# LINEAR SYSTEMS ANALYSIS AND DECODING OF CONVOLUTIONAL CODES

A Dissertation

Submitted to the Graduate School

of the University of Notre Dame

in Partial Fulfillment of the Requirements

for the Degree of

Doctor of Philosophy

by

Brian Michael Allen, B.A., M.A., M.S.

Joachim Rosenthal, Director

Department of Mathematics

Notre Dame, Indiana

June 1999

## ABSTRACT

In this dissertation the tools of linear systems theory are used to study convolutional codes and to develop decoding algorithms for them.

In particular, the input-state-output representation of a convolutional code is examined. Properties of this representation are explored and its connection with various other representations is detailed. Notable among these are the connections between the syndrome former matrix and the local description of codewords obtained via the input-state-output representation. Also, for codes with rate $k/n \leq 1/2$, the output-state-input representation is developed and the close connections with the input-state-output description are detailed. These connections are then used to enhance certain algebraic decoding schemes for convolutional codes.

Turbo codes are also given a brief treatment. A linear systems representation of these codes is developed and a few remarks on the decoding of turbo codes utilizing this representation are given.

In a much broader application of the input-state-output representation, a matrix Euclidean algorithm is developed. Not only does this algorithm efficiently decide if a given convolutional encoder is catastrophic, it can compute the greatest common left divisor of the encoder matrix in a straightforward manner. This has applications in many areas including finding minimal bases of rational vector spaces, obtaining irreducible matrix fraction descriptions of transfer functions and for obtaining a basis for the free module generated by the columns of the matrix.

Finally, the class of Reed-Solomon convolutional codes is presented. These codes are shown to possess maximum distance separable generator and parity check subcodes. This property, along with the ability to use Berlekamp-Massey decoding on the subcodes make these convolutional codes ideal for the algebraic decoding scheme presented in this dissertation. Other properties of these codes are then examined, including their column distance function. Some theoretical results, examples and open problems regarding this issue are presented.



## ACKNOWLEDGEMENTS

I need to thank the people who encouraged and supported my work toward this dissertation. First, my advisor, Joachim Rosenthal, has encouraged, enlightened and nurtured me as a mathematician. For that, I will always be grateful. I have benefitted greatly from the friendship and advice of my colleagues including Eric York, Paul Weiner, Steve Walk, Jeff Igo, Chris Monico and Roxana Smarandache.

I would like to thank the members of my defense committee, Heide Glüsing-Lüerßen, Amarjit Budhiraja and Thomas Fuja, for their time, effort and observations. I would like to thank all of the faculty, staff and graduates students of the Mathematics Department at the University of Notre Dame. I would especially like to thank Alex Hahn and Juan Migliore for all they have done for me. I am also very grateful to Daniel Costello Jr. for all his advice and help.

I need to acknowledge all of the generous institutions which have supported my work. The Department of Mathematics and the Graduate School of the University of Notre Dame have provided an extraordinary learning environment. I am indebted to the Arthur J. Schmitt Foundation and to the Center for Applied Math at Notre Dame for their generous fellowships. I must also thank the Institute for Mathematics and its Applications and the National Science Foundation for their support.

Thanks must be given to my wonderful wife who has supported and encouraged me to an extent that I can never repay. I would also like to thank my family for all they have given me. I will never be able to show them just how much I love them.

I also would like to thank the Lady on The Dome and the entire Notre Dame family for a truly enriching and wonderful experience.

CONTENTS


CHAPTER 1 INTRODUCTION ... 2
1.1 Communication, Coding and Shannon ... 2
1.2 Block Codes ... 3
1.3 Convolutional Codes ... 4
1.4 Organization of this Dissertation ... 4

CHAPTER 2 LINEAR SYSTEMS AND CONVOLUTIONAL CODES ... 6
2.1 Behaviors ... 6
2.2 Realization ... 7
2.3 Special Properties of the ISO Representation ... 9
2.4 An Algebraic Model for Turbo Codes ... 11

CHAPTER 3 CONNECTIONS BETWEEN REPRESENTATIONS ... 15
3.1 Generator Matrices and Sequences ... 15
3.2 Parity Check Matrices ... 16
3.3 Shift Registers ... 17
3.4 From ISO Representations to Generator Matrices ... 18
3.5 From the Local Description to Syndrome Formers ... 18

CHAPTER 4 A MATRIX EUCLIDEAN ALGORITHM ... 21
4.1 Background Material ... 21
4.2 A Brief History of the Problem ... 22
4.3 A Realization Algorithm ... 22
4.4 The Controllability Space ... 24
4.5 The Refining Algorithm ... 26
4.6 The Situation of Constant Rows ... 27
4.7 The Algorithm ... 28
4.8 Examples ... 29

CHAPTER 5 DECODING ERROR CORRECTING CODES ... 32
5.1 Channel Models and Maximum Likelihood Decoding ... 32
5.2 Decoding Parameters ... 33
5.3 A Few Decoding Algorithms ... 34
5.4 Majority Logic Decoding of Convolutional Codes ... 34
5.5 Using the Local Description for Feedback Decoding ... 36

CHAPTER 6 ALGEBRAIC DECODING OF CONVOLUTIONAL CODES USING THE LOCAL DESCRIPTION ... 38
6.1 A Basic Algorithm ... 38
6.2 Some Classes of Binary Input-State-Output Convolutional Codes ... 40
6.3 Example Codes. ... 41
6.4 An Alternative to State Estimation ... 42
6.5 Some Notes on the State Elimination Algorithm ... 42
6.6 Enhanced State Estimation ... 43
6.7 Analysis of the Enhanced Algorithm ... 43
6.8 An Algebraic Look at Decoding Turbo Codes ... 45

CHAPTER 7 MDS CONVOLUTIONAL CODES ... 47
7.1 Reed-Solomon Convolutional Codes ... 47
7.2 Further Properties of Reed-Solomon Convolutional Codes ... 48
7.3 Some Insights into the Conjecture ... 50
7.4 The Case when $\delta = 2$ ... 52
7.5 The Road Ahead and a Stronger Conjecture ... 54

BIBLIOGRAPHY ... 57

CHAPTER 1

INTRODUCTION

## 1.1 Communication, Coding and Shannon

One may reasonably define communication as the conveyance, reliable or otherwise, of information. Reliable and efficient communication is becoming an increasingly indispensable tool of the modern world. Indeed, as we are firmly entrenched in the "information age" our society is utterly dependent on reliable communication. Evidence of this dependence can be seen everywhere. Financial markets rely on computers to accurately receive and oversee trading activity. Banking transactions are "wired" instead of delivered by armored cars. Credit cards are verified automatically over phone lines or the internet for each purchase. All of these things involve communication in one form or another. Whether it is a person entering information into a computer, a computer displaying information to a person, two computers sharing data, or simply two people communicating with each other; all are forms of communication.

Of course, communication has come a long way. The earliest forms of communications between humans probably involved some sort of gesturing and primitive oral languages. These evolved into written languages with grammar and syntax. Today, communication has moved into the digital age. Information is encoded as sequences of 0's and 1's to allow for virtually instantaneous micro-chip controlled communication.

Reliability in communication has always been important. Indeed almost all communication methods have some inherent error tolerances. In oral language, it is possible, to a large extent, for two people speaking the same language but with different accents and even dialects to communicate very well. In written language, errors in spelling are often easily corrected by the reader. For example, if one encounters the passage "The chilwren did very well in school", one could assume that "children" was the intended word. In fact, my automated spell-checker has been urging me to make that very change.

Modern digital communication also has these tolerances for error. It comes in the form of error correcting codes. A reasonable definition of an error correcting code is a pair $(\mathcal{M}, \mathcal{R})$ and two maps associated with these sets:

$$\psi : \mathcal{M} \to \mathcal{R} \quad \text{and} \quad \beta : \mathcal{R} \to \psi(\mathcal{M})$$

Here, $\mathcal{M}$ is the set of all possible messages, $\mathcal{R}$ is the set of all possible received messages, the map $\psi$ is injective and is called the encoder and the map $\beta$ is called the estimator or preliminary decoder. Given this framework, we can present a model of communication employing an error correcting code. Suppose the message $m \in \mathcal{M}$ is to be transmitted. Then we have the following situation.

$$m \overset{\psi}{\longmapsto} r \overset{noise}{\longmapsto} \tilde{r} \overset{\beta}{\longmapsto} \bar{r} \overset{\psi^{-1}}{\longmapsto} \bar{m}$$

This process can be explained as follows. The message $m$ is encoded under the map $\psi$ to the transmitted message $r$. As $r$ is transmitted over the communication channel it is subjected to various kinds of interference and distortion which is collectively labeled 'noise'. For example, thunderstorms often affect the quality of telephone connections and dust may affect the quality of sound produced by a compact disc. This noise may transform $r$ into a possibly different $\tilde{r} \in \mathcal{R}$. Thus, the distorted message $\tilde{r}$ is received and the decoder must map $\tilde{r}$ to a reasonable element, $\bar{r}$, of $\psi(\mathcal{M})$ using the estimator map $\beta$. Then, the encoding is reversed by applying $\psi^{-1}$ to obtain the decoded message $\bar{m} \in \mathcal{M}$. The transmission successfully communicates the message if and only if $m$ equals $\bar{m}$. It is one of the main focuses of coding theory to develop codes which achieve a high rate of successful communication over a particular channel.

**Example 1.1.1** Suppose two possible messages are to be sent. These messages can be represented in a digital communication system by 0 and 1. Hence, $\mathcal{M} = \{0, 1\}$. We can define $\psi$ by $\psi(0) = 00000$ and $\psi(1) = 11111$. We can define $\beta$ by the rule "estimate as 00000 if the received message has more 0's than 1's, and vice versa". Thus if 01011 is received, then $\beta(01011) = 11111$ and this is decoded as $\psi^{-1}(11111) = 1$.

If we know that for our channel, each bit that is transmitted will be received erroneously with a probability of 0.1, it can easily be computed that a message will be decoded incorrectly with a probability of 0.00856. This means that there will be 11 times fewer errors by using this code as opposed to using no code at all!

The tradeoff in this situation is that instead of having to transmit only one digit to send each message, five digits have to be sent. Over time, this can add up to a lot more time and money to transmit these messages. So the notion of the *rate* of the error correcting code must considered. The *rate* of a code is simply the ratio of the number of information bits (*i.e.* the length of a message, $m \in \mathcal{M}$) to the number of code bits (*i.e.* the length of $\psi(m)$). For the code of this example, the rate is $1/5$.

It was Claude Shannon in 1948 [64] who showed that the goal of finding error correcting codes that allowed for a high probability of successful transmission was possible. Shannon showed that each channel has a constant associated with it called the *channel capacity*. Furthermore, he showed that there exists error correcting codes that achieve a successful transmission with probability arbitrarily close to 1 with the rate of the code arbitrarily close to (but below) the channel capacity. His proof showed only the existence of such codes. There is little indication of how to obtain such codes. The construction of these codes, along with efficient decoding algorithms, is the goal of modern coding theory. Error correcting codes generally fall into two categories: block codes and convolutional codes.

### 1.2 Block Codes

Let $\mathcal{A}$ be a (finite) set of symbols, called the message alphabet. Define $\mathcal{M}$ to be the set consisting of all sequences of symbols from $\mathcal{A}$ of length $k$. Also, define $\mathcal{R}$ to be the set consisting of all sequences of symbols from $\mathcal{A}$ of length $n$. We define $k$ and $n$ to be positive integers with $k \leq n$.

**Definition 1.2.1** A *block code of rate $k/n$* over the alphabet $\mathcal{A}$ is defined as the pair $(\mathcal{M}, \mathcal{R})$ together with an injective map $\psi : \mathcal{M} \to \mathcal{R}$ and an estimator map $\beta : \mathcal{R} \to \psi(M)$.

An important special case of the above definition is when $\mathcal{A}$ is a finite field, $\mathbb{F}_q = GF(q)$. In this case, and when $\psi$ is a linear map $\psi : \mathbb{F}_q^k \to \mathbb{F}_q^n$, the code is called a *linear block code*.

From elementary linear algebra, we know that the map $\psi$ can be represented by a scalar matrix, $G$. This full rank $k \times n$ matrix is known as a generator matrix of the code. Using this generator matrix description, the encoding process for a message $m \in \mathbb{F}_q$ can be described as:

$$m \stackrel{\psi}{\longmapsto} mG$$

Using the above definitions, it is clear that for a linear block code, the set of codewords, $\psi(\mathcal{M})$, is a linear subspace of $\mathbb{F}_q^n$. Another way to describe this linear subspace is through the use of a kernel representation. Indeed, there exists a scalar matrix $H$ of size $(n-k) \times n$, called the parity check matrix, such that a vector $x \in F_q^n$ is a codeword if and only if $xH^T = 0$.

The code in Example 1.1.1 is a linear block code over $\mathbb{F}_2$. Indeed, a generator matrix is given by $G = [\, 1\ 1\ 1\ 1\ 1 \,]$. Also, a parity check matrix is given by

$$H = \begin{bmatrix} 1 & 1 & 0 & 0 & 0 \\ 1 & 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 & 1 \end{bmatrix}$$

**Remark 1.2.2** Although error correcting codes are defined as a pair of sets together with an encoder map and an estimator map, in practice, all of these may not be specified. Often, when the focus is not on decoding, the estimator map is omitted. When the code is linear, it is common practice to identify the code solely by a generator or parity check matrix. Further, by abuse of notation, a code is often specified simply by the set of codewords $\psi(\mathcal{M})$. Hence, identifying the linear subspace that comprises the set of codewords is enough to identify the code.

This discussion of block codes is ended with some basic definitions.

**Definition 1.2.3** The *Hamming weight* or simply *weight* of a vector in $\mathbb{F}^n$ is defined as the number of nonzero components of the vector. Notation: the weight of a vector $x$ is given by $\mathrm{wt}\,(x)$.

**Definition 1.2.4** The *Hamming distance* or simply *distance* between two vectors in $\mathcal{A}^n$ is the number of components in which they disagree. Notation: the distance between $x$ and $y$ is given by $\mathrm{dist}\,(x, y)$.

**Definition 1.2.5** The *distance of a code* is the minimum distance between any two codewords.

If the code is linear, then the distance is equal to the minimum weight of all the nonzero codewords.

### 1.3 Convolutional Codes

We now turn our attention to the other major class of error correcting codes. Suppose we have a rate $k/n$ linear block code over $\mathbb{F}_q$ with scalar generator matrix $G$. Assume we have a sequence of messages $u_0, u_1, u_2, \ldots, u_r$ from $\mathcal{M}$ that we wish to send. We could write this sequence as a polynomial over $\mathbb{F}_q^k$ in some indeterminate $s$ as follows:

$$u_0, u_1, u_2, \ldots, u_r \;\longleftrightarrow\; u_0 + u_1 s + u_2 s^2 + \cdots + u_r s^r$$

If we denote this polynomial by $u(s)$, then the encoding of the entire sequence can be written simply as $u(s) \mapsto u(s)G$, where the matrix multiplication is done on each coefficient of the polynomial.

It was Elias in 1955 [22] who suggested that the matrix G need not be scalar. In doing so, the notion of a convolutional code was born. The classical definition of convolutional codes can now be stated.

**Definition 1.3.1** [23, 52] A rate $k/n$ convolutional code is defined as a $k$ dimensional $\mathcal{F}$-linear subspace of $\mathcal{F}^n$, where $\mathcal{F}$ is either the field of rational functions $\mathbb{F}(s)$ or the field of formal Laurent series $\mathbb{F}((s))$. In either case, a $k \times n$ full rank matrix $G(s)$, with entries in $\mathbb{F}[s]$ will serve as a generator matrix. The highest degree polynomial occurring in $G(s)$ is called the *memory*, $m$, of the encoder. In short, the encoding of a message depends, not only on the bits to be sent at a certain time, but also on the bits sent in the previous $m$ time intervals. Of course, polynomial parity check matrices exist as well, and are sometimes referred to as syndrome formers.

It should be remarked that the notions of Hamming weight and distance are easily extended to convolutional codes. The block code notion of the distance of a code also is readily extended, but it takes on the special name of *free distance* when it is applied to convolutional codes.

Since convolutional codes can potentially have infinite message sequences, some unique situations may arise. In particular, it may happen that an infinite weight message sequence is encoded as a finite weight codeword. If this happens, then a finite number of errors during transmission can lead the estimator map to estimate this received word as the all zero sequence, which, of course, is a valid codeword. Hence, a finite number of transmission errors can lead to an infinite number of errors in the message word. This unfortunate situation is called *catastrophic encoding* and is to be avoided. Fortunately, the following proposition, proven by Massey and Sain [51] characterize this situation completely in terms of the generator matrix.

**Proposition 1.3.2** *A convolutional encoder $G(s)$ is non-catastrophic if and only if the greatest common divisor of its full size minors is $s^l$, for some non-negative integer $l$. If $l = 0$ the encoder is called observable.*

### 1.4 Organization of this Dissertation

An overview of this dissertation will now be presented. Chapter 2 will introduce some basic concepts of linear systems and how they can be used to define a convolutional code. The idea of turbo codes will be reviewed and a corresponding linear systems representation for these codes will be given.

Chapter 3 will discuss further properties of convolutional codes including the relationships between their various representations. In Chapter 4 the linear systems representation of convolutional codes is used to investigate catastrophic encoding. In the process, a matrix Euclidean algorithm will be developed and its many applications discussed. Some basic concepts of decoding error correcting codes will be presented in Chapter 5. Some of the major decoding schemes for linear block codes and convolutional codes will be given a brief survey. In Chapter 6 some algebraic decoding techniques for convolutional codes based on their linear systems representation are introduced. Various constructions of binary convolutional codes are developed to suit the algorithm. Some alternatives to the algebraic decoding scheme are put forward and their effectiveness in various situations is discussed and analyzed. In Chapter 7 Reed-Solomon convolutional codes are introduced and their effectiveness in one of the enhanced algebraic decoding schemes is shown. Exploring these codes further, the column distance functions of the codes will also be discussed. Examples and results will be given for lower complexities, and some open problems will be stated.

# CHAPTER 2

# LINEAR SYSTEMS AND CONVOLUTIONAL CODES

In this Chapter the basic notions of linear systems will be reviewed, starting with the concept of behaviors. These ideas are used to form an alternate definition of convolutional code. This definition is then refined by giving a first order representation and exploring its various properties. For a more thorough discussion of the systems theory which dominates the early part of this chapter, the reader is referred to [70, 36, 35, 68, 69, 37].

## 2.1 Behaviors

In this section the notion of dynamical systems and behaviors as well as some related concepts are introduced and briefly discussed. This is a review of the material contained in [58, 61, 71].

**Definition 2.1.1** A *dynamical system* $\Sigma$ is a triple $\Sigma = (T, \mathcal{A}, \mathfrak{B})$, where $T \subseteq \mathbb{R}$ is the time axis, $\mathcal{A}$ is the signal alphabet and $\mathfrak{B} \subseteq \mathcal{A}^T$ is called the *behavior*. The elements of $\mathfrak{B}$ are called trajectories.

We will work with these abstract notions only long enough to write down an alternate definition of convolutional code, so we will omit any illuminating examples of the above definitions.

For our purposes we will take the discrete time axis $T = \mathbb{Z}_+$. Let $\mathbb{F} = \mathbb{F}_q$ and let the signal space be $\mathcal{A} = \mathbb{F}^n$. Hence, our behaviors will consist of subsets of the set of one sided infinite sequences of vectors in $\mathbb{F}^n$.

**Definition 2.1.2** Define the *left shift operator,* $\sigma$, and the *right shift operator,* $\sigma^{-1}$, on the sequence space $\mathcal{A}^T$ by

$$\begin{aligned} \sigma(a_0, a_1, a_2, \dots) &= (a_1, a_2, a_3, \dots) \\ \sigma^{-1}(a_0, a_1, a_2, \dots) &= (0, a_0, a_1, a_2, \dots) \end{aligned}$$

A subset $\mathcal{C} \subseteq \mathcal{A}^T$ is right (left) shift invariant if $\sigma^{-1}\mathcal{C} \subseteq \mathcal{C}$ ( $\sigma\mathcal{C} \subseteq \mathcal{C}$ ). Also if for every element of $\mathcal{C}$, at most finitely many components are nonzero, we say $\mathcal{C}$ has *compact support.*

With these definitions in hand, we are led to the following rather cryptic definition of a convolutional code.

**Definition 2.1.3** A subset $\mathcal{C} \subseteq \mathcal{A}^T$ is called a convolutional code if $\mathcal{C}$ is linear (as a vector space over $\mathbb{F}_q$ with component-wise addition), right shift invariant and has compact support.

These requirements may seem odd, but are actually quite natural. Certainly, we have seen that a linear subspace is desired. The right shift invariance means only that a valid codeword is still a valid codeword if it is delayed before it is sent. The condition of finite support is new. Obviously this contradicts the classical definition of a convolutional code given in Definition 1.3.1. In practice this restriction has little effect since one would never want to send an infinite codeword. More academically, this can be justified by the lack of a widely accepted definition of convolutional code.

With the usual identification between finite sequences and polynomials, the following theorem translates the new definition of convolutional code into a more recognizable form.

**Theorem 2.1.4** *[71, Theorem 3.1.2] Let $\mathcal{C} \subseteq \mathcal{A}^T$. Then $\mathcal{C}$ is a convolutional code if and only if $\mathcal{C}$ is a $\mathbb{F}[s]$-submodule of $\mathbb{F}^n[s]$.*

**Corollary 2.1.5** *[71, Corollary 3.1.3] As a submodule of a free module (over a PID), $\mathcal{C}$ has a well defined rank $k$. Hence there exists an injective module homomorphism*

$$\begin{aligned} \psi : \mathbb{F}^k[s] &\longrightarrow \mathbb{F}^n[s] \\ u(s) &\longmapsto v(s) \end{aligned}$$

*Equivalently, there is a full rank $k \times n$ polynomial matrix $G(s)$ such that*

$$\mathcal{C} = \{ v(s) \mid \exists\, u(s) \in \mathbb{F}^k[s] : v(s) = u(s)G(s) \}$$

Let us introduce some terminology and notation. As before, the rate of the convolutional code is given by $k/n$. Given an encoder $G(s)$, the maximum degree polynomial occurring in row $i$ is called the *row degree* of row $i$ and denoted by $\nu_i$. After a possible reordering of the rows, we will assume that $\nu_1 \geq \nu_2 \geq \ldots \geq \nu_k$. As before, the *memory* of the encoder, $m$, is equal to $\nu_1$. Of course, reordering the encoder might seem to change the code it defines, so let us resolve some issues of uniqueness with regard to encoders with the following proposition.

**Proposition 2.1.6** *[71, Lemma 3.1.6] Two encoders, $G_1(s)$ and $G_2(s)$, define the same convolutional code if and only if there exists a $k \times k$ unimodular matrix $U(s)$ such that $U(s)G_1(s) = G_2(s)$.*

The *complexity of an encoder*, $\delta(G(s))$, is given as the sum of the row degrees, $\delta(G(s)) = \Sigma\nu_i$. However, in light of the above proposition, there is a much better way to define the complexity of a code. The *complexity of a convolutional code*, $\delta(\mathcal{C})$ is the maximum degree of all the full size minors of *any* encoder $G(s)$. This complexity is often referred to as the *McMillan degree* in the systems theory literature. When the context is clear, the notation will often be shortened to simply $\delta$.

An encoder is *minimal* if $\delta(G(s)) = \delta(\mathcal{C})$. It can be shown that any two minimal encoders must have the exact same row degrees (up to reordering). These row degrees for a minimal encoder are known as the *Kronecker indices* of the code.

## 2.2 Realization

In this section we will develop first order representations of convolutional codes as defined in the previous section. Again this section is a review of previous works including [58, 61, 71].

The following theorem proves the existence of a first order realization. It should be remarked that the results here are 'dual' statements of those in [37] as observed in [58].

**Theorem 2.2.1** *[58] Let $\mathcal{C} \subseteq \mathbb{F}^n[s]$ be a rate $k/n$ convolutional code of complexity $\delta$. Then there exist size $(\delta + n - k) \times \delta$ matrices $K$, $L$ and a size $(\delta + n - k) \times n$ matrix $M$ (all with scalar entries in $\mathbb{F}$) such that the code $\mathcal{C}$ is defined by*

$$\mathcal{C} = \{v(s) \in \mathbb{F}^n[s] \mid \exists\, x(s) \in \mathbb{F}^\delta[s] : sKx(s) + Lx(s) + Mv(s) = 0\}.$$

*Further, $K$ has full column rank, $[K \mid M]$ has full row rank and* $\operatorname{rank}[s_0K + L \mid M] = \delta + n - k,\ \forall\, s_0 \in \bar{\mathbb{F}}$.

*A triple $(K, L, M)$ satisfying the above is called a minimal representation of $\mathcal{C}$.*

**Proposition 2.2.2** *[58] If $(\bar{K}, \bar{L}, \bar{M})$ is another representation of the convolutional code $\mathcal{C}$ then there exist unique invertible (scalar) matrices $T$ and $S$ such that*

$$(\bar{K}, \bar{L}, \bar{M}) = (TKS^{-1}, TLS^{-1}, TM).$$

It can be shown, after a suitable transformation permitted by the above proposition, and possibly a reordering of the components of the code (obviously resulting in an 'equivalent' code) that the triple $(K, L, M)$ can be written in the following special form.

$$K = \begin{bmatrix} -I \\ 0 \end{bmatrix}, \quad L = \begin{bmatrix} A \\ C \end{bmatrix}, \quad M = \begin{bmatrix} 0 & B \\ -I & D \end{bmatrix}$$

Here, $A$ is size $\delta \times \delta$, $B$ is $\delta \times k$, $C$ is $(n-k) \times \delta$ and $D$ is $(n-k) \times k$.

This rewriting of the first order representation allows us to define a convolutional code in terms of the more familiar $(A, B, C, D)$ representation in systems theory. Hence, we arrive at the *input-state-output* representation.

**Definition 2.2.3** [Input-State-Output Definition of Convolutional Code] [61]
Let $\mathbb{F} = \mathbb{F}_q$ be the Galois field of $q$ elements and consider the matrices $A \in \mathbb{F}^{\delta\times\delta},\ B \in \mathbb{F}^{\delta\times k},\ C \in$

$\mathbb{F}^{(n-k)\times\delta}$, and $D \in \mathbb{F}^{(n-k)\times k}$. A rate $\frac{k}{n}$ convolutional code $\mathcal{C}$ of complexity $\delta$ can be described by the linear system governed by the equations:

$$\begin{aligned} x_{t+1} &= Ax_t + Bu_t, \\ y_t &= Cx_t + Du_t, \\ v_t &= \begin{pmatrix} y_t \\ u_t \end{pmatrix}, \; x_0 = 0. \end{aligned} \tag{2.2.1}$$

Hence the input to the encoder at time $t$ is the *information or message vector*, $u_t$. The encoder creates the *parity vector*, $y_t$, and the *code vector*, $v_t$, is transmitted across the channel. We will refer to the convolutional code created in this way by $\mathcal{C}(A, B, C, D)$.

From a systems theory point of view, the variable $x_t$ is referred to as the *state* of the system at time $t$. The input vector, $u_t$, is combined with the current state, $x_t$, to create the output, $y_t$. Also, the current input is used to update the state for the next time interval, $x_{t+1}$.

Some enlightening examples, as well as the connections between the various representations of convolutional codes will be given in Chapter 3.

The set of code words are, by definition, equal to the set of trajectories $\left\{\binom{y_t}{u_t}\right\}_{t\geq 0}$ of the dynamical system (2.2.1). The following Proposition characterizes those trajectories.

**Proposition 2.2.4 (Local Description of Trajectories)** *[61] Let $\tau, \gamma$ be positive integers with $\tau < \gamma$. Assume that the encoder is at state $x_\tau$ at time $t = \tau$. Then any code sequence $\left\{\binom{y_t}{u_t}\right\}_{t\geq 0}$ governed by the dynamical system* (2.2.1) *must satisfy:*

$$\begin{pmatrix} y_\tau \\ y_{\tau+1} \\ \vdots \\ \vdots \\ y_\gamma \end{pmatrix} = \begin{pmatrix} C \\ CA \\ \vdots \\ \vdots \\ CA^{\gamma-\tau} \end{pmatrix} x_\tau + \begin{pmatrix} D & 0 & \cdots & 0 \\ CB & D & \ddots & \vdots \\ CAB & CB & \ddots & \ddots \\ \vdots & & \ddots & \ddots \; 0 \\ CA^{\gamma-\tau-1}B & CA^{\gamma-\tau-2}B & \cdots & CB \;\; D \end{pmatrix} \begin{pmatrix} u_\tau \\ u_{\tau+1} \\ \vdots \\ \vdots \\ u_\gamma \end{pmatrix}$$

*Moreover the evolution of the state vector $x_t$ is given over time as:*

$$x_t = A^{t-\tau}x_\tau + \begin{pmatrix} A^{t-\tau-1}B & \ldots & B \end{pmatrix} \begin{pmatrix} u_\tau \\ \vdots \\ u_{t-1} \end{pmatrix}; \quad t = \tau+1, \tau+2, \ldots, \gamma+1. \tag{2.2.2}$$

*Proof:* This follows easily by iterating the equations that define the system. □

For $\gamma \geq 1$ we will define the following notation:

$$M_\gamma(A, B, C, D) = \begin{bmatrix} D & 0 & \cdots & 0 \\ CB & D & \ddots & \vdots \\ CAB & CB & \ddots & \ddots \\ \vdots & & \ddots & \ddots \; 0 \\ CA^{\gamma-2}B & CA^{\gamma-3}B & \cdots & CB \;\; D \end{bmatrix} \tag{2.2.3}$$

Some or all of the parameters may be omitted when the context is clear. This should not cause confusion with the $M$ from the $(K, L, M)$ first order representation because that representation will not be discussed any further in this dissertation.

Next the set of valid codewords is given as the kernel of a parity check matrix.

**Proposition 2.2.5 (Global Description of Trajectories)** *[61]*
$\left\{ \binom{y_t}{u_t} \in \mathbb{F}^n \mid t = 0, \ldots, \gamma \right\}$ *represents a valid code word if and only if:*

$$\left( \begin{array}{ccc|ccccc} 0 & \cdots & 0 & A^{\gamma}B & A^{\gamma-1}B & \cdots & AB & B \\ \hline & & & D & & & & \\ & & & CB & D & & & \\ & -I & & CAB & CB & \ddots & & \\ & & & \vdots & & \ddots & \ddots & \\ & & & CA^{\gamma-1}B & CA^{\gamma-2}B & \cdots & CB & D \end{array} \right) \begin{pmatrix} y_0 \\ y_1 \\ \vdots \\ y_{\gamma} \\ \hline u_0 \\ u_1 \\ \vdots \\ u_{\gamma} \end{pmatrix} = 0. \tag{2.2.4}$$

*Proof*: Setting $\tau = 0$ in Proposition 2.2.4 gives the bottom portion of the matrix. Since $x_{\gamma+1} = 0$ the top row of the matrix follows from the second part of Proposition 2.2.4. □

Let $A, B, C$ be scalar matrices over $\mathbb{F}$ of size $\delta \times \delta$, $\delta \times k$ and $(n-k) \times \delta$ respectively. Let $j$ be a positive integer and define

$$\Phi_j(A, B) := \left( \begin{array}{cccc} A^{j-1}B & A^{j-2}B \ldots & AB & B \end{array} \right), \tag{2.2.5}$$

$$\Omega_j(A, C) := \begin{pmatrix} C \\ CA \\ CA^2 \\ \vdots \\ CA^{j-1} \end{pmatrix}. \tag{2.2.6}$$

**Definition 2.2.6** Let $A, B$ be matrices of size $\delta \times \delta$ and $\delta \times k$ respectively. Then $(A, B)$ is called a *controllable pair* if

$$\operatorname{rank} \Phi_{\delta}(A, B) = \delta. \tag{2.2.7}$$

If $(A, B)$ is a controllable pair then we call the smallest integer $\kappa$ having the property that $\operatorname{rank} \Phi_{\kappa}(A, B) = \delta$ the *controllability index* of $(A, B)$.

In a similar fashion we define:

**Definition 2.2.7** Let $A, C$ be matrices of size $\delta \times \delta$ and $(n-k) \times \delta$ respectively. Then $(A, C)$ is called an *observable pair* if

$$\operatorname{rank} \Omega_{\delta}(A, C) = \delta. \tag{2.2.8}$$

If $(A, C)$ is an observable pair then we call the smallest integer $\eta$ having the property that $\operatorname{rank} \Omega_{\eta}(A, C) = \delta$ the *observability index* of $(A, C)$.

It is true that the $(A, B, C, D)$ representation of the convolutional code is minimal (in terms of complexity $\delta$) if and only if $(A, B)$ is a controllable pair [61]. Further, the convolutional code will be observable (*i.e.* non-catastrophic with delay 0) if and only if $(A, C)$ form an observable pair [61]. Hence, it is natural to only consider codes whose representations satisfy these conditions.

Finally, let us translate the result of Proposition 2.2.2 to the input-state-output representation.

**Proposition 2.2.8** *For $T \in Gl_{\delta}(\mathbb{F})$ we have:*

$$\mathcal{C}(A, B, C, D) \quad = \quad \mathcal{C}(TAT^{-1}, TB, CT^{-1}, D)$$

## 2.3 Special Properties of the ISO Representation

In this section, the properties of the ISO representation (2.2.1) will be discussed. In particular, for the special case when the rank of $D$ is $k$, which forces the rate of the code to be at most $1/2$, it is possible to 'invert' the system. That is, the system can be rewritten so that the output drives the state and the input. The connections between these two representations, since they will play a fundamental role in what is to follow, will be discussed here. The connections between these representations and the other "classical" representations of convolutional codes will follow later in Chapter 3.

Let us consider a convolutional code defined by the matrices $(A, B, C, D)$ as is (2.2.1). For this section we will assume that $\operatorname{rank} D = k = n - k$. However, we will note that what follows can also be done with some minor modifications for the case where $\operatorname{rank} D = k < n - k$. This situation will not be of use to us in this dissertation and it will be much clearer if those details are omitted here.

By simple algebraic manipulations of (2.2.1), we arrive at the following representation.

**Proposition 2.3.1 (Output-State-Input Representation)**

*Given the $(A, B, C, D)$ representation with conditions discussed above (most importantly that $D$ is invertible), the following linear system defines the same convolutional code.*

$$\begin{aligned} x_{t+1} &= (A - BD^{-1}C)x_t + BD^{-1}y_t, \\ u_t &= -D^{-1}Cx_t + D^{-1}y_t, \\ v_t &= \begin{pmatrix} y_t \\ u_t \end{pmatrix}, \; x_0 = 0. \end{aligned} \tag{2.3.9}$$

*Proof:* Clear, since the defining equations are simple algebraic manipulations of the previous ones. □

In most practical cases, $D$ can be chosen to be the identity which will significantly improve the above notation. Also, we will employ the following shorthand notation for the above matrices.

$$\tilde{A} := (A - BD^{-1}C), \; \tilde{B} := BD^{-1}, \; \tilde{C} := -D^{-1}C, \; \tilde{D} := D^{-1}$$

The following connection between $A$ and $\tilde{A}$ will be useful.

**Lemma 2.3.2** *For $i \geq 1$ we have*

$$A^i = \tilde{A}^i + \begin{bmatrix} \tilde{A}^{i-1}\tilde{B} \; \tilde{A}^{i-2}\tilde{B} \cdots \tilde{A}\tilde{B} \; \tilde{B} \end{bmatrix} \begin{bmatrix} C \\ CA \\ \vdots \\ CA^{i-1} \end{bmatrix}.$$

*Proof:* By induction on $i$. For $i = 1$, the result follows immediately from the definition of $\tilde{A}$. Assume the result holds for $i \leq j$ and show for $i = j + 1$. Hence, we have

$$\tilde{A}^j = A^j - \Phi_j(\tilde{A}, \tilde{B})\, \Omega_j(A, C).$$

Multiplying on the left by $(A - BD^{-1}C)$ gives

$$\tilde{A}^{j+1} = A^{j+1} - BD^{-1}CA^j - \begin{bmatrix} \tilde{A}^j\tilde{B} \; \tilde{A}^{j-1}\tilde{B} \cdots \tilde{A}^2\tilde{B} \; \tilde{A}\tilde{B} \end{bmatrix} \Omega_j(A, C).$$

The new 'extraneous' term is exactly the term 'missing' from the matrix multiplication. Realizing this gives the desired result. □

The output-state-input representation serves just as well as the traditional input-state-output description. In particular, a local description of the trajectories can be obtained in the same way as Proposition 2.2.4 by simply replacing $(A, B, C, D)$ with $(\tilde{A}, \tilde{B}, \tilde{C}, \tilde{D})$. The same substitution into (2.2.2) will give another equation for the state based on the new representation. Similarly, the same is true for the global description of the trajectories provided in Proposition 2.2.5. Of course, we may also define the controllability matrices, $\Phi_\gamma(\tilde{A}, \tilde{B})$, and observability matrices, $\Omega_\gamma(\tilde{A}, \tilde{C})$, with respect to this representation. It is natural to ask how these various objects from the two representations are related. It happens that the relationship is quite nicely described, and we will do so in the following sequence of lemmas.

**Lemma 2.3.3** *Recall the definition of $M_\gamma$ in (2.2.3). Then*

$$M_\gamma(A, B, C, D) = M_\gamma(\tilde{A}, \tilde{B}, \tilde{C}, \tilde{D})^{-1}.$$

*Proof:* We will show that $M_\gamma(\tilde{A},\tilde{B},\tilde{C},\tilde{D})\, M_\gamma(A,B,C,D) = I_{k\gamma}$. The fact that the entries above the diagonal are 0 is trivial since we are multiplying two lower diagonal matrices. The diagonal consists of all 1's since $\tilde{D}D = I_k$. The next lower diagonal is all 0's since $\tilde{C}\tilde{B}D + \tilde{D}CB = -D^{-1}CBD^{-1}D + D^{-1}CB = 0$. All lower diagonals are zero since each block entry can be written in the form

$$\tilde{C}\,\left[\tilde{A}^j + \Phi_j(\tilde{A},\tilde{B})\,\Omega_j(A,C) - A^j\right]\,B$$

An immediate application of Lemma 2.3.2 shows this to be 0. The fact that the reverse multiplication also gives the identity is clear, but can also be proven directly by obtaining an analog of Lemma 2.3.2 with the 'tildes' and 'non-tildes' switched. □

**Lemma 2.3.4**

$$\begin{aligned}\Phi_\gamma(A,B)\,M_\gamma(\tilde{A},\tilde{B},\tilde{C},\tilde{D}) &= \Phi_\gamma(\tilde{A},\tilde{B})\\ M_\gamma(\tilde{A},\tilde{B},\tilde{C},\tilde{D})\,\Omega_\gamma(A,C) &= \Omega_\gamma(\tilde{A},\tilde{C})\end{aligned}$$

*Proof:* Both statements are a direct consequence of Lemma 2.3.2. □

The above lemmas show that the input-state-output representation is very naturally related to the output-state-input representation. In Chapter 3, we will see how these representations are related to the more classical representations of convolutional codes.

## 2.4 An Algebraic Model for Turbo Codes

Turbo codes were introduced by Berrou, Glavieux and Thitimajshima in 1993 [14]. Their idea of using parallel concatenation of recursive systematic convolutional codes (RSCs) (See Definition 3.1.2.) with an interleaver was a major step in terms of achieving low bit error rates (BERs) at signal-to-noise ratios near the Shannon limit. (See Chapter 5 for a more thorough explanation of these terms.) However, their ideas were originally met with skepticism, partly because of the phenomenal performance of the codes and partly because it was not clear why they worked. Many papers have since been devoted to these questions, e.g. [54, 20, 21, 19]. Much of this work has focused on the improved weight distribution obtained by using a suitable (usually random) interleaver. Others focused on developing a more suitable soft-decision decoder to be used in the turbo-decoding process, e.g. [13, 31]. This section will focus on developing turbo codes in the framework of the input-state-output representation for convolutional codes.

We will give a brief review of turbo codes. We will then present the natural generator and parity check descriptions of turbo codes that follow from the local description of Proposition 2.2.4. The decoding of turbo codes using this new description will be discussed in Chapter 6.

### 2.4.1 A Brief Review of Turbo Codes

Here the basic structure of turbo codes will be reviewed. The following is the design of the most simple of turbo codes. Only two parallel identical RSCs are used and no puncturing of the output bits is employed. Turbo codes could have many parallel RSCs, but typically only two are used, and even more typically, the RSCs will be rate 1/2 and be identical. Puncturing is used very often to improve the rate of the code by 'splicing' together the separate output streams. We will omit this part of the turbo coding scheme simply for clarity of presentation.

The input, $u$, to the turbo encoder is used in 3 ways. First, it is sent directly to the channel as part of the codeword. Thus, turbo codes are systematic. Second, the input is sent into the first RSC and the output, say $y$, of this convolutional encoder (separated from the input) is sent along the channel as the second part of the codeword. Finally, the input is also sent into an interleaver or permutation $S$, which rearranges the order of the input bits. The scrambled input bits are then sent into the RSC and its output, say $\hat{y}$, is sent as the final piece of the codeword. This general scheme is outlined in figure 2.1.

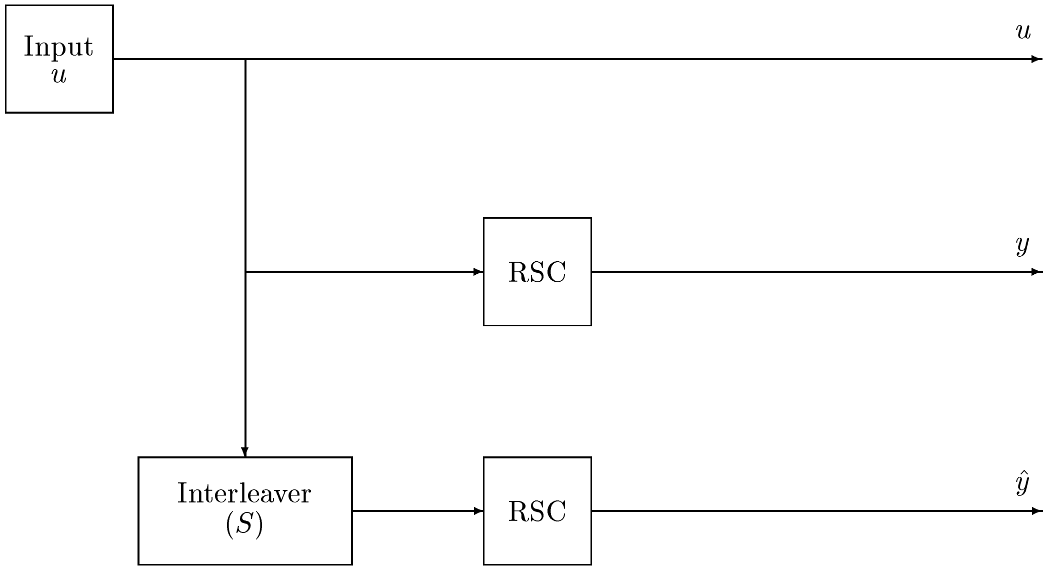


FIGURE 2.1: The basic encoding structure of turbo codes.

Here are a few points worth mentioning. First, the overall rate of the code as just described is 1/3, although typically, as mentioned above, puncturing is used to increase the rate to 1/2. Secondly, the size of the interleaver, $S$, plays a crucial role. The interleaver is just a known permutation on $N$ elements, or inputs bits in this case. So we need to wait for each block of $N$ input bits before we encode. Unfortunately, the best results of turbo coding come when $N$ is extremely large. A typical choice for $N$ is about 65,000. Needless to say this is not well suited to certain applications where delays of information cannot be tolerated. Regardless, some work has been done for smaller $N$, e.g. [33, 32].

### 2.4.2 A New Description of Turbo Codes

Now we will use the representation of a convolutional code presented in Section 2.2 to obtain representations of turbo codes. In particular, we will obtain a generator matrix and parity check matrix description of turbo codes.

First, let us develop our turbo code (as described in Figure 2.1). To do this we specify the size of our interleaver and choose an interleaver. We will arbitrarily denote the size as $N$ and the interleaver by $S$. Next we must specify which RSC we will use. Of course, this can be done by specifying scalar matrices $(A, B, C, D)$ of the appropriate size. A concrete example will be presented at the end of this section.

Although not necessarily required, we will make the typical assumption that after each $N$ time intervals BOTH RSCs will be back in the all zero state. (The paper by Blackert *et al.* [16] gives conditions on the interleaver which guarantee this possible. For this dissertation, we will assume all interleavers are chosen with this property. Other inquiries into interleaver design are given in [10, 11].) This will make more clear the prevailing notion that turbo codes can be thought of as large block codes.

We can represent each $N$-block of output from the RSC as follows using Proposition 2.2.4. For $\tau \geq 0$ we have:

$$\begin{pmatrix} y_{\tau N} \\ y_{\tau N+1} \\ \vdots \\ \vdots \\ y_{\tau N+N-1} \end{pmatrix} = \begin{pmatrix} D & 0 & \cdots & & 0 \\ CB & D & \ddots & & \vdots \\ CAB & CB & \ddots & \ddots & \\ \vdots & & \ddots & \ddots & 0 \\ CA^{N-2}B & CA^{N-3}B & \cdots & CB & D \end{pmatrix} \begin{pmatrix} u_{\tau N} \\ u_{\tau N+1} \\ \vdots \\ \vdots \\ u_{\tau N+N-1} \end{pmatrix}. \tag{2.4.10}$$

Reverting to our earlier notation, the large matrix on the right hand side of the above equation will be denoted $M_N(A, B, C, D)$, or simply $M$. We are now ready to give a parity check matrix and a pseudo-generator matrix description of the turbo code. The proofs of these assertions are straightforward.

**Theorem 2.4.1** *A vector $V \in \mathbb{F}^{3N}$ is a valid $N$-block of the turbo code if and only if*

$$\begin{bmatrix} \Phi_N(A,B) & 0 & 0 \\ M & -I & 0 \\ MS & 0 & -I \end{bmatrix} V = 0,$$

*where $S$ is the interleaver (permutation) matrix. Thus any sequence in $\mathbb{F}$ is a codeword of the turbo code if and only if it is a sequence of valid $N$-blocks.*

**Lemma 2.4.2** *If $u \in \mathbb{F}^N$ and $u \in ker\ \Phi_N(A,B)$, then the image $Gu$, where*

$$G = \begin{bmatrix} I \\ M \\ MS \end{bmatrix}$$

*is a valid $N$-block of the turbo code.*

This description is not satisfying since the input, $u$, is constrained. This can be overcome with the following observation. With reasonable assumptions on $A$ and $B$, including that $(A, B)$ be a controllable pair and that the characteristic polynomial of $A$ equals the minimum polynomial, Cayley's theorem implies that $\Phi_N(A,B)u = 0$ if and only if $p_A(s)|u(s)$, where $p_A(s)$ is the minimum polynomial of the matrix $A$, and $u(s)$ is the polynomial whose coefficients are the entries of the vector $u$. We know the degree of $p_A(s)$ is $\delta$. We write $p_A(s) = p_\delta s^\delta + \ldots + p_1 s + p_0$. We define the following matrix, $P$, of size $N \times N - \delta$, (Hence, we assume $N > \delta$.) with the property that the image of $P$ is precisely the kernel of $\Phi_N(A,B)$:

$$P = \begin{bmatrix} p_0 & & & 0 \\ p_1 & p_0 & & \\ \vdots & p_1 & \ddots & \\ p_\delta & \vdots & \ddots & p_0 \\ & p_\delta & & p_1 \\ & & \ddots & \vdots \\ 0 & & & p_\delta \end{bmatrix}$$

We are now ready to present the generator matrix description of a turbo code.

**Theorem 2.4.3** *The valid $N$-blocks for the turbo code are generated by the following matrix $G$. i.e. For any $v \in \mathbb{F}^{N-\delta}$, $Gv$ is a valid $N$-block. Also every valid $N$-block is of this form.*

$$G = \begin{bmatrix} P \\ MP \\ MSP \end{bmatrix}$$

*Proof:* Follows from definition of $P$ and Lemma 2.4.2. □

**Remark 2.4.4** In the above generator matrix, the input $v$ is not the input to the encoder, rather $Gv$ is the input to the encoder.

**Example 2.4.5** A typical RSC chosen for binary turbo codes is given by $[1 \;\; \frac{s^2+1}{s^2+s+1}]$. We will develop the representations of this section for this encoder. First, the matrices $(A, B, C, D)$ are given by:

$$A = \begin{bmatrix} 0 & 1 \\ 1 & 1 \end{bmatrix}, \quad B = \begin{bmatrix} 0 \\ 1 \end{bmatrix}, \quad C = [0 \;\; 1], \quad D = [1].$$

Clearly, and not surprisingly, $P_A(s) = s^2 + s + 1$. For the sake of exposition, we will choose a ridiculously small value $N$=5. The matrix P and the matrix M are :

$$P = \begin{bmatrix} 1 & 0 & 0 \\ 1 & 1 & 0 \\ 1 & 1 & 1 \\ 0 & 1 & 1 \\ 0 & 0 & 1 \end{bmatrix} \qquad M = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & 0 \\ 1 & 1 & 1 & 0 & 0 \\ 0 & 1 & 1 & 1 & 0 \\ 1 & 0 & 1 & 1 & 1 \end{bmatrix}$$

# CHAPTER 3

# CONNECTIONS BETWEEN REPRESENTATIONS

In this chapter we will present some examples of convolutional codes using each of the various representations. In particular, example codes will be developed with the generator matrix description. From there, it will be shown how to transform this description into a parity check description and input-state-output description. Also, the generator sequence representation and electronic shift register descriptions will be introduced and derived from the generator description. More importantly, codes developed using the input-state-output representation will be transformed into generator and parity check descriptions.

## 3.1 Generator Matrices and Sequences

We have already seen in Definition 1.3.1 that a convolutional code can be described in terms of a polynomial generator matrix, $G(s)$. Before proceeding any further let us give some concrete examples.

**Example 3.1.1** Let us define two binary convolutional encoders, $G^1(s)$ and $G^2(s)$ by,

$$\begin{aligned} G^1(s) &:= \begin{bmatrix} s+1 & s^3+s+1 \end{bmatrix} \\ G^2(s) &:= \begin{bmatrix} 1 & 0 & s^2+1 \\ 0 & 1 & s \end{bmatrix} \end{aligned}$$

For $G^1(s)$ it is clear that $m = \nu_1 = 3$ and $\delta(G^1(s)) = 3$. For $G^2(s)$ it is easy to see that $m = \nu_1 = 2$, $\nu_2 = 1$ and $\delta(G^2(s)) = 3$. Let us denote the codes they describe by $\mathcal{C}_1$ and $\mathcal{C}_2$ respectively. It is easy to check that $\delta(\mathcal{C}_1) = 3$ and that $\delta(\mathcal{C}_2) = 2$. Hence, $G^2(s)$ is not a minimal encoder. The 'problem' with this encoder is that $G^2(s)$ is not a *minimal basis of the rational vector space* generated by its rows as in the sense of [24]. This issued will be touched upon in Chapter 4.

**Definition 3.1.2** A generator matrix is *systematic* if it is in the form $[\,\tilde{G}(s) \quad I\,]$, where $\tilde{G}(s)$ has entries in $\mathbb{F}(s)$. The bits of the codeword corresponding to the identity matrix are called the *information* bits of the codeword since they are exactly the bits of the message, or information, vector. A generator matrix is said to be *recursive* if it contains non-polynomial entries.

We can see that $G^2(s)$ is systematic but $G^1(s)$ is not.
For any generator matrix, we may write in the obvious way:

$$G(s) \;=\; G_m s^m \;+\; G_{m-1}s^{m-1} \;+\; \cdots \;+\; G_1 s \;+ G_0$$

where each $G_i$ is simply the $k \times n$ matrix whose entries are the scalar coefficients of $s^i$ in the corresponding polynomial entries of $G(s)$. In doing so, the sequence

$$G_0 \; G_1 \; \ldots G_{m-1} \; G_m$$

is called a *generating sequence* of the code. It is clear that there is a bijection between generator matrices and generating sequences and that transformation between the two is obvious.

Using generating sequences it is easy to write down a scalar generating matrix (as for block codes) for a convolutional code. However, since convolutional codes can have message words and codewords of arbitrary length, the generator matrix must accommodate. We arrive at the following *semi-infinite generator matrix* description.

**Definition 3.1.3**

$$G \;=\; \begin{bmatrix} G_0 & G_1 & G_2 & \cdots & G_m & & & \cdots \\ 0 & G_0 & G_1 & \cdots & G_{m-1} & G_m & & \\ 0 & 0 & G_0 & \cdots & G_{m-2} & G_{m-1} & G_m & \\ \vdots & & & & & & & \ddots \end{bmatrix}$$

### 3.2 Parity Check Matrices

Since we can also define a linear subspace as the kernel of a matrix, it is clear that a convolutional code can be described as the kernel of a polynomial matrix. The matrix is called a *parity check matrix* or *syndrome former*. If one has a generator matrix for a convolutional code, how does one obtain a parity check matrix? The answer is a rather simple application of linear algebra. Take the generator matrix, $G(s)$ and append the $n \times n$ identity matrix below it. Then perform elementary (over $\mathbb{F}[s]$) column operations on this matrix until the top $k$ rows come into the form $[\,L(s) \mid 0\,]$. Then the bottom $n$ rows and last $n-k$ columns form the transpose of a parity check matrix. One may refer to [40, 39, 23] for a more in depth discussion.

**Remark 3.2.1** The matrix $L(s)$ is a greatest common left divisor of the matrix $G(s)$. This concept will be explained thoroughly in Chapter 4. For now it suffices to know that if $L(s)$ is $I_k$, then the above algorithm also gives a polynomial inverse, $G^{-1}(s)$, for the encoder $G(s)$ in the first $k$ columns of the bottom $n$ rows. If $L(s)$ is not the identity, then $G(s)$ is *catastrophic* as we have defined it before, and more importantly for our purposes here, the parity check matrix will, depending on how we choose to define a convolutional code, actually define a larger code than the one defined by $G(s)$. This point will be illustrated in the following examples.

**Example 3.2.2** [Continuation of Example 3.1.1] For $G^1(s)$, we have

$$\begin{bmatrix} s+1 & s^3+s+1 \\ 1 & 0 \\ 0 & 1 \end{bmatrix} \text{after column operations is} \begin{bmatrix} 1 & 0 \\ s^2+s & s^3+s+1 \\ 1 & s+1 \end{bmatrix}$$

Hence, $H^1(s) = \begin{bmatrix} s^3+s+1 & s+1 \end{bmatrix}$. Similarly, it can be shown that $H^2(s) = \begin{bmatrix} s^2+1 & s & 1 \end{bmatrix}$ is the parity check matrix for $G^2(s)$.

**Example 3.2.3** Consider the binary catastrophic convolutional encoder

$$G^3(s) := \begin{bmatrix} s & 0 & s \\ 1 & s+1 & s^2+s+1 \end{bmatrix}$$

After performing the above algorithm we arrive at

$$\begin{bmatrix} s & 0 & 0 \\ 1 & s+1 & 0 \\ 1 & 0 & 1 \\ 0 & 1 & s \\ 0 & 0 & 1 \end{bmatrix}$$

This shows that the 'greatest common left divisor', $L(s)$, of $G^3(s)$ is

$$L(s) = \begin{bmatrix} s & 0 \\ 1 & s+1 \end{bmatrix}.$$

In Chapter 4, an alternative way based on state space realizations is developed for computing such divisors. However, the above calculation also shows that $H^3(s) = [1 \mid s \mid 1]$. The polynomial vector $[1 \mid 1 \mid s+1]$ is in the kernel of $H^3(s)$. However, no polynomial vector will multiply by $G^3(s)$ to generate this vector. The rational vector $[\frac{1}{s+1} \;\; \frac{1}{s+1}]$ does generate this codeword. Therefore, $H^3(s)$ defines a larger convolutional code if we employ Definition 1.3.1 as opposed to the one given in Corollary 2.1.5. This seemingly critical difference in the two definitions is made irrelevant since we have already remarked that catastrophic encoders are to be avoided. Any catastrophic encoder, $G(s)$, can be replaced by a non-catastrophic encoder $L^{-1}(s)G(s)$.

Finally, let us remark that, in parallel with generating sequences, parity check sequences are also employed. Similarly, there exists *semi-infinite parity check* matrix descriptions of convolutional codes. We will refer to the semi-infinite parity check matrices as syndrome formers and their polynomial counterparts as parity check matrices, although typically the two terms are interchangeable.

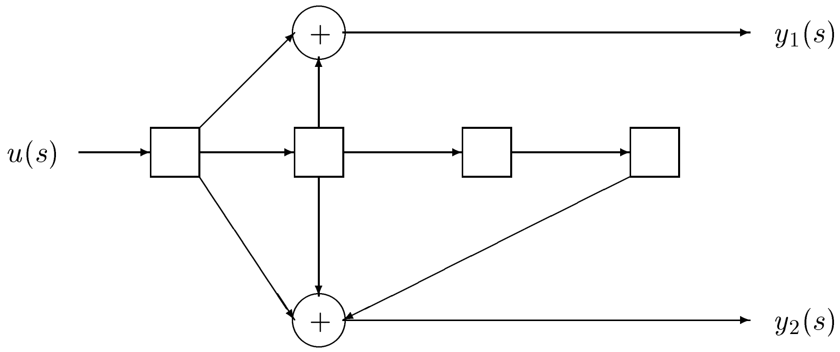


FIGURE 3.1: Shift register for $\mathcal{C}_1$.

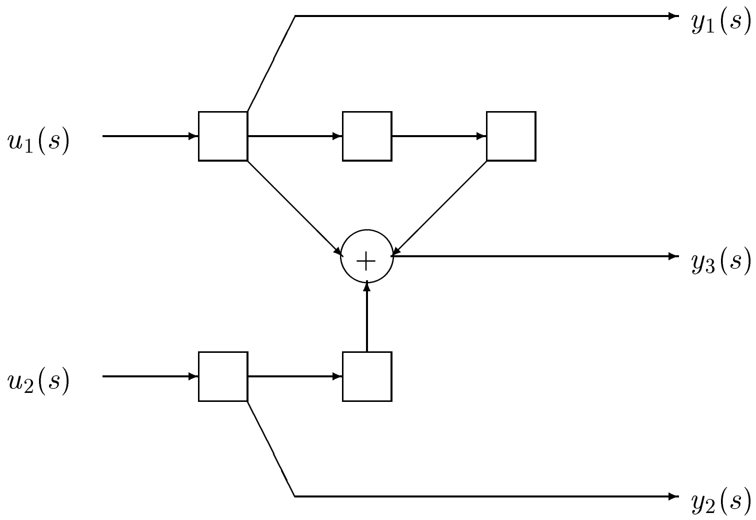


FIGURE 3.2: Shift register for $\mathcal{C}_2$.

### 3.3 Shift Registers

Convolutional codes are widely used in digital communication systems today. One of the primary reasons for this is that there exists an efficient method of implementing the encoding process. Shift registers are an extremely simple and fast way to implement binary convolutional encoding as we shall see.

A shift register, informally, is a circuit consisting of memory elements, $\square$, and modulo-2 adders, $\bigoplus$. (For non-binary codes, constant multipliers can be used as well.) For a rate $k/n$ convolutional encoder with row degrees $\nu_1, \nu_2, \ldots, \nu_k$, the shift register consists of $k$ rows of memory elements, with $\nu_i + 1$ elements in row $i$. These memory elements are initially filled with 0's. At the first time interval the $k$ bits, $u_i$ of the message word are each entered into the first memory element of their respective row. The contents of all the other elements are shifted to the next memory element in the row, with the contents of the last element being discarded. This is the memory side of the encoding process. We still need to address how the codeword is output.

For each time interval, each of the $n$ outputs, $y_i$, of the convolutional code is formed by adding modulo-2 (or whatever field the code is defined over) the contents of the 'appropriate' memory elements. Which elements are 'appropriate' is determined, of course, by the generator matrix (or other suitable description of the convolutional code). Suffice it to say that a few examples should make the process clear. The shift registers for $\mathcal{C}_1$ and $\mathcal{C}_2$ are presented in Figures 3.1 and 3.2.

Of course, it is also quite simple to go from a shift register representation to a generator matrix description. The process of interchanging shift registers and $(A, B, C, D)$ representations is also quite natural. This process is detailed nicely in Section 7.3 of [42].

### 3.4 From ISO Representations to Generator Matrices

Given matrices $(A, B, C, D)$ which define a convolutional code by (2.2.1), it is natural to wonder what the generator matrix of the code is. In principle it is an easy task to compute the generator matrix.

If we take any codeword $\binom{y(s)}{u(s)}$ and its corresponding sequence of states $x(s)$ (represented in polynomial form in the reverse way: $(a_0, a_1, \ldots, a_n) \leftrightarrow a_0 s^n + a_1^{n-1} + \cdots + a_n$), then the following equation holds:

$$\begin{bmatrix} sI - A & 0 & -B \\ -C & I & -D \end{bmatrix} \begin{bmatrix} x(s) \\ y(s) \\ u(s) \end{bmatrix} = 0.$$

**Proposition 3.4.1** *[58] There exists polynomial matrices $X(s)$, $Y(s)$ and $U(s)$ with size $\delta \times k$, $(n-k) \times k$and $k \times k$ respectively, such that*

$$\ker \begin{bmatrix} sI - A & 0 & -B \\ -C & I & -D \end{bmatrix} = \operatorname{im} \begin{bmatrix} X(s) \\ Y(s) \\ U(s) \end{bmatrix}.$$

*Also, the matrix $G(s) = [\,Y(s)' \quad U(s)'\,]$ forms a polynomial encoder.*

In short the above Proposition states that a generator matrix can be obtained by computing a basis for the kernel of the matrix on the left hand side of the equation. This can be done by hand or by computer for larger matrices.

**Example 3.4.2** Consider the binary code defined by the matrices:

$$A := \begin{bmatrix} 1 & 1 \\ 1 & 0 \end{bmatrix}, \quad B := \begin{bmatrix} 1 \\ 0 \end{bmatrix}, \quad C := [1 \;\; 0], \quad D := [1]$$

A simple calculation reveals that $G(s) = [\,s^2 + 1 \;\; s^2 + s + 1\,]$.

The next issue is to go from a generator matrix description to a first order representation. This process is called *realization* and was first discussed in Section 2.2. The actual process is simplified if we have a parity check matrix rather than a generator matrix. Then obtaining a $(K, L, M)$ description is rather straightforward and is described in detail in [57]. We are more interested, however, in obtaining an $(A, B, C, D)$ representation. We will show how this is done, and give an immediate application, in Chapter 4.

### 3.5 From the Local Description to Syndrome Formers

In this section we will show how the local description in Proposition 2.2.4 contains or can easily be made to contain a parity check matrix, or syndrome former, for the convolutional code. Let us start by recalling the local description in a slightly different form:

$$\begin{pmatrix} C \\ CA \\ \vdots \\ CA^{\gamma-\tau} \end{pmatrix} x_\tau = \left(\begin{array}{cccc|cccc} D & 0 & \cdots & 0 & I & & & \\ CB & D & \ddots & \vdots & & I & & \\ \vdots & & \ddots & 0 & & & \ddots & \\ CA^{\gamma-\tau-1}B & CA^{\gamma-\tau-2}B & \cdots & D & & & & I \end{array}\right) \begin{pmatrix} -u_\tau \\ -u_{\tau+1} \\ \vdots \\ -u_\gamma \\ \hline y_\tau \\ y_{\tau+1} \\ \vdots \\ y_\gamma \end{pmatrix} \tag{3.5.1}$$

For simplicity of notation, let us restrict ourselves to rate 1/2 codes. Let the minimum polynomial of the matrix $A$ be denoted by $p_A(s)$. To omit degenerate cases we will assume the degree of $p_A(s)$ to be $m$. Here $m$ is the 'memory' of the encoder in the usual sense. This is also equal to $\delta$ in the rate 1/n case.

**Lemma 3.5.1** *By performing elementary row operations on the $(m+1)$st row block of the above matrix equation in accordance with the polynomial $p_A(s)$ (e.g. if $p_A(s) = s^3 + s + 1$, then add the row block beginning with $C$ and the one beginning with $CA$ to the row block beginning with $CA^3$), then the (m+1)st row block of the matrix on the left hand side will be* $\mathbf{0}$. *Repeating this operation mutatis mutandis for lower row blocks will eliminate all lower blocks of the matrix on the left hand side. The resulting matrix at and below the (m+1)st row block on the right hand side will be (up to row operations) the parity check matrix or syndrome former (in semi-infinite block form) for the convolutional code.*

*Proof:* The first two statements follow immediately from the definition of minimum polynomial. The last statement is trivial since the lower portion of the resulting matrix has a shifted block structure and, by the resulting equation, multiplies with the code bits to equal 0. Thus, it must be a syndrome former, differing from the 'standard' one by at most elementary row operations. □

Letting S denote the top $m$ rows of the observability matrix, we arrive at the following situation:

$$\left(\frac{S}{0}\right) x_\tau = \left(\frac{\text{State Equations}}{\text{Syndrome Former}}\right)(\text{Codeword}) \tag{3.5.2}$$

Simply put, the above equation says that the top portion of the matrix consists of equations on the bits of the codeword that depend on the state at any given time $\tau$. The bottom portion of the matrix consists of the syndrome former matrix as shown in the lemma (3.5.1). Let us clarify these points with a few examples.

**Example 3.5.2** Let us consider the following code over $\mathbb{F}_2$:

$$A = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \end{bmatrix} \qquad B = \begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix}$$

$$C = \begin{bmatrix} 0 & 0 & 1 & 1 & 1 \end{bmatrix} \qquad D = \begin{bmatrix} 1 \end{bmatrix}$$

$$\left[\begin{array}{ccccc} 0 & 0 & 1 & 1 & 1 \\ 0 & 1 & 1 & 1 & 0 \\ 1 & 1 & 1 & 0 & 0 \\ 1 & 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 \\ \hline 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{array}\right] x_\tau = \left[\begin{array}{cccccccc|cccccccc} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ \hline 1 & 1 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 1 & 1 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 1 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{array}\right] \left[\begin{array}{c} u_\tau \\ u_{\tau+1} \\ u_{\tau+2} \\ u_{\tau+3} \\ u_{\tau+4} \\ u_{\tau+5} \\ u_{\tau+6} \\ u_{\tau+7} \\ \hline y_\tau \\ y_{\tau+1} \\ y_{\tau+2} \\ y_{\tau+3} \\ y_{\tau+4} \\ y_{\tau+5} \\ y_{\tau+6} \\ y_{\tau+7} \end{array}\right]$$

**Remark 3.5.3** The last 3 rows of the above matrix on the right-hand side are shifted versions of the syndrome former matrix for this encoder. In the above example, since the matrix $A$ has a minimum polynomial of a very nice form, $p_A(s) = s^5$, no row operations were necessary to obtain the syndrome former structure on the bottom half of the matrix equation. In general, whenever $p_A(s) = s^i$ (for some positive integer $i$) the *parity parallelogram* [43][p. 423] used in definite decoding of convolutional codes is nothing more than the Markov parameters $CA^iB$ [66][p. 213]. As we will see later, similar results hold for the parity triangle of feedback decoding. Let us consider an example where this is not the case.

**Example 3.5.4** Let us examine the code given by $G(s) = [\, s^2+1 \quad s^2+s+1 \,]$. We have already seen that a realization of this code is given by:

$$A = \begin{bmatrix} 1 & 1 \\ 1 & 0 \end{bmatrix} \quad B = \begin{bmatrix} 1 \\ 0 \end{bmatrix}$$

$$C = \begin{bmatrix} 1 & 0 \end{bmatrix} \quad D = \begin{bmatrix} 1 \end{bmatrix}$$

$$\left[\begin{array}{cc} 1 & 0 \\ 1 & 1 \\ \hline 0 & 0 \\ 0 & 0 \\ 0 & 0 \end{array}\right] x_{\mathcal{T}} = \left[\begin{array}{ccccc|ccccc} 1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ \hline 1 & 0 & 1 & 0 & 0 & 1 & 1 & 1 & 0 & 0 \\ 0 & 1 & 0 & 1 & 0 & 0 & 1 & 1 & 1 & 0 \\ 0 & 0 & 1 & 0 & 1 & 0 & 0 & 1 & 1 & 1 \end{array}\right] \left[\begin{array}{c} u_{\mathcal{T}} \\ u_{\mathcal{T}+1} \\ u_{\mathcal{T}+2} \\ u_{\mathcal{T}+3} \\ u_{\mathcal{T}+4} \\ \hline y_{\mathcal{T}} \\ y_{\mathcal{T}+1} \\ y_{\mathcal{T}+2} \\ y_{\mathcal{T}+3} \\ y_{\mathcal{T}+4} \end{array}\right]$$

We will further examine the relationship between the syndrome former and the local description in Section 5.5 where we utilize these concepts to explore various majority logic decoding schemes.

# CHAPTER 4

# A MATRIX EUCLIDEAN ALGORITHM

In this chapter, the question of determining if a generator matrix for a convolutional code is catastrophic, non-catastrophic or observable is addressed. Recall that these terms were discussed in Proposition 1.3.2. As will be discussed below, this problem also has much broader implications and many various applications. Hence, let us step back and analyze this problem in terms of computing common divisors for matrices.

We will develop an efficient algorithm for determining the greatest common left divisor (GCLD) of two polynomial matrices. Knowing this divisor allows for several immediate applications: In coding theory, a non-catastrophic convolutional encoder can be derived from an arbitrary one. In systems theory, irreducible matrix fraction descriptions of transfer function matrices can be found. In linear algebra, the greatest common divisor can be seen as a basis for a free module generated by the columns of the matrices.

The approach taken is based on recent ideas from systems theory. A minimal state space realization is obtained with minimal calculations, and from this the controllability matrix is analyzed to produce the GCLD. It will be shown that the derived algorithm is a natural extension of the Euclidean algorithm to the matrix case.

The results of this chapter were reported by the author in [5].

## 4.1 Background Material

Let $\mathbb{F}$ be an arbitrary field and consider the polynomial ring $\mathbb{F}[s]$. If we are given two polynomial matrices $E(s)$ and $F(s)$ each with $k$ rows then we may define a greatest common left divisor (GCLD) to be any $k \times k$ polynomial matrix $L(s)$ satisfying:

1. There exists polynomial matrices $\bar{E}(s)$ and $\bar{F}(s)$ such that $L(s)\bar{E}(s) = E(s)$ and $L(s)\bar{F}(s) = F(s)$.
2. If $\bar{L}(s)$ is any other divisor of $E(s)$ and $F(s)$ then there exists a polynomial matrix $D(s)$ such that $\bar{L}(s)D(s) = L(s)$.

By an arbitrary choice, we will work with left divisors. The theory holds *mutatis mutandis* for right divisors.

Notice that GCLDs are not unique. For our applications we will assume that the matrix $[E(s) \mid F(s)]$ is full rank. This implies that all GCLDs will be nonsingular and differ by a unimodular right factor [35]. Note also that the columns of the GCLD of the full rank polynomial matrix $[E(s) \mid F(s)]$ form a basis for the free module spanned by the columns of $[E(s) \mid F(s)]$ in $\mathbb{F}^k[s]$. Two matrices are said to be coprime if their GCLD is a unimodular matrix.

Instead of starting with two separate matrices and then combining them into one, we may be given, as in the case of a convolutional encoder, a single full rank matrix $P(s)$ of size $k \times n$. We can speak of the GCLD of this single matrix by writing $P(s) = [E(s)|F(s)]$ where usually $E(s)$ is of size $k \times k$ and $F(s)$ is of size $k \times (n-k)$, and hence the GCLD of $P(s)$ is then the GCLD of $E(s)$ and $F(s)$. Obviously the GCLD does not depend on how we choose the division. Equivalently, we could define a GCLD of $P(s)$ to be a matrix $L(s)$ such that $L(s)\bar{P}(s) = P(s)$, where $\bar{P}(s)$ is a polynomial matrix whose Smith, or equivalently, Hermite form is $[I_k \mid 0]$.

With this last description we are able to see several immediate applications. First, if we are given $P(s)$ as a polynomial basis for a rational vector space [24], then by dividing by $L(s)$ (*i.e.* taking $\bar{P}(s)$) we get a minimal polynomial basis for the vector space (as defined in [24]). Secondly, if we are given $P(s)$ as a generating set for its column module over $\mathbb{F}[s]$, then we observed earlier that the columns of the GCLD, $L(s)$, of $P(s)$ form a basis of the column module of $P(s)$. In particular if $P(s)$ is of size $1 \times 2$ and has the form $P(s) = (p(s), q(s))$, $p(s), q(s) \in \mathbb{F}[s]$ then the GCLD of $P(s)$ is nothing else than the greatest common divisor (g.c.d.) of $p(s), q(s)$. Moreover our algorithm is in this case equivalent to Euclid's algorithm. Finally, if we are given $P(s)$ as a convolutional encoder, then $P(s)$ is an observable (*i.e.* non-catastrophic with delay 0) encoder if and only if $L(s)$ is a unimodular matrix [2, 18].

Closely related to this last application, we can think of $P(s)$ as describing over the real numbers $\mathbb{R}$ a linear behavior in the sense of Willems [69]:

$$\mathfrak{B} = \{w(t) \in C^{\infty}(\mathbb{R}, \mathbb{R}^n) \mid P\left(\frac{d}{dt}\right) w(t) = 0\}.$$

The computation of the GCLD is then needed for the computation of the controllable sub-behavior of $\mathfrak{B}$.

The approach that will be taken here is to obtain a minimal state space representation of the associated behavior $\mathfrak{B}$ with little or no calculation [57]. This state space representation will be controllable if and only if our behavioral system (or encoder) is observable. Further, the contribution of this chapter will be to calculate a GCLD of $P(s)$ directly from the controllability matrix of this state space representation. As we shall see, the algorithm presented will be a natural generalization of the Euclidean algorithm to polynomial matrices.

## 4.2 A Brief History of the Problem

The problem of finding GCLDs is not new, and, indeed, there are several algorithms in existence. The most obvious way is to append the two matrices together as $[E(s) \mid F(s)]$ and perform polynomial column operations (over $\mathbb{F}[s]$) to bring the matrix to Smith or Hermite form [8]. The obvious drawback is that polynomial column operations can become quite tedious, especially if the degrees of the polynomial entries are high. This problem was overcome by Kung *et al.* [38, 15] with their approach using generalized Sylvester matrices. A problem with that algorithm is that the scalar matrices obtained from the original polynomial matrices were often quite large.

Several more recent works, using somewhat similar but distinct methods to the one proposed here, have appeared: Fuhrmann [25] obtained an algorithm using a matrix continued fraction representation. Antoulas [6] has done considerable work on the subject using recursive and partial realizations.

An excellent reference on the various techniques of computing GCDs in the case $k = 1$ can be found in [12]. In fact, the section on "G.C.D. Using Companion Matrix" from this book give exactly our algorithm in the simple case $k = 1$. In this reference it was, unfortunately, not observed by the author that the companion matrix was, in fact, a realization of the polynomial matrix. This prevented the extension to the general case, where the author of that paper instead presents the algorithm of Kung *et al.*.

## 4.3 A Realization Algorithm

We now present the realization algorithm we will need, proceeded by some notation. For a more thorough account of the ideas involved, please refer to [57].

Partition $P(s)$ into $P(s) = [\,E(s) \mid F(s)\,]$, where $E(s)$ is $k \times k$ and $F(s)$ is $k \times (n-k)$. After some unimodular row operations we can assume that $P(s)$ is row proper with row degrees (Kronecker indices) $\nu_1 \geq \cdots \geq \nu_k$. After a possible right multiplication by an $n \times n$ invertible matrix we can assume that the high order coefficient matrix, $P_h$, has the form $[I_k \mid 0]$. Assume that $P(s)$ has no constant rows, i.e. $\nu_k \geq 1$. For $i, j = 1, \ldots, k$ let

$$e_{i,j}(s) \;=\; \sum_{\alpha=0}^{\nu_i} e_{i,j}^{\alpha} s^{\alpha} \qquad\qquad \mathbf{f}_i(s) \;=\; \sum_{\alpha=0}^{\nu_i - 1} \mathbf{f}_i^{\alpha} s^{\alpha}$$

denote the polynomial entries of E(s) and the $i^{th}$ row of $F(s)$ respectively.

Define for $i = 1, \ldots, k$ matrices of sizes $\nu_i \times \nu_i$, $\nu_i \times (n-k)$ and $1 \times \nu_i$ respectively:

$$A_{i,i} := \begin{bmatrix} 0 & \dots & \dots & \dots & -e_{i,i}^0 \\ 1 & 0 & & & -e_{i,i}^1 \\ 0 & 1 & \ddots & & \vdots \\ \vdots & & \ddots & 0 & \vdots \\ 0 & \dots & 0 & 1 & -e_{i,i}^{\nu_i - 1} \end{bmatrix}, \; B_i := \begin{bmatrix} \mathbf{f}_i^0 \\ \mathbf{f}_i^1 \\ \vdots \\ \mathbf{f}_i^{\nu_i - 1} \end{bmatrix}, \; C_i := [0, \ldots, -1]. \tag{4.3.1}$$

For $i, j = 1, \ldots, k$, $i \neq j$ define matrices of size $\nu_i \times \nu_j$:

$$A_{i,j} := \begin{bmatrix} 0 & \ldots & 0 & -e^0_{i,j} \\ \vdots & & \vdots & -e^1_{i,j} \\ \vdots & & \vdots & \vdots \\ 0 & \ldots & 0 & -e^{\nu_i-1}_{i,j} \end{bmatrix} \tag{4.3.2}$$

The matrices $A_{i,i}$ are just the companion matrices for the polynomials $e_{i,i}(s)$, while the matrices $A_{i,j}$ are just $\nu_j - 1$ columns of zeroes with the coefficient vector of the polynomial $e_{i,j}(s)$ appended on the right. Similarly each $B_i$ is just the coefficient vectors of all the polynomials in the $i^{th}$ row of $F(s)$. So these matrices are obtained with no calculations at all, provided that the matrix $P(s)$ meets the somewhat stringent conditions imposed. If $P_h$ does not have the form $P_h = [I_k \mid 0]$ then it can be brought into this form with the unimodular operations outlined above.

Notice also the requirement that $P(s)$ has no constant rows. If $P(s)$ has $\kappa$ constant rows then the row and column operations outlined above will transform $P(s)$ into:

$$\hat{P}(s) = \left[\begin{array}{cc|c} \hat{E}_1(s) & \hat{E}_2(s) & \hat{F}(s) \\ 0 & I_\kappa & 0 \end{array}\right] \tag{4.3.3}$$

and $[\,\hat{E}_1(s) \mid \hat{F}(s)\,]$ has no constant rows.

Right unimodular operations will not affect the GCLD, however left operations will have to be 'undone' once the GCLD of the resulting matrix is calculated. So all of these conditions can be met at the expense of some efficiency. From here on, assume that $P(s)$ meets these requirements.

**Theorem 4.3.1** *[70, 57] Given $P(s) = [\,E(s) \mid F(s)\,]$, satisfying $P_h = [\,I_k \mid 0\,]$ and let $A_{i,j}, B_i, C_i$ be defined as above. Let*

$$A := \begin{bmatrix} A_{1,1} & \cdots & A_{1,k} \\ \vdots & \ddots & \vdots \\ A_{k,1} & \cdots & A_{k,k} \end{bmatrix}, \quad B := \begin{bmatrix} B_1 \\ \vdots \\ B_k \end{bmatrix}, \quad C := \begin{bmatrix} C_1 & & 0 \\ & \ddots & \\ 0 & & C_k \end{bmatrix}$$

*and let $\sigma$ represent either the shift operator or the differential operator $\frac{d}{dt}$. Then*

$$\begin{array}{rcl} \sigma x(t) & = & Ax(t) + Bu(t), \\ y(t) & = & Cx(t) \end{array} \tag{4.3.4}$$

*represents a minimal state space realization of the system*

$$E(\sigma)y(t) + F(\sigma)u(t) = 0. \tag{4.3.5}$$

We see that $A$ has size $\delta \times \delta$ (where $\delta = \sum_{i=1}^{k} \nu_i$), $B$ is $\delta \times (n-k)$, and $C$ is $k \times \delta$.

The idea here is that controllability of the state space representation is equivalent to the controllability of the behavioral system given by $P(s)$ which is equivalent to $P(s)$ being an observable encoder [53, 57].

The relationship between the polynomial matrix $P(s)$ and the matrices $(A, B, C)$ is expressed in the following way: Consider the $k \times \delta$ matrix

$$X(s) = \begin{bmatrix} 1 & s & \cdots & s^{\nu_1-1} & 0 & 0 & \cdots & 0 & & & & & \\ 0 & 0 & \cdots & 0 & 1 & s & \cdots & s^{\nu_2-1} & & & & & \\ & & & & 0 & 0 & \cdots & 0 & & & & & \\ & & & & & & & & \ddots & 0 & 0 & \cdots & 0 \\ & & & & & & & & & 1 & s & \cdots & s^{\nu_k-1} \end{bmatrix} \tag{4.3.6}$$

which was called a basis matrix of size $\nu = [\nu_1, \ldots, \nu_k]$ in [57] since it has the property that every polynomial $k$-vector $\varphi(s) \in \mathbb{F}^k[s]$ whose $i$-th component has degree at most $\nu_i - 1$ can uniquely be described through $\varphi(s) = X(s)\alpha, \alpha \in \mathbb{F}^\delta$.

A direct calculation reveals that $P(s)$ and the matrices $(A, B, C)$ are related by:

$$X(s)[sI - A \mid B] \quad = \quad P(s) \begin{bmatrix} -C & 0 \\ O & I_{n-k} \end{bmatrix} \tag{4.3.7}$$

Of course, we can multiply $X(s)$ by an invertible matrix $T \epsilon GL_\delta$, on the right and obtain the equivalent realization $(T^{-1}AT, T^{-1}B, CT)$. We will make use of this fact in Section 4.5 to obtain a more suitable realization.

## 4.4 The Controllability Space

We are given a $k \times n$ full rank polynomial matrix $P(s)$ and wish to determine its GCLD, $L(s)$. Write $P(s) = L(s)\bar{P}(s)$, where $\bar{P}(s)$ has Smith form $[I_k \mid 0]$. We will assume that the rows of $\bar{P}(s)$ form a minimal basis in the sense of Forney [24]. The row degrees $(\mu_1, \ldots, \mu_k)$ of $\bar{P}(s)$ are therefore the minimal indices of the rational vector space generated by the rows of the matrix $P(s)$. We will not assume that $(\mu_1, \ldots, \mu_k)$ are ordered by size. Also write $P(s) = [E(s) \mid F(s)]$ and let $P_h = [I_k \mid 0]$ be the high order coefficient matrix.

Since $L(s)$ is determined uniquely up to unimodular right multiplication, we have a choice as to which $L(s)$ to work with, and hence which $\bar{P}(s)$ to work with. The following lemma relates $L(s)$ and $\bar{P}(s)$ and it singles out a nice choice:

**Lemma 4.4.1** *If the rows of $\bar{P}(s)$ form a minimal basis having row degrees $\mu_1, \ldots, \mu_k$ then $L(s)$ is uniquely determined from the identity $P(s) = L(s)\bar{P}(s)$. The $(i,j)$-entry of $L(s)$ has degree at most $(\nu_i - \mu_j)$ or the entry is zero.*

*It is possible to choose $\bar{P}(s)$ such that the scalar matrix $L_\infty$ whose $(i,j)$-entry is the coefficient of $s^{\nu_i - \mu_j}$ in the $(i,j)$-entry of $L(s)$ is lower triangular.*

*Proof:* The first part of the lemma is a direct consequence of [24]. The second part will be established by induction. Using elementary column operations on $L(s)$ (this corresponds to elementary row operations on $\bar{P}(s)$) it will be possible to eliminate all entries of the first row of $L_\infty$ with the exception of one entry. After a possible permutation of the columns we can assume that the first row of $L_\infty$ has with the exception of the entry $(1,1)$ all entries equal to zero. Proceeding inductively row by row will establish the claim. $\square$

Let $\bar{P}_h$ be the high order coefficient matrix of $\bar{P}(s)$. From the fact that both $\bar{P}_h$ and $P_h$ have rank $k$ and from the identity $P_h = L_\infty \bar{P}_h$ it follows that $L_\infty$ is invertible. As a direct consequence we have:

**Lemma 4.4.2** *Let $d := \sum \mu_i$ be the McMillan degree of $\bar{P}(s)$. Then*

$$\deg \det L(s) = \delta - d = \delta - \operatorname{rank} \Phi(A, B).$$

Lemma 4.4.2 establishes a first relation between the GCLD, $L(s)$, and the controllability matrix $\Phi(A, B)$. It should be noted that this result, for the case $k = 1$, was known already in 1950 by MacDuffee [45] if not earlier. It is the goal of this and the next section to show that under certain conditions it is possible to compute $L(s)$ from the column space of $\Phi(A, B)$, i.e. from the reachability space of $(A, B)$.

Since the high order coefficient matrix of $P(s)$ has the form $[I_k \mid 0]$ we can realize $P(s)$ by inspection to obtain the scalar matrices $A$, $B$, and $C$ relative to the basis matrix $X(s)$. Hence the following equation holds:

$$X(s)[sI - A \mid B] \quad = \quad [-E(s)C \mid F(s)] \tag{4.4.8}$$

Note that in order to realize $\bar{P}(s)$, we need $\bar{P}_h = [I_k \mid 0]$ and that the row degrees, $\mu_i$, of $\bar{P}(s)$ are at least one. To satisfy the first requirement, in general, we will have to multiply $\bar{E}(s)$ by $T$ to obtain a realizable form. *i.e.* $\bar{P}_r(s) = [\bar{E}(s)T \mid \bar{F}(s)]$. The second requirement cannot be guaranteed. For this section and the following one we will assume that $\bar{P}(s)$ has no constant rows, and the case where there are constant rows is considered in Section 4.6.

Now, realize $\bar{P}_r(s)$ to obtain matrices $\bar{A}$, $\bar{B}$, and $\bar{C}$, relative to the canonical basis matrix $\tilde{X}(s)$. Hence the following equation holds:

$$\tilde{X}(s)[sI - \bar{A} \mid \bar{B}] \;=\; [-\bar{E}(s)T\bar{C} \mid \bar{F}(s)] \tag{4.4.9}$$

The controllability matrix of the pair $(\bar{A}, \bar{B})$ can also be computed. However, the usual definition of the controllability matrix is that of a $d \times d(n-k)$ matrix, where $\bar{B}$ is of size $d \times (n-k)$. We can, however, naturally extend the size of this matrix to $d \times \delta(n-k)$. This is necessary for the following key result.

**Theorem 4.4.3**

$$L(s)\tilde{X}(s)\Phi(\bar{A}, \bar{B}) \;=\; X(s)\Phi(A, B)$$

*Proof:* Repeated applications of (4.4.8) give:

$$X(s)\Phi(A,B) \;=$$

$$\left[F \mid sF + ECB \mid \cdots \mid s^{\delta-1}F + s^{\delta-2}ECB + s^{\delta-3}ECAB + \cdots + ECA^{\delta-2}B\right]$$

Repeated applications of (4.4.9) give:

$$L(s)\tilde{X}(s)\Phi(\bar{A}, \bar{B}) =$$

$$\left[F \mid sF + ET\bar{C}\bar{B} \mid \cdots \mid s^{\delta-1}F + s^{\delta-2}ET\bar{C}\bar{B} + s^{\delta-3}ET\bar{C}\bar{A}\bar{B} + \cdots + ET\bar{C}\bar{A}^{\delta-2}\bar{B}\right]$$

By examining the above expressions, it is clear that the only step remaining in the proof is to show that $CA^iB = T\bar{C}\bar{A}^i\bar{B}$ for all non-negative integer $i$.

We notice that $(sI - A)^{-1} = \sum_{i=0}^{\infty} \frac{A^i}{s^{i+1}}$. Starting with the equation $X(s)(sI - A) = -E(s)C$, we apply this inverse to obtain $X(s) = -E(s)\sum_{i=o}^{\infty} \frac{CA^i}{s^{i+1}}$. Further:

$$F(s) = X(s)B = -E(s)\sum_{i=o}^{\infty} \frac{CA^iB}{s^{i+1}}.$$

Similarly, we have the equations:

$$\bar{F}(s) = \tilde{X}(s)\bar{B} = -\bar{E}(s)T\sum_{i=o}^{\infty} \frac{\bar{C}\bar{A}^i\bar{B}}{s^{i+1}}.$$

Multiplying the last equation by $L(s)$ results in

$$F(s) = -E(s)\sum_{i=o}^{\infty} \frac{T\bar{C}\bar{A}^i\bar{B}}{s^{i+1}}.$$

Since $E(s)$ has high order coefficient matrix $I_k$, the columns of $E(s)$ are linearly independent over $\mathbb{F}[s]$ and we get:

$$\sum_{i=o}^{\infty} \frac{CA^iB}{s^{i+1}} \;=\; \sum_{i=o}^{\infty} \frac{T\bar{C}\bar{A}^i\bar{B}}{s^{i+1}}$$

Equating coefficients in the above expression gives us the desired equality and completes the proof. $\square$

This theorem is the key to the entire algorithm as the following corollary shows.

**Corollary 4.4.4** *There exists an invertible matrix $W \in Gl_{(n-k)\delta}$ such that*

$$X(s)\Phi(A,B)W = \left[s^{\mu_1-1}\ell_1 \dots s\ell_1\, \ell_1 \mid \dots \mid s^{\mu_k-1}\ell_k \dots s\ell_k\, \ell_k \;\mid\; O_{k\times((n-k)\delta-d)}\right]. \tag{4.4.10}$$

*In this representation the $k\times k$ matrix*

$$L(s) = [\ell_1, \dots, \ell_k]$$

*represents a greatest common left divisor of $[E(s)\;\; F(s)]$ and $\mu_1,\dots,\mu_k$ are the row degrees of $[\bar{E}(s)\;\; \bar{F}(s)]$.*

*Proof:* Since $\bar{P}_r(s)$ is a minimal basis, its realization, $(\bar{A},\bar{B})$, must be a controllable pair. Therefore, there exists a scalar matrix $W \in Gl_{(n-k)\delta}$ such that $\Phi(\bar{A},\bar{B})W = \left[I_d \mid O_{k\times((n-k)\delta-d)}\right]$. Hence, the theorem implies that $X(s)\Phi(A,B)$ is column equivalent to a matrix whose columns are exactly the columns of a GCLD and also multiples of these columns (as the multiplication $\tilde{X}(s)\Phi(\bar{A},\bar{B})$ indicates). □

## 4.5 The Refining Algorithm

By Theorem 4.4.3 and Corollary 4.4.4, the columns of $L(s)$ are contained in a matrix that is column equivalent (over $\mathbb{F}$) to $X(s)\Phi(A,B)$. The question is now, how to select these $k$ columns of $L(s)$ from the $(n-k)\delta$ columns of the controllability matrix? The answer is fairly simple: column reduce and then choose the appropriate $k$ columns in a manner that will be described below. However, we must first reconsider our choice of basis matrix $X(s)$. The reason we have started with the one we have chosen is that it allows us to write down the matrices $A$ and $B$ a little easier. The downside is that when we column reduce the controllability matrix we start by eliminating the lower degree terms of the polynomials in row 1 of the corresponding matrix $X(s)\mathcal{C}(A,B)$. It would make much more sense to start eliminating the highest degree terms in each row. We accomplish this by replacing the standard basis matrix $X(s)$ introduced in (4.3.6) with the basis matrix $\mathfrak{X}(s) =$

$$\begin{bmatrix} s^{\nu_1-1} & 0 & & & s^{\nu_1-2} & 0 & & & \cdots & s^{\nu_1-\nu_1} & 0 & & 0 \\ 0 & s^{\nu_2-1} & & & 0 & s^{\nu_2-2} & & & \cdots & 0 & s^{\nu_2-\nu_1} & & \\ & & \ddots & & & & \ddots & & & & & \ddots & 0 \\ & & & s^{\nu_k-1} & 0 & & & s^{\nu_k-2} & \cdots & 0 & & 0 & s^{\nu_k-\nu_1} \end{bmatrix}$$

In this representation, the monom $s^\beta$ and the corresponding column is omitted as soon as the exponent $\beta < 0$. $\mathfrak{X}(s)$ and $X(s)$ are related by a simple permutation of the columns, i.e. there is a permutation matrix $U$ such that $\mathfrak{X}(s) = X(s)U$. This permutation transforms the controllability matrix $\Phi(A,B)$ into $U^{-1}\Phi(A,B)$.

Although it is much simpler to explain the algorithm by performing the $U$ transformation as above, in practice the computer would automatically perform the realization with respect to the new basis matrix $\mathfrak{X}$ and arrive at $U^{-1}AU$ and $U^{-1}B$ instead of $A$ and $B$. The realization with respect to the new basis matrix is just as simple to compute as the original, yet it is in a more practical form and, by arriving at it directly, will not waste time by transforming basis matrices.

As mentioned earlier, the basis matrix $\mathfrak{X}(s)$ (as well as the basis matrix $X(s)$) has the property that every polynomial $k$-vector $\varphi(s) \in \mathbb{F}^k[s]$ whose $i$-th component has degree at most $\nu_i - 1$ can uniquely be described through $\varphi(s) = \mathfrak{X}(s)\alpha, \alpha \in \mathbb{F}^\delta$. It is therefore possible to identify $\varphi(s)$ with the $\delta$-vector $\alpha$. We will say that $\alpha$ is the *coordinate vector* of $\varphi(s)$ with respect to the basis matrix $\mathfrak{X}(s)$.

**Theorem 4.5.1** *Assume $P(s)$ has Kronecker indices $\nu_1 \geq \dots \geq \nu_k$ and minimal indices $\mu_1,\dots,\mu_k$ none of which equal zero. Let $L(s) = [\ell_1,\dots,\ell_k]$ be a GCLD whose $(i,j)$-entry has degree at most $\nu_i - \mu_j$ or is zero. Assume that the matrix $L_\infty$ is lower triangular (by Lemma 4.4.1) and let $d = \sum_{i=1}^k \mu_i$. Then the $\delta\times d$ scalar matrix whose columns form the coordinate vectors of*

$$\left[s^{\mu_1-1}\ell_1 \dots s\ell_1\, \ell_1 \mid \dots \mid s^{\mu_k-1}\ell_k \dots s\ell_k\, \ell_k\right] \tag{4.5.11}$$

*is after a possible permutation of the columns in column echelon form.*

*Proof:* Immediate consequence from the fact that $L_\infty$ is lower triangular, has nonzero diagonal elements and the specific choice of the basis matrix $\mathfrak{X}(s)$. $\square$

As a consequence of this theorem we can immediately read out the minimal indices $\mu_1, \ldots, \mu_k$ from the pivot indices of the column echelon form of $\Phi(A, B)$. A priori it is not true that $\mathfrak{X}(s)\Phi(A, B)$ has the particular form (4.5.11) even if $\Phi(A, B)$ is in column echelon form. One observes however that elementary column operations on $\Phi(A, B)$ correspond to unimodular operations on $\mathfrak{X}(s)\Phi(A, B)$. By Theorem 4.4.3 we also know that the columns of $\mathfrak{X}(s)\Phi(A, B)$ are in the column module of $L(s)$. By the above remarks it is possible to identify $k$ columns $[c_1, \ldots, c_k]$ from the column echelon form of $\Phi(A, B)$ such that $\mathfrak{X}(s)[c_1, \ldots, c_k]$ forms a GCLD of $P(s)$. In the sequel we make this selection process more precise.

Assume that the controllability matrix $\Phi(A, B)$ is in column echelon form. We can think of the controllability matrix as being divided into row blocks. The top row block consists of $k$ rows and corresponds (under multiplication by $\mathfrak{X}(s)$) to coefficients of degree $\nu_i - 1$ for each respective row $i$. The next lower block corresponds to coefficients of degree $\nu_i - 2$. Each lower block is similarly defined. If $\nu_j - \beta < 0$ then no row corresponding to row $j$ occurs in row block $\beta$ (or any subsequent blocks). Based on this we define:

**Definition 4.5.2** 1. A column in the controllability matrix $\Phi(A, B)$ is said to "take its order in row $i$" if the leading coefficient occurs in a row which corresponds (under multiplication by $\mathfrak{X}(s)$) to an entry in row $i$ of the resulting polynomial $k$-vector.

2. For each row $i$, $1 \leq i \leq k$, consider all the column vectors of the controllability matrix taking their order in row $i$. From this set, the column vector whose leading coefficient is lowest (in the matrix, not necessarily in value), is called the "row leader for row $i$".

**Theorem 4.5.3** *If the column echelon form of $\Phi(A, B)$ has $k$ row leaders $[c_1, \ldots, c_k]$ then $\mathfrak{X}(s)[c_1, \ldots, c_k]$ forms a GCLD of $P(s)$.*

*Proof:* It follows from our definition of row leaders $[c_1, \ldots, c_k]$ that

$$\deg \det \mathfrak{X}(s)[c_1, \ldots, c_k] = \sum_{i=1}^{k} \nu_i - \sum_{i=1}^{k} \mu_i = \delta - d.$$

Since $\mathfrak{X}(s)[c_1, \ldots, c_k]$ is a subset of the column module of $L(s)$ it follows that the columns of $\mathfrak{X}(s)[c_1, \ldots, c_k]$ generate this column module and this completes the proof. $\square$

**Remark 4.5.4** It can be shown and it is illustrated in an example in Section 4.8 that in the case of $(n - k) = k = 1$, i.e. in the situation where $P(s) = (p_1(s), p_2(s))$ the column reduction of the controllability matrix $\Phi(A, B)$ is exactly the Euclidean algorithm. The presented algorithm generalizes in this way Euclid's algorithm.

**Remark 4.5.5** The column reduction of $\Phi(A, B)$ can be done very efficiently by iteratively computing the vectors $A^i b_j$, where $b_j$ is the $j$-th column of $B$. (See [66, 2] for more details). Due to the very sparse structure of $(A, B)$ the column reduction is even easier.

### 4.6 The Situation of Constant Rows

As remarked earlier, the matrix $\bar{P}(s)$ that is used in the proof of our algorithm could have constant rows, and that poses problems when we try to realize this matrix. In this section we will deal with this case. In particular, assume that $\bar{P}(s)$ has $0 \leq \lambda \leq k$ constant rows ($\mu_i = 0$ for $1 \leq i \leq \lambda$).

Similar to before, we know that $\bar{P}(s)$ has (after possible right scalar multiplication) the form:

$$\bar{P}(s) = \left[ \begin{array}{cc|c} I_k & 0 & 0 \\ 0 & \bar{E}(s) & \bar{F}(s) \end{array} \right] \tag{4.6.12}$$

Letting $\bar{P}_r(s) = [\bar{E}(s) \mid \bar{F}(s)]$, we can obtain the realization matrices $\bar{A}$, $\bar{B}$, and $\bar{C}$ relative to the basis matrix $\tilde{X}(s)$ for $\bar{P}_r(s)$. The following result can easily be shown using arguments mirroring those in the proof of Theorem 4.4.3:

**Theorem 4.6.1**

$$L(s)\begin{bmatrix} 0 \\ \tilde{X}(s) \end{bmatrix}\Phi(\bar{A},\bar{B}) = X(s)\Phi(A,B)$$

In analogy to Corollary 4.4.4 we have:

**Corollary 4.6.2** *Let $\mu_{\lambda+1},\ldots,\mu_k$ be the nonzero minimal indices of $P(s)$. Then there exists an invertible matrix $W \in Gl_{(n-k)\delta}$ such that*

$$X(s)\Phi(A,B)W = \left[s^{\mu_{\lambda+1}-1}\ell_{\lambda+1}\ldots s\ell_{\lambda+1}\,\ell_{\lambda+1} \mid \ldots \mid s^{\mu_k-1}\ell_k \ldots s\ell_k\,\ell_k \;\mid\; O_{k\times((n-k)\delta-d)}\right]. \quad (4.6.13)$$

*In this representation the $[\ell_{\lambda+1},\ldots,\ell_k]$ represent $k-\lambda$ generators of the column module of $P(s)$.*

By Lemma 4.4.2 we know that the rank $\Phi(A,B) = \sum_{i=1}^{k}\mu_i = d$. Combining this with Corollary 4.6.2 and Theorem 4.5.3 results in:

**Theorem 4.6.3** *If $P(s)$ has $\lambda$ zero minimal indices then the column echelon form of $\Phi(A,B)$ has $k-\lambda$ row leaders $[c_{\lambda+1},\ldots,c_k]$ and the columns of $\mathfrak{X}(s)[c_{\lambda+1},\ldots,c_k]$ form an independent set of generators for a GCLD of $P(s)$.*

By this last theorem we will be able to compute the number, $\lambda$, of nonzero minimal indices of $P(s)$ and we always will be able to identify $k-\lambda$ 'row leaders' from the echelon form of $\Phi(A,B)$. This is very important. Otherwise, we could perform the algorithm, get $k$ columns and think we are done, when in reality we would have selected columns that are unimodularly equivalent and ended up with a singular matrix!

The question now turns to: How do we select the remaining $\lambda$ columns to fill up our matrix and arrive at a GCLD?

The answer is actually quite simple. For this consider the high order coefficient matrix $H$ of the $k\times(k-\lambda)$ matrix $\mathfrak{X}(s)[c_{\lambda+1},\ldots,c_k]$. This high order coefficient matrix is a sub-matrix of the matrix $L_\infty$, introduced in Lemma 4.4.1. The high order coefficient matrix of $P(s)$ is assumed to be $P_h = [I_k\ 0]$. It is therefore possible to augment $H$ with columns from $P_h$ such that the overall matrix $L_\infty$ becomes invertible. Correspondingly we have a way of selecting $\lambda$ columns from the first $k$ columns of $P(s)$ such that $\mathfrak{X}(s)[c_{\lambda+1},\ldots,c_k]$ augmented by these columns results in a GCLD of $P(s)$. Simply put: For every row, $i$, which does not have a row leader, simply select column $i$ from the matrix $P(s)$ to be in $L(s)$.

## 4.7 The Algorithm

We now present the algorithm of computing a GCLD in a concise form:

Step 1 We are given a full rank polynomial matrix $P(s)$.

Step 2 Check if the high order coefficient matrix $P_h$ has the form $[I \mid 0]$. If not, then use right and left unimodular operations to bring it into this form. Keep track of any left unimodular operations in the matrix $V(s)$.

Step 3 Check if $P(s)$ has any constant rows. If $P(s)$ has $\kappa$ constant rows and is in the form (4.3.3) then the sub-matrix $\left[\begin{smallmatrix}\hat{E}_2(s)\\ I_\kappa\end{smallmatrix}\right]$ of (4.3.3) defines $\kappa$ generators of a GCLD $L(s)$. Continue the algorithm with the reduced matrix $[\,\hat{E}_1(s) \mid \hat{F}(s)\,]$ in order to find the remaining $k-\kappa$ columns of the GCLD.

Step 4 Obtain the realization matrices $A$ and $B$ relative to the basis matrix $\mathfrak{X}(s)$ 'by inspection'.

Step 5 Calculate the controllability matrix $\mathcal{C}(A,B)$ and column reduce it. (This may be done simultaneously to greatly improve efficiency [66, 2].)

Step 6 Pick out the 'row leaders' from the column reduced controllability matrix $\mathcal{C}(A,B)$. Multiply the 'row leaders' by $\mathfrak{X}(s)$ and place them in the GCLD.

Step 7 If there are $k$ row leaders, then go to step 8. If there are less than $k$ row leaders, then follow the algorithm of Section 4.6.

Step 8 Multiply the GCLD on the left by $V^{-1}$ and stop.

**Remark 4.7.1** The steps which take the most time are steps 2 and 5. Step 2 is not necessary when $P(s)$ is in the desired form. Of course, in general we will not know or can not guarantee what form a matrix will have. However, in certain applications, such as searching for observable convolutional encoders [2, 18], we may prescribe what form the matrices will have.

After having computed the GCLD, $L(s)$, there might arise the need to compute the 'controllable part' $\bar{P}(s)$ as well. Let $\mathbf{p}_i(s)$ and $\tilde{\mathbf{p}}_i(s)$ denote the $i$th column of $P(s)$ and $\bar{P}(s)$ respectively, $i = 1, \ldots, n$. Consider for each index $i$ the equation

$$L(s)\tilde{\mathbf{p}}_i(s) = \mathbf{p}_i(s). \tag{4.7.14}$$

We can view (4.7.14) as a system of $\delta + k$ linear equations in $d + k$ unknowns. We therefore have to solve simultaneously $n$ systems of equations in $d + k$ unknowns. Due to the fact that the matrix $L_\infty$ is already in lower triangular form it follows that the coefficient matrix appearing in (4.7.14) is already in triangular form as well. A solution of (4.7.14) can therefore be computed very efficiently and the method will be illustrated in the next section.

## 4.8 Examples

We have included some examples to aid in the understanding of the algorithm.

**Example 4.8.1** First, take the case when $P(s)$ is a $1 \times 2$ matrix. In this case we are just determining the GCD of two polynomials. Notice that $P(s)$ will trivially satisfy all of the conditions unless the two polynomials have the same degree. In that case divide one into the other, and take the remainder in place of the original polynomial.

Let us work through the following example. Let $P(s) =$

$$[\,s^6 + 5s^5 - 464s^4 + 1123s^3 - 887s^2 + 234s + 72 \qquad s^5 - 2s^4 - 342s^3 + 1177s^2 - 1170s + 504\,]$$

We get the following realization:

$$A = \begin{bmatrix} -5 & 1 & 0 & 0 & 0 & 0 \\ 464 & 0 & 1 & 0 & 0 & 0 \\ -1123 & 0 & 0 & 1 & 0 & 0 \\ 887 & 0 & 0 & 0 & 1 & 0 \\ -234 & 0 & 0 & 0 & 0 & 1 \\ -72 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} \qquad B = \begin{bmatrix} 1 \\ -2 \\ -342 \\ 1177 \\ -1170 \\ 504 \end{bmatrix}$$

relative to the basis matrix $\mathfrak{X}(s) = [s^5\; s^4\; s^3\; s^2\; s\; 1]$. The corresponding column reduced controllability matrix is:

$$\Phi(A,B) = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ -2 & 1 & 0 & 0 & 0 & 0 \\ -342 & \frac{-65}{3} & 1 & 0 & 0 & 0 \\ 1177 & \frac{221}{3} & -19 & 0 & 0 & 0 \\ -1170 & \frac{-220}{3} & 23 & 0 & 0 & 0 \\ 504 & 32 & -12 & 0 & 0 & 0 \end{bmatrix}$$

Since there is only one row of $\mathfrak{X}(s)\mathcal{C}(A,B)$, the row leader must be the rightmost nonzero column. Hence the GCLD is $s^3 - 19s^2 + 23s - 12$.

Notice that the first column of the above matrix corresponds with polynomial of lesser degree from our original matrix. The second column corresponds with the 'first remainder' that one obtains when applying the Euclidean algorithm to the two polynomials in our matrix. The third column corresponds with the 'second remainder', and also the last nonzero one, of the Euclidean algorithm. Because of this, our algorithm can be seen as an extension of the Euclidean algorithm to matrices.

**Example 4.8.2** Now let us look at a more nontrivial example.

$$P(s) := \begin{bmatrix} s^5 & s^4+s^2 & s^4+2\,s^2 \\ s & s^3+s^2+s+1 & 2\,s+3 \end{bmatrix}$$

Here $\nu_1 = 5, \nu_2 = 3$ and the realization matrices are

$$A = \begin{bmatrix} 0 & -1 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & -1 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ -1 & -1 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & -1 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & -1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} \qquad B = \begin{bmatrix} 1 \\ 0 \\ 0 \\ 2 \\ 2 \\ 3 \\ 0 \\ 0 \end{bmatrix}$$

relative to the basis matrix

$$\mathfrak{X}(s) = \begin{bmatrix} s^4 & 0 & s^3 & 0 & s^2 & 0 & s & 1 \\ 0 & s^2 & 0 & s & 0 & 1 & 0 & 0 \end{bmatrix}.$$

The column reduced controllability matrix is

$$\mathcal{C}(A,B) = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & -1 & 1 & 0 & 0 & 0 & 0 & 0 \\ 1 & -1 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

We see that columns 2 and 3 take their order in row 2, while column 1 is the only column taking its order in row 1. Hence column 1 is the 'row leader' for row 1 and column 3 is the 'row leader' for row 2. It follows that $\mu_1 = 1$ and $\mu_2 = 2$. As an independent verification, we can also see directly from $\mathfrak{X}(s)\mathcal{C}(A,B)$ that column 2 is just $s-1$ times column 3 and hence they are dependent.

$$\mathfrak{X}(s)\mathcal{C}(A,B) = \begin{bmatrix} s^4 & s^3-s^2 & s^2 & 0 & 0 & 0 & 0 & 0 \\ 1 & s^2-1 & s+1 & 0 & 0 & 0 & 0 & 0 \end{bmatrix},$$

so the GCLD is

$$L(s) = \begin{bmatrix} s^4 & s^2 \\ 1 & s+1 \end{bmatrix}.$$

We can now also easily compute $\bar{P}(s)$ by solving the following linear system of equations:

$$\begin{bmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} a_1 & a_2 & a_3 \\ b_1 & b_2 & b_3 \\ c_1 & c_2 & c_3 \\ d_1 & d_2 & d_3 \\ e_1 & e_2 & e_3 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 1 & 1 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \\ 1 & 1 & 2 \\ 0 & 1 & 2 \\ 0 & 1 & 3 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix}$$

This corresponds to the equation $L(s)\bar{P}(s) = P(s)$, where $\bar{P}(s)$ is represented by the matrix:

$$\left[ \begin{array}{ccc} a_1 s + b_1 & a_2 s + b_2 & a_3 s + b_3 \\ c_1 s^2 + d_1 s + e_1 & c_2 s^2 + d_2 s + e_2 & c_3 s^2 + d_3 s + e_3 \end{array} \right].$$

The left-hand matrix in the above equation comes easily from the column reduced controllability matrix. It consists of the 'row leaders' plus 'shifts' of the row leaders. To be precise, for each $i$, the row leader for column $i$ occurs, along with $\mu_i$ upward 'shifts' of the row leader. Note that this necessitates adding another row block to the top of the scalar matrix.

The right-hand matrix is simply the coefficients of the matrix $P(s)$ with respect to the 'augmented basis matrix' $\left[ \begin{array}{cc|c} s^5 & 0 & \multirow{2}{*}{$\mathfrak{X}(s)$} \\ 0 & s^3 & \end{array} \right]$.

Not only is the left-hand matrix easily constructed, but it will be lower diagonal (up to column permutations) so that the above system can be solved instantaneously! The resulting matrix $\bar{P}(s)$ can now be stated:

$$\bar{P}(s) = \left[ \begin{array}{ccc} s & 0 & 1 \\ 0 & s^2+1 & 2 \end{array} \right].$$

CHAPTER 5

# DECODING ERROR CORRECTING CODES

This chapter is included as a basic introduction to the decoding of error correcting codes. It is intended to provide the necessary background for subsequent chapters for those readers who are not familiar with this subject. A few of the more basic concepts are presented in detail, while some others are only briefly introduced with references to works containing a more thorough discussion. The reader is referred to [43, 44, 49] for general references on decoding. In addition, Section 5.5 provides new insight into *majority logic decoding* methods using the local description of convolutional codes presented in 2.2.4.

## 5.1 Channel Models and Maximum Likelihood Decoding

We must be able to decode the information that we encode if it will be of any use. The fact that our encoded information is often corrupted by channel noise makes this task non-trivial. We have seen in Section 1.1 that the decoding process can be divided into two function $\beta$ and $\psi^{-1}$. The map $\beta$ attempts to remove the noise from the received word, thereby estimating which codeword was indeed sent. The map $\psi^{-1}$ reverses the encoding process to determine what message was encoded. In practice, the map $\psi^{-1}$ is already specified by choosing the encoder, $\psi$, and hence can be considered part of the encoding process. So the heart of the decoding task is to find an efficient and effective $\beta$.

Let us discuss what we mean by efficient and effective. By efficient we mean the decoder should work in a timely fashion and should be reasonably cheap and easy to build and maintain. These are concepts mainly associated with the 'engineering side' of coding theory, and hence, fall outside the domain of this dissertation. Although, it is certainly a mathematical issue in evaluating the complexity of the decoding algorithm prescribed by $\beta$. By effective we mean, informally, that the map $\beta$ chooses the correct codeword with very high probability. Let us introduce some of the concepts that will make this idea more concrete.

**Definition 5.1.1** Given a received word $\mathbf{r}$, and the set of all possible codewords $\{\mathbf{x_i}\}_{i\in I}$. Let $Pr(\mathbf{x_i}|\mathbf{r})$ denote the probability that the codeword $\mathbf{x_i}$ was sent given that the word $\mathbf{r}$ was received. Then, defining $\beta$ as follows, $\beta$ is said to be a *maximum likelihood decoder.*

$$\beta(\mathbf{r}) \;=\; \mathbf{x_j}, \text{ where } Pr(\mathbf{x_j}|\mathbf{r}) \;=\; \max_{i\in I} Pr(\mathbf{x_i}|\mathbf{r})$$

It is clear that this is intuitively the correct approach for constructing a good decoder. However, there are two immediate issues which arise from this definition. First, how does one compute the indicated probabilities. Second, for large codes it is impossible to compute all of these probabilities individually in a timely manner.

Let us address the first issue. (We will delay discussion of the second issue until Section 5.3.) To compute these probabilities it is necessary to give our communication channel a 'model'. This model must accurately reflect the real-life channel that it represents and, hopefully, it should have some definable mathematical properties. In reality it is not surprising that there are many different kinds of channels and hence we must develop a separate model for each of them. We will content ourselves by considering two of the major channels.

**Definition 5.1.2** A *binary symmetric channel* (BSC) is one such that $\mathcal{A} = \mathbb{F}_2$ and if we define $Pr(0|1) = p_0$ and $Pr(1|0) = p_1$ to be the probabilities that a transmitted 0 will be received as a 1, and that a 1 will be received as a 0 respectively, then $p_0 = p_1 := p$. Similarly, a *q-ary symmetric channel* is one such that $\mathcal{A} = \mathbb{F}_q$ where $Pr(a_i|a_i) = 1 - p$ for all $a_i \in \mathbb{F}_q$ and $Pr(a_i|a_j) = p/(q-1)$ for all $a_i \neq a_j$.

So, if we are given a block code over a BSC with 'transition probability' $p$, and a received word $\mathbf{r}$ it is a simple matter of computing binomial probabilities to determine which codeword is the most likely. Unless we are transmitting at a rate above the channel capacity, it is true that $p < 1/2$.

Therefore the issue of finding the most likely codeword is the same as finding the codeword which differs from $\mathbf{r}$ in the fewest components. That is, $\beta$ should choose $\mathbf{x_j}$ such that

$$\operatorname{dist}(\mathbf{x_j}, \mathbf{r}) = \min_{i \in I} \operatorname{dist}(\mathbf{x_i}, \mathbf{r})$$

Such a decoding scheme is called *nearest neighbor decoding.* For many channels, nearest neighbor decoding and maximum likelihood decoding are equivalent.

Our first channel model makes use of *hard decision* decoding in that each bit of the received word is immediately estimated as either a 0 or 1. The probability of each estimate being incorrect is given by the channel transition probability $p$. While mathematically this is very nice, it is often not the best channel model. From a practical point of view one can think of the transmission of 0's and 1's by sending a voltage of -1 or 1. (This is an extremely oversimplified description.) The noise occurring during transmission over the channel can be modeled by adding or subtracting some amount of voltage. So for each bit the decoder will receive some real-valued voltage. Instead of making a hard decision on the bit (*i.e.* decoding all negative voltages as 0 and all positive voltages as 1), the decoder can assign a probability that each bit is a 1 (or 0) by using some channel information. In particular, it is reasonable to assume that the noise adheres to some sort of normal distribution with mean 0 and standard deviation $\sigma$. The standard deviation depends (primarily) on the relative strength of the energy per message bit $E_b$ to the noise power spectral density $N_0/2$. The ratio $E_b/N_0$ is known as the *signal to noise ratio* (SNR). Thankfully, one need not know what a power spectral density is to compute $\sigma$. In practice, the knowledge of $E_b/N_0$ given in decibels and the rate, $R = k/n$ are all that is needed to obtain $\sigma$.

$$\sigma = 10^{-E_b/20N_0}\sqrt{1/2R}$$

**Definition 5.1.3** The channel we have just described is called an *additive white Gaussian noise* (AWGN) channel or simply a Gaussian channel.

A more thorough examination of this and other channels can be found in [17]. It is clear that the probabilities we desired to compute can be gotten easily from the normal distribution associated with the noise.

### 5.2 Decoding Parameters

Let us consider a hard decision channel where maximum likelihood decoding is equivalent to nearest neighbor decoding (e.g. a BSC). A very important parameter in determining the effectiveness of a code and decoding algorithm is the *codeword error rate.* Simply put, this is just the percentage of transmitted codewords that are decoded incorrectly. For block codes using a nearest neighbor decoding algorithm, the following theorem and corollary state the importance of the minimum distance of the code in regards to the codeword error rate.

**Theorem 5.2.1** *[49, amongst many others] Let a block code, $\mathcal{C}$, have minimum distance $d$. Let $t = \lfloor (d-1)/2 \rfloor$. Then the code can <u>detect</u> up to $d-1$ errors and it can <u>correct</u> up to to $t$ errors.*

*Proof:* Let $\mathbf{x}$ be the sent codeword and $\mathbf{r} = \mathbf{x}+\mathbf{e}$ be the received word. An error will be undetected if $\mathbf{r} = \mathbf{x_1} \in \mathcal{C}$ ($\mathbf{x_1} \neq \mathbf{x}$). If less than $d$ errors occur, then $\operatorname{wt} \mathbf{e} < d$ and hence by the triangle inequality (Hamming distance defines a metric!), $\mathbf{r}$ cannot be another codeword. Hence the error will be detected.

Consider the vector space $\mathbb{F}^n$ and our code $\mathcal{C} \subset \mathbb{F}^n$. For each codeword we may define a ball of radius $\rho$ for any $\rho \geq 0$. Simply, these balls contain all vectors in $\mathbb{F}^n$ whose distance from the center (codeword) are less than or equal to $\rho$. When $\rho \leq t$, it is clear that the balls are disjoint. Hence, when $t$ or less errors occur, $\mathbf{r}$ will lie in exactly one of the balls. Nearest neighbor decoding specifies that we decode as the codeword in the center of that ball. □

**Corollary 5.2.2** *The codeword error rate, $P_e$, for a rate $k/n$ (binary) block code with minimum distance $d$ ($t = \lfloor (d-1)/2 \rfloor$) transmitted over a BSC with channel transition probability $p$ is given by*

$$P_e \leq \sum_{i=t+1}^{n} \binom{n}{i} p^i (1-p)^{n-i}$$

*Proof:* The right side of the above inequality is just the probability of more than $t$ errors occurring in any codeword. (The inequality can be explained by the fact that for most codes, the spheres of radius $t$ about each codeword do not cover the entire vector space $\mathbb{F}^n$. If one of these vectors is received, the decoder may 'luck' into a correct decoding.) $\square$

On the other hand, when a soft-decision decoding algorithm is employed or for convolutional codes the notion of codeword error is not as important as the *bit error rate* or *bit error probability.* Simply, this is just the probability of a message bit being decoded incorrectly, and is denoted by $P_b$. Similar formulas exists for computing (or bounding) this probability for the various channel models.

## 5.3 A Few Decoding Algorithms

We would like to address the second issue which arose in the discussion on computing the relative probabilities of codewords given a received word. Namely, that it is infeasible to compute every such probability. We shall present some of the most fundamental decoding techniques to show how this problem can be overcome.

*Syndrome decoding* is a hard decision method for decoding block codes. We start with a parity check matrix, $H$, for the code. Then we compute the cosets of the code as an additive subgroup of $\mathbb{F}^n$. For each coset, we choose a vector with minimal weight and call it the *coset leader.* For every vector $\mathbf{x} \in \mathbb{F}^n$, the vector $\mathbf{x}H^T$ is called the syndrome of the vector. It is true that all vectors in the same coset have the same syndrome. Since every vector $\mathbf{c}$ in our code satisfies $\mathbf{c}H^T = 0$, all of the codewords lie in the coset whose leader is the all zero vector and whose syndrome is the all zero vector.

Given a received word $\mathbf{r} = \mathbf{c} + \mathbf{e}$, where $\mathbf{e}$ is the error vector, we decode in the following way. Compute the syndrome of $\mathbf{r}$, which equals the syndrome of $\mathbf{e}$. This syndrome must equal one of the syndromes of the coset leaders. The error vector must be one of the vectors in that coset. The most likely error vector is the one with minimal weight, *i.e.* the coset leader. (If more than one minimal weight vector exist, choose one.) Subtract the coset leader from the received word to obtain the decoded word.

This is a very nice scheme from a mathematical viewpoint, but even the task of storing the coset leaders and their syndromes can become quite costly and cumbersome for large codes. Some codes have even more efficient ways of syndrome decoding. Hamming codes and BCH codes [49] are good examples of codes which can compute the error vector algebraically rather than having to store the complete list of coset leaders. There is also the technique of *majority logic decoding* using *orthogonal parity checks.* We will explore this technique as it is used to decode convolutional codes in Section 5.4 and then proceed to relate this technique to our local description of convolutional codes in Section 5.5.

Soft-decision decoding of block codes is becoming increasingly popular. Gallager developed a soft-decision algorithm based roughly on majority logic decoding in the early Sixties [26, 27]. While his methods showed promise for large block sizes, the method was widely ignored, probably due to the lack of sufficient technology to implement the decoding practically. The method was rediscovered and expanded on by MacKay *et al.* [46, 47]. With improvements in the technology used to test and implement error correcting codes, the codes developed for use with these methods are comparable with the highly touted turbo codes in terms of bit error rate performance [48, 67].

There are several prominent algorithms for the decoding of convolutional codes. Foremost among them is the Viterbi decoding algorithm. Sequential decoding, with its many variations, is another popular method. The reader is referred to [43, 40] for an excellent treatment of these 'classical' topics. More recently, soft-decision decoding techniques have gained some attention since they can be use quite effectively in the decoding of turbo codes. Some of these include the BCJR algorithm [9] and various soft-output Viterbi algorithms (SOVA) [13, 28]. We shall discuss some new algebraic ideas for decoding convolutional codes in Chapter 6.

## 5.4 Majority Logic Decoding of Convolutional Codes

This section will provide a brief background on majority logic decoding of convolutional codes. (The underlying syndrome decoding technique can, of course, be used for block codes.) In particular, definite decoding, feedback decoding and threshold decoding will be discussed. For a more thorough investigation of these topics the reader is referred to [50, 55, 63, 43, 40].

The general idea of majority logic decoding is to use the syndrome former matrix of the convolutional code to obtain parity checks on the bits of the codeword. These parity checks are then used in somewhat different ways by the three basic types of majority logic decoding. First, let us focus on how the parity checks are obtained.

The syndrome former matrix (see Section 3.2) is created with $(2m+1)n$ columns. This matrix defines a set of equations on each window of length $(2m+1)n$ bits of the received sequence. A special time interval is selected from the window and a set of orthogonal parity checks is obtained from this set of equations on this time interval. Simply put, a set of parity checks is orthogonal on a bit $u_i$ if each check involves $u_i$ and no other bit is checked by more than one of the checks. For definite decoding, the 'middle' time interval of the parity parallelogram is selected, and a set of orthogonal parity checks is created for each of the $k$ information bits of that time interval. For feedback and threshold decoding, only the middle and right hand bits of the 'parallelogram' are used, thus resulting in the 'parity triangle'. We will soon see a more accurate description of this process using the representations of Section 3.5.

For the hard decision techniques, namely definite and feedback decoding, the decoding process is straightforward. The received bits are given a hard decision estimate based on the channel model and are fed into the majority logic decoder. That is, the bits are fed into the orthogonal parity check equations and each parity check is evaluated as a 0 or a 1. For each information bit, if a majority of its syndromes are 1, it is assumed to be in error and (assuming a binary code) its hard decision channel estimate is reversed. On the other hand, if a majority of its orthogonal parity checks are 0, it assumed to be correct and its hard decision channel estimate stands.

Now comes the main difference between definite and feedback decoding. In definite decoding, the decisions made by the majority logic decoder are not used to update the bits in future time intervals. In feedback decoding, the decoding made by the majority logic decoder is used in future computations. In particular, if the bit $u_i$ is determined to be in error, then all the values of the orthogonal parity checks are changed. In both methods, the sequence of syndromes is shifted by one time interval to prepare for the next set of incoming bits (rather than redundantly recalculate the previous syndromes).

For threshold decoding, the decoder does not immediately make a hard decision estimate. Therefore the syndromes will not evaluate to 0 or 1 and, hence, the decoder decision cannot be based simply on the majority of the orthogonal parity checks. In general, the procedure is to obtain a real valued function on the orthogonal parity checks and declare a bit to be in error if the function exceeds some 'threshold' value.

Let us review the basic ideas involved by deriving the parity parallelograms and triangles for the code in Example 3.5.2.

**Example 5.4.1** The 22 column syndrome former matrix, in which the parity parallelogram (See Remark 3.5.3.) is clearly visible, is:

$$\left[\begin{array}{ccccccccccc|ccccccccccc}
1 & 1 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\
0 & 1 & 1 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\
0 & 0 & 1 & 1 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\
0 & 0 & 0 & 1 & 1 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\
0 & 0 & 0 & 0 & 1 & 1 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\
0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1
\end{array}\right]$$

The syndrome equations derived are as follows:

$$\begin{bmatrix} s_\tau \\ s_{\tau+1} \\ s_{\tau+2} \\ s_{\tau+3} \\ s_{\tau+4} \\ s_{\tau+5} \end{bmatrix} = \begin{bmatrix} \mathbf{1} & \mathbf{1} & \mathbf{1} & 0 & 0 & \mathbf{1} & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 & 1 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & \mathbf{1} & \mathbf{1} & \mathbf{1} & 0 & 0 & \mathbf{1} & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 & 1 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & \mathbf{1} & \mathbf{1} & \mathbf{1} & 0 & 0 & \mathbf{1} \end{bmatrix} \begin{bmatrix} u_{\tau-5} \\ u_{\tau-4} \\ u_{\tau-3} \\ u_{\tau-2} \\ u_{\tau-2} \\ u_{\tau-1} \\ u_\tau \\ u_{\tau+1} \\ u_{\tau+2} \\ u_{\tau+3} \\ u_{\tau+4} \\ u_{\tau+5} \end{bmatrix} + \begin{bmatrix} y_\tau \\ y_{\tau+1} \\ y_{\tau+2} \\ y_{\tau+3} \\ y_{\tau+4} \\ y_{\tau+5} \end{bmatrix}$$

Here, the syndromes $s_\tau$, $s_{\tau+3}$ and $s_{\tau+5}$ form a set of orthogonal parity checks. Thus a majority logic decoder can correct a single error in any of the 13 bits that are checked by the orthogonal parity checks.

On the other hand, the parity triangle for feedback and threshold decoding is given by:

$$\begin{bmatrix} s_\tau \\ s_{\tau+1} \\ s_{\tau+2} \\ s_{\tau+3} \\ s_{\tau+4} \\ s_{\tau+5} \end{bmatrix} = \begin{bmatrix} \mathbf{1} & 0 & 0 & 0 & 0 & 0 \\ 0 & \mathit{1} & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ \mathbf{1} & 0 & 0 & \mathbf{1} & 0 & 0 \\ \mathbf{1} & \mathbf{1} & 0 & 0 & \mathbf{1} & 0 \\ \mathbf{1} & \mathit{1} & \mathbf{1} & 0 & 0 & \mathbf{1} \end{bmatrix} \begin{bmatrix} u_\tau \\ u_{\tau+1} \\ u_{\tau+2} \\ u_{\tau+3} \\ u_{\tau+4} \\ u_{\tau+5} \end{bmatrix} + \begin{bmatrix} y_\tau \\ y_{\tau+1} \\ y_{\tau+2} \\ y_{\tau+3} \\ y_{\tau+4} \\ y_{\tau+5} \end{bmatrix}$$

Here, the syndromes $s_\tau$, $s_{\tau+3}$, $s_{\tau+4}$ and the check formed as the sum $(s_{\tau+1} + s_{\tau+5})$ are a set of orthogonal parity checks. Thus a majority logic decoder can correct up to two errors in any of the 11 bits that are checked by the orthogonal parity checks.

## 5.5 Using the Local Description for Feedback Decoding

In the usual setting of convolutional codes the parity triangle is obtained directly from the syndrome former, as shown in the above examples. However, the local description offers valuable insight into the nature of feedback decoding. Instead of identifying the parity triangle as the right half of the parity parallelogram, it should rather be identified as the top portion of the matrix in (3.5.2). It is in this setting that relationship between the state of the encoder and the syndromes is fully revealed.

To be exact, we take the top $m+1$ rows of equation (3.5.2). The left hand side consists of the first $m$ rows of the observability matrix, followed by a row of zeros, times the state at time $\tau$. The right hand side consists of the parity triangle times the sequence of inputs plus the sequence of outputs.

**Theorem 5.5.1** *The updated syndromes for feedback decoding for times $\tau$ through $\tau+m$ are given by:*

$$\begin{bmatrix} s_\tau \\ s_{\tau+1} \\ \vdots \\ s_{\tau+m-1} \\ s_{\tau+m} \end{bmatrix} = \begin{bmatrix} C \\ CA \\ \vdots \\ CA^{m-1} \\ 0 \end{bmatrix} \hat{x}_\tau + \begin{bmatrix} D & 0 & \cdots & & 0 \\ CB & D & \cdots & & 0 \\ \vdots & \ddots & \ddots & & \vdots \\ CA^{m-2}B & \cdots & CB & D & 0 \\ X & X & X & X & D \end{bmatrix} \begin{bmatrix} \tilde{u}_\tau \\ \tilde{u}_{\tau+1} \\ \vdots \\ \tilde{u}_{\tau+m-1} \\ \tilde{u}_{\tau+m} \end{bmatrix} + \begin{bmatrix} \tilde{y}_\tau \\ \tilde{y}_{\tau+1} \\ \vdots \\ \tilde{y}_{\tau+m-1} \\ X \end{bmatrix}$$

*where the last row depends on the matrix $A$ as described in Proposition (3.5.1). (The $\tilde{u}$'s and $\tilde{y}$'s are the received bits. $\hat{x}_\tau$ is the state of 'encoder' on the receiving end based on the decoded input bits up to time $\tau - 1$.)*

*Proof:* The proof follows from equation (2.2.2).

$$\begin{bmatrix} C \\ CA \\ CA^2 \\ \vdots \\ CA^{m-1} \\ 0 \end{bmatrix} \hat{x}_\tau = \begin{bmatrix} C \\ CA \\ CA^2 \\ \vdots \\ CA^{m-1} \\ 0 \end{bmatrix} [A^{\tau-1}B \ \cdots \ AB \ B] \begin{bmatrix} \hat{u}_0 \\ \vdots \\ \hat{u}_{\tau-1} \end{bmatrix}$$

Where the $\hat{u}$'s are the decoded inputs from previous time intervals. Multiplying out the right hand side gives exactly the left half of the parity parallelogram. The result is now immediate. □

**Remark 5.5.2** The above theorem states that having no errors in the feedback at any given time is equivalent to knowing the correct state of the encoder at that time. Also, if the matrix $A$ is nilpotent, then only the most recently decoded input bits are required since the rest of the above matrix will be 0. In that case, we will get the usual parity parallelogram.

CHAPTER 6

# ALGEBRAIC DECODING OF CONVOLUTIONAL CODES USING THE LOCAL DESCRIPTION

In this chapter we will describe an algebraic decoding scheme for convolutional codes that was first put forth in [56]. Some example codes for this algorithm will be presented. We will proceed to make significant improvements in this algorithm and discuss more applications including the decoding of turbo codes.

## 6.1 A Basic Algorithm

We begin by restating the notion of nearest neighbor decoding for convolutional codes. This is then used to develop a basic algorithm for the algebraic decoding of convolutional codes.

Assume a code sequence $\{v_t\}_{t\geq 0} = \left\{\binom{y_t}{u_t}\right\}_{t\geq 0}$ was sent and the sequence

$$\{\hat{v}_t\}_{t\geq 0} = \left\{\begin{pmatrix}\hat{y}_t\\ \hat{u}_t\end{pmatrix}\right\}_{t\geq 0}$$

has been received. The decoding problem then asks for the minimization of the error

$$\text{error} := \min_{\{v_t\}\in\mathcal{C}} \sum_{t=0}^{\infty} \operatorname{dist}(v_t, \hat{v}_t) = \min\left(\sum_{t=0}^{\infty}[\operatorname{dist}(u_t,\hat{u}_t) + \operatorname{dist}(y_t,\hat{y}_t)]\right). \tag{6.1.1}$$

If no transmission error occurs then $\{\hat{v}_t\}_{t\geq 0}$ is a valid trajectory and the error value in (6.1.1) is zero. Let

$$\left\{\begin{pmatrix}f_t\\ e_t\end{pmatrix}\right\}_{t\geq 0} := \left\{\begin{pmatrix}\hat{y}_t - y_t\\ \hat{u}_t - u_t\end{pmatrix}\right\}_{t\geq 0} \tag{6.1.2}$$

be the sequence of errors.

We consider the received codeword sequence $\{v_t\}_{t\geq 0}$ as being divided into time intervals of size $T+1$. There is also a positive integer, $\Theta$, chosen for each code subject to the following considerations:

- $\Theta$ should be less than $T$. In particular, it is desirable for $\lfloor (T+1)/\Theta \rfloor$ to be greater than 1.
- We will want (the columns of) $\Omega_\Theta(A,C)$ to be the generator matrix of a block code with sufficiently good distance. The matrix $\Omega_\Theta(A,C)$ must be full rank, so $\Theta$ must be at least $\nu$, the observability index of $(A,C)$. It is easy to see that the distance of the code is a non-decreasing function of $\Theta$. However, a balance must be found, as we shall see, between the distance of this code and the desire to maximize $(T+1)/\Theta$.

**Algorithm:** Assume initially that

$$\hat{u}_{T-\Theta+1}, \hat{u}_{T-\Theta+2}, \ldots, \hat{u}_T \tag{6.1.3}$$

has been correctly transmitted. From Proposition 2.2.4 it follows that

$$\begin{pmatrix} y_{T-\Theta+1}\\ y_{T-\Theta+2}\\ \vdots\\ \vdots\\ y_T \end{pmatrix} - \begin{pmatrix} D & 0 & \cdots & & 0\\ CB & D & \ddots & & \vdots\\ CAB & CB & \ddots & \ddots & \\ \vdots & & \ddots & \ddots & 0\\ CA^{\Theta-2}B & CA^{\Theta-3}B & \cdots & CB & D \end{pmatrix} \begin{pmatrix} u_{T-\Theta+1}\\ u_{T-\Theta+2}\\ \vdots\\ \vdots\\ u_T \end{pmatrix}$$

is in the column space of the block code generated by the columns of

$$\begin{pmatrix} C \\ CA \\ \vdots \\ CA^{\Theta-1} \end{pmatrix}. \tag{6.1.4}$$

The larger the value of $\Theta$, the bigger, in general, the distance of this block code. Assume, now, that this block code has distance $d_{gen}$, and let $t_{gen} = \left\lfloor \frac{d_{gen}-1}{2} \right\rfloor$. Then, as long as there are $t_{gen}$ or fewer errors in the sequence

$$\hat{y}_{T-\Theta+1}, \hat{y}_{T-\Theta+2}, \ldots, \hat{y}_T$$

we can compute, using various standard decoding techniques for block codes, the error sequence

$$f_{T-\Theta+1}, f_{T-\Theta+2}, \ldots, f_T$$

as well as the state vector $x_{T-\Theta+1}$ from the local description:

$$\begin{bmatrix} \hat{y}_{T-\Theta+1} \\ \hat{y}_{T-\Theta+2} \\ \vdots \\ \vdots \\ \hat{y}_T \end{bmatrix} - \begin{bmatrix} D & 0 & \cdots & & 0 \\ CB & D & \ddots & & \vdots \\ CAB & CB & \ddots & \ddots & \\ \vdots & & \ddots & \ddots & 0 \\ CA^{\Theta-2}B & CA^{\Theta-3}B & \cdots & CB & D \end{bmatrix} \begin{bmatrix} u_{T-\Theta+1} \\ u_{T-\Theta+2} \\ \vdots \\ \vdots \\ u_T \end{bmatrix} = \begin{bmatrix} C \\ CA \\ \vdots \\ \vdots \\ CA^{\Theta-1} \end{bmatrix} x_{T-\Theta+1}. \tag{6.1.5}$$

If a decoding error is made here, *i.e.* if more than $t_{gen}$ errors occur in the above sequence of $\hat{y}_i$'s, we call it an error of TYPE I.

Since the state vector $x_{T-\Theta+1}$ is also equal to

$$x_{T-\Theta+1} = \begin{pmatrix} A^{T-\Theta}B & \ldots & B \end{pmatrix} \begin{pmatrix} u_0 \\ \vdots \\ u_{T-\Theta} \end{pmatrix}$$

it is possible to compute the error sequence $e_0, \ldots, e_{T-\Theta}$ from the syndrome vector

$$\begin{pmatrix} A^{T-\Theta}B & \ldots & B \end{pmatrix} \begin{pmatrix} \hat{u}_0 \\ \vdots \\ \hat{u}_{T-\Theta} \end{pmatrix} - x_{T-\Theta+1}. \tag{6.1.6}$$

We see now that the key to decoding this part of the code sequence is to have $(A^{T-\Theta}B \ \ldots \ B)$ be a parity check matrix for a code with good distance, or have some properties that make it easy to decode, *e.g.* low-density. In this way, one can either choose to decode algebraically with hard decision and require a good distance, or one can decode using Gallager-type algorithms or other soft-decision algorithms where structure is more important than distance. Let the distance of this parity check code be $d_{par}$, and let $t_{par} = \left\lfloor \frac{d_{par}-1}{2} \right\rfloor$. Hence, if we decode algebraically, we can correct up to $t_{par}$ errors in the sequence $\hat{u}_0, \ldots, \hat{u}_{T-\Theta}$. Decoding errors made in this stage are called errors of TYPE II. If desired, the error sequence $f_0, \ldots, f_{T-\Theta}$ can now be computed from the description of the code provided in Proposition 2.2.4, or in Proposition 2.2.5.

Let us now deal with the situation where the received sequence in (6.1.3) is not correct. Several things might happen then. First it is possible that we cannot compute the state vector $x_{T-\Theta+1}$ from identity (6.1.5) in which case we conclude that the sequence given in (6.1.3) was not correct. It is possible that a wrong state vector $x_{T-\Theta+1}$ is computed. This will lead to a wrong syndrome vector in (6.1.6) and the computed error sequence $e_0, \ldots, e_{T-\Theta}$ will then have weight more than $t_{par}$. At that point, we conclude that there must be an error in the sequence in (6.1.3). (The other possibility is that the state has been correctly estimated, but there are actually more than $t_{par}$ errors in $\hat{u}_0, \ldots, \hat{u}_{T-\Theta}$. In this case, the algorithm won't be able, at this stage, to decode correctly.) Finally, there might occur the situation where we compute (by chance) the correct state vector $x_{T-\Theta+1}$ from identity (6.1.5). In this situation we correctly will find the error sequence $e_0, \ldots, e_{T-\Theta}$ as well as the error sequence $f_0, \ldots, f_{T-\Theta}$ and in a later stage of the algorithm we will recognize that (6.1.3) was not correct. We will handle this case by saying only that it is 'improbable' and affects at most $\Theta$ inputs (if one is concerned about bit error rate).

Assume now that it has been recognized that (6.1.3) is not correct. In this case we replace the sequence (6.1.3) with the sequence

$$\hat{u}_{T-2\Theta+1}, \hat{u}_{T-2\Theta+2}, \ldots, \hat{u}_{T-\Theta} \tag{6.1.7}$$

and we assume that this is a correctly transmitted sequence. Again it might turn out that (6.1.7) is not a correctly transmitted sequence. One iteratively proceeds until one finds a correct sequence

$$\hat{u}_{T-h\Theta+1}, \hat{u}_{T-h\Theta+2}, \ldots, \hat{u}_{T-(h-1)\Theta}. \tag{6.1.8}$$

If we are unable to find such a correct sequence, then we say an error of TYPE III occurs. Such a sequence can be found with high probability, depending on the channel, and, more importantly, the values of $T$ and $\Theta$. This sequence will then allow one to correctly compute the state vector $x_{T-h\Theta+1}$ as well as the errors $\left\{ \binom{f_t}{e_t} \mid 0 \le t \le T - h\Theta \right\}$. After having computed $x_{T-h\Theta+1}$ we reiterate the algorithm by attempting to compute the state vector $x_{2T-(h+1)\Theta+1}$ making the assumption that the sequence

$$\hat{u}_{2T-(h+1)\Theta+1}, \hat{u}_{2T-(h+1)\Theta+2}, \ldots, \hat{u}_{2T-\Theta}. \tag{6.1.9}$$

has been transmitted correctly.

## 6.2 Some Classes of Binary Input-State-Output Convolutional Codes

We will present two ideas for how to select the matrices $(A, B, C, D)$ so as to construct codes with properties desirable for this decoding scheme. BCH type codes over larger fields have already been constructed using these ideas in [61, 62, 59, 71]. We will examine 'maximum distance separable' convolutional codes over larger fields in Chapter 7. Our construction here will focus on permutation matrices and matrices with large order over the binary field.

**Definition 6.2.1**

- Let $A$ be a nonsingular matrix with minimum polynomial $p_A(s)$. Then the *order*, $n$, of the matrix $A$ is equal to the order of the polynomial $p_A(s)$, which is the smallest integer, $n$, such that $p_A(s)$ divides $s^n - 1$. Equivalently, $n$ is the smallest integer such that $A^n = I$. See [41], for example.

- If $A$ is size $\delta \times \delta$, then $A$ is *primitive*, (as well as its minimum polynomial) if its order is the maximum, $q^\delta - 1$ (over $\mathbb{F}_q$).

We will restrict ourselves to convolutional codes of rate $\frac{1}{2}$, so our matrices will have sizes: $A = \delta \times \delta$, $B = \delta \times 1$, $C = 1 \times \delta$, and $D$ will be a $1 \times 1$ matrix (usually the identity).

Notice that we now have a bound on the size of $T - \Theta$. Since we require $\Phi_{T-\Theta+1}(A, B)$ to be a parity check matrix, if $T - \Theta \ge n$, the order of $A$, then the distance of that code will necessarily be 2, and hence will provide for no error correction.

This bound emphasizes the need to choose a matrix, $A$, with large order. At this point we can go in several directions. Obviously, if we wish to maximize order, we may choose $A$ as a companion matrix of a primitive polynomial. Notice that in this case, the choice of $B$ is irrelevant, so long as $B$ is not the zero vector. However, a matrix with high order doesn't necessarily correspond to good distance. In fact, the parity check subcode is generated by the minimum polynomial of the matrix, so the distance is immediately upper bounded by the weight of this polynomial (and will decrease as $T - \Theta$ increases). We will make this notion more precise.

There is a one-one correspondence between polynomials of degree $d$ and vectors of length $d+1$ given easily by: $a_0 + a_1 s + a_2 s^2 + \ldots + a_d s^d \leftrightarrow [a_0, a_1, a_2, \ldots, a_d]$. Every codeword (here, codeword refers to any vector $\mathbf{u}$, such that $[BAB \cdots A^{T-\Theta-1}B]\mathbf{u}' = \mathbf{0}$, *i.e.* a codeword of the parity check subcode) is a multiple of the minimum polynomial, under the above correspondence.

**Definition 6.2.2** The ideal of all polynomials generated by $p_A(s)$ is denoted as $(p_A(s))$. Let the intersection of this ideal with the set of all polynomials of degree less or equal to $d$ be denoted by $(p_A(s))_{<d}$.

Using these definitions we may reformulate the above discussion:

**Proposition 6.2.3** *For each value of $T-\Theta$, the set of codewords for the parity check subcode is precisely $(p_A(s))_{<(T-\Theta)}$. The distance of this code, is exactly the weight of the minimum weight non-zero polynomial in this set.*

Hence, the order of a matrix is not the sole determining factor in the quality of the code. Nevertheless, there are enough good codes that randomly selecting primitive polynomials with a high weight will shortly lead to a good parity check subcode. Since the generator subcode also depends heavily on $A$ and not so much on $C$, this code also tends to have good distance properties at the same time. An example of a code constructed using these ideas is found below in Example 6.3.2.

Although primitive matrices have, in general, good distance properties, the corresponding parity check subcode lacks any real structure. If we are willing to give up a little distance for some structure, we can quite possibly take advantage of some soft decision decoding techniques. Permutation matrices have the advantages of possessing excellent structure and having potentially large order. Most importantly, if we select a sparse column, with say, t ones, as our matrix $B$, then each column of $\Phi_{T-\Theta+1}(A,B)$ will have exactly t ones.

## 6.3 Example Codes.

**Example 6.3.1** Let us construct a simple example. Let us choose a permutation which is the direct sum of cycles of length 3, 5, 7, and 11. For simplicity we choose the cycles: (2 3 1)(5 6 7 8 4)(10 11 12 13 14 15 9)(17 18 19 20 21 22 23 24 25 26 16). The resulting matrix is $26 \times 26$, and has order $3 \times 5 \times 7 \times 11 = 1155$. We choose as our $B$ matrix, the vector with all zeros except a 1 in rows 1, 4, 9, and 16. The corresponding matrix $\Phi_{68}(A,B)$ has distance 5. We choose $T-\Theta$ to be 67. For a randomly chosen $C$ matrix with weight 13 the code generated by the columns of $\Omega_\Theta(A,C)$ has distance 3 when $\Theta = 34$. Hence, we may let $T = 101$, and $\Theta = 34$, so $(T+1)/\Theta = 3$; we will have three intervals to work with for estimating the state.

We now give error probabilities for this code assuming nearest neighbor decoding over a BSC for various values of the channel transition probability $p$.

TABLE 6.1: ERROR PROBABILITIES FOR THE CODE OF EXAMPLE 6.3.1

| p | TYPE I | TYPE II | TYPE III | Block Error |
|---|---|---|---|---|
| .03 | .2717 | .3343 | .2683 | .6453 |
| .01 | .0454 | .0310 | .0242 | .0974 |
| .001 | .0005 | .00007 | .000037 | .0006 |

**Example 6.3.2** We now present a simple example code based on a primitive matrix. Let $A$ be the companion matrix for the primitive polynomial $s^{20}+s^{19}+s^{18}+s^{17}+s^{14}+s^{11}+s^{10}+s^{9}+s^{8}+s^{7}+s^{4}+s^{2}+1$ over $\mathbb{F}_2$. Choose $B$ and $C$ as simply the first unit vector, and $D = 1$. It turns out that for $\Theta = 33$, the generator subcode has distance 5. Choosing $T = 65$ provides a parity check subcode with distance 5. The error probabilities for the code are presented in the following table:

We defer a rigorous analysis of the decoding properties of this decoding algorithm until further refinements are presented. This analysis appears in Section 6.7.

TABLE 6.2: ERROR PROBABILITIES FOR THE CODE OF EXAMPLE 6.3.2

| p | TYPE I | TYPE II | TYPE III | Block Error |
|---|---|---|---|---|
| .03 | .0756 | .3178 | .2549 | .5301 |
| .01 | .0044 | .0287 | .0225 | .0547 |
| .001 | .000005 | .000044 | .000034 | .00008 |

### 6.4 An Alternative to State Estimation

In this section we refine the previous algebraic decoding algorithm by altering our treatment of the state. The approach is based on making use of both the input-state-output and output-state-input representations. For simplicity we will limit the presentation to codes of rate proportional to 1/2. The adaptation of the method is easily done for rates $1/n$. Some effort is required to extend this method to general rates below 1/2 and there is no known extension to higher rates.

Let us recall the state representation given in (2.2.2) and the corresponding equation for the output-state-input representation. If we subtract these equations, we arrive at the following:

$$(A^{t-\tau} - \tilde{A}^{t-\tau})x_\tau = [\,A^{t-\tau-1}B \cdots B|\tilde{A}^{t-\tau-1}\tilde{B} \cdots \tilde{B}\,] \begin{bmatrix} -u_\tau \\ \vdots \\ -u_{t-1} \\ \hline y_\tau \\ \vdots \\ y_{t-1} \end{bmatrix} \tag{6.4.1}$$

This equation immediately suggests an iterative parity check decoding scheme: Assume the code has been correctly decoded up to time $\tau$. In particular, $x_\tau$ is known. Then (6.4.1) allows us to use parity check decoding on the next $t - \tau$ time intervals of the codeword. Correctly decoding this part of the codeword gives the state $x_t$ which enables us to repeat this process through the entire codeword.

The efficiency of this algorithm depends on how the parity check decoding is accomplished and on how the matrices $(A, B, C, D)$ are chosen. As before, there are two basic choices for how to accomplish parity check decoding. First, one can use a straightforward nearest neighbor decoding scheme based solely on the minimum distance of the code. This scheme will turn out to be most efficient if the matrix appearing in (6.4.1) is the parity check matrix for some good block code. Binary codes that satisfy this requirement are elusive to find. In fact, if $k = 1$ (hence $n = 2$) this parity check code must have distance 2. Only by increasing $k$ and choosing a suitably complex matrix $D$ can the distance be increased. For larger field sizes, some examples do exist (see [58, 61, 71]), however the algorithm is still limited by the elimination of information inherent in this process. These limitations will be described more fully in Section 6.5.

Alternatively, one can choose to implement a Gallager-type low-density soft-decision decoding algorithm such as in [27, 46]. Low-density decoding depends on having the parity check matrix with nearly uniform column and row densities. These techniques can overcome a lack of good distance properties. However, the basic strength of soft-decision techniques is that they make full use of all information available at the expense of complexity in order to decode. This refinement, as we shall see in Section 6.5, contradicts this idea by sacrificing information in favor of efficiency. Thus, this refinement is not well suited for soft-decision decoding methods.

### 6.5 Some Notes on the State Elimination Algorithm

We now offer some insight as to the effectiveness of the algorithm presented in Section 6.4.

In order to make full use of the error correcting capabilities of the convolutional codes with the iterative manner of the decoding schemes of this chapter, one must use all the information available when making decoding decisions. We will see that the algorithm presented in section 6.4 does not make full use of this information. That is, the decoding algorithm will not find the optimal solution. The algorithm is designed to trade off some error correcting capability for reduced complexity.

Let us make these remarks more precise. The full information of the convolutional code is presented in the global description of Proposition 2.2.5. The choice to use an iterative decoding method already limits the amount of information available to the decoder to the information provided by the local description of Proposition 2.2.4.

**Theorem 6.5.1** *The code defined by the parity check matrix of* (6.4.1) *is larger than the code defined by the parity check matrix of* (3.5.1)*. i.e.:*

$$\ker\left[\, M_\gamma(A,B,C,D) \quad | \quad I\,\right] \subset \ker\left[\, \Phi_\gamma(A,B) \quad | \quad \Phi_\gamma(\tilde{A},\tilde{B})\,\right]$$

*Proof:* This follows immediately by multiplying the left-hand matrix above by $\Phi_\gamma(\tilde{A},\tilde{B})$ and applying lemma 2.3.4. □

The theorem states that the proposed state elimination algorithm of Section 6.4 does not make full use of the information in the local description. Hence, the algorithm will not perform optimally for an iterative decoding scheme. However, the parity check matrix involved is considerably smaller and hence offers the possibility of providing a reduced complexity for decoding.

## 6.6 Enhanced State Estimation

In section 6.4, the issue of finding a correct state in the algebraic decoding algorithm was sidestepped by 'canceling' the state from the equations. However, the resulting parity check matrix was less than ideal for error correction. Our new approach is to increase the number of available 'windows' that can be used to estimate a correct state. The idea is to make direct use of Lemmas 2.3.3 and 2.3.4. Namely, by multiplying equation (6.1.5) by $M_\Theta(\tilde{A},\tilde{B},\tilde{C},\tilde{D})$ we arrive at the following:

$$\begin{bmatrix} \tilde{D} & 0 & \cdots & 0 \\ \tilde{C}\tilde{B} & \tilde{D} & \ddots & \vdots \\ \tilde{C}\tilde{A}\tilde{B} & \tilde{C}\tilde{B} & \ddots & \ddots & \\ \vdots & & \ddots & \ddots & 0 \\ \tilde{C}\tilde{A}^{\Theta-2}\tilde{B} & \tilde{C}\tilde{A}^{\Theta-3}\tilde{B} & \cdots & \tilde{C}\tilde{B} & \tilde{D} \end{bmatrix} \begin{bmatrix} y_{T-\Theta+1} \\ y_{T-\Theta+2} \\ \vdots \\ \vdots \\ y_T \end{bmatrix} - \begin{bmatrix} \hat{u}_{T-\Theta+1} \\ \hat{u}_{T-\Theta+2} \\ \vdots \\ \vdots \\ \hat{u}_T \end{bmatrix} = \begin{bmatrix} \tilde{C} \\ \tilde{C}\tilde{A} \\ \vdots \\ \vdots \\ \tilde{C}\tilde{A}^{\Theta-1} \end{bmatrix} x_{T-\Theta+1}. \tag{6.6.2}$$

From this equation it is clear that we may now examine any $\Theta$ outputs to estimate the state $x_{T-\Theta+1}$. Of course, when we do this, the matrix $\Omega_\Theta(\tilde{A},\tilde{C})$ will be used as the generator matrix for a linear block code, so it is desirable that this code have a large distance.

Taking this idea even further, it is possible to decode the first $T-\Theta+1$ time intervals using the received outputs. This follows from the output-state-input analog of (6.1.6), namely:

$$\left(\tilde{A}^{T-\Theta}\tilde{B}\ \ldots\ \tilde{B}\right)\begin{pmatrix} \hat{y}_0 \\ \vdots \\ \hat{y}_{T-\Theta} \end{pmatrix} - x_{T-\Theta+1}. \tag{6.6.3}$$

This idea depends on $\Phi(\tilde{A},\tilde{B})$ being the parity check matrix of a linear block code with good distance. Once the outputs are known, it is a simple task to produce the input (information) vector.

## 6.7 Analysis of the Enhanced Algorithm

Let us consider the decoding performance of the above algorithm given a rate $k/n$ convolutional code, $(A,B,C,D)$, with complexity $\delta$, transmitted over a q-ary symmetric channel with transition probability $p$.

Denote the minimum distances of the parity check codes given by $\Phi_{T-\Theta+1}(A,B)$ and $\Phi_{T-\Theta+1}(\tilde{A},\tilde{B})$ (for some fixed T and $\Theta$) by $d_{par}$ and $\tilde{d}_{par}$ respectively. Similarly, denote the distances of the block codes generated by $\Omega_\Theta(A,C)$ and $\Omega_\Theta(\tilde{A},\tilde{C})$ by $d_{gen}$ and $\tilde{d}_{gen}$ respectively. Finally, let $I = \lfloor (T+1)/\Theta \rfloor$, the number of distinct intervals from which to make an estimate of the state. The following lemma provides an upper bound on the probability that there are no $\Theta$ consecutive correctly received inputs or outputs from which to estimate the state (*i.e.* the probability of type III errors).

**Lemma 6.7.1**

$$\begin{aligned}
Pr(\textit{no error in } \Theta \textit{ inputs}) &= (1-p)^{k\Theta} \\
Pr(\textit{no error in } \Theta \textit{ outputs}) &= (1-p)^{(n-k)\Theta} \\
Pr(\textit{all } I \textit{ input intervals are corrupt}) &= [\,1 - (1-p)^{k\Theta}\,]^I \\
Pr(\textit{all } I \textit{ output intervals are corrupt}) &= [\,1 - (1-p)^{(n-k)\Theta}\,]^I \\
Pr(\textit{Type III error}) &\le [\,1 - (1-p)^{k\Theta}\,]^I\,[\,1 - (1-p)^{(n-k)\Theta}\,]^I
\end{aligned}$$

*For the special case where $k = n$:*

$$Pr(\textit{Type III error}) \le [\,1 - (1-p)^{k\Theta}\,]^{2I}$$

*Proof:* Follows from elementary properties of probability and binomial distributions. □

**Remark 6.7.2** The above lemma is only an upper bound on the probability of type III errors because we have restricted ourselves to investigate only distinct intervals in search of $\Theta$ consecutive correctly transmitted inputs or outputs. If the first interval is corrupt it is not necessary to 'slide' $\Theta$ time units. It is possible (although unlikely) that only the last time unit in the interval was corrupted and that it would suffice to slide only one unit before recomputing the state. This however would lead to an extreme rise in complexity. A more reasonable alternative would be to slide about $\Theta/2$ intervals. The code designer must make this determination in evaluating complexity limitations versus error correcting needs.

Now, assume that a Type III error has been avoided and we have found $\Theta$ consecutive correctly transmitted inputs or outputs. The following lemma gives the probability that a correct state estimate can not be obtained (*i.e.* a type I error occurs).

**Lemma 6.7.3** *If the correctly transmitted sequence is comprised of inputs:*

$$Pr(\textit{Type I error}) = 1 - \sum_{i=0}^{t_{gen}} \binom{k\Theta}{i} p^i (1-p)^{k\Theta - i}$$

*If the correctly transmitted sequence is comprised of outputs:*

$$Pr(\textit{Type I error}) = 1 - \sum_{i=0}^{\tilde{t}_{gen}} \binom{(n-k)\Theta}{i} p^i (1-p)^{(n-k)\Theta - i}$$

*where*

$$\begin{aligned}
t_{gen} &= \lfloor (d_{gen} - 1)/2 \rfloor \\
\tilde{t}_{gen} &= \lfloor (\tilde{d}_{gen} - 1)/2 \rfloor .
\end{aligned}$$

*Proof:* These statements are just the binomial probability that number of errors does not exceed the error correcting capabilities of the generator subcodes. With this condition, the state estimate can be obtained by from (6.1.5) or (6.6.2) respectively. □

**Remark 6.7.4** If the distance of the generator subcodes was not a factor, we would be able to significantly reduce the probability of Type III errors. We have only considered 'solving' (6.1.5) for all the inputs or all the outputs. In general, simple row operations would enable us to solve for either the input or output of each time unit. That would increase the number of possible 'windows' in each interval from 2 up to $2^{\Theta}$. Theoretically this would reduce the the probability of Type III errors to virtually 0. Besides the complexity issues associated with this approach, the corresponding generator subcodes would change with each arrangement. It would be an impossible task to design a code with all $2^{\Theta}$ of these codes possessing good distance properties. It is extraordinary that we are able to find codes with good generator subcode distance properties for just our two chosen arrangements. Such codes will be presented in Section 7.1.

Similarly we can compute the probability of Type II errors.

**Lemma 6.7.5**

$$Pr(\textit{Type II error}) \quad = \quad 1 \; - \; \sum_{i=0}^{t_{par}} \binom{T - \Theta + 1}{i} p^i (1-p)^{T-\Theta+1-i}$$

*where* $t_{par} = \lfloor (d_{par} - 1)/2 \rfloor$.

**Remark 6.7.6** In theory, it would be possible to use both the input and output sequences to avoid Type II errors. That is, if the received input sequence has too many errors then one could try to decode using the received output sequence using (6.6.3). The problem is that in many cases, it is not known when a decoding error occurs. Of course, there are many equations which we can check our decoded inputs with. This would be done at the expense of complexity. A simpler alternative might be to decode on both sides, *i.e.* use (6.1.6) and (6.6.3) simultaneously and compare the results. This would help affirm a correct decoding, but there would be no obvious 'tie-breaker' if the two decoded sequences disagreed.

We now have the following obvious upper bound on the probability of decoding error in the first block. Furthermore, if we assume that we have the correct state at time $\tau$, then the same bound holds for the block beginning with $\tau$. (If the state $x_\tau$ is incorrect, we may obtain it in the same way we obtain the state above. *i.e.* The error propagation effect inherent in this incremental decoding scheme can be limited by the use of our state estimation algorithm.)

**Theorem 6.7.7** *The probability of block error in the enhanced decoding algorithm is upper bounded by*

*1 - [1 - Pr(type I)][1 - Pr(type II)][1 - Pr(type III)].*

Examples of codes suitable for this decoding scheme will be presented in Section 7.1.

### 6.8 An Algebraic Look at Decoding Turbo Codes

The basic idea in turbo decoding is to have two separate decoders working together to iteratively decode the codeword. The first decoder receives the first output stream, $y$, and makes a soft decision estimate for each bit using any of a number of proposed schemes for decoding a RSC. This estimate is passed along to the second decoder which also receives the other output stream, $\hat{y}$. Using both streams of data, the second encoder makes an updated estimate of the codeword. This data is then sent back to the original decoder, which tries to improve its estimate based on this new data. This swapping of information between encoders is repeated until a 'reliable' decoding decision is obtained.

Again, there is a choice as to which soft-decision decoding algorithm to use in the above process. The original work done in this area used a modified version of the BCJR decoding algorithm [7]. More recently the focus has shifted to a soft output Viterbi algorithm (SOVA) [28, 13]. Another approach is offered in [31]. Regardless of which approach is used, the basic principle is the same. Decoding is based on the fact that several parity check matrices are available to us. The first two are immediate from the input-state-output representation and the fact that we require each valid

$N$-block to begin and end with both encoders in the all zero state. The last two follow from the output-state-input representation (See Proposition 2.3.1).

$$\begin{aligned}
\Phi_N(A,B)\,u &= 0 && (6.8.1)\\
\Phi_N(A,B)\,S\,u &= 0 && (6.8.2)\\
\Phi_N(\tilde{A},\tilde{B})\,y &= 0 && (6.8.3)\\
\Phi_N(\tilde{A},\tilde{B})\,\hat{y} &= 0 && (6.8.4)
\end{aligned}$$

Again, the key idea, to maximize the decoding potential of the turbo code, is the sharing of the soft decision estimates between the decoders

**Remark 6.8.1** Nowhere is the choice between hard-decision and soft-decision decoding so clear as here. We can see that if $N$ is greater than the order of the matrix $A$ (which is less than $2^\delta$) then the parity check matrices must have distance 2 and offer no hard-decision decoding ability. On the other hand, if $N$ is less than or equal to the order of $A$ then the only possible choice for $S$, with the requirements we have imposed, is the identity matrix and our turbo code becomes little more than a repetition code composed with an RSC. Thus, soft-decision decoding is the only viable choice.

CHAPTER 7

# MDS CONVOLUTIONAL CODES

In this chapter the class of MDS convolutional codes given by a Reed-Solomon type construction will be examined. In particular, it will be shown that this class of codes is extremely well suited to the algebraic decoding techniques of Section 6.6. Then we will investigate the column distance function for this class of codes and compare them with theoretical limits.

## 7.1 Reed-Solomon Convolutional Codes

The natural question that arises from the description of the enhanced algebraic decoding algorithm of Section 6.6 is whether there exist codes which have good distance properties for each of the subcodes. Ideally, these subcodes would also possess some other properties which would aid in the decoding process. The answer to this question is yes, if one considers fields larger than $\mathbb{F}_2$. Consider the following code construction which was presented in [65]; similar codes were presented in [34].

We will only consider rate 1/2. The extension to rate $1/n$ is simple. The existence of these codes at higher rates is the content of [60]. Let us fix a complexity $\delta$. Choose $\mathbb{F}_q$ so that $q \geq 3\delta - 1$. Find a primitive element, $\alpha$, of $\mathbb{F}_q$. Define the following matrices:

$$A = \begin{bmatrix} \alpha & 0 & \cdots & 0 \\ 0 & \alpha^2 & & 0 \\ \vdots & & \ddots & \vdots \\ 0 & 0 & \cdots & \alpha^\delta \end{bmatrix}, \quad B = \begin{bmatrix} 1 \\ 1 \\ \vdots \\ 1 \end{bmatrix}, \quad D = [1] \text{ , and}$$

$$A' = \begin{bmatrix} \alpha^{\delta+1} & 0 & \cdots & 0 \\ 0 & \alpha^{\delta+2} & & 0 \\ \vdots & & \ddots & \vdots \\ 0 & 0 & \cdots & \alpha^{2\delta} \end{bmatrix}$$

The definition of the matrix $C$ requires some computation. We would like for $\tilde{A} = A - BC$ to be similar to $A'$. This can be done by computing the characteristic polynomial of $A - BC$ (with the entries of C denoted by variables), setting this polynomial equal to the polynomial of $A'$ and solving for the entries of C.

**Definition 7.1.1** The codes defined above are called *Reed-Solomon convolutional codes.*

**Theorem 7.1.2 (MDS Property of Reed-Solomon Codes)**

1. *The parity check subcodes defined by $\Phi_{T-\Theta+1}(A, B)$ and $\Phi_{T-\Theta+1}(\tilde{A}, \tilde{B})$ of Reed-Solomon convolutional codes are maximum distance separable (MDS) codes and therefore have minimum distance $\delta + 1$ for any $T - \Theta$ such that $\delta \leq T - \Theta \leq q - 2$.*

2. *The generator subcodes defined by $\Omega_\Theta(A, C)$ and $\Omega_\Theta(\tilde{A}, \tilde{C})$ of Reed-Solomon convolutional codes are MDS block codes and therefore have minimum distance $\Theta - \delta + 1$ for any $\Theta$ such that $\delta \leq \Theta \leq q - 2$.*

*Proof:*

1. The matrix $\Phi_{T-\Theta+1}(A, B)$ is a Vandermonde matrix and hence every full-size minor is nonzero. Therefore, any vector in the kernel of this matrix (*i.e.* a codeword) must have weight at least $\delta + 1$. On the other hand, since $(\tilde{A}, \tilde{B}, \tilde{C}, \tilde{D})$ also represents the same code and has the same (minimal) complexity, it must be true that $(\tilde{A}, \tilde{B})$ form a controllable pair. Since $\tilde{B} = B$ and $\tilde{A} = A - BC = S^{-1}A'S$ for some invertible scalar matrix $S$, we conclude that $(A', SB)$ form a controllable pair as well. It follows that $SB$ has no zero entries. From this we conclude that $\Phi_{T-\Theta+1}(A', SB)$ is a Vandermonde matrix. The result follows by noting that $\Phi_{T-\Theta+1}(A', SB) = S\Phi_{T-\Theta+1}(\tilde{A}, \tilde{B})$.

2. The result for the observability matrices will follow in the same line of reasoning as above once we can establish that the matrix $C$ contains no zero entries. This is equivalent to saying that the pair $(A, C)$, and hence the convolutional code, is observable. This can be seen as follows. The corresponding polynomial generator matrix representation of these codes is simply $[\, p_A(s) \;\; p_{A'}(s) \,]$, the characteristic polynomials of the matrices $A$ and $A'$. Since these polynomials have distinct roots ($\{\alpha, \alpha^2, \ldots, \alpha^\delta\}$ and $\{\alpha^{\delta+1}, \alpha^{\delta+2}, \ldots, \alpha^{2\delta}\}$ respectively) the gcd of the full-size minors is trivial and hence the code is observable.

□

**Remark 7.1.3** Not only do these codes posess excellent subcode distance properties, but the subcodes, being BCH codes (usually not technically Reed-Solomon) may be decoded efficiently via the Berlekamp-Massey algorithm.

Table 7.1 presents the probability of block error for many values of $\delta$, $q$, $\Theta$, $T$ and transition probability $p$ when the enhanced decoding algorithm of Section 6.6 is used on a $q$-ary symmetric channel. Figure 7.1 graphs a few of these codes over a wide range of transition probabilities. The two charts show that, in general, there is not one code in this class that is optimal for all channel transition probabilities. In particular, it is apparent that lower complexity codes perform better on noisier channels, while higher complexity codes achieve better error rates on better quality channels.

TABLE 7.1: ERROR PROBABILITIES FOR VARIOUS REED-SOLOMON CONVOLUTIONAL CODES USING THE ENHANCED DECODING ALGORITHM

| Code Parameters | | | | Probabilities of Block Error | | | |
|---|---|---|---|---|---|---|---|
| $q$ | $\delta$ | $\Theta$ | $T+1$ | p=.03 | p=.01 | p=.005 | p=.001 |
| 7 | 2 | 4 | 8 | .0105 | .0012 | .0003 | 1.2e-5 |
| 11 | 3 | 5 | 10 | .0173 | .0019 | .0005 | 2.0e-5 |
| 13 | 4 | 6 | 12 | .0137 | .0015 | .0004 | 1.5e-5 |
| 13 | 4 | 8 | 16 | .0049 | .0001 | 1.6e-5 | 1.2e-7 |
| 16 | 5 | 7 | 14 | .0193 | .0021 | .0005 | 2.1e-5 |
| 16 | 5 | 9 | 18 | .0072 | .0002 | 2.4e-5 | 1.7e-5 |
| 32 | 10 | 12 | 24 | .0570 | .0063 | .0016 | 6.6e-5 |
| 32 | 10 | 14 | 42 | .0096 | .0003 | 4.4e-5 | 3.6e-7 |
| 32 | 10 | 16 | 32 | .0232 | .0005 | 3.6e-5 | 6.5e-8 |
| 47 | 15 | 19 | 57 | .0253 | .0008 | .0001 | 9.6e-7 |
| 47 | 15 | 21 | 63 | .0143 | .0001 | 4.5e-6 | 6.0e-9 |
| 47 | 15 | 23 | 69 | .0169 | 8.0e-5 | 1.8e-6 | 1.7e-10 |
| 64 | 21 | 27 | 81 | .0389 | .0003 | 1.4e-5 | 1.8e-8 |
| 64 | 21 | 29 | 87 | .0423 | .0003 | 6.5e-6 | 6.6e-10 |
| 128 | 42 | 50 | 150 | .2415 | .0039 | .0001 | 1.6e-8 |

### 7.2 Further Properties of Reed-Solomon Convolutional Codes

We continue our exploration of Reed-Solomon convolutional codes by examining another distance parameter of these codes, namely the column distance function. For background on this distance parameter, please see [43, 30].

**Definition 7.2.1** For any convolutional code, let $\mathbf{v}$ and $\tilde{\mathbf{v}}$ be any codewords. Also, for any codeword, let

$$[\mathbf{v}]_\gamma \;=\; (v_0, v_1, v_2, \ldots, v_\gamma)$$

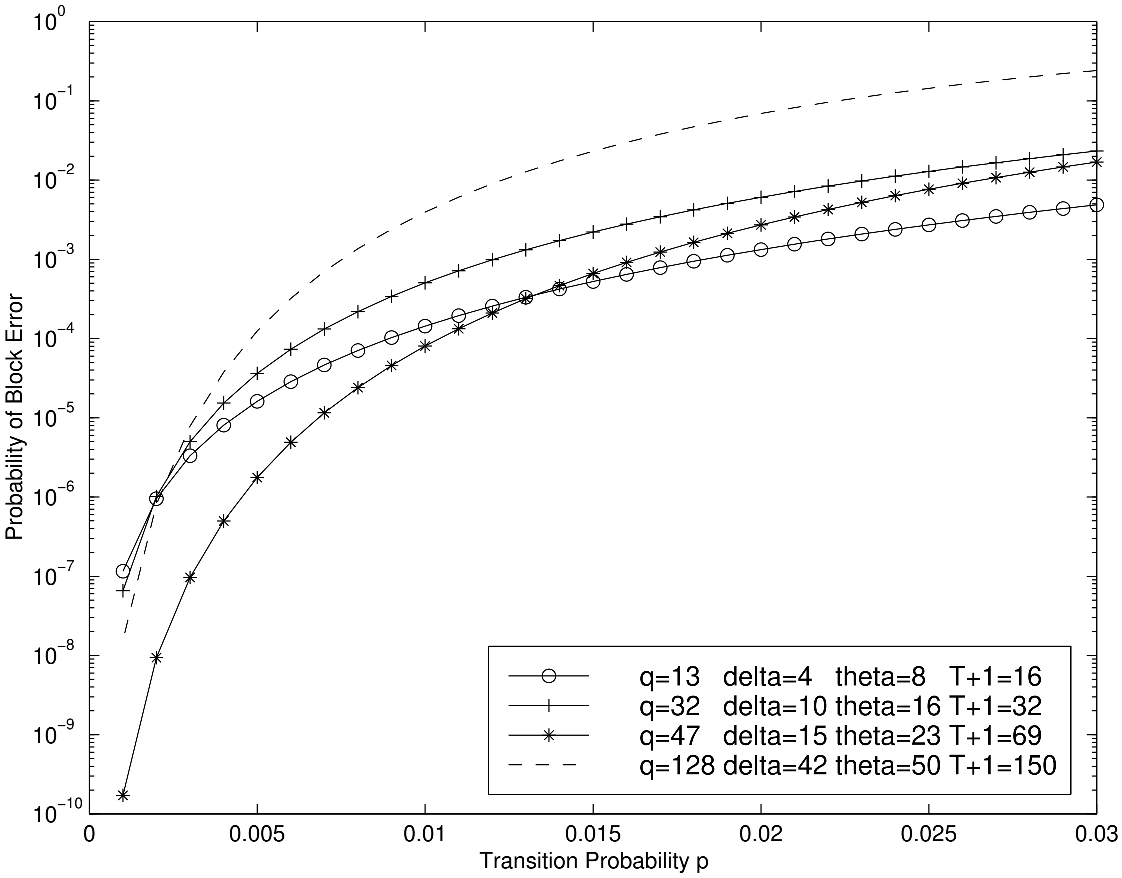


FIGURE 7.1: Plot of block error probabilities for selected Reed-Solomon convolutional codes.

denote the $\gamma^{th}$ truncation. Then the *column distance function* (CDF) of order $\gamma$ is denoted by $d_\gamma$ and defined as

$$\begin{aligned} d_\gamma &= \min\{\mathrm{dist}\,(\,[\mathbf{v}]_\gamma, [\tilde{\mathbf{v}}]_\gamma\,) \;:\; [\mathbf{v}]_0 \neq [\tilde{\mathbf{v}}]_0\} \\ &= \min\{\mathrm{wt}\,([\mathbf{v}]_\gamma) \;:\; [\mathbf{v}]_0 \neq 0\}. \end{aligned}$$

For the special case of when $\gamma = m$ (*i.e.* over one constraint length), $d_m$ is called the *minimum distance*, $d_{min}$, of the convolutional code.

It is clear that the CDF is a nondecreasing function of $\gamma$ and that for all $\gamma$ it must be that $d_\gamma \leq d_{free}$. This distance parameter is important for many decoding schemes including sequential decoding and majority logic decoding.

We now state the conjecture which we will discuss for the remainder of this chapter.

**Conjecture 7.2.2** For a rate 1/2 Reed-Solomon convolutional code with complexity $\delta$, and for $\delta - 1 \leq \gamma \leq 3\delta - 1$, then

$$d_\gamma \;\geq\; \gamma - \delta + 3$$

In particular, for $\gamma = 3\delta - 1$, we have $d_{3\delta-1} = 2\delta + 2$.

**Remark 7.2.3** It was shown in [65] that the free distance of these codes is $2\delta + 2$. So we can restate the above conjecture in these terms: If at any time, $\tau$, two codewords are in the same state, $x_\tau$, and do not agree at time $\tau + 1$, then over the interval $(\tau, \tau + 1, \ldots, \tau + 3\delta - 1)$ the codewords must differ by the full free distance, $2\delta + 2$, of the code. This would be an important result since the full 'decoding power' of the code would be contained in any interval of length $3\delta$.

There is a wealth of empirical evidence to support this conjecture, yet neither a proof nor a counterexample has been discovered. We offer several example codes, for small $\delta$, which satisfy the properties of the conjecture.

**Example 7.2.4** For $\delta = 1$ and $\mathbb{F}_4$, let $\alpha$ be a root of $s^2 + s + 1$ over $\mathbb{F}_4$. Define the Reed-Solomon convolutional code using the following matrices:

$$A = [\alpha], \ A' = [\alpha + 1], \ B = [1], \ C = [1], \ D = [1]$$

One can easily verify by hand that this code has $d_2 = 4$.

**Example 7.2.5** For $\delta = 2$ and $\mathbb{F}_7$, choose $\alpha = 3$ for the primitive element. Then the defining matrices of our code are given by:

$$A = \begin{bmatrix} 3 & 0 \\ 0 & 2 \end{bmatrix}, \ A' = \begin{bmatrix} 6 & 0 \\ 0 & 4 \end{bmatrix}, \ B = \begin{bmatrix} 1 \\ 1 \end{bmatrix}, \ C = [\,3 \ \ 6\,], \ D = [1]$$

Again, an easy calculation verifies that this code has $d_5 = 6$.

**Example 7.2.6** For $\delta = 3$ and $\mathbb{F}_{11}$, choose $\alpha = 2$ as the primitive element. Then we may define our code using the following matrices:

$$A = \begin{bmatrix} 2 & 0 & 0 \\ 0 & 4 & 0 \\ 0 & 0 & 8 \end{bmatrix}, \ A' = \begin{bmatrix} 5 & 0 & 0 \\ 0 & 10 & 0 \\ 0 & 0 & 9 \end{bmatrix}, \ B = \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix},$$

$$C = [\,8 \ \ 1 \ \ 3\,], \ D = [1]$$

A short computer verification (although possible to do by hand) reveals that this code has $d_8 = 8$.

**Example 7.2.7** For $\delta = 4$ and $\mathbb{F}_{13}$, choose $\alpha = 7$. Then the following matrices define our code:

$$A = \begin{bmatrix} 7 & 0 & 0 & 0 \\ 0 & 10 & 0 & 0 \\ 0 & 0 & 5 & 0 \\ 0 & 0 & 0 & 9 \end{bmatrix}, \ A' = \begin{bmatrix} 11 & 0 & 0 & 0 \\ 0 & 12 & 0 & 0 \\ 0 & 0 & 6 & 0 \\ 0 & 0 & 0 & 3 \end{bmatrix}, \ B = \begin{bmatrix} 1 \\ 1 \\ 1 \\ 1 \end{bmatrix},$$

$$C = [\,11 \ \ 2 \ \ 6 \ \ 6\,], \ D = [1]$$

Again, a computer verification yields that $d_{11} = 10$.

**Remark 7.2.8** It should be noted that for each of the example codes above, the full conjecture holds true, not just for when $\gamma = 3\delta - 1$. In fact, for smaller $\gamma$ the bound is not tight. In particular, for the example when $\delta = 4$, it can be shown that $d_3 = 5$ while the bound of the conjecture is merely $d_3 \geq 2$. A better conjecture for smaller $\gamma$ will follow.

### 7.3 Some Insights into the Conjecture

We will analyze the nature of these codes in order to better understand why the conjecture should hold. What we learn will enable us to handle the conjecture for small values of $\delta$.

**Remark 7.3.1** Let us make some observations which will simplify our analysis. First, we assume without loss of generality that the first input, $u_0$, is 1 (hence $y_0 = 1$). If the conjecture is false then there is a partial code sequence $\{\binom{y_0}{u_0}, \binom{y_1}{u_1}, \ldots, \binom{y_{3\delta-1}}{u_{3\delta-1}}\}$ with total weight less than $2\delta + 2$. This implies either the input sequence or the output sequence has weight less than $\delta + 1$. Since these codes have symmetric distance properties for the input-state-output and output-state-input representations, we can assume without loss of generality that the input sequence has weight less than $\delta + 1$. In particular this implies that any counterexample cannot be a complete codeword since at least $\delta + 1$ nonzero inputs are required to bring the underlying linear system into the all zero state once it has left (since $\Phi(A, B)$ has distance $\delta + 1$).

Consider the special case when only the first $w$ inputs are allowed to be nonzero and all other inputs are zero. Then the input is $1, u_1, u_2, \ldots, u_{w-1}, 0, \ldots, 0$, and the local description of the code becomes:

$$\begin{bmatrix} y_0 \\ y_1 \\ \vdots \\ y_{w-1} \\ \hline y_w \\ y_{w+1} \\ \vdots \\ y_\gamma \end{bmatrix} = \begin{bmatrix} D & 0 & \cdots & 0 \\ CB & D & \ddots & \vdots \\ \vdots & \ddots & \ddots & 0 \\ CA^{w-2}B & \cdots & CB & D \\ \hline CA^{w-1}B & \cdots & CA^2B & CB \\ CA^wB & \cdots & CA^3B & CAB \\ \vdots & & \vdots & \vdots \\ CA^{\gamma-1}B & \cdots & CA^{\gamma-w+1}B & CA^{\gamma-w} \end{bmatrix} \begin{bmatrix} 1 \\ u_1 \\ \vdots \\ u_{w-1} \end{bmatrix}$$

We will obtain a lower bound on the weights of the the last $\gamma - w + 1$ outputs $y_w, \ldots, y_\gamma$ by rearranging the bottom half of the matrix on the right side of the above equation as follows:

$$\begin{bmatrix} y_w \\ y_{w+1} \\ \vdots \\ y_\gamma \end{bmatrix} = \begin{bmatrix} B^T \\ B^TA \\ \vdots \\ B^TA^{\gamma-w} \end{bmatrix} \begin{bmatrix} c_1 & & \\ & \ddots & \\ & & c_\delta \end{bmatrix} \begin{bmatrix} A^{w-1}B & \cdots & AB & B \end{bmatrix} \begin{bmatrix} 1 \\ u_1 \\ \vdots \\ u_{w-1} \end{bmatrix}$$

This equation says that the last $\gamma - w + 1$ outputs form a codeword of a code generated by a Vandermonde matrix. This code has distance $\gamma - \delta - w + 2$. Hence, if the input to this matrix is not the all zero vector then the last $\gamma - w + 1$ outputs must have weight at least $\gamma - \delta - w + 2$.

Let us analyze the input to this matrix to ensure that the all zero vector is not the input. First, since all the $c_i$ are non-zero, the second matrix on the right side in the above equation is nonsingular. All that remains is to show that

$$[\, A^{w-1}B \cdots AB\ B \,]\, [\, 1\ u_1\ \cdots\ u_{w-1} \,]'$$

is nonzero. However, from (2.2.2), we know that this expression is exactly the state at time $w$, $x_w$. Remark 7.3.1 specifically excludes the possibility that we re-enter the all zero state. Hence, the input into the Vandermonde matrix is not the all zero vector, and we have proven the following lemma.

**Lemma 7.3.2** *Let $\delta \le \gamma \le 3\delta - 1$ and $w \le \gamma - \delta$, then when the input is of the form $1, u_1, u_2, \ldots, u_{w-1}, 0, \ldots, 0$ then the outputs $y_w, y_{w+1}, \ldots, y_{\gamma-1}$ have weight at least $\gamma - \delta - w + 2$.*

For the special case when the first $w$ inputs are nonzero, the conjectured property must hold. This is true since the inputs have weight $w$ and the outputs have weight at least $\gamma - \delta - w + 3$ (since $y_0 \neq 0$). This lemma will be a valuable tool for use in analyzing the conjecture.

Let us consider an alternate approach based on induction on $\gamma$. This approach will further clarify some of the properties of these codes and will serve to produce a conjecture for the exact column distance function of these codes, rather than the previously stated lower bound conjecture.

There are $2\gamma + 2$ bits in each partial codeword we are considering. The conjecture is that at least $\gamma - \delta + 3$ of these are nonzero. We know that $u_0$ and $y_0$ are nonzero, so $\gamma - \delta + 1$ of the remaining $2\gamma$ bits must be nonzero for the conjecture to hold. This means that at most $\gamma + \delta - 1$ of these bits are zero. In this situation there are at least $\delta - 1$ time units in which both the input and output bits are zero. Let us consider a time unit, $\tau$, where this is the case. From the input-state-output representation, it follows that $Cx_\tau = 0$, hence $x_\tau \in \ker C$.

If there exists a counterexample to the conjecture, then there must be at least $\delta$ time units in which the input and output bits are zero. This corresponds to $\delta$ states in the the kernel of $C$. Since, $\ker C$ has dimension $\delta - 1$, it may be possible to prove the conjecture by proving some sort of linear independence property of the states.

Although the above argument has not led to a proof of the conjecture, it has led us to a plausible conjecture for the actual column distance function of these codes. We begin with a definition.

**Definition 7.3.3** For any given input sequence $u$ of length $\gamma+1$ (with $u_0 \neq 0$), denote by $K_{\gamma,u}$ the number of time units in which both the input and output bits are zero. Denote by $D_{\gamma,u}$ the number of time units in which both the input and output bits are nonzero.

The following proposition is clear.

**Proposition 7.3.4** *The weight of the partial codeword corresponding to the input sequence $u$ is given by $\gamma+1+D_{\gamma,u}-K\gamma,u$. The minimum such weight over all appropriate input sequences is $d_\gamma$.*

**Definition 7.3.5** Consider all input sequences of the form $(1, u_1, u_2, \ldots, u_\gamma)$, and the corresponding sequence of states in the partial codeword each input sequence generates. For each such sequence, consider the dimension of the space spanned by the states that are in $\ker C$. Denote by $N_{\gamma,w}$ the maximum such dimension over all the input sequences of weight $w$.

**Conjecture 7.3.6**

$$N_{\gamma,w} = \max_{\{u\,:\,\mathrm{wt}\,(u)=w\}} (K_{\gamma,u} - D_{\gamma,u} + 1)$$

**Remark 7.3.7** The above conjecture, when combined with Proposition 7.3.4, states that $d_\gamma$ is dependent on the number of states in a code sequence that are in $\ker C$. That fact is trivial. What the conjecture intends to point out is that it requires nonzero inputs to drive the system into a state which facilitates low output, *i.e.* you cannot reduce output weight without increasing input weight and conversely. Since $N_{\gamma,w}$ is at most $\delta-1$, we see that Conjecture 7.3.6 combined with Proposition 7.3.4 reduces to Conjecture 7.2.2. Computation of the exact values of $N_{\gamma,w}$, as difficult as that may be, leads to the following conjectured exact value for $d_\gamma$.

**Conjecture 7.3.8** Define:

$$N_\gamma = \max_{1\leq w\leq\delta} N_{\gamma,w}$$

Then Proposition 7.3.4 and Conjecture 7.3.6 combine to state:

$$d_\gamma = \gamma + 2 - N_\gamma$$

## 7.4 The Case when $\delta = 2$

We will give a thorough examination of the case when $\delta = 2$. (The case when $\delta = 1$, is completely trivial.) Even for this small case, there is no known proof of Conjecture 7.2.2. However, it can be easily shown that $d_5 \geq 5$, which is just as good from a hard decision error correction point of view.

**Proposition 7.4.1** *For Reed-Solomon convolutional codes with $\delta = 2$ and $1 \leq \gamma \leq 3$, we have*

$$d_\gamma \geq \gamma - \delta + 3 = \gamma + 1$$

*Proof:* For $\gamma = 1$, the statement is trivial since $u_0 \neq 0$ and $y_0 \neq 0$. For the larger $\gamma$ it suffices to consider input sequences of weight 1 in light of Remark 7.3.1. Thus, $u_0 = 1$ and all other inputs are 0. We also have that $y_0 = Du_0 \neq 0$. The remaining outputs are given by:

$$\begin{aligned} y_1 &= CB \\ y_2 &= CAB \\ y_3 &= CA^2B \end{aligned}$$

The fact that at least two of these must be nonzero is a direct consequence of Lemma 7.3.2, or more directly, a consequence of the fact that the observability and controllability matrices are MDS. □

Let us consider the case when $\gamma = 4$. We wish to show that $d_4 \geq 5$. Here it suffices to consider input sequences of weight at most 2. An argument identical to that in the above proposition proves that any weight 1 input sequence must result in an output sequence of weight at least 4. So we need only consider the case when the input sequence has weight exactly 2. This means we let $u_0 = 1$ and $u_i \neq 0$ for exactly one $i$ such that $1 \leq i \leq 4$.

The case when $u_1 \neq 0$ is easily handled by Lemma 7.3.2. The case when $u_4 \neq 0$ is proven by combining the result of Proposition 7.4.1 for $\gamma = 3$ with the fact that $u_4 \neq 0$ to conclude $d_4 \geq 5$.

Now consider the case when $u_2 \neq 0$. We must show that at least 2 of $\{y_1, y_2, y_3, y_4\}$ are nonzero. These outputs are given by:

$$\begin{aligned} y_1 &= CB \\ y_2 &= CAB + u_2 \\ y_3 &= CA^2B + CBu_2 \\ y_4 &= CA^3B + CABu_2 \end{aligned}$$

We know that at most one of $\{CB, CAB, CA^2B, CA^3B\}$ can be zero. We will treat each of these possibilities separately starting with the case when all are nonzero. In this situation $y_1 \neq 0$ and at least one of $\{y_3, y_4\}$ must be nonzero because of MDS arguments similar to those of Lemma 7.3.2 and the desired result is obtained. When $CA^3B = 0$ it must be that $y_1 \neq 0$ and $y_4 \neq 0$. When $CA^2B = 0$ it must be that $y_1 \neq 0$ and $y_3 \neq 0$. Also, when $CAB = 0$ it is true that $y_1 \neq 0$ and $y_2 \neq 0$. The only case that remains is when $CB = 0$. If this is the case then $y_3 \neq 0$. Then either $y_2$ or $y_4$ must be nonzero unless $CA^3B - CABCAB = 0$. However, we also have

$$\begin{aligned} \tilde{C}\tilde{A}^3\tilde{B} &= -CA^3B + 2CA^2BCB + (CAB)^2 - 3CAB(CB)^2 + (CB)^4 \\ & \phantom{=} \text{Taking into account that } CB = 0 \text{ gives} \\ \tilde{C}\tilde{A}^3\tilde{B} &= CABCAB - CA^3B \end{aligned}$$

Hence, $CA^3B - CABCAB = 0$ implies that $\tilde{C}\tilde{A}^3\tilde{B} = 0$ which is impossible since $\tilde{C}\tilde{B} = -CB = 0$. This finishes the case for when $u_2 \neq 0$.

The last case we need to consider is that when $u_3 \neq 0$. Again we must show that at least 2 of $\{y_1, y_2, y_3, y_4\}$ are nonzero. These outputs are given by:

$$\begin{aligned} y_1 &= CB \\ y_2 &= CAB \\ y_3 &= CA^2B + u_3 \\ y_4 &= CA^3B + CBu_3 \end{aligned}$$

The cases when $CB = 0$, $CA^2B = 0$, $CA^3B = 0$ and when none are zero are easily treated in a fashion similar to the case when $u_2 \neq 0$. The only remaining case is when $CAB = 0$. Here, $y_1 \neq 0$ and either $y_3$ or $y_4$ is nonzero unless $CA^3B - CBCA^2B = 0$. There is no known argument which excludes this possibility, nor are there any known examples of when this situation arises. In any event, the probability of this case is rare and any code can easily be checked against these conditions. This results in the following proposition.

**Proposition 7.4.2** *When $\delta = 2$, $d_4$ is 'generically' at least 5. In particular, $d_4 \geq 5$ unless $CAB = 0$ and $CA^3B - CBCA^2B = 0$.*

We wrap up our discussion of the $\delta = 2$ case by considering $d_5$. Conjecture 7.2.2 claims this should be 6. This result has not been explicitly proven, although it can be shown that it holds 'generically' and the set of equations governing this situation can be computed explicitly as in the above case for $\gamma = 4$. However, since the hard decision error correcting capability remains the same if we show the distance is 5 instead of the full 6, we will content ourselves with showing the much easier result that $d_5 \geq 5$.

**Theorem 7.4.3** *For rate 1/2 Reed-Solomon convolutional codes with $\delta = 2$, we have $d_5 \geq 5$.*

*Proof:* Again it suffices to consider input sequences of weight at most 2. Also, the input sequences of weight 1 are again seen to generate output sequences of weight at least 5 (1 more than we need), so we need consider only the weight 2 input sequences. The cases when $u_1 \neq 0$ and $u_5 \neq 0$ are easily taken care of by Lemma 7.3.2 and Proposition 7.4.2 respectively.

For $u_2 \neq 0$, we have

$$\begin{aligned} y_1 &= CB \\ y_2 &= CAB + u_2 \\ y_3 &= CA^2B + CBu_2 \\ y_4 &= CA^3B + CABu_2 \\ y_5 &= CA^4B + CA^2u_2 \end{aligned}$$

The usual arguments show that at least 2 of $\{y_3, y_4, y_5\}$ are nonzero. Since $u_0$, $y_0$ and $u_2$ are already nonzero the result follows.

Similarly for $u_4 \neq 0$, we have

$$\begin{aligned} y_1 &= CB \\ y_2 &= CAB \\ y_3 &= CA^2B \\ y_4 &= CA^3B\,u_4 \\ y_5 &= CA^4B + CBu_4 \end{aligned}$$

Again, at least 2 of $\{y_1, y_2, y_3\}$ must be nonzero.

Finally, for the case when $u_3 \neq 0$ we have

$$\begin{aligned} y_1 &= CB \\ y_2 &= CAB \\ y_3 &= CA^2B + u_3 \\ y_4 &= CA^3B + CBu_3 \\ y_5 &= CA^4B + CABu_3 \end{aligned}$$

At least one of $\{y_1, y_2\}$ must be nonzero and at least one of $\{y_4, y_5\}$ must be nonzero.

This completes the proof. □

**Remark 7.4.4** Although a general proof of the conjectures remains elusive, the cases for small $\delta$ can be treated similarly to the case $\delta = 2$ above. These small cases further support the conjecture, but since they increase factorially in complexity it is difficult to proceed much further by hand.

## 7.5 The Road Ahead and a Stronger Conjecture

We conclude with a discussion of the importance of the conjectures of this chapter by expanding upon Remark 7.2.3. We will also introduce and discuss a stronger conjecture regarding the existence of rate 1/2 convolutional codes with the smallest possible column distance function order necessary to achieve the free distance.

If Conjecture 7.2.2 holds true, then it is possible, at least theoretically, to decode any block of length $3\delta$ of a received word as long as at most $\delta$ errors occurred. That is, we can decode if at most $\delta$ of any $6\delta$ consecutive bits (including inputs and outputs) are in error regardless of in which position the errors may arise.

On one hand, we may compare this to the algebraic decoding algorithm of Chapter 6. In that algorithm it was critically important as to where the errors occurred. At most $t_{par}$ errors could occur in the $(T - \Theta)$-length sequence of inputs; at most $t_{gen}$ errors could occur in the $\Theta$-length sequence of outputs. In addition, it is necessary for an entire $\Theta$-length sequence of inputs to be error free. The enhanced decoding algorithm of Section 6.6 allows for a fraction more freedom of where the errors can occur, but the basic restraints are still in place. Using the Reed-Solomon codes of this chapter as an example, we see that since $\Theta \approx 3\delta/2$ (see Table 7.1) we require at most

$\delta/2$ errors in the last $3\delta$ bits (and all of them either inputs or outputs). Further, at most $\delta/2$ errors can occur in the first $T - \Theta \approx 5\delta/2$ inputs. Altogether this amounts to correcting about $\delta$ errors in a sequence of $11\delta/2$ bits. This is slightly better than the conjectured result until one accounts for the 'structured error' requirements imposed by the algorithm.

On the other hand, we may compare the conjectured results with the theoretical limits of the CDF.

**Theorem 7.5.1** *For a rate 1/2 convolutional code with complexity $\delta$ and $d_{free} = 2\delta + 2$, the smallest possible $\gamma$ such that $d_\gamma = d_{free}$ is $2\delta$.*

*Proof:* Assume there is some $\gamma < 2\delta$ such that $d_\gamma = d_{free}$. Let the code be given by the polynomial encoder matrix $[p(s) \;\; q(s)]$. We can assume that $p(0) \neq 0$, hence $p(s)$ divides $s^c - l$ for any c divisible by the order of $p(s)$. Given some such $c$ that is also greater than $\gamma$, there is a polynomial $u(s)$ such that $u(s)p(s) = s^c - 1$. Consider the first $\gamma + 1$ time units of the codeword $u(s)[p(s) \;\; q(s)]$. The first polynomial has only the degree 0 term as being nonzero (since $c > \gamma + 1$). If every coefficient of $u(s)q(s)$ (inside the first $\gamma + 1$ time units) is nonzero then the weight of this partial codeword becomes $\gamma + 2 < 2\delta + 2 = d_{free}$. From this it is clear that the smallest possible $\gamma$ is $2\delta$.

□

The above theorem states that, at best, we can expect to decode up to $\delta$ errors for every $4\delta$ bits. This is not surprising since this is the MDS bound for rate 1/2 block codes. Finding codes which satisfy this property is not an easy task. In general, the field size may need to be very large.

When $\delta = 1$, we have that $2\delta = 3\delta - 1$ so that Reed-Solomon convolutional codes actually satisfy this property. For larger $\delta$, Reed-Solomon codes do not necessarily have this property. Based on empirical evidence, very few, if any, Reed-Solomon convolutional codes for $\delta > 1$ have this property. None of the examples provided earlier in this chapter satisfy this stronger distance condition.

For $\delta > 1$ it is not clear that there even exists convolutional codes which satisfy this stronger condition. The following is one example for $\delta = 2$.

**Example 7.5.2** The convolutional code over $\mathbb{F}_{11}$ generated by

$$\left[\; s^2 + 10s + 4 \quad s^2 + s + 9 \;\right]$$

has $d_{free} = d_4 = 6$.

Larger examples are much more difficult to come by. Conjecture 7.3.8 should offer a reasonable starting place to look for more systematic means of discovering these codes. We will instead look to guarantee the existence of these codes.

**Conjecture 7.5.3** For a large enough field size, given a complexity $\delta$, there exists a rate 1/2 convolutional code such that $d_{2\delta} = d_{free} = 2\delta + 2$.

*Proof:* [with a gap] Let $g_1(s) = \sum_{i=0}^{\delta} a_i s^i$ and $g_2(s) = \sum_{i=0}^{\delta} b_i s^i$ be polynomials over a 'large enough' finite field, $\mathbb{F}$. For any arbitrary $\phi(s)$, with $\phi(0) \neq 0$, let

$$\phi(s)\; [g_1(s) \;\; g_2(s)] \;=\; \left[\sum x_i s^i \quad \sum y_i s^i\right].$$

We wish to show there exists some polynomials $g_1(s)$ and $g_2(s)$ over some finite field such that $\mathrm{wt}\,(x_0, \ldots, x_{2\delta+1}, y_0, \ldots, y_{2\delta+1}) \geq 2\delta + 2$.

This is equivalent to the existence of the following syndrome former matrix with the property that any set of $2\delta + 2$ columns which includes the first column must be linear independent.

$$\left[\begin{array}{cccccc|cccccc} b_0 & & & & & & a_0 & & & & & \\ \vdots & \ddots & & & & & \vdots & \ddots & & & & \\ \vdots & & \ddots & & & & \vdots & & \ddots & & & \\ b_\delta & & & \ddots & & & a_\delta & & & \ddots & & \\ & \ddots & & & \ddots & & & \ddots & & & \ddots & \\ & & b_\delta & \cdots & \cdots & b_0 & & & a_\delta & \cdots & \cdots & a_0 \end{array}\right] \left[\begin{array}{c} x_0 \\ \vdots \\ \vdots \\ x_{2\delta+1} \\ \hline -y_0 \\ \vdots \\ \vdots \\ -y_{2\delta+1} \end{array}\right] = 0$$

Denote the large matrix above by $H$. Form a $2\delta+2$ subset of columns from $H$ by selecting $x_0$ and any other $t-1$ of the $x_i$'s and also any $\ell$ of the $y_i$'s (with $t+\ell = 2\delta+2$). We then have a morphism:

$$\begin{array}{rcl} F \;:\; \mathbb{F}^{2\delta+1} \times \mathbb{F}^{\delta+1} \times \mathbb{F}^{\delta+1} & \longrightarrow & \mathbb{F}^{2\delta+2} \\ (x_{i_2}, \ldots, x_{i_t}, -y_{j_1}, \ldots, -y_{j_\ell}; a_0, \ldots, a_\delta; b_0, \ldots, b_\delta) & \longmapsto & H \begin{bmatrix} x_0 \\ x_{i_1} \\ \vdots \\ -y_{j_\ell} \end{bmatrix} \end{array}$$

The set of all points $P$ with $F(P) = 0$ forms an affine variety $\mathfrak{X}$. We claim that

$$\dim \mathfrak{X} \;\leq\; \dim \mathrm{domain}\,(F) \;-\; 2\delta + 2 \;=\; 2\delta + 1$$

To prove this claim we compute the Jacobian $J(F(\vec{x}, \vec{0}, \vec{0}))$. If one can show that this determinant is nonzero for some point $(\vec{x}, \vec{0}, \vec{0})$ in a 'large' irreducible component then we could conclude that the tangent space at that point would have dimension at most $2\delta+1$ and hence the dimension of $\mathfrak{X}$ would be at most $2\delta+1$. With this bound on the dimension of $\mathfrak{X}$, we see that the projection of $\mathfrak{X}$ onto $\mathbb{F}^{\delta+1} \times \mathbb{F}^{\delta+1}$ could not be a surjective function, which means that there must exist $g_1(s)$ and $g_2(s)$ which satisfy Conjecture 7.5.3. $\square$

## BIBLIOGRAPHY

[1] B.M. Allen. An algebraic framework for turbo codes. In *Proc. of the 36-th Allerton Conference on Communication, Control, and Computing*, pages 27–28, 1998.

[2] B.M. Allen and J. Rosenthal. Analysis of convolutional encoders via generalized sylvester matrices and state space realization. In *Proc. of the 34-th Allerton Conference on Communication, Control, and Computing*, pages 893–902, 1996.

[3] B.M. Allen and J. Rosenthal. Analyzing convolutional encoders using realization theory. In *Proceedings of the 1997 IEEE International Symposium on Information Theory*, page 287, Ulm, Germany, 1997.

[4] B.M. Allen and J. Rosenthal. Parity-check decoding of convolutional codes whose systems parameters have desirable algebraic properties. In *Proceedings of the 1998 IEEE International Symposium on Information Theory*, page 307, Boston, MA, 1998.

[5] B.M. Allen and J. Rosenthal. A matrix Euclidean algorithm induced by state space realization. *Linear Algebra Appl.*, 288:105–121, 1999.

[6] A.C. Antoulas. On recursiveness and related topics in linear systems. *IEEE Trans. Automat. Contr.*, AC-31(12):1121–1135, 1986.

[7] L.R. Bahl, J. Cocke, F. Jelinek, and J. Raviv. Optimal decoding of linear codes for minimizing symbol error rate. *IEEE Trans. Inform. Theory*, IT-20:284–287, March 1974.

[8] J. A. Ball, I. Gohberg, and L. Rodman. *Interpolation of Rational Matrix Functions*. Birkhäuser Verlag, Basel-Berlin-Boston, 1990.

[9] J.S. Baras, R. W. Brockett, and P.A. Fuhrmann. State space models for infinite-dimensional systems. *IEEE Trans. Automat. Contr.*, AC-19:693–700, 1974.

[10] A.S. Barbulescu and S.S. Pietrobon. Interleaver design for turbo codes. *Electronics Letters*, 30(25):2107–2108, 1994.

[11] A.S. Barbulescu and S.S. Pietrobon. Terminating the trellis of turbo-codes in the same state. *Electronics Letters*, 31(1):22–23, 1995.

[12] S. Barnett. *Polynomials and linear control systems*. M. Dekker, New York, 1983.

[13] C. Berrou, P. Adde, E. Angui, and S. Faudeil. A low complexity soft-output Viterbi decoder architecture. In *Proc. of IEEE Int. Conference on Communication*, pages 737–740, Geneva, Switzerland, May 1993.

[14] C. Berrou, A. Glavieux, and P. Thitimajshima. Near Shannon limit error-correcting coding and decoding: Turbo-codes. In *Proc. of IEEE Int. Conference on Communication*, pages 1064–1070, Geneva, Switzerland, May 1993.

[15] R. R. Bitmead, S. Y. Kung, B. D. O. Anderson, and T. Kailath. Greatest common divisors via generalized Sylvester and Bézout matrices. *IEEE Trans. Automat. Control*, AC-23:1043–1047, 1978.

[16] W.J. Blackert, E.K. Hall, and S.G. Wilson. Turbo code termination and interleaver conditions. *Electronics Letters*, 31(24):2082–2083, 1995.

[17] T. Cover and J. Thomas. *Elements of Information Theory*. Wiley & Sons, New York, 1991.

[18] A. Dholakia. *Introduction to Convolutional Codes with Applications*. Kluwer Academic Publishers, 1994.

[19] D. Divsalar and R.J. McEliece. Effective free distance of turbo codes. *Electronics Letters*, 32(5):445–446, 1996.

[20] D. Divsalar and F. Pollara. On the design of turbo codes. *TDA Progress Report*, 42-123:99–121, November 15 1995.

[21] S. Dolinar and D. Divsalar. Weight distributions for turbo codes using random and nonrandom permutations. *TDA Progress Report*, 42-122:56–65, August 15 1995.

[22] P. Elias. Coding for noisy channels. *IRE Conv. Rec.*, 4:37–46, 1955.

[23] G. D. Forney. Convolutional codes I: Algebraic structure. *IEEE Trans. Inform. Theory*, IT-16(5):720–738, 1970.

[24] G. D. Forney. Minimal bases of rational vector spaces, with applications to multivariable linear systems. *SIAM J. Control Optim.*, 13(3):493–520, 1975.

[25] Paul A. Fuhrmann. A matrix euclidean algorithm and matrix continued fraction expansions. *Systems and Control Letters*, 3:263–271, 1983.

[26] R.G. Gallager. Low-density parity-check codes. *IRE Trans. on Info. Theory*, IT-8:21–28, 1962.

[27] R.G. Gallager. *Low-Density Parity Check Codes.* M.I.T. Press, Cambridge, MA, 1963. Number 21 in Research monograph series.

[28] J. Hagenauer and P. Hoeher. A viterbi algorithm with soft-decision outputs and its applications. In *Proc. of IEEE Globecom '89*, pages 47.11–47.17, 1989.

[29] U. Helmke, J. Rosenthal, and J. M. Schumacher. A controllability test for general first-order representations. *Automatica*, 33(2):193–201, 1997.

[30] R. Johannesson and K. Sh. Zigangirov. *Fundamentals of Convolutional Coding.* IEEE Press, New York, 1999.

[31] P. Jung. Novel low complexity decoder for turbo codes. *Electronics Letters*, 31(2):86–87, 1995.

[32] P. Jung. Comparison of turbo-code decoders applied to short frame transmission systems. *IEEE Journal on Selected Areas in Communications*, 14(3):530–537, 1996.

[33] P. Jung and M. Nasshan. Performance evaluation of turbo codes for short frame transmission systems. *Electronics Letters*, 30(2):111–113, 1994.

[34] J. Justesen. An algebraic construction of rate $1/\nu$ convolutional codes. *IEEE Trans. Inform. Theory*, IT-21(1):577–580, 1975.

[35] T. Kailath. *Linear Systems.* Prentice-Hall, Englewood Cliffs, N.J., 1980.

[36] R. E. Kalman. Mathematical description of linear dynamical systems. *SIAM J. Control Optim.*, 1:152–192, 1963.

[37] M. Kuijper. *First-Order Representations of Linear Systems.* Birkhäuser, Boston, 1994.

[38] S.-Y. Kung, T. Kailath, and M. Morf. A generalized resultant matrix for polynomial matrices. In *Proc. IEEE Conf. on Decision and Control (Florida)*, pages 892–895, 1976.

[39] L.H. Charles Lee. Computation of the right-inverse of G(D) and the left-inverse of H(D). *Electronics Letters*, 26(13):904–906, 1990.

[40] L.H. Charles Lee. *Convolutional Coding: Fundamentals and Applications.* Artech House, Norwalk, MA, 1997.

[41] R. Lidl and H. Niederreiter. *Introduction to Finite Fields and their Applications.* Cambridge University Press, Cambridge, London, 1986.

[42] R. Lidl and H. Niederreiter. *Introduction to Finite Fields and their Applications.* Cambridge University Press, Cambridge, London, 1994. Revised edition.

[43] S. Lin and D. Costello. *Error Control Coding: Fundamentals and Applications.* Prentice-Hall, Englewood Cliffs, NJ, 1983.

[44] J. H. van Lint. *Introduction to Coding Theory.* Springer Verlag, Berlin, New York, 1982.

[45] C. C. MacDuffee. Some applications of matrices in the theory of equations. *American Mathematical Monthly*, 57:154–161, 1950.

[46] D.J.C. MacKay. Good error-correcting codes based on very sparse matrices. *IEEE Trans. Inform. Theory*, 45(2):399–431, 1999.

[47] D.J.C. MacKay and R.M. Neal. Near shannon limit performance of low density parity check codes. *Electronics Letters*, 32(18):1645–1646, 1996.

[48] D.J.C. MacKay, S.T. Wilson, and M.C. Davey. Comparison of constructions of irregular gallager codes. In *Proc. of the 36-th Allerton Conference on Communication, Control, and Computing*, pages 220–229, 1998.

[49] F. J. MacWilliams and N. J.A. Sloane. *The Theory of Error-Correcting Codes.* North Holland, Amsterdam, 1977.

[50] J. L. Massey. *Threshold decoding.* MIT Press, Cambridge, Massachusetts, 1963.

[51] J. L. Massey and M. K. Sain. Inverses of linear sequential circuits. *IEEE Trans. on Computers*, C-17(4):330–337, 1968.

[52] Ph. Piret. *Convolutional Codes, an Algebraic Approach.* MIT Press, Cambridge, MA, 1988.

[53] M. S. Ravi, J. Rosenthal, and J. M. Schumacher. Homogeneous behaviors. *Math. Contr., Sign., and Syst.*, 10:61–75, 1997.

[54] P. Robertson. Illuminating the structure of code and decoder of parallel concatenated recursive systematic (turbo) codes. In *Proceedings GLOBECOM '94*, volume 3, pages 1298–1303, San Francisco, CA, November 1994.

[55] J.P. Robinson. Error propogation and definite decoding of convolutional codes. *IEEE Trans. Inform. Theory*, IT-14(1):121–128, January 1968.

[56] J. Rosenthal. An algebraic decoding algorithm for convolutional codes. In G. Picci and D.S. Gilliam, editors, *Dynamical Systems, Control, Coding, Computer Vision: New Trends, Interfaces, and Interplay*, pages 343–360. Birkäuser, Boston-Basel-Berlin, 1999.

[57] J. Rosenthal and J. M. Schumacher. Realization by inspection. *IEEE Trans. Automat. Contr.*, AC-42(9):1257–1263, 1997.

[58] J. Rosenthal, J. M. Schumacher, and E.V. York. On behaviors and convolutional codes. *IEEE Trans. Inform. Theory*, 42(6):1881–1891, 1996.

[59] J. Rosenthal and R. Smarandache. Construction of convolutional codes using methods from linear systems theory. In *Proc. of the 35-th Annual Allerton Conference on Communication, Control, and Computing*, pages 953–960, 1997.

[60] J. Rosenthal and R. Smarandache. Maximum distance separable convolutional codes. Technical Report 1998-074, MSRI, Berkeley, California, 1998. to appear in Appl. Algebra Engrg. Comm. Comput.

[61] J. Rosenthal and E.V. York. BCH convolutional codes. Technical report, University of Notre Dame, Dept. of Mathematics, October 1997. Preprint # 271. Available at http://www.nd.edu/~rosen/preprints.html. To appear in *IEEE Trans. Inform. Theory.*

[62] J. Rosenthal and E.V. York. A construction of binary BCH convolutional codes. In *Proceedings of the 1997 IEEE International Symposium on Information Theory*, page 291, Ulm, Germany, 1997.

[63] L. D. Rudolph. Generalized threshold decoding of convolutional codes. *IEEE Trans. Inform. Theory*, IT-16(6):739–745, November 1970.

[64] C.E. Shannon. A mathematical theory of communication. *Bell Syst. Tech. J.*, 27:379–423 and 623–656, 1948.

[65] R. Smarandache and J. Rosenthal. A state space approach for constructing MDS rate $1/n$ convolutional codes. In *Proceedings of the 1998 IEEE Information Theory Workshop on Information Theory*, pages 116–117, Killarney, Kerry, Ireland, June 1998.

[66] E. D. Sontag. *Mathematical Control Theory: Deterministic Finite Dimensional Systems.* Springer Verlag, New York, 1990.

[67] D.A. Spielman. Finding good LDPC codes. In *Proc. of the 36-th Allerton Conference on Communication, Control, and Computing*, pages 211–219, 1998.

[68] J. C. Willems. From time series to linear system. Part I: Finite dimensional linear time invariant systems. *Automatica*, 22:561–580, 1986.

[69] J. C. Willems. Paradigms and puzzles in the theory of dynamical systems. *IEEE Trans. Automat. Control*, AC-36(3):259–294, 1991.

[70] W. A. Wolovich. *Linear Multivariable Systems*, volume 11 of *Appl. Math. Sc.* Springer Verlag, New York, 1974.

[71] E.V. York. *Algebraic Description and Construction of Error Correcting Codes, a Systems Theory Point of View.* PhD thesis, University of Notre Dame, 1997. Available at http://www.nd.edu/~rosen/preprints.html.