Convolutional Reed-Solomon Codes

By P. M. EBERT and S. Y. TONG

(Manuscript received July 28, 1968)

We derive a new family of convolutional character-error-correcting codes which are a convolutional form of the Reed-Solomon block codes and as such have nonbinary symbols. We also derive a bound on the error correcting capabilities of these codes in which the error-correcting capability per constraint length grows approximately with the square root of the constraint length.

When these codes are used on a binary channel they are effective for both random and burst error correction because a single character spans several channel digits.

These codes have greater error-correcting capabilities than the Robinson-Bernstein self-orthogonal codes but are harder to decode. The single-character-error-correcting codes, when interleaved, are shown to be more powerful than the equivalent Hagelbarger code and appear to be simpler to implement. They are also slightly better than the interleaved version of Berlekamp's code.

We discuss encoding and decoding algorithms and illustrate a simple decoding algorithm for some of the codes. These codes are closely related to the Bose–Chaudhuri–Hocquenghem block codes and share with them the decoding simplification for character erasures in place of errors. Any Bose–Chaudhuri–Hocquenghem decoding algorithm can be used to decode these codes.

I. INTRODUCTION

This paper is concerned with a family of character error correcting convolutional codes which are derived from Reed-Solomon block codes.[1] The derivation of the error correcting capabilities is easy because of the algebraic nature of the code; the convolutional nature of the code allows the use of a simple encoder even at high rates. Also, sequential decoding techniques might be applicable.

Throughout the paper we use elements of a finite field as symbols

instead of just 1 and 0. The elements can be represented by k-tuples of ones and zeros if the field has 2^k elements; these k tuples are called symbols or characters. Thus with this code we are able to correct character errors which may be bursts of binary errors. All that one need know about finite fields is that each nonzero element has an inverse and that there exists at least one element which when taken to successive powers will generate the entire field with the exception of the zero element. This is called a primitive element.

The capability of the codes is given by the number of errors that can be corrected within a fixed number of characters (the constraint length). Suppose a code can correct three errors within a constraint length of 12. Then the code can correct any pattern of errors as long as no sequence of 12 characters has more than three errors in it. In the context of error correcting ability one can define a minimum distance of the code. For linear codes the minimum distance (d), is equal to the minimum weight code word segment, one constraint length long, which has a nonzero first character. With this definition the code can correct up to $(d\text{-}1)/2$ errors occurring within one constraint length.

The rate of a code, R, is the fraction of characters used as information symbols.

A convolutional code has its check symbols formed as a convolution or weighted sum of a fixed number of the past information symbols. The weighted sum is formed in the algebra of the finite field.

The codes described in Section II are capable of correcting t character errors within a constraint length of $(2t^2 - t + 1)/(1 - R)$ channel characters. The channel characters must be elements of a finite field of size at least $[R(2t - 1)\ (t - 1) + 1]/(1 - R)$. If one uses as channel symbols elements of a much larger field it is possible to construct a code which will correct t errors with a constraint length of only $2t/(1 - R)$. Those codes can be encoded by a standard convolutional encoder or by a number of accumulators which can also perform multiplication. Decoding can be accomplished by a modified Bose-Chaudhuri-Hocquenghem decoder.

II. CONSTRUCTION OF THE CODES

For a code of rate $R = (b - 1)/b$ one can use every bth symbol as a check symbol. To define the code completely one need only give the weights used in the convolution of past channel symbols. For convenience we put these weights in an N by $b = 1/(1-R)$ matrix, **B**. $B_{i,j}=$

W_{i-1+bi} (The constraint length is Nb.) If x_i is a check symbol we write

$$x_i = -\sum_{k=1}^{bN-1} x_{i-k}W_k, \qquad i = b, 2b, \cdots$$

or

$$\sum_{k=0}^{bN-1} x_{i-k}W_k = 0, \qquad W_0 = 1. \tag{1}$$

Following Wyner and Ash we take the N by b matrix called **B** and form it into an infinite **A** matrix by shifting **B** to the right b places and down one place:[2]

$$\mathbf{A} = \begin{bmatrix} B & 00 & 00 & \cdot \\ 00 & B & 00 & \cdot \\ 00 & 00 & B & \cdot \\ & & \cdots & \end{bmatrix}$$

Then by (1) any code sequence written as a vector **x** will satisfy

$$\mathbf{Ax} = \mathbf{0}. \tag{2}$$

We define the code by defining the elements of the matrix **B**. **B** is used instead of the weights W_i because the notation is clearer. Denote the elements of **B** by B_{ij} . Then let $B_{ij} = \beta_j\gamma_j^{i-1}$ where the b γ_i's are α^0, α^1, $\cdots$, α^{b-2}, 0. The symbol α is a primitive element of the field, and 0^0 is taken as 1. The γ_i's are called locators. Let: $\beta_j = \alpha^{v(j)\ (b-1)}$, where

$$v^{(j)} = \sum_{i=0}^{j-2} i = \frac{(j-1)(j-2)}{2}.$$

Thus for any b, N, and finite field a code is defined.

As an example we take the special case which was presented by Wyner, where $N = 2$ and the B_{ij} are given by[3]

$$B_{ij} = \begin{cases} \alpha^{(i-1)(j-1)} & i \neq b \\ 0^{(j-1)} & i = b \end{cases}$$

$$\mathbf{B} = \begin{bmatrix} 1 & 1 & \cdot & \cdots & 1 & 1 \\ 1 & \alpha & \alpha^2 & \cdots & \alpha^{b-2} & 0 \end{bmatrix}. \tag{3}$$

This code has $d = 3$ and thus can correct single errors, or double erasures in a constraint length of $2b$. It has the additional advantages

that it can be easily implemented, it is optimal,* and it has no error propagation. We describe, in Sections III through V, the implementation and some properties of the single-character-correcting codes.

III. DECODING TECHNIQUE FOR SINGLE CHARACTER CORRECTING CODES

Single errors are particularly easy to correct because the first syndrome (difference between the calculated check character and the received check character) is equal to the error, and the second syndrome is equal to the error multiplied by the error location. Since we take

$$\begin{aligned} \gamma_i &= \alpha^{i-1}, \qquad i \neq b \\ \gamma_i &= 0, \qquad\quad i = b \end{aligned} \tag{4}$$

the error location i can be found by dividing the second syndrome S_1 by α and comparing the result to the first syndrome, S_0. This is repeated until they agree. The division can be implemented by a shift register whose feedback path corresponds to the coefficients of $g(x)$ the generator polynominal of α, the primitive element.[4]

IV. HARDWARE IMPLEMENTATION FOR SINGLE CHARACTER CORRECTING CODE

The implementation is described through the example: symbols in $GF(2^k)$ $k = 2$, $b = 4$. The elements of $GF(4)$ are represented as binary 2-tuples. Since $x^2 + x + 1$ is a primitive polynomial in $GF(2)$, division by α can be instrumented by a two-state shift register with appropriate taps. The decoder takes form of Fig. 1. The entire received vector is shifted into the data buffer and then the syndromes S_0 and S_1 are computed and stored in two registers R_0 and R_1, respectively. S_0 identifies the error pattern. If S_0 is nonzero, it is assumed that a single character error of the pattern shown occurred. For each character shifted out of data buffer the R_1 register is shifted once with the feedback path connected (which corresponds to a division by α). The error pattern S_0 in R_0 matches that contained in register R_1 when the erroneous character just emerges out of the data buffer. This character need only have S_0 subtracted from it to complete correction. It is possible to do all of this with simple logic circuitry and a storage of $2bk$ bits.

---

*No other code, of the same constraint length, which corrects single errors, can have a higher rate (that is, fewer check symbols).

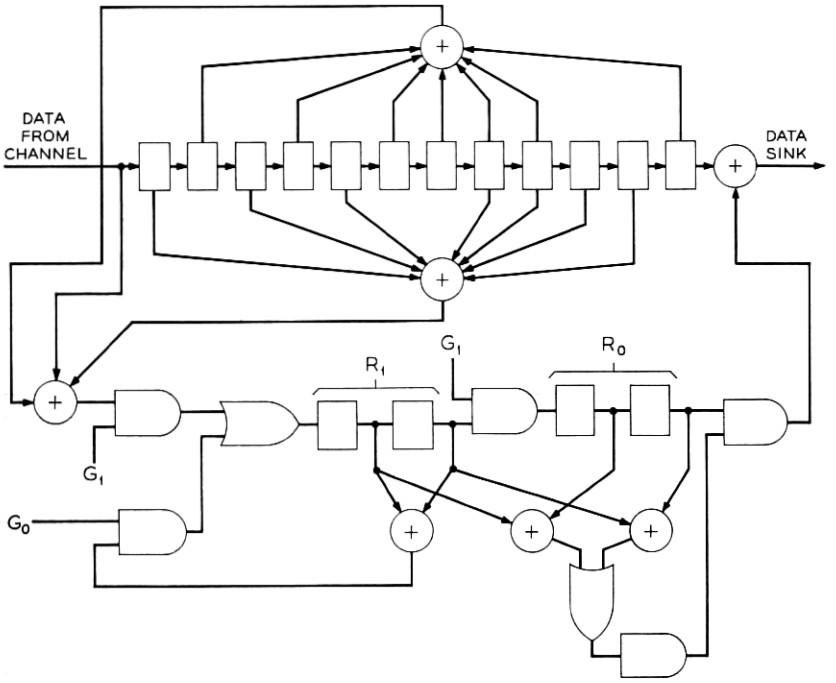

Fig. 1 — Decoder for single error correcting code. G_0 and G_1 are timing signals: G_0 is high during the time information characters appear; G_1 is high when parity character appears.

V. PROPERTIES OF THE SINGLE CHARACTER CORRECTING CODE

Observe that for every b character one must decide, based on the current syndromes of $2k$ bits, the error pattern and the location of the error. Since there are b possible error locations one needs at least $[\log_2 b] = k$ bits for identification if $2^{k-1} < b$, and as there are 2^k error patterns in a character (including the no-error pattern) that calls for k bits of information; thus the lower bound on the number of syndrome bits is $2k$ which shows the code is optimal* when $b > 2^{k-1}$.

Notice that since the syndrome is reset to zero after correction (corresponding to the removal of the effect of the error on syndrome) no error propagation is possible.

Interleaving can be applied to this class of codes to achieve a class of near-optimal burst-error correcting codes. If the interleaving degree is

* It is optimal in the number of check symbols.

m then two adjacent characters in the original code are separated by $m - 1$ characters or $L = k(m - 1)$ binary digits. A burst of length $L + 1$ bits will never affect two adjacent characters of the original codes; hence the interleaved code is capable of correcting an $L + 1$ bit burst. Since the original code has a storage requirement of $2bk$ bits, the interleaved code requires, at most, $2kbm$ bits. Observe that to correct a particular character one need not store the last character of the other $m - 1$ interleaved codes; therefore the storage requirement is $2\ bkm - (m - 1)k = 2\ bk(1 + L/k) - L$.

The required guard space is simply one less than the constraint length.* This can be seen by observing that in correcting a burst the syndrome must be set to zero when the last character in the burst is corrected. It follows that the guard space must always be shorter than the constraint length. Table I compares some members of this class of codes with some Hagelbarger burst-error-correcting recurrent codes as well as the Berlekamp burst-error-correcting recurrent codes (assuming Massey's decoding algorithm).[5-7] Table I shows this class of codes requires less guard space and hence is more powerful.

TABLE I—COMPARISON OF BURST CORRECTING CODES

Rate	Burst	Decoder cost*			Guard space		
		A	B	C	A	B	C
$R = 1/2$	20	44	—	61	61	—	60
	19	—	60	58	—	61	57
$R = 2/3$	21	132	—	112	170	—	111
	22	—	120	—	—	122	—
$R = 3/4$	20	183	—	—	223	—	—
	21	—	168	156	—	171	155
$R = 4/5$	20	326	—	—	384	—	—
	21	—	225	—	—	229	—
	22	—	—	219	—	—	218

* More precisely, the number of shift registers.

Note: A – Hagelbarger codes.
B – Berlekamp's type B2 burst-error correcting codes modified by interleaving.
C – Single-character correcting codes modified by interleaving.

Although the number of shift register stages is not an accurate measure of decoder cost, it is seen that, except for the rate of ½, Hagelbarger's codes generally cost more, especially at higer rates. Although

* In our case the constraint length is equal to the storage requirement.

the interleaved single character correcting codes are slightly better than Berlekamp's code for correcting type B1 bursts, they are both the same (and optimal) for type B2 burst correction.[2]

VI. GENERALIZATION

In order to demonstrate the minimum distance of codes with a longer constraint length we rely on the following lemma.

Lemma: *If the code with* $N = N_1 \geqq 1$ *has a minimum distance of at least* d_1 *, then the code with* $N_2 = N_1 + d_1 - 1$ *has a minimum distance of at least* $d_1 + 1$.

The proof of this lemma depends on showing that no code word with d_1 or less nonzero elements can satisfy $\mathbf{Ax} = \mathbf{0}$, which is equivalent to showing that any d_1 columns of $\mathbf{A}$ form a matrix of rank at least d_1 and thus equation (2) can be satisfied only by $\mathbf{x} = \mathbf{0}$ among all $\mathbf{x}$ with weight d_1 or less.

Proof: By the assumption that the code with $N = N_1$ has minimum distance d_1 , there must be at least d_1 nonzero elements in the first $N_1 b$ symbols of $\mathbf{x}$. Assume that there are just d_1 and no more, as well as no additional nonzero symbols in the rest of the constraint length N_2 . We write $\mathbf{x}'$ as a d_1 dimensional vector consisting of only the nonzero elements of $\mathbf{x}$. Accordingly $\mathbf{A}'\mathbf{x}' = \mathbf{0}$, where $\mathbf{A}'$ is a N_2 by d_1 matrix with columns corresponding to the elements of $\mathbf{x}'$. We must choose d_1 rows of $\mathbf{A}'$ which have a nonzero determinant. Assume for the moment that none of error locations are zero. We then choose the bottom d_1 of $\mathbf{A}'$. We have a d_1 by d_1 array of nonzero elements. Each of the matrix element is guaranteed to be nonzero by the fact that all the nonzero elements of $\mathbf{x}$ lie within the first $N_1 b$ elements and consequently $\mathbf{A}'$ can only have no zeros in the d_1th or lower rows.

We now take the d_1 by d_1 array and divide each column by the first element. The first row is now all 1's and the kth row is

$$\frac{B_i(j+k-1)}{B_{ij}} = \frac{\beta_{j+k-1}\gamma_i^{j+k-2}}{\beta_j\gamma_i^{j-1}} = (\alpha^{j(b-1)}\gamma_i)^{k-1}\alpha^{[(k-1)(k-4)(b-1)/2]}. \tag{5}$$

We next divide each row by $\alpha^{[(k-1)(k-4)(b-1)/2]}$ thus obtaining a Vander-Monde matrix. The determinant of the matrix cannot be zero since the quantity

$$\alpha^{j(b-1)}\gamma_i \tag{6}$$

is unique for each column. If q of the locators are zero, we choose the bottom $d_1 - q$ rows and those q rows which correspond to the 1's in the q zero locator columns. This determinant is nonzero by the same logic. Consequently, the equation $\mathbf{A}'\mathbf{x}' = \mathbf{0}$, can only be satisfied by $\mathbf{x}' = \mathbf{0}$, which contradicts our assumption. Therefore, the assumption is untrue and the minimum weight code word which satisfies (2) must have weight $d_1 + 1$ or larger, therefore providing the lemma.

We now prove the general result by induction. When $N = 1$ the matrix $\mathbf{1}$ obviously has rank 1 and thus the minimum distance is 2. To obtain a minimum distance of 3 we need to add $2 - 1 = 1$ to b, thus $N = 2$. Each time we increase the minimum distance by 1 we increase N by $d - 1$, thus

$$N = 1 + \sum_{i=1}^{d-2} i = \frac{(d-2)(d-1)}{2} + 1$$

$2N - 4 = d(d - 3)$ for any value of $d \geqq 2$. Since this is a lower bound on d we can be assured that for any $d \geqq 2$ we can build a code with $2N - 4 \leqq d(d - 3)$.

The field must be of sufficiently large size so that all the "locators" of (6) be different. This can be done if the field has at least $(b - 1)$ $[(d - 2)\ (d - 3) + 2]/2$ nonzero elements. Thus for example we could build a code which corrects two errors $(d = 5)$, has a rate of $\frac{3}{4}$ $(b = 4)$ with a 16 element alphabet. The constraint length would be $Nb = 7 \times 4 = 28$. This could be implemented by using 4-tuples as symbols with an overall constraint of 112 binary digits.

VII. ALTERNATIVE CODE

It is also possible to obtain a minimum distance of $N + 1$ if one uses only a certain set of symbols as locators. In the previous section one needed a field with at least $(b - 1)$ $[(d - 2)\ (d - 3) + 2]/2$ nonzero symbols. In this section we show that N can be made equal to $(d - 1)$ if one is willing to use symbols from a much larger field. For this purpose we define B by:

$$\mathbf{B} = \begin{bmatrix} 1 & 1 & \cdots & 1 \\ \gamma_1^1 & \gamma_2^2 & \cdots & \gamma_b^b \\ \beta\gamma_1^2 & \beta\gamma_2^2 & \cdots & \beta\gamma_b^2 \\ \cdots & \cdots & \cdots & \cdots \\ \beta^{v(N)}\alpha_1^{N-1} & \cdots & \cdots & \beta^{v(N)}\alpha_b^{N-1} \end{bmatrix}$$

$$v(i) = \sum_{i=1}^{j-2} i = \frac{(j-2)(j-1)}{2}$$

where γ_i is the locator and β will be defined later. As in Section VI we wish to show that

$$\mathbf{Ax} = \mathbf{0} \tag{7}$$

only if $\mathbf{x}$ has d or more nonzero elements, given that there is at least one nonzero element among the first b. Suppose there exists a code word $\mathbf{x}$ with weight $d-1$ or less such that (7) is met. Then take the N by $d-1$ matrix consisting of the $d-1$ columns of $\mathbf{A}$ corresponding to the nonzero components of $\mathbf{x}$, and call it $\mathbf{A}'$. The determinant of $\mathbf{A}'$ is a polynomial in β, where the lowest power of β is no lower than that found along the main diagonal. This comes about because the right-hand columns are shifted down and thus contribute least to the power of β in the bottom rows. The highest power of β is shown in the appendix to differ from the lowest power by no more than:

$$\left[\frac{N}{3}\right]\left[\frac{N+1}{3}\right]\left\{2\left[\frac{N+2}{3}\right]-1\right\}^{*}.$$

If we call the difference between the highest and lowest power of β, r, then the determinant can only be zero when β is a root of an rth order or smaller polynomial with coefficients from the field containing the γ_i's. Consequently we can choose β from an $r+1$ order extension field such that it is a root of an irreducible $r+1$ order polynomial. Then it cannot be the root of a polynomial of degree r or less, and the determinant cannot be zero. Consequently $\mathbf{x} = \mathbf{0}$ for (7) to be met.

We have now defined α as a member of the field with q^{r+1} elements where $GF(q)$ is the locator field. In other words our symbols are taken from a q^{r+1} element field and the locators α_i are confined to the q element subfield. For example, suppose we require a double-error-correcting code ($N = 4$) with a rate $\frac{3}{4}$. We use the four element field as locators, and since $r+1 = 4$ the symbols must come from a $4^4 = 256$ element field. This results in an overall constraint length of $8 \times 4 \times 4 = 128$ bits.

VIII. IMPLEMENTATION

The encoding can be accomplished by the standard convolutional encoder. The decoding presents the same problems as the decoding of

* $[\cdot]$ is the symbol for the integer part of $\cdot$.

a nonbinary Bose-Chaudhuri-Hocquenghem code. One takes increasing estimates of the number of errors within a constraint length and solves for the error locations and values. If the estimate is incorrect there is no consistent solution for the error values. The problem of error propagation can be easily handled by introducing a second set of syndrome calculators which are not changed when errors are corrected. A guard space equal to the constraint length, with no errors, will then produce zeros in all these syndromes, indicating that there have been no errors in the last constraint length.

In both coding and decoding, symbols must be added and multiplied. This is easy to implement when the symbols are represented as binary k tuples. Addition is just bit by bit modulo two addition; multiplication by α (the primitive element) is accomplished by a linear shift register.[4]

IX. CONCLUSIONS

We have presented a class of algebraic convolutional codes. These codes can be compared to other convolutional codes, such as Robinson and Bernstein's self-orthogonal codes.[8] The Robinson–Bernstein codes are binary error correcting codes which have a constraint length bounded by $N \geqq [(2t^2 - t)(b - 1) + 1]b$. Our codes have a constraint length bounded by $N \leqq (2t^2 - t + 1)b$, which is less than the above for $b \geqq 2$. Ignoring the fact that our code is a character error correcting code, our code is more powerful than the Robinson–Bernstein code.

If one takes account of the fact that characters are several bits long, one can still make a comparison by taking the constraint length of our code as

$$N \leqq (2t^2 - t + 1)b(\{ n_2[(b - 1)(2t^2 - 3t + 1) + b]\} + 1),$$

$[\cdot]$ integer of.

One now finds that our code is more powerful than the Robinson–Bernstein code for sufficiently large b. More specifically, if $t^2 \leqq 2^b/16b$ our code is more powerful. The Robinson-Bernstein codes have the advantage that they can be decoded by majority logic devices while we require much more complex devices.

The codes described in this section may be useful when one desires a high rate code. Here the amount of storage needed at the encoder can be made equal to the redundancy per constraint length. The decoder must store the entire constraint length in order to make cor-

rections but the part of storage needed to compute the error locations and values is only equal to the redundancy. The disadvantage is that the decoder must perform finite field multiplication, division, and addition.

The code is also capable of correcting erasures; decoding is much simpler in this case. Therefore it may be advantageous to let the k tuples (which represent symbols) be code words in a binary subcode. The subcode then could be used to detect binary errors (producing an erasure), and the convolutional code used to correct these erasures. This is a convolutional version of Forney's concatenated block codes.[9]

The decoding algorithm is also simple for any code of distance four or less. This includes the single-error correcting code, the double-erasure correcting code, and the single-error-plus-single-erasure code. The decoding complexity is comparable to that of Hamming codes.

APPENDIX

Bound on the Powers of β

We wish to bound the difference between the maximum and minimum power of β in the determinant of $\mathbf{A}'$. The matrix will be made up of columns taken from $\mathbf{B}$ or columns from $\mathbf{B}$ shifted downward. This shifting downward of columns is the only parameter which effects our bound. Therefore we take Z_i as the amount that the ith column is shifted; the ith column is headed by Z_i zeros. By ordering we make Z_i a nondecreasing function of i. We can assume that $Z_i < i$; otherwise this would put a zero on the main diagonal and produce a zero determinant. One could then take the largest nonsingular upper left minor $\mathbf{A}''$ and write $\mathbf{A}''\mathbf{X}'' = \mathbf{0}$. The only difference here is that $\mathbf{A}''$ has smaller size; but the bound will still be valid.

The determinant of the matrix is given by

$$\sum_{\sigma} \prod_{i=1}^{N} b_{i,\sigma(i)} \tag{8}$$

where σ is a permutation of the numbers 1 to N. The term $b_{i,\sigma(i)}$ is zero if $\sigma(i) \leqq Z_i$; otherwise it contains β to the $\{[\sigma(i) - Z_i - 2] \cdot [\sigma(i) - Z_i - 1]\}/2$ power. For any given σ the product of (8) contains β to the

$$\tfrac{1}{2} \sum_{i=1}^{N} [\sigma(i) - Z_i - 2][\sigma(i) - Z_i - 1] \tag{9}$$

power, provided that for all i $\sigma(i) > Z_i$. Equation (9) can be rewritten

$$\tfrac{1}{2}\sum_{i=1}^{N}\sigma^2(i) - 3\sigma(i) - 2(i)Z_i + (Z_i + 2)(Z_i + 1). \tag{10}$$

The only term which depends on σ is

$$-\sum_{i=1}^{N}\sigma(i)Z_i\ . \tag{11}$$

The minimum power of β in (8) is given by (10) when we minimize (11). Since Z_i is nondecreasing the term which minimizes (11) is clearly $\sigma(i) = i$. Furthermore, the only other terms of (8) which contribute to this lowest power of β come from σ's which permute the i's over regions where Z_i is constant.

The difference between the maximum power of β and the minimum power is

$$\begin{aligned} D &= \sum_{i=1}^{N} iZ_i - \operatorname*{Min}_{\sigma,\sigma(i)>Z_i}\sum_{i=1}^{N}\sigma(i)Z_i \\ &= \operatorname*{Max}_{\sigma,\sigma(i)>Z_i}\sum_{i=1}^{N}[i - \sigma(i)]Z_i = \sum_{i=1}^{N}(i - \sigma_{\min}(i)]Z_i\ . \end{aligned} \tag{12}$$

The minimization in (12) is accomplished by assigning the smallest i to the largest Z_i, subject to the constraint that $\sigma(i) > Z_i$. This can be done by starting with Z_N and assigning the smallest possible integer to $\sigma(N)$, namely $Z_N + 1$. The term Z_{N-1} then receives the next smallest integer which is still free, and so on. This is the minimum as any other acceptable σ can be converted into this $\sigma_{\min}$ by a sequence of pairwise permutations which do not increase the sum. In Fig. 2 we show an example of the assignment on a square array where the 0's indicate the σ which minimizes the sum and the x's that which maximize it $[\sigma(i) = i]$. The 0's in Fig. 2 are not exactly as described above but are permuted somewhat above equal value Z_i's for reasons which will be clarified in this appendix.

We now show that we can increase various Z_i's to Z_i''s so that $\Delta_i' = Z_{i+1}' - Z_i' \leqq 1$ (in Fig. 2 we increase Z_4, Z_5 and Z_8) and D will not decrease. When we change the Z_i, $\sigma_{\min}(i)$ does not change. When $\Delta_i \geqq 1$, our construction of $\sigma_{\min}(i)$ gives $\sigma_{\min}(i) = Z_{i+1} < i + 1$; thus $\sigma_{\min}(i) \leqq i$, and $i - \sigma_{\min}(i) \geqq 0$. Once we have the new Z_i' with $Z_{i+1}' - Z_i' \leqq 1$ we write $D' \geqq D$ in terms of a sequence

$$\Delta i' = Z_{i+1}' - Z_i' = \begin{cases} 0 \\ \blacksquare\cdot \\ 1 \end{cases}$$

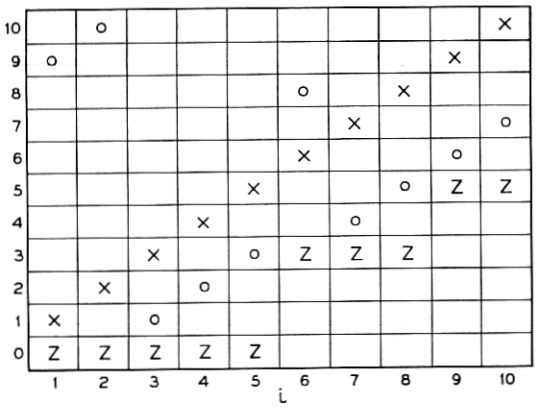

Fig. 2 — Bounds on the minimum and maximum powers of β in the determinant of **A**.

Since

$$Z_i' = \sum_{j=1}^{i-1} \Delta_j' ,$$

then

$$\begin{aligned} D' &= \sum_{i=1}^{N} [i - \sigma(i)] Z_i' = \sum_{i=1}^{N} [i - (i)] \sum_{j=1}^{i-1} \Delta_j' \\ &= \sum_{j=1}^{N-1} \Delta_j' \sum_{i=j+1}^{N} [i - \sigma(i)] \\ &= \sum_{j=1}^{N-1} \Delta_j' \left[\frac{(N-j)(N+j+1)}{2} - \sum_{i=j+1}^{N} \sigma(i) \right]. \end{aligned} \tag{13}$$

We observe that our choice of $\sigma(i)$ gives

$$\sum_{i=j+1}^{N} \sigma(i) = \sum_{i=Z_j+2}^{N-j+Z_j+1} i = [N-j] \left[\frac{N-j+3}{2} + Z_j \right],$$

when $\Delta_j' = 1$,

and (13) becomes

$$\sum_{j=1}^{N-1} \Delta_j' (N-j)(j-1-Z_j).$$

For each value of $Z_j(0, 1, 2, \cdots)$ there can be only one nonzero Δ_j'; if

one takes K to be the number of nonzero Δ_i', then

$$D' = \sum_{k=0}^{K-1} (N - f_k)(f_k - 1 - k),$$

where f_k is that value of j for which $Z_i = k$ and $\Delta_i' = 1$. We now show that

$$\sum_{k=0}^{K-1} (N - f_k)(f_k - 1 - k) \leqq \sum_{i=f_0}^{N} (N - i)(f_0 - 1).$$

The terms on the right are all nonnegative and for every k on the left there is an i on the right with $i = f_k$. For that term $f_k \leqq f_0 + k$, as f_k cannot increase faster than k.

$$f_k - 1 - k \leqq f_0 - 1,$$

$$(N - f_k)(f_k - 1 - k) \leqq (N - f_k)(f_0 - 1).$$

$$D' \leqq \sum_{i=f_0}^{N} (N - i)(f_0 - 1) = (f_0 - 1)\frac{(N - f_0)(N - f_0 + 1)}{2}. \quad (14)$$

The integer value of f_0 which maximizes this is

$$f_0 = \left[\frac{N}{3}\right] + 1, \qquad [\cdot] = \text{integer part of } \cdot$$

and

$$D \leqq \left[\frac{N}{3}\right]\left[\frac{N+1}{3}\right]\left[\left\{2\left[\frac{N+2}{3}\right] - 1\right\}\right]. \quad (15)$$

This bound on D can actually be achieved for any N.

REFERENCES

1. Reed, I. S., and Solomon, G., "Polynomial Codes over Certain Finite Fields," J. Soc. Ind. Appl. Math., *8*, No. 2 (June 1960), pp. 300–304.
2. Wyner, A., and Ash, R., "Analysis of Recurrent Codes," IEEE Trans. on Inform. Theory, *9*, No. 3 (July 1963), pp. 143–160.
3. Wyner, A., "Some Results on Burst-Correcting Recurrent Codes," IEEE Conv. Rec., Part 4 (1963), 139–152.
4. Peterson, W., *Error Correcting Codes*, Cambridge: M.I.T. Press, 1961.
5. Hagelbarger, D. W., "Recurrent Codes; Easily Mechanized Burst-Correcting Codes," B.S.T.J., *38*, No. 4 (July 1959), pp. 969–984.
6. Berlekamp, E. R., "Note on Recurrent Codes," IEEE Trans. on Inform. Theory, *10*, No. 3 (July 1964), p. 257.
7. Massey, J. L., "Implementation of Burst-Correcting Convolutional Codes," IEEE Trans. on Inform. Theory, *11*, No. 3 (July 1965), pp. 416–422.
8. Robinson, J. P. and Bernstein, A. J., "A Class of Binary Recurrent Codes with Limited Error Propagation," IEEE Trans. on Inform. Theory, *13*, No. 1 (January 1967), pp. 106–113.
9. Forney, G. D., *Concatenated Codes*, Cambridge: M.I.T. Press, 1966.