. (1994). The matrix sign decomposition and its relation to the polar decomposition, *Linear Algebra and Appl.* **212/213**: 3–20.

Higham, N. J. (1995a). Stability of the diagonal pivoting method with partial pivoting, *Numerical Analysis Report No. 265*, Manchester Centre for Computational Mathematics, Manchester, England. To appear in SIAM J. Matrix Anal. Appl.

Higham, N. J. (1995b). The Test Matrix Toolbox for MATLAB (version 3.0), *Numerical Analysis Report No. 276*, Manchester Centre for Computational Mathematics, Manchester, England.

Higham, N. J. (1996). *Accuracy and Stability of Numerical Algorithms*, Society for Industrial and Applied Mathematics, Philadelphia, PA, USA.

Miller, W. (1975). Software for roundoff analysis, *ACM Trans. Math. Software* **1**(2): 108–128.

Miller, W. & Spooner, D. (1978b). Software for roundoff analysis, II, *ACM Trans. Math. Software* **4**(4): 369–387.

Miller, W. & Spooner, D. (1978a). Algorithm 532: Software for roundoff analysis, *ACM*

Trans. Math. Software **4**(4): 388–390.

Miller, W. & Wrathall, C. (1980). *Software for Roundoff Analysis of Matrix Algorithms*, Academic Press, New York.

Nelder, J. A. & Mead, R. (1965). A simplex method for function minimization, *Computer Journal* **7**: 308–313.

Oettli, W. & Prager, W. (1964). Compatibility of approximate solution of linear equations with given error bounds for coefficients and right-hand sides, *Numer. Math.* **6**: 405–409.

Rigal, J. L. & Gaches, J. (1967). On the compatibility of a given solution with the data of a linear system, *J. Assoc. Comput. Mach.* **14**(3): 543–548.

Rowan, T. H. (1990). *Functional Stability Analysis of Numerical Algorithms*, PhD thesis, University of Texas at Austin, Austin, TX, USA.

Stewart, G. W. (1977). Research, development, and LINPACK, *in* J. R. Rice (ed.), *Mathematical Software III*, Academic Press, New York, pp. 1–14.

Stewart, G. W. (1980). The efficient generation of random orthogonal matrices with an application to condition estimators, *SIAM J. Numer. Anal.* **17**(3): 403–409.

Sun, J. & Sun, Z. (1997). Optimal backward perturbation bounds for underdetermined systems, *SIAM J. Matrix Anal. Appl.* To appear.

Todd, J. (1954). The condition of the finite segments of the Hilbert matrix, *in* O. Taussky (ed.), *Contributions to the Solution of Systems of Linear Equations and the Determination of Eigenvalues*, number 39 in *Applied Mathematics Series*, National Bureau of Standards, United States Department of Commerce, Washington, D. C., pp. 109–116.

Waldén, B., Karlson, R. & Sun, J. (1995). Optimal backward perturbation bounds for the linear least squares problem, *Numerical Linear Algebra with Applications* **2**(3): 271–286.

Wilkinson, J. H. (1963). *Rounding Errors in Algebraic Processes*, Notes on Applied Science No. 32, Her Majesty's Stationery Office, London. Also published by Prentice-Hall, Englewood Cliffs, NJ, USA. Reprinted by Dover, New York, 1994.