Math 75 notes

P. Pollack and C. Pomerance

The unifying theme in this course is *finite fields*. We will be taking a concrete approach, as opposed to a purely theoretical approach. And we will be concerned with applications.

So, what is a finite field? Answer: It's a field with a finite number of elements.

OK, what's a field? Let's start with some examples and non-examples, and then try for a definition. Here are some examples of fields:

$$\mathbb{Q},\ \mathbb{R},\ \mathbb{C},\ \mathbb{Q}[i],\ \mathbb{Q}(x),\ \mathbb{Z}/(p) \quad \text{with } p \text{ prime},$$

where $\mathbb{Q}, \mathbb{R}, \mathbb{C}$ are, respectively, the sets of rational numbers, real numbers, complex numbers. Also, $\mathbb{Q}[i]$ is the set of complex numbers $a + bi$ where $a, b \in \mathbb{Q}$, $\mathbb{Q}(x)$ is the set of rational functions (quotients of polynomials) with rational number coefficients, and the notation $\mathbb{Z}/(p)$ means "the set of integers mod p". Note that only the last set is finite, and it is the only finite field in this list. Here are some examples of non-fields:

$$\mathbb{Z},\ \mathbb{Q}[x],\ \mathbb{R}_+,\ \mathbb{Z}/(n) \quad \text{with } n \text{ composite or } 1,$$

where $\mathbb{Q}[x]$ is the set of polynomials with rational number coefficients and $\mathbb{R}_+$ is the set of positive reals.

So, if you don't already know the definition, maybe you can glean that the examples of non-fields are all missing something that the examples of fields have. To understand the definition of a field, one must first know the definition of an abelian group. This is a set G closed under a binary operation (a function that takes pairs of elements of G to elements of G), call it $\circ$, and a distinguished element called the identity element, say it is e, such that

$$e \circ a = a, \quad a \circ b = b \circ a, \quad a \circ (b \circ c) = (a \circ b) \circ c$$

each hold for all $a, b, c \in G$, and for all $a \in G$, there is some $b \in G$ (shown to be unique) with $a \circ b = e$. A field is a set F which is an abelian group with an operation usually denoted as $+$ and with identity usually denoted as 0, such that $F \setminus \{0\}$ is an abelian group with an operation usually denoted by $\cdot$ or concatenation and with identity element usually denoted as 1, and such that times distributes over plus:

$$a(b + c) = ab + ac$$

for all $a, b, c \in F$. (We use the standard priority queue for arithmetic, so when not counter-indicated by parentheses, multiplication precedes addition.) The inverse of a under $+$ is denoted $-a$ and the multiplicative inverse of $a \neq 0$ is denoted a^{-1} or $1/a$.

So, we can indeed see that the examples of non-fields really are non-fields: $\mathbb{Z}$ lacks many multiplicative inverses as does $\mathbb{Q}[x]$, while $\mathbb{R}_+$ does not even have an additive identity. The set $\mathbb{Z}/(n)$ for n composite lacks multiplicative inverses for primes dividing n (and perhaps other elements as well), while $\mathbb{Z}/(1)$ has just 1 element, while a field always has at least two elements, namely 0 and 1.

The examples of fields are mostly seen easily to satisfy the various rules, but maybe the examples $\mathbb{C} = \mathbb{R}[i]$, $\mathbb{Q}[i]$, and $\mathbb{Z}/(p)$ need some further explanation on why they have multiplicative inverses for nonzero elements. Showing this is very instructive, and we will try to find some sort of unified argument for the 3 examples. Or rather, we will find 3 arguments, and try to unify them.

Let's start with $\mathbb{C} = \mathbb{R}[i]$. If an element $\alpha = a + bi$, with $a, b \in \mathbb{R}$ is nonzero, then not both a, b are zero. Thus, $a^2 + b^2 > 0$, so that

$$\frac{a}{a^2+b^2} - \frac{b}{a^2+b^2}i$$

is in $\mathbb{C}$, and it is readily observed that this is the multiplicative inverse of α. Actually, the exact same proof works for $\mathbb{Q}[i]$. Let's try for a *harder* proof that is hopefully more generalizable. Consider the polynomials $bx + a$ and $x^2 + 1$. We see that if "x" is replaced with i, the first is the element α, while the second is 0 (so that $x^2 + 1$ is the *minimum polynomial* for i). Let's use high school algebra and divide $bx + a$ into $x^2 + 1$. The quotient is $\frac{1}{b}x - \frac{a}{b^2}$ and the remainder is $\frac{a^2+b^2}{b^2}$. Thus,

$$x^2 + 1 = \left(\frac{1}{b}x - \frac{a}{b^2}\right)(bx + a) + \frac{a^2+b^2}{b^2}.$$

We recognize the remainder to be nonzero, so we can multiply both sides of this equation by the reciprocal, and reorganizing, we get

$$\frac{b^2}{a^2+b^2}(x^2+1) - \frac{b^2}{a^2+b^2}\left(\frac{1}{b}x - \frac{a}{b^2}\right)(bx + a) = 1.$$

Now replacing x with i, this equation becomes the assertion

$$-\frac{b^2}{a^2+b^2}\left(\frac{1}{b}i - \frac{a}{b^2}\right)\alpha = 1,$$

or

$$\left(\frac{a}{a^2+b^2} - \frac{b}{a^2+b^2}i\right)\alpha = 1.$$

Whew! This seems like a lot more work, and we snuck in the observation anyway that $a^2+b^2 \neq 0$ (do you see where?). But we shall see that dealing with polynomial arithmetic is a great way to generalize.

So, what about $\mathbb{Z}/(p)$ for p a prime? Let's illustrate for $p = 17$ and show that 7 has a multiplicative inverse. Trying to do something like the above longer argument, we divide 7 into 17 getting quotient 2 and remainder 3, so that

$$17 = 2 \cdot 7 + 3.$$

At this stage above, we then multiplied by the inverse of the remainder. Well, we set up this problem so that it is easy to see the inverse of 3, it is 6, since $6 \cdot 3 = 1$ in $\mathbb{Z}/(17)$. Thus, we have

$$6 \cdot 17 - 6 \cdot (2 \cdot 7) = 1.$$

Replacing 17 in the above with 0, we get that the inverse of 7 is -12, namely 5. So, this calculation looks superficially the same, and we shall see that the resemblance is not just superficial.

In fact, we have chosen these examples as illustrating a way to create new fields. For example, suppose F is the finite field $\mathbb{Z}/(3)$. Can we create a new finite field $F[i]$ which would be the set of pairs $a + bi$ with $a, b \in F$, and where $i^2 = -1$? Indeed we can, and this is a finite field with 9 elements. In fact, we'll create *all* finite fields by an elaboration of this idea. So, it is a really promising approach.

Coding theory

One rich area where finite fields play a prominent role is coding theory. When one hears the word "code" it is natural to think of cryptography, where one is trying to hide a message from prying eyes. But that is not what is meant here (though cryptography itself is another area that makes good use of finite fields). Coding theory is the exact opposite, namely the message is supposed to be as readable and transparent as possible. The problem that coding theory tries to solve is the presence of static or "noise", which tends to make transmissions not so intelligible. We're not talking about talking! But rather, electronic transmissions where content is digitized in some way, whether it be for satellite or cable television signals, cell-phone transmissions, cd's and dvd's, etc.

There are very plain ways to digitize content. For example, if we are transmitting a written message, we can let "A" be "01", "B" be "02", etc., continuing on to lower case letters, say "a" is "27", etc., spaces as "00", punctuation, and so on. So a message gets transformed into a number. And the number can be broken into pieces, so we have a sequence of handleable numbers.

Natural language is very redundant; we're sure you've seen puzzles where there is some message with all of the vowels and spaces removed, and you have to guess the meaning. For example:

NWSTHTMFRLLGDPPLTCMTTHDFTHRPRT.

Maybe not so easy, but also not impossible to decipher. That's how the human brain works, but in this digital age, we would like to find transmissions systems that are easily decipherable by computers, even in the presence of a small amount of unavoidable error.

Let's look at three examples. First is the (8,7) parity check code. Here, each code word is a string of 8 symbols, each 0 or 1. There are a total of $2^8 = 256$ such strings, but we put an extra condition on, namely that the final symbol is 1 if there are an odd number of 1's among the first 7 symbols, and it is 0 otherwise. Thus, the first 7 symbols can be arbitrary, but they determine the 8th symbol. There are thus actually $2^7 = 128$ possible code words. Each of these expressions might stand for something that is very plain, and then we can string sequences of them together and get as long a transmission as we would like.

For example, 01100101 is a valid code word, but 01100100 is not.

One should observe that by making the final bit correspond to the first 7, we can detect single errors. That is, if there is exactly one error among the 8 symbols, so that either some 0 is mistaken for a 1 or vice versa and everything else is correct, then the final bit will not check out with the first 7. We can detect errors in this way, and perhaps ask for a retransmission to get the true message. Note that if there are an even number of errors, this code will not detect them.

Now let's consider the triple repetition code. Say we consider code words of length 9, where each symbol is 0 or 1, but the first 3 symbols are all the same, the second three symbols are all the same, and the last three symbols are all the same. So, for example 111000000 is a valid code word, but 101000001 is not. Of the $2^9 = 512$ possible strings of length 9, just 8 of them are valid code words! However, this code not only can detect single errors, it can correct them. For example, the above non-code word corrects most naturally, via majority rule, to the above code word. In fact, this example has corrected two errors, though some other examples with two errors might lead to the incorrect code word.

Now let us consider the triple check code. Now we are considering strings of 0 and 1 of length 6, where the first 3 symbols, say they are abc, are arbitrary, and the last 3, say xyz are all check bits. In particular, x is 1 if the number of 1's in ab is odd and 0 otherwise, while y checks ac and z checks bc. Thus, of the $2^6 = 64$ strings of length 6, there are 8 valid code words. This is thus intermediate between the first two examples in the density of code words among all possible strings. Here's an example of a code word: 101101, and here's an example of a non-code word: 101100. This code, like the triple repetition code, also can detect and correct single errors. Note that the last 3 bits in a code word here will always have an even number of 1's, so the example 101100 is seen not to be a code word. Note that the ab and ac checks are valid, but the bc check fails. If we change one of abc, we'll still have something failing, since the changed letter will be in two checks. So, the abc part is correct, and the error is among the check bits. Here's another example: 100101. Now two of the checks fail, the two that have "c" in common, so that bit must be in error.

Each of these three examples have something in common. They are *linear codes* over the finite field $\mathbb{Z}/(2)$. This finite field has just 2 elements, namely 0 and 1. And note that the symbols used in the code words are just 0 and 1. Further, the code words might be viewed as *vectors*. The first example has vectors with 8 coordinates, the second has 9 coordinates, and the third has 6 coordinates. And in each case, the vectors that comprise the code words are closed under addition. (You should check this.) We'll see that there is a whole family of codes where we copy this procedure, using an arbitrary finite field for the "alphabet" of allowable entries, and using certain convenient linear subspaces of vectors.