NONSYMMETRIC LANCZOS AND FINDING ORTHOGONAL POLYNOMIALS ASSOCIATED WITH INDEFINITE WEIGHTS

Daniel L. BOLEY [1,*], Sylvan ELHAY [2], Gene H. GOLUB [3,**] and Martin H. GUTKNECHT [4]

[1] *Computer Science Department, University of Minnesota, Minneapolis, MN 55455, U.S.A.*
[2] *Computer Science Department, University of Adelaide, Adelaide, South Australia, 5000*
[3] *Computer Science Department, Stanford University, Stanford, CA 94305, U.S.A.*
[4] *Interdisciplinary Project Center for Supercomputing, ETH Zurich, ETH-Zentrum, CH-8092 Zurich, Switzerland*

Received 26 September 1990

The nonsymmetric Lanczos algorithm reduces a general matrix to tridiagonal by generating two sequences of vectors which satisfy a mutual bi-orthogonality property. The process can proceed as long as the two vectors generated at each stage are not mutually orthogonal, otherwise the process breaks down. In this paper, we propose a variant that does not break down by grouping the vectors into clusters and enforcing the bi-orthogonality property only between different clusters, but relaxing the property within clusters. We show how this variant of the matrix Lanczos algorithm applies directly to a problem of computing a set of orthogonal polynomials and associated indefinite weights with respect to an indefinite inner product, given the associated moments. We discuss the close relationship between the modified Lanczos algorithm and the modified Chebyshev algorithm. We further show the connection between this last problem and checksum-based error correction schemes for fault-tolerant computing.

Subject classifications: AMS (MOS): 42C05, 65F30, 68M15, 65D99.

Keywords: Orthogonal polynomials, modified Chebyshev algorithm, nonsymmetric Lanczos algorithm, based fault tolerance

1. Introduction

The Lanczos algorithm was originally proposed by Lanczos [21] as a method for the recursive computation of minimal polynomials for symmetric and nonsymmetric matrices. Soon it became viewed as an efficient means to reduce a general matrix to tridiagonal form, from which the result can be determined.

* The research reported by this author was supported in part by NSF grant CCR-8813493.
** The research reported by this author was supported in part by ARO grant DAAL03-90-G-0105 and in part by NSF grant DCR-8412314.

More recently the same algorithm has become popular as a method to compute some eigenvalues of large sparse matrices. In this context, for symmetric matrices, the Lanczos algorithm has been studied extensively [5,24]. In particular, the convergence of the algorithm, when used to compute eigenvalues, has been extensively analysed in [19,23,28,30] [32, pp270ff]. However, the nonsymmetric Lanczos algorithm has received much less attention. Besides some numerical stability problems, the method suffers from the possibility of a breakdown from which the only way to "recover" was to restart the whole process from the beginning with different starting vectors [32, pp388ff]. Methods to continue in the face of a breakdown have been proposed by [26], and more recently by [13,17,25]. In this paper, we propose a very similar way to continue the process in the face of a breakdown, but defined from a somewhat different point of view.

The connection between the Lanczos algorithm and orthogonal polynomials has also been studied extensively (e.g. [6,3,27,11,12,14,4,20]). In this paper, we take one particular problem in the area of orthogonal polynomials and reduce it to a matrix variant of the nonsymmetric Lanczos algorithm. The problem we address is that of computing a set of indefinite weights for a discrete "inner product" and the associated orthogonal polynomials given a set of initial moments or modified moments. It is well known that under certain (generic) normality assumptions the solution can be found with the *modified Chebyshev algorithm* for generating the recurrence coefficients for a sequence of orthogonal polynomials [7,29,8,20,31]. In [8] the same problem is addressed in the context of continous inner products and associated infinite sets of recurrence coefficients, and the nongeneric modified Chebyshev algorithm developed therein is equivalent to ours when the support of the weight function is just a finite set of points. We attempt to address this problem in the context of finite dimensional matrix relations, and show how a nonsymmetric Lanczos algorithm applied with a particular pair of initial vectors may be used to generate the polynomials orthogonal with respect to the unknown set of indefinite weights. The resulting method is a simple generalization of the "lower triangular Lanczos" algorithm of Kautsky and Golub [18].

Checksum schemes play a central role in Algorithm-based Fault Tolerant Computing, in which temporary hardware errors occurring during the factorization of a matrix (or related problems) can be detected and corrected without having to repeat the entire computation from scratch [15,16,22]. We use the simplest checksum scheme to illustrate the relation between it and the process of obtaining the weights from the moments of a sequence of polynomials. We show how a nonsymmetric Lanczos algorithm may be used to solve for the corrections given a set of checksums.

The rest of this paper is organized as follows. We first describe the proposed variant to the nonsymmetric Lanczos algorithm, then discuss the connection to the problem of reconstruction of a sequence of orthogonal polynomials and the associated indefinite weights, and then discuss the connection to the checksum-based error correction problem. We conclude with a short discussion of some numerical examples (collected in the appendices) and some conclusions.

2. Nonsymmetric Lanczos

We describe a modified nonsymmetric Lanczos process which recovers when the left and right vectors are orthogonal, exactly the situation when the ordinary nonsymmetric Lanczos process breaks down and must be restarted. We refer the reader to [32, pp388ff] for the details for the standard nonsymmetric Lanczos process upon which the following development is based. To orient the reader to what follows, we summarize the process as follows. We are given a matrix $\boldsymbol{A}$ and two initial vectors $\boldsymbol{x}_0$ and $\boldsymbol{y}_0$. In the standard process, we assume that these two initial vectors are not mutually orthogonal. Assuming it does not break down, the process generates a set of vectors $\boldsymbol{X} = [\boldsymbol{x}_0, \ldots, \boldsymbol{x}_{n-1}]$ and $\boldsymbol{Y} = [\boldsymbol{y}_0, \ldots, \boldsymbol{y}_{n-1}]$ and a tridiagonal matrix $\boldsymbol{H}$ with unit subdiagonal such that

$$\boldsymbol{A}\boldsymbol{X} = \boldsymbol{X}\boldsymbol{H}, \tag{1a}$$

$$\boldsymbol{A}^{\mathrm{T}}\boldsymbol{Y} = \boldsymbol{Y}\boldsymbol{H}, \tag{1b}$$

$$\boldsymbol{Y}^{\mathrm{T}}\boldsymbol{X} = \boldsymbol{D} \text{ (a nonsingular diagonal matrix)}. \tag{1c}$$

Let $\boldsymbol{X}_r \equiv [\boldsymbol{x}_0, \ldots, \boldsymbol{x}_{r-1}]$ denote the first r columns of $\boldsymbol{X}$ and $\boldsymbol{Y}_r$ denote the first r columns of $\boldsymbol{Y}$. At the r-th stage of the process, the vectors $\boldsymbol{x}_r$ and $\boldsymbol{y}_r$ are generated. The vector $\boldsymbol{x}_r$ is generated by forming $\boldsymbol{b}_r = \boldsymbol{A}\boldsymbol{x}_{r-1}$, and then setting $\boldsymbol{x}_r = \boldsymbol{b}_r - [\boldsymbol{x}_0, \ldots, \boldsymbol{x}_{r-1}]\boldsymbol{h}_{r-1}$, where the coefficients $\boldsymbol{h}_{r-1}$ are chosen so that the bi-orthogonality condition $\boldsymbol{x}_r^{\mathrm{T}}[\boldsymbol{y}_0, \ldots, \boldsymbol{y}_{r-1}] = 0$ holds. The vector $\boldsymbol{y}_r$ is computed analogously. In exact arithmetic, only the last two pairs of vectors $\boldsymbol{x}_{r-2}$, $\boldsymbol{y}_{r-2}$ and $\boldsymbol{x}_{r-1}$, $\boldsymbol{y}_{r-1}$ enter into the computation of the next pair of vectors $\boldsymbol{x}_r$, $\boldsymbol{y}_r$. We could rescale the vectors $\boldsymbol{x}_r$ and $\boldsymbol{y}_r$ to make $\boldsymbol{y}_r^{\mathrm{T}}\boldsymbol{x}_r = 1$, so that $\boldsymbol{D} = \boldsymbol{I}$ (in which case, the $\boldsymbol{H}$ in (1b) must be replaced by $\boldsymbol{H}^{\mathrm{T}}$). We have chosen not to do so in this exposition so that the polynomials that are generated by this algorithm in the next section remain monic and to keep the exposition simple. However, such scaling would enhance the numerical stability of the algorithm.

When this algorithm is used for its original purpose to find eigenvalues, the algorithm reduces $\boldsymbol{A}$ to tridiagonal form $\boldsymbol{H}$, from which the eigenvalues can be more easily determined. The generated vectors $\boldsymbol{X}$, $\boldsymbol{Y}$ can provide information about the eigenspaces of $\boldsymbol{A}$, although in practice this is not made use of due to memory limitations. The generalization we propose below can also be used for the same purpose, but beyond noting that it does reduce a matrix to a block tridiagonal matrix $\boldsymbol{H}$, we do not address this further in this paper.

The standard Lanczos process breaks down if at any stage the two computed vectors $\boldsymbol{x}_r$, $\boldsymbol{y}_r$ are mutually orthogonal. At this stage, some recovery procedure is required. In [32, pp388ff], the only recovery procedure proposed was to restart the whole process from scratch with different starting vectors. In this case, all the computation up to this point must be thrown away. In many cases, such as in the next section, the starting vectors are fixed by the problem and cannot be changed. Thus some method of recovery is essential.

In our generalized Lanczos method, the algorithm groups the vectors $\boldsymbol{X}$ and $\boldsymbol{Y}$ into clusters $\boldsymbol{X} = [X_1, \ldots, X_m]$ and $\boldsymbol{Y} = [Y_1, \ldots, Y_m]$, where the clusters satisfy the orthogonality conditions

$$Y_i^{\mathrm{T}} X_j = 0 \text{ for } i \neq j, \tag{2a}$$

$$Y_i^{\mathrm{T}} X_i \equiv D_i \text{ is nonsingular, except maybe for the last cluster.} \tag{2b}$$

As each new vector is generated, the algorithm determines whether it is appended to the last cluster X_k, Y_k, or else the new vector starts a new cluster, the former cluster being considered *complete*. This choice is determined at each step based on whether condition (2b) is not satisfied, or is satisfied, respectively, at the current stage. Under normal operation, each cluster consists of exactly one vector, so that the algorithm is identical to the standard Lanczos process. At each stage the two new vectors are generated as follows. As before, $\boldsymbol{b}_r$ and $\boldsymbol{c}_r$ are formed by applying $\boldsymbol{A}$ to the last generated vector $\boldsymbol{x}_{r-1}$ and $\boldsymbol{A}^{\mathrm{T}}$ to the last generated $\boldsymbol{y}_{r-1}$, respectively. Next a pair of vectors $\boldsymbol{h}_{r-1}$, $\boldsymbol{g}_{r-1}$ is found such that $\boldsymbol{b}_r - [X_1, \ldots, X_k]\boldsymbol{h}_{r-1}$ is orthogonal to $[Y_1, \ldots, Y_k]$ and $\boldsymbol{c}_r - [Y_1, \ldots, Y_k]\boldsymbol{g}_{r-1}$ is orthogonal to $[X_1, \ldots, X_k]$, where X_i, Y_k denote the clusters containing the last set of $\boldsymbol{x}$, $\boldsymbol{y}$ vectors generated, which may or may not be completed. If $D_k \equiv Y_k^{\mathrm{T}} X_k$ is nonsingular, such a $\boldsymbol{h}_{r-1}$ and $\boldsymbol{g}_{r-1}$ can be found. In this case the clusters X_k, Y_k are ended, and the new vectors $\boldsymbol{x}_r = \boldsymbol{b}_r - [\boldsymbol{x}_0, \ldots, \boldsymbol{x}_{r-1}]\boldsymbol{h}_{r-1}$, $\boldsymbol{y}_r = \boldsymbol{c}_r - [\boldsymbol{y}_0, \ldots, \boldsymbol{y}_{r-1}]\boldsymbol{g}_{r-1}$ start new clusters X_{k+1}, Y_{k+1}, respectively.

The breakdown in the standard algorithm corresponds to the situation where the diagonal block D_k is singular. In this case, the new vector $\boldsymbol{x}_r$ is chosen in the following manner and does not start a new cluster. The vector $\boldsymbol{b}_r$ is projected onto the orthogonal complement of the last cluster X_k by forming $\hat{\boldsymbol{b}}_r = \boldsymbol{b}_r - X_k X_k^+ \boldsymbol{b}_r$, where $X_k X_k^+$ is the orthogonal projector onto the space $\textsc{colsp}\, X_k$. Then the coefficients $\hat{\boldsymbol{h}}_{r-1}$ are computed so that $\hat{\boldsymbol{b}}_r - [X_1, \ldots, X_{k-1}]\hat{\boldsymbol{h}}_{r-1}$ is orthogonal to $[Y_1, \ldots, Y_{k-1}]$. The new vector is then $\boldsymbol{x}_r = \hat{\boldsymbol{b}}_r - [X_1, \ldots, X_{k-1}]\hat{\boldsymbol{h}}_{r-1}$. The result is that the new vector $\boldsymbol{x}_r$ is orthogonal to $[Y_1, \ldots, Y_{k-1}]$ as in the standard process, but, unlike the standard process, it is orthogonal to X_k instead of Y_k.

The vector $\boldsymbol{y}_r$ is generated in the analogous way, by forming $\boldsymbol{c}_r = \boldsymbol{A}^{\mathrm{T}} \boldsymbol{y}_{r-1}$, projecting it onto the orthogonal complement of Y_k to obtain $\hat{\boldsymbol{c}}_r$, and subtracting multiples of columns $[Y_1, \ldots, Y_{k-1}]$ so that the final resulting vector $\boldsymbol{y}_r = [\boldsymbol{I} - Y_k Y_k^+)\boldsymbol{c}_r - [Y_1, \ldots, Y_{k-1}]\hat{\boldsymbol{g}}_{r-1}$ is orthogonal to $[X_1, \ldots, X_{k-1}]$ and Y_k. Here $\hat{\boldsymbol{g}}_{r-1}$ is a vector of coefficients analogous to $\hat{\boldsymbol{h}}_{r-1}$.

After the vectors $\boldsymbol{x}_r$, $\boldsymbol{y}_r$ are determined in this manner, they are appended to the last clusters X_k, Y_k, respectively. In this way the clusters Y_k, X_k are extended together until $D_k \equiv Y_k^{\mathrm{T}} X_k$ becomes nonsingular, when both clusters are ended and new ones started. Normally this latter condition happens at every stage. The resulting set of vectors will satisfy

$$AX = XH, \tag{3a}$$

$$A^{\mathrm{T}}Y = YG, \tag{3b}$$

$$Y^{\mathrm{T}}X = D, \text{ a block diagonal matrix,} \tag{3c}$$

where H and G are unit upper Hessenberg matrices. Since $Y^{\mathrm{T}}AX = Y^{\mathrm{T}}XH = DH$, and $X^{\mathrm{T}}A^{\mathrm{T}}Y = X^{\mathrm{T}}YG = D^{\mathrm{T}}G$, we have the relation $G^{\mathrm{T}}D = DH$. Since a block diagonal matrix times an upper Hessenberg matrix is block upper Hessenberg, it follows that G and H are block tridiagonal, with the partitioning defined by the cluster dimensions. This implies that in computing the coefficients h_{r-1}, g_{r-1} or $\hat{h}_{r-1}$, $\hat{g}_{r-1}$ at each stage, only the last two pairs of clusters X_i, Y_i, $i = k-1$, k, must be used, at least in exact arithmetic. From [13] it follows moreover that on completion of a cluster, only the first pair of vectors from the clusters X_{k-1}, Y_{k-1} contribute to x_r and y_r, while before completion of a cluster, none of the vectors from X_{k-1}, Y_{k-1} contribute, cf. the example in appendix 2.

The vectors x_{r-1} and y_{r-1} are generated in this manner until one of them is found to be all zero. When this happens, we could choose to continue with the zero vector, or with a randomly chosen b_r or c_r and re-orthogonalize, until both x_r, y_r are zero, for some r. However, for our application below, it suffices to terminate the process when just one vector is zero. We summarize the entire process in appendix 1.

Let $K_r \equiv [x_0, \ldots, A^{r-1}x_0]$ be the Krylov sequence generated by x_0 and $L_r \equiv [y_0, \ldots, (A^{\mathrm{T}})^{r-1}y_0]$ be the Krylov sequence generated by y_0. We note that each x_r is a linear combination of Ax_{r-1}, $x_0, \ldots, x_{r-1}$, for all r, regardless of how the vectors are grouped into clusters. Hence $\text{COLSP}X_r = \text{COLSP}K_r$ and $\text{COLSP}Y_r = \text{COLSP}L_r$ for every r. Hence if r is any index for which the matrix $L_r^{\mathrm{T}}K_r$ is nonsingular, then x_r, y_r are vectors that start a new pair of clusters, and they are the unique vectors defined (up to scaling) by $x_r = [I - K_r(L_r^{\mathrm{T}}K_r)^{-1}L_r^{\mathrm{T}}]Ax_{r-1}$ and $y_r = [I - L_r(K_r^{\mathrm{T}}L_r)^{-1}K_r^{\mathrm{T}}]A^{\mathrm{T}}y_{r-1}$. As noted above, the other vectors must be chosen so that all the vectors in each cluser are independent, and we have chosen (arbitrarily) to make the vectors in each cluster mutually orthogonal. Thus we conclude that the first vector in each cluster is uniquely defined up to a scale factor, but the remaining vectors are not uniquely defined.

There is some freedom in the choice of $\hat{b}_r$, $\hat{c}_r$, and at least one choice leads to a very special structure in the block diagonal matrix D. As we have described it, $\hat{b}_r$ is chosen orthogonal to the part of the last cluster X_k generated up to that point. But the modified Lanczos process can work if $\hat{b}_r$ is any linear combination of $b_r \equiv Ax_{r-1}$ and columns of X_k as long as the chosen vector is linearly independent of X_k. Unless $b_r \in \text{COLSP}X_k$ (in which case $x_r = 0$), it is necessary and sufficient that b_r contribute to this linear combination. In particular, we could choose $\hat{b}_r = b_r$, after checking that it is independent of X_k. Such a choice leads to a very particular structure in the diagonal blocks D_k, when they are bigger than

1×1, which we now demonstrate. Suppose y_j, y_{j+1} are two consecutive vectors in cluster Y_k,k and x_{r-1}, x_r are in the corresponding cluster X_k. Suppose further that we choose $\hat{b}_{i+1} = b_{i+1}$ and $\hat{c}_{i+1} = c_{i+1}$ at every step i. Then the matrix element $D_{j,r}$ lies within the diagonal block D_k and satisfies the identity:

$$D_{j,r} = y_j^{\mathrm{T}} x_r = y_j^{\mathrm{T}}\left(b_r - [X_1, \ldots, X_{k-1}]\hat{h}_{r-1}\right) = y_j^{\mathrm{T}} b_r = y_j^{\mathrm{T}} A x_{r-1}.$$

We also have a similar identity for the element $D_{j+1,r-1}$, which also lies within the diagonal block D_k:

$$D_{j+1,r-1} = y_{j+1}^{\mathrm{T}} x_{r-1} = \left(c_{j+1} - [Y_1, \ldots, Y_{k-1}]\hat{g}_j\right)^{\mathrm{T}} x_{r-1} = c_{j+1}^{\mathrm{T}} x_{r-1} = y_j^{\mathrm{T}} A x_{r-1}.$$

Hence $D_{j+1,r-1} = D_{j,r}$. Since this is true for any j, r for which the indicated elements of D lie within the same diagonal block, it follows that the diagonal blocks D_k are Hankel matrices. Furthermore, since *every* leading principle submatrix of D_k is singular, it also follows that D_k is lower anti-triangular, by which we mean that all entries above the main anti-diagonal are zero. We repeat that this is under the assumption that the vectors within each individual cluster are not mutually orthogonalized. In our implementation, we have chosen not to make this choice, but rather we have chosen to orthogonalize the vectors within each cluster in order to enhance numerical independence. Then D_k is still lower anti-triangular with identical elements on the anti-diagonal. Yet other choices are discussed in [13]; in particular, by redefining the elements of X_k, Y_k after completing the cluster one can make D_k an anti-diagonal unit matrix.

3. Indefinite weights and polynomials from the moments

We show how the nonsymmetric Lanczos process may be used to solve the problem of computing a sequence of polynomials h_0, $h_1, \ldots$ orthogonal with respect to some unknown inner product given only the generalized moments defined in terms of another known sequence of polynomials t_0, $t_1, \ldots$. Specifically, we are given a sequence of n monic polynomials $t^{\mathrm{T}}(z) \equiv [t_0(z), \ldots, t_{n-1}(z)]$ of exact degrees generated by the recurrence $zt^{\mathrm{T}}(z) = t^{\mathrm{T}}(z)T + t_n(z)e_n^{\mathrm{T}}$, where T is the matrix of *recurrence coefficients*. Here $e_n^{\mathrm{T}} \equiv [0, \ldots, 0, 1]$ is the n-th coordinate unit vector. Since all the polynomials are monic and of exact degree, T is a unit upper Hessenberg matrix (i.e., the subdiagonal entries are all 1's). We call T the *recurrence matrix* for the sequence of polynomials $t^{\mathrm{T}}(z)$. If T is given, the n-th degree polynomial $t_n(z)$ is the monic polynomials whose zeroes $z_0, \ldots, z_{n-1}$ are the eigenvalues of T. If the polynomial $t_n(z)$ is given, the last column of T can be chosen so that the eigvenvalues of T are exactly the zeroes $z_0, \ldots, z_{n-1}$ of $t_n(z)$, without affecting any of the preceding polynomials $t_0, \ldots, t_{n-1}$. We define the vector of zeroes $z \equiv [z_0, \ldots, z_{n-1}]$.

Next, we are given the n *initial moments* $s_{i0} \equiv \langle t_i, t_0 \rangle_w$, $i = 0, \ldots, n-1$, where $\langle \cdot, \cdot \rangle_w$ is the discrete indefinite inner product defined in terms of some unknown, but arbitrary, set of weights $w \equiv [w_0, \ldots, w_{n-1}]$ over the given set of points

z. We wish to compute another sequence of monic polynomials $\boldsymbol{h}^{\mathrm{T}}(z) = [h_0(z), \ldots, h_{n-1}(z)]$ of exact degree which are orthogonal with respect to this unknown set of weights, and then to determine the weights themselves. In this particular situation where we are given n points and n initial moments, the weights are uniquely determined, but below we discuss the case where we are given fewer moments, in which case an additional condition is imposed to uniquely determine the weights. It is not always possible to find such a sequence $\boldsymbol{h}^{\mathrm{T}}(z)$, but in any case we would like to find some sequence $\boldsymbol{h}^{\mathrm{T}}(z)$ such that $h_0, \ldots, h_{i-1}$ are all orthogonal (*wrt* to this inner product) to $h_i, \ldots, h_{n-1}$, for any i for which this is possible (a detailed discussion of this can be found in [8]).

Since the polynomials h_i are all monic and of exact degree we have that $h_0(z) = t_0(z) = 1$. Analogous to the relations for $\boldsymbol{t}$, we have the relations $z\boldsymbol{h}^{\mathrm{T}}(z) = \boldsymbol{h}^{\mathrm{T}}(z)\boldsymbol{H} + h_n(z)\boldsymbol{e}_n^{\mathrm{T}}$, where $\boldsymbol{H}$ is the unit upper Hessenberg recurrence matrix for the polynomial sequence $\boldsymbol{h}^{\mathrm{T}}(z)$. We assume that $h_n(z)$ is also the polynomial whose zeroes are exactly the points z (this is always possible), that is $h_n = t_n = \prod(z - z_i)$. In the case where both sequences $\boldsymbol{t}$ and $\boldsymbol{h}$ are orthogonal to some ordinary definite inner product, both $\boldsymbol{T}$ and $\boldsymbol{H}$ are tridiagonal, and the method we propose is then equivalent to the "LTL" algorithm [18] for generating the matrix $\boldsymbol{H}$ directly from $\boldsymbol{T}$.

Though the matrix $\boldsymbol{T}$ can represent an arbitrary sequence of n monic polynomials, two particular sequences are typically encountered. Often the monomials $1, z, z^2, \ldots, z^{n-1}$ are used. In this case, the recurrence matrix $\boldsymbol{T}$ is the upper Hessenberg companion matrix corresponding to $t_n(z)$, whose zeros are exactly the points $z_0, \ldots, z_{n-1}$. Another typical sequence is the set of polynomials orthogonal with respect to ordinary definite continuous inner product defined over some interval. In this case, it is well known that the sequence $\boldsymbol{t}$ is generated by a three term recurrence, so $\boldsymbol{T}$ is a tridiagonal matrix [10,7].

In either case, we can select the first $n+1$ such polynomials, for any arbitrary value of n depending only on the number of polynomials we wish to compute or the number of moments we may know. In the case of the second choice of the polynomials orthogonal with respect to a continuous inner product, it is known that these $n+1$ polynomials will also be mutually orthogonal with respect to a discrete inner product defined over a finite collection of points and weights which are, respectively, the eigenvalues and the squares of the first components of the normalized eigenvectors for the $n \times n$ matrix $\boldsymbol{T}$ [10]. This matrix $\boldsymbol{T}$ can be generated by the symmetric Lanczos algorithm [2,3].

We define the *generalized Vandermonde matrices* $\boldsymbol{V_T}$ and $\boldsymbol{V_H}$, whose i, j-th entries are, respectively, $t_j(z_i)$ and $h_j(z_i)$ for i, $j = 0, \ldots, n-1$:

$$\boldsymbol{V_T} = \begin{bmatrix} \boldsymbol{t}^{\mathrm{T}}(z_0) \\ \vdots \\ \boldsymbol{t}^{\mathrm{T}}(z_{n-1}) \end{bmatrix}, \quad \boldsymbol{V_H} = \begin{bmatrix} \boldsymbol{h}^{\mathrm{T}}(z_0) \\ \vdots \\ \boldsymbol{h}^{\mathrm{T}}(z_{n-1}) \end{bmatrix}. \tag{4}$$

If we define the diagonal matrix $Z \equiv \text{DIAG}(z_0, \ldots, z_{n-1})$, then V_T satisfies $ZV_T = V_T T$. (Recall that $t_n(z_i) = h_n(z_i) = 0$.) Furthermore V_T is nonsingular. And we have the analogous relation $ZV_H = V_H H$.

The two sequences t and h are related by $h_j(z) = t_j(z) + t_{j-1}(z)u_{j-1,j} + \ldots + t_0(z)u_{0j}$, for $j = 0, 1, \ldots, n-1$, and scalars u_{ij}. We can group all the u_{ij}'s into a unit upper triangular matrix U, and collect all these relations into one matrix relation $h^{\mathrm{T}} = t^{\mathrm{T}}U$. In terms of the Vandermonde matrices: $V_H = V_T U$. We can define the *Gram matrix* of *mixed moments* $S \equiv V_T^{\mathrm{T}} W V_H$, where $W \equiv \text{DIAG}(w_0, \ldots, w_{n-1})$. Notice that the first column s_0 of S is exactly the given data $[s_{00}, \ldots, s_{n-1,0}]^{\mathrm{T}}$, since $h_0 = t_0 = 1$.

We can construct some relations among the various matrices that will be seen to be exactly those appearing in the Lanczos process. First we see that $ZV_H = V_H H = V_T U H$ and $ZV_H = ZV_T U = V_T T U$. Hence, $TU = UH$.

We also have that $T^{\mathrm{T}}S = T^{\mathrm{T}}V_T^{\mathrm{T}}WV_H = V_T^{\mathrm{T}}ZWV_H$. Since Z and W are both diagonal and hence commute, we can continue the equality with $V_T^{\mathrm{T}}WZV_H = V_T^{\mathrm{T}}WV_H H = SH$.

Finally we have that the matrix $S^{\mathrm{T}}U = V_H^{\mathrm{T}}WV_T U = V_H^{\mathrm{T}}WV_H$ is just the Gram matrix of moments of the polynomials h_i, and by assumption this matrix is supposed to be diagonal, or if not possible at least block diagonal.

We summarize these relations as follows:

$TU = UH$, (5a)

$T^{\mathrm{T}}S = SH$, (5b)

$S^{\mathrm{T}}U = D$, a diagonal or block diagonal matrix, (5c)

the first column s_0 of S is the vector of given initial modified moments

$[s_{00}, \ldots, s_{n-1,0}]^{\mathrm{T}}$, (5d)

the first column u_0 of U is $e_1 = [1, 0, \ldots, 0]^{\mathrm{T}}$, (5e)

T is given. (5f)

These are just the right ingredients to run the nonsymmetric Lanczos process starting with the matrix $A = T$ and initial vectors $x_0 = e_1$, $y_0 = s_0$, to generate U, S and H, corresponding to the X, Y, H, in the notation of the previous section. The choice $x_0 = e_1$ is not artificial, it expresses the initial condition relating the two sequences t and h, namely that $h_0 = t_0$. In the case that a diagonal D exists, the generated vectors X and Y will be exactly the U and S. Otherwise the vectors X will still be a unit upper triangular matrix U, which defines a sequence of polynomials $h^{\mathrm{T}} \equiv t^{\mathrm{T}}U$. However, Y will not be exactly the matrix S. But it will still be true that for every i, the first i columns of Y and of S will span the same space, namely the Krylov space $\text{COLSP}[s_0, T^{\mathrm{T}}s_0, \ldots, (T^{\mathrm{T}})^{i-1}s_0]$. Since the first column y_{i_k} of each generated Lanczos cluster Y_k is uniquely defined up to scaling, the corresponding column s_{i_k} of S must be the same, up to scaling. Hence the orthogonality conditions that $[x_0, \ldots, x_{i_{k-1}}]$ is orthogonal to

$[\boldsymbol{x}_{i_k}, \ldots, \boldsymbol{x}_{n-1}]$ will still be valid for each index i_k denoting the first column of a cluster.

The matrix $\boldsymbol{U}$ defines the polynomials $h_i(z)$ in terms of the given polynomials $t_i(z)$, $i = 0, 1, \ldots,$ so the Lanczos process has, in effect, generated the polynomials $h_i(z)$, as illustrated by the example in appendix 3. Once the $\boldsymbol{U}$ is known, one can solve for the weights themselves using the first column of the relation $\boldsymbol{S} = \boldsymbol{V}_{\boldsymbol{T}}^{\mathrm{T}} \boldsymbol{W} \boldsymbol{V}_{\boldsymbol{H}}$, namely

$$s_0 = \boldsymbol{V}_{\boldsymbol{T}}^{\mathrm{T}} \boldsymbol{w}. \tag{6}$$

Note that (6) can be used directly without recourse to the Lanczos algorithm, but the above discussion shows how the polynomial sequence $\boldsymbol{h}^{\mathrm{T}}(z)$ can be generated in terms of $\boldsymbol{t}^{\mathrm{T}}(z)$ directly from the moments without computing the corresponding weights at all.

In the case that only l of the weights are nonzero, the Lanczos process will terminate at the l-th stage. This can be seen because the polynomials h_l, $h_{l+1}, \ldots$ will all be zero on those points with nonzero weights, hence the corresponding columns of $\boldsymbol{S}$ will be entirely zero. Let $\boldsymbol{U}_l$, $\boldsymbol{S}_l$ denote the partial $\boldsymbol{U}$, $\boldsymbol{S}$, respectively, generated in this case (i.e. columns $0, \ldots, l$). The Lanczos process has generated the first $l+1$ polynomials $h_i(z)$, $i = 0, \ldots, l$. We can compute the first $l+1$ columns of $\boldsymbol{V}_H$ as $\boldsymbol{V}_{\boldsymbol{T}} \boldsymbol{U}_l$. The last column of $\boldsymbol{V}_{\boldsymbol{T}} \boldsymbol{U}_l$ will be the vector $[h_l(z_0), \ldots, h_l(z_{n-1})]^{\mathrm{T}}$ and have zero entries exactly on those points with nonzero weights. Thus one can determine from this last column which points have nonzero weights.

Knowing which entries of $\boldsymbol{w}$ are nonzero, we can delete the zero entries of $\boldsymbol{w}$ together with the corresponding rows of $\boldsymbol{V}_{\boldsymbol{T}}$ in formula (6). We are then left with n equations in l unknowns. We can then select just the first l of these equations to get an $l \times l$ system of linear equations for the l nonzero weights $w_{i_1}, \ldots, w_{i_l}$:

$$\begin{bmatrix} s_{00} \\ \cdot \\ \vdots \\ \cdot \\ s_{l-1,0} \end{bmatrix} = \left(\boldsymbol{V}_{\boldsymbol{T}}^{\mathrm{T}} \right)_{i_1 \ldots i_l}^{0 \ldots l-1} \begin{bmatrix} w_i \\ \cdot \\ \vdots \\ \cdot \\ w_{i_l} \end{bmatrix}. \tag{7}$$

In the above, the notation $(\boldsymbol{V}_{\boldsymbol{T}}^{\mathrm{T}})_{i_1 \ldots i_l}^{0 \ldots l-1}$ denotes the submatrix of $\boldsymbol{V}_{\boldsymbol{T}}^{\mathrm{T}}$ obtained by extracting rows $0 \ldots l-1$ and columns $i_1 \ldots i_l$. It is easily verified that this submatrix is nonsingular. Note that in (7), only the first l mixed moments appear.

If it is known in advance that the number l of nonzero weights is at most some given number d, then the Lanczos process is guaranteed to terminate in at most d steps, and only the first $d+1$ rows and columns of $\boldsymbol{S}$ are required to solve for the weights. But the top left $(d+1) \times (d+1)$ parts of $\boldsymbol{S}$ and $\boldsymbol{U}$ are completely determined through the Lanczos process by the first $m \equiv 2d+1$ initial moments $s_{00}, \ldots, s_{2d,0}$, due to the fact that the generating matrix $\boldsymbol{T}$ in the Lanczos process is upper Hessenberg and the generated $\boldsymbol{U}$ is upper triangular. Hence if only the first m moments are known, one can fill out the remaining entries in the initial vector s_0 with arbitrary *fill* values, and then carry out the Lanczos process as before. In this case, the Lanczos process should be terminated when the first $d+1$ entries of a generated y_l are all zero, corresponding to the first $d+1$ entries of column s_l being all zero. If l weights are nonzero, this condition cannot occur before the l-th stage, since the leading top left $l \times l$ block of $\boldsymbol{S}$ will be nonsingular. The matrix U_l generated by the Lanczos process is still upper triangular and hence is independent of the choice of fill values for s_0. In fact, only the first m entries in each generated Lanczos vector need be computed, and only the top left $m \times m$ part of $\boldsymbol{T}$ need be referenced. Thus the generated U_l can be used to solve for the weights in exactly the same way as outlined above. If any multiple vector clusters arise, it is necessary to relax the orthogonality condition within the $\boldsymbol{S}$ (but not the $\boldsymbol{U}$) clusters to linear independence, by enforcing orthogonality among the first m entries of each generated s vector, but this does not affect the generated $\boldsymbol{U}$ matrix.

If there is an infinite sequence of modified moments s_{i0}, $i = 0, 1, 2, \ldots,$ arising from a continuous formal inner product, the same argument can be used to show that the Lanczos algorithm can be applied using only the first $m = 2d+1$ modified moments to obtain the first d formal orthogonal polynomials h_j, $j = 0, \ldots, d-1$ with respect to this formal inner product. This is discussed further in the next section.

4. Equivalence of the Lanczos and the modified Chebyshev algorithms

The problem of the previous section can also be solved with the modified Chebyshev algorithm [7,8,20,29,31] or, in cases where the classical algorithm breaks down, by its nongeneric extension [8]. This approach and the connection between the nonsymmetric Lanczos and the modified Chebyshev algorithms will be discussed in this section.

Some relations between the two algorithms have been observed before, e.g. in [20]. Clearly, the two methods produce recurrence coefficients of formally orthogonal polynomials. But while the Lanczos algorithm has as input an $n \times n$ matrix $\boldsymbol{A}$ and two n-vectors $\boldsymbol{x}_0$ and $\boldsymbol{y}_0$, the Chebyshev algorithm starts with a set of initial modified moments s_{i0} and a corresponding recurrence matrix $\boldsymbol{T}$. (The unmodified moments are the special case of modified moments where $\boldsymbol{T}$ is the downshift matrix with ones on the first subdiagonal and zeroes elsewhere.) Normally it takes $2n$ modified moments s_{i0}, $i = 0, \ldots, 2n-1$, to determine the n columns of the normalized tridiagonal matrix $\boldsymbol{H}$. The $n \times n$ lower triangular matrix $\boldsymbol{S}$ is therefore extended to a $2n \times n$ matrix $\bar{\boldsymbol{S}}$, of which the modified Chebyshev algorithm generates recursively column by column along with the columns of $\boldsymbol{H}$. Actually only the elements s_{ik}, $i = k,\ k+1, \ldots, 2n-k-1$ of column k $(k = 0, 1, \ldots, n-1)$ are computed, since $s_{ik} = 0$ if $i < k$, while for $i \geqslant 2n-k$ the moment $s_{2n,0}$ would be needed. (For simplicity we assume the normal (generic) case where the algorithm does not break down, but our argu-

ments extend to the general case.) If instead Lanczos input is given, one can choose any $2n \times 2n$ recurrence matrix $\boldsymbol{T}$ (unit Hessenberg), compute the corresponding polynomials t_i, $i = 0, \ldots, 2n-1$, and the modified moments
$s_{i0} \equiv \boldsymbol{y}_0^{\mathrm{T}} t_i(\boldsymbol{A}) \boldsymbol{x}_0, \quad i = 0, \ldots, 2n-1.$
Started with these data the modified Chebyshev algorithm produces then exactly the same $\boldsymbol{H}$ and thus the same polynomials $h_0, \ldots, h_{n-1}$, h_n as one obtains with the Lanczos algorithm. If $\boldsymbol{A}$ is a nonderogatory matrix of rank n, and $\boldsymbol{x}_0$ and $\boldsymbol{y}_0$ are not unluckily chosen, h_n is the minimal polynomial of $\boldsymbol{A}$.

However, in order to prove that the two algorithms are equivalent, we need conversely to be able to specify for given modified moments $s_{00}, \ldots, s_{2n-1,0}$ and given recurrence matrix $\boldsymbol{T}$ some Lanczos input $\boldsymbol{A}$, $\boldsymbol{x}_0$, $\boldsymbol{y}_0$ that will produce the same $\boldsymbol{H}$.

The situation of the previous section is special, since there the set of zeroes $z_0, \ldots, z_{n-1}$ of t_n is prescribed, and the support of the weight function is a subset of it. We can then assume that $t_{n+i}(z) = z^i t_n(z)$, $i = 0, 1, \ldots$, and thus $s_{n0} = s_{n+1,0} = \cdots = s_{2n-1,0} = 0$ since $s_{i0} \equiv \langle t_i, t_0 \rangle_w$. Application of the modified Chebyshev algorithm to these data yields the $n \times n$ matrices $\boldsymbol{S}$ and $\boldsymbol{H}$. (All additional elements of $\bar{\boldsymbol{S}}$ that occur in the formulas are zero.) But the first $n/2$ columns of $\boldsymbol{H}$ are independent of the choice $s_{i0} = 0$, $i = n, n+1, \ldots$, since these first $n/2$ columns of $\boldsymbol{H}$ are normally determined by the first n moments, as noted above.

As suggested in the previous section, cf. (5), we can then apply the nonsymmetric Lanczos algorithm with $\boldsymbol{A} \equiv \boldsymbol{T}$, $\boldsymbol{x}_0 \equiv \boldsymbol{e}_1$, $\boldsymbol{y}_0 \equiv \boldsymbol{s}_0$ to generate $\boldsymbol{H}$. Like the modified Chebyshev algorithm it generates $\boldsymbol{Y} \equiv \boldsymbol{S}$ and $\boldsymbol{H}$ column by column, but it produces additionally $\boldsymbol{X} \equiv \boldsymbol{U}$. Note that this Lanczos problem has then a very special structure. Since $\boldsymbol{x}_0 = \boldsymbol{e}_1$, the matrix $\boldsymbol{U}$ is unit upper triangular, and, in view of $\boldsymbol{S}^{\mathrm{T}}\boldsymbol{U} = \boldsymbol{D}$, the matrix $\boldsymbol{S}$ is lower triangular (or block tower triangular), as we know of course from its original definition. Therefore, once the first j (block) columns of $\boldsymbol{S}$ and $\boldsymbol{D}$ are known, the first j (block) columns of $\boldsymbol{U}$ could be found by the standard recurrence for inverting a triangular matrix.

In the more general situation where the support of the weight function is not known and thus possibly $h_n \neq t_n$, the relations (5a) and (5b) need no longer hold for $n \times n$ matrices. But, assuming that infinite sequences of polynomials t_j and modified moments s_{i0} are given, we can still derive the analogue of (5a)–(5f) for infinite matrices $\boldsymbol{H}$, $\boldsymbol{S}$, $\boldsymbol{T}$, and $\boldsymbol{U}$, cf. [8], with $\boldsymbol{T}$ being arbitrary unit upper Hessenberg. We could then formally run both the modified Chebyshev algorithm and the Lanczos algorithm, starting the latter with $\boldsymbol{A} \equiv \boldsymbol{T}$, $\boldsymbol{x}_0 \equiv \boldsymbol{e}_1$, $\boldsymbol{y}_0 \equiv \boldsymbol{s}_0$, and they would produce the same result. Further inspection of the dependencies shows that for producing the first n columns of $\boldsymbol{H}$ it normally suffices to know the first $2n$ modified moments and to compute a (decreasing) finite number of elements of the columns of $\boldsymbol{S}$ and $\boldsymbol{U}$. So, after all, we still only need finite sequences $\{t_j\}$ and $\{s_{i0}\}$ if the support of the weight function is a finite set and thus $\boldsymbol{A}$ has finite rank.

We conclude that the nonsymmetric Lanczos algorithm and the modified Chebyshev algorithm are indeed equivalent in the sense that, theoretically, all the problems that can be solved with one can as well be solved with the other, whereby both produce identical results. Since $\boldsymbol{T}$ can be chosen to be the shift matrix, it follows that also the classical (unmodified) Chebyshev algorithm is equivalent to the Lanczos algorithm. We have moreover seen from this equivalence that when nonsymmetric Lanczos is applied to an upper Hessenberg matrix $\boldsymbol{T}$ and is started with $\boldsymbol{x}_0 \equiv \boldsymbol{e}_1$, it suffices to compute the columns of $\boldsymbol{S}$. Likewise, if it were applied to a lower Hessenberg matrix and started with $\boldsymbol{y}_0 \equiv \boldsymbol{e}_1$, it would be sufficient to compute the columns of $\boldsymbol{U}$. Computational costs and storage could thus be reduced by half. However, in practice such a special choice of one initial vector may not be advisable.

5. Application to weighted checksum error correction scheme

We examine a particular application of the above reconstruction that occurs in the use of weighted checksum schemes for detection and correction of temporary or transient hardware errors that might occur during the course of a computation. The problem we address is to detect and correct up to d temporary hardware errors that might occur during the solution of an $n \times n$ system of linear equations of the form $\boldsymbol{A}\boldsymbol{x} = \boldsymbol{b}$. We consider algorithms that solve such systems of linear equations by factoring the matrix $\boldsymbol{A}$ into $\boldsymbol{A} = \boldsymbol{P}^{\mathrm{T}}\boldsymbol{L}\boldsymbol{R}$ (as in Gaussian elimination with partial pivoting), $\boldsymbol{A} = \boldsymbol{Q}\boldsymbol{R}$, or some similar form, where $\boldsymbol{P}$ is a permutation matrix, $\boldsymbol{L}$ is unit lower triangular, $\boldsymbol{R}$ is upper triangular, and $\boldsymbol{Q}$ is an orthogonal matrix [9]. To simplify the exposition, we let $\boldsymbol{Q}$ denote the factoring matrix: either the orthogonal matrix just noted or the product $\boldsymbol{P}^{\mathrm{T}}\boldsymbol{L}$ for Gaussian elimination. In either case, $\boldsymbol{Q}^{-1}$ represents the "row operations" applied to the matrix $\boldsymbol{A}$ to reduce it to the upper triangular form $\boldsymbol{R} = \boldsymbol{Q}^{-1}\boldsymbol{A}$.

To detect and correct up to d hardware errors, we need at least $m \equiv 2d + 1 \ll n$ checksums on each row of $\boldsymbol{A}$. Let $\boldsymbol{M} = [\boldsymbol{I} \mid \boldsymbol{V}_m]$ be the $n \times (n+m)$ matrix of checksums, where $\boldsymbol{V}_m$ is the $n \times m$ matrix of *checksum coefficients*, consisting of the first m columns of $\boldsymbol{V}_T$ from (4). Given any matrix $\boldsymbol{A}$, we can compute m independent checksums on each row, to obtain the $n \times m$ matrix of checksums $\boldsymbol{A}_{\mathrm{c}} \equiv \boldsymbol{A}\boldsymbol{V}_m$. We can append this to $\boldsymbol{A}$ to form the augmented matrix $\boldsymbol{A}_M \equiv \boldsymbol{A}\boldsymbol{M} = [\boldsymbol{A} \mid \boldsymbol{A}_{\mathrm{c}}]$. The operations represented by $\boldsymbol{Q}^{-1}$ are applied to $\boldsymbol{A}_M$, yielding the checksummed factorization $\boldsymbol{A}_M = \boldsymbol{Q}\boldsymbol{R}_M = \boldsymbol{Q}[\boldsymbol{R} \mid \boldsymbol{R}_{\mathrm{c}}]$, where $\boldsymbol{R}_{\mathrm{c}} = \boldsymbol{Q}^{-1}\boldsymbol{A}_{\mathrm{c}}$ is the result of applying the row operations represented by $\boldsymbol{Q}^{-1}$ to the checksums $\boldsymbol{A}_{\mathrm{c}}$. The checksums can then be used to detect errors by forming the $n \times m$ *checksum difference* matrix $\boldsymbol{F} = \boldsymbol{R}_M\boldsymbol{N}$, where

$$\boldsymbol{N} = \begin{bmatrix} \boldsymbol{V}_m \\ -\boldsymbol{I} \end{bmatrix} ((m+n) \times m)$$

is the *weighted checksum matrix*. The checksum difference matrix $\boldsymbol{F} = \boldsymbol{Q}^{-1}[\boldsymbol{A} \mid \boldsymbol{A}_c]\boldsymbol{N} = \boldsymbol{R}\boldsymbol{V}_m - \boldsymbol{Q}^{-1}\boldsymbol{A}_c$ is simply the difference between the checksums computed from the resulting $\boldsymbol{R}$ and the row operations $\boldsymbol{Q}^{-1}$ applied to the original checksums $\boldsymbol{A}_c$ from the starting matrix $\boldsymbol{A}$. Hence if no hardware errors occur, $\boldsymbol{F}$ will be entirely zero. Under the assumption that at most d errors have occurred in any row, it can be shown conversely that if $\boldsymbol{F}$ is entirely zero, then no errors have occurred during the computation [1]. Therefore an error is detected in a row of $\boldsymbol{R}_M$ exactly when the corresponding row of $\boldsymbol{F}$ is not entirely zero. Furthermore, the nonzero entries of $\boldsymbol{F}$ can be used to determine what that error was and where it occured, so that a correction can be applied.

Suppose $\boldsymbol{f}^{\mathrm{T}}$ is a nonzero row of $\boldsymbol{F}$. Then the corresponding row of the computed $\boldsymbol{R}_M$ has an error, which we now show how it may be computed. Let $\boldsymbol{r}_M^{\mathrm{T}} \equiv [\boldsymbol{r}^{\mathrm{T}} \mid \boldsymbol{r}_c^{\mathrm{T}}]$ be the corresponding row of the *correct* $\boldsymbol{R}_M$ to be determined. Let $\boldsymbol{e}_M^{\mathrm{T}} = [\boldsymbol{e}^{\mathrm{T}} \mid \boldsymbol{e}_c^{\mathrm{T}}]$ be the net error in that row of $\boldsymbol{R}_M$. That is, the row of $\boldsymbol{R}_M$ actually computed is $\boldsymbol{r}_M^{\mathrm{T}} + \boldsymbol{e}_M^{\mathrm{T}}$ and satisfies $(\boldsymbol{r}_M^{\mathrm{T}} + \boldsymbol{e}_M^{\mathrm{T}})\boldsymbol{N} = \boldsymbol{f}^{\mathrm{T}}$. Since $\boldsymbol{r}_M^{\mathrm{T}}$ is the correct row, we have $\boldsymbol{r}_M^{\mathrm{T}}\boldsymbol{N} = 0$ and $\boldsymbol{e}_M^{\mathrm{T}}\boldsymbol{N} = \boldsymbol{f}^{\mathrm{T}}$. If we assume (as in [1]) that the errors occur in the matrix $\boldsymbol{R}$ itself, not in the checksums $\boldsymbol{R}_c$, then $\boldsymbol{e}_c^{\mathrm{T}} = 0$, and we have the relation

$$\boldsymbol{e}^{\mathrm{T}}\boldsymbol{V}_m = \boldsymbol{f}^{\mathrm{T}}. \tag{8}$$

We obtain the m-vector $\boldsymbol{f}^{\mathrm{T}}$ from the checksum difference matrix, and our goal is to solve for the n-vector of errors $\boldsymbol{e}^{\mathrm{T}}$. In the case $m = n$, the matrix $\boldsymbol{V}_m$ is square and nonsingular, so $\boldsymbol{e}^{\mathrm{T}}$ can be obtained directly by solving the system (8). But more typically, we assume that at most $d \ll n$ entries of the vector $\boldsymbol{e}^{\mathrm{T}}$ are nonzero, meaning at most d individual errors occurred during the factorization. In this case, we compute only $m = 2d + 1 \ll n$ checksums per row. We know at most d entries of $\boldsymbol{e}$ are nonzero, but we do not know exactly how many are nonzero nor which ones they are.

We can apply the following interpretation to (8). Since $\boldsymbol{V}_m$ is generated by a sequence of polynomials $\boldsymbol{t}^{\mathrm{T}}$, we can interpret the vector $\boldsymbol{e}$ as a vector of unknown weights. Then the vector $\boldsymbol{f}$ is seen to be a vector of moments: $f_i = \langle t_0, t_i \rangle_e$. Thus we can interpret $\boldsymbol{f}$ as the vector of initial moments $\boldsymbol{s}_0$ of the previous section, and $\boldsymbol{e}$ as the vector of weights $\boldsymbol{w}$, and use the nonsymmetric Lanczos algorithm as outlined above to solve for $\boldsymbol{e}$ from the "moments" $\boldsymbol{f}$. We run the Lanczos algorithm with matrix $\boldsymbol{T}$ and starting vectors $\boldsymbol{u}_0 = [1, 0, \ldots, 0]^{\mathrm{T}}$ and $\boldsymbol{s}_0 = \boldsymbol{f}$, where $\boldsymbol{T}$ is the recurrence matrix for the polynomials defining the given checksum coefficients $\boldsymbol{V}_m$. We can also use the Chebyshev algorithm to accomplish the same task.

If $l \leqslant d$ is the number of errors that have occurred in the row of $\boldsymbol{R}$, this process will stop after l steps, generating $l + 1$ columns $\boldsymbol{U}_l$ on the right and $\boldsymbol{S}_l$ on the left, as outlined in the previous section just above eq. (7). The zero entries in the last column of $\boldsymbol{V}_m\boldsymbol{U}_l$ mark the positions of the nonzero entries in the error vector $\boldsymbol{e}^{\mathrm{T}}$. Then the values of these errors are obtained by solving the linear system (7). Note that, unlike the process in [1], it is not necessary to solve an eigenproblem to determine the positions of the errors.

In order to use this method, we need the recurrence matrix T and the first m columns of the generalized Vandermonde matrix V_m. If T is given, we may apply the recurrences $ZV_m = V_mT$ to obtain V_m, where Z is the diagonal matrix of eigenvalues of T. If we desire instead to use the polynomials orthogonal with respect to some given inner product, it is necessary to generate T. This can be done by the symmetric Lanczos algorithm [10,3,2], or somewhat less stably by the nonsymmetric Lanczos algorithm as outlined in the previous section, starting with the shift matrix (i.e. the recurrence matrix for the monomials z^i) and the ordinary moments with respect to the given inner product.

6. Numerical examples

We first illustrate the generalized Lanczos algorithm with an 8×8 example. The input A, x_0, y_0, and the generated vectors X, Y, coefficients H, G, and the products $D = Y^TX$, Y^TAX are all listed in appendix 2. The method was run twice, once as described, and once with the orthogonalization within each cluster turned off. This last choice was made to illustrate the Hankel structure in the D. The method returned four clusters of dimensions 2, 4, 1, 1, respectively, indicated by the partitioning lines.

One can note several properties we have indicated above, besides the Hankel structure in D when orthogonalization within clusters is turned off. Even when this orthogonalization is applied, the diagonal blocks D_k still maintain a Hankel-like, lower anti-triangular, structure. Also note that the initial vectors in each cluster remain unchanged in the second run. These initial vectors are x_i, y_i, $i = 0$, 2, 6, 7. One can also see that the H and G do not turn out identical in this algorithm, unlike the situation in the standard Lanczos method. But they do turn out identical when the orthogonalization within clusters is turned off. We note that one sees much more structure in the numerical examples than was indicated above.

In appendix 3 we give a short example illustrating the reconstruction of a set of indefinite weights, given a few initial moments. This corresponds exactly to the problem of reconstructing an error vector given a checksum difference vector computed using checksums during the course of some matrix factorization algorithm. In the example, we have used a set of checksum coefficients generated using a sequence of orthogonal polynomials, but it would be equally possible to do it with the monomials, in which case the checksum coefficients would be just the entries from the ordinary Vandermonde matrix. Using orthogonal polynomials tends to yield checksum coefficients that are better scaled, and if we relaxed the requirement that the polynomials be monic, we would improve the scaling even further.

7. Conclusions

We have outlined a nonsymmetric matrix Lanczos procedure that can continue when a generated pair of vectors are mutually orthogonal. This process has been defined in constructive fashion and entirely in terms of matrices. We then showed the connection between the matrix process and the moments and weights with respect to a discrete indefinite inner product. We showed how the matrix process with a particular pair of initial vectors corresponds exactly to the process of generating a matrix of recurrence coefficients for polynomials orthogonal with respect to an unknown, indefinite, weight function, starting with a set of initial moments, as done by the Chebyshev algorithm. We finally indicated how the problem of error correction in an algorithm-based checksum scheme embedded in a matrix factorization process can be cast as a problem of computing indefinite weights from the moments for a set of orthogonal polynomials. In the appendices 2 and 3 we illustrate the Lanczos process with non-diagonal blocks and the process of computing the unknown weights for an inner product with some numerical examples.

There is much more structure in the results of the Lanczos algorithm than we have indicated here. We have not addressed the issue of how the nonsymmetric Lanczos algorithm might be used to solve the matrix eigenproblem. We have indicated that there are several places within the algorithm where we have choice of scaling, and we have chosen certain scalings to enhance the clarity of the presentation of the algorithm, and to keep the corresponding polynomials monic. We have not discussed how the choice of scalings might be used to improve the numerical stability, nor have we addressed the need for re-orthogonalization of the Lanczos vectors to maintain biorthogonality. All these aspects are left for future work.

References

[1] C.F. Anfinson, R.P. Brent and F.T. Luk, A theoretical foundation for the weighted checksum scheme in: *Proc. SPIE vol 975 Advanced Algorithms and Architectures for Signal Processing III*, paper 2, 1988.

[2] D.L. Boley and G.H. Golub, A survey of matrix inverse eigenvalue problems, Inverse Problems, Vol. 3 (Physics Trust Publications, Bristol, England, 1987) pp 595–622.

[3] C. de Boor and G. Golub, The numerically stable reconstruction of a Jacobi matrix from spectral data, Lin. Alg and Appl. 21 (1978) 245–260.

[4] C. Brezinski, *Padé-Type Approximants and General Orthogonal Polynomials*, ISNM, Vol. 50 (Birkhäuser, Basel/Stuttgart, 1980).

[5] J. Cullum and R. Willoughby, *Lanczos Algorithms for Large Symmetric Eigenvalue Computations*, Vol. I: Theory (Birkhäuser, Boston, 1985).

[6] G. Cybenko, An explicit formula for Lanczos polynomials, Lin. Alg. and Appl. 88/89 (1987) 99–115.

[7] W. Gautschi, On generating orthogonal polynomials, SIAM J. Sci. and Stat. Comput. 3 (1982) 289–317.

[8] G. Golub and M. Gutknecht, Modified moments for indefinite weight functions, Numer. Math. 57 (1990) 607–624.

[9] G. Golub and C. Van Loan, *Matrix Computations* 2/e (Johns Hopkins, 1989).

[10] G. Golub and J. Welsch, Calculation of Gauss quadrature rules, Math. Comp. 23 (1969) 221–230.

[11] W.B. Gragg, The Padé table and its relation to certain algorithms of numerical analysis, SIAM Rev. 14 (1972) 1–62.

[12] W.B. Gragg, Matrix interpretations and applications of the continued fraction algorithm, Rocky Mountain J. Math. 4 (1974) 213–225.

[13] M.H. Gutknecht, A completed theory for the unsymmetric Lanczos process and related algorithms, SIAM J. Matrix Anal., 1989, submitted.

[14] M.H. Gutknecht, The unsymmetric Lanczos algorithms and their relations to Padé approximation, continued fractions, the qd algorithm, biconjugate gradient squared algorithms, and fast Hankel solvers, preprint, 1990.

[15] K.H. Juang and J.A. Abraham, Algorithm-based fault tolerance for matrix operations, IEEE Trans. Comput. C-33 Nr. 6 (June 1984) 518–528.

[16] J.Y. Jou and J.A. Abraham, Fault-tolerant matrix arithmetic and signal processing on highly concurrent computing structures; Proc. IEEE 74 Nr. 5, special issue on fault tolerance (May 1986) 732–741.

[17] W. Joubert, Lanczos methods for the solution of nonsymmetric systems of linear equations in: *Proc. Copper Mtn. Conf. on Iterative Methods*, April 1–5, 1990; submitted to SIAM J. on Sci. and Stat. Comput.

[18] J. Kautsky and G.H. Golub, On calculation of Jacobi matrices, Lin. Alg. and Appl. 52/53 (1983) 439–455.

[19] S. Kaniel, Estimates for some computational techniques in linear algebra, Math. Comp. 20 (1966) 369–378.

[20] M. Kent, Chebyshev, Krylov, Lanczos: Matrix Relationships and Computations, Ph.D. Thesis, Stanford Univ. Computer Sci. Report STAN-CS-89-1271, June 1989.

[21] C. Lanczos, An iteration method for the solution of the eigenvalue problem linear differential and integral operators, J. Res. Natl. Bur. Stand. 45 (1950) 255–282.

[22] F.T. Luk and H. Park, An analysis of algorithm-based fault tolerance, J. Parallel Distr. Comput. 5 (1988) 172–184.

[23] C.C. Paige, The computation of eigenvalues and eigenvectors of very large sparse matrices, Ph.D. Thesis, London Univ., 1971.

[24] B. Parlett, *The Symmetric Eigenvalue Problem* (Prentice Hall, 1980).

[25] B.N. Parlett, Reduction to tridiagonal form and minimal realizations; preprint submitted to SIAM J. Matrix Anal., 1990.

[26] B.N. Parlett, D.R. Taylor and Z.A. Liu, A look-ahead Lanczos algorithm for unsymmetric matrices, Math. Comp. 44 (1985) 105–124.

[27] H. Rutishauser, Der Quotienten-Differenzen-Algorithmus, Mitt. Inst. angew. Math. ETH, Nr. 7 (Birkhäuser, Basel/Stuttgart, 1957).

[28] Y. Saad, On the rates of convergence of the Lanczos and the block Lanczos methods, SIAM J. Num. Anal. 17 (1980) 687–706.

[29] R. Sack and A. Donovan, An algorithm for Gaussian quadrature given modified moments, Numer. Math. 18 (1972) 465–478.

[30] D. Scott, Analysis of the symmetric Lanczos process, Univ. of Calif., Berkeley, Electronic Res. Lab. Report UCB/ERL M78/40, 1978.

[31] J. Wheeler, Modified moments and Gaussian quadrature, Rocky Mtn. J. Math. 4 (1974) 287–296.

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

Appendix 1

MODIFIED NONSYMMETRIC LANCZOS ALGORITHM

Below is a simple summary of the modified nonsymmetric Lanczos algorithm. The computational details (especially in steps 2.2, 2.3) are omitted to ease the clutter in this summary. For example, it was shown in the text just below eq. (3c) that $\boldsymbol{H}$ and $\boldsymbol{G}$ are block tridiagonal, hence many of the entries in $\hat{\boldsymbol{h}}$, $\hat{\boldsymbol{g}}$ are zero, at least in exact arithmetic. But computational experience with Lanczos methods in floating point arithmetic has shown the need to re-bi-orthogonalize in order to maintain the properties of bi-orthogonality among the vectors $\boldsymbol{X}$, $\boldsymbol{Y}$, and block tridiagonality of $\boldsymbol{H}$, $\boldsymbol{G}$ (cf. [32, p391], [9], etc.).

```
**Input**: $n \times n$ matrix $\boldsymbol{A}$ and two $n$-vectors $\boldsymbol{x}_0$, $\boldsymbol{y}_0$.
1.  ( * initialization * )
    **Set** all clusters $X_l$, $Y_l$ (for all $l$) to "empty".
    **Set** first clusters $X_0 := [\boldsymbol{x}_0]$, $Y_0 := [\boldsymbol{y}_0]$.
    **If** $\boldsymbol{y}_0^{\mathrm{T}} \boldsymbol{x}_0 = 0$ **then set** $k := 0$ **else set** $k := 1$; ( * current cluster index * )
    **Set** $i := 0$, ( * vector index * )
2.  **Until** $\boldsymbol{x}_i = 0$ or $\boldsymbol{y}_i = 0$ **do begin**
      ( * main loop * )
      2.1. ( * apply matrix operator to expand Krylov sequence * )
           **Set** $\hat{\boldsymbol{b}} := \boldsymbol{A}\boldsymbol{x}_i$ and $\hat{\boldsymbol{c}} := \boldsymbol{A}^{\mathrm{T}}\boldsymbol{y}_i$.
      2.2. ( * bi-orthogonalize against previous clusters * )
           **Find** coefficient vectors $\hat{\boldsymbol{h}}_i$, $\hat{\boldsymbol{g}}_i$ such that
             $\boldsymbol{b} := \hat{\boldsymbol{b}} - [X_0, \ldots, X_{k-1}]\hat{\boldsymbol{h}}_i$ is orthogonal to $[Y_0, \ldots, Y_{k-1}]$ and
             $\boldsymbol{c} := \hat{\boldsymbol{c}} - [Y_0, \ldots, Y_{k-1}]\hat{\boldsymbol{g}}_i$ is orthogonal to $[X_0, \ldots, X_{k-1}]$.
      2.3. ( * if current cluster nonempty, orthogonalize within the cluster * )
           **If** cluster pair $X_k$, $Y_k$ nonempty **then**
             **find** coefficient vectors $\boldsymbol{h}_i$, $\boldsymbol{g}_i$ so that
                 $\boldsymbol{x}_{i+1} := \hat{\boldsymbol{b}} - X_k \boldsymbol{h}_i$ is orthogonal to $X_k$ and
                 $\boldsymbol{y}_{i+1} := \hat{\boldsymbol{c}} - Y_k \boldsymbol{g}_i$ is orthogonal to $Y_k$.
      2.4. ( * append pair of new vectors to current cluster. * )
           **Set** $X_k := [X_k, \boldsymbol{x}_{i+1}]$.
           **Set** $Y_k := [Y_k, \boldsymbol{y}_{i+1}]$.
      2.5. ( * if current cluster complete, make next cluster "current",... * )
           ( * ...so next vector will be the first vector in that next cluster * )
           **If** $Y_k^{\mathrm{T}} X_k$ is non-singular **then**
             **Set** $k := k + 1$.
      2.6. **Set** $i := i + 1$.
**End** Until Loop.
**Results**: vectors $\boldsymbol{X} = [X_0, \ldots, X_k]$, $\boldsymbol{Y} = [Y_0, \ldots, Y_k]$, and coefficients $\boldsymbol{H}$, $\boldsymbol{G}$, satis-
         fying (3a)–(3c).
```

Appendix 2

LANCZOS ALGORITHM EXAMPLES

Input matrix A

0	0	0	0	0	0	0	−1
1	0	0	0	0	0	0	0
0	1	0	0	0	0	0	0
0	0	1	0	0	0	0	0
0	0	0	1	0	0	0	0
0	0	0	0	1	0	0	0
0	0	0	0	0	1	0	0
0	0	0	0	0	0	1	0

Initial vectors

x_0	y_0
1	0
1	0
0	1
0	0
0	0
0	−1
0	0
0	0

Generated right vectors X

1.0	−0.5	1.0	0	0	−0.5	1.0	1.0
1.0	0.5	0	1.0	0	0	−1.0	1.0
0	1.0	0	0	1.0	0	−1.0	−1.0
0	0	1.0	0	0	0.5	1.0	1.0
0	0	0	1.0	0	0	−1.0	1.0
0	0	0	0	1.0	0	−1.0	−1.0
0	0	0	0	0	1.0	1.0	−1.0
0	0	0	0	0	0	1.0	1.0

Generated left vectors Y

0	0	1.000	−0.167	0.182	−0.300	1.000	1.000
0	1.000	−1.000	0.167	−0.182	0.300	−1.000	−1.000
1.000	0	1.000	−0.167	0.182	−0.300	1.000	1.000
0	0	−1.000	0.167	−0.182	−0.700	−1.000	−1.000
0	−1.000	1.000	−0.167	−0.818	−0.150	1.000	1.000
−1.000	0	−1.000	−0.833	−0.091	−0.350	−1.000	−1.000
0	0	0	0	−1.000	0.150	1.000	1.000
0	0	0	−1.000	0.091	0.350	1.000	−1.000

Right-Hand coefficients H

0.5	−0.75	0	0	0	−1.0	0	0
1.0	0.50	0	0	0	0	0	0
0	1.00	0	0	0.5	−0.5	−2.0	0
0	0	1.0	0	0	1.5	0	0
0	0	0	1.0	0	1.0	0	0
0	0	0	0	1.0	−1.0	0	0
0	0	0	0	0	1.0	0	−1.0
0	0	0	0	0	0	1.0	0

Left-Hand coefficients G

0	−1.000	0	0	0	−1.000	0	0
1.000	1.000	0	0	0	0	0	0
0	1.000	−0.833	−0.028	0.197	−0.742	−2.000	0
0	0	1.000	−0.076	0.537	0.705	0	0
0	0	0	1.000	0.059	0.628	0	0
0	0	0	0	1.000	−0.150	0	0
0	0	0	0	0	1.000	0	−1.000
0	0	0	0	0	0	1.000	0

Block diagonal matrix $D = Y^{\mathrm{T}}X$

0	1.000	0	0	0	0	0	0
1.000	0.500	0	0	0	0	0	0
0	0	0	0	0	−1.000	0	0
0	0	0	0	−1.000	0.167	0	0
0	0	0	−1.000	0.091	−1.182	0	0
0	0	−1.000	0.150	−0.650	−0.050	0	0
0	0	0	0	0	0	2.000	0
0	0	0	0	0	0	0	−2.000

Block Tridiagonal matrix $Y^{\mathrm{T}}AX$

1.000	0.500	0	0	0	0	0	0
1.000	−0.500	0	0	0	−1.000	0	0
0	0	0	0	−1.000	1.000	0	0
0	0	0	−1.000	0.167	−1.167	0	0
0	0	−1.000	0.091	−1.182	−0.227	0	0
0	−1.000	0.150	−0.650	−0.550	0.125	2.000	0
0	0	0	0	0	2.000	0	−2.000
0	0	0	0	0	0	−2.000	0

Below are the results when the vectors within each cluster are not orthogonalized.

X with non-orthogonalized clusters

1	0	1	0	0	0	1	1
1	1	0	1	0	0	−1	1
0	1	0	0	1	0	−1	−1
0	0	1	0	0	1	1	1
0	0	0	1	0	0	−1	1
0	0	0	0	1	0	−1	−1
0	0	0	0	0	1	1	−1
0	0	0	0	0	0	1	1

Y with non-orthogonalized clusters

$$\begin{array}{cc|cccc|c|c}
0 & 0 & 1 & -1 & 1 & -1 & 1 & 1 \\
0 & 1 & -1 & 1 & -1 & 1 & -1 & -1 \\
1 & 0 & 1 & -1 & 1 & -1 & 1 & 1 \\
0 & 0 & -1 & 1 & -1 & 0 & -1 & -1 \\
0 & -1 & 1 & -1 & 0 & 0 & 1 & 1 \\
-1 & 0 & -1 & 0 & 0 & -1 & -1 & -1 \\
0 & 0 & 0 & 0 & -1 & 1 & 1 & 1 \\
0 & 0 & 0 & -1 & 1 & -1 & 1 & -1
\end{array}$$

H with non-orthogonalized clusters

$$\begin{array}{cc|cccc|c|c}
0 & -1 & 0 & 0 & 0 & -1 & 0 & 0 \\
1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\
\hline
0 & 1 & 0 & 0 & 0 & 0 & -2 & 0 \\
0 & 0 & 1 & 0 & 0 & 2 & 0 & 0 \\
0 & 0 & 0 & 1 & 0 & 1 & 0 & 0 \\
0 & 0 & 0 & 0 & 1 & -1 & 0 & 0 \\
\hline
0 & 0 & 0 & 0 & 0 & 1 & 0 & -1 \\
\hline
0 & 0 & 0 & 0 & 0 & 0 & 1 & 0
\end{array}$$

G with non-orthogonalized clusters

$$\begin{array}{cc|cccc|c|c}
0 & -1 & 0 & 0 & 0 & -1 & 0 & 0 \\
1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\
\hline
0 & 1 & 0 & 0 & 0 & 0 & -2 & 0 \\
0 & 0 & 1 & 0 & 0 & 2 & 0 & 0 \\
0 & 0 & 0 & 1 & 0 & 1 & 0 & 0 \\
0 & 0 & 0 & 0 & 1 & -1 & 0 & 0 \\
\hline
0 & 0 & 0 & 0 & 0 & 1 & 0 & -1 \\
\hline
0 & 0 & 0 & 0 & 0 & 0 & 1 & 0
\end{array}$$

$D = Y^{\mathrm{T}}X$ with non-orthogonalized clusters

$$\begin{array}{cc|cccc|c|c}
0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\
1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\
\hline
0 & 0 & 0 & 0 & 0 & -1 & 0 & 0 \\
0 & 0 & 0 & 0 & -1 & 1 & 0 & 0 \\
0 & 0 & 0 & -1 & 1 & -2 & 0 & 0 \\
0 & 0 & -1 & 1 & -2 & 1 & 0 & 0 \\
\hline
0 & 0 & 0 & 0 & 0 & 0 & 2 & 0 \\
\hline
0 & 0 & 0 & 0 & 0 & 0 & 0 & -2
\end{array}$$

Y^TAX with non-orthogonalized clusters

1	1	0	0	0	0	0	0
1	0	0	0	0	−1	0	0
0	0	0	0	−1	1	0	0
0	0	0	−1	1	−2	0	0
0	0	−1	1	−2	1	0	0
0	−1	1	−2	1	−1	2	0
0	0	0	0	0	2	0	−2
0	0	0	0	0	0	−2	0

Appendix 3

POLYNOMIAL AND WEIGHT RECONSTRUCTION EXAMPLE

We use the vector of points $z = [1, 2, \ldots, 8]^T$. Assume only $d = 2$ weights are nonzero, so we need only $m = 2d + 1 = 5$ initial moments $s_{00}, \ldots, s_{40}$.

In the context of checksumming the LU factorization [9] of a matrix $A = LU$, this corresponds to computing $m = 5$ checksums on each row of the 8×8 matrix A, allowing up to $d = 2$ errors per row. If V_5 is the 8×5 matrix of checksum coefficients, then the checksums computed on A will be AV_5. The checksum difference vector f, a 5-vector, is one row of the checksum difference matrix F, which is the difference between the checksums on the resulting U and the row operations L applied to the original checksums AV_5.

One can use the symmetric Lanczos algorithm [10,3,2] to find the tridiagonal matrix T that generates the monic polynomials orthogonal over points z with weights $1, \ldots, 1$:

4.500	5.250	0	0	0	0	0	0
1.000	4.500	4.000	0	0	0	0	0
0	1.000	4.500	3.536	0	0	0	0
0	0	1.000	4.500	3.048	0	0	0
0	0	0	1.000	4.500	2.462	0	0
0	0	0	0	1.000	4.500	1.762	0
0	0	0	0	0	1.000	4.500	0.942
0	0	0	0	0	0	1.000	4.500

The corresponding polynomials are (from $zt^T(z) = t^T(z)T + t_n(z)e_n^T$)

$t_0 = 1$

$t_1 = zt_0 - 4.5t_0 \qquad = z - 4.5$

$t_2 = zt_1 - 4.5t_1 - 5.25t_0 = z^2 - 9z + 15$

...

The checksum coefficients V_5 can be found by evaluating the first five such polynomials on the points z. One could also use the non-symmetric Lanczos algorithm itself to generate $\boldsymbol{T}$, V_T directly, starting with the moments of the monomials 1, z, z^2, $z^3, \ldots$ over the points z with weights $1, \ldots, 1$.

The vector of initial moments we use is $s_0 = [-4, -25, -34, 90, 277.714, *, *, *]^{\mathrm{T}}$, where the $*$'s stand for unspecified *fill* values. In our checksum example, we use $\boldsymbol{f}^{\mathrm{T}} = [-4, -25, -34, 90, 277.714]$ as our example.

The Lanczos process terminates after 2 steps, generating the vectors

	U_3			S_3	
1.000	−6.250	4.000	−4.000	0	0
0	1.000	−2.000	−25.000	101.250	0
0	0	1.000	−34.000	202.500	0
0	0	0	90.000	−405.000	*
0	0	0	277.714	*	*
0	0	0	*	*	*
0	0	0	*	*	*
0	0	0	*	*	*

The $*$'s denote entries depending on the fill values in s_0. These need not be computed.

The first three polynomials orthogonal with respect to the unknown inner product are derived from U_3:

$$h_0 = t_0 \qquad = 1$$

$$h_1 = t_1 - 6.25 t_0 \qquad = z - 10.75$$

$$h_2 = t_2 - 2t_1 - 4t_0 = z^2 - 11z + 28.$$

Note the zeroes of h_2 are $z_3 = 4$ and $z_6 = 7$, as is shown in the Generalized Vandermonde matrix $V_T U_3$:

j	z_j	$V_T U_3$		
		$[h_0(z_j)$	$h_1(z_j)$	$h_2(z_j)]$
0	1.00	1.00	−9.75	18.00
1	2.00	1.00	−8.75	10.00
2	3.00	1.00	−7.75	4.00
3	4.00	1.00	−6.75	0
4	5.00	1.00	−5.75	−2.00
5	6.00	1.00	−4.75	−2.00
6	7.00	1.00	−3.75	0
7	8.00	1.00	−2.75	4.00

Corresponding to the zeroes in the last column of $V_T U_3$, the nonzero weights are w_3, w_6. We solve the system (7):

$$\begin{bmatrix} -4.0 \\ -25.0 \end{bmatrix} = \begin{bmatrix} 1.0 & 1.0 \\ -0.5 & 2.5 \end{bmatrix} \begin{bmatrix} w_3 \\ w_6 \end{bmatrix}$$

to obtain the solution $w_3 = 5$, $w_6 = -9$. So the final set of weights, corresponding to the error vector in the checksum problem is $\boldsymbol{w} = \boldsymbol{e} = [0, 0, 0, 5, 0, 0, -9, 0]^{\mathrm{T}}$.

We could also do the same computation using the monomials $t_i(z) = z^i$, $i = 0, 1, \ldots, 7$. In this case $\boldsymbol{T}$ would be the companion matrix for the polynomial $t_8 = (z-1)\ldots(z-8)$. The steps are exactly the same as above.