Algorithmic Fault Tolerance Using the Lanczos Method[1]

Daniel L. Boley
Department of Computer Science
University of Minnesota

Richard P. Brent
Computer Sciences Laboratory
Australian National University

Gene H. Golub
Department of Computer Science
Stanford University

Franklin T. Luk
School of Electrical Engineering
Cornell University

Dedicated by Boley, Brent, and Luk, to their teacher
Gene Golub on the occasion of his 15th birthday

In memory of Jeffrey Speiser

Abstract

We consider the problem of algorithm-based fault tolerance, and make two major contributions. First, we show how very general sequences of polynomials can be used to generate the checksums, so as to reduce the chance of numerical overflows. Second, we show how the Lanczos process can be applied in the error location and correction steps, so as to save on the amount of work and to facilitate actual hardware implementation.

1. Background. Many important signal processing and control problems require computational solution in real time. Much research has gone into the development of special purpose algorithms and associated hardware. The latter are usually called systolic arrays in academia, and application specific integrated circuits (ASICs) in industry. In many critical situations, so much depends on the ability of the combined software/hardware system to deliver reliable and accurate numerical results that fault tolerance is indispensable. Often, weight constraints forbid the use of multiple modular redundancy and one must resort to a software technique to handle errors. A top choice is *Algorithm-Based Fault Tolerance* (ABFT), originally developed by Abraham and students [9, 10], to provide a low-cost error protection for basic matrix operations. Their work was extended by Luk et al. [11, 13, 14] to applications that include matrix equation solvers, triangular decompositions, and recursive least squares. A theoretical framework for error correction was developed for the cases of one error [10], two errors [1], and multiple errors [7]. Interestingly, the model in [7] turns out to be the Reed-Solomon code [17]. However, the

[1] Appeared in *SIAM J. on Matrix Analysis* (Gene H. Golub 60th birthday issue) 13 (1992), 312–332.
 rpb124 typeset using LaTeX

procedure proposed in [7], and implicit in [17], is cumbersome in work and quite suspect in its numerical stability.

A lot has already appeared in the literature on fault tolerant matrix algorithms, e.g., [9, 10, 11, 13, 14, 16]. A simple example is matrix multiplication. Let A and B be given square matrices of order $n+1$, and C be the desired matrix product AB. A way to achieve fault tolerance is to append the matrix B with say $m+1$ checksum columns, with $m \leq n$, and to calculate a checksum matrix product. Details can be found in [1] and [10]. Briefly, define

$$B_r \equiv (\, B \quad S_B \,) \quad \text{and} \quad C_r \equiv (\, C \quad S_C \,),$$

where S_B and S_C denote $(n+1) \times (m+1)$ checksum matrices on B on C, respectively. Then

$$C_r = AB_r, \quad \text{and so} \quad S_C = AS_B.$$

The matrix S_C is used to detect, locate and correct errors in C. Fault tolerant matrix multiplication is simple in that the rows of C_r can be examined independently. Indeed, denote the i-th row of C_r by

$$(\, \xi_0, \xi_1, \cdots, \xi_n, \eta_0, \eta_1, \cdots, \eta_m \,), \tag{1.1}$$

where the ξ_j's represent data and the η_k's represent checksums. In [9] it is explained how, even if only one processor in the parallel system malfunctions temporarily, multiple errors will be present in the computed matrix product. In [1] it is shown that, with a judicious choice of checksum coefficients, the use of $\eta_0, \eta_1, \cdots, \eta_m$ can detect up to $m+1$ errors in $\xi_0, \xi_1, \cdots, \xi_n$, and correct up to $\lfloor (m+1)/2 \rfloor$ errors therein. A method for handling these errors is presented in [7]. Unfortunately, the procedure is quite complex, for it includes determining the rank of a matrix (to calculate the number of errors) and solving a Hankel matrix equation (to locate the errors).

A major contribution of this paper is to show how a clever use of just the Lanczos algorithm suffices for fault tolerance. The Lanczos algorithm was originally devised by Lanczos as a procedure for reducing an arbitrary matrix to a tridiagonal form having the same eigenvalues as the original matrix. The method has been found to be particularly useful for large sparse matrices and has a number of optimality properties with respect to convergence in the symmetric case. For non-symmetric matrices the algorithm has been less well understood, but in recent years there has been major progress in the development and understanding of the algorithm (see, e.g., [3] and references therein). Our work in this paper is important in at least two aspects: (1) only simple additional hardware is necessary to implement the Lanczos scheme; (2) through the use of orthogonal polynomials, our error correction problem is numerically well-conditioned. As first pointed out in [11], exponential growth in the checksum coefficients leads directly to ill-conditioning of the associated checksum matrices, which in turn leads to loss of accuracy in the computations. It is well known (see, e.g., [8]) that in solving, for example, a set of linear equations in the presence of round-off errors, a condition number of 10^x leads to a loss of about x digits of accuracy in the solutions. Ways to alleviate this difficulty were attempted in [6, 11, 16], albeit without the success here to achieve essentially "optimal conditioning" (see Section 1.3). Examples illustrating the importance of matrix conditioning in fault tolerant computing can be found in [12]. We stress here once more that our signal processing applications mandate the use of floating-point data types and operations.

This paper is organized as follows. The error correction problem is described in the next two subsections. Section 2 presents the recurrence relationships for the polynomials that generate the checksum coefficients. Sections 3 and 4 introduce the notions of Krylov matrices and error locator polynomial, respectively. The application of the Lanczos procedure to the error correction problem is discussed in Section 5. The Lanczos process is further simplified to a column elimination scheme in Section 6, and two numerical examples illustrating our ideas are given in Section 7.

1.1. Problem Description. The data $\{\xi_j\}$ and the checksums $\{\eta_i\}$ are related via:

$$\eta_i = \sum_{j=0}^{n} \nu_{ij}\xi_j, \tag{1.2}$$

where $i = 0, 1, \cdots, m$, and the coefficients $\{\nu_{ij}\}$ are pre-chosen. Suppose now that faulty computation has given us possibly corrupted data $\{\hat{\xi}_j\}$, but that the checksum values $\{\eta_i\}$ stay intact. The general case that includes errors in checksums will be discussed in Section 1.3. Hence the errors ω_j can be defined by

$$\omega_j \equiv \hat{\xi}_j - \xi_j, \tag{1.3}$$

for $j = 0, 1, \cdots, n$. Define another set of checksums $\{\hat{\eta}_i\}$ from the faulty data:

$$\hat{\eta}_i = \sum_{j=0}^{n} \nu_{ij}\hat{\xi}_j, \tag{1.4}$$

where $i = 0, 1, \cdots, m$. We note here that a major difficulty in ABFT is the proper choice of the coefficients $\{\nu_{ij}\}$. Taking difference of (1.2) and (1.4), we get

$$\hat{\eta}_i - \eta_i = \sum_{j=0}^{n} \nu_{ij}\left(\hat{\xi}_j - \xi_j\right).$$

Defining a set of *syndromes* $\{\sigma_i\}$ by

$$\sigma_i \equiv \hat{\eta}_i - \eta_i, \qquad \text{for} \quad i = 0, 1, \cdots, m,$$

we get

$$\sigma_i = \sum_{j=0}^{n} \nu_{ij}\omega_j, \qquad \text{for} \quad i = 0, 1, \cdots, m. \tag{1.5}$$

Furthermore, define an $(m+1)$-element syndrome vector $\mathbf{s}$, an $(m+1)\times(n+1)$ "generator" matrix G, and an $(n+1)$-element error vector $\mathbf{w}$ by

$$\mathbf{s} \equiv \begin{pmatrix} \sigma_0 \\ \sigma_1 \\ \vdots \\ \sigma_m \end{pmatrix}, \quad G \equiv \begin{pmatrix} \nu_{00} & \nu_{01} & \cdots & \nu_{0n} \\ \nu_{10} & \nu_{11} & \cdots & \nu_{1n} \\ \vdots & \vdots & \ddots & \vdots \\ \nu_{m0} & \nu_{m1} & \cdots & \nu_{mn} \end{pmatrix}, \quad \text{and} \quad \mathbf{w} \equiv \begin{pmatrix} \omega_0 \\ \omega_1 \\ \vdots \\ \omega_n \end{pmatrix}.$$

We can write out (1.5) in matrix form:

$$\mathbf{s} = G\mathbf{w}. \tag{1.6}$$

Given $\mathbf{s}$ and G, our *problem* is to solve for $\mathbf{w}$. Analogously, we also define the data vectors

$$\mathbf{x} = \begin{pmatrix} \xi_0 \\ \xi_1 \\ \vdots \\ \xi_n \end{pmatrix}, \quad \hat{\mathbf{x}} = \begin{pmatrix} \hat{\xi}_0 \\ \hat{\xi}_1 \\ \vdots \\ \hat{\xi}_n \end{pmatrix}$$

In this paper, we choose G as a *generalized Vandermonde* matrix:

$$G = \begin{pmatrix} p_0(x_0) & p_0(x_1) & p_0(x_2) & \cdots & p_0(x_n) \\ p_1(x_0) & p_1(x_1) & p_1(x_2) & \cdots & p_1(x_n) \\ p_2(x_0) & p_2(x_1) & p_2(x_2) & \cdots & p_2(x_n) \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ p_m(x_0) & p_m(x_1) & p_m(x_2) & \cdots & p_m(x_n) \end{pmatrix}, \tag{1.7}$$

where $p_i(x)$ denotes a polynomial of exact degree i, for $i = 0, 1, \cdots, m$, and $\{x_j\}$ denotes a set of distinct points that we call the *knots*. Hence

$$\nu_{ij} = p_i(x_j). \tag{1.8}$$

We will also scale the zero degree polynomial to unity:

$$p_0(x) \equiv 1.$$

In most previous work, e.g., [6, 7, 9, 10, 11], the polynomials $\{p_i(x)\}$ were chosen to be the *monomials*, viz.,

$$p_i(x) \equiv x^i, \qquad \text{for} \quad i = 0, 1, \cdots, m. \tag{1.9}$$

Then G is the ordinary Vandermonde matrix:

$$G = \begin{pmatrix} x_0^0 & x_1^0 & x_2^0 & \cdots & x_n^0 \\ x_0^1 & x_1^1 & x_2^1 & \cdots & x_n^1 \\ x_0^2 & x_1^2 & x_2^2 & \cdots & x_n^2 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ x_0^m & x_1^m & x_2^m & \cdots & x_n^m \end{pmatrix}. \tag{1.10}$$

Jou and Abraham [10] chose the knots as

$$x_j = 2^j, \qquad \text{for} \quad j = 0, 1, \cdots, n.$$

Recognizing that such a choice could easily lead to numerical overflow of the coefficients $\{\nu_{ij}\}$, Luk [11] proposed that

$$x_j = j + 1, \qquad \text{for} \quad j = 0, 1, \cdots, n, \tag{1.11}$$

which would grow at a somewhat slower pace. Brent et al. [6] showed how one could keep ν_{ij} from exceeding a prime number that is only a little bigger than n. However, their scheme is usable only for error detection, and not for error correction. Nair and Abraham [16] explored how standard codes over a finite field may be converted to corresponding codes over the reals with various properties. In this paper, we show how other sequences of polynomials $\{p_i(x)\}$ of exact degree i can be chosen that would yield coefficients $\{\nu_{ij}\}$ that are better scaled than those arising from the monomials. In this way an efficient Lanczos method can be used for error correction. Numerical issues, however, will not be discussed, even though the sensitivity of ABFT techniques to roundoff errors is well recognized; see, e.g., [5] and [12].

1.2. Error Location and Correction. In [1, 9, 10] a linear algebraic model of the weighted checksum scheme is developed, allowing parallels to be drawn between algorithm-based fault tolerance and coding theory. An assumption that we must make for our correction procedure is that no errors occur in the checksums. We now show that equation (1.6) with G of the form (1.7) always has at least one solution.

Theorem 1.1. For any n, let the knots $x_0, x_1, \cdots, x_n$ be distinct. Choose $m \le n$. For each i, where $i = 0, 1, \cdots, m$, let the polynomial $p_i(x)$ have exact degree i. Then the matrix G of (1.7) has full row rank, and so any $m+1$ columns of G form an $(m+1)\times(m+1)$ nonsingular matrix.

Proof: We show that $\mathbf{v}^T G = 0$ implies $\mathbf{v} = 0$. For any $(m+1)$-vector $\mathbf{v}$, define the polynomial $q(x)$ of degree at most m as

$$q(x) \equiv \mathbf{v}^T \begin{pmatrix} p_0(x) \\ p_1(x) \\ \vdots \\ p_m(x) \end{pmatrix}.$$

There is a one-to-one correspondence between vectors $\mathbf{v}$ and such polynomials $q(x)$. Then $\mathbf{v}^T G$ is an $(n+1)$-vector whose entries are the values that $q(x)$ takes on at all the knots x_i:

$$(q(x_0), \cdots, q(x_n)) = \mathbf{v}^T G.$$

If $q(x_i) = 0$ for all i, then $q(x)$ must be the zero polynomial. Hence $\mathbf{v} = \mathbf{0}$. The submatrix formed by extracting any $m+1$ columns of G also has the form (1.7) with $m+1$ distinct knots, so this submatrix is square and has full rank. □

We say that a coding scheme has *detected* the presence of errors if the syndrome vector $\mathbf{s}$ is nonzero, and that it can *correct* the errors if $\mathbf{x}$ can be recovered from $\hat{\mathbf{x}}$. From Theorem 1.1 it follows that $\mathbf{s}$ is guaranteed to be nonzero as long as the number of nonzero ω_i's (or errors) is between one and $m+1$. Hence our coding scheme can detect up to $m+1$ errors. The problem of error correction is harder. If $m = n$, then

$$\mathbf{w} = G^{-1}\mathbf{s}.$$

But when $m < n$, the solution to (1.6) is no longer unique, for we can find $\mathbf{w}$ via inverting any $(m+1)\times(m+1)$ submatrix of G. A usual choice (cf. [1]) is to restrict the number of errors so that there is only one solution. Let

$$\gamma = \begin{cases} m+1 & \text{if} \quad m = n \\ \lfloor (m+1)/2 \rfloor & \text{if} \quad m < n \end{cases} \tag{1.12}$$

and assume that at most γ errors have occurred. We claim that this $\mathbf{w}$ is unique. Assume that there is a different vector $\tilde{\mathbf{w}}$ with at most γ nonzero entries, that also satisfies (1.6). So,

$$G\mathbf{w} = \mathbf{s} \qquad \text{and} \qquad G\tilde{\mathbf{w}} = \mathbf{s}.$$

But then

$$G\,(\mathbf{w} - \tilde{\mathbf{w}}) = \mathbf{0},$$

and the difference vector $(\mathbf{w}-\tilde{\mathbf{w}})$ will have between one and $m+1$ nonzero elements, contradicting Theorem 1.1. From here on, we will assume that at most γ errors are present in the data, and in Section 6, a Lanczos method will be presented for finding the $\mathbf{w}$ that contains at most γ nonzero elements. We summarize our results as follows.

Fact 1.1. With G defined as in Theorem 1.1, our coding scheme can detect up to $m+1$ errors, for $m+1$ given checksums, or syndromes. This coding scheme can also correct up to γ errors, in the sense that for a given set of $m+1$ syndromes, there is at most one solution to (1.6) with between 1 and γ nonzero ω_j's. □

1.3. Errors in Checksums and Matrix Conditioning. Let us illustrate some of the numerical advantages of the extra flexibility we gain from using general sets of polynomials and knots. To account for errors also in the checksums, we append an equal number of "parity" values to each of the data rows of the matrix, where the parity values are set just so the checksums are zero. Specifically, we append an equal number of parity values $\{\pi_0, \pi_1, \cdots, \pi_m\}$ to the data vector and then compute the checksums from the $(n+m+2)$-element vector of data and parity values. To do this, we need $m+1$ extra knots $x_{n+1}, x_{n+2}, \cdots, x_{n+m+1}$ corresponding to the parity values. Thus, the checksums are computed by the formula

$$\begin{pmatrix} \eta_0 \\ \eta_1 \\ \vdots \\ \eta_m \end{pmatrix} = G \begin{pmatrix} \xi_0 \\ \xi_1 \\ \vdots \\ \xi_n \end{pmatrix} + F \begin{pmatrix} \pi_0 \\ \pi_1 \\ \vdots \\ \pi_m \end{pmatrix}, \tag{1.13}$$

where F is an $(m+1) \times (m+1)$ matrix whose (i,j)-th entry is $F_{ij} = p_i(x_{n+j})$. The simplest choice is to set the η_k's to zero, in which case the parity values are computed from the data values by

$$\begin{pmatrix} \pi_0 \\ \pi_1 \\ \vdots \\ \pi_m \end{pmatrix} = -F^{-1} G \begin{pmatrix} \xi_0 \\ \xi_1 \\ \vdots \\ \xi_n \end{pmatrix}. \tag{1.14}$$

Since the parity values are related to the data in the same way as the original checksums via the new coefficient matrix $F^{-1}G$, they may be carried along with the data row during all the floating-point operations in the same way as the original checksums can. With this choice for the parity values, the checksums are identically zero and hence need not be computed at all. Thus the amount of data that must be carried during the computation using this set of parity values is the same as that using the former checksum scheme with no parity values, but we gain tolerance for errors among the error check (parity) values as well as among the original data. This scheme corresponds to a *systematic linear code* in the parlance of algebraic coding theory.

When using the monomials as in [10, 11], the condition number of F can be high. As an example, with $m = 5$ and the knots of [11]: $x_j = j+1$, the condition number of the corresponding 6×6 matrix F will be at least 7×10^5, meaning that the computed parity values will have at least five fewer digits of accuracy than the original data. This would make it impossible to detect any errors occurring in the low order five digits of any data item. For larger values of m, this effect is even more marked: the condition number of the 8×8 Vandermonde matrix using knots $1, 2, \cdots, 8$ is almost 10^9.

On the other hand, if we choose the polynomials to be the Chebyshev polynomials of the first kind, we can choose the knots to substantially reduce the condition number of F. This is illustrated in the first numerical example in Section 7.

The accuracy of the computed parity values will make a big difference in the ability to detect and correct errors that occur in the lower part of the mantissa part of the floating-point words. When errors approach the lower part, they begin to become indistinguishable from rounding errors, and if a severe loss of accuracy occurs during the computation of parity values, hardware errors in the corresponding last digits of the floating-point word will be undetectable or uncorrectable, as they will be indistinguishable from rounding errors.

2. Recurrence Relations. Define $p_{n+1}(x)$ to be the monic polynomial of degree $n+1$ whose zeros are the given knots, viz.,

$$p_{n+1}(x) = \prod_{j=0}^{n} (x - x_j) = x^{n+1} + \sum_{j=0}^{n} \zeta_j x^j, \tag{2.1}$$

for some coefficients ζ_j. The two vectors $\mathbf{s}$ and $\mathbf{w}$ are related via

$$\mathbf{s} = G\mathbf{w} = GD_\omega \mathbf{e}, \tag{2.2}$$

where D_ω is an $(n+1) \times (n+1)$ diagonal matrix given by

$$D_\omega \equiv \operatorname{diag}(\omega_0, \omega_1, \cdots, \omega_n), \tag{2.3}$$

and $\mathbf{e}$ is an $(n+1)$-vector of all ones, viz., $\mathbf{e} \equiv (1, 1, \cdots, 1)^T$.

The polynomials $\{p_i(x)\}$ satisfy a set of recurrence relations which can be grouped into the matrix expression:

$$x\ (p_0(x), \cdots, p_n(x)) \ = \ (p_0(x), \cdots, p_n(x))\ Z \ + \ (0, \cdots, 0, 1)\ p_{n+1}(x)\ \zeta_{n+1}, \tag{2.4}$$

where ζ_{n+1} equals some unspecified scalar, and Z denotes an $(n+1) \times (n+1)$ *irreducible upper Hessenberg* matrix, i.e., a matrix whose immediate subdiagonal elements are all nonzero. If all the polynomials are monic then the subdiagonals of Z are all ones. Formula (2.4) expresses each polynomial $p_{k+1}(x)$ as a linear combination of $x\,p_k(x)$ and all the previous polynomials $p_0(x)$, $p_1(x)$, $\cdots, p_k(x)$. For example, if we choose the $\{p_i(x)\}$ as the monomials, then Z is the companion matrix for the polynomial $p_{n+1}(x)$:

$$Z = \begin{pmatrix} 0 & & & & & & \zeta_0 \\ 1 & 0 & & & & & \zeta_1 \\ & 1 & 0 & & & & \zeta_2 \\ & & 1 & \cdot & & & \zeta_3 \\ & & & \cdot & \cdot & & \cdot \\ & & & & \cdot & 0 & \zeta_{n-1} \\ & & & & & 1 & \zeta_n \end{pmatrix}. \tag{2.5}$$

In the procedure that we will describe, these scalars ζ_i will play no role in the actual computation, and so the matrix Z functions essentially as the "shift-down" matrix for the case of the monomials.

From here on, until the middle of Section 5, we will *assume* that

$$m = n.$$

In Section 5.1 we will show how the algorithms will still work when $m < n$. If we evaluate (2.4) at each of the knots, we obtain a relation for G:

$$D_x G^T = G^T Z, \tag{2.6}$$

where

$$D_x \equiv \operatorname{diag}\ (x_0, x_1, \cdots, x_n), \tag{2.7}$$

because $p_{n+1}(x_i) = 0$ for $i = 0, 1, \cdots, n$. Note that we have just used our assumption that $m = n$; the matrix G is now square. in computing the product $Z^T\mathbf{s}$. Equation (2.6) yields the relations

$$D_x^j G^T = G^T Z^j \qquad \text{for any } j, \tag{2.8}$$

and

$$q(D_x)G^T = G^T q(Z) \qquad \text{for any polynomial } q(x). \tag{2.9}$$

Furthermore, from (2.2) and (2.6), we derive the relation:

$$Z^T\mathbf{s} = Z^T G D_\omega \mathbf{e} = G D_x D_\omega \mathbf{e} = G D_\omega D_x \mathbf{e}, \tag{2.10}$$

which yields the two equations:

$$(Z^T)^j\mathbf{s} = G D_\omega D_x^j \mathbf{e} = G D_\omega \begin{pmatrix} x_0^j \\ x_1^j \\ \vdots \\ x_n^j \end{pmatrix} \qquad \text{for any } j, \tag{2.11}$$

and

$$q(Z^T)\mathbf{s} = G D_\omega\, q(D_x)\mathbf{e} \qquad \text{for any polynomial } q(x). \tag{2.12}$$

3. Krylov Matrices. We define two sequences of Krylov matrices $\{B_i\}$ and $\{C_i\}$, to be generated by the two matrices Z and Z^T. Let $\mathbf{e}_1$ denote the $(n+1)$-element first coordinate unit vector, viz., $(1, 0, \cdots, 0)^T$. The matrix B_j is $(n+1)\times(j+1)$, and given by

$$B_j = \Big(\, \mathbf{e}_1,\ Z\mathbf{e}_1,\ Z^2\mathbf{e}_1,\ \cdots,\ Z^j\mathbf{e}_1 \,\Big). \tag{3.1}$$

Since Z is an irreducible upper Hessenberg matrix, the matrix B_j has full column rank and is upper triangular. The column space of B_j is the same as the column space of the first $j+1$ columns of the identity matrix. The other matrix C_j has dimensions $(n+1)\times(j+1)$, and is defined by

$$C_j = \Big(\, \mathbf{s},\ Z^T\mathbf{s},\ (Z^T)^2\mathbf{s},\ \cdots,\ (Z^T)^j\mathbf{s} \,\Big). \tag{3.2}$$

Note again how we have used our assumption that $m = n$. Utilizing (2.11) we may write C_j as

$$C_j = G D_\omega V_j^T, \tag{3.3}$$

where

$$V_j = \begin{pmatrix} 1 & 1 & 1 & \cdots & 1 \\ x_0 & x_1 & x_2 & \cdots & x_n \\ x_0^2 & x_1^2 & x_2^2 & \cdots & x_n^2 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ x_0^j & x_1^j & x_2^j & \cdots & x_n^j \end{pmatrix}.$$

So, V_j consists of the first $(j+1)$ rows of an ordinary $(n+1)\times(n+1)$ Vandermonde matrix.

Now, how do we determine how many errors have occurred? Suppose that the number is k. Recall our assumption that

$$k \le \gamma. \tag{3.4}$$

In (3.3), the matrix D_ω has rank k, and from Theorem 1.1, the matrix G is nonsingular and V_j^T has full column rank which equals $(j+1)$. Hence the rank of the matrix C_j is given by

$$\text{rank } (C_j) = \begin{cases} j+1 & \text{if } \ j+1 < k \\ k & \text{if } \ j+1 \ge k \end{cases} \tag{3.5}$$

It also follows from Theorem 1.1 and (3.3) that the first l rows of C_j has maximal rank given by $\min\{j+1, k\}$, for any $l \geq k$. In particular, we have the following result.

Lemma 3.1. Let k be the number of errors (nonzero ω_j's). Denote the first k rows of B_j and C_j by $B_j^{(k)}$ and $C_j^{(k)}$, respectively. Then

$$\text{rank } \left(B_{k-1}^{(k)}\right) = \text{rank } \left(B_{k-1}\right) = k,$$

and

$$\text{rank } \left(C_{k-1}^{(k)}\right) = \text{rank } \left(C_{k-1}\right) = k.$$

Hence $C_{k-1}^T B_{k-1} = (C_{k-1}^{(k)})^T B_{k-1}^{(k)}$ is a $k \times k$ nonsingular matrix. □

Again, for the special case where $\{p_i(x)\}$ are the monomials, we get that

$$B_j = \begin{pmatrix} 1 & 0 & 0 & \cdots & 0 \\ 0 & 1 & 0 & \cdots & 0 \\ 0 & 0 & 1 & \cdots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \cdots & 1 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \cdots & 0 \end{pmatrix} \tag{3.6}$$

and

$$C_j = \begin{pmatrix} \sigma_0 & \sigma_1 & \sigma_2 & \cdots & \sigma_j \\ \sigma_1 & \sigma_2 & \sigma_3 & \cdots & \sigma_{j+1} \\ \sigma_2 & \sigma_3 & \sigma_4 & \cdots & \sigma_{j+2} \\ \vdots & \vdots & \vdots & \iddots & \vdots \\ \sigma_{n-j} & \sigma_{n-j+1} & \sigma_{n-j+2} & \cdots & \sigma_n \\ \sigma_{n-j+1} & \sigma_{n-j+2} & \sigma_{n-j+3} & \cdots & \times \\ \vdots & \vdots & \vdots & \iddots & \vdots \\ \sigma_{n-2} & \sigma_{n-1} & \sigma_n & \cdots & \times \\ \sigma_{n-1} & \sigma_n & \times & \cdots & \times \\ \sigma_n & \times & \times & \cdots & \times \end{pmatrix},$$

where $\times$ denotes elements that may not interest us.

4. Error Locator Polynomial. We have just seen how the number of errors can be calculated from the rank of $\{C_j\}$. A more difficult task is to find out which k of the ω_j's are nonzero. Labelling the errors as $\omega_{j_1}, \omega_{j_2}, \cdots, \omega_{j_k}$, we will show how to find the indices $j_1, j_2, \cdots, j_k$ by determining the corresponding knots $x_{j_1}, x_{j_2}, \cdots, x_{j_k}$.

Definition 4.1. The *error locator polynomial* is a polynomial whose zeros are precisely the knots corresponding to the nonzero ω_j's. □

Consider the $k \times (k+1)$ homogeneous system

$$C_{k-1}^T B_k \mathbf{a} = 0. \tag{4.1}$$

Denote the elements of $\mathbf{a}$ by

$$\mathbf{a} \equiv (\alpha_0, \alpha_1, \cdots, \alpha_k)^T. \tag{4.2}$$

From Lemma 3.1, a nonzero solution with $\alpha_k \neq 0$ to (4.1) exists, and is unique up to scaling. For example, if the $\{p_i(x)\}$ were the monomials, then

$$C_{k-1}^T B_k = \begin{pmatrix} \sigma_0 & \sigma_1 & \sigma_2 & \cdots & \sigma_{k-2} & \sigma_{k-1} & \sigma_k \\ \sigma_1 & \sigma_2 & \sigma_3 & \cdots & \sigma_{k-1} & \sigma_k & \sigma_{k+1} \\ \sigma_2 & \sigma_3 & \sigma_4 & \cdots & \sigma_k & \sigma_{k+1} & \sigma_{k+2} \\ \vdots & \vdots & \vdots & \iddots & \vdots & \vdots & \vdots \\ \sigma_{k-2} & \sigma_{k-1} & \sigma_k & \cdots & \sigma_{2k-4} & \sigma_{2k-3} & \sigma_{2k-2} \\ \sigma_{k-1} & \sigma_k & \sigma_{k+1} & \cdots & \sigma_{2k-3} & \sigma_{2k-2} & \sigma_{2k-1} \end{pmatrix} \tag{4.3}$$

is a Hankel matrix of syndrome values, and thus (4.1) can be regarded as *permuted Yule-Walker* equations [8, p. 184], obtained by reversing the order of the rows. With other choices for the polynomials $\{p_i(x)\}$, we can consider (4.1) as a generalization of the permuted Yule-Walker equations.

Now, in association with (4.2), define a k-th degree polynomial $q(x)$ by

$$q(x) \equiv \alpha_0 + \alpha_1 x + \cdots + \alpha_{k-1}x^{k-1} + x^k. \tag{4.4}$$

We will show that the k zeros of $q(x)$ are precisely the knots $x_{j_1}, x_{j_2}, \cdots, x_{j_k}$, corresponding to the nonzero ω-values. That is, we will show that the polynomial $q(x)$ is the error locator polynomial. Using the identity

$$\begin{aligned} G^T\, B_k\, \mathbf{a} &= G^T\, \left(\mathbf{e}_1, Z\mathbf{e}_1, Z^2\mathbf{e}_1, \cdots, Z^k\mathbf{e}_1\right)\, \mathbf{a} \\ &= \left(\mathbf{e}, D_x\mathbf{e}, D_x^2\mathbf{e}, \cdots, D_x^k\mathbf{e}\right)\, \mathbf{a} \\ &= q(D_x)\,\mathbf{e}, \end{aligned} \tag{4.5}$$

we expand (4.1) as follows:

$$0 = C_{k-1}^T\, B_k\, \mathbf{a} = V_{k-1}\, D_\omega\, G^T\, B_k\, \mathbf{a} = V_{k-1}\, D_\omega\, q(D_x)\,\mathbf{e} = V_{k-1} \begin{pmatrix} \omega_0 q(x_0) \\ \omega_1 q(x_1) \\ \vdots \\ \omega_n q(x_n) \end{pmatrix}.$$

If we extract only those entries involving the nonzero ω-values, we obtain a $k \times k$ nonsingular, homogeneous system:

$$0 = \begin{pmatrix} 1 & 1 & 1 & \cdots & 1 \\ x_{j_1} & x_{j_2} & x_{j_3} & \cdots & x_{j_k} \\ x_{j_1}^2 & x_{j_2}^2 & x_{j_3}^2 & \cdots & x_{j_k}^2 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ x_{j_1}^{k-1} & x_{j_2}^{k-1} & x_{j_3}^{k-1} & \cdots & x_{j_k}^{k-1} \end{pmatrix} \begin{pmatrix} \omega_{j_1} q(x_{j_1}) \\ \omega_{j_2} q(x_{j_2}) \\ \omega_{j_3} q(x_{j_3}) \\ \vdots \\ \omega_{j_k} q(x_{j_k}) \end{pmatrix}. \tag{4.6}$$

Hence

$$q(x_{j_i}) = 0 \quad \text{for} \quad i = 1, 2, \cdots, k,$$

as we desired.

We can also express the error locator polynomial as a linear combination of the original set of polynomials $\{p_i(x)\}$, which may be useful in the computational procedure. We define a polynomial $r(x)$ by

$$r(x) \equiv (p_0(x), p_1(x), \cdots, p_n(x))\;\; B_k\mathbf{a}. \tag{4.7}$$

From the upper triangular structure of the matrix B_k, we see that $r(x)$ is a polynomial of degree at most k. If we evaluate $r(x)$ at each knot, we see from (4.5) that it agrees with $q(x)$ at every knot:

$$\begin{pmatrix} r(x_0) \\ r(x_1) \\ \vdots \\ r(x_n) \end{pmatrix} = G^T B_k \mathbf{a} = q(D_x)\mathbf{e} = \begin{pmatrix} q(x_0) \\ q(x_1) \\ \vdots \\ q(x_n) \end{pmatrix}. \tag{4.8}$$

So,

$$r(x) \equiv q(x). \tag{4.9}$$

We summarize our results as follows.

Fact 4.1. Suppose that there are exactly k errors, and that the syndrome vector $\mathbf{s}$ has been computed using G, which is generated via the recurrence matrix Z. Let B_k and C_{k-1} be the two Krylov matrices generated by Z, e_1 and $\mathbf{s}$. Then the error locator polynomial $q(x)$ is defined by (4.4), where the vector of coefficients $\mathbf{a}$ is the unique (up to scaling) nonzero solution to (4.1). Furthermore, we may express $q(x)$ as a linear combination of the original polynomials $\{p_i(x)\}$, in which case the coefficients are simply the entries of the vector $B_k\mathbf{a}$, as proved in (4.7) and (4.9). $\square$

5. Lanczos Process. The nonsymmetric Lanczos Algorithm, described in detail in [3], is a recursive process that starts with the matrix A and two vectors $\mathbf{r}_0$ and $\mathbf{l}_0$, and generates two sequences of matrices $\{R_j\}$ and $\{L_j\}$, given by

$$R_j \equiv (\mathbf{r}_0, \mathbf{r}_1, \cdots, \mathbf{r}_j) \tag{5.1}$$

and

$$L_j \equiv (\mathbf{l}_0, \mathbf{l}_1, \cdots, \mathbf{l}_j), \tag{5.2}$$

for $j = 0, 1, \cdots, n$. Hence R_j and L_j are both $(n+1) \times (j+1)$ matrices. Let $\mathrm{Sp}(M)$ denote the column space of a matrix M. Then, for every j, the following four relations will be satisfied:

$$\mathrm{Sp}(R_j) = \mathrm{Sp}\left(\mathbf{r}_0,\, A\,\mathbf{r}_0,\, A^2\mathbf{r}_0, \cdots,\, A^j\mathbf{r}_0\right), \tag{5.3}$$

$$\mathrm{Sp}(L_j) = \mathrm{Sp}\left(\mathbf{l}_0,\, A^T\mathbf{l}_0,\, (A^T)^2\mathbf{l}_0, \cdots,\, (A^T)^j\mathbf{l}_0\right), \tag{5.4}$$

and, if $L_j^T R_j$ is nonsingular, then

$$\mathbf{l}_{j+1}^T R_j = 0, \tag{5.5}$$

$$\mathbf{r}_{j+1}^T L_j = 0. \tag{5.6}$$

Property (5.4) implies that $\mathbf{l}_{j+1}$ is a linear combination of $A^T\mathbf{l}_j$ and $\mathbf{l}_0, \mathbf{l}_1, \cdots, \mathbf{l}_j$. If the matrix $L_j^T R_j$ is nonsingular, then the particular linear combination is chosen to satisfy (5.5). Otherwise, we have some freedom in choosing $\mathbf{l}_{j+1}$, and we may pick $\mathbf{l}_{j+1} = A^T\mathbf{l}_j$. Other choices satisfying (5.3)-(5.4) are possible, but this particular one will lead to a computational simplification, as will be seen in the next section. We can derive similar results for the $\{\mathbf{r}_i\}$ vectors. The process terminates when either $\mathbf{l}_j = 0$ or $\mathbf{r}_j = 0$, for some j.

For our situation, we propose to use the Lanczos process with the matrix $A = Z$, and the starting vectors $\mathbf{r}_0 = \mathbf{e}_1$ and $\mathbf{l}_0 = \mathbf{s}$. With this choice, we get

$$\mathrm{Sp}(R_j) = \mathrm{Sp}(B_j) \tag{5.7}$$

and

$$\mathrm{Sp}(L_j) = \mathrm{Sp}(C_j), \tag{5.8}$$

for $j = 0, 1, \cdots, n$. Since the matrix Z is irreducible upper Hessenberg, the matrix R_j will be upper triangular, and will have full column rank $j+1$, for every $j < k$. Hence the Lanczos process will terminate at the k-th step with $\mathbf{l}_k = 0$, by property (3.4). Since the matrix $C_{k-1}^T B_{k-1}$ is nonsingular, we get from (5.6) that $\mathbf{r}_k$ will be the vector in the column space of R_k that is orthogonal to L_{k-1}, or equivalently in the column space of B_k that is orthogonal to C_{k-1}. But this means that the vector $\mathbf{r}_k$ equals the vector $B_k\mathbf{a}$ defined by (4.1), up to a scaling constant. Hence the Lanczos Algorithm may be used to generate the error locator polynomial $q(x)$ as a linear combination of the original set of polynomials $\{p_i(x)\}$.

Fact 5.1. Suppose that we have run the nonsymmetric Lanczos process with the matrix Z, and the starting vectors $\mathbf{e}_1$ and $\mathbf{s}$. The process will terminate at the k-th step with $\mathbf{l}_k = 0$, and the vector $\mathbf{r}_k$ will equal the vector $B_k\mathbf{a}$, except possibly for a scaling factor. Hence the entries of $\mathbf{r}_k$ give us the coefficients of the error locator polynomial $q(x)$ in terms of the original set of polynomials $\{p_i(x)\}$, as shown in (4.7) and (4.9). □

5.1. Case Where $m < n$. We now examine the case where there are fewer syndromes than data values, i.e., $m < n$. Indeed in practice, usually $m << n$. As noted just below equation (3.4), we may check the rank of C_j for every j by just checking the first l rows, as long as $l \geq k$. If γ, the maximum number of errors, is known, then it suffices to examine the first γ rows of C_j, or equivalently of L_j, and the Lanczos process is guaranteed to terminate in at most γ steps. Note that each $\mathbf{l}_{j+1}$ in the Lanczos process is a linear combination of $Z^T\mathbf{l}_j$ and of $\mathbf{l}_0, \mathbf{l}_1, \cdots, \mathbf{l}_j$. Since Z^T is lower Hessenberg and $\mathbf{l}_j$ has j leading zero elements, to find the top ρ elements of $Z^T\mathbf{l}_j$, we need to know only the top $\rho + 1$ elements of $\mathbf{l}_j$. It follows that the first $\gamma + 1$ values of the generated vectors $\mathbf{l}_0, \cdots, \mathbf{l}_{\gamma-1}$ depend only upon the first 2γ values of the initial vector $\mathbf{l}_0 = \mathbf{s}$. Therefore, it suffices to compute only 2γ syndrome values in order to generate the coefficients in (4.1) that are needed to solve for $\mathbf{a}$. Recall from (1.12) our assumption that

$$\gamma = \lfloor (m+1)/2 \rfloor,$$

and so, given γ, one should choose m so that

$$m + 1 = 2\gamma.$$

Fact 5.2. If the number of errors is at most γ, then 2γ syndrome values are necessary and sufficient to determine the error locator polynomial and its zeros by means of equation (4.1). □

6. Column Elimination Scheme. The Lanczos process as described simplifies somewhat for the particular purpose we are using it here, that of computing the error locator polynomial. With our particular starting data, the right Lanczos matrix R_j will be upper triangular and will have full column rank, for every j. The left Lanczos matrix L_j may assume several forms. We first examine the "generic" case that $L_j^T R_j$ is nonsingular for every j. Then the condition (5.5) is equivalent to forcing L_{j+1} to be lower triangular. In the Lanczos algorithm, this structure is obtained by subtracting multiples of previous columns $\{\mathbf{l}_i\}$ from the one that has just been generated as $Z^T\mathbf{l}_j$. That is, we perform "column operations" akin to "row operations" in ordinary Gaussian elimination. (Note that using another elimination scheme such as an orthogonal decomposition would destroy the properties (5.3) and (5.4) as well as the triangular structure of the generated matrices.) Thus, at stage j of the process, we generate $\mathbf{l}_{j+1}$ from $\mathbf{l}_{j-1}$ and $\mathbf{l}_j$ as follows. The vector $\mathbf{l}_{j-1}$ has $(j-1)$ leading zero entries, the vector $\mathbf{l}_j$ has j leading zero entries, and the vector $Z^T\mathbf{l}_j$ $(\equiv \tilde{\mathbf{l}}_{j+1})$ has $(j-1)$ zero entries:

$$
(\,\mathbf{l}_{j-1},\quad \mathbf{l}_j,\quad \tilde{\mathbf{l}}_{j+1}\,) = \begin{pmatrix} 0 & 0 & 0 \\ \vdots & \vdots & \vdots \\ 0 & 0 & 0 \\ l_{j-1,j-1} & 0 & \tilde{l}_{j-1,j+1} \\ l_{j,j-1} & l_{j,j} & \tilde{l}_{j,j+1} \\ l_{j+1,j-1} & l_{j+1,j} & \tilde{l}_{j+1,j+1} \\ \vdots & \vdots & \vdots \end{pmatrix}.
$$

We must eliminate the two elements $\tilde{l}_{j-1,j+1}$ and $\tilde{l}_{j,j+1}$ to obtain an $\mathbf{l}_{j+1}$ that has $j+1$ leading zero entries. These two eliminations are done by subtracting from $\tilde{\mathbf{l}}_{j+1}$ suitable multiples of $\mathbf{l}_{j-1}$ and $\mathbf{l}_j$, respectively.

We now examine the "nongeneric" case. Suppose that for some particular value of j, the matrix $L_j^T R_j$ is singular, and $L_{j-1}^T R_{j-1}$ is nonsingular. (The following also applies for the case where $j=0$, i.e., $\mathbf{l}_0^T\mathbf{r}_0 = 0$.) This means that $\mathbf{l}_j$ must have more than j leading zero entries. Suppose there are i extra leading zero entries, for a total of $j+i$ leading zero entries:

$$
\mathbf{l}_j = \begin{pmatrix} 0 \\ \vdots \\ 0 \\ l_{j+i,j} \\ l_{j+i+1,j} \\ \vdots \end{pmatrix} \tag{6.1}
$$

Then the next $(i+1)$ vectors $\tilde{\mathbf{l}}_{j+1}, \tilde{\mathbf{l}}_{j+2}, \cdots, \tilde{\mathbf{l}}_{j+i+1}$ are defined simply by

$$
\tilde{\mathbf{l}}_{j+l} = Z^T\tilde{\mathbf{l}}_{j+l-1} = (Z^T)^l\tilde{\mathbf{l}}_j,
$$

for $l = 1, 2, \cdots, i+1$. Due to the lower Hessenberg form of Z^T, these vectors have a lower anti-triangular form, as illustrated below for $i=3$:

$$
\left(\mathbf{l}_{j-1}, \mathbf{l}_j, \tilde{\mathbf{l}}_{j+1}, \tilde{\mathbf{l}}_{j+2}, \tilde{\mathbf{l}}_{j+3}, \tilde{\mathbf{l}}_{j+4}\right) = \begin{pmatrix} 0 & 0 & 0 & 0 & 0 & 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ 0 & 0 & 0 & 0 & 0 & 0 \\ l_{j-1,j-1} & 0 & 0 & 0 & 0 & \tilde{l}_{j-1,j+4} \\ l_{j,j-1} & 0 & 0 & 0 & \tilde{l}_{j,j+3} & \tilde{l}_{j,j+4} \\ l_{j+1,j-1} & 0 & 0 & \tilde{l}_{j+1,j+2} & \tilde{l}_{j+1,j+3} & \tilde{l}_{j+1,j+4} \\ l_{j+2,j-1} & 0 & \tilde{l}_{j+2,j+1} & \tilde{l}_{j+2,j+2} & \tilde{l}_{j+2,j+3} & \tilde{l}_{j+2,j+4} \\ l_{j+3,j-1} & l_{j+3,j} & \tilde{l}_{j+3,j+1} & \tilde{l}_{j+3,j+2} & \tilde{l}_{j+3,j+3} & \tilde{l}_{j+3,j+4} \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \end{pmatrix}. \tag{6.2}
$$

We wish to "exhibit" the rank of this matrix by reducing it to a column permutation of a lower triangular matrix. To preserve the Krylov sequence property (5.4), we set

$$
\mathbf{l}_j = \tilde{\mathbf{l}}_j,\ \mathbf{l}_{j+1} = \tilde{\mathbf{l}}_{j+1},\ \cdots,\ \mathbf{l}_{j+i} = \tilde{\mathbf{l}}_{j+i}.
$$

Then to form $\mathbf{l}_{j+i+1}$ we must eliminate the leading $(i+2)$ nonzero entries of $\tilde{\mathbf{l}}_{j+i+1}$, namely $\tilde{l}_{j-1,j+i+1}, \tilde{l}_{j,j+i+1}, \cdots, \tilde{l}_{j+i,j+i+1}$, by means of column operations. Note that property (5.4) is still preserved for all the index values j, $j+1$, $\cdots$, $j+i+1$, and that this column elimination scheme is essentially the Berlekamp-Massey algorithm [2, 15] for solving the permuted Yule-Walker problem (4.1) when the coefficient matrix is given by (4.3). The scheme requires $i+2$ column operations, less than the $2(i+1)$ that would have been required for the generic case.

Whether the generic or nongeneric elimination scheme is used, the result after k steps is a full rank matrix L_{k-1} which is either lower triangular, or a column permutation of a lower triangular matrix. Therefore, to solve for $\mathbf{a}$ in (4.1), it suffices to choose an arbitrary nonzero value for $\tilde{\alpha}_k$ and solve the following system for $\tilde{\alpha}_0, .., \tilde{\alpha}_{k-1}$:

$$L_{k-1}^T (R_k \tilde{\mathbf{a}}) = \mathbf{0}. \tag{6.3}$$

As $\mathrm{Sp}\,(C_{k-1}) = \mathrm{Sp}\,(L_{k-1})$, and hence $\mathrm{Null}\left(C_{k-1}^T\right) = \mathrm{Null}\left(L_{k-1}^T\right)$, the right annihilating vector $B_k\mathbf{a}$ of C_{k-1}^T in (4.1) is the same as the right annihilating vector $R_k\tilde{\mathbf{a}}$ of L_{k-1}^T in (6.3). That is,

$$B_k\mathbf{a} = R_k\tilde{\mathbf{a}}, \tag{6.4}$$

except for a scaling constant. The matrix R_k is upper triangular, so it suffices to extract only the first $(k+1)$ rows of L_{k-1}. The $(k+1)$-st row of L_{k-1} enters only into the part depending on $\tilde{\alpha}_k$, so (6.3) is a $k \times k$ system for the remaining $\tilde{\alpha}$-values. Since L_{k-1} is lower triangular (at least within a column permutation), solving (6.3) for $R_k\tilde{\mathbf{a}} = B_k\mathbf{a}$ requires a back-substitution step, and to obtain $\mathbf{a}$ itself requires another back-substitution step. If we are interested only in the locations of the zeros of the error locator polynomial, as opposed to the coefficients of the polynomial itself, it suffices to solve (6.3) for the right annihilating vector $B_k\mathbf{a}$ and substitute this result directly into (4.8), yielding directly the values of the error locator polynomial evaluated at every knot.

We summarize the steps to obtain $\mathbf{a}$ as follows:

0. Start with $\mathbf{l}_0 = \mathbf{s}$.
1. For $i = 0, 1, \cdots$, compute $\mathbf{l}_{i+1}$ by forming $\tilde{\mathbf{l}}_{i+1} = Z^T\mathbf{l}_i$, and annihilating the first two nonzero entries by two "column operations". (In the nongeneric case described above, we form the vectors as illustrated in (6.2) and follow the prescription described thereafter.)
2. The process in Step 1 continues until $\mathbf{l}_k = 0$ for some value k. This value is the number of errors in the data.
3. Solve system (6.3) for $B_k\mathbf{a}$ using (6.4).
4. If the error locator polynomial itself is desired, as opposed to its zeros, then generate the Krylov matrix B_k, and back-solve to obtain $\mathbf{a}$.

Step 1 is guaranteed to terminate in at most γ steps. This requires that only the first $m+1$, which equals 2γ, syndromes be carried, and that only the first $m+1-j$ entries in each $\mathbf{c}_j$ vector be computed. Therefore, to carry out Step 1 requires at most γ applications of the matrix Z^T and 2γ "column operations".

If the bandwidth of Z equals b, then each application of Z or Z^T to the leading $m+1$ entries of a vector costs $(m+1)b$ operations. Since only the leading principal submatrix of Z participates in the computations, the bandwidth of (2.5) is $b = 1$. Indeed, that particular choice would incur only shifting and no arithmetic costs. If the original set of polynomials $\{p_i(x)\}$ were the Chebyshev polynomials, or some other sequence of orthogonal polynomials, then Z would be tridiagonal and the bandwidth would be $b = 3$, as in the numerical example in the next section.

Only Steps 1 and 3 require computation for the specific syndrome values: we approximate their costs using $k \leq \gamma$ and $m = 2\gamma - 1$.

Cost of Step 1 $= k(mb + 2m) \leq 2\gamma^2(b+2)$.
Cost of Step 3 $=$ Cost of Back Substitutions $= k^2/2 \leq \gamma^2/2$.

Total Cost $\leq \gamma^2(2b+4.5)$.

To summarize, we have described a method which computes a lower triangular basis for the Krylov space, Sp (C_{k-1}). By recursively carrying out column eliminations as each new column is generated, we are able to exhibit the maximal rank of R_j and L_j at each stage j. We then use the lower triangular basis L_{k-1} to solve for the vector $B_k\mathbf{a}$ representing the error locator polynomial. In principle, we could use *any* basis for Sp (C_{k-1}). If we used instead an orthonormal basis, we would enhance the numerical stability of the method at a cost of more arithmetic. Such an orthonormal basis can be generated recursively by an Arnoldi process (see e.g. [4]), and the rank would be exhibited in the same way as in the above process. But in this paper we focus on the lower triangular basis because it is simple to compute and because it is closely related to the nonsymmetric Lanczos and Berlekamp-Massey algorithms.

7. Numerical Examples. Except for the possible goal of reducing the condition number of the relevant matrices, the choice of polynomials and knots is arbitrary as long as the polynomials are of increasing degree and the knots are distinct. These are the only conditions required to apply the Lanczos-based paradigm. Different choices lead to a wide variety of different schemes, including many of the standard ones. In this section we illustrate our method with two particular numerical examples in which we use the Chebyshev polynomials and the monomials to generate the checksum coefficients and the knots. In printing the numbers, we have rounded them to the digits shown, even though the computations were carried out in Lisp on a Sun workstation with IEEE arithmetic using a precision of about 16 decimal digits. The first two Chebyshev polynomials are

$$p_0(x)=1, \quad p_1(x)=x,$$

and it is well known that the subsequent polynomials are generated by the recurrence

$$p_{i+1}(x)=2x\,p_i(x)-p_{i-1}(x), \qquad \text{for } i=1,2,\cdots.$$

the Chebyshev polynomials $p_0(x), p_1(x), \cdots$ are related via the recurrence (2.4) and the recurrence matrix

$$\frac{1}{2}\begin{pmatrix} 0 & 1 & 0 & 0 & \\ 2 & 0 & 1 & 0 & \ddots \\ 0 & 1 & 0 & 1 & \ddots \\ 0 & 0 & 1 & 0 & \ddots \\ 0 & 0 & 0 & 1 & \ddots \\ 0 & 0 & 0 & 0 & \ddots \\ & \ddots & \ddots & \ddots & \ddots \end{pmatrix}, \tag{7.1}$$

and it is well known that the zeroes of the polynomial p_k are all real, simple, and are the same as the eigenvalues of the leading $k\times k$ principal submatrix of (7.1).

Example 1. We illustrate the process of determining the errors that might be present in a given row $(\xi_0, \xi_1, \cdots, \xi_n)$ of an $(n+1)\times(n+1)$ matrix A. In order to compute the checksums, we need $n+1$ knots $x_0, x_1, \cdots, x_n$, from which we determine the matrix G of checksum coefficients (1.7) using the Chebyshev polynomials.

In order to illustrate process for handling also errors in the checksums, we suppose that $m+1=6$ parity values $\pi_0, \pi_1, \cdots, \pi_5$ have been appended to the matrix row, and that an extra six knots have been chosen. Corresponding to the parity values are six extra knots $x_{n+1}, x_{n+2}, \cdots, x_{n+6}$. Define the 6×6 matrix F by $F_{ij}=p_{i-1}(x_{n+j})$, for $i,j=1,2,\cdots,6$.

We then define the parity values by (1.14) so that the checksums (1.13) computed on the entire "data sequence":

$$\xi_0, \xi_1, \cdots, \xi_n, \pi_0, \pi_1, \cdots \pi_5, \tag{7.2}$$

are zero.

We may choose the knots to be any set of distinct numbers, so we make the following arbitrary choice. The last eight knots are chosen as the zeros of p_8:

$$\begin{aligned}
x_{n-1} &= \cos 15\pi/16 = -0.980785 \\
x_{n+0} &= \cos 13\pi/16 = -0.831470 \\
x_{n+1} &= \cos 11\pi/16 = -0.555570 \\
x_{n+2} &= \cos \ \ 9\pi/16 = -0.195090 \\
x_{n+3} &= \cos \ \ 7\pi/16 = +0.195090 \\
x_{n+4} &= \cos \ \ 5\pi/16 = +0.555570 \\
x_{n+5} &= \cos \ \ 3\pi/16 = +0.831470 \\
x_{n+6} &= \cos \ \ \ \ \pi/16 = +0.980785
\end{aligned}$$

and the $n-1$ remaining knots are chosen as the zeroes of p_{n-1}, which are all distinct from those of p_8 for any $n-1$ relatively prime to 8. All $n+7$ knots are guaranteed to be distinct. This is a modification of a periodic code in the sense of [16]. We have that $m = 5$ and $\gamma = 3$. The combined matrix $(G \mid F)$ of checksum coefficients is given by

$$(G \mid F) = \\ \left(\begin{array}{ccc|cccccc}
\cdots & 1.0000 & 1.0000 & 1.0000 & 1.0000 & 1.0000 & 1.0000 & 1.0000 & 1.0000 \\
\cdots & -0.9808 & -0.8315 & -0.5556 & -0.1951 & 0.1951 & 0.5556 & 0.8315 & 0.9808 \\
\cdots & 0.9238 & 0.3827 & -0.3827 & -0.9238 & -0.9238 & -0.3827 & 0.3827 & 0.9238 \\
\cdots & -0.8315 & 0.1951 & 0.9808 & 0.5556 & -0.5556 & -0.9808 & -0.1951 & 0.8315 \\
\cdots & 0.7071 & -0.7071 & -0.7071 & 0.7071 & 0.7071 & -0.7071 & -0.7071 & 0.7071 \\
\cdots & 0.5556 & 0.9808 & -0.1951 & -0.8315 & 0.8315 & 0.1951 & -0.9808 & 0.5556
\end{array}\right)$$

We note that the condition number of F is 89. If instead we use the polynomials (1.9) and knots (1.11), then G and F would both be ordinary Vandermonde matrices, and the condition number of F would be in excess of 7×10^5. This means that by using the Chebyshev polynomials, we may compute the parity values (1.14) to almost full machine accuracy, whereas when using (1.9) and (1.11), the last five digits of the computed parity values are guaranteed to be in error from the ill-conditioning of F. In the latter case, we would not be able to detect any errors that might occur in the last five digits of any data item. We note that if we choose the last six knots as the zeroes of p_6 and the remaining $n+1$ knots as the zeroes of p_{n+1} then it can be shown that the rows of F would be mutually orthogonal and the condition number of F would be reduced to only 2. We use the choice of p_8 and p_{n-1} to illustrate that many different choices can lead to much improvement in the condition number.

Given the $n+7$ knots, the checksum coefficients are generated by the first six Chebyshev polynomials, which satisfy the recurrence (2.4) where the recurrence matrix Z is just the leading 6×6 principal submatrix of (7.1). The Krylov sequence $\{B_j\}$ depends only on the recurrence matrix Z. In particular, B_5 is given by

$$B_5 = \begin{pmatrix}
1.0 & 0 & 0.5 & 0 & 0.375 & 0 \\
0 & 1.0 & 0 & 0.75 & 0 & 0.6250 \\
0 & 0 & 0.5 & 0 & 0.500 & 0 \\
0 & 0 & 0 & 0.25 & 0 & 0.3125 \\
0 & 0 & 0 & 0 & 0.125 & 0 \\
0 & 0 & 0 & 0 & 0 & 0.0625
\end{pmatrix}. \tag{7.3}$$

Since the parity values were chosen just to make the checksums zero, the syndrome values are obtained by applying the checksum coefficient matrix $(G \mid F)$ to the augmented data (7.2). We suppose that the resulting six $(= 2\gamma)$ syndromes are

$$\mathbf{s} = \begin{pmatrix} -2.000000 \\ -2.443301 \\ 4.460885 \\ 2.612462 \\ -4.242641 \\ -1.364308 \end{pmatrix}.$$

The other Krylov sequence $\{C_j\}$ is calculated as

$$C_j = \begin{pmatrix} -2.0000 & -2.4433 & 1.2304 & \cdots \\ -2.4433 & 1.2304 & -1.1794 & \cdots \\ 4.4609 & 0.0845 & 0.6698 & \cdots \\ 2.6125 & 0.1091 & 0.3543 & \cdots \\ -4.2426 & 0.6241 & \times & \cdots \\ -1.3643 & \times & \times & \cdots \end{pmatrix},$$

where $j \geq 2$, and the symbol "$\times$" stands for entries depending on the further syndrome values that we do not have available, and do not need under the assumption that no more than three errors have occurred. If we carry out column eliminations to reduce C_j to a lower triangular form, we obtain

$$L_j = \begin{pmatrix} -2.0000 & 0 & 0 & \cdots \\ -2.4433 & 4.2153 & 0 & \cdots \\ 4.4609 & -5.3651 & 0 & \cdots \\ 2.6125 & -3.0824 & 0 & \cdots \\ -4.2426 & 5.8071 & \times & \cdots \\ -1.3643 & \times & \times & \cdots \end{pmatrix},$$

where $j \geq 2$. Note that the third column is all zero, so number k of errors equals 2. We then use the top left 3×3 part of B_5 and the top left 3×2 part of L_2 to solve equation (6.3):

$$0 = \begin{pmatrix} -2.0000 & -2.4433 & 4.4609 \\ 0 & 4.2153 & -5.3651 \end{pmatrix} \begin{pmatrix} 1.0 & 0 & 0.5 \\ 0 & 1.0 & 0 \\ 0 & 0 & 0.5 \end{pmatrix} \begin{pmatrix} \alpha_0 \\ \alpha_1 \\ 1 \end{pmatrix}$$

for the 3-vector $B_2\mathbf{a}$. This is a 2×2 system of equations for α_0, α_1, but we can solve directly for $B_2\mathbf{a}$. The results are

$$B_2^{(3)}\mathbf{a} = \begin{pmatrix} 0.3378 \\ 0.6364 \\ 0.5000 \end{pmatrix} \quad \text{and} \quad \mathbf{a} = \begin{pmatrix} -0.1622 \\ 0.6364 \\ 1.0000 \end{pmatrix}.$$

Thus the error locator polynomial is

$$q(x) = x^2 + 0.6364x - 0.1622.$$

We can find the zeroes of $q(x)$ directly, or substitute $B_2^{(3)}\mathbf{a}$ directly into (4.8) to obtain the vector of evaluations of the error locator polynomial at all the knots:

$$(G\,|\,F)^T B_2\mathbf{a} = \begin{pmatrix} \vdots & \vdots & \vdots \\ 1.0 & -0.9808 & 0.9238 \\ 1.0 & -0.8315 & 0.3827 \\ 1.0 & -0.5556 & -0.3827 \\ 1.0 & -0.1951 & -0.9238 \\ 1.0 & 0.1951 & -0.9238 \\ 1.0 & 0.5556 & -0.3827 \\ 1.0 & 0.8315 & 0.3827 \\ 1.0 & 0.9808 & 0.9238 \end{pmatrix} \begin{pmatrix} 0.3378 \\ 0.6364 \\ 0.5000 \end{pmatrix} = \begin{pmatrix} \vdots \\ 0.1756 \\ O(10^{-16}) \\ -0.2071 \\ -0.2483 \\ O(10^{-16}) \\ 0.5000 \\ 1.0583 \\ 1.4238 \end{pmatrix}.$$

The locations of the $O(10^{-16})$ entries indicate that the zeroes of the error locator polynomial are -0.8315 and -0.1951, which are the knots corresponding to the locations of the nonzero ω-values: ω_n and ω_{n+3}, which in turn are the errors in the last data item ξ_n and the third parity value π_2, respectively. We can then extract the corresponding columns from equation (1.6) to obtain the 2×2 system to solve for the ω-values:

$$\begin{pmatrix} -2.0000 \\ -2.4433 \end{pmatrix} = \begin{pmatrix} 1.0000 & 1.0000 \\ -0.8315 & 0.1951 \end{pmatrix} \begin{pmatrix} \omega_n \\ \omega_{n+3} \end{pmatrix}$$

yielding the result

$$\begin{pmatrix} \omega_n \\ \omega_{n+3} \end{pmatrix} = \begin{pmatrix} 2.0000 \\ -4.0000 \end{pmatrix}.$$

□

Example 2. We consider a second numerical example illustrating the nongeneric procedure and the permuted lower triangular structure (6.2). To simplify the exposition, we do not use any parity values and ignore conditioning issues. We suppose we have knots $x_j = j+1$, $j = 0, 1, 2, \cdots$, and polynomials $p_i(x) = x^i$, $i = 0, 1, 2, \cdots$. The matrix G is the ordinary Vandermonde matrix

$$G = \begin{pmatrix} 1 & 1 & 1 & 1 & \cdots \\ 1 & 2 & 3 & 4 & \cdots \\ 1 & 4 & 9 & 16 & \cdots \\ 1 & 8 & 27 & 64 & \cdots \\ \vdots & \vdots & \vdots & \vdots & \ddots \end{pmatrix}.$$

Suppose that we start with the syndrome vector

$$\mathbf{s} = (1, 0, 0, 0, -24, -240, -1560, -8400, \cdots)^T.$$

We have eight syndrome values, allowing up to four errors. Then the Krylov sequence would be generated by a "shift down" matrix yielding

$$C_4 = \begin{pmatrix} 1 & 0 & 0 & 0 & -24 \\ 0 & 0 & 0 & -24 & -240 \\ 0 & 0 & -24 & -240 & -1560 \\ 0 & -24 & -240 & -1560 & -8400 \\ -24 & -240 & -1560 & -8400 & \times \\ -240 & -1560 & -8400 & \times & \times \\ -1560 & -8400 & \times & \times & \times \\ -8400 & \times & \times & \times & \times \end{pmatrix}.$$

The Krylov matrix B_4 of (3.6) is just the first five columns of an identity matrix. When we attempt to reduce C_4 to the lower triangular form L_4 by column operations, we find that the second column $\mathbf{c}_1$ has three leading zeroes. Hence, we get

$$\mathbf{l}_1 = \mathbf{c}_1,$$

without any elimination at all, and furthermore we have two additional leading zero elements. So we generate the next two $\mathbf{l}$ vectors by

$$\mathbf{l}_2 = \mathbf{c}_2 \qquad \text{and} \qquad \mathbf{l}_3 = \mathbf{c}_3.$$

The next vector $\mathbf{l}_4$ is obtained by eliminating the first four elements of $\tilde{\mathbf{l}}_4$ (in this case the same as $\mathbf{c}_4$) by means of column operations. The result is

$$L_4 = (\mathbf{l}_0, \mathbf{l}_1, \mathbf{l}_2, \mathbf{l}_3, \mathbf{l}_4) = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & -24 & 0 \\ 0 & 0 & -24 & -240 & 0 \\ 0 & -24 & -240 & -1560 & 0 \\ -24 & -240 & -1560 & -8400 & \times \\ -240 & -1560 & -8400 & \times & \times \\ -1560 & -8400 & \times & \times & \times \\ -8400 & \times & \times & \times & \times \end{pmatrix}. \tag{7.4}$$

Since the last column is zero, the maximal rank k equals 4, so we have four errors. We then solve (4.1) for the coefficients $\mathbf{a}$ of the error locator polynomial $q(x)$. That is, we solve

$$\begin{pmatrix} 1 & 0 & 0 & 0 & -24 \\ 0 & 0 & 0 & -24 & -240 \\ 0 & 0 & -24 & -240 & -1560 \\ 0 & -24 & -240 & -1560 & -8400 \end{pmatrix} \mathbf{a} = \mathbf{0},$$

obtaining the error locator polynomial

$$q(x) = x^4 - 10x^3 + 35x^2 - 50x + 24.$$

The zeroes of q are 1, 2, 3, and 4, indicating that the errors occur in the first four positions. To find the actual errors, we extract the top left 4×4 part of (1.6):

$$\begin{pmatrix} 1 \\ 0 \\ 0 \\ 0 \end{pmatrix} = \begin{pmatrix} 1 & 1 & 1 & 1 \\ 1 & 2 & 3 & 4 \\ 1 & 4 & 9 & 16 \\ 1 & 8 & 27 & 64 \end{pmatrix} \begin{pmatrix} \omega_0 \\ \omega_1 \\ \omega_2 \\ \omega_3 \end{pmatrix},$$

and solve this to obtain the errors

$$\begin{pmatrix} \omega_0 \\ \omega_1 \\ \omega_2 \\ \omega_3 \end{pmatrix} = \begin{pmatrix} 4 \\ -6 \\ 4 \\ -1 \end{pmatrix}.$$

□

8. Acknowledgements. The authors acknowledge their research sponsors: D. L. Boley was supported in part by the National Science Foundation under grant CCR-8813493, G. H. Golub by the Army Research Office under contract DAAL03-90-G-0105, and F. T. Luk by the Army Research Office under contract DAAL03-90-G-0104. This paper is dedicated to the memory of Jeffrey Speiser; we thank him for his invaluable role in cross-fertilizing the fields of numerical analysis, scientific computing, and signal processing.

9. References.

[1] C. J. Anfinson and F. T. Luk, "A linear algebraic model of algorithm-based fault tolerance," *IEEE Transactions on Computers, Special Issue on Parallel and Distributed Algorithms*, Vol. C-37 (Dec. 1988), pp. 1599-1604.

[2] E. R. Berlekamp, *Algebraic Coding Theory*, McGraw-Hill, New York, NY, 1968.

[3] D. L. Boley, S. Elhay, G. H. Golub, M. H. Gutknecht, "Nonsymmetric Lanczos and finding orthogonal polynomials associated with indefinite weights," *Numerical Algorithms* Vol. 1 (1991), pp. 21-44.

[4] D. L. Boley, G. H. Golub, "The Lanczos algorithm and controllability," *System and Control Letters*, Vol. 4 (1984), pp. 317-324.

[5] D. L. Boley, G. H. Golub, S. Makar, N. Saxena and E. J. McCluskey, "Backward error assertions for checking solutions to systems of linear equations," Report NA-89-12, Computer Science Dept., Stanford Univ., Nov. 1989.

[6] R. P. Brent, F. T. Luk and C. J. Anfinson, "Choosing small weights for multiple error detection," *Proceedings of SPIE Vol. 1058, IS&T High Speed Computing II* (1989), pp. 130-136.

[7] R. P. Brent, F. T. Luk and C. J. Anfinson, "Checksum schemes for fault tolerant systolic computing," *Mathematics in Signal Processing II*, J. G. McWhirter, Ed., Clarendon Press, Oxford, 1990, pp. 791-804.

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

[9] K. H. Huang and J. A. Abraham, "Algorithm-based fault tolerance for matrix operations," *IEEE Trans. Comput.*, Vol. C-33 (June 1984), pp. 518-528.

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

[11] F. T. Luk, "Algorithm-based fault tolerance for parallel matrix equation solvers," *Proceedings of SPIE Vol. 564, Real Time Signal Processing VIII* (1985), pp. 49-53.

[12] F. T. Luk and H. Park, "An analysis of algorithm-based fault tolerance techniques," *Journal of Parallel and Distributed Computing*, Vol. 5 (1988), pp. 172-184.

[13] F. T. Luk and H. Park, "Fault-tolerant matrix triangularizations on systolic arrays," *IEEE Transactions on Computers*, Vol. C-37 (Nov. 1988), pp. 1434-1438.

[14] F. T. Luk, E. K. Torng and C. J. Anfinson, "A novel fault tolerance technique for recursive least squares minimization," *Journal of VLSI Signal Processing*, Vol. 1 (1989), pp. 181-188.

[15] J. L. Massey, "Shift register synthesis and BCH decoding," *IEEE Trans. Inform. Theory*, Vol. IT-15 (1967), pp. 122-127.

[16] V. S. S. Nair and J. A. Abraham, "Real-number codes for fault-tolerant matrix operations on processor arrays," *IEEE Trans. Comput., Special Issue on Fault-Tolerant Computing*, Vol. C-39 (April 1990), pp. 426-435.

[17] I. S. Reed and G. Solomon, "Polynomial codes over certain finite fields," *J. Soc. Ind. Appl. Math.*, Vol. 8 (1960), pp. 300-304.