PROFESSOR: Today we finish up chapter 13. I think we are finished, but I'll take questions. We get into chapter 14, which starts to get into coding for bandwidth-limited channels. Here we're coding directly in terms of Euclidean space rather than in terms of Hamming space. And there's this wonderful quote from Neil Sloane. "Euclidean space coding is to Hamming space coding as classical music is to rock'n'roll." Meaning that, in the Euclidean space, things are continuous, whereas we were discrete back in Hamming space.

Nonetheless, there are strong connections between the two, which in two lectures I'll basically only have a chance to hint at. You were supposed to hand in problem set nine last Friday, or today if you haven't already. Today, we'll hand out the solutions for nine, and then chapter 14 and its problems, which are problem set 10. These are not due. As always, I recommend you do them. The solutions will be handed out Wednesday.

However, on the exam, the exam will not cover chapter 14. So everything we do this week is gravy. The exam is a week from tomorrow in the morning. Ashish will administer it because I won't be here. For this exam, it's closed book, except you're allowed to make five pages of notes, as you did three pages on the midterm. And I do recommend you bring a calculator, because for one of the problems the calculator will be helpful. And I was told that erasable memory should be cleared on the calculator, so point of honor if you do that.

Any questions about the exam or chapter 13? Yes?

AUDIENCE: Are you including the midterm [UNINTELLIGIBLE]?

PROFESSOR: Yeah, oh, everything in the course up to chapter 13. The rule is everything that we covered in class. If we didn't cover it in class, then you're not responsible for it. Other questions? Any last questions on chapter 13? This is kind of the finale for binary codes. We found that, at least on the binary erasure channel, we had techniques that would allow us to design LDPC codes that would get us as close to

capacity as we liked, or at least I made that assertion, directed you to where you could find that done in the literature.

And I further asserted that, at least for binary symmetric input channels, like the binary additive white Gaussian noise channels, you could do much the same thing using techniques like density evolution or EXIT charts. Again, the issue is to design code, such as this iterative decoding, after many, many, many iterations, will, with probability one, converge to the correct solution (or with probably one minus epsilon).

So that's the great conclusion to the story for binary codes, channels in which we want to have binary inputs. And way back in the beginning, we said that that was perfectly sufficient, really, for the general additive white Gaussian channel. When we were talking about binary code rates less than 1/2, or nominal spectral efficiencies less than one bit per second per Hertz, which are equivalent statements.

So what happens if we want to go over channels where we want to send at higher spectral efficiencies? We want to send four bits per second per Hertz, as we certainly do over, say, telephone line channels, which are 3,000, or 4,000-Hertz channels, over which we want to send tens of thousands of bits per second. Or many radio channels will support high numbers of bits per second per Hertz, and so forth. Well, obviously, you can't do that with binary codes.

So, today and Wednesday in this chapter, we very quickly, of course, go through the series of techniques that have been developed to go over bandwidth-limited channels such as the bandwidth-limited additive white Gaussian noise channel. Clearly, you're going to need a non-binary constellation, and somehow code on that constellation.

But what you'll see is that both the techniques and much of the development are very reminiscent of the binary case. Historically, the binary case was always done first, and then the generalizations to non-binary followed. And so we'll see -- you can start off with very simple, hand-designed constellations, as we were doing back in the early chapters, four and five. We played around with little m equals eight

constellations and tried to optimize them from the point of view of, say, minimizing power for a fixed minimum distance between points in a certain number of dimensions in Euclidean space.

Now, we want to go up to considerably longer constellations. So that was kind of at the level of Hamming codes and single parity-check codes and very elementary binary coding techniques like that. So we want more complicated codes, let's say moderate complexity codes. We will find things that are analogous to block codes, linear block codes, namely lattices. These are structures in Euclidean space that have the group property, as linear block codes do. And then we'll find trellis codes, which are the Euclidean space analog of convolutional codes.

And basically, our goal in the next two lectures is to get to the level of a union-bound estimate analysis of the performance of these kinds of coding techniques, which, like their binary analogs, sort of sit in a moderate complexity, pretty good performance, get up to 6 dB coding gain, can't get to capacity. So that's as far as we'll be able to go in chapter 14.

In the last page or two of chapter 14, I briefly mention higher performance techniques. And it's generally accepted that you can get as close to capacity on these bandwidth-limited channels, additive white Gaussian noise channels as you can on the binary channels, in some cases just by adapting binary techniques in a kind of force-fit way, and others that are somewhat more principled, I might say.

But in any case, by adapting the binary techniques, you can get turbo codes, low-density parity-check codes, capacity approaching codes of various types for the bandwidth-limited channel. But I just hint how that can be done in one page, and I can't include it in the course. So that's where we're going.

So basically, we're trying to design a constellation set of signal points. Constellation c or signal set or alphabet, script c, if you like. And we're doing the kind of things we were doing back in chapter four and five. What are some of the things that the parameters we have-- we're going to do this in n-dimensional space. We're going to be concerned with the size of the constellation, the log of the size, the number of log

2 of the size, the number of bits we can send.

We call this m at one point. That will determine the number of bits we can send per n dimensions, or we can normalize it per two dimensions. Typically, I said, in bandwidth-limited regime, we'll normalize everything per two dimensions. We're going to have an average power of the constellation, which I'm just going to write like that. We're going to have a minimum squared distance between points in the constellation, which I'm going to write like that.

And the idea is to optimize the trade-off between this and this. Clearly, if I take a given constellation and I scale it up, I'm going to get better minimum squared distance without the cost of increasing power. So I either fix the minimum squared distance between points and try to minimize the power, the average energy over the centroids of a bunch of spheres, or vice versa.

So when we do this, this is kind of the game, we get into a classical problem called sphere-packing. If we say, hold the minimum squared distance constant, that means we want to have a certain minimum distance. Here I'll make it d_min over 2. That means that there's going to be a sphere of radius d_min over 2 around each point in this space, which must not be violated by the sphere of any other point. So basically, we're trying to pack spheres as closely together as we can. And we're maybe given the dimension. Here I have two dimensions to play with. We may be given the number, m.

In general now, we're going to be thinking of m becoming very large. We want to go to arbitrarily large number of bits per second per Hertz. So think of schemes that will continue to work as we add more and more points. And if I do that in two dimensions, we get penny-packing. Take a bunch of seven pennies out of your pocket, and try to squeeze them together as tightly as you can. That's effectively what we're trying to do. They maintain their fixed radius, and you get something that quickly starts to look like what this is called, a hexagonal sphere-packing, because the centers line up on a hexagonal grid.

And in fact, I forget exactly what we did. We did some of these back in the early

chapters. And let me draw a 16-point constellation, which I remember we did draw. Here's a constellation consisting of 16 pennies. And subject to my drawing ability, it's packed as closely as possible for a fixed minimum distance between the points.

And as far as anyone knows, this constellation, consisting a set of points on a hexagonal grid, is the most dense packing of 16 points in two dimensions, as measured by the average power of the centers. And it's not very symmetrical, but you can see it's based on a symmetrical packing. This hexagonal, this is basically a subset of 16 points from the hexagonal lattice. If we extend these points in all directions-- again, I haven't done it very well-- but the hexagonal lattice is simply the underlying lattice here.

Let me just draw a set of points that look like this. I'm going to fail again. Sorry, I'm better at rock'n'roll than I am at classical music. Anyway, this is the lattice, and it's called hexagonal because, if we draw the decision regions for this lattice, if we consider this as a set of points in a constellation and draw the decision regions, it looks like that, this little hexagon, defined by the six nearest neighbors. They define six sides of a hexagon.

And so each lattice point has a little region of space that belongs to it, you can think of as its decision region. Now, what makes this a lattice? What's the definition of a lattice? We'll say an n-dimensional lattice. Very simple and elegant definition for a lattice. It's a discrete subset. That means a set of discrete points of Rn, sitting in Euclidean n-space, that has the group property. And you now all remember what that means. It means the group property, under ordinary addition of vectors in Euclidean n-space, addition or subtraction, of course.

What are some of the implications of having a group property? That means that the sum of any two vectors is another vector in the lattice. Does the lattice have to include the 0, the origin point? Anybody? Oh, good. I'm going to ask this on the final. Come on. Does it or doesn't it?

AUDIENCE: It does.

PROFESSOR: It does. Why?

AUDIENCE: [INAUDIBLE].

PROFESSOR: OK, that's one good answer. It has to be closed under addition and subtraction. Subtraction of the point from itself is going to give you 0. A more mathematically trained person would probably, say, well, it obviously has to have an identity. Every group has an identity element. And that means, if lambda is in the lattice, does minus lambda have to be in the lattice? Yeah, every element has to have an additive inverse, and so forth.

Well, so if I have a lattice like this, that means that one of these points has to be the origin, say this one. Now, of course, you can have a translate of a lattice, and geometrically it has all the same properties as the lattice itself, at any fixed t to all the lattice points. In other words, make the origin somewhere else. But that's not itself a group. That's the co-set of a group. It would translate as a co-set operation.

So lambda must be a group, and lambda plus t, any lambda translate is a co-set of lambda. So most of the things you say about lambda are also true for lambda plus t, but in order to fix a group, we need the origin to be part of the points set. OK, if it's a group, what does that mean? That means, if we take any particular point, say this one, regard it as a vector, then call this lambda_1 -- or let me it g1, more suggestively. Does g1 plus g1 have to be in the group? Yes.

So that forces this point to be in there, 2g1, 3g1, so forth. All of these have to be in there, as well as minus g1, minus 2g1, and so forth. So basically, once we found a generator, all integers times that generator have to be in the lattice. So that means there's a cross-section along the axis defined by g1, such that these points are simply the integers, scaled by whatever the norm of g1 is.

And of course, that holds for any of these neighbors here. You want to look for nearest neighbors of 0's. Take g2 and then, of course, all of its integer multiples have to be in the lattice. So we've got a set here that, in some sense, spans n-space. We have now two vectors, g1, g2, which are not linearly dependent. They're

sitting in 2-space. So if we take all real linear combinations of these two vectors, we're going to get all of 2-space.

If we take all integer linear combinations of these two vectors, they all must be in the lattice, right? By the group property? We've already proved that all the integer multiples of either of them is in the lattice, so any thing plus or minus those two. So certainly, the lattice includes the set of all integer multiples of a_i times gi, where i equals 1 to 2, where a1, a2 is in the two-dimensional integer, lattice Z-squared.

Could there be any other points? There actually could, if you made a mistake and picked your initial vector to be 2g1, because then you wouldn't have taken account of g1. So you've got to take your two initial vectors to be two of the minimum norm points in the lattice. But if you do that, then you can easily prove that this is all of the lattice points.

So in fact, a two-dimensional lattice is equal to a set of all integer linear combinations of two generators, which we can write as a little two-by-two generator matrix, G. Or very briefly, we could write lambda as G times Z-squared, and we can view it as just some linear transformation of Z-squared.

So in this hexagonal lattice, we can view it as starting out with a square grid. Of course, Z-squared itself is a lattice. Z-squared is a group. This is just the square lattice in two dimensions. We have Z_n as a lattice in n dimensions, in general. So I take Z-squared, Z_n, more generally, and I distort it. I just run it through some g, which is a linear transformation, and slants some of the generator points.

And that's a general way of getting a lattice. That's not the way we usually visualize the hexagonal lattice. We usually visualize it in such a way as to recognize that it has six-fold symmetry. But that's certainly a way to get it. And this has one interesting consequence. One of the measures we want for a lattice is the volume per lattice point. In here, the volume, you can divide this up into volumes that are all of equal size, and we call that the volume of A2. This lattice -- hexagonal lattice, is called A2, and it's the volume of that hexagon.

1 over the volume of that hexagon is the density of points in 2-space, if you like. So density is 1/volume. I mean, think of every point as having a unique volume, and if we basically consider the cells around each lattice point, they fill up 2-space, or the perfect tessellation tiling of 2-space. These hexagons will.

One easy consequence of the group property is that all the decision regions have to be congruent, in fact, just translates of one another by lattice points. The decision region around this point is the same as the decision region about 0, translated by a lattice point, namely the center of that decision region. I won't prove that but it's true and easy to prove.

So that's one way of finding what's the volume. The volume will mean the volume of 2-space per lattice point. But what's another way to do that? Another way is just to take this little parallelotope, whose volume is easier to calculate. And we can see that we can equally well tile 2-space with these little parallelograms, just anchor -- in this case, this is the one that's anchored in the lower left corner by the origin.

Then we take another one that's anchored by this point. We take another one that's anchored by this point, another one that's anchored by this point. And it's clear that's another tiling of 2-space by regions, one for each lattice point that completely fills 2-space with overlaps only on the boundary, which don't count.

So the volume of A2 is also the volume of that, what's called a fundamental parallelotope. And what's the volume of that fundamental parallelotope? Well, one way of calculating it is to take the volume of Z-squared. We can view this as the distortion of Z-squared, which looks like this. So if this is Z-squared, what's the fundamental volume of Z-squared? The volume of Z-squared per lattice point. This is (0,1,-1,1, so forth,1,1). What's the volume of Z-squared per--

AUDIENCE: 1

PROFESSOR: --1. Clearly. And we have this decision region. This is little square side 1, or this parallelotope is also little square side 1. It's volume is 1. General geometric principle, if we take this and distort it by G, then what's the volume of, say, this

parallelotope going to be? This parallelotope goes by G into this one, and its volume is the Jacobian, which is the determinant of G.

So the volume of a lattice is the determinant of any generator matrix for that lattice. Just a nice little side fact, but it's an example of the kind of things you get in Euclidean space that you don't get in Hamming space. Of course, there are various generator matrices that you could use to generate this lattice. But whichever one you choose, this is going to be true.

So this is an invariant among all the generating matrices. So that's a lattice. I haven't mentioned what's the most important thing for us about a lattice. A lattice is a group. What's the fundamental property of a group, which I guess was embodied in the alternate set of axioms for the group? If I take the group plus any element of the group, what do I get?

AUDIENCE: [INAUDIBLE].

PROFESSOR: The group again. Let's interpret that geometrically. One of the nice things about this subject is you're constantly going back and forth between algebra and geometry. Geometrically, what does that mean? That means, if we take this lattice, say the origin is here, and we shift it, say, so that the origin is up here, then nothing changes. Everything is invariant to this shift. We can do this for any lattice point.

That shows you that all the decision regions have to be congruent. It shows you that the number of nearest neighbors, the minimum distance from any point to its nearest neighbors is always the same for any lattice point. Basically -- it's a lattice is geometrically uniform. That means you can stand on any of the points and the world around you looks the same, as if you stood on any of the other points.

Think of these as stars in the universe, and somebody blindfolds you and drops you on one of these points. Can you tell which one of those points you've been dropped on if nobody's written anything on these points? No, you can't tell. The world looks the same from whichever one you drop on. So that means the minimum square distance from any point is the same as every other.

So the number of nearest neighbors at minimum square distance is the same. It means the total distance profile, starting from any point, is the same. It means if we were to take one of these points and send it over a Gaussian channel from a very large set, and we take one of the points from the interior and send it over an additive white Gaussian noise channel, the probability of error is going to be the same, regardless of which point we send it from, because the probability of error is the probability of falling outside your decision region. And since the decision region always has the same shape, it's invariant to which point you send.

So this geometrical uniformity property, which that's the general name for it and it has all these specific consequences that I just listed, is the principal geometrical property of a lattice. This just follows from its group property. You don't have to be a lattice to have this property. For instance, the translate of a lattice is not a group, technically, but it obviously has the same geometrical uniformity property. And there are more elaborate examples of things that look less like lattices that are nonetheless geometrically uniform. On the other hand, lattices definitely are geometrically uniform.

So this seems to be a good thing to do, because what's our objective? We're trying to have d_min-squared be the same for every point in our constellation. We want to tack these pennies as close together as we can. That means we want to hold d_min-squared constant for every point that we're going to use. And so lattices have that property. So that seems like a good way to do sphere-packing.

And in fact, sphere-packing, in n-dimensions, is a very old mathematical subject. How many spheres can you pack around a three sphere, is it 12 or 13? That was debated in the 18th and 19th centuries, and I think it's only finally just been resolved a few years ago. This turns out to be not an easy problem. The densest sphere-packing in four dimensions was found in the mid-19th century. In eight dimensions, I think around the turn of the 20th century. In 16 dimensions, it's the Barnes-Wall lattice, which we'll see a little later, which was found in '54, about the same time as Reed-Muller codes, or maybe it was '59, maybe a little later than Reed-Muller codes.

The Leech lattice was certainly found a little earlier than that, but around the same time as the Golay code, to which it's a very close cousin. And so forth. So you might think that these questions had all been settled, but in fact they're still open for most specific dimensions, n. Conway and Sloane have the authoritative book on this, which basically lists what we know about the densest sphere-packings in all dimensions.

So here's our idea. Our idea is we're going to go consult the mathematicians and find out what the densest sphere-packings that they found are in various dimensions. In two dimensions, it's the hexagonal lattice. In one dimension, it's the integer, and basically the integers are the only packing with regularity properties in one dimension. And two dimensions, you could consider Z-squared, or you can consider A2. These are the two obvious candidates, the square and the hexagonal packings.

In three dimensions, you get face-centered cubic packing. And what's the BCC? The base-centered cubic packing, I guess.

AUDIENCE: Body-centered.

PROFESSOR: Say again.

AUDIENCE: Body center.

PROFESSOR: Body-centered, thank you. I'm sure you're right. So anyway, there's some things that are known about this from classical mathematics and current mathematics. So our idea is we're going to try to find the densest possible lattice in n dimensions. But for digital communications, for coding, we only want a finite number of points from that constellation. So what we're going to do is we're going to create our constellation based on a lattice and on a region of n-space. And the idea is simple. We're just going to intersect the lattice. Or maybe let me allow a translate of a lattice with the region.

And make this a finite, compact region. And this will give us a certain finite subset of

the points in the lattice. Now, when I draw a constellation like this one, of a size 16, that's what I've done. I've basically taken A2. Or it's actually a translate of A2. I think the origin is in here somewhere. And I draw a sphere around the origin. And I take the 16 points that happen to fall within that sphere. And I call that my constellation.

So that's the idea in general. Think of the region, r, as a cookie cutter. We're going to take the cookie cutter and chop out a finite number of points from this nice, regular, infinite grid. And that will give us a constellation. So especially for large constellations, we'll be able to make some approximations that will allow us to analyze that kind of constellation very easily.

And actually, the best constellations that we know are of this type. We've taken this baseline, m-PAM. Well, what's m-PAM? We take the lattice plus t, let's say to be Z plus 1/2. So that means we don't take the origin, but we take the points 1/2 minus 1/2, 3/2. They're equally spaced points, but we've shifted them so as to be symmetric about the origin. 5/2 minus minus, so forth. So this is Z plus 1/2 is our lattice. And then if we want 4-PAM, we take a cookie cutter consisting of this region.

So this is the region going from minus 2 to 2. It's not unreasonable to expect that, if we take a region of length 4, we're going to get 4 points, if, again, it's symmetric around the corner, because each of these points takes up about one unit of space over here. So if we made it from minus 4 to plus 4, we'd have a region of length 8, and we'd get 8 points. And so forth. So that's how we do m-PAM, and then scale it if we want to. But this is the basic idea.

Similarly, for m by m QAM, like everything else, it's just doing the same thing independently in two dimensions. Here we take our lattice to be Z plus 1/2 squared, which is just the offset Z-squared, offset so as to be four-way, to be roughly symmetrical to four-way rotations, 90-degree rotations.

So here is Z-squared plus (1/2, 1/2). And then, again, here to get 16-QAM, for instance, we might take the region to be what? (-2,2)-squared which, again, has area 16. You might think of this as just taking the little square around each of these. Each of these squares has area 1. So when we take the union of all those squares,

we get the region.

That's another way of characterizing this region. It's just the union of all the decision regions of the points that we want. If you had started from the points, you could certainly invent a region that is the union of those decision regions. It would have an area equal to the number of points times the volume of 2-space per each point.

That's the kind of approximation we're going to use. The bounding region is clearly going to have something like the number of points in the constellation, times the fundamental volume as its volume. So this is our idea. This is what we're going to call a lattice constellation. And we're going to investigate the properties of lattice constellations, not the most general thing we could think of but good enough.

Seem to have the right sort of property. And we're going to take it at least as far as the union-bound estimate, so we can get a good estimate of probability of error if we take one of these constellations, like m-by-m QAM, and use it to transmit over a Gaussian noise channel. And furthermore, for large constellations, meaning as we get well up into the bandwidth-limited regions of large nominal spectral densities, we're going to be able to separate the analysis into lattice properties and region properties. And we'll basically be able to just add these up to get the overall analysis, by making some judicious approximations that become exact in the large constellation limit.

So let's now restrict ourselves to lattices, or translates of lattices, and start to analyze them. What are their key parameters for our application, which is digital communications? Well, one is the minimum squared distance between lattice points. We've seen that doesn't depend on the particular point. So this is the minimum squared distance from any lattice point. The number of nearest neighbors. The volume per lattice point. And is there anything else? It seems to me there's a fourth one. No. I guess that's all we need to know.

So I want to combine these into a key figure of merit for coding. Again, we did this back in chapter four for a complete constellation. We found the figure of merit by holding the average power fixed for a fixed number of points, m, on a fixed

dimension, n, and maximizing the minimum square distance, or, conversely, holding the minimum squared distance fixed and minimizing the average power. So we get some figure of merit that was kind of normalized, and we could just optimize that one thing.

The key figure of merit was called by 19th-century mathematician, Hermite's parameter, but we are going to call it the nominal coding gain because that's what it's going to turn out to be. And it's just defined as follows. The nominal coding gain of a lattice is just this minimum squared distance, so it goes up as the minimum squared distance. But then we want to normalize this. And the obvious thing to normalize it by, we want something that's going to scale as a two-dimensional quantity. If the lattice was scaled by alpha, then this would go up as alpha squared. So we want something that'll scale in the same way.

And what we choose to normalize it by is the volume of lambda per two dimensions, if you like. So you can see immediately, this is invariant to scaling, that the nominal coding gain of lambda is the same as the nominal coding gain of alpha lambda. Is alpha lambda a lattice, by the way? If lambda's a group, and we have any scale factor alpha greater than 0, then alpha times lambda is also a group. Everything's just multiplied by alpha. So alpha lambda is a lattice.

And this is actually one of the homework problems. If you scale everything by alpha, d_min-squared goes up by alpha squared. The volume goes up by alpha to the n, for an n-dimensional lattice. But when we take 2 over n, we're going to get back to alpha squared. So alpha c of lambda is alpha c of alpha lambda. So this is what you want. You want a kind of scale-free version of a lattice, because, obviously, when we use this, we're going to feel free to amplify it or compress it any way we want, so we do with QAM.

All right, so this is normalized in that way. We prove in the homework it's also invariant to other things. If you took, say, an orthogonal transformation of this lattice, a rotation, reflection, multiply it by a matrix that doesn't change the scale, think of it geometrically as an orthogonal transformation, is that going to be a lattice? It is

going to be a lattice. Everything's just going to have some h out here, times G times Z-squared, and that's still of the form of a lattice. Or call it U for orthogonal, unitary.

So we multiply it by some orthogonal matrix, U. It's still a lattice. If we look in here, orthogonal transformations don't change squared distances. We're not going to change this -- it doesn't matter. Actually, I might. No. It's a rigid rotation, I'm sorry. Everything here stays the same. Hexagonal lattice still is hexagonal lattice, because it's a rigid rotation or reflection geometrically. And the volume clearly stays the same, so none of these things changes. And orthogonal transformation isn't going to change this quantity either.

So we can rotate this or do whatever. And then, finally, there's an even more interesting one, which is, if we take the lattice, lambda to the m, this is simply the Cartesian product. It's the set of all lambda_1, lambda_2, lambda_m, such that each of these lambda_k is in Lambda. So it's the set of all n-tuples of elements of Lambda. This clearly lives in dimension mn, so it's a much higher-dimensional lattice. The simplest case is Z to the m - it lives in m dimensions, whereas Z is down in one dimension.

So it lives in dimension mn. It's a way of constructing high-dimensional lattices from low-dimensional lattices. It's something that, in communications, we consider to be almost a non-event. Sending a sequence of elements of lambda is the same as sending a sequence of elements of lambda_m. In fact, if we just send a sequence of things from here, and a sequence of things from lambda, there's no difference from a receiver's point of view. So we regard these two lattices as equivalent, from a communications point of view.

Well, what happens to d_min-squared? d_min-squared doesn't change, because we can always-- d_min-squared is, by the way, the weight of the smallest-- should have said that. You can always do things relative to 0. d_min-squared, therefore, is the squared norm or lowest squared energy of any non-zero lattice point, same as we did with Hamming codes, just by the group property. K_min is the number of such lowest Euclidean weight points.

OK, here, what's the lowest weight point? It would look something like this, where this is one of the lowest weight points in lambda. So d_min-squared doesn't change. This changes, but we're not considering that. The volume, what is the volume? It turns out that it's not hard to show that V(lambda to the m) is V(lambda) to the m. And therefore, you can show that this is figure of merit doesn't change, even if you take Cartesian products.

So for instance, I chose that the nominal coding gain of to the m is equal to the nominal coding gain of Z. Which is what, by the way? I should have done this as the first example. What's the minimum squared distance of Z? 1. What's the volume of Z? 1. So this is 1, or 0 dB. We're going to measure nominal coding gain in dB, which, well, we're getting us into communications.

What's the nominal coding gain of the hexagonal lattice? By asking this question, I'm basically asking, is the hexagonal lattice denser than Z-squared in two dimensions than the square lattice in two dimensions? And if so, quantitatively, how much denser is it?

AUDIENCE: [INAUDIBLE].

PROFESSOR: OK, well, there's a root 3 in it, but that isn't exactly the answer, I don't think. There are various ways that you can do it, but one of the ways is to take, as a generator matrix, (1, 0; 1/2, root 3 over 2). You like that as a generator matrix for the-- That's taking this and this as the two generators, and recognizing this as 1/2, and this as root 3 over 2. That makes the length of this equal to 1/4 plus 3/4, equals 1 square root, and it's 1.

So these are my two generators of length 1 in a hexagonal lattice of minimum squared distance 1. So based on that, I can say that the volume of A2 is what?

AUDIENCE: [INAUDIBLE].

PROFESSOR: Root 3 over 2. Not hard to take the determinant of that, right? And what's the minimum squared distance in this scaling? What's the minimum non-zero weight -- squared norm, it's 1. So this is equal to 1 over root 3 over 2, which is equal to 2 over

root 3. Which is equal to what? This is 3 dB. This is 4.8 dB, roughly. Square root is 2.4 dB. So this is about 0.6 dB.

That's where it pays to be facile with dB's. This area, by the way, if you tried to compute the area of this little hexagon here, what would it be? If the area of this parallelotope is the square root of 3/2, what's the area of this hexagon? The same, right? They're both space filling when centered on lattice points. So they have to have the same area. So that's an easy way of computing the area of a hexagon, or of this particular hexagon anyway.

But you would get the same result in either way. So comparing this with Z-squared, I now have a quantitative measure of how much denser A2 is than Z-squared. Put in one way, if I fix the minimum squared distance of each of them to be 1, if I'm doing penny-packing and I say I want pennies of a certain radius to be packed into two-dimensional space, then this is 1 over the volume, which is basically the density. The density of A2 is 2 over root 3, times [UNINTELLIGIBLE] as the density of points in z squared. I get more points per unit area with the same distance between them using A2 than I can with Z-squared.

So that's the significance of this parameter. So it's an obvious thing for Hermite to have come up with. It's going to turn out to be equally fundamental for us. So that may be all we want to do on lattices for the time being. Yeah, seems to be.

So we're talking about lattice constellations. The other thing we have to develop is the properties of regions. So let me talk about regions, R -- I'm thinking of a compact region in an n-dimensional space. So again, let me fix the dimension to n. And we're going to have a certain volume. That's obviously an important characteristic of a region. And I'm going to assume that this is finite.

And what's the other thing that we basically want? Well, when I pick a constellation like this, by intersecting a lattice with a region, eventually I'm going to let the points in this constellation be equiprobable, as I always do in digital communications. 16 points. Say they all have probably 1/16, the uniform probability over the discrete points.

One of the approximations I am going to make -- so what's the average power of that constellation? We could figure it out by taking 1/16 times the energy. I should say the average energy, I'm sorry. We're not talking power here. We're talking energy, but I read it as P. What's the average energy? It's 1/16 times the L2 norm of each of these points, if you like that. It's simply the Euclidean squared norm.

And we get some average. But here's the key approximation principle. I'm going to say the average power of this constellation, especially as the constellations get large, is simply going to be approximated as the average energy of a uniform distribution over this bounding region. Suppose I simply put a uniform continuous distribution over this circle, this sphere, two sphere that I've used to be my cookie cutter. I'm going to say that's going to be a good approximation to the average energy of the 16 discrete points, which are, by construction, they're spread uniformly over this region.

The approximation is, I'm kind of approximating a unit impulse of weight there by distributed weight across its-- well, actually across its whole little region. If we have a hexagon around each one, uniform distribution over the hexagon. And if that's a reasonable approximation, then it's reasonable to say that the average power of the whole constellation is just the average power of a uniform distribution over this region.

It's good exercise to try it for some moderate size cases. And you'll find it is a very good approximation. In any case, it's the one we're going to use. So the other key parameter is going to be P(R), which is, in words, the average energy of uniform distribution over R. Sorry, script R. Now, when we write this out, this actually gets a little formidable. So I think it's better to remember what we're actually trying to compute. Oh, and one last thing, we're going to normalize it per dimension.

So what do I mean? What's a uniform distribution over R? This is P(x equals 1) over the volume of R for x in R and 0 otherwise. Uniform within, 0 outside. And what does it have to equal? It has to be equal to 1 over V(R), because its integral has to be equal to 1 if it's a probability distribution.

So to compute P(R), we basically take the integral from over R, of this probability distribution, 1 over V(R), times what? The energy per dimension. The total energy of x is x-squared, the L2 norm of x. Normalize per dimension -- we're going to put an n over there -- dx. So that's an equation for what it is I wanted to compute.

I think it's much harder to remember the terms in this than to remember this verbal description of what we want. But you may be more literate than I. Now, let's define -- we want to normalize quantity comparable to Hermite or coding gain over here. The normalized quantity we're going to use -- we want to take P(R) -- and what should we normalize it by to make it scale-invariant, let's say again?

Again, if I take the region, I scale the region by alpha and get alpha_R. What's going to happen to the average energy of a uniform distribution over R?

AUDIENCE: [INAUDIBLE].

PROFESSOR: It's going to go up as alpha squared. Energy scarce, goes as the square of the scale factor. So again, I want to normalize it by V(R) over 2 to the n, because this is also something that will go up as alpha squared if I scale R by alpha.

So this is what's known as the dimension-less or normalized-- it has both names-- second moment. Another name for this is the second moment per dimension of a uniform distribution over R. This is a normalized or dimension-less second moment. I think I call it normalized in the notes, but it's always called G in the literature.

And let me just introduce here-- well, what's the normalized second moment of an interval? Say, a symmetric interval around the origin. G to the minus 1 to the 1, which is like the interval that we used to pull out the m-PAM signal structure. So let's take n to be equal to 1, R just to be the interval from -1 to +1. And V(R) equals what? It's the length of R, which is 2.

What's P(R)? The integral from -1 to 1, x squared, dx, which is 2/3. I think so.

AUDIENCE: [INAUDIBLE].

PROFESSOR: Right, 1/2. Thank you. That's 1/3. All right, so what is G(R) going to be, or G going to be? This is going to be P(R) over V(R)-squared. And so this is going to be 1/3 over 4, or 1/12. OK, that's not as nice as this over here, but it is what it is.

There's a number that shows up in quantization, for instance, uniform scalar quantizer, you'll find a 1/12 in there. Why? It's because of this equation. If we ask again about the invariances of this-- this is the last homework problem for this week-- it's invariant not only to scaling, but, again, it's invariant to orthogonal transformations, because orthogonal transformations won't affect either of these things. A rigid rotation, about the origin, won't affect V(R) or P(R). And furthermore, it's invariant to Cartesian products, again.

If we just have a series of regions, a region made up basically as the Cartesian product, like an n-cube is a Cartesian product. We just pick each of the dimensions independently from some set R. Then that's not going to affect the normalized second moment either. Exercise for the student.

So from that result, I can say that G([-1 1]) to the m. Which is what? This is an m-cube of side 2. If I say each of the coordinates, x1, x2, x3, up to xm, has got to be xk n times [-1 1], then what I'm going to get, the region defined by that is an m-cubed, centered on the origin of side 2.

So the normalized second moment of that is going to be 1/12. This is where the shortcut begins to help you. I guess I'm trying to put too much on one board. But the last thing I want to put up on here is the shaping gain. The shape gain of a region is defined as the shaping gain of the region is equal to 1/12. So we're kind of taking as a baseline, the m-cube, in any number of dimensions n, over G(the region).

In other words, how much less is the dimension-less second moment, the normalized second moment, than what you would get for an m-cube? In every dimension except for one, a sphere is going to be a better region, from a second moment point of view, than an m-cube. An m-cube has corners. Really, it's pretty obvious that, if you want to minimize the normalized second moment in any number of dimensions, you would choose an m-sphere, because if you have any wrinkle

that comes out of an m-sphere, anything that's not an m-sphere, you should try to squash it back in it. And that'll make it an m-sphere.

You'll take higher energy and make it lower energy for the same little unit of Play-Doh or mass stuff. So a sphere will give you the best normalized second moment in any number of dimensions n, so you're always going to get a G(R) that's less than 1/12. That's good. And this measures the amount of gain.

Again, you can measure this in dB. That's going to translate directly to dB. I suppose I could do an example for the sphere in two dimensions, but I won't. How are we doing on time? Let me see if I can do the union-bound estimate. That would be a trick, but I think it's doable, because this is where we're going.

So the union-bound estimate, which is something we did way back in chapter five or something, is basically, given a constellation, which we're going to have a lattice constellation based on some lattice lambda and some region. So this is like our m-by-m QAM, or it might be this guy, which is better than 16-QAM, in some sense. In two senses, actually.

The union-bound estimate, what is it? The probability of error is approximately equal to-- let me do it per block. So I'll have the number of nearest neighbors per block of our constellation times Q to the square root of d_min-squared of our constellation, over what is it? 4 sigma squared? 4 sigma squared.

And if you remember how we got this, we took the union-bound, we took all the pairwise error probabilities, and from that, we get contributions from kd at every kd contributions at distance d. We added them all up in the union-bound. We said, well, for moderate complexity and performance, at least this is going to be dominated by the minimum distance. So this is the minimum distance terms. This is the number of minimum distance terms. And this is error probability -- the pairwise error probability we get for each minimum distance term.

Now, to compute this, I'm going to use three approximations, but two important ones, this continuous approximation that I've already referred to. First, I'm going to

say that the average power of my constellation is approximately equal to just the second moment of R. The average energy per dimension of R.

And then I'm going to say, furthermore, how many points are there in the constellation? This will affect what rate I can go at. The average number of points in the constellation -- also mention this-- is going to be approximately equal to simply the volume of my bounding region, V(R), over the volume taken up by each point in my lattice.

That makes sense, doesn't it? What I want to do here to get 16 points is I take a sphere whose area is roughly 16 times the area of one of these little hexagons or one of these little parallelograms. And I'll get about 16 points, because that's the density of it.

And there's a final little approximation, which is that I'm going to assume that we're somewhere in the interior of the lattice, so that the average number of nearest neighbors in the constellation is simply equal to the number of nearest neighbors in the lattice. So that's a reasonable approximation, and that's a very obvious one. If I take any constellation based on the hexagonal lattice, for a large enough constellation, the average number of nearest neighbors is going to be 6. If I take it on the square lattice, the average number of nearest neighbors is going to be 4.

But we know we're not terribly interested in that either. Let me do a little manipulation here. I'm going to write this as Q of d_min-squared of c, which clearly I'm going to also assume that d_min-squared of c is equal to d_min-squared of the lattice. So I'll make that d_min-squared of the lattice. And I want to normalize that by v of lambda, to the 2/n, whatever dimension n I'm in, to get something that we're familiar with. So I need a V(n) over 2 to the n.

And anticipating a little, I'm going to put a V(R) over 2 to the n. And then over here, I'm going to put V(R) over-- Is this going to come out right?-- over 2 to the n, over-- this seems upside-down. Ah, no, it isn't upside-down. Because I'm using this expression here, I do want 1 over G(R).

So this is 1 over G(R). If I multiply this by 1/12, now I get 1 over G(R), so I need to put a 1/12 here. And then I have a P(R) over 4 sigma squared. So does everyone agree I can write it that way? This'll be good. Go out with a bang.

So what are all these things? This is the coding gain of lambda. This, according to this, is the size of the constellation to the 2 over n, approximately. The rate of the constellation is log 2 times the size of lambda, R. So the rate is equal to just log 2 of this quantity. So I can say 2 to the R. This is 2 to the R equals this. So this would be 2 to the 2 over n, 2R over n. Is that right? Again, I'm unsure if that's what I want.

This is the shaping gain of the region, these three things together, times 3, times-- what's P(R) over sigma squared? That's the SNR. That's the average power per dimension. And this is the average noise power per dimension. I've used up these in that. So I've got a 1 over 1/12. That's 12 times that. So this is 3 times SNR.

And let's see. This is 2R over n is just 2 to the--

AUDIENCE: Isn't that just 2R? Because [UNINTELLIGIBLE] normalized by the dimension.

PROFESSOR: Yeah. And furthermore, it's minus, because it's upside-down. So it's 2 to the - 2R. Actually, what I want is the spectral efficiency. And the spectral efficiency is the rate per two dimensions. So this is just 2 to the minus rho. So with a little help for my friends, what's SNR over 2 to the rho?

AUDIENCE: [INAUDIBLE].

PROFESSOR: It's approximately SNR norm. Certainly true as row becomes large. It's actually SNR over 2 to the rho minus 1. So this is SNR norm. OK, so with a little hand waving and a few high SNR approximations, we get that, in summary, for the UBE, the probability of error is approximately K_min(lambda) times Q to the square root of the coding gain of the lattice, times the shaping gain of the region, times 3SNR norm. And that's a wonderful approximation.

Let me just call out some of its properties to you, and then we'll talk about them next time. But first of all, there's nothing here-- all these terms involve either lambda or

R. So we have completely separated, in this analysis, the effects of the region from the effects of the lattice. We can choose the lattice and the region independently. And they independently affect how good a performance we get. For the baseline, which was either m-PAM or m-by-m QAM, we have, for the baseline PAM or QAM, we've set it up so that both the coding gain of the lattice and the shaping gain of the region are equal to 1.

So for the baseline, this gives us what we already developed back in the early stages. I won't worry about this -- Q-hat, but it does give you the right coefficient out here, too, depending on whether you normalize it per two dimensions or what. It just gives you Q to the square root of 3SNR norm, which I hope you remember from chapter five. So this is our baseline curve for m-PAM or m-by-m QAM.

We just plotted versus SNR norm, the probability of error per two dimensions, which is what I recommended. And we got some curve. So this is the baseline. And now we've got everything separated out in a nice way, so that if somebody gives us a lattice with a certain coding gain and a region with a certain shaping gain, those both just add up as gains.

If that's all we had, forget the coefficient, we would just move to the right here, and it would give us effective reductions and the required SNR norm by the coding gain of the lattice and the shaping gain of the region. And we'd get the same curve moved out over here.

Again, as in the binary case, typically we're going to get a larger number of near neighbors, leading to a larger coefficient out here. We want to normalize this per two dimensions. Per two dimensions, it's only 4 for the baseline. But in general, we're going to get a KS of lambda, the number of nearest neighbors per two dimensions, which is going to raise this a little. And that will lower the effective coding gain down here.

But we've basically, by these large constellation, high SNR approximations, we've really reduced the analysis of lattice constellations to a very simple task, just to analyze what's the coding gain, what's the shaping gain, number of nearest

neighbors, plot the performance curve, end of story. So we'll start there next time.