Contents

Chapter 4

Error correction codes

We now come to the last topic of this course. We will discuss how we can *protect* the information which is stored on the disc from errors which are introduced either due to physical changes (scratches on the disk, dust particles which block the laser beam, ...) or due to natural variations which occur probabilistically (e.g., so called thermal noise which is introduced in the electrical system). We will accomplish this by introducing *redundant* bits. These redundant bits can then be used in the *decoding process* to reconstruct the original information even if some bits are missing or have been changed.

For reasons of complexity it is useful to introduce the redundant bits in a *linear* manner. This brings us into the realm of *linear* and *modern* algebra. Before we can therefore get started with the task at hand, let us review some basic notions of algebra.

4.1 Some Useful Notions of Linear/Modern Algebra

4.1.1 Fields

You are all familiar with algebraic manipulations over the reals ($\mathbb{R}$) or complex numbers ($\mathbb{C}$). Both of them are *fields.* Casually speaking, this means that you can perform all the familiar operations (addition, multiplication, and their inverses, subtraction and division). More formally, a field F is a set with the two basic operations $+$ (addition) and $*$ (multiplication) which obey the following basic rules (recall the notion of a *group* which we discussed in Section **??**):

	$+$	$*$
closure: $\forall a, b \in F$	$a + b \in F$	$a * b \in F$
associativity: $\forall a, b, c \in F$	$a + (b + c) = (a + b) + c$	$a * (b * c) = (a * b) * c$
identity: $\exists 0, 1 \in F, 0 \neq 1,$ so that	$0 + a = a$	$1 * a = a$
inverse: $\forall a \in F, \exists (-a), a^{-1} \in F$ so that	$a + (-a) = 0$	$a * (a^{-1}) = 1, a \neq 0$
commutativity: $\forall a, b \in F$	$a + b = b + a \in F$	$a * b = b * a \in F$

In addition we have the distributive law: for all $a, b, c \in F$, $a * (b + c) = a * b + a * c$.

Information is inherently discrete (a set of bits). For our purpose it is therefore natural to work with *finite fields*, i.e., with fields that have a finite number of elements. The simplest such fields are the fields F_p, where p is a prime. Finite fields are also called *Galois* fields, in honor of Jean-Baptiste Galois. As the story goes, he jotted down the basic theory of finite fields (and much more) in the fortnight of a duel which ended his life at the young age of 21 (her name was Stéphanie).

Example 1 (The Fields F_p)**.** *We encountered the field F_p already in Section* **??**. *There we denoted it as $\mathbb{Z}_p$. It is the set $\{0, \cdots, p-1\}$ under the standard integer arithmetic but modulo p. In order to show that $\mathbb{Z}_p$ indeed forms a field we need to verify all the above properties. Most of them are quite straightforward and are inherited from the usual properties of integer arithmetic. The only one that needs some thought is the existence of an inverse with respect to multiplication. Let us repeat the argument here which shows that this inverse exists. Let $a \in \mathbb{Z}_p$, $a \neq 0$. Then by the extended Euclidean algorithm there exist two integers α and π so that*

$$a\alpha + p\pi = \gcd(a, p) = 1,$$

where we have used the fact that p is a prime and that $0 < a < p$ so that $\gcd(a, p) = 1$. If we take this relationship modulo p then we get $a\alpha = 1$. In words, α is the multiplicative inverse of a, $a^{-1} = \alpha$.

Example 2 (The Field F_2)**.** *The* binary *field is of particular importance. F_2 consists of only two elements, namely* 0 *and* 1.

An alternative way of representing F_2 is to think of the two elements 0 *and* 1 *as logical values. From this point of view "addition" corresponds to taking the XOR whereas "multiplication" corresponds to taking the AND. This is shown in detail in the following tables. This point of view is convenient since these operations can be implemented efficiently on a computer.*

$+$	0	1
0	0	1
1	1	0

$*$	0	1
0	0	0
1	0	1

The field F_2 has the following curious property. If $a \in F_2$ then $a + a = 0$. Put differently, $a = -a$. In words, addition and subtraction are the same and signs do not matter.

Everything you have learned in linear algebra (how to solve systems of equations using matrices, the condition for a system to have a unique solution in terms of the rank of a matrix, ...) stays valid if we use a finite field F instead of $\mathbb{R}$ or $\mathbb{C}$.

Example 3 (System of Equations over F_7)**.** *Consider the linear system of equations*

$$\begin{aligned} 3x_1 + 4x_2 &= 2, \\ x_1 + 2x_2 &= 0, \end{aligned}$$

where all variables are elements of F_7. To solve it we proceed in exactly the same manner as we would if the equations were over $\mathbb{R}$. We rewrite the system in matrix form as $\boldsymbol{A}x^T = (2, 0)^T$, where

$$\boldsymbol{A} = \begin{pmatrix} 3 & 4 \\ 1 & 2 \end{pmatrix}, \qquad \boldsymbol{x} = (x_1, x_2).$$

We know that the system has a unique solution if $\boldsymbol{A}$ has full rank. This in turn we can check by computing the determinant of $\boldsymbol{A}$. Indeed, in our case we get $|\boldsymbol{A}| = 3 \cdot 2 - 1 \cdot 4 = 2 \neq 0$, where all computations are in F_7. We get

$$\boldsymbol{A}^{-1} \overset{(i)}{=} \frac{\begin{pmatrix} \boldsymbol{A}_{22} & -\boldsymbol{A}_{12} \\ -\boldsymbol{A}_{21} & \boldsymbol{A}_{11} \end{pmatrix}}{|\boldsymbol{A}|} = \frac{\begin{pmatrix} 2 & -4 \\ -1 & 3 \end{pmatrix}}{|\boldsymbol{A}|} = \frac{\begin{pmatrix} 2 & 3 \\ 6 & 3 \end{pmatrix}}{2} \overset{(ii)}{=} \begin{pmatrix} 2 & 3 \\ 6 & 3 \end{pmatrix} 4 = \begin{pmatrix} 1 & 5 \\ 3 & 5 \end{pmatrix}.$$

In step (i) we have used Cramer's rule to compute the inverse. This is easily verified: if $|\boldsymbol{A}| \neq 0$ then

$$\frac{\begin{pmatrix} \boldsymbol{A}_{11} & \boldsymbol{A}_{12} \\ \boldsymbol{A}_{21} & \boldsymbol{A}_{22} \end{pmatrix}\begin{pmatrix} \boldsymbol{A}_{22} & -\boldsymbol{A}_{12} \\ -\boldsymbol{A}_{21} & \boldsymbol{A}_{11} \end{pmatrix}}{|\boldsymbol{A}|} = \frac{|\boldsymbol{A}|\begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}}{|\boldsymbol{A}|} = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}.$$

In step (ii) we have used the fact that the multiplicative inverse of 2 *is* 4 *(just try it: $2 * 4 = 8 = 1$* mod 7*). Therefore, instead of dividing by* 2 *we can multiply by* 4*. It follows that the solution is $\boldsymbol{x}^T = \boldsymbol{A}^{-1}(2, 0)^T = (2, 6)$.*

4.1.2 Vector Space

Just in case you forgot, let us review some basic notions of linear algebra.

Let $\mathcal{V}$ be a commutative group with respect to addition (see Section **??**), whose elements $\boldsymbol{u}, \boldsymbol{v}, \ldots$ are called *vectors* and let F be a field whose elements $a, b, \ldots$ are called *scalars*. The set $\mathcal{V}$ is a *vector space* over F if it has the following properties: for all $\boldsymbol{u}, \boldsymbol{v} \in \mathcal{V}$ and all $a, b \in F$,

(i) closure: $a\boldsymbol{v} \in \mathcal{V}$

(ii) associativity for scalar multiplication: $a(b\boldsymbol{v}) = (ab)\boldsymbol{v}$

(iii) identity: $1\boldsymbol{v} = \boldsymbol{v}$

(iv) distributivity: $a(\boldsymbol{u} + \boldsymbol{v}) = a\boldsymbol{u} + a\boldsymbol{v}$ and $(a + b)\boldsymbol{u} = a\boldsymbol{u} + b\boldsymbol{u}$

The best known example of a vector space is the set $\mathbb{R}^n$ of real number tuples. Indeed, we can show that $\mathcal{V} = \mathbb{R}^n$ is a commutative group with respect to addition and that $F = \mathbb{R}$ is a field and that the above properties (closure, associativity, identity, distributivity) are fulfilled.

A subset $\mathcal{S}$ of $\mathcal{V}$ is a *subspace* if $\mathcal{S}$ itself is a vector space over F, *i.e.,* if it is a commutative group which fulfills the above properties. Since $\mathcal{S}$ is a subset of $\mathcal{V}$, which by assumption forms a vector space, associativity and distributivity are already ensured; therefore, it suffices to check that

$$a\boldsymbol{u} - \boldsymbol{v} \in \mathcal{S} \text{ for all } a \in F \text{ and } \boldsymbol{u}, \boldsymbol{v} \in \mathcal{S}.$$

Taking $\boldsymbol{u} = \boldsymbol{v}$ and $a = 1$, it follows that an additive identity exists; that an additive inverse exists follows by setting $\boldsymbol{u} = \boldsymbol{0}$; taking $a = 1$, additive closure is guaranteed; finally, by choosing $\boldsymbol{v} = 0$, multiplicative closure can be shown.

Example 4. *We already know that $\mathbb{R}^2$ is a vector space. Let us show that the set of vectors $\mathcal{S} = (u, 3u)$ forms a subspace.*

$$\begin{aligned} a \cdot (u, 3u) - (v, 3v) &\overset{?}{\in} \mathcal{S} \\ (au - v, 3au - 3v) = (au - v, 3(au - v)) &\overset{\surd}{\in} \mathcal{S} \end{aligned}$$

This confirms, first of all, that $\mathcal{S}$ together with addition is a commutative group and, second, that $\mathcal{S}$ is a vector space over F.

4.1.3 Metric Space

We need more than just a vector space. We also need the notion of a *distance* or *metric*.

A *metric space* $(\mathcal{V}, d(\cdot, \cdot))$ is a vector space $\mathcal{V}$ together with a *distance* (or *metric*) $d(\boldsymbol{u}, \boldsymbol{v})$ between pairs of vectors $\boldsymbol{u}, \boldsymbol{v} \in \mathcal{V}$. The distance is a map $\mathcal{V} \times \mathcal{V} \to \mathbb{R}$ satisfying the three following axioms: for all $\boldsymbol{u}, \boldsymbol{v}, \boldsymbol{w} \in \mathcal{V}$,

(i) $d(\boldsymbol{u}, \boldsymbol{v}) \geq 0$ and $d(\boldsymbol{u}, \boldsymbol{v}) = 0$ if and only if $\boldsymbol{u} = \boldsymbol{v}$

(ii) symmetry: $d(\boldsymbol{u}, \boldsymbol{v}) = d(\boldsymbol{v}, \boldsymbol{u})$

(iii) triangle inequality: $d(\boldsymbol{u}, \boldsymbol{v}) \leq d(\boldsymbol{u}, \boldsymbol{w}) + d(\boldsymbol{w}, \boldsymbol{v})$

Example 5. *$\mathbb{R}^2$ together with the Euclidean distance is a metric space, where if $\boldsymbol{u} = (u_1, u_2)$ and $\boldsymbol{v} = (v_1, v_2)$, $u_1, u_2, v_1, v_2 \in \mathbb{R}$,*

$$d(\boldsymbol{u}, \boldsymbol{v}) = \sqrt{(u_1 - v_1)^2 + (u_2 - v_2)^2}.$$

4.1.4 Linear Combinations and the Notion of Basis

Let $\mathcal{V}$ be a vector space, and let $\{\boldsymbol{v}_i\}_{i=1}^m$ be a set of vectors of $\mathcal{V}$. A *linear combination* of the vectors $\boldsymbol{v}_i \in \mathcal{V}$ is a sum of the form

$$\boldsymbol{u} = \sum_{i=1}^{m} a_i \boldsymbol{v}_i, \tag{4.1}$$

where the coefficients $a_i \in F$. We claim that the set of vectors $\boldsymbol{u}$ generated by all linear combinations of m vectors $\{\boldsymbol{v}_i\}$ forms a subspace which we call *span*$\{\boldsymbol{v}_i\}$.

The set of vectors $\{\boldsymbol{v}_i\}_{i=1}^m$ is *linearly independent* if and only if $\boldsymbol{u} = \sum_{i=1}^m a_i \boldsymbol{v}_i = \boldsymbol{0}$ implies that all the coefficients a_i are zero. In this case, the representation (4.1) is unique: there is only one set of coefficients $\{a_i\}$ which allows us to write the vector $\boldsymbol{u}$ in this manner.

The vectors $\{\boldsymbol{v}_i\}$ *span* the vector space $\mathcal{V}$ if and only if any vector of $\mathcal{V}$ can be written as a linear combination of these vectors.

The set of vectors $\{\boldsymbol{v}_i\}$ forms a *basis* of the vector space $\mathcal{V}$ if these vectors are linearly independent and span the vector space $\mathcal{V}$. The number m of elements of a basis is the *dimension* of this vector space. If $\mathcal{V}$ has dimension m, then a set containing more than m vectors is necessarily linearly dependent.

Example 6. *We can see that the following set of vectors, taken in $\mathbb{R}^3$*

$$\{(1,0,0),\ (0,1,0),\ (0,0,1)\}$$

is linearly independent, because any linear combination will be of the form

$$(a_1 + a_2 \cdot 0 + a_3 \cdot 0, a_1 \cdot 0 + a_2 + a_3 \cdot 0, a_1 \cdot 0 + a_2 \cdot 0 + a_3) = (a_1, a_2, a_3)$$

and there is no way that the resulting vector is zero unless $a_1 = a_2 = a_3 = 0$.

We can see that these three vectors span $\mathbb{R}^3$ because any vector $\boldsymbol{u} = (u_1, u_2, u_3)$ can be written as a linear combination by choosing $a_1 = u_1$, $a_2 = u_2$ and $a_3 = u_3$. This set of vectors thus forms a basis of $\mathbb{R}^3$. This means that $\mathbb{R}^3$ is of dimension 3.

We can easily see that adding a fourth vector to this set would make them linearly dependent, because the fourth could always be written as a linear combination of the first three. For example

$$(3, 2, 0) = 3 \cdot (1, 0, 0) + 2 \cdot (0, 1, 0).$$

4.1.5 The Vector Space F^n and the Hamming Metric

We are interested in vector spaces over a finite field F. We therefore consider the vector space $\mathcal{V}$ which is formed by vectors $\boldsymbol{v} = (v_1, \ldots, v_n)$ where $v_i \in F$. All our vectors are *row* vectors. We will see in Exercise 1 that $\mathcal{V}$ is a vector space. Clearly, its dimension is n as it has the basis of the n linearly independent vectors $(1, 0, \ldots, 0)$, $(0, 1, \ldots, 0)$, ..., $(0, 0, \ldots, 1)$.

We define the *Hamming weight* or *weight* of $\boldsymbol{v}$ as

$$w(\boldsymbol{v}) = |\{i : v_i \neq 0\}|.$$

In words, the Hamming weight of a vector $\boldsymbol{v}$ is equal to the number of non-zero components in the tuple $\boldsymbol{v} = (v_1, \ldots, v_n)$.

The distance between two vectors $\boldsymbol{u}$ and $\boldsymbol{v}$, called the *Hamming distance*, is the number of positions in which u_i and v_i differ:

$$d(\boldsymbol{u}, \boldsymbol{v}) = w(\boldsymbol{u} - \boldsymbol{v}) = |\{i : (u_i - v_i) \neq 0\}|.$$

Example 7 ($F = F_2$)**.** *Because* $\boldsymbol{u} = (1, 0, 1, 1, 1, 0)$ *and* $\boldsymbol{v} = (1, 0, 0, 1, 1, 1)$ *differ in their third and last positions, we see that their Hamming distance is 2. Indeed,*

$$d(\boldsymbol{u}, \boldsymbol{v}) = w(\boldsymbol{u} - \boldsymbol{v}) = w(\ (0, 0, 1, 0, 0, 1)\ ) = 0 + 0 + 1 + 0 + 0 + 1 = 2.$$

Remark: Note the different meanings of 0 *and* 1 *in this equation. In the expression* $w(\ (0, 0, 1, 0, 0, 1)\ )$, 0 *and* 1 *are elements of* F_2*, whereas on the right, in the expression* $0 + 0 + 1 + 0 + 0 + 1$*,* 0 *and* 1 *are integers! Note also that for* $F = F_2$*,* $d(\boldsymbol{u}, \boldsymbol{w}) = \sum_{i=1}^{n} |u_i - v_i|$.

You will show in Exercise 1 that $d(\cdot, \cdot)$ is indeed a distance as defined in Section 4.1.3. Therefore, F^n is a metric space as defined in Section 4.1.3.

4.2 What is a Code

In most of the following examples we use $F = F_2$, i.e., the field with two elements, but all statements are true for the general case.

We start with an example.

Example 8. *Consider the following block code of* length $n = 7$ *over* $F = F_2$ *with* 8 codewords*.*

$k = 3$		$n = 7$	$w(x)$
000	$\mapsto$	0000000	0
001	$\mapsto$	0011100	3
010	$\mapsto$	0111011	5
011	$\mapsto$	0100111	4
100	$\mapsto$	1110100	4
101	$\mapsto$	1101000	3
110	$\mapsto$	1001111	5
111	$\mapsto$	1010011	4

Imagine that you want to store a file. To keep the exposition at a manageable level let us look at a very short file, consisting of only three bits. Of course, any real file would be considerably larger.

A priori we do not know what the contents of the file will be. We therefore have to provision for all possible cases. There are $2^3 = 8$ possible files consisting of 3 bits: these are, 000, 001, $\cdots$, 111. We map each of these 8 possible information vectors of length 3, each of them representing one possible realization of the file, into a distinct codeword*. For our case the codewords are of length $n = 7$. This means we expand the length of the file from 3 to 7. We hope that we can use the* redundant *information contained in the codeword at a later stage when we want to reconstruct the original file from a possibly corrupted observation. The map which maps the information (the file) to the codeword is called the* encoding map*.*

The code *$\mathcal{C}$ is the set of all codewords. We say that $\mathcal{C}$ is a block code since we map blocks of data into blocks of codewords. We say that the code has* length *$n = 7$. We also say that the code has* rate *$r = \frac{k}{n} = \frac{3}{7}$, since we map blocks of length $k = 3$ into blocks of length $n = 7$. The code itself as well as the map between the information vectors is fixed once and for all and it is known both to the person that writes the data onto a disk (the sender) as well as the person that reads the data back (the receiver).*

Definition 1 (Block Code)**.** *A block* code *$\mathcal{C}$ of length n is a set of n-tuples over F. The* rate *of the code is defined as $r = \frac{1}{n}\log_{|F|}|\mathcal{C}|$.*

Discussion: For our previous example we have $r = \frac{1}{7}\log_2(8) = \frac{3}{7}$, which is consistent with our previous definition.

So far we have seen that a block code has the following parameters. It has a length n and a rate r. How shall we choose the code? What makes a code good? We will see that the third most important parameter of a block code is its *minimum distance*.

Definition 2 (Minimum Distance)**.** *The* minimum *distance of a block code $\mathcal{C}$, denote it by $d_{min}(\mathcal{C})$, is given by*

$$d_{min}(\mathcal{C}) = \min_{\boldsymbol{x},\boldsymbol{x'} \in \mathcal{C}; \boldsymbol{x} \neq \boldsymbol{x'}} d(\boldsymbol{x}, \boldsymbol{x'}).$$

In words, the minimum distance of a code is the closest any pair of distinct codewords is to each other. As we will see shortly: a good code is a code that has a large minimum distance.

Example 9 (Minimum Distance of Our Running Example)**.** *As we see from the previous definition, in order to determine the minimum distance of a given code with M codewords we need to take the minimum over all distinct pairs of codewords. This means that we need to check $\binom{M}{2} = \frac{M(M-1)}{2}$ pairs. (Imagine that you are at a party and that everyone wants to shake hands with everyone else; if there are M people at the party then the total number of handshakes required is $\binom{M}{2}$). For our running example we have $M = 8$, so that we need to check $\binom{8}{2} = 28$ pairs. This is manageable and we get $d_{min}(\mathcal{C}) = 3$. For larger codes a direct brute force check might not be possible. E.g., we will soon learn Reed-Solomon codes. For the specific codes we discuss the number of codewords are of the order of $M = 256^{128}$. Clearly, we will need to think in such cases of cleverer schemes of determining the minimum distance.*

4.3 Channel Models

The basic motivation for using coding is that when you store information on a compact disk and read the information back at a later time it can happen that the result differs from what was stored. The physical process that can lead to this change can be arbitrarily complicated and we would like to avoid having to consider the exact such process. We therefore construct a so called *channel model*. Such a channel model is a mathematical abstraction which describes only the part of the physical channel which is important to us but abstracts from all unnecessary details. We discuss two such models: the *erasure* channel and the *symmetric channel*.

Definition 3 (Erasure Channel)**.** *In the erasure channel we assume that each component of the codeword is either received perfectly or is* erased. *Erasure means that the transmitted symbol in the erased position is replaced by a special symbol, let us use the symbol "?". The receiver therefore knows which positions are erased and it has to guess the information that is contained in these positions. In other words, the channel hides certain parts of the transmitted word.*

Example 10 (Erasure Channel)**.** *Assume that the transmitted word is* $\boldsymbol{x} = (0100111)$. *Let* $\boldsymbol{y}$ *denote the output of the channel (see Figure* **??***). Assume that* $\boldsymbol{y} = (0?001?1)$*: the channel has erased bits* 2 *and* 6*. We say that the erasure pattern has* weight 2*.*

Definition 4 (Symmetric Channel)**.** *In the symmetric channel each component of the codeword is either received perfectly or it is changed to one of the other* $|F| - 1$ *symbols, uniformly. This means some positions are changed. If* $F = F_2$*, then this means that some positions are "flipped" from a* 0 *to a* 1 *or vice versa.*

Example 11 (Symmetric Channel)**.** *Consider again our running example. Assume that the transmitted codeword is* $\boldsymbol{x} = (0100111)$*. Let* $\boldsymbol{y}$ *be the output of the channel. Assume we have* $\boldsymbol{y} = (0000101)$*. If we compare this to* $\boldsymbol{x}$ *we see that the channel introduced two errors, namely at position* 2 *and* 6*. Define* $\mathbf{e} = \boldsymbol{y} - \boldsymbol{x}$*. For our example we have* (0010010)*. We call* $\boldsymbol{e}$ *the* error vector*. It is non-zero exactly at those positions in which the channel introduced an error. We have* $w(\boldsymbol{e}) = 2$*, i.e., the weight of* $\mathbf{e}$ *is equal to the number of errors that the channel introduced. Also note that* $\boldsymbol{y} = \boldsymbol{x} + \mathbf{e}$*. In words, the effect of the channel is to add an error vector to the codeword.*

4.4 Basic Principles of Error Correction

We now see why the minimum distance plays a key role in coding.

4.4.1 Error Detection

Assume an operator of a nuclear power plant wants to send a signal to the reactor to shut down. Slightly less dramatic, assume you want to save your bank record on a disk. In both cases we would like to ensure that the sent signal (the stored information) is received (read back) correctly. In other words, you would like to make sure that errors in the transmission are detected.

Let the code $\mathcal{C}$ be given. Assume we choose a codeword $\boldsymbol{x} \in \mathcal{C}$ and that $\boldsymbol{x}$ is sent over a symmetric channel. The receiver observes the channel output which is called $\boldsymbol{y}$. Of course, the receiver *does not know* the sent codeword $\boldsymbol{x}$. He only knows the received word $\boldsymbol{y}$. If $\boldsymbol{y} \notin \mathcal{C}$ then the receiver knows that the channel introduced some errors. In this case the receiver can raise a flag, alerting the user to the fact that something went wrong during transmission. We say in this case that the receiver can *detect the error.*

Lemma 1 (Error Detection)**.** *A code $\mathcal{C}$ of minimum distance $d_{min} = d_{min}(\mathcal{C})$ can detect all errors of weight $t \leq d_{min} - 1$ or less.*

Proof. Let $\boldsymbol{x}$ be the transmitted word and $\boldsymbol{y}$ be the received word. Recall that we have $\boldsymbol{y} = \boldsymbol{x} + \boldsymbol{e}$, where $\boldsymbol{e}$ is the error vector. Assume that at most $d_{\min} - 1$ errors occurred, i.e., that $w(\boldsymbol{e}) \leq d_{\min} - 1$. Then $d(\boldsymbol{x}, \boldsymbol{y}) = w(\boldsymbol{e}) \leq d_{\min} - 1$. It follows that $\boldsymbol{y}$ can not be a codeword (so that the error is detectable) since $\boldsymbol{x}$ is a codeword and by definition of the code any two codewords have distance at least $d_{\min}$. $\square$

Example 12 (Error Detection)**.** *Consider our running example. Assume that $\boldsymbol{x} = (0100111)$ and that $\boldsymbol{e} = (0100010)$ so that $\boldsymbol{y} = (0000101)$. We have $w(\boldsymbol{e}) = 2 \leq d_{min} - 1 = 2$. By Lemma 1 we know that $\boldsymbol{y}$ can not be a codeword and that the error is detectable. Indeed, if we go down the list of codewords for our running example, we see that $\boldsymbol{y}$ is not a codeword.*

4.4.2 Erasure Correction

Lemma 2 (Erasure Correction)**.** *Let the code $\mathcal{C}$ be given. We choose a codeword $\boldsymbol{x} \in \mathcal{C}$ and send it over the erasure channel. Let $\boldsymbol{y}$ be the output of the channel. If $\boldsymbol{y}$ contains at most $t \leq d_{min} - 1$ erasures then we can reconstruct the transmitted codeword uniquely.*

Proof. Given $\boldsymbol{y}$ which has t erasures find all codewords $\boldsymbol{x'} \in \mathcal{C}$ which agree with $\boldsymbol{y}$ in all *known* positions. Clearly, the transmitted word $\boldsymbol{x} \in \mathcal{C}$ is one of those. If it is the unique such codeword then we can reconstruct $\boldsymbol{x}$ uniquely. But if $\boldsymbol{x'}$ was another codeword that agrees with $\boldsymbol{y}$ on all known positions then it can differ from $\boldsymbol{x}$ in at most t positions. This leads to a contradiction since $t(\boldsymbol{x}, \boldsymbol{x'}) \leq t \leq d_{\min} - 1$; but any two codewords must have distance at least $d_{\min}$. $\square$

Example 13 (Erasure Correction)**.** *Consider our running example. Assume we receive $\boldsymbol{y} = (0?001?1)$. Since $t = 2 \leq d_{min} - 1 = 2$ we know that we can reconstruct the transmitted word uniquely. Indeed, if we go down the list of all codewords we see that $\boldsymbol{x} = (0100111)$ is the only codeword that is compatible with $\boldsymbol{y}$.*

4.4.3 Error Correction

Lemma 3 (Error Correction)**.** *We are given a code $\mathcal{C}$. Choose a codeword $\boldsymbol{x} \in \mathcal{C}$ and transmit it over the symmetric channel. Let $\boldsymbol{y}$ be the output of the channel. If the channel introduces t errors, where $t \leq \frac{d_{min}-1}{2}$, then we can correct the errors.*

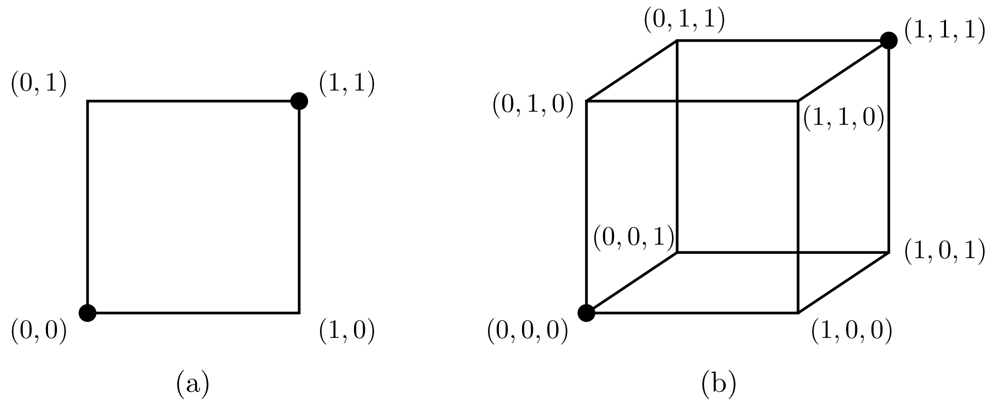


Figure 4.1: (2, 1) (a) and (3, 1) (b) repetition codes

Proof. To correct the errors use the following procedure. Given $\boldsymbol{y}$ find all codewords $\boldsymbol{x'} \in \mathcal{C}$ so that $d(\boldsymbol{y}, \boldsymbol{x'}) \leq \frac{d_{\min}-1}{2}$. By hypothesis the transmitted word $\boldsymbol{x}$ is in this list. We claim that this is the only one. Indeed, if there is another word $\boldsymbol{x'} \in \mathcal{C}$ so that $d(\boldsymbol{y}, \boldsymbol{x'}) \leq \frac{d_{\min}-1}{2}$ this would imply by the triangle inequality that $d(\boldsymbol{x}, \boldsymbol{x'}) \leq d(\boldsymbol{x}, \boldsymbol{y}) + d(\boldsymbol{y}, \boldsymbol{x'}) \leq \frac{d_{\min}-1}{2} + \frac{d_{\min}-1}{2} = d_{\min} - 1$, a contradiction. □

Example 14 (Error Correction)**.** *Assume we receive the word* $\boldsymbol{y} = (0000111)$. *If we go down the list of all codewords we see that the only element of* $\mathcal{C}$ *which is within distance* $\frac{d_{min}-1}{2} = 1$ *from* $\boldsymbol{y}$ *is the word* $\boldsymbol{x} = (0100111)$. *Hence, we declare* $\boldsymbol{x}$ *to be the transmitted word. If there was at most one error in the channel then we decoded correctly.*

From our previous discussion it is hopefully clear that the basic problem of coding theory is to construct codes that have a large number of codewords (large rate) and at the same time a large minimum distance (resilience against errors). If you look at Exercises 9 and 15 you will discover that these are somewhat conflicting goals. Given a desired rate (and blocklength) the minimum distance can not be increased beyond a certain point. Our aim is then to operate as close to the optimum as possible.

4.5 Linear Codes and Some Simple Examples

So far a code $\mathcal{C}$ was simply a collection of n-tuples over some finite field F that were separated by a distance of at least $d_{\min}$ from each other. In order to find efficient procedures to accomplish the encoding and the decoding operation it is useful to require that the codewords have some additional structure.

Definition 5 (Linear Code)**.** *We say that a code* $\mathcal{C}$ *over the field* F *is* linear *if for all* $\boldsymbol{x}, \boldsymbol{x'} \in \mathcal{C}$ *and all* $a \in F$ *we have*

$$a\boldsymbol{x} - \boldsymbol{x'} \in \mathcal{C}.$$

This is important: As we have discussed in Section 4.1.5, F^n is a vector space over the field F. The above condition therefore is equivalent to saying that a linear code is a subspace of F^n. In the sequel, if the dimension of the code $\mathcal{C}$ of length n equals k and the minimum distance is $d_{\min}$ then we say that the code has parameters $[n, k, d_{\min}]$.

Example 15 (Our Running Example is a Linear Code)**.** *We can check that the above condition is fulfilled for our running example. We conclude that this code is linear.*

The first important simplification that arises when we consider a linear code is that its minimum distance can be computed much more efficiently.

Lemma 4. *Let $\mathcal{C}$ be a linear code. Then*

$$d_{min} = \min_{\boldsymbol{x} \in \mathcal{C}; \boldsymbol{x} \neq 0} w(\boldsymbol{x}).$$

Proof. Starting from Definition 2 we have

$$d_{\min} = \min_{\boldsymbol{x}, \boldsymbol{x'} \in \mathcal{C}, \boldsymbol{x} \neq \boldsymbol{x'}} d(\boldsymbol{x}, \boldsymbol{x'}) = \min_{\boldsymbol{x}, \boldsymbol{x'} \in \mathcal{C}, \boldsymbol{x} \neq \boldsymbol{x'}} w(\boldsymbol{x} - \boldsymbol{x'}) = \min_{\boldsymbol{x} \in \mathcal{C}, \boldsymbol{x} \neq 0} w(\boldsymbol{x}),$$

where in the last step we have made use of the fact that due to the linearity of the code $\boldsymbol{x} - \boldsymbol{x'}$ is again a (non-zero) codeword. □

Let us look at a few simple linear codes.

Example 16 ($(n, n-1)$ Single Parity-Check Code.)**.** *The words of the code are obtained by adding a single parity-check bit r_i to each message $\boldsymbol{u}_i$ which is chosen so that the number of non-zero bits in the codeword $\boldsymbol{y}_i$ is even. For example, $\boldsymbol{u} = (1,0,1,1,0,0,1)$ becomes $\boldsymbol{y} = (1,0,1,1,0,0,1,0)$. The distance $d_{\min}$ of this code is thus 2, and according to our previous discussion this code can detect any single error, can correct any single erasure, but can not correct any errors. (In fact, we notice that this code detects any odd number of errors and erroneously decodes any configuration which has an even number of errors.)*

Example 17 ($(n, 1)$ Repetition Code.)**.** *Here there is only one information bit and $n-1$ control bits, which are generated by repetition of the information bit. The distance of this code is thus $d_{min} = n$, which allows detection of any configuration of $n-1$ errors, any correction of $n-1$ erasures, or any correction of configurations of no more than $(n-1)/2$ errors. Obviously, while the detection/correction capacity of this code is very high, its rate is very low. As an example, we have represented the $(2,1)$ and $(3,1)$ repetition codes in Fig. 4.1, which graphically illustrates the error correction and detection capacities of these codes.*

4.5.1 Generator Matrix of a Linear Code

Since a linear code $\mathcal{C}$ is a subspace it has a basis: any codeword $\boldsymbol{x} \in \mathcal{C}$ can be written in a unique way as a linear combination of these basis vectors. It is convenient to write this relationship in terms of matrix notation. The codewords of an (n, k) linear code over F_2 are the 2^k tuples spanned by a *generator matrix* G of k rows and n columns. The k row vectors of the matrix G are chosen so as to be linearly independent. In particular, the rows of G are k vectors of the code and the null vector belongs to the code.

Let $\boldsymbol{u}$ be a k-tuple to be encoded. The corresponding codeword $\boldsymbol{x}$ is

$$\boldsymbol{x} = \boldsymbol{u}G. \tag{4.2}$$

Example 18. *A generator matrix which produces our running example is*

$$\boldsymbol{G} = \begin{bmatrix} 1 & 1 & 1 & 0 & 1 & 0 & 0 \\ 0 & 1 & 1 & 1 & 0 & 1 & 1 \\ 0 & 0 & 1 & 1 & 1 & 0 & 0 \end{bmatrix}. \tag{4.3}$$

The subspace spanned by the rows of $\boldsymbol{G}$ is not changed by a permutation of the rows or by adding one row to the other. Therefore, we can apply elementary row operations to get $\boldsymbol{G}$ in row-reduced echelon form. But we can do more. Assume that we also allow column permutations. Strictly speaking this changes the subspace. But the main characteristics of the code (the rate and the minimum distance) stay unchanged. Using column permutations if necessary, we can write $\boldsymbol{G}$ in systematic form:

$$\boldsymbol{G} = [I_k \quad P], \tag{4.4}$$

where I_k is the identity matrix of order k (a matrix where all the diagonal elements are 1 and all the non diagonal elements are 0) and P is a matrix whose size is $k \times (n-k)$. For instance, the matrix $\boldsymbol{G}$ given in (4.3) can be written in the form

$$\boldsymbol{G} = \begin{bmatrix} 1 & 0 & 0 & 1 & 1 & 1 & 1 \\ 0 & 1 & 0 & 0 & 1 & 1 & 1 \\ 0 & 0 & 1 & 1 & 1 & 0 & 0 \end{bmatrix} \tag{4.5}$$

by using elementary operations on its rows (add the first row to the second). The codewords remain the same, but their mapping onto the k-tuples $\boldsymbol{u}_i$ is modified. The linear code is then said to be *systematic* and simply consists in leaving the k information carrying bits unchanged and adding the $n-k$ control bits r_i onto the end:

$$\boldsymbol{x} = (x_1, \ldots, x_n) = (u_1, u_2, \ldots, u_k, r_1, r_2, \ldots, r_{n-k}).$$

The parity-check bits are thus calculated as follows from the information bits:

$$\begin{aligned} r_1 &= u_1 + u_3, \\ r_2 &= u_1 + u_2 + u_3, \\ r_3 &= u_1 + u_2, \\ r_4 &= u_1 + u_2. \end{aligned}$$

A systematic linear code has the advantage of being easy to implement with logical circuits and shift registers, as we only need to store the $k \times (n-k)$ matrix P.

4.5.2 Parity-Check Matrix

The *parity-check matrix* H of an (n,k) code generated by $\boldsymbol{G}$ is a $(n-k)\times n$ matrix whose $(n-k)$ rows are independent, and has the property that for any $\boldsymbol{x}$ belonging to the code, we have

$$\boldsymbol{x}H^T = \mathbf{0}. \tag{4.6}$$

It is easy to check that such a matrix always exists: if $\boldsymbol{G}$ is written in systematic form, then H can be constructed as

$$\boldsymbol{H} = [-P^T \quad I_{n-k}]. \tag{4.7}$$

Note that we have

$$\boldsymbol{G}H^T = [I_k \quad P]\begin{bmatrix} -P \\ I_{n-k} \end{bmatrix} = -P + P = 0.$$

In the case that the code is binary, signs do not matter and the parity-check matrix has the form $\boldsymbol{H} = [P^T \quad I_{n-k}]$.

Example 19. *The matrix H corresponding to the matrix $\boldsymbol{G}$ given by (4.5) is*

$$\boldsymbol{H} = \begin{bmatrix} 1 & 0 & 1 & 1 & 0 & 0 & 0 \\ 1 & 1 & 1 & 0 & 1 & 0 & 0 \\ 1 & 1 & 0 & 0 & 0 & 1 & 0 \\ 1 & 1 & 0 & 0 & 0 & 0 & 1 \end{bmatrix}. \tag{4.8}$$

The relations given by (4.6) are the *parity-check relations*:

$$\begin{aligned} x_1 + x_3 + x_4 &= 0 \\ x_1 + x_2 + x_3 + x_5 &= 0 \\ x_1 + x_2 + x_6 &= 0 \\ x_1 + x_2 + x_7 &= 0. \end{aligned}$$

4.5.3 Syndrome

Let us recall that $\boldsymbol{x}$ is the emitted codeword and $\boldsymbol{y} = \boldsymbol{x} + \boldsymbol{e}$ is the received word. A common technique to detect or correct errors is to calculate the vector $\boldsymbol{s}$ of $n-k$ bits, called *syndrome* and defined by

$$\boldsymbol{s} = \boldsymbol{y}H^T. \tag{4.9}$$

We see that

$$\boldsymbol{s} = \boldsymbol{x}H^T + \boldsymbol{e}H^T = \boldsymbol{e}H^T \tag{4.10}$$

is zero if there are no errors. On the other hand, all the errors represented by a vector which is not orthogonal to the rows of H are detected.

Suppose we use the systematic code given in example 19 and that $\boldsymbol{y} = (1,0,1,0,1,1,1)$ is the received vector. The syndrome $\boldsymbol{s} = (0,1,0,0)$ shows that the received word is incorrect.

Error *detection* can thus be very easily achieved by computing the syndrome.

Error *correction* is more complex. If $H = (\boldsymbol{h_1}, \boldsymbol{h_2}, \dots, \boldsymbol{h_n})$ where $\boldsymbol{h_i}$ is the ith column vector of H, (4.10) can be rewritten in the form

$$\boldsymbol{s} = e_1\boldsymbol{h_1} + e_2\boldsymbol{h_2} + \ldots + e_n\boldsymbol{h_1} = \sum_{i=1}^{n} e_i\boldsymbol{h_i} \tag{4.11}$$

and the syndrome is a linear combination of the column vectors of the parity-check matrix. This relation can be used to correct the received word, if it is possible to compute the coefficients e_i which satisfy (4.11) in a unequivocal way.

So we can see that if the n columns of H are all distinct and non-zero, a linear code can correct all *simple errors* for binary codes. Indeed, an error in the kth position produces a syndrome which is simply the $\boldsymbol{h_k}$ column, according to (4.11), and the error can be localized and thus corrected.

Example 20. *In the preceding example, we see that* $\boldsymbol{s} = (0, 1, 0, 0) = \boldsymbol{h_5}$. *Consequently,* $e_5 = 1$ *and* $\boldsymbol{e} = (0, 0, 0, 0, 1, 0, 0)$, *so the emitted word was* $\boldsymbol{x} = \boldsymbol{y} + \boldsymbol{e} = (1, 0, 1, 0, 0, 1, 1)$.

4.5.4 Hamming Code

In order to maximize the rate $r = k/n$ of such a code, we can wonder what is the maximum length n of codewords for which any simple error can be corrected, if the number $n - k$ of parity-check bits remains fixed. Indeed, $n - k = n(1 - r)$. So if $n - k$ is fixed and n increases, then r increases.

As the column vectors of the matrix H are $(n-k)$-tuples, we can form at most $2^{n-k} - 1$ of them such that they are all non-zero and distinct. We saw in the preceding section that a code can correct any simple error if the n columns are non-zero and distinct. We must therefore have

$$n \leq 2^{n-k} - 1$$

and the maximum word length is reached when $n = 2^{n-k} - 1$. The code is then a *Hamming code.* The following table gives a few Hamming codes and their rates:

n	3	7	15	31	63	127
k	1	4	11	26	57	120
$n - k$	2	3	4	5	6	7
r	0.333	0.571	0.733	0.839	0.905	0.945

By encoding long messages, we can see that it is possible to get a large rate.

The parameters of a Hamming code are $[n, k, d_{\min}] = [2^m - 1, 2^m - m - 1, 3]$, where m represents the number of parity-check bits $m = n - k$. The minimal distance is $d_{\min} = 3$, i.e., we can correct single errors. The proof of this fact is in Section 4.5.3

Example 21. *Take the following generating matrix of a $(7,4)$ Hamming code:*

$$\boldsymbol{G} = \begin{bmatrix} 1 & 0 & 0 & 0 & 1 & 0 & 1 \\ 0 & 1 & 0 & 0 & 1 & 1 & 0 \\ 0 & 0 & 1 & 0 & 1 & 1 & 1 \\ 0 & 0 & 0 & 1 & 0 & 1 & 1 \end{bmatrix}$$

such that if $\boldsymbol{x}$ is a codeword matching the message $\boldsymbol{u}$,

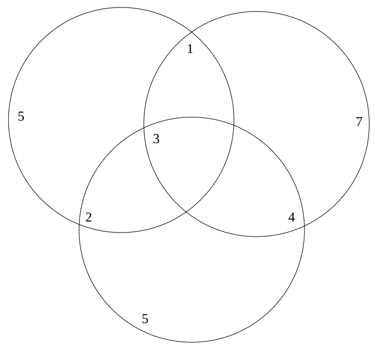
$$\begin{aligned} x_1 &= u_1 \\ x_2 &= u_2 \\ x_3 &= u_3 \\ x_4 &= u_4 \\ x_5 &= u_1 + u_2 + u_3 \\ x_6 &= u_2 + u_3 + u_4 \\ x_7 &= u_1 + u_3 + u_4 \end{aligned}$$

The parity-check matrix is

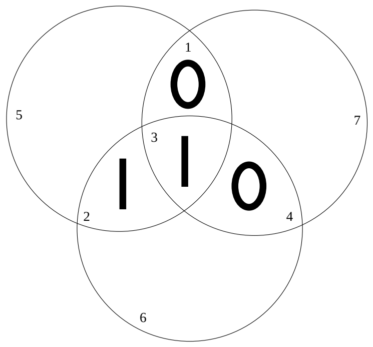
$$\boldsymbol{H} = \begin{bmatrix} 1 & 1 & 1 & 0 & 1 & 0 & 0 \\ 0 & 1 & 1 & 1 & 0 & 1 & 0 \\ 1 & 0 & 1 & 1 & 0 & 0 & 1 \end{bmatrix}.$$

The encoding of $\boldsymbol{u} = (0,1,1,0)$ is $\boldsymbol{x} = (0,1,1,0,0,0,1)$. Suppose that the received word is $\boldsymbol{y} = (0,1,1,1,0,0,1)$. We then compute the syndrome $\boldsymbol{s} = \boldsymbol{y}\boldsymbol{H}^T = (0,1,1)$. Assuming that there is only one error, we know from Section 4.5.3 that if the syndrome is to be found in the kth column of H, then the error is in the kth position. From this, we can tell that the simple error is situated in position 4.

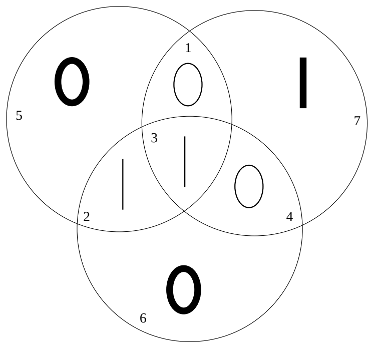
Another way of understanding this is to view the parity bit x_5 as checking the parity of u_1, u_2, and u_3, the parity bit x_6 as checking the parity of u_2, u_3, and u_4, and the third parity bit x_7 as checking the parity of u_1, u_3, and u_4. This can be visualized in the following Venn diagram, with each circle containing the elements of one parity group. Note that u_3 is checked by all three parity bits and is thus in the middle.



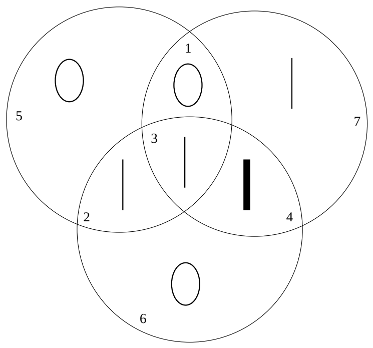
We now place the bits of the word we wish to encode in positions $1, 2, 3$ and 4.



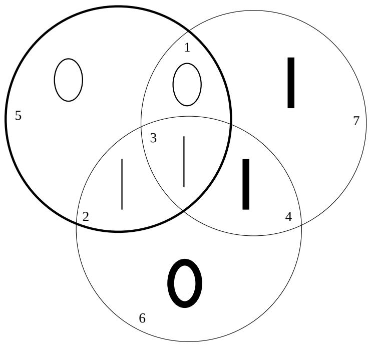
Within each circle, the number of "1" bits must be even. So the bits in positions $5, 6$ and 7 are computed to fulfill this constraint.



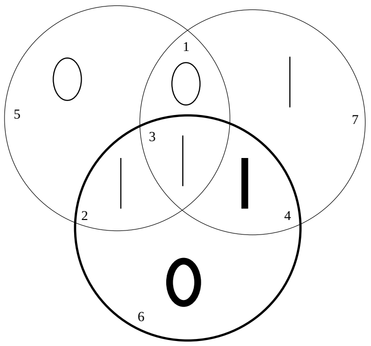
Suppose an error occurs during transmission and the bit in fourth position is incorrect.



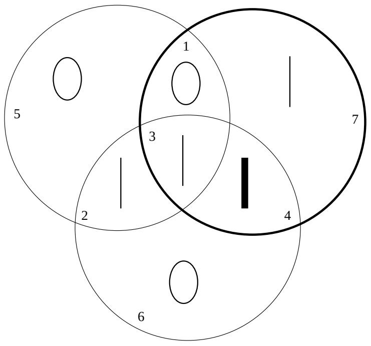
On receiving the codeword, we place it in the diagram and analyze each circle in turn. We assume that there is only a simple error. This error is either inside or outside each circle. The first circle is consistent (it has an even number of "1" bits) so all the bits inside this circle must be correct.



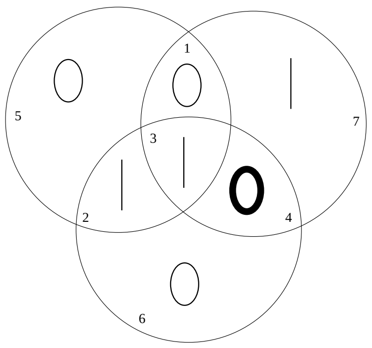
The next circle is not consistent. This means that the error is somewhere inside the circle, which implies that all bits outside the circle are correct.



The last circle is also inconsistent. We can again be certain that all bits outside that circle are correct, leaving us with only one possible incorrect bit, the bit in position 4.



All that remains now is to correct this bit in order to get the corrected codeword.



This gives us the corrected codeword, $\boldsymbol{x} = (0110001)$, thus informing us that the original word $\boldsymbol{u}$ must have been 0110.

An applet with demonstrates this method can be found *http://www.systems.caltech.edu/EE/Faculty/rjm/* under the *Hamming Code Animation* link.

4.6 Reed-Solomon Codes

The best code we have seen so far has a minimum distance of 3. This is good enough to show the basic idea of coding but next to useless for any real application. On a compact disk you can make a $1mm$ radial cut and still the code is powerful enough to recover the information. Given that a bit occupies roughly 10^{-6} meters on the disk this means that such a cut covers on the order of thousand bits along each track. How can we construct codes with a large minimum distance? This is not a simple problem and researchers have spent considerable effort in the last 60 years to find better and better codes. The problem is difficult for binary codes, i.e., codes over F_2. But if we look at larger alphabets then an optimal solution has been proposed already in the 1950's by Irvine Reed and Gus Solomon. The codes are called Reed-Solomon (RS) codes in their honor and they are one of the most used codes to date. At any given second there are hundreds of millions of RS codes which are in operation and they ensure that most of our communications takes place (essentially) error free.

Given how powerful RS codes are, their definition is remarkably simple.

Definition 6 (Reed-Solomon Codes)**.** *Given a finite field F, choose n and k such that $n \leq |F|$ and $1 \leq k \leq n$. To construct a Reed-Solomon (RS) code with parameters n and k over the field F choose n* distinct *elements from F, call them $a_0, \cdots, a_{n-1}$. Let $F[x]$ denote the set of polynomials with coefficients in F in the indeterminate x equipped with the standard addition and multiplication of polynomials. Then $\mathcal{C}$ is defined as*

$$\mathcal{C} = \{(A(a_0), \cdots, A(a_{n-1})) : A(x) \in F[x] \ s.t. \ \deg(A(x)) < k\}.$$

In words, we consider the set of polynomials with coefficients in F and degree at most $k-1$. We evaluate each such polynomial at the n distinct points a_i, $i \in [n]$, and the result of these evaluations forms the n components of a codeword.

This is best explained by an example.

Example 22 (Reed-Solomon Codes over F_5)**.** *Let us build a RS code over F_5. We can choose any blocklength n between 1 and $5 = |F_5|$. Let us choose the maximum possible, namely $n = 5$. Next we can choose the dimension k of the code. We choose $k = 2$. This gives us a code of rate $r = k/n = 2/5$. We start by writing down the set of all polynomials $A(x)$ of degree at most $k-1 = 1$ with coefficients in F_5. This set of polynomials is $\{0, 1, 2, 3, 4, x, 1+x, 2+x, 3+x, 4+x, 2x, 1+2x, 2+2x, 3+2x, 4+2x, 3x, 1+3x, 2+3x, 3+3x, 4+3x, 4x, 1+4x, 2+4x, 3+4x, 4+4x\}$. Since we can choose each of the two coefficients from F_5 and the choice of the two coefficients is independent, we get in total $5^2 = 25$ such polynomials. According to the definition we need now 5 distinct field elements a_i. Since in total there are only 5 distinct elements we let $a_i = i$, $i = 0, \cdots, 4$. We now take in turn each of the 25 polynomials. Each we evaluate at the 5 field elements a_i. E.g., take the constant polynomial $A(x) = 0$. If we evaluate it at the 5 field elements we get the all-zero vector 00000. This is one codeword. In the same manner if we take any of the constant polynomials we get the constant vectors. As a further example, take the polynomial $A(x) = 3 + 2x$. If we evaluate at the 5 field elements we get 30241. In this way we get a total of 25 codewords.*

Lemma 5 (Reed-Solomon Codes)**.** *A RS code as defined above with parameters n and k over the field F has dimension k and minimum distance $d_{min} = n - k + 1$. This is the largest minimum distance any code with parameters n and k can have.*

Proof. To see this claim, first note that the code is linear since the evaluation map is a linear map. In more detail, if $A(x)$ and $B(x)$ are two elements of $F[x]$ of degree at most $k-1$ and if $\alpha, \beta \in F$, then $C(x) = \alpha A(x) + \beta A(x)$ is also an element of $F[x]$ of degree at most $k-1$. Further, for any $a_i \in F$, $\alpha A(a_i) + \beta B(a_i) = C(a_i)$. This shows that any linear combination of codewords is again a codeword.

Before we can proceed we need to review one fact. Recall that if you have a polynomial of degree d with complex coefficients then such a polynomial has exactly d complex roots, no more and no less. This is called the fundamental theorem of algebra. For general fields it can happen that you have less. E.g., the polynomial $1 + x^2$, considered as a polynomial over the reals has no real roots (its roots are complex). But over *any* field (so also for any finite field) the number of roots (in F) of a polynomial of degree d over a field F is at most d. This is not a completely trivial fact. Unfortunately we do not have time to prove this fact here so that we have to take it by faith. Now we can proceed.

To see that the code has dimension k first note that there are exactly $|F|^k$ distinct elements $A(x)$ of $F[x]$ of degree at most $k-1$. It remains to verify that the evaluation of two distinct polynomials results in distinct codewords. Let $A(x)$ and $B(x)$ be two distinct polynomials of degree at most $k-1$ and let $C(x) = A(x) - B(x)$, which is also a polynomial in $F[x]$ of degree at most $k-1$. If

$$(A(a_0), \cdots, A(a_{n-1})) = (B(a_0), \cdots, B(a_{n-1})),$$

then

$$(C(a_0), \cdots, C(a_{n-1})) = 0.$$

But by the fundamental theorem of algebra the polynomial $C(x)$ can have at most $k-1 < n$ zeros since it has degree at most $k-1$. It follows that any non-zero codeword has weight at least $n-k+1$. Therefore, $d_{\min}(\mathcal{C}) \geq n-k+1$. But as discussed in Problem 15, the minimum distance of *any* code of length n and dimension k is upper bounded by $n-k+1$. This bound is called the *Singleton bound.* We therefore conclude that $d_{\min}(\mathcal{C}) = n-k+1$. □

For the purpose of implementation it is convenient to give a slightly more traditional representation of the code in terms of its generator matrix.

Example 23. *Consider the RS code over F_5 with $n=5$ and $k=2$ considered in Example 22. We can rewrite the evaluation map which takes a polynomial $A(x)$ and evaluates at the five points a_i to give rise to the codeword $\boldsymbol{x}$: we have $\boldsymbol{x} = \boldsymbol{uG}$, where $\boldsymbol{u}$ is the vector of length k which contains the information, and $\boldsymbol{G}$ is the generator matrix. Set $\boldsymbol{u} = (A_0, A_1)$, i.e., the components of the information word are the coefficients of the polynomial $A(x)$. Further, define the matrix $\boldsymbol{G}$ as*

$$\boldsymbol{G} = \begin{pmatrix} a_0^0 & a_1^0 & a_2^0 & a_3^0 & a_4^0 \\ a_0^1 & a_1^1 & a_2^1 & a_3^1 & a_4^1 \end{pmatrix} = \begin{pmatrix} 1 & 1 & 1 & 1 & 1 \\ 0 & 1 & 2 & 3 & 4 \end{pmatrix}.$$

Now the evaluation map has just the standard form $\boldsymbol{x} = \boldsymbol{uG}$. E.g., if we start with $A(x) = 3+2x$ then $\boldsymbol{u} = (3,2)$. The corresponding codeword is then $(3,2)\boldsymbol{G} = 30241$, the same result we got in the previous example.

According to Lemma 5, a RS code of parameters n and k has a minimum distance of $d_{\min} = n-k+1$. For our example we get $d_{\min} = 5-2+1 = 4$. We conclude that such a code can correct any erasure pattern of weight at most $d_{\min} - 1 = 3$ erasures.

Example 24 (Erasure Correction for RS Codes)**.** *Assume that we use the RS code which we constructed in Example 23 and that we receive the word $\boldsymbol{y} = (??2?1)$. There are three erasures. Since $3 \leq d_{min} - 1 = n-k = 3$, we know that we can uniquely reconstruct the transmitted word. Let $\boldsymbol{x}$ be the transmitted word. We know that $\boldsymbol{x} = \boldsymbol{uG}$. Further, we know that $\boldsymbol{x}$ and $\boldsymbol{y}$ agree on positions 3 and 5. We therefore have the system of equations*

$$(y_3 y_5) = (21) = (u_1 u_2) \begin{pmatrix} 1 & 1 \\ 2 & 4 \end{pmatrix}.$$

We can solve this system of equations (over F_5) in the same manner as we discussed in Example 3. We determine that

$$A^{-1} = \begin{pmatrix} 2 & 2 \\ 4 & 3 \end{pmatrix},$$

which gives us $(u_1 u_2) = (32)$. We have therefore recovered the sent information. If we want to we can also determine the corresponding transmitted codeword. We get $\boldsymbol{x} = \boldsymbol{uG} = 30241$.

For more realistic examples the matrix equation is typically quite large and we have to solve a lets say 100×100 system of equations. In these cases we can no longer use Cramer's rule but we have to resort to Gaussian elimination. Over finite fields Gaussian elimination is considerably easier than over the reals or complex numbers. There are no issues of ill-conditioned matrices or precision of the computation. Bring the matrix into an upper triangular form by suitable basic row operations (and switching of columns if necessary). Then use back-substitution to solve the system.

When we construct a RS code we typically have a large degree of freedom. Once the field has been fixed and the blocklength n has been chosen, any choice of n distinct field elements $a_0, \cdots, a_{n-1}$ will do. In order for different devices to be interoperable we need to fix these choices once and for all. The following choice is used often since it leads to very efficient implementations. This is also the choice that you should use for your project.

Example 25. *Consider the field F_7. As discussed in Exercise 14, take the field element* 3 *and consider the powers* $\{3^0, 3^1, 3^2, 3^3, 3^4, 3^5\} = \{1, 3, 2, 6, 4, 5\}$. *As you can see the set of powers of* 3 *covers all non-zero field elements. In other words, the field F_7 contains an element so that* $\{a^0, a^1, \cdots, a^{|F|-2}\}$ *covers all non-zero field elements. This is not a special property of F_7. In fact, any finite field has at least one such element. We call such an element a* generator *of the field.*

Given a finite field F, a blocklength n, where $n \leq |F| - 1$, a dimension k, $1 \leq k < n$, and a generator of the field a, the canonical *RS code is defined as follows: pick the n distinct field elements to be $a_i = a^i$, $i = 0, \cdots, n-1$.*

Consider again a RS code over F_5 (see Example 23). We can verify that $a = 2$ is a generator of the field since $\{2^0 = 1, 2^1 = 2, 2^2 = 4, 2^3 = 3\}$ are all distinct. Note that the largest length we can pick is $n = 4$ since a only generates the non-zero elements of the field. If we pick $k = 3$ the generator matrix has the form

$$\begin{aligned} \boldsymbol{G} &= \begin{pmatrix} a_0^0 & a_1^0 & a_2^0 & a_3^0 \\ a_0^1 & a_1^1 & a_2^1 & a_3^1 \\ a_0^2 & a_1^2 & a_2^2 & a_3^2 \end{pmatrix} = \begin{pmatrix} a^{0\cdot 0} & a^{1\cdot 0} & a^{2\cdot 0} & a^{3\cdot 0} \\ a^{0\cdot 1} & a^{1\cdot 1} & a^{2\cdot 1} & a^{3\cdot 1} \\ a^{0\cdot 2} & a^{1\cdot 2} & a^{2\cdot 2} & a^{3\cdot 2} \end{pmatrix} \\ &= \begin{pmatrix} a^0 & a^0 & a^0 & a^0 \\ a^0 & a^1 & a^2 & a^3 \\ a^0 & a^2 & a^4 & a^6 \end{pmatrix} = \begin{pmatrix} 1 & 1 & 1 & 1 \\ 1 & 2 & 4 & 3 \\ 1 & 4 & 1 & 4 \end{pmatrix}. \end{aligned}$$

Exercise 14 hints at why it is convenient to use a canonical RS code. In this case the evaluation of the polynomial $A(x)$ at the positions $x = a^i$ corresponds to computing a Fourier transform and such a Fourier transform can be computed very efficiently.

4.7 The Field F_{256}

Now that we know how to construct RS codes for any given finite field F there is only one thing left to do. We want to construct the field F_{256}. This is the field that is used in most systems today. Why F_{256}? As we can see from its name, F_{256} has 256 field elements. This is convenient since $256 = 2^8$. This means that we can label the 256 field elements by 8 bit values. But 8 bits is exactly one byte, which is the fundamental information quantity in any digital system.

We will keep our exposition of F_{256} to the bare minimum. There are many beautiful properties of finite fields that you will learn at later stages of your studies. But for now we are content to learn how to construct this field and how to perform computations in it.

First note that we can not define F_{256} as the set of integers modulo 256, since 256 is not prime and so in general there is no multiplicative inverse of a given element. Instead we will use the following construction.

Definition 7 (The Construction of F_{256})**.** *Define $f(x) = x^8 + x^4 + x^3 + x^2 + 1$, where $f(x)$ is a polynomial in the indeterminant x with coefficients in F_2.*

The field F_{256} consists of the 256 distinct binary polynomials of degree at most 7, i.e., all the polynomials $\{0, 1, x, x+1, x^2, x^2+1, x^2+x, x^2+x+1, \cdots, x^7+x^6+x^5+x^4+x^3+x^2+x+1\}$. It remains to define the two basic operations, namely addition and multiplication.

Addition is the usual polynomial addition, where the binary components are added over the field F_2.

To perform a multiplication of the two polynomials $a(x), b(x) \in F_{256}$, first perform a standard polynomial multiplication. This gives lets say the polynomial $\tilde{c}(x)$ which might have degree as high as 14. Reduce this polynomial $\tilde{c}(x)$ modulo $f(x)$. This means, write $\tilde{c}(x)$ as $\tilde{c}(x) = f(x)\alpha(x) + c(x)$, where $c(x)$ is a polynomial of degree at most 7. Note that there is a unique such way of writing $\tilde{c}(x)$. Define *the result of multiplying $a(x)$ with $b(x)$ to be $c(x)$.*

All this is best understood by an example.

Example 26. *Let $a(x) = x^4 + x^2 + 1$, $b(x) = x^7 + x^4$. Then*

$$a(x) + b(x) = x^7 + x^2 + 1.$$

Note that any time we add two polynomials of degree at most 7 we get a polynomial of degree at most 7, which is again a field element.

Multiplication is a little bit more complex. To compute $a(x) \cdot b(x)$, where both are considered to be elements of the field F_{256}, we first compute the ordinary polynomial multiplication, let us denote it by $a(x) * b(x)$. We get

$$a(x) * b(x) = (x^4 + x^2 + 1) * (x^7 + x^4) = x^{11} + x^9 + x^8 + x^7 + x^6 + x^4.$$

The right hand side is not an element of F_{256} since its degree is 11. We need to "reduce" it. This means we want to subtract from it a suitable multiple of $f(x)$ so that the remainder is a polynomial of degree at most 7. This is done by what is called long division*. We have*

$$x^{11} + x^9 + x^8 + x^7 + x^6 + x^4 = (x^8 + x^4 + x^3 + x^2 + 1)(x^3 + x + 1) + (x^4 + x^3 + x^2 + x + 1).$$

Therefore, we say that $x^{11} + x^9 + x^8 + x^7 + x^6 + x^4$ modulo $f(x) = x^8 + x^4 + x^3 + x^2 + 1$ is equal to $x^4 + x^3 + x^2 + x + 1$.

Let us explain what we mean by long division in more detail. For integers one can perform division with remainder: for any two integers a and b we can find uniquely q and r such that

$a = qb + r$ with $r < b$. Similarly, given two polynomials $a(x)$ and $b(x)$, with coefficients in a field, we can find polynomials $q(x)$ and $r(x)$ with $degr(x) < degb(x)$ such that $a(x) = q(x)b(x) + r(x)$. We do not formally prove this fact but just remark that the important condition is to have coefficients of the polynomials belonging to a field. This is fulfilled for our construction of F_{256} where the coefficients of the polynomials belong to the filed F_2. Let us illustrate the process of long division on our example:

$$
\begin{array}{r|l}
x^{11} + x^9 + x^8 + x^7 + x^6 + x^4 & x^8 + x^4 + x^3 + x^2 + 1 \\
\hline
x^{11} + x^7 + x^6 + x^5 + x^3 & x^3 \\
\hline
x^9 + x^8 - x^5 + x^4 - x^3 & \\
x^9 + x^8 + x^5 + x^4 + x^3 & \quad + x \\
\hline
x^9 + x^5 + x^4 + x^3 + x & \\
\hline
x^8 - x & \\
x^8 + x & \qquad + 1 \\
\hline
x^8 + x^4 + x^3 + x^2 + 1 & \\
\hline
-x^4 - x^3 - x^2 + x - 1 & \\
x^4 + x^3 + x^2 + x + 1 &
\end{array}
$$

We see that the division operation is very similar to the one for integers.The analogy can be pushed further and one can define the greatest common divisor for two polynomials $\gcd(a(x), b(x))$ as the (monic) polynomial $c(x)$ which divides both of them and such that any other $d(x)$ which divides them also divides $c(x)$. The gcd can be found by the Euclidean algorithm:

$$
\begin{aligned}
a(x) &= q(x)b(x) + r(x) & \deg(r(x)) &< \deg(b(x)) \\
b(x) &= q_1(x)r(x) + r_1(x) & \deg(r_1(x)) &< \deg(r(x)) \\
r(x) &= q_2(x)r_1(x) + r_2(x) & \deg(r_2(x)) &< \deg(r_1(x)) \\
r_2(x) &= q_3(x)r_2(x)
\end{aligned}
$$

Since the degree of the remainder decreases the process will terminate (here we have assumed it terminates at the fouth step). This is best illustrated on our example:

$$
\begin{aligned}
(x^{11} + x^9 + x^8 + x^7 + x^6 + x^4) &= (x^3 + x + 1)(x^8 + x^4 + x^3 + x^2 + 1) + (x^4 + x^3 + x^2 + x + 1) \\
(x^8 + x^4 + x^3 + x^2 + 1) &= (x^4 + x^3 + 1)(x^4 + x^3 + x^2 + x + 1) + (x^3 + x) \\
(x^4 + x^3 + x^2 + x + 1) &= (x + 1)(x^3 + x) + 1 \\
(x^3 + x) &= (x^3 + x).1 + 0
\end{aligned}
$$

This also shows that

$$\gcd(x^{11} + x^9 + x^8 + x^7 + x^6 + x^4; x^8 + x^4 + x^3 + x^2 + 1) = 1.$$

As for integers, by reverting the process, it can be checked that the gcd can be written in the form

$$\gcd = \alpha(x)a(x) + \beta(x)b(x).$$

In the example above you can check that

$$\gcd = r_2(x) = (1 + q_2(x)q_1(x))a(x) - (q(x)(1 + q_2(x)q_1(x)) - q_2(x))b(x).$$

Most of the properties that a field has to fulfill (see the table in Section 4.1.1) are very easy to verify. The most difficult one is to ensure that every non-zero field element has an inverse with respect to multiplication. This means, we need to check that for each $a(x) \in F_{256}$ there exists a polynomial, call it $a^{-1}(x) \in F_{256}$ such that $a(x) \cdot a^{-1}(x) = 1$. One tedious, albeit straightforward way, to verify this is to multiply a given element $a(x) \in F(x)$ by all field elements $b(x) \in F(x)$ and to check that there is one such $b(x)$ so that $a(x) \cdot b(x) = 1$.

The reason why this is guaranteed to work is that the modulo operation is done with respect to an irreducible polynomial $f(x)$. Irreducible polynomials play a similar role to prime numbers. By definition we cannot decompose them into factors of smaller degree. Moreover as for integers we have the following two lemmas.

Lemma 6 (Divisor Property)**.** *If $f(x)$ is irreducible and divides the product $a(x)b(x)$ then it must divide $a(x)$ or $b(x)$ or both.*

Proof. Suppose $f(x)$ divides $a(x)$. Then we are done. If $f(x)$ does not divide $a(x)$ then $\gcd(f(x), a(x)) = 1$ since $f(x)$ is irreducible. Thus by the Euclidean algorithm there exist $\phi(x)$ and $\alpha(x)$ such that $1 = \phi(x)f(x) + \alpha(x)a(x)$. Multiplying by $b(x)$ we get

$$b(x) = \phi(x)f(x)b(x) + \alpha(x)a(x)b(x)$$

Now it suffices to note that $f(x)$ divides the right hand side of the equation (by the hypothesis) so that it must divide the left hand side also. □

Lemma 7 (Unique Factorization Property)**.** *Any polynomial with coefficients in a field can be decomposed in to a product of irreducible factors. This decomposition is unique up to the order of the factors and a choice of global coefficients for the factors.*

We do not use this lemma explicitly so we omit the proof but just point out that it is a direct corollary of the divisor property.

We are now ready to explain why it is guaranteed that each non-zero element has a unique inverse. If we are only interested in the *existence* of such an element we can proceed as follows. We first claim that there can be *at most* one such inverse. Assume to the contrary that for a given non-zero field element $a(x)$ both $b(x)$ and $c(x)$ are inverses, $b(x) \neq c(x)$. Then from $a(x) \cdot b(x) = a(x) \cdot c(x)$ we conclude that $a(x) \cdot (b(x) - c(x)) = a(x) \cdot d(x) = 0$, where $d(x)$ is a non-zero element of the field. This means that $a(x) * d(x)$ must be a multiple of $f(x)$, i.e., $a(x) * d(x) = f(x) * e(x)$ for some binary polynomial $e(x)$. From the divisor property this would imply that $f(x)$ divides $a(x)$ or $d(x)$ or both. But this is not possible since $f(x)$ is irreducible so that $\gcd(f, a) = \gcd(f, d) = 1$. This would imply that either $a(x)$ or $b(x)$ (or both) must share a factor in common with $f(x)$. That $f(x)$ is irreducible in our running example can be checked by dividing it by all polynomials of degree at most 4 (there are 32 of them)

Here comes the punch line: since by our previous argument the multiplication of a given non-zero field element $a(x)$ with the set of all field elements must each yield a distinct element of the field, one of them must be the field element 1. This proves that the inverse exists.

If we want to efficiently *compute* the inverse we use exactly the same technique as for computing the inverse of an element of F_p. This means for a given non-zero field element we use the extended Euclidean algorithm to compute two polynomials $\alpha(x)$ and $\phi(x)$ so that

$$a(x) * \alpha(x) + f(x) * \phi(x) = \gcd(a(x), f(x)) = 1.$$

where $\gcd(a(x), f(x)) = 1$ since $f(x)$ is irreducible. If we now consider this equation modulo $f(x)$ we see that $\alpha(x)$ is the sought after inverse.

Finally we explain how one can set a table in order to efficiently do operations in a field like F_{256}. But we do this for the smaller field F_{16} (note that $16 = 2^4$) to keep the discussion manageable. This field consists of all binary polynomials of degree smaller or equal to 3 $\{0, 1, x, x^2, x^3, x + 1, x^2 + 1, ...x^3 + x^2 + x + 1\}$ where multiplication is defined modulo the irreducible polynomial $f(x) = x^4 + x^3 + 1$. In this representation addition is very easy since we add the polynomials as usual by adding the coefficients modulo 2. This can also be easily in binary notation with the obvious correspondence $\{0000, 0001, 0010, 0100, 0011, 0101, ...1111\}$. However it is less convenient for multiplication or for finding an inverse. For these operations it is better to represent the field elements by $\{0, 1, x, x^2, x^3, x^4, x^5, ..., x^{14}\}$. Since we work modulo $x^4 + x^3 + 1$ we have

$$\begin{aligned}
&x^4 = -x^3 - 1 = x^3 + 1\\
&x^5 = x.(x^3 + 1) = x^4 + x = x^3 + x + 1\\
&x^6 = x^4 + x^2 + x = x^3 + x^2 + x + 1\\
&x^7 = x^4 + x^3 + x^2 + x = 2x^3 + x^2 + x + 1 = x^2 + x + 1\\
&\text{etc.}
\end{aligned}$$

You are also advised to check that $x^{15} = 1$. Therefore we can set up a table:

We can now perform multiplication and inversion easily. For example

$$\begin{aligned}
(1110).(0101) &= x^8.x^9 = x^{17} = x^2 = (0100)\\
(0111)^{-1} &= x^{-7} = x^{-7}.x^{15} = x^8 = (1110)
\end{aligned}$$

Conclusion: there is a general theory of finite fields along the lines above. In fact, one can prove that all finite fields have p^k elements where p is prime and k is an integer. Further, such a field can be constructed as the set of polynomials with coefficients in F_p modulo an irreducible (does not factor over F_p) polynomial of degree k. Addition of two polynomials is done component-wise and multiplication is regular polynomial multiplication followed by a reduction by the irreducible polynomial.

multiplication	addition	binary representation
0	0	0000
1	1	0001
x	x	0010
x^2	x^2	0100
x^3	x^3	1000
x^4	x^3+1	1001
x^5	x^3+x+1	1011
x^6	x^3+x^2+x+1	1111
x^7	x^2+x+1	0111
x^8	x^3+x^2+x	1110
x^9	x^2+1	0101
x^{10}	x^3+x	1010
x^{11}	x^3+x^2+1	1101
x^{12}	$x+1$	0011
x^{13}	x^2+x	0110
x^{14}	x^3+x^2	1100
x^{15}	1	0001

Table 4.1: Operations in F_{16}.

4.8 Bibliographical References

S. Lin and D. J. Costello, Jr., *Error Control Coding: Fundamentals and Applications.* Prentice Hall, 1983.

R. E. Blahut, *Theory and Practice of Error-Control Codes.* Addison-Wesley, 1983.

W. C. Huffman and V. Pless, *Fundamentals of Error Correcting Codes.* Cambridge University Press, 2003

R. J. McEliece, *The Theory of Information and Coding.* Cambridge University Press

4.9 Exercises

Exercise 1. Let $F = F_5$. Consider the system of linear equations

$$\begin{aligned} x_1 + 2x_2 + 3x_3 &= 2, \\ 3x_1 + 2x_2 + 4x_3 &= 1, \\ 3x_1 + x_2 + x_3 &= 0. \end{aligned}$$

a) Write this system in matrix form as $\boldsymbol{A}\boldsymbol{x}^T = \boldsymbol{u}^T$. What are $\boldsymbol{A}$, $\boldsymbol{x}$, and $\boldsymbol{u}$, and what are their dimensions?

b) Assume that you know that the determinant of $\boldsymbol{A}$ considering it as a matrix with elements over the integers is 7. What is the determinant of $\boldsymbol{A}$, considering it as a matrix over F_5?

c) Show that the system can be solved uniquely over F_5?

d) Solve the system over F_5 by using Gaussian elimination.

e) If you had to solve a $n \times n$ system. How complex is Gaussian elimination, i.e., how many elementary operations (addition, multiplication, etc.) will you need?

Hints: a) This is a 3×3 system; b) It is 2; c) $|\boldsymbol{A}| = 2 \neq 0$; d) The solution is $(1, 0, 2)$; e) n^3;

Exercise 2. Let $\mathcal{V}$ be the set formed by the vectors $\boldsymbol{v} = (v_1, \ldots, v_n)$ where v_i is equal to 0 or 1, and the field F be the Galois field F_2.

a) Show that this set is a vector space. What is its dimension? Repeat the same for v_i are taken over F_p, where p is prime.

b) Show that the *Hamming distance* between two vectors $\boldsymbol{u}$ and $\boldsymbol{v}$,

$$d(\boldsymbol{u}, \boldsymbol{v}) = \sum_{i=1}^{n} |u_i - v_i|.$$

defines a metric on this vector space.

Hint: b) We want to show that $d(\boldsymbol{u}, \boldsymbol{v}) \leq d(\boldsymbol{u}, \boldsymbol{w}) + d(\boldsymbol{w}, \boldsymbol{v})$; It suffices to consider a single component, call them $\boldsymbol{u}_i$, $\boldsymbol{w}_i$, and $\boldsymbol{v}_i$; fix $\boldsymbol{u}_i$ and consider the four possible cases where $\boldsymbol{w}_i$ and $\boldsymbol{v}_i$ are either the same or differ from $\boldsymbol{u}_i$;

Exercise 3. Let $\mathcal{S}$ be a subspace of the vector space $\mathcal{W}$ over the field F. The set

$$\mathcal{S}^{\perp} = \{\boldsymbol{w} \in \mathcal{W} : \boldsymbol{w} \bullet \boldsymbol{s} = 0 \text{ for all } \boldsymbol{s} \in \mathcal{S}\}$$

is called the *dual space*. Here, $\boldsymbol{w} \bullet \boldsymbol{s}$ is defined $\sum_i \boldsymbol{w}_i \boldsymbol{s}_i$, where all computations are done in F.

a) Show that $\mathcal{S}^{\perp}$ is a subspace.

b) Show that if $\mathcal{S}$ is a subspace of dimension k then $\mathcal{S}^{\perp}$ is a subspace of dimension $n - k$.

b) Take $F = F_2$ and $\mathcal{S} = \{0000, 0011, 1100, 1111\}$. Determine the dual $\mathcal{S}^{\perp}$.

Hint: a) Take $\boldsymbol{w}, \boldsymbol{w}' \in \mathcal{W}$. Show that $\alpha \boldsymbol{w} - \boldsymbol{w}' \in \mathcal{W}$; b) We need to show that if $\boldsymbol{A}$ is a $k \times n$ matrix, $k \leq n$, with independent rows then the set of solutions $\boldsymbol{A}\boldsymbol{w}^T = 0$ has dimension

$n-k$; bring this equation into upper triangular form by Gaussian elimination; show that you can choose the lowest $n-k$ components of $\boldsymbol{w}$ in an arbitrary manner and that the remaining components are then fixed;

Exercise 4. We consider a binary code generated by the matrix

$$\boldsymbol{G} = \begin{bmatrix} 1 & 0 & 0 & 1 & 0 & 1 \\ 0 & 1 & 0 & 0 & 1 & 1 \\ 0 & 0 & 1 & 1 & 1 & 1 \end{bmatrix}.$$

This means that

$$\mathcal{C} = \{\boldsymbol{x} : \boldsymbol{x} = \boldsymbol{u}\boldsymbol{G}, \boldsymbol{u} \in F_2^3\}.$$

a) Is the word $(1,0,1,0,1,0)$ a codeword ?

b) How many words are generated by this code? Give a list of them.

c) The first two bits of the word $(x,x,0,1,1,0)$ were deleted. What was the emitted word? What is the maximum number of binary symbols which can be removed while still being able to find the emitted codeword.

d) Let $\mathcal{C}$ be the vector space of the words generated by this code, $\mathcal{C}^{\perp}$ is its orthogonal complement; find $\mathcal{C}^{\perp}$ (give the list of all its elements.)

e) $\mathcal{C}^{\perp}$ is also a vector space. Give a basis for this space. Such a basis is usually denoted by the matrix $\boldsymbol{H}$. It is called the parity-check matrix (see Section 4.5.2).

f) Let $\hat{\boldsymbol{y}}$ be the received word. The vector given by $\boldsymbol{s} = \hat{\boldsymbol{y}}\boldsymbol{H}^T$ is called the *syndrome* (see Section 4.5.3). If the received word is $(0,0,1,1,0,1)$, what is the resulting syndrome? What is the word which was most likely emitted?

g) What is the smallest number of errors (Hamming distance) which could change one codeword into another codeword?

Hints: a) combine the first and third row of $\boldsymbol{G}$; b) 8; c) $(1,1,0,1,1,0)$; $d_{\text{min}} = 3$, $dmin - 1 = 2$, so we can tolerate at most two erasures; d+e) $\boldsymbol{G}$ is in systematic form; it is therefore easy to find $\boldsymbol{H}$; f) $(0,0,1,1,1,1)$ g) $d_{\text{min}} = 3$

Exercise 5. Let n be the blocklength. a) What are the generating matrices and parity-check matrices of a single parity-check code? b) What about a repetition code?

Hint: a) $\boldsymbol{G}$ has dimensions $(n-1)\times n$; b) $\boldsymbol{G}$ has dimensions $1 \times n$; We have not learned this in class but the two matrices are actually the dual of each other;

Exercise 6. The two words of a $(3,1)$ repetition code, $(0,0,0)$ and $(1,1,1)$, have the same probability.

a) The code works as an error detection code. if p is the probability of error on one bit, what is the probability that an incorrect word not being detected? Evaluate this probability for $p = 1/3$.

b) The code now works as an error correction code. Depending on the received word, when do we decide that the word $(0,0,0)$ was emitted? What is the probability that an incorrect word be corrected to the wrong value? Evaluate this probability for $p = 1/3$.

Exercise 7. The 16 codewords of a $(7, 4)$ code are $(0, 0, 0, 0, 0, 0, 0)$, $(1, 1, 1, 1, 1, 1, 1)$, and all cyclic permutations of $(0, 0, 0, 1, 0, 1, 1)$ and $(0, 0, 1, 1, 1, 0, 1)$.

a) What is the systematic generating matrix of this code?

b) What is the corresponding parity-check matrix?

c) What is the minimal distance of this code? What is its rate? Is this code equivalent to a $(7, 4)$ Hamming code, given in Section **??**?

Exercise 8. In practice, a systematic generating matrix of a linear code never has a column of zeros. Why?

Hint: Assume you scratch this column, i.e., assume you do not transmit the corresponding component. What changes for the receiver? Does the receiver loose any information?

Exercise 9. Take a binary block code which has m words of length n. That is, each codeword is a sequence of n bits. We suppose that this code can correct up to e errors.

a) For a codeword x, we call S the set of binary sequences for which the decoder returns x. Show that the total number of sequences contained in the set $|S|$ has the following lower bound:

$$|S| \geq \sum_{i=0}^{e} \binom{n}{i}$$

where $\binom{n}{i}$ is a binomial coefficient, giving the number of possible combinations for placing i elements in n places.

b) Show that the number M of codewords satisfies:

$$M \leq \frac{2^n}{\sum_{i=0}^{e} \binom{n}{i}}$$

This upper bound is known as the "sphere packing bound". Because it is equivalent to considering a sphere (of dimension n) with radius e, centered on each codeword and filling in the space of possible sequences without overlapping of spheres.

c) We have seen in the course (Section **??**) that Hamming codes can correct up to 1 error. Fir a bloc length of $n = 2^m - 1$ these codes contain $2^{2^m - m - 1}$ codewords. Show that Hamming codes satisfy the previous bound by equality. Such codes are called *perfect*.

Exercise 10. Show that in a linear binary code, either all the codewords contain an even number of 1's or half the codewords have an even number of 1's and the other half has an odd number of 1's.

Hint: Note that the sum of two words of odd weight gives a word of even weight, the sum of two words of even weight gives a word of even weight and the sum of a word of even weight and a word of odd weight gives a word of odd weight. Assume therefore that not all codewords have an even number of ones. Pick one codeword of odd weight, call it $\boldsymbol{w}$. Pair up all codewords. To every codeword $\boldsymbol{x} \in \mathcal{C}$ associate the codeword $\boldsymbol{x} + \boldsymbol{w}$. Argue that this proves the claim.

Exercise 11. Construct a RS code of length $n = 7$ and dimension $k = 3$ over F_7.

a) What is its generating matrix $\boldsymbol{G}$?

b) Assume that the information word is $\boldsymbol{u} = (123)$. What is the corresponding codeword?

c) Assume that a codeword is sent over the erasure channel and that you observed (64?4?0?). Can you recover the transmitted odored? If so, what is it.

Hint: a) The rows of $\boldsymbol{G}$ are $((1,2,3,4,5,6,0),(1,4,2,2,4,1,0),(1,4,2,2,4,1,0))$; b) $(6,1,6,0,4,4,0)$; c) $(6,4,1,4,6,0,0)$

Exercise 12. Consider the field F_{256} generated by the primitive polynomial $f(x) = x^8+x^4+x^3+x^2+1$. As discussed in class this means the following: the field elements are the 256 polynomials of degree at most 7 with binary coefficients. Addition is the usual polynomial addition, where the binary components are added over the field F_2. To perform a multiplication of the two polynomials $a(x)$ and $b(x)$ first perform a standard polynomial multiplication. This gives lets say the polynomial $\tilde{c}(x)$ which might have degree as high as 14. Reduce this polynomial $\tilde{c}(x)$ modulo $f(x)$. This means, write $\tilde{c}(x)$ as $\tilde{c}(x) = f(x)\alpha(x) + c(x)$, where $c(x)$ is a polynomial of degree at most 7. Note that there is a unique such way of writing $\tilde{c}(x)$. Define now the result of multiplying $a(x)$ with $b(x)$ to be $c(x)$.

a) What is $(a(x) + b(x))c(x)$, where $a(x) = 1 + x^4 + x^7$, $b(x) = x + x^2 + x^6$, and $c(x) = 1 + x^3$.

Without proof we note that the field has the property that the set $\{x^0, x^1, \cdots, x^{254}\}$ if reduced modulo the primitive polynomial $f(x)$ consists of all non-zero elements of the field and that $x^{255} = x^0 = 1$. In principle you verify this by direct computation. There is therefore a one-to-one correspondence between the non-zero field elements and the first 255 powers of x.

b) Using this property, what is x^{-247} if written as an element of F_{256}?

c) Assume you store a table in which each (non-zero) field element is written both as a polynomial $a(x)$ of degree at most 7 as well as x^i, where $0 \leq i \leq 254$. Write down the first 10 rows of this table, corresponding to the entries x^i, $0 \leq i \leq 9$.

d) Assume you want to compute the multiplication of two generic elements $a(x)$ and $b(x)$. How can you do this efficiently using the above table? Demonstrate this with the simple example $a(x) = x^3$ and $b(x) = x^5 + x^4 + x^3 + x$.

Exercise 13. Construct the RS code over F_{11} of length $n = 8$ and with $k = 4$. Use the non-zero field elements $x_i = i$, $i = 1, \cdots, 8$ for the construction.

a) Write down the corresponding generator matrix $\boldsymbol{G}$.

b) Assume that you want to transmit the information vector $\boldsymbol{u} = (1407)$. What is the corresponding codeword?

c) Assume you received the word (62?90???). You want to determine the received codeword. Write down the corresponding system of equations. Now use the Gaussian elimination algorithm to recover the transmitted information. What was the transmitted codeword?

Exercise 14 In the first module of this course you have encountered the N-point Fourier transform over the complex numbers. Fourier transforms play a fundamental part in much of signal processing and you will encounter them as a basic tool throughout your studies. This exercise will show you in terms of a concrete example that the concept of Fourier transforms is very general.

a) Consider the field F_7. Compute explicitly the set $\{3^i \mod 7\}$, $i = 0, \cdots, 5$. What is 3^6 mod 7? A field element a such that its powers a^i, $i = 0, \cdots, |F| - 2$, cover all the non-zero elements of the field is called a *generator*. We have seen already in Exercise 12 that x is a generator for the field F_{256} produced by the primitive polynomial $f(x) = x^8 + x^4 + x^3 + x^2 + 1$.

b) Consider a vector $\boldsymbol{u}$ of length 6 whose components are elements of F_7. Define the following "Fourier Transform Pair" between the vectors $\boldsymbol{u}$ and $\hat{\boldsymbol{u}}$ of length 6 with components in F_7. We have

$$\hat{\boldsymbol{u}}_i = \sum_{j=0,\cdots,5} \boldsymbol{u}_j 3^{ij}, \text{Fourier transform}$$
$$\boldsymbol{u}_j = 6 \sum_{i=0,\cdots,5} \hat{\boldsymbol{u}}_i 5^{ij}, \text{inverse Fourier transform}$$

where all computations are done over F_7. Note that 5 is the multiplicative inverse of 3.

c) Start with $\boldsymbol{u} = (123456)$. Compute $\hat{\boldsymbol{u}}$. Then verify that you get back $\boldsymbol{u}$ by computing the inverse transform.

d) Compute the cyclic convolution of the two vectors $\boldsymbol{u} = (123456)$ and $\boldsymbol{v} = (121212)$. Then compute the Fourier transform of the result. Alternatively, compute the Fourier transform of $\boldsymbol{u}$ and $\boldsymbol{v}$ and then take the component-wise multiplication. Any comments?

e) (Bonus Question) Can you show in general that if you start with $\boldsymbol{u}$ and compute first $\hat{\boldsymbol{u}}$ by applying the Fourier transform and then compute the inverse Fourier transform of $\hat{\boldsymbol{u}}$ that you get back $\boldsymbol{u}$?

Exercise 15 Consider a binary linear code $\mathcal{C}$ of length n and dimension k. This means that $\mathcal{C}$ is a subspace of $\{0, 1\}$ of dimension k. Let $d_{\min}$ be the minimum distance of the code $\mathcal{C}$.

a) Prove the following fundamental inequality due to Singleton

$$d \leq n - k + 1.$$

Hint: Write down the 2^k codewords in a $(2^k) \times n$ binary matrix. Delete all except the first k columns of this matrix. Note that the distance between two codewords is the sum of the difference among the first k components and the difference among the last $(n - k)$ components.

b) Give an example of a code which achieves the preceding upper bound with parameters n and $k = 1$.

Hint: repetition code