Draft of May 8, 1996
7:10 p.m.

# The Algebraic Theory of Convolutional Codes

Robert J. McEliece
Department of Electrical Engineering
California Institute of Technology
Pasadena, California, USA

**Contents:**

Section 1. Preface ..... p. 1
Section 2. Introduction. From Block to Convolutional Codes. ..... p. 3
Section 3. Polynomial Generator Matrices ..... p. 12
Section 4. The Hilbert Series and Free Distance Bounds ..... p. 18
Section 5. Physical Encoder Complexity ..... p. 27
Section 6. Catastrophic Generator Matrices and Encoders. ..... p. 40
Section 7. The Dual Code. Wyner-Ash Codes. ..... p. 47
Section 8. Puncturing Convolutional Codes. ..... p. 54
Appendix A. Basic and Reduced Matrices. ..... p. 64
Appendix B. The Extended Smith Algorithm and Applications. ..... p. 68
References. ..... p. 78

## 1. Preface.

This chapter of the *Handbook of Coding Theory* differs from the others in that it deals exclusively with the class of *convolutional*, rather than *block*, codes. In some ways the two classes are quite similar, but in many other ways they are very different.

The simplest way to contrast block and convolutional codes is to compare their encoders. An encoder for an $[n, k]$ linear block code can be thought of as a one-to-one linear mapping $\phi$ from $F^k$ to $F^n$, where $F$ is the "symbol field," usually $GF(2)$. Thus if $\boldsymbol{u} \in F^k$ is a $k$-dimensional "information word," the corresponding $n$-dimensional "codeword" is $\boldsymbol{x} = \phi(\boldsymbol{u})$; and if the encoder is presented with a sequence $\boldsymbol{u}(1), \boldsymbol{u}(2), \ldots$ of information words, it produces a sequence of codewords, $\boldsymbol{x}(1), \boldsymbol{x}(2), \ldots$, where $\boldsymbol{x}(i) = \phi(\boldsymbol{u}(i))$. An encoder for an $(n, k)$ *convolutional* code, on the other hand, can also be thought of as a linear mapping from a sequence of $k$-dimensional information words to a sequence of $n$-dimensional codewords, i.e.,

$$\phi : \boldsymbol{u}(1), \boldsymbol{u}(2), \ldots \rightarrow \boldsymbol{x}(1), \boldsymbol{x}(2), \ldots .$$

The key difference is that for a convolutional code the $i$th codeword $\boldsymbol{x}(i)$ is a function not only of $\boldsymbol{u}(i)$, but also of a fixed number of "earlier" information words, $\boldsymbol{u}(i-1), \ldots, \boldsymbol{u}(i-M)$, where the integer $M$ is called the *memory* of the encoder.* From this viewpoint, a block code is just a convolutional code without memory, which makes the theory of block codes appear to be a degenerate special case of the theory of convolutional codes. This viewpoint is technically correct, but it is nevertheless misleading. For in block coding the general tendency is to consider codes for which $n$ and $k$ are large, whereas for convolutional codes, the usual approach is to study families of codes with small, fixed, values of $n$ and $k$, for various values of $M$. For example, much is known about binary block codes of length (say) 1023, but little or nothing is known about convolutional codes of this length. On the other hand, there is a large body of research devoted to the class of $(2, 1)$ convolutional codes, whereas the class of $[2, 1]$ block codes is completely trivial.

Convolutional codes were invented in 1955 by Elias [11], and were re-introduced, under the name "recurrent codes," by Hagelbarger in 1959 [22]. With the discovery that convolutional codes can be decoded with sequential decoding [58] [12] [26], threshold decoding [37], and above all, Viterbi decoding [54][46][15], they became quite widespread in practice. However, the *theory* of convolutional codes was only imperfectly understood until the classic series of papers by Forney [13][16][17], who showed that the algebra of $k \times n$ matrices over the field of rational functions in the delay operator $D$ over $F$, denoted by $F(D)$, played the same role for convolutional codes, as the algebra of $k \times n$ matrices over $F$ plays for linear block codes. Unfortunately, the algebra of matrices over $F(D)$ is much more complex than the algebra of matrices over $F$, and so the algebraic "basics" for convolutional codes lie much deeper than the corresponding facts for block codes. This chapter is best thought of as a 1996 "spin" on Forney's papers. (Chapter 2 in the book by

---

* This definition is not completely accurate, since there are convolutional encoders which have infinite memory. A precise definition will be given in Section 2.

Piret [48] is also a nice intrduction to the "Forney theory.") This chapter is by no means self-contained, and it assumes that the reader is already familiar with the basic algebraic theory of block codes, and is comfortable with abstract algebra.

*Advice for novices:* The theory of convolutional codes contains much more than the algebraic basics we shall cover here. Indeed, we have not even mentioned a number of important topics, for example, the combinatorial theory of convolutional codes (i.e., the state and trellis diagrams of the code), the decoding of convolutional codes, the construction of good convolutional codes, and above all, the practical applications of convolutional codes. For the reader interested in these topics, we refer to the monograph by Massey [40], and the books by Lin and Costello [35], Dholakia [9], and Wicker [57]. The book by Dholakia contains almost 500 references, and is a good place for a reader who is interested in knowing more about convolutional codes to start.

*Advice for experts:* If you are already familiar with the Forney theory, you will find little that is new here. However, since even after all these years no standard set of terms for many of the basic concepts has emerged, I have introduced a number of new terms, and redefined some old ones, in a serious attempt to arrive at a standard nomenclature. Thus for example you will read here about, for the first time, the *internal degree* and *external degree* of a polynomial matrix. I have retained the definition of the "memory" of a convolutional code, as in "unit-memory" codes, but coined the term *degree* to denote the number of storage elements required by a minimal encoder. I have rechristened the "generalized constraint lengths" as the *Forney indices*, a tribute I think long overdue. Thus the memory of a code is the largest Forney index, while the degree is the sum of the Forney Indices. I have retired the venerable but ambiguous term *constraint length* altogether. Following a recent suggestion of Forney, Johannesson, and Wan, I have interchanged the traditional, but misleading, definitions of *minimal* and *canonical* generator matrices. I have tried to lay to rest the erroneous but surprisingly persistent belief that some convolutional codes are "catastrophic." I think you will enjoy learning about the Hilbert series for a convolutional code (Theorem 4.1), which I learned about from Richard Stanley. I have tried to promote the Wyner-Ash codes to the prominent place they deserve. The "extended Smith algorithm" in Appendix B may prove to be new, too. It gives a practical way to produce a basic generator matrix, or to test for "noncatastrophicness," without actually computing the gcd of all those $k \times k$ submatrices.

I have learned a lot from my friends during the preparation of this chapter, and I would particularly like to thank Ivan Onsyzchuk, Richard Stanley, and P. P. Vaidyanathan for lending me their insights. I hope I can return the favor someday!

## 2. Introduction. From Block to Convolutional Codes.

*Abstract: In this section, we will give a definition of a convolutional code, by first giving an unorthodox definition of an ordinary linear block code, and then generalizing this unorthodox definition into the "standard" definition of a convolutional code. We assume the reader is already familiar with the elementary theory of linear block codes.*

Let $\boldsymbol{C}$ be an $[n, k]$ linear block code over a field $F$, called the "symbol field." If $G$ is a generator matrix for $\boldsymbol{C}$, i.e., a $k \times n$ matrix of rank $k$ with entries from $F$ such that $\boldsymbol{C}$ is the $F$-rowspace of $G$, then $G$ can be used as an *encoder* for $\boldsymbol{C}$, as follows. If $\boldsymbol{u} \in F^k$ is a $k$-dimensional "information vector," or "information word," then with $G$ as encoder, the "codeword" corresponding to $\boldsymbol{u}$ is

$$\boldsymbol{x} = \boldsymbol{u}G. \tag{2.1}$$

Normally, an encoder is used to encode not just one, but a sequence of information words, say $\boldsymbol{u}(0), \boldsymbol{u}(1), \boldsymbol{u}(2), \ldots$. In engineering parlance, $\boldsymbol{u}(i)$ might be called "the information word transmitted at time $i$," and $i$ might be called the "time index" of $\boldsymbol{u}(i)$. In any case, the $i$th codeword $\boldsymbol{x}(i)$ corresponding to the $i$th information word $\boldsymbol{u}(i)$ is

$$\boldsymbol{x}(i) = \boldsymbol{u}(i)G \qquad \text{for } i = 0, 1, \ldots. \tag{2.2}$$

In keeping with this view, let us temporarily *redefine* the code $\boldsymbol{C}$ generated by $G$ to be the set of all possible code *sequences* $(\boldsymbol{x}(0), \boldsymbol{x}(1), \ldots)$ which can be produced by (2.2).

Equation (2.2) can be expressed more concisely if we use *generating functions*.* To do this, let $D$ be an indeterminate, which we call the *delay operator*. Multiply both sides of (2.2) by $D^i$, and sum for $i \geq 0$:

$$\sum_{i \geq 0} \boldsymbol{x}(i) D^i = \left( \sum_{i \geq 0} \boldsymbol{u}(i) D^i \right) G. \tag{2.3}$$

Now if we define the vector generating functions

$$\boldsymbol{X}(D) = \sum_{i \geq 0} \boldsymbol{x}(i) D^i \tag{2.4}$$

$$\boldsymbol{U}(D) = \sum_{i \geq 0} \boldsymbol{u}(i) D^i, \tag{2.5}$$

then (2.3) becomes simply

$$\boldsymbol{X}(D) = \boldsymbol{U}(D)G. \tag{2.6}$$

---

* In the engineering literature, generating functions are often called *z-transforms*, and the indeterminate $z^{-1}$ is used instead of $D$.

Equation (2.1) shows how the encoder $G$ works on a single information vector $\boldsymbol{u}$, while eq. (2.6) shows how it works on an infinite sequence of information vectors. The two equations are identical, except that in (2.1) $\boldsymbol{x}$ and $\boldsymbol{u}$ are elements of the vector spaces $F^n$ and $F^k$, while in (2.6), $\boldsymbol{X}(D)$ and $\boldsymbol{U}(D)$ are more complex objects. $\boldsymbol{X}(D)$ is an $n$-dimensional vector, and $\boldsymbol{U}(D)$ is a $k$-dimensional vector, whose components are *formal power series* of the form

$$A(D) = \sum_{i \geq 0} a_i D^i, \tag{2.7}$$

where $a_i \in F$ for all $i$. The set of all power series of the form (2.7) forms a commutative integral domain, denoted by $F[[D]]$, which can be extended in a natural way to a *field* $F((D))$ consisting of all one-sided *formal Laurent series* of the form

$$A(D) = \sum_{i \geq m} a_i D^i, \tag{2.8}$$

where $m$ can be any integer, positive, negative, or zero. The field $F((D))$ is complete with respect to the exponential valuation $w(A(D)) = \min\{i : a_i \neq 0\}$, and the series in (2.8) always converges with respect to this valuation, no matter what coefficients $a_i$ are chosen.

More about the algebraic properties of $F((D))$ can be found in [55, Chapter 10], [56, Chapter 18], or [3, Chapter 1], and more about the combinatorial properties can be found in [51, Chapter 1]. For future reference, we define here six special kinds of Laurent series that will play important roles in the rest of this Chapter.

• Laurent series of the form $A(D) = a_0 + \cdots + a_L D^L$ are called *polynomials*. The set of all polynomials is denoted by $F[D]$.

• Laurent series of the special form (2.7) are said to be *causal*. Thus a causal Laurent series is just an ordinary power series, and every polynomial is causal. As we have already noted, the set of causal Laurent series is denoted by $F[[D]]$.

• Laurent series of the form

$$A(D) = \sum_{i=-m}^{0} a_i D^i, \tag{2.9}$$

where $m \geq 0$, are said to be *anticausal*. Thus an anticausal Laurent series is just a polynomial in $D^{-1}$; the set of anticausal Laurent series is denoted $F[D^{-1}]$.

• Every rational function $P(D)/Q(D)$ (where $P(D)$ and $Q(D)$ are polynomials with $Q(D) \neq 0$) has a unique expansion as a Laurent series. Such Laurent series are (naturally) called *rational* Laurent series. The set of all rational Laurent series forms a subfield of $F((D))$, called the *rational subfield* of $F((D))$, and denoted by $F(D)$. $F(D)$ is the smallest subfield of $F((D))$ containing $D$ and $F$.

• A Laurent series that is both causal and rational is called a *realizable* Laurent series. Thus a Laurent series is realizable if and only if it is of the form $P(D)/Q(D)$, where $P(D)$ and $Q(D)$ are polynomials, and $Q(0) \neq 0$.

• The number of nonzero coefficients of a Laurent series is called the *weight* of the series, and a Laurent series with only a finite number of nonzero coefficients is called a *finite weight* series. Every finite-weight Laurent series is of the form $P(D)/D^L$, where $P(D)$ is a polynomial and $L$ is a nonnegative integer. Note that a causal, finite weight, Laurent series is simply a polynomial, and that every finite-weight Laurent series is rational.

Let us now further modify the definition of the code generated by $G$. Instead of saying that the code is the set of *sequences* $(\boldsymbol{x}(i))$ produced by (2.2), let us agree that the code is the corresponding set of *generating functions*, i.e., the set of $\boldsymbol{X}(D)$'s that can appear on the left side of (2.6). If we choose to enlarge the set of possible "information sequences" $\boldsymbol{U}(D)$ on the right side of (2.6) from $F[[D]]^k$ to $F((D))^k$ (which amounts to nothing more than allowing the input to the encoder to "start" at any time, rather than just at $i = 0$), then (2.6) says that $\boldsymbol{C}$ is simply the $F((D))$-rowspace of $G$, which, since we have assumed that $G$ has rank $k$, is a $k$-dimensional subspace of $F((D))^n$. Thus an $[n, k]$ linear block code can be viewed as a special kind of $k$-dimensional subspace of $F((D))^n$, and so we offer the following unorthodox definition:

**2.1 Definition.** *An $[n, k]$ linear block code over $F$ is a $k$-dimensional subspace of $F((D))^n$, with a basis consisting entirely of vectors from $F^n$.*

Of course, since the basis of $\boldsymbol{C}$ is required to be in $F^n$, we lose nothing essential by considering only those codewords which also lie in $F^n$, which we might call the *scalar* subcode of $\boldsymbol{C}$. In particular, the minimum weight codewords of $\boldsymbol{C}$ lie in the scalar subcode, and all basis vectors for possible generator matrices lie in the scalar subcode. Thus Definition 2.1 is equivalent, in all important respects, to the usual definition:

**2.2 Definition.** *An $[n, k]$ linear block code over $F$ is a $k$-dimensional subspace of $F^n$.*

Having now taken this unnecessarily general view of block codes, we begin our discussion of convolutional codes, again beginning with an encoder. An $(n, k)$ *convolutional encoder* is a linear device which maps a sequence of $k$-dimensional information words $\boldsymbol{u}(0), \boldsymbol{u}(1), \ldots$, into a sequence of $n$-dimensional codewords $\boldsymbol{x}(0), \boldsymbol{x}(1), \ldots$. In this sense it is the same as block encoder. The difference is that the convolutional encoder has an internal $m$-dimensional "storage vector" or "state vector," $\boldsymbol{s}(i)$, and the $i$th codeword $\boldsymbol{x}(i)$ is a linear function not only of the $i$th input word $\boldsymbol{u}(i)$, but also of the $i$th state $\boldsymbol{s}(i)$. The formal description of the encoder is: $\boldsymbol{s}(0) = 0$ and for $i \geq 0$,

$$\boldsymbol{s}(i+1) = \boldsymbol{s}(i)\mathcal{A} + \boldsymbol{u}(i)\mathcal{B} \tag{2.10}$$

$$\boldsymbol{x}(i) = \boldsymbol{s}(i)\mathcal{C} + \boldsymbol{u}(i)\mathcal{D}, \tag{2.11}$$

where the four matrices $\mathcal{A}$, $\mathcal{B}$, $\mathcal{C}$, and $\mathcal{D}$ have entries from $F$ and dimensions

$$\begin{aligned} \mathcal{A} &: m \times m \\ \mathcal{B} &: k \times m \\ \mathcal{C} &: m \times n \\ \mathcal{D} &: k \times n. \end{aligned} \tag{2.12}$$

The integer $m$ is called the *degree* of the encoder. Thus while an encoder for an $[n, k]$ *block* code is characterized by a *single* $k \times n$ matrix $G$, an encoder for an $(n, k)$ *convolutional* code is characterized by *four* matrices $(\mathcal{A}, \mathcal{B}, \mathcal{C}, \mathcal{D})$ whose dimensions are given by (2.12). If $m = 0$, the matrices $\mathcal{A}$, $\mathcal{B}$, and$C$ disappear and the encoding equations (2.11) and (2.10) reduce to $\boldsymbol{x}(i) = \boldsymbol{u}(i)\mathcal{D}$, which is the same as (2.1), with $G$ replaced by $\mathcal{D}$. Thus a block code is simply a "degree 0" (or *memoryless*) convolutional code. The definition embodied in eqs. (2.11) and (2.10) is called the *state space description* of a convolutional code. It is familiar to system theorists who study discrete-time, multi-input, multi-output linear time-invariant system [52, section 13.4]. The earliest use of the state-space representation for convolutional codes appears to be [38].

The *convolutional code* associated with the encoding matrices $(\mathcal{A}, \mathcal{B}, \mathcal{C}, \mathcal{D})$ is then defined to be the set of all possible output sequences $\boldsymbol{x}(0), \boldsymbol{x}(1), \ldots$ produced by the encoding equations (2.11) and (2.10). Alternatively, and in keeping with the previous discussion of block codes, the convolutional code can be defined as the set of generating functions ($n$-vectors of causal Laurent series) $\boldsymbol{X}(D) = \boldsymbol{x}(0) + \boldsymbol{x}(1)D + \boldsymbol{x}(2)D^2 + \cdots$ corresponding to the encoder output (cf. (2.4)).

It is useful to state the encoding equations in generating-function form. To do this, multiply both sides of (2.11) and (2.10) by $D^i$ and sum over all $i$ (using the fact that $\boldsymbol{u}(i), \boldsymbol{s}(i), \boldsymbol{x}(i)$ are all zero for $i < 0$). The result is

$$\boldsymbol{S}(D)D^{-1} = \boldsymbol{S}(D)\mathcal{A} + \boldsymbol{U}(D)\mathcal{B} \tag{2.13}$$

$$\boldsymbol{X}(D) = \boldsymbol{S}(D)\mathcal{C} + \boldsymbol{U}(D)\mathcal{D}, \tag{2.14}$$

where $\boldsymbol{X}(D)$ and $\boldsymbol{U}(D)$ are as defined in (2.4) and (2.5) and $\boldsymbol{S}(D)$ is defined similarly as

$$\boldsymbol{S}(D) = \sum_{i \geq 0} \boldsymbol{s}(i)D^i. \tag{2.15}$$

It is easy to solve eqs. (2.14) and (2.13) to get an explicit expression for $\boldsymbol{S}(D)$ and $\boldsymbol{X}(D)$ in terms of the input $\boldsymbol{U}(D)$. The result is

$$\boldsymbol{S}(D) = \boldsymbol{U}(D)E(D) \tag{2.16}$$

$$\boldsymbol{X}(D) = \boldsymbol{U}(D)G(D), \tag{2.17}$$

where the matrices $E(D)$ (which is $k \times m$) and $G(D)$ (which is $k \times n$) are given by

$$E(D) = \mathcal{B}(D^{-1}I_m - \mathcal{A})^{-1} \tag{2.18}$$

$$\begin{aligned} G(D) &= \mathcal{D} + E(D)\mathcal{C} \\ &= \mathcal{D} + \mathcal{B}(D^{-1}I_m - \mathcal{A})^{-1}\mathcal{C}, \end{aligned} \tag{2.19}$$

where in (2.18) and (2.19), $I_m$ denotes the $m \times m$ identity matrix.

If, as we did with block codes, we enlarge the set of possible input series $\boldsymbol{U}(D)$ in (2.17) from $F[[D]]^k$ to $F((D))^k$, it follows that the convolutional code associated with the encoding matrices $(\mathcal{A}, \mathcal{B}, \mathcal{C}, \mathcal{D})$ is the $F((D))$-rowspace of the matrix $G(D)$. For this reason, $G(D)$ is called a *generator matrix* for the code, and the quadruple $(\mathcal{A}, \mathcal{B}, \mathcal{C}, \mathcal{D})$ is called a *state-space realization* of the matrix $G(D)$.

By (2.19), the entries in $G(D)$ are rational functions in $D$. We recall that the set of all rational functions in $D$ forms a subfield, denoted by $F(D)$, of $F((D))$, so that according to (2.17), any $(n, k)$ convolutional code is a $k$-dimensional subspace of $F((D))^n$, with a basis consisting entirely of vectors from $F(D)^n$. Conversely, if $G(D)$ is any $k \times n$ matrix over $F(D)$, then $G(D)$ is row-equivalent (with row operations entirely over $F(D)$) to a *causal* matrix $G'(D)$, i.e., each entry in $G'(D)$ is causal. By a standard result in linear system theory [52, Chapter 13], $G'(D)$ can be realized by a quadruple $(\mathcal{A}, \mathcal{B}, \mathcal{C}, \mathcal{D})$, and so the $F((D))$-rowspace of $G(D)$ is the convolutional code generated by $(\mathcal{A}, \mathcal{B}, \mathcal{C}, \mathcal{D})$. Thus the set of $(n, k)$ convolutional codes is identical to the set of $k$-dimensional subspaces of $F((D))^n$ with bases from $F(D)^n$, and so we offer the following definition.

**2.3 Definition.** *An $(n, k)$ convolutional code is a $k$-dimensional subspace of $F((D))^n$, with a basis consisting entirely of vectors from $F(D)^n$.*

Since the basis vectors of a convolutional code are required to lie in the space $F(D)^n$, nothing essential is lost by considering the *rational subcode* of the code, i.e., those codewords whose components all lie in $F(D)$. For example, all *finite weight* codewords, which are the only ones that can occur in engineering practice, are in the rational subcode. Similarly, the only generator matrices that correspond to *realizable* encoders are those with causal rational entries. Thus we shall use the following definition of a convolutional code in the rest of this Chapter.

**2.4 Definition.** *An $(n, k)$ convolutional code is a $k$-dimensional subspace of $F(D)^n$.*

A final concept that we need is that of the *free distance* of a convolutional code. The free distance of a convolutional code is defined to be the minimum weight of any nonzero codeword, where the weight of a codeword $\boldsymbol{X}(D) = (X_1(D), \ldots, X_n(D))$ is defined to be the sum of the weights of its components $X_i(D)$. (Each component is a Laurent series, and the weight of a Laurent series has already been defined as the number of its nonzero coefficients.) If a linear block code is viewed as a special case of a convolutional code, then it is easy to see that the free distance of the code is the same as the minimum distance, as normally defined. Among codes of a fixed rate $k/n$, the free distance is arguably the single most important parameter determining the performance; the larger the free distance, the better the code, as a rule.

**2.5 Example:** We conclude this introductory section with a detailed look at one particular $n = 2$, $k = 1$ convolutional code over $F = GF(2)$. We begin with an encoder of degree $m = 2$. The input, output, and state at time $i$ will be denoted by $u(i)$, $(x_1(i), x_2(i))$, and

$(s_1(i), s_2(i))$, respectively. The encoding matrices $\mathcal{A}$, $\mathcal{B}$, $\mathcal{C}$, and $\mathcal{D}$ are given by

$$\begin{aligned} \mathcal{A} &= \begin{pmatrix} 0 & 1 \\ 0 & 0 \end{pmatrix} \\ \mathcal{B} &= \begin{pmatrix} 1 & 0 \end{pmatrix} \\ \mathcal{C} &= \begin{pmatrix} 1 & 0 \\ 1 & 1 \end{pmatrix} \\ \mathcal{D} &= \begin{pmatrix} 1 & 1 \end{pmatrix}. \end{aligned} \tag{2.20}$$

Then according to (2.18) and (2.19), we have

$$\begin{aligned} E(D) &= \mathcal{B}(D^{-1} I_m - \mathcal{A})^{-1} \\ &= \begin{pmatrix} 1 & 0 \end{pmatrix} \begin{pmatrix} D^{-1} & 1 \\ 0 & D^{-1} \end{pmatrix}^{-1} \\ &= \begin{pmatrix} D & D^2 \end{pmatrix}, \end{aligned}$$

and

$$\begin{aligned} G(D) &= \mathcal{D} + E(D)\mathcal{C} \\ &= \begin{pmatrix} 1 & 1 \end{pmatrix} + \begin{pmatrix} D & D^2 \end{pmatrix} \begin{pmatrix} 1 & 0 \\ 1 & 1 \end{pmatrix} \\ &= \begin{pmatrix} 1 + D + D^2 & 1 + D^2 \end{pmatrix}. \end{aligned}$$

Hence

$$G(D) = \begin{pmatrix} 1 + D + D^2 & 1 + D^2 \end{pmatrix} \tag{2.21}$$

is a generator matrix for the code. Our calculations show that $G(D)$ can be realized by the four matrices in (2.20). This realization can be made more concrete with the diagram of Figure 2.1.

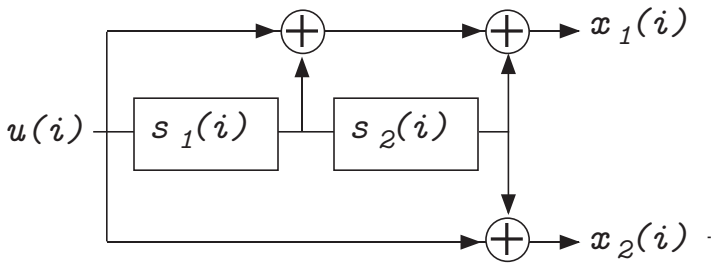


**Figure 2.1.** A physical realization for the generator matrix $\begin{pmatrix} 1 + D + D^2 & 1 + D^2 \end{pmatrix}$. This encoder has degree 2 and memory 2.

Figure 2.1 can be interpreted in a number of ways. To a hardware engineer, for example, it represents a blueprint for an actual physical device that can be built from commercially available parts: the "$\oplus$" symbols represent modulo two adders (`XOR` gates), and the rectangles represent delay flip-flops. Also, it shows how, despite the abstract approach we have chosen, that a convolutional encoder indeed converts a single ($k = 1$) information bit stream into two ($n = 2$) coded bit streams. For now, however, the figure is simply a shorthand notation for the equations

$$\begin{aligned} \boldsymbol{s}(i+1) &= (s_1(i+1), s_2(i+1)) = (u(i), s_1(i)) \\ \boldsymbol{x}(u) &= (x_1(i), x_2(i)) = (u(i) + s_1(i) + s_2(i), u(i) + s_2(i)), \end{aligned} \tag{2.22}$$

which, as is easily verified, correspond to the general equations (2.11) and (2.10) when the specific matrices in (2.20) are substituted for $\mathcal{A}$, $\mathcal{B}$, $\mathcal{C}$, and $\mathcal{D}$. In this particular case, it is possible to eliminate the state variables $s_1(i)$ and $s_2(i)$ from (2.22), to obtain an explicit representation of the encoder's output in terms of its input:

$$(x_1(i), x_2(i)) = (u(i) + u(i-1) + u(i-2), u(i) + u(i-2)). \tag{2.23}$$

From (2.23), it is seen that the output at time $i$ depends not only on the input at time $i$ but also the inputs from the earlier times $i-1$ and $i-2$. This fact is expressed formally by saying that the encoder has *memory* 2, since it needs to "remember" the inputs from two units in the past. In this case, the memory and the degree of the encoder are the same. In general, these quantities are different. Indeed, for some encoders of finite degree, the memory turns out to be *infinite*, as we will now see.

Since $G(D) = (\, 1 + D + D^2 \quad 1 + D^2 \,)$ is a generator matrix for the code, any matrix which is $F(D)$-row equivalent to $G(D)$ is also a generator matrix. For example, if we divide the entries in $G(D)$ by $1 + D + D^2$, we obtain

$$G'(D) = \left(\, 1 \quad \tfrac{1+D^2}{1+D+D^2} \,\right), \tag{2.24}$$

which, as we will learn in Section 5, is called a *systematic* generator matrix. In Figure 2.2 we see a physical realization for $G'(D)$.

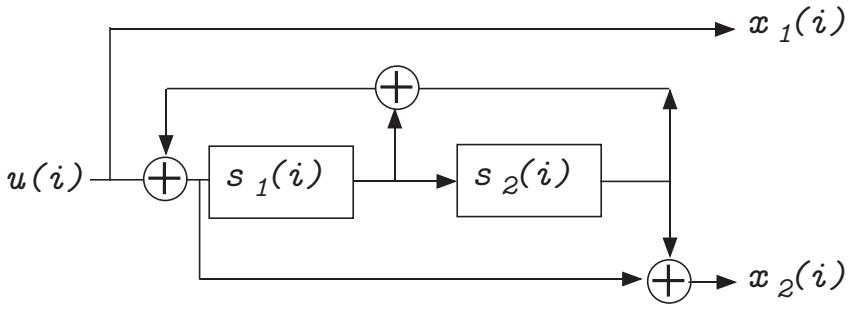


**Figure 2.2.** A physical realization for the generator matrix $G'(D) = (\, 1 \quad (1 + D^2)/(1 + D + D^2) \,)$. This encoder has degree 2 and infinite memory.

Figure 2.2 is shorthand for the equations

$$\begin{aligned}(s_1(i+1), s_2(i+1)) &= (s_1(i) + s_2(i) + u(i), s_1(i)) \\ (x_1(i), x_2(i)) &= (u(i), s_1(i) + u(i))^*.\end{aligned} \tag{2.25}$$

The equations in (2.25), in turn, are equivalent to the "$(\mathcal{A}, \mathcal{B}, \mathcal{C}, \mathcal{D})$" representation in (2.10) and (2.11) with

$$\begin{aligned}\mathcal{A} &= \begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix} \\ \mathcal{B} &= \begin{pmatrix} 1 & 0 \end{pmatrix} \\ \mathcal{C} &= \begin{pmatrix} 0 & 1 \\ 0 & 0 \end{pmatrix} \\ \mathcal{D} &= \begin{pmatrix} 1 & 1 \end{pmatrix}.\end{aligned} \tag{2.26}$$

Indeed, if we use the matrices in (2.26) in eqs. (2.18) and (2.19), we find that the corresponding $G(D)$ is as given in (2.24). In this case, if we attempt to eliminate the state variables $s_1(i)$ and $s_2(i)$ from (2.25) in order to express the output $x_1(i)$ and $x_2(i)$ in terms of the input $u(i)$, a strange thing happens. Instead of a closed form expression, we obtain

$$\begin{aligned}x_1(i) &= u(i) \\ x_2(i) &= u(i) + u(i-1) + u(i-3) + u(i-4) + \cdots.\end{aligned}$$

Thus, despite the fact that the encoder in Figure 2.2 has only two storage elements, the current output $x_2(i)$ depends on inputs from the arbitrarily distant past. We say that this encoder has *infinite memory*. It is for this reason that the *degree* of an encoder is a more useful concept than its *memory*.

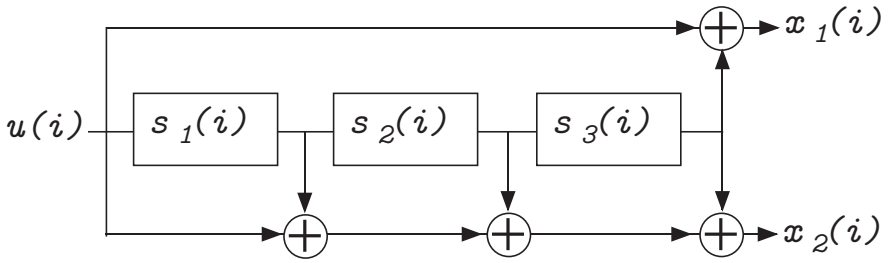


**Figure 2.3.** A physical realization for the "catastrophic" generator matrix $G''(D) = (\,1 + D^3 \quad 1 + D + D^2 + D^3\,)$. This encoder has degree 3 and memory 3.

* According to Figure 2.2, we have $x_2(i) = (u(i) + s_1(i) + s_2(i)) + s_2(i)$. But since the underlying field is $GF(2)$, the $s_2(i)$'s cancel, leaving $x_2(i) = s_1(i) + u(i)$, as given in (2.25).

Finally, if we multiply the entries of $G(D)$ as given in (2.21) by $1 + D$, we obtain

$$G''(D) = (\,1 + D^3 \quad 1 + D + D^2 + D^3\,), \tag{2.27}$$

which, as we will learn in Section 6, is a *catastrophic* generator matrix. A physical realization of this generator is shown in Figure 2.3. From Figure 2.3 we deduce that the "input-output" representation of this encoder is

$$\begin{aligned}(s_1(i+1), s_2(i+1), s_3(i+1)) &= (u(i), s_1(i), s_2(i)) \\ (x_1(i), x_2(i)) &= (u(i) + s_3(i), u(i) + s_1(i) + s_2(i) + s_3(i))\end{aligned} \tag{2.28}$$

which is equivalent to the "$(\mathcal{A}, \mathcal{B}, \mathcal{C}, \mathcal{D})$" representation with

$$\begin{aligned}\mathcal{A} &= \begin{pmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ 0 & 0 & 0 \end{pmatrix} \\ \mathcal{B} &= (\,1 \quad 0 \quad 0\,) \\ \mathcal{C} &= \begin{pmatrix} 0 & 1 \\ 0 & 1 \\ 1 & 1 \end{pmatrix} \\ \mathcal{D} &= (\,1 \quad 1\,).\end{aligned} \tag{2.29}$$

In this case we can eliminate the state variables $s_1(i)$, $s_2(i)$, and $s_3(i)$ from (2.28) to obtain

$$\begin{aligned}x_1(i) &= u(i) + u(i-3) \\ x_2(i) &= u(i) + u(i-1) + u(i-2) + u(i-3),\end{aligned}$$

so that this encoder has memory 3.

Despite their differences, it is important to remember that the three encoders in Figures 2.1, 2.2, and 2.3, all generate the same code. The free distance of this code turns out to be $d_{\text{free}} = 5$, which can be verified by an examination of cases, and the only codewords of weight 5 are the words of the form $(D^L(1 + D + D^2), D^L(1 + D^2))$, where $L$ is any integer, positive, negative, or zero. ▪

This concludes our introduction to the theory of convolutional codes. Here is a brief outline of the rest of the Chapter. In Section 3, we will study various properties of generator matrices for convolutional codes, and identify a preferred class of generator matrices, the *canonical* generator matrices. In Section 4, we will be concerned with the free distance of a convolutional code, derive a useful upper bound on $d_{\text{free}}$ as a function of the code's parameters, and give some short tables of good convolutional codes. In Section 5, we will study the complexity of physical encoders (like the one in Figure 2.1), and show that the canonical generators can be realized in an especially simple and efficient way. In Section 6 we will study a class of generator matrices that must be avoided at all costs, the *catastrophic* generator matrices. In Section 7 we will define and study the *dual code* of a convolutional code, and as an application define and derive the important properties of the *Wyner-Ash* codes. Finally, in Section 8, we will describe how the *puncturing* process can be used to create good new convolutional codes from old ones.

## 3. Polynomial Generator Matrices.

*Abstract: In Section 2, we defined an $(n, k)$ convolutional code $\boldsymbol{C}$ to be a $k$-dimensional subspace of $F(D)^n$, and a generator matrix of $\boldsymbol{C}$ to be a $k \times n$ matrix over $F(D)$, whose rowspace is $\boldsymbol{C}$. In this section we will discuss the various kinds of generator matrices that a convolutional code can have, and settle on the canonical polynomial generator matrices as the preferred kind. We will identify certain important parameters of convolutional codes, the degree and the Forney indices, which have no analog for block codes. Some of the background mathematics needed has been relegated to the appendices.*

If $\boldsymbol{C}$ is an $(n, k)$ convolutional code over the symbol field $F$, a *generator matrix* $G(D)$ for $\boldsymbol{C}$ is a $k \times n$ matrix over $F(D)$ whose rows form a basis for $\boldsymbol{C}$. If the entries of $G(D)$ are polynomials, then $G(D)$ is called a *polynomial generator matrix* (PGM) for $\boldsymbol{C}$. Any convolutional code has a polynomial generator matrix, since if $G$ is an arbitrary generator matrix for $\boldsymbol{C}$, the matrix obtained from $G$ by multiplying each row by the least common multiple of the denominators of the entries in that row is a PGM for $\boldsymbol{C}$.

Let $G(D) = (g_{ij}(D))$ be a $k \times n$ PGM for $\boldsymbol{C}$. We denote the $i$th row of $G$, i.e., the $n$-vector $(g_{i1}(D), \ldots, g_{in}(D))$, by $\boldsymbol{g}_i(D)$, and we define the *degree* of $\boldsymbol{g}_i(D)$ as the maximum degree of its components. In a similar way we define the degree of any $n$-tuple of polynomials as the maximum degree of any component. We now define the *internal degree* and *external degree* of $G(D)$ as follows.

$$\begin{aligned} \operatorname{intdeg} G(D) &= \text{maximum degree of } G(D)\text{'s } k \times k \text{ minors} \\ \operatorname{extdeg} G(D) &= \text{sum of the row degrees of } G(D). \end{aligned}$$

The following two definitions will be essential in our discussion of convolutional codes.

**3.1 Definition.** *A $k \times n$ polynomial matrix $G(D)$ is called basic if, among all polynomial matrices of the form $T(D)G(D)$, where $T(D)$ is a nonsingular $k \times k$ matrix over $F(D)$, it has the minimum possible internal degree.*

**3.2 Definition.** *A $k \times n$ polynomial matrix $G(D)$ is called reduced if, among all matrices of the form $T(D)G(D)$, where $T(D)$ is unimodular,* $G(D)$ has the minimum possible external degree. Since any unimodular matrix is a product of elementary matrices, an equivalent definition is that a matrix is reduced if its external degree cannot be reduced by a sequence of elementary row operations.*

Before continuing along our main line, we pause to present a simple theorem that provides several useful facts about the internal and external degrees of a polynomial matrix.

**3.3 Theorem.** *Let $G(D)$ be a $k \times n$ polynomial matrix.*

---

* *A unimodular matrix is a square polynomial matrix whose determinant is a nonzero scalar, i.e., an element of $F$.*

(a) If $T(D)$ is any nonsingular $k \times k$ polynomial matrix, then

$$\operatorname{intdeg} T(D)G(D) = \operatorname{intdeg} G(D) + \deg \det T(D).$$

In particular $\operatorname{intdeg} T(D)G(D) \geq \operatorname{intdeg} G(D)$, with equality if and only if $T(D)$ is unimodular.

(b) $\operatorname{intdeg} G(D) \leq \operatorname{extdeg} G(D)$.

**Proof:** (a) The $k \times k$ submatrices of $T(D)G(D)$ are just the $k \times k$ submatrices of $G(D)$, each multiplied by $T(D)$. Thus the $k \times k$ minors of $T(D)$ are just the $k \times k$ minors of $G(D)$, each multiplied by $\det T(D)$. The result now follows.

(b) Denote the degree of the $i$th row of $G(D)$ by $e_i$. In the expansion of any $k \times k$ minor of $G(D)$, each term is the product of $k$ entries of $G(D)$, one from each row (and column). Since each entry from the $i$th row has degree $\leq e_i$, it follows that the degree of any $k \times k$ minor is at most $e_1 + \cdots + e_k = \operatorname{extdeg} G(D)$. ▪

Basic and reduced polynomial matrices enjoy many useful and surprising properties. In Appendix A, we will prove the following two theorems, which delineate these properties. We will refer to these two theorems constantly in the rest of this chapter.

**Theorem A.1.** *A $k \times n$ polynomial matrix $G(D)$ is basic (see Definition 3.1) if and only if any one of the following six conditions is satisfied.*

(1) The invariant factors of $G(D)$ are all 1.

(2) The gcd of the $k \times k$ minors of $G(D)$ is 1.

(3) $G(\alpha)$ has rank $k$ for any $\alpha$ in the algebraic closure of $F$.

(4) $G(D)$ has a right $F[D]$ inverse, i.e., there exists a $n \times k$ polynomial matrix $H(D)$ such that $G(D)H(D) = I_k$.

(5) If $\boldsymbol{x}(D) = \boldsymbol{u}(D)G(D)$, and if $\boldsymbol{x}(D) \in F[D]^n$, then $\boldsymbol{u}(D) \in F[D]^k$. ("Polynomial output implies polynomial input.")

(6) $G(D)$ is a submatrix of a unimodular matrix, i.e., there exists a $(n-k) \times n$ matrix $L(D)$ such that the determinant of the $n \times n$ matrix $\begin{pmatrix} G(D) \\ L(D) \end{pmatrix}$ is a nonzero element of $F$.

**Theorem A.2.** *A $k \times n$ polynomial matrix $G(D)$ is reduced (see Definition 3.2) if and only if one of the following three conditions is satisfied.*

(1) If we define the the "indicator matrix for the highest-degree terms in each row" $\overline{G}$ by

$$\overline{G}_{ij} = \operatorname*{coeff}_{D^{e_i}} g_{ij}(D),$$

where $e_i$ is the degree of $G(D)$'s $i$th row, then $\overline{G}$ has rank $k$.

(2) $\operatorname{extdeg} G(D) = \operatorname{intdeg} G(D)$.

(3) The "predictable degree property": For any $k$-dimensional polynomial vector, i.e., any $\boldsymbol{u}(D) = (u_1(D), \ldots, u_k(D)) \in F[D]^k$

$$\deg(\boldsymbol{u}(D)G(D)) = \max_{1 \le i \le k} (\deg u_i(D) + \deg \boldsymbol{g}_i(D)).$$

**3.4 Example:** Here are eight generator matrices for a $(4, 2)$ convolutional code over $GF(2)$. Of these eight, six, $G_2$ through $G_7$, are PGM's.

$$G_1 = \begin{pmatrix} \frac{1}{1+D+D^2} & 1 & \frac{1+D^2}{1+D+D^2} & \frac{1+D}{1+D+D^2} \\ 1 & \frac{1+D+D^2}{D} & D & \frac{1}{D} \end{pmatrix}$$

$$G_2 = \begin{pmatrix} 1 & 1+D+D^2 & 1+D^2 & 1+D \\ D & 1+D+D^2 & D^2 & 1 \end{pmatrix}$$

$$G_3 = \begin{pmatrix} 1 & 1+D+D^2 & 1+D^2 & 1+D \\ 0 & 1+D & D & 1 \end{pmatrix}$$

$$G_4 = \begin{pmatrix} 1 & D & 1+D & 0 \\ 0 & 1+D & D & 1 \end{pmatrix}$$

$$G_5 = \begin{pmatrix} 1+D & 0 & 1 & D \\ D & 1+D+D^2 & D^2 & 1 \end{pmatrix}$$

$$G_6 = \begin{pmatrix} 1 & 1 & 1 & 1 \\ 0 & 1+D & D & 1 \end{pmatrix}$$

$$G_7 = \begin{pmatrix} 1+D & 0 & 1 & D \\ 1 & D & 1+D & 0 \end{pmatrix}$$

$$G_8 = \begin{pmatrix} 1 & 0 & \frac{1}{1+D} & \frac{D}{1+D} \\ 0 & 1 & \frac{D}{1+D} & \frac{1}{1+D} \end{pmatrix}$$

Here is a table listing the properties of the generator matrices $G_1, \ldots, G_8$. (Note that only the polynomial generator matrices, viz., $G_2$–$G_7$, have external or internal degrees, or can be basic or reduced.)

**Table 3.1. Properties of $G_1, \ldots, G_8$.**

| | Basic? | Reduced? | intdeg | extdeg |
|---|---|---|---|---|
| $G_1$ | – | – | – | – |
| $G_2$ | No | No | 3 | 4 |
| $G_3$ | Yes | No | 1 | 3 |
| $G_4$ | Yes | No | 1 | 2 |
| $G_5$ | No | Yes | 3 | 3 |
| $G_6$ | Yes | Yes | 1 | 1 |
| $G_7$ | No | Yes | 2 | 2 |
| $G_8$ | – | – | – | – |

These properties are easily verified by referring to Theorems A.1 and A.2. For example, $G_3$ is *basic* because the gcd of its $2 \times 2$ minors is 1 (Theorem A.1, part (2)), and it is *not reduced* because its internal and external degrees are unequal (Theorem A.2, part (2)). ▪

It follows from Definition 3.1 that among all PGM's for a given convolutional code, those for which the *internal* degree is as small as possible are exactly the *basic* PGM's. It turns out, however, that the set of PGM's for which the *external* degree is as small as possible form a much more interesting class, the class of *canonical* PGM's.

**3.5 Definition.** *Among all PGM's for a given convolutional code $\boldsymbol{C}$, those for which the external degree is as small as possible are called canonical PGM's. This minimal external degree is called the degree of the code $\boldsymbol{C}$, and denoted* $\deg \boldsymbol{C}$.

As we shall see, canonical generator matrices have many remarkable properties. The key to these properties is the following theorem.

**3.6 Theorem.** *A PGM $G(D)$ for the convolutional code $\boldsymbol{C}$ is canonical if and only if it is both basic and reduced.*

**Proof:** First we will show that a canonical PGM must be both basic and reduced. Then we will show that a PGM that is both basic and reduced must be canonical.

To prove the first assertion, denote by $m_0$ the common internal degree of all the basic PGM's for $\boldsymbol{C}$, and among all the basic PGM's choose one, say $G_0(D)$, for which the *external* degree is as small as possible. Then $G_0$ must be reduced, since if $T(D)$ is unimodular, $\operatorname{intdeg} T(D)G_0(D) = \operatorname{intdeg} G_0(D) = m_0$ by Theorem 3.3, and so by the definition of $G_0$, $\operatorname{extdeg} T(D)G_0(D) \geq \operatorname{extdeg} G_0(D)$. Now let $G(D)$ be any canonical PGM for $\boldsymbol{C}$. Then we must have

$$\operatorname{intdeg} G_0 \leq \operatorname{intdeg} G \leq \operatorname{extdeg} G \leq \operatorname{extdeg} G_0. \tag{3.1}$$

(The first inequality in (3.1) follows from the fact that $G_0$ is chosen to have minimum possible internal degree. The second inequality follows from Theorem 3.3, part (b). The

third inequality is because $G$, as a canonical PGM, has minimum possible eternal degree.) But since $G_0(D)$ is reduced, we have by Theorem A.2, part (2) that $\operatorname{intdeg} G_0 = \operatorname{extdeg} G_0$, so that equality holds throughout (3.1). Thus $\operatorname{intdeg} G = \operatorname{intdeg} G_0 = m_0$, so $G$ is basic; and $\operatorname{intdeg} G = \operatorname{extdeg} G$, so that $G$ is reduced, by Theorem A.2, part (2).

Conversely, suppose that $G(D)$ is basic and reduced, and let $G_0(D)$ be any other PGM for $\boldsymbol{C}$. Then by Theorem 3.3(b), $\operatorname{extdeg} G_0(D) \geq \operatorname{intdeg} G_0(D)$. Since $G(D)$ is basic, we have $\operatorname{intdeg} G_0(D) \geq \operatorname{intdeg} G(D)$; since $G(D)$ is reduced, we have by Theorem 3.3 that $\operatorname{intdeg} G(D) = \operatorname{extdeg} G(D)$. Combining these inequalities, we have that $\operatorname{extdeg} G_0(D) \geq \operatorname{extdeg} G(D)$, which proves that $G(D)$ is canonical. ▪

In the proof of Theorem 3.6, we saw that $m_0$, the common internal degree for all basic PGM's for $\boldsymbol{C}$, was equal to $\deg \boldsymbol{C}$, i.e., the minimum possible external degree. Thus we have the following two corollaries to Theorem 3.6.

**3.7 Corollary.** *The minimal internal degree of any PGM for a given convolutional code $\boldsymbol{C}$ is equal to the degree of $\boldsymbol{C}$.*

**3.8 Corollary.** *If $G$ is any basic generator matrix for $\boldsymbol{C}$, then* $\operatorname{intdeg} G = \deg \boldsymbol{C}$.

In general, a convolutional code can have several different canonical polynomial generator matrices. The following theorem shows, however, that all canonical polynomial generator matrices share many common features.

**3.9 Theorem.** *If $e_1 \leq e_2 \leq \cdots \leq e_k$ are the row degrees of a canonical generator matrix $G(D)$ for a convolutional code $\boldsymbol{C}$, and if $f_1 \leq f_2 \leq \cdots \leq f_k$ are the row degrees of any other polynomial generator matrix, say $G'(D)$, for $\boldsymbol{C}$, then $e_i \leq f_i$, for $i = 1, \ldots, k$.*

**Proof:** If the statement is false, there exists an index $j$ such that $e_1 \leq f_1, \ldots, e_j \leq f_j$, but $e_{j+1} > f_{j+1}$. It then follows from the "canonicality" of $G(D)$ (use the properties in Theorem A.1, part (5), and Theorem A.2, part (3)) that the first $j+1$ rows of $G'$ must be polynomial linear combinations of the first $j$ rows of $G$, which contradicts the fact that the rows of $G'$ are linearly independent. ▪

**3.10 Theorem.** *The set of row degrees is the same for all canonical PGM's for a given code.*

**Proof:** This result follows immediately from Theorem 3.9. ▪

The row degrees referred to in Theorems 3.9 and 3.10, say $(e_1, e_2, \ldots, e_k)$, are called the *Forney indices* of the code. The sum $e_1 + \cdots + e_k$ of the Forney indices, i.e., the minimum possible external degree of any PGM for $\boldsymbol{C}$, is the degree of the code.* The maximum of the Forney indices is called the *memory* of the code. From now on, we

* There are a myriad of alternative terms for this important quantity, including *constraint length*, *overall constraint length*, *state complexity*, and *memory*.

reserve the letter $m$ to denote the degree of a given convolutional code, and refer to an $(n, k)$ code with degree $m$ as an $(n, k, m)$ code. We reserve the letter $d$ to denote the free distance of the code, and an $(n, k, m)$ code with free distance $d$ will be called a $(n, k, m, d)$ code. An $(n, k, m)$ code is called *optimal* (more precisely, *distance optimal*) if it has the maximum possible free distance among all codes with the same value of $n$, $k$, and $m$.

**3.11 Example:** Continuing the study of the $(4, 2)$ code from Example 3.4, we see that of the eight given generator matrices, only $G_6$ is canonical (it satisfies condition (2) of Theorem A.1 and condition (2) of Theorem A.2, so it is both basic and reduced), so that the *degree* of the code $\boldsymbol{C}$ is 1, and the *Forney indices* are $(0, 1)$. The code is thus a $(4, 2, 1)$ code. In Example 4.6, below, we will see that $d = 4$ and that it is in fact an *optimal* $(4, 2, 1)$ code. ▪

## 4. The Hilbert Series and Free Distance Bounds.

*Abstract: In this section we will study the polynomial subcodes of a convolutional code, and will see that the dimensions of the polynomial subcodes can be computed from the code's Forney indices. We will then use the fact that each polynomial subcode is a block code to obtain a useful family of upper bounds on the free distance of a convolutional code in terms of the maximum possible minimum distance for the polynomial subcodes. We also include tables of canonical generator matrices for some "small" convolutional codes, together with their free distances.*

If $\boldsymbol{C}$ is a fixed $(n,k)$ convolutional code, a *polynomial codeword* of $\boldsymbol{C}$ is a codeword all of whose components are polynomials. Recalling that the degree of a polynomial vector is defined to be the maximum degree of any component, for any integer $L \geq 0$ we define $\boldsymbol{C}_L$ to be the set of *polynomial codewords of degree* $\leq L$. $\boldsymbol{C}_L$ is a vector space over $F$. Indeed, it is a subspace of the set of all possible $n$-dimensional polynomial vectors of degree $\leq L$ over $F$. We denote the $F$-dimension of $\boldsymbol{C}_L$ by $\delta_L$. The following theorem shows that the $\delta_L$'s can be computed from the Forney indices.

**4.1 Theorem.** *If $\boldsymbol{C}$ is an $(n,k)$ convolutional code with Forney indices $(e_1,\ldots,e_k)$ and polynomial subcode dimensions $\delta_L$, then*

$$\sum_{L\geq 0} \delta_L t^L = \frac{t^{e_1}+\cdots+t^{e_k}}{(1-t)^2}. \tag{4.1}$$

*Note: The power series appearing in Theorem 4.1 is called the Hilbert series for the code. It was first introduced in [44]. The first general results on Hilbert series (needless to say, having nothing to do with convolutional codes), were obtained by Hilbert in 1890 [23]. For the "modern" theory of Hilbert series, see [50].*

**Proof:** Let $G(D)$ be a canonical PGM for $\boldsymbol{C}$, whose row degrees are the ordered Forney indices, say $e_1 \leq \cdots \leq e_k$, and let $\boldsymbol{g}_1,\ldots,\boldsymbol{g}_k$ be the rows of $G(D)$. Let $\boldsymbol{x}(D)$ be any polynomial codeword of degree $\leq L$. Then it follows from Theorem A.1, part 5, that $\boldsymbol{x}(D) = \boldsymbol{u}(D)G(D)$, where $\boldsymbol{u}(D) = (u_1(D),\ldots,u_k(D))$ is a $k$-vector of polynomials, and it follows from the predictable degree property (Theorem A.2, part 3), that $\deg u_i + e_i \leq L$, for $i = 1,\ldots,k$. Thus a basis for the $F$-space $\boldsymbol{C}_L$ is the set $\{D^j \boldsymbol{g}_i(D) : j + e_i \leq L\}$. Hence

$$\begin{aligned}
\sum_{L\geq 0} \delta_L t^L &= \sum_{i=1}^{k} \sum_{j\geq 0} \left(t^{e_i+j} + t^{e_i+j+1} + \cdots\right) \\
&= \sum_{i=1}^{k} \sum_{j\geq 0} \frac{t^{e_i+j}}{1-t} \\
&= \sum_{i=1}^{k} \frac{t^{e_i}}{(1-t)^2}. \quad \blacksquare
\end{aligned}$$

**4.2 Corollary.** *(Forney [13]). We have the following explicit formula for $\delta_L$:*

$$\delta_L = \sum_{i=1}^{k} \max(L + 1 - e_i, 0). \tag{4.2}$$

**Proof:** By elementary calculus, we have that $(1-t)^{-2} = \sum_{j\geq 0}(j+1)t^j$. Applying this fact to the formula (4.1), we obtain

$$\begin{aligned}\sum_{L\geq 0} \delta_L t^L &= \sum_{i=1}^{k} \frac{t^{e_i}}{(1-t)^2} \\ &= \sum_{i=1}^{k} \sum_{j\geq 0} (j+1) t^{e_i+j} \\ &= \sum_{i=1}^{k} \sum_{j\geq e_i} (j+1-e_i) t^j.\end{aligned}$$

Thus the coefficient of $t^L$ in the Hilbert series $\sum_{L\geq 0} \delta_L t^L$ is $\sum_{i=1}^{k} \max(L+1-e_i, 0)$, which is what we wanted to prove. ∎

**4.3 Corollary.** *Let $\boldsymbol{C}$ be an $(n, k, m)$ convolutional code. Then for all $L \geq 0$, we have*

$$\delta_L \geq \max((L+1)k - m, 0). \tag{4.3}$$

*Furthermore, we have equality for all $L \geq 0$ in (4.3) if and only if the Forney indices assume only the two values $\lfloor m/k \rfloor$ and $\lceil m/k \rceil$. A code for which this is true will be called a compact code.*

**Proof:** Since $\max(x, 0) \geq x$, eq. (4.2) implies that $\delta_L \geq \sum_{i=1}^{k}(L+1-e_i) = (L+1)k - \sum_{i=1}^{k} e_i = (L+1)k - m$. Since $\delta_L \geq 0$, too, it follows that $\delta_L \geq \max((L+1)k - m, 0)$ for all $L \geq 0$. Assuming that the Forney indices are ordered so that $e_1 \leq \cdots \leq e_k$, it follows from (4.2) that $\delta_L = (L+1)k - m = \max((L+1)k - m, 0)$ if $L + 1 \geq e_k$, and $\delta_L = 0 = \max((L+1)k - m, 0)$ if $L + 1 \leq e_1$. If $e_k - e_1 \leq 1$, one of these alternatives must hold for all $L$. On the other hand, if $e_k - e_1 \geq 2$, then there is at least one value of $L$ for which $e_1 < L + 1 < e_k$, in which case $0 < \delta_L < (L+1)k - m$, so that $\delta_L \neq \max((L+1)k - m, 0)$. ∎

It is easy to show that in a compact code, there are exactly $(m \bmod k)$ Forney indices equal to $\lceil m/k \rceil$, and $k - (m \bmod k)$ Forney indices equal to $\lfloor m/k \rfloor$. Thus for example if $k = 4$ and $m = 13$, a compact code will have Forney indices $(3, 3, 3, 4)$.

Since the subcode $\boldsymbol{C}_L$ forms an $[n(L+1), \delta_L]$ linear block code over $F$, we know that the *free distance* of $\boldsymbol{C}$, i.e., the minimum nonzero distance between codewords of $\boldsymbol{C}$, cannot exceed the minimum distance of $\boldsymbol{C}_L$, for $L = 0, 1, \ldots$. Hence we have Theorem 4.4.

**4.4 Theorem.** *If $\boldsymbol{C}$ is an $(n,k)$ convolutional code over the finite field $GF(q)$, with Forney indices $(e_1,\ldots,e_k)$, then*

$$d_{\text{free}}(C) \le \min_{L\ge 0} \Delta_q(n(L+1),\delta_L),$$

*where $\Delta_q(n,k)$ denotes the maximum possible minimum distance of an $[n,k]$ linear block code over $GF(q)$, and the parameters $\delta_L$ are determined from the Forney indices by (4.2).*

We note from Corollary 4.3 that $\delta_L$ is minimized for all $L$, and so $\Delta_q(n(L+1),\delta_L)$ is maximized for all $L$, for a compact code. This suggests, but does not prove, that among all $(n,k,m)$ codes, the compact codes will have the largest free distances.* In any case, for applications we will need a bound on $d_{\text{free}}$ that applies to all $(n,k,m)$ codes, regardless of their Forney indices. Thus we offer the following Corollary to Theorem 4.4. It is good to bear in mind, however, that it may be possible to improve the bound if the code is noncompact.

**4.5 Corollary.** *If $\boldsymbol{C}$ is an $(n,k,m)$ code, then*

$$d_{\text{free}}(C) \le \min_{L\ge 0} \Delta_q(n(L+1),k(L+1)-m).$$

**Proof:** Combine Theorem 4.4 with Corollary 4.3. ▪

The upper bound on $d_{\text{free}}$ of Theorem 4.4 is attained for many, but not all, values of $n$, $k$, and $m$. The following examples, and the tables at the end of the section, illustrate this. (All examples are over the field $GF(2)$.)

**4.6 Example:** Continuing the study of the $(4,2,1)$ code in Examples 3.4 and 3.11, we see that since the Forney indices are $(0,1)$, by Theorem 4.1 the "Hilbert Series" for the code is

$$\frac{1+t}{(1-t)^2} = 1 + 3t + 5t^2 + \cdots + (2L+1)t^L + \cdots.$$

Thus the dimension of the $L$th subcode $\boldsymbol{C}_L$ is $2L+1$. It then follows from Theorem 4.4 that the free distance of the code satisfies

$$d_{\text{free}}(C) \le \min_{L\ge 0} \Delta_2(4(L+1),2L+1).$$

In particular, for $L=0$ the above bound gives $d_{\text{free}}(C) \le \Delta_2(4,1) = 4$. But in fact $d_{\text{free}} = 4$ for this code (use the generator matrix $G_4$ to check this fact), so this particular code has the largest possible free distance for a $(4,2,1)$ convolutional code, i.e., it is an *optimal* $(4,2,1)$ code. ▪

---

* Indeed, in Section 6, we will see that all extended Wyner-Ash codes are noncompact, despite being optimal. For example, the $(4,3,3,4)$ extended Wyner-Ash code has Forney indices $(0,1,2)$—see Table 3.2.

**4.7 Example:** Consider $(n, k, m) = (2, 1, 2)$ codes. Since $k = 1$, there is only one Forney index, and so any $(2, 1, 2)$ code is compact. By Corollary 4.2, we have $\delta_0 = 0$, and, for $L \geq 1$, $\delta_L = L - 1$, so that by Theorem 4.4, we have (here and hereafter we use the tables of Brouwer and Verhoeff [6], or those in "Chapter Brouwer" of the values of $\Delta_q(n, k)$ when $q = 2$)

$$\begin{aligned} d_{\text{free}} &\leq \min(\Delta(2,0), \Delta(4,0), \Delta(6,1), \Delta(8,2), \Delta(10,3) \ldots) \\ &= \min(\infty, \infty, 6, 5, 5, \ldots) \\ &= 5. \end{aligned}$$

In fact, there is a well-known $(2, 1, 2)$ code with $d_{\text{free}} = 5$, whose (unique) canonical generator matrix is

$$G(D) = (\, 1 + D + D^2 \quad 1 + D^2 \,). \tag{4.4}$$

(See CiteExamplePrologue.1 or [41, Chapter 9].) It follows then that the $(2, 1, 2)$ code defined by (4.4) is optimal. ▪

**4.8 Example:** Consider binary $(4, 3, 2)$ codes. The Forney indices of such codes must be either $(0, 1, 1)$ (compact) or $(0, 0, 2)$ (noncompact). In the first case we have by Corollary 4.2 $\delta_0 = 1$, $\delta_1 = 4$, $\delta_2 = 7$, etc., and so by Theorem 4.4, we have

$$\begin{aligned} d_{\text{free}} &\leq \min(\Delta(4,1), \Delta(8,4), \Delta(12,7), \Delta(16,10), \ldots) \\ &= \min(4, 4, 4, 4, \ldots) \\ &= 4. \end{aligned}$$

However, it turns out that there is no $(4, 3, 2)$ code with $d_{\text{free}} = 4$ (Theorem 7.10, below). The largest possible $d_{\text{free}}$ turns out to be $d_{\text{free}} = 3$, which is achieved by the second-order Wyner-Ash code (Theorem 7.9). A canonical PGM for this code is

$$\begin{pmatrix} 1 & 1 & 1 & 1 \\ 1+D & 1 & 0 & 0 \\ 0 & 1+D & 1 & 0 \end{pmatrix}.$$

On the other hand, if the Forney indices are $(0, 0, 2)$, then Corollary 4.2 tells us that $\delta_0 = 2$, $\delta_1 = 4$, $\delta_2 = 7$, etc., and so by Theorem 4.4, we have

$$\begin{aligned} d_{\text{free}} &\leq \min(\Delta(4,2), \Delta(8,4), \Delta(12,7), \Delta(16,10), \ldots) \\ &= \min(2, 4, 4, 4, \ldots) \\ &= 2. \end{aligned}$$

And indeed there is a $(4, 3, 2)$ code with Forney indices $(0, 0, 2)$ and $d_{\text{free}} = 2$. A canonical PGM for one such code is

$$\begin{pmatrix} 1 & 0 & 0 & 1 \\ 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 1+D^2 \end{pmatrix}.$$

We therefore see that among $(4,3,2)$ codes, only the compact ones can be optimal. ▪

We conclude this section with five tables. In the tables, $\Delta(n,k,m)$ denotes the largest possible free distance for a $(n,k,m)$ code convolutional code over $GF(2)$, and $\Delta^*(n,k,m)$ denotes the upper bound on $\Delta(n,k,m)$ obtained from Corollary 4.5, viz.,

$$\Delta^*(n,k,m) \stackrel{\text{def}}{=} \min_{L \geq 0} \Delta_2(n(L+1), k(L+1)-m). \tag{4.5}$$

We have also given an explicit canonical generator matrix for an optimal $(n,k,m)$ code in each case, i.e., an $(n,k,m)$ code with $d_{\text{free}} = \Delta(n,k,m)$. The values of $\Delta(n,k,m)$, and the corresponding generator matrices, are taken from tables in the literature, primarily [45], [32] and [47]. The values of $\Delta^*(n,k,m)$ were computed by hand using the formula in (4.5) and the tables of $\Delta(n,k)$ in [6], or "Chapter Brouwer." In the tables, boldface entries in the $\Delta^*$ column indicate instances where the bound of (4.5) does not give the exact value of $\Delta(n,k,m)$. Also, in some cases, the value of $\Delta(n,k,m)$ is the same as the previous value $\Delta(n,k,m-1)$. When this happens, rather than list an $(n,k,m)$ code with the maximal free distance, we have instead written "see previous entry."

In the tables every polynomial is coded in *octal*. For example, to code the polynomial $1 + D + D^3 + D^6$, it is first written in the form $D^6 + D^3 + D + 1$, then converted to the binary string 1001011, then parsed from right to left into groups of three, 001 001 011, which is then read as the octal string 113. In a few cases, we find that $\Delta(n,k,m)$ turns out to be the same as the "previous value," i.e., $\Delta(n,k,m-1)$. When this happens, we write "see previous entry," rather than list an inefficient $(n,k,m)$ code whose free distance is not better than the best $(n,k,m-1)$ code.

Table 4.1. Some Optimal $(2,1,m)$ Codes.

| $m$ | $\Delta(2,1,m)$ | $\Delta^*(2,1,m)$ | canonical $G(D)$ |
|---|---|---|---|
| 0 | 2 | 2 | ( 1 1 ) |
| 1 | 3 | **4** | ( 1 3 ) |
| 2 | 5 | 5 | ( 5 7 ) |
| 3 | 6 | 6 | ( 13 17 ) |
| 4 | 7 | **8** | ( 23 35 ) |
| 5 | 8 | 8 | ( 53 75 ) |
| 6 | 10 | 10 | ( 133 171 )$^*$ |
| 7 | 10 | **11** | (see previous entry) |
| 8 | 12 | 12 | ( 561 753 ) |
| 9 | 12 | **13** | (see previous entry) |
| 10 | 14 | 14 | ( 2335 3661 ) |

Table 4.2. Some Optimal $(3, 1, m)$ Codes

| $m$ | $\Delta(3,1,m)$ | $\Delta^*(3,1,m)$ | canonical $G(D)$ |
|---|---|---|---|
| 0 | 3 | 3 | $(1 \quad 1 \quad 1)$ |
| 1 | 5 | **6** | $(1 \quad 3 \quad 3)$ |
| 2 | 8 | 8 | $(5 \quad 7 \quad 7)$ |
| 3 | 10 | 10 | $(13 \quad 15 \quad 17)$ |
| 4 | 12 | 12 | $(25 \quad 33 \quad 37)$ |
| 5 | 13 | 13 | $(47 \quad 53 \quad 75)$ |
| 6 | 15 | 15 | $(133 \quad 145 \quad 175)$ |
| 7 | 16 | 16 | $(225 \quad 331 \quad 367)$ |
| 8 | 18 | 18 | $(557 \quad 663 \quad 711)$ |
| 9 | 20 | 20 | $(1117 \quad 1365 \quad 1633)$ |
| 10 | 22 | 22 | $(2353 \quad 2671 \quad 3175)$ |

* This inconspicuous entry occupies a special place in the history of science. Discovered by Odenwalder in the late 1960's [45], the $(2, 1, 6, 10)$ code was adopted by NASA as a standard in the late 1970's, and was used to encode the historic photos of Jupiter, Saturn, Uranus, and Neptune that the *Voyager* spacecraft transmitted to earth in the 1980's. It has also been used in many other space communication applications. See "Chapter Wicker" for details.

Table 4.3. Some Optimal $(3, 2, m)$ Codes

| $m$ | $\Delta(3,2,m)$ | $\Delta^*(3,2,m)$ | canonical $G(D)$ | Forney indices |
|---|---|---|---|---|
| 0 | 2 | 2 | $\begin{pmatrix} 1 & 0 & 1 \\ 0 & 1 & 1 \end{pmatrix}$ | $(0,0)$ |
| 1 | 2 | 3 | (see previous entry) | |
| 2 | 3 | **4** | $\begin{pmatrix} 3 & 2 & 3 \\ 2 & 1 & 1 \end{pmatrix}$ | $(1,1)$ |
| 3 | 4 | 4 | $\begin{pmatrix} 1 & 2 & 3 \\ 4 & 1 & 7 \end{pmatrix}$ | $(1,2)$ |
| 4 | 5 | **6** | $\begin{pmatrix} 7 & 4 & 1 \\ 2 & 5 & 7 \end{pmatrix}$ | $(2,2)$ |
| 5 | 6 | 6 | $\begin{pmatrix} 3 & 6 & 7 \\ 14 & 1 & 17 \end{pmatrix}$ | $(2,3)$ |
| 6 | 7 | **8** | $\begin{pmatrix} 13 & 6 & 13 \\ 6 & 13 & 17 \end{pmatrix}$ | $(3,3)$ |
| 7 | 8 | 8 | $\begin{pmatrix} 3 & 6 & 15 \\ 34 & 31 & 17 \end{pmatrix}$ | $(3,4)$ |
| 8 | 8 | 8 | (see previous entry) | |
| 9 | 9 | **10** | $\begin{pmatrix} 25 & 30 & 17 \\ 50 & 7 & 65 \end{pmatrix}$ | $(4,5)$ |
| 10 | 10 | 10 | $\begin{pmatrix} 63 & 54 & 31 \\ 26 & 53 & 43 \end{pmatrix}$ | $(5,5)$ |

Table 4.4. Some Optimal $(4, 1, m)$ Codes

| $m$ | $\Delta(4,1,m)$ | $\Delta^*(4,1,m)$ | canonical $G(D)$ |
|---|---|---|---|
| 0 | 4 | 4 | $(1 \quad 1 \quad 1 \quad 1)$ |
| 1 | 7 | **8** | $(1 \quad 3 \quad 3 \quad 3)$ |
| 2 | 10 | 10 | $(5 \quad 7 \quad 7 \quad 7)$ |
| 3 | 13 | 13 | $(13 \quad 15 \quad 15 \quad 17)$ |
| 4 | 16 | 16 | $(25 \quad 27 \quad 33 \quad 37)$ |
| 5 | 18 | 18 | $(53 \quad 67 \quad 71 \quad 75)$ |
| 6 | 20 | 20 | $(135 \quad 135 \quad 147 \quad 163)$ |
| 7 | 22 | 22 | $(235 \quad 275 \quad 313 \quad 357)$ |
| 8 | 24 | 24 | $(463 \quad 535 \quad 733 \quad 745)$ |
| 9 | 27 | 27 | $(1117 \quad 1365 \quad 1633 \quad 1653)$ |
| 10 | 29 | 29 | $(2387 \quad 2353 \quad 2671 \quad 3175)$ |

Table 4.5. Some Optimal $(4, 3, m)$ Codes

| $m$ | $\Delta(4,3,m)$ | $\Delta^*(4,3,m)$ | canonical $G(D)$ | Forney indices |
|---|---|---|---|---|
| 0 | 2 | 2 | $\begin{pmatrix} 1 & 0 & 0 & 1 \\ 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 1 \end{pmatrix}$ | $(0,0,0)$ |
| 1 | 2 | 2 | (see previous entry) | |
| 2 | 3 | **4** | $\begin{pmatrix} 1 & 1 & 1 & 1 \\ 3 & 1 & 0 & 0 \\ 0 & 3 & 1 & 0 \end{pmatrix}$ | $(0,1,1)^*$ |
| 3 | 4 | 4 | $\begin{pmatrix} 1 & 1 & 1 & 1 \\ 0 & 3 & 2 & 1 \\ 0 & 2 & 5 & 5 \end{pmatrix}$ | $(0,1,2)^{**}$ |
| 4 | 4 | 4 | (see previous entry) | |
| 5 | 5 | **6** | $\begin{pmatrix} 3 & 2 & 2 & 3 \\ 4 & 3 & 0 & 7 \\ 0 & 2 & 5 & 5 \end{pmatrix}$ | $(1,2,2)$ |
| 6 | 6 | 6 | $\begin{pmatrix} 3 & 4 & 0 & 7 \\ 6 & 1 & 4 & 3 \\ 2 & 6 & 7 & 1 \end{pmatrix}$ | $(2,2,2)$ |
| 7 | 6 | **8** | (see previous entry) | |
| 8 | 7 | **8** | $\begin{pmatrix} 7 & 6 & 2 & 1 \\ 14 & 5 & 0 & 15 \\ 10 & 4 & 17 & 1 \end{pmatrix}$ | $(2,3,3)$ |
| 9 | 8 | 8 | $\begin{pmatrix} 1 & 14 & 16 & 3 \\ 10 & 13 & 2 & 7 \\ 16 & 0 & 3 & 13 \end{pmatrix}$ | $(3,3,3)$ |

* The $(4, 3, 2, 3)$ Wyner-Ash code. See Section 7.

** The $(4, 3, 3, 4)$ extended Wyner-Ash code. It is optimal, but noncompact. See Section 7.

## 5. Physical Encoder Complexity.

*Abstract: In this section we will see that any polynomial generator matrix, and more generally, any causal generator matrix, for an $(n, k)$ convolutional code can be used as a blueprint for building a physical encoder for the code, i.e., a device which can be used to transform $k$ parallel streams of information symbols into $n$ parallel streams of coded symbols. If we define the degree of such an encoder as the number of delay (memory) elements it contains, we will see that the "direct-form," or "controller canonical form" physical realization of a canonical encoder has minimum degree among all encoders. This minimum is however not attained uniquely by the canonical encoders; there is a larger class, the minimal encoders, which includes the "systematic" encoders, any member of which can be realized with the same number of delays as a canonical encoder. This minimum encoder degree turns out to be the degree of the code, as defined in Section 3. Except as otherwise noted, all of the results in this section are taken from Forney's original paper [13].*

Recall that an $(n, k)$ convolutional code $\boldsymbol{C}$ over a field $F$ is a $k$-dimensional subspace of $F(D)^n$. An *abstract linear encoder* $\phi$ for $\boldsymbol{C}$ is a linear mapping from $F(D)^k$ onto $\boldsymbol{C}$. Thus $\phi$ provides a way of transforming, or *encoding*, an arbitrary $k$-tuple of rational functions $\boldsymbol{u}(D) = (u_1(D), \ldots, u_k(D))$ over $F$ into a codeword $\boldsymbol{x}(D) = (x_1(D), \ldots, x_n(D))$ from $\boldsymbol{C}$.

If $\phi$ is an abstract linear encoder for $\boldsymbol{C}$, it is fully determined by its action on any basis for $F(D)^k$. If we choose the "natural" basis $(\boldsymbol{\epsilon}_1(D), \ldots, \boldsymbol{\epsilon}_k(D))$ for $F(D)^k$ defined by

$$\begin{aligned}
\boldsymbol{\epsilon}_1(D) &= (1, 0, 0, \ldots, 0) \\
\boldsymbol{\epsilon}_2(D) &= (0, 1, 0, \ldots, 0) \\
&\vdots \\
\boldsymbol{\epsilon}_k(D) &= (0, 0, 0, \ldots, 1),
\end{aligned}$$

then for $i = 1, 2, \ldots, k$, if we define $\boldsymbol{g}_i(D) = (g_{i,1}(D), \ldots, g_{i,n}(D)) = \phi(\boldsymbol{\epsilon}_i(D))$, and the $k \times n$ matrix $G(D)$ as

$$G(D)_{i,j} = g_{i,j}(D),$$

then it follows from linear algebra that $G(D)$ is a generator matrix for $\boldsymbol{C}$, and that for any $\boldsymbol{u}(D) \in F(D)^k$,

$$\phi(D) = \boldsymbol{u}(D) \cdot G(D).$$

Conversely, if $G(D)$ is a arbitrary generator matrix for $\boldsymbol{C}$, then the mapping $\phi : \boldsymbol{u}(D) \rightarrow u(D)G(D)$ is an abstract linear encoder for $\boldsymbol{C}$. In other words, the *abstract linear encoders* for $\boldsymbol{C}$ are in one-to-one correspondence with the *generator matrices* for $\boldsymbol{C}$.

If $\phi$ is a given abstract linear encoder for $\boldsymbol{C}$, with corresponding generator matrix $G(D)$, a *physical realization* $E$ of $\phi$ is a circuit built from adders, multipliers,* and memory

* We will illustrate everything with the field $GF(2)$, so that no multipliers will be necessary.

elements (with respect to the underlying field $F$) that represents $\phi$. That is, $E$ is a $k$-input, $n$-output linear time-invariant circuit with the property that if an arbitrary $k$-tuple of rational functions over $F$, say $\boldsymbol{u}(D)$, as represented by $k$ parallel sequences of elements from $F$ (the coefficients of the Laurent series expansions of the components of $\boldsymbol{u}(D)$), is presented to $E$ as the input, then the output will be a $n$ parallel sequences of elements from $F$, representing the codeword $\boldsymbol{x}(D) = \phi(D) = \boldsymbol{u}(D)G(D)$.

In Figures 5.2 through 5.8, we see signal flow graphs which represent physical realizations for the abstract encoders $G_2(D), \ldots, G_8(D)$ given in Example 3.4. (There can be no physical realization for $G_1(D)$, since it is noncausal, and therefore cannot be realized by a circuit that contains only delay elements.) In these figures, in order to save space, we have adopted a shorthand notation for the circuit elements, which is described in Figure 5.1. (In Figures 5.2–5.8, there are no arrows showing the direction of signal flow. It is to be understood, however, that all signals flow either *downward* or *rightward*.) The number of delay elements in a physical encoder is of key importance, so we pause for a definition.

**5.1 Definition.** *The number of delay elements in a physical encoder is called the degree of the encoder.*

As an illustration, let's verify the correctness of the circuit in Figure 5.2, which is a degree 3, memory 2, physical realization of the abstract linear encoder represented by the generator matrix

$$G_2(D) = \begin{pmatrix} 1 & 1+D+D^2 & 1+D^2 & 1+D \\ D & 1+D+D^2 & D^2 & 1 \end{pmatrix}.$$

The circuit in Figure 5.2 has two "input signal" wires, labelled $u_1$ and $u_2$, and four "output signal" wires, labelled $x_1$, $x_2$, $x_3$, and $x_4$. Each of these six signals should be thought of as a sequence of elements from the symbol field, which is this case is $GF(2)$. Thus the value of the signal $u_1$ at time $i$ is denoted by $u_1(i)$, etc. The circuit is designed so that the two input signals $u_1$ and $u_2$ are added and delayed in various ways, using the rules of Figure 5.1, so that at the top right part of the circuit, the five parallel horizontal wires carry the following five signals:

$$\begin{aligned} z_1(i) &= u_1(i) \\ z_2(i) &= u_2(i) \\ z_3(i) &= u_1(i-2) + u_2(i-2) + u_1(i-1) \\ z_4(i) &= u_1(i-1) + u_2(i-1) \\ z_5(i) &= u_2(i-1) \end{aligned} \tag{5.1}$$

As we did in Section 2, we will find it convenient to work with the generating-fucntion form of these signals. If we write (5.1) in generating-function form, we obtain (here $Z_1(D) =$

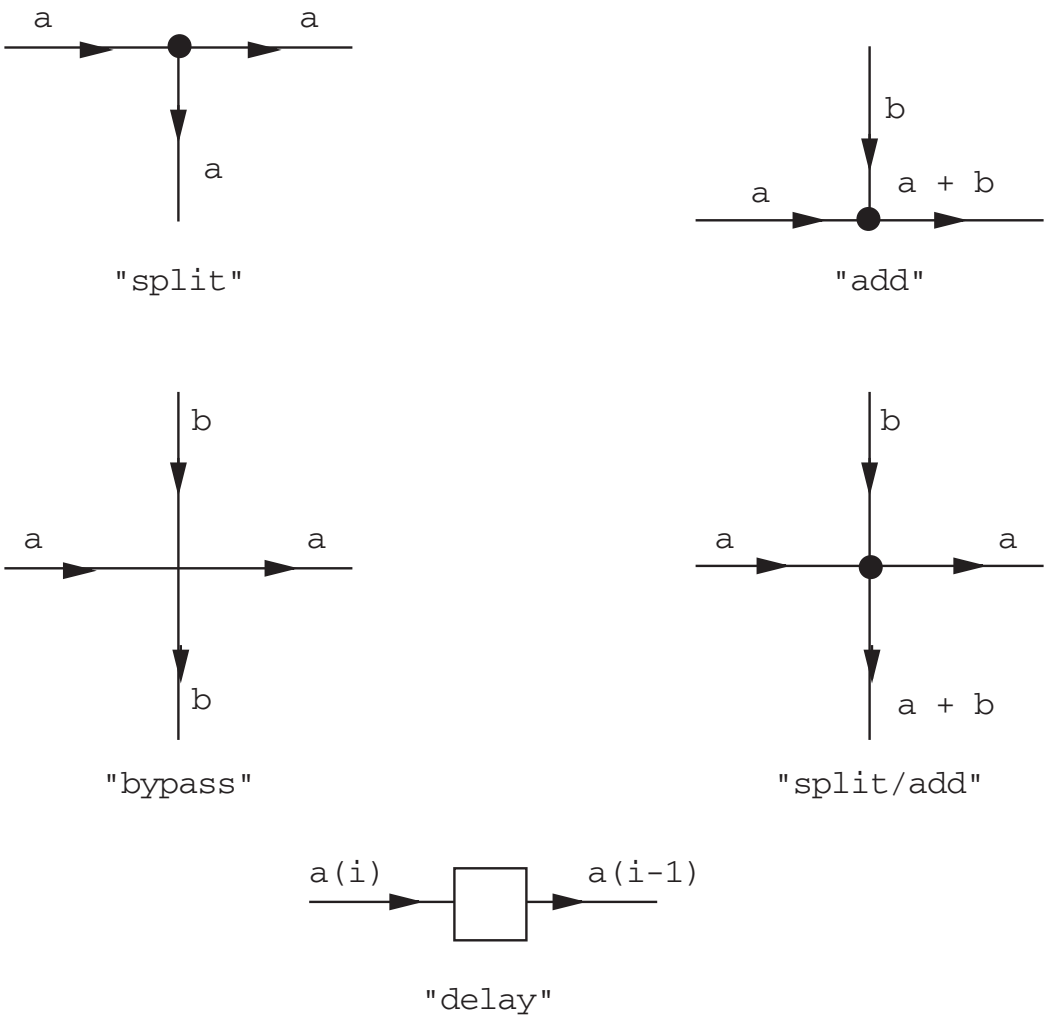


**Figure 5.1.** Explanation of the circuit elements used in the signal flowgraphs of Figures 5.2–5.8.

$\sum_i z_1(i)D^i$, and so forth)

$$
\begin{aligned}
Z_1(D) &= U_1(D) \\
Z_2(D) &= U_2(D) \\
Z_3(D) &= (D + D^2)U_1(D) + D^2U_2(D) \\
Z_4(D) &= DU_1(D) + DU_2(D) \\
Z_5(D) &= DU_2(D).
\end{aligned} \tag{5.2}
$$

If these five signals are added together, as shown in Figure 5.2, we obtain the following

formulas for the four output signals:

$$\begin{aligned}
X_1(D) &= Z_1(D) + Z_5(D) \\
&= U_1(D) + DU_2(D) \\
X_2(D) &= Z_1(D) + Z_2(D) + Z_3(D) + Z_5(D) \\
&= (1 + D + D^2)U_1(D) + (1 + D + D^2)U_2(D) \\
X_3(D) &= Z_1(D) + Z_3(D) + Z_4(D) + Z_5(D) \\
&= (1 + D^2)U_1(D) + D^2U_2(D) \\
X_4(D) &= Z_1(D) + Z_2(D) + Z_4(D) + Z_5(D) \\
&= (1 + D)U_1(D) + U_2(D),
\end{aligned}$$

which is equivalent to the matrix equation

$$(X_1(D), X_2(D), X_3(D), X_4(D)) = (U_1(D), U_2(D))G_2(D).$$

The verification of the correctness of the encoders in Figures 5.3–5.7 is similar.

The design of circuits of the kind depicted in Figures 5.2–5.8 is a bit tricky, especially if the goal is to produce a physical realization with minimum possible degree, but a discussion this is beyond the scope of this article. For a thorough explanation of circuits of this kind, we refer the reader to Vaidyanathan [52], especially Section 2.1.3. However, note that the circuits in Figures 5.5, 5.6, and 5.7 are relatively straightforward. For example, consider the circuit of Figure 5.5, which is a physical realization of the abstract linear encoder corresponding to the generator matrix

$$G_5(D) = \begin{pmatrix} 1 + D & 0 & 1 & D \\ D & 1 + D + D^2 & D^2 & 1 \end{pmatrix}.$$

The intermediate wires $z_1, z_2, z_3, z_4, z_5$ carry only single inputs, with or without delays:

$$\begin{aligned}
z_1 &= u_1 \\
z_2 &= u_2 \\
z_3 &= u_1D \\
z_4 &= u_2D^2 \\
z_5 &= u_2D,
\end{aligned}$$

so that the computation of the four output signals is quite easy (for simplicity of notation we will no longer distingush between a signal and its generating function):

$$\begin{aligned}
x_1 &= z_1 + z_3 + z_5 = u_1(1 + D) + u_2D \\
x_2 &= z_2 + z_4 + z_5 = u_2(1 + D + D^2) \\
x_3 &= z_1 + z_4 = u_1 + u_2D^2 \\
x_4 &= z_2 + z_3 = u_1D + u_2.
\end{aligned}$$

Circuits of this elementary type are called *direct-form* realizations of the corresponding circuits. (An alternate term is "controller canonical form.") The circuits in Figures 5.5–5.7 are all direct form realizations. By definition, the direct-form realization of a generator matrix $G(D)$ with row degrees $d_1, \ldots, d_k$ has exactly $d_1 + \cdots + d_k = \operatorname{extdeg} G(D)$ delay elements, i.e., it is an encoder of degree $\operatorname{extdeg} G(D)$.

Finally, note that the circuit of degree 1 depicted in Figure 5.8 contains *feedback*, which deserves special consideration. The first horizontal wire in Figure 5.8 carries the signal $z_1 = u_1$, the second carries $z_2 = u_2$, and because of the little feedback circuit, the third horizontal wire carries the signal $z_3 = \frac{D}{1+D}(u_1 + u_2)$. Thus the four outputs $(x_1, x_2, x_3, x_4)$ are given by

$$\begin{aligned} x_1 &= z_1 = u_1 \\ x_2 &= z_2 = u_2 \\ x_3 &= z_1 + z_3 \\ &= u_1 \frac{1}{1+D} + u_2 \frac{D}{1+D} \\ x_4 &= z_2 + z_3 \\ &= u_1 \frac{D}{1+D} + u_2 \frac{1}{1+D}, \end{aligned}$$

which is equivalent to the equation $(x_1, x_2, x_3, x_4) = (u_1, u_2)G_8(D)$. Because of the feedback, this encoder has infinite memory (See Figure 2.2).

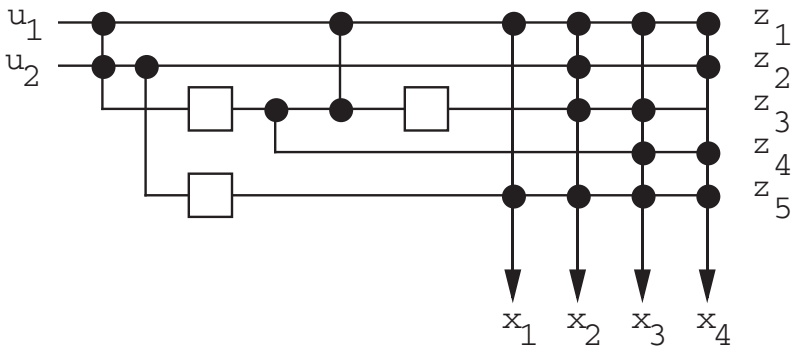


**Figure 5.2.** A degree-3 physical realization of the abstract encoder $G_2(D)$. (For explanation of the signals $z_1, \ldots, z_5$, see text.)

Let $\boldsymbol{C}$ be a fixed $(n, k, m)$ convolutional code over the field $F$. For a given generator matrix $G(D)$, there will in general be many possible physical realizations for $G(D)$. The *McMillan degree* of $G(D)$ is defined to be the minimum degree of any realization of $G(D)$ (see [52, Chapter 13], for details). This quantity is important, both because it tells us

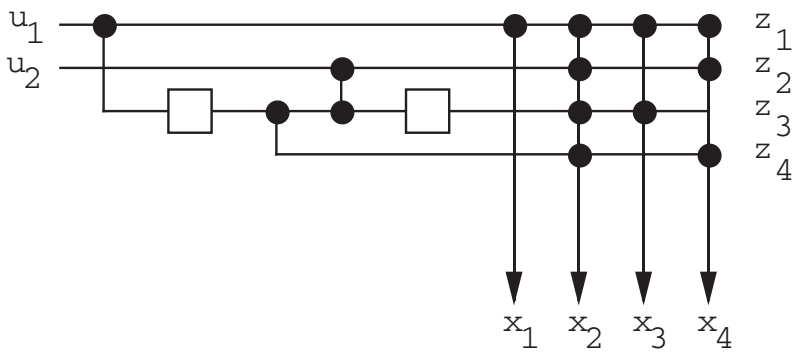


**Figure 5.3.** A degree-2 physical realization of the abstract encoder $G_3(D)$.

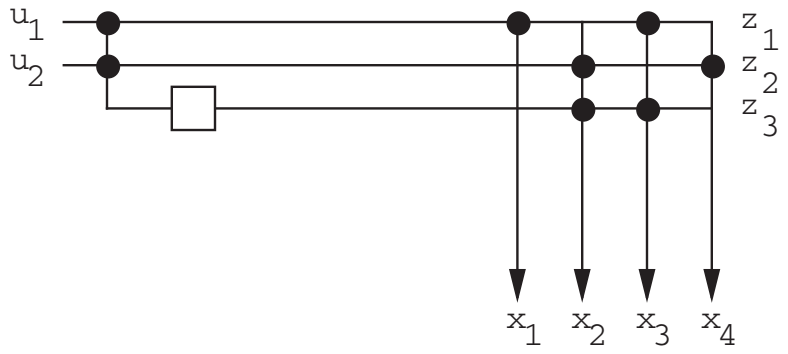


**Figure 5.4.** A degree-1 physical realization of the abstract encoder $G_4(D)$.

something about the complexity of building an encoder for $G$, and also because it represents the minimum dimension of the state space for any realization of $G(D)$; and the complexity of the Viterbi decoding algorithm is an exponential function of the degree of the encoder [41, Chapter 9], [57, Chapter 12]. It turns out that each of the physical realizations shown in Figures 5.2 through 5.8 has minimum degree among all realizations of the corresponding $G(D)$, and so the McMillan degree of each of the generator matrices $G_2(D), \ldots, G_8(D)$ is as given in Table 5.1. (In general, as we shall see below, the term "minimal" means that no encoder can have smaller degree. For this particular code, the minimum possible degree is one, so that only the encoders which use exactly one delay, i.e., the encoders in Figures 5.4, 5.6, and 5.8, are minimal.)

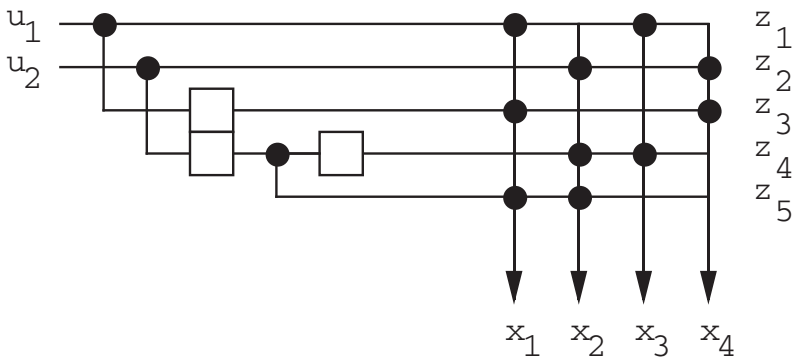


**Figure 5.5.** A degree-3 physical (direct-form) realization of the abstract encoder $G_5(D)$.

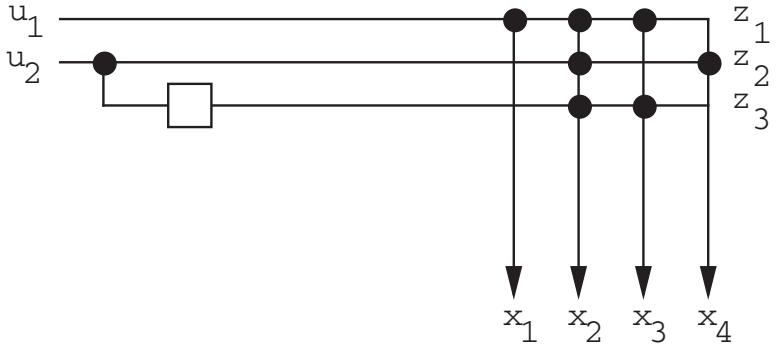


**Figure 5.6.** A degree-1 physical (direct-form) realization of the abstract encoder $G_6(D)$.

**Table 5.1. Some generator matrices for a $(4, 2, 1, 4)$ code.**

| | Basic? | Reduced? | Minimal? | intdeg | Mcdeg | extdeg |
|---|---|---|---|---|---|---|
| $G_2$ | No | No | No | 3 | 3 | 4 |
| $G_3$ | Yes | No | No | 1 | 2 | 3 |
| $G_4$ | Yes | No | Yes | 1 | 1 | 2 |
| $G_5$ | No | Yes | No | 3 | 3 | 3 |
| $G_6$ | Yes | Yes | Yes | 1 | 1 | 1 |
| $G_7$ | No | Yes | No | 2 | 2 | 2 |
| $G_8$ | – | – | Yes | – | 1 | – |

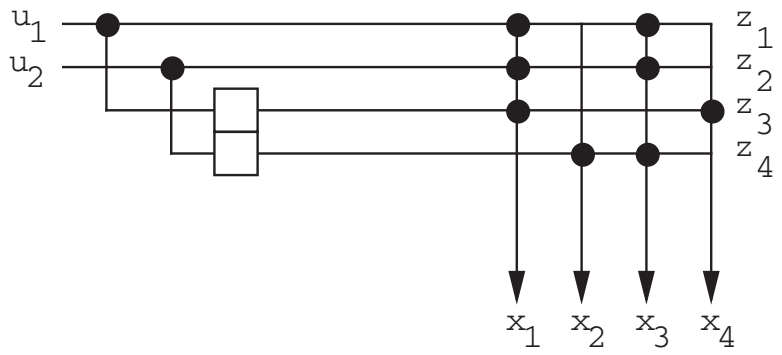


**Figure 5.7.** A degree-2 physical (direct-form) realization of the abstract encoder $G_7(D)$.

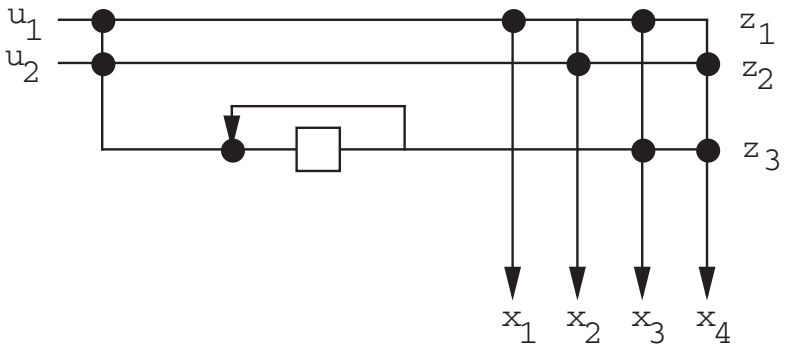


**Figure 5.8.** A degree-1 physical realization of the (systematic) abstract encoder $G_8(D)$.

In Table 5.1, it is seen "experimentally" that for any polynomial generator matrix $G(D)$, we have

$$\text{intdeg}\, G(D) \leq \text{Mcdeg}\, G(D) \leq \text{extdeg}\, G(D). \tag{5.3}$$

It turns out that these two inequalities are true in general. We shall give a proof of the upper bound on $\text{Mcdeg}\, G(D)$ in eq. (5.3) in Theorem 5.5, below, but the proof of the lower bound is too long to include here.

The above table also shows "experimentally" that, among all encoders for a given code, the minimum McMillan degree is equal to the degree of the code. This fact too is true in general, and in the remainder of this section we will give a complete proof of this fact. Among other things, we will see that a direct-form realization of a canonical generator matrix will always give the minimum possible McMillan degree. Our proofs will depend on certain well-developed ideas from linear system theory.

Any rational function of the form $p(D)/q(D)$, with $q(D) \neq 0$, can expressed as a Laurent series in ascending powers of $D$, viz,

$$\frac{p(D)}{q(D)} = \sum_{k=k_0}^{\infty} a_k D^k, \tag{5.4}$$

where $k_0$ is possibly negative. Recall that if $k_0 \geq 0$, i.e., if the Laurent series for $p(D)/q(D)$ involves only nonnegative powers of $D$, the function is said to be *causal*. This is equivalent to saying that the denominator $q(D)$ has the property that $q(0) \neq 0$.

If $p(D)/q(D)$ isn't causal, we define its *causal part* to be the power series obtained by taking only the terms of nonnegative degree in $D$ from the Laurent series, i.e.,

$$\text{causal part } \frac{p(D)}{q(D)} = \sum_{k=\max(k_0,0)}^{\infty} a_k D^k. \tag{5.5}$$

More generally, the *causal part* of a codeword $(x_1(D), \ldots, x_n(D))$ from a convolutional code is defined to be $(x_1(D)^*, \ldots, x_n(D)^*)$, where $x_i(D)^*$ is the causal part of $x_i(D)$, for $i = 1, 2, \ldots, n$.

Now suppose that $\boldsymbol{C}$ is an $(n, k)$ convolutional code with Forney indices $(e_1, e_2, \ldots, e_k)$, and degree $m = e_1 + \cdots + e_k$. Define $\boldsymbol{C}^*$ to be the set of codewords of $\boldsymbol{C}$ *whose causal part is also a codeword*. Both $\boldsymbol{C}$ and $\boldsymbol{C}^*$ are (infinite-dimensional) vector spaces over the base field $F$, and $\boldsymbol{C}^*$ is a subspace of $\boldsymbol{C}$. The *abstract state space of* $\boldsymbol{C}$, denoted by $\Sigma_C$, is defined as the corresponding quotient space:

$$\Sigma_{\boldsymbol{C}} = \boldsymbol{C}/\boldsymbol{C}^*. \tag{5.6}$$

The main result about the abstract state space is given in the following theorem.

**5.2 Theorem.** *If $\Sigma_{\boldsymbol{C}}$ is the abstract state space for a given convolutional code, as defined in (5.6), then*

$$\dim \Sigma_{\boldsymbol{C}} = \deg \boldsymbol{C} = m.$$

**Proof:** Let $G(D)$ be a canonical generator matrix for $\boldsymbol{C}$, with rows $\boldsymbol{g}_1(D), \ldots, \boldsymbol{g}_k(D)$, where $\boldsymbol{g}_i(D) \in F[D]^n$, and $\deg \boldsymbol{g}_i(D) = e_i$, for $i = 1, 2, \ldots, k$. Now consider the following $m$ codewords from $\boldsymbol{C}$:

$$\begin{aligned} \boldsymbol{b}_{i,j}(D) = D^j \boldsymbol{g}_i(D), \quad & i = 1, \ldots, k \\ & j = -1, \ldots, -e_i. \end{aligned}$$

We will show that $\{\boldsymbol{b}_{i,j}(D)\}$ is a basis for $\Sigma_{\boldsymbol{C}}$; since this set contains exactly $e_1 + \cdots + e_k = m$ elements, this will prove the theorem. First we will show that $\{\boldsymbol{b}_{i,j}(D)\}$ spans $\Sigma_{\boldsymbol{C}}$; then we'll show that the $\boldsymbol{b}_{i,j}$'s are linearly independent modulo $\boldsymbol{C}^*$.

Thus let $\boldsymbol{x}(D) = \sum_{i=1}^{k} u_i(D)\boldsymbol{g}_i(D)$ be an arbitrary codeword from $\boldsymbol{C}$, where $u_i(D) = \sum_j u_{i,j}D^j$ is the expansion of $u_i(D)$ as a Laurent series. Define

$$u'_i(D) = \sum_{j=-e_i}^{-1} u_{i,j}D^j$$

and

$$\boldsymbol{x}'(D) = \sum_{i=1}^{k} u'_i(D)\boldsymbol{g}_i(D).$$

Then

$$\begin{aligned}\boldsymbol{x}'(D) &= \sum_{i=1}^{k}\sum_{j=-e_i}^{-1} u_{i,j}D^j\boldsymbol{g}_i(D)\\ &= \sum_{i=1}^{k}\sum_{j=-e_i}^{-1} u_{i,j}\boldsymbol{b}_{i,j},\end{aligned}$$

so that $\boldsymbol{x}'(D)$ is in the $F$-space spanned by the $\boldsymbol{b}_{i,j}$'s. But also

$$\begin{aligned}\boldsymbol{x}(D) - \boldsymbol{x}'(D) &= \sum_{i=1}^{k}\sum_{j} u_{i,j}D^j\boldsymbol{g}_i(D) - \sum_{i=1}^{k}\sum_{j=-e_i}^{-1} u_{i,j}D^j\boldsymbol{g}_i(D)\\ &= \sum_{i=1}^{k}\left(\sum_{j<-e_i} u_{i,j}D^j\right)\boldsymbol{g}_i(D) + \sum_{i=1}^{k}\left(\sum_{j\geq 0} u_{i,j}D^j\right)\boldsymbol{g}_i(D)\\ &= \text{an anticausal codeword} + \text{a causal codeword}.\end{aligned}$$

Hence $\boldsymbol{x}(D)-\boldsymbol{x}'(D) \in \boldsymbol{C}^*$, so that $\{\boldsymbol{b}_{i,j}\}$ does span $\boldsymbol{C}/\boldsymbol{C}^*$. (Note that this half of the proof does not use the fact that $G(D)$ is canonical, and shows that $\dim \boldsymbol{C}/\boldsymbol{C}^* \leq \operatorname{extdeg} G(D)$ for any polynomial generator matrix for $\boldsymbol{C}$.)

Now we need to show that the $\boldsymbol{b}_{i,j}$'s are linearly independent over $\boldsymbol{C}/\boldsymbol{C}^*$. In other words, if the causal part of some linear combination of the $\boldsymbol{b}_{i,j}$'s is a codeword, i.e.,

$$\boldsymbol{x}(D) = \sum_{i=1}^{k}\sum_{j=-e_i}^{-1} u_{i,j}\boldsymbol{b}_{i,j}(D) \in \boldsymbol{C}^*, \tag{5.7}$$

we must show that $u_{i,j} = 0$ for all $i, j$. To do this note first that (5.7) can be written as

$$\boldsymbol{x}(D) = \sum_{i=1}^{k} u_i(D)\boldsymbol{g}_i(D), \tag{5.8}$$

where $\deg \boldsymbol{u}_i(D)\boldsymbol{g}_i(D) \leq e_i - 1$. If $\boldsymbol{x}^*(D)$ denotes the causal part of $\boldsymbol{x}(D)$, and if $\boldsymbol{x}^*(D)$ is a codeword, then we have

$$\boldsymbol{x}^*(D) = \sum_{i=1}^{k} u'_i(D)\boldsymbol{g}_i(D), \tag{5.9}$$

for rational fundtions $u'_1(D), \ldots u'_k(D)$. But we are assuming that $G(D)$ is canonical, and so basic, and thus by Theorem A.1, part (5), the $u'_i(D)$ are in fact polynomials. Thus the "strictly anticausal part" of $\boldsymbol{x}(D)$, i.e., the terms in $\boldsymbol{x}(D)$ of negative degree, can be written as

$$\begin{aligned} \boldsymbol{x}(D) - \boldsymbol{x}^*(D) &= \sum_{i=1}^{k} (u_i(D) - u'_i(D))\boldsymbol{g}_i(D) \\ &= \sum_{i=1}^{k} u''_i(D)\boldsymbol{g}_i(D). \end{aligned} \tag{5.10}$$

Now if even one of the $u'_i(D)$'s is nonzero, then $\deg u''_i(D) \geq 0$, so that $\deg u''_i \boldsymbol{g}_i(D) \geq e_i \geq 0$, so that by the "predictable degree property" (Theorem A.2, part (3)), $\deg(\boldsymbol{x}(D) - \boldsymbol{x}'(D)) \geq 0$, which is a contradiction. Thus $u'_i(D) = 0$, for $i = 1, \ldots, k$, which by (5.9) means that $\boldsymbol{x}^*(D) = 0$. But by the predictable degree property again, (5.7) impies that $\deg \boldsymbol{x}(D) \geq 0$, unless all of the $u_{i,j}$'s are zero. This completes the proof. ▪

**5.3 Theorem.** *If $K$ is any physical encoder for $\boldsymbol{C}$, then* $\deg K \geq \dim \Sigma_{\boldsymbol{C}}$.

**Proof:** Let $\boldsymbol{x}_1(D)$ and $\boldsymbol{x}_2(D)$ be two codewords which are inequivalent mod $\boldsymbol{C}^*$, i.e., such that the causal part of $\boldsymbol{x}_1(D) - \boldsymbol{x}_2(D)$ isn't a codeword. Let $\sigma_1$ and $\sigma_2$ be the corresponding $K$-physical states, i.e., the contents of the delay elements of $K$, corresponding to these two codewords at $t = 0$. If these two states were identical, the $t = 0$ physical state of the codeword $\boldsymbol{x} = \boldsymbol{x}_1 - \boldsymbol{x}_2$ would be zero. But since by definition, a codeword is the output of an encoder in response to a given input, *assuming the initial state of the encoder to be 0*, this would mean that the part of $\boldsymbol{x}$ beginning at $t = 0$, i.e., the causal part of $\boldsymbol{x}$, is a codeword, contradicting our assumption that $\boldsymbol{x}_1 - \boldsymbol{x}_2 \notin \boldsymbol{C}^*$. Thus since distinct *code states* lead to distinct *$K$-physical* states, $\dim \Sigma_{\boldsymbol{C}} \leq \deg K$. ▪

**5.4 Theorem.** *If $G(D)$ is any causal generator matrix for $\boldsymbol{C}$, then* $\operatorname{Mcdeg} G \geq \deg \boldsymbol{C} = e_1 + \cdots + e_k$.

**Proof:** Combine Theorems 5.2 and 5.3. ▪

**5.5 Theorem.** *If $G(D)$ is a polynomial generator matrix for the code $\boldsymbol{C}$, then*

$$\operatorname{Mcdeg} G(D) \leq \operatorname{extdeg} G(D),$$

*and equality holds if $G(D)$ is canonical.*

**Proof:** As we observed earlier, the direct-form realization of any polynomial generator matrix requires exactly $\operatorname{extdeg} G(D)$ delay elements. Since $\operatorname{Mcdeg} G(D)$ is defined to be the minimum possible number of delay elements in any realization of $G(D)$, it follows that $\operatorname{Mcdeg} G(D) \leq \operatorname{extdeg} G(D)$. Also, from Theorem 5.4, we know that $\operatorname{Mcdeg} G(D) \geq \deg \boldsymbol{C}$. But from the results in Section 3, we know that if $G(D)$ is canonical, then $\operatorname{extdeg} G(D) = \deg \boldsymbol{C}$, so that $\operatorname{extdeg} G(D) = \operatorname{Mcdeg} G(D)$. ▪

**5.6 Theorem.** *Among all realizable generator matrices for $\boldsymbol{C}$, the smallest possible McMillan degree is the degree of the code. Furthermore, the direct-form realization of any canonical generator matrix for $\boldsymbol{C}$ achieves this minimum.*

**Proof:** According to Theorem 5.4, if $G(D)$ is realizable, then $\operatorname{Mcdeg} G(D) \geq \deg \boldsymbol{C}$. On the other hand, if $G(D)$ is a canonical generator matrix for $\boldsymbol{C}$ its direct-from realization requires $\operatorname{extdeg} G(D)$ delay elements; but by definition 3.5, $\operatorname{extdeg} G(D) = \deg \boldsymbol{C}$. ▪

Theorem 5.6 shows that the direct-form realizations of canonical generator matrices require as few delay elements as possible. However, it is important to note that there can be other encoders that do as well. For example, for the $(4, 2)$ code illustrated in Table 5.1, for which $\deg \boldsymbol{C} = 1$, we see from Figures 5.4, 5.6, and 5.8 that the encoders for $G_4(D)$, $G_6(D)$, and $G_8(D)$ all have degree 1, but only the encoder in Figure 5.6 for $G_6$ is a direct-form realization for a canonical generator matrix. In general, generator matrices whose McMillan degree equals $\deg \boldsymbol{C}$ are called *minimal* generator matrices. The following theorem, whose proof is too long to include here, gives a useful characterization of minimal generator matrices. (See [13], [14], [27], [19], and [53] for a complete history and discussion of this interesting result. Also see [19] for a different characterization of minimal generator matrices.)

**5.7 Theorem.** *If $G(D)$ is a realizable generator matrix for $\boldsymbol{C}$, then $G(D)$ is minimal, i.e.,*

$$\operatorname{Mcdeg} G(D) = \deg \boldsymbol{C}, \tag{5.11}$$

*if and only if $G(D)$ has both a polynomial right inverse, i.e., a right inverse whose entries are each polynomials in $D$, and an antipolynomial right inverse, i.e., one whose entries are polynomials in $D^{-1}$.* ▪

**5.8 Example:** Consider the following generator matrix, which is taken from Table 5.1.

$$G_4(D) = \begin{pmatrix} 1 & D & 1+D & 0 \\ 0 & 1+D & D & 1 \end{pmatrix}.$$

This matrix is not canonical, since it is not reduced. However, in Figure 5.4 we see a physical implementation of $G_4(D)$ of degree 1, so $G_4(D)$ is minimal. Thus according to Theorem 5.7, $G_4(D)$ ought to have both a polynomial and an antipolynomial right inverse. Indeed, a simple calculation shows that

$$K_1(D) = \begin{pmatrix} 1+D & 0 \\ 1 & 0 \\ 0 & 0 \\ 1+D & 1 \end{pmatrix}$$

is a polynomial right inverse, and

$$K_2(D) = \begin{pmatrix} 0 & 0 \\ D^{-1} & 0 \\ 0 & 0 \\ 1+D^{-1} & 1 \end{pmatrix}$$

is an antipolynomial right inverse. ▪

An important class of nonpolynomial realizable generator matrices for convolutional codes are the *systematic* generator matrices, which are defined as those of the form $G(D) = (I_k|X)$, where $I_k$ is the $k \times k$ identity matrix, and $X$ is a $k \times (n-k)$ matrix of causal rational functions. For example, the generator matrix

$$G_8(D) = \begin{pmatrix} 1 & 0 & 1/(1+D) & D/(1+D) \\ 0 & 1 & D/(1+D) & 1/(1+D) \end{pmatrix}$$

illustrated in Figure 5.8, is a systematic generator matrix. Since a matrix of the form $G(D) = (I_k|X)$ has the matrix $H(D) = \begin{pmatrix} I_k \\ 0 \end{pmatrix}$ for a right inverse, and $H(D)$ is both polynomial and antipolynomial, it follows from Theorem 5.7 that *every systematic generator matrix is minimal.*

It turns out that there are exactly $q^{mk}\prod_{i=0}^{k-1}(q^k - q^i)$ minimal generator matrices for an $(n,k,m)$ convolutional code over $GF(q)$, a result due to Dickinson [10]. For our $(4,2,1)$ example, this number is 24. $G_4$, $G_6$, and $G_8$ account for only three such. According to Dickinson's results, the complete set of 24 minimal generator matrices for this particular code are the matrices of the form $G = QH$ where $Q$ is one of the six $2 \times 2$ nonsingular scalar matrices over $GF(2)$, and $H$ is one of the following 4 generator matrices:

$$H_1(D) = \begin{pmatrix} 1 & 1 & 1 & 1 \\ 0 & D+1 & D & 1 \end{pmatrix}$$
$$H_2(D) = \begin{pmatrix} 1 & 1 & 1 & 1 \\ D & 1 & 0 & D+1 \end{pmatrix}$$
$$H_3(D) = \begin{pmatrix} 1 & 1 & 1 & 1 \\ 0 & 1 & \frac{D}{D+1} & \frac{1}{D+1} \end{pmatrix}$$
$$H_4(D) = \begin{pmatrix} 1 & 1 & 1 & 1 \\ \frac{D}{D+1} & \frac{1}{D+1} & 0 & 1 \end{pmatrix}.$$

For example the minimal generator matrix $G_4(D)$ from Example 5.8 is given by

$$G_4(D) = \begin{pmatrix} 1 & 1 \\ 0 & 1 \end{pmatrix} H_3(D).$$

The number of *canonical* generator matrices is also known [36]. For the $(4,2,1)$ code of Example 3.4, the number is 4. Finally, it is not hard to show that the number of merely *basic* generator matrices is infinite.

## 6. Catastrophic Generator Matrices and Encoders.

*Abstract: We saw in Section 5 that any causal generator matrix for a convolutional code can be used as a blueprint for a physical encoder for the code. Some encoders are to be preferred to others, because of complexity considerations, but one class of encoders must be avoided at all costs. These are the catastrophic encoders. If a catastrophic encoder is used in an application, the resulting decoder bit error probability will approach 1/2, no matter how good the underlying code, and how "quiet" the transmission channel, may be.*

We begin this section with a dictum, which we hope will dispel a widely-believed myth:

**There is no such thing as a catastrophic convolutional code.**

But there *is* such a thing as a catastrophic generator matrix (encoder) for a convolutional code. Let us explain why, by thinking about a general communication system using a given $(n, k, m)$ convolutional code $\boldsymbol{C}$, with a generator matrix $G(D)$ as encoder. An information word $\boldsymbol{u}(D)$ is encoded into the codeword $\boldsymbol{x}(D) = \boldsymbol{u}(D)G(D)$, and $\boldsymbol{x}(D)$ is then transmitted over a noisy channel and received, possibly badly garbled, as $\boldsymbol{y}(D)$. The decoder must make an estimate of $\boldsymbol{u}(D)$, based on the garbled codeword $\boldsymbol{y}(D)$. The decoding algorithm can be broken down into two steps, the "hard" job and the "easy" job. The hard job is to find a codeword, say $\widehat{\boldsymbol{x}}(D)$, which is "close" to $\boldsymbol{y}(D)$, the hope being that $\widehat{\boldsymbol{x}}(D) = \boldsymbol{x}(D)$. The easy job is then to calculate the information word $\widehat{\boldsymbol{u}}(D)$ corresponding to the codeword $\widehat{\boldsymbol{x}}(D)$. It is in the easy step that catastrophes can occur!

If the decoder's estimate of $\boldsymbol{x}(D)$ is $\widehat{\boldsymbol{x}}(D)$, its estimate of $\boldsymbol{u}(D)$ will be

$$\widehat{\boldsymbol{u}}(D) = \widehat{\boldsymbol{x}}(D) \cdot K(D), \tag{6.1}$$

where $K(D)$ is any $n \times k$ right inverse for $G(D)$, i.e., an $n \times k$ matrix over $F(D)$ such that $G(D)K(D) = I_k$, where $I_k$ is the $k \times k$ identity matrix. If we define the *codeword error* by $\boldsymbol{e}_C(D) = \boldsymbol{x}(D) - \widehat{\boldsymbol{x}}(D)$, and the *information error* by $\boldsymbol{e}_I(D) = \boldsymbol{u}(D) - \widehat{\boldsymbol{u}}(D)$, it follows from (6.1) that

$$\boldsymbol{e}_I(D) = \boldsymbol{e}_C(D) \cdot K(D). \tag{6.2}$$

A decoding "catastrophe" is said to have occurred if the codeword error has finite weight, but the corresponding information error has infinite weight. Since $\boldsymbol{e}_C(D)$ is a difference of codewords, it is also a codeword. Thus we are led to the following definition.

**6.1 Definition.** *A generator matrix $G(D)$ is said to be catastrophic if there is an infinite-weight vector $\boldsymbol{u}(D) \in F(D)^k$ such that the corresponding codeword $\boldsymbol{x}(D) = \boldsymbol{u}(D)G(D)$ has finite weight.*

**6.2 Example:** Let's consider the $(2, 1, 2, 5)$ convolutional code $\boldsymbol{C}$ defined by the canonical generator matrix

$$G_0(D) = \begin{pmatrix} 1 + D + D^2 & 1 + D^2 \end{pmatrix}.$$

(This code was discussed in Example 2.5 and appears as the $m = 2$ entry in Table 4.1.) If we multiply $G_0(D)$ by $1 + D$, we obtain the alternative generator matrix

$$G(D) = (\, 1 + D^3 \quad 1 + D + D^2 + D^3 \,),$$

which is catastrophic. Let us see why. A simple calculation shows that

$$K(D) = \begin{pmatrix} \frac{D}{D+1} \\ 1 \end{pmatrix} \tag{6.3}$$

is a right inverse for $G(D)$. Now let us suppose that an information word $\boldsymbol{u}(D)$ is encoded using $G(D)$ as the encoder, resulting in the codeword $\boldsymbol{x}(D) = \boldsymbol{u}(D)G(D)$, which is transmitted over a noisy channel and received as $\boldsymbol{y}(D)$. Assume that $\boldsymbol{y}(D)$ is mis-decoded by the decoder as $\widehat{\boldsymbol{x}}(D)$, and that the codeword error is

$$\boldsymbol{e}_C(D) = \widehat{\boldsymbol{x}}(D) - \boldsymbol{x}(D) = (\, 1 + D + D^2 \quad 1 + D^2 \,),$$

which is one of the nonzero codewords of least weight in $\boldsymbol{C}$. Then by (6.2), the corresponding information error is

$$\begin{aligned} \boldsymbol{e}_I(D) &= \boldsymbol{e}_C(D)K(D) \\ &= (\, 1 + D + D^2 \quad 1 + D^2 \,) \begin{pmatrix} \frac{D}{D+1} \\ 1 \end{pmatrix} \\ &= \frac{1}{1+D} \\ &= 1 + D + D^2 + D^3 + D^4 + \cdots. \end{aligned}$$

Thus an error pattern of weight 5 in "code space" has propagated into an infinite number of errors in "information space," an event surely worthy of the name "catastrophe." In Theorem 6.3, below, we will see that the essential problem is that the inverse matrix $K(D)$ in (6.3) has infinite weight. ▪

In 1968 Massey and Sain [39] proved the following theorem, which allows us to tell whether a given *polynomial* generator matrix is catastrophic, or not.

**6.3 Theorem.** *If $G(D)$ is a polynomial generator matrix for a convolutional code, the following three conditions are equivalent. A polynomial generator matrix satisfying any one (and therefore all three) of these properties is called noncatastrophic. Otherwise, it is called catastrophic.*

(a) *No infinite-weight input $\boldsymbol{u}(D)$ can produce a finite weight output $\boldsymbol{x}(D) = \boldsymbol{u}(D)G(D)$.*

(b) *The* gcd *of the $k \times k$ minors of $G(D)$ is a power of $D$.*

(c) *$G(D)$ has a right finite weight (FW) inverse*.*

* This means that $G(D)$ has a right inverse $K(D)$, all of whose entries are of finite weight. In the standard parlance of system theory, this would be called a *finite impulse response* (FIR) inverse.

**Proof:** The proof is organized as follows: (a) $\rightarrow$ (b) $\rightarrow$ (c) $\rightarrow$ (a).

• (a) $\rightarrow$(b). We shall in fact prove the contrapositive, i.e., that if the gcd of the $k \times k$ minors is not a power of $D$, then there exists an infinite-weight input that produces a finite-weight output. Suppose then that $p(D)$ is an irreducible polynomial, not a power of $D$, that divides each of the $k \times k$ minors of $G(D)$. Let $\alpha$ be a zero of $p(D)$ in some extension field of $F$, i.e., $p(\alpha) = 0$, and consider the matrix $G(\alpha)$. Each of the $k \times k$ minors of $G(\alpha)$ is zero, so $\operatorname{rank} G(\alpha) < k$. Thus by a sequence of elementary row operations on $G(\alpha)$, we can produce a matrix $G'(\alpha)$ whose last row is all zeros. Alternatively, there is a $k \times k$ elementary matrix $U(\alpha)$ such that $G'(\alpha) = U(\alpha)G(\alpha)$. If now we replace each of the occurrences of $\alpha$ in this equation with the indeterminate $D$, we obtain

$$G'(D) = U(D)G(D),$$

where $U(D)$ is a unimodular $k \times k$ matrix and each of the entries in the last row of $G'(D)$ is divisible by $p(D)$. If now we define the $k$-dimensional row vector $\boldsymbol{u}'(D) = (0, 0, \ldots, 0, 1/p(D))$, it follows that $\boldsymbol{x}(D) = \boldsymbol{u}'(D)G'(D)$ is a vector all of whose entries are polynomials. Thus if we define $\boldsymbol{u}(D) = \boldsymbol{u}'(D)U(D)$, then $\boldsymbol{u}(D)G(D) = \boldsymbol{u}'(D)G'(D) = \boldsymbol{x}(D)$. Thus $\boldsymbol{u}(D)$ is an infinite input that produces a finite output.

• (b) $\rightarrow$(c). Let's denote the $k \times k$ submatrices of $G(D)$ by $G_\nu(D)$, and the corresponding minors (subdeterminants) by $\Delta_\nu(D)$, for $\nu = 1, \ldots, \binom{n}{k}$. If we denote the transposed cofactor matrix of $G_\nu(D)$ by $K_\nu(D)$, it follows from the Cofactor Theorem of linear algebra [2, Theorem 3.12] that $G_\nu(D)K_\nu(D) = \Delta_\nu(D)I_k$, for $\nu = 1, \ldots, \binom{n}{k}$. By placing $n - k$ appropriately chosen rows of zeros in $K_\nu(D)$, one then easily constructs an $n \times k$ polynomial matrix $K'_\nu(D)$ such that $G(D)K'_\nu(D) = \Delta_\nu I_k$, for all $\nu$.

Since we are assuming that $\gcd_\nu(\Delta_\nu(D)) = D^L$ for some $L \geq 0$, it follows from Euclid's algorithm [41, Section 8.4] that there exist polynomials $h_\nu(D)$ such that $\sum_\nu h_\nu(D)\Delta_\nu(D) = D^L$. Thus if we define

$$K'(D) = \sum_\nu h_\nu(D)K'_\nu(D),$$

it follows that

$$\begin{aligned} G(D)K'(D) &= \sum_\nu h_\nu(D)G(D)K'_\nu(D) \\ &= \sum_\nu h_\nu(D)\Delta_\nu(D)I_k \\ &= (\sum_\nu h_\nu(D)\Delta_\nu(D))I_k \\ &= D^L I_k. \end{aligned}$$

Thus the matrix $K(D) = D^{-L}K'(D)$ is an FW inverse for $G(D)$.

• (c) $\rightarrow$(a). Suppose that $K(D)$ is an FW inverse for $G(D)$, and suppose that $\boldsymbol{x}(D) = \boldsymbol{u}(D)G(D)$. Then multiplying both sides of this equation on the right by $K(D)$, we find that $\boldsymbol{x}(D)K(D) = \boldsymbol{u}(D)$. Thus if $\boldsymbol{x}(D)$ is finite, $\boldsymbol{u}(D)$ must be finite, too, since $K(D)$ is FW. ▪

The proof of the Massey-Sain theorem is important enough to merit some illustrative examples.

**6.4 Example:** To illustrate the (a) $\rightarrow$ (b) part of the proof, consider the generator matrix

$$G(D) = \begin{pmatrix} 1 & D & 1 \\ D & 1+D & 1+D^2 \end{pmatrix}. \tag{6.4}$$

If we denote the $2 \times 2$ minor of $G(D)$ corresponding to columns $i$ and $j$ of $G(D)$ by $\Delta_{ij}$, we have

$$\Delta_{12} = 1 + D + D^2, \quad \Delta_{13} = 1 + D + D^2, \quad \Delta_{23} = 1 + D^3,$$

so that $\gcd(\Delta_{12}, \Delta_{13}, \Delta_{23}) = 1 + D + D^2$. Since $1 + D + D^2$ is irreducible, we may take $p(D) = 1 + D + D^2$, which has a root $\alpha$ in $GF(2^2)$. Then substituting $\alpha$ for $D$ in eq. (6.4), we obtain

$$G(\alpha) = \begin{pmatrix} 1 & \alpha & 1 \\ \alpha & \alpha^2 & \alpha \end{pmatrix}.$$

We note that $G(\alpha)$, as predicted, has rank less than 2, and indeed if we multiply the first row of $G(\alpha)$ by $\alpha$ and add it to the second row, or alternatively multiply $G(\alpha)$ by $U(\alpha) = \begin{pmatrix} 1 & 0 \\ \alpha & 1 \end{pmatrix}$, we obtain

$$G'(\alpha) = U(\alpha)G(\alpha) = \begin{pmatrix} 1 & \alpha & 1 \\ 0 & 0 & 0 \end{pmatrix}. \tag{6.5}$$

Substituting $D$ for $\alpha$, in (6.5) and defining $G'(D) = U(D)G(D)$, we find that

$$G'(D) = \begin{pmatrix} 1 & 0 \\ D & 1 \end{pmatrix} G(D) = \begin{pmatrix} 1 & D & 1 \\ 0 & 1+D+D^2 & 1+D+D^2 \end{pmatrix}.$$

Each entry in the second row of $G'(D)$ is divisible by $1 + D + D^2$, so that

$$(0, \frac{1}{1+D+D^2})G'(D) = (0, 1, 1),$$

Thus if we define

$$\boldsymbol{u}(D) = (0, \frac{1}{1+D+D^2})U(D) = (\frac{D}{1+D+D^2}, \frac{1}{1+D+D^2}),$$

then we have $\boldsymbol{u}(D)G(D) = (0, 1, 1)$, i.e., $\boldsymbol{u}(D)$ is an infinite input that produces a finite output.∎

**6.5 Example:** To illustrate the (b) $\rightarrow$ (c) part of the proof, consider the generator matrix

$$G(D) = \begin{pmatrix} 1+D & D & 1+D \\ D & 1 & 1+D \end{pmatrix}.$$

The $2 \times 2$ minors of $G(D)$ are $\Delta_{12} = 1 + D + D^2$, $\Delta_{13} = 1 + D$, $\Delta_{23} = 1 + D^2$. We have $\gcd(\Delta_{12}, \Delta_{13}, \Delta_{23}) = 1$, and indeed $\Delta_{12} + D \cdot \Delta_{13} = 1$. The transposed cofactor matrices of $G_{12}$ and $G_{13}$ are

$$K_{12} = \mathrm{Cof}(G_{12})^T = \begin{pmatrix} 1 & D \\ D & 1+D \end{pmatrix}, \quad K_{13} = \mathrm{Cof}(G_{13})^T = \begin{pmatrix} 1+D & 1+D \\ D & 1+D \end{pmatrix},$$

and so if we define

$$K'_{12} = \begin{pmatrix} 1 & D \\ D & 1+D \\ 0 & 0 \end{pmatrix}, \quad K'_{13} = \begin{pmatrix} 1+D & 1+D \\ 0 & 0 \\ D & 1+D \end{pmatrix},$$

we have $G(D)K'_{12}(D) = (1 + D + D^2)I_2$, $G(D)K'_{13}(D) = (1 + D)I_2$. Finally then, if we define

$$K(D) = K'_{12}(D) + DK'_{13}(D) = \begin{pmatrix} 1+D+D^2 & D^2 \\ D & 1+D \\ D^2 & D+D^2 \end{pmatrix},$$

it follows that $G(D)K(D) = I_2$, i.e., $K(D)$ is an FW (polynomial, in this case) inverse for $G(D)$.∎

Note that the conditions of Theorem 6.3 are similar to, but not identical with, the conditions of Theorem A.1. Indeed, we see by comparing Theorem A.1 with Theorem 6.3 that every basic generator matrix is noncatastrophic. However, the converse is not true. For example, the matrix

$$G_7(D) = \begin{pmatrix} 1+D & 0 & 1 & D \\ 1 & D & 1+D & 0 \end{pmatrix}$$

in our long-running example is noncatastrophic but not basic, since the gcd of its $2 \times 2$ minors is $D$, whereas the matrix $H(D)$ defined by

$$H(D) = \begin{pmatrix} 0 & 1 \\ 0 & 0 \\ 0 & 0 \\ D^{-1} & 1+D^{-1} \end{pmatrix},$$

is an FW inverse for $G_7(D)$.

The Massey-Sain theorem does not apply directly to arbitrary generator matrices, i.e., those which may have non-polynomial entries, but it can easily be made to do so, by replacing condition ($b$), with a more general condition ($b'$).

**6.6 Theorem.** *If $G(D)$ is a rational generator matrix for a convolutional code, the following three conditions are equivalent. A generator matrix satisfying any one (and therefore all three) of these properties is called noncatastrophic. Otherwise, it is called catastrophic.*

(a) *No infinite-weight input $\boldsymbol{u}(D)$ can produce a finite weight output $\boldsymbol{x}(D) = \boldsymbol{u}(D)G(D)$.*

(b′) *Let $\beta(D)$ be the least common multiple of the denominators in $G(D)$, let $G'(D)$ be the matrix obtained from $G(D)$ by multiplying each entry by $\beta(D)$, and let $\alpha(D)$ be the* gcd *of the $k \times k$ minors of $G'(D)$. When the ratio $\alpha(D)/\beta(D)$ is reduced to lowest terms, say $\alpha'(D)/\beta'(D)$, then $\alpha'(D)$ is a power of $D$.*

(c) *$G(D)$ has a right FW inverse.*

We omit the proof of Theorem 6.6, but we will illustrate its use in the following example.

**6.7 Example:** Consider the following two generator matrices for the $(4, 2, 1, 4)$ code.

$$G_1(D) = \begin{pmatrix} \frac{1}{1+D+D^2} & 1 & \frac{1+D^2}{1+D+D^2} & \frac{1+D}{1+D+D^2} \\ 1 & \frac{1+D+D^2}{D} & D & \frac{1}{D} \end{pmatrix}$$

$$G_8(D) = \begin{pmatrix} 1 & 0 & \frac{1}{1+D} & \frac{D}{1+D} \\ 0 & 1 & \frac{D}{1+D} & \frac{1}{1+D} \end{pmatrix}$$

For $G_1$, the lcm of the denominators is $\beta_1(D) = D(1 + D + D^2)$, and the matrix obtained by multiplying each of the entries of $G_1(D)$ by $\beta_1(D)$ is

$$G_1'(D) = \begin{pmatrix} D & D(1+D+D^2) & D+D^3 & D+D^2 \\ D(1+D+D^2) & (1+D+D^2)^2 & D^2(1+D+D^2) & 1+D+D^2 \end{pmatrix}.$$

The gcd of the $2 \times 2$ minors of $G_1'(D)$ is $\alpha_1(D) = D(1+D+D^2)^2$, so that the ratio $\alpha_1/\beta_1$ is

$$\frac{\alpha_1(D)}{\beta_1(D)} = \frac{D(1+D+D^2)^2}{D(1+D+D^2)} = \frac{(1+D+D^2)}{1} = \frac{\alpha_1'(D)}{\beta_1'(D)}.$$

Thus $\alpha_1'(D) = (1 + D + D^2)$, which is not a power of $D$, so $G_1(D)$ is a catastrophic generator matrix. Indeed, it is easy to verify directly that $\boldsymbol{u}(D) = (1, D^3/(1 + D + D^2))$ is a "catastrophic input," since $\boldsymbol{u}(D)$ has infinite weight but

$$\boldsymbol{u}(D)G_1(D) = (1 + D, 1 + D^2, 1 + D + D^2, 1)$$

has finite weight.

For $G_8$, the lcm of the denominators is $\beta_8(D) = 1 + D$, and the matrix obtained by multiplying each of the entries of $G_8(D)$ by $\beta_8(D)$ is

$$G_8'(D) = \begin{pmatrix} 1+D & 0 & 1 & D \\ 0 & 1+D & D & 1 \end{pmatrix}.$$

The gcd of the $2 \times 2$ minors of $G_8'(D)$ is $\alpha_8(D) = 1 + D$, so that the ratio $\alpha_8/\beta_8$ is

$$\frac{\alpha_8(D)}{\beta_8(D)} = \frac{1+D}{1+D} = \frac{1}{1} = \frac{\alpha_8'(D)}{\beta_8'(D)}.$$

Thus $\alpha_1'(D) = 1$, which is a power of $D$, so $G_8(D)$ is a noncatastrophic generator matrix. A FW inverse for $G_8(D)$ is self-evident:

$$K(D) = \begin{pmatrix} 1 & 0 \\ 0 & 1 \\ 0 & 0 \\ 0 & 0 \end{pmatrix},$$

which shows that *any systematic generator matrix is noncatastrophic.* ▪

Finally, we illustrate Theorems 6.3 and 6.6 with Table 6.1, which tells which of the polynomial generator matrices $G_1(D)$ through $G_8(D)$ from Example 3.4, are catastrophic, and which are not. In this table, the quantity $\Delta$ is the gcd of the $2 \times 2$ minors of $G(D)$, if $G(D)$ is polynomial, and is $\alpha'(D)$ if $G(D)$ is not polynomial.

**Table 6.1. The "Catastrophicness" of $G_1(D), \ldots, G_8(D)$.**

| | $\Delta$ | Catastrophic? | FW inverse | catastrophic input |
|---|---|---|---|---|
| $G_1$ | $1 + D + D^2$ | Yes | (none) | $\begin{pmatrix} 1 & \frac{D^3}{1+D+D^2} \end{pmatrix}$ |
| $G_2$ | $1 + D + D^2$ | Yes | (none) | $\begin{pmatrix} \frac{D}{1+D+D^2} & \frac{1}{1+D+D^2} \end{pmatrix}$ |
| $G_3$ | 1 | No | $\begin{pmatrix} 1 & 0 & 0 & 0 \\ 1+D & 0 & 0 & 1 \end{pmatrix}^T$ | (none) |
| $G_4$ | 1 | No | $\begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}^T$ | (none) |
| $G_5$ | $1 + D + D^2$ | Yes | (none) | $\begin{pmatrix} \frac{D^2}{1+D+D^2} & \frac{1}{1+D+D^2} \end{pmatrix}$ |
| $G_6$ | 1 | No | $\begin{pmatrix} 1 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 \end{pmatrix}^T$ | (none) |
| $G_7$ | $D$ | No | $\begin{pmatrix} 0 & 0 & 0 & D^{-1} \\ 1 & 0 & 0 & 1+D^{-1} \end{pmatrix}^T$ | (none) |
| $G_8$ | 1 | No | $\begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \end{pmatrix}^T$ | (*none*) |

## 7. The Dual Code. Wyner-Ash Codes.

*Abstract: In this section we will define and study the dual code for a convolutional code. The main result here is that the degree of the dual codes is the same as the degree of the original code. As an application, we will focus on the class of $(n, n-1)$ codes, whose duals are $(n, 1)$ codes, and we will, in effect, give a complete classification of all such codes with $d_{\text{free}} = 3$ or $d_{\text{free}} = 4$. We will see that the Wyner-Ash codes, introduced in 1963, are the "best" possible $(n, n-1)$ codes with $d_{\text{free}} = 3$ or $d_{\text{free}} = 4$. The Wyner-Ash codes are thus the convolutional code analogs of the Hamming block codes.*

Recall that for an ordinary block code $\boldsymbol{C}$, the *dual code* $\boldsymbol{C}^\perp$ is defined to be the set of vectors which are orthogonal to every vector in $\boldsymbol{C}$. Since we have defined a convolutional code to be a block code over the rational field $F(D)$, no change in the definition of dual code is necessary for convolutional codes:

**7.1 Definition.** *If $\boldsymbol{C}$ is an $(n, k)$ convolutional code over the field $F$, its dual code, denoted by $\boldsymbol{C}^\perp$, is defined as follows:*

$$\boldsymbol{C}^\perp = \{\boldsymbol{y} \in F(D)^n : \boldsymbol{y} \cdot \boldsymbol{x} = 0 \text{ for all } \boldsymbol{x} \in \boldsymbol{C}.\}$$

By a standard result in linear algebra, it follows that $\boldsymbol{C}^\perp$ is an $(n, n-k)$ convolutional code over $F$. The following theorem, due to Forney [13], lies deeper.

**7.2 Theorem.** $\deg \boldsymbol{C} = \deg \boldsymbol{C}^\perp$, *i.e., if $\boldsymbol{C}$ is an $(n, k, m)$ convolutional code, then $\boldsymbol{C}^\perp$ is a $(n, n-k, m)$, code.*

**Proof:** Let $G$ be a basic $k \times n$ generator matrix for $\boldsymbol{C}$, and let $L$ be an $(n-k) \times n$ polynomial matrix such that $A = \begin{pmatrix} G \\ L \end{pmatrix}$ is unimodular. (Such an $L$ exists by part 6 of Theorem A.1). Now let $B = A^{-1}$, and write $B = (\, K^T \quad H^T \,)$, where $K$ is $k \times n$ and $H$ is $(n-k) \times n$. Both $K$ and $H$ are polynomial matrices, since $A$ is unimodular. Then since $AB = I_n$, it follows that

$$AB = \begin{pmatrix} G \\ L \end{pmatrix} (\, K^T \quad H^T \,) = \begin{pmatrix} GK^T & GH^T \\ LK^T & LH^T \end{pmatrix} = I_n = \begin{pmatrix} I_k & 0_{k \times n-k} \\ 0_{n-k \times k} & I_{n-k} \end{pmatrix}.$$

From this we see that $GH^T = 0_{k \times n-k}$, so that the $n-k$ rows of $H$ are each orthogonal to the $k$ rows of $G$. Furthermore, since $H$ is a submatrix of the nonsingular matrix $B$, $H$ has rank $n-k$, so that $H$ is a generator matrix for the dual code $\boldsymbol{C}^\perp$. But even more is true: since $B$ is unimodular, it follows from Theorem A.1, part 6, that *$H$ is a basic generator matrix for $\boldsymbol{C}^\perp$.*

Furthermore, by the "minors of the inverse" theorem [20], [25], the $k \times k$ minors of $G$ are, except for scalar factors, identical to the $(n-k) \times (n-k)$ minors of $H$, so that in particular, the internal degree of $G$ (the maximum degree of any $k \times k$ minor of $G$) is the same as the internal degree of $H$ (the maximum degree of any $(n-k) \times (n-k)$ minor of $H$.) But by Theorem 3.8, the internal degree of a basic generator matrix for a convolutional code equals the degree of the code. ▪

**7.3 Corollary.** *If $\boldsymbol{C}$ is an $(n, n-1, m)$ convolutional code, with basic generator matrix $G(D)$, and if the $(n-1)\times(n-1)$ minors of $G$ are $\Delta_1, \Delta_2, \ldots, \Delta_n$ (here $\Delta_i$ is the determinant of $G$ with its $i$th column removed), the $1 \times n$ matrix*

$$H = (\, -\Delta_1 \quad +\Delta_2 \quad \cdots \quad (-1)^n \Delta_n \,) \tag{7.1}$$

*is a canonical generator matrix for $\boldsymbol{C}^\perp$.*

**Proof:** According to the proof of Theorem 7.2, if $G(D)$ is completed to an $n\times n$ unimodular matrix $A$ by adding an $n$th row of polynomials to $G$, and if $B = A^{-1}$, then the $n$th column of $B$ (when transposed) will be a basic generator matrix for $\boldsymbol{C}^\perp$. However, since $\det A$ is a scalar, the inverse of $A$ is, by the "cofactor theorem" of linear algebra [2, Theorem 3.12], a scalar times the transpose of the cofactor matrix of $A$. Thus the $n$th column of $B$ is a scalar times $(\, -\Delta_1 \quad +\Delta_2 \quad \cdots \quad (-1)^n \Delta_n \,)^T$. This proves that $H$, as defined in eq. (7.1) is a basic generator matrix for $\boldsymbol{C}^\perp$. Since however the dimension of $\boldsymbol{C}^\perp$ is one, any basic generator matrix is also canonical. ▪

In analogy with the situation for block codes, we now define a *parity-check* matrix for a convolutional code.

**7.4 Definition.** *If $\boldsymbol{C}$ is an $(n, k, m)$ convolutional code, and if $H(D)$ is a generator matrix for $\boldsymbol{C}^\perp$, $H(D)$ is called a parity-check matrix for $\boldsymbol{C}$. Alternatively, $H(D)$ is an $(n-k)\times n$ matrix over $F(D)$ of rank $n-k$ whose rows are orthogonal to each of the codewords in $\boldsymbol{C}$.*

**7.5 Example:** Consider the $(4, 2, 1)$ code from Example 3.4. Since

$$G_8(D) = \begin{pmatrix} 1 & 0 & 1/(1+D) & D/(1+D) \\ 0 & 1 & D/(1+D) & 1/(1+D) \end{pmatrix}$$

is a systematic generator matrix of the form $(\, I_k \quad A \,)$, it follows from the general theory of block codes that the matrix $(\, -A^T \quad I_{n-k} \,)$ is a generator matrix for the dual code. Thus

$$H_1 = \begin{pmatrix} \frac{1}{1+D} & \frac{D}{1+D} & 1 & 0 \\ \frac{D}{1+D} & \frac{1}{1+D} & 0 & 1 \end{pmatrix}$$

is a systematic generator matrix for the dual code. If we want a *polynomial* generator matrix, we can multiply each row of $H_1$ by $1+D$, thus obtaining

$$H_2 = \begin{pmatrix} 1 & D & 1+D & 0 \\ D & 1 & 0 & 1+D \end{pmatrix}.$$

However, $H_2$ is not basic. We can make it basic by adding row 2 to row 1, and then dividing row 1 by $1+D$, obtaining

$$H_3 = \begin{pmatrix} 1 & 1 & 1 & 1 \\ D & 1 & 0 & 1+D \end{pmatrix},$$

which is not only basic, but also canonical. The close similarity between $H_3$ and the canonical generator matrix $G_6$ for the primal code tells us that the code is essentially self-dual.▪

**7.6 Example:** Consider the following two canonical generator matrices for two different $(3, 2, 2)$ convolutional codes:

$$G_1 = \begin{pmatrix} D & 1 & D \\ 1 & 1+D & 1+D \end{pmatrix}$$
$$G_2 = \begin{pmatrix} 0 & 1 & 1 \\ 1+D+D^2 & 1+D^2 & 0 \end{pmatrix}$$

The first of these codes, call it $\boldsymbol{C}_1$, is a compact code with Forney indices $(1, 1)$, while the second is noncompact and has Forney indices $(0, 2)$. The free distances are 3 and 2, respectively.

We can compute the dual codes for these two codes by using Corollary 7.3. For $\boldsymbol{C}_1$, the $2 \times 2$ minors are

$$\Delta_1 = 1 + D^2, \Delta_2 = D^2, \Delta_3 = 1 + D + D^2.$$

On the other hand, for $\boldsymbol{C}_2$, the minors are

$$\Delta_1 = 1 + D^2, \Delta_2 = 1 + D + D^2, \Delta_3 = 1 + D + D^2.$$

Thus by Corollary 7.3, we find the following generator matrices for the dual codes $\boldsymbol{C}_1^\perp$ and $\boldsymbol{C}_2^\perp$:

$$H_1 = (\, 1 + D^2 \quad D^2 \quad 1 + D + D^2 \,)$$
$$H_2 = (\, 1 + D^2 \quad 1 + D + D^2 \quad 1 + D + D^2 \,).$$

Thus as predicted by Theorem 7.2, both dual codes are $(3, 1, 2)$ codes. But in this simple case, both dual codes also have the same Forney index, viz., (2). Thus $\boldsymbol{C}_1^\perp$ and $\boldsymbol{C}_2^\perp$ are different codes with the *same* Forney indices, whose dual codes (the original codes $\boldsymbol{C}_1$ and $\boldsymbol{C}_2$) have *different* Forney indices. Thus while $\deg \boldsymbol{C}^\perp$ can be predicted from $\deg \boldsymbol{C}$, the Forney indices of $\boldsymbol{C}^\perp$ cannot be predicted from those of $\boldsymbol{C}$. ▪

The following theorem gives useful partial information about the Forney indices of a $(n, n-1, m)$ convolutional code, in terms of its parity-check matrix. It shows explicitly that the structure of the Forney indices of a code depends on much more than simply the Forney indices of the dual code.

**7.7 Theorem.** *Let $\boldsymbol{C}$ be a $(n, n-1, m)$ convolutional code with basic parity-check matrix*

$$H = (\, h_1(D) \quad \cdots \quad h_n(D) \,),$$

*where $\max_i \deg h_i(D) = m$. Let $r$ denote the dimension of the $F$-vector space spanned by the $n$ polynomials $h_1(D), \ldots, h_n(D)$, i.e., the set of polynomials of the form $\alpha_1 h_1(D) +$*

$\cdots + \alpha_n h_n(D)$, *where* $\alpha_1, \ldots, \alpha_n \in F$. *Then the number of nonzero Forney indices for* $\boldsymbol{C}$ *is* $r-1$.

**Proof:** As in Section 4, let $\delta_0$ denote the dimension of the subcode of $\boldsymbol{C}$ consisting of the codewords of dimension zero, i.e., the set of $F$-vectors $(x_1, \ldots, x_n)$ such that $\sum_{i=1}^{n} x_i h_i(D) = 0$. Then by the "dimension = rank + nullity" theorem [2, Theorem 2.3], it follows that $n = r + \delta_0$, i.e., $\delta_0 = n - r$. But according to Corollary 4.2, $\delta_0$ is the number of Forney indices which are equal to zero. Since the total number of Forney indices for a $(n, n-1, m)$ code is $n-1$, it follows that the number of nonzero Forney indices is $(n-1) - (n-r) = r-1$. ▪

**7.8 Example:** Let $m \geq 2$, and consider the $(2^m, 2^m - 1, m)$ convolutional code over $GF(2)$ defined by the parity-check matrix

$$H_m = (\, 1 \quad 1+D \quad 1+D^2 \quad \cdots \quad 1+D+\cdots+D^m \,), \tag{7.2}$$

where the entries in $H_m$ are the $2^m$ distinct polynomials of degree $\leq m$ with constant term 1. We assert that this code has $d_{\text{free}} = 3$. To see this, note that $\boldsymbol{C}$ has no words of weight 1, since no entry of $H_m$ is zero; and it has no words of weight 2, since there can be no identity of the form $D^{e_1} h_i(D) + D^{e_2} h_j(D) = 0$, where $h_i(D)$ and $h_j(D)$ are the $i$th and $j$th components of $H_m$. On the other hand, since, e.g., $[1, D, 1] \cdot [1, 1+D, 1+D+D^2] = 0$, $\boldsymbol{C}$ has words of weight 3, so $d_{\text{free}} = 3$ as asserted. ▪

The $(2^m, 2^m - 1, m, 3)$ code defined by eq. (7.2) is called the $m$th order *Wyner-Ash* code, in honor of its discoverers [59]. The next theorem identifies the Forney indices of the Wyner-Ash codes.

**7.9 Theorem.** *The* $(2^m, 2^m - 1, m, 3)$ $m$*th order Wyner-Ash code has Forney indices* $(0^{2^m - m - 1}, 1^m)$, *and so it is a compact code.*

**Proof:** We use Theorem 7.7. The set of polynomials in the parity-check matrix (7.2) spans a vector space of dimension $m+1$, viz., the set of all polynomials of degree $\leq m$, so that, by Theorem 7.7, the number of nonzero Forney indices is $m$. But since the degree of the $m$th order Wyner-Ash code is $m$, and the sum of the Forney indices is the degree, it follows that each of the nonzero Forney indices must be equal to one. ▪

The next theorem establishes an important optimality property of W-A codes.

**7.10 Theorem.** *Any* $(n, n-1, m)$ *code with* $d_{\text{free}} \geq 3$ *must have* $n \leq 2^m$; *if* $n = 2^m$, *the code must have* $d_{\text{free}} = 3$, *and apart from possible "delays" in the codeword components, must be the Wyner-Ash code.*

**Proof:** Suppose $\boldsymbol{C}$ is a $(n, n-1, m)$ code with $d_{\text{free}} \geq 3$, and let

$$H = (\, h_1(D) \quad h_2(D) \quad \cdots \quad h_n(D) \,)$$

be a canonical parity-check matrix for $\boldsymbol{C}^{\perp}$. Then by Theorem 7.2, $\max_i \deg h_i(D) = m$. Now since $d_{\text{free}} \geq 3$, no $h_i(D)$ can be 0 (else $d_{\text{free}} = 1$); also no pair $(h_i(D), h_j(D))$ can

satisfy $D^e h_i = h_j$ (else $d_{\text{free}} = 2$). Thus for each nonzero polynomial $h(D)$ of degree $\leq m$, at most one polynomial in the "equivalence class" $\{D^e h(D) : e = 0, \pm 1, \pm 2, \ldots\}$ can be included among the $h_i(D)$'s. But since each equivalence class contains exactly one polynomial of least degree, namely the one with constant term 1, the number of equivalence classes equals the number of polynomials of degree at most $m$ with constant term 1, which is $2^m$. Thus $n \leq 2^m$, as asserted. ▪

**7.11 Example:** Now suppose that $m \geq 3$, and consider the binary $(2^{m-1}, 2^{m-1} - 1, m)$ code defined by the parity-check matrix

$$H'_m = (\, 1 + D^m \quad \cdots \quad 1 + D + \cdots + D^m \,), \tag{7.3}$$

where the entries in $H'_m$ are the $2^{m-1}$ polynomials of the form $1 + a_1 D + \cdots + a_{m-1} D^{m-1} + D^m$, $a_i = 0$ or 1. This code has $d_{\text{free}} = 4$. To see this, note that there can be no words of weight 1, since no entry of $H'_m$ is zero; there are no words of weight 2, since no two entries of $H'_m$ differ only by a power of $D$. If there were a word of weight three, such a word would imply the existence of an equation of the form

$$D^{e_1} h_1(D) + D^{e_2} h_2(D) + D^{e_3} h_3(D) = 0, \tag{7.4}$$

with $e_1 \leq e_2 \leq e_3$, where $h_1(D)$, $h_2(D)$, and $h_3(D)$ are three (not necessarily distinct) entries from $H'_m$. But then $e_1 = e_2$, in order that the lowest degree terms in the sum in (7.4) cancel, and $e_2 = e_3$, in order that the highest degree terms cancel. Thus if (7.4) holds, then in fact

$$h_1(D) + h_2(D) + h_3(D) = 0,$$

which is clearly not possible, since e.g. each of the $h_i(D)$ has constant term 1. Thus $d_{\text{free}} \geq 4$. On the other hand, the free distance of this code is not more than 4, since, for example, the equation

$$(1 + D^m) + (1 + D + D^m) + (1 + D^2 + D^m) + (1 + D + D^2 + D^m) = 0$$

represents a sum of four columns of $H'_m$ equal to zero, if $m \geq 3$. We call the code with parity-check matrix given by (7.3) the *extended Wyner-Ash code of order m.* ▪

The following theorem shows that the extended Wyner-Ash codes are noncompact, despite being optimal.

**7.12 Theorem.** *The $(2^{m-1}, 2^{m-1} - 1, m, 4)$ mth order extended Wyner-Ash code has Forney indices $(0^{2^{m-1}-m}, 1^{m-2}, 2)$, and so it is not a compact code.*

**Proof:** The set of polynomials in the parity-check matrix (7.3) spans a binary vector space of dimension $m$, viz., the set of polynomials of the form $p_0 + p_1 D + \cdots + p_m D^m$ with $p_0 = p_m$. Thus, by Theorem 7.7, the number of nonzero Forney indices for the $m$th order extended Wyner-Ash code is $m - 1$. But since the degree of the code is $m$, and the sum of the Forney indices is equal to the degree, it follows that among the nonzero Forney indices, one must be equal to two and the rest must be equal to one. ▪

**7.13 Theorem.** *Any $(n, n-1, m)$ code with $d_{\text{free}} \geq 4$ must have $n \leq 2^{m-1}$.*

**Proof:** Suppose on the contrary that $\boldsymbol{C}$ is a $(n, n-1, m)$ code with $d_{\text{free}} \geq 4$, with $n \geq 2^{m-1}+1$, and let

$$H = (\, h_1(D) \quad h_2(D) \quad \cdots \quad h_n(D)\,)$$

be a parity-check matrix for $\boldsymbol{C}$. If some $h_i(D)$ is divisible by $D$, then dividing $h_i(D)$ by $D$ will not change $d_{\text{free}}$, so without loss of generality we can assume each $h_i(D)$ has the form

$$h_i(D) = 1 + h_{i,1}D + \cdots + h_{i,m}D^m.$$

Now consider the $n-1$ polynomials

$$k_i(D) = D^{-e_i}(h_1(D) + h_i(D)) \quad \text{for } i = 2, \ldots, n,$$

where $e_i \geq 1$ is the largest integer such that $D^{e_i}$ divides $h_1(D) + h_i(D)$. These are $n-1 \geq 2^{m-1}$ distinct polynomials of degree less than $m$, with constant term 1. Similarly, the polynomials $h_i(D)$ are $n \geq 2^{m-1}+1$ distinct polynomials of degree $\leq m$, with constant term 1. Together, then, the $h_i$'s and the $k_i$'s account for at least $2^{m-1}+2^{m-1}+1 = 2^m+1$ polynomials of degree $\leq m$ with constant term 1. But since there are only $2^m$ total polynomials of this form, it follows that $k_j(D) = h_i(D)$ for some $i$ and $j$, which implies an equation of the form $D^{-e_j}(h_1 + h_j) = h_i$ , i.e., $h_1(D) + h_j(D) = D^{e_j}h_i(D)$, which, being a linear dependence among three columns of $H$, implies that $d_{\text{free}} \leq 3$, a contradiction. ▪

Theorems 7.10 and 7.13 are due to Ytrehus [61][62]. We might summarize them by saying that the Wyner-Ash codes and the extended Wyner-Ash codes are *length optimal*. More generally, Ytrehus shows that there exists a $(n, n-r, m, 3)$ convolutional code with $n = (2^r - 1)2^m$, for any $m \geq 1$ and $r \geq 1$, and that if $\boldsymbol{C}$ is a $(n, n-r, m)$ convolutional code with $d_{\text{free}} \geq 3$, then $n \leq (2^r-1)2^m$. Similarly, he constructs a $(n, n-r, m, 4)$ code with $n = 2^{m+r-1}$ for any $m \geq 1$, $r \geq 2$, and shows that any $(n, n-r, m, 4)$ code with $r \geq 2$ must have $n \leq 2^{m+r-1}$.

Our last theorem about Wyner-Ash codes states that they are *distance optimal*, i.e., have the greatest possible $d_{\text{free}}$ for their $(n, k, m)$ paramenters.

**7.14 Theorem.** *The $(2^m, 2^m-1, m, 3)$ Wyner-Ash codes, and the $(2^{m-1}, 2^{m-1}-1, m, 4)$ extended Wyner-Ash codes, are distance-optimal.*

**Proof:** . The $(2^m, 2^m-1, m, 3)$ Wyner-Ash code is optimal, since by Theorem 7.13, any $(n, n-1, m, 4)$ code must have $n \leq 2^{m-1}$. The $(2^{m-1}, 2^{m-1}-1, m, 4)$ extended Wyner-Ash code is optimal, since by Corollary 4.5 (with $L = 0$), any $(2^{m-1}, 2^{m-1}-1, m)$ code must have $d_{\text{free}} \leq \Delta(2^{m-1}, 2^{m-1}-1-m)$, and it is easy to show that this last value is 4. ▪

Let us denote by $f_d(m)$ the largest integer $n$ such that there exists an $(n, n-1, m)$ convolutional code with $d_{\text{free}} \geq d$. Then Theorem 7.10 says that $f_3(m) = 2^m$, for $m \geq 2$, and Theorem 7.13 says that $f_4(m) = 2^{m-1}$, for $m \geq 3$. If would interesting to know the

value of $f_d(m)$ for fixed $d$ and large $m$ for $d \geq 5$. For example, using Corollary 4.5, with $L = 1$, it is possible to show that

$$f_5(m) \leq 2^{(m+1)/2},$$

but we do not know how accurate this estimate may be.

## 8. Puncturing Convolutional Codes.

*Abstract: In this section we will introduce a simple method for constructing new convolutional codes from old ones, via puncturing. This method has enjoyed widespread success because the best codes constructed by puncturing are typically as powerful as other codes with the same parameters, but are considerably easier to implement than "nonpunctured" codes.*

We begin with an illustrative example. Consider the encoder for the $(2, 1, 2)$ convolutional code with generator matrix $G(D) = (1 + D^2, 1 + D + D^2)$ shown in Figure 8.1. We can describe the encoding process as follows.

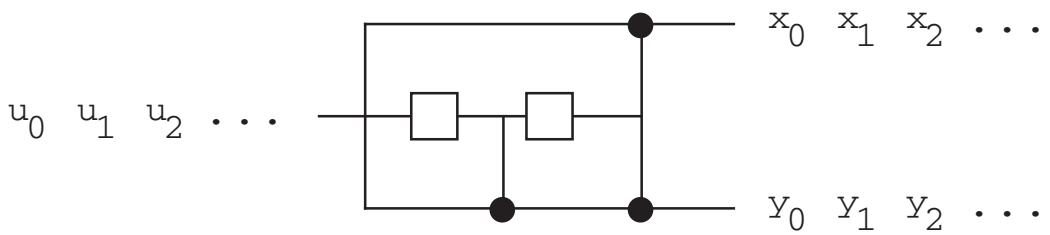


**Figure 8.1.** A Physical Encoder for the $(2, 1, 2, 5)$ code with $G(D) = (1 + D^2, 1 + D + D^2)$.

At the $i$th "tick" of an imaginary clock, the encoder accepts one input bit ($u_i$), and in response, produces two output bits ($x_i$ and $y_i$). But this view is somewhat arbitrary. We could equally well imagine a clock that ticks only half as fast, in which case the encoder would accept *two* input bits ($u_{2i}$ and $u_{2i+1}$) at the $i$th tick, and in response produce *four* outputs $(x_{2i}, x_{2i+1}, y_{2i}, y_{2i+1})$. In other words, by "blocking" the input sequence into blocks of 2 bits each, the original $(2, 1)$ code has become a $(4, 2)$ code. Of course, this change of viewpoint doesn't really change the code in any significant way, but bear with us. With the "blocked" code still in hand, let us delete, or "puncture," the bit $x_{2i+1}$ from each four-bit output block, so that for every tick, the encoder accepts two input bits ($u_{2i}$ and $u_{2i+1}$), and in response produces *three* output bits $(x_{2i}, y_{2i}, y_{2i+1})$. We can summarize the process of blocking and puncturing in this example with the following display:

$$\begin{pmatrix} x_0 & x_1 & x_2 & x_3 & x_4 & x_5 \cdots \\ y_0 & y_1 & y_2 & y_3 & y_4 & y_5 \cdots \end{pmatrix} \Longrightarrow \begin{pmatrix} x_0 & & x_2 & & x_4 & \cdots \\ y_0 & y_1 & y_2 & y_3 & y_4 & y_5 \cdots \end{pmatrix}.$$

Notice the periodic pattern of the output, which can be summarized by the $2 \times 2$ "puncturing matrix"

$$P = \begin{pmatrix} 1 & 0 \\ 1 & 1 \end{pmatrix}.$$

In this way the original $(2, 1)$ code, after being *punctured with pattern* $P$, has become a $(3, 2)$ code. It is not hard to see that the free distance of this new code is $d_{\text{free}} = 3$, and as

we will see below the degree of the new code is $m = 2$, the same as the parent code. This "punctured" $(3, 2, 2, 3)$ convolutional code is, by Theorem 7.10, optimal. Furthermore, because it is a descendant of a simple parent, it naturally inherits a simple structure that makes it easier to decode with Viterbi's algorithm than an arbitrary $(3, 2, 2)$ code. (For more on the decoding of punctured convolutional codes, see the article by Cain, Clark, and Geist [7], who introduced the notion of puncturing.)

In general, puncturing can be described as follows. We begin with a "parent" $(n, k, m)$ convolutional code[1], and block it to depth $M$, i.e., group the input bit stream into blocks of $M$ bits each. The result, as is intuitively plausible and as we shall show formally below (Theorem 8.6), is an $(nM, kM, m)$ convolutional code with the same free distance as the parent code. Now let $P$ be a $n \times M$ 0-1 "puncturing" matrix, with a total of $N$ 1's and $nM - N$ 0's. If we now delete, or "puncture," all but $N$ bits from each $nM$-bit output block, using the matrix $P$ as a template, the result is an $(N, kM, m)$ convolutional code[2]. In many cases, this punctured code will turn out to be "good" in the sense of having a relatively large free distance, or even *optimal* in the sense of having the largest possible free distance, for the parameters $(N, kM, m)$. This is surprising, since the class of punctured codes is a small subset of the class of unrestricted convolutional codes, and it is useful for two reasons. First, searching for a good punctured code with a given set of parameters is much simpler than searching for an arbitrary code, and second, the complexity of implementing Viterbi's maximum-likelihood decoding algorithm on an $(N, kM, m)$ punctured convolutional code is less than that for an arbitrary $(N, kM, m)$ code by a factor of $2^{M-1}$. (From a theorist's viewpoint, however, the subject is somewhat unsatisfactory because in general, the choice of the parent code and the "puncturing pattern" must be found by trial and error via a computer search.)

In the remainder of this section, we will formally study the method of puncturing. Our main goal will be to see how to produce a generator matrix for a punctured code, given a generator matrix $G(D)$ for the parent code and a puncturing matrix $P$. We will give several examples, but the interested reader should consult the literature for extensive tables of good punctured codes [4][7][8][9][21][60], especially [9, Table A.8].

We begin by returning to the simple *blocking* idea, which, as we have seen, is a method of taking a given $(n, k)$ convolutional code $C$ and from it producing an infinite sequence of codes, with parameters $(2n, 2k), (3n, 3k), \ldots$ . The derived "blocked" codes do not differ from the original code in any substantive way, but still it is interesting to ask the following question: if we start with an $(n, k)$ convolutional code with (canonical) generator matrix $G(D)$, and block it to depth $M$, what is a corresponding (canonical) generator matrix, say $G^{[M]}$, for the resulting $(nM, kM)$ code? For example, what is a canonical generator

[1] In most practical applications, the parent code has $k = 1$, but it is not much more difficult to treat the general case, and we shall do so.

[2] In fact, the degree of the punctured code may be less than $m$, but for most interesting punctured codes no memory reduction will take place.

matrix for the $(4,2)$ code obtained from the $M = 2$ blocking of the code in Figure 8.1? One answer turns out to be

$$G^{[2]}(D) = \begin{pmatrix} 1+D & 1+D & 0 & 1 \\ 0 & D & 1+D & 1+D \end{pmatrix}.$$

To see why this is so, we reason as follows. If the input sequence is $\boldsymbol{u}(D) = u_0 + u_1 D + u_2 D^2 + \cdots$, the resulting codeword $\boldsymbol{x}(D) = \boldsymbol{u}(D)G(D)$ can be written as

$$\boldsymbol{x}(D) = u_0 G(D) + u_1 DG(D) + u_2 D^2 G(D) + \cdots.$$

Thus the code itself*, i.e., the set of all possible codewords, is the set of all binary linear combinations of the rows of the matrix $\overline{G}(D)$ defined by

$$\overline{G}(D) = \begin{pmatrix} G(D) \\ DG(D) \\ D^2 G(D) \\ D^3 G(D) \\ \vdots \end{pmatrix}. \tag{8.1}$$

Now let us write the generator matrix $G(D) = (1 + D^2, 1 + D + D^2)$ as

$$G(D) = (1,1) + (0,1)D + (1,1)D^2. \tag{8.2}$$

We call (8.2) the *polynomial decomposition* of $G(D)$. Combining (8.2) with (8.1), we obtain

$$\overline{G}(D) = \begin{pmatrix} (1,1) & (0,1)D & (1,1)D^2 & & & \\ & (1,1)D & (0,1)D^2 & (1,1)D^3 & & \\ & & (1,1)D^2 & (0,1)D^3 & (1,1)D^4 & \\ & \vdots & & & & \ddots \end{pmatrix} \tag{8.3}$$

We can simplify the notation a little by indexing the rows and columns of $\overline{G}(D)$ with appropriate powers of $D$, to indicate the "timing" of the encoding process with respect to an imaginary clock:

$$\overline{G}(D) = \begin{array}{c} \\ 1 \\ D \\ D^2 \\ D^3 \\ D^4 \end{array} \begin{array}{c} \begin{array}{ccccccc} 1 & D & D^2 & D^3 & D^4 & D^5 & \cdots \end{array} \\ \begin{pmatrix} 11 & 01 & 11 & & & & \\ & 11 & 01 & 11 & & & \\ & & 11 & 01 & 11 & & \\ & & & 11 & 01 & 11 & \\ & & & & \ddots & & \end{pmatrix} \end{array} \tag{8.4}$$

* Technically, the set of *causal codewords* of the code.

In this notation, the single row indexed by $D^i$ corresponds to the encoder input at the $i$th clock tick, and the two columns indexed by $D^i$ correspond to the encoder output at the $i$th tick. Now if we "run the clock at half speed", so to speak, the matrix $\overline{G}(D)$ can be written as

$$\overline{G}(D) = \begin{array}{c} \\ 1 \\ 1 \\ D \\ D \\ D^2 \end{array} \begin{array}{c} \begin{array}{cccccc} 1 & 1 & D & D & D^2 & \cdots \end{array} \\ \begin{pmatrix} 11 & 01 & 11 & & & \\ & 11 & 01 & 11 & & \\ & & 11 & 01 & 11 & \\ & & & 11 & 01 & 11 \\ & & & & \ddots & \end{pmatrix} \end{array}. \tag{8.5}$$

In (8.5), the *two* rows indexed by $D^i$ correspond to the encoder *inputs* at the $i$th tick, and the *four* columns indexed by $D^i$ correspond to the encoder *outputs* at the $i$th tick. Of course, eq. (8.5) represents no substantive change from (8.4), since only the row and column labels have changed. But in this "half-speed" format, the two rows of $\overline{G}(D)$ indexed by "1" form the $2 \times 4$ polynomial matrix $G^{[2]}(D)$, given by

$$G^{[2]}(D) = \begin{array}{c} \\ 1 \\ 1 \end{array} \begin{array}{c} \begin{array}{cccc} 1 & 1 & D & D \end{array} \\ \begin{pmatrix} 11 & 01 & 11 & \\ & 11 & 01 & 11 \end{pmatrix} \end{array}$$

$$= \begin{pmatrix} 1+D & 1+D & 0 & 1 \\ 0 & D & 1+D & 1+D \end{pmatrix} \tag{8.6}$$

And hence (see (8.1)) $\overline{G}(D)$ can be expressed as

$$\overline{G}(D) = \begin{pmatrix} G^{[2]}(D) \\ DG^{[2]}(D) \\ D^2G^{[2]}(D) \\ \vdots \end{pmatrix}.$$

The conclusion is that $G^{[2]}(D)$ is a generator matrix for the $M = 2$ blocking of the original code, as depicted in Figure 1. In fact, it is easy to check that $G^{[2]}(D)$ is a canonical generator matrix. Similarly, if we want to find the generator matrix for the $M = 3$ blocking, we need only take the first three rows of $\overline{G}(D)$, as follows:

$$G^{[3]}(D) = \begin{array}{c} \\ 1 \\ 1 \\ 1 \end{array} \begin{array}{c} \begin{array}{ccccc} 1 & 1 & 1 & D & D \end{array} \\ \begin{pmatrix} 11 & 01 & 11 & & \\ & 11 & 01 & 11 & \\ & & 11 & 01 & 11 \end{pmatrix} \end{array}$$

$$= \begin{pmatrix} 1 & 1 & 0 & 1 & 1 & 1 \\ D & D & 1 & 1 & 0 & 1 \\ 0 & D & D & D & 1 & 1 \end{pmatrix} \tag{8.7}$$

It follows that $G^{[3]}(D)$ is a generator matrix for the $M = 3$ blocking of the original code; it too turns out to be canonical.

This procedure can be continued indefinitely, in a straightforward way. But in order to give an algebraic description of the general generator matrix $G^{[M]}(D)$ of the $M$th blocking of an arbitrary $k \times n$ polynomial matrix $G(D)$, we need to introduce the notion of the *polyphase decomposition* of a polynomial, or power series.

**8.1 Definition.** *If*

$$f(D) = a_0 + a_1 D + a_2 D^2 + \cdots$$

*is a power series (or possibly just a polynomial) in the indeterminate $D$, then for any integer $M \geq 1$, the $M$th polyphase decomposition of $f(D)$ is the following list of $M$ power series:*

$$\begin{aligned} &(f_{0,M}(D), f_{1,M}(D), \ldots, f_{M-1,M}(D)) \\ &\stackrel{\text{def}}{=} \left( \sum_{k \geq 0} a_{kM} D^k,\ \sum_{k \geq 0} a_{kM+1} D^k,\ \ldots,\ \sum_{k \geq 0} a_{kM+(M-1)} D^k \right) \end{aligned} \tag{8.8}$$

*The $j$th component of the list in (8.8) is called the $(j, M)$th polyphase component of $f(D)$.*

The notation in (8.8) may obscure the simple underlying idea, which is that the $(j, M)$th polyphase component of $f(D)$ is the power series whose coefficients are obtained from those of of $f(D)$ by starting with the $j$th coefficient, and taking every $M$th term. For example, if

$$f(D) = 3 + D + 4D^2 + D^3 + 5D^4 + 9D^5 + 2D^6 + 6D^7, \tag{8.9}$$

then the third polyphase decomposition of $f(D)$ is

$$\left(3 + D + 2D^2, 1 + 5D + 6D^2, 4 + 9D\right). \tag{8.10}$$

The components of the $M$th polyphase decomposition can be used to form an important matrix, called the $M$th *polycyclic pseudocirculant* (or PCPC for short) associated with $f(D)$.

**8.2 Definition.** *If*

$$f(D) = a_0 + a_1 D + a_2 D^2 + \cdots$$

*is a power series (or possibly just a polynomial) in the indeterminate $D$, then for any integer $M \geq 1$, if $(f_0, f_1, \ldots, f_{M-1})$ is the $M$th polyphase decomposition of $f(D)$, the $M$th polycyclic pseudocirculant associated with $f(D)$ is the $M \times M$ polynomial matrix $f^{[M]}(D)$ defined as follows:*

$$f^{[M]}(D) = \begin{pmatrix} f_0 & f_1 & \cdots & f_{M-1} \\ Df_{M-1} & f_0 & \cdots & f_{M-2} \\ Df_{M-2} & Df_{M-1} & \cdots & f_{M-3} \\ \vdots & \vdots & \ddots & \vdots \\ Df_1 & Df_2 & \cdots & f_0 \end{pmatrix}$$

For example, if $f(D)$ is as given in (8.9), then (cf. (8.10)), the third PCPC associated with $f(D)$ is

$$f^{[3]}(D) = \begin{pmatrix} 3+D+2D^2 & 1+5D+6D^2 & 4+9D \\ 4D+9D^2 & 3+D+2D^2 & 1+5D+6D^2 \\ D+5D^2+6D^3 & 4D+9D^2 & 3+D+2D^2 \end{pmatrix}$$

We now return to the problem of finding a generator matrix for a blocked convolutional code, again using the simple $(2,1,2,5)$ code with generator matrix $G(D) = (g_1(D), g_2(D)) = (1+D^2, 1+D+D^2)$ to illustrate the general ideas. Let $M = 2$. Then the second polyphase decomposition of $g_1(D) = 1+D^2$ is $(1+D, 0)$, and so the second PCPC associated with $g_1(D) = 1+D^2$ is

$$g_1^{[2]}(D) = \begin{pmatrix} 1+D & 0 \\ 0 & 1+D \end{pmatrix} \tag{8.11}$$

Similarly, the second polyphase decomposition of $g_2(D) = 1+D+D^2$ is $(1+D, 1)$, and so the second PCPC associated with $g_2(D) = 1+D+D^2$ is

$$g_2^{[2]}(D) = \begin{pmatrix} 1+D & 1 \\ D & 1+D \end{pmatrix}. \tag{8.12}$$

Thus the block matrix $\begin{pmatrix} g_1^{[2]}(D) & g_2^{[2]}(D) \end{pmatrix}$ is given by

$$\begin{pmatrix} g_1^{[2]}(D) & g_2^{[2]}(D) \end{pmatrix} = \begin{pmatrix} 1+D & 0 & 1+D & 1 \\ 0 & 1+D & D & 1+D \end{pmatrix},$$

which, apart from a permutation of the columns, is identical to the matrix $G^{[2]}(D)$ in (8.6). Indeed, if we *interleave* the columns of $\begin{pmatrix} g_1^{[2]}(D) & g_2^{[2]}(D) \end{pmatrix}$ to depth 2, i.e., take the columns in order $1, 3, 2, 4$, the result is exactly $G^{[2]}(D)$.

Similarly, with $M = 3$, we find that the third PCPC's associated with $g_1(D)$ and $g_2(D)$ are

$$g_1^{[3]}(D) = \begin{pmatrix} 1 & 0 & 1 \\ D & 1 & 0 \\ 0 & D & 1 \end{pmatrix}$$

$$g_2^{[3]}(D) = \begin{pmatrix} 1 & 1 & 1 \\ D & 1 & 1 \\ D & D & 1 \end{pmatrix},$$

from which we see that the block matrix $\begin{pmatrix} g_1^{[3]}(D) & g_2^{[3]}(D) \end{pmatrix}$, when interleaved to depth 3, i.e., if the columns are taken in order $1, 4, 2, 5, 3, 6$, is the same as the matrix $G^{[3]}(D)$ given in (8.7). A straightforward elaboration of this argument leads to the following theorem and corollary.

**8.3 Theorem.** *If $\boldsymbol{C}$ is an $(n, k)$ convolutional code, then the $M$th blocking of $\boldsymbol{C}$, denoted by $\boldsymbol{C}^{[M]}$, is an $(nM, kM)$ convolutional code. If $G(D) = (g_{i,j}(D))$ is a $k \times n$ polynomial generator matrix for the original code $\boldsymbol{C}$, then a generator matrix for $C^{[M]}$, say $G^{[M]}(D)$, can be obtained from $G(D)$ by replacing each entry $g_{i,j}(D)$ from $G(D)$ with the corresponding $M$th PCPC, and then interleaving the columns to depth $M$.*

**8.4 Corollary.** *If the row degrees of the generator matrix $G(D)$ are $(e_1, \ldots, e_k)$, then the row degrees of $G^{[M]}(D)$ are*

$$(\lfloor e_i/M \rfloor, \lfloor (e_i + 1)/M \rfloor, \ldots, \lfloor (e_i + M - 1)/M \rfloor) \quad \text{for } i = 1, \ldots, k.$$

*(In other words, a single row degree of $e$ in $G(D)$ is replaced with $M$ row degrees in $G^{[M]}(D)$. If $e = e'M + r$ with $0 \leq r < M$, then $M - r$ of these degrees will be equal to $e'$ and $r$ will be equal to $e' + 1$. The sum of the row degrees $G^{[M]}(D)$, i.e., the external degree of $G^{[M]}(D)$, is thus the same as for $G(D)$.)*

The following "corollaries" to Theorem 8.3 do not really follow in a simple manner from Theorem 8.3, but we include them here because of their importance. We refer the interested reader to [52, Chapters 10 and 13] for the proofs. However, the fact that the degree is the same for all blocked codes is intuitively clear, since as we saw in Chapter 4, the degree $m$ represents the smallest possible number of memory elements needed to implement a physical encoder, and there is no essential difference in the encoders for the various blocked versions of the original code. The fact that $d_{\text{free}}$ is the same for all of the blocked codes is apparent, since the set of codewords is the same for all values of $M$.

**8.5 Theorem.** *If $G(D)$ is a canonical generator matrix for $\boldsymbol{C}$, then $G^{[M]}(D)$ will be a canonical generator matrix for $\boldsymbol{C}^{[M]}$. Thus the degree of the blocked code $\boldsymbol{C}^{[M]}$ is the same as that for the original code.*

**Proof:** (Sketch only) We need to show that $G^{[M]}(D)$ is both basic and reduced, given that $G(D)$ is. Since $G(D)$ is basic, by Theorem A.1, $G(D)$ will have a polynomial right inverse, say $H(D)$. Then $H^{[M]}(D)$ will be a polynomial right inverse for $G^{[M]}(D)$, so $G^{[M]}(D)$ is basic. Since by Corollary 8.4 the external degree of $G^{[M]}(D)$ is the same as that of $G(D)$, and the external degree of $G(D)$ is the degree of the original code, it follows that the degree of the blocked code is $\leq m$. But we saw in Chapter 4 that the degree of a code represents the smallest possible number of memory element needed to implement a physical encoder. If the degree of $G^{[M]}(D)$ were less than $m$, there would be an encoder for the blocked code which used fewer than $m$ memory elements. But there is no essential difference between the parent code and the blocked code, so this is impossible. It follows that the external degree of $G^{[M]}(D)$ cannot be reduced, and so by Definition 3.2, it is a reduced matrix. ▪

**8.6 Corollary.** *The $M$th blocking of a $(n, k, m, d)$ convolutional code is a $(nM, kM, m, d)$ convolutional code. If the Forney indices of the original code are $(e_1, \ldots, e_k)$, then the Forney indices of $\boldsymbol{C}^{[M]}$ are*

$$(\lfloor e_i/M \rfloor, \lfloor (e_i + 1)/M \rfloor, \ldots, \lfloor (e_i + M - 1)/M \rfloor) \quad \text{for } i = 1, \ldots, k.$$

**Proof:** This follows from Theorem 8.5 and Corollary 8.4, together with the observation that $d_{\text{free}}$ is the same for all blocked versions of the parent codes, since the set of codewords is the same for all values of $M$. ▪

**8.7 Example:** Starting from the $(2, 1, 2, 5)$ code of Figure 8.1, the blocking technique leads to codes with parameters $(4, 2, 2, 5)$, $(6, 3, 2, 5)$, $(8, 4, 2, 5)$, .... Note that by Corollary 4.5 with $L = 1$, the free distance of *any* $(4, 2, 2)$ code is at most $\Delta(8, 2) = 5$, so that the $(4, 2, 2, 5)$ code whose generator matrix is given by (8.6) is optimal. However, there is a $(6, 3, 2, 6)$ code (see [33, Table II]), so the $(6, 3, 2, 5)$ code produced by blocking is not optimal. ▪

**8.8 Example:** Let us take as the parent code the $(2, 1, 3)$, $d_{\text{free}} = 6$ with canonical generator matrix is $G(D) = (1 + D + D^3, 1 + D + D^2 + D^3)$ [Table 4.1]. If we block this code with $M = 2$, it becomes a $(4, 2, 3)$ code, still with $d_{\text{free}} = 6$. Using Theorem 8.3, we find that a canonical generator matrix $M = 2$ blocked version of this code is

$$G^{[2]}(D) = \begin{pmatrix} 1 & 1+D & 1+D & 1+D \\ D+D^2 & D+D^2 & 1 & 1+D \end{pmatrix}.$$

According to Corollary 4.5 with $L = 2$, the free distance of *any* $(4, 2, 3)$ code can be at most $\Delta(12, 3) = 6$, so that this code is an optimal $(4, 2, 3)$ code. Similary, if we block the code with block length 3, we obtain a $(6, 3, 3)$ code with $d_{\text{free}} = 6$, with canonical generator matrix

$$G^{[3]}(D) = \begin{pmatrix} 1+D & 1+D & 1 & 1 & 0 & 1 \\ 0 & D & 1+D & 1+D & 1 & 1 \\ D & D & 0 & D & 1+D & 1+D \end{pmatrix}.$$

This code too is optimal, for by the bound of Corollary 4.5 with $L = 1$, the free distance of *any* $(6, 3, 3)$ code can be at most $\Delta(12, 3) = 6$. Similar calculations show that the $M = 5$ and $M = 6$ blockings of the $(2, 1, 3)$ code are also optimal. Strangely, however, the $M = 4$ blocking, which yields a $(8, 4, 3)$ code with $d_{\text{free}} = 6$, is not optimal since there is an $(8, 4, 3)$ code with $d_{\text{free}} = 8$ (see [1][33, Table I]). ▪

Having now described *blocking*, we come finally to the formal definition of *puncturing*. We need the notion of a *puncturing matrix*, which we illustrate by example. Starting with the $(2, 1, 2, 5)$ code with generator matrix $G(D) = (g_1(D), g_2(D)) = (1 + D^2, 1 + D + D^2)$, we saw above that the $M = 2$ PCPC's were

$$(g_1^{[2]}(D), g_2^{[2]}(D)) = \left( \begin{pmatrix} 1+D & 0 \\ 0 & 1+D \end{pmatrix}, \begin{pmatrix} 1+D & 1 \\ D & 1+D \end{pmatrix} \right). \tag{8.13}$$

The display in (8.13) is a bit unorthodox, but nevertheless it can be viewed as a matrix with four columns, indexed $(1, 1)$, $(1, 2)$, $(2, 1)$ and $(2, 2)$. If we now delete column $(1, 2)$, we obtain the following $2 \times 3$ matrix

$$\left( \begin{pmatrix} 1+D \\ 0 \end{pmatrix}, \begin{pmatrix} 1+D & 1 \\ D & 1+D \end{pmatrix} \right). \tag{8.14}$$

We will say that the matrix in (8.14) has been obtained from the matrix in (8.13), by *puncturing according to the pattern*

$$P = \begin{pmatrix} 1 & 0 \\ 1 & 1 \end{pmatrix}. \tag{8.15}$$

The 1's in the matrix $P$ indicate columns that are retained; the 0's indicate columns that are deleted. Alternatively, we can say that the matrix in (8.14) has been obtained from the original matrix $G(D) = (1 + D^2, 1 + D + D^2)$, by *puncturing with pattern* $P = \begin{pmatrix} 1 & 0 \\ 1 & 1 \end{pmatrix}$, and denote the resulting matrix by $G_P(D)$. The code with generator matrix $G_P(D)$ is called the $P$-puncturing of the original code. We can summarize this example with the following display:

$$G(D) = \begin{pmatrix} 1 + D^2 & 1 + D + D^2 \end{pmatrix}, \quad P = \begin{pmatrix} 1 & 0 \\ 1 & 1 \end{pmatrix}$$

$$\Downarrow$$

$$G_P(D) = \begin{pmatrix} 1 + D & 1 + D & 1 \\ 0 & D & 1 + D \end{pmatrix}$$

(This is the example we began the section with.)

The general definition follows.

**8.9 Definition.** *If $G(D)$ is a $k \times n$ polynomial matrix, and if $P$ is a $n \times M$ matrix of 0's and 1's, the $nM$ columns of the matrix $G^{[M]}(D)$ are in natural one-to-one correspondence with the $nM$ entries of $P$, and the matrix $G_P(D)$ is defined to be the matrix obtained from $G^{[M]}(D)$ by deleting those columns which correspond to the 0 entries in $P$. The code defined by the generator matrix $G_P(D)$ is called the $P$-punctured version of $\boldsymbol{C}$.*

By Corollary 8.6, it follows that if $\boldsymbol{C}$ is an $(n, k, m, d)$ code, the $P$-puncturing of $\boldsymbol{C}$ is an $(N, kM, m', d')$ code, where $N$ is the Hamming weight of $P$, $m' \leq m$ and $d' \leq d$. Unfortunately, little more about $m'$ and $d'$ can be said in general, and the exact determination of these parameters must be done on a case-by case basis, using numerical methods. The determination of $m'$ is usually quite easy, since if the parent generator matrix $G(D)$ is canonical, then $G_P(D)$, is likely to be canonical, in which case $m' = m$, or very close to canonical, in which case a few row operations on $G_P(D)$ will produce a canonical generator matrix, from which $m'$ can be determined by inspection. The determination of $d'$, however, is rather more difficult, and we refer the reader to [4][7][8][9][21][60], especially [9, Section 4.8].

**8.10 Example:** If we start with the $(2, 1, 3)$ code from Example 8.8, and take $P = \begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix}$, we obtain a $(3, 2)$ code with generator matrix

$$\begin{pmatrix} 1 & 1 + D & 1 + D \\ D + D^2 & D + D^2 & 1 \end{pmatrix}.$$

This generator matrix is canonical, and so the Forney indices are $(1, 2)$, and the code's degree is 3. Its free distance turns out to be 4, so the $\begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix}$-puncturing of the parent code is a $(3, 2, 3, 4)$ code. By Theorem 7.13, this code is optimal. Similarly, if we consider the $\begin{pmatrix} 1 & 0 & 0 \\ 1 & 1 & 1 \end{pmatrix}$-puncturing of the same parent code, we obtain a $(4, 3)$ code with generator matrix

$$\begin{pmatrix} 1+D & 1+D & 1 & 1 \\ 0 & D & 1+D & 1 \\ D & D & D & 1+D \end{pmatrix}.$$

This generator matrix is canonical, and so the code's degree is 3. The free distance turns out to be 4. The bound of Corollary 4.5, when applied to $(4, 3, 3)$ codes, gives $d_{\text{free}} \leq 4$, so this code is also optimal. Similarly Yasuda, Kashiki, and Hirata [60] have shown that by suitably puncturing the $M = 4$ and $M = 5$ blockings of the parent $(2, 1, 3)$ code, it is possible to obtain optimal $(5, 4, 3, 3)$ and $(6, 5, 3, 3)$ codes. ▪

In all our examples so far, puncturing the code did not result in a decrease in the code's degree. The final example in this section shows that a decrease in degree can sometimes happen.

**8.11 Example:** (Taken from [4, Table I].) Here the parent code is a $(3, 1, 4, 9)$ code with generator matrix

$$G(D) = \begin{pmatrix} 1 + D^2 + D^3 & 1 + D^3 & 1 + D + D^2 + D^4 \end{pmatrix}.$$

In this case the parent code is not optimal, since there is a $(3, 1, 4, 12)$ code [Table 4.2]. The puncturing matrix is

$$P = \begin{pmatrix} 1 & 0 \\ 0 & 1 \\ 0 & 1 \end{pmatrix}.$$

The matrix $G^{[2]}(D)$ is

$$G^{[2]}(D) = \left( \begin{pmatrix} 1+D & D \\ D^2 & 1+D \end{pmatrix}, \begin{pmatrix} 1 & D \\ D^2 & 1 \end{pmatrix}, \begin{pmatrix} 1+D+D^2 & 1 \\ D & 1+D+D^2 \end{pmatrix} \right).$$

If we puncture $G^{[2]}(D)$ using $P$ as the template, we obtain

$$G_P = \begin{pmatrix} 1+D & D & 1 \\ D^2 & 1 & 1+D+D^2 \end{pmatrix}$$

It is easy to see that $G_P(D)$ is canonical, so that the punctured code is a $(3, 2, 3)$ code. Note that degree of the punctured code is strictly less than that of the parent code. The free distance turns out to be $d_{\text{free}} = 4$, so that we obtain an $(3, 2, 3, 4)$ code, just as in Example 8.10, but in a completely different way. ▪

## Appendix A. Basic and Reduced Matrices.

*Abstract: In this appendix we collect for reference many technical, but useful, properties of basic and reduced matrices, which were defined in Section 3. The main results are presented in two theorems. Theorem A.1 gives six properites of a polynomial matrix, which are equivalent to its being basic, and Theorem A.2 gives three properties which are equivalent to being reduced.*

**A.1 Theorem.** *A $k \times n$ polynomial matrix $G(D)$ is basic (see Definition 3.1) if and only if any one of the following six conditions is satisfied.*

(1) The invariant factors of $G(D)$ are all 1.

(2) The gcd of the $k \times k$ minors of $G(D)$ is 1.

(3) $G(\alpha)$ has rank $k$ for any $\alpha$ in the algebraic closure of $F$.

(4) $G(D)$ has a right $F[D]$ inverse, i.e., there exists a $n \times k$ polynomial matrix $H(D)$ such that $G(D)H(D) = I_k$.

(5) If $\boldsymbol{x}(D) = \boldsymbol{u}(D)G(D)$, and if $\boldsymbol{x}(D) \in F[D]^n$, then $\boldsymbol{u}(D) \in F[D]^k$. ("Polynomial output implies polynomial input.")

(6) $G(D)$ is a submatrix of a unimodular matrix, i.e., there exists a $(n-k) \times n$ matrix $L(D)$ such that the $n \times n$ matrix $\begin{pmatrix} G(D) \\ L(D) \end{pmatrix}$ has determinant which is a nonzero scalar.

**Proof:** The proof is logically rather involved. We shall prove the following implications: (Basic) $\rightarrow$ (1) $\rightarrow$ (2) $\rightarrow$ (4) $\rightarrow$ (5) $\rightarrow$ (Basic); (2) $\leftrightarrow$ (3); (1) $\leftrightarrow$ (6).

• (Basic) $\rightarrow$ (1): Suppose that $\Gamma$ is the $k \times n$ invariant-factor form for $G$, i.e., $\Gamma = \operatorname{diag}(\gamma_1, \gamma_2, \ldots, \gamma_k)$ where $\gamma_i = \Delta_i/\Delta_{i-1}$, $\Delta_i$ being the gcd of the $i \times i$ minors of $G$. (We take $\Delta_0 = 1$ by convention.) Then there is a $k \times k$ unimodular matrix $X$ and an $n \times n$ unimodular matrix $Y$ such that

$$XGY = \Gamma. \tag{A.1}$$

(For a discussion of this "invariant factor decomposition," see Appendix B. For detailed proofs, see [29, Theorem 6.3.16].) Thus if $\Gamma_k$ denotes the $k \times k$ matrix formed by the leftmost $k$ columns of $\Gamma$, the matrix $G' = \Gamma_k^{-1} XG$ is a polynomial matrix equivalent to $G$. Furthermore, since $\det(\Gamma_k^{-1}X) = \det \Gamma_k^{-1} \det X = (\det \Gamma_k)^{-1} \det X = s(\gamma_1 \cdots \gamma_k)^{-1}$, where $s$ is a nonzero scalar, unless the $\gamma_i$'s are all 1, the internal degree of $G'$ is strictly less than that of $G$. Thus if the invariant factors of $G$ are not all 1, then $G$ is not basic, which proves (Basic) $\rightarrow$ (1).

• (1) $\leftrightarrow$ (2): According to the definitions given in the previous paragraph, the product of

the invariant factors of $G$ is

$$\begin{aligned}\gamma_1\gamma_2\cdots\gamma_k &= \frac{\Delta_1}{\Delta_0}\cdot\frac{\Delta_2}{\Delta_1}\cdots\frac{\Delta_k}{\Delta_{k-1}} \\ &= \frac{\Delta_k}{\Delta_0} \\ &= \Delta_k = (\text{the gcd of the } k\times k \text{ minors of } G.)\end{aligned}$$

Hence the $\gamma_i$'s are all 1 if and only if $\Delta_k = 1$.

• (2) $\rightarrow$ (4): Suppose that the gcd of the $k\times k$ minors of $G(D)$ is 1, and denote the individual minors by $\Delta_\nu(D)$, for $\nu = 1, 2, \ldots, \binom{n}{k}$. Then by Cramer's rule for each $\nu$, there will exist a "pseudo-inverse" for $G(D)$, with factor $\Delta_\nu(D)$, i.e., an $n\times k$ matrix $H_\nu(D)$ such that $G(D)H_\nu(D) = \Delta_\nu(D)I_k$. Since the gcd of the $\Delta_\nu(D)$'s is 1, there exists a polynomial linear combination of the $\Delta_\nu(D)$'s equal to 1, say $\sum_\nu \lambda_\nu(D)\Delta_\nu(D) = 1$. It follows that $H(D) = \sum_\nu \lambda_\nu(D)H_\nu(D)$ is an $n\times k$ polynomial inverse for $G(D)$.

• (4) $\rightarrow$ (5): Suppose that $G(D)$ has an $n\times k$ polynomial inverse $H(D)$, and that $\boldsymbol{u}(D) = (u_1(D), \ldots, u_k(D))$ is a $k$-vector of rational functions such that $\boldsymbol{x}(D) = \boldsymbol{u}(D)G(D)$ is an $n$-vector of polynomials. Multiplying this equation on the right by $H(D)$, we obtain $\boldsymbol{x}(D)H(D) = \boldsymbol{u}(D)$, which implies that $\boldsymbol{u}(D)$ is in fact a polynomial vector.

• (5) $\rightarrow$ (Basic): Suppose that (5) holds, and let $T(D)$ be an arbitrary nonsingular $k\times k$ matrix of rational functions such that $G' = TG$ is a polynomial matrix. Then by property (5), $T$ must in fact be a polynomial matrix, so that by Theorem 3.3(a), $\operatorname{intdeg} G' \geq \operatorname{intdeg} G$, which means that $G$ is basic.

• (2) $\leftrightarrow$ (3): Suppose that the gcd of the $k\times k$ minors of $G(D)$ is 1, let $\alpha$ be an arbitrary element of the algebraic closure of $F$, and let $p(D)$ be the minimal polynomial of $\alpha$. Then there must be at least one $k\times k$ subdeterminant of $G(D)$ which is not divisible by $p(D)$, which means that the corresponding $k\times k$ submatrix of $G(\alpha)$ is nonsingular. Thus $G(\alpha)$ must have rank $k$. Conversely, suppose that the gcd of the $k\times k$ minors of $G$ is not 1, which means that it is divisible by some irreducible polynomial $p(D)$. If $\alpha$ is a root of $p(D)$ in some extension field of $F$, it follows that every $k\times k$ minor of $G(\alpha)$ is zero, which in turn means that $G(\alpha)$ has rank less than $k$.

• (1) $\leftrightarrow$ (6): Suppose the invariant factors of $G$ are all 1. Then the invariant-factor decomposition in eq. (A.1) can be written as

$$G = A\,(\,I_k \quad 0_{k,n-k}\,)\,B,$$

where $A = X^{-1}$ and $B = Y^{-1}$. Thus if $B = \begin{pmatrix} B_U \\ B_L \end{pmatrix}$ where $B_U$ is $k\times n$ and $B_L$ is $(n-k)\times n$, it follows that $G = AB_U$. But the matrix $\begin{pmatrix} AB_U \\ B_L \end{pmatrix}$ is unimodular, since it is obtained from the unimodular matrix $B$ via a sequence of elementary row operations on the first $k$ rows. Conversely, if $B = \begin{pmatrix} G(D) \\ H(D) \end{pmatrix}$ is unimodular, then the equation $G(D) = I_k\,(\,\Gamma_k \quad 0_{k,n-k}\,)\,B$ shows that the invariant factors of $G(D)$ are all 1. ▪

**A.2 Theorem.** *A $k \times n$ polynomial matrix is reduced (see Definition 3.2) if and only if one of the following three conditions is satisfied.*

(1) If we define the the "indicator matrix for the highest-degree terms in each row" $\overline{G}$ by

$$\overline{G}_{ij} = \operatorname*{coeff}_{D^{e_i}} g_{ij}(D),$$

where $e_i$ is the degree of $G(D)$'s $i$th row, then $\overline{G}$ has rank $k$.

(2) $\operatorname{extdeg} G(D) = \operatorname{intdeg} G(D)$.

(3) The "predictable degree property": For any $k$-dimensional polynomial vector, i.e., any $\boldsymbol{u}(D) \in F[D]^k$

$$\deg(\boldsymbol{u}(D)G(D)) = \max_{1 \le i \le k} (\deg u_i(D) + \deg \boldsymbol{g}_i(D)).$$

**Proof:** The logical organization of this proof is as follows: we shall prove that (Reduced) $\rightarrow$ (1) $\rightarrow$ (2) $\rightarrow$ (Reduced), and (1) $\leftrightarrow$ (3).

• (Reduced) $\rightarrow$ (1): Suppose (1) is false. Then there is a nonzero $k$-dimensional vector from $F$, say $\boldsymbol{\alpha} = (\alpha_1, \ldots, \alpha_k)$, such that $\boldsymbol{\alpha}\overline{G} = 0$. Now suppose that the rows of $G$ are $(\boldsymbol{g}_1, \ldots, \boldsymbol{g}_k)$ with $\deg \boldsymbol{g}_i = e_i$, and $e_1 \le e_2 \le \cdots \le e_k$. Then from $\boldsymbol{\alpha}\overline{G} = 0$, it follows that the coefficient of $D^{e_k}$ in the linear combination

$$\boldsymbol{g}'_k = \alpha_1 D^{e_k - e_1}\boldsymbol{g}_1 + \alpha_2 D^{e_k - e_2}\boldsymbol{g}_2 + \cdots + \alpha_k D^{e_k - e_k}\boldsymbol{g}_k$$

is zero, so that the unimodular transformation of $G(D)$ that replaces $\boldsymbol{g}_k$ with $\boldsymbol{g}'_k$, and leaves the remaining rows of $G$ unchanged, reduces the external degree of $G$. In other words, if (1) is false, $G$ is not reduced, which is (the contrapositive of) what we set out to prove.

• (1) $\rightarrow$ (2): Suppose that $\overline{G}$ has rank $k$, and denote the $k \times k$ submatrices of $\overline{G}$ by $\overline{G}_\nu$, for $\nu = 1, 2, \ldots, \binom{n}{k}$. Then since $\operatorname{rank} \overline{G} = k$, there is at least one index $\nu_0$ such that $\det \overline{G}_{\nu_0} \neq 0$. If now the row degrees of $G$ are $e_1, \ldots, e_k$, then (cf. the proof of Theorem 3.3(b)) the coefficient of $D^{e_1 + \cdots + e_k}$ in $\det G_{\nu_0}$ is $\det \overline{G}_{\nu_0} \neq 0$. Thus $\operatorname{intdeg} G \ge \operatorname{extdeg} G$. The opposite inequality is true for any matrix, as was shown in Theorem 3.3(b).

• (2) $\rightarrow$ (Reduced): Suppose $\operatorname{intdeg} G(D) = \operatorname{extdeg} G(D)$, and that $T(D)$ is an arbitrary $k \times k$ unimodular matrix. Then $\operatorname{extdeg} TG \ge \operatorname{intdeg} TG$ by Theorem 3.3(b); $\operatorname{intdeg} TG = \operatorname{intdeg} G$, by Theorem 3.3(a); and $\operatorname{intdeg} G = \operatorname{extdeg} G$, by assumption. Combining this string of inequalities and equalities, we get $\operatorname{extdeg} TG \ge \operatorname{extdeg} G$, which proves that $G$ is reduced.

• (1) $\leftrightarrow$ (3): Let $\boldsymbol{u}(D) = (u_1(D), \ldots, u_k(D))$ be a $k$-vector of polynomials, and let $\boldsymbol{x}(D) = (x_1(D), \ldots, x_n(D))$ be defined by the equation $\boldsymbol{x}(D) = \boldsymbol{u}(D)G(D)$. If the $k$ rows of $G(D)$ are denoted by $\boldsymbol{g}_1(D), \ldots, \boldsymbol{g}_k(D)$, then we have

$$\begin{aligned} \boldsymbol{x}(D) &= \boldsymbol{u}(D)G(D) \\ &= u_1(D)\boldsymbol{g}_1(D) + \cdots + u_k(D)\boldsymbol{g}_k(D). \end{aligned} \tag{A.2}$$

If the degree of $u_i(D)$ is $d_i$, for $i = 1, \ldots, k$, and the degree of $\boldsymbol{g}_i(D)$ is $e_i$, for $i = 1, \ldots, k$, it follows from eq. (A.2) that the degree of $\boldsymbol{x}(D)$ is at most $d = \max_i(d_i + e_i)$. We call $d$ the "prediction" of the degree of $\boldsymbol{x}(D)$. To test the prediction, we note that the vector of coefficients of $D^d$ in $x(D)$ is $\boldsymbol{\alpha} = (\alpha_1, \ldots, \alpha_k)\overline{G}$, where $\alpha_i$ is the coefficient of $D^{d-e_i}$ in $u_i(D)$. (At least one of the $\alpha_i$ is nonzero, since $d_i + e_i = d$ must hold for at least one index $i$.) But $\boldsymbol{\alpha}\overline{G} \neq 0$ for all nonzero $\boldsymbol{\alpha}$'s if and only if $\overline{G}$ has rank $k$, and so the prediction is true for all $\boldsymbol{u}(D)$'s if and only if $\operatorname{rank} \overline{G} = k$. ▪

## Appendix B. The Extended Smith Algorithm and Applications.

*Abstract: In this appendix, we shall show that by using the algebra of $k \times n$ polynomial matrices, in particular the invariant-factor theorem (a.k.a. the Smith Form), one can transform an arbitrary generator matrix for an $(n, k)$ convolutional code $\boldsymbol{C}$ into a noncatastrophic, basic, and ultimately a canonical, encoder for $\boldsymbol{C}$. We will also show how to find a polynomial inverse for a basic encoder for $\boldsymbol{C}$, and a basic encoder for the dual code $\boldsymbol{C}^{\perp}$. Our main tool will be an algorithm, which we call the "extended Smith algorithm," for finding the invariant factors of an arbitrary $k \times n$ matrix over an Euclidean domain, which bears the same relationship to the usual invariant factor algorithm as the "extended Euclid's algorithm" bears to the usual Euclid's algorithm. Most of these results first appeared in [43].**

We recall that the goal of the *Euclidean algorithm* (see e.g. Chapter 1 of this *Handbook*, or [29, Sec. 1.1], [42, Chapter 2]) is to take a pair $(a, b)$ of elements from a Euclidean domain $R$, and by repeated application of the division algorithm for $R$ to find the greatest common divisor $d$ of $a$ and $b$. The goal of the *extended Euclidean algorithm* (see e.g. ([31, Sec. 4.5.2]) or ([41, Sec. 8.4]) is to take the same pair, and not only find $d$, but also to find elements $s$ and $t$ of $D$, such that $sa+tb = d$. The extended Euclidean algorithm has several important applications in coding theory. For example, it can be used to compute inverses in finite fields ([42, Example 4.2]), decode BCH codes ([41, Section 8.5]), and to find inverses for noncatastrophic $(n, 1)$ convolutional encoders ([5, Sec. 12.2]).

Similarly, the goal of the *Smith algorithm*, which is often called the *invariant-factor algorithm*, (see e.g. [20, Sec. 6.2.4], [29, Sec. 6.3.3], or [49, Sec. 12.2]) is to take an arbitrary $k \times n$ matrix $G$ (with $k \leq n$) over a Euclidean domain $R$, and by a sequence of elementary row and column operations, to reduce $G$ to a $k \times n$ diagonal matrix $\Gamma = \mathrm{diag}(\gamma_1, \ldots, \gamma_k)$, whose diagonal entries are the invariant factors of $G$, i.e., $\gamma_i = \Delta_i/\Delta_{i-1}$, where $\Delta_i$ is the gcd of the $i \times i$ minors of $G$. (We take $\Delta_0 = 1$ by convention.) The goal of the *extended Smith algorithm*, which was introduced in [43], is to take the same input, and not only find $\Gamma$, but also to find a $k \times k$ unimodular matrix $X$, and an $n \times n$ unimodular matrix $Y$, such that $XGY = \Gamma$. In particular, when the extended Smith algorithm is applied to a $1 \times 2$ matrix, say $G = (\, a \quad b\,)$, it produces the matrix $G' = (\, d \quad 0\,)$, where $d = \gcd(a, b)$, as well as a $2 \times 2$ unimodular matrix $Y = \begin{pmatrix} s & s' \\ t & t' \end{pmatrix}$ such that $GY = G'$, i.e., $as + bt = d$ and $as' + bt' = 0$, which is exactly what the extended Euclid's algorithm produces when given the input $(a, b)$.

We begin with a careful description of Smith's algorithm (which is often called the invariant-factor algorithm), but we will not give a formal proof of its correctness. For that, we refer the reader to [20, Sec. 6.2.4], [29, Sec. 6.3.3], or [49, Sec. 12.2].

---

* An alternative, and in some respects simpler, technique for transforming encoders to canonical form can be based in the *Hermite* canonical form for a polynomial matrix. For a discussion of this, see [48, Section 2.3.2.]

The central part of Smith's algorithm is the following sub-algorithm "E," which takes as input an arbitrary matrix $A$ with entries from a Euclidean domain $R$, at least one of which is nonzero, and, via elementary row and column operations, transforms $A$ into a matrix with the $(1,1)$ entry nonzero, every other entry in row 1 and column 1 zero, and every other entry in the matrix divisible by the $(1,1)$ entry.

E1. Move an entry in $A$ of least size to position $(1,1)$.

E2. If there is a nonzero entry in either row 1 or column 1 which is not divisible by $a_{1,1}$, use it to reduce the size of $a_{1,1}$, as follows.

E2a. If there is a nonzero entry, say $a_{1,j}$, in row 1 which is not divisible by $a_{1,1}$, then by the division algorithm there exist nonzero elements $q$ and $r$ such that $a_{1,j} = qa_{1,1} + r$, with $\text{size}\, r < \text{size}\, a_{1,1}$. If we subtract $q$ times column 1 from column $j$, the $(1,j)$ entry will be changed to $r$, which has smaller size than $a_{1,1}$. If we move this entry to position $(1,1)$, we will have reduced the size of $a_{1,1}$.

E2b. If there is a nonzero entry, say $a_{i,1}$, in column 1 which is not divisible by $a_{1,1}$, we repeat the procedure outlined in step 2a, with rows and columns interchanged.

E3. Change all the entries in row 1 and column 1 (except $a_{1,1}$ itself) to zero, as follows.

E3a. Since (by step E2) $a_{1,j}$ is divisible by $a_{1,1}$, we have $a_{1,j} = q_j a_{1,1}$. Thus by subtracting $q_j$ times column 1 from column $j$ the $(1,j)$ entry will be transformed to zero. Repeating this step for $j = 2, \ldots, r$, we will "zero out" the first row.

E3b. Repeat step 3a with rows and columns interchanged.

E4. If there is a nonzero entry in $A$, say $a_{i,j}$, which is not divisible by $a_{1,1}$, add column $j$ to column 1. (This produces a nonzero entry in column 1 which is not divisible by $a_{1,1}$.) Return to step E2.

E5. Stop. The matrix $A$ now has the desired property.

In the Smith algorithm, one applies Algorithm E successively to the original matrix $G$, then to the matrix obtained from $G$ by deleting row 1 and column 1, then by deleting row 2 and column 2, etc., until either an all-zero matrix is encountered, or until all rows and columns have been processed. The result is a $k \times n$ diagonal matrix $\Gamma = \text{diag}(\gamma_1, \ldots, \gamma_k)$, whose diagonal entries are the invariant factors of $G$. The following example illustrates the Smith algorithm, when the underlying Euclidean domain is the ring of polynomials over the field $GF(2)$. In this domain, the "size" of an element is its degree. (See [42, Chapter 2], for a discussion of more general Euclidean domains.)

**B.1 Example:** Consider the following $2 \times 4$ matrix with entries which are polynomials over $GF(2)$ (cf. Example 3.4, matrix $G_2(D)$):

$$G = G_0 = \begin{pmatrix} 1 & 1+D+D^2 & 1+D^2 & 1+D \\ D & 1+D+D^2 & D^2 & 1 \end{pmatrix}.$$

Since a lowest-degree term already appears in the $(1,1)$ position, we skip step E1. Since the $(1,1)$ entry is 1, every nonzero entry in row 1 and column 1 is divisible by the $(1,1)$

entry, so we skip step E2 as well. Executing Step 3: we "zero out" the first row by adding $(1 + D + D^2)$ times column 1 to column 2, $(1 + D^2)$ times column 1 to column 3, and $(1 + D)$ times column 1 to column 4, thereby obtaining successively

$$G_1 = \begin{pmatrix} 1 & 0 & 1+D^2 & 1+D \\ D & 1+D^3 & D^2 & 1 \end{pmatrix}$$
$$G_2 = \begin{pmatrix} 1 & 0 & 0 & 1+D \\ D & 1+D^3 & D+D^2+D^3 & 1 \end{pmatrix}$$
$$G_3 = \begin{pmatrix} 1 & 0 & 0 & 0 \\ D & 1+D^3 & D+D^2+D^3 & 1+D+D^2 \end{pmatrix}.$$

To zero the entry in position $(2, 1)$, we add $D$ times row 1 to row 2, thereby obtaining

$$G_4 = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1+D^3 & D+D^2+D^3 & 1+D+D^2 \end{pmatrix}.$$

This completes the operation of Algorithm E on the original matrix $G$. We now apply Algorithm E to the $1 \times 3$ matrix obtained by deleting row 1 and column 1 from $G_4$. The lowest degree term in $G_4$ appears in position $(2, 4)$, so by interchanging columns 2 and 4, we move it to position $(2, 2)$:

$$G_5 = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1+D+D^2 & D+D^2+D^3 & 1+D^3 \end{pmatrix}.$$

At this point, all of the remaining entries in row 2 are divisible by the $(2, 2)$ entry: $D + D^2 + D^3 = D \cdot (1 + D + D^2)$, and $1 + D^3 = (1 + D) \cdot (1 + D + D^2)$. Thus by adding $D$ times column 2 to column 3, and then $(1 + D)$ times column 2 to column 4, we compute successively

$$G_6 = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1+D+D^2 & 0 & 1+D^3 \end{pmatrix}$$
$$G_7 = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1+D+D^2 & 0 & 0 \end{pmatrix}.$$

The matrix $G_7$ is the invariant-factor form for $G$; in particular, the invariant factors of $G$ are $\gamma_1 = 1$ and $\gamma_2 = 1 + D + D^2$. ▪

We have now seen in computational detail how the Smith algorithm works. Formally, it can be described as follows. Beginning with the matrix $G_0 = G$, it produces a sequence of $k \times n$ matrices $G_1, G_2, \ldots$, where $G_{i+1}$ is derived from $G_i$ by either an elementary row operation or an elementary column operation. We can represent this algebraically as

$$G_{i+1} = E_{i+1} G_i F_{i+1}, \tag{B.1}$$

where $E_{i+1}$ and $F_{i+1}$ are $k \times k$ and $n \times n$ elementary matrices, respectively. If $G_{i+1}$ is obtained from $G_i$ via a row operation, then $F_{i+1} = I_n$, but if $G_{i+1}$ is obtained from $G_i$ via a column operation, then $E_{i+1} = I_k$. After a finite number $N$ of steps, we obtain $G_N = \Gamma$.

The extended Smith algorithm builds on the Smith algorithm. Whereas the Smith algorithm works only with the sequence $G_0, G_1, \ldots, G_N$, the extended Smith algorithm also works with a sequence of unimodular $k \times k$ matrices $X_0, \ldots, X_N$, and a sequence of unimodular $n \times n$ matrices $Y_0, \ldots, Y_N$. The sequences $(X_i)$ and $(Y_i)$ are initialized as $X_0 = I_k$, $Y_0 = I_n$, and updated via the rule (cf. Eq. (B.1))

$$X_{i+1} = E_{i+1} X_i \tag{B.2}$$

$$Y_{i+1} = Y_i F_{i+1}. \tag{B.3}$$

The following simple lemma is the key to the extended Smith algorithm.

**B.2 Lemma.**

$$X_i G Y_i = G_i \qquad \textit{for } i = 0, 1, \ldots, N. \tag{B.4}$$

**Proof:** Since $X_0 = I_k$ and $Y_0 = I_n$, (B.4) holds for $i = 0$. Assuming now that (B.4) holds for a given value of $i$, we multiply both sides of (B.4) on the left by $E_{i+1}$ and on the right by $F_{i+1}$:

$$E_{i+1} X_i G Y_i F_{i+1} = E_{i+1} G_i F_{i+1}. \tag{B.5}$$

By (B.1), the right side of (B.5) is equal to $G_{i+1}$. Thus

$$(E_{i+1} X_i) G (Y_i F_{i+1}) = G_{i+1}. \tag{B.6}$$

But $E_{i+1} X_i = X_{i+1}$ by (B.2) and $Y_i F_{i+1} = Y_{i+1}$ by (B.3), so that (B.6) becomes simply $X_{i+1} G Y_{i+1} = G_{i+1}$, which proves (B.4) by induction ∎

If we specialize (B.4) with $i = N$, we get

$$X_N G Y_N = \Gamma, \tag{B.7}$$

which is the desired "extended Smith diagonalization" of $G$.

A convenient way to implement the extended Smith algorithm is to extend $G$ to dimensions $(n + k) \times (n + k)$ as follows:

$$G' = \begin{pmatrix} G & I_k \\ I_n & 0_{n \times k} \end{pmatrix}. \tag{B.8}$$

Then if the sequence of elementary row and column operations generated by the Smith algorithm applied to $G$ are performed on the extended matrix $G'$, after $i$ iterations, the resulting matrix $G'_i$ has the form

$$G'_i = \begin{pmatrix} G_i & X_i \\ Y_i & 0_{n \times k} \end{pmatrix}, \tag{B.9}$$

Thus after $N$ steps, the matrices $X_N$ and $Y_N$ in (B.7) appear as the upper right and lower left blocks of $G'_N$, respectively. We illustrate all this by "extending" Example B.1.

**B.3 Example:** We begin as we did in Example B.1, with the following $2 \times 4$ matrix:

$$G = G_0 = \begin{pmatrix} 1 & 1+D+D^2 & 1+D^2 & 1+D \\ D & 1+D+D^2 & D^2 & 1 \end{pmatrix}.$$

Then the corresponding matrix $G'$ (cf. Eq. (B.8)) is

$$G' = G'_0 = \begin{pmatrix} 1 & 1+D+D^2 & 1+D^2 & 1+D & 1 & 0 \\ D & 1+D+D^2 & D^2 & 1 & 0 & 1 \\ 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \end{pmatrix}$$

In Example B.1, we obtained $G_1$, $G_2$, and $G_3$ from $G_0$ by successively adding $(1+D+D^2)$ times column 1 to column 2, $(1+D^2)$ times column 1 to column 3, and $(1+D)$ times column 1 to column 4. If we perform the same sequence of operations to $G'_0$, we then obtain

$$G'_1 = \begin{pmatrix} 1 & 0 & 1+D^2 & 1+D & 1 & 0 \\ D & 1+D^3 & D^2 & 1 & 0 & 1 \\ 1 & 1+D+D^2 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \end{pmatrix}$$

$$G'_2 = \begin{pmatrix} 1 & 0 & 0 & 1+D & 1 & 0 \\ D & 1+D^3 & D+D^2+D^3 & 1 & 0 & 1 \\ 1 & 1+D+D^2 & 1+D^2 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \end{pmatrix}$$

$$G'_3 = \begin{pmatrix} 1 & 0 & 0 & 0 & 1 & 0 \\ D & 1+D^3 & D+D^2+D^3 & 1+D+D^2 & 0 & 1 \\ 1 & 1+D+D^2 & 1+D^2 & 1+D & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \end{pmatrix}$$

Next, adding $D$ times row 1 to row 2, we obtain

$$G'_4 = \begin{pmatrix} 1 & 0 & 0 & 0 & 1 & 0 \\ 0 & 1+D^3 & D+D^2+D^3 & 1+D+D^2 & D & 1 \\ 1 & 1+D+D^2 & 1+D^2 & 1+D & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \end{pmatrix}.$$

Interchanging columns 2 and 4, we obtain

$$G_5' = \begin{pmatrix} 1 & 0 & 0 & 0 & 1 & 0 \\ 0 & 1+D+D^2 & D+D^2+D^3 & 1+D^3 & D & 1 \\ 1 & 1+D & 1+D^2 & 1+D+D^2 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \end{pmatrix}.$$

Finally, adding $D$ times column 2 to column 3 and $(1+D)$ times column 2 to column 4, we compute successively

$$G_6' = \begin{pmatrix} 1 & 0 & 0 & 0 & 1 & 0 \\ 0 & 1+D+D^2 & 0 & 1+D^3 & D & 1 \\ 1 & 1+D & 1+D & 1+D+D^2 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & D & 0 & 0 & 0 \end{pmatrix}$$

and

$$G_7' = \begin{pmatrix} 1 & 0 & 0 & 0 & 1 & 0 \\ 0 & 1+D+D^2 & 0 & 0 & D & 1 \\ 1 & 1+D & 1+D & D & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & D & 1+D & 0 & 0 \\ . & & & & & \end{pmatrix}$$

Thus the extended Smith decomposition of the original matrix $G$ is given by

$$\begin{pmatrix} 1 & 0 \\ D & 1 \end{pmatrix} \cdot G \cdot \begin{pmatrix} 1 & 1+D & 1+D & D \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & D & 1+D \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1+D+D^2 & 0 & 0 \end{pmatrix}.$$

We will use this decomposition in the analysis of the $(4,2)$ convolutional code generated by $G$, in Example B.5. ▪

Now we will see how the extended Smith algorithm can be used to efficiently transform an arbitrary generator matrix for an $(n,k)$ convolutional code $\boldsymbol{C}$ into a noncatastrophic, basic, and ultimately a canonical, encoder for $\boldsymbol{C}$. We will also see how to find a polynomial inverse for a basic encoder for $\boldsymbol{C}$, and a basic encoder for the dual code $\boldsymbol{C}^{\perp}$.

Assume then that $G$ is a $k \times n$ polynomial generator matrix (we assume $\mathrm{rank}(G) = k$) for a convolutional code $\boldsymbol{C}$ over a finite field $F$. Since $F[D]$ is a Euclidean domain, we may apply the extended Smith algorithm described above, thereby obtaining a decomposition of the form (B.7). In what follows, the matrices $X_N$ and $Y_N$ produced by the extended Smith algorithm will be denoted simply by $X$ and $Y$.

The matrices $X$, $Y$, and $\Gamma$, contain much valuable information about the code $\boldsymbol{C}$ and the generator matrix $G$. To extract this information, however, we need to define several useful "pieces" of these matrices, which we call $\Gamma_k$, $\widetilde{\Gamma}_k$, $K$, and $H$ (in what follows, $r = n - k$):

$(B.10)$ $\Gamma_k$ = leftmost $k$ columns of $\Gamma = \mathrm{diag}(\gamma_1, \ldots, \gamma_k)$. *(dimensions: $k \times k$.)*

$(B.11)$ $\widetilde{\Gamma}_k = \gamma_k \cdot \Gamma_k^{-1} = \mathrm{diag}(\gamma_k/\gamma_1, \ldots, \gamma_k/\gamma_k)$. *(dimensions: $k \times k$.)*

$(B.12)$ $K$ = leftmost $k$ columns of $Y$. *(dimensions: $n \times k$.)*

$(B.13)$ $H$ = rightmost $r$ columns of $Y$. *(dimensions: $n \times r$.)*

The following theorem describes the useful "outputs" of the extended Smith algorithm, when applied to $G$.

**B.4 Theorem.** *With the matrices $\Gamma_k$, $\widetilde{\Gamma}_k$, $K$, and $H$ defined as in (B.10)–(B.13), we have the following:*

(a) *A basic encoder for $\boldsymbol{C}$:* $G_{\mathrm{basic}} = \Gamma_k^{-1} XG$. (That is, $G_{\mathrm{basic}}$ is obtained by dividing the $i$th row of $XG$ by the invariant factor $\gamma_i$, for $i = 1, \ldots, k$.)

(b) *A polynomial inverse for $G_{\mathrm{basic}}$ :* $K$

(c) *A polynomial pseudo-inverse for $G$, with factor $\gamma_k$:* $K\widetilde{\Gamma}_k X$. (In particular, if $G$ is already basic, i.e., if $\Gamma_k = I_k$, then $KX$ is a polynomial inverse for $G$.)

(d) *A basic encoder for $\boldsymbol{C}^\perp$, i.e., parity-check matrix for $\boldsymbol{C}$:* $H^T$.

**Proof:** From (B.7) with $X = X_N$ and $Y = Y_N$, we have

$$XG = \Gamma B \qquad \text{where } B = Y^{-1}. \tag{B.14}$$

Now $B$ is an $n \times n$ unimodular polynomial matrix. Let $U$ be the $k \times n$ matrix consisting of the first $k$ rows of $B$, and $L$ be the $r \times n$ matrix consisting of the last $r$ rows of $B$. Then since $\Gamma = (\,\Gamma_k \quad 0_{k\times r}\,)$, where "$0_{k\times r}$" is a $k \times r$ all-zeros matrix, it follows that

$$\Gamma B = (\,\Gamma_k \quad 0_{k\times r}\,) \begin{pmatrix} U \\ L \end{pmatrix} = \Gamma_k U, \tag{B.15}$$

so that, combining (B.14) and (B.15), we have

$$U = \Gamma_k^{-1} XG. \tag{B.16}$$

From (B.16) we see that $U$ is equivalent to $G$. Furthermore, since $U = (\,I_k \quad 0_{k\times r}\,)\, B$, it follows that $U$ is basic. This proves (a).

To prove (b), we use the fact that $Y$ and $B$ are inverse matrices, so that

$$(B.17) \qquad BY = \begin{pmatrix} U \\ L \end{pmatrix} ( K \quad H ) = \begin{pmatrix} UK & UH \\ LK & LH \end{pmatrix} = I_n = \begin{pmatrix} I_k & 0_{k\times r} \\ 0_{r\times k} & I_r \end{pmatrix}.$$

It follows from (B.17) that $UK = I_k$, so that $K$ is a polynomial inverse for the basic encoder $U$, which proves (b).

To prove (c), suppose that $J$ is an $n \times k$ polynomial pseudo-inverse for $G$ with (polynomial) factor $\psi(D)$, i.e., that $GJ^T = \psi I_k$. Then we have

$$(B.18) \qquad XGYY^{-1}JX^{-1} = XGJX^{-1} = X(\psi I_k)X^{-1} = \psi I_k.$$

But from (B.7) we know that $XGY = \Gamma = ( \Gamma_k \quad 0_{k\times r)} )$. Thus if we denote the matrix $Y^{-1}JX^{-1}$ by $J'$, we note that $J'$ is a polynomial matrix (since $X$ and $Y$ are unimodular) and (B.18) becomes

$$(B.19) \qquad ( \Gamma_k \quad 0 ) J' = \psi I_k.$$

If now $J' = \begin{pmatrix} J'_0 \\ J'_1 \end{pmatrix}$ where $J'_0$ is $k \times k$ and $J'_1$ is $r \times k$, it follows from (4.10) that $J'_0 = \psi\Gamma_k^{-1}$. But since $J'$ is a polynomial matrix, and $\Gamma_k^{-1} = \mathrm{diag}(\gamma_1^{-1}, \ldots, \gamma_k^{-1})$, it follows that $\psi$ must be a multiple of $\gamma_k$. Conversely, if we take $\psi = \gamma_k$, so that $J'_0 = \gamma_k\Gamma_k^{-1} = \widetilde{\Gamma}_k$, and $J'_1 = 0_{r\times k}$, then $J' = \begin{pmatrix} \widetilde{\Gamma}_k \\ 0_{r\times k} \end{pmatrix}$. Thus $J = YJ'X = K\widetilde{\Gamma}_k X$ is a polynomial pseudo-inverse with factor $\gamma_k$, which proves (c).

To prove (d), we have to show that the set of codewords in the code generated by $G$, i.e., the set of vectors $\boldsymbol{x}$ of the form $\boldsymbol{x} = \boldsymbol{u}G$, is identical to the set of vectors $\boldsymbol{x}$ such that $\boldsymbol{x}H = 0$. Thus let $X_1 = \{\boldsymbol{x} : \boldsymbol{x} = \boldsymbol{u}G\}$, and $X_2 = \{\boldsymbol{x} : \boldsymbol{x}H = 0\}$. We will show that $X_1 = X_2$.

Since as we have seen in part (a), the matrix $U$, defined to be the first $k$ rows of $B$, is a basic encoder equivalent to $G$, it follows that $X_1 = \{\boldsymbol{x} : \boldsymbol{x} = \boldsymbol{u}U\}$. Thus if $\boldsymbol{x} \in X_1$, $\boldsymbol{x} = \boldsymbol{u}U$ for some $\boldsymbol{u}$ and so $\boldsymbol{x}H = (\boldsymbol{u}U)H = \boldsymbol{u}(UH) = 0$, since by (B.17), $UH = 0_{k\times r}$. Thus $X_1 \subseteq X_2$. On the other hand, if $\boldsymbol{x}$ is arbitrary, then since $B^{-1} = Y$, we have

$$(B.20) \quad \boldsymbol{x} = \boldsymbol{x}I_n = \boldsymbol{x}(YB) = \boldsymbol{x}((\, K \quad H\,) \begin{pmatrix} U \\ L \end{pmatrix}) = \boldsymbol{x}(KU + HL) = (\boldsymbol{x}K)U + (\boldsymbol{x}H)L.$$

Now suppose $\boldsymbol{x} \in X_2$, i.e., $\boldsymbol{x}H = 0$. Then from (B.20), we have $\boldsymbol{x} = (\boldsymbol{x}K)U = \boldsymbol{u}U$, where $\boldsymbol{u} = \boldsymbol{x}K$, and so $\boldsymbol{x} \in X_1$. Thus $X_2 \subseteq X_1$, which completes the proof of part (d). ▪

**B.5 Example:** To illustrate the results of this appendix, we consider the following generator matrix $G$ for a $(4, 2)$ binary convolutional code:

$$G = \begin{pmatrix} 1 & 1+D+D^2 & 1+D^2 & 1+D \\ D & 1+D+D^2 & D^2 & 1 \end{pmatrix}.$$

In Example B.3, we found the extended invariant-factor decomposition of $G$ to be $XGY = \Gamma$, where

$$\Gamma = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1+D+D^2 & 0 & 0 \end{pmatrix}$$
$$X = \begin{pmatrix} 1 & 0 \\ D & 1 \end{pmatrix}$$
$$Y = \begin{pmatrix} 1 & 1+D & 1+D & D \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & D & 1+D \end{pmatrix}.$$

Thus the matrices defined in (B.10)–(B.13) are as follows.

$$\Gamma_k = \begin{pmatrix} 1 & 0 \\ 0 & 1+D+D^2 \end{pmatrix}$$
$$\widetilde{\Gamma}_k = \begin{pmatrix} 1+D+D^2 & 0 \\ 0 & 1 \end{pmatrix}$$
$$K = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 1+D & 0 & 0 & 1 \end{pmatrix}^T$$
$$H = \begin{pmatrix} 1+D & 0 & 1 & D \\ D & 1 & 0 & 1+D \end{pmatrix}^T$$

Using the prescriptions in Theorem B.4, we now quickly obtain the following.

- *A basic encoder for* $\boldsymbol{C}$:

$$G_{\text{basic}} = \Gamma_k^{-1} X G = \begin{pmatrix} 1 & 1+D+D^2 & 1+D^2 & 1+D \\ 0 & 1+D & D & 1 \end{pmatrix}.$$

- *A polynomial inverse for* $G_{\text{basic}}$ :

$$K = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 1+D & 0 & 0 & 1 \end{pmatrix}^T.$$

- *A polynomial pseudo-inverse for* $G$, *with factor* $\gamma_2 = 1+D+D^2$:

$$K\widetilde{\Gamma}_k X = \begin{pmatrix} 1 & 0 & 0 & D \\ 1+D & 0 & 0 & 1 \end{pmatrix}^T.$$

- *A (basic) encoder for* $\boldsymbol{C}^\perp$:

$$H^T = \begin{pmatrix} 1+D & 0 & 1 & D \\ D & 1 & 0 & 1+D \end{pmatrix}.$$

Finally, we note that in this case neither of the basic encoders $G_{\text{basic}}$ or $H$ is canonical, since they are not reduced. To reduce them, we can follow the simple algorithm originally described in [13], or perhaps more lucidly in [29, Sec. 6.3.2], (where canonical matrices are however called "row-reduced"), which is based on part (1) of Theorem A.2. For example, to reduce $G_{\text{basic}}$, we first write down the indicator matrix for the highest-degree terms in each row, which is

$$\overline{G}_{\text{basic}} = \begin{pmatrix} 0 & 1 & 1 & 0 \\ 0 & 1 & 1 & 0 \end{pmatrix}.$$

This matrix does not have full rank, which means that the external degree of $G_{\text{basic}}$ can be reduced by an elementary row operation. Indeed, we can multiply the second row by $D$ and add it to the first row, obtaining

$$G'_{\text{basic}} = \begin{pmatrix} 1 & 1 & 1 & 1 \\ 0 & 1+D & D & 1 \end{pmatrix}.$$

The highest-degree indicator matrix for $G'_{\text{basic}}$ is

$$\overline{G}'_{\text{basic}} = \begin{pmatrix} 1 & 1 & 1 & 1 \\ 0 & 1 & 1 & 0 \end{pmatrix},$$

which does have full rank, so by Theorem A.2, part (1), $G'_{\text{basic}}$ is reduced, and is therefore a canonical generator matrix for the code. Similarly, to reduce $H$, we simply add row 2 to row 1, thereby obtaining

$$H_{\text{can}} = \begin{pmatrix} 1 & 1 & 1 & 1 \\ D & 1 & 0 & 1+D \end{pmatrix},$$

which is, again by Theorem A.2, a reduced and therefore canonical generator matrix for the dual code.∎

**References.**


1. K. Abdel-Ghaffar, R. J. McEliece, and G. Solomon, "Some partial unit memory convolutional codes," *JPL TDA Progress Report*, vol. 42-107 (November 1991), pp. 57–72. Also see Proc. ISIT 1991, p. 196.

2. T. M. Apostol, *Calculus, Vol. II, 2nd Ed.* New York: John Wiley & Sons, 1969.

3. E. Artin, *Algebraic Numbers and Algebraic Functions.* New York: Gordon amd Breach, 1967.

4. G. Bégin and D. Haccoun, "High-rate punctured convolutional codes: structure properties and construction techniques," *IEEE Trans. Comm.*, vol. COM-37 (December 1989), pp. 1381–1385.

5. R. E. Blahut, *Theory and Practice of Error-Correcting Codes.* Reading, Mass.: Addison-Wesley, 1983.

6. A.E. Brouwer and T. Verhoeff, "An updated table of minimum-distance bounds for binary linear codes," *IEEE Trans. Inform. Theory* vol. IT-39 (March 1993), pp. 662–677.

7. J. B. Cain, G. C. Clark, and J. M. Geist, "Punctured convolutional codes of rate $(n-1)/n$ and simplified maximum likelihood decoding," *IEEE Trans. Inform. Theory*, vol. IT-25 (January 1979), pp. 97–100.

8. D. G. Daut, J. W. Modestino, and L. D. Wismer, "New short constraint length convolutional code construction for selected rational rates," *IEEE Trans. Inform. Theory*, vol. IT-28 (September 1982), pp. 794–800.

9. A. Dholakia, *Introduction to Convolutional Codes with Applications.* Boston: Kluwer Academic Publishers, 1994.

10. B. W. Dickinson, "A new characterization of canonical convolutional encoders," *IEEE Trans. Inform. Theory*, vol. IT-22 (May 1976), pp. 352–354.

11. P. Elias, "Coding for noisy channels," *1955 IRE International Convention Record* (part 4), pp. 37–46.

12. R. M. Fano, "A heuristic discussion of probabilistic decoding," *IEEE Trans. Inform. Theory*, vol. IT-9 (April 1963). pp. 64–74.

13. G. D. Forney, Jr., "Convolutional codes I: algebraic structure," *IEEE Trans. Inform. Theory*, vol. IT-16 (November 1970), pp. 720–738.

14. G. D. Forney, Jr., "Correction to 'Convolutional Codes I: Algebraic Structure,' " *IEEE Trans. Inform. Theory*, vol. IT-17 (May 1971), p. 360.

15. G. D. Forney, Jr., "The Viterbi algorithm," *Proc. IEEE*, vol. 61 (March 1973), pp. 268–276.

16. G. D. Forney, Jr., "Structural analysis of convolutional codes via dual codes," *IEEE Trans. Inform. Theory*, vol. IT-19 (July 1973), pp. 512–518.

17. G. D. Forney, Jr., "Minimal bases of rational vector spaces with applications to multivariable linear systems," SIAM J. Control 13 (1975), pp. 493–502.

18. G. D. Forney, Jr., "Algebraic structure of convolutional codes, and algebraic system theory," in *Mathematical System Theory*, A. C. Antoulas, ed., pp. 527–557. Berlin: Springer-Verlag, 1991.

19. G. D. Forney, Jr., R. Johannesson, and Zhe-xian Wan, "Minimal and canonical rational generator matrices for convolutional codes," submitted to *IEEE Trans. Inform. Theory*, December 1995.

20. F. R. Gantmacher, *The Theory of Matrices, Vol. I, 2nd Ed.* New York: Chelsea, 1959.

21. D. Haccoun and G. Bégin, "High-rate punctured convolutional codes for Viterbi and sequential decoding," *IEEE Trans. Comm.*, vol. COM-37 (November 1989). pp. 1113–1125.

22. D. W. Hagelbarger, "Recurrent codes: Easily mechanized, burst-correcting binary codes," *Bell Syst. Tech. J.*, vol. 38 (July 1959), pp. 969–984.

23. D. Hilbert, "Ueber die Theorie der algebraischen Formen," *Math. Ann.*, vol. 36(1890), pp. 471-534.

24. K. Hoffman and R. Kunze, *Linear Algebra.* Englewood Cliffs, N. J. : Prentice-Hall, 1961.

25. R. A. Horn and C. R. Johnson, *Matrix Analysis*. Cambridge: Cambridge University Press, 1985.

26. F. Jelinek, "A fast sequential decoding algorithm using a stack," *IBM J . Res. Dev.*, vol. 13 (November 1969), pp. 675–685.

27. R. Johannesson and Z.-x. Wan, "A linear algebra approach to minimal convolutional encoders," *IEEE Trans. Inform. Theory*, vol. IT-39 (July 1993), pp. 1219–1233.

28. R. Johannesson and Z.-x. Wan, "On canonical encoding matrices and the generalized constraint lengths for convolutional codes," pp. 187–199 in *Communications and Cryptography*, R. E. Blahut et al., eds. Boston: Kluwer, 1994.

29. T. Kailath. *Linear Systems.* Englewood Cliffs, N. J.: Prentice Hall, 1980.

30. D. E. Knuth, *The Art of Computer Algorithms, Vol. 1: Fundamental Algorithms.* Reading, Mass.: Addison-Wesley, 1969.

31. D. E. Knuth, *The Art of Computer Algorithms, Vol. 2: Seminumerical Algorithms (2nd Ed.)*. Reading, Mass.: Addison-Wesley, 1981.

32. K. J. Larsen, "Short convolutional codes with maximal free distance for rates 1/2, 1/3, and 1/4, *IEEE Trans. Inform. Theory*, vol. IT-18 (May 1973), pp. 371–372.

33. G. S. Lauer, "Some optimal partial-unit-memory codes," *IEEE Trans. Inform. Theory*, vol. IT-25 (March 1979), pp. 240–243.

34. L.–N. Lee, "Short unit-memory byte-oriented binary convolutional codes having maximal free distance," *IEEE Trans. Inform. Theory*, vol. IT-22 (May 1976), pp. 349–352.

35. S. Lin and D. J. Costello, *Error-Control Coding—Fundamentals and Applications*. Englewood Cliffs, N. J.: Prentice-Hall, 1983.

36. K. Lumbard and R. J. McEliece, "Counting minimal generator matrices," *Proc. 1994 IEEE Inter. Symp. Inform. Theory*, (Trondheim, Norway, June 1994), p. 18.

37. J. L. Massey, *Threshold Decoding*. Cambridge, Mass.: M.I.T. Press, 1963.

38. J. L. Massey and M. K. Sain, "Codes, automata, and continuous systems: explicit interconnections." *IEEE Trans. Automatic Control*, vol. AC-12 (December 1967), pp. 644–649.

39. J. L. Massey and M. K. Sain, "Inverses of linear sequential circuits," *IEEE Trans. Comp.*, vol. COM-17 (1968), pp. 330–337.

40. J. L. Massey, *Error Bounds for Tree Codes, Trellis Codes, and Convolutional Codes, with Encoding and Decoding Procedures*, in *Coding and Complexity*, G. Longo, ed., CISM Courses and lectures no. 216. Wien-New York: Springer-Verlag, 1977.

41. R. J. McEliece, *The Theory of Information and Coding*. Reading, Mass.: Addison-Wesley, 1977.

42. R. J. McEliece, *Finite Fields for Computer Scientists and Engineers*. Boston: Kluwer, 1987.

43. R. J. McEliece and I. Onyszchuk, "The analysis of convolutional codes via the extended Smith algorithm," *Jet Propulsion Laboratory TDA Progress Report*, vol. 42–112 (February 15, 1993), pp. 22–30. See also Proc.1993 International Symposium Information Theory, pp. 142.

44. R. J. McEliece and R. P. Stanley, "The general theory of convolutional codes," *Jet Propulsion Laboratory TDA Progress Report*, vol. 42–113 (May 15, 1993), pp. 89–98.

45. J. P. Odenwalder, "Optimal decoding of convolutional codes," Ph.D. thesis, UCLA System Sciences Dept., Los Angeles, 1970.

46. J. K. Omura, "On the Viterbi decoding algorithm," *IEEE Trans. Inform. Theory*, vol. IT-15 (January 1969), pp. 177–179.

47. E. Paaske, "Short binary convolutional codes with maximal free distance," *IEEE Trans. Inform. Theory*, vol. IT-20 (September 1974), pp. 683–688.

48. P. Piret, *Convolutional Codes: An Algebraic Approach*. Cambridge, Mass.: The MIT Press, 1988.

49. H. J. S. Smith, "On systems of linear indeterminate equations and congruences,"

*Philos. Trans. Royal Soc. London* vol. 151 (1861), pp. 293–326.

50. R. P. Stanley, "Hilbert functions of graded algebras," *Advances in Math.* vol. 28 (1978), pp. 57–83.

51. R. P. Stanley, *Enumerative Combinatorics, Vol. I.* Monterey, Calif.: Wadsworth and Brooks-Cole, 1986.

52. P. P. Vaidyanathan, *Multirate Systems and Filter Banks.* Englewood Cliffs, N. J.: Prentice Hall, 1993.

53. P. P. Vaidyanathan and T. Chen, "Role of anticausal inverses in multirate filterbanks—Part I: System-theoretic fundamentals," *IEEE Trans. Signal Processing*, vol. 43 (May 1995), pp. 1090–1102.

54. A. J. Viterbi, "Error bounds for convolutional codes and an asymptotically optimum decoding algorithm," *IEEE Trans. Inform. Theory*, vol. IT-13 (April 1967), pp. 260–269.

55. B. L. van der Waerden, *Modern Algebra, vol. 1.* New York: Frederick Ungar, 1953.

56. B. L. van der Waerden, *Algebra, vol. 2.* New York: Frederick Ungar, 1970.

57. S. B. Wicker, *Error Control Systems.* Englewood Cliffs, N.J.: Prentice–Hall, 1995.

58. J. M. Wozencraft and B. Reiffen, *Sequential Decoding.* Cambridge, Mass.: The M.I.T. Press, 1961.

59. A. D. Wyner and R. B. Ash, "Analysis of recurrent codes," *IEEE Trans. Inform. Theory*, vol. IT-9 (1963), pp. 143–156.

60. Y. Yasuda, K. Kashiki, and Y. Hirata, "High-rate punctured convolutional codes for soft decision Viterbi decoding," *IEEE Trans. Comm.*, vol. COM-32 (March 1984). pp. 315–318.

61. Ø. Ytrehus, "A note on high rate binary convolutional codes." University of Bergen *Report in Informatics no. 68* (August 1992).

62. Ø. Ytrehus, "Ascetic convolutional codes," pp. 382–390 in Proc. 33rd Allerton Conference on Communication, Control, and Computing, October 1995.