PROFESSOR: Good moRning. Again, we're left with a little stub of chapter six to finish. This was our first chapter on binary linear block codes.

Very briefly, last time we developed the family of RM, Reed-Muller codes, parameterized by R and M. These are codes of length two to the M, and distance 2 to the M minus R for all reasonable combinations of M and R, integers. And the reason I do these first is because it gives us a nice big -- in fact, infinite -- family of codes that kind of cover the waterfront. Let us see what we can reasonably get in terms of the parameters n k d in codes.

And, of course, n k t d tell us most of what we know, want to know about the performance of moderate complexity codes in terms of coding gain. And it's easy to see that among these codes, we have sequences in which the coding gain goes -- the nominal coding gain goes to infinity. And we also know that there's at least one sequence, the bi-orthogonal sequence, for which the effective coding gain completely closes the gap to the Shannon limit. And in all likelihood, there are many such sequences, though I'm not aware of the actual result there.

So it seems like we might be almost finished here, except I asked about what possible fly there could be in this ointment last time. And one of you alertly reported that we still really hadn't worried about decoding. And the implicit decoding algorithm here is minimum distance decoding. Exhaustive minimum distance decoding. We basically have to compute the minimum distance to each of the 2 to the k code words. And that's going to get a little tedious as soon as k gets up to 16 or 107 or whatever.

So it was recognized pretty early in the development of coding theory that this was the essential problem. The problem is not to construct codes whose performance under maximum likelihood decoding is near that of the Shannon limit -- you can use random codes for that. But in constructing these more algebraically structured codes, what we're really trying to do is to facilitate the decoding problem, which is the central problem. We need to find a decoding method that is actually feasible.

And, of course, Shannon didn't address that at all. He just addressed the ultimate potential of these codes with the optimal decoding technique and didn't worry about its complexity.

All right. So I'd say the fundamental problem here is decoding complexity. Now, one of the -- an awful lot of the early work on coding was about binary codes, of course. But it also considered them in the context of binary decoding algorithms. In other words, what's assumed is what you send as a series of bits and what you receive as a series of bits.

That's not the picture we have here in this course, where what we receive is a series of real numbers. Because we're going over a continuous additive white Gaussian noise channel. Of course, there are channels which are inherently digital, at least by the time the coding engineer can get his hands on them. What he's presented with is a string of bits.

So it makes sense to consider codes for bit error correction. Classic codes of this type are, first of all, the repetition code. If I send this -- if I use the 7 1 7 code, the repetition code of length 7, I'm going to be able to correct up to three errors. Three bit errors. Because four bits are still going to be correct. In a majority vote, we'll do the decoding.

Most of you have probably seen Hamming codes. These are codes with minimum Hamming distance 3. That means if you make one error, you're still within the Hamming sphere of radius 1 around the original code word. That's disjoined from all the other Hamming spheres, and so in principle you ought to be able to do single error correction with Hamming codes. And there are very simple algorithms for doing that.

OK. But that may be what you have to do. The final point of chapter six is that in the additive white Gaussian noise context, that's not what you want to do at all. And I just want to say a few words about that because this is one of the Achille's heels of a lot of classical coding theory, which was addressed to the binary in, binary out situation.

So let's talk about the penalty for making hard decisions. That is, by hard decision, we mean a binary decision. 0, 1, plus or minus 1. And I'll do this fairly quickly, but in a way that I hope will stick.

Let me draw our block diagram of our coding scheme. We've got a code. We've got an encoder, which basically puts out a code word. An n-tuple in the code. I'm going to consider the individual elements of this n-tuple. They're just elements of the binary field F2 And we go through our standard 2-PAM map, which maps them to s of xk, which is going to be in plus or minus alpha in general. Just plus or minus 1. Doesn't matter up to scaling.

In order to send them, we're going to send this sequence of real numbers over the white Gaussian noise channel. And as a result, we're going to get out s -- let's call it a received symbol -- rk, which is the transmitted symbol plus some Gaussian noise symbol, nk. This is a real number.

And then -- then what we do? Now we're into the decoder. This is what we received, so it's free to us to choose what to do next. I'm going to say observe, first of all, that without loss of generality, we can decompose rk into a sign and magnitude. So this out here is going to be a sign of rk. And this is going to be the magnitude of rk. And let me just give this a name. Let me call this yk, and let me call this beta k.

OK. So what is this? This is in plus or minus 1, and this is in the positive reals, so the non-negative reals. OK. And clearly, if I keep both of these, I haven't lost any information. This is just a way of representing rk in sign and magnitude form.

Now, this is what's called the hard decision. Actually, let me do this differently. Let me pass this through the inverse to a 2-PAM map. So out of this, I get a yk, which is in F2

OK. Either of these -- the real sign or the binary hard decision -- could be considered to be the hard decision. Now, what am I doing here? I'm saying, well, if you force me to make a guess on whether this individual transmission was sending a 0 or 1, this is the most obvious guess. If I send -- here's minus 1, here's our minus alpha, here's plus 1 -- if I sent a plus 1, then my maximum likelihood per bit decision would be to decide the plus 1 was sent. If I get a positive sign in -- what I'm really mapping here is rk, and minus 1 if I get a negative side. So that's what's called a hard decision. I'm going to decide right here what I think it was, what it most likely was. And this is the best way to do it. Just take the sign.

This here is a real valued weight, often called the reliability, which is helpful to retain. That's the main point of what I'm going to talk to you about. If this is 0 -- that means what we received was right on this boundary between the positive and negative side -- then the reliability is 0. In likelihood terms, that means it's equally likely that what was sent was a plus 1 or a minus 1. So in some sense, we get no information when the reliability is 0. Fails to discriminate between plus 1 and minus 1.

The larger this is, the further we get out here. Or symmetrically, the further we get out here, the more certain we are that in that particular symbol, neither a plus or a minus was sent. OK And it actually tuRns out, if you work out the Gaussian numbers, that beta k is the log likelihood ratio up to scale of the more likely versus the log of the likelihood, or the more likely versus the less likely symbol.

So it has a minimum of 0. The bigger it is, the more reliable. So it's natural to call it the reliability.

OK. An awful lot of traditional algebraic decoding neglects this channel down here. It just says, OK, let's take these hard decisions and try to decode them. If you've previously had a coding class, then that's probably what they did. Or they assumed a context in which this was never available in the first place.

And my only point here, my main point, is that that's a very bad thing to do. How can we evaluate that? Suppose we just take this top channel and don't look at the reliability. Then basically we have the channel model becomes a binary symmetric channel. We have bits in, bits out, and we have probably one minus p that if we send a 0 we get a 0. And since it's symmetric, the same probability -- if we send a 1, we get a 1 -- and a probability p of making an error. So we get the traditional memoryless binary symmetric channel model.

We can compute the capacity of this model and compare it to the capacity of the additive white Gaussian channel -- this model. And depending on the signal to noise ratio, from a capacity calculation, we find that there is of the order of 2 or 3 dB loss.

Well, as we go on in this course, we're going to find that 1 dB is a very worthwhile coding game. So to throw away 2 or 3 dB just by throwing away this information, is a bad thing to do.

I can make this point in a different way by simply looking at what's the optimum decision rule. For say, let me take very simple code, just the 2-1-2 repetition code. That's just 0,0,1,1. This is the code where you either send plus, plus, plus alpha, plus alpha, or minus alpha, minus alpha. And what's the decision region? It's obviously this 45 degree line. And what is this squared distance to any decision region? It's 2 alpha squared, right?

So basically, the probability of making an error is the probability that the noise variable has a magnitude in this dimension, and in this direction, greater than the square root of 2 alpha squared. You've been through all those calculation several times now.

All right. So this is -- if I keep reliability info -- in other words, if I keep the full received signal -- if I discard reliability, then basically what can I get out? Then my y, as I say, is in 0, 0, 0, 1, 1, 0, 1, 1. There are only four possible things I can see in two transmissions through this channel. They're all binary two-tuples.

And what does that mean? That means if my actual r is in this quadrant, then I'm going to make a hard decision of 0 and 0 both times. So I'm going to say I'm in this quadrant. And likewise, I'm basically going to decide which of these four quadrants I'm in, and that's all the information the decoder is going to have.

So here I am at this point here. Now the decoder simply knows, via these two bits, which quadrant you're in. Now, it has to decide which of these two code words were sent. What's its maximum likelihood decision rule given just this information?

AUDIENCE: 0, 0, [INAUDIBLE] 0, 0, [INAUDIBLE] 1, 1, 1, 1, 1. And in the other case, any one.

PROFESSOR: OK. That's correct. So clearly, if you land in this quadrant, you decide 0, 0. Now, if you land down here, you decide 1, 1 in this quadrant.

And what am I going to do here or here? Actually, here the evidence is totally balanced. I have nothing that tells me whether to go one way or another. I could make an arbitrary decision in the hope of minimizing my error probability. Flip a coin. But whichever decision I make, what I'm going to find is that there's a probability of error that -- there's a certain noise, namely the noise that takes me from here to this decision boundary that is going to cause an error. Or if I decide the other way, it would be the one that goes from here to here. So I'm going to be stuck. There are going to be certain noise variables of length, now, merely alpha, or square distance alpha squared, that are going to cause me to make a decision error, ultimately. Regardless of how I set up this final block here.

Decoder. Whatever rule I give to this decoder, it's only going to take, worst case, a noise of squared magnitude alpha squared to cause an error.

AUDIENCE: [UNINTELLIGIBLE PHRASE]

PROFESSOR: That's right. I shouldn't have thrown this away. That's the elementary point I'm trying to make here. All right.

AUDIENCE: [UNINTELLIGIBLE PHRASE] the distance, d_min is different, essentially. I mean, we can put it that way.

PROFESSOR: That's where I'm going, is that the effective minimum squared distance here is 2 alpha squared. And this is what shows up in our union-bound estimate in all of our -- we do this kind of minimum distance decoding. But if I throw away this important information, then my effective minimum squared distance is only alpha squared. That is the bottom line here.

In dB terms, how much of a cost is that? 3 dB, right? So I've cost myself a factor of

2 in noise margin. So 3 dB loss.

Because of lack of time, I won't go through the argument in the notes, which shows that, in fact, exactly the same thing occurs for any -- whenever the minimum Hamming distance is even here in this code that I started from, you lose precisely 3 dB. It's not quite as clean an argument when the minimum Hamming distance is odd. Then you lose up to 3 dB, and it goes to 3 dB as the distance increases. But there's a very, pretty elementary geometric argument that hard decisions, again, cost you 3 dB loss, which is consistent with what the capacity calculation gives you.

All right. Well, obviously the reason we did this is we were trying to simplify things. What would be the first step to unsimplify things and get some of this loss back?

Let me suggest that what this amounts to is a two-level quantization of the received real number rk. What's the next number higher than 2? Integer.

AUDIENCE: 3.

PROFESSOR: 3. All right. How about a 3 level quantization here? OK.

So what we're talking about is a highly quantized magnitude where instead of just making a decision boundary here, which is effectively what we did for 2 level quantization, let's make some null zone here between some threshold plus t and some threshold minus t. And we'll say, in this region, the received symbol is unreliable. OK. That's called an erasure. So we make a quantized magnitude where rk -- doesn't fit so neatly here. Let me just say this quantized magnitude is going to give me out something which is either going to be minus 1, a 0, or plus 1. It's going to be three levels, this is minus 1, 0, plus 1.

Or I don't want the minus 1 there. The reliability is either a fixed reliability, or it's 0. OK.

So now I've got a channel model that looks like this. If you work it out, it's called the binary erasure channel with errors. All transitions are possible. I can send a 0,1. I can receive, let's call it, a 0, a 1, or a question mark for -- this is called an erasure.

And this is 1 minus p minus q, and this is p and this is q symmetrically, for some p and q, which you can evaluate.

And let's choose this threshold t to optimize things and get the maximum capacity. Now if you do the capacity calculation, you'll find there's only 1 to 1.5 dB loss. Again, depending on the signal to noise ratio. So in effect, you've already bought back, just with this very simple method, half of the loss that you inflicted on yourself by making hard decisions.

So making erasures is a good first step towards correcting this problem. And let's think about it again for this simple code.

Suppose I establish these thresholds at plus t and minus t. Plus t and minus t. So now when I consider the problem that the decoder has, I have nine possible regions, all right, each with three possible decisions in two dimensions.

And so what's my decision rule going to be now? For this 0, 0 decision, I'm going to include, of course, this region. But now if I land in this region, that means one of the received symbols was erased, but the other one gave me a definite indication. So the weight of evidence still goes up here. And similarly, like so. So my decision region for 0 -- definitely for 0, 0 is this region. And my decision region definitely for 1, 1 is the symmetric region.

And, of course, I still have three regions where I really can't say anything. This is two erasures. This is 0, 1 -- two conflicting pieces of evidence. This is 1, 0 -- two conflicting pieces of evidence. So I still have to flip a coin out in here.

But how I improve things, that's measured by what's the minimum distance to -- what's the minimum size of error it takes to make a decision error. And that's going to be either this length or this length. You see that? Just pretty clear, intuitively?

So the game here is we first of all choose t so that these two lengths are the same. That's done by squeezing t. You see these go in opposite directions as t is raised or lowered. So we find the t such that these two are the same. Having equalized t, we find some value for the effective minimum squared distance, which is between alpha squared and 2 alpha squared. As I remember, this somehow gains about 1.5 dB. The moral is about the same. You can get about half of your loss back by using erasures.

And again, this holds for any code which has an even minimum Hamming distance, if you can accept this method of analysis.

All right. So a first step, even if you want to stay in the binary world, is to allow yourself to use erasures as well. That will claw back half of the loss that you might have incurred by using hard decisions.

And, of course, even better would be to use a more highly quantized reliability and somehow have a decoding algorithm that can use soft decisions. Soft decisions are decisions that have reliability metrics attached to them. And in the early days of coding, people evaluated capacity. They evaluated this sort of thing. And it was pretty generally agreed that eight level quantization was going to be practically good enough. 16 levels, certainly good enough. Nowadays you typically go up to 64 levels. You might go up to much higher numbers if you have little other things to worry about like timing recovery and so forth.

But if you're purely in a synchronized, symbol by symbol transmission, then three or four bits of reliability information are going to be enough. So from an engineering point of view, that's a good way to go. Now, of course, you're going to need a decoding algorithm that can use soft decisions.

So as we go along, I'm going to talk about error correcting decoding algorithms that are not much, because correcting errors only is a very bad thing to do here. Errors and erasure. Correcting decoding algorithms, which are fairly easy in algebraic code context. And then finally soft decision decoding algorithms, which is the kind we really want on this channel. We really want to use the soft decisions. We can't afford these huge losses. We can't even afford 1 to 1 and a 1/2 dB loss.

OK, so there's more set on this in the last part of chapter six, but that's all the time I want to spend on it in class. Does anyone have any questions? yes?

AUDIENCE: [INAUDIBLE] transformation before we do the quantization, can we change the shape of the [UNINTELLIGIBLE] regions such that we can further improve the [INAUDIBLE PHRASE] improve the decoding?

PROFESSOR: Well, that's an interesting idea. But think about it from an information theoretic point of view. Can you actually get more information by introducing a one-to-one transformation, whether it's linear or nonlinear, whatever?

AUDIENCE: The point is, the original quantization is based on a straight line, [INAUDIBLE PHRASE]. [UNINTELLIGIBLE PHRASE].

PROFESSOR: You want to put a curved line on this plane?

AUDIENCE: [INAUDIBLE].

PROFESSOR: OK. That's going to imply decisions that are not symbol by symbol decisions. That's going to imply that you need to save r1 and r 2 in order even to just decide where you're in this and then go through some kind of quantization on the plane. And once you've got r1 and r2 and you're trying to draw crazy boundaries, I suggest it's going to be simpler to just compute the Euclidean distance to each of these points, which results in this decision region.

AUDIENCE: [INAUDIBLE PHRASE].

PROFESSOR: OK. I don't see the rationale for it yet, but there's been a million innovations in this business, and you might have a good idea.

Any other comments? I really appreciate comments that go anywhere. I like to address them now when they're ripe. OK.

So let's go on to chapter seven and eight. [UNINTELLIGIBLE] here.

Chapters seven and eight are closely related. This is really my bow to algebraic coding theory. I expect that if any of you have had a close course in coding before, that it was primarily on algebraic coding theory. How many of you have had a course in coding theory before? One, two, three. Not so many. Was it on algebraic coding theory? Yes?

AUDIENCE: I [UNINTELLIGIBLE] two, one engineering and one algebraic.

PROFESSOR: Excuse me?

AUDIENCE: Two -- one for -- actually, two algebraic and one out of kind of this stuff.

PROFESSOR: OK. Well, you're a ringer then. What was the one on this stuff?

AUDIENCE: It was out of Wicker's book.

PROFESSOR: Out of?

AUDIENCE: Wicker's book.

PROFESSOR: Wicker's book on turbo codes.

AUDIENCE: No, the data storage one.

PROFESSOR: Data storage. OK. I don't know that book. But it talks about sophisticated soft decision type coding, and so forth.

What was the course you took?

AUDIENCE: Algebra.

PROFESSOR: Algebraic. OK.

All right. Well, let me just make a broad brush comment, which may or may not be supported by your previous exposure to coding theory. Which is that for many, many years, when people said coding theory, they meant algebraic coding theory, coding theory of block codes that are constructed by algebraic techniques like Reed-Muller codes, like Reed-Solomon codes, BCH codes, cyclic codes. And if you said, I want to leaRn some coding theory, asked your graduate student to go buy a textbook for you, it's probably going to be a textbook on algebraic coding theory.

Meanwhile, there was a bunch of us who were off actually trying to construct codes for real channels. And conceRning with really how complex this is to implement, and what kind of performance can we really get. And we hardly ever used algebraic codes. Initially, we used convolutional codes with various kinds of decoding algorithms, threshold decoding, sequential decoding, the Viterbi algorithm. And these get more and more elaborate, and then finally the big step, well, the big step was to capacity approaching codes. Which now people are starting to call modeRn coding theory. Long, random-like codes that have an iterative decoding algorithm. And that's what we'll get to towards the end of this course.

So from an engineers point of view, to some extent, of all this work on n k d and algebraic decoding algorithms and so forth was a massive distraction. It really missed the point. Sometimes because it assumed hard decisions. Or after assuming hard decisions, it assumed bounded distance decoding algorithms. But it was all -- from Shannon's perspective, it was too deterministic, too constructed, too structured. You want more random elements in your coding scheme.

However, two things: one, this theory is a beautiful theory, both the mathematical theory of finite fields and the coding theory, particularly of Reed-Solomon codes, which are uniquely the greatest accomplishment of algebraic coding theory, and which have proved to be very useful. And B, Reed-Solomon codes are something that as engineers, ought to be part of your tool kit, and you will find very useful a variety of situations.

So the objective of this part of the course is, within some proportion within the overall scheme of the course, to give you exposure to this lovely theory. You can leaRn all about it just by reading a book or two. Probably less than one book. And to give you some exposure to Reed-Solomon codes, which have been used in practice. For more than 20 years, the deep space standard was a concatenated code. That means a sequence of two codes, of which the inner code was a convolutional code decoded by the Viterbi algorithm, which we'll talk about after this.

And then the outer code, the code that cleans up errors made by the inner code, was a Reed-Solomon code of length 255 over the finite field with 256 elements. And that's the dynamite combination that dominated the power-limited coding world for at least 20 years. So you ought to know, as a minimum, about Reed-Solomon codes.

So that's my objective in going through these next two chapters. However, because these have not basically been on the winning path to get to the Shannon limit, I'm trying to limit it to three weeks max. And therefore, that means that the presentation is going to be a lot faster than what you're used to. I'll probably do as much in these three weeks as most people do in a term in an entire algebraic coding theory course. That's an exaggeration -- people do a lot more -- but I try to at least hit all the main points that you need to know.

So I apologize in advance if it seems that now we're on a very fast moving train. But you ought to know this stuff. You ought not to spend too much time on it. I'm trying to reconcile those two points of view.

Any questions or comments? All right. Let's go.

Chapter seven. We go through a series of algebraic objects. First, the integers, then groups. You certainly ought to know -- you do know about integers. You ought to know about groups, at least a little bit. Let's see. What's next? Then fields, particularly finite. Then polynomials over fields, particularly finite. And then finally, from this we actually get to construct finite fields.

And these are all things that you'll encounter again and again, have encountered. I think you've probably encountered everything except possibly for groups and finite fields. You certainly encountered polynomials over real and complex fields. And so in some sense, this is a broadening of what you already know.

So I assume we start with the integers for two reasons. One is you already know about the factorization properties of the integers. We're going to -- that leads to a little bit of number theory, which we're going to use when we talk about groups, finite groups, cyclic groups. So this is just to remind you, but it's also to put up a template. And we're going to find the algebraic properties of polynomials are very,

very similar to those of the integers. This is because they're both rings. They're both in fact principal ideal domains. They're Euclidean domains. They both have a Euclidean division algorithm as their key mathematical property.

And so pretty much any time you want to develop a result about the polynomials, you can start from the comparable results that you know about the integers and just change the notation. And you'll have a sketch of the proof for polynomials. So it's important to remind ourselves about the integers first.

Now, when I say the factorization properties of the integers, I'm not talking about anything more complicated than what you leaRned in grade school. I just want to introduce a few terms. We have the notion of divisors. a divides b means that a times some quotient equals b. I could be talking about plus or minus, positive or negative integers here. Everything divides 0.

OK. We have the idea of units, which are the invertible integers under multiplication. And what are those? Which integers have a multiplicative inverse? [UNINTELLIGIBLE] any of them?

AUDIENCE: [INAUDIBLE PHRASE].

PROFESSOR: [INAUDIBLE]. All right.

That's a little exercise that shows you that, in general, the invertible elements of any set that has an operation like multiplication form a group. [INAUDIBLE] group thing, these clearly form a group under multiplication that's isomorphic to the binary group. Well, in fact, we've already been using it -- F2.

OK. So we have units, and whenever we talk about the divisors, factorization, primes, so forth, the units are a nuisance. Because we can always multiply things by units and it doesn't affect factorization properties. This is [UNINTELLIGIBLE] a little bit more out there.

What I eventually want to get to is -- skipping a few things that are said in the notes -- is unique factorization. Maybe before that I should define primes: are positive integers. The reason we stay positive is, again -- of course, we could have a negative prime integer, but let's pick a unique representative of two units. Positive integers -- I should've said in the notes greater than 1, we don't consider 1 to be a prime -- that have no non-trivial divisors.

And, of course, every integer has trivial divisors. Namely, plus or minus 1 and plus or minus itself. Let me give this a name. i. Not a very good name. How about n?

So if we have any integer n, it's divisible by plus or minus one, plus or minus n -- that doesn't count. So a prime is an integer that doesn't have any other divisors than these trivial ones.

And then I want to get to unique factorization, which I'll just state, not prove. Which is that every integer is equal to a unique product of primes. And we always have to add up to units. We can add as many plus or minus 1's to the product as we want, as long as we have an even number of them. And we'll have another factorization, so here, we want to exclude the units. And then we have unique factorization.

OK. This is all grade school stuff, high school stuff maybe. Again, when I go through polynomials, you'll see there is a set of concept exactly like this. Now, where we want to get to is mod-n arithmetic, which again, you all know about, I assume.

This is based on the fact -- we can think of it as being based on Euclidean division algorithm, which basically proves that given any two integers m and n, we can write m as some qn plus r where 0 is less than or equal to r, less than or equal to n minus 1, and is called the remainder of m, modulo n.

And how is this proved? Let's say these are both positive integers. Say m is a big integer, n is a small integer. We just keep subtracting integer multiples of n from m until we -- you know, the remainder will start decreasing until we get the remainder in this range, and it's obvious that we can always get a remainder that's in that range. And further, that this decomposition is unique under this restriction on r.

All right. So we write the m is congruent or equivalent to r mod n. That means it differs from r by a multiple of n where r is unique and r is in the set 0, 1, up through n minus 1, which we say are the residue classes mod n. It has to be one of those n things. OK?

No one is having the slightest trouble with this, right? Boring. You've seen it all before. I hope.

OK. So now you ask, suppose m equals r mod-n, and some other w equals s mod-n. What is m plus w mod-n, and you quickly prove that it's equal to r plus s mod-n. By transferring, by just checking the properties of ordinary arithmetic, you can use this kind of expression: m must equal something times n plus r. w must equal something times n plus s. So if we add m plus w, we're going to get something times n plus r plus s. And that is, by definition, going to be r plus s mod-n.

So we get a well-defined addition rule for these residues, for the elements of this set of n residues Rn, or remainders. OK, so addition is well-defined.

And similarly, multiplication is well-defined. mw mod-n, again, you can prove -- and this is done in the notes -- that if mw mod-n can be found by taking rs and reducing it, mod-n. Maybe I should have congruent signs here, but I won't worry about it. So this is mod-n, addition and multiplication.

So what this says is that Rn, combined with this mod-n addition operation, which in the notes I write in that way, and this mod-n multiplication operation -- this is a well-defined set of n elements with a well-defined addition operation and a well-defined multiplication operation. And I don't prove any properties about it here, but I come back to prove that this has almost all of the properties that we want of a field. In fact, if n is a prime, it does have the properties that we want of a field.

So we'll later prove that the finite field with p elements is simply rp with mod-p addition and multiplication. And, of course, for the particular case p equals 2, we already have a lot of experience with this. That's how we get the binary field. We just take the 0 and 1, considered as residues mod-2. And then the field addition and multiplication operations give us something that satisfies the axioms of the field. We'll find that works for every prime, and it doesn't work for non-primes.

Can anyone quickly see why this isn't going to work for non-primes? Suppose n equals 6. Anyone quickly see why this set under these two operations isn't going to be a field? This, of course, assumes that you know what a field is, which we haven't actually said yet.

AUDIENCE: [INAUDIBLE].

PROFESSOR: Excuse me?

AUDIENCE: [INAUDIBLE].

PROFESSOR: There's no inverse. In a field, every non-zero element has to have an inverse, just like in the real and complex field. But in the integers mod-6, 3 times 2 is equal to 6, which is equal to 0. So there are two non-zero elements that multiplied together give 0. And that means that neither 3 nor 2 can possibly have a multiplicative inverse.

So that's why you have to have a prime. This doesn't happen, of course, if n were a prime.

OK. So that's just a little warm up to remind you what mod-n arithmetic is like. And in particular, this is going to be the key element of a lot of proofs -- the Euclidean division algorithm -- both for integers and for polynomials.

OK. So that, I hope, was review. Now we're going to talk about groups. So let's do it.

How many of you feel you know what a group is? Less than half the class. OK. Well, we've been talking about groups so far here, but let's take a little bit more of an axiomatic approach. In the notes, I actually give two possible sets of axioms for a group.

Let me start with a standard one. You've heard of something called the group property in this course. Well, first of all, what am I talking about when I'm talking about a group? I'm talking about a set, g, and I'm also talking about some operation on that set, which I'll use the mod addition circle-plus as the group property. Even if it's multiplication or matrix inversion or -- no, it couldn't be matrix inversion.

What this means is that if we take any two elements of the group, then a plus b is well-defined. So you can think there's a table. Here are the elements of the group. a, b, c, d, e. a, b, c, d, e. And at this point, all we know is that we can fill out the table. We get something that goes in all here. All right? So that defines the group.

AUDIENCE: [INAUDIBLE PHRASE].

PROFESSOR: That means if I give a and b, then the quantity a plus b is -- I can tell you what c is. There's a unique element in each of the cells of this addition table, or the group table.

OK. Group property, also called closure, is the obvious one. What's an elementary condition of what all these sums have to be for this to be a group? Have to be all in the group.

If I say I have a group that consists of 1,2,3,4,5, 1,2,3,4,5 and my group operation is ordinary addition, then I can certainly fill out the table. And I get 2,3,4,5,6 -- whoops. Yeah, that's the element that's not in the group. So in order to have a group, I want a -- I mean, it's a plus b is in the group, all a and b. So the group is closed under addition. This was the property we used to define linearity for binary linear block codes. The sum of any two n-tuples had to be another n-tuple in the code.

We have the associativity operation. This is a requirement on the binary operation, really. a plus b plus c has got to be equal to a plus b plus c. In other words, you can get rid of the parentheses. And this, an expression like that, is meaningful. It doesn't matter how you group, whether you add these two things first and then that, or add these two things first and then that.

Maybe now is a good time to talk about the special property, which is not an axiom. So let me call it x abelian, or the commutative property. This is true if a plus b is equal to b plus a, all a, b group. Now this is, of course, true for ordinary addition and ordinary multiplication. It's not true for matrix multiplication. Say if I have a group of matrices, then in general, ab is not equal to ba. So for some groups this holds, for some groups it doesn't hold.

In this course, almost without exception, every group we're going to talk about is an abelian group. That means that in adding things up, the order doesn't matter. So if that were true, then it in a plus b plus c, that means that's the same as c plus b plus a, or c plus a plus b, or whatever. All the properties that you're accustomed to from ordinary addition.

But this is not one of the group axioms, and, of course, there are non-abelian groups. The smallest non-abelian group has size 6, and is the permutation group on three elements. So small groups are all -- size 5 or less are all abelian. Because that's the only way we can do the group table.

All right. So that's terminology at this point. It's not part of the axiom. But that's the reason that I use a symbol that looks like a plus, is I'm pretty much always thinking an abelian group, and I don't want to trouble you with thinking about non-abelian groups.

OK. Then we have the identity property. There is an identity, which, in an abelian group, is always called -- ordinarily called 0. In an additive group it's always called 0. So I'll call it 0. In more abstract treatments, it's called 1 or E for -- what does E stand for? Eigen something. There's a 0 in g such that 0 plus a equals -- or a plus 0 -- is always equal to a. The 0 is the null operator on any element of the group. If you add 0 to anything, you get what you started with.

Clearly, we have such a thing in ordinary addition. Suppose I was talking in the real numbers under multiplication, is that a group, you guess? It's actually not. Suppose I talk about the non-zero real numbers under multiplication. Is that a group? Yes, and its identity is 1. 1 as the thing that if you multiply it by anything, it leaves what you started with.

There's the inverse, which we don't have in the reals under multiplication. For all a and g, there exists something called minus a -- or a to the minus 1 in multiplicative notation -- in g such that a plus minus a equals 0. That's an additive inverse. So in the real numbers, the inverse of a is minus a. In the integers, the inverse of a is minus a under addition.

In the non-zero real numbers under multiplication, every non-zero real number does have a multiplicative inverse, which is called 1 over a, or a minus 1. That's why we have to exclude 0. 0 does not an inverse. So a good example to think about is r-star, which is non-zero reals under ordinary multiplication, is a group. Is it an abelian group? Is ab equal to ba? Yeah. So this is an infinite abelian group.

All right. OK. I want to show that the axioms lead to a permutation property. Because this is something that's going to come up again and again. One way of saying this is if I have, if a plus b equals c -- suppose I write down that symbolic equation -- then I can solve for any one of the three given the other two.

How would I do that? If I'm given a and b, obviously, that tells me c from the table. If I'm given a and c, then a is equal to c minus b -- I'm sorry -- b is equal to c minus a.

I should say here, from this inverse, we're going to write b plus minus a in a simpler form. It's just b minus a.

So the fact that we have an inverse sort of defines another operation, which we call subtraction, which is not abelian. Not commutative. a minus b is not equal to b minus a. But it's basically implicit in this that we subtract group elements. We can cancel group elements. If you have a plus c equal to b plus c, then a must be equal to b. Because you can subtract c from both sides.

All right. OK. What does this mean that this gives us a constraint on the addition table of a group? Suppose I take a four element group, g, and one of the elements is going to be the zero element, and the other let's just call a, b, c. 0, a, b, c. And I have a well-defined addition table, which includes, in here, only a 0, a, b, c.

From the property of the identity, I can fill in some of these things. And now what freedom do I have to fill in the rest? How many group tables are there of size four?

Well, the main point that I want to make with permutation property is that every row and every column has to be a permutation of the four elements 0, a, b, c. Why is that? Suppose I had another a in this column here. Then I would have a plus 0 is equal to a, and also a plus c is equal to a. But that would imply that c equals 0, which it isn't.

All right. So in order to be able to solve this equation, this has to be something different from a, and likewise. Since I have, in this case, a finite group, I basically just have to write the group elements down here again in some order.

It tuRns out that for four elements, there are exactly two ways of doing that, generically. One is sort of the cyclic group. I'm sorry -- a, b, c, 0, a, c, 0, a, b. That's one way of filling out the table such that each row is a permutation of 0, a, b, c. Each column is a permutation of 0, a, b, c. So that's a legitimate group table.

Let me just, for your interest, show you that that's not the only one. We could also do 0, a, b, c, 0, a, b, c. If I write 0 here, then I've got to write c there and b there. Got to write c here and b there. And then down here, the only choice I have is this. This is called the cyclic group of four elements. This is called the Klein four group. And this is all there is. Both of them tuRn out to be abelian if you check it out.

And basically, this permutation property restricts these to be the possible groups of size four, abelian or non-abelian, just to satisfy the axiom.

I actually prove in the notes that if you replace axioms a and b with the permutation property, you get another equivalent set of axioms for a group. So the permutation of properties is very basic. All you need is -- for a set and a binary operation to form a group, all you need is identity, associativity, and the permutation property for the group table, the addition table. Simple proof of that in the notes.

But this observation is going to be important as we go forward. It's a fundamental thing we're going to use. So you see a group can't just be anything. It already has very definite restrictions on it.

All right. Anything else I want to say right here? OK, subtraction, cancellation.

OK, I guess we can go into the next property, which is cyclic groups. And I'm only going to talk about finite cyclic groups, or finite groups in general. Now, a cyclic group is probably the simplest kind of group.

Let me first put up the canonical cyclic group, which is the integers mod-n, which means the set of residues mod-n and the mod-n addition operation. Which is written just as Zn -- there are various notations for it. But this, of course, we'll just write it as Zn. The integers mod-n.

Does this form a group? Let's check the axioms. Certainly, let's take the original axioms. So the Rn is closed under addition mod-n. If we add two elements, we're going to get another element. The mod-n addition operation is associative. In fact, it's abelian. a plus b plus c is b plus c plus a -- it doesn't matter what order, from the same property for ordinary addition.

Is there an identity in here? Yes. The 0 is one of the elements here and acts as the identity. 0 plus anything mod-n is equal to the same thing mod-n.

And is there an inverse? OK. Let's explicitly write out Rn as 0, 1 up to n minus 1. And then I claim that if I take any of these i, then n minus i is its inverse. n minus i is in the right range if not zero. The inverse of 0 is 0. The inverse of i between 1 and n minus 1 is going to be n minus i because if I add i to n minus i -- is equal to n, take mod-n, that's equal to 0. So every element has the additive inverse.

And so the group table has a very boring form, which does satisfy the permutation property. If I take Z5, for instance. 0,1,2,3,4. 0 plus anything looks like that. 1 times anything is just 1,2,3,4 and 1 plus 4 is 0 mod 5. 2,3,4,0,1, and so forth. 3, 4, 0, 1, 2, 4, 0, 1, 2, 3. Each row or column is just a cyclic shift to the previous row or column. Just fill it out. And does it satisfy the permutation property? Yes, it does.

This one here, for instance, you can recognize as a cyclic group table. And so, apart from relabeling, this is the group table of Z4. Or we take a, b, c equal to 1, 2, 3.

If two groups have the same group table up to relabelling, they're said to be isomorphic. OK? That's an important concept in math. And all it means is the two groups act the same, except that we've changed the names of the group elements.

OK. So what I'm going to show is that all cyclic -- this group is sort of cyclic because, as you see in the notes, we can think of adding 1 as just being cycling around a circle. In this case, with 5 points on it. And adding 2 is going two steps around the circle. And the elements on the circle are the group elements 0, 1, 2, 3, 4. And then we get to 5. We identify that with 0. So we close the loop. One we identify with 6. This with 7. This with 8. Up to as many as you like.

So that's the intuition why these are called cyclic groups. The addition operation cycles rather than being on a line, as it would be with the ordinary integers, where we go infinitely in either direction.

So the formal definition of a cyclic group is that g has a single generator. Call it a little g. And -- am I running over? Close to. Is that really right? I only have five minutes left. We're not going to go at the speed I had hoped. g is -- what does that mean? If we know that g contains g, and this is never going to be equal to 0, then we know that g contains two elements at least: 0 and g. And by the group axioms, g contains g, g plus g, g plus g plus g -- we can keep adding them.

I'm going to call these the integer multiples of g. So this will be 1g, 2g, 3g. That's what I mean when I write an integer in front of g -- I mean g plus itself, the integer number of times. And all those things, by the group properties, have to be in the group. All right?

Again, restrict this to finite. It's a finite cyclic group. Then at some point, I have to -- this is all the elements of the group. So at some point, I have to come along and say that ng is 0. If just by -- when I say generator, the set generated by g is just 1g, 2g, 3g. And that's got to be all the elements of the group, so n times g has got to be 0.

And that means, without putting too fine a point on it, that if n were 5, for instance, the group has to consist of 0g, 1g, 2g, 3g, 4g, and then 5g is equal to 0. So we call this 0g or 5g. And furthermore, the addition table of the group, we can, you know -- however, if we add 2 g's to 3 g's, we're going to 5 g's, which is 0. All right? So 2g plus 3g has got to be 5 g's, which is 0g.

So this is what the addition table of the group has to look like. It's exactly the same, except we just make a one-to-one correspondence like that.

So the conclusion is a finite cyclic group with n elements -- I'll write the size of g by this absolute value thing -- is isomorphic to the integers mod-n -- this canonical cyclic group that we started with -- must be isomorphic to Zn with the isomorphism being given by the one-to-one correspondence. i_g in the group corresponds to simply i in Zn, or 0 is less than or equal to i is less than or equal to n minus 1. The addition rule is exactly the same.

OK, so this is a pretty sweeping conclusion. It says that really the only finite cyclic group up to relabeling is Zn. OK, so if you understand Zn, you understand all finite cyclic groups. Just in a general group, we're going to have some element which basically represents 1, called g, and its integer multiples, which represent 2, 3, 4, and so forth. But we can operate with a group just as though it was Zn. So it's easy to understand finite cyclic groups. They all just look like this.

Question? Yeah.

AUDIENCE: Is the generator unique?

PROFESSOR: No, it's not. No, it's not. Next time we'll start to talk about subgroups of Zn, for instance. In the set of Z5 -- well, let me take this. All right. In Z5 -- let me not even do that. In Z5, 1 is the generator, but because 5 is a prime, 2 is a generator, too. All right?

Another way of writing. Let's take a generator equals 2 in Z5. Then g equals 2, 2g equals 4. 3g equals 6, equals 1 mod-5. 4g equals 3. And 5g, of course, equals 5, which equals zero. So 2 is also a generator. And, in fact, 3 and 4 are also generators. There are four generators. This is getting us right into the number theory.

OK. So no, g does not have to be 1. g is just something that if you take all its integer multiples, you generate the group.

OK, I guess we stop there, and we'll resume next time.