Numerical Analysis Project
Manuscript NA-89-12

November 1989

Backward Error Assertions for Checking Solutions to Systems of Linear Equations

by

Daniel Boley
Gene H. Golub
Samy Makar
Nirmal Saxena
Edward J. McCluskey

Numerical Analysis Project
Computer Science Department
Stanford University
Stanford, California 94305

Backward Error Assertions for Checking Solutions to Systems of Linear Equations

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 943052140
U.S.A.

Samy Makar
Nirmal Saxena
Edward J. McCluskey

Center for Reliable Computing
Computer Systems Laboratory
Stanford University
Stanford, CA 943054055
U.S.A.

ABSTRACT

This paper presents ***an*** assertion scheme based on ***the backward error analysis*** for error detection in algorithms that solve a system of linear equations, Ax = b. This Backward Error Assertion Model 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 Ax = b. Unlike other proposed error detection methods, this assertion model does not require any encoding of matrix ***A***. Experimental results under various error models are presented to validate the effectiveness of these assertions.

1. Introduction

Some issues in the robust solution of systems of linear equations are addressed. The specific object is the design of a watchdog scheme that can guarantee that the solution is correct in some sense, or else signal that an error occurred during the solution process, such as from temporary hardware faults. Properties of some techniques for the solution of this problem are discussed.

Many papers [1], [2], [3] have been devoted to the study of checksum schemes for algorithms such as Gaussian Elimination on a matrix ***A***. We do not attempt to fully describe these methods. 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. [2]).

It is well known (e.g. [8], sec 2.4) that in floating point arithmetic, the numerical solutions to many problems are almost never exact, but they 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 [4] 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 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*** a ***a priori*** error analysis for the particular method used. Suitable ***a priori*** bounds are given in the Appendix.

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, as well as Orthogonal **Triangularization.** 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 we **used.**

2. What Checksum Schemes Won't Find

We illustrate the checksum scheme with a simpleminded example which we then use to point out a major weakness of this approach. Suppose we want to solve the system Ax = **b** in 3 decimal digit rounded arithmetic, where

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

The method we use is **GE with** Partial Pivoting [5] 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 $A = P^T LU$. 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. In this example, **pairwise** pivoting **([6], p236-9)** is the same as partial pivoting. In **pairwise** pivoting, only neighboring rows are swapped in such a way that the Pivot row is bubbled up to occupy the position of the ***k*-th *row.***

We construct a Checksum matrix

$$H = (I \mid H_c) = \begin{bmatrix} 1 & 0 & | & 1 & 2 \\ 0 & 1 & | & 1 & 1 \end{bmatrix},$$

where the number of additional checksum columns ***H_c* depends on the dimension** of the matrix. The row operations are carried out on the extended matrix

$$A_w = AH = (A \mid A_c) = \begin{bmatrix} 1.00 & 2.00 & | & 3.00 & 4.00 \\ 1.00e\text{-}3 & 1.00 & | & 1.00 & 1.00 \end{bmatrix}.$$

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

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

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

$$U_w = (U \mid U_c) = \begin{bmatrix} 1.00 & 2.00 & & 3.00 & 4.00 \\ 0 & 1.00 & | & 1.00 & 1.00 \end{bmatrix}.$$

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 x = $(1.00, 1.00)^T$, which is **almost** correct to the accuracy shown. The true answer (to 15 digits of accuracy) is x = $(1.00200400801603, 0.99899799599198)^T$, which when rounded to 3 digits is x = $(1.00, .999)^T$.

It is generally assumed that temporary errors can occur in the entries of A or among the multipliers in L 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 [4] that errors to the matrix entries may also affect the row order, but such errors will 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 the sign bit 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

$$\hat{A}_w = \hat{A}H = (\hat{A} \mid \hat{A}_c) = \begin{bmatrix} 1.00e\text{-}3 & 1.00 & | & 1.00 & 1.00 \\ 1.00 & 2.00 & | & 3.00 & 4.00 \end{bmatrix},$$

and the result of the elimination would be

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

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

$$\hat{U}_w = (\hat{U} \mid \hat{U}_c) = \begin{bmatrix} 1.00e\text{-}3 & 1.00 & | & 1.00 & 1.00 \\ 0 & -1.00e{+}3 & | & -1.00e{+}3 & -1.00e{+}3 \end{bmatrix}.$$

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. 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.

3. Backward Error Assertion Model

In this section we outline another 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. [8] pp53, 56-7)

$$\|\mathbf{x}\|_1 \equiv \sum_i |x_i|, \quad \|\mathbf{x}\|_2 \equiv \left[\sum_i |x_i|^2\right]^{\frac{1}{2}}, \quad \|\mathbf{x}\|_\infty \equiv \max_i |x_i|,$$

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

We examine **three methods:** Gaussian Elimination **(GE)** with **Partial** Pivoting, **GE with** Complete Pivoting, and Orthogonal Triangularization using Householder Transformations **(QR** Factorization). It is well known from the landmark work of J. H. Wilkinson (e.g. [6] pp157-160, 209-215, 236, 247-252) 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 [5] 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 [5].

Our approach is to check whether the computed solution meets this guarantee. 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 temporary 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

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

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 this E (1) 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:

1. Use GE with Partial Pivoting [5] to factor A into $PA \approx L_cU_c$, where L_c, U_c are lower and upper triangular, and P is a permutation. (cost: $O(n^3)$)
2. Solve triangular systems $L_c\mathbf{y} = P\mathbf{b}$ for $\mathbf{y}_c$ and $U_c\mathbf{x} = \mathbf{y}_c$ 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-(1) satisfies the bound (A3):

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

(cost $O(n^2)$)

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 ([8], 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 maximumgrowthis

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

though Wilkinson [7] points out that it is extremely rare to encounteramatrixwithgrowthgreaterthat

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

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. This property is 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:

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

2' Solve triangular systems $L_c\mathbf{y} = P\mathbf{b}$ for $\mathbf{y}_c$ and $U_c\mathbf{z} = \mathbf{y}_c$ for $\mathbf{z}_c$, then form solution $\mathbf{x}_c \equiv Q^T\mathbf{z}_c$. (cost: $O(n^2)$)

Then the bound in step 4 will again be (2), but with a different growth factor g [7]:

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

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 [5] 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. [8], sec 5.2.1). In this method, steps 1 and 2 are replaced by

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 $R_c \mathbf{x} = Q_c^T \mathbf{b}$ for $\mathbf{x}_c$. (cost: $O(n^2)$)

Normally Q_c is left as a list of Elementary Reflectors known as Householder Transformations 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):

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

Note that also in this case we can compute the norm of E directly from the norms of $\mathbf{r}_c$ and $\mathbf{x}_c$ without explicitly forming E ([8], p60).

4. Numerical Experiments

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 x = $(1.00, 1.00)^T$ and $\hat{x} = (0, 1.00)^T$, corresponding to the elimination without and with a row swap, respectively. The two corresponding residuals are $\mathbf{r}$ = (0, 1e-3), and $\hat{r}$ = (1, 0), though the first residual will be exactly zero if computed in the 3 digit arithmetic of the processor itself. The norms of the matrix E and $\hat{E}$ (1) are, respectively, 10^{-3} and 1. The right hand side of (2) with (3) is 9.8e-2 indicating that x is accepted, and $\hat{x}$ is rejected. Even with the very limited accuracy of 3 digit arithmetic, this method can guarantee -accuracy in excess of one digit.

To validate the effectiveness of the Backward Error Assertion Model on larger examples, a series of preliminary 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 (left-most), the exponent in bits 62-52, the mantissa in bits 51-O.

In the experiments, a matrix with random elements uniformly distributed in the interval $(-1,1)$ was chosen and factored into $A = LU$ using GE with Partial Pivoting. Then certain entries in the L and U were chosen, and errors injected into each bit of each chosen entry. Figures 1-4 show how the behavior of the error detection model as a function of the bit with the error. An error was detected if the residual exceeded the theoretical bound (2). Two different error detection models were tested: the hard bound (3) and the heuristic bound (4). Also two different error injection models were tested: the Single Element Error Model, in which errors were injected into a single, randomly chosen, element of L or U at a time, and the All Element Error Model, in which errors were injected into all the elements of L and U at once. In each case, the errors were injected once for each of the 64 bits in the word. In all cases, the results represent an average over several randomly chosen matrices, and in the case of the Single Element Error Model, an average over several randomly chosen elements in the L and U.

Preliminary experiments are reported in Figures 1-4, and more experiments are under way. The Figures show the percentage of detected errors versus the bit location of the injected error. As expected, Figures l-2 show clearly how the hard bound (3) becomes less sensitive as the matrix size grows; in extreme cases only errors in the sign bit or the exponent fields can be detected. However, as shown in Figures 34, the heuristic bound (4) detected errors in the upper half of the word reliably for all matrix sixes. The error in the computed solutions themselves can be bounded once the condition number of the matrix is known. Thus this error detection model would guarantee at least single precision accuracy. The results illustrate how errors in the low order bits are not detected, but are treated instead as “acceptable round-off noise”.

5. 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 outlined a methodology 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 on the one hand the validity of this overall approach and on the other hand the limitations of the particular error bounds used. 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. In addition, this model allows the simultaneous use of other assertion schemes, such as a checksum method, at no extra cost other than the separate costs of the methods used.

Acknowledgements

This research was supported in part by the Hewlett-Packard resident Fellowship, by the National Science Foundation under grants MIP-8709128, CCR-8821078, and CCR-8813493, by the Army Research Office under grant DAAL03-87-K-0095, and by the Innovative Science and Technology Office of the Strategic Defense Initiative Organization administered through the Office of Naval Research under contract N00014-85-K-0600. The work of the first author was carried out while on leave at the Stanford University Computer Science Dept., whose hospitality is gratefully acknowledged.

Appendix

In this appendix we outline the derivation far 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 [5] 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 operationsthatmakeuptheeliminationitself. 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 [5]. 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 [5]

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

where the error perturbation matrices satisfy the bounds

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

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 (Al)

$$(A + E_{LU})\mathbf{x}_c \equiv (A + E_1 + L_cE_3 + E_2U_c + E_2E_3)\mathbf{x}_c = \mathbf{b},$$

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

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

From [5], [7], *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) [7], as already mentioned above.

We can carry out a similar analysis far the Orthogonal Triangularization method, otherwise known as the QR Decomposition. To solve Ax = b, we factor $A = QR$, where Q is orthogonal, and solve the system $R\mathbf{x} = \mathbf{y} \equiv Q^T\mathbf{b}$. In floating point arithmetic, we actually compute ([6], pp157-160, 236, 250) the approximate factorization Q_cR_c and approximate vectors $\mathbf{y}_c, \mathbf{x}_c$ which satisfy:

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

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 ([6], pp157-160, 236, 250):

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

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

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

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 [5].

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

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

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 (AS) to obtain the final bound

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

REFERENCES

[1] K.H. Huang and J.A. Abraham: "Algorithm-based fault tolerance for matrix operations"; ***IEEE Trans.*** Comput. C-33 **#6, pp518-528,** June 1984.

[2] **J.Y.Jou** and J.A. Abraham: "Fault-tolerant matrix arithmetic and signal processing on highly concurrent computing structures"; ***Proc. IEEE 74 #5, Special Issue on Fault Tolerance,*** **pp732-741,** May 1986.

[3] F.T. Luk and H. **Park:** "An analysis of algorithm-based fault tolerance"; ***J. Parallel Distr. Comput. 5,*** **pp172-84,** 1988.

[4] H Park **"On** multiple error correction in matrix triangularizations using checksum schemes"; submitted to ***J. Parallel Distr. Comput., 1989.***

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

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

[7] J.H. Wilkinson: "Error analysis of direct methods of matrix inversion"; ***J. A.C.M.*** 8, **pp281-330,** 1961.

[8] G.H. Golub and C. Van Loan: ***Matrix Computations;*** Johns Hopkins, 1983.

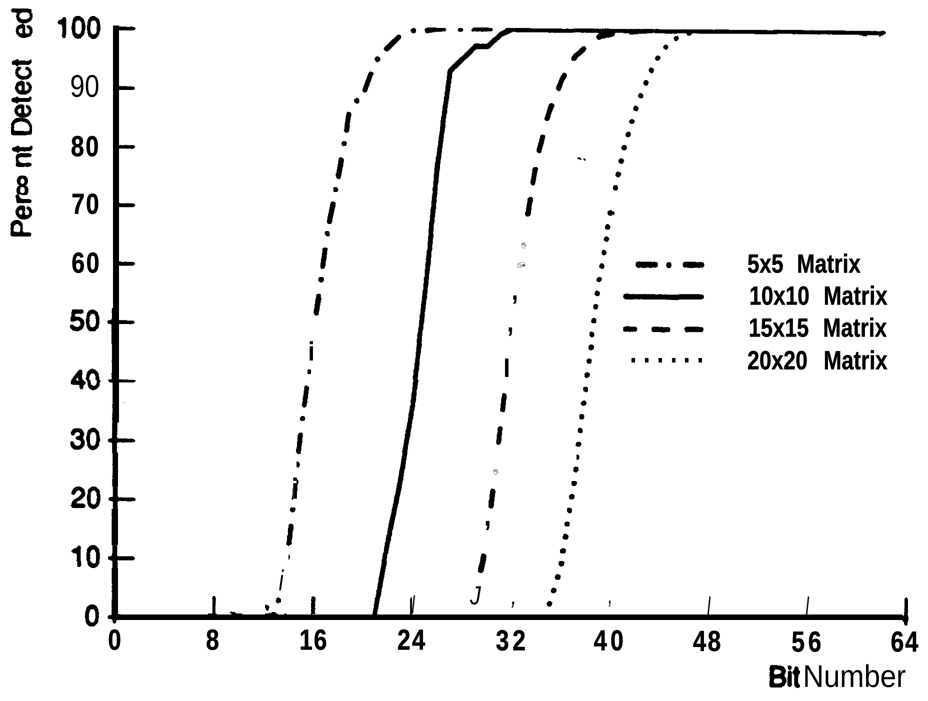

Figure 1. Detection of Single Element Errors Using Hard Bound (3)

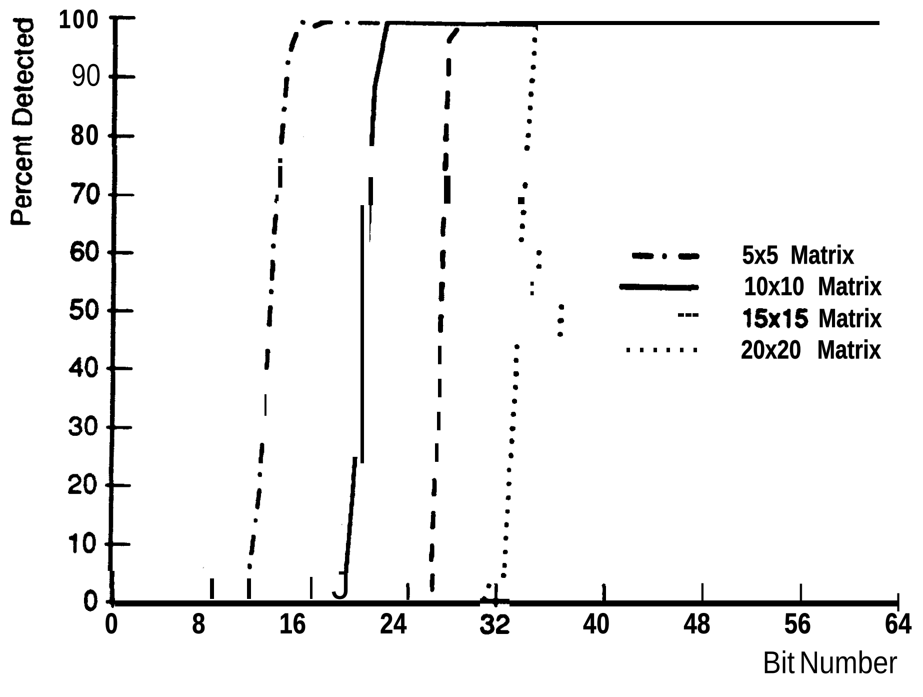

Figure 2. Detection of All Element Errors Using Hard Bound (3)

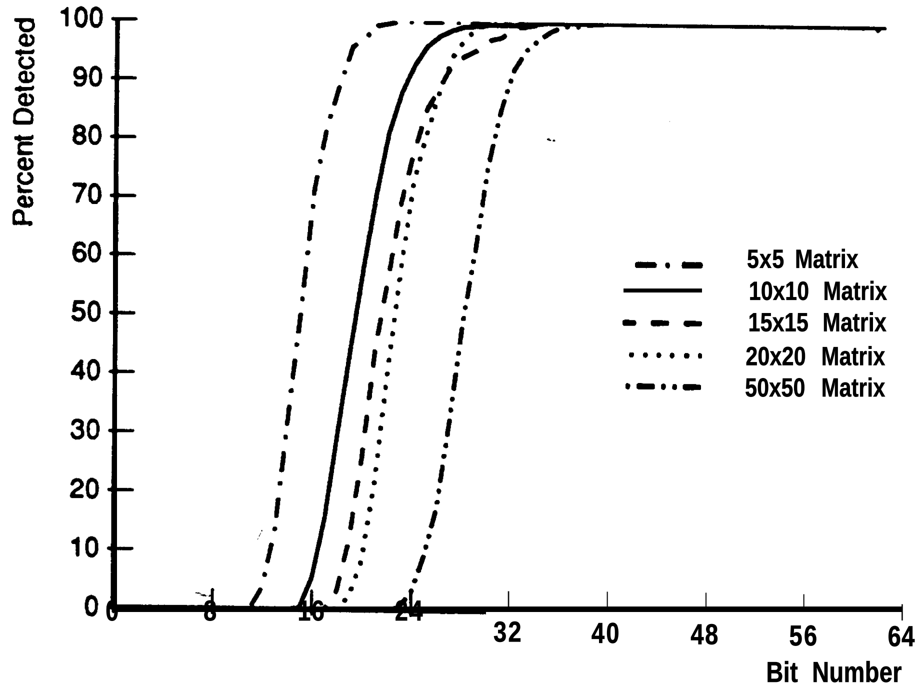

Figure 3. Detection of Single Element Errors Using Heuristic Bound (4)

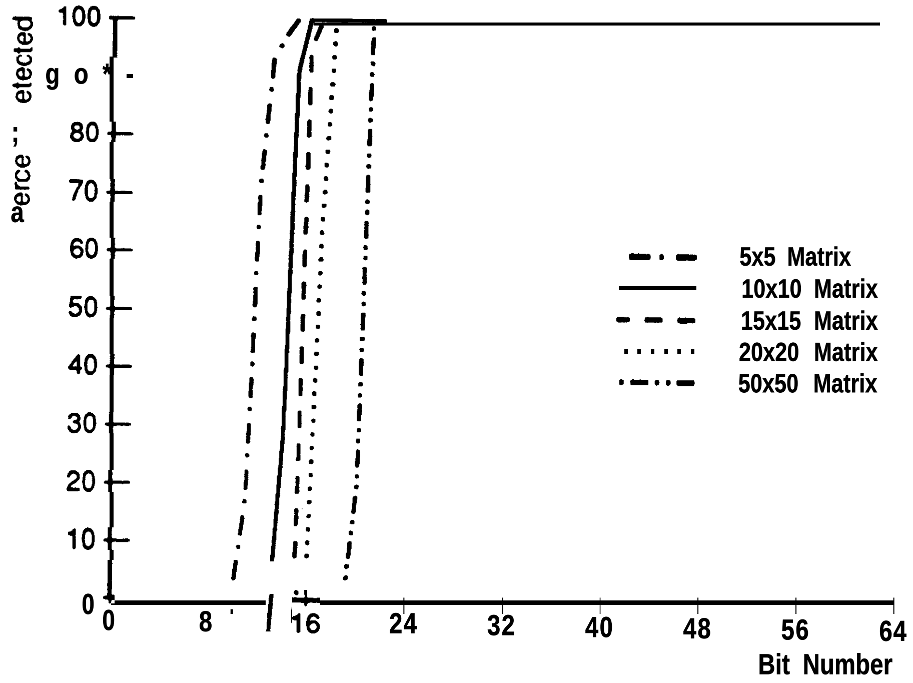

Figure 4. Detection of All Element Errors Using Heuristic Bound (4)