Optimization by direct search in matrix computations

Higham, Nicholas J.

1993

MIMS EPrint: **2006.167**

ISSN 1749-9097

SIAM J. MATRIX ANAL. APPL.
Vol. 14, No. 2, pp. 317–333, April 1993

001

OPTIMIZATION BY DIRECT SEARCH IN MATRIX COMPUTATIONS*

NICHOLAS J. HIGHAM†

Abstract. A direct search method attempts to maximize a function $f : \mathbb{R}^n \to \mathbb{R}$ using function values only. Many questions about the stability and accuracy of algorithms in matrix computations can be expressed in terms of the maximum value of some easily computable function f. For a variety of algorithms it is shown that direct search is capable of revealing instability or poor performance, even when such failure is difficult to discover using theoretical analysis or numerical tests with random or nonrandom data. Informative numerical examples generated by direct search provide the impetus for further analysis and improvement of an algorithm. The direct search methods used are the method of alternating directions and the multi-directional search method of Dennis and Torczon. The problems examined include the reliability of matrix condition number estimators and the stability of Strassen's fast matrix inversion method.

Key words. optimization, matrix computations, direct search method, numerical stability, Gaussian elimination, matrix condition number estimation, fast matrix multiplication, Vandermonde system, matrix inverse

AMS(MOS) subject classifications. 65F05, 65G05, 65K10

1. Introduction. Is Algorithm X numerically stable? How large can the growth factor be for Gaussian elimination with pivoting strategy P? By how much can condition estimator C underestimate the condition number of a matrix? These types of questions are fundamental in the analysis of algorithms in matrix computations. Usually, one attempts to answer such questions by a combination of theoretical analysis and numerical experiments with random and nonrandom data. In this work we show that a third approach can be a valuable supplement to the first two: phrase the question as an optimization problem and apply a direct search method.

A direct search method for the problem

$$\max_{x \in \mathbb{R}^n} f(x), \qquad f : \mathbb{R}^n \to \mathbb{R} \tag{1.1}$$

is a numerical method that attempts to locate a maximizing point using function values only, and which does not attempt to estimate derivatives of f. Such methods are usually based on heuristics that do not involve assumptions about the function f. Various direct search methods have been developed; for surveys, see [43], [52], and [53]. Most of these methods were developed in the 1960s, in the early days of numerical optimization. For problems in which f is smooth, direct search methods have largely been supplanted by more sophisticated optimization methods that use derivatives (such as quasi-Newton methods and conjugate gradient methods), but they continue to find use in applications where f is not differentiable, or even not continuous. These applications range from chemical analysis [46], where direct search methods have found considerable use, to the determination of drug doses in the treatment of cancer [4]; in both applications the evaluation of f is affected by experimental errors. Lack of smoothness of f, and the difficulty of obtaining derivatives when they exist, are characteristic of the optimization problems we consider here.

* Received by the editors February 25, 1991; accepted for publication (in revised form) October 15, 1991.

† Department of Mathematics, University of Manchester, Manchester, M13 9PL, United Kingdom (na.nhigham@na-net.ornl.gov).

Our aims and techniques can be illustrated using the example of Gaussian elimination (GE). Wilkinson's classic backward error analysis [60] shows that the stability of the process for $A \in \mathbb{R}^{n\times n}$ is determined by the size of the growth factor

$$\rho_n(A) = \frac{\max_{i,j,k} |a_{ij}^{(k)}|}{\max_{i,j} |a_{ij}|},$$

where the $a_{ij}^{(k)}$ are the intermediate elements generated during the elimination. For a given pivoting strategy we would therefore like to know how big $\rho_n(A)$ can be. To obtain an optimization problem of the form (1.1) we let $x = \mathrm{vec}(A) \in \mathbb{R}^{n^2}$, where $\mathrm{vec}(A)$ comprises the columns of A strung out into one long vector, and we define $f(x) = \rho_n(A)$. Then we wish to determine

$$\max_{x\in\mathbb{R}^{n^2}} f(x) \equiv \max_{A\in\mathbb{R}^{n\times n}} \rho_n(A).$$

Suppose, first, that no pivoting is done. Then f is defined and continuous at all points where the elimination does not break down, and it is differentiable except at points where there is a tie for the maximum in the numerator or denominator of the expression defining $\rho_n(A)$. We took $n = 4$ and applied the direct search maximizer MDS (described in §3) to $f(x)$, starting with the identity matrix $A = I_4$. After 11 iterations and 433 function evaluations, the maximizer converged,[1] having located the matrix[2]

$$B = \begin{bmatrix} 1.7846 & -0.2760 & -0.2760 & -0.2760 \\ -3.3848 & 0.7240 & -0.3492 & -0.2760 \\ -0.2760 & -0.2760 & 1.4311 & -0.2760 \\ -0.2760 & -0.2760 & -0.2760 & 0.7240 \end{bmatrix},$$

for which $\rho_4(B) = 1.23 \times 10^5$. (The large growth is a consequence of the submatrix $B(1\colon 3, 1\colon 3)$ being ill conditioned; B itself is well conditioned.) Thus the optimizer readily shows that $\rho_n(A)$ can be very large for GE without pivoting.

Next, consider GE with partial pivoting. Here, at the kth stage of the elimination, rows are interchanged so that $|a_{kk}^{(k)}| \geq |a_{ik}^{(k)}|$, $i = k\colon n$. Now f is defined everywhere but is usually discontinuous when there is a tie in the choice of pivot element, because then an arbitrarily small change in A can alter the pivot sequence. We applied the maximizer MDS to f, this time starting with the orthogonal matrix $A \in \mathbb{R}^{4\times 4}$ with $a_{ij} = (2/\sqrt{2n+1})\sin(2ij\pi/(2n+1))$ [34], for which $\rho_4(A) = 2.32$. After 29 iterations and 1169 function evaluations the maximizer converged to a matrix B with $\rho_4(B) = 5.86$. We used this matrix to start the maximizer AD (described in §3); it took five iterations and 403 function evaluations to converge to the matrix

$$C = \begin{bmatrix} 0.7248 & 0.7510 & 0.5241 & 0.7510 \\ 0.7317 & 0.1889 & 0.0227 & -0.7510 \\ 0.7298 & -0.3756 & 0.1150 & 0.7511 \\ -0.6993 & -0.7444 & 0.6647 & -0.7500 \end{bmatrix},$$

[1] In the optimizations of this section we used the convergence tests described in §3 with tol $= 10^{-3}$.
[2] All numbers quoted are rounded to the number of significant figures shown.

for which $\rho_4(C) = 7.939$. Note that this matrix is not of the form

$$A = \begin{bmatrix} 1 & & & 1 \\ -1 & 1 & & 1 \\ -1 & -1 & 1 & 1 \\ -1 & -1 & -1 & 1 \end{bmatrix}$$

identified by Wilkinson [60] as yielding the maximum possible growth $\rho_n = 2^{n-1}$ for partial pivoting. The whole set of matrices $A \in \mathbb{R}^{n\times n}$ for which $\rho_n(A) = 2^{n-1}$ is described in [34], and C is one of these matrices, modulo the convergence tolerance.

These examples, and others presented below, illustrate the following attractions of using direct search methods to aid the understanding of algorithms in matrix computations.

(1) The simplest possible formulation of optimization problem is often sufficient to yield useful results. Derivatives are not needed, and direct search methods tend to be insensitive to lack of smoothness in the objective function f. Unboundedness of f is a favourable property—direct search methods usually quickly locate large values of f.

(2) Good progress can often be made from simple starting values, such as an identity matrix. However, prior knowledge of the problem may provide a good starting value that can be substantially improved (as in the partial pivoting example).

(3) Usually it is the global maximum of f in (1.1) that is desired (although it is often sufficient to know that f can exceed a specified value). When a direct search method converges it will, in general, at best have located a *local* maximum—and in practice the maximizer may simply have stagnated, particularly if a slack convergence tolerance is used. However, further progress can often be made by *restarting* the same (or a different) maximizer, as in the partial pivoting example. This is because for methods that employ a simplex (such as the MDS method), the behaviour of the method starting at x_0 is determined not just by x_0 but also by the $n+1$ vectors in the initial simplex constructed at x_0.

(4) The numerical information revealed by direct search provides a starting point for further theoretical analysis. For example, the GE experiments above strongly suggest the (well-known) results that $\rho_n(A)$ is unbounded without pivoting and bounded by 2^{n-1} for partial pivoting, and inspection of the numerical data suggests the methods of proof.

When applied to smooth problems the main disadvantages of direct search methods are that they have at best a linear rate of convergence and they are unable to determine the nature of the point at which they terminate (since derivatives are not calculated). These disadvantages are less significant for the problems we consider, where it is not necessary to locate a maximum to high accuracy and objective functions are usually nonsmooth. (Note that these disadvantages are not necessarily shared by methods that implicitly or explicitly *estimate* derivatives using function values, such as methods based on conjugate directions [43], [44]; however, these are not normally regarded as direct search methods.)

The rest of this paper is organized as follows. In §2 we summarize related work and explain what is new about our approach. In §3 we describe the alternating directions (AD) method and the multidirectional search (MDS) method that we have used in this work. All our experiments were done using the interactive package MATLAB [40]. We used 80286 and 80386 PC-compatible machines that are of similar overall speed to a Sun 3/50 workstation. Most of the optimization runs that we describe took less than an hour of computing time.

In §4 we show how direct search methods can provide insight into the performance of matrix condition number estimators, for which the construction of "counterexamples" is usually difficult. In §5 we describe some other, miscellaneous problems in matrix computations that can be successfully explored using direct search. Finally, in §6, we offer some conclusions.

2. Related work. This work was inspired by two papers published in the 1970s by Miller [37], [38] and by recent work of Rowan [47]. Miller's papers describe a way of using a computer to search for numerical instability in algebraic processes, and so to an extent they are concerned with "automatic rounding error analysis." In [37] Miller defines 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. He then defines 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 analyzed are required to contain no loops or conditional branches and are presented to Miller's Fortran software in a special numeric encoding. The software automatically computes the partial derivatives needed to evaluate $\rho(d)$, and attempts to maximize ρ using the method of alternating directions. Miller gives several examples illustrating the scope of his software; he shows, for example, that it can identify the instability of the Gram–Schmidt method for orthogonalizing a set of vectors.

In [38] Miller and Spooner extend the work in [37] in several ways. The algorithm to be analyzed is expressed in a Fortran-like language that allows for-loops but not logical tests. The definition of ρ is generalized, and a method of computing it is developed that involves solving a generalized eigenvalue problem. The book [39] gives a thorough development of the work of [38] and provides further examples of the use of the software. The potential of Miller and Spooner's software for exposing numerical instability is clearly demonstrated in [37], [38], and [39], yet the software has apparently not been widely used. We suspect this is largely due to the inability of the software to analyze algorithms expressed in Fortran, or any other standard language.

A different approach to algorithm analysis is taken in [35], [36]. Here errors are measured in a relative rather than an absolute sense, and the stability is analyzed at *fixed data* instead of attempting to maximize instability over all data; however, the analysis is still linearized.

Statistical modelling of rounding errors in an algorithm has been developed by Chatelin and Brunet, and by Vignes. Their techniques involve randomly perturbing the result of every floating point operation and using statistics to measure the effect on the output of the algorithm; see [9], [10], and the references therein. In [8] Fortran preprocessor tools are developed for implementing the statistical approach described in [9] and the local relative error approach of [36].

We take an approach different from those described above. We note that for many algorithms one can define an *easily computable* function f that gives an a posteriori measure of the degree of success or the stability of the algorithm. Our approach is to try to maximize f over the space of problem data using direct search and any available implementation of the algorithm. This approach has several advantages.

• Any algorithm for which a suitable f can be defined and computed can be tested. There are no constraints on the algorithm or the choice of f.

• When f measures numerical stability the interpretation of the results is straightforward, since f reflects the actual rounding errors sustained, instead of being a bound from a linearized or statistical model of rounding error propagation.

• Existing software implementing the algorithm can be utilized.

In a recent Ph.D. thesis Rowan [47] 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 [41]). 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 [47] 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 [47] show that Rowan's software is capable of detecting numerical instability in a wide variety of numerical algorithms. Rowan also gives two specific examples of the approach we are advocating here: he uses the subplex method to find a matrix for which the LINPACK condition estimator performs poorly (see §4), and to find unit upper triangular matrices R with $|r_{ij}| \le 1$ that maximize $\kappa_1(R)$.

3. Two direct search methods. We have experimented with two direct search methods. The first is the alternating directions (AD) method. Given a starting value x it attempts to solve the problem (1.1) by repeatedly maximizing over each coordinate direction in turn:

repeat
    % One iteration comprises a loop over all components of x.
    for $i = 1\colon n$
        find α such that $f(x + \alpha e_i)$ is maximized (line search)
        set $x \leftarrow x + \alpha e_i$
    end
until converged

AD is one of the simplest of all optimization methods and its fundamental weakness, that it ignores any interactions between the variables, is well known. Despite the poor reputation of AD we have found that it can perform well on the types of problems considered here. In our implementation of AD the line search is done using a crude scheme that begins by evaluating $f(x + he_i)$ with $h = 10^{-4}x_i$ (or $h = 10^{-4}\max(\|x\|_\infty, 1)$ if $x_i = 0$); if $f(x + he_i) \le f(x)$ then the sign of h is reversed. Then if $f(x + he_i) > f(x)$, h is doubled at most 25 times until no further increase in f is obtained. Our convergence test checks for a sufficient relative increase in f between one iteration and the next: convergence is declared when

$$f_k - f_{k-1} \le \text{tol}\,|f_{k-1}|, \tag{3.1}$$

where f_k is the highest function value at the end of the kth iteration. The AD method has the very modest storage requirement of just a single n-vector.

The second method is the multidirectional search method (MDS) of Dennis and Torczon [55], [56]. This method employs a simplex, which is defined by $n + 1$ vectors $\{v_i\}_0^n$ in $\mathbb{R}^n$. One iteration in the case $n = 2$ is represented pictorially in Fig. 3.1, and may be explained as follows.

The initial simplex is $\{v_0, v_1, v_2\}$ and it is assumed that $f(v_0) = \max_i f(v_i)$. The purpose of an iteration is to produce a new simplex at one of whose vertices f exceeds $f(v_0)$. In the first step the vertices v_1 and v_2 are reflected about v_0 along the lines

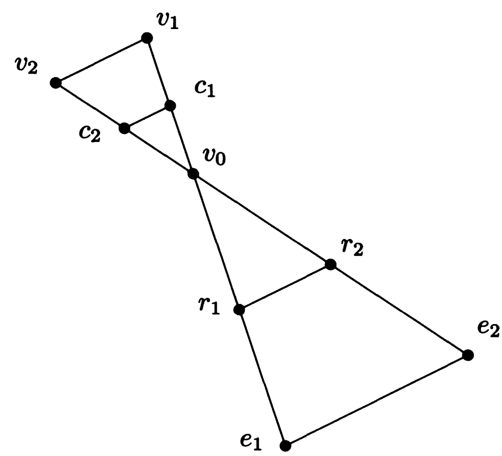

FIG. 3.1. *The possible steps in one iteration of the* MDS *method when* $n = 2$.

joining them to v_0, yielding r_1 and r_2 and the reflected simplex $\{v_0, r_1, r_2\}$. If this reflection step is successful, that is, if $\max_i f(r_i) > f(v_0)$, then the edges from v_0 to r_i are doubled in length to give an expanded simplex $\{v_0, e_1, e_2\}$. The original simplex is then replaced by $\{v_0, e_1, e_2\}$ if $\max_i f(e_i) > \max_i f(r_i)$, and otherwise by $\{v_0, r_1, r_2\}$. If the reflection step is unsuccessful then the edges $v_0 - v_i$ of the original simplex are shrunk to half their length to give the contracted simplex $\{v_0, c_1, c_2\}$. This becomes the new simplex if $\max_i f(c_i) > \max_i f(v_i)$, in which case the current iteration is complete; otherwise the algorithm jumps back to the reflection step, now working with the contracted simplex. For further details of the MDS method, see [15], [55], and [56].

The MDS method requires at least $2n$ independent function evaluations per iteration, which makes it very suitable for parallel implementation. Generalizations of the MDS method that are even more suitable for parallel computation are described in [15]. The MDS method requires $O(n^2)$ elements of storage for the simplices, but this can be reduced to $O(n)$ (at the cost of extra bookkeeping) if an appropriate choice of initial simplex is made [15].

Unusually for a direct search method, the MDS method possesses some convergence theory. Torczon [56] shows that if the level set of f at v_0^0 is compact and f is continuously differentiable on this level set then a subsequence of the points v_0^k (where k denotes the iteration index) converges to a stationary point of f. Moreover, she gives an extension of this result that requires only continuity of f and guarantees convergence to either a stationary point of f or a point where f is not continuously differentiable. No such convergence results are known for the Nelder–Mead direct search method [16], [41], which also employs a simplex but which is fundamentally different from the MDS method. Our limited experiments with the Nelder–Mead method indicate that while it can sometimes outperform the MDS method, the MDS method is generally superior for our purposes.

Our implementation of the MDS method provides two possible starting simplices, both of which include the starting point x_0: a regular one (all sides of equal length) and a right-angled one based on the coordinate axes, both as described in [55]. The scaling is such that each edge of the regular simplex, or each edge of the right-angled simplex that is joined to x_0, has length $\max(\|x_0\|_\infty, 1)$. Also as in [55], the main termination test halts the computation when the relative size of the simplex is no larger than a tolerance tol, that is, when

$$\frac{1}{\max(1, \|v_0\|_1)} \max_{1\leq i\leq n} \|v_i - v_0\|_1 \leq \text{tol}. \tag{3.2}$$

Unless otherwise stated, we used tol $= 10^{-3}$ in (3.1) and (3.2) in all our experiments.

It is interesting to note that the MDS method and our particular implementation of the AD method do not exploit the numerical values of f: their only use of f is to compare two function values to see which is the larger!

Our MATLAB implementations of the AD and MDS methods can be obtained from netlib [18] by sending electronic mail to `netlib@ornl.gov` comprising the message **`send dsmax from matlab/optimization`**.

4. Condition estimators. Condition estimation is the problem of computing an inexpensive but "reliable" estimate of $\kappa(A) = \|A\|\|A^{-1}\|$, for some matrix norm, given a factorization of the nonsingular matrix A. (Other condition numbers of A are also of interest, but we will concentrate on this standard condition number.) The best known condition estimator is the one used in LINPACK [17]; it makes use of an LU factorization of A and works with the 1-norm. Its development is described in [12].[3] Several years after [12] was published several counterexamples to the LINPACK condition estimator were discovered by Cline and Rew [13]; by a counterexample we mean a parametrized matrix for which the quotient "condition estimate divided by true condition number" can be made arbitrarily small (or large, depending on whether the estimator produces a lower bound or an upper bound) by varying a parameter. Despite the existence of these counterexamples the LINPACK estimator has been widely used and is regarded as being almost certain to produce an estimate correct to within a factor ten in practice [27].

Another 1-norm condition estimation algorithm was developed by Higham [29], [30], building on an algorithm of Hager [25]. This estimator is in the NAG library and is being used throughout LAPACK [2]. The general algorithm estimates $\|B\|_1$ given a means for forming matrix-vector products Bx and B^Ty. By taking $B = A^{-1}$ and using an LU factorization of A we obtain an estimator with the same functionality as the LINPACK estimator. Counterexamples to the general algorithm are identified in [29].

A 2-norm condition estimator was developed by Cline, Conn, and Van Loan [11, Algorithm 1]; see also [58]. The algorithm builds on the ideas underlying the LINPACK estimator and estimates $\sigma_{\min}(R) = \|R^{-1}\|_2^{-1}$ or $\sigma_{\max}(R) = \|R\|_2$ for a triangular matrix R. Here, $\sigma_{\min}$ and $\sigma_{\max}$ denote the smallest and largest singular values, respectively. Full matrices can be treated if a factorization $A = QR$ is available (Q orthogonal, R upper triangular), since R and A have the same singular values. The estimator performs extremely well in numerical tests [11], [27], often producing an estimate having some correct digits. No counterexamples to the estimator were known until Bischof [5] obtained counterexamples as a by-product of the analysis of a different, but related, method.

We have experimented with MATLAB implementations of the three condition estimators discussed above. RCOND is the LINPACK estimator as built into MATLAB. SONEST implements the algorithm of [29] as applied to estimating $\kappa_1(A)$. SIGMAN

[3] It is not widely known that a precursor to the LINPACK condition estimator is presented in [24]. I thank G. W. Stewart for pointing this out to me.

is an implementation of the algorithm of [11] for estimating $\sigma_{\min}(R)$, where R is upper triangular.

For RCOND we define $x = \mathrm{vec}(A)$, $A \in \mathbb{R}^{n\times n}$, and

$$f(x) = \frac{\kappa_1(A)}{\mathrm{est}(A)},$$

where $\mathrm{est}(A) \le \kappa_1(A)$ is the condition estimate computed by RCOND. The same definition is used for SONEST, which also computes a lower bound. We note that since the algorithms underlying RCOND and SONEST contain tests and branches, for certain A an arbitrarily small change in A can completely change the condition estimate; hence for both algorithms f has points of discontinuity.

Since SIGMAN was designed for upper triangular matrices $R \in \mathbb{R}^{n\times n}$ we take in this case $x = \overline{\mathrm{vec}}(R)$, where $\overline{\mathrm{vec}}$ is the vec operator modified to skip over elements in the lower triangle of its argument, and we define

$$f(x) = \frac{\mathrm{est}(R)}{\sigma_{\min}(R)},$$

where $\mathrm{est}(R) \ge \sigma_{\min}(R)$ is the estimate.

We applied the MDS maximizer to RCOND starting at $A = I_4$. After 30 iterations and 1009 function evaluations the maximizer had located the matrix

$$A = \begin{bmatrix} 1.1380 & 0.1380 & -2.52\times 10^6 & 0.1380 \\ 0.1380 & 1.1380 & -1.34\times 10^7 & 0.1380 \\ 0.1380 & 0.1380 & 1.1380 & 0.1380 \\ 0.1380 & 0.1380 & 1.59\times 10^7 & 1.1380 \end{bmatrix},$$

for which

$$\kappa_1(A) = 9.88\times 10^{14}, \quad \mathrm{est}(A) = 2.17\times 10^{10}, \quad \frac{\kappa_1(A)}{\mathrm{est}(A)} = 4.56\times 10^4.$$

(For comparison, with the same starting matrix the AD maximizer yielded $f = 18.2$ after six iterations and 950 function evaluations.) This matrix A is badly scaled, and its validity as a counterexample could be questioned on the grounds that for linear equations the condition number $\kappa_1(A)$ is not necessarily an appropriate measure of problem sensitivity when A is badly scaled (see, for example, [22, §3.5.2]). This objection can be overcome by maximizing f subject to a constraint that ensures A is reasonably well scaled. A simple, but effective, constraint is $\kappa_1(A) \le \theta$, where θ is a suitable tolerance. To incorporate this constraint we use a crude penalty function approach in which f is redefined so that $f(x) = -10^{300}$ whenever the constraint is violated. Applying the MDS maximizer with starting matrix $A = \mathrm{diag}(1, -1, 1, -1)$, $\theta = 10^4$, and $\mathrm{tol} = 10^{-9}$ in (3.2), we obtained after 124 iterations and 4625 function evaluations the well-scaled matrix

$$A = \begin{bmatrix} 0.1094 & 0.4559 & 0.1434 & 0.1461 \\ 1.1989 & -0.8617 & 0.1359 & 0.1383 \\ 0.1375 & 2.2531 & 2.2017 & 0.1383 \\ 0.1404 & -2.6932 & 0.1383 & -0.8617 \end{bmatrix},$$

for which

$$\kappa_1(A) = 7.47\times 10^3, \quad \mathrm{est}(A) = 5.37, \quad \frac{\kappa_1(A)}{\mathrm{est}(A)} = 1.39\times 10^3.$$

We note that the parametrized counterexamples in [13] all become badly scaled when the parameter is chosen to make the condition estimate poor. The only previously known well-scaled counterexample to the LINPACK condition estimator is an $n \times n$ lower triangular matrix L in [13] for which $\kappa_1(L)/\mathrm{est}(L) = 2^{n-1}$.

For SONEST, the two maximizers make extremely slow progress starting with $A = I_4$. A better starting value for both maximizers is the 4×4 version of the $n \times n$ matrix with $a_{ij} = \cos((i-1)(j-1)\pi/(n-1))$ [34]. After 11 iterations and 1001 function evaluations the AD maximizer had determined a (well-scaled) matrix A for which

$$\kappa_1(A) = 2.94 \times 10^5, \quad \mathrm{est}(A) = 4.81, \quad \frac{\kappa_1(A)}{\mathrm{est}(A)} = 6.11 \times 10^4.$$

Applying MDS to SIGMAN, starting with $R = I_4$, we obtained after 65 iterations and 1511 function evaluations a matrix R such that

$$\sigma_{\min}(R) = 3.25 \times 10^{-1}, \quad \mathrm{est}(R) = 2.00 \times 10^1, \quad \frac{\mathrm{est}(R)}{\sigma_{\min}(R)} = 6.16 \times 10^1.$$

Using this matrix to start the AD maximizer led after two iterations and a further 93 function evaluations to $\overline{R}$ such that

$$\sigma_{\min}(\overline{R}) = 3.25 \times 10^{-1}, \quad \mathrm{est}(\overline{R}) = 4.31 \times 10^1, \quad \frac{\mathrm{est}(\overline{R})}{\sigma_{\min}(\overline{R})} = 1.33 \times 10^2.$$

These results are surprising. With little effort on our part in the choice of starting matrix the maximizers have discovered examples where each of the condition estimators fails to achieve its objective of producing an estimate correct to within an order of magnitude. Such numerical examples have apparently never been observed in practical computation, or in tests with random matrices such as those in [27]. The value of direct search maximization in this context is clear: it can readily demonstrate the fallibility of a condition estimator—a task that can be extremely difficult to accomplish using theoretical analysis or tests with random matrices. Moreover, the numerical examples obtained from direct search may provide a starting point for the construction of parametrized theoretical ones, or for the improvement of a condition estimation algorithm.

An area of current research in condition estimation is the derivation of algorithms appropriate in applications such as signal processing where a matrix undergoes repeated low rank updates. Several algorithms have been developed [42], [50], but counterexamples to them are not known. Direct search on the appropriate ratio f could provide further insight into these methods.

We note that the direct search approach provides an alternative to the usual way of assessing the quality of condition estimators, which is to examine the quality of the estimates produced for random matrices [27]. One could instead measure the difficulty that an optimizer has in "defeating" a condition estimator—perhaps over a large number of trials with random starting matrices.

As well as measuring the quality of a single algorithm, direct search can be used to compare two competing algorithms, in order to investigate whether one algorithm performs uniformly better than the other. We applied the MDS maximizer to the function

$$f(x) = \frac{\mathrm{estS}(A)}{\mathrm{estR}(A)},$$

where estS(A) and estR(A) are the condition estimates from SONEST and RCOND, respectively. If $f(x) > 1$ then SONEST has produced a larger lower bound for $\kappa_1(A)$ than RCOND. Starting with $A = I_4$, the MDS maximizer converged after 22 iterations to a matrix A for which $\mathrm{estS}(A) = \kappa_1(A)$ and $f(x) = 150.1$. With f defined as $f(x) = \mathrm{estR}(A)/\mathrm{estS}(A)$, and starting with a random matrix $B \in \mathbb{R}^{4\times 4}$, the MDS maximizer converged after 40 iterations to a matrix with $f(x) = 63.90$. This experiment shows that neither estimator is uniformly superior to the other. This conclusion would be onerous to reach by theoretical analysis of the algorithms.

Finally, we use direct search to investigate an open question raised in [27]. If $A\Pi = QR$ is a QR factorization with column pivoting of $A \in \mathbb{R}^{n\times n}$, then [27, Thm. 6.2]

$$\frac{1}{|r_{nn}|} \le \|R^{-1}\|_{1,2} \le \frac{2^{n-1}}{|r_{nn}|}. \tag{4.1}$$

Moreover, there is much experimental evidence to show that $1/|r_{nn}|$ is rarely more than ten times smaller than $\|R^{-1}\|_2$. The question raised in [27] is whether for R from the QR factorization with column pivoting the estimate $\mathrm{est}(R) \le \kappa_1(R)$ produced by the LINPACK estimator has the desirable property that

$$\mathrm{est}(R) \ge \frac{\|R\|_1}{|r_{nn}|}, \tag{4.2}$$

that is, whether the LINPACK estimator always performs at least as well as the trivial lower bound from (4.1). We applied the MDS maximizer to $f(x) = |r_{nn}|^{-1}\|R\|_1/\mathrm{est}(R)$, where $A \in \mathbb{R}^{n\times n}$, $x = \mathrm{vec}(A)$, and $A\Pi = QR$. Starting with $A = I_4$, the maximizer achieved $f(x) = 520.4$ after 67 iterations and 2929 function evaluations. Thus (4.2) is not satisfied—not even to within a reasonable constant factor. However, we have not been able to generate any matrix A for which SIGMAN produces an estimate $\mathrm{est}(R) > |r_{nn}|$ (with $A\Pi = QR$), so it is an open question as to whether $\mathrm{est}(R) \le |r_{nn}|$ always holds for SIGMAN.

5. Other topics. In this section we describe five further topics in matrix computations in which direct search yields interesting results. The first four examples are all concerned with the instability of an algorithm in the presence of rounding errors; here, unlike in the applications considered so far, the objective function depends in an essential way on rounding errors. In our MATLAB computing environment the unit roundoff $u \approx 1.11 \times 10^{-16}$.

5.1. Fast matrix inversion. First, we discuss an example for which there is no existing error analysis, and for which direct search reveals numerical instability. In [51] Strassen gives a method for multiplying two $n\times n$ matrices in $O(n^{\log_2 7})$ operations ($\log_2 7 \approx 2.801$); he also gives a method for inverting an $n \times n$ matrix with the same asymptotic cost. The inversion method is based on the following formulae, where

$$A = \begin{bmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{bmatrix} \in \mathbb{R}^{n\times n}, \quad A_{ij} \in \mathbb{R}^{m\times m}, \quad n = 2m,$$

and $C = A^{-1}$:

$$P_1 = A_{11}^{-1}, \quad P_2 = A_{21}P_1,$$
$$P_3 = P_1A_{12}, \quad P_4 = A_{21}P_3,$$

$$P_5 = P_4 - A_{22}, \quad P_6 = P_5^{-1},$$

$$C = \begin{bmatrix} P_1 - P_3P_6P_2 & P_3P_6 \\ P_6P_2 & -P_6 \end{bmatrix}.$$

(These formulae are easily derived via a block LU factorization of A.) The matrix multiplications are done by Strassen's method and the inversions determining P_1 and P_6 are done by recursive invocations of the method itself. The inversion method is clearly unstable for general A, because the method breaks down if A_{11} is singular. Indeed Strassen's inversion method has been implemented on a Cray-2 in [3] and tested for $n \leq 2048$, and it is observed empirically in [3] that the method has poor numerical stability. Here we use direct search to investigate the numerical stability.

With $x = \mathrm{vec}(A) \in \mathbb{R}^{n^2}$, define the stability measure

$$f(x) = \frac{\min\{\|A\widehat{C} - I\|_\infty, \|\widehat{C}A - I\|_\infty\}}{\|A\|_\infty \|\widehat{C}\|_\infty}, \tag{5.1}$$

where $\widehat{C}$ is the inverse of A computed using Strassen's inversion method. This definition of f is appropriate because, as shown in [19], for most conventional matrix inversion methods either the left residual $\widehat{C}A - I$ or the right residual $A\widehat{C} - I$ is guaranteed to have norm of order $u\|\widehat{C}\|\|A\|$. To treat Strassen's inversion method as favourably as possible we use just one level of recursion; thus P_1 and P_6 are computed using Gaussian elimination with partial pivoting, but the multiplications are done with Strassen's method. We applied the MDS maximizer, with tol $= 10^{-9}$ in (3.2), starting with the 4×4 Vandermonde matrix whose (i, j) element is $((j-1)/3)^{i-1}$. After 34 iterations the maximizer had converged with $f = 0.838$, which represents complete instability. The corresponding matrix A is well conditioned, with $\kappa_2(A) = 82.4$. For comparison, the value of f when A is inverted using Strassen's method with *conventional* multiplication is $f = 6.90 \times 10^{-2}$; this confirms that the instability is not due to the use of fast multiplication techniques—it is inherent in the inversion formulae.

If A is a symmetric positive definite matrix then its leading principal submatrices are no more ill conditioned than the matrix itself, so one might expect Strassen's inversion method to be stable for such matrices. To investigate this possibility we carried out the same maximization as before except we enforced positive definiteness as follows: when the maximizer generates a vector $x \equiv \mathrm{vec}(B)$, A in (5.1) is defined as $A = B^TB$. Starting with a 4×4 random matrix A with $\kappa_2(A) = 6.71 \times 10^7$ the maximization yielded the value $f = 3.32 \times 10^{-8}$ after 15 iterations, and the corresponding value of f when conventional multiplication is used is $f = 6.61 \times 10^{-11}$ (the "maximizing" matrix A has condition number $\kappa_2(A) = 3.58 \times 10^9$).

The conclusion from these experiments is that Strassen's inversion method cannot be guaranteed to produce a small left or right residual even when A is symmetric positive definite and conventional multiplication is used. Hence the method must be regarded as being fundamentally unstable. (The analyses in [14, §2.3] and [19, §2.2] can be used to obtain further insight into this instability.)

5.2. Fast Vandermonde system solvers. In 1970 Björck and Pereyra [7] published two algorithms for solving the Vandermonde systems $Vx = b$ and $V^Ta = f$ in $O(n^2)$ operations, where $V = (\alpha_j^{i-1}) \in \mathbb{R}^{n \times n}$. These algorithms have been used in various applications [1], [48], [57] and generalized in several ways [6], [28], [31], [54], and their numerical stability has been investigated [26], [31], [59]. In [7] it was pointed out that the algorithms sometimes produce surprisingly accurate results, and an explanation for this was given in [26]. However, analysis in [31] predicts that the algorithms can be moderately unstable, and an example of instability is given in [31], together with suggested remedies. The example of [31] appears to be rare since no instances of instability of the Björck–Pereyra algorithms were reported in the first twenty years following their publication. It is therefore interesting to see whether instability can be located by direct search.

Consider the dual system $V^T a = f$ and define the relative residual

$$\begin{aligned} g(x) &\equiv \frac{\|V^T\widehat{a} - f\|_\infty}{\|V^T\|_\infty \|\widehat{a}\|_\infty + \|f\|_\infty} \\ &= \min\{\epsilon : (V + \Delta V)^T \widehat{a} = f + \Delta f, \\ &\qquad \|\Delta V\|_\infty \le \epsilon \|V\|_\infty,\ \|\Delta f\|_\infty \le \epsilon \|f\|_\infty\}, \end{aligned}$$

where $x = [\alpha_1, \ldots, \alpha_n, f_1, \ldots, f_n]^T \in \mathbb{R}^{2n}$ and $\widehat{a}$ is the computed solution from the Björck–Pereyra dual algorithm. The equality above says that the relative residual is equal to the normwise backward error and is well known (see [45]). It is desirable to constrain the points α_i to be distinct and in increasing order (since this ordering is standard and usually helps the numerical stability [31]). To do so we redefine g so that $g(x) = -10^{300}$ if the ordering conditions are not satisfied. We applied direct search to g with $n = 25$, starting with the points α_i equally spaced on $[-1, 1]$ and with $f_i \equiv 1$. Using a combination of MDS and AD maximizer invocations we obtained the value $g(x) = 2.57 \times 10^{-13}$, after a total of approximately 2100 function evaluations.

Thus, given "innocuous" starting values, the maximizers find moderate instability of size three orders of magnitude. The vector produced by the maximization does not represent a pathological problem: the points α_i are approximately equally spaced between -0.996 and 0.588, the vector f has positive elements lying between 0.309 and 2.42, and the Vandermonde matrix V satisfies $\kappa_2(V) = 2.27 \times 10^{10}$. The $n = 25$ problem specified in [31, eq. (6.2)] yields $g(x) = 2.03 \times 10^{-12}$, so the value of g located by direct search is not a global maximum, but it is sufficiently large to reveal instability. We also tried using the problem just mentioned as a starting point for direct search. The AD maximizer increased g to 6.79×10^{-12} in two iterations, but was unable to increase g further.

5.3. Matrix inverse. Numerical analysts universally deprecate the idea of solving a linear system $Ax = b$ by forming $x = A^{-1} \times b$. The reasons are that Gaussian elimination with partial pivoting (GEPP) is generally less expensive and more numerically stable than use of a computed matrix inverse. The difference in computational expense is easily explained and demonstrated (classically, the difference is a factor of 3). The difference in numerical stability is more subtle, and is rarely discussed in the literature, although an excellent analysis is given in [21, §4.7.2]. The instability of the inversion approach is easily demonstrated using direct search. For the linear system $Ay = b$ where $A \in \mathbb{R}^{n\times n}$ let

$$f(x) = \frac{\|b - A\widehat{y}\|_\infty}{\|A\|_\infty \|\widehat{y}\|_\infty + \|b\|_\infty},$$

where A^{-1} is computed via GEPP, $\widehat{y}$ is the computed version of $y = A^{-1} \times b$, and $x \in \mathbb{R}^{n^2+n}$ contains the elements of A and b. We applied the MDS maximizer, with $\mathrm{tol} = 10^{-9}$ in (3.2), taking as initial data the Hilbert matrix of order 4 and the vector of all ones. After 27 iterations and 1741 function evaluations the maximizer converged with $f(x) = 2.91 \times 10^{-9}$. In contrast, with y computed from GEPP, and using the same starting values, the maximizer was unable to drive f above the unit roundoff level 1.11×10^{-16}.

For the matrix inversion method the final A and b found by the maximizer satisfy

$$\kappa_2(A) = 1.05 \times 10^{10}, \quad \|A\|_2\|y\|_2 = 40.2, \quad \|b\|_2 = 2.50;$$

thus b is a right-hand side for which y does not reflect the ill condition of A. As explained in [21], it is for such A and b that the instability of matrix inversion as a means of solving $Ax = b$ is most pronounced.

5.4. Complex matrix multiplication. It is well known that two complex numbers can be multiplied using only three real multiplications. An analogous result holds for matrices. Let $A = A_1 + iA_2, B = B_1 + iB_2$, where $A_i, B_i \in \mathbb{R}^{n\times n}$. If $C = C_1 + iC_2 = AB$, then

$$C_1 = A_1B_1 - A_2B_2, \tag{5.2a}$$

$$C_2 = (A_1 + A_2)(B_1 + B_2) - A_1B_1 - A_2B_2, \tag{5.2b}$$

and these expressions can be evaluated using three real matrix multiplications and five real matrix additions. This yields a saving in arithmetic operations of about 25 percent compared to the usual way of forming C, which involves four real matrix multiplications. However, this alternative method is known to be less stable than conventional multiplication in the sense that the computed imaginary part of the product can be relatively inaccurate [33]. Let

$$f(x) = \frac{\|\widehat{C}_2 - \overline{C}_2\|_\infty}{\|\widehat{C}_2\|_\infty},$$

where $x = [\mathrm{vec}(A_1)^T \ \mathrm{vec}(A_2)^T \ \mathrm{vec}(B_1)^T \ \mathrm{vec}(B_2)^T]^T \in \mathbb{R}^{4n^2}$ and $\widehat{C}_2$ and $\overline{C}_2$ are the computed imaginary parts from conventional multiplication and the formula (5.2b), respectively. A large value for $f(x)$ implies that $\overline{C}_2$ is inaccurate, since we know from standard error analysis that $\widehat{C}_2$ will always be as accurate as can be expected. To test how readily direct search can expose the instability of (5.2a) we applied the MDS maximizer to f with the starting data $A = E + i\alpha E$, $B = E + i\alpha E$, where $E \in \mathbb{R}^{n\times n}$ is the matrix of 1's. With $n = 4$, tol $= 10^{-6}$ in (3.2), and $\alpha = 1$, the maximizer converged after 12 iterations with $f(x) = 3.40 \times 10^{-16}$. With the same n and tol, and $\alpha = 5$, the maximizer converged with $f(x) = 9.99 \times 10^{-11}$ after 24 iterations. Thus, with a little experimentation in the choice of starting value, the instability is easily revealed.

5.5. QR factorization. If $A = QR \in \mathbb{R}^{n\times n}$ is a QR factorization then $\sigma_{\min}(A) = \sigma_{\min}(R) \le \min_i |r_{ii}|$. If column pivoting is used in the QR factorization then this inequality differs from equality by at most a factor 2^{n-1}, as shown by (4.1). But in general the inequality can be arbitrarily weak, as is well known. This is easily confirmed by direct search. Let $f(x) \equiv \min_i |r_{ii}|/\sigma_{\min}(A)$, where $x = \mathrm{vec}(A)$ and $A = QR$. We applied the MDS maximizer followed by the AD maximizer, with tol $= 10^{-3}$ and starting with $A = I_4$, and we obtained $f(x) = 6.58 \times 10^7$ after approximately 930 function evaluations. In fact, the maximizers make rapid progress in increasing f for every starting value we have tried. This is perhaps not surprising, since Foster

[20] gives bounds on the probability that f exceeds β for a class of random matrices, and the probabilities are significant even for large β. For the QR factorization with column pivoting, starting with $A = I_4$, the same maximization produced $f(x) = 2.57$, which is well short of the global maximum $2^{n-1} = 8$.

6. Conclusions. Our experience in using direct search methods has convinced us that they are a useful supplement to the traditional means of analyzing algorithms in matrix computations, such as rounding error analysis and numerical testing with random data. Indeed, tests with random data tend to reveal the average-case behaviour of an algorithm, but the worst-case is also of interest. An underlying theme of this work is that direct search can be vastly more effective than Monte Carlo testing at revealing worst-case behaviour (this is particularly true for the condition estimators of §3).

As we have shown, direct search is sometimes capable of exposing failure of algorithms even when given a trivial starting value such as an identity matrix. An informed choice of starting value coming from partial understanding of the algorithm increases the chance of a revealing optimization. Unsuccessful optimizations can also provide useful information. As Miller and Spooner explain [38, p. 370], "Failure of the maximizer to find large values of ω (say) can be interpreted as providing evidence for stability equivalent to a large amount of practical experience with low-order matrices."

To make use of direct search one has to be able to express the question of interest as an unconstrained optimization problem. As we have shown, this can often be done by employing an appropriate residual or overestimation ratio. If numerical stability is of interest and a suitable objective function cannot be defined then the approach of Rowan [47] is attractive, since it automatically constructs stability estimates given only the ability to execute the algorithm at two different precisions.

Direct search optimization is potentially useful in other areas of numerical analysis besides matrix computations. An experiment in which direct search is used to reveal the fallibility of an adaptive quadrature routine is described in [47]; direct search is used to investigate the accuracy of floating point summation in [32]; and direct search has been used to help tune heuristic parameters in Fortran codes for the numerical solution of ordinary differential equations [49]. We hope that in addition to encouraging researchers in numerical analysis to experiment with direct search optimization, this work will encourage optimization researchers to devote more attention to the rather neglected area of direct search. The multidirectional search method of Dennis and Torczon performed extremely well in our experiments, and alternating directions performed much better than the textbooks might lead one to expect. Parallel direct search methods, such as those in [15], seem particularly attractive for tackling difficult problems such as maximizing the growth factor for Gaussian elimination with complete pivoting [23].

Acknowledgments. I thank Des Higham and Nick Trefethen for their many helpful comments on this work.

REFERENCES

[1] M. Almacany, C. B. Dunham, and J. Williams, *Discrete Chebyshev approximation by interpolating rationals*, IMA J. Numer. Anal., 4 (1984), pp. 467–477.

[2] E. Anderson, Z. Bai, C. H. Bischof, J. W. Demmel, J. J. Dongarra, J. J. Du Croz, A. Greenbaum, S. J. Hammarling, A. McKenney, S. Ostrouchov, and D. C. Sorensen, *LAPACK Users' Guide*, Society for Industrial and Applied Mathematics, Philadelphia, 1992.

[3] D. H. Bailey and H. R. P. Ferguson, *A Strassen–Newton algorithm for high-speed parallelizable matrix inversion*, in Proceedings of Supercomputing '88, IEEE Computer Society Press, New York, 1988, pp. 419–424.

[4] M. C. Berenbaum, *Direct search methods in the optimisation of cancer chemotherapy*, Br. J. Cancer, 61 (1991), pp. 101–109.

[5] C. H. Bischof, *Incremental condition estimation*, SIAM J. Matrix Anal. Appl., 11 (1990), pp. 312–322.

[6] Å. Björck and T. Elfving, *Algorithms for confluent Vandermonde systems*, Numer. Math., 21 (1973), pp. 130–137.

[7] Å. Björck and V. Pereyra, *Solution of Vandermonde systems of equations*, Math. Comp., 24 (1970), pp. 893–903.

[8] B. Bliss, M.-C. Brunet, and E. Gallopoulos, *Automatic program instrumentation with applications in performance and error analysis*, in Expert Systems for Scientific Computing, E. N. Houstis, J. R. Rice, and R. Vichnevetsky, eds., North-Holland, 1992, pp. 235–260.

[9] M.-C. Brunet, *Contribution à la Fiabilité de Logiciels Numériques et à L'analyse de Leur Comportement: Une Approche Statistique*, Ph.D. thesis, U.E.R. Mathematiques de la decision, Université de Paris IX Dauphine, Jan. 1989.

[10] F. Chatelin and M.-C. Brunet, *A probabilistic round-off error propagation model. Application to the eigenvalue problem*, in Reliable Numerical Computation, M. G. Cox and S. J. Hammarling, eds., Oxford University Press, Oxford, U.K., 1990, pp. 139–160.

[11] A. K. Cline, A. R. Conn, and C. F. Van Loan, *Generalizing the LINPACK condition estimator*, in Numerical Analysis, Mexico 1981, J. P. Hennart, ed., Lecture Notes in Mathematics 909, Springer-Verlag, Berlin, 1982, pp. 73–83.

[12] A. K. Cline, C. B. Moler, G. W. Stewart, and J. H. Wilkinson, *An estimate for the condition number of a matrix*, SIAM J. Numer. Anal., 16 (1979), pp. 368–375.

[13] A. K. Cline and R. K. Rew, *A set of counter-examples to three condition number estimators*, SIAM J. Sci. Statist. Comput., 4 (1983), pp. 602–611.

[14] J. W. Demmel and N. J. Higham, *Stability of block algorithms with fast level* 3 *BLAS*, ACM Trans. Math. Software, 18 (1992), pp. 274–291.

[15] J. E. Dennis, Jr. and V. J. Torczon, *Direct search methods on parallel machines*, SIAM J. Optimization, 1 (1991), pp. 448–474.

[16] J. E. Dennis, Jr. and D. J. Woods, *Optimization on microcomputers: The Nelder–Mead simplex algorithm*, in New Computing Environments: Microcomputers in Large-Scale Computing, A. Wouk, ed., Society for Industrial and Applied Mathematics, Philadelphia, 1987, pp. 116–122.

[17] J. J. Dongarra, J. R. Bunch, C. B. Moler, and G. W. Stewart, *LINPACK Users' Guide*, Society for Industrial and Applied Mathematics, Philadelphia, 1979.

[18] J. J. Dongarra and E. Grosse, *Distribution of mathematical software via electronic mail*, Comm. ACM, 30 (1987), pp. 403–407.

[19] J. J. DuCroz and N. J. Higham, *Stability of methods for matrix inversion*, IMA J. Numer. Anal., 12 (1992), pp. 1–19.

[20] L. V. Foster, *The probability of large diagonal elements in the* QR *factorization*, SIAM J. Sci. Statist. Comput., 11 (1990), pp. 531–544.

[21] P. E. Gill, W. Murray, and M. H. Wright, *Numerical Linear Algebra and Optimization, Volume* 1, Addison-Wesley, Reading, MA, 1991.

[22] G. H. Golub and C. F. Van Loan, *Matrix Computations*, 2nd Ed., Johns Hopkins University Press, Baltimore, MD, 1989.

[23] N. I. M. Gould, *On growth in Gaussian elimination with complete pivoting*, SIAM J. Matrix Anal. Appl., 12 (1991), pp. 354–361.

[24] W. B. Gragg and G. W. Stewart, *A stable variant of the secant method for solving nonlinear equations*, SIAM J. Numer. Anal., 13 (1976), pp. 889–903.

[25] W. W. Hager, *Condition estimates*, SIAM J. Sci. Statist. Comput., 5 (1984), pp. 311–316.

[26] N. J. Higham, *Error analysis of the Björck–Pereyra algorithms for solving Vandermonde systems*, Numer. Math., 50 (1987), pp. 613–632.

[27] ———, *A survey of condition number estimation for triangular matrices*, SIAM Rev., 29 (1987), pp. 575–596.

[28] N. J. HIGHAM, *Fast solution of Vandermonde-like systems involving orthogonal polynomials*, IMA J. Numer. Anal., 8 (1988), pp. 473–486.

[29] ———, *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 (1988), pp. 381–396.

[30] N. J. HIGHAM, *Experience with a matrix norm estimator*, SIAM J. Sci. Statist. Comput., 11 (1990), pp. 804–809.

[31] ———, *Stability analysis of algorithms for solving confluent Vandermonde-like systems*, SIAM J. Matrix Anal. Appl., 11 (1990), pp. 23–41.

[32] ———, *The accuracy of floating point summation*, Numerical Analysis Report No. 198, Univ. of Manchester, England, Apr. 1991; SIAM J. Sci. Comput., 14 (1993), to appear.

[33] ———, *Stability of a method for multiplying complex matrices with three real matrix multiplications*, SIAM J. Matrix Anal. Appl., 13 (1992), pp. 681–687.

[34] N. J. HIGHAM AND D. J. HIGHAM, *Large growth factors in Gaussian elimination with pivoting*, SIAM J. Matrix Anal. Appl., 10 (1989), pp. 155–164.

[35] J. L. LARSON, M. E. PASTERNAK, AND J. A. WISNIEWSKI, *Algorithm* 594: *Software for relative error analysis*, ACM Trans. Math. Software, 9 (1983), pp. 125–130.

[36] J. L. LARSON AND A. H. SAMEH, *Algorithms for roundoff error analysis—A relative error approach*, Computing, 24 (1980), pp. 275–297.

[37] W. MILLER, *Software for roundoff analysis*, ACM Trans. Math. Software, 1 (1975), pp. 108–128.

[38] W. MILLER AND D. SPOONER, *Software for roundoff analysis*, II, ACM Trans. Math. Software, 4 (1978), pp. 369–387.

[39] W. MILLER AND C. WRATHALL, *Software for Roundoff Analysis of Matrix Algorithms*, Academic Press, New York, 1980.

[40] C. B. MOLER, J. N. LITTLE, AND S. BANGERT, PC-*Matlab User's Guide*, The MathWorks, Inc., Natick, MA, 1987.

[41] J. A. NELDER AND R. MEAD, *A simplex method for function minimization*, Comput. J., 7 (1965), pp. 308–313.

[42] D. J. PIERCE AND R. J. PLEMMONS, *Fast adaptive condition estimation*, SIAM J. Matrix Anal. Appl., 13 (1992), pp. 274–291.

[43] M. J. D. POWELL, *A survey of numerical methods for unconstrained optimization*, SIAM Rev., 12 (1970), pp. 79–97.

[44] ———, *A view of unconstrained minimization routines that do not require derivatives*, ACM Trans. Math. Software, 1 (1975), pp. 97–107.

[45] J. L. RIGAL AND J. GACHES, *On the compatibility of a given solution with the data of a linear system*, J. Assoc. Comput. Mach., 14 (1967), pp. 543–548.

[46] M. W. ROUTH, P. A. SWARTZ, AND M. B. DENTON, *Performance of the super modified simplex*, Analytical Chemistry, 49 (1977), pp. 1422–1428.

[47] T. H. ROWAN, *Functional Stability Analysis of Numerical Algorithms*, Ph.D. thesis, Univ. of Texas at Austin, May 1990.

[48] L. F. SHAMPINE, I. GLADWELL, AND R. W. BRANKIN, *Reliable solution of special event location problems for* ODE*s*, ACM Trans. Math. Software, 17 (1991), pp. 11–25.

[49] P. W. SHARP, Personal communcation, 1991.

[50] G. M. SHROFF AND C. H. BISCHOF, *Adaptive condition estimation for rank-one updates of* QR *factorizations*, SIAM J. Matrix Anal. Appl., 13 (1992), pp. 1264–1279.

[51] V. STRASSEN, *Gaussian elimination is not optimal*, Numer. Math., 13 (1969), pp. 354–356.

[52] W. H. SWANN, *Direct search methods*, in Numerical Methods for Unconstrained Optimization, W. Murray, ed., Academic Press, New York, 1972, pp. 13–28.

[53] ———, *Constrained optimization by direct search*, in Numerical Methods for Constrained Optimization, P. E. Gill and W. Murray, eds., Academic Press, New York, 1974, pp. 191–217.

[54] W. P. TANG AND G. H. GOLUB, *The block decomposition of a Vandermonde matrix and its applications*, BIT, 21 (1981), pp. 505–517.

[55] V. J. TORCZON, *Multi-directional Search: A Direct Search Algorithm for Parallel Machines*, Ph.D. thesis, Dept. of Mathematical Sciences, Rice University, Houston, TX, May 1989.

[56] ———, *On the convergence of the multidirectional search algorithm*, SIAM J. Optimization, 1 (1991), pp. 123–145.

[57] G. E. TRAPP AND W. SQUIRE, *Solving nonlinear Vandermonde systems*, Comput. J., 18 (1975), pp. 373–374.

[58] C. F. VAN LOAN, *On estimating the condition of eigenvalues and eigenvectors*, Linear Algebra Appl., 88/89 (1987), pp. 715–732.

[59] J. M. VARAH, *Errors and perturbations in Vandermonde systems*, manuscript, 1990.

[60] J. H. WILKINSON, *Error analysis of direct methods of matrix inversion*, J. Assoc. Comput. Mach., 8 (1961), pp. 281–330.