Chapter 6

Modifying Codes

If one code is in some sense good, then we can hope to find from it similar and related codes that are also good. In this chapter we discuss some elementary methods for modifying a code in order to find new codes. In two further sections we discuss special cases related to generalized Reed-Solomon codes.

6.1 Six basic techniques

A code C has three fundamental parameters—its length n, its dimension k, and its redundancy $r = n-k$. Each of these parameters has a natural interpretation for linear codes, and although the six basic modification techniques are not restricted to linear codes it will be easy initially to describe them in these terms. Each fixes one parameter and increases or decreases the other two parameters accordingly. We have:

(i) *Augmenting.* Fix n; increase k; decrease r.
(ii) *Expurgating.* Fix n; decrease k; increase r.
(iii) *Extending.* Fix k; increase n; increase r.
(iv) *Puncturing.* Fix k; decrease n; decrease r.
(v) *Lengthening.* Fix r; increase n; increase k.
(vi) *Shortening.* Fix r; decrease n; decrease k.

The six techniques fall naturally into three pairs, each member of a pair the inverse process to the other. Since the redundancy of a code is its "dual dimension," each technique also has a natural dual technique.

6.1.1 Augmenting and expurgating

In augmenting or expurgating a code we keep its length fixed but vary its dimension and redundancy.

When *augmenting* a code C we add codewords to C. augmenting expurgating

The inverse process of *expurgating* a code is the throwing out of codewords. Notice that augmentation may cause the minimum distance to decrease, while expurgation will not decrease minimum distance and may, in fact, increase it. For generalized Reed-Solomon codes, we always have

$$\mathrm{GRS}_{n,k-1}(\boldsymbol{\alpha},\mathbf{v}) \leq \mathrm{GRS}_{n,k}(\boldsymbol{\alpha},\mathbf{v})\,.$$

Therefore the second code results from augmenting the first, and the first from expurgating the second. In this case the expurgated code has larger minimum distance.

A linear code can be easily augmented by adding rows to a generator matrix and expurgated by taking away rows. A typical way of augmenting a linear code is by adding the row vector composed entirely of 1's to its generator matrix. (Of course, in certain cases this vector will already belong to the code; so the action is inappropriate.)

Increasing the size of a linear code is the same as decreasing the size of its dual, so these two techniques are dual to each other as well as inverse. The dual of augmenting a code by the all 1's vector is expurgating by keeping only those codewords whose entries sum to 0.

These techniques are used for nonlinear codes as well. Consider a linear code C that is a subcode of the linear code D. We can create new codes, not necessarily linear, that are augmentations of C and expurgations of D, by taking the union of certain cosets of C in D. For instance, we might choose those cosets whose coset leaders had largest weight. This method has produced some nonlinear codes that have better properties (in this case minimum distance) than any linear code with the same length and size.

If K is a subfield of the field F and C is a code over F, then we can expurgate C by keeping only those codewords all of whose entries belong to K. This *subfield subcode* inherits many of the properties of the original code and may have further nice properties as well. This extremely important type of expurgation will be discussed at length in a later chapter.

subfield subcode

6.1.2 Extending and puncturing

In extending or puncturing a code we keep its dimension fixed but vary its length and redundancy. These techniques are exceptional in that they are one-to-one. Issues related to the extending and puncturing of *GRS* codes will be discussed in the next two sections.

extending
puncturing

When *extending* a code we add extra redundancy symbols to it. The inverse is *puncturing*, in which we delete redundancy symbols. Puncturing may cause the minimum distance to decrease, but extending will not decrease the minimum distance and may, in fact, increase it. (See Problem 6.1.1 below.) To extend a linear code we add columns to its generator matrix, and to puncture the code we delete columns from its generator.

coordinate extension

Let us call the $[n+1,k]$ linear code C^+ a *coordinate extension* of C if it results from the addition of a single new redundancy symbol to the $[n,k]$ linear code C over the field F. Each codeword $\mathbf{c}^+ = (c_1, \ldots, c_n, c_{n+1})$ of the extended code C^+ is constructed by adding to the codeword $\mathbf{c} = (c_1, \ldots, c_n)$ of C a new coordinate $c_{n+1} = \sum_{i=1}^n a_i c_i = \mathbf{a} \cdot \mathbf{c}$, for some fixed $\mathbf{a} = (a_1, \ldots, a_n) \in F^n$. Here we imply that the new coordinate is the last one, but this is not necessary. A coordinate extension can add a new position at any place within the original code.

Although it is formally possible, it makes little sense to add a coordinate determined by a vector $\mathbf{a}$ of $C^\perp$, since each new c_{n+1} would be 0. We thus assume that $\mathbf{a} \notin C^\perp$. In this case, the subcode $C_0 = C \cap \mathbf{a}^\perp$ of C has dimension $k-1$. We call C_0 the *kernel* of the extension. Replacing the vector $\mathbf{a}$ by any other vector of the coset $\mathbf{a} + C^\perp$ leaves the extension C^+ and the kernel C_0 unchanged. Replacing $\mathbf{a}$ by a nonzero scalar multiple gives an extension of C diagonally equivalent to C^+ and with the same kernel C_0. kernel

The kernel C_0 is that subcode of C that has 0 added in the extension position c_{n+1} of C^+. If G_0 is a generator matrix for the kernel C_0 and $\mathbf{c} \in C - C_0$, then one generator matrix for the coordinate extension C^+ is

$$\left[\begin{array}{c|c} G_0 & \mathbf{0} \\ \hline \mathbf{c} & c_{n+1} \end{array}\right] ,$$

where $\mathbf{0}$ is a column vector of $k-1$ entries equal to 0. Conversely, for any linear $[n, k-1]$ subcode C_0 of C, there is a coordinate extension C^+ of C with kernel C_0. The extension C^+ can be constructed via a generator matrix as above or, equivalently, by choosing a vector $\mathbf{a}$ in $C_0^\perp - C^\perp$.

The most typical method of extending a code is the appending of an overall parity check symbol, a final symbol chosen so that the entries of each new codeword sum to zero. This corresponds to a coordinate extension in which $\mathbf{a}$ has all of its entries equal to -1. For binary codes, this is the usual requirement that the 1's of a codeword in the extended code have even parity. For C a binary Hamming code, this leads to the extended Hamming codes as constructed in Section 4.3, although there we added the new coordinate at the front of the codeword rather than the rear. An extended binary Hamming code has minimum distance 4. No matter what position we choose to puncture an extended binary Hamming code, we are left with a code of minimum distance 3. This code must again be a Hamming code by Problem 4.1.3.

(6.1.1) Problem. *Let C^+ be a coordinate extension of C with kernel C_0. If $\mathrm{d}_{\min}(C_0) > \mathrm{d}_{\min}(C)$, prove that $\mathrm{d}_{\min}(C^+) = \mathrm{d}_{\min}(C) + 1$.*

Let $\mathbf{x}'$ denote the vector of length $n-1$ that is gotten by deleting the last entry from the vector $\mathbf{x}$ of length n. Then $\mathrm{GRS}_{n,k}(\boldsymbol{\alpha}, \mathbf{v})$ can be punctured to $\mathrm{GRS}_{n-1,k}(\boldsymbol{\alpha}', \mathbf{v}')$. Clearly this can be repeated and not always in the final coordinate. The relationship between puncturing and correction of erasures in *GRS* codes is discussed in the next section of this chapter. On the other hand $\mathrm{GRS}_{n,k}(\boldsymbol{\alpha}, \mathbf{v})$ is an extension of $\mathrm{GRS}_{n-1,k}(\boldsymbol{\alpha}', \mathbf{v}')$. Extensions of this kind are always possible as long as $n < |F|$. We talk about further extension of *GRS* codes in the final section of the chapter.

(6.1.2) PROBLEM. *Let C be a code with minimum distance d. Prove that C can correct any pattern of g erasures and e errors provided*

$$g + 2e + 1 \leq d\,.$$

*(*HINT: *For a given pattern, consider the code that is C punctured at the erasure locations.)*

6.1.3 Lengthening and shortening

In lengthening or shortening a code we keep its redundancy fixed but vary its length and dimension.

lengthening
shortening

When *lengthening* a code C we increase the length and add codewords to C. The inverse process of *shortening* a code involves the throwing out of codewords and deleting coordinate positions. As such, these operations can be thought of as combinations of the ones discussed above. Lengthening is extending followed by augmenting, and shortening is expurgating followed by puncturing. Since the two constituent operations tend to have opposing influence on the minimum distance, the actual effect of a lengthening or shortening operation upon distance will depended upon the situation.

For linear codes lengthening corresponds to bordering a generator matrix by adding new columns (extending) and the same number of new rows (augmenting). A standard method is to add to the original generator a final column that is entirely 0, and then add a row that is nonzero in this new column, for instance, the vector of all 1's. Thus a coordinate extension D^+ of a linear code D is a lengthening of its kernel $C = D_0$. Lengthening a code is dual to extending, and the special case of adding an all 0 column and all 1 row for C corresponds to extending $C^\perp$ by an overall parity check symbol. Thus in Section 4.3, we started with a lexicographic generator matrix L_m for the dual Hamming code C and bordered it to construct a generator EL_m for the first order Reed-Muller code $RM(1, m)$ whose dual is the extended Hamming code.

(6.1.3) PROBLEM. *Let C^+ be a coordinate extension of the linear code C with kernel C_0. Prove that $(C^+)^\perp$ is an extension of $C_0^\perp$ and a lengthening of $C^\perp$.*

Shortening undoes lengthening by removing a border from a generator matrix. To reverse the standard 0 column lengthening just described, we first find a generator matrix for the longer code that has a unique row in which the last column is nonzero. Then delete that row (expurgating) and the final column (puncturing), leaving a generator matrix for the original code. In fact this reconstructs the original code as the kernel of a coordinate extension in the overall parity check position. Of course this type of shortening can be done with respect to any column. There will also be various other shortenings available, corresponding to deleting borders whose columns have more nonzero entries. Shortening plays a role in constructing noncyclic CRC codes from cyclic codes, as discussed in Section 8.4.1.

Using the $\mathbf{x}$, $\mathbf{x}'$ notation of the previous subsection, we see that the canonical generator matrix for the code $\mathrm{GRS}_{n,k}(\boldsymbol{\alpha}, \mathbf{v})$ is obtained by bordering the canonical generator matrix for $\mathrm{GRS}_{n-1,k-1}(\boldsymbol{\alpha}', \mathbf{v}')$. Therefore we can shorten $\mathrm{GRS}_{n,k}(\boldsymbol{\alpha}, \mathbf{v})$ to $\mathrm{GRS}_{n-1,k-1}(\boldsymbol{\alpha}', \mathbf{v}')$. In general this will not be the same as the $[n-1, k-1]$ shortened code constructed above by deleting the last position from all codewords that finish with 0, the kernel of $\mathrm{GRS}_{n,k}(\boldsymbol{\alpha}, \mathbf{v})$ viewed as an extension in its last coordinate.

(6.1.4) PROBLEM. *Let C be $\mathrm{GRS}_{n,k}(\boldsymbol{\alpha}, \mathbf{v})$, and shorten C to the $[n-1, k-1]$ code D by taking all codewords of C that end in 0 and then deleting this last coordinate. Find vectors $\boldsymbol{\beta}$ and $\mathbf{u}$ with $D = \mathrm{GRS}_{n-1,k-1}(\boldsymbol{\beta}, \mathbf{u})$.*

We can also lengthen and shorten nonlinear codes. Choose a coordinate position, and select from the code only those words that have some fixed entry in that place. Then delete that position from each of these words. This process can be repeated any number of times, leaving the residue of all codewords that match some specific pattern on some specific set of positions, those positions then deleted. We can use this approach to prove the rest of the Asymptotic Plotkin Bound 2.3.9(2):

$$\alpha_m(\delta) \leq 1 - \frac{m}{m-1}\delta\,, \text{ for } 0 \leq \delta \leq \frac{m-1}{m}\,.$$

PROOF OF COROLLARY 2.3.9.

Consider a family $\{C_n\}$ of m-ary codes of unbounded length n and such that the limits

$$\lim_{n\to\infty} d(C_n)/n = \lim_{n\to\infty} \delta(C_n) = \delta$$

and

$$\lim_{n\to\infty} k(C_n)/n = \lim_{n\to\infty} \kappa(C_n) = \kappa$$

both exist. Assume additionally that $\delta \leq (m-1)/m$. We wish to prove

$$\kappa \leq 1 - \frac{m}{m-1}\delta\,.$$

Clearly we may assume that $\delta \neq 0$, so $d(C_n)$ goes to infinity with n. In particular, there is an integer N such that $d(C_n) \geq m$, for all $n > N$.

Let $n > N$, and set $C = C_n$ and $d = d(C_n)$. Define

$$n' = \left\lfloor (d-1)\frac{m}{m-1} \right\rfloor = (d-1) + \left\lfloor \frac{d-1}{m-1} \right\rfloor \;\geq d\,.$$

Since $1 > (m-1)/m \geq 0$,

$$d - 2 < \frac{m-1}{m}n' \leq d-1\,.$$

Let $\mathbf{x}$ be an $(n-n')$-tuple. Shorten the code to a set $C' = C'(\mathbf{x})$ of n'-tuples by choosing all codewords ending in the $(n-n')$-tuple $\mathbf{x}$ and then deleting these last $(n-n')$ positions from the chosen words. Then either C' is empty or the shortened code C' has length n' and minimum distance $d' \geq d$. Furthermore

$$\frac{m-1}{m} < \frac{d}{n'} \leq \frac{d'}{n'}\,,$$

Therefore the Plotkin Bound 2.3.8 can be applied to C' to yield

$$|C'| \leq \frac{d'}{d' - \frac{m-1}{m}n'} \leq \frac{d}{d - \frac{m-1}{m}n'} \leq d\,,$$

since the function $f(x) = x/(x-c)$ is decreasing. There are $m^{n-n'}$ possible choices for $\mathbf{x}$, and each $C'(\mathbf{x})$ has size at most d; so

$$|C| \leq dm^{n-n'}\,.$$

Taking logarithms and dividing by n, we reach

$$\begin{aligned}\frac{\log_m(|C|)}{n} &\leq \frac{\log_m(d) + n - n'}{n}\\ &\leq \frac{\log_m(n)}{n} + 1 - \frac{m}{m-1}\frac{d-2}{n}\,.\end{aligned}$$

Therefore, for all $n > N$,

$$\kappa(C_n) \leq 1 - \frac{m}{m-1}\delta(C_n) + n^{-1}\left(\log_m(n) + 2\frac{m}{m-1}\right)\,,$$

which, in the limit, is the desired bound. □

6.2 Puncturing and erasures

In Subsection 6.1.2 we saw that the correction of erasures and errors can be dealt with through correction of errors for a suitable punctured code. Indeed the following theorem is a special case of Problem 6.1.2.

(6.2.1) THEOREM. *The code* $\mathrm{GRS}_{n,k}(\boldsymbol{\alpha},\mathbf{v})$ *can be used to correct any pattern of g erasures and e errors provided*

$$g + 2e \leq n - k\,.$$

We are interested in proving the theorem by displaying a specific algorithm for decoding. We shall see that a simple modification of Euclidean algorithm decoding allows us to find the error and erasure locations, at which point the algorithm of Proposition 3.3.3 can be used to find all values.

Remember that for the code $C = \mathrm{GRS}_{n,k}(\boldsymbol{\alpha},\mathbf{v})$ we defined

$$L(x) = \prod_{i=1}^{n}(x - \alpha_i)$$

and

$$L_i(x) = L(x)/(x-\alpha_i)\,.$$

Let J be a subset of the coordinate positions, and consider the code $C_J = \mathrm{GRS}_{n-g,k}(\boldsymbol{\alpha}_J, \mathbf{v}_J)$, gotten by puncturing C at the coordinate positions of the set $\bar{J}$, the complement of J, with $|\bar{J}| = g$. For C_J we have

$$L_J(x) = \prod_{i\in J}(x-\alpha_i)$$

and

$$L_{J,i}(x) = L_J(x)/(x-\alpha_i)\,,$$

for $i \in J$. If we let

$$L_{\bar{J}}(x) = \prod_{i\in \bar{J}}(x-\alpha_i) = \prod_{i\not\in J}(x-\alpha_i)\,,$$

then

$$L(x) = L_J(x)L_{\bar{J}}(x) \quad \text{and} \quad L_i(x) = L_{J,i}(x)L_{\bar{J}}(x)\,.$$

By Theorem 5.1.6 the dual of C is $\mathrm{GRS}_{n,n-k}(\boldsymbol{\alpha}, \mathbf{u})$, where

$$u_i = \frac{1}{v_i L_i(\alpha_i)}\,;$$

and the dual of C_J is $\mathrm{GRS}_{n-g,n-g-k}(\boldsymbol{\alpha}_J, \tilde{\mathbf{u}})$, where

$$\tilde{u}_i = \frac{1}{v_i L_{J,i}(\alpha_i)}\,,$$

for $i \in J$. Notice that we do not get $\tilde{\mathbf{u}}$ by simple puncturing of $\mathbf{u}$ (and so we do not write $\mathbf{u}_J$.) Nevertheless $\tilde{\mathbf{u}}$ is easy to calculate. For $i \in J$,

$$\begin{aligned}
\tilde{u}_i &= \frac{1}{v_i L_{J,i}(\alpha_i)} \\
&= \frac{1}{v_i(L_i(\alpha_i)/L_{\bar{J}}(\alpha_i))} \\
&= \frac{L_{\bar{J}}(\alpha_i)}{v_i L_i(\alpha_i)} \\
&= L_{\bar{J}}(\alpha_i)u_i\,.
\end{aligned}$$

We have proven

(6.2.2) PROPOSITION. *The dual of* $\mathrm{GRS}_{n-g,k}(\boldsymbol{\alpha}_J, \mathbf{v}_J)$ *is*

$$\mathrm{GRS}_{n-g,n-g-k}(\boldsymbol{\alpha}_J, \tilde{\mathbf{u}})\,,$$

where $\tilde{u}_i = L_{\bar{J}}(\alpha_i)u_i$. □

We return to Theorem 6.2.1. Suppose we receive the word $\mathbf{p}$ which contains g erasures at the positions $\bar{J}$. To find the locations of the errors (as opposed to erasures) we decode the punctured received word $\mathbf{p}_J$ using the punctured code C_J. As C_J corrects $\lfloor (n-g-k+1)/2 \rfloor$ errors, this already proves the theorem without an algorithm.

We now describe the error location algorithm, following that of Theorem 5.2.4. We first calculate the J-syndrome:

$$\begin{aligned} S_J(z) &= \sum_{i \in J} \frac{L_{\bar{J}}(\alpha_i) u_i p_i}{1-\alpha_i z} \pmod{z^{r-g}} \\ &= \sum_{i=1}^{n} \frac{L_{\bar{J}}(\alpha_i) u_i p_i}{1-\alpha_i z} \pmod{z^{r-g}}\,. \end{aligned}$$

Next we step through the Euclidean algorithm with the initialization

$$a(z) = z^{r-g} \text{ and } b(z) = S_J(z)$$

until a step j is reached where $\deg(r_j(z)) < (r-g)/2$. We can then find the error locations I in J from the calculated $\sigma_J(z)$ (and $\omega_J(z)$).

Of course we could at the same time find the error values at the locations in I, but we would still need to find the values at the erasure locations in $\bar{J}$. It is probably more efficient to use the algorithm of Proposition 3.3.3 to find all values at the $g + e \le d-1$ error and erasure locations $I \cup \bar{J}$ simultaneously.

6.3 Extended generalized Reed-Solomon codes

Let $n > 1$, and consider n-tuples from the field F with the following properties:

(i) $\mathbf{w} = (w_1, w_2, \ldots, w_n) \in F^n$ has all its entries w_i not 0;

(ii) $\boldsymbol{\beta} = (\beta_1, \beta_2, \ldots, \beta_n) \in F^n$ and $\boldsymbol{\gamma} = (\gamma_1, \gamma_2, \ldots, \gamma_n) \in F^n$ satisfy

$$\beta_i \gamma_j \neq \beta_j \gamma_i\,, \text{ for all } i \neq j\,.$$

extended generalized Reed-Solomon code

For $k > 0$ the *extended generalized Reed-Solomon code* $\mathrm{EGRS}_{n,k}(\boldsymbol{\beta}, \boldsymbol{\gamma}; \mathbf{w})$ is the code C composed of all codewords

$$\mathbf{ev}_{\boldsymbol{\beta},\boldsymbol{\gamma};\mathbf{w}}(f) = (w_1 f(\beta_1, \gamma_1), \ldots, w_i f(\beta_i, \gamma_i), \ldots, w_n f(\beta_n, \gamma_n))\,,$$

where $f = f(x, y)$ runs through all polynomials of $F[x, y]$ that are homogeneous of degree $k-1$:

$$f(x, y) = f_0 y^{k-1} + f_1 x y^{k-2} + f_2 x^2 y^{k-3} + \cdots + f_{k-1} x^{k-1} \text{ with } f_i \in F\,.$$

The condition (ii) states that, for all distinct i, j, there is no $c \in F$ with $(\beta_i, \gamma_i) = c(\beta_j, \gamma_j)$. It should be thought of as saying that

$$\beta_i/\gamma_i \neq \beta_j/\gamma_j\,, \text{ for all } i \neq j\,,$$

but care must be taken since we allow the possibility $\gamma_j = 0$. There is at most one j with $\gamma_j = 0$, and in that case $\beta_j \neq 0$ since $n > 1$.

For each i, let α_i be the ratio β_i/γ_i, where we write ∞ for $\beta_j/\gamma_j = \beta_j/0$. Then by (ii) the α_i are all distinct. Let $\boldsymbol{\alpha} = (\alpha_1, \alpha_2 \dots, \alpha_n)$. Further let $\boldsymbol{\alpha}'$ be $\boldsymbol{\alpha}$ punctured at position j where $\alpha_j = \infty$, if such a position exists.

For each i, set $v_i = w_i\gamma_i^{k-1}$; and let $\mathbf{v} = (v_1, v_2, \dots, v_n)$. All entries of $\mathbf{v}$ are nonzero with the possible exception of that v_j where $\gamma_j = 0$ and $\alpha_j = \infty$, if such a j exists. In that case let $\mathbf{v}'$ be $\mathbf{v}$ punctured at position j.

We first check that the code we have defined is really an extension of a generalized Reed-Solomon code.

(6.3.1) Theorem. *Let $C = \mathrm{EGRS}_{n,k}(\boldsymbol{\beta}, \boldsymbol{\gamma}; \mathbf{w})$. If $\alpha_j = \infty$ and $\gamma_j = 0$ then the code gotten by puncturing C at position j is $\mathrm{GRS}_{n-1,k}(\boldsymbol{\alpha}', \mathbf{v}')$. If no such j exists, then $C = \mathrm{GRS}_{n,k}(\boldsymbol{\alpha}, \mathbf{v})$.*

Proof. Let C' be the code gotten by puncturing C at j where $\alpha_j = \infty$. If no such j exists, let $C' = C$.

With each degree $k-1$ homogeneous polynomial $f(x, y)$ as above, we associate a polynomial $\hat{f}$ in the single indeterminate $\frac{x}{y}$:

$$\hat{f}\left(\frac{x}{y}\right) = \frac{1}{y^{k-1}} f(x,y) = f_0 + f_1\left(\frac{x}{y}\right) + f_2\left(\frac{x}{y}\right)^2 + \cdots + f_{k-1}\left(\frac{x}{y}\right)^{k-1}.$$

The polynomial $\hat{f}(\frac{x}{y})$ has degree at most $k-1$ and satisfies

$$\hat{f}(\alpha_i) = \frac{1}{\gamma_i^{k-1}} f(\beta_i, \gamma_i).$$

Therefore, for any i with $\alpha_i \neq \infty$, the i^{th} entry of the codeword

$$\mathbf{ev}_{\boldsymbol{\beta},\boldsymbol{\gamma};\mathbf{w}}(f)$$

in the code $C = \mathrm{EGRS}_{n,k}(\boldsymbol{\beta}, \boldsymbol{\gamma}; \mathbf{w})$ equals that of the codeword

$$\mathbf{ev}_{\boldsymbol{\alpha}',\mathbf{v}}(\hat{f})$$

in the generalized Reed-Solomon code $\mathrm{GRS}_{n',k}(\boldsymbol{\alpha}', \mathbf{v}')$. That is,

$$C' = \mathrm{GRS}_{n',k}(\boldsymbol{\alpha}', \mathbf{v}'). \qquad \Box$$

A canonical generator matrix for C has rows $\mathbf{ev}_{\boldsymbol{\beta},\boldsymbol{\gamma};\mathbf{w}}(f)$ as $f = f(x, y)$ runs through the basis $y^i x^{k-1-i}$ of the space of homogeneous polynomials of degree $k-1$. This matrix is also obtained by adding to the canonical generator matrix for $\mathrm{GRS}_{n-1,k}(\boldsymbol{\alpha}', \mathbf{v}')$ at position j (where $\alpha_j = \infty$) a column that is all 0 except for the entry $w_j\beta_j^{k-1}$ in its last row. (Compare Problem 5.1.4.) In particular $\mathrm{GRS}_{n-1,k-1}(\boldsymbol{\alpha}', \mathbf{v}')$ is revealed as the kernel of the coordinate extension of $\mathrm{GRS}_{n-1,k}(\boldsymbol{\alpha}', \mathbf{v}')$ at position j that produces the extended code $\mathrm{EGRS}_{n,k}(\boldsymbol{\beta}, \boldsymbol{\gamma}; \mathbf{w})$. In particular, the next theorem is a consequence of the previous theorem and Problem 6.1.1. Instead we give a proof following that of the corresponding result for *GRS* codes, Theorem 5.1.1.

(6.3.2) THEOREM. *The code* $\mathrm{EGRS}_{n,k}(\boldsymbol{\beta}, \boldsymbol{\gamma}; \mathbf{w})$ *is an* $[n, k]$ *linear code over* F *with minimum distance* $n - k + 1$.

PROOF. The only thing that needs careful checking is that the minimum distance is at least $n - k + 1 = n - (k - 1)$.

Let $f(x, y)$ be a homogeneous polynomial of degree $k - 1$, and let $\hat{f}(\frac{x}{y})$ be its associated polynomial. As all the w_i are nonzero, the number of entries 0 in $\mathbf{f} = \mathbf{ev}_{\boldsymbol{\beta},\boldsymbol{\gamma};\mathbf{w}}(f)$ equals the number of i with $f(\beta_i, \gamma_i) = 0$. We must prove there are at most $k-1$ such i. There are two cases to consider, depending upon whether or not $f(\beta_j, \gamma_j) = 0$ for a j with $\gamma_j = 0$ and $\alpha_j = \infty$.

First consider those 0's of $\mathbf{f}$ that occur at positions i for which $\gamma_i \neq 0$. Each corresponding α_i is a root of the polynomial $\hat{f}$, and there are at most $\deg(\hat{f})$ roots. In particular, in the case where all 0's of $\mathbf{f}$ occur at such positions i, there are at most $k - 1 \leq \deg(\hat{f})$ places equal to 0, as required.

Now assume that $\gamma_j = 0$ and $f(\beta_j, 0) = 0$, that is,

$$\begin{aligned} 0 &= f_0 0^{k-1} + f_1\beta_j 0^{k-2} + f_2\beta_j^2 0^{k-3} + \cdots + f_{k-2}\beta_j^{k-2}0^1 + f_{k-1}\beta_j^{k-1} \\ &= f_{k-1}\beta_j^{k-1}\,. \end{aligned}$$

As $\beta_j \neq 0$, we must have $f_{k-1} = 0$ in this case. Therefore the degree of $\hat{f}$ is in fact at most $k-2$. So even here there are at most $k-1$ places where $\mathbf{f}$ is 0, one at position j and at most $\deg(\hat{f}) \leq k - 2$ at other locations. □

(6.3.3) PROBLEM. *Prove that the dual of an EGRS code is also an EGRS code.*

(6.3.4) THEOREM. *Let* $a, b, c, d, e \in F$ *with*

$$ad - bc \neq 0 \textit{ and } e \neq 0\,.$$

Then

$$\mathrm{EGRS}_{n,k}(\boldsymbol{\beta}, \boldsymbol{\gamma}; \mathbf{w}) = \mathrm{EGRS}_{n,k}(\tilde{\boldsymbol{\beta}}, \tilde{\boldsymbol{\gamma}}; \tilde{\mathbf{w}})\,,$$

where

$$\begin{aligned} \tilde{\beta}_i &= a\beta_i + b\gamma_i, \\ \tilde{\gamma}_i &= c\beta_i + d\gamma_i, \\ and \tilde{w}_i &= ew_i\,. \end{aligned}$$

PROOF. The proof consists mainly of calculation. The crucial observation is that, for any homogeneous polynomial $f(x, y)$ of degree $k - 1$ and for the quadruple $r, s, t, u \in F$, the polynomial $f(rx + sy, tx + uy)$ is also homogeneous of degree $k - 1$, provided $rx + sy \neq 0 \neq tx + uy$.

The number $\Delta = ad - bc$ is the determinant of the matrix

$$\begin{bmatrix} a & b \\ c & d \end{bmatrix} .$$

As Δ is nonzero we can solve for β_i and γ_i and find that

$$\begin{aligned} \beta_i &= \Delta^{-1}(d\tilde{\beta}_i - b\tilde{\gamma}_i) , \\ \gamma_i &= \Delta^{-1}(-c\tilde{\beta}_i + a\tilde{\gamma}_i) . \end{aligned}$$

As $e \neq 0$, the vector $\tilde{\mathbf{w}}$ has no entries 0, since $\mathbf{w}$ has none. We also check that

$$\tilde{\beta}_i\tilde{\gamma}_j - \tilde{\beta}_j\tilde{\gamma}_i = \Delta(\beta_i\gamma_j - \beta_j\gamma_i) \neq 0 ,$$

giving the defining conditions (i) and (ii) for the vectors $\tilde{\boldsymbol{\beta}}$, $\tilde{\boldsymbol{\gamma}}$, and $\tilde{\mathbf{w}}$.

Starting with the degree $k-1$ homogeneous polynomial $f(x,y)$, we define the new polynomial

$$g(x,y) = \frac{1}{e\Delta^{k-1}} f(dx - by, -cx + ay) .$$

Then

$$\mathbf{ev}_{\boldsymbol{\beta},\boldsymbol{\gamma};\mathbf{w}}(f) = \mathbf{ev}_{\tilde{\boldsymbol{\beta}},\tilde{\boldsymbol{\gamma}};\tilde{\mathbf{w}}}(g) .$$

Therefore each codeword of the first code is also in the second code. As both codes have the same dimension, they must be equal. □

Problems 5.1.2 and 5.1.3 are special cases of this theorem.

The $q+1$ possible ratios $\alpha_i = \beta_i/\gamma_i$ from $\{\infty\} \cup \mathbb{F}_q$ are identified with the projective line over $\mathbb{F}_q$. The *EGRS* codes can thus be thought of as codes defined by functions on the projective line. The group of 2×2 matrices that appears in Theorem 6.3.4 acts naturally on the projective line.

(6.3.5) THEOREM. *If $n \leq |F|$, then $C = \mathrm{EGRS}_{n,k}(\boldsymbol{\beta},\boldsymbol{\gamma};\mathbf{w})$ is equal to $\mathrm{GRS}_{n,k}(\boldsymbol{\alpha},\mathbf{v})$ over F, for appropriate $\boldsymbol{\alpha}$ and $\mathbf{v}$. If $n < |F|$, then $\boldsymbol{\alpha}$ may be chosen with all its entries not equal to 0.*

PROOF. If $n \leq |F|$, then some possible ratio α does not occur among the $\alpha_i = \beta_i/\gamma_i$. If the ratio $\alpha = \infty$ is missing, then C is a *GRS* code by Theorem 6.3.1. If $\gamma_j = 0$ and $\alpha \neq \infty$, then any transformation

$$\begin{bmatrix} a & b \\ c & d \end{bmatrix} = \begin{bmatrix} a & b \\ -1 & \alpha \end{bmatrix}$$

in Theorem 6.3.4 takes $\boldsymbol{\gamma}$ to a vector $\tilde{\boldsymbol{\gamma}}$ with no entry 0; so C is again a *GRS* code by Theorem 6.3.1. If $n < |F|$ then the the values of a, b, c, and d in the transformation can be chosen so that both $\tilde{\boldsymbol{\beta}}$ and $\tilde{\boldsymbol{\gamma}}$ avoid 0. Then $C = \mathrm{GRS}_{n,k}(\tilde{\boldsymbol{\alpha}},\mathbf{v})$ with each entry $\tilde{\alpha} = \tilde{\beta}/\tilde{\gamma}$ of $\tilde{\boldsymbol{\alpha}}$ nonzero. □