MITOCW | 11. LTI channel and intersymbol interference

HARI BALAKRISHNAN: Hi. My name is Hari Balakrishnan. I'm your replacement professor. You guys know about replacement referees?

AUDIENCE: Yes.

HARI BALAKRISHNAN: No. So I'm the other professor in the class. And I generally start with materials that's a little bit later. But what we thought we'd do today is to talk about this Audiocom library. So before I get started and show you how to use this and all the different ways in which things will break because it's something the real world, let me see a show of hands. How many people have tried to install or run or do something with it?

OK, how many people don't have any idea of what I'm talking about? OK. And how many people have managed to get something to work? By "work," it's if you got the preamble to decode, it means something is working. All right how many people have got nothing? OK. Well, it'll all work. It's just a question of figuring out how to get it to work.

OK, so here's the story. In order to do something practical with all the theory that you're learning, it's important to try to implement something. Now, the theory we're learning applies to a wide range of communication channels. It applies to radio. It applies to wired links-- you know, ethernets or cables. It applies to optical links like infrared or free-space optical links.

And it also applies to audio. And we've decided to use audio as the vehicle to bring these ideas into the lab for two reasons. The first and most important reason is that it turns out it's the easiest piece of hardware. It's the most convenient piece of hardware that everybody has access to. All you actually need is a computer with a microphone and a speaker.

And in fact, I would imagine that in the next couple of years, it'll run on this thing too. So you could write apps on this that will make this work. Already, we have stuff working on an Android and iPhone to do much of the stuff. So the first one is just-- it's everywhere. Everybody has it. You don't have to do anything special to get it.

The second is that it turns out it actually illustrates many of the problems that we're talking about-- noise and this idea of how linear time-invariant systems-- an understanding of linear time-invariant systems applies to what we're trying to build to communication channels and how you use the ideas you've learned about LTI systems to improve the performance of whatever you implement.

I'm going to restrict myself to this Audiocom system, but these ideas apply across the board. So let me tell you how this system does its job. And this goes back to what Professor Verghese was telling you before. Ultimately, all of the information is modulated on top of a carrier. And a carrier, as you know, are sinusoid. So time goes this way, and this is the amplitude, or which we also call the voltage. So it's a time-invariant waveform.

What your receiver is capable-- so the transmitter is capable of generating these kinds of waveforms at different frequencies. So this might be, for example, 1 kilohertz or 1,000 hertz. We can go up to-- I've got mine to work up to 20 kilohertz, which is fortunate because I can't hear it. And there's a wide range of frequencies this could work in. In the lab, I found that things tend to work between 1 kilohertz and only about 3 or 4 kilohertz. So that's the range we're talking about. So you pick a carrier waveform for your transmission.

What the receiver's capable of doing is-- it's a digital receiver. So what it's capable of doing is receiving data, receiving signals from the sound card at a certain number of samples per second. We're going to call that the sampling rate. We've used this term before. The sampling rate can be anything. By default in the system, we've picked the highest possible sampling rate of 48 kilohertz. In some cases, you might have to yank it down. You might have to go as low as 8 kilohertz. But 48 kilohertz seems to work in the lab. It works on a bunch of machines that I've looked at.

So we have two things. We have the 1 kilohertz, which is the carrier waveform. We'll use the notation FC for that. And we can receive samples at 48 kilohertz. And that's how the transmitter also, when it wants to send stuff on the air, it picks a sampling rate. It's going to pick 48 kilohertz. The receiver and sender need to agree on the sampling rate.

What happens then is very straightforward. If you have a carrier waveform that looks like this, what 1 kilohertz means is that we do the cycle 1,000 times a second, which means that one period of the cycle is 1/1000th of a second, or 1 millisecond.

Now, if I sample this at 48,000, or 48 kilohertz, or 48,000 times a second, in one of these periods, how many samples do I get?

AUDIENCE: [INAUDIBLE]

HARI BALAKRISHNAN: This is where you have to tell me something. Sorry?

AUDIENCE: 48.

HARI BALAKRISHNAN: 48. Great. So I sample at 48,000 times a second. Each of these things is 1/1000th of a second, so I get 48. What that means is that we pick a sample here. We pick a sample here. We pick a sample here. 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, and we keep going. 12, 12, et cetera. So we get 48, OK?

So if I were to send a pure sinusoid at the receiver-- and the way the sinusoid is transmitted is by providing these samples, each sample being equal to this voltage value picked from this curve. And I send each of those samples, and each of those in our implementation is a floating-point number, OK? Like a 32-bit floating-point number.

I pick a number here, 0. And then I pick something here, which is the value of the sinusoid at this point in time, and this, and this, and this, and all the way down. And I send those samples. The receiver is going to be listening at 48 kilohertz and picking up whatever it's getting on the audio channel. And that's what it's going to be assuming. That's what it's going to assume was sent.

Of course, in the real world, if I send at 1 volt, what might be received is 0.01 volts, or 0.2 volts, or whatever, because signals may, in fact, lose amplitude as you transmit them over the air. So that's the overall context.

I'm going to start by showing you a few things here. The first thing I'll show is this grapher program that actually is part of the package. But you need some GTK toolkits for this to work. So you need this for the lab, but this is just a useful test.

If I transmit a waveform, this program assumes that I'm transmitting data at 1 kilohertz. If I send data at 1 kilohertz-- and this is a pure sinusoid. Let me do that I'll explain its parameters in a bit.

[TONE PLAYS]

You can see that what's going on here is if I'm silent, the points get demodulated at plus 1 and minus 1 that correspond to the top and the bottom of the waveform.

[CLAPPING]

And if I make a noise, the effect of noise get visible. And in fact, you can see that sometimes the guy on the left goes to the right, and vice versa, OK? So that's actually a very visual effect of what noise does to your transmission. I'm going to show this again. I'm going to do this again so you can see what's going on.

As the amount of noise increases, you can see that-- you can imagine that what's going on here in the signal-- and this is called bipolar signaling, where you might send bit 0 as a particular kind of sinusoid, and a bit 1 at the other way, as the other kind. So the idea would be that the 0's get mapped to the left. The 1's get mapped to the right. And as we increase the amount of noise, you can start to see that they cross over, and we'd be making errors in our estimation.

[TONE PLAYS]

So things are working fine now. But as I [CLAPS] make noise, or as you start talking, and you create additional noise, we're going to start to see the effects of the noise. And this is annoying, so I'm going to turn it off. OK? So that's the first order way in which you determine that things are working. You don't need to use the graphing program to do that. And this is actually annoying to the eyes, so let me turn that off too.

So you run, and the documentation we've given you tells you how you go about step by step trying to debug this. So let me tell you what these options mean. It's all written up, so you don't have to take notes right now. Everything is there. In that command line, the dash little s 256 refers to the number of samples per bit. I'll get to that point in a second.

The dash capital S 1 means that we send-- capital S is the kind of source. You know, it's something about the nature of the source. Capital S 1 means that my entire information is sent as 1's, which means that I'm sending a pure sinusoidal carrier. All I'm doing is sending the sinusoidal carrier. Dash g just means, show me some graphs. And n refers to the number of bits I wish to send, OK? So these are all documented, and you don't have to worry about it.

So what we're going to do now is actually to show you what happens when we transmit some data. This is actually the very first task in the lab. You transmit some data, and you plot what the noise histogram looks like.

[TONE PLAYS]

So what we did was we sent a couple hundred bits at some appropriate number of samples per bit. You know, the way you tell if stuff's working is that line there. If it received the preamble-- and there's this long preamble. I'll explain what that word means. But if you receive the preamble, it means that stuff's working. If you didn't get the preamble, it means that-- it doesn't mean that things are completely broken. It just means you got to do a little bit more work.

We also plot out the signal-to-noise ratio of the transmission. And that shows up there. If that signal-to-noise ratio is something like 20 dB or 15 dB, things are fine. This system doesn't really work below 15 or 10 decibels. It means that the signal-to-noise ratio is too low, which means you either have to yank up the volume, or you have to go to a quieter location. Or-- and I'll explain this-- you have to change some of the parameters in the program.

Now, what does this graph look like? We'll ignore this for a second. For this number of samples, that was the normal distribution. At 0.44, you could look at it, and it sort of looks Gaussian. It's not quite a Gaussian. I'll explain why in a little bit.

What this picture show is this is what the noise-- this is what the received samples look like. And it has a mean value that's in the center. And then it's kind of got a shape on both sides.

This picture in the middle are the samples post-demodulation. It's actually want you received at the receiver after we ran the demodulation step.

And the stuff on top-- the blue refers to what was transmitted, which in this case was a pure carrier. The green shows what was received, which is some noisy version of the carrier, because we didn't send information other than just the carrier. This is effectively the first lab task. We have to do nothing more than run this a few times. And most of the first task in the lab, the first two tasks in the lab is just making sure that the stuff works.

Now, this entire system-- you know, everything works if this thing called the preamble is decoded. So what is this preamble? And why do we need it? Well, the problem is that, as you saw in this graphing receiver when I showed this to you, the audio hardware is going to be-- what it's doing is it's always listening. When you run the program, it's always listening on the channel. So it's getting data. Even when I'm not sending anything, it's getting something. There's always something on the audio channel.

So the question is, how does the receiver know that what it's receiving is part of a legitimate transmission? All communication systems need to solve this problem of synchronizing between the sender and receiver, and that's done using the preamble. The preamble is nothing more than a well-known bit sequence, sequence of bits, that the sender and receiver both agree on.

So in this case, our preamble is that long sequence there-- 1 0 1 1 0 1 1 1, et cetera. And there's some guidelines and rules of thumb that go into what makes a preamble good and what makes it not so good. We're not going to worry about that here. I'm just telling you that's the preamble. So as long as the receiver is successfully able to decode the preamble, it means that it knows that there's a legitimate transmission over the air, and it can start listening to it. So if you don't get the preamble, all bets are off as to what the heck's going on on the channel. And sometimes, you may not get the preamble.

Now, one of the things you would have to do, you would want to do, is when don't get a preamble, usually, it's a sign that either some samples are being lost because the audio hardware isn't able to cope with it. And the documentation describes how you deal with that problem.

Or it means that the sampling rate is too high, and maybe something is running on your computer that's-- I don't know. You have some video going on in the background, and it's not keeping up to read at 48,000 hertz. Or it's a sign that the signal-to-noise ratio is too low. Maybe you're in a very crowded location, or maybe the volume is too low. So those are usually the ways in which you go about fixing this problem with the preamble.

Now I want to show you one more noise graph. What I'd like you to do now is actually make-- we're going to try to do this with some music on the side. What I want to show you is that as you increase the amount of noise, the noise is captured in the variance of the Gaussian distribution. The more the noise, the bigger the variance in the distribution. So I'm going to try to do this by sending 1,000 bits. So this will take a little bit of time. So let's go send it without--

[TONE PLAYS]

So we'll send this without a huge amount of noise. I mean, there's some ambient noise in the room. And this will take a little bit of time to demodulate and get working. There's a couple of errors. The bit error rate is 0.002 bits, so we had an error here. And we saw something like this. It isn't quite a Gaussian in this case. But we saw something like that.

Now, what I'd like you to do is kind of start-- when I say yes, just start making some noise. Just clap, or just talk, or whatever. Just turn on your phones. Do something.

[CHUCKLING]

[TONE PLAYS]

Yeah.

[CHEERING AND APPLAUSE]

All right, let's see what-- All right, we're done. All right.

[LAUGHTER]

AUDIENCE: I had to.

[CHUCKLING]

HARI BALAKRISHNAN: I was actually going to play-- I found some nice YouTube clips of a music group, one of the MIT a cappella music groups. OK, we really got toasted here because you guys started talking, and we couldn't recover the preamble. But we've probably got--

AUDIENCE: [LAUGHTER]

HARI BALAKRISHNAN: That was the noise distribution that we saw. And you can see that it's actually kind of a [INAUDIBLE] distribution. I'll show you what this picture is. This is a picture that essentially will become your best friend, or maybe your worst enemy. This is called an eye diagram. And it looks completely messed up. So I'm going to show you what that is. And we're going to talk about it.

So let me explain to you-- but this noise was too high. And the point here is that that's shown in this picture here where we ended up with enough of a variation that we couldn't distinguish between 0's and 1's. We sent something which had a little bit of 0's in the beginning. And then the entire 1 distribution was spread between 0.1 volts to 0.9 volts. So the variance is extremely high.

Now let's do one thing. I'm going to change this to just transmit random pieces of information here. When I don't give anything, it means that the data that's being sent is just a random sequence of 0's and 1's. Let me change this to 200 bits.

[TONE PLAYS]

All right, so we got those bits through. And that's a beautiful eye diagram. I'll explain what this means. But the point is that when you see the separation, and you see a point in the middle here with a big gap on this eye diagram graph. And then you see a separation between, these were the 0's, and those were the 1's, which means you could threshold somewhere in the middle and separate out which bits were 0's and which bits were 1. It means you're cooking things a bit, OK?

So you can see that there's a distribution here of what the 1's looked like in the empirical data. There's a distribution here of what the 0's look like. This is not going to be a Gaussian at all. I'll explain why that is in a moment, OK?

So I have to do two more things today, and then I'll turn it over to Professor Verghese. The first one I want to explain to you is what this eye diagram is and why it's kind of useful. And why, as you add more noise into the system, the combination of something called intersymbol interference when noise gets in the way of decoding the bits.

Now, on this channel, there are two things that distort the quality of communication. The first is noise, and you kind of saw the effect of that. The second is this thing that we've been studying by modeling it as a linear time-invariant system. The idea is that when you have a sequence of 0's on the input, and then you go into a sequence of 1 samples, that sudden sharp input transition does not immediately get captured at the receiver. It takes time for the 0 to settle into a 1 and a 1 to settle into a 0. And you can see that in this picture here.

Now let's focus in in this picture here and look carefully at this place here. Oops. All right, let's do this again. I learned about this, like, 15 minutes ago. So bear with me.

All right, so they were 0's at the bottom, and then we bumped up to 1 on the input. The input bumped from 0 to a 1. This is after we demodulate it. So if things work perfectly, you would see at the output the 0 immediately goes to a 1. But what do you actually see? You see that it takes a while for the 1 to settle down, right? It goes from 0 to 1 sharply on the input, but it takes a while to go from 0 to 1. And then I go up like that. And then I want to go from 1 to 0. And you can see that as it comes down to 0, it takes a while to settle down.

Do you guys all understand why that happens? Like, I don't mean the physics of why it happens. But what I mean is how that idea relates to this idea of the unit step response and the unit sample response? This is nothing more than the unit step response of this channel, right? I have a 0, and I've assumed I'm on 0 for a while. I bump up to a 1, and it takes a while for it to go from 0 to 1. And similarly, it takes a while for it to go from a 1 back to a 0.

Now, suppose I end up switching 0, 1, 1, 0, 0, 1, 0, 1, 0, 1 very, very quickly. And I don't give enough time for the whole thing to settle down. In other words, as I go from a 0 to a 1, before it settles down into 1, if the next bit is a 0, and then I start coming down. And before the next bit settles down to 0, I get to a 1. And before it comes back to 0-- I keep doing that. What I'm going to end up with is this combination of 0's and 1's that are sort of random start confusing the receiver because we're not giving enough time for the 0 to settle down.

And similarly, when we go from 0 to 1, we're not giving enough time for the 1 to settle down. That's what this eye diagram was referring to where we found in this confused case-- and then when you have noise, some of the 0's get moved to 1 anyway, and 1's get moved to 0. And we end up with this very crowded picture.

The way you tackle this problem is it all has to do with the number of samples that you decide to use within 1 bit. So coming back to this picture, if I do this at 48,000 samples per second, in any one of these carriers, I have 48 samples that I use. But now I get to decide how many samples corresponds to 1 bit of information? When I have a bit that I want to transmit-- let's say a 1-- how many samples?

What that effectively means is to transmit a 1, how many of these periods of this waveform do I want to use to transmit a 1? So for example, if I decide I want to send a 1 as three carriers of the waveform, then in any one of these carriers, I have 48 samples. Therefore, I represent a 1 as 48 times 3, which is 144 samples per bit.

Now, to transmit a 0, I could do a bunch of different things. I could decide to keep the channel silent. If I do that, it's called on-off keying. And that's what we're using here. So we're sending 1 as-- if we decide to use 144 samples per bit, then it means we represent a 1 as 124 samples, which corresponds to three of these. And then we represent 0 as nothing. So we don't send anything. If we do that, this is called on-off keying because we send on for 1, where we send a sinusoid. And for sending us a 0 bit, we send nothing.

Now, if the number of samples I select per bit is too small, you end up with this effect that I don't give enough time for the 0 to settle down to-- when I make the transition from a 0 to a 1, if I pick too small samples per bit, before we settle into a 1, the next bit may show up as a 0. And then before we settle down into a 0, the next bit shows up perhaps as a 1. And we end up commingling the 0's and the 1's. And we're not able at the receiver to tell the difference between these different-volted samples after we demodulate. So we don't know what happened.

So if the eye diagram is a way to capture that, what the eye diagram does is-- it's a kind of a clever hack. What it does is-- thought I had it somewhere. There it is. The way you generate an eye diagram-- you don't have to worry about writing the software for it. But you will look at a lot of these kinds of pictures in lab 5. You transmit a random sequence of bits. And then you look at the samples that were received at the receiver after demodulation was done.

The output of the demodulation is a set of voltage values. What you do is you look at three bit periods. In other words, you look at a sequence of time, a number of samples that corresponds to three bit periods. So for example, if I pick 144 samples per bit, what I do is I take 144 and multiply that by 3 bit periods. And I look at that many samples. So for every three bit periods, I take all the samples, and I plot them, OK?

So a particular sequence of bits-- in this case, it might be a 0 and a 1 and a 0. This is a 0 followed by a 1 followed by a 0. A different sequence of bits could be a 1 followed by 1 followed by 1. A different sequence of bits could be a 0 followed by 0 followed by 0. So there are eight combinations of sequences of three bits each. So there are eight-- you have this clean eye diagram like this. You're going to see a variety of different lines corresponding to all the places where different 3-bit sequences appeared in your input.

So for any given 3-bit sequence of the input, there's a 3-bit sample sequence at the output that corresponds to a number of samples equal to the samples per bit multiplied by 3. And each of those generates one of those trajectories through this picture. If you generate that picture, and you find that there's a very clean gap between all possible combinations of these bit sequences, as is in this case here, then it means that you're very likely to be able to decode.

Because what you can do is, essentially, the receiver can decide that when 1's happen, it corresponds to something over here at 0.35 or 0.4 volts. When a 0 happens, it may not be a 0, but it may correspond to something like a 0.1 volt here, which means I can pick the middle point and slice it at that middle point to determine whether the received bits were 0 or 1.

Now, if we were to run this experiment again and make a lot of noise-- let's try that-- what would happen is we may not be able to decode at all. So I'm going to request you guys to make a little bit of noise after I start. And then we'll see how it goes. All right, start.

[WHISTLING]

[TONE PLAYS]

[APPLAUSE]

AUDIENCE: [SCREAMS]

HARI BALAKRISHNAN: You know what? It wasn't loud enough. It worked great. But let's look at what the eye diagram looks like. Well, it's a little worse than the other one, but I think the energy is down in this room. So let's do it one more time. I want to be in a position where nothing decodes.

[TONE PLAYS]

[SCREAMING AND APPLAUSE]

Well, 15% better is, in fact, my preamble decoded, which is amazing.

[INTERPOSING VOICES]

HARI BALAKRISHNAN: And that's what it looked like. OK, this is an eye diagram even a mother wouldn't like. All right, so I'm going to stop here. Before I turn it over to Professor Verghese, are there any questions? Anything at all.

I know some of you have had issues. And I met one or two of you this morning to fix your computers. And I'm happy to do that. I'll be in the lab from 4 o'clock. We'll get it working on your computers. If it doesn't work, then you're going to have to use the lab machines.

Let's take some questions or comments. Any people have any questions about this stuff? Do people get an idea of what's going on and what do you need to do? Once you get this working, the labs sort of-- they'll write themselves. You just have to do a little bit of work. Questions?

This can go up to higher frequencies too. So if the sound's annoying your ears, you can actually get it to work. I know one of you guys is trying to get this to work with ultrasonic reception. That's challenging, but we can probably help you on. Yeah.

AUDIENCE: What do we get for the uniform noise since [INAUDIBLE]?

HARI BALAKRISHNAN: Uniform noise?

AUDIENCE: Yeah. So if we just have another tone on the other side, how do we get to [INAUDIBLE]?

HARI BALAKRISHNAN: Yeah, you mean if you were to make another transmission at exactly the same frequency? The beauty of this is that there's different demodulation schemes. Right now, we're using something called the envelope demodulation, which Professor Verghese talked about before. All that's doing is it's taking the absolute value of every received sample and then running a simple averaging filter. And that's what you write in the lab. It's a very, very simple demodulation.

We'll study something called quadrature demodulation in probably next lecture or the one after that, which means that'll have the property that, if you transmit at a certain frequency, and somebody else interferes and transmits at a different carrier frequency, you're still going to be able to recover your transmission. But if the other transmitter has significant signal strength in the same frequencies that you're transmitting in, then he's going to start to look like noise to you, and that's going to not work.

So what happened here is when you guys were all whistling and were clapping and so on, there were signals generated across all frequencies, including the frequency at which I was transmitting. And that's what caused the signal to have noise. It's not like you were all transmitting at 1,000 hertz. It just so happens that that combination of noise you were making had a component at 1,000 hertz. Does that answer your question?

AUDIENCE: Yeah.

HARI BALAKRISHNAN: Any other questions, comments, remarks? OK.

GEORGE VERGHESE: Yeah, we've been talking about modeling the baseband channel. And we've said we'll focus on LTI channels. So I just wanted to take advantage of this being up here to have you think about what this tells you about how close to linear this channel is, and how close to time-invariant it is.

What do you think? What we're seeing here is, for instance, a step response to something that's at 0 and then goes up to 1 and stays at 1. We're seeing a superposition of many such 0-to-1 transitions, some 1-to-0 transitions, later 1-to-0 transitions, and so on. So we're really looking at a superposition of step responses staggered in time and going from 0 to or 1 to 0.

Do you think time invariance is maybe a good assumption for this channel? Plausible, right? Because the stuff that we get here looks a lot like the stuff we're getting here. The deviations might be noise, but the more structured parts of the waveform here match the structured part of the waveform here.

And what you're really seeing is the loudspeaker here reverberating through the room. You're all staying fixed, so the echoes are from fixed locations. The walls are fixed. And so what we're seeing is the step response of the room, in effect. If you hit the room a little bit later, well, you get the same response, but a little later.

Does this look like it's very linear? Would linearity be a good assumption for this channel? I mean this is very partial information, but does it give you enough to judge? So why do you think linearity might be good here? Were you saying, yes, it's a good assumption?

AUDIENCE: Yeah, because time is a pattern.

GEORGE VERGHESE: Because what?

AUDIENCE: Like, if you have a signal go five seconds from a different signal, it'll still appear in the center.

GEORGE VERGHESE: But that's the time-invariance argument that you're giving me. What's the linearity argument? Yeah.

AUDIENCE: When you scale the intact input, the output gets scaled accordingly.

GEORGE VERGHESE: OK, so when you scale it-- so we're not really seeing too much. Well, we are seeing scaling. What kind of scaling are you seeing here? We have 0-to-1 transitions, but we also have 1-to-0 kinds of transitions, right? What would you want to see for a linear channel for how the 1-to-0 transition behaves relative to the 0-to-1 transition? What would you expect on the linear channel? Yeah.

AUDIENCE: [INAUDIBLE]

GEORGE VERGHESE: So from superposition, you would really expect to see the same shape on the downslope that you see on the upslope, right? So when you look at the upward transition here, you see a certain shape. Well, it's not quite matched on the downward slope here. And the reason is that the particular simple demodulation that we're using here for this on-off scheme actually makes that not look very linear for signals that are for channels that have this kind of an overshoot to them.

But the other scheme that Hari mentioned, the quadrature demodulation, will actually do a lot better. We, in fact, saw that, right? We saw that there was modulation that essentially pulled out the absolute value of what you were sending. And then there was another modulation scheme that actually pulled out the signal itself. And so if you're doing things like taking absolute values somewhere in the middle there, then you're going to start losing the ability to model it as linear.

OK, so linear time-invariant models are not necessarily good for all channels. It's something that you've got to look for. There are good reasons to try and structure a channel so that it's close to linear and time-invariant because then you can do a lot of analysis and design for it.

OK, so I want to continue talking about our models for LTI channels. And still in the time domain, next time, we'll start to look at this in the frequency domain. Hari mentioned frequency several times here. So we're talking about an LTI channel, Linear and Time-Invariant. We talked about characterizing it by its unit sample response.

And so let's see. Unit sample response means put in a unit sample function. You get out an output that you're going to call the unit sample response. Put in an input x(n). That is a summation of such things. Let's say x(k) delta of n minus k summed over all k, right? That's the general input represented as a weighted combination of unit samples.

Well, what comes out in that case? What is y of n? If we're talking about a linear time-invariant system, then it's the same weighted combination of the responses to these unit samples. So we're going to get summation x(k) h of n minus k, right? So this is the convolution expression that we talked about last time.

So what I want to do in the rest of the lecture is give you some other ways to think about this. Our notation for this was y of n equals x convolved with h evaluated at time n, right?

Another way to think about this-- let's see. Let me actually first do an example and then give you another way to think about this. Suppose I have a system whose effect is to multiply the input by A, some number A, and delay by some number D.

And I tell you that this is LTI. Actually, I don't have to tell you that it's LTI. You can prove that it's LTI. If I tell you have a system whose only action on the input is to delay the input by capital D and scale the input by capital A, you can actually prove that it satisfies time-invariance and that you can superimpose, OK? So this is LTI.

So what's the unit sample response? If I put in the unit sample function of the input, what's the output?

AUDIENCE: A delta n.

GEORGE VERGHESE: Anyone?

AUDIENCE: A delta n minus D.

GEORGE VERGHESE: Yeah, A delta of n minus d, right? And if I put some general input function in here, what's the output?

AUDIENCE: [INAUDIBLE]

GEORGE VERGHESE: Without telling you about convolution. Somebody-- I heard a voice here. Yeah.

AUDIENCE: A of s times [INAUDIBLE] D.

GEORGE VERGHESE: Right. We didn't have to do any convolution to figure this out, right, because I described the system to you in a simple way. You could tell me what it does to the output.

All right, so I want to give you another way to think about a system where the unit sample response of h events. I'm talking about a system with the unit sample response h of n. So what does that mean? That means that at time 0, I evaluate 0. When I put out and put in a unit sample at time 1, I get some h1. At time 2, get some h2, and so on. This is a hn, OK?

So if I give you a system and tell you that the unit sample response is this function, h, Here's a way to think of what it is. Inside there, here's what my system can be thought of as doing. Inside here, I've got many parallel paths. I've got a system that scales by h0, delays by 0, and in parallel with the system that scales by-- let me put it up here-- scales by h1, delay by 1, and so on.

OK, so all of these in parallel-- each one is very simple. Each one is as simple as the example I showed you. OK, I've got a whole bunch of these parallel systems, each one as simple as this. If I put a unit sample in here, what comes out? It's going to be exactly that, right, because the unit sample would get scaled by 0, delayed by nothing, will come out there. Then unit sample will get through this path scaled by h1, delayed by 1, come out there. When you assemble all of these with this summer here, you're going to get exactly that response.

So here's another way to think about what's sitting inside a system, an LTI system whose unit sample response is given to be that, OK? So if I put x and n, what is it that's going to come out? If I put x and n, what comes out? Well, through this path, I get x(n) scaled by 0 and delayed by nothing. Through this path, I get x(n) scaled by h1 and delayed by 1, and so on.

So what comes out? y of n is equal to h(m) x of n minus m over all m, OK? So I take x(n). I shift it by nothing. I scale by h0. That's one of these terms. The term corresponding to m equals 0. I take x(n), scale it by h1, delay it by 1. That gives me the term with m equals 1. So here's another way to write it, OK?

Last time, I said that actually you can write convolution in this form or with the operation reversed. So it actually doesn't matter which order you write things in. This would be something you might write as h convolved with x at time n. So two different ways of writing the convolution and two different ways of thinking about how the output gets represented that way.

You can easily get from one representation to the other by just making a change of variables. Like, let n minus k equals m, and you get from this representation to the other. This is a more mechanistic way of thinking about why these two representations work.

OK, so with that is a given, let me show you how to actually carry out these operations graphically. In either form, here is a simple graphical way to think about what's going on. So to determine y at time n, this is the operation that I have to carry out, all right? To define y at time n, the output at time n, I've got to find a way to implement this operation.

So how are we going to think of this graphically? I want to sketch the signal x. I want to sketch the signal h and then do something with these two signals to construct this, OK? So when I plot x, and I plot h to implement this operation, what's the name on my time axis? Is it m that I'm going to stick here or k or what? I want to implement this. I want to draw these two time functions, the functions of k, right? And it's just the number that I'm specifying here.

OK, so on the k-axis, I'm going to take x and plot it. So x is some time function. Here's my x. I won't label them all because that would get crowded. And just to keep things clean, let's change colors here.

How am I going to plot h of n minus k? Well, let's start thinking about the n equals 0 case. So for n equals 0, I've got to plot h of minus k. So how does h of minus k relate to the unit sample response h of k or h of n? If I tell you that I have a system-- we had an example up here, didn't we? I lost it.

If I tell you that I have a system with this unit sample response-- let's do something simple. Let's say that this is 1/2 to the n times u of n. So for positive time, it's kind of a decaying geometric series. And for negative time, it's 0, OK? So that's an example of a unit sample response of a system.

So if that's h of n, what does h of minus k look like? Yeah, anyone?

AUDIENCE: Reflection.

GEORGE VERGHESE: Sorry?

AUDIENCE: Reflection [INAUDIBLE].

GEORGE VERGHESE: It's the reflection, OK? So if I want for n equals 0 to plot just the h of minus k that I need here, it's that reversed and plotted. OK, so this is an h of minus k here.

What about h of minus k? Sorry, what about h of n minus k? Suppose I had n equals 3 now. How do I get h of 3 minus k? So I want to get h of 3 minus k. So that corresponds to sliding this over. Do I slide it to the right by 3, to the left by 3?

You can tell by looking at the argument here. Whatever used to happen at 0 has now got to happen k equals 3. So that means a rightward shift. OK, so you flip this over, and then you slide it by n steps, OK? So you take the h of k, flip it around to get h of minus k, slide it by n steps. So if n is positive, you're sliding it to the right. If n is negative, you're sliding it to the left.

So now you're on a single figure. You've managed to plot these two. What's the remaining operation? What you've got to do is the point-by-point product of these two waveforms and sum over the entire time axis. It's like taking a dot product, right? So you're going to take this value of the red curve-- well, actually, let me slide it over.

Let's do this case. This is the slid-over case, right? You're going to take every one of the purple values multiplied by the white and sum over the entire time axis. That's an implementation of this. So in recitation tomorrow, you'll get practice on this.

But what you want to think of, sort of the mantra for graphical implementation of convolution is you've got to do the flip of one of the time functions, slide by n, and then the product, OK? So you slide it to get one particular value of n. If you want the next value of n, you slide it 1 over and go through this whole thing. So it's flipping, sliding by the right number of spots, doing the inner product. That gives you one value of the answer. And then you repeat. All right, you'll get more practice in recitation.