slide1:
Hello everyone, in this lecture, we'll continue our foundational story and focus on understanding the discrete Fourier transform, or DFT, and its inverse, the IDFT. These slides and my book draft will be your main reference, and they provide a clear, step-by-step path for following the key steps and get the main idea.
The content you see here is part of a larger foundation in medical imaging. Once you have a solid grasp of these concepts, it will be much easier to understand advanced topics like X-ray tomography, nuclear imaging, and MRI.
Fourier analysis is central to all of these areas. We'll use it to analyze and process signals in both continuous and discrete forms, and to move back and forth between the time or spatial domain and the frequency domain, in either a continuous or discrete form.
In practice, we digitize signals and then perform operations such as convolution, Fourier transforms, and inverse Fourier transforms in one dimension, two dimensions, and beyond. This lecture will begin building the tools you need for that process.

slide2:
We're moving along right on track in our journey through this course.
If you've had the chance to preview the reading materials for today, that's great — it will help you connect the concepts more quickly. If not, that's fine too.
Just follow along closely, and make time afterward to review the main ideas. Building this habit of reinforcing concepts as we go will make your understanding stronger and more intuitive, especially as the topics become more mathematical.

slide3:
In our last lecture, we examined the sampling theorem in detail, focusing on key concepts such as the sampling rate and the Nyquist rate. By now, you should have a clear idea of why, under certain conditions, we can perfectly recover a continuous signal from its sampled version.
Here's the key idea:#We start with a continuous signal, f of t, shown here. When we sample it at regular time intervals — let's call that interval delta-t — we obtain a discrete sequence of values. The sampling theorem tells us that, if certain conditions are met, we can reconstruct the original continuous signal from these samples without losing any information.
That's remarkable — because even though we only record values at specific time points, and there's "missing" information in between, the theorem guarantees perfect recovery. But this only works if the original signal's Fourier transform is band-limited. That means its frequency spectrum is essentially zero outside a certain range, from minus w to plus w.
In the frequency domain, sampling corresponds to a convolution between the original continuous Fourier spectrum and a periodic train of delta functions. This delta train in frequency comes from the Fourier transform of the impulse train in time. The spacing between the deltas in frequency — which we'll call P — is the reciprocal of the sampling interval delta-t.
To avoid any distortion, P must be at least 2 W. This ensures there is no overlap between the repeated copies of the spectrum in the frequency domain. You can think of each delta function as a "copying machine" — each one shifts and reproduces the original spectrum at regular intervals. If the copies don't overlap, the original information is preserved perfectly.
In practice, we can apply a low-pass filter to keep only the central copy of the spectrum, then perform an inverse Fourier transform to reconstruct the exact original signal.
Now, a small note about the special case where P equals exactly 2 W: here, the copies just touch at a single point. This doesn't cause problems unless that single point is a delta function with a finite value — in which case it could contribute to the result. But in most practical signals, that single point has no significant effect.

slide4:
Now let's look at the special case where P equals exactly 2 w.

Suppose our continuous signal is a pure sine wave with a frequency of 1 hertz. That means it completes exactly one cycle in one second. In the frequency domain, this is a band-limited signal — its maximum frequency is 1 hertz.
If we sample at exactly twice that frequency — so at 2 hertz — we meet the Nyquist rate exactly. In this case, we take two samples within each cycle. For a pure sinusoid, two unknowns fully describe it: the amplitude and the phase. In theory, two samples per cycle should be enough to determine those two unknowns.
However, here's the problem. Depending on where the samples fall, we might end up with the exact same sample value each time — for example, always sampling at the zero crossings of the sine wave. In that case, all our samples would be zero, and we would have no reliable way to determine the original signal.
In the Fourier domain, a pure sine wave corresponds to a pair of delta functions located at its positive and negative frequency components. When we sample at exactly 2 w, the duplicated spectra in the frequency domain just touch each other. If the touching points happen to be delta functions, those deltas overlap, and we can no longer separate the contributions from each one.
Mathematically, it's like having the equation X plus Y equals C — you know the sum, but not the individual values. This is an aliasing problem.
That's why, in practice, sampling exactly at 2 w is risky. To be safe, we require PPP — the spacing between repeated spectra in frequency — to be slightly greater than 2 w. In other words, the sampling rate should be a little higher than twice the maximum frequency in the signal. This extra margin ensures that no overlap occurs, and reconstruction remains reliable.

slide5:
Here, we have the whole derivation of the sampling theorem summarized on one page.
You see a few steps underlined in red. These are the important turning points in the math — the places where something key happens. In a detailed lecture, we would walk through each of them slowly, but here, I want to focus on the big picture.
We start in the frequency domain, where the spectrum of our signal is multiplied by a periodic train of impulses. When we bring this back to the time domain, that multiplication turns into a convolution. The result of that convolution involves a special function called "sinc," which naturally appears when you take the Fourier transform of a rectangular shape in frequency.
Next, when we write out the convolution, we see it as an infinite sum — a series of shifted sinc functions. Each one is scaled by the value of our signal at a sample point. The distance between these samples is delta-t, and that's simply one over P, the spacing in the frequency domain.
The final line is the famous Shannon interpolation formula. In words, it says: take each sample of your signal, put a sinc function centered at that sample, scale it by the sample's value, and then add them all together. Do this for every sample — stretching infinitely in both directions — and you get your original continuous signal back exactly, as long as the sampling conditions are satisfied.
This is a beautiful result. It tells us that perfect reconstruction is not just possible — it's a direct consequence of the sampling theorem. And this formula is one of the core foundations of all digital signal processing.

slide6:
Let's look at a good case — when our sampling rate is high enough to meet the sampling theorem's requirement.
The black curve here is our original, continuous signal. Think of it as the "true" signal. The blue points show where we took our samples — a limited number of values, spaced evenly in time.
Here's the magic: if the sampling rate is greater than twice the maximum frequency in the signal — the Nyquist rate — then we can take these discrete samples and use the Shannon interpolation formula to reconstruct the original signal perfectly.
When we do that here, the reconstructed signal (shown in red) lies exactly on top of the original black curve. They match so perfectly that you can't even see the difference.
This is the sampling theorem in action. For band-limited signals, as long as we

sample fast enough, we can go back and forth between continuous and discrete forms without losing information. You can repeat this process over and over, with different sample sets, and it will work every time.
In short — with the right sampling rate, theory and practice agree perfectly.

slide7:
Now, here's a bad case — when the sampling rate is too low.
The black curve is again our original, continuous signal. The blue points are where we took our samples. And the red curve is what we get when we try to reconstruct the signal from those samples.
At the sample locations, the reconstructed red curve passes exactly through the blue points — so it matches the original at those specific spots. But between the sample points, the red curve drifts far away from the true black signal.
This mismatch is caused by aliasing. Because our sampling rate is less than twice the maximum frequency in the signal, the repeated spectra in the frequency domain overlap. That overlap distorts the information, and once it happens, no amount of interpolation can recover the original shape.
So, while the reconstruction looks fine exactly at the sampled points, the continuous waveform in between is completely wrong. This is why respecting the sampling theorem is essential — if you don't sample fast enough, the damage is permanent.

slide8:
Let's step back and see the big picture.
There are really two parts here. On the top half, we have discretized the signal in the time domain. This means we start with a continuous signal, sample it at regular intervals, and get a discrete set of values — something we can store and process in a digital computer. If the sampling theorem is satisfied, this step causes no information loss.
But the story doesn't end there. Even though our signal is now discrete in time, its Fourier spectrum is still continuous. And a continuous spectrum cannot be stored directly in a computer either.
That's where the second part comes in — discretizing the spectrum. This is the central topic of this lecture. The idea is to take the continuous Fourier spectrum and multiply it by a periodic train of delta functions in the frequency domain. This multiplication in frequency corresponds to convolution in the time domain, which effectively copies our time-domain signal over and over again, spaced apart by a fixed period t.
In other words, sampling in one domain causes repetition — or duplication — in the other domain. When we discretize both the time-domain signal and the frequency-domain spectrum, we end up with a periodic structure in both domains.
The key point here is that now, both our signal in time and its spectrum in frequency are discrete and periodic. This makes them fully compatible with digital processing. And this is the foundation for moving toward the discrete Fourier transform.

slide9:
Signal sampling is a fundamental concept, but it can be confusing if you only look at it quickly. So let's slow down and really understand it.
At the top, we have our continuous signal, f x. This is defined for every value of x and could represent anything — a sound wave, an image profile, or a measured signal.
Next, we have what's called a Shah function, or an impulse train. This is simply a series of delta functions, each spaced t units apart. Mathematically, it's written as the sum of delta functions, where each one is located at x equals n T, with n being an integer. You can think of this impulse train as a ruler that marks the exact points where we'll take our samples.
When we multiply the continuous signal f of x by this impulse train, we get the sampled function. This multiplication is done point-by-point: at each delta function's location, we keep the value of f of x; everywhere else, the result is zero.
So the sampled signal is really the original signal's values, "attached" to the impulses in the train. The impulses mark the positions, and the heights of the impulses carry the signal's values at those positions.

In other words, the sampled function is just the original function, but only at discrete, regularly spaced points. This is the first step in moving from the continuous world to the digital world.

slide10:
When we sample a signal in the time domain — or the spatial domain — it creates a very specific effect in the frequency domain. Sampling in one domain is equivalent to performing a convolution in the other domain.
Let’s walk through it.#We start with our original spectrum, which we’ll call F of u. Here, u represents frequency, and u-max is the highest frequency present in the signal — its bandwidth.
When we multiply our signal in time by an impulse train — a set of regularly spaced spikes — that multiplication turns into a convolution in frequency. The result is that the original spectrum gets repeated again and again, spaced apart by the sampling frequency.
In this diagram, the middle shape is the original spectrum. To the left and right, you can see identical copies. The spacing between them is one over capital T, where capital T is the sampling interval in time.
If our sampling frequency — which is one over T — is at least twice the maximum frequency in the signal, then these repeated copies don’t touch each other. In that case, we can simply keep the central copy and reconstruct the original signal perfectly.
But if we sample too slowly, the copies overlap. This overlap distorts the spectrum, and once that happens, there’s no way to separate the original from the distortion. That’s the aliasing problem we saw in the earlier “bad case” example.

slide11:
Here’s the Nyquist theorem in action.
In the diagram, you can see the original spectrum and its repeated copies created by sampling. If our sampling rate is too low, these copies start to overlap. When that happens, the frequency components from one copy mix with those from another. This is aliasing — and once it occurs, we can’t tell which part of the spectrum came from the original signal and which part came from the overlap.
To avoid aliasing, we must make sure that the highest frequency in the signal — we call it u-max — is less than one divided by twice the sampling interval. In spoken terms, the sampling frequency must be greater than twice u-max. This threshold is called the Nyquist frequency.
When the condition is satisfied, we can use a rectangular filter — often called a gate function — to select just the central copy of the spectrum. That single copy contains all the information needed to perfectly reconstruct the original signal.
It’s like DNA testing — you don’t need the whole organism, just a small sample, because it contains the complete blueprint. In the same way, one clean copy of the spectrum contains everything we need.
But if the sampling rate is too low and overlap happens, no filter can separate the mixed components. The information is lost forever.

slide12:
Now let’s see why aliasing makes recovery non-unique.
In this diagram, the red and blue curves represent two different sets of frequency components. When aliasing happens, these components overlap in the frequency domain. At the points where they overlap, what we measure is just the sum of their values.
For example, imagine that at a certain frequency, the blue component has a value of A, and the red component also has some value. When they overlap, all we see is their sum. That means many different combinations of blue and red could produce the exact same total.
In other words, you can’t tell exactly how much came from the blue curve and how much came from the red one. The information about their contributions is lost.
This is why aliasing is such a problem — once two spectra overlap, there is no unique way to separate them. No matter how you process the data, you can’t reconstruct the original spectrum with certainty.

That’s why the Nyquist theorem’s condition — sampling faster than twice the highest frequency — is so important. It prevents this kind of overlap, keeping the spectrum unique and recoverable.

slide13:
Up to this point, we’ve been talking about aliasing and how to avoid it. Let’s now assume we’ve done that — our sampling rate is always high enough, and the spectra are well separated. Aliasing is no longer an issue.
On the top row, you see our original continuous-time signal on the left. On the right, it’s been sampled — we now have a series of discrete values at regular time intervals.
On the bottom row, you see what this means in the frequency domain. On the left is the original spectrum, limited to a maximum frequency, u-max. On the right is the sampled spectrum — multiple, non-overlapping copies, spaced apart by the sampling frequency, which is one over capital T.
So yes, the signal is now discrete in time, and its spectrum is nicely replicated without overlap. But our job isn’t finished yet. The spectrum is still continuous in frequency. And if we want to fully digitize the signal — so that both the time domain and the frequency domain are discrete — we still have another step to go.
That step is discretizing the spectrum, which is exactly where we’re heading next.

slide14:
Now that our signal is discrete in time and aliasing is avoided, the next step is to discretize the Fourier spectrum.
On the right, we start with the continuous, periodic Fourier spectrum of our sampled signal. The peaks are separated because our time-domain sampling rate was high enough. That’s good — no overlap here.
Our goal is to make the frequency domain discrete as well. To do that, we multiply the spectrum by a periodic train of impulses — shown here in green. This is just like what we did before in the time domain, but now we’re applying it in the frequency domain.
Multiplication in the frequency domain corresponds to convolution in the time domain. So, when we apply this green impulse train to the spectrum, the effect in the time domain is that our discrete signal gets duplicated at regular intervals, spaced by capital T.
Here, delta-u — the spacing between impulses in the frequency domain — equals one over capital T, where capital T is the period of each repeated signal in the time domain.
As before, we must choose our frequency-domain sampling rate carefully. The impulses in frequency must be close enough together — in other words, our spacing delta-u must be small enough — to avoid overlaps in the time domain. This is the same logic as before, just in the other domain.
By doing this, we end up with a signal that is discrete in both the time domain and the frequency domain. This is the essential step that brings us to the discrete Fourier transform, or DFT.

slide15:
Let’s restate the big idea in another way.
We start with a continuous function in the time domain, which we’ll call f of t. It has a continuous Fourier transform, capital F of u, in the frequency domain. These two are perfectly related — one can be transformed into the other with no loss of information.
Now, if we sample f of t at the Nyquist rate or higher, we get a discrete version of the signal — here shown as g of t. And because our sampling rate satisfies the theorem, there’s no loss of information in moving from the continuous signal to its discrete version.
That discrete-time signal, g of t, also has its own Fourier transform — here represented as capital G of u. And again, as long as we have sampled properly, this discrete spectrum contains the same information as the continuous spectrum, capital F of u.
The key point is that the Fourier transform is an invertible process. That means moving from f of t to capital F of u, or from g of t to capital G of u, can be

done in either direction without losing information — as long as the sampling theorem is respected.
So whether we choose to work in the continuous domain or in the discrete domain, we can perform equivalent signal analysis. The difference is that in the real world, our computers are digital. That means we do everything in the discrete domain — but thanks to the theory, we can be confident that our results match what we would get in the continuous world.

slide16:
Let’s now bring everything together in one big picture, but with a bit more detail.
We start with a continuous function, f of t, and its continuous Fourier spectrum, capital F of u. Our goal is to sample both the time domain and the frequency domain in a way that makes sense for digital processing.
First, let’s ask: how many samples will we have in the time domain?
This depends on two things:
The sampling interval in time, delta-t. This is equal to one over capital P, where capital P is the spacing in the frequency domain. The smaller the delta-t, the more data points we collect.
The total duration of the signal, capital T. The longer we record, the more samples we have.
The total number of samples in the time domain — call it N — is simply the total duration T divided by delta-t. Since delta-t is one over P.
We can write:#N equals T multiplied by P.
Now let’s switch to the frequency domain.
Here, the total span of one period in frequency is capital P. The sampling interval in frequency, delta-u, is equal to one over capital T. So the total number of samples in frequency over one period — call it M — is the total span P divided by delta-u. That’s again P multiplied by T.
This symmetry is important:#The number of data points in the time domain and the number of data points in the frequency domain are the same — both equal to P multiplied by T.
So whether we’re sampling in time or in frequency, the total number of samples is determined by the product of the total span in one domain and the sampling density in the other.
This is the balanced relationship that underlies the discrete Fourier transform, and it’s why we can move between time and frequency representations so efficiently in digital signal processing.

slide17:
Now let’s highlight the key variables and the important relationship between them.
In the time domain, the total number of samples is T times P. Here, capital T is the total duration of the signal, and capital P is the spacing in the frequency domain. So we have N equal to T multiplied by P.
In the frequency domain, we have the exact same number of samples, also T multiplied by P. We call this number M, and here P is the total span of one period in frequency, while T controls the sampling density in frequency.
If we take the reciprocal of N, we get a very useful relationship:#1 over N equals delta-t times delta-u,#where delta-t is the sampling step size in time, and delta-u is the sampling step size in frequency.
This is an expression of the duality between time and frequency. If we fix the total number of samples N, making delta-t smaller — meaning we sample more finely in time — will make delta-u larger, meaning the frequency samples are spaced farther apart. And the reverse is also true.
In simple terms: if you zoom in more on one domain, you automatically zoom out in the other. This balance is built into the mathematics of the Fourier transform and is something we’ll use later when we look at applications.

slide18:
Now we’re ready to move from the continuous Fourier transform to the discrete Fourier transform.
When we sample a continuous signal, it stops being a smooth curve that exists at every moment in time. Instead, it becomes the product of the original signal and

a series of equally spaced spikes — what we call delta functions. These spikes occur at time intervals of delta-t, which means each spike is separated by one divided by capital P in time. In other words, we only keep the signal's value at these regular time points, and everywhere else is zero.
We can describe this mathematically as a sum of delta spikes. Each spike occurs at a time equal to n divided by P, where n goes from zero up to N minus one. The height of each spike is simply the value of the original signal at that time. You can imagine it as a row of vertical lines, each carrying the amplitude of the signal at its sampling point.
This gives us the discrete version of the signal in the time domain. Even though it's just a set of separate values, it still represents a function of time, so we can take its Fourier transform.
When we do that, the Fourier transform turns into a sum over all the sampled points. The oscillating factor — what we call the kernel — still has the form "e to the power of minus j times two-pi times the frequency u times n divided by P," but now we only evaluate it at those sampled time points.
The outcome is what we call the direct Fourier transform of the sampled signal. This is how we connect the continuous theory to the practical, digital form of the discrete Fourier transform that we use in computation.

slide19:
The discrete Fourier transform we just discussed is actually a periodic function. Let's understand why.
Think back to the Fourier series. In the time domain, a Fourier series is built from sinusoidal components — each one a sine or cosine wave. And by definition, each of these components is periodic. Whether we have just one frequency or many, the result will still be periodic.
Now, in this expression at the top, the left-hand side shows the Fourier series in the time domain, with terms involving t over capital T. On the right-hand side, we have an almost identical formula — but here the variable t has been replaced by u, and T has been replaced by P. This tells us that in the frequency domain, we also get a periodic function.
The periodicity comes directly from the fact that the Fourier transform of a sampled signal is made up of these repeating sinusoidal components. Each one repeats over and over, giving us a continuous function in the frequency domain that is also periodic.
This is exactly what we saw in the big picture earlier: when we sample in one domain, the Fourier transform becomes periodic in the other domain.

slide20:
Up to now, we've mostly talked about sampling in the time domain. But we can also sample directly in the Fourier domain.
Let's start with the Fourier transform of our signal, f-hat of u. If we multiply this spectrum by a train of delta functions in the frequency domain — just like we did earlier in the time domain — we get a sampled version of the spectrum. That means we only keep its values at certain frequency points, which we can label as u-zero, u-one, all the way up to u (N minus 1).
Here, the spacing between these frequency samples is 1 over capital T, where capital T is the period in the time domain. The index m runs from zero to N minus one, just like it did in the time-domain case. And the total number of samples in the frequency domain is the same as in the time domain — both equal to capital N.
Mathematically, we can write the sampled spectrum at position u m as a sum over all the time-domain samples f of n over P, multiplied by a complex exponential. The exponent has a nice symmetry: it's e to the power of minus j, 2 pi, n times m, divided by capital N.
This symmetry is important — it shows that sampling in the Fourier domain has exactly the same mathematical structure as sampling in the time domain. The formulas look almost identical, just with the roles of time and frequency swapped.
This parallel is one of the reasons the discrete Fourier transform is so elegant — it's the same process, whether you're thinking in terms of time samples or frequency samples.

slide21:
Now we can state the discrete Fourier transform
We've sampled the signal at times#t n equals to n divided by capital P, for n equals to 0, 1, …, N minus 1.#We'll read the spectrum at frequencies#u m equals to m divided by capital T, for m equals to 0, 1, …, N minus 1,#with N equals to T times P.
The DFT value at u m is
"sum from n equals zero to N minus one, f of n over P, into e to the power of minus j, two pi, m n over N."
This is just an inner product with a complex sinusoid of frequency index m.#Each output is complex: its magnitude tells us how much of that frequency is present, and its angle (the phase) tells us the shift.
A few quick anchors:
t n equals to n over P are the time samples.
u m equals to m over T are the frequency samples.
The exponential "e to the minus j two pi m n over N" is the rotating basis wave.
Next, we'll write the inverse DFT and see why a factor of one over N naturally appears so we can reconstruct the time samples exactly.

slide22:
Up to now, we've been writing our time samples as t-zero, t-one, t-two, and so on, and our frequency samples as u-zero, u-one, u-two, and so on. But when we work with sampled data, what really matters are the integer positions of these samples.
So, to make things simpler, we'll label them using integers in square brackets.
For example:
f of zero means the value of the function at time sample t-zero.
f of one means the value at time sample t-one.
f of N minus one means the value at the last time sample, t N minus one.
We do the same in the frequency domain:
f-hat of zero means the Fourier transform value at frequency sample u-zero.
f-hat of one is at u-one.
And so on, up to f-hat of N minus one.
With this notation, the discrete Fourier transform reads like this:
f-hat of m equals the sum from n equals zero to N minus one of f of n, multiplied by e to the power of minus j, two pi, m times n divided by N.
Here, n is the index for time-domain samples, and m is the index for frequency components.
Each term is just the sample value f of n multiplied by a sinusoidal basis wave at frequency index m. Adding all those terms together gives the strength of that frequency in the signal — that's our Fourier coefficient.
We can also see this as a matrix multiplication:
The time samples f of zero, f of one, …, f of N minus one form a column vector.
They are multiplied by an N-by-N matrix whose entries are these complex exponentials e to the power of minus j, two pi, m times n divided by N.
The result is another vector containing all the frequency samples f-hat of zero, f-hat of one, …, f-hat of N minus one.
For the first frequency index, m equals zero, every exponential equals one, so the first Fourier coefficient is simply the sum of all the time samples.
Later, we'll talk about scaling factors to account for the sampling step size, but for now this integer-index form keeps the DFT definition clean and easy to use.

slide23:
Once we have all the discrete Fourier coefficients — that is, f-hat of m for m equals zero up to N minus one — we can recover the original sampled signal values f of n by using the inverse discrete Fourier transform.
In words, the formula says:
f of n equals one over capital N, times the sum from m equals zero to N minus one of f-hat of m, multiplied by e to the power of plus j, two pi, m times n divided by N.
Here, n is the time-sample index, and m is the frequency-sample index.
The plus sign in the exponent is important — it's the opposite of the minus sign we used in the forward discrete Fourier transform. This sign change is what

allows us to reverse the process.
The factor of one over capital N also plays a key role. When we computed f-hat of zero in the forward transform, we were summing all the time samples, but we didn't average them. This factor in the inverse transform takes care of that averaging, so we get the correct values back.
Some people prefer a symmetric version: instead of putting all the scaling in the inverse transform, they split it evenly — using one over the square root of N in both the forward and inverse formulas. This is purely a matter of definition; the math works either way.
You can also see this in matrix form:
In the forward transform, we multiply the time-sample vector by the Fourier matrix of complex exponentials.
In the inverse transform, we multiply the frequency-sample vector by the inverse of that matrix, which has the plus sign in the exponent and the one over N factor out front.
With these two formulas — forward and inverse — we can move back and forth between a discretized time-domain signal and its discretized frequency-domain spectrum, with no information loss, as long as the sampling theorem was satisfied in the first place.

slide24:
Let's talk about why we have this factor of one over capital N in the inverse discrete Fourier transform.
Earlier, I mentioned that 1 over N is equal to delta-t times delta-u. Here, delta-t is the sampling interval in the time domain, and delta-u is the sampling interval in the frequency domain.
Think of it this way: when we move from the continuous Fourier transform to the discrete version, we're replacing continuous integrals with sums. In the time domain, we can picture our original continuous function as being approximated by a collection of rectangles — each rectangle has a height equal to the sample value f of n , and a width of delta-t.
Likewise, in the frequency domain, the Fourier spectrum is also represented as a collection of rectangles — each one has a height equal to f-hat of m and a width of delta-u.
When we go from the spectrum back to the signal, we're essentially adding up all of these little rectangular pieces. The factor of delta-t times delta-u captures the combined effect of this discretization in both domains. And since 1 over N equals delta-t times delta-u, that's why the factor appears in the inverse DFT.
This is the discrete equivalent of what happens in the continuous Fourier transform, where we also have a scaling factor in the inverse formula. The difference is that here, the scaling is tied directly to our sampling steps in both domains.

slide25:
The factor of 1 over capital N comes directly from how the continuous Fourier transform becomes a discrete one when we sample.
In the continuous world, both the forward and inverse Fourier transforms are integrals. But when we digitize the process, we replace those integrals with sums. That's what we see in this figure — the smooth area under the curve is replaced by a set of thin rectangular strips. Each strip has a height equal to the function value at a sample point, and a width equal to the sampling interval.
In the time domain, that width is delta-t, the spacing between time samples. In the frequency domain, it's delta-u, the spacing between frequency samples.
When we perform a forward Fourier transform, the discretization introduces a factor of delta-t. When we perform the inverse transform, the discretization introduces a factor of delta-u. Multiply those together, and you get delta-t times delta-u — which we know is equal to one over capital N.
This is why, in the inverse discrete Fourier transform, we include the 1 over N factor — it accounts for both steps of discretization: first in time, then in frequency.
And remember, there's one more difference between the forward and inverse formulas: the forward transform uses a minus sign in the exponent, and the inverse transform uses a plus sign. This sign change is what lets the two

operations perfectly undo each other.
So, the 1 over N is there because of how sampling replaces integrals with sums in both domains. It’s the scaling factor that ensures the forward and inverse transforms work together exactly.

slide26:
The second way to look at the discrete Fourier transform is from the perspective of harmonics.
When we do a forward DFT, we can think of it as multiplying our vector of time samples by a square matrix of complex exponentials. This matrix acts like a rotation in a high-dimensional space — it changes our coordinates from the time-domain representation to the frequency-domain representation.
The set of Fourier basis functions — the sinusoids at different discrete frequencies — form an orthonormal basis. That means they are all perpendicular to each other in this mathematical space. The forward transform rotates our coordinates from the “time axis” basis to the “frequency axis” basis. The inverse transform simply rotates them back.
This idea generalizes. In one dimension, the Fourier basis is just sines and cosines. But in two or three dimensions, or on a sphere, we can define more complex sinusoidal patterns, called harmonics. On a sphere, these are the spherical harmonics — each one corresponding to a certain “frequency” pattern over the surface.
The key property is the same: if you take two different harmonics and compute their inner product, the result is zero — they are orthogonal. If you take one harmonic and do an inner product with itself, you get one — it’s normalized.
So whether we are talking about simple sines and cosines, or more complex spherical harmonics, the Fourier transform is still just expressing a signal in terms of an orthogonal set of basis functions, then rotating between these bases.

slide27:
Here we’re looking at a particular orthonormal basis used in Fourier analysis.
Our basis functions have the form:
e to the power of j, two pi, m times t divided by capital T
where m is the frequency index.
If we take two of these basis functions, one with index m and the other with index n, and compute their inner product — meaning we integrate the product of one and the complex conjugate of the other over the full interval from zero to capital T — two things can happen:
If m and n are different, the integral is zero. That means the functions are orthogonal — they share no common component.
If m and n are the same, the integral is one. That means each basis function has a unit norm, or length equal to one.
This is exactly what “orthonormal” means: the functions are orthogonal to each other, and each one has length one.
Because the Fourier basis has this orthonormal property, we can represent any signal as a sum of these basis functions without losing information. And we can go back and forth between the time-domain representation and the frequency-domain representation using the discrete Fourier transform and its inverse, knowing that the relationship is exact when the sampling theorem is satisfied.

slide28:
What we see here is just another way to write the discrete Fourier transform and its inverse — same math, just different symbols. You’ll often see different papers or books use different notations, so it’s important to recognize that they all mean the same thing.
Let’s say we have N data points in the time domain. We’ll call them h k, where k runs from zero to N minus one. These are our measured samples — they could represent anything, such as temperature values taken at different times.
To get the frequency-domain representation, we compute H n, the Fourier coefficients, using the formula:
H n equals the sum from k equals zero to N minus one of h k times e to the power of minus j, two pi, k n divided by N.
This is the forward discrete Fourier transform. The minus sign in the exponent

tells us we’re rotating our coordinates in one direction in this N-dimensional space.
The inverse transform takes us back from the frequency coefficients H n to the time samples h k:
h k equals one over N, times the sum from n equals zero to N minus one of H n times e to the power of plus j, two pi, k n divided by N.
Here, the plus sign in the exponent means we’re rotating back, and the factor of one over N is the scaling we discussed earlier — it accounts for the sampling steps in both time and frequency. Some definitions split the scaling evenly between the forward and inverse transforms to make them look perfectly symmetric, but that’s just a matter of convention.
The key points:
We have N samples, so we only need N orthogonal basis functions.
Those basis functions are harmonics whose frequencies differ by a constant increment.
The forward and inverse transforms are nearly symmetric — the main difference is the sign in the exponent and where we put the scaling factor.
From a computational point of view, each row of the Fourier transform matrix has N elements. To compute one Fourier coefficient, we do N multiplications and N additions. Since we have N coefficients to compute, the total work is proportional to N-squared.
This N-squared growth in computations was a big deal in the early days of signal processing, when N could be very large. That’s why the development of the Fast Fourier Transform, or FFT, was such a breakthrough — it reduced this cost dramatically.

slide29:
The Fast Fourier Transform, or FFT, is an efficient algorithm for computing the discrete Fourier transform. It was published by Cooley and Tukey in 1965, and it completely changed the way we do signal processing.
To give you an idea of its impact, in 1969, analyzing 2,048 data points from a seismic trace using the standard DFT took more than 13 hours. Using the FFT on the exact same machine, the same analysis took just 2.4 seconds. That’s an enormous improvement.
I remember when I was in primary school, a 2,000-point seismic data analysis could take more than a full day to process. But with FFT, the same task could be finished in under three seconds.
This kind of speedup is critical in many applications, especially those that need real-time results — whether it’s analyzing signals from an airplane’s sensors, monitoring seismic activity, or performing real-time image reconstruction.
The reason FFT is so powerful is that it reduces the computational complexity. A naive DFT requires about N-squared operations — that’s N multiplications and additions for each of the N outputs.
The FFT reduces this to N times log-base-2 of N operations.
That’s a huge difference. For example, if N is 1,000, the log-base-2 of N is about 10. So instead of doing a million operations, we only need about ten thousand. The larger N gets, the more dramatic the savings become.
Because of this efficiency, FFT still plays an essential role in modern signal processing, and it’s also widely used in areas like medical imaging, machine learning, and convolution-based computations.

slide30:
Because the inverse discrete Fourier transform is so similar to the forward transform, we can design a fast algorithm for it as well. This is called the inverse FFT, or IFFT.
In MATLAB, the fft function computes the forward FFT, and the ifft function computes the inverse FFT. For example, fft of X, N computes an N-point FFT. If the vector X has fewer than N points, it pads with zeros; if it has more, it truncates. The IFFT works the same way, just in reverse.
If we were to implement the Fourier transform exactly as in the mathematical definition, we would do a summation for each frequency index k. Each summation is essentially an inner product — it requires N multiplications and N additions. And we need to do this for each of the N output points. That’s N times N

operations — N squared in total.
The FFT and IFFT are like built-in shortcuts. You don't need to know all the details of how they work to use them — just like you don't need to know how your phone connects to your friend when you press the call button. You can treat FFT and IFFT as black boxes: you give them the data, and they give you the result quickly and efficiently.
With FFT and IFFT, Fourier analysis becomes practical for real-time work. Once we can efficiently move between the time domain and the frequency domain, we can use this power for many applications — performing convolution, estimating spectra, removing noise, detecting patterns or contours in images, and much more.

slide31:
Let's look at our first example: how to compute convolution using the Fast Fourier Transform, or FFT.
You already know the direct method — the one we practiced in class. You take one sequence, flip it, shift it, multiply corresponding terms, and sum them up. Then you repeat for each shift. This gives the convolution directly in the time domain.
If the first sequence has length n and the second has length m, the convolution result has n plus m minus 1 points. That's how many shifts we need to perform.
And in terms of computation, this direct method takes time proportional to n squares, where n is roughly the size of the data.
But there's a much faster, indirect method. Instead of doing the convolution directly in the time domain, we switch to the frequency domain using the FFT.
Here's the MATLAB approach:
First, compute the FFT of x — that gives us its frequency spectrum.
Then, compute the FFT of y — the spectrum of the filter or second sequence.
In the frequency domain, convolution turns into multiplication. So we simply multiply the two spectra, element by element.
Finally, apply the inverse FFT to that product to return to the time domain.
This works because of the convolution theorem — it says convolution in the time domain is equivalent to multiplication in the frequency domain.
If you compare the results from the direct method and the FFT-based method in MATLAB, you'll see they match exactly. The key difference is efficiency:
Direct convolution: about N square operations.
FFT-based convolution: about n log n operations — much faster for large n.
One small detail: in discrete Fourier analysis, both the time domain and the frequency domain are treated as periodic. That means this FFT-based convolution actually performs circular convolution unless you handle padding carefully.

slide32:
Let's look at a second example, and here's where things get interesting.
We again compute convolution in two ways:
First, the direct method in the time domain.
Second, the FFT-based method, multiplying in the frequency domain and taking the inverse FFT.
This time, the results are different. Why?
It comes down to the fact that in discrete Fourier analysis, our signals are treated as periodic, not single, isolated sequences, but infinitely repeated copies.
Imagine x as one segment in time, and y as another. In the continuous world, when we shift one relative to the other, we only overlap the part that exists in the original copy. But in the discrete Fourier model, when you shift, parts of one copy can wrap around and overlap with the start of the other copy.
It's as if your "neighbor" signal is spilling into your yard — and at the same time, you're spilling into your other neighbor's yard on the left.
That wrap-around interaction is exactly what circular convolution means:
The signals are treated as if arranged around a circle.
When you slide one past the end, it reappears at the other side.
The FFT-based convolution performs this circular convolution by default, which is why the result here doesn't match the direct, linear convolution.
Later, we'll see how to avoid this wrap-around effect so the FFT method matches the true linear convolution.

slide33:
We just saw that FFT-based convolution gives us circular convolution by default, where the “neighboring” copies wrap around and interfere.
The simplest way to avoid that wrap-around is to make sure those neighbors are far enough away so they can’t overlap during the shift. That’s what zero padding does.
We take our original signals and append enough zeros to both so that the total length is at least N plus M minus 1, where N and M are the original lengths of the two sequences.
In this example, each signal is only 5 samples long, but we pad them out to length 16. Now, when we perform the FFT, multiply in the frequency domain, and take the inverse FFT, the “wrap-around” region is all zeros — so the result matches the direct, linear convolution exactly.
In other words, zero padding is like making your yard so wide that your neighbors are too far away to cause any trouble.

slide34:
This picture shows the idea of zero padding in a visual way.
On top, the red triangles represent one signal, and the blue triangles represent the other. Without padding, these shapes are repeated periodically, so when you slide them for convolution, parts from the “neighboring copies” wrap around and interfere — that’s the circular convolution effect we saw earlier.
Now, by adding enough zeros, we artificially extend the period. This pushes the neighboring copies farther apart, so when we perform the convolution, only the single red copy and the single blue copy overlap. The neighbors never get close enough to contribute.
The result is that our circular convolution now matches the linear convolution exactly — because there’s no wrap-around. The earlier mismatch happened simply because we didn’t have enough zeros, so the neighbors still overlapped.
This is the essence of zero padding: make the “yard” wide enough so your neighbors can’t mess up your calculation.

slide35:
If you’d like to explore these ideas further, MathWorks has some excellent tutorials and examples in its documentation.
This particular page walks through the relationship between linear and circular convolution in MATLAB. It explains the key conditions under which the two are equivalent, and shows how zero padding can be used to make circular convolution behave like linear convolution.
You’ll also find clear MATLAB examples here — creating vectors, padding them to the right length, applying the FFT, multiplying in the frequency domain, and using the inverse FFT to get back to the time domain.
So, if you want to reinforce what we’ve just covered or see it implemented step-by-step, this is a great place to start.

slide36:
Our second example is spectral analysis.
Here, we use an important relationship: one divided by capital N is equal to delta t multiplied by delta nu.#In words, delta t is the sampling interval in the time domain, and delta nu — sometimes called delta u — is the spacing between frequency samples in the Fourier domain.
What this tells us is that if we keep delta t fixed and the total number of samples N fixed, then our frequency spacing delta nu is also fixed. That means, if a signal contains two frequencies that are very close together, the spectrum may not have enough resolution to separate them.
One way to improve the resolution is called zero padding. This means we add zeros to the end of the time-domain signal, which increases the total number of samples N. Increasing N makes the spacing in the frequency domain, delta nu, smaller — giving us a finer frequency grid and better spectral resolution.
In this MATLAB example, we create a signal by adding two sinusoids.#The first term is the cosine of two times pi times one hundred times t.#The second term is the sine of two times pi times two-hundred two point five times t.
We sample t from zero to one second in steps of zero point zero zero one

seconds, which corresponds to a sampling frequency of one kilohertz.#Without zero padding, the frequency step size may be too large to clearly separate the peaks at one hundred hertz and one-hundred two point five hertz.#With zero padding, the peaks become distinct.
If you want the full MATLAB walkthrough, you can follow the link at the bottom of the slide.

slide37:
Now, let’s see what happens when we perform the Fourier transform without using zero padding.
We take our signal x and compute its discrete Fourier transform using the FFT function.#We keep only the first half of the spectrum, because for a real-valued signal, the second half is just the mirror image.#Then we normalize by dividing by the length of x, and scale the amplitudes appropriately.#The frequency axis is calculated from zero to half the sampling rate, in steps of the frequency resolution.
On the plot, you see two peaks in blue. The first peak is at exactly one hundred hertz — that one is well captured because it matches the frequency grid.#The second component, however, is at one hundred two point five hertz, which does not align neatly with the frequency samples.
As a result, the peak is lower than it should be — you can see that it only reaches about half of its true amplitude — and it spreads into nearby frequency bins.
This illustrates the limitation of the frequency resolution when we don’t use zero padding.

slide38:
Now, let’s see what happens when we use zero padding.
We take our original signal and extend it in the time domain by adding a large number of zeros at the end.#This does not change the actual information in the signal, but it does increase the total number of samples.
When we compute the Fourier transform of this longer sequence, the frequency spacing between the points in the spectrum becomes smaller.
This means we get a finer frequency grid — more points between zero and the Nyquist frequency.
As a result, when we plot the spectrum, both peaks — the one at one hundred hertz and the one at one hundred two point five hertz — are now represented much more accurately.
You can see that their amplitudes are correctly estimated, and the peaks are sharper.
So, zero padding is a simple but powerful trick for improving the visual resolution of your spectrum.#It doesn’t actually add new information, but it lets you see the frequency components more clearly.

slide39:
Before we wrap up, I want to highlight something that often feels like magic in signal processing.
A signal cannot be both time-limited and band-limited at the same time.
If a signal is strictly limited in time — meaning it exists only within a certain interval — its frequency spectrum will stretch infinitely.#And if a signal is strictly limited in frequency — meaning it contains only a finite range of frequencies — then it must extend infinitely in time.
In practice, we often work with signals that are approximately time-limited and approximately band-limited.#This approximation, along with clever techniques like sampling, zero padding, and windowing, allows us to do what might seem impossible — turning the “impossible” into something that works perfectly well for our needs.#That’s the kind of “magic” you see every day in signal processing

slide40:
Now let’s step back and look at the big picture.#Everything we’ve discussed so far seems to work perfectly — but there’s an important approximation hidden here.
We often say: if a signal is finite in time — meaning it has a limited duration — then its Fourier spectrum is also limited to a finite interval.

In reality, that’s not strictly true.#If a signal is perfectly finite in the time domain, its spectrum will actually extend infinitely. It will get smaller and smaller as you move away from the center, but it never truly stops.
You might argue that if we take the frequency range from minus W to plus W, and make W large enough, the parts outside this range become so small that they can be ignored.#And yes, in practical engineering, we often do that. But from a mathematical point of view, we have to be careful.
Why?#Because when we discretize, the spectrum is periodically repeated — we get infinitely many copies of it.#Even if a single copy has very small values far away from the center, those small values from every copy can add up.#The further away a copy is, the less it contributes — but since there are infinitely many of them, the sum of all those tiny contributions might still be significant.
This is why convergence is not always guaranteed and why mathematical rigor is important when making these approximations.

slide41:
Let me give you a simple example to show why “getting smaller” isn’t always enough.
Imagine a curve where the height at any point is equal to one divided by the position along the horizontal axis. As you move farther to the right, the values get smaller and smaller — for example, at position one the value is one, at position two it’s one-half, at position three it’s one-third, at position four it’s one-fourth, and so on.
You might think: “These numbers are tiny — if we keep adding them, the total should stay small.”
But here’s the surprising fact: even if you start far out, like at one over one million, and then keep adding one over two million, one over three million, and so on, the total will still grow without limit.
In other words, an infinite number of tiny contributions can still add up to something infinitely large.
And this is exactly the kind of situation we face in Fourier analysis: very small contributions from infinitely many frequency copies can still combine into something significant.

slide42:
This idea connects to the famous wheat and chessboard problem, which is a classic example of exponential growth.
Imagine a chessboard with sixty-four squares. On the first square, you place one grain of wheat. On the next square, you double it — two grains. On the third square, you double it again — four grains. Then eight, sixteen, thirty-two, and so on. By the time you fill all the squares, the total number of grains becomes unimaginably large — enough to cover the Earth with a thick layer of wheat.
This illustrates how quickly exponential growth can get out of control.
Now, in our Fourier problem, for our approximation trick to work, the leftover terms — those tiny contributions from far-away frequencies — must shrink faster than one divided by the position number. That means the function must be smooth enough so that its spectrum decays quickly.
In fact, for most practical functions, after some pre-processing and smoothing, the decay is even faster — often like one divided by the position number squared. And in that case, adding up all the small contributions still gives you a small total. This is what allows our approach to work.

slide43:
So, at this point, we have completed the main part of our discussion on Fourier analysis. In preparing these lectures, I’ve drawn from multiple resources — including this excellent textbook from Stanford University’s course “The Fourier Transform and Its Applications” — as well as my own insights and examples.
This textbook is designed for a full semester, and it’s quite comprehensive. My goal here has been to distill the most essential concepts for our biomedical engineering context. I’ve also added examples and explanations from other sources, as well as my own understanding, to create what I like to think of as a “BME version” of linear systems and Fourier analysis.
So far, you have all the core material. In the next lecture, our teaching assistant will go over some homework problems to help reinforce your

understanding and sharpen your skills. After that, we’ll move on to our next major topic — network image quality.
That means the most mathematically challenging part of the course — Fourier analysis — is now behind us.