Applied and industrial algebra

Tim Hsu, San José State University

May 12, 2018

Contents

Introduction

(definition of applied math)
drives topics

- Cryptography
- Euclidean algorithm
- Error-correcting codes
- Finite fields
- Discrete Fourier Transform and Fast Fourier Transform

informed consumer
ulterior motive

$$\text{Abstraction} \Rightarrow \text{Simplifcation} \Rightarrow \text{Generalization} \Rightarrow \text{Power}$$

outline of rest of book
there are 80 problems.

Chapter 1

Numbers, polynomials, rings, and fields

(chapter intro)

1.1 Numbers

We begin by specifying what we will assume as background knowledge.

We assume you are familiar with the *natural numbers* and the integers:

$$\mathbf{N} = \{1, 2, 3, \dots\}, \qquad \mathbf{Z} = \{\dots, -2, -1, 0, 1, 2, \dots\}. \tag{1.1.1}$$

Note that we use the convention of starting $\mathbf{N}$ with 1 instead of starting with 0.

You should also be familar with the *rational numbers:*

$$\mathbf{Q} = \left\{ \frac{k}{n} \middle| \, k, n \in \mathbf{Z}, \ n \neq 0 \right\}. \tag{1.1.2}$$

In other words, the rationals $\mathbf{Q}$ are precisely all numbers formed as the legitimate (thus the condition $n \neq 0$) ratio of two integers. The fussy reader may note that (1.1.2) is not a definition *per se*, just a list of the numbers within the real numbers that happen to be rational — to which we reply, buckle up, friend, you're going to find this book to be pretty un-fussy by your standards. (Did we mention that this is a book for the enlightened *consumer* of applied and industrial algebra?)

Speaking of the real numbers, in this book, we will all agree to pretend that we know what the *real numbers* $\mathbf{R}$ are. Which is the sort of thing that is obvious unless you think about it, right? I mean, you've used them all your life, you actually used deep properties about them when you took calculus, they form the number line, they're all possible numbers that can be expressed as a decimal, finite or infinite... sure, that all sounds fine! So we'll just all agree to avoid unpleasant questions like "What is the actual, precise definition of the real numbers?".*

*This is the heart of introductory analysis, one of the more difficult classes in the undergraduate math curriculum.

The first number system we will really use that may be somewhat new to you is the complex numbers $\mathbf{C}$. One way to think of complex numbers is that they consist of all expressions of the form $a + bi$, where a and b are real numbers, and i is a new (to-be-defined) symbol:

$$\mathbf{C} = \{a + bi \mid a, b \in \mathbf{R}\}. \tag{1.1.3}$$

Addition and multiplication of elements of $\mathbf{C}$ is then defined to be multiplication of "polynomials in i", with the additional rule that $i^2 = -1$. For example:

$$\begin{aligned}(3 + 7i)(4 - 5i) &= 3(4) - 3(5i) + (7i)(4) - (7i)(5i) \\ &= 12 - 15i + 28i - 35(i^2) \\ &= 12 - 15i + 28i + 35 \\ &= 47 + 13i.\end{aligned} \tag{1.1.4}$$

For the fussy reader, if you accept the existence of the real numbers, we can define a complex number to be a pair (a, b) of real numbers $a, b \in \mathbf{R}$, with addition and multiplication defined by

$$(a, b) + (c, d) = (a + c, b + d), \qquad (a, b) \cdot (c, d) = (ac - bd, ad + bc), \tag{1.1.5}$$

or in other words,

$$(a + bi) + (c + di) = (a + c) + (b + d)i, \tag{1.1.6}$$

$$(a + bi)(c + di) = (ac - bd) + (ad + bc)i. \tag{1.1.7}$$

In practice, the fussy definition is no more enlightening than the unfussy one, but at least we can be reassured that the complex numbers are no less rigorous than the real numbers.

Another way to look at complex numbers that will be very useful to us is to draw them in the *complex plane*. As shown in Figure 1.1.1, the idea is that we draw $a + bi$ as the point (a, b) in the xy-plane. This ties naturally into two other operations one can do on a complex number $a + bi$, namely, the *modulus*, or *absolute value*, of $a + bi$:

$$|a + bi| = \sqrt{a^2 + b^2}; \tag{1.1.8}$$

and the (complex) *conjugate* of $a + bi$:

$$\overline{a + bi} = a - bi. \tag{1.1.9}$$

Note that if $a \in \mathbf{R}$, then the modulus $|a + 0i| = \sqrt{a^2}$ is the usual real absolute value of a; in other words, the modulus is a generalization of the real absolute value of a number.

The modulus and the conjugate have the following algebraic properties.

Theorem 1.1.1. *For $z, w \in \mathbf{C}$, we have that*

1. $|zw| = |z|\,|w|$;

2. $z\overline{z} = |z|^2$;

3. $|z| \geq 0$, *and* $|z| = 0$ *if and only if* $z = 0$; *and*

4. *If* $z \neq 0$, *then*

$$z\left(\frac{\overline{z}}{|z|^2}\right) = 1. \tag{1.1.10}$$

We'll see that the above property that is most useful to us is (1.1.10).

Proof. Problem 1.1.4. □

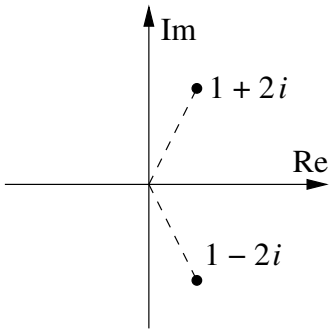

Figure 1.1.1: Two conjugate complex numbers

The modulus and the complex conjugate also have natural geometric interpretations in the complex plane, as shown in Figure 1.1.1: The modulus of $a + bi$ can be pictured as the distance from the point $a+bi$ to the origin, and the conjugate can be pictured as the mirror image of $a + bi$ in the real axis (x-axis).

Taking things up a notch, we will take it as a fact[†] that for any $z \in \mathbf{C}$, the *complex exponential* e^z can be defined starting with *Euler's formula*

$$e^{ix} = \cos x + i \sin x, \tag{1.1.11}$$

where x is any real number, and cos and sin are calculated in radians (as any good calculus student knows). As special cases of (1.1.11), we have

$$e^{\pi i} = -1, \qquad\qquad e^{2\pi i} = 1. \tag{1.1.12}$$

(The part of (1.1.12) is sometimes known as *Euler's identity.*) Since we will also take it as fact that the usual exponential law

$$e^{z+w} = e^z e^w \tag{1.1.13}$$

[†]A fact proven in analysis, most likely in a second course. Did we mention that analysis is difficult?

holds for all $z, w \in \mathbf{C}$, we can extend Euler's formula (1.1.11) to give

$$e^{x+yi} = e^x e^{iy} = e^x \cos y + (e^x \sin y)i \tag{1.1.14}$$

for any $x + yi \in \mathbf{C}$, where e^x is the usual real exponential function.

We note that the identity known as *de Moivre's theorem* is a special case of applying the usual exponential laws to complex exponentials; that is, for $n \in \mathbf{N}$ and $x \in \mathbf{R}$,

$$(\cos x + i \sin x)^n = (e^{ix})^n = e^{inx} = \cos(nx) + i \sin(nx). \tag{1.1.15}$$

Again, in this book, we will take it as given that this all works; we hope you find it plausible, or at least internally consistent.

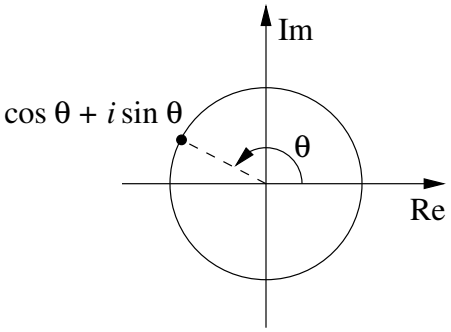

Figure 1.1.2: Complex exponentials on the unit circle

Complex exponentials also have the following geometric interpretation. Drawing (1.1.11) in the complex plane, we see that the complex numbers of the form $e^{i\theta}$, $\theta \in \mathbf{R}$, are precisely the points on the unit circle, with θ being the angle of $\cos\theta + i \sin\theta$ in the usual sense; see Figure 1.1.2. More generally, for an arbitrary complex number $a + bi \neq 0$, if $r = |a + bi|$ is the modulus of $a + bi$, then $\dfrac{a + bi}{r}$ is a point $e^{i\theta}$ on the unit circle, which means that

$$a + bi = re^{i\theta}. \tag{1.1.16}$$

Here θ is called the *argument* of $a + bi$, and (1.1.16) is called the *modulus-argument* representation of $a+bi$. The argument θ can be interpreted as the angle (again, in radians) going from the positive x-axis to the ray from the origin through $a + bi$; again, see Figure 1.1.2. Note that much like, for example, angles in polar coordinates, changing the argument of a complex number z from θ to $\theta + 2\pi$ does not change z at all; in other words, the argument of z is defined only up to adding multiples of 2π.

The reason we are so interested in complex numbers, especially complex exponentials, is the answer to the following question.

Question 1.1.2. What are the solutions $z \in \mathbf{C}$ to the equation $z^n = 1$?

In fact, we find the answer to Question 1.1.2 so interesting, we introduce the following terminology.

Definition 1.1.3. For $n \in \mathbf{N}$, we define an *nth roots of unity* to be a complex number z such that $z^n = 1$. Note that an nth root of unity is always an nth root of unity for infinitely many n; for example, 1 itself is an nth root of unity for all $n \in \mathbf{N}$. We therefore define a *primitive nth root of unity* to be an nth root of unity z such that $z^k \neq 1$ for any $k \in \mathbf{N}$, $k < n$.

For example, when $n = 2$, the nth roots of unity are $z = \pm 1$, and the only primitive nth root of unity is $z = -1$. When $n = 4$, the nth roots of unity are $z = 1, i, -1, -i$, and the primitive nth roots of unity are $z = \pm i$. For more about roots of unity, see Problems 1.1.1–1.1.3.

Problems

1.1.1. (a) What is $(e^{2\pi i/3})^3$? $(e^{4\pi i/3})^3$? Can you find all three 3rd roots of unity?

(b) Can you find one solution $z \in \mathbf{C}$, $z \neq 1$, to the equation $z^5 = 1$? How about others? How many can you find?

(c) Can you guess what all solutions to $z^n = 1$ are for general $n \in \mathbf{N}$?

1.1.2. (a) Find all solutions $z \in \mathbf{C}$ to the equation $z^6 = 1$. Which of these are primitive 6th roots of unity (Definition 1.1.3)?

(b) Same, but for $z^{15} = 1$.

(c) Can you guess which nth roots of unity are primitive roots of unity for general $n \in \mathbf{N}$?

1.1.3. Draw all solutions $z \in \mathbf{C}$ to the equation $z^8 = 1$ in the complex plane. What is the analogy of this picture for general n?

1.1.4. Let $z = a + bi$ and $w = c + di$ be complex numbers.

(a) Prove that $|zw| = |z|\,|w|$.

(b) Prove that $|z|^2 = z\overline{z}$.

(c) Prove that $|z| \geq 0$, and $|z| = 0$ if and only if $z = 0$. (Suggestion: You can use the fact that if $a \in \mathbf{R}$, then $a^2 \geq 0$, and $a^2 = 0$ if and only if $a = 0$.)

(d) Prove that if $z \neq 0$, then $z\left(\dfrac{\overline{z}}{|z|^2}\right) = 1$.

1.2 Greatest common divisors

The first motivating problem we consider in this book comes from the following ideas.

Definition 1.2.1. For $a, d \in \mathbf{Z}$, to say that d *divides* a means that $a = qd$ for some $q \in \mathbf{Z}$. (The letter q is chosen to remind you that if $d \neq 0$, then q is the quotient $\dfrac{a}{d}$, and to say that d divides a is to say that $\dfrac{a}{d}$ is an integer.)

Definition 1.2.2. Let a and b be natural numbers. We define a *common divisor* of a and b to be an integer d that divides both a and b. We define the *greatest common divisor*, or *gcd*, of a and b to be the largest natural number that is a common divisor of a and b. We sometimes denote the gcd of a and b by $\gcd(a, b)$, or even by (a, b).

Note that later, we will slightly modify the definition of $\gcd(a, b)$ so it generalizes better; see (blah). In any case, we may now state our first motivating problem.

Motivating Problem 1.2.3. Given $a, b \in \mathbf{N}$, how can we compute $\gcd(a, b)$ efficiently?

The money-making aspect of Motivating Problem 1.2.3 may not be clear, but for now, we hope you can at least accept it as a toy problem that is not too difficult to explain; we'll see later that it *is* actually a money-making algorithm, or at least sets a standard for what it means for a certain class of algorithms to be money-making.

The key word in Motivating Problem 1.2.3 is *efficiently*, since there are lots of simple but inefficient ways to compute $\gcd(a, b)$. For example:

1. By trying the divisors $1, \ldots, a - 1$, find all positive divisors of a.
2. By trying the divisors $1, \ldots, b - 1$, find all positive divisors of b.
3. The largest number on both lists is $\gcd(a, b)$.

For a slightly more efficient version, assuming $a \leq b$:

1. By trying the divisors $1, \ldots, a - 1$, find all positive divisors $d_1, \ldots, d_n$ of a.
2. See which of the numbers $d_1, \ldots, d_n$ divide b.
3. The largest number d_i dividing b is $\gcd(a, b)$.

The problem with both of these naive algorithms is that they could both potentially require roughly $2b$ or $2a$ trial divisions. In computer science terminology that we will explain in more detail later, if we start knowing only that $1 \leq a, b \leq n$, then these algorithms require $O(n)$ trial divisions, meaning there exists some constant C such that no more than Cn trial divisions will be needed to complete the algorithm ($C = 2$ is a natural choice here).

Now, an $O(n)$ algorithm may seem reasonable if $n = 100$ or $n = 1000$. However, if $n = 10^{100}$ or 10^{1000}, as it is in applications, what we actually need is something more like an $O(\log n)$ algorithm, that is, an algorithm that requires no more than $C \log n$ trial divisions to finish for some constant C. Describing that algorithm, and its generalizations via abstract algebra, is the main goal of this chapter.

Problems

1.2.1. Can you improve the naive gcd algorithms listed above so they finish in at most $O(\sqrt{n})$ divisions? That's still not much help when $n = 10^{1000}$, but is certainly better than $O(n)$. (Suggestion: If $n = ab$, $a, b, n \in \mathbf{N}$, one of a or b has to be greater than or equal to the other.)

1.2.2. (try some naive gcd computations)

1.3 The integers mod n, informally

In this section, we give an informal introduction to the integers mod n. Rest assured, we will later define and discuss the integers mod n formally and precisely, but that will require a fair amount of theory, and the integers mod n are too useful for what we're doing not to introduce them early on.

One classic motivation for modular arithmetic comes from "clock arithmetic." That is, when you're counting time by the hour, the time goes from 11 o'clock, to 12 o'clock, and then back to 1 o'clock. Mathematically, we can think of this as saying that $13 = 1$ (in o'clocks), or better yet, $12 = 0$. Another motivation comes from days of the week: If we number Sunday $= 0$, Monday $= 1$, and so on to Saturday $= 6$, we see that the day after Saturday is $6 + 1 = 0$ (in days of the week). In other words, when counting days of the week, we set $7 = 0$.

Along these lines, we can think of arithmetic (mod n) as being "what happens to the integers when we set $n = 0$." To be somewhat (though not completely) more precise:

Not Quite a Definition 1.3.1. We define the *integers mod* n, denoted by $\mathbf{Z}/n$, to be the following number system:

- The underlying set of $\mathbf{Z}/n$ is $\{0, 1, 2, \ldots, n-1\}$.
- We define addition in $\mathbf{Z}/n$ to be addition "reduced mod n". That is, to add a and b in $\mathbf{Z}/n$, calculate $a + b$ as an integer, and define $a + b$ as an element of $\mathbf{Z}/n$ to be the remainder r that you get when you divide $a + b$ by n:

$$(a + b) = qn + r. \tag{1.3.1}$$

- Similarly, to multiply a and b in $\mathbf{Z}/n$, calculate ab as an integer, and define ab as an element of $\mathbf{Z}/n$ to be the remainder r that you get when you divide ab by n:

$$ab = qn + r. \tag{1.3.2}$$

Notation 1.3.2. We see in Not Quite a Definition 1.3.1 that, working mod n, all that really matters about an integer a is the remainder that you get when you divide a by n. We therefore define the statement $a \equiv b \pmod{n}$ to mean that a and b have the same remainder when you divide them by n, or in other words, that a and b differ by a multiple of n. For example, in this notation, (1.3.1) and (1.3.2) become

$$a + b \equiv r \pmod{n}, \qquad ab \equiv r \pmod{n}, \tag{1.3.3}$$

respectively, illustrating the "set $n = 0$" description of arithmetic mod n.

Example 1.3.3. The elements of $\mathbf{Z}/17$ are $\{0, 1, \ldots, 16\}$, and mod 17, we have:

$$5 \equiv 22 \equiv 39 \equiv 170000039 \pmod{17}. \tag{1.3.4}$$

Here are some sample calculations in $\mathbf{Z}/17$:

$$13 + 16 = 29 \equiv 12 \pmod{17}, \tag{1.3.5}$$

$$7(11) = 77 \equiv 9 \pmod{17}. \tag{1.3.6}$$

If you put in a little practice, you may start to see that you can do addition and multiplication in $\mathbf{Z}/n$ using only ordinary $+$ and $\times$ and the division theorem, so operationally, mod n arithmetic is not that big a deal. On the other hand, if you think a bit about what you're doing (always dangerous), it may become less and less clear that mod n arithmetic is consistent, or that it satisfies the usual properties of arithmetic. We will return to this last point later.

Digression. One amusing and (sort of) practical application of mod 7 arithmetic, due to the mathematician John H. Conway[‡], starts with the following observation.

> In any given year, the days April 4, June 6, August 8, October 10, and December 12 (i.e., 4/4, 6/6, 8/8, 10/10, and 12/12) all fall on the same day of the week.

Check a calendar and see! Then for any given year, we define the *Doomsday* for that year to be the day of the week on which those dates all fall. Conway's algorithm for determining the day of the week then goes roughly as follows.

1. Figure out a mnemonic (memorization trick) for remembering one Doomsday in each month. The above observation takes care of all even-numbered months except February; for the odd months, the phrases "seven-eleven" and "nine to five job" remind us 7/11 (July 11), 11/7, 5/9, and 9/5 are all Doomsdays, and 3/7 you just have to memorize. Because of leap years, January and February are trickier; one mnemonic is that the last day of February, either 2/28 in a non-leap year or 2/29 in a leap year, as is the last day of January — as long as we define the last day of January in a leap year to be January 32 (or equivalently mod 7, January 25, 18, 11, or 4).

2. Suppose we know which day of the week Doomsday is for a given year; for example, in 2018 (the time of this writing), Doomsday is a Wednesday, or 3 (mod 7) in our coding. Then, taking June for concreteness, the day of the week on which June the nth falls is $3 + (n - 6)$ (mod 7). For example, since

$$3 + (24 - 6) = 21 \equiv 0 \pmod{7}, \tag{1.3.7}$$

   June 24, 2018 is a Sunday.

In practice (and the author does use this occasionally in real life), it's probably better to think of the above idea less formulaically, and more just in terms of the idea that multiples of 7 are the same as 0. For example, for June 24, 2018, it's simpler to remember that since 21 is a multiple of 7, $24 \equiv 3 \pmod{7}$, and 3 is 3 less than 6, so the day of the week is 3 before Wednesday, or Sunday.

[‡]Who is, full disclosure, the author's Ph.D. advisor.

Of course, this idea goes being finding days of the week in whatever year it is now. Conveniently, since $365 \equiv 1 \pmod 7$, the value of Doomsday increases by 1 each non-leap year, and by 2 each leap year (since $366 \equiv 2 \pmod 7$), so for example, Doomsday is Thursday in 2019, and Saturday (+2) in the leap year 2020. Beyond that, since any 12-year span not crossing a century contains exactly 3 leap years, every 12-year span within a century advances Doomsday by

$$9 + 3(2) = 15 \equiv 1 \pmod 7. \tag{1.3.8}$$

(Conway mnenomic: "A dozen years is but one day".) And then we get to memorizing century days, and the turn-of-the-century rules for leap years, and the Julian vs. the Gregorian calendars.... But that's enough for now.

Problems

1.3.1. 265 hours from now.

1.3.2. 2018: Doomsday is a Wednesday.

(a) random days of the year

1.3.3. find dates for this year

1.3.4. Casting nines problem

1.4 Complexity and big-O notation

We come to a key idea of this book. Since many, or maybe most, of the problems we consider can be solved by relatively simple "no-brainer" methods that are too slow to work in practice, what makes the algorithms we discuss useful is the fact that they are faster than the corresponding no-brainer method. We can sometimes measure this kind of speed-up by doing computer experiments, but it helps to have some conceptual way to discuss the speed of an algorithm.

One way to describe the speed of an algorithm comes from describing the time $T(n)$ the algorithm takes to finish a calculation, given an input of size n. More precisely, the following terminology can be used to give both a quantitative and a qualitative description of such a "worst-case time" function $T(n)$.

Definition 1.4.1. Let $T(n)$ and $f(n)$ be real-valued functions with domain the natural numbers $\mathbf{N}$ (in fancy function notation, $T : \mathbf{N} \to \mathbf{R}$ and $f : \mathbf{N} \to \mathbf{R}$). To say that $T(n) = O(f(n))$ means that there exists some constant C such that $T(n) \leq Cf(n)$ for all $n \in \mathbf{N}$. The notation $O(f(n))$ is known as *big O notation.*

To make big O notation truly useful, we need to understand some standard "comparison functions" $f(n)$ that we can use to describe our worst-case time functions $T(n)$. More specifically, the following idea helps in sorting out the relative sizes of comparison functions.

Definition 1.4.2. Suppose $f(n)$ and $g(n)$ are real-valued functions on n. To say that $f(n) << g(n)$ means that

$$\lim_{n\to\infty} \frac{f(n)}{g(n)} = 0. \tag{1.4.1}$$

Now, (1.4.1) is written in a mathematically precise way, but even if you don't remember exactly what $\lim_{n\to\infty}$ means, hopefully you remember enough about it to see that the point of (1.4.1) is that as n gets very large, $g(n)$ is far bigger than $f(n)$. With that mind, we come to the following theorem.

Theorem 1.4.3 (The Asymptotics Theorem)**.** *For fixed $0 < p$ and $a > 1$, we have that*

$$\log n << n^p << a^n << n!. \tag{1.4.2}$$

In other words, in terms of growth as n approaches $+\infty$, logarithmic growth ($\log n$) is much less than polynomial growth (n^p) is much less than exponential growth (a^n) is much less than combinatorial growth ($n!$). An algorithm with worst-case finishing time $T(n) = O(\log n)$ is called a *logarithmic-time* algorithm, and *polynomial-time*, *exponential-time*, and *combinatorial-time* algorithms are defined similarly.

The worst-case time $T(n)$ of an algorithm is called its *complexity*, and the study of algorithm speed is known as *complexity theory*. In complexity theory, we often consider problems solvable in polynomial time ($T(n) = O(n^p)$) to be solvable in practice, whereas problems only solvable in exponential or combinatorial time are not. Note, however, that for industrial purposes, even an $O(n^4)$ or $O(n^3)$ algorithm may be too slow, depending on how big n is.

We next come to an idea that is often helpful in studying the complexity of an algorithm.

Theorem 1.4.4 (The Addition Principle)**.** *If $f(n) << g(n)$, then $f(n) + g(n) = O(g(n))$.*

In other words, the slowest part of an algorithm dominates its runtime, so you can ignore the faster parts.

Example 1.4.5. We finally get around to explaining why the Euclidean Algorithm for integers runs in $O(\log n)$ time. The key point is that (Problem 1.4.1) at division step n:

$$r_{n-2} = q_n r_{n-1} + r_n, \qquad 0 \le r_n < r_{n-1}, \tag{1.4.3}$$

we have that $r_n \le \dfrac{r_{n-2}}{2}$. If we then assume, for simplicity, that the algorithm always finishes with the last nonzero remainder r_N after an even number of steps ($N = 2k$), we see that

$$1 \le r_N \le \frac{r_0}{2^k}. \tag{1.4.4}$$

Since this holds for every k, it follows that if the algorithm finishes in more than $2K$ steps, then we must have $r_0 > 2^K$, or taking the contrapositive, if $r_0 \le 2^K$, the algorithm must finish in at most $2K \ge 2\log_2(r_0)$ steps.

For more examples of runtime estimates, see the problems below.

Remark 1.4.6. Note that by the change of base formula for logs, $O(\log_a n)$ has the same meaning for any base $a > 1$.

Problems

1.4.1. Suppose $a, d \in \mathbf{N}$, and we divide a by d as per the Division Theorem:

$$a = qd + r, \qquad 0 \leq r < d. \tag{1.4.5}$$

Explain why (prove that) $r \leq \frac{a}{2}$. (Suggestion: Either $d \leq \frac{a}{2}$ or $d > \frac{a}{2}$.)

1.4.2. sum of $1^k, 2^k, \ldots, n^k$. lower and upper bounds.

1.4.3. n days of christmas

1.4.4. writing down a number

1.4.5. dot products

1.4.6. cross product

1.4.7. insertion sort

1.4.8. merge sort

1.4.9. count permutations

1.4.10. multiply two polynomials

1.4.11. determinants by minors

1.4.12. gaussian reduction

Chapter 2

Cheaper: Error-correcting codes

2.1 The idea of an error-correcting code

Let's start by stating the main problem that error-correcting codes solve in vague terms that you may not know; then we'll gradually fill those terms in.

Motivating Problem 2.1.1. Suppose we are transmitting a message over a noisy channel. Is there some way that we can detect, or better yet, correct errors that occur in transmission?

By the term *message* in Motivating Problem 2.1.1, we just mean a finite sequence, or *string*, of letters. More conveniently, we can encode letters as numbers by some agreed-upon scheme (e.g., ASCII), so we may as well consider messages that are sequences of numbers. In fact, by writing those numbers in binary, we can restrict our attention to strings of binary digits, or *bits*, break those *bitstrings* into blocks of some fixed length n, and consider what happens one block at a time.

Next, the term *noisy channel* in Motivating Problem 2.1.1 refers to any means of communication that might change some of the 0's in a bitstring to 1's, and vice versa. In other words, an *error* in the transmission of a block of n bits is a sent 0 that is received as a 1, or vice versa.

We therefore come to the following more specific version of Motivating Problem 2.1.1.

Motivating Problem 2.1.2. Suppose we are transmitting a bitstring of length n and one or more errors occurs in transmission. Is there some way that we can detect that an error or errors has occurred? Better yet, is there some way that we can correct an error or errors?

In the terms of Motivating Problem 2.1.2, the basic idea of *error-correcting codes* is that we can detect, or even correct, transmission errors by having both sender and receiver agree ahead of time that only certain bitstrings of length n are valid messages, or *codewords*. The reason for choosing only certain bitstrings to be codewords is that if a non-codeword is received in transmission, the receiver will know that something has gone wrong. Moreover, if we choose our code very carefully, the receiver may actually be able to figure out which codeword the sender most likely intended.

For example, if we have n data bits $x_1, \ldots, x_n$ to transmit, we can add a *parity check* bit

$$x_0 = x_1 + \cdots + x_n \pmod 2 \tag{2.1.1}$$

to our message, and transmit the bitstring $(x_0, x_1, \ldots, x_n)$. In other words, the codewords in the *parity check code of length* $n+1$ are all bitstrings $(x_0, x_1, \ldots, x_n)$ such that

$$x_0 + x_1 + \cdots + x_n = 0 \pmod 2. \tag{2.1.2}$$

Now, suppose sender and receiver agree ahead of time to use the parity check code of length $n+1$, and in transmission, bit x_n is flipped from 0 to 1, or 1 to 0. Since $1 + 1 = 0 \pmod 2$, we can model this one-bit error by adding 1 to $x_n \pmod 2$; that is, the receiver will receive the bitstring $(x_0, x_1, \ldots, x_n + 1)$. However, if the receiver then sums this bitstring, they get

$$x_0 + x_1 + \cdots + x_{n-1} + (x_n + 1) = 1 \pmod 2, \tag{2.1.3}$$

and comparing (2.1.2), the receiver will then know that an error has occurred. Note, however, that the receiver will not know which bit has been flipped, because flipping any bit x_i (including x_0) has the same effect on the sum as flipping x_n. Note also that if *two* errors occur in transmission, everything will appear to be fine; more generally, any odd number of errors can be detected, and any even number of errors cannot be.

In this chapter and in subsequent chapters, we will study (e.g., in Section 2.7) efficient methods for transmitting bitstrings with error correction. However, as a benchmark, we should note that there is a simple, if inefficient, way to transmit bits with error correction, namely, *repetition codes*. Suppose we want to transmit a single bit x. In the repetition code of length 3, we simply repeat our data bit 3 times. (We said this would be inefficient!) In other words, to send the message bit 0, we transmit the bitstring 000, and to send the message bit 1, we transmit 111. And indeed, this code corrects one error: If we transmit 000 and the receiver receives (say) 010, as long as no more than one error has occurred, the receiver can conclude that the original message was 000. (This procedure is known as *majority logic* decoding.)

So the question arises, can we do better? That is:

Motivating Problem 2.1.3. Can we transmit bitstrings in blocks of some length n, with one error-correction per block, at a cost of less than 3 transmitted bits per 1 message bit?

Motivating Problem 2.1.3 also explains the title of this chapter: The point is not just that we can do error-correction in communication, but as we'll see by the end of this chapter, we can do it using fewer transmitted bits per message bit, i.e., cheaper.

Problems

2.1.1. $(5, 1)$ repetition: how many errors corrected? $(6, 1)$ how many corrected and detected? Pattern?

2.2 Matrices with entries in a field F

Sections 2.2–2.3 form a crash course in linear algebra over fields. Now, if you don't feel like reading these sections, you can make your own textbook on the same material using the following simple recipe:

1. Find your favorite linear algebra textbook.
2. Everywhere you see an $\mathbf{R}$, representing the real numbers, replace it with F, representing an arbitrary field.

Congratulations: Now you have a textbook for linear algebra over an arbitrary field F!

More seriously, that's not quite completely true, but it ends up being surprisingly close to being true — the basic ideas really are quite independent of the particular field F. Nevertheless, we'll start from the beginning.

$\vdots$

(Matrices over fields: See textbook, sections 8A–8B, and just always assume we have a field instead of an arbitrary ring R.)

$\vdots$

In particular, $n \times 1$ matrices are called *column vectors*, and $1 \times n$ matrices are called *row vectors*. If we want to refer to either column or row vectors, but not specify which, we simply talk about *vectors*. Depending on context, the *zero vector* $\mathbf{0}$ either denotes the column vector $\begin{bmatrix} 0 \\ \vdots \\ 0 \end{bmatrix}$ or the row vector $[0 \ \cdots \ 0]$.

We now come to our first key concept.

Definition 2.2.1. Let F be a field and $n \in \mathbf{N}$. We define F^n to be the set of all $n \times 1$ column vectors with entries in F. In other words,

$$F^n = \left\{ \left. \begin{bmatrix} x_1 \\ \vdots \\ x_n \end{bmatrix} \right| x_i \in F \right\}. \tag{2.2.1}$$

Equivalently,

$$F^n = \left\{ \begin{bmatrix} x_1 & \cdots & x_n \end{bmatrix} \mid x_i \in F \right\}. \tag{2.2.2}$$

(We hope even the fussiest reader will accept that in this context, row and column vectors are just different notations for the same idea.)

$\vdots$

We come to a crucial observation (Remark 2.2.3) that requires the following definition to explain.

Definition 2.2.2. Let $\mathbf{a}_1, \ldots, \mathbf{a}_n$ be column vectors in F^k. An an *F-linear combination*, or if F is understood, simply a *linear combination*, of the vectors $\mathbf{a}_1, \ldots, \mathbf{a}_n$ is an expression of the form

$$c_1\mathbf{a}_1 + \cdots + c_n\mathbf{a}_n, \tag{2.2.3}$$

where $c_1, \ldots, c_n \in F$.

Remark 2.2.3. Perhaps the most important way of looking at matrix-vector multiplication is as follows: If A is a $k \times n$ matrix whose n columns are the $k \times 1$ column vectors $\mathbf{a}_1, \ldots, \mathbf{a}_n$, and $\mathbf{x} = \begin{bmatrix} x_1 \\ \vdots \\ x_n \end{bmatrix}$ is an $n \times 1$ column vector, then

$$A\mathbf{x} = x_1\mathbf{a}_1 + \cdots + x_n\mathbf{a}_n. \tag{2.2.4}$$

Comparing (2.2.3) and (2.2.4), we see that $A\mathbf{x}$ is the linear combination of the columns of A with coefficients given by the entries of $\mathbf{x}$. This idea comes in very handy later; see Section 2.4.

$\vdots$

2.3 Linear equations over a field F

Throughout, let F be a field.

Definition 2.3.1. A *linear equation* is an equation of the form

$$a_1x_1 + \cdots + a_nx_n = b, \tag{2.3.1}$$

where $a_i, b \in F$ are constants, and the x_i are the unknowns. A *system of k linear equations in n variables* has the form

$$\begin{aligned} a_{11}x_1 + \cdots + a_{1n}x_n &= b_1, \\ &\vdots \\ a_{k1}x_1 + \cdots + a_{kn}x_n &= b_k. \end{aligned} \tag{2.3.2}$$

Note that if

$$A = \begin{bmatrix} a_{11} & \cdots & a_{1n} \\ \vdots & \ddots & \vdots \\ a_{k1} & \cdots & a_{kn} \end{bmatrix}, \ \mathbf{x} = \begin{bmatrix} x_1 \\ \vdots \\ x_k \end{bmatrix}, \ \mathbf{b} = \begin{bmatrix} b_1 \\ \vdots \\ b_k \end{bmatrix}, \tag{2.3.3}$$

then (2.3.2) can be rewritten as $A\mathbf{x} = \mathbf{b}$. We also sometimes represent (2.3.2) by its *augmented matrix*

$$[A|\mathbf{b}] = \left[\begin{array}{ccc|c} a_{11} & \cdots & a_{1n} & b_1 \\ \vdots & \ddots & \vdots & \vdots \\ a_{k1} & \cdots & a_{kn} & b_k \end{array}\right]. \tag{2.3.4}$$

Finally, the *solution set* of (2.3.2) is the set of all $\mathbf{x} = \begin{bmatrix} x_1 \\ \vdots \\ x_k \end{bmatrix} \in F^k$ such that all equations in (2.3.2) hold, or in other words, such that $A\mathbf{x} = \mathbf{b}$.

As you may recall, if a linear system is represented by a matrix of the following type, then it is particularly easy to solve.

Definition 2.3.2. Let A be a matrix with entries in F. To say that A is in *row-echelon form*, or *REF*, means that:

1. The leftmost entry of each nonzero row of A is 1. (These are called *leading* 1*s*.)
2. The leading 1s move strictly to the right as we go down the rows of A. For example, if the leading 1 is the 3rd entry of row i, then if row $i+1$ is nonzero, its leading 1 must be entry j for some $j \geq 4$.
3. Any all-0 rows of A must be in its final row(s).

If A is in REF, we call the columns containing leading 1s the *leading columns* of A. Note that by property 2, if A is in REF, all entries underneath a leading 1 must be 0. If A is in REF, and in addition, all entries *above* every leading 1 are 0, we say that A is in *reduced row-echelon form*, or *RREF*.

As promised, here is the method for writing down the solution to a system of linear equations whose augmented matrix is in RREF.

Algorithm 2.3.3. Suppose we have a system of linear equations whose augmented matrix $[A|\mathbf{b}]$ is in RREF, and suppose A is a $k \times n$ matrix.

1. If the last nonzero row of $[A|\mathbf{b}]$ has the form $[0 \ldots 0|1]$, then the corresponding equation is $0 = 1$, and the system has no solutions (empty solution set).
2. Otherwise, all of the leading 1s of $[A|\mathbf{b}]$ are found in A. Call the variables corresponding to leading columns of A the *leading variables*, and call the other variables of the system the *free variables*.
3. Let $\{x_{j_1}, \ldots, x_{j_f}\}$ be the f free variables. Add an equation $x_{j_m} = x_{j_m}$ for each free variable x_{j_m}; we now have n equations in n variables.
4. Rewrite each of the n equations as $x_i =$ (everything else in the equation moved to the right-hand side). We then see that our solution set is the set of all $\mathbf{x}$ of the form

$$\mathbf{x} = \mathbf{b}' + x_{j_1}\mathbf{a}'_1 + \cdots + x_{j_f}\mathbf{a}'_f, \tag{2.3.5}$$

where $\mathbf{b}'$ is $\mathbf{b}$ with 0s inserted at the locations $j_1, \ldots, j_f$, and each $\mathbf{a}'_m$ is the negative of column j_m of A, with a 1 inserted at location j_m and a 0 inserted at every other location in $j_1, \ldots, j_f$.

That last description is awkward and probably only useful if you already know what it says, but an example should make things clearer.

Example 2.3.4. Consider the system with augmented matrix

$$[A|\mathbf{b}] = \left[\begin{array}{ccccc|c} 1 & 2 & 0 & 0 & -3 & 1 \\ 0 & 0 & 1 & 0 & 5 & -6 \\ 0 & 0 & 0 & 1 & 7 & 4 \end{array}\right]. \tag{2.3.6}$$

There are no $0 = 1$ rows, and our free variables are x_2 and x_5. Adding the corresponding $x_j = x_j$ rows, we get

$$\begin{array}{rcrcrcrcrcl} x_1 & + & 2x_2 & & & - & 3x_5 & = & 1, & & \\ & & x_2 & & & & & = & & x_2, & \\ & & & x_3 & & + & 5x_5 & = & -6, & & \\ & & & & x_4 & + & 7x_5 & = & 4, & & \\ & & & & & & x_5 & = & & x_5. & \end{array} \tag{2.3.7}$$

Rearranging and rewriting in column vector form, we get

$$\begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \end{bmatrix} = \begin{bmatrix} 1 \\ 0 \\ -6 \\ 4 \\ 0 \end{bmatrix} + x_2 \begin{bmatrix} -2 \\ 1 \\ 0 \\ 0 \\ 0 \end{bmatrix} + x_5 \begin{bmatrix} 3 \\ 0 \\ -5 \\ -7 \\ 1 \end{bmatrix}. \tag{2.3.8}$$

Note where the 0s and 1s have been inserted, and also note the annoying sign changes we are forced to remember (because life is cruel). Also, we haven't specified the field F over which this is all happening, but it doesn't matter — the same calculation works no matter what F is.

So how can we put a given system of linear equations in RREF? To start, as you may recall from linear algebra over $\mathbf{R}$, the following three operations on a system of linear equations, or equivalently, on its augmented matrix $[A|\mathbf{b}]$, do not change its solution set.

Definition 2.3.5. The *elementary operations* on a system of linear equations, or matrix, are:

1. Switch equation i and equation j, or equivalently, switch row i and row j of $[A|\mathbf{b}]$.

2. For $a \in F$, $a \neq 0$, multiply both sides of equation i by a, or equivalently, multiply row i of $[A|\mathbf{b}]$ by a.

3. For $a \in F$, add a times equation i to equation j, or equivalently, add a times row i of $[A|\mathbf{b}]$ to row j.

Note that the proof that elementary operations do not change the solution set of $[A|\mathbf{b}]$ is just that all three operations are reversible. Crucially, though, we need to use the fact that F is a field to reverse multiplication of row i by $a \neq 0$.

Gauss devised the following algorithm to combine the elementary operations to put any matrix in RREF.

Algorithm 2.3.6. *Gaussian reduction* is the following (recursive) algorithm for putting a $k \times n$ matrix A into REF.

1. If A is the $k \times n$ zero matrix, done.
2. Otherwise, swap the rows of A (elementary operation type 1) to get a leftmost nonzero entry $a \neq 0$ in the top row of A.
3. Multiply the top row of A by a^{-1} to make the leftmost nonzero entry of the top row equal to 1. (Here, we again rely on the assumption that F is a field.)
4. Add appropriate multiples of the top row of A to the other rows of A to make all entries underneath the leading 1 of the top row equal to 0.
5. Go back to step 1 and apply Gaussian reduction to the $k-1$ rows of A beneath the top row.

We can then put A into RREF by adding appropriate multiples of each nonzero row to the rows above it, to make all entries above each leading 1 also equal to 0.

Again, an example will probably be clearer than a formal description of the algorithm.

Example 2.3.7. Let $F = \mathbf{F}_7$, and keep in mind that mod 7, $4 = -3$, $5 = -2$, $6 = -1$, and $2 \cdot 4 = 3 \cdot 5 = 1$. To put $A = \begin{bmatrix} 0 & 3 & 3 & 0 & 3 & 0 \\ 3 & 6 & 5 & 1 & 3 & 5 \\ 5 & 6 & 6 & 4 & 2 & 1 \end{bmatrix}$ in RREF, we first swap rows 1 and 2, multiply the new top row by $3^{-1} = 5 = -2$, and then add 2 times row 1 to row 3, to get

$$\to \begin{bmatrix} 3 & 6 & 5 & 1 & 3 & 5 \\ 0 & 3 & 3 & 0 & 3 & 0 \\ 5 & 6 & 6 & 4 & 2 & 1 \end{bmatrix} \to \begin{bmatrix} 1 & 2 & 4 & 5 & 1 & 4 \\ 0 & 3 & 3 & 0 & 3 & 0 \\ 5 & 6 & 6 & 4 & 2 & 1 \end{bmatrix} \to \begin{bmatrix} 1 & 2 & 4 & 5 & 1 & 4 \\ 0 & 3 & 3 & 0 & 3 & 0 \\ 0 & 3 & 0 & 0 & 4 & 2 \end{bmatrix}. \tag{2.3.9}$$

Next, multiply row 2 by $5 = -2$ and then add -3 times the (new) row 2 to row 3 to get

$$\to \begin{bmatrix} 1 & 2 & 4 & 5 & 1 & 4 \\ 0 & 1 & 1 & 0 & 1 & 0 \\ 0 & 3 & 0 & 0 & 4 & 2 \end{bmatrix} \to \begin{bmatrix} 1 & 2 & 4 & 5 & 1 & 4 \\ 0 & 1 & 1 & 0 & 1 & 0 \\ 0 & 0 & 4 & 0 & 1 & 2 \end{bmatrix}. \tag{2.3.10}$$

Finally, multiply row 3 by 2 and then clear the entries above the leading 1s, to get

$$\to \begin{bmatrix} 1 & 2 & 4 & 5 & 1 & 4 \\ 0 & 1 & 1 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 2 & 4 \end{bmatrix} \to \begin{bmatrix} 1 & 2 & 0 & 5 & 0 & 2 \\ 0 & 1 & 0 & 0 & 6 & 3 \\ 0 & 0 & 1 & 0 & 2 & 4 \end{bmatrix} \to \begin{bmatrix} 1 & 0 & 0 & 5 & 2 & 3 \\ 0 & 1 & 0 & 0 & 6 & 3 \\ 0 & 0 & 1 & 0 & 2 & 4 \end{bmatrix}, \tag{2.3.11}$$

the RREF of our original matrix A.

Combining Algorithms 2.3.6 and 2.3.3, we see that we can solve any system of linear equations over an arbitrary field F, just like we can when $F = \mathbf{R}$; see the problems for examples.

We end this section with a bit of theory that will come in handy.

Theorem 2.3.8. *Let A be a $k \times n$ matrix with $k < n$. Then $A\mathbf{x} = \mathbf{0}$ has at least one nonzero solution $\mathbf{x}$.*

Proof. First, since the last column of $[A|\mathbf{0}]$ will always be $\mathbf{0}$, no matter what row operations we do, we will never get the unsolvable equation $0 = 1$, which means that $A\mathbf{x} = \mathbf{0}$ has at least one solution. Furthermore, since $k < n$, the RREF of $[A|\mathbf{0}]$ has at least one free variable, which means that we can set that variable to 1, obtaining a nonzero solution $\mathbf{x}$. □

Problems

2.3.1. Let $A = \begin{bmatrix} 4 & 6 & 0 & 3 \\ 3 & 0 & 1 & 7 \\ 3 & 4 & 6 & 1 \end{bmatrix}$.

(a) Use Gaussian reduction to put A in RREF, taking entries to be in the field $\mathbf{F}_2$.

(b) Same, but in the field $\mathbf{F}_3$.

(c) Same, but in the field $\mathbf{F}_5$.

(d) Same, but in the field $\mathbf{F}_7$.

2.3.2. Let $A = \begin{bmatrix} 2 & 4 & 0 & 6 & 2 \\ 1 & 4 & 2 & 2 & 6 \\ 1 & 2 & 1 & 3 & 6 \\ 1 & 0 & 7 & 4 & 4 \end{bmatrix}$.

(a) Use Gaussian reduction to put A in RREF, taking entries to be in the field $\mathbf{F}_2$.

(b) Same, but in the field $\mathbf{F}_3$.

(c) Same, but in the field $\mathbf{F}_5$.

(d) Same, but in the field $\mathbf{F}_7$.

2.3.3. Consider the system of linear equations given by the augmented matrix

$$[A|\mathbf{b}] = \left[\begin{array}{cccc|c} 3 & 0 & 0 & 4 & 7 \\ 2 & 0 & 7 & 3 & 6 \\ 1 & 7 & 3 & 1 & 1 \\ 1 & 6 & 2 & 0 & 2 \end{array}\right]. \tag{2.3.12}$$

(a) Find the solution set (possibly empty) for this linear system, taking entries to be in the field $\mathbf{F}_2$.

(b) Same, but in the field $\mathbf{F}_3$.

(c) Same, but in the field $\mathbf{F}_5$.

(d) Same, but in the field $\mathbf{F}_7$.

2.3.4. Consider the system of linear equations given by the augmented matrix

$$[A|\mathbf{b}] = \left[\begin{array}{ccc|c} 3 & 2 & 0 & 5 \\ 5 & 3 & 1 & 0 \\ 3 & 0 & 6 & 7 \\ 6 & 0 & 0 & 4 \\ 4 & 4 & 3 & 6 \end{array}\right]. \tag{2.3.13}$$

(a) Find the solution set (possibly empty) for this linear system, taking entries to be in the field $\mathbf{F}_2$.

(b) Same, but in the field $\mathbf{F}_3$.

(c) Same, but in the field $\mathbf{F}_5$.

(d) Same, but in the field $\mathbf{F}_7$.

2.3.5. Consider the system of linear equations given by the augmented matrix

$$[A|\mathbf{b}] = \left[\begin{array}{cccccc|c} 7 & 2 & 5 & 4 & 4 & 4 & 6 \\ 7 & 1 & 2 & 2 & 2 & 4 & 1 \\ 5 & 4 & 3 & 2 & 2 & 2 & 7 \\ 4 & 1 & 2 & 6 & 3 & 1 & 1 \\ 2 & 7 & 7 & 5 & 6 & 7 & 3 \end{array}\right]. \tag{2.3.14}$$

(a) Find the solution set (possibly empty) for this linear system, taking entries to be in the field $\mathbf{F}_2$.

(b) Same, but in the field $\mathbf{F}_3$.

(c) Same, but in the field $\mathbf{F}_5$.

(d) Same, but in the field $\mathbf{F}_7$.

2.4 Subspaces of F^n

Continuing our campaign to replace $\mathbf{R}$ with an arbitrary field F, we have the following crucial idea, which is actually the principal motivation for our sudden interest in linear algebra.

Definition 2.4.1. For $n \in \mathbf{N}$, a *subspace* of F^n is a subset W of F^n that satisfies three properties:

1. W contains the zero vector $\mathbf{0}$;
2. (Closed under $+$) For any $\mathbf{v}, \mathbf{w} \in W$, we have $\mathbf{v} + \mathbf{w} \in W$; and

3. (Closed under scalar multiplication) For any $\mathbf{v} \in W$ and $a \in F$, we have $a\mathbf{v} \in W$.

Example 2.4.2. The two extreme examples of subspaces of F^n are, on the one hand, F^n itself, and on the other hand, the *zero subspace* $\{\mathbf{0}\}$.

One way to describe or define a particular subspace is to use the following construction.

Definition 2.4.3 (Span)**.** Let $\mathbf{v}_1, \ldots, \mathbf{v}_k$ be vectors in F^n. The *span* of $\{\mathbf{v}_1, \ldots, \mathbf{v}_k\}$ is defined to be the set of all F-linear combinations of $\mathbf{v}_1, \ldots, \mathbf{v}_k$. In other words,

$$\operatorname{span}\{\mathbf{v}_1, \ldots, \mathbf{v}_k\} = \{a_1\mathbf{v}_1 + \cdots + a_k\mathbf{v}_k \mid a_i \in F\}. \tag{2.4.1}$$

Conversely, if W is a subspace of F^n, to say that $\{\mathbf{v}_1, \ldots, \mathbf{v}_k\}$ *spans* W means that $W = \operatorname{span}\{\mathbf{v}_1, \ldots, \mathbf{v}_k\}$. In particular, if $\{\mathbf{v}_1, \ldots, \mathbf{v}_k\}$ spans W, then each vector $\mathbf{v}_i$ must be in W.

And indeed, the span of a subset of F^n is a sub*space* of F^n:

Theorem 2.4.4. *If $\mathbf{v}_1, \ldots, \mathbf{v}_k$ are vectors in F^n, then $\operatorname{span}\{\mathbf{v}_1, \ldots, \mathbf{v}_k\}$ is a subspace of F^n.*

Proof. This proof is both interesting and not too difficult, so it has been left to you (Problem 2.4.1). □

In matrix terms, for a given subset $\{\mathbf{v}_1, \ldots, \mathbf{v}_k\}$ of F^n, let A be the $n \times k$ matrix whose columns are $\mathbf{v}_1, \ldots, \mathbf{v}_k$. Since the linear combinations of $\mathbf{v}_1, \ldots, \mathbf{v}_k$ are precisely the vectors of the form $A\mathbf{x}$ for some $\mathbf{x} \in F^k$ (Remark 2.2.3), we have the following computational description of spanning.

Theorem 2.4.5. *Let $\{\mathbf{v}_1, \ldots, \mathbf{v}_k\}$ be a subset of F^n, and let A be the $n \times k$ matrix whose columns are $\mathbf{v}_1, \ldots, \mathbf{v}_k$. Then:*

1. *The vector $\mathbf{b} \in F^n$ is contained in $\operatorname{span}\{\mathbf{v}_1, \ldots, \mathbf{v}_k\}$ if and only if $A\mathbf{x} = \mathbf{b}$ has a solution $\mathbf{x} \in F^k$.*
2. *For a subspace W of F^n, the set $\{\mathbf{v}_1, \ldots, \mathbf{v}_k\}$ spans W if and only if $\{\mathbf{v}_1, \ldots, \mathbf{v}_k\}$ is contained in W and for every $\mathbf{b} \in W$, $A\mathbf{x} = \mathbf{b}$ has a solution $\mathbf{x} \in F^k$.* □

The situation in Theorem 2.4.5 is so useful, we give it a name.

Definition 2.4.6. Let A be an $n \times k$ matrix over a field F. The *column space of A*, or $\operatorname{Col}(A)$, is defined to be the span of the columns of A (a subspace of F^n).

The next idea can be thought of as complementary to spanning.

Definition 2.4.7 (Linear independence)**.** Let $\mathbf{v}_1, \ldots, \mathbf{v}_k$ be vectors in F^n. To say that $\{\mathbf{v}_1, \ldots, \mathbf{v}_k\}$ is *linearly independent* means that the only way to get

$$a_1\mathbf{v}_1 + \cdots + a_k\mathbf{v}_k = \mathbf{0} \tag{2.4.2}$$

for coefficients $a_i \in F$ is if all of the $a_i = 0$ (i.e., $a_1 = \cdots = a_k = 0$). If there exist some $a_1, \ldots, a_k \in F$ with not all $a_i = 0$ such that (2.4.2) holds, we say that $\{\mathbf{v}_1, \ldots, \mathbf{v}_k\}$ is *linearly dependent*.

Again thanks to Remark 2.2.3, we also have a useful computational description of linear independence.

Theorem 2.4.8. *Let $\{\mathbf{v}_1, \ldots, \mathbf{v}_k\}$ be a subset of F^n, and let A be the $n \times k$ matrix whose columns are $\mathbf{v}_1, \ldots, \mathbf{v}_k$. Then $\{\mathbf{v}_1, \ldots, \mathbf{v}_k\}$ is linearly independent if and only if the only solution to $A\mathbf{x} = \mathbf{0}$ is $\mathbf{x} = \mathbf{0}$.* □

Note that by Theorem 2.3.8, it follows that if $k > n$, then $\{\mathbf{v}_1, \ldots, \mathbf{v}_k\}$ cannot be a linearly independent subset of F^n.

In any case, if you have the sneaking feeling this is all leading up to something, you're right, and here it is.

Definition 2.4.9. Let W be a subspace of F^n. A *basis* for W is a linearly independent subset $\{\mathbf{v}_1, \ldots, \mathbf{v}_k\}$ of W that also spans W. To say that W has *dimension* k means that W has a basis with k vectors in it.

Now, even a minimallly fussy reader should worry a little about the definition of dimension in Definition 2.4.9. Specifically, what happens if a subspace has one basis with, say, 5 vectors in it, and another basis with, say, 7 vectors in it? Fortunately, the following theorem ensures that troubling scenario is impossible.

Theorem 2.4.10. *Let W be a subspace of F^n.*

1. *If $\{\mathbf{v}_1, \ldots, \mathbf{v}_s\}$ spans W and $\{\mathbf{w}_1, \ldots, \mathbf{w}_\ell\}$ is a linearly independent subset of W, then $\ell \leq s$.*

2. *Any two bases for W must have the same size k (i.e., W cannot have more than one dimension), where $k \leq n$.*

Proof. Suppose $\{\mathbf{v}_1, \ldots, \mathbf{v}_s\}$ spans W, $\{\mathbf{w}_1, \ldots, \mathbf{w}_k\}$ is a subset of W, and $k > s$. To prove part 1 of the theorem, it suffices to show that $\{\mathbf{w}_1, \ldots, \mathbf{w}_k\}$ must be linearly dependent.

Let A be the $n \times s$ matrix whose columns are $\mathbf{v}_1, \ldots, \mathbf{v}_s$, and let B the $n \times k$ matrix whose columns are $\mathbf{w}_1, \ldots, \mathbf{w}_k$. Since $\{\mathbf{v}_1, \ldots, \mathbf{v}_s\}$ spans W, by Theorem 2.4.5, for $1 \leq i \leq m$, there exists some $\mathbf{x}_i \in F^s$ such that $A\mathbf{x}_i = \mathbf{w}_i$. Therefore, if we let X be the $s \times k$ matrix whose columns are $\mathbf{x}_1, \ldots, \mathbf{x}_k$, we have that

$$AX = A[\mathbf{x}_1 \ \cdots \ \mathbf{x}_m] = [A\mathbf{x}_1 \ \cdots \ A\mathbf{x}_m] = [\mathbf{w}_1 \ \cdots \ \mathbf{w}_m] = B. \tag{2.4.3}$$

However, since $s < k$, by Theorem 2.3.8, there exists some $\mathbf{y} \in F^k$ such that $X\mathbf{y} = \mathbf{0}$ and $\mathbf{y} \neq \mathbf{0}$. Furthermore,

$$B\mathbf{y} = AX\mathbf{y} = A\mathbf{0} = \mathbf{0}, \tag{2.4.4}$$

which means, by Theorem 2.4.8, that $\{\mathbf{w}_1, \ldots, \mathbf{w}_k\}$ is linearly dependent. Part 1 of the theorem follows.

As for part 2 of the theorem, suppose we have two bases of size k and m for W. Since the first basis spans W and the second basis is linearly independent, $k \geq m$, and since the second basis spans W and the first basis is linearly independent, $m \geq k$. In other words, $k = m$, and the theorem follows. □

(To be written later: In fact, any subspace of F^n has some dimension $k \leq n$; see problems.)

Problems

2.4.1. Suppose $\mathbf{v}_1, \ldots, \mathbf{v}_k$ are vectors in F^n, and let

$$W = \operatorname{span}\{\mathbf{v}_1, \ldots, \mathbf{v}_k\} = \{a_1\mathbf{v}_1 + \cdots + a_k\mathbf{v}_k \mid a_i \in F\}. \tag{2.4.5}$$

In other words, a vector $\mathbf{x}$ of F^n is in W exactly when

$$\mathbf{x} = a_1\mathbf{v}_1 + \cdots + a_k\mathbf{v}_k \tag{2.4.6}$$

for some choice of coefficients $a_i \in F$.

(a) Explain why $\mathbf{0}$ is in W. (Suggestion: What a_i should you choose to get $\mathbf{0}$ in (2.4.6)?)

(b) Suppose $\mathbf{x}$ and $\mathbf{y}$ are in W. Explain why $\mathbf{x} + \mathbf{y}$ must be in W. (Suggestion: Both $\mathbf{x}$ and $\mathbf{y}$ have a form similar to (2.4.6); what does $\mathbf{x} + \mathbf{y}$ look like?)

(c) Suppose $\mathbf{x}$ is in W and $a \in F$. Explain why $a\mathbf{x}$ must be in W.

2.4.2. Let $\mathbf{v}_1 = \begin{bmatrix}4\\0\\0\\3\\1\end{bmatrix}$, $\mathbf{v}_2 = \begin{bmatrix}4\\4\\2\\0\\2\end{bmatrix}$, and $\mathbf{v}_3 = \begin{bmatrix}1\\2\\3\\2\\2\end{bmatrix}$ be vectors in $\mathbf{F}_5^5$.

(a) Determine if $\mathbf{b}_1 = \begin{bmatrix}0\\4\\3\\4\\1\end{bmatrix} \in \mathbf{F}_5^5$ is in $\operatorname{span}\{\mathbf{v}_1, \mathbf{v}_2, \mathbf{v}_3\}$.

(b) Determine if $\mathbf{b}_2 = \begin{bmatrix}2\\3\\2\\4\\1\end{bmatrix} \in \mathbf{F}_5^5$ is in $\operatorname{span}\{\mathbf{v}_1, \mathbf{v}_2, \mathbf{v}_3\}$.

2.4.3. Let $\mathbf{v}_1 = \begin{bmatrix}1\\2\\1\\1\\0\end{bmatrix}$, $\mathbf{v}_2 = \begin{bmatrix}0\\2\\1\\1\\2\end{bmatrix}$, $\mathbf{v}_3 = \begin{bmatrix}1\\0\\0\\0\\3\end{bmatrix}$, $\mathbf{v}_4 = \begin{bmatrix}0\\0\\0\\2\\1\end{bmatrix}$ be vectors in $\mathbf{F}_5^5$.

(a) Show that $\{\mathbf{v}_1, \mathbf{v}_2, \mathbf{v}_3, \mathbf{v}_4\}$ is linearly dependent (i.e., not linearly independent).

(b) Find a way to remove one vector from $\{\mathbf{v}_1, \mathbf{v}_2, \mathbf{v}_3, \mathbf{v}_4\}$ and leave a linearly independent set. Justify your answer, either by computation or by an explanation.

2.4.4. relatively tractable $\mathbf{F}_2^7$ (6 vecs, remove 2).

2.4.5. Let W be the subspace of $\mathbf{F}_2^7$ defined by

$$W = \text{span}\left\{ \begin{bmatrix} 1 \\ 1 \\ 1 \\ 0 \\ 0 \\ 0 \\ 1 \end{bmatrix}, \begin{bmatrix} 0 \\ 1 \\ 1 \\ 0 \\ 0 \\ 1 \\ 1 \end{bmatrix}, \begin{bmatrix} 0 \\ 1 \\ 0 \\ 1 \\ 1 \\ 1 \\ 1 \end{bmatrix} \right\}. \tag{2.4.7}$$

You may assume that the above vectors are linearly independent.

(a) What is the dimension d of W? Can you use d to predict how many vectors there will be in W?

(b) Write down every element of W.

2.4.6. calculate dimension of subspace from spanning set.

2.4.7. extension theorem.

2.4.8. dimension exists.

2.5 Linear transformations

For the last part of our quick trip through linear algebra, we consider matrix multiplication as a function.

Definition 2.5.1. Let A be a $k \times n$ matrix with entries in a field F. We define the *linear transformation associated with* A to be the function T with inputs in F^n and outputs in F^k (i.e., with *domain* F^n and *codomain* F^k) defined by

$$T(\mathbf{x}) = A\mathbf{x}. \tag{2.5.1}$$

We will use the idea of matrix multiplication as a function several times in this book, but our immediate reason for introducing the idea is the following definition.

Definition 2.5.2. Let A be a $k \times n$ matrix with entries in a field F. We define the *nullspace* of A, or equivalently, the *nullspace* of the associated linear transformation T, to be the set of all $\mathbf{x}$ such that $A\mathbf{x} = \mathbf{0}$. In symbols, this is

$$\text{Null}(T) = \text{Null}(A) = \{\mathbf{x} \in F^n \mid A\mathbf{x} = \mathbf{0}\}. \tag{2.5.2}$$

As the name indicates, nullspaces are subspaces:

Theorem 2.5.3. *Let A be a $k \times n$ matrix with entries in a field F. Then* Null(A) *(the nullspace of A) is a subspace of F^n.*

Proof. As with Theorem 2.4.4, this proof is also both interesting and not too difficult, so it has been left to you (Problem 2.5.1). □

Computationally, it is also true that our standard algorithm for solving $A\mathbf{x} = \mathbf{0}$ always produces a basis for Null(A). However, it is probably better to illustrate this first with an example, instead of a theorem.

Example 2.5.4. Suppose A is a 3×5 matrix with entries in F_{13} such that the RREF of A is

$$\begin{bmatrix} 1 & 2 & 0 & 0 & 3 \\ 0 & 0 & 1 & 0 & 4 \\ 0 & 0 & 0 & 1 & 5 \end{bmatrix}. \tag{2.5.3}$$

Solving the equation $A\mathbf{x} = \mathbf{0}$, our free variables are x_2 and x_5, and adding the rows $x_2 = x_2$ and $x_5 = x_5$, we get

$$\begin{array}{rcrcrcrcrcl} x_1 & + & 2x_2 & & & + & 3x_5 & = & 0, & & \\ & & x_2 & & & & & = & & x_2, & \\ & & & x_3 & & + & 4x_5 & = & 0, & & \\ & & & & x_4 & + & 5x_5 & = & 0, & & \\ & & & & & & x_5 & = & & x_5. & \end{array} \tag{2.5.4}$$

Rearranging and rewriting in column vector form, we get

$$\begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \end{bmatrix} = x_2 \begin{bmatrix} -2 \\ 1 \\ 0 \\ 0 \\ 0 \end{bmatrix} + x_5 \begin{bmatrix} -3 \\ 0 \\ -4 \\ -5 \\ 1 \end{bmatrix}. \tag{2.5.5}$$

A brief computation shows that $\left\{ \begin{bmatrix} -2 \\ 1 \\ 0 \\ 0 \\ 0 \end{bmatrix}, \begin{bmatrix} -3 \\ 0 \\ -4 \\ -5 \\ 1 \end{bmatrix} \right\}$ is linearly independent, so $\left\{ \begin{bmatrix} -2 \\ 1 \\ 0 \\ 0 \\ 0 \end{bmatrix}, \begin{bmatrix} -3 \\ 0 \\ -4 \\ -5 \\ 1 \end{bmatrix} \right\}$ is a basis for Null(A), and $\dim \text{Null}(A) = 2$.

As it turns out, there is nothing special about Example 2.5.4, and applying our standard methods for solving $A\mathbf{x} = \mathbf{0}$ always gives a basis for Null(A). We record this fact in a theorem.

Theorem 2.5.5. *Applying the standard algorithm for solving a system of linear equations in RREF (Algorithm 2.3.3) in vector form gives a basis for* Null(A) *whose vectors correspond bijectively (one-to-one and onto) to the free variables coming from the RREF of A. Consequently,* $\dim(\text{Null}(A))$ *is precisely the number of free variables in the RREF of A.* □

This last fact about the dimension of Null(A) can be rephrased as a result (Corollary 2.5.7) whose statement is made more succinct by the following definitions.

Definition 2.5.6. Let A be a $k \times n$ matrix over a field F. We define rank(A), the *rank of* A, to be the number of leading columns in the RREF of A, and we define nullity(A), the *nullity of* A, to be dim(Null(A)), the dimension of the nullspace of A.

Since each of the n columns of a $k \times n$ matrix in RREF is either a leading column or a free column, Theorem 2.5.5 then implies the following result.

Corollary 2.5.7 (Rank-nullity)**.** *Let A be a $k \times n$ matrix over a field F. Then*

$$\text{rank}(A) + \text{nullity}(A) = n. \tag{2.5.6}$$

for later: inverses and FFT

Problems

2.5.1. Let A be a $k \times n$ matrix with entries in a field F, and again, let Null(A) be the set of all $\mathbf{x} \in F^n$ such that $A\mathbf{x} = \mathbf{0}$.

(a) Explain why $\mathbf{0}$ is in Null(A).

(b) Suppose $\mathbf{x}$ and $\mathbf{y}$ are in Null(A). Explain why $\mathbf{x} + \mathbf{y}$ must be in Null(A). (Suggestion: What does it mean to say that $\mathbf{x}$ is in Null(A)? What does it mean to say that $\mathbf{x} + \mathbf{y}$ is in Null(A)?)

(c) Suppose $\mathbf{x}$ is in Null(A) and $a \in F$. Explain why $a\mathbf{x}$ must be in W.

2.5.2. Let $A = \begin{bmatrix} 6 & 6 & 3 & 2 & 1 \\ 5 & 5 & 1 & 4 & 3 \\ 4 & 0 & 3 & 5 & 1 \end{bmatrix}$, a 3×5 matrix with entries in $\mathbf{F}_7$. Find a basis for Null(A).

2.5.3. all elements of nullspace over $\mathbf{F}_3$.

2.6 Binary linear codes

So, you may ask, this linear algebra nonsense is all well and good, but what does it have to do with error-correction? Well, as we already saw in Section 2.1, but without the benefit of the linear-algebraic language we have now developed:

- A bit is precisely an element of $\mathbf{F}_2$.
- A "flip" error in a bit x is modelled by adding 1 to x in $\mathbf{F}_2$, which changes 0 to 1 and 1 to 0.
- A bitstring of length n is precisely an element of $\mathbf{F}_2^n$.

(We pause here to note a strange coincidence of modern life, to which several generations of algebraists owe our continued employment: That binary digits, or bits, can be modelled usefully as elements of $\mathbf{F}_2$.)

So, thinking of binary codes of length n as subsets of $\mathbf{F}_2^n$, we can now define the following large and important class of codes in a succinct manner.

Definition 2.6.1. A *binary linear code $\mathcal{C}$ of length n* is a subspace $\mathcal{C}$ of $\mathbf{F}_2^n$. Elements (vectors) of a binary linear code are called *codewords*.

In other words, a binary linear code is a way to declare that only certain bitstrings of length n are valid codewords, with the valid codewords forming a subspace $\mathcal{C}$ of $\mathbf{F}_2^n$. Consequently, for the rest of this chapter, we assume, without having to restate it, that all linear algebra (matrices, vectors, etc.) is over $\mathbf{F}_2$ and all arithmetic is mod 2.

To discuss how, and how well, binary linear codes work, it is helpful to have the following standard framework.

Definition 2.6.2 (Standard framework)**.** When we discuss a binary linear code in operation, we use the following *standard framework*.

Suppose a sender, named Xavier, wants to send bitstrings, broken up into blocks, to a receiver, named Yolanda, using the binary linear code $\mathcal{C}$. To establish common terminology and notation, here's what happens next.

1. We denote the message bitstring that Xavier wants to send by $\mathbf{m}$.

2. Xavier maps the message $\mathbf{m}$ to some codeword $\mathbf{x} \in \mathcal{C}$, a process known as *encoding*. This could be done, for example, by letting $\mathbf{x} = G\mathbf{m}$ for some matrix G, though other schemes are sometimes more natural or useful.

3. Xavier transmits $\mathbf{x}$ over the communications channel, and Yolanda receives $\mathbf{y}$. If no errors have occured in transmission, $\mathbf{y} = \mathbf{x}$, but if errors have occurred, then $\mathbf{y} \neq \mathbf{x}$.

4. Yoland interprets the received transmission $\mathbf{y}$ as a received message $\mathbf{m}'$, a process known as *decoding*. A communication is successful exactly when $\mathbf{m}' = \mathbf{m}$. Note that decoding can sometimes be broken down into two steps

   (a) First, Yolanda *corrects* $\mathbf{y}$ to a valid codeword $\mathbf{y}' \in \mathcal{C}$, using methods that are more often a matter of cleverness, and not just linear algebra.

   (b) Yolanda then *reads* $\mathbf{y}'$ as a message $\mathbf{m}'$, possibly by letting $\mathbf{m}' = G'\mathbf{y}'$.

In the terms of the standard framework, we can model transmission errors in the following linear-algebraic manner. Let $\mathbf{e}_i$ be the vector in $\mathbf{F}_2^n$ whose ith coordinate is 1 and whose other coordinates are all 0. In that notation, if exactly one error happens in transmission, flipping bit i, then

$$\mathbf{y} = \mathbf{x} + \mathbf{e}_i. \tag{2.6.1}$$

(Remember: Adding 1 (mod 2) flips 0 to 1 and 1 to 0.) Similarly, if errors happen in bits i and j, we have

$$\mathbf{y} = \mathbf{x} + \mathbf{e}_i + \mathbf{e}_j. \tag{2.6.2}$$

We next look at specific examples of binary linear codes. Usually, we define a binary linear code $\mathcal{C}$ in one of two ways:

- We can define $\mathcal{C}$ to be the nullspace of a *parity check matrix* H.

- We can define $\mathcal{C}$ to be the span of the columns of a *generator matrix* G.

Returning to the two examples of families of codes given in Section 2.1, it turns out that one example is conveniently defined using a parity check matrix, and the other is conveniently defined using a generator matrix.

Example 2.6.3 (Parity check code)**.** The *parity check code of length* $n+1$ is defined to be the nullspace $\mathcal{C}$ of the $1 \times (n+1)$ matrix $H = [1 \ \ldots \ 1]$. In other words, $\mathbf{x} \in \mathbf{F}_2^{n+1}$ is in $\mathcal{C}$ exactly when $H\mathbf{x} = 0$ (a 1×1 matrix!). Problem 2.6.1 gives an equivalent description of $\mathcal{C}$ in terms of a generator matrix.

To send the message bitstring $\mathbf{m} = \begin{bmatrix} x_1 \\ \vdots \\ x_n \end{bmatrix}$, Xavier adds the parity check bit

$$x_0 = x_1 + \cdots + x_n \tag{2.6.3}$$

and transmits $\mathbf{x} = \begin{bmatrix} x_0 \\ x_1 \\ \vdots \\ x_n \end{bmatrix}$. When Yolanda receives $\mathbf{y}$, she can check the validity of $\mathbf{y}$ by calculating $H\mathbf{y}$. If $\mathbf{y} = \mathbf{x}$, then $H\mathbf{y} = H\mathbf{x} = 0$, but if bit i has been flipped, or in other words, $\mathbf{y} = \mathbf{x} + \mathbf{e}_i$, then

$$H\mathbf{y} = H\mathbf{x} + H\mathbf{e}_i = 0 + 1 = 1. \tag{2.6.4}$$

Unfortunately, since the same error message is produced no matter which bit is flipped, Yolanda cannot tell which bit has been flipped, and she can only ask Xavier to resend. More generally, any odd number of errors can be detected, but not corrected, and any even number of errors will not be detected. (See Problem 2.6.2.)

Example 2.6.4 (Repetition code)**.** Let $n = 2m+1$ be an odd positive integer. The *repetition code of length* n is defined to be the span $\mathcal{C}$ of the (single) column of the $n \times 1$ generator matrix

$$G = \begin{bmatrix} 1 \\ \vdots \\ 1 \end{bmatrix}. \tag{2.6.5}$$

Since the only possible linear combinations of $\{G\}$ are G itself and the zero vector, $\mathcal{C} = \{\mathbf{0}, G\}$. Problem 2.6.3 gives an equivalent description of $\mathcal{C}$ in terms of a parity check matrix.

To send the message bit x, Xavier transmits the vector xG; that is, if $x = 1$, Xavier transmits $\mathbf{x} = G$, and if $x = 0$, Xavier transmits $\mathbf{x} = \mathbf{0}$. When Yolanda receives $\mathbf{y}$, she can check the validity of $\mathbf{y}$ by check if all of its bits are equal. If not all of the bits of $\mathbf{y}$ are equal, then Yolanda can correct $\mathbf{y}$ to $\mathbf{y}'$ by the *majority logic* method of choosing whichever of 0 or 1 occurs more often. She may thereby correct up to m errors, where again, $n = 2m+1$. (For a repetition code of even length, she can correct up to $(m/2) - 1$ errors and detect, but not correct, $m/2$ errors.)

We conclude this section by highlighting certain statistics that are important for evaluating the effectiveness of a binary linear code.

Notation 2.6.5. An $[n, k, d]$ *binary code* is a binary linear code $\mathcal{C}$ such that:

- $\mathcal{C}$ has length n;
- $\dim \mathcal{C} = k$; and
- d is the smallest number of nonzero coordinates appearing in a nonzero codeword of $\mathcal{C}$.

The numbers n, k, and d are called the *length*, *dimension*, and *minimum distance* of $\mathbf{C}$, respectively.

Example 2.6.6. Let $\mathcal{C}$ be the parity check code of length $n+1$ (Example 2.6.3). By rank-nullity (Corollary 2.5.7), $\dim C = n$, and since vectors are codewords of $\mathcal{C}$ exactly when they contain an even number of 1's, the minimum distance of $\mathcal{C}$ is 2. In other words, $\mathcal{C}$ is an $[n+1, n, 2]$ code.

Example 2.6.7. Let $\mathcal{C}$ be the repetition code of length n (Example 2.6.4). By the definition of dimension, $\dim C = 1$, and since the all-1's vector is the only nonzero codeword of $\mathcal{C}$, the minimum distance of $\mathcal{C}$ is n. In other words, $\mathcal{C}$ is an $[n, 1, n]$ code.

Note that if $\mathcal{C}$ is an $[n, k, d]$ code, then k gives the number of "message bits" contained in each codeword, and, as we will see in the next section, d gives the number of errors that can be detected or corrected (at least in principle). The game of error-correcting can therefore be summarized as trying to maxmimize both k and d for a given length n.

Problems

2.6.1. Let $\mathcal{C}$ be the nullspace of the $1 \times (n+1)$ matrix

$$H = [1 \ \dots \ 1]. \tag{2.6.6}$$

(a) Use our standard methods to find a basis $\mathbf{v}_1, \dots, \mathbf{v}_n$ for $\mathcal{C}$.

(b) Let G be the matrix whose columns are $\mathbf{v}_1, \dots, \mathbf{v}_n$. For a given message bitstring $\mathbf{m} = \begin{bmatrix} x_1 \\ \vdots \\ x_n \end{bmatrix}$, explain why the encoding procedure of Example 2.6.3 is equivalent to transmitting $\mathbf{x} = G\mathbf{m}$.

2.6.2. even flips vs odd flips

2.6.3. repetition code parity check matrix

2.7 The Hamming 7- and 8-codes

Definition 2.7.1. The *Hamming 7-code* $\mathcal{H}_7$ can be defined in one of two ways:

- $\mathcal{H}_7$ is the nullspace of the parity check matrix

$$H_7 = \begin{bmatrix} 1 & 0 & 1 & 0 & 1 & 0 & 1 \\ 0 & 1 & 1 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 \end{bmatrix}. \tag{2.7.1}$$

- $\mathcal{H}_7$ is the column space of the generator matrix

$$G_7 = \begin{bmatrix} 1 & 1 & 0 & 1 \\ 1 & 0 & 1 & 1 \\ 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}. \tag{2.7.2}$$

Note that the parity check matrix H_7 has the very useful property that the ith column of H_7 is precisely the binary digits of the number i, written upside down (i.e., the top digit is the 1's digit, then the 2's, then the 4's) so that H_7 is in RREF.

Following our standard framework (Definition 2.6.2), here's how the Hamming 7-code is used in practice.

1. Suppose Xavier wants to send a message bitstring $\mathbf{m} \in \mathbf{F}_2^4$.
2. Xavier encodes $\mathbf{m}$ by taking $\mathbf{x} = G_7\mathbf{m}$. Note that bits x_3, x_5, x_6, and x_7 are precisely the contents of $\mathbf{m}$.
3. Xavier transmits $\mathbf{x}$ and Yolanda receives $\mathbf{y}$.
4. Yolanda then decodes $\mathbf{y}$, using the following procedure.
   (a) First, Yolanda corrects $\mathbf{y}$ to a codeword $\mathbf{y}'$ as follows. Let $\mathbf{s} = H_7\mathbf{y} \in \mathbf{F}_2^3$ be the *syndrome* of $\mathbf{y}$. If $\mathbf{s} = \mathbf{0}$, then $\mathbf{y}$ is a codeword of $\mathcal{H}_7$, so Yolanda chooses $\mathbf{y}' = \mathbf{y}$ (the most reasonable assumption). Otherwise, Yolanda reads $\mathbf{s}$ as the binary digits of a number i, assumes that a transmission error has occurred in bit i, and chooses $\mathbf{y}'$ to be $\mathbf{y}$ with its ith bit flipped (i.e., $\mathbf{y}' = \mathbf{y} + \mathbf{e}_i$).
   (b) To finish, Yolanda reads the message $\mathbf{m}'$ off of bits 3, 5, 6, and 7 of $\mathbf{y}'$.

Example 2.7.2. Suppose Xavier wants to send the message $\mathbf{m} = \begin{bmatrix} 0 \\ 1 \\ 0 \\ 1 \end{bmatrix}$. He calculates

$$\mathbf{x} = G_7\mathbf{m} = \begin{bmatrix} 0 \\ 1 \\ 0 \\ 0 \\ 1 \\ 0 \\ 1 \end{bmatrix} \tag{2.7.3}$$

(matrix multiplication details omitted — try it yourself!) and transmits $\mathbf{x}$. If a transmission error now occurs in, for example, bit 3, Yolanda then receives

$$\mathbf{y} = \mathbf{x} + \mathbf{e}_3 = \begin{bmatrix} 0 \\ 1 \\ 1 \\ 0 \\ 1 \\ 0 \\ 1 \end{bmatrix}. \tag{2.7.4}$$

In the correction stage, Yolanda calculates

$$H_7\mathbf{y} = \begin{bmatrix} 1 \\ 1 \\ 0 \end{bmatrix}, \tag{2.7.5}$$

which she interprets as $i = 1(2^0) + 1(2^1) + 0(2^2) = 3$. She therefore chooses $\mathbf{y}' = \begin{bmatrix} 0 \\ 1 \\ 0 \\ 0 \\ 1 \\ 0 \\ 1 \end{bmatrix}$ (i.e., $\mathbf{y}$ with its 3rd bit changed) and reads the message $\mathbf{m}' = \begin{bmatrix} 0 \\ 1 \\ 0 \\ 1 \end{bmatrix}$ from bits 3, 5, 6, and 7 of $\mathbf{y}'$.

The remarkable feature of the Hamming 7-code is that Example 2.7.2 is no accident: Any single transmission error can be corrected. To be precise, we have the following theorem.

Theorem 2.7.3. *In our standard framework (Definition 2.6.2), if* $\mathbf{y} = \mathbf{x} + \mathbf{e}_i$ *(i.e., a single transmission error occurs, in bit* i*), then the syndrome* $\mathbf{s} = H_7\mathbf{y}$ *is precisely the binary digits of* i*. Consequently, any single transmission error can be corrected (by flipping the* i*th bit back).*

Proof. Since $\mathbf{x} \in \mathcal{H}_7 = \text{Null}(H_7)$, $H_7\mathbf{x} = \mathbf{0}$. Therefore,

$$\mathbf{s} = H_7\mathbf{y} = H_7(\mathbf{x} + \mathbf{e}_i) = H_7\mathbf{x} + H_7\mathbf{e}_i = H_7\mathbf{e}_i, \tag{2.7.6}$$

the ith column of H_7. However, H_7 is defined to be the matrix whose ith column is the binary digits of i (Definition 2.7.1), so the theorem follows. □

We can extend $\mathcal{H}_7$ with a parity check bit as follows.

Definition 2.7.4. The *Hamming 8-code* $\mathcal{H}_8$ is defined to be the nullspace of the parity check matrix

$$H_8 = \begin{bmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 1 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \end{bmatrix}. \tag{2.7.7}$$

Note that if we delete the first row and first column of H_8, we get H_7, the parity check matrix for the Hamming 7-code. Therefore, to be consistent with the Hamming 7-code, we write an arbitrary element of $\mathcal{H}_8$ as $\begin{bmatrix} x_0 \\ x_1 \\ \vdots \\ x_7 \end{bmatrix} \in \mathcal{H}_8$.

The key properties of the Hamming 8-code are summarized in the following theorem; the details are left for you to explore in Problems 2.7.4–2.7.6.

Theorem 2.7.5. *The Hamming 8-code* $\mathcal{H}_8$ *corrects 1 error and detects 2 errors.*

Proof. Problem 2.7.4 determines a generator matrix for $\mathcal{H}_8$ and works out its dimension; Problem 2.7.5 describes decoding. □

Now, even after going through the details (or maybe even more so after going through the details!), it may seem somewhat miraculous that the Hamming 7- and 8-codes work the way they do. If you're in an enterprising mood, however, you might wonder: How can we come up with some other code that's just as good, or maybe even better? One answer to this question, at least in principle, lies in the following idea.

Definition 2.7.6. Let $\mathbf{x}, \mathbf{y}$ be vectors in $\mathbf{F}_2^n$ (or more generally, F^n for a field F). The *Hamming distance* between $\mathbf{x}$ and $\mathbf{y}$ is:

$$\begin{aligned} d(\mathbf{x}, \mathbf{y}) &= \text{the number of coordinates in which } \mathbf{x} \text{ and } \mathbf{y} \text{ differ} \\ &= \text{the number of nonzero coordinates in } \mathbf{x} - \mathbf{y} \\ &= \text{the number of coordinate changes needed to go from } \mathbf{x} \text{ to } \mathbf{y}. \end{aligned} \tag{2.7.8}$$

The *Hamming weight* $\mathrm{wt}(\mathbf{x})$ of $\mathbf{x}$ is the number of nonzero coordinates of $\mathbf{x}$. In other words,

$$\mathrm{wt}(\mathbf{x}) = d(\mathbf{x}, 0), \qquad d(\mathbf{x}, \mathbf{y}) = \mathrm{wt}(\mathbf{x} - \mathbf{y}). \tag{2.7.9}$$

Example 2.7.7. The Hamming distance between $\mathbf{x} = \begin{bmatrix} 1 \\ 0 \\ 0 \\ 1 \\ 0 \end{bmatrix}$ and $\mathbf{y} = \begin{bmatrix} 1 \\ 1 \\ 0 \\ 1 \\ 1 \end{bmatrix}$ is 2 because $\mathbf{x}$ and $\mathbf{y}$ differ in exactly coordinates 2 and 5.

Definition 2.7.8. To repeat some definitions first seen in Notation 2.6.5, if $\mathcal{C}$ is a binary linear code, then the *minimum distance* of $\mathcal{C}$ is the smallest possible weight of a nonzero vector of $\mathcal{C}$, and if $\mathcal{C}$ has length n, dimension k, and minimum distance d, we say that $\mathcal{C}$ is an $[n, k, d]$ code.

Example 2.7.9. By listing all of the vectors of $\mathcal{H}_7$, we see that $\mathcal{H}_7$ has minimum distance 3. Combining that with what we already know about $\mathcal{H}_7$, we see that $\mathcal{H}_7$ is a $[7, 4, 3]$ code. Similarly, Problems 2.7.4 and 2.7.4 together show that $\mathcal{H}_8$ is an $[8, 4, 4]$ code.

The following theorem quantifies the idea that if a binary linear code $\mathcal{C}$ has a higher minimum distance, then $\mathcal{C}$ will be able correct and detect more errors, at least in principle. (Compare Problem 2.1.1.) Namely, we can use the following error-correction method.

Definition 2.7.10. The *nearest neighbor* error-correction method, applied to a code $\mathcal{C}$, is the following procedure for error correction, expressed in the terms of our standard framework (Definition 2.6.2):

- If there is a unique $\mathbf{y}' \in \mathcal{C}$ such that $d(\mathbf{y}, \mathbf{y}')$ is minimized, we correct $\mathbf{y}$ to $\mathbf{y}'$. (For example, if $\mathbf{y} \in \mathcal{C}$, then $\mathbf{y}' = \mathbf{y}$ is the unique codeword that minimizes $d(\mathbf{y}, \mathbf{y}')$, as $d(\mathbf{y}, \mathbf{y}) = 0$.)
- If there is more than one vector $\mathbf{y}' \in \mathcal{C}$ such that $d(\mathbf{y}, \mathbf{y}')$ is minimized, we state that $\mathbf{y}$ has been detected as an erroneous transmission, but cannot be corrected, as the different $\mathbf{y}'$ minimizing $d(\mathbf{y}, \mathbf{y}')$ are equally probable as intended transmissions.

To state the relationship between the minimum distance of a code and the number of errors that can be corrected, we first need a bit of notation: The *floor* of a real number x, or $\lfloor x \rfloor$, is the greatest integer less than or equal to x. In other words, $\lfloor x \rfloor$ is x rounded down to the nearest integer.

Theorem 2.7.11. *Let $\mathcal{C}$ be a binary linear code with minimum distance d. Then the nearest neighbor method, applied to $\mathcal{C}$, corrects $\lfloor (d-1)/2 \rfloor$ errors and detects $\lfloor d/2 \rfloor$ errors.*

Proof. We first note that by (2.7.9), if $\mathbf{x}$ and $\mathbf{y}$ differ in e coordinates, then $\mathrm{wt}(\mathbf{x} - \mathbf{y}) = e$, and vice versa. It follows that the minimum distance d of $\mathcal{C}$ is also the minimum distance between any two distinct codewords $\mathbf{x}, \mathbf{y} \in \mathcal{C}$.

So now, suppose $e \leq \lfloor (d-1)/2 \rfloor$, or in other words, $2e+1 \leq d$. The error-correcting claim in the theorem means precisely that the nearest neighbor method can correct e errors. So suppose now that we take a codeword $\mathbf{x}$ and apply e errors to it, to get a received transmission $\mathbf{y}$. In that case, on the one hand, since we can change $\mathbf{x}$ to $\mathbf{y}$ by changing e coordinates, we have that $d(\mathbf{x}, \mathbf{y}) = e$. On the other hand, if there were some other codeword $\mathbf{z} \in \mathcal{C}$ such that $d(\mathbf{z}, \mathbf{y}) \leq e$, then it would be possible to change the codeword $\mathbf{x}$ to the codeword $\mathbf{z}$ by changing at most $2e$ coordinates, violating our assumption that the minimum distance between any two distinct codewords is at least $2e+1$. Therefore, $\mathbf{x}$ is the unique nearest neighbor of $\mathbf{y}$, and the nearest neighbor method corrects up to e errors.

The analogous proof that the nearest neighbor method detects $\lfloor d/2 \rfloor$ errors is left to you (Problem 2.7.7). □

Note that the nearest neighbor method is not necessarily efficient, as it essentially involves making a lookup table with a nearest neighbor listed for each vector of $\mathbf{F}_2^n$ (where n is the length of $\mathcal{C}$). In fact, for any particular code, it may be better to use a different method that ends up having the same effect, like our decoding method for the Hamming 7-code. The point of Theorem 2.7.11 is that for an $[n, k, d]$ code, there is at least *some* method for correcting $\lfloor (d-1)/2 \rfloor$ errors and detecting $\lfloor d/2 \rfloor$ errors, so it is valuable to look for codes with as large a minimum distance d as possible.

Problems

2.7.1. Let H_7 and G_7 be the parity check and generator matrices for the Hamming 7-code (Definition 2.7.1).

(a) Find the matrix product $H_7 G_7$.

(b) Now suppose H and G are parity check and generator matrices for the same code $\mathcal{C}$, and suppose that H is in RREF and the columns of G are linearly independent. Suppose also that $\mathcal{C}$ has length n and dimension k. What will the matrix product HG be? (For example, what will the size of HG be?)

2.7.2. Let $\mathcal{H}_7$ be the Hamming 7-code (Definition 2.7.1).

(a) Write out all of the codewords in $\mathcal{H}_7$.

(b) Find all shortest nonzero codewords in $\mathcal{H}_7$ (i.e., the nonzero codewords with the fewest number of 1's).

2.7.3. Suppose Yolanda is receiving transmissions sent using the Hamming 7-code, and she receives $\mathbf{y} = \begin{bmatrix} 1 \\ 0 \\ 1 \\ 1 \\ 1 \\ 0 \\ 0 \end{bmatrix}$. Correct $\mathbf{y}$ to a codeword $\mathbf{y}'$, if necessary, and read off the message bits

3, 5, 6, and 7 to find the intended message $\mathbf{m}'$. Do the same for $\mathbf{y} = \begin{bmatrix} 1 \\ 0 \\ 1 \\ 0 \\ 1 \\ 0 \\ 1 \end{bmatrix}$ and $\mathbf{y} = \begin{bmatrix} 1 \\ 1 \\ 1 \\ 0 \\ 0 \\ 0 \\ 1 \end{bmatrix}$.

2.7.4. Let $\mathcal{H}_8$ be the Hamming 8-code (Definition 2.7.4).

(a) Use the fact that $\mathcal{H}_8 = \text{Null}(H_8)$ to find a basis for $\mathcal{H}_8$.

(b) Find the dimension of $\mathcal{H}_8$.

2.7.5. Let $\mathcal{H}_8$ be the Hamming 8-code (Definition 2.7.4), let $\mathbf{x} \in \mathcal{H}_8$ be a transmitted codeword, and let $\mathbf{y}$ be the corresponding received transmission. Let $\mathbf{s} = H_8\mathbf{y} = \begin{bmatrix} s_0 \\ s_1 \\ s_2 \\ s_3 \end{bmatrix}$ be the syndrome of $\mathbf{y}$.

(a) Suppose $\mathbf{y}$ is $\mathbf{x}$ with a single bit flipped, or in other words, $\mathbf{y} = \mathbf{x} + \mathbf{e}_i$ $(0 \leq i \leq 7)$. Explain why it must be the case that $s_0 = 1$, and explain how to use (s_1, s_2, s_3) to figure out what i is.

(b) Suppose $\mathbf{y}$ is $\mathbf{x}$ with two bits flipped, or in other words, $\mathbf{y} = \mathbf{x} + \mathbf{e}_i + \mathbf{e}_j$ $(0 \leq i, j \leq 7$, $i \neq j)$. Explain why it must be the case that $s_0 = 0$, and explain why we cannot have $(s_1, s_2, s_3) = (0, 0, 0)$.

(c) Find two different pairs i, j with $0 \leq i, j \leq 7$ and $i \neq j$ that produce the same syndrome $H_8\mathbf{y}$. (This is why we can only detect two errors, not correct them.)

2.7.6. Min distance for $\mathcal{H}_8$. Generalize.

2.7.7. nearest neighbor detects e errors if $2e \leq d$.

Chapter 3

Stronger: BCH codes

3.1 The burst error problem

Suppose we have a communications channel where errors aren't evenly spread, but instead clumped together in bunches, or *bursts*. This happens with many real-life communications situations, such as the following.

- *Transmissions over the internet:* Instead of "static" or "noise" errors, you might instead see errors coming from some kind of disconnection that would brief by the human sense of time, but would nevertheless totally mangle a long string of bits.
- *Data on a hard drive:* As of 2018, data is still often stored on rotating disk at high density. A scratch on such a disk might again mangle a long string of bits.
- *DVDs:* As of 2018, people still buy or rent movies on plastic discs called DVDs, which have the same vulnerability to scratches that hard drives do.*

As always, we should try to look at some "duh" solutions to this problem, for comparison. More to the point, here are some problems any solution to the burst error problem must overcome.

- First off, we need to make our codewords very long, or else a burst error will randomize an entire codeword, making error-correction impossible.
- More subtly, in the terms of Notation 2.6.5, as a code gets very long ($n \to \infty$), it becomes very difficult to keep both the *information rate* k/n and the *error-correction rate* d/n high.

So it's probably the case that something has to give. Therefore, instead of looking for a code that can do it all, let's look at the following problem, which is not quite as difficult. (After all, this is still a math book, which means we still have the option of making the problem easier until we can actually solve it.)

*Author's note: I realize I sound like an alien being describing everyday human life, but this is here as a hedge for the future. Even 10 years ago, I would have used CDs for this example, but by now in 2018, I suspect some readers will have never encountered a CD.

Motivating Problem 3.1.1. Create an error-correcting code $\mathcal{C}$ that will correct relatively long burst errors. We allow for the possibility that $\mathcal{C}$ is not so great at correcting randomly scattered errors (i.e., has a low d/n).

Problems

3.1.1. Consider a binary linear error-correcting code $\mathcal{C}$. In this problem, we look at how burst errors interact with not just one codeword, but a *sequence* of codewords.

(a) Suppose $\mathcal{C}$ has length 10. What is the length b of the smallest burst of bit errors (consecutive string of bit errors) that can affect (create errors within) two consecutively transmitted codewords? Draw a picture to explain your answer.

(b) Now suppose $\mathcal{C}$ has length 13. What is the length b of the smallest burst of bit errors that can affect three consecutively transmitted codewords? Again, explain with a picture.

(c) Finally, suppose $\mathcal{C}$ has some arbitrary length n, and $m \geq 1$. In terms of n and m, what is the length b of the smallest burst of bit errors that can affect m consecutively transmitted codewords? Explain with a picture.

3.2 Cyclic codes

To construct the class of codes that solves Motivating Problem 3.1.1, we will first need to generalize the idea of a binary linear code (Definition 2.6.1).

Definition 3.2.1. Let $q = p^r$ be a prime power, and let $\mathbf{F}_q$ be the (unique) field of order q. A *linear code $\mathcal{C}$ of length n over $\mathbf{F}_q$* is a subspace $\mathcal{C}$ of $\mathbf{F}_q^n$. Elements (vectors) of a linear code are again called *codewords*.

Remark 3.2.2. Since all of the codes we discuss will be linear, we will sometimes just call them codes. In case you were wondering, though, there are such things as nonlinear codes, which are just defined to be sub*sets* of $\mathbf{F}_q^n$, and in fact, you can construct nonlinear codes that have excellent communications statistics (i.e., the nonlinear equivalent of a high k/n and d/n). However, nonlinear codes are more difficult to use because we can no longer think of them in terms of linear algebra. For more on nonlinear codes, see ???.

Definition 3.2.3. Let $\mathcal{C}$ be a linear code of length n over $\mathbf{F}_q$. To say that $\mathcal{C}$ is *cyclic* means that it is closed under cyclic permutation of coordinates. That is, to say that $\mathcal{C}$ is cyclic means that if $\begin{bmatrix} c_0 \\ c_1 \\ c_2 \\ \vdots \\ c_{n-1} \end{bmatrix}$ is in $\mathcal{C}$, then so are $\begin{bmatrix} c_{n-1} \\ c_0 \\ c_1 \\ \vdots \\ c_{n-2} \end{bmatrix}$, $\begin{bmatrix} c_{n-2} \\ c_{n-1} \\ c_0 \\ \vdots \\ c_{n-3} \end{bmatrix}$, and so on.

The symmetry condition in Definition 3.2.3 is reasonably natural, as it is often the case that "good" codes are more symmetric than usual. However, the really useful thing about cyclic codes can be seen by writing codewords using the following notation.

Notation 3.2.4. The *polynomial notation* for vectors in $\mathbf{F}_q^n$ represents the vector $\begin{bmatrix} c_0 \\ \vdots \\ c_{n-1} \end{bmatrix}$ as the polynomial

$$c_0 + c_1 x + c_2 x^2 + \cdots + c_{n-1} x^{n-1} \tag{3.2.1}$$

in the ring $R = \mathbf{F}_q[x]/(x^n - 1)$ (i.e., setting $x^n \equiv 1$).

Note that in $R = \mathbf{F}_q[x]/(x^n - 1)$, addition of polynomials and multiplication of a polynomial by a scalar both work exactly like they do for the corresponding vectors. What makes the polynomial notation useful is that if

$$c(x) = c_0 + c_1 x + c_2 x^2 + \cdots + c_{n-1} x^{n-1} \tag{3.2.2}$$

is a codeword, then since $x^n \equiv 1$ in R, we have that

$$\begin{aligned} xc(x) &= c_0 x + c_1 x^2 + c_2 x^3 + \cdots + c_{n-2} x^{n-1} + c_{n-1} x^n \\ &= c_{n-1} + c_0 x + c_1 x^2 + c_2 x^3 + \cdots + c_{n-2} x^{n-1}, \end{aligned} \tag{3.2.3}$$

which is precisely the (one-step) cyclic permutation of the coordinates of $c(x)$. Similarly, $x^2 c(x)$ cyclically permutes the coordinates of $c(x)$ by two steps, and in general, $x^k c(x)$ cyclically permutes the coordinates of $c(x)$ by k steps. As we'll see momentarily, that means that cyclic codes can be characterized algebraically using the following idea.

Definition 3.2.5. Let R be a (commutative) ring. An *ideal* of R is a subset I of R satisfying the following three axioms:

1. (Zero) The zero element of R is contained in I.
2. (Closed under addition) If $x, y \in I$, then $x + y \in I$.
3. (Closed under R-scalar multiplication) If $x \in I$ and $r \in R$, then $rx \in I$.

Theorem 3.2.6. *Let $\mathcal{C}$ be a linear code of length n over $\mathbf{F}_q$. In polynomial notation, $\mathcal{C}$ is cyclic if and only if it is an ideal of the ring $\mathbf{F}_q[x]/(x^n - 1)$.*

Proof. On the one hand, suppose $\mathcal{C}$ is an ideal of $\mathbf{F}_q[x]/(x^n - 1)$. Since $\mathcal{C}$ is closed under multiplication by $x \in \mathbf{F}_q[x]/(x^n - 1)$, by the previous discussion if $c(x) \in \mathcal{C}$, so are all of its cyclic permutations $x^k c(x)$. It follows that $\mathcal{C}$ is cyclic.

For the converse, see Problem 3.2.1. □

Problems

3.2.1. Suppose $\mathcal{C}$ is a cyclic code of length n over $\mathbf{F}_q$. We continue to use the polynomial notation (Notation 3.2.4) and think of $\mathbf{F}_q^n$ as $\mathbf{F}_q[x]/(x^n-1)$. The goal of this problem is to prove that (explain why) $\mathcal{C}$ is an ideal (Definition 3.2.5) of $\mathbf{F}_q[x]/(x^n-1)$.

(a) Explain why $\mathcal{C}$ contains zero and is closed under addition.

(b) Suppose $c(x) \in \mathcal{C}$. Explain why $xc(x) \in \mathcal{C}$.

(c) Suppose $c(x) \in \mathcal{C}$. Explain why $(ax + bx^2)c(x) \in \mathcal{C}$.

(d) Suppose $c(x) \in \mathcal{C}$ and

$$f(x) = a_0 + a_1x^1 + \cdots + a_kx^k \tag{3.2.4}$$

is an element of $F_q[x]$. Explain why $f(x)c(x) \in \mathcal{C}$.

3.3 Ideals and minimal polynomials

We repeat Definition 3.2.5.

Definition 3.3.1. Let R be a (commutative) ring. An *ideal* of R is a subset I of R satisfying the following three axioms:

1. (Zero) The zero element of R is contained in I.
2. (Closed under addition) If $x, y \in I$, then $x + y \in I$.
3. (Closed under R-multiplication) If $x \in I$ and $r \in R$, then $rx \in I$.

Like any other piece of pure abstract nonsense, the definition of ideal will only make sense after we consider some examples.

Example 3.3.2 (Even integers)**.** Let $R = \mathbf{Z}$, and let I be the set of even integers. Then I is an ideal because:

1. 0 is even.
2. The sum of two even numbers is even.
3. The last axiom is worth considering in more detail. The point is that, not only is the product of even numbers even, but also the product of an *arbitrary* integer, even or odd, and an even number is even.

Example 3.3.3 (Silly examples)**.** For a ring R, the set $\{0\}$ is an ideal of R called the *zero ideal*. The ring R is also an ideal of itself.

Example 3.3.4 (Multiples of m)**.** For a fixed $m \in \mathbf{Z}$, the set

$$m\mathbf{Z} = \{km \mid k \in \mathbf{Z}\} \tag{3.3.1}$$

is an ideal of $\mathbf{Z}$.

Example 3.3.5 (Multiples of $m(x)$)**.** Similarly, if F is a field, for a fixed $m(x) \in F[x]$, the set

$$(m(x)) = \{k(x)m(x) \mid k(x) \in F[x]\} \tag{3.3.2}$$

is an ideal of $F[x]$.

Example 3.3.6 (Principal ideals)**.** For a ring R and a fixed $a \in R$, the set

$$(a) = \{ra \mid r \in R\} \tag{3.3.3}$$

is an ideal of R called the *principal ideal generated by* a; see Problem 3.3.1. Note that Examples 3.3.4 and 3.3.5, which we have also seen before in other contexts, are both principal ideals.

Example 3.3.7 (Finitely generated ideals)**.** For a ring R and fixed $a, b \in R$, the set

$$(a, b) = \{ra + sb \mid r, s \in R\} \tag{3.3.4}$$

is an ideal of R called the *ideal generated by* a *and* b; see Problem 3.3.2.

As always, our favorite rings to consider are the integers $\mathbf{Z}$ and polynomials $F[x]$ (F a field), and both of these rings have the following very useful property.

Theorem 3.3.8. *Let R be either $\mathbf{Z}$ or $F[x]$. Then every ideal of R is principal.*

Proof. For the case $R = \mathbf{Z}$, let I be an ideal of R. If $I = \{0\}$, then $I = (0)$, and I is principal. Otherwise, I contains nonzero elements of $\mathbf{Z}$; in fact, since I is closed under multiplication by $-1 \in \mathbf{Z}$, I must contain positive elements.

So let d be the smallest positive element of I. For any other $a \in I$, by the Division Algorithm, we have that

$$a = qd + r \qquad \text{where } 0 \leq r < d. \tag{3.3.5}$$

However, since $-q \in R$, $d \in I$, and I is an ideal, $-qd \in I$; and since $a \in I$, $-qd \in I$, and I is an ideal, $a - qd \in I$. Therefore, $r = a - qd \in I$, and since d is the smallest positive element of I, we must have $r = 0$. It follows that $I = (d)$.

For a field F, the case $R = F[x]$ follows entirely analogously; see Problem 3.3.3. □

The fact that every ideal in $F[x]$ is principal leads to the following definition.

Definition 3.3.9. Let F be a field, and let I be an ideal of $F[x]$. To say that $d(x)$ is the *minimal polynomial* of I means that $I = (d(x))$. (Note that if we choose the leading term of $d(x)$ to be 1, then $d(x)$ really is unique and not just "unique up to multiplication by a unit.")

Now, the proof of Theorem 3.3.8 is *nonconstructive*, in that it does not describe a method for computing the minimal polynomial $d(x)$ in practice, partly because we have not yet specified the data structure that defines I (e.g., some kind of list of polynomials). However, if, for example, we begin with a finite set of generators for I, we can compute $d(x)$, and you'll never guess what we use to do that. (OK, at this point in the book, maybe you can guess.)

Theorem 3.3.10. *Let F be a field, and consider the idea $I = (a(x), b(x))$ of $F[x]$, where $a(x)$ and $b(x)$ are nonzero polynomials in $F[x]$. Then the minimal polynomial of I is $\gcd(a(x), b(x))$, which can be computed by the Euclidean algorithm.*

Proof. Let $d(x) = \gcd(a(x), b(x))$. On the one hand, since $d(x)$ divides both $a(x)$ and $b(x)$, $d(x)$ divides every element of $I = \{r(x)a(x) + s(x)b(x) \mid r(x), s(x) \in F[x]\}$, which means that I is contained in $(d(x))$. Conversely, by Bézout's identity, we know that

$$d(x) = r(x)a(x) + s(x)b(x) \tag{3.3.6}$$

for some $r(x), s(x) \in F[x]$, which means that $d(x)$ and all of its $F[x]$-multiples are contained in I; in other words, $(d(x))$ is contained in I. It follows that $I = (d(x))$, and the theorem follows. □

Problems

3.3.1. Let R be a commutative ring and $a \in R$, and define

$$(a) = \{ra \mid r \in R\}. \tag{3.3.7}$$

The goal of this problem is to prove that (a) is an ideal of R.

(a) Explain how you know that $0 \in (a)$.

(b) What do two random elements of (a) look like? Explain why their sum must be in (a).

(c) For $s \in R$ and $x \in (a)$, explain why $sx \in (a)$.

3.3.2. Let R be a commutative ring and $a, b \in R$, and define

$$(a, b) = \{ra + sb \mid r, s \in R\}. \tag{3.3.8}$$

The goal of this problem is to prove that (a, b) is an ideal of R.

(a) Explain how you know that $0 \in (a, b)$.

(b) What do two random elements of (a, b) look like? Explain why their sum must be in (a, b).

(c) For $s \in R$ and $x \in (a, b)$, explain why $sx \in (a, b)$.

3.3.3. Prove that every ideal of $F[x]$ is principal.

3.3.4. Consider the ideal $I = (x^3 + 2x + 2, x^4 + 2x^3 + x^2 + x + 3)$ in $\mathbf{F}_5[x]$. Compute the minimal polynomial of I.

3.3.5. Compute minimal polynomial for ideal generated by three.

3.4 Cyclic codes and generator polynomials

Returning to cyclic codes, we first have the following variation on Theorem 3.3.8.

Theorem 3.4.1. *Let F be a field, fix $n \in \mathbf{N}$, and let $\mathcal{C}$ be a cyclic code of length n, i.e., an ideal of $R = F[x]/(x^n - 1)$. Then $\mathcal{C}$ is principal; moreover, the minimal polynomial of $\mathcal{C}$ is a divisor of $x^n - 1$.*

Proof. The argument of Theorem 3.3.8 shows that if I is not the zero ideal, then every element of I is a multiple of some lowest-degree element $d(x)$. However, the same argument also applies to $x^n - 1$, and the theorem follows. □

Definition 3.4.2. Let $\mathcal{C}$ be a cyclic code of length n over $\mathbf{F}_q$. We define the *generator polynomial* of $\mathcal{C}$ to be the minimal polynomial of $\mathcal{C}$ (as an ideal in $\mathbf{F}_q[x]/(x^n - 1)$).

Therefore, if we want to study the cyclic codes of length n over $\mathbf{F}_q$, it suffices to consider each possible factor $g(x)$ of $x^n - 1$ over $\mathbf{F}_q$ and study the corresponding principal ideals $(g(x))$. Still easier said than done! But at least this considerably limits the kind of example we need to look at.

Before we turn to one particular method of making good cyclic codes (Sections 3.5–3.7), we first establish some facts that hold for cyclic codes in general, beginning with the dimension of a cyclic code.

Theorem 3.4.3. *Let $\mathcal{C}$ be a cyclic code of length n over $\mathbf{F}_q$ that is generated by the divisor $g(x) \in \mathbf{F}_q[x]$ of $x^n - 1$. If $\deg g(x) = r$, then the set*

$$\mathcal{B} = \left\{ g(x), xg(x), \ldots, x^{(n-1)-r} g(x) \right\} \tag{3.4.1}$$

is a basis for $\mathcal{C}$. Consequently, the dimension of $\mathcal{C}$ is $k = n - r$.

Proof. We first show that $\mathcal{B}$ is linearly independent. Let

$$g(x) = c_0 + c_1 x + \cdots + c_{r-1} x^{r-1} + c_r x^r, \tag{3.4.2}$$

where $c_r \neq 0$. Then if G is the matrix whose columns are $\mathcal{B}$ in ordinary vector form, we have

$$G = \begin{bmatrix} c_0 & 0 & 0 & \cdots & 0 \\ c_1 & c_0 & 0 & \cdots & 0 \\ c_2 & c_1 & c_0 & \cdots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ c_r & c_{r-1} & c_{r-2} & \cdots & 0 \\ 0 & c_r & c_{r-1} & \cdots & c_0 \\ 0 & 0 & c_r & \cdots & c_1 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \cdots & c_r \end{bmatrix}, \tag{3.4.3}$$

where the lower right-hand entry corresponds to the term $c_r x^{n-1}$ in $x^{(n-1)-r}g(x)$. Then multiplying each of the bottom $n-r$ rows by c_r^{-1}, we get a matrix in REF in which every column is a leading column. It follows that $A\mathbf{x} = \mathbf{0}$ has $\mathbf{x} = \mathbf{0}$ as its only solution, or in other words, $\mathcal{B}$ is linearly independent.

To see why $\mathcal{B}$ spans $\mathcal{C}$, see Problem 3.4.2. □

Encoding and reading an error-free transmission are then straightforward:

- To encode a message $\mathbf{m} = \begin{bmatrix} m_0 \\ \vdots \\ m_{n-r-1} \end{bmatrix}$, we write $\mathbf{m}$ in polynomial notation as

$$m(x) = m_0 + m_1 x + \cdots + m_{n-r-1}x^{n-r-1} \tag{3.4.4}$$

  and transmit the codeword

$$m(x)g(x) = m_0 g(x) + m_1 x g(x) + \cdots + m_{n-r-1}x^{n-r-1}g(x). \tag{3.4.5}$$

- Conversely, to read a received codeword $y(x)$ that has had no errors in transmission, we use polynomial division to recover

$$m(x) = y(x)/g(x). \tag{3.4.6}$$

  Note that if $g(x)$ does not divide $y(x)$, then $y(x)$ is not an element of $\mathcal{C} = (g(x))$, and an error must have occurred in transmission.

Of course, as is always the case with error-correcting codes (compare Section 2.6), actually *correcting* errors is more difficult, and the error-correcting procedures we use change greatly in different examples.

We end this section by looking at a few examples of cyclic codes; see Problems 3.4.5–3.4.6 for some more examples.

Example 3.4.4 (Parity check as cyclic)**.** For $n \in \mathbf{N}$, note that

$$x^n - 1 = (x-1)(1 + x + \cdots + x^{n-1}) \tag{3.4.7}$$

in $\mathbf{F}_2[x]$ (or actually, in $F[x]$ for any field F). Let $\mathcal{C}$ be the cyclic code of length n over $\mathbf{F}_2$ that is generated by the divisor $1+x$ (which is the same as $x-1$ with coefficients (mod 2)). Then $\mathcal{C}$ is precisely the parity check code of length n from Example 2.6.3; see Problem 3.4.3.

Example 3.4.5 (Repetition as cyclic)**.** Let $\mathcal{C}$ be the cyclic code of length n over $\mathbf{F}_2$ that is generated by $(1 + x + \cdots + x^{n-1})$. Then $\mathcal{C}$ is precisely the repetition code of length n from Example 2.6.4; see Problem 3.4.4.

Example 3.4.6 ($\mathcal{H}_7$ as cyclic)**.** Let $\mathcal{C}$ be the cyclic code of length 7 over $\mathbf{F}_2$ that is generated by $1 + x + x^3$. We now show that, after a change of coordinates, $\mathcal{C}$ is the Hamming 7-code $\mathcal{H}_7$.

First, following the proof of Theorem 3.4.3, we see that

$$G = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 \\ 0 & 1 & 1 & 0 \\ 1 & 0 & 1 & 1 \\ 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{3.4.8}$$

is a generator matrix for $\mathcal{C}$.

Next, we claim that

$$H = \begin{bmatrix} 1 & 0 & 1 & 1 & 1 & 0 & 0 \\ 1 & 1 & 1 & 0 & 0 & 1 & 0 \\ 0 & 1 & 1 & 1 & 0 & 0 & 1 \end{bmatrix} \tag{3.4.9}$$

is a parity check matrix for $\mathcal{C}$. While it may be unclear how H was derived, the fact that HG is the zero matrix (check this for yourself!) and $\operatorname{rank} H = 3 = 7 - 4$ means that the columns of G are a basis for $\operatorname{Null}(H)$, which must therefore be $\mathcal{C}$.

We then observe that the columns of H are the binary digits (written backwards) of $3, 6, 7, 5, 1, 2, 4$, respectively. It follows that $\mathcal{C}$ is the Hamming 7-code, but with bits $x_1, \ldots, x_7$ moved to positions $5, 6, 1, 7, 4, 2, 3$, respectively.

Example 3.4.6 certainly shows that it is possible to find good cyclic codes. The rest of this chapter is therefore devoted to a solution to the following problem:

Motivating Problem 3.4.7. Given $\mathbf{F}_q$, how can we choose $n \in \mathbf{N}$ and $g(x)$ dividing $x^n - 1$ so that the cyclic code $\mathcal{C}$ generated by $g(x)$ has both a relatively large dimension and a large minimum distance?

Problems

3.4.1. Let $\mathcal{C}$ be the cyclic code of length 9 over $\mathbf{F}_2$ generated by $x^5 + x^4 + 1$, or in other words, let $\mathcal{C}$ be the principal ideal $(x^5 + x^4 + 1)$. Find the generating polynomial (minimal polynomial) $g(x)$ of $\mathcal{C}$. (Suggestion: Note that by Theorem 3.4.1, $g(x)$ must divide both $x^5 + x^4 + 1$ and $x^9 - 1$.)

3.4.2. Let $\mathcal{C}$ be a cyclic code of length n over $\mathbf{F}_q$ that is generated by the divisor $g(x) \in \mathbf{F}_q[x]$ of $x^n - 1$, and suppose that $\deg g(x) = r$. Let

$$\mathcal{B} = \left\{ g(x), xg(x), \ldots, x^{(n-1)-r} g(x) \right\}, \tag{3.4.10}$$

and suppose $f(x) \in \mathcal{C}$. Since we are working mod $(x^n - 1)$, we may assume that $\deg f \leq n - 1$.

(a) Explain why it must be the case that $f(x) = q(x)g(x)$ for some polynomial $q(x)$ of degree at most $(n-1)-r$.

(b) Prove that $f(x)$ is a linear combination of the elements of $\mathcal{B}$.

3.4.3. Let $\mathcal{C}$ be the cyclic code of length n over $\mathbf{F}_2$ that is generated by $1+x$.

(a) Write out the generator matrix of $\mathcal{C}$, as in the proof of Theorem 3.4.3. What is the dimension of $\mathcal{C}$?

(b) Prove that each basis element from part (a) is a codeword of the parity check code of length n (Example 2.6.3). (Since the two codes have the same dimension, it follows that they must be equal.)

3.4.4. Let $\mathcal{C}$ be the cyclic code of length n over $\mathbf{F}_2$ that is generated by $1+x+\cdots+x^{n-1}$.

(a) Write out the generator matrix of $\mathcal{C}$, as in the proof of Theorem 3.4.3. What is the dimension of $\mathcal{C}$?

(b) Prove that each basis element from part (a) is a codeword of the repetition code of length n (Example 2.6.4). (Since the two codes have the same dimension, it follows that they must be equal.)

3.4.5. Stats for example of binary cyclic code, length 8.

(a) check divides

(b) basis

(c) dimension

3.4.6. Stats for example of binary cyclic code, length 9.

(a) check divides

(b) basis

(c) dimension

3.5 Primitive roots and minimal polynomials

To be able to state, let alone explain, the theorem we will present that gives one solution to Motivating Problem 3.4.7, we need the following definition.

Definition 3.5.1. A *finite extension field* of $\mathbf{F}_q$ is a finite field E that contains $\mathbf{F}_q$ as a subfield (i.e., a subset of E that is itself a field, using the same operations of $+$ and $\cdot$). For brevity, we sometimes just say that E is an *extension* of $\mathbf{F}_q$.

We restrict our attention to two examples of extension fields, the first of which is silly (but, as we shall see, useful).

Example 3.5.2. The field $\mathbf{F}_q$ is an extention of itself.

Example 3.5.3. Let $q = 2^r$. Then the field $\mathbf{F}_q$ is an extention of $\mathbf{F}_2$.

We note that if E is an extension of $\mathbf{F}_q$, then a polynomial that is irreducible in $\mathbf{F}_q[x]$ may be reducible in the larger ring $E[x]$. In other words, when we factor over E, we may be able to factor polynomials that are irreducible over $\mathbf{F}_q$. For example, x^3+x+1 is irreducible over $\mathbf{F}_2$, but if α is a root of $x^3 + x + 1$ in $\mathbf{F}_8$, then

$$x^3 + x + 1 = (x - \alpha)(x - \alpha^2)(x - \alpha^4). \tag{3.5.1}$$

(Try it yourself; see Problem 3.5.1.) We will see momentarily where the form of this factorization comes from.

Returning to our main thread, we may now state the theorem we will use to construct examples of cyclic codes that correct many errors per codeword.

Theorem 3.5.4. *Let $\mathcal{C}$ be a cyclic code of length n over $\mathbf{F}_q$ that is generated by the divisor $g(x) \in \mathbf{F}_q[x]$ of $x^n - 1$. Suppose E is an extension field of $\mathbf{F}_q$ such that for some $\delta \in \mathbf{N}$ and some primitive nth root of unity α in E, we have that*

$$0 = g(\alpha) = g(\alpha^2) = g(\alpha^3) = \cdots = g(\alpha^{\delta-1}). \tag{3.5.2}$$

Then the minimum distance d of $\mathcal{C}$ is at least δ, i.e., $d \geq \delta$.

Since the smaller $\deg g(x)$ is, the larger the dimension of $\mathcal{C}$ is, we see that it will be helpful if, given some finite extension field E of $\mathbf{F}_q$ and $\alpha \in E$, we can find a polynomial $g(x)$ of smallest possible degree such that $g(\alpha) = 0$. To achieve this goal, we will need the following facts about finite fields.

Theorem 3.5.5. *Let q be a prime power.*

1. *The field $\mathbf{F}_q$ has a primitive root; i.e., there exists some $\alpha \in \mathbf{F}_q$ such that*

$$\mathbf{F}_q = \left\{0, 1, \alpha, \ldots, \alpha^{q-2}\right\} \tag{3.5.3}$$

   and $\alpha^{q-1} = 1$.

2. *Every element of $\mathbf{F}_q$ is a root of $x^q - x$, and consequently,*

$$x^q - x = \prod_{\beta \in \mathbf{F}_q} (x - \beta). \tag{3.5.4}$$

3. *If E is a finite extension of $\mathbf{F}_q$, then the roots of $x^q - x$ are precisely $\mathbf{F}_q$ (a subfield of E of order q); in other words, $\beta \in E$ is a root of $x^q - x$ if and only if $\beta \in \mathbf{F}_q$.*

4. *Suppose E is a finite extension of $\mathbf{F}_q$, and define a function $\rho_q : E \to E$ by the formula*

$$\rho_q(\beta) = \beta^q. \tag{3.5.5}$$

   Then ρ_q is an automorphism of E fixing exactly the subfield $\mathbf{F}_q$, or in other words:

(a) ρ_q *is a bijection;*

(b) For all $\beta_1, \beta_2 \in E$*, we have*

$$\rho_q(\beta_1 + \beta_2) = \rho_q(\beta_1) + \rho_q(\beta_2), \qquad \rho_q(\beta_1\beta_2) = \rho_q(\beta_1)\rho_q(\beta_2); \tag{3.5.6}$$

and

(c) For all $\beta \in E$*,* $\rho_q(\beta) = \beta$ *if and only if* $\beta \in \mathbf{F}_q$*.*

Functions of the form $\rho_q(\beta) = \beta^q$ (assertion (4)) are called *Frobenius automorphims.*

Proof. (1) will be proven later (or possibly never).

For (2), let α be a primitive root of $\mathbf{F}_q$. For $\beta \in \mathbf{F}_q$, either $\beta = 0$, in which case certainly $\beta^q - \beta = 0$, or $\beta = \alpha^i$ for some i, in which case

$$\beta^q - \beta = \beta((\alpha^i)^{q-1} - 1) = \beta((\alpha^{q-1})^i - 1) = \beta(1^i - 1) = 0. \tag{3.5.7}$$

Therefore, each element of $\mathbf{F}_q$ is a root of $x^q - x$. Furthermore, the Root Theorem then implies that

$$r(x) = \prod_{\beta \in \mathbf{F}_q} (x - \beta) \tag{3.5.8}$$

divides $x^q - x$, and since $\deg r = q$ and r is monic (leading coefficient 1), it must be the case that $r(x) = x^q - x$.

For (3), on the one hand, part (2) of the theorem implies that each $\beta \in \mathbf{F}_q$ is a root of $x^q - x$, and $x^q - x$ can have at most q roots.

Finally, for (4), it is a fact (proven later or never?) that E must have order q^e for some $e \in \mathbf{N}$. In those terms, we first note that assertion (4c) follows from assertion (3). For assertion (4a), if α is a primitive root of E, applying ρ_q e times, we get

$$\rho_q^e(\alpha^i) = (\alpha^{q^e})^i = \alpha^i, \tag{3.5.9}$$

which means that ρ_q^e is the identity function, implying that ρ_q is a bijection.

Last but not least, we consider assertion (4b) in the special case where $q = 2$. (The general case is similar but more complicated.) Certainly $(\beta_1\beta_2)^2 = \beta_1^2\beta_2^2$, so it remains to show that $(\beta_1 + \beta_2)^2 = \beta_1^2 + \beta_2^2$. However, since we are still working with coefficients mod 2, we have that

$$(\beta_1 + \beta_2)^2 = \beta_1^2 + 2\beta_1\beta_2 + \beta_2^2 = \beta_1^2 + \beta_2^2. \tag{3.5.10}$$

The theorem follows. □

Turning to the problem of finding a "smallest" $g(x)$ such that $g(\beta) = 0$, the following theorem makes the idea of "smallest" precise.

Theorem 3.5.6. *Let* E *be an extension of* $\mathbf{F}_q$*, fix some* $\beta \in E$*, and let*

$$I = \{f(x) \in \mathbf{F}_q[x] \mid f(\beta) = 0\}. \tag{3.5.11}$$

Then I *is an ideal of* $\mathbf{F}_q[x]$*, and consequently,* $I = (m(x))$ *for some* $m(x) \in \mathbf{F}_q[x]$*.*

We call the polynomial $m(x) \in \mathbf{F}_q[x]$ in Theorem 3.5.6 the *minimal polynomial of* β *over* $\mathbf{F}_q$.

Proof. Problem 3.5.2 shows that I is an ideal, and the last assertion follows because every idea of $\mathbf{F}_q[x]$ is principal (Theorem 3.3.8). □

As you may have gathered from the statement of Theorem 3.5.4, or as we will see soon, to make Theorem 3.5.4 work effectively, we choose a generator that is a least common multiple of minimal polynomials. It will therefore be useful to establish notation for minimal polynomials.

Notation 3.5.7. If E is an extension of $\mathbf{F}_q$, and we fix a primitive root $\alpha \in E$, then $m_i(x)$ denotes the minimal polynomial of α^i over $\mathbf{F}_q$.

We can compute $m_i(x)$ efficiently using the following idea.

Definition 3.5.8. Let E be an extension of $\mathbf{F}_q$, and let β be in E. The *Frobenius orbit of* β is the set

$$\{\beta, \rho_q(\beta), \rho_q(\rho_q(\beta)), \dots\}. \tag{3.5.12}$$

Note that since a finite power of ρ_q is the identity, every Frobenius orbit is finite.

Theorem 3.5.9 (Orbit Theorem)**.** *Let E be an extension of $\mathbf{F}_q$, let β be in E, and suppose the Frobenius orbit of β is $\{\beta_1, \dots, \beta_s\}$, where $\beta_1 = \beta$. Then the minimal polynomial of β over $\mathbf{F}_q$ is*

$$m(x) = (x - \beta_1)(x - \beta_2)\dots(x - \beta_s). \tag{3.5.13}$$

Proof. First, we need to show that $m(x) \in \mathbf{F}_q[x]$. Observe that if we apply the Frobenius automorphism ρ_q to (the coefficients of) $m(x)$, because ρ_q is an automorphism (Theorem 3.5.5(4)), we get

$$\begin{aligned}
\rho_q(m(x)) &= \rho_q((x - \beta_1)(x - \beta_2)\dots(x - \beta_{s-1})(x - \beta_s)) \\
&= (x - \rho_q(\beta_1))(x - \rho_q(\beta_2))\dots(x - \rho_q(\beta_{s-1}))(x - \rho_q(\beta_s)) \\
&= (x - \beta_2)(x - \beta_3)\dots(x - \beta_s)(x - \beta_1) \\
&= m(x).
\end{aligned} \tag{3.5.14}$$

Therefore, by Theorem 3.5.5(4), the coefficients of $m(x)$ are actually in $\mathbf{F}_q$, and not just in E; in other words, $m(x) \in \mathbf{F}_q[x]$.

So now, as in Theorem 3.5.6, let

$$I = \{f(x) \in \mathbf{F}_q[x] \mid f(\beta) = 0\}. \tag{3.5.15}$$

On the one hand, since $m(\beta) = m(\beta_1) = 0$, we see that $m(x) \in I$, which means that $(m(x))$ is contained in I, since I is closed under taking $F_q[x]$-scalar multiples.

On the other hand, suppose $f(x) \in I$, or in other words, suppose that

$$f(x) = a_0 + a_1 x + \dots + a_k x^k \in \mathbf{F}_q[x] \tag{3.5.16}$$

and $f(\beta) = 0$. Applying ρ_q to both sides of $f(\beta) = 0$, we see that

$$\rho_q(a_0 + a_1\beta + \cdots + a_k\beta^k) = \rho_q(0). \tag{3.5.17}$$

Again using the fact that ρ_q is an automorphism (Theorem 3.5.5(4)), we see that

$$\rho_q(a_0) + \rho_q(a_1)\rho_q(\beta) + \cdots + \rho_q(a_k)\rho_q(\beta)^k = 0. \tag{3.5.18}$$

However, since $a_i \in \mathbf{F}_q$, $\rho_q(a_i) = a_i$, and furthermore, $\rho_q(\beta) = \rho_q(\beta_1) = \beta_2$. It follows that

$$f(\beta_2) = a_0 + a_1\beta_2 + \cdots + a_k\beta_2^k = 0. \tag{3.5.19}$$

Repeating this process, we see that $f(\beta_i) = 0$ for $1 \leq i \leq s$, which means that $m(x)$ divides $f(x)$, by the Root Theorem. It follows that I is contained in $(m(x))$, and the theorem follows. □

Example 3.5.10. Consider the extension $E = \mathbf{F}_8$ of $\mathbf{F}_2$, and let α be a primitive root of E. To compute the Frobenius orbit of α^i, we start with α^i and square what we have until we get back to α^i, keeping in mind that $\alpha^7 = 1$. So the Frobenius orbit of α^1 is

$$O_1 = \left\{\alpha^1, \alpha^2, \alpha^4\right\},$$

because when we get to α^8, we return to $\alpha^8 = \alpha^1$. Note that O_1 is also the Frobenius orbit of α^2 and α^4. Putting that together with Theorem 3.5.9, we see that

$$m_1(x) = m_2(x) = m_4(x) = (x - \alpha)(x - \alpha^2)(x - \alpha^4). \tag{3.5.20}$$

Similarly, the Frobenius orbit of α^3 is

$$O_1 = \left\{\alpha^3, \alpha^6, \alpha^5\right\}, \tag{3.5.21}$$

and

$$m_3(x) = m_5(x) = m_6(x) = (x - \alpha^3)(x - \alpha^5)(x - \alpha^6). \tag{3.5.22}$$

Problems

3.5.1. Let α be an element of $\mathbf{F}_8$ such that $\alpha^3 + \alpha + 1 = 0$. Verify by direct computation that

$$x^3 + x + 1 = (x - \alpha)(x - \alpha^2)(x - \alpha^4). \tag{3.5.23}$$

3.5.2. Let E be an extension of $\mathbf{F}_q$, fix some $\beta \in E$, and let

$$I = \{f(x) \in \mathbf{F}_q[x] \mid f(\beta) = 0\}. \tag{3.5.24}$$

This problem proves that I is an ideal in $\mathbf{F}_q[x]$. Note that $0 \in I$ because plugging anything into the zero polynomial gives 0. As for the other parts of the definition of ideal:

(a) Prove that I is closed under addition. (Suggestion: What does it mean to say that $f_1(x), f_2(x) \in I$?)

(b) Prove that I is closed under multiplication by $h(x) \in \mathbf{F}_q[x]$.

3.5.3. Consider the extension $E = \mathbf{F}_{256}$ of $\mathbf{F}_2$, and let α be a primitive root of E. Find the product formulas for $m_1(x), \ldots, m_5(x)$, as in Example 3.5.10.

3.5.4. Consider the extension $E = \mathbf{F}_{512}$ of $\mathbf{F}_2$, and let α be a primitive root of E. Find the product formulas for $m_1(x), \ldots, m_5(x)$, as in Example 3.5.10.

3.6 BCH codes

We first restate the BCH theorem, named after its discoverers Bose and Chaudhuri [citation?], and independently, Hocquenghem [citation?].

Theorem 3.6.1. *Let $\mathcal{C}$ be a cyclic code of length n over $\mathbf{F}_q$ that is generated by the divisor $g(x) \in \mathbf{F}_q[x]$ of $x^n - 1$. Suppose E is an extension field of $\mathbf{F}_q$ such that for some $\delta \in \mathbf{N}$ and some primitive nth root of unity α in E, we have that*

$$0 = g(\alpha) = g(\alpha^2) = g(\alpha^3) = \cdots = g(\alpha^{\delta-1}). \tag{3.6.1}$$

Then the minimum distance d of $\mathcal{C}$ is at least δ, i.e., $d \geq \delta$.

A code of the form described by Theorem 3.5.4 is called a *BCH code*. If $\mathcal{C}$ is a BCH code, We call the quantity δ in Theorem 3.5.4 the *designed distance* of $\mathcal{C}$, because while the theorem assures us that $d \geq \delta$, it may actually be the fact that $d > \delta$, i.e., our code actually turns out to be better than intended (see Example 3.6.5 for an example). In any case, we can now describe the following recipe for building BCH codes.

Algorithm 3.6.2 (Constructing a BCH code)**.** The following recipe constructs a BCH code $\mathcal{C}$ over $\mathbf{F}_q$.

1. Choose an extension E of $\mathbf{F}_q$, and let q^e be the order of E.
2. Choose a primitive nth root of unity $\alpha \in E$ for some $n \in \mathbf{N}$. (If $n = q^e - 1$, then α is also a primitive root of E.)
3. Choose a designed distance $\delta \in \mathbf{N}$.
4. Let $g(x)$ be the least common multiple of $m_1(x), \ldots, m_{\delta-1}(x)$, i.e., remove repetitions of minimal polynomials and take the resulting product.

Then $\mathcal{C}$ is the cyclic code generated by $g(x)$.

Example 3.6.3. Let $E = \mathbf{F}_{128}$, the field of order $128 = 2^7$ (i.e., $e = 7$), and let α be a primitive root of E (so in particular, $\alpha^{127} = 1$). We begin by constructing the corresponding BCH code $\mathcal{C}$ of designed distance $\delta = 13$.

To construct this code, we need to compute $m_i(x)$ by computing Frobenius orbits. The first such orbit is

$$\left\{\alpha^1, \alpha^2, \alpha^4, \alpha^8, \alpha^{16}, \alpha^{32}, \alpha^{64}\right\}, \tag{3.6.2}$$

and we know this orbit is complete because $\alpha^{128} = \alpha^1$. To avoid writing α over and over, we abbreviate this orbit as $\{1, 2, 4, 8, 16, 32, 64\}$.

Next, since we need $g(\alpha^i) = 0$ for $1 \leq i \leq 12$, the next power of α not yet accounted for is α^3, so we compute the orbit (again writing only the exponents and omitting α)

$$\{3, 6, 12, 24, 48, 96, 65\}. \tag{3.6.3}$$

We again finish because $\alpha^{130} = \alpha^3$, and we again see that the next power not yet accounted for is α^5. Continuing this process until we account for all i from 1 to 12, in total, we get:

$$\begin{aligned} &\{1, 2, 4, 8, 16, 32, 64\}, && \{3, 6, 12, 24, 48, 96, 65\}, \\ &\{5, 10, 20, 40, 80, 33, 66\}, && \{7, 14, 28, 56, 112, 97, 67\}, \\ &\{9, 18, 36, 72, 17, 34, 68\}, && \{11, 22, 44, 88, 49, 98, 69\}. \end{aligned} \tag{3.6.4}$$

It follows that $\mathcal{C}$ has generator polynomial

$$g(x) = m_1(x)m_3(x)m_5(x)m_7(x)m_9(x)m_{11}(x), \tag{3.6.5}$$

where, for example,

$$m_7(x) = (x - \alpha^7)(x - \alpha^{14})(x - \alpha^{28})(x - \alpha^{56})(x - \alpha^{112})(x - \alpha^{97})(x - \alpha^{67}). \tag{3.6.6}$$

We similarly see that each $m_i(x)$ ($i = 1, 3, 5, 7, 9, 11$) has degree 7, so in total, $\deg g(x) = 42$.

By Theorem 3.4.3, we see that $\dim \mathcal{C} = 127 - 42 = 85$, and so $\mathcal{C}$ is a $[127, 85, d]$ code, where $d \geq 13$. Note that since $13 = 2(6) + 1$, $\mathcal{C}$ corrects 6 errors per codeword, at least in the abstract.

Remark 3.6.4. We do not discuss explicit error-correction methods here, but efficient error-correction methods do exist; see ??.

Example 3.6.5. Continuing Example 3.6.3, suppose we now try to construct the corresponding BCH code of designed distance $\delta = 16$. As before, we get orbits

$$\{13, 26, 52, 104, 81, 35, 70\}, \qquad \{15, 30, 60, 120, 113, 99, 71\}. \tag{3.6.7}$$

So if

$$g(x) = m_1(x)m_3(x)m_5(x)m_7(x)m_9(x)m_{11}(x)m_{13}(x)m_{15}(x), \tag{3.6.8}$$

we see that $g(\alpha^i) = 0$ for $1 \leq i \leq 15$, as desired. However, looking back at (3.6.4), we see that 16, 17, and 18 have previously appeared in our orbits, which means that

$$g(\alpha^{16}) = g(\alpha^{17}) = g(\alpha^{18}) = 0 \tag{3.6.9}$$

as well. It follows that the corresponding code $\mathcal{C}$ actually has minimum distance at least 19, providing an example of a code where the actual minimum distance is greater than the designed distance $\delta = 16$.

In any case, since $\deg g(x) = 56$, $\dim \mathcal{C} = 127 - 56 = 71$, and $\mathcal{C}$ is a $[112, 56, d]$ code, where $d \geq 19$. Again, because $d = 2(9) + 1$, $\mathcal{C}$ corrects 9 errors per codeword.

Before we can finally prove the BCH Theorem, we record the following linear-algebraic fact.

Lemma 3.6.6. *Let F be a field, and let $\alpha_1, \ldots, \alpha_k$ be pairwise distinct elements of F (i.e., $\alpha_i \neq \alpha_j$ for $i \neq j$). Then the matrix*

$$V = \begin{bmatrix} 1 & 1 & 1 & \cdots & 1 \\ \alpha_1 & \alpha_2 & \alpha_3 & \cdots & \alpha_k \\ \alpha_1^2 & \alpha_2^2 & \alpha_3^2 & \cdots & \alpha_k^2 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ \alpha_1^{k-1} & \alpha_2^{k-1} & \alpha_3^{k-1} & \ldots & \alpha_k^{k-1} \end{bmatrix} \tag{3.6.10}$$

is invertible, and consequently, its columns are linearly independent.

A matrix V of the form in (3.6.10), or the transpose of such a matrix, is called a *Vandermonde matrix*.

Idea of proof. This follows because

$$\det V = \prod_{1 \leq i < j \leq k} (\alpha_j - \alpha_i), \tag{3.6.11}$$

which is nonzero as long as $\alpha_i \neq \alpha_j$ for $i \neq j$. □

Proof of BCH Theorem. if $c(x) = c_0 + \cdots + c_{n-1}x^{n-1} \in \mathcal{C}$, $c(x) = q(x)g(x)$, so $c(\alpha^i) = 0$. In other words,

$$c_0 + c_1\alpha + c_2\alpha^2 \cdots + c_{n-1}\alpha^{n-1} = 0. \tag{3.6.12}$$

In vector form, (3.6.12) becomes

$$[1 \ \alpha \ \alpha^2 \ \ldots \ \alpha^{n-1}] \begin{bmatrix} c_0 \\ \vdots \\ c_{n-1} \end{bmatrix} = 0. \tag{3.6.13}$$

For any fixed $\mathbf{c} = \begin{bmatrix} c_0 \\ \vdots \\ c_{n-1} \end{bmatrix} \in \mathcal{C}$, (3.6.13) holds for $0 \leq i \leq \delta - 1$, so if we define a $(\delta - 1) \times n$ matrix H by

$$H = \begin{bmatrix} 1 & \alpha & \alpha^2 & \cdots & \alpha^{n-1} \\ 1 & \alpha^2 & \alpha^{2(2)} & \cdots & \alpha^{(n-1)2} \\ 1 & \alpha^3 & \alpha^{2(3)} & \cdots & \alpha^{(n-1)3} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & \alpha^{\delta-1} & \alpha^{2(\delta-1)} & \ldots & \alpha^{(n-1)(\delta-1)} \end{bmatrix}, \tag{3.6.14}$$

then for any $\mathbf{c} \in \mathcal{C}$, we have that $H\mathbf{c} = \mathbf{0}$; in other words, $\mathcal{C} \subseteq \text{Null}(H)$.

So now, let $\mathbf{c} \in \mathbf{F}_q^n$ be a vector of (Hamming) weight at most $\delta - 1$, or in other words, suppose $c_i = 0$ except possibly for $\delta - 1$ coordinates $c_{i_1}, \ldots, c_{i_{\delta-1}}$. The corresponding columns $i_1, \ldots, i_{\delta-1}$ of H then form a $(\delta-1) \times (\delta-1)$ matrix

$$H_{\mathbf{c}} = \begin{bmatrix} \alpha^{i_1} & \alpha^{i_2} & \alpha^{i_3} & \cdots & \alpha^{i_{\delta-1}} \\ \alpha^{2i_1} & \alpha^{2i_2} & \alpha^{2i_3} & \cdots & \alpha^{2i_{\delta-1}} \\ \alpha^{3i_1} & \alpha^{3i_2} & \alpha^{3i_3} & \cdots & \alpha^{3i_{\delta-1}} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ \alpha^{(\delta-1)i_1} & \alpha^{(\delta-1)i_2} & \alpha^{(\delta-1)i_3} & \ldots & \alpha^{(\delta-1)i_{\delta-1}} \end{bmatrix} \tag{3.6.15}$$

such that $H_{\mathbf{c}}\mathbf{c} = \mathbf{0}$.

If we now multiply column j of $H_{\mathbf{c}}$ by α^{-i_j} for each j between 1 and $\delta - 1$, an operation that does not change invertibility, we get another matrix

$$V = \begin{bmatrix} 1 & 1 & 1 & \cdots & 1 \\ \alpha^{i_1} & \alpha^{i_2} & \alpha^{i_3} & \cdots & \alpha^{i_{\delta-1}} \\ \alpha^{2i_1} & \alpha^{2i_2} & \alpha^{2i_3} & \cdots & \alpha^{2i_{\delta-1}} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ \alpha^{(\delta-2)i_1} & \alpha^{(\delta-2)i_2} & \alpha^{(\delta-2)i_3} & \ldots & \alpha^{(\delta-2)i_{\delta-1}} \end{bmatrix}. \tag{3.6.16}$$

However, since α is a *primitive* nth root of unity, all powers of α between 0 and $n - 1$ are different, which means that V is a Vandermonde matrix, and therefore, invertible (Lemma 3.6.6).

It follows that $H_{\mathbf{c}}$ is also invertible, and therefore, since $H_{\mathbf{c}}\mathbf{c} = \mathbf{0}$, we must have $\mathbf{c} = \mathbf{0}$. Consequently, $\mathcal{C}$ has no nonzero codewords $\mathbf{c}$ such that $\mathrm{wt}(\mathbf{c}) \leq \delta - 1$, and the theorem follows. □

Remark 3.6.7. Fact: long BCH codes are bad.

Problems

3.6.1. Let $E = \mathbf{F}_{16}$, and let α be a primitive root of unity of E (i.e., $n = 15$).

(a) Let $\delta = 3$, and let $\mathcal{C}$ be the corresponding BCH code obtained by Algorithm 3.6.2. Find the generating polynomial $g(x)$ of $\mathcal{C}$ and $k = \dim \mathcal{C}$. You do not need to multiply $g(x)$ out over $\mathbf{F}_2$; just leave it in product form.

(b) Same, but for $\delta = 5$.

(c) Give an example of a BCH code where the actual minimal distance is greater than the designed minimal distance.

3.6.2. Let $E = \mathbf{F}_{32}$, and let α be a primitive root of unity of E (i.e., $n = 31$).

(a) Let $\delta = 5$, and let $\mathcal{C}$ be the corresponding BCH code obtained by Algorithm 3.6.2. Find the generating polynomial $g(x)$ of $\mathcal{C}$ and $k = \dim \mathcal{C}$. You do not need to multiply $g(x)$ out over $\mathbf{F}_2$; just leave it in product form.

(b) Same, but for $\delta = 7$.

3.6.3. Let $E = \mathbf{F}_{64}$, and let α be a primitive root of unity of E (i.e., $n = 63$).

(a) Let $\delta = 5$, and let $\mathcal{C}$ be the corresponding BCH code obtained by Algorithm 3.6.2. Find the generating polynomial $g(x)$ of $\mathcal{C}$ and $k = \dim \mathcal{C}$. You do not need to multiply $g(x)$ out over $\mathbf{F}_2$; just leave it in product form.

(b) Same, but for $\delta = 7$.

(c) Same, but for $\delta = 9$.

3.7 Reed-Solomon codes

Reed-Solomon codes are constructed by taking the special case of BCH codes when $E = \mathbf{F}_q$ ($q = 2^e$) and thinking of each "byte" in $\mathbf{F}_q$ as a string of e "bits" in $\mathbf{F}_2$. As we will soon see, Reed-Solomon codes are not particularly good codes if errors are randomly scattered in a transmission, but they provide an effective and practical solution to the burst error problem (Motivating Problem 3.1.1).

We now give the recipe for constructing a Reed-Solomon code. First, however, recall that for $q = 2^e$, $\mathbf{F}_q = \mathbf{F}_2[x]/(m(x))$, where $m(x)$ is a degree e polynomial that is irreducible over $\mathbf{F}_2$. In practice, that means that elements of $\mathbf{F}_q$ can be represented as polynomials

$$a_0 + a_1 x + a_2 x^2 + \cdots + a_{e-1} x^{e-1} \in \mathbf{F}_2[x], \tag{3.7.1}$$

which in turn can be represented as a "byte" $(a_0, \ldots, a_{e-1})$ of e bits. Therefore, when we construct a Reed-Solomon code $\mathcal{C}$, we will need to distinguish between $\mathbf{F}_q$-lengths and dimensions, which we denote by the capital letters N and K, and $\mathbf{F}_2$-lengths and dimensions, which we denote by lower-case letters n and k. Note that since each "byte" (element of F_q) is made from e "bits" (elements of $\mathbf{F}_2$), we have that $n = Ne$ and $k = Ke$. We similarly use D and d to denote the minimum F_q-distance and $\mathbf{F}_2$-distance of $\mathcal{C}$.

Algorithm 3.7.1 (Constructing a Reed-Solomon code)**.** The following recipe constructs a Reed-Solomon code $\mathcal{C}$ over a field $\mathbf{F}_q$.

1. Choose $q = 2^e$ for some $e \in \mathbf{N}$, and let $E = \mathbf{F}_q$.
2. Choose a primitive Nth root of unity $\alpha \in E$ for some $N \in \mathbf{N}$. (If $N = q - 1$, then α is also a primitive root of E.)
3. Choose a designed distance $\Delta \in \mathbf{N}$.
4. Since $m_i(x) = (x - \alpha^i)$, let

$$g(x) = (x - \alpha^1)(x - \alpha^2)(x - \alpha^3) \ldots (x - \alpha^{\Delta - 1}). \tag{3.7.2}$$

Then $\mathcal{C}$ is the cyclic code generated by $g(x)$.

First, the bad news: If $\mathcal{C}$ is a Reed-Solomon code, and we look at $\mathcal{C}$ as a code over $\mathbf{F}_2$ with randomly scatttered bit errors, then $\mathcal{C}$ will generally not be so great as a code. The problem is that the bit length $n = Ne$, or e times the byte length, but each randomly scattered bit error can potentially cause a byte error, which effectively makes $d = D$. The correction rate is therefore effectively

$$\frac{d}{n} = \frac{D}{Ne} = \frac{1}{e}\left(\frac{D}{N}\right), \tag{3.7.3}$$

or in other words, the random-bit-error-correction rate is reduced by a factor of e from the random-byte-error-correction rate.

The good news is that Reed-Solomon codes are very good at correcting burst bit errors, as shown by the following example.

Example 3.7.2. Let $E = \mathbf{F}_{512}$, the field of order $512 = 2^9$ (i.e., $e = 9$), and let α be a primitive root of E (so in particular, $\alpha^{511} = 1$). Let $\mathcal{C}$ be the Reed-Solomon code of designed distance $\Delta = 201$. We see that

$$g(x) = \prod_{i=1}^{200}(x - \alpha^i), \tag{3.7.4}$$

a polynomial of degree 200, and since $N = 511$, the "byte" dimension of $\mathbf{C}$ is $K = 311$. Since $e = 9$, we also see that $n = 9(511) = 4599$ and $k = 9(311) = 2799$. Futhermore, since $\Delta = 201 = 2(100) + 1$, we can correct 100 randomly occurring"byte" errors in any codeword, but as mentioned above, we are also only guaranteed to be able to correct 100 randomly occurring"bit" errors, which is not a great ratio when compared to the bit length of 4599.

However, suppose that our bit errors occur not randomly, but instead, in a single burst of length b. We see from Figure 3.7.1, below, that if $b \leq 99(9)+1$ (but not if $b = 99(9)+2$), then that burst will be contained within 100 "bytes", and can therefore be corrected. It follows that within any codeword of length 4599, we can correct any single burst of length at most $99(9) + 1 = 892$.

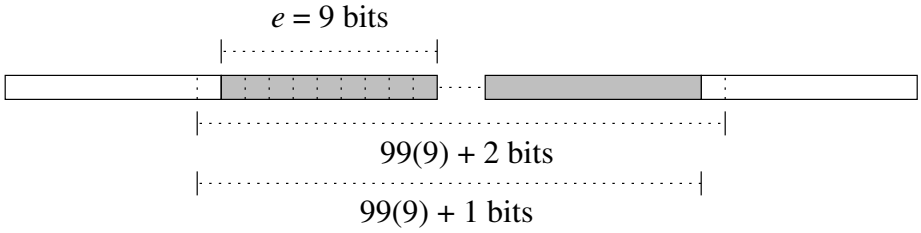

Figure 3.7.1: The longest correctible burst error

There's nothing special about the parameters of Example 3.7.2, so exactly the same argument results in the following theorem.

Theorem 3.7.3 (Reed-Solomon burst correction)**.** *Let $\mathcal{C}$ be a Reed-Solomon code constructed over $\mathbf{F}_q$ ("bytes") with $q = 2^e$ and designed $\mathbf{F}_q$-distance $\Delta = 2T + 1$. Then any burst ("bit") error of length at most $b = (T - 1)e + 1$ within a single codeword can be corrected.*

Proof. Problem 3.7.2. □

Problems

3.7.1. Let $E = \mathbf{F}_{256}$, and let α be a primitive root of unity of E (i.e., $N = 255$).

(a) For the Reed-Solomon code $\mathcal{C}$ with designed $\mathbf{F}_q$-distance $\Delta = 15$, find the generator polynomial $g(x)$, the $\mathbf{F}_q$-dimension K, the $\mathbf{F}_2$-dimension k, and the maximum burst error correction length b.

(b) Same, but for $\Delta = 19$.

3.7.2. longest burst that can be corrected.

Chapter 4

Theory: Groups

4.1 Groups

axioms and whatnot.

4.2 Subgroups

(This follows 11B of your textbook.)

Notation 4.2.1. We use the notation $H \leq G$ to say that H is a subgroup of a group G. The point of using $\leq$ is to distinguish between H being a sub*group* of G and H being merely a sub*set* of G ($H \subseteq G$).

Definition 4.2.2. We define C_n to be the set of all nth roots of unity in $\mathbf{C}$. In other words:

$$C_n = \{z \in \mathbf{C} \mid z^n = 1\}. \tag{4.2.1}$$

As we saw earlier (reference?), if $\omega = e^{2\pi i/n}$, then

$$C_n = \left\{1, \omega, \omega^2, \ldots, \omega^{n-1}\right\}. \tag{4.2.2}$$

Theorem 4.2.3. *For $n, k \in \mathbf{N}$, we have that:*

1. *C_n is a subgroup of $U(\mathbf{C})$ (the nonzero complex numbers under multiplication), and*
2. *If k divides n, then C_k is a subgroup of C_n.*

Note that statement (2) of Theorem 4.2.3 is important to us because subgroup chains like

$$C_2 \leq C_4 \leq C_8 \leq C_{16} \leq \cdots \tag{4.2.3}$$

will later become the basis of the Fast Fourier Transform.

Proof. Problem 4.2.1 shows that:

1. C_n contains the complex number 1;
2. C_n is closed under multiplication; and
3. C_n is closed under inverses.

The first statement of the theorem follows because C_n is a subset of $U(\mathbf{C})$. As for the second statement, if $z^k = 1$ and $n = kd$, then $z^n = z^{kd} = (z^k)^d = 1$, which means that C_k is a subset of C_n. The theorem follows. □

Problems

4.2.1. Prove C_k subgp of C_n.

4.2.2. This problem explores the orders of different elements in C_n.

(a) Let $\omega = e^{2\pi i/10}$. Find the orders of $1, \omega, \omega^2, \ldots, \omega^9 \in C_{10}$.

(b) Now let $\omega = e^{2\pi i/12}$. Find the orders of $1, \omega, \omega^2, \ldots, \omega^{11} \in C_{12}$.

(c) Now consider the general case of $\omega = e^{2\pi i/n}$. What do you think the order of ω^k is? See if you can guess a formula that is a function of n and k.

4.3 Cosets

(This follows 11C of your textbook.)

Theorem 4.3.1. *Let H be a subgroup of a group G, and let a be an element of G. If b is an element of aH, then $aH = bH$.*

Proof. Suppose $b \in aH$, or in other words, $b = ah_0$ for some $h_0 \in H$. Then for any $bh \in bH$, since H is closed under multiplication, $bh = a(h_0 h) \in AH$; and for any $ah \in aH$, since H is closed under multiplication and inverses, $ah = b(h_0^{-1} h) \in bH$. The theorem follows. □

Theorem 4.3.1 motivates the following definition.

Definition 4.3.2. Let H be a subgroup of a group G, and let a be an element of G. A *representative* of the coset aH is an element b of aH. Note that by Theorem 4.3.1, if b is a representative of aH, then bH is an alternate name for aH.

Theorem 4.3.1 also has the following important corollary.

Corollary 4.3.3. *Let H be a subgroup of a group G, and let a and b be an element of G. Then aH and bH are either disjoint or equal.*

In other words, cosets are either disjoint are equal.

Proof. If aH and bH are not disjoint, then there exists some $g \in aH \cap bH$, and by Theorem 4.3.1, $aH = gH = bH$. □

As always, the first thing you should do when you see a new abstract definition is to make up an example and play around with it.

Example 4.3.4. Let $G = U_{13}$, the units of $\mathbf{F}_{13}$, and let $H = \langle 5 \rangle$, the cyclic subgroup of G generated by $5 \in G$. Since $5^2 = 12$, $5^3 = 8$, and $5^4 = 1 \pmod{13}$, we see that

$$H = \{1, 5, 12, 8\}. \tag{4.3.1}$$

Choosing (randomly) $6 \in G$, we see that

$$6H = \{6, 30, 72, 48\} = \{6, 4, 7, 9\} \pmod{13}. \tag{4.3.2}$$

By Theorem 4.3.1, we know that $7H = 6H$, but let's try it anyway. We get:

$$7H = \{7, 35, 84, 56\} = \{7, 9, 6, 4\} \pmod{13}, \tag{4.3.3}$$

which is the same set as $6H$, just written in a different order; in other words, $7H = 6H$.

To get a different coset, we need to choose a representative not already contained in H or $6H$, so let's try $3 \in G$:

$$3H = \{3, 15, 36, 24\} = \{3, 2, 10, 11\} \pmod{13}. \tag{4.3.4}$$

Note that $G = H \cup 6H \cup 3H$, and that all three cosets are disjoint.

Putting all of this coset nonsense together, we have the following important picture: Given a group G and a subgroup H, we can *partition* G into cosets of H, as shown in Figure 4.3.1. The point is that every element of G appears in exactly one coset, which means that, for example, we can "divide and conquer" a sum over G by summing over each coset and summing the totals from each coset, without worrying about missing an element of G or counting an element of G twice.

$G =$

H	aH	bH
cH	dH	gH

Figure 4.3.1: A group G partitioned into six cosets of a subgroup H

To be precise:

Definition 4.3.5. Let X be a set, and let $\{A_1, \ldots, A_n\}$ be a collection of subsets of X. To say that $\{A_1, \ldots, A_n\}$ *partition* X means that:

1. (Nonempty) Each $A_i \neq \emptyset$;
2. (Cover) $X = \bigcup_{i=1}^{n} A_i$ (i.e., X is the union of the A_i); and

3. (Pairwise disjoint) If $i \neq j$, then $A_i \cap A_j = \emptyset$.

Theorem 4.3.6. *Let G be a finite group and let H be a subgroup of G. Consider all left cosets of H, and choose one element a_i from each coset of H so that $\{a_1H, \ldots, a_nH\}$ contains each coset of H exactly once. Then $\{a_1H, \ldots, a_nH\}$ partitions G.*

Proof. Comparing Definition 4.3.5, we first observe that each coset a_iH is nonempty because $a_i \in a_iH$. Next, any $g \in G$ is contained in gH, which must be equal to some a_iH because of the way we chose the a_i. Finally, Corollary 4.3.3 implies that the a_iH are pairwise disjoint. □

Definition 4.3.7. Let G be a finite group, and let H be a subgroup of G. A choice of coset representatives like the set $\{a_1, \ldots, a_n\}$ in the statement of Theorem 4.3.6 is called a *complete set of coset representatives* for H in G.

Example 4.3.8. Returning to the example $G = U_{13}$, $H = \langle 5 \rangle$ of Example 4.3.4, we see that $\{1, 6, 3\}$ is a complete set of coset representatives for H in G, as are $\{1, 7, 3\}$, $\{12, 9, 10\}$, and so on.

Problems

4.3.1. Let $\omega = e^{2\pi i/12}$, and consider the subgroups C_2 and C_6 of C_{12}. (In fact, $C_2 \leq C_6 \leq C_{12}$.)

(a) What are the elements of C_2 in terms of ω? Explain. And what are the elements of C_6 in terms of ω?

(b) Write C_{12} as a disjoint union of cosets of C_2.

(c) Find a complete set of coset representatives for C_2 in C_{12}. See if you can find one chosen by some orderly method.

(d) Find a complete set of coset representatives for C_2 in C_6.

Chapter 5

Faster: The Fast Fourier Transform

5.1 Can we make multiplication faster?

The Fast Fourier Transform (FFT) has been cited as one of the ten most important algorithms of the 20th century, and its applications are almost too numerous to mention (though ?? does a pretty good job of it). However, because this is an algebra book, we'll use a motivating problem that's easy to describe: multiplying large numbers.

Motivating Problem 5.1.1. Using the standard grade-school algorithm for long multiplication, the time required to multiply two N-digit numbers is $O(N^2)$ (see Problem 5.1.1. Is there a faster algorithm for multiplying two N-digit numbers, e.g., an algorithm that's $O(N \log N)$?

If you give Problem 5.1.1 proper consideration, it should blow your mind. How could there possibly be a new and dramatically faster way to perform a calculation that people have probably been doing for thousands of years? Nonetheless, in 1971, Schönhage and Strassen (reference?) devised a practical method for multiplying two N-digit numbers that doesn't quite get the time required down to $O(N \log N)$, but is pretty close. As we'll see, the key idea is that the Discrete Fourier Transform (DFT) reduces standard multiplication to an $O(N)$ operation and so the bottleneck becomes the DFT itself, a problem solved by the FFT. We start that discussion in our next section.

Problems

5.1.1. Recall the usual grade-school algorithm for multiplication of two N-digit numbers.

(a) Suppose we multiply $1{,}234 \times 5{,}432$. Carefully count the number of single-digit multiplications and the number of additions that are required to carry out the usual grade-school algorithm on these two numbers. (Ignore carrying.)

(b) Generalize the previous part to see how long it takes to multiply two N-digit numbers. How many single-digit multiplications, as a function of N? Find an upper bound for the number of additions as well (easier than the exact number). (You can again ignore carrying, which is an $O(N)$ factor, but in perhaps a non-obvious way.)

5.2 The Discrete Fourier Transform

To provide just a tiny bit of context and motivation for the *Discrete Fourier Transform* (DFT), we define the following class of functions.

Definition 5.2.1. Fix $N \in \mathbf{N}$. We define a *signal* to be a function $f : \mathbf{Z}/N \to \mathbf{C}$, or in other words, a complex-valued function with domain $\mathbf{Z}/N$. Note that a signal f is defined by its N values $f(0), \ldots, f(N-1) \in \mathbf{C}$, so we sometimes represent a signal f in vector form as $\begin{bmatrix} f(0) \\ \vdots \\ f(N-1) \end{bmatrix}$.

Definition 5.2.2. Fix $N \in \mathbf{N}$, let $\omega = e^{2\pi i/N}$ be the natural primitive Nth root of unity in $\mathbf{C}$, and let $f : \mathbf{Z}/N \to \mathbf{C}$ be a signal. We define the *Discrete Fourier Transform*, or *DFT*, of f to be the function $\hat{f} : \mathbf{Z}/N \to \mathbf{C}$ given by

$$\hat{f}(k) = \frac{1}{N} \sum_{n=0}^{N-1} f(n)\omega^{-nk}. \tag{5.2.1}$$

Note that while the DFT of a signal f is, by definition, itself a signal, it's more helpful to think of $\hat{f}(k)$ as the *spectrum* of f, as roughly speaking, $\hat{f}(k)$ measures the strength of the part of f that has "frequency k". (See Theorem 5.2.6, below, for a justification for this idea.)

Remark 5.2.3. Looking at (5.2.1) for $k = 0, 1, 2, 3$, we get

$$\begin{aligned}
\hat{f}(0) &= \frac{1}{N}(f(0) + f(1) + f(2) + \cdots + f(N-1)), \\
\hat{f}(1) &= \frac{1}{N}(f(0) + \omega^{-1}f(1) + \omega^{-2}f(2) + \cdots + \omega^{-(N-1)}f(N-1)), \\
\hat{f}(2) &= \frac{1}{N}(f(0) + \omega^{-2}f(1) + \omega^{-2(2)}f(2) + \cdots + \omega^{-2(N-1)}f(N-1)), \\
\hat{f}(3) &= \frac{1}{N}(f(0) + \omega^{-3}f(1) + \omega^{-3(2)}f(2) + \cdots + \omega^{-3(N-1)}f(N-1)).
\end{aligned} \tag{5.2.2}$$

In fact, if we write out (5.2.1) for $k = 0, \ldots, N-1$ and express the result in terms of matrix-vector multiplication, we get

$$\begin{bmatrix} \hat{f}(0) \\ \vdots \\ \hat{f}(N-1) \end{bmatrix} = \frac{1}{N} \begin{bmatrix} 1 & 1 & 1 & \ldots & 1 \\ 1 & \omega^{-1} & \omega^{-2} & \ldots & \omega^{-(N-1)} \\ 1 & \omega^{-2} & \omega^{-2(2)} & \ldots & \omega^{-2(N-1)} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & \omega^{-(N-1)} & \omega^{-2(N-1)} & \ldots & \omega^{-(N-1)(N-1)} \end{bmatrix} \begin{bmatrix} f(0) \\ \vdots \\ f(N-1) \end{bmatrix}. \tag{5.2.3}$$

Consequently, the DFT itself is an $O(N^2)$ process, just like multiplying an $N \times N$ matrix times a column vector of length N.

Our next goal is to show that the DFT is invertible, and that its inverse is a process that is very similar to the DFT itself. We therefore optimistically make the following definition.

Definition 5.2.4. Let $\hat{f} : \mathbf{Z}/N \to \mathbf{C}$ be a spectrum function. The *inverse DFT* of $\hat{f}$ is defined to be

$$\sum_{k=0}^{N-1} \hat{f}(k)\omega^{kn}. \tag{5.2.4}$$

Note that this is the same transformation as the DFT (Definition 5.2.2), but with a sign change and without the factor of $\frac{1}{N}$.

Before we can prove that the inverse DFT lives up to its name, we need the following lemma.

Lemma 5.2.5 (Orthogonality Lemma)**.** *Fix $N \in \mathbf{N}$ and let $\omega = e^{2\pi i/N}$ be the natural primitive Nth root of unity in $\mathbf{C}$. For $t \in \mathbf{Z}/N$, we have:*

$$\sum_{k=0}^{N-1} \omega^{kt} = \begin{cases} N & \text{if } t = 0 \pmod N, \\ 0 & \text{otherwise.} \end{cases} \tag{5.2.5}$$

Proof. See Problem 5.2.1. □

We now prove that the inverse DFT works as advertised.

Theorem 5.2.6 (Inversion Theorem)**.** *Fix $N \in \mathbf{N}$, let $\omega = e^{2\pi i/N}$ be the natural primitive Nth root of unity in $\mathbf{C}$, and let $f : \mathbf{Z}/N \to \mathbf{C}$ be a signal. If $\hat{f}$ is the DFT of f, then*

$$f(n) = \sum_{k=0}^{N-1} \hat{f}(k)\omega^{kn}. \tag{5.2.6}$$

In other words, taking the inverse transform, which again is basically the DFT with a sign change, recovers our original signal f.

Proof. Considering the right-hand side of (5.2.6) with the variable x in place of the variable n, we see that

$$\begin{aligned} \sum_{k=0}^{N-1} \hat{f}(k)\omega^{kx} &= \frac{1}{N} \sum_{k=0}^{N-1} \left(\sum_{n=0}^{N-1} f(n)\omega^{-nk} \right) \omega^{kx} \\ &= \frac{1}{N} \sum_{n=0}^{N-1} f(n) \sum_{k=0}^{N-1} \omega^{k(x-n)} \qquad (*) \\ &= \frac{1}{N}(f(x)N) \qquad (**) \\ &= f(x), \end{aligned} \tag{5.2.7}$$

where (*) follows by switching the order of summation, and (**) follows by the Orthogonality Lemma 5.2.5. The theorem follows. □

Remark 5.2.7. Extending Remark 5.2.3, we see that Theorem 5.2.6 implies that

$$\begin{aligned}&\left(\frac{1}{N}\begin{bmatrix}1 & 1 & 1 & \dots & 1\\ 1 & \omega^{-1} & \omega^{-2} & \dots & \omega^{-(N-1)}\\ 1 & \omega^{-2} & \omega^{-2(2)} & \dots & \omega^{-2(N-1)}\\ \vdots & \vdots & \vdots & \ddots & \vdots\\ 1 & \omega^{-(N-1)} & \omega^{-2(N-1)} & \dots & \omega^{-(N-1)(N-1)}\end{bmatrix}\right)^{-1}\\ &=\begin{bmatrix}1 & 1 & 1 & \dots & 1\\ 1 & \omega^{1} & \omega^{2} & \dots & \omega^{(N-1)}\\ 1 & \omega^{2} & \omega^{2(2)} & \dots & \omega^{2(N-1)}\\ \vdots & \vdots & \vdots & \ddots & \vdots\\ 1 & \omega^{(N-1)} & \omega^{2(N-1)} & \dots & \omega^{(N-1)(N-1)}\end{bmatrix}.\end{aligned} \tag{5.2.8}$$

Problems

5.2.1. (*Proves Lemma 5.2.5*) Fix $N \in \mathbf{N}$, and let $\omega = e^{2\pi i/N}$. Let $f(x) = x^N - 1$.

(a) Explain why

$$f(x) = x^N - 1 = (x-1)\left(\sum_{k=0}^{N-1} x^k\right). \tag{5.2.9}$$

(Suggestion: Try writing out the sum as $1 + x + \dots$.)

(b) Explain why for any $t \in \mathbf{Z}/N$, $f(\omega^t) = 0$.

(c) Prove that for any $t \in \mathbf{Z}/N$,

$$\sum_{k=0}^{N-1} \omega^{kt} = \begin{cases} N & \text{if } t = 0 \pmod N,\\ 0 & \text{if } t \neq 0 \pmod N.\end{cases} \tag{5.2.10}$$

Suggestion: Use the fact that $a, b \in \mathbf{C}$, if $ab = 0$, then either $a = 0$ or $b = 0$.

5.3 Convolution

The DFT, like all Fourier transforms, has many remarkable properties that we won't have time to get into. The one we will look at is its interaction with the property known as *convolution*.

Definition 5.3.1. Let $f, g : \mathbf{Z}/N \to \mathbf{C}$ be signals. We define the *convolution* of f and g to be the signal $f * g : \mathbf{Z}/N \to \mathbf{C}$ defined by

$$(f * g)(n) = \frac{1}{N}\sum_{t=0}^{N-1} f(n-t)g(t). \tag{5.3.1}$$

Definition 5.3.1 will almost certainly seem strange and unmotivated to you, but it can be justified in terms of two important properties. The first is in terms of multiplication of polynomials.

Theorem 5.3.2. *Let $f, g : \mathbf{Z}/N \to \mathbf{C}$ be signals. Then in the ring $\mathbf{C}[x]/(x^N - 1)$, we have that*

$$\left(\frac{1}{N}\sum_{k=0}^{N-1} f(k)x^k\right)\left(\frac{1}{N}\sum_{m=0}^{N-1} g(m)x^m\right) = \frac{1}{N}\sum_{n=0}^{N-1}(f * g)(n)x^n. \tag{5.3.2}$$

In other words, after scaling appropriately, the convolution $f * g$ gives the coefficients of the product of the polynomials whose coefficients are given by f and g, as long as we are working (mod $(x^N - 1)$) (i.e., taking $x^N = 1$).

Before proving Theorem 5.3.2, we'll need the following lemma.

Lemma 5.3.3 (Substitution Lemma)**.** *Let $F : \mathbf{Z}/N \to \mathbf{C}$ be a complex-valued function on $\mathbf{Z}/N$. Then for any $t \in \mathbf{Z}/N$, we have that*

$$\sum_{k=0}^{N-1} F(k) = \sum_{n=0}^{N-1} F(n - t). \tag{5.3.3}$$

Note that we can think of (5.3.3) as justifying "summation by substitution $k = n - t$", in analogy with integration by substitution.

Proof. See Problem 5.3.1. □

Proof of Theorem 5.3.2. Replacing m with t in (5.3.2), we have that

$$\begin{aligned}\left(\frac{1}{N}\sum_{k=0}^{N-1} f(k)x^k\right)\left(\frac{1}{N}\sum_{t=0}^{N-1} g(t)x^t\right) &= \frac{1}{N^2}\sum_{t=0}^{N-1}\sum_{k=0}^{N-1} f(k)g(t)x^{k+t} \\ &= \frac{1}{N^2}\sum_{t=0}^{N-1}\sum_{n=0}^{N-1} f(n-t)g(t)x^{(n-t)+t} \quad (*) \\ &= \frac{1}{N}\sum_{n=0}^{N-1}\left(\frac{1}{N}\sum_{t=0}^{N-1} f(n-t)g(t)\right)x^n \\ &= \frac{1}{N}\sum_{n=0}^{N-1}(f * g)(n)x^n,\end{aligned} \tag{5.3.4}$$

where we switch the order of summation twice, and we use the substitution $k = n - t$ in (*), by the Substitution Lemma 5.3.3. The theorem follows. □

The other key property of convolution is that the DFT turns convolution into pointwise multiplication. More precisely, we have the following theorem.

Theorem 5.3.4. *Let $f, g : \mathbf{Z}/N \to \mathbf{C}$ be signals. We have that*

$$\widehat{(f * g)}(k) = \hat{f}(k)\hat{g}(k). \tag{5.3.5}$$

The significance of Theorem 5.3.4 is that computing the coefficients of $f * g$ directly is an $O(N^2)$ operation, as it is essentially the same as long multiplication of N-digit numbers, whereas computing $\hat{f}(k)\hat{g}(k)$ for all $k \in \mathbf{Z}/N$ is only an $O(N)$ operation. Now, on the one hand, the DFT is itself an $O(N^2)$ operation, which would seem to make this "shortcut" not very useful. However, if we can replace the direct DFT with something faster, then since the DFT is now the bottleneck in multiplication, we will then get a faster algorithm for multiplication.

Proof of Theorem 5.3.4. We have that

$$\begin{aligned}
\widehat{(f * g)}(k) &= \frac{1}{N}\sum_{n=0}^{N-1}(f * g)(n)\omega^{-nk} && (*)\\
&= \frac{1}{N}\sum_{n=0}^{N-1}\frac{1}{N}\left(\sum_{t=0}^{N-1} f(n-t)g(t)\right)\omega^{-nk} && (**)\\
&= \frac{1}{N^2}\sum_{t=0}^{N-1}\sum_{u=0}^{N-1} f(u)g(t)\omega^{-(u+t)k} && (***)\\
&= \left(\frac{1}{N}\sum_{u=0}^{N-1} f(u)\omega^{-uk}\right)\left(\frac{1}{N}\sum_{u=0}^{N-1} g(t)\omega^{-tk}\right)\\
&= \hat{f}(k)\hat{g}(k),
\end{aligned} \tag{5.3.6}$$

where (*) is the definition of the DFT of $f * g$ (Definition 5.2.2), (**) is the definition of $f * g$ (Definition 5.3.1), and (***) is the substitution $u = n - t$ (Substitution Lemma 5.3.3). The theorem follows. □

Problems

5.3.1. (a) Consider $\mathbf{Z}/N = \{0, 1, \ldots, N-1\}$ and suppose $t \in \mathbf{Z}/N$. What set do we get if we add t to each element of $\mathbf{Z}/N$? (Suggestion: Try some random values of N and t and see what happens; then generalize.)

(b) Now let $F : \mathbf{Z}/N \to \mathbf{C}$ be a complex-valued function on $\mathbf{Z}/N$, and fix some $t \in \mathbf{Z}/N$. Explain why

$$\sum_{k=0}^{N-1} F(k) = \sum_{n=0}^{N-1} F(n-t). \tag{5.3.7}$$

Suggestion: Try writing out both sides of (5.3.7).

5.3.2. compare time estimates for multiplications

5.4 The Fast Fourier Transform

As mentioned in the previous section, we now have the following motivating problem.

Motivating Problem 5.4.1. If we can speed up the computation of the DFT, we can speed up multiplication of polynomials of degree N, and therefore, multiplication of N-digit numbers.

The basic method of speeding up the DFT is called the *Fast Fourier Transform*, or *FFT*. The idea is due to Cooley and Tukey, though our description of it is based on Diaconis and Rockmore.

Algorithm 5.4.2 (FFT)**.** Fix $N \in \mathbf{N}$ and $\omega = e^{2\pi i/N}$. Let

$$C_1 = H_0 \leq H_1 \leq \cdots \leq H_{n-1} \leq H_n = C_N \tag{5.4.1}$$

be a chain of subgroups of C_N that starts at the trivial subgroup $C_1 = \{1\}$ and ends at the group C_N itself. We define the *Fast Fourier Transform based on the subgroup chain* $H_0 \leq \cdots \leq H_n$ as follows.

Throughout this algorithm, let $\mathbf{x} = \begin{bmatrix} x(0) \\ \vdots \\ x(N-1) \end{bmatrix}$ represent the current state of our calculation and let $\mathbf{y} = \begin{bmatrix} y(0) \\ \vdots \\ y(N-1) \end{bmatrix}$ represent the new state of our calculation after the current step. The goal is to start with $\mathbf{x} = \begin{bmatrix} f(0) \\ \vdots \\ f(N-1) \end{bmatrix}$ and end with $\mathbf{x} = \begin{bmatrix} \hat{f}(0) \\ \vdots \\ \hat{f}(N-1) \end{bmatrix}$.

1. *Initialize.* Set $\mathbf{x} = \begin{bmatrix} f(0) \\ \vdots \\ f(N-1) \end{bmatrix}$.

2. *Main loop.* For $i = 1$ to n:

   (a) *Notation.* Suppose that $H_{i-1} = \langle \omega^m \rangle$ and $H_i = \langle \omega^k \rangle$ for the standard smallest possible positive choices of m and k. (Special case: For $i = 1$ and $H_{i-1} = H_0 = \{1\}$, we take $m = N$.) Since $H_{i-1} \leq H_i$, we must have that k divides m, or $m = kd$ for some positive integer d. Use the standard complete set of coset representatives $1, \omega^k, \omega^{2k}, \ldots, \omega^{(d-1)k}$ for H_{i-1} in H_i.

(b) *Set entries of* $\mathbf{y}$ *corresponding to the new subgroup* H_i. For $j = 0$ to $(N/k) - 1$ (i.e., jk ranges over all exponents of ω appearing in H_i), set

$$\begin{aligned} y(jk) &= \sum_{r=0}^{d-1} x(jm + kr)\omega^{-rkj} \\ &= x(jm) + x(jm + k)\omega^{-kj} + x(jm + 2k)\omega^{-2kj} + \dots \\ &\qquad + x(jm + (d-1)k)\omega^{-(d-1)kj}. \end{aligned} \tag{5.4.2}$$

As will be clearer in examples, in the sum on the right-hand side of (5.4.2), we can think of jm as an "offset" (starting point coming from the old subgroup H_{i-1}) and kr as stepping through the exponents of the coset representatives $1, \omega^k, \omega^{2k}, \dots, \omega^{(d-1)k}$. Note that in the expression $x(jm + kr)$ in (5.4.2), we calculate $jm + kr$ as an integer (mod N), so for example, as j ranges from 0 to N/k, jm cycles through each of the integers $0, m, 2m, \dots, N - m$ exactly d times.

(c) *Translate the subgroup fill to entries corresponding to the cosets of* H_i *in* C_N. That is, do exactly the same filling as in step 2b, but with the indices of y and x increased by $1, \dots, k-1$. To be precise, for $\ell = 1$ to $k-1$ and $j = 0$ to $(N/k)-1$, we set

$$y(jk + \ell) = \sum_{r=0}^{d-1} x(jm + kr + \ell)\omega^{-jkr}. \tag{5.4.3}$$

Note that in principle (and certainly in any code that you would actually want to run) we could actually combine this step and the previous step into one big loop, starting with for $\ell = 0$ to $k-1$, but we hope that separating the two makes our algorithm slightly more human-readable.

(d) *Set current state to new state and loop.* Set $\mathbf{x} = \mathbf{y}$ and go to the next step of the main loop.

3. *Rescale.* Divide every entry of $\mathbf{x}$ by N.

Example 5.4.3. For any $N \in \mathbf{N}$, consider the FFT based on the subgroup chain $C_1 \leq C_N$; in other words, $H_0 = C_1$ and $H_1 = C_N$. Note that since we only have one step in the subgroup chain, $n = 1$. Therefore, stepping through the algorithm, we get:

1. *Initialize.* Set $\mathbf{x} = \begin{bmatrix} f(0) \\ \vdots \\ f(N-1) \end{bmatrix}$.

2. *Main loop.* This has one step in it, with $i = n = 1$.

(a) *Notation.* Since $H_0 = \langle \omega^N \rangle$ and $H_1 = \langle \omega^1 \rangle$, we have $m = N$, $k = 1$, $d = N$, and our coset representatives for H_0 in H_1 are $1, \omega, \omega^2, \dots, \omega^{N-1}$, or in other words, the elements of $G = C_N$.

(b) *Fill entries of* $\mathbf{y}$ *corresponding to* H_1. Since $d = N$, $k = 1$, $N/k = N$, and $m = 0 \pmod N$, for $j = 0$ to $N - 1$, set

$$y(j) = \sum_{r=0}^{N-1} x(r)\omega^{-rj}. \tag{5.4.4}$$

(c) *Translate to cosets.* Since the only coset of $H_1 = C_N$ in C_N is itself, nothing happens in this step.

(d) Set $\mathbf{x} = \mathbf{y}$.

3. *Rescale.* Divide every entry of $\mathbf{x}$ by N.

Comparing (5.4.4) and (5.2.1), we see that after the rescaling step, we indeed have $\mathbf{x} = \begin{bmatrix} \hat{f}(0) \\ \vdots \\ \hat{f}(N-1) \end{bmatrix}$. However, we have saved no time: Since we are doing exactly the same computation as the regular DFT, this "FFT" is just the DFT in more complicated notation, and is therefore still an $O(N^2)$ algorithm.

While Example 5.4.3 does reassure us that we haven't gone completely off the deep end here, we also see that if we do not try to cut N up into small pieces, then no time savings will occur. More precisely, in general, if we take D to be the largest value of d that appears in any step of the main loop, then since $n = O(\log N)$, we have that the time complexity of our FFT will be

$$O(DnN) = O(DN \log N). \tag{5.4.5}$$

So as long as D is independent of N and very small compared to the size of N ($D = 2$ is common, but $D = 3$ or $D = 5$ might also be reasonable), then we can expect that the FFT will run in time $O(N \log N)$, where the D has been absorbed into the unstated constant.

We next consider a more typical example of an FFT.

Example 5.4.4. For $N = 6$, consider the FFT based on the subgroup chain $C_1 \leq C_2 \leq C_6$ (so $n = 2$); that is, $H_0 = C_1$, $H_1 = C_2$, and $H_2 = C_6$. Stepping through the algorithm:

1. *Initialize.* Set $\mathbf{x} = \begin{bmatrix} f(0) \\ \vdots \\ f(N-1) \end{bmatrix}$.

2. *Main loop.* This has two steps in it, with $i = 1$ and $i = 2$.

- $i = 1$.

(a) *Notation.* Since $H_0 = \langle \omega^6 \rangle$ and $H_1 = \langle \omega^3 \rangle$, we have $m = 6$, $k = 3$, $N/k = 2$, $d = 2$, with coset representatives $1, \omega^3$.

(b) *Fill entries of* $\mathbf{y}$ *corresponding to* H_1. Going from $j = 0$ to 1, we have

$$\begin{aligned} y(0) &= x(0) + x(3)\omega^{-0} = f(0) + f(3), \\ y(3) &= x(0) + x(3)\omega^{-3} = f(0) + f(3)\omega^{-3}. \end{aligned} \tag{5.4.6}$$

(c) *Translate to cosets.* Similarly, we have

$$\begin{aligned} y(1) &= x(1) + x(4)\omega^{-0} = f(1) + f(4), \\ y(4) &= x(1) + x(4)\omega^{-3} = f(1) + f(4)\omega^{-3}, \\ y(2) &= x(2) + x(5)\omega^{-0} = f(2) + f(5), \\ y(5) &= x(2) + x(5)\omega^{-3} = f(2) + f(5)\omega^{-3}. \end{aligned} \tag{5.4.7}$$

(d) Set

$$\mathbf{x} = \mathbf{y} = \begin{bmatrix} f(0) + f(3) \\ f(1) + f(4) \\ f(2) + f(5) \\ f(0) + f(3)\omega^{-3} \\ f(1) + f(4)\omega^{-3} \\ f(2) + f(5)\omega^{-3} \end{bmatrix}. \tag{5.4.8}$$

- $i = 2$.
  (a) *Notation.* Since $H_1 = \langle \omega^3 \rangle$ and $H_2 = \langle \omega^1 \rangle$, we have $m = 3$, $k = 1$, $N/k = 6$, $d = 3$, with coset representatives $1, \omega, \omega^2$.
  (b) *Fill entries of* $\mathbf{y}$ *corresponding to* H_2. Going from $j = 0$ to 5, we have

$$\begin{aligned} y(0) &= x(0) + x(1)\omega^{-0} + x(2)\omega^{-0} \\ &= f(0) + f(3) + f(1) + f(4) + f(2) + f(5), \\ y(1) &= x(3) + x(4)\omega^{-1} + x(5)\omega^{-2} \\ &= f(0) + f(3)\omega^{-3} + f(1)\omega^{-1} + f(4)\omega^{-4} + f(2)\omega^{-2} + f(5)\omega^{-5}, \\ y(2) &= x(0) + x(1)\omega^{-2} + x(2)\omega^{-4} \\ &= f(0) + f(3) + f(1)\omega^{-2} + f(4)\omega^{-2} + f(2)\omega^{-4} + f(5)\omega^{-4}, \\ y(3) &= x(3) + x(4)\omega^{-3} + x(5)\omega^{-6} \\ &= f(0) + f(3)\omega^{-3} + f(1)\omega^{-3} + f(4) + f(2) + f(5)\omega^{-3}, \\ y(4) &= x(0) + x(1)\omega^{-4} + x(2)\omega^{-8} \\ &= f(0) + f(3) + f(1)\omega^{-4} + f(4)\omega^{-4} + f(2)\omega^{-2} + f(5)\omega^{-2}, \\ y(5) &= x(3) + x(4)\omega^{-5} + x(5)\omega^{-10} \\ &= f(0) + f(3)\omega^{-3} + f(1)\omega^{-5} + f(4)\omega^{-2} + f(2)\omega^{-4} + f(5)\omega^{-1}. \end{aligned} \tag{5.4.9}$$

  (c) *Translate to cosets.* $H_2 = C_6$, so nothing happens.
  (d) Set $\mathbf{x} = \mathbf{y}$.

3. *Rescale.* Divide every entry of $\mathbf{x}$ by $N = 6$.

And lo and behold, our final answer is

$$\mathbf{x} = \begin{bmatrix} f(0) + f(1) + f(2) + f(3) + f(4) + f(5) \\ f(0) + f(1)\omega^{-1} + f(2)\omega^{-2} + f(3)\omega^{-3} + f(4)\omega^{-4} + f(5)\omega^{-5} \\ f(0) + f(1)\omega^{-2} + f(2)\omega^{-4} + f(3) + f(4)\omega^{-2} + f(5)\omega^{-4} \\ f(0) + f(1)\omega^{-3} + f(2) + f(3)\omega^{-3} + f(4) + f(5)\omega^{-3} \\ f(0) + f(1)\omega^{-4} + f(2)\omega^{-2} + f(3) + f(4)\omega^{-4} + f(5)\omega^{-2} \\ f(0) + f(1)\omega^{-5} + f(2)\omega^{-4} + f(3)\omega^{-3} + f(4)\omega^{-2} + f(5)\omega^{-1} \end{bmatrix} = \begin{bmatrix} \hat{f}(0) \\ \vdots \\ \hat{f}(N-1) \end{bmatrix}. \tag{5.4.10}$$

In other words, it works!

Remark 5.4.5. Examining Example 5.4.4, when we look at the time the algorithm takes, we see that:

- Each step in the main loop requires dN multiplications and $(d-1)N$ additions, where $d = 2$ for $i = 1$ and $d = 3$ for $i = 2$.
- There are n steps in the main loop.

So in total, the algorithm requires no more than $2dnN$ operations.

More generally, for any case of the FFT, if D is the maximum value of d as i goes from 1 to n, we see that the FFT takes $O(DnN)$ time. If D stays constant as N increases (e.g., if D is fixed at 2 or 3), then by absorbing D into the unstated constant, we see that the FFT takes

$$O(nN) = O(N \log N) \tag{5.4.11}$$

time. Here $n \leq \log_2 N$ because the size of the subgroup H_i grows by a factor of at least 2 each time, starting with $|H_0| = 1$ and increasing to $|H_N| = N$.

Now, to make any progress in life, one should go as far as any sane person would go and then keep going. With that in mind, we have the following example.

Example 5.4.6. For $N = 16$, consider the FFT based on the subgroup chain $C_1 \leq C_2 \leq C_4 \leq C_8 \leq C_{16}$ (so $n = 4$); in other words, $H_0 = C_1$, $H_1 = C_2$, $H_2 = C_4$, $H_3 = C_8$, and $H_4 = C_{16}$. Let's step through the algorithm as in Example 5.4.6, though we will omit many details, for the sake of both typography and the limits of your patience.

1. *Initialize.* Set $\mathbf{x} = \begin{bmatrix} f(0) \\ \vdots \\ f(N-1) \end{bmatrix}$.
2. *Main loop.* This goes from $i = 1$ to 4.
   - $i = 1$.
     (a) *Notation.* Since $H_0 = \langle \omega^{16} \rangle$ and $H_1 = \langle \omega^8 \rangle$, we have $m = 16$, $k = 8$, $N/k = 2$, $d = 2$, with coset representatives $1, \omega^8$.

(b) *Fill entries of* $\mathbf{y}$ *corresponding to* H_1. We have

$$\begin{aligned} y(0) &= x(0) + x(8)\omega^{-0} = f(0) + f(8), \\ y(8) &= x(0) + x(8)\omega^{-8} = f(0) + f(8)\omega^{-8}. \end{aligned} \tag{5.4.12}$$

(c) *Translate and loop.* We then set, for example

$$\begin{aligned} y(1) &= x(1) + x(9)\omega^{-0} = f(1) + f(9), \\ y(2) &= x(2) + x(10)\omega^{-0} = f(2) + f(10), \\ y(9) &= x(1) + x(9)\omega^{-8} = f(1) + f(9)\omega^{-8}, \\ y(10) &= x(2) + x(10)\omega^{-8} = f(2) + f(10)\omega^{-8}, \end{aligned} \tag{5.4.13}$$

and so on. Then set $\mathbf{x} = \mathbf{y}$, as usual.

- $i = 2$.
  (a) *Notation.* $H_1 = \langle \omega^8 \rangle$, $H_2 = \langle \omega^4 \rangle$, $m = 8$, $k = 4$, $N/k = 4$, $d = 2$, representatives $1, \omega^4$.
  (b) *Fill entries of* $\mathbf{y}$ *corresponding to* H_2.

$$\begin{aligned} y(0) &= x(0) + x(4)\omega^{-0} = f(0) + f(8) + f(4) + f(12), \\ y(4) &= x(8) + x(12)\omega^{-4} = f(0) + f(8)\omega^{-8} + f(4)\omega^{-4} + f(12)\omega^{-12}, \\ y(8) &= x(0) + x(4)\omega^{-8} = f(0) + f(8) + f(4)\omega^{-8} + f(12)\omega^{-8}, \\ y(12) &= x(8) + x(12)\omega^{-12} = f(0) + f(8)\omega^{-8} + f(4)\omega^{-12} + f(12)\omega^{-4}. \end{aligned} \tag{5.4.14}$$

  (c) *Translate and loop.* Translate $y(0)$ to $y(1)$, $y(2)$, and $y(3)$ (details omitted) and similarly for all other empty entries; and set $\mathbf{x} = \mathbf{y}$.
- $i = 3$.
  (a) *Notation.* $H_2 = \langle \omega^4 \rangle$, $H_3 = \langle \omega^2 \rangle$, $m = 4$, $k = 2$, $N/k = 8$, $d = 2$, representatives $1, \omega^2$.
  (b) *Fill entries of* $\mathbf{y}$ *corresponding to* H_3. See Figure 5.4.1.
  (c) *Translate and loop.* The one coset translation is precisely Figure 5.4.1 with every index in $x(n)$ or $y(n)$ (but not $f(n)$) increased by 1, so we omit the details. We then set $\mathbf{x} = \mathbf{y}$ and loop.
- $i = 4$.
  (a) *Notation.* $H_3 = \langle \omega^2 \rangle$, $H_3 = \langle \omega^1 \rangle$, $m = 2$, $k = 1$, $N/k = 16$, $d = 2$, representatives $1, \omega$.
  (b) *Fill entries of* $\mathbf{y}$ *corresponding to* H_4. This now means that we fill every row, and in fact, each row is filled with the corresponding correct value of the DFT, up to rescaling. We omit the details here, due to space and typography, but you should check this yourself by trying a few entries, starting from the $i = 3$ state (i.e., Figure 5.4.1 and its translate to the odd-numbered entries).

3. *Rescale.* Divide every entry of $\mathbf{x}$ by $N = 16$.

$$
\begin{aligned}
y(0) &= x(0) + x(2)\omega^{-0} \\
&= f(0) + f(4) + f(8) + f(12) \\
&\quad + f(2) + f(6) + f(10) + f(14) \\
y(2) &= x(4) + x(6)\omega^{-2} \\
&= f(0) + f(4)\omega^{-4} + f(8)\omega^{-8} + f(12)\omega^{-12} \\
&\quad + f(2)\omega^{-2} + f(6)\omega^{-6} + f(10)\omega^{-10} + f(14)\omega^{-14}, \\
y(4) &= x(8) + x(10)\omega^{-4} \\
&= f(0) + f(4)\omega^{-8} + f(8) + f(12)\omega^{-8} \\
&\quad + f(2)\omega^{-4} + f(6)\omega^{-12} + f(10)\omega^{-4} + f(14)\omega^{-12}, \\
y(6) &= x(12) + x(14)\omega^{-6} \\
&= f(0) + f(4)\omega^{-12} + f(8)\omega^{-8} + f(12)\omega^{-4} \\
&\quad + f(2)\omega^{-6} + f(6)\omega^{-2} + f(10)\omega^{-14} + f(14)\omega^{-10}, \\
y(8) &= x(0) + x(2)\omega^{-8} \\
&= f(0) + f(4) + f(8) + f(12) \\
&\quad + f(2)\omega^{-8} + f(6)\omega^{-8} + f(10)\omega^{-8} + f(14)\omega^{-8}, \\
y(10) &= x(4) + x(6)\omega^{-10} \\
&= f(0) + f(4)\omega^{-4} + f(8)\omega^{-8} + f(12)\omega^{-12} \\
&\quad + f(2)\omega^{-10} + f(6)\omega^{-14} + f(10)\omega^{-2} + f(14)\omega^{-6}, \\
y(12) &= x(8) + x(10)\omega^{-12} \\
&= f(0) + f(4)\omega^{-8} + f(8) + f(12)\omega^{-8} \\
&\quad + f(2)\omega^{-12} + f(6)\omega^{-4} + f(10)\omega^{-12} + f(14)\omega^{-4}, \\
y(14) &= x(12) + x(14)\omega^{-14} \\
&= f(0) + f(4)\omega^{-12} + f(8)\omega^{-8} + f(12)\omega^{-4} \\
&\quad + f(2)\omega^{-14} + f(6)\omega^{-10} + f(10)\omega^{-6} + f(14)\omega^{-2}.
\end{aligned}
$$

Figure 5.4.1: FFT, $N = 16$, $i = 3$, subgroup fill

Problems

5.4.1. For $N = 8$, consider the FFT based on the subgroup chain $C_1 \leq C_2 \leq C_4 \leq C_8$ (so $n = 3$). Step through the entire algorithm as in Example 5.4.4.

5.4.2. For $N = 12$, consider the FFT based on the subgroup chain $C_1 \leq C_3 \leq C_6 \leq C_{12}$ (so $n = 3$). Step through the entire algorithm as in Example 5.4.4.

5.5 A "proof of concept" FFT multiplication algorithm

crude convolution-based multiplication

5.6 The Schönhage-Strassen multiplication algorithm

Schönhage and Strassen.

Bibliography