Floating Point Fault Tolerance with Backward Error Assertions

Daniel Boley

Computer Science Dept.
4-192 EE/CSci Building
University of Minnesota
Minneapolis, MN 55455
U.S.A.

Gene H. Golub[*]

Computer Science Dept.
Building 460
Stanford University
Stanford, CA 94305-2140
U.S.A.

Samy Makar
Nirmal Saxena
Edward J. McCluskey[†]

Center for Reliable Computing
Computer Systems Laboratory
Stanford University
Stanford, CA 94305-4055
U.S.A.

ABSTRACT

This paper introduces an assertion scheme based on the *backward error analysis* for error detection in algorithms that solve dense systems of linear equations, $A\mathbf{x} = \mathbf{b}$. Unlike previous methods, this Backward Error Assertion Model is specifically designed to operate in an environment of floating point arithmetic subject to round-off errors, and can be easily instrumented in a Watchdog processor environment. The complexity of verifying assertions is $O(n^2)$ compared to the $O(n^3)$ complexity of algorithms solving $A\mathbf{x} = \mathbf{b}$. Unlike other proposed error detection methods, this assertion model does not require any encoding of the matrix A. Experimental results under various error models are presented to validate the effectiveness of this assertion scheme.

[*]The work of this author was supported in part by the National Science Foundation under Grant NSF CCR-8821078.
[†]The work of this author was supported in part by the Innovative Science and Technology Office of the Strategic Defense Initiative and administered through the Office of Naval Research under Contract No. N0001492-J-1782, and in part by the National Science Foundation under Grant No. MIP-9107760.

1. Introduction

In many applications it has been found that the most effective way to solve problems of very large order is via parallel processing, and the advent of VLSI technology has made possible the construction of computers involving thousands of processors capable of handling such large problems. To take a specific example, the systolic array is a parallel processing paradigm which was pioneered by Kung [11], which is particularly well suited for implementation in VLSI, and which has been particularly successful in signal processing applications (see e.g. [4] [6] [15] [16]). However, individual processors in a processor array can suffer a hard failure or a transient error (an error that may occur only occasionally or irregularly), giving rise to erroneous results which may be difficult to detect. In the realm of floating point matrix computations, where no computation is exact, the difficulty of error detection and correction is exacerbated by the presence of round-off errors. In this paper we use some basic algorithms in linear algebra as a vehicle to demonstrate a new paradigm for detecting and partially correcting faults when using floating point arithmetic. Specifically, we examine algorithms to solve dense systems of linear equations and linear least squares problems. These algorithms can be easily implemented on a systolic array [4] [11] and are fundamental to many methods in signal processing and parameter identification [6]. We illustrate by a simple example how a classical checksum-based approach can completely fail to detect catastrophic errors in a floating point computation, and we propose a new approach specifically designed to detect faults during a floating point matrix computation, capable of guaranteeing the accuracy and integrity of the computed results in the absence of faults, and capable of correcting some errors.

This paper introduces a new way to guarantee the correctness of solutions of sets of linear equations obtained via floating point arithmetic in the presence of transient hardware faults. The specific object is the design of a Watchdog scheme that can guarantee that the solution is correct to the extent that any floating point computation can be guaranteed correct, correct many transient errors, or else signal that an uncorrectable transient fault occurred during the solution process. A Watchdog process is a small process running concurrently with the main computation process that detects errors by monitoring the behavior of the main computation process [14]. If it were possible to carry out the computations in exact arithmetic, then a reversal check [12] could be used to verify that the computed solution does indeed exactly satisfy the original set of equations. But floating point computations always involve some error, normally on the order of the unit round-off of the machine. So it is generally impossible to guarantee the exact correctness of any computed floating point solution down to the last bit, or even that the original equations are satisfied exactly, even when the computation is carried out correctly using the best available algorithm in floating point arithmetic of a given precision. Hence, to detect the presence of errors from hardware faults, it is necessary to distinguish such errors from those that could be caused just by normal round-off errors. Indeed, floating point arithmetic generally precludes the use of a simple reversal check [12, p106]. By using a technique (based on iterative refinement) designed to reduce round-off level errors to a minimum, we can automatically correct many errors, whether from round-off or from hardware faults, that are "small" enough. This method is based on the mathematical bounds that *must* be satisfied by any solution obtained using one of the standard floating point solution algorithms discussed in this paper. The method is guaranteed to detect and signal errors that are too large or catastrophic to be corrected. Hence any solution obtained using this Watchdog system will either be correct to whatever accuracy would normally be expected for a "correct" floating point solution, or an error will be signaled. The reliability of our error detection method depends on the reliable computation of the appropriate bounds and residual vectors. But since these extra computations cost much less than the original solution process, we can afford extra overhead to ensure the correctness of these computations. This is discussed further below.

Most previous approaches for error detection and correction of any computation have been based on the use of checksums. Many papers [8], [10], [13] have been devoted to the study of checksum schemes for algorithms in floating point arithmetic such as Gaussian Elimination on a matrix A. The basic idea of these methods is to extend the matrix A with some additional columns which represent weighted checksums of the matrix rows using different, linearly independent weight vectors. As Gaussian Elimination (**GE**) proceeds by row operations, these checksums are preserved and can be used to detect, and in some cases correct, temporary errors in the elements of the matrix or in the multipliers during the course of the elimination. A description of this scheme can be found in many places (cf. [10]).

It is well known (e.g. [5, sec 2.4]) that floating point arithmetic presents distinctive problems in that computations are no longer exact. Almost all computations suffer from contamination arising out of the round-off error. Hence any method that attempts to detect or correct errors must account for the fact that some error occurs in normal processing in floating point arithmetic. In [17] there is an extensive discussion of the behavior of checksum schemes for detecting or correcting multiple errors when floating point arithmetic is used.

In this paper, we propose another scheme, not based on the checksum approach, to certify the correctness of a solution to a dense set of linear equations when operating in an environment of floating point arithmetic. The basis of our approach is the error analysis for the method used to compute the solution. We check that the *computed* solution satisfies an *a priori* error bound for the particular method used. Suitable *a priori* bounds are given in the Appendix. A preliminary version of this paper appeared as [2].

The rest of this paper is organized as follows. We illustrate the checksum scheme with a simple example, which we then use to show a major weakness of this scheme, as it is commonly defined. We then describe our approach using **GE** with two different pivoting strategies, used to solve sets of linear equations, as well as Orthogonal Triangularization (**QR** Factorization), used to solve linear least squares problems. We illustrate the approach with some numerical experiments and finish with some concluding remarks. In an Appendix we sketch the derivation of the bounds from the backward error analysis of the numerical methods used.

2. What Checksum Schemes Won't Find

We illustrate the checksum scheme with a simple-minded example which we then use to point out a major weakness of this approach. Suppose we want to solve the system $A\mathbf{x} = \mathbf{b}$ in 3 decimal digit rounded arithmetic, where

$$A = \begin{bmatrix} 1.00 & 2.00 \\ 1.00\text{e-}3 & 1.00 \end{bmatrix}; \quad \mathbf{b} = \begin{bmatrix} 3.00 \\ 1.00 \end{bmatrix}.$$

The method we use is **GE** with Partial Pivoting [3] to factor the matrix A as the product of a permutation matrix P, a unit lower triangular matrix of multipliers L and an upper triangular matrix U, to get $A = P^{\mathrm{T}}LU$. At each stage k of **GE** with Partial Pivoting, the k-th column is searched from the diagonal down for the largest entry in magnitude. The row containing this entry then becomes the new Pivot row for this stage. The Pivot row is swapped with the k-th row, and then multiples of it are added to rows $k+1, \ldots, n$ to annihilate all the entries in column k below the diagonal. We remark that Pairwise Pivoting [20] suffers in the same way as Partial Pivoting in this example.

We construct a Checksum matrix

$$H \equiv (I \mid H_c) \equiv \left[\begin{array}{cc|cc} 1 & 0 & 1 & 2 \\ 0 & 1 & 1 & 1 \end{array}\right],$$

where the checksum coefficients H_c are chosen so that any pair of rows are independent, theoretically allowing the detection of up to 2 errors ([8] [10] [13]). (In this case, this trivially reduced to nonsingular.) The row operations are carried out on the extended matrix

$$A_w = AH = (A \mid A_c) = \left[\begin{array}{cc|cc} 1.00 & 2.00 & 3.00 & 4.00 \\ 1.00\text{e-}3 & 1.00 & 1.00 & 1.00 \end{array}\right].$$

In the pivoting algorithm used, no row swap occurs, so that we get a multiplier matrix:

$$L = \begin{bmatrix} 1.00 & 0 \\ 1.00\text{e-}3 & 1.00 \end{bmatrix},$$

and A is overwritten with the extended upper triangular matrix. After rounding to 3 decimal digits, the result is:

$$U_w = (U \mid U_c) = \left[\begin{array}{cc|cc} 1.00 & 2.00 & 3.00 & 4.00 \\ 0 & 1.00 & 1.00 & 1.00 \end{array}\right].$$

To verify that no errors occurred, we form the Checksum Difference Matrix $D = UH_c - U_c$ and note that all all its entries are zero. In this particular case, even in the face of round-off errors, the Difference Matrix D is exactly zero. If we carry out back-substitution on this result, we arrive at the solution $\mathbf{x} = (1.00, 1.00)^{\mathrm{T}}$, which is almost correct to the accuracy shown. The true answer (to 15 digits of accuracy) is $\mathbf{x} = (1.00200400801603, 0.998997995599198)^{\mathrm{T}}$, which when rounded to 3 digits is $\mathbf{x} = (1.00, .999)^{\mathrm{T}}$.

It is generally assumed that temporary errors can occur in the entries of A or among the multipliers in L at any time during the course of the elimination. This checksum scheme will detect and in some cases correct such errors. However, temporary errors could affect intermediate results that are not stored either in A or in L. Such intermediate results are used to determine the order of the rows in pivoting. It has been shown in [17] that errors to the matrix entries may also affect the row order, but such errors can be detected by a checksum scheme, even if correction is precluded by the catastrophic cancellation resulting from the incorrect row ordering. We show by example that errors in intermediate results may also result in incorrect row orderings, giving rise to possible catastrophic cancellation, and that such errors may be completely undetected by the checksum scheme.

Consider the effect if a temporary error occurs in one of the compares during the search for the Pivot row. In our particular example, we end up with a row swap where none was needed. The matrix A becomes

$$\hat{A}_w = \hat{A}H = (\hat{A} \mid \hat{A}_c) = \left[\begin{array}{cc|cc} 1.00\text{e-}3 & 1.00 & 1.00 & 1.00 \\ 1.00 & 2.00 & 3.00 & 4.00 \end{array}\right],$$

and the result of the elimination would be

$$\hat{L} = \begin{bmatrix} 1.00 & 0 \\ 1.00\text{e+}3 & 1.00 \end{bmatrix}$$

(where the permutation representing the row swap has been combined into $\hat{L}$) and

$$\hat{U}_w = (\hat{U} \mid \hat{U}_c) = \left[\begin{array}{cc|cc} 1.00\text{e-}3 & 1.00 & 1.00 & 1.00 \\ 0 & -1.00\text{e+}3 & -1.00\text{e+}3 & -1.00\text{e+}3 \end{array}\right].$$

Again, the Checksum Difference Matrix $\hat{D} = \hat{U}H_c - \hat{U}_c$, computed in the precision of the processor (3 digits), is exactly zero. However, back-substitution on this result yields the solution $\hat{\mathbf{x}} = (0, 1.00)^T$, which has no digits of accuracy at all! We have illustrated that catastrophic loss of accuracy can occur as a result of a temporary error not detected by the checksum scheme. Note that none of this error can be explained by possible ill-conditioning of the problem, since in this case the matrix A has a condition number under 10 (i.e. well-conditioned). This is an extreme example, but it does show that a zero Checksum Difference Matrix may not guarantee the accuracy or correctness of the computed answer in floating point arithmetic.

This may be considered an artificial example, since it is often possible to supplement the checksum-based scheme with other error detection methods such as replicating the computation on different processors, or at least the critical parts of the computation such as pivot row selection. However these methods entail modifications of the solution code itself, perhaps including the sharing of data between the different processors as the computation proceeds. We propose a method that (a) does not depend on any modification of the basic code, and (b) does not preclude the use of any error detection method which does require such modifications.

3. Backward Error Assertion Model

In this section we propose *Backward Error Assertions* as a way to check for errors in the solution of a set of linear equations. In what follows, we use the subscript $_c$ to denote numerically computed quantities, possibly with errors. The vector and matrix norms we use are defined as follows (cf. [5, pp53, pp56-7])

$$\|\mathbf{x}\|_1 \equiv \sum_i |x_i|, \quad \|\mathbf{x}\|_2 \equiv \left(\sum_i |x_i|^2\right)^{1/2}, \quad \|\mathbf{x}\|_\infty \equiv \max_i |x_i|,$$

$$\|A\|_1 \equiv \max_i \sum_j |a_{ij}|, \quad \|A\|_\infty \equiv \max_j \sum_i |a_{ij}|, \quad \|A\|_F \equiv \left(\sum_{i,j} |a_{ij}|^2\right)^{1/2}.$$

We examine three methods: Gaussian Elimination (**GE**) with Partial Pivoting, **GE** with Complete Pivoting, and Orthogonal Triangularization using Householder Transformations (**QR** Factorization). We also examine the addition of iterative refinement. It is well known from the landmark work of J. H. Wilkinson (e.g. [21, pp157-160, pp209-215, p236, pp247-252]) that these methods are all backward stable. That is, if the methods are used to find the solution to $A\mathbf{x} = \mathbf{b}$, the computed solution $\mathbf{x}_c$ will exactly satisfy the approximate system $(A + E)\mathbf{x}_c = \mathbf{b}$, and in each case a bound on the norm of E can be given in terms of the original data and the floating point precision of the processor. It is well known [3] that the relative error in the solution is bounded by $K(A)\cdot\|E\|/\|A + E\|$, where $K(A) \equiv \|A\|\cdot\|A^{-1}\|$ is the *condition number* of A. The accuracy of solutions computed in floating point arithmetic is guaranteed only indirectly through this relationship [3].

Our approach is to use the guarantee as a Backward Error Assertion by checking whether the computed solution meets this guarantee. Regardless of whether or not temporary errors occur, if the solution meets the guarantee, then it is as close to the true solution as the method can make it anyway. In this case, the solution will be just as acceptable as the computed solution that would be obtained in floating point arithmetic with no transient errors.

How does one check that it meets the guarantee? Let $\mathbf{x}_c$ be the computed solution that may have been subject to temporary errors. We can compute its *residual*: $\mathbf{r}_c \equiv A\mathbf{x}_c - \mathbf{b}$. It is easy to show that $\mathbf{x}_c$ must satisfy the approximate equation $(A + E)\mathbf{x}_c = \mathbf{b}$, where

$$E \equiv \frac{\mathbf{r}_c \mathbf{x}_c^{\mathrm{T}}}{\mathbf{x}_c^{\mathrm{T}} \mathbf{x}_c}. \tag{1}$$

Indeed, this is the smallest E in the F-norm for which $\mathbf{x}_c$ will satisfy the approximate equation. Therefore, the computed solution $\mathbf{x}_c$ meets the guarantee if the error matrix E satisfies the *a priori* bound for the particular method.

To describe our validating procedure in detail, we use **GE** with Partial Pivoting as an example. For this method, the basic steps are as follows:

Algorithm 1. Solve $A\mathbf{x} = \mathbf{b}$ for $\mathbf{x}$ with **GE** with partial pivoting and validation of the result.

1. Use **GE** with Partial Pivoting [3] to factor A into $PA \approx L_c U_c$, where L_c, U_c are lower and upper triangular, and P is a permutation. (*cost:* $O(n^3)$)
2. Solve triangular systems $L_c \mathbf{y}_c = P\mathbf{b}$ for $\mathbf{y}_c$ and $U_c \mathbf{x}_c = \mathbf{y}_c$ for $\mathbf{x}_c$. (*cost:* $O(n^2)$)
3. Compute Residual $\mathbf{r}_c \equiv A\mathbf{x}_c - \mathbf{b}$. (*cost:* $O(n^2)$)
4. Use the residual $\mathbf{r}_c$ to check that the matrix E (equation (1)) satisfies the bound (A3) (*cost:* $O(n^2)$):

$$\|E\|_\infty \le \frac{\|\mathbf{r}_c\|_\infty \cdot \|\mathbf{x}_c\|_1}{\mathbf{x}_c^{\mathrm{T}} \mathbf{x}_c} \le g\varepsilon 1.02 \left[n^3 + 2n^2 + \frac{n}{100} \right]. \tag{2}$$

In the above, ε is the "machine epsilon", also known as the unit round-off for the floating point arithmetic. Note that the norm of E in (2) can be computed directly in terms of the norms of $\mathbf{r}_c$ and $\mathbf{x}_c$ (the left-most equality) without explicitly forming E [5, p60]. Also, in the above formula the growth factor g represents the maximum possible value that can occur in a matrix entry during the elimination process. For Partial Pivoting, the maximum growth is

$$g = 2^{n-1} \|A\|_\infty, \tag{3}$$

though Wilkinson [22] points out that it is extremely rare to encounter a matrix with growth greater that

$$g = 8\|A\|_\infty. \tag{4}$$

In those rare cases where this last heuristic bound is exceeded, those cases are exactly the ones in which great improvement in accuracy could be achieved by using one of the other methods mentioned here, which have smaller possible growth factors. So it would be legitimate to use the heuristic bound instead of the hard bound in the Watchdog checking process. In those cases where it fails, it will be either due to temporary errors or (less likely) be one of those rare cases where Partial Pivoting has large growth. In either case, the solution should be re-attempted with one of the other two more robust algorithms.

Steps 1 and 2 describe the basic underlying method for solving a system of linear equations. Steps 3 and 4 together make up the validating procedure. Note that the total cost of the validating procedure is $O(n^2)$, compared with $O(n^3)$ for the basic method. The validating procedure also requires only the original unmodified input data, whose validity can be verified by traditional checksum schemes. These properties are maintained for the other methods described below.

As used in this model, the underlying method (steps 1 and 2) is used with no modifications. It is not a difficult matter to use a different method for this portion of the computation. It is only necessary to replace steps 1 and 2 with the new method and to replace the bound (2) with the new bound appropriate for that method. For **GE** with Complete Pivoting, we replace steps 1 and 2 with this method:

First Update to Algorithm 1. Use complete pivoting instead of partial pivoting.

1′ Factor $PAQ \approx L_c U_c$ where P, L_c, U_c are defined as before, and Q is another permutation. (*cost:* $O(n^3)$)

2′ Solve triangular systems $L_c \mathbf{y}_c = P\mathbf{b}$ for $\mathbf{y}_c$ and $U_c \mathbf{z}_c = \mathbf{y}_c$ for $\mathbf{z}_c$, then form solution $\mathbf{x}_c \equiv Q^{\mathrm{T}} \mathbf{z}_c$. (*cost:* $O(n^2)$)

Then the bound in step 4 will again be (2), but with a different growth factor g [22]:

$$g = 1.8 n^{(1/4)\log n}. \tag{5}$$

We remark again that for both pivoting strategies, g is a bound on the growth in the matrix elements that can occur during the elimination process. If a slight modification to the software in steps 1 and 2 is allowed, one can monitor this growth and if necessary abort if this growth factor is exceeded. In particular, for Partial Pivoting, it is known [3] that at the k-th stage, the maximum possible growth is

$$g = 2^{k-1} \|A\|_\infty,$$

so that a further check can be had by monitoring this growth during the elimination.

Likewise, we can use the method of Orthogonal Triangularization, also known as the **QR** Decomposition (cf. [5], sec 5.2.1). In this method, steps 1 and 2 are replaced by

Second Update to Algorithm 1. Use **QR** instead of **GE**.

1″ Factor $A \approx Q_c R_c$ where Q_c is an orthogonal matrix, and R_c is an upper triangular matrix. (*cost:* $O(n^3)$)

2″ Solve triangular system $R_c \mathbf{x}_c = Q_c^{\mathrm{T}} \mathbf{b}$ for $\mathbf{x}_c$. (*cost:* $O(n^2)$)

Normally Q_c is left as a list of Elementary Reflectors known as Householder Transformations [5, pp195-9] whose product is Q_c. These Householder Transformations are exactly those generated by the method itself. Multiplication by Q_c^{T} is accomplished by applying the individual Householder Transformations in reverse order. In this case, the error bound equivalent to (2) becomes (A9):

$$\|E\|_F \le \frac{\|\mathbf{r}_c\|_2 \cdot \|\mathbf{x}_c\|_2}{\mathbf{x}_c^{\mathrm{T}} \mathbf{x}_c} \le \varepsilon \|A\|_F (1.18 n^2 + 30n). \tag{6}$$

Note that also in this case we can bound the norm of E directly from the norms of $\mathbf{r}_c$ and $\mathbf{x}_c$ without explicitly forming E [5, p60].

4. Iterative Refinement

Iterative refinement is a technique that can be applied to any solution method for systems of linear equations. It can reduce the residual to the minimum possible [3] [5]. It yields finer error detection through a tighter backward error bound, and at the same time yields partial error correction through its iterative convergence property, even with just one iteration. Although computational experience has shown that the accuracy obtained in practice from **GE** with partial pivoting, or with **QR**, is usually quite satisfactory, recent work has shown that the *guaranteed a-priori* bounds on the norm of the residual after just one step of iterative refinement are much tighter than the bounds for the residual from the original solution method. Early work [9] [19] was based on norm-wise bounds, along the lines of the bounds in the previous section. But recent work [1] [7] has adopted a component-wise analysis that in many cases yields much tighter bounds on the sizes of the individual components of the residual vector. In this section, we sketch one of these results that is of particular interest as a backward Error Assertion.

In this paper, if M and N denote two conformal vectors or matrices, we use the notation $|M|$ to denote the vector or matrix obtained by taking the absolute value of each entry of M, and the notation $M \le N$ as meaning that each entry of M is less than or equal to the corresponding entry of N.

One step of iterative refinement can be appended to either **GE** or **QR** algorithms as follows:

Algorithm 2. Solve $A\mathbf{x} = \mathbf{b}$ for $\mathbf{x}$ with a standard matrix factorization and one step of iterative refinement.

```
1. Solve $A\mathbf{x}_c = \mathbf{b}$ for $\mathbf{x}_c$ and the residual $\mathbf{r}_c \equiv A\mathbf{x}_c - b$ using QR or GE.
2. Solve $A\mathbf{e}_c = \mathbf{r}_c$ using the factorization already obtained as part of step 1.
3. Form refined solution $\mathbf{x}_I \equiv \mathbf{x}_c - \mathbf{e}_c$.
4. Form new residual $\mathbf{r}_I \equiv A\mathbf{x}_I - \mathbf{b}$.
```

When **QR** is used, the bound on the residual $\mathbf{r}_I$ is given in [7] and is as follows. Assume A is a dense, nonsingular matrix, and $\mathbf{x}_c$ is obtained using the **QR** decomposition using Householder transformations or Givens rotations. If

$$\varepsilon \| |A| \cdot |A^{-1}| \|_\infty \sigma(A, \mathbf{x}_I) \le f\left(\|A\|_F (1.18n^2 + 30n) \right)^{-1} \tag{7}$$

then

$$|\mathbf{r}_I| \le \frac{2n\varepsilon}{1 - n\varepsilon} |A| \cdot |\mathbf{x}_I| \tag{8}$$

where

$$\sigma(A, \mathbf{x}) = \frac{\max_i (|A| \cdot |\mathbf{x}|)_i}{\min_i (|A| \cdot |\mathbf{x}|)_i} \text{ and } f(t) \approx 2 \frac{(t + n + 2)^2}{1 + n},$$

and $n\varepsilon \le .01$. By using (1), the component-wise bound (8) can be converted into a simple norm bound on the perturbation $E = \Delta A$ independent of $\mathbf{x}_I$:

$$\|E_I\|_F \le \frac{\|\mathbf{r}_I\|_2 \cdot \|\mathbf{x}_I\|_2}{\mathbf{x}_I^{\mathrm{T}} \mathbf{x}_I} = \frac{\|\mathbf{r}_I\|_2}{\|\mathbf{x}_I\|_2} \le \frac{2n\varepsilon}{1 - n\varepsilon} \|A\|_F \tag{9}$$

To interpret this result for the purposes of checking correctness, it is not necessary to check condition (7). If the bound (8) (or alternatively (9)) is satisfied, then the solution is guaranteed to be accurate to the extent possible in floating point arithmetic, whether or not (7) also happens to be satisfied.

If the bound (8) (or alternatively (9)) is not satisfied, then either the hypothesis (7) fails or a hardware fault has occurred. The failure of the hypothesis (7) means that the original system is very ill-conditioned or badly scaled, making it extremely unlikely that any method, even a "correct" one, can yield a solution of high accuracy. If a hardware fault has occurred, then this fault has resulted in a catastrophic loss of accuracy in the solution. In either case, a failure to satisfy the bound (9) implies that the solution cannot be trusted, and an uncorrectable error is signaled. A new attempt to solve the problem is then necessary. If hardware faults are rare enough, the same computer can be used to rule out the hardware as the source of the error. But to rule out the possibility that the failure lies with the conditioning of the underlying system, the system must be solved using extra precision arithmetic, or using an algorithm less sensitive to round-off errors, or possibly by re-formulating the problem to obtain a better scaled system to solve. Since the way such catastrophic errors must be handled depends on the particular application and the accuracy desired, we limit ourselves in this paper to detecting and signaling such catastrophic errors.

We obtain the overall fault-tolerant solution procedure by combining one of the standard matrix factorizations such as **QR** with iterative refinement and backward error validation:

```
Algorithm 3.  Solve $A\mathbf{x} = \mathbf{b}$ with validation of the result and partial fault tolerance.
  1. Use QR to factor $A$ into $A \approx Q_c R_c$.  (cost: $O(n^3)$)
  2. Solve triangular system $R_c \mathbf{x}_c = Q_c^{\mathrm{T}} \mathbf{b}$ for $\mathbf{x}_c$.  (cost: $O(n^2)$)
  3. Compute Residual $\mathbf{r}_c \equiv A\mathbf{x}_c - \mathbf{b}$.  (cost: $O(n^2)$)
  4. Solve triangular system $R_c \mathbf{e}_c = Q_c^{\mathrm{T}} \mathbf{r}_c$ for $\mathbf{e}_c$.  (cost: $O(n^2)$)
  5. Form refined solution $\mathbf{x}_I = \mathbf{x}_c - \mathbf{e}_c$.  (cost: $O(n)$)
  6. Compute Residual $\mathbf{r}_I \equiv A\mathbf{x}_I - \mathbf{b}$.  (cost: $O(n^2)$)
  7. Check that the residual $\mathbf{r}_I$ satisfies the bound (8) or (9). (cost: $O(n^2)$)
```

When used as a Backward Error Assertion, (8) or (9) each yield a simple tight formula that can be checked in $O(n^2)$ operations. In fact, the entire computation of the single step of iterative refinement plus the validation procedure (steps 3-7 above) takes only $O(n^2)$ steps, once the **QR** or **GE** factorizations have been computed (needed just once). We can then check the condition (9). If (9) fails, then either a transient error occurred during the solution, refinement, or validation process, or the original system is too ill-conditioned to admit an accurate solution in finite precision arithmetic. If (9) is satisfied, then the solution $\mathbf{x}_I$ is guaranteed to be as accurate as one can expect from the method. Note that this does not say that no transient error occurred, but it does guarantee that if any error has occurred, it has been corrected silently. Any such error must be located in the lower order bits of a floating point word, so that it mimics a slight loss of precision. Hence, this procedure is fault-tolerant in the sense that many errors are silently corrected, failing only on the more catastrophic errors.

The success of the Backward Error Assertion scheme depends on carrying out the computations of Algorithm 3 in a reliable way even in the presence of transient faults. The Backward Error Assertions (steps 3-7) are intended to detect errors in steps 1 and 2. Faults in the error detection steps could lead to a reduction in the error coverage (fraction of errors detected) or false alarms (errors signaled when none occurred). To avoid these situations, it is necessary to carry out steps 3-7 in a fault tolerant manner. The simplest way to detect errors in steps 3-7 is to replicate the computations. Since steps 3-7 cost only $O(n^2)$, much less than the cost of step 1, the replication will be a minor extra expense. However, a more efficient approach can be designed using forward error bounds for matrix-matrix and matrix-vector multiplications. These bounds lead to effective error detection schemes developed in [18] which are applicable to the computations in steps 3-7.

The robustness of Backward Error Assertions against undetected errors and false alarms depends on the mathematical theory in [7]. The computed solution to the exact set of equations $A\mathbf{x} = \mathbf{b}$ must satisfy exactly the approximate set of equations $(A + E)\mathbf{x} = \mathbf{b}$, where $\|E\|$ is bounded by the backward error bounds (8) or (9). The backward error bounds are tight enough that the exact and approximate set of equations must agree to many digits of accuracy. Hence any solution satisfying the backward error bounds is guaranteed to solve the exact set of equations as well as the correctly computed solution. If the system is also reasonably well conditioned, this also implies that the solution has many digits of accuracy.

5. Numerical Experiments

In order to validate this method, we demonstrate by example that this method will not accept any answer with less accuracy than that obtainable without transient errors. The results also verify the theory that any answer produced without transient errors will be accepted and that many errors can be silently corrected.

As a simple illustration of the Backward Error Assertion Model, we apply the method to the example in Section 1. There are two computed solutions $\mathbf{x} = (1.00, 1.00)^T$ and $\hat{\mathbf{x}} = (0, 1.00)^T$, corresponding to the elimination without and with a row swap, respectively. The two corresponding residuals are $\mathbf{r} = (0, 1e\text{-}3)$, and $\hat{\mathbf{r}} = (1, 0)$, though $\mathbf{r}$ will be exactly zero if computed in the 3 digit arithmetic of the processor itself. The norms of the matrices E and $\hat{E}$ define by (1) are, respectively, 10^{-3} and 1. The right hand side of (2) with (3) is 9.8e-2 indicating that $\mathbf{x}$ is accepted, and $\hat{\mathbf{x}}$ is rejected. Even with the very limited accuracy of 3 digit arithmetic, this method can guarantee accuracy in excess of one digit, without even resorting to the one step of iterative refinement. However, because the bound (2) has the factor n^3, it becomes much too loose a bound as the dimension increases. However, if one step of iterative refinement is applied, the dependence becomes linear in n, so that the guaranteed bound becomes much tighter, even if in practice the unrefined answer already has almost the maximum accuracy. Hence we use iterative refinement to validate this method for larger examples.

To validate the effectiveness of the Backward Error Assertion Model on larger examples, a set of simple prototype numerical experiments was performed. Double precision floating point was used, in which each word is 64 bits long, allocated as follows: the sign bit in bit 63 (leftmost), the exponent in bits 62-52, the mantissa in bits 51-0.

In the experiments, a matrix with random elements uniformly distributed in the interval $(-1,1)$ was chosen and factored into $A = QR$ using **QR**. Then certain entries in the Q and R were chosen at random, and errors injected into a bit of each chosen entry. Then the perturbed QR factors were used to solve the system followed by one step of iterative refinement. The residual obtained after the one step of iterative refinement was then compared to the bound (8). An error was *silently corrected* if the residual satisfied the theoretical bound (8). Answers that were silently corrected in this way were *accepted.* An error was *detected and signaled* if the residual exceeded the theoretical bound (8). In these tests, we used two error injection models: the Single Error Model, in which errors were injected into a single, randomly chosen, element of Q or R at a time, and the Multiple Error Model, in which several errors were injected into different randomly chosen elements. Prototype tests, shown in Figure 1, were carried out with randomly generated well-conditioned 50×50 matrices, and in the case of the Multiple Error Model, with five errors. The tests show that errors injected in the lower 30 bits were almost always silently corrected, and multiple errors injected in the upper 25 bits were almost always detected and signaled without being corrected. Single errors in the upper half were about half detected and half silently corrected. A single error is a rank one change, which can often be completely corrected in one step of iterative refinement. The maximum relative error in any solution that was accepted in this test was 7.3122e-13, meaning all the accepted answers had at least 12 digits of accuracy.

In order to validate the method under increasing ill-conditioning, we ran a second set of tests shown in Figures 2 and 3, with the Single Error Model, using randomly chosen bits, against a range of condition numbers. In Figure 2 we show the percentage of solutions that were silently corrected versus the condition number of the generated matrix. In this test we used the Single Error Model with randomly chosen bits. In Figure 3, we show the maximum relative error in any accepted solution (solid line). This is compared with the theoretical bound on the relative error in the solution (dashed line in Figure 3), given by the formula [5, p82], assuming $\delta \cdot \mathrm{K}(A) < 1$:

$$\frac{\|\Delta \mathbf{x}\|}{\|\mathbf{x}\|} \le \frac{2\delta \cdot \mathrm{K}(A)}{1 - \delta \cdot \mathrm{K}(A)},$$

where $\mathrm{K}(A) = \|A\| \cdot \|A^{-1}\|$ is the condition number of A, and $\delta \equiv \|\Delta A\| / \|A\|$ is the maximum relative error in the matrix coefficients. Since we can model residuals either as errors in the matrix using (1) or in the right hand side, we have chosen to simplify the analysis of [5] by choosing the matrix error model. If no transient errors occur, then $\delta \cdot \|A\|$ will be bounded by (2), (6), (9) as appropriate, in any case $O(\varepsilon)$. Figure 3 shows that the theoretical bound tends to be pessimistic by a small constant factor, so that any solution satisfying it should be as satisfactory as the solution obtained with no errors. Note that when the condition number exceeded 10^{12} , the assumption $\delta \cdot \mathrm{K}(A) < 1$ failed, so there is theoretically no bound.

Finally, we ran a third set of experiments shown in Figure 4 to validate the same method on Gaussian Elimination in which incorrect pivot rows can be selected. This was the basis for the example in section 2 on which the checksum scheme failed to detect an error. We used two sets of sample matrices: one set chosen with entries of the form 100^{ν}, where ν was a normally distributed random variable with mean 0 and variance 1, and the other set chosen with random entries uniformly distributed in [0,1]. The matrices in the first set were chosen to be badly scaled, since the entries range over several orders of magnitude from 10^{-5} to 10^{+3} with most entries on the order of unity. This was chosen so that if the wrong pivot rows are selected, rather severe cancellation would occur. Unfortunately, one side effect of this was that the matrices were relatively badly conditioned with a condition number on the order of 3e+6. But even so, the results from Figure 3 show that we can expect at least 7 (and probably at least 10 in practice) digits of accuracy in the final answer. The matrices in the second set were chosen to be similar to the those in the earlier experiments, and had relatively low condition numbers on the order of 3e+2. A set of simple experiments with the Single Error Model was conducted using Gaussian Elimination. To emulate incorrect choices in pivot row selection in a simple way, we turned off pivoting entirely. The results in Figure 4 show that even with this crude strategy, errors in the lower order bits were often corrected in spite of poor choice of pivot rows. For the badly scaled cases, bit errors even in the low order bits very often propagated through the matrix resulting in errors too large to be corrected. On the other hand, the ill-conditioning resulted in a somewhat looser Backward Error Assertion bound, guaranteeing only 7 to 10 digits of accuracy instead of over 12. But even though the bound is tight enough to guarantee substantial accuracy, it still tends to be conservative in practice, as illustrated by the fact that the maximum error in any accepted solution in this experiment was under 10^{-12}, and many solutions flagged as erroneous were close enough that a second round of iterative refinement would have been sufficient to reach an acceptable solution. This simple experiment is sufficient to show that incorrect pivoting can often lead to more catastrophic errors than isolated bit errors. However, the Backward Error Assertion bounds depend only on the original data and are independent of the pivoting strategy used or the actual pivot rows selected.

6. Conclusions

We have shown by example that the checksum scheme may fail to detect certain errors, and in some cases even catastrophic errors. We then proposed the methodology of Backward Error Assertions, based on the backward error analysis to verify the correctness of numerical results. We applied this methodology to three different numerical methods for the solution of systems of linear equations. The numerical experiments showed that the Backward Error Assertion Model is effective in detecting errors that other schemes might not detect. The results show the validity of this overall approach. The Backward Error Assertion Model will detect errors that have the greatest effect on the accuracy of the final result, namely errors in the high order part of the floating point word. Errors in the lower order part of the word will generally be silently corrected. If two or more steps of iterative refinement were used, more errors might be correctable, allowing faster recovery from errors. But some catastrophic errors will never be correctable, so there is always the chance the whole computation will need to be repeated. The Backward Error Assertion Model can easily be combined with the techniques of [18] to achieve error detection capability on the entire computation including the Watchdog process.

We contrast this to the traditional approach where errors are corrected by first detecting and locating them, and then applying an explicit correction. In floating point arithmetic, errors may be so catastrophic that they may be uncorrectable, and only these errors are signaled. Iterative refinement silently corrects many errors automatically. Hence in the Backward Error Assertion Model, certain errors will be silently corrected without being explicitly detected, and other errors will be detected and signaled without being corrected. In addition, this model allows the simultaneous use of other traditional assertion schemes, such as a checksum method, at no extra cost other than the separate costs of the methods used. These methods may be used to detect some hardware faults (even those corrected by the Backward Error Assertion Method) occurring in the floating point computation, but they may also miss some errors as illustrated in section 2.

The Backward Error Assertion Model can be easily implemented in a parallel environment. In fact, the basic solver process for the set of linear equations can be implemented in whatever way is most appropriate, such as a systolic array, without regard to the Backward Error Assertions. The Assertions can be implemented in a separate Watchdog processor operating independent of and parallel to the main processor(s), or as a postprocessor to the basic solver. The Watchdog processor requires access only to the original input data and the computed solution, but no other intermediate result. The main computation would proceed without any degradation from the Watchdog processor, unless an error is signaled. The basic tasks in the Assertion processor are Matrix Vector Products and Back-substitutions, which are both parallelizable in their own right [5]. In addition, the use of Backward Error Assertions does not preclude the use of any other error detection or correction scheme, such as further use of iterative refinement, replication of the backward error computation, or a checksum based scheme. Further study will show the effectiveness of further steps of iterative refinement in correcting more errors, but since there could always be catastrophic uncorrectable errors, one must always allow for the necessity to repeat the computation from scratch. Under the assumption that such errors will be relatively rare, the extra cost of repeating the entire computation occasionally will be modest.

REFERENCES

[1] **M. Arioli, J. Demmel, I. S. Duff,** Solving Sparse Linear Systems with Sparse Backward Error, *SIAM J. Matr. Anal.* 10, pp. 165-190, 1989.

[2] **D. L. Boley, G. H. Golub, S. Makar, N. Saxena, E. J. McCluskey,** *Backward Error Assertions for Checking Solutions to Systems of Linear Equations*, Stanford Univ. Numerical Analysis Project, report NA-89-12, November 1989.

[3] **G. Forsythe, C. Moler,** *Computer Solution of Linear Algebraic Systems*, Prentice Hall, 1967.

[4] **W. M. Gentleman, H. T. Kung,** Matrix Triangularization by Systolic Arrays, in *Proc. SPIE* 298, Real-Time Signal Processing IV, pp. 298-303, 1981.

[5] **G. H. Golub, C. Van Loan,** *Matrix Computations*, 2nd ed., Johns Hopkins, 1989.

[6] **S. Haykin,** *Adaptive Filter Theory* (2nd ed.), Prentice Hall, 1991.

[7] **N. J. Higham,** Iterative Refinement Enhances the Stability of QR Factorization Methods for Solving Linear Equations, *BIT* 31, pp. 447-468, 1991.

[8] **K. H. Huang, J. A. Abraham,** Algorithm-based fault tolerance for matrix operations, *IEEE Trans. Comput.* C-33 #6, pp. 518-528, June 1984.

[9] **M. Jankowski, H. Wozniakowski,** Iterative Refinement Implies Numerical Stability, *BIT* 17, pp. 303-311, 1977.

[10] **J. Y. Jou, J. A. Abraham,** Fault-tolerant matrix arithmetic and signal processing on highly concurrent computing structures, *Proc. IEEE* 74 #5, Special Issue on Fault Tolerance, pp. 732-741, May 1986.

[11] **H. T. Kung, C. E. Leiserson,** Systolic Arrays (for VLSI), in *Sparse Matrix Proceedings 1978* (I. S. Duff and G. W. Stewart ed.), pp. 256-282, SIAM, Philadelphia, 1979,.

[12] **P. A. Lee, T. Anderson,** *Fault Tolerance, Principles and Practice* (2nd ed.), Springer Verlag, 1990.

[13] **F. T. Luk, H. Park,** An analysis of algorithm-based fault tolerance, *J. Parallel Distr. Comput.* 5, pp. 172-84, 1988.

[14] **A. Mahmood, E. J. McCluskey,** Concurrent Error Detection using Watchdog Processors - a Survey, *IEEE Trans. Comput.* 37 #2, pp. 160-174, 1988.

[15] **J. G. McWhirter,** Recursive Least-Squares Minimization using a Systolic Array, in *Proc. SPIE* 431, Real-Time Signal Processing VI, pp. 105-112, 1983.

[16] **J. G. McWhirter,** Algorithmic Engineering -- an Emerging Discipline, in *Proc. SPIE* 1152, Advanced Algorithms and Architectures for Signal Processing IV (F. T. Luk ed.), pp. 2-15, 1989.

[17] **H. Park,** On multiple error correction in matrix triangularizations using checksum schemes, *J. Parallel Distr. Comput.* 14, pp. 90-97, 1992.

[18] **A. Roy-Chowdhury, P. Banerjee,** Tolerance Determination for Algorithm Based Checks using Simple Error Analysis Techniques, in *Fault Tolerant Computing Symp. FTCS-23*, pp. 290-298, IEEE Press, 1993.

[19] **R. D. Skeel,** Iterative Refinement Implies Numerical Stability for Gaussian Elimination, *Math. Comp.* 35, pp. 817-832, 1980.

[20] **D. C. Sorensen,** Analysis of pairwise pivoting in Gaussian elimination, *IEEE Trans. Comput.* C-34, pp. 274-278, 1985.

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

[22] **J. H. Wilkinson,** Error analysis of direct methods of matrix inversion, *J. A. C. M.* 8, pp. 281-330, 1961.

Appendix

In this appendix we outline the derivation for the error bounds used for the three methods we have considered in this paper. In the case of **GE** with Pivoting, it is a classical result [3] that once one knows what the row/column interchanges will be, one can carry out all those interchanges and then carry out all the row operations that make up the elimination itself. Thus, for the purpose of this error analysis, we can assume that A has already been permuted into the right order so that no further permutation is necessary. The following analysis is well-known and comes from [3]. To solve a set of linear equations using **GE**, we use three steps: (a) factor $A = LU$, (b) solve $L\mathbf{y} = \mathbf{b}$, (c) solve $U\mathbf{x} = \mathbf{y}$. In floating point arithmetic, what we compute are the approximate factors L_c, U_c and approximate solutions $\mathbf{y}_c$, $\mathbf{x}_c$, which satisfy [3]

$$L_c U_c = A + E_1,\ (L_c + E_2)\mathbf{y}_c = \mathbf{b},\ (U_c + E_3)\mathbf{x}_c = \mathbf{y}_c \qquad \text{(A1)}$$

where the error perturbation matrices satisfy the bounds

$$\|E_1\|_\infty \le g\varepsilon n^2,\ \|E_2\|_\infty \le g\varepsilon \frac{n(n+1)}{2}1.01,\ \|E_3\|_\infty \le \varepsilon \frac{n(n+1)}{2}1.01 \tag{A2}$$

where g is the "growth factor" (the maximum element that ever occurs during the elimination), n is the dimension of the system, and ε is the "machine epsilon", otherwise known as the unit round-off error. Throughout this whole analysis, we make the implicit assumption that $n\varepsilon \le 0.01$. The factor 1.01 in (A2) can reduced closer to 1 by reducing this implicit bound on $n\varepsilon$. The final solution $\mathbf{x}_c$ satisfies from (A1)

$$(A + E_{LU})\mathbf{x}_c \equiv (A + E_1 + L_c E_3 + E_2 U_c + E_2 E_3)\mathbf{x}_c = \mathbf{b},$$

and E_{LU} can be bounded from (A2) by

$$\|E_{LU}\|_\infty \le g\varepsilon 1.02\left[n^3 + 2n^2 + \frac{n}{100}\right] \tag{A3}$$

From [3], [22], *a priori* bounds for g are given for Complete Pivoting by (5), and for Partial Pivoting by (3), though in this last case g is almost always bounded by (4) [22], as already mentioned above.

We can carry out a similar analysis for the Orthogonal Triangularization method, otherwise known as the **QR** Decomposition. To solve $A\mathbf{x} = \mathbf{b}$, we factor $A = QR$, where Q is orthogonal, and solve the system $R\mathbf{x} = \mathbf{y} \equiv Q^{\mathrm{T}}\mathbf{b}$. In floating point arithmetic, we actually compute [21, pp157-160, p236, p250] the approximate factorization $Q_c R_c$ and approximate vectors $\mathbf{y}_c$, $\mathbf{x}_c$ which satisfy:

$$Q_c R_c = A + E_4,\ \mathbf{y}_c = P(\mathbf{b} + \mathbf{e}_5),\ (R_c + E_6)\mathbf{x}_c = \mathbf{y}_c,$$

where P is a true orthogonal matrix close to Q^{T}. Assuming Q is left as a product of Householder Transformations, we have the following bounds [21, pp157-160, p236, p250]:

$$\|E_4\|_F \le 12.36(n-1)(1 + 12.36\varepsilon)^{n-2}\varepsilon\|A\|_F,$$

$$\|\mathbf{e}_5\|_2 \le 12.36(n-1)(1 + 12.36\varepsilon)^{n-2}\varepsilon\|\mathbf{b}\|_2,$$

$$\|E_6\|_F \le n(n-1)1.01\varepsilon\|R\|_F.$$

where $\|R\|_F = \|A\|_F$. This bound is not derived directly from (A2), but rather from a bound on the individual entries in E_6 in back-substitution found in [3].

Using the bound $(1 + 12.36\varepsilon)^{n-2} \le 1.1316$, valid if $n\varepsilon \le 0.01$, we rewrite the above bounds as

$$\|E_4\|_F \le 14(n-1)\varepsilon\|A\|_F$$

$$\|\mathbf{e}_5\|_2 \le 14(n-1)\varepsilon\|\mathbf{b}\|_2 \le 0.14\|\mathbf{b}\|_2, \tag{A4}$$

$$\|E_4\|_F + \|E_6\|_F \le \varepsilon\|A\|_F(1.16n^2 + 13n). \tag{A5}$$

As in the case of **GE**, we combine the above relations to find that $\mathbf{x}_c$ exactly satisfies

$$(P(A+E_4)+E_6)\mathbf{x}_c = P(\mathbf{b}+\mathbf{e}_5). \tag{A6}$$

We would like to reduce this to the form $(A+E)\mathbf{x}_c = \mathbf{b}$, putting all the perturbation into the coefficients A. We can do this by rewriting (A6) as

$$(PA+E_{QR})\mathbf{x}_c \equiv \left(PA+PE_4+E_6+P\mathbf{e}_5\frac{\mathbf{x}_c^{\mathrm{T}}}{\mathbf{x}_c^{\mathrm{T}}\mathbf{x}_c}\right)\mathbf{x}_c = P\mathbf{b}. \tag{A7}$$

Take norms and combine (A4), (A5):

$$\|E_{QR}\|_F \le \varepsilon\left(\|A\|_F(1.16n^2+13n)+14n\frac{\|\mathbf{b}\|_2}{\|\mathbf{x}_c\|_2}\right). \tag{A8}$$

We now need a lower bound on $\|\mathbf{x}_c\|_2$. To obtain this, take norms in (A6) and use (A4):

$$(\|A\|_F+\|E_4\|_F+\|E_6\|_F)\|\mathbf{x}_c\|_2 \ge \|\mathbf{b}+\mathbf{e}_5\|_2 \ge \|\mathbf{b}\|_2-\|\mathbf{e}_5\|_2 \ge 0.86\|\mathbf{b}\|_2,$$

and use (A5) to arrive at

$$\|\mathbf{x}_c\|_2 \ge \frac{0.86\|\mathbf{b}\|_2}{\|A\|_F[1+\varepsilon(1.16n^2+13n)]}.$$

Use this lower bound in (A8) to obtain the final bound

$$\|E_{QR}\|_F \le \varepsilon\|A\|_F(1.18n^2+30n). \tag{A9}$$

The bound (8) was obtained from the following result in [7]: Let A be nonsingular. Suppose that the linear system $A\mathbf{x}=\mathbf{b}$ is solved using solver S using one step of iterative refinement, in floating point arithmetic with a unit round-off of ε. Assume that the residual is computed in the conventional manner via inner products or vector-vector adds. Assume also that the computed solution from solver S satisfies

$$|\mathbf{r}_c| \le \varepsilon(G|A|\cdot|\mathbf{x}_c|+H|\mathbf{b}|).$$

Then there is a function

$$f(t,s) \approx \left(\frac{(t+n+1)}{\||A|\cdot|A^{-1}|\|_\infty}+2(t+n+2)^2(1+\varepsilon s)^2\right)\frac{s}{n+1}$$

such that if

$$\||A|\cdot|A^{-1}|\|_\infty\sigma(A,\mathbf{x}_I) \le (f(\|G\|_\infty,\|H\|_\infty)\varepsilon)^{-1}$$

then

$$|\mathbf{b} - A\mathbf{x}_I| \le \frac{2(n+1)\varepsilon}{1-n\varepsilon}|A|\cdot|\mathbf{x}_I|. \qquad \text{(A10)}$$

The bound (8) was then obtained from (A10) by absorbing H into G (i.e., treating all the errors as if they were in the matrix) and by using the bound (A9) as a worst case bound for G.

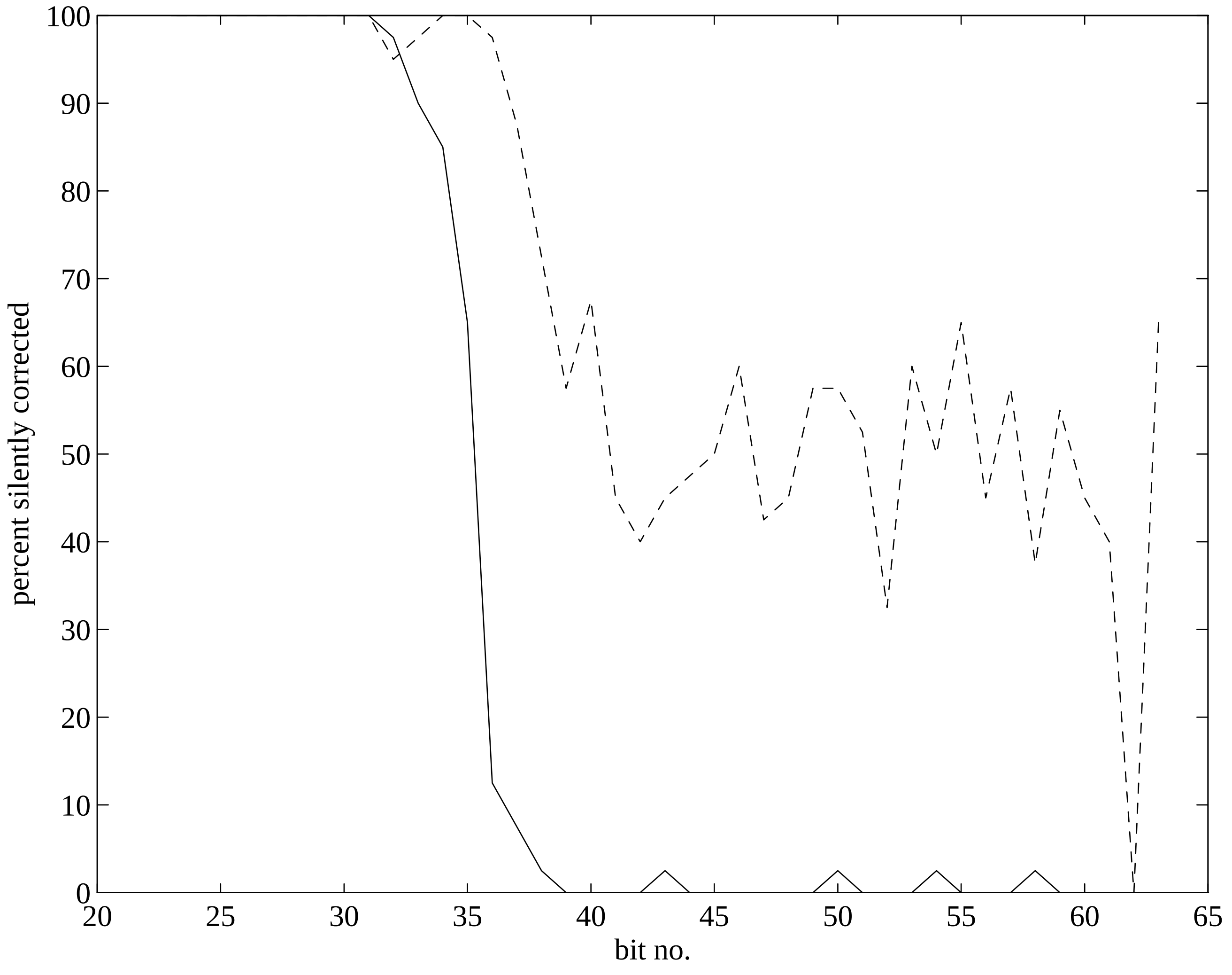

Figure 1 -- Percentage of Errors Silently Corrected vs Bit Number (**QR**).
(solid line = multiple (5) error model, dashed line = single error model)
All errors not silently corrected were detected and signaled.

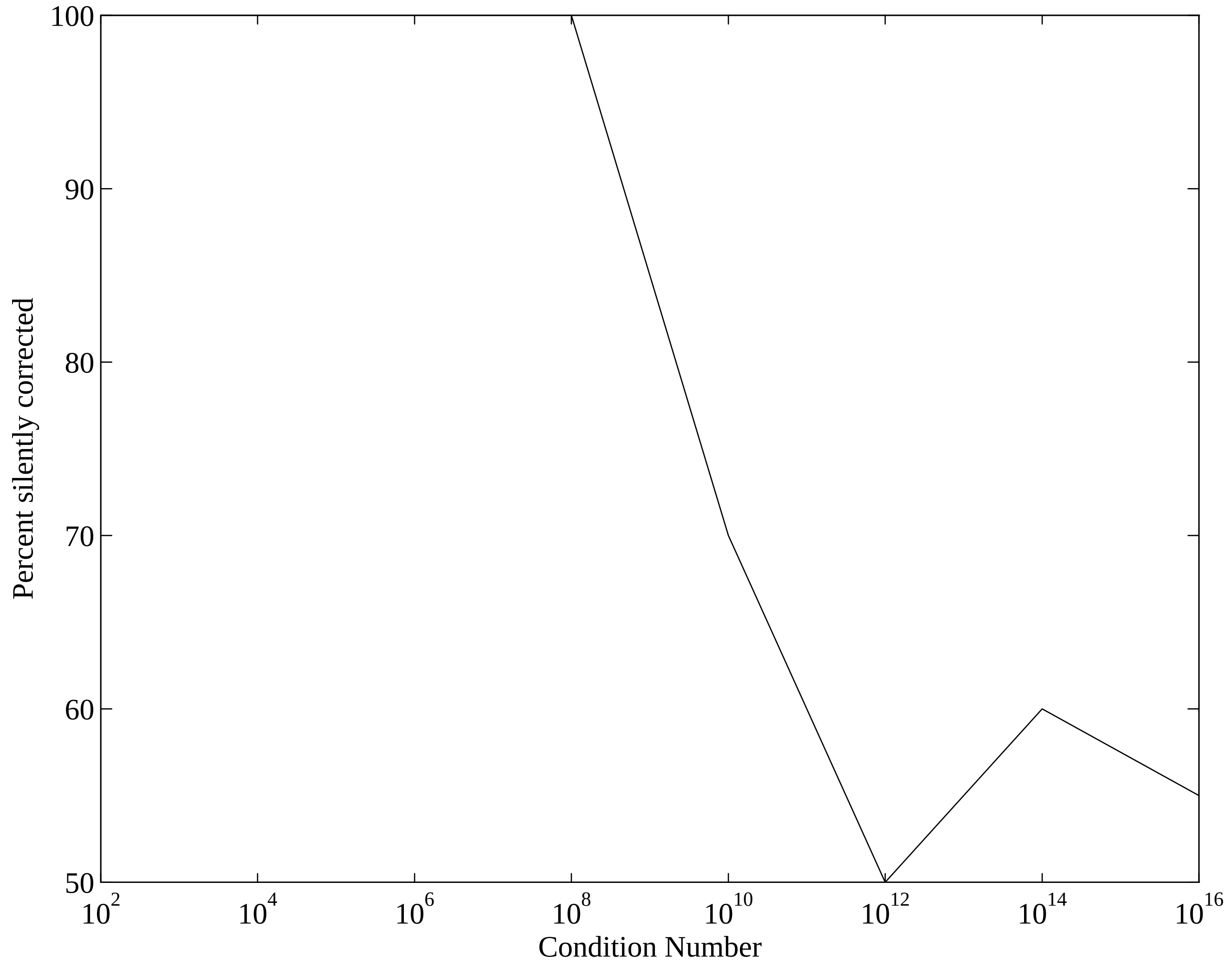

Figure 2 -- Percentage of Errors Silently Corrected vs Condition Number (**QR**).
All errors not silently corrected were detected and signaled.

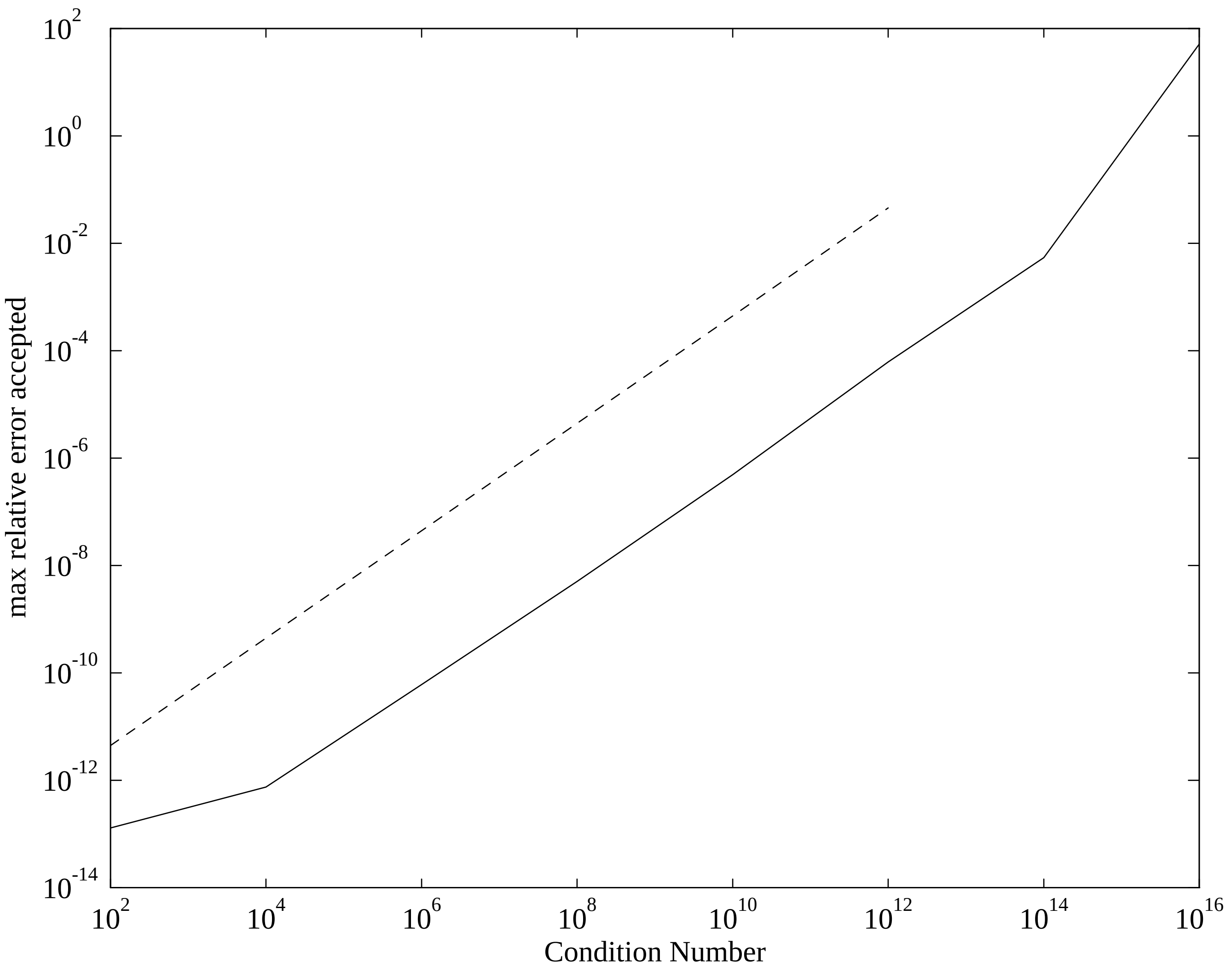

Figure 3 -- Maximum Relative Error in Accepted Answer vs Condition Number.
(solid line = result from experiments, dashed line = theoretical bound)

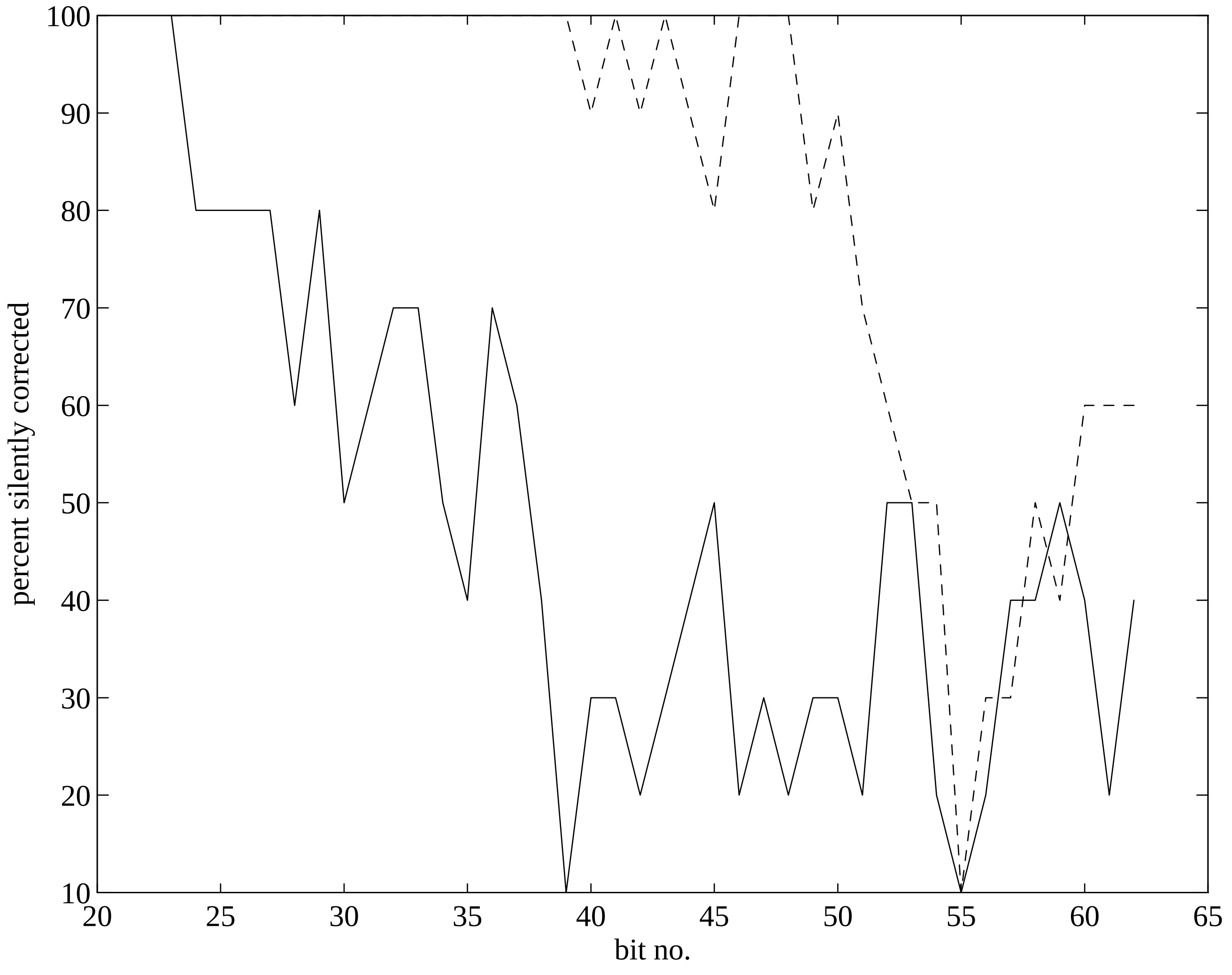

Figure 4 -- Percentage of Errors Silently Corrected vs Bit Number (**GE** with wrong pivoting strategy).
(solid line = badly scaled matrix, dashed line = uniformly random matrix)
All errors not silently corrected were detected and signaled.