An Intuitive Guide to the Discrete Fourier Transform for Music

Andrew Davis

August 12, 2020

The Discrete Fourier Transform is an integral part of digital signal applications for music. Understanding the inner workings of the Discrete Fourier Transform will lead to a more complete understanding of filter design, spectral processing, pitch shifting, time stretching, and a host of other digital audio effects. The purpose of this document is to approach the Discrete Fourier Transform from an intuitive perspective and to consider it solely in the context of music and audio. One should understand, however, that the Discrete Fourier Transform has applications in many domains and is one of the great tools we have in solving complex scientific and mathematical problems.

Introduction

Intended Audience

This document is intended for musicians and upper-level undergraduates interested in learning more about the Discrete Fourier Transform and digital signal processing. The goal is to present the DFT, a difficult and complicated topic, in an intuitive format that shows its use for audio applications. Where possible I will try and eschew cumbersome math. However, an understanding of math is essential for even a basic understanding of the Discrete Fourier Transform. Appendix A is a helpful guide for reviewing important mathematical topics. I would suggest reviewing that document before proceeding with the main guide. What math is required? Readers should have an understanding of trigonometry and trigonometric identities, but topics like calculus are unnecessary. I suspect that many reading this document already have an understanding of core audio principles like sampling, sampling rate, aliasing, etc. Many of these topics can be found in an introductory course on digital audio so I will assume the reader has familiarity with those topics.

Why do we need the Discrete Fourier Transform?

Before we start examining the Discrete Fourier Transform (abbreviated DFT for the remainder of this document), we must first motivate why we even need it in the first place. Sound can be expressed in two domains: the time domain and the frequency domain. The former is the more natural way to think about sound. We perceive sound as an evolution over time. Songs have verses and choruses and those formal parts happen at specific locations. We simply cannot absorb the entirety of a song in one swoop as we could with a painting. The time domain representation of sound mimics the way we perceive it. It expresses sound as a fluctuation of amplitude over time where amplitude represents the change in pressure that are detected by our ears. In the digital world, the amplitude is represented as a sequence of samples. Each sample contains a measurement of the sound's amplitude, a value typically between -1 and 1.

The frequency domain is perhaps a more surprising way to think about sound, though equally valid. The frequency domain expresses sound as a set of frequencies. It turns out that any sound can be decomposed into a sum of sinusoids having fixed amplitude, frequency and phase. A sinusoid is simply any wave of the form

$$A\sin(2\pi ft + \phi)$$

where A is the amplitude, f is the frequency, and ϕ is the phase.[1] Mathematically then, any sound can be expressed as a sum of the following form:

[1] Note that a cosine wave is also a sinusoid as it is equivalent to a sine wave with a phase shift.

$$A_1 \sin(2\pi f_1 t + \phi_1) + A_2 \sin(2\pi f_2 t + \phi_2) + A_3 \sin(2\pi f_3 t + \phi_3) + ...$$

This was an amazing discovery in the mid-nineteenth century and is attributed to Joseph Fourier after whom the Discrete Fourier Transform is named. What this discovery implies is that you can recreate any sound (i.e., a song, field recording, speech) by simply summing together a series of sine waves, assuming you have the right amplitudes, frequencies, and phases. Wild! The frequency domain, then, is the collection of sinusoids that constitute a sound. It expresses all the frequencies found in some sound and their respective amplitudes and phases. I sincerely hope that you are questioning how it is possible that *any* sound can be constructed from a set of sine waves. It is by no means obvious. Sadly, we will not prove this truth. The math is beyond the scope of this document. We will simply have to take it as gospel and use it to our advantage as we progress through our understanding of the DFT.

We now have two equally valid ways to think about sound. One is to consider sound as a change in amplitude over time. The other is to consider the various sinusoids that make up that sound. It may be surpising at first to recognize that the frequency domain makes no consideration of time. It feels strange that a series of unchanging sine waves added together can really express the evolution of sound over time that we experience when we listen to music, but it is indeed true.

But how do we actually figure out the sinusoids that make up a given sound? The answer: the Discrete Fourier Transform! The DFT is a tool to convert a sound from the time domain signal to the frequency domain.. Once in the frequency domain we can do various spectral manipulations to change the sinusoidal components of the original sound and convert it back to the time domain using the Inverse Discrete Fourier Transform (abbreviated as IDFT). Together the DFT and the IDFT are the two tools we need to convert between the time domain and frequency domain of any sound.

Getting Our Terminology Right

You may have heard of several different "flavors" of tools with the name Fourier in them: Fourier Transform, Fourier Series, Discrete Fourier Transform, Discrete-Time Fourier Transform, Short-Time Fourier Transform, ... etc. It's difficult to keep all of these different concepts straight. Each one though is a tool to convert a time domain signal to its frequency domain. The differences lie in whether time is continuous or discrete and whether the transformation produces a continuous or discrete frequency domain.

We perceive sound as a continuous stream over some duration. It is continuous because at any given instant we could measure the air pressure processed by our ears. Recall that we represent air pressure using ampltiude. Therefore, we can say every instant in time is associated with a particular amplitude. This is what is meant by continuous time. So exactly how many instants of time are there in a given piece of music? The answer: infinite! Unfortunately, it is impossible for digital systems and computers to record the amplitude for every instant in time. We would need an infinite number of amplitudes for our infinite moments in time to represent every recorded sound properly. Computers and digital systems can only store a finite amount of data. Sampling is a compromise for this limitation. We can get the overall picture of a sound by measuring the amplitude at a small enough period that the original sound can be rendered almost exactly the same as the original. For example, we might sample 30 seconds of a song by taking a million samples. Even though a million numbers is quite large, our modern machines can comfortably store that amount of data. Sampling generates a finite amount of data, and any finite set of data can be stored on a computer as long as its not too large.

Similarly, a sound in the frequency domain can be made of an infinite number of sinusoids. If we want to store the frequency domain of a sound in a computer, we will only be able to store a subset of those sinusoids. It is simply the limitations of the technology we have available to us. Akin to sampling amplitude, we want to sample the sinusoids from the frequency domain at a high enough resolution that we can get an overall sense of its continuous frequency domain.

So why do we care about the Discrete Fourier Transform in particular? Because it converts a finite discrete signal (i.e., a finite number of samples) into a finite discrete frequency domain (i.e., a finite number of sinusoids with particular phases and amplitudes). Some of these other tools like the Fourier transform for example take a signal of continuous time and convert it to a continuous and infinite frequency spectrum. Others like the Discrete-Time Fourier Transform work with infinitely sampled signals and convert them to a continuous frequency spectrum. The Discrete Fourier Transform is the practical option for us as musicians because we can process a finite number of samples from a song say and get an approximate rendering of its sinusoidal components using a computer. Courses in digital signal processing will spend much more time dealing with these other transforms.

One final point. If you spend enough time studying electronic music or audio technology, you will invariably come across the Fast Fourier Transform. The Fast Fourier Transform (abbreviated as FFT) is identical to the Discrete Fourier Transform except that it is... faster! The FFT is a way to calculate the computations needed for the DFT more quickly . It was one of the most important discoveries in technology in the twentieth century and revolutionized our ability to process signals in a timely fashion. Most software and audio programming languages reference the FFT. Just know that there is no difference between the FFT and the DFT other than computational speed.

Detecting Sinusoids

Before we attempt to transform a time domain signal to the frequency domain, we will need to solve a simpler problem. Let us first try and develop a tool to detect whether a particular frequency or sinusoid is present in some sequence of audio samples. If we can do that, we can test our samples against a whole range of frequencies, allowing us to construct a good sketch of the frequency domain of the sound. In essence, this is what the DFT does, and the tool we use to detect frequencies is called the "inner product."

Orthogonality and the Inner Product

Orthogonality is an important mathematical concept that will help us detect the presence of sinusoids in sound. Orthogonality has several definitions depending upon the application. In geometry, we say two lines are orthogonal if they form a 90 degree angle (i.e., perpendicular). We can also say two vectors are orthogonal if their inner product equals zero. We can think of a vector as a sequence of numbers. For example, $v = \langle 1, 2, 3 \rangle$ is the vector v with the sequence 1, 2, and 3. The inner product of two vectors is the sum of their pointwise products. For example, if we have $v = \langle 1, 2, 3 \rangle$ and $u = \langle 1, -1, 0 \rangle$, then the inner product of these two vectors is $(1 * 1) + (2 * -1) + (3 * 0) = -1$. In this example, we take the first numbers of each vector and multiply them. Then we multiply the second numbers and then the third numbers and add them all up. In this example, that sum is -1.

The term"vector" is generally used in the context of mathematics. In digital signal processing, we use the term "sequence". A sequence is just like a vector. For example, the sequence $x[n] = [1, 2, 3]$ is equivalent to the vector v from before. Just like vectors, we can take the inner product of two sequences. For example, the inner product of the sequences $x[n] = [1, 2, 3]$ and $y[n] = [1, -1, 0]$ also yields -1. Are $x[n]$ and $y[n]$ orthogonal? No, because their inner product is not zero. For a time-domain sequence, the numbers represent amplitudes. So a sequence like $x[n] = [0.5, 0.2, 0]$ is a sound file with three samples of amplitudes 0.5, 0.2, and 0.

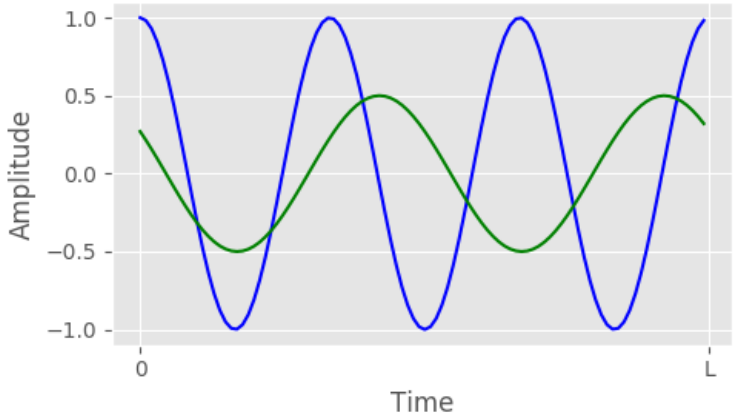
We can also take the inner product of two functions as well. Figure 1 shows two different sinusoids. The green sine wave can be expressed as .... and the blue sine wave can be expressed as .... As with sequences, we take the inner product of two functions by summing their pointwise products.

Why do we care about orthogonality? It turns out that orthogonality and sinusoids have an important mathematical relationship. If we take any two sinusoids and take their inner product, the result is zero (i.e., orthogonal) **except when the two sinusoids have the same frequency**. This simple property is the key to understanding how the DFT works and for determing whether a particular signal contains some frequency. Note that there are some caveats to this claim which we will discuss soon.

Consider the two sinusoids in Figure 1, which will serve as a good example for illustrating orthogonality. To take the inner product of these two sinusoids over some interval from 0 to some farther point in time L, one must sum the pointwise products of the amplitudes for every instant in time between 0 and L. The first pointwise product would be at $t = 0$ where t represents time. At that moment, the green sinusoid has an amplitude of about 0.25 and the blue sinusoid has an amplitude of 1.0. The product of these two values would be $1.0 * 0.25 = 0.25$. That constitutes the first pointwise product. To calculate the sum of all the other pointwise products, we would need to do this same procedure for every value of t between 0 and L. That's an infinite number of points! Fortunately, if we can express these sinusoids in the form $A\sin(2\pi ft + \phi)$, then it becomes relatively trivial with calculus and we can show that the result is indeed 0!

Any two sinusoids of different frequencies are orthogonal no matter their phase or amplitude. A key caveat though is that the duration over which we take the inner product must usually be infinite in order to make this claim. In music though, we never deal with infinitely long sine waves. Songs are made up of finite length sine waves. We can still have orthogonality for any two sine waves over some finite interval from 0 to L, but we must provide the extra condition that the waves are **periodic** along that interval.

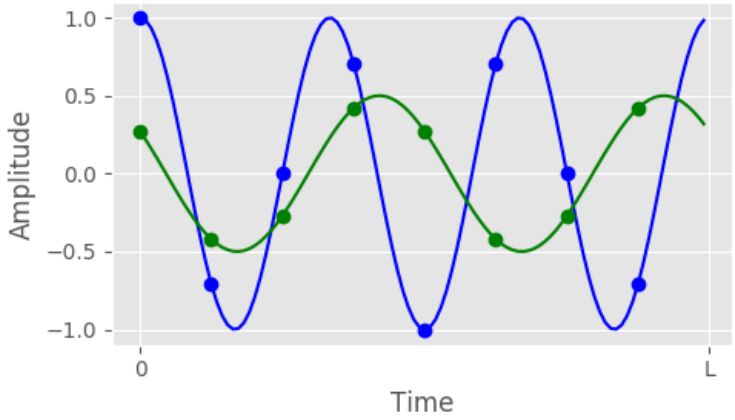
Figure 1: Two sinusoids that are periodic along the interval 0 to L.



This is a crucial point and one that will be paramount for understanding the limitations of the Discrete Fourier Transform. If we look back at Figure 1, we will see that both the blue and the green sinusoids start and end at the same place in their respective curve. If this were not true, we could **not** make the claim that these two sine waves were orthogonal along that interval.

Two periodic sinusoids are also orthogonal even when we sample them. This is important because we will never deal with periodic continuous sinusoids in a computer like those found in Figure 1. We will be dealing with sampled signals and sinusoids. To illustrate and provide a little proof on concept, let's sample the two sinusoids from Figure 1. For ease, let's assume that $L = 1$ so we will sample for one second from both the green and blue sinusoids. If you look at Figure 1, we can see that the green sinusoid completes two complete cycles over a period of 1 second. Therefore, it has a frequency of 2Hz. The blue sinusoid completes three full cycles so it has a frequency of 3Hz. We need to choose a sampling rate that will not cause aliasing for these two signals. Anything above 6Hz will suffice, so let's choose 8 Hz. Figure 2 shows where those samples lie on our two sinusoids. The samples for the blue sinusoid are approximately $b[n] = [1, -0.707, 0, 0.707, -1, 0.707, 0, -0.707]$ and the samples for the green sinusoid are $g[n] = [0.27, -0.421, -0.27, 0.421, 0.27, -0.421, -0.27, 0.421]$. The inner product of $b[n]$ and $g[n]$ is equal to (1 * 0.27) + (-0.707 * -0.421) + (0 * -0.27) + (0.707 * 0.421) + (-1 * 0.27) + (0.707 * -0.421) + (0 * -0.27) + (-0.707 * 0.421) = 0. Indeed these two sinusoids are orthogonal even when sampled.

Figure 2: Two sinusoids that are periodic along the interval 0 to L.



Let's briefly summarize the conditions when two sinusoids of different frequencies are orthogonal:

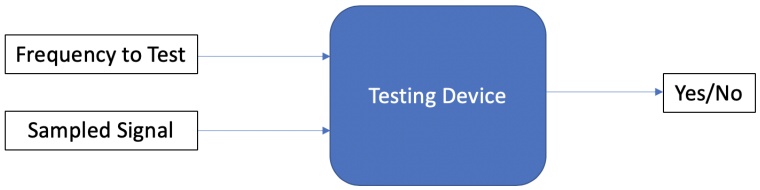
- the two sinusoids are periodic along some time interval
- if sampled, the sample rate is greater than twice the frequency of both sinusoids

If you are curious more about how we can prove orthogonality and some of the issues with orthogonality for continuous and discrete sinusoids, refer to Appendix A.

Probing for Sinusoids

Ideally we would like some tool to be able to test whether a particular frequency is part of some arbitrary signal. We should be able to provide this tool some signal like a song or speech and some frequency and our tool should be able to report back to use that either, yes, this frequency is part of the signal or, no, the frequency is not part of the signal. Figure 3 graphically depicts this procedure. Given such a tool, we could then iteratively test our signal for a whole range of frequencies to construct the frequency domain of our signal. This is exactly the procedure that the DFT performs. It tests a signal for a whole range of frequencies to determine which are part of the signal and which are not.

Figure 3: A sketch of the desired procedure for determining frequency in a signal.



Orthogonality will be our testing tool. Let's imagine that our sampled signal was a simple sinusoid of arbitrary frequency, phase, and amplitude. Ahead of time, we do not know anything about the sinusoid other than it is periodic along some time interval L. Let's choose a frequency to test, say 100Hz, that we know is also periodic along L. If we take the inner product between the samples of the unknown sinusoid and the samples of 100Hz, the result can provide some clues as to whether they are the same signal. What do we know if the result is non-zero? We can definitively conclude that they are the same frequency because all periodic sinusoids of different frequencies are orthogonal. What do we know if the result is zero? The easy conclusion is to say that they must be different frequencies. But that would be a mistake! We have not said anything yet about the inner product of two sinusoids of the **same** frequency. As it turns out, the inner product will almost always be non-zero unless the two sinusoids are out of phase by $\pi/2$. So an inner product of zero almost always means that the two sinusoids are of different frequency but not definitively.

The inner product then is almost the perfect tool to help us distinguish between two periodic frequencies. Unfortunately, the pesky phase issue of $\pi/2$ for two sinusoids of the same frequency throws a wrench into the system. In the next section, we will look at a way to correct this problem. But for now, let's assume that the inner product can perfectly distinguish between the frequencies of two sinusoids.

We have shown how orthogonality can work as our testing tool for Figure 3 when our signal is a sinusoid. But rarely would our signal ever be just a simple sinusoid. How can we apply the inner product to an arbitrary signal like a song? Recall that any song, speech, soundscape, or any other kind of audio can be broken down into a sum of sinusoids.

$$A_1 \sin(2\pi f_1 t + \phi_1) + A_2 \sin(2\pi f_2 t + \phi_2) + A_3 \sin(2\pi f_3 t + \phi_3) + ...$$

It turns out that the inner product is distributive. So the inner product of some arbitrary signal applies to each sinusoidal element that composes the signal. The term "distributive" in mathematics applies to many different operations. For example, we know from alebgra that the distributive property of multiplication over addition means that $x * (y + z) = x * y + x * z$. We see that the multiplication operator can apply to each individual component and the result is just the same. If we have an arbitrary signal x and another arbitrary signal composed of components y and z, then the distributive property for inner products means that $\langle x, y + z\rangle = \langle x, y\rangle + \langle x, z\rangle$. We will not spend any time proving this truth here. But it is actually not too difficult to do so. The inner product is simply a series of multiplications and additions. So with some clever rearranging of terms, we can show that the distributive property is true for inner products.

This is a wonderful result because we know now that when the inner product of some test frequency is applied to some arbitrary signal composed of sinusoids, the result is equivalent to the inner product of our test frequency with each sinusoidal component. We just analyzed the meaning of the inner product

of two sinusoids. Anything that is non-zero means that two sinusoids are the same frequency. Anything zero is **almost** always means the two sinusoids are of different frequency. Therefore, a non-zero result of the inner product of some test frequency with an arbitrary signal means that the frequency is a part of that signal. A result of zero means that the frequency is likely not a part of the signal. So the inner product is our tool to implement Figure 3.

Figure 4: A realization of the desired procedure for determining frequency in a signal.

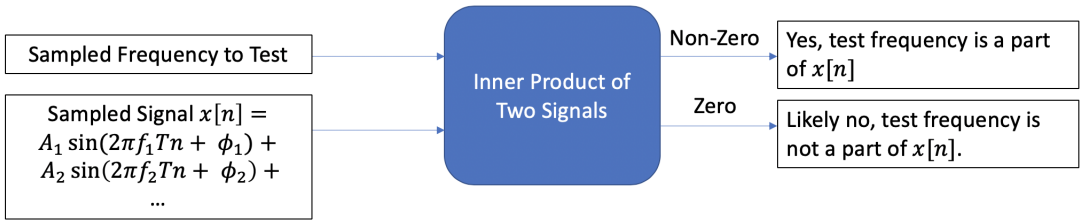


Figure 4 summarizes our methodology for implementing Figure 3. The inner product is a tool that can distinguish whether two sinusoids are the same frequency with near certainty. The next section will show a way to make it foolproof. Because the inner product is distributive, when it is applied to an arbitrary signal and some test sinusoid, the inner product of the test with each sinusoidal component of different frequency will be zero. If any of the frequency components match our test frequency, we will likely get a non-zero result indicating that the test frequency is present in our signal. Again, our test mechanism depicted in Figure 4 only works under the condition that the test frequency and the sampled signal are periodic along the same interval.

The Pesky $\pi/2$ Problem

We know that the inner product of two sinusoids at different frequencies is always zero, assuming that the sinusoids are periodic over some interval L. However, that is not the **only** way two sinusoids are orthogonal. It turns out that the inner product of two sinusoids of the **same** frequency is zero when the sinusoids are separated by a phase of $\pi/2$. This is a problem if we want to use the inner product to test the similarity of two frequencies. We do not want the inner product to be zero when the frequencies are the same. That should be reserved for different frequencies.

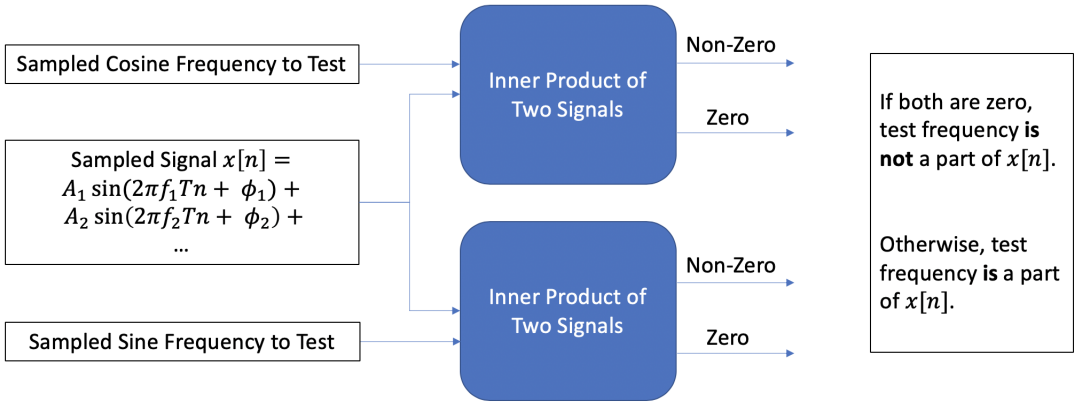
How can we ensure that we get a non-zero inner product for sinusoids of the same frequency? The answer: take **two** inner products. Suppose we have some test frequency and some signal and we would like to know whether that test frequency is a part of the signal just as described in Figure 3. If we take the inner product between the test frequency and the signal, we run the risk that the answer could be zero even if the test frequency is a part of the signal. But if we have two test sinusoids of the same frequency but with different phases, then at least one of them will yield a non-zero inner product with the signal if the signal contains the frequency of the test sinusoid.

It may not be readibly apparent why using two sinusoids makes any difference. For simplicity, let us say the two sinusoids are sine and cosine. Sine and cosine are separated by a phase difference of $\pi/2$. Can you think of another sinusoid of the same frequency with a phase different of $\pi/2$ for **both** of them? The answer is no. A sinusoid that $\pi/2$ away from cosine is either a sine wave or π radians away from sine. In either case, that sinusoid would have a non-zero inner product with sine. Therefore, using two inner products, one with sine and one with cosine, creates a wonderful system where a zero inner product from both cosine and sine means the frequency is conclusively not a part of the signal.

Figure 5 shows our completed implementation of the testing mechanism outlined from Figure 3. Note how we now need two sampled sinusoids: one cosine and one sine. But now see how an inner product of zero for both the sine test and cosine test definitively concludes that the frequency is not a part of the signal.

If you are curious about how we can prove that $\pi/2$ is an issue, see Appendix B which walks through the math.

Figure 5: Complete solution for testing frequency components of a signal.



Imaginary Numbers

Figure 5 shows how we need to take the inner product of our signal with both a cosine and sine wave to test whether a frequency is present in some signal. Therefore, we will have two results, one for each inner product. It's a little cumbersome in math to express two separate results from some operation. Generally, if we compute something like $\sin(\pi/2)$, it yields a single numerical result.

One way to do this would be to use vectors. As we discussed above, vectors store a sequence of data and we could put the result from the first inner product in the first slot of the vector and the second inner product in the second slot. This would work perfectly well. But there actually is a better solution: complex numbers. Perhaps you have encountered complex numbers somewhere in your mathematical training. Complex numbers come in the form $a + bi$ where i is called the imaginary unit or number and is defined as $i^2 = -1$. Already, this feels perilous on some level. Why are we using complex numbers for audio signals? There is nothing "imaginary" about an audio signal, sine wave, or cosine wave. So why use them?

In general, complex numbers are important mathematical tools for solving problems in mathematics, engineering and the physical sciences. This includes signals. In fact, complex numbers are just as "real" as say negative numbers. It's hard to find a physical analogy to negative numbers. We cannot pick up -4 rocks, for example. Yet, we accept negative numbers because they allow us to work more easily with operators like subtraction. Complex numbers are similar. They are a mathematical tool that allows us to solve problems we would not otherwise be able to solve. For our purposes, they serve a similar role to vectors. When we have a number like $a + bi$, we separate the real component a from the imaginary component b. You cannot add a and b. They are treated as separate and distinct units. We could very well use a complex number to express the number of apples and oranges. Perhaps the real component represents the number of apples and the imaginary component represents the number of oranges. Then, to express that I have 3 apples and 4 oranges, I could write that as a single number $3+4i$. So one way to view a complex number is simply as a single number that has two distinct parts. This is what we need. We can use the real component to hold the result of one inner product and the complex component to hold the other.

Complex numbers have one final important property that make them the ideal choice for our problem. The great mathematician, Leonhard Euler, discovered one of the fundamental mathematical equations that relates numbers such as i, e and sin and cos. It is called Euler's formula:

$$e^{ix} = \cos(x) + i\sin(x) \tag{1}$$

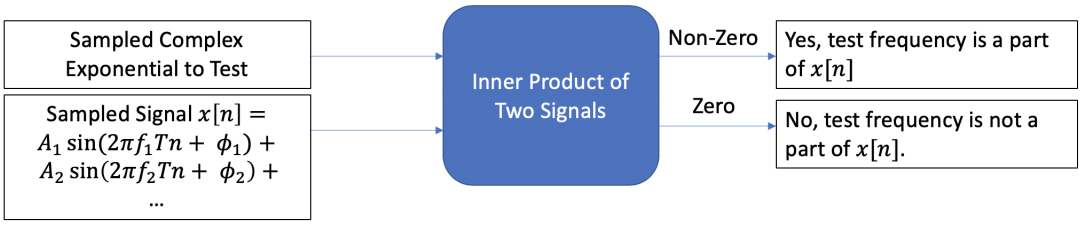
This is one of the most remarkable equations in all of mathematics. The key takeaway is that we can use a complex exponential to express both a real cosine term and an imaginary sine term. Again there is nothing "imaginary" about the sine wave. The complex exponential can simply be expressed as two **separate** sinusoids. Fortuitously, a complex exponential is just what we need for our current problem. Remember we want to take the inner product of a signal with a cosine and sine wave, and we want to keep the results separate. All we need to do then is to take the inner product of our signal with a complex exponential. For example, if we have a signal $x[n]$ and we want to test it for frequency f_{test}, we can compute $\langle x[n], e^{2\pi f_{test} i} \rangle$. This notation says to take the inner product with the real cosine term

of frequency f_{test} and the inner product with imaginary sine term of frequency f_{test}. Because the real and imaginary numbers are kept separate, we will get an answer of the form $a + bi$ where a is the result of the inner product with cosine and b is the result of the inner product with sine.

For digital signal processing, we will sometimes see Euler's formula express with a different variable ω which stands for angular frequency. Angular frequency is related to frequency by the equation $\omega = 2\pi f$. So we can rewrite Equation 1 as $e^{i\omega} = \cos(\omega) + i\sin(\omega)$ or $e^{2\pi fi} = \cos(2\pi f) + i\sin(2\pi f)$. This better expresses the relationship between the frequency of the sinusoids and the complex exponential.

Figure 6 shows the updated version of our procedure for calculating the inner product.

Figure 6: Complete solution using complex exponentials.



It is important to understand that Figure 6 is no different from Figure 5. The inner product in Figure 6 produces two inner products stored in the real and imaginary components, respectively. Those are the same inner products calculated in Figure 5. If both the real and imaginary component are zero then we know the test frequency is not a part of our signal $x[n]$.

Unrelated to the issue of inner products, Euler's formula also allows us to translate back and forth between sinusoids like sine and cosine and exponentials. Exponentials are much easier to work with and allows us to quickly prove equations like trigonometric identities that would be much harder if we simply had to work with just sine and cosine. In general, if you plan to learn more about digital signal processing, you will need to get comfortable with complex exponentials.

****NEED TO MENTION ABOUT NEGATIVE AND SAY -SIN****

The Discrete Fourier Transform

The DFT Equation

We now have all the necessary knowledge to understand how the Discrete Fourier Transform works. Here is the equation for it below:

$$X_k = \sum_{n=0}^{N-1} x[n] \cdot e^{-i\frac{2\pi}{N}kn} \tag{2}$$

There is a lot to unpack here. At a high level X_k represents how much of some frequency we have in some signal $x[n]$. X_k is a complex number and is the result of the inner product of our signal $x[n]$ and some test frequency denoted by the complex exponential $e^{i\frac{2\pi}{N}kn}$. If X_k is zero, then we know we do not have that frequency in $x[n]$. If X_k is non-zero, then we do have some amount of that frequency in $x[n]$. The DFT equation is simply a mathematical description of the procedure described in Figure 6. Though Equation 2 may look daunting, its simply calculating the inner product of two signals just as we have done before.

Noticeably absent though from Equation 2 is any mention of frequency, usually described with the variable f. Understand that the DFT is in fact calculating the inner product of the signal $x[n]$ with some frequency. However, we cannot determine what the frequency is without knowing the sampling rate. This is true if we were to look at any sequence of samples including those operated on by the DFT. If we were to simply look at a sequence of samples, say from just a simple sinusoid, we cannot make any claims about the frequency of that sinusoid. We need to know how fast those samples will be played back. The faster the playback, then the higher the frequency. Similarly, the slower the playback, then the lower the frequency. Frequency can only be determined by the combination of a sequence of samples in conjunction with a sampling rate, usually described as f_s. With the DFT, we make no assumptions about the sampling rate of the signal $x[n]$. We treat $x[n]$ as just a sequence of samples and we test every sinusoid that could be periodic along that sequence of samples.

Remember that our conditions for orthogonality only hold when we deal with two sinusoids that are periodic over some interval. In the DFT, that interval is the number of samples of our audio signal $x[n]$ denoted by N. A sinusoid is periodic along some interval if it competes an integer number of cycles. To test all the sinusoids that are periodic along N, we need to calculate the inner product with the sinusoids that evenly complete one cycle, two cycles, three cycles, four cycles, etc. The variable k denotes the number of complete cycles that our testing sinusoid completes. If we know the sample rate f_s, we can determine the frequency of the sinusoid we are testing using k and N. The sample rate is a ratio of the number of samples taken per second. If we divide the sample rate by N, we can calculate how many periods of N it takes to span one second of time. For example if $f_s = 1000$Hz and $N = 250$, then $f_s/N = 4$ tells us that four cycles of N constitutes one second of time. If we know that our testing sinusoid completes k cycles over those N samples, then it must complete $k \cdot f_s/N$ across one second which is simply the frequency of the sinusoid. Therefore, we can state that $f = \frac{k}{N} f_s$. Therefore, we know the frequency represented by X_k once we have the sampling rate. In fact, you can see in the complex sinusoid all the components for determing frequency (i.e., $\frac{k}{N}$) except the sampling rate. So the DFT does in fact encode all the information for the frequency of the test sinusoid once the sample rate as known.

Perhaps you may be wondering why the sample rate is simply not included in the DFT equation. That's a reasonable question and would make sense in the domain of audio. In fact, we could write a variation of the DFT equation that takes into account the sample rate of our signal.

$$X_f = \sum_{n=0}^{N-1} x[n] \cdot e^{-i2\pi \frac{f}{f_s} n} \tag{3}$$

Here we take the same inner product of $x[n]$ with some complex sinusoid of frequency f and known sample rate f_s. It is easy to see that this new variation is equivalent to Equation 2 by the fact that $f = \frac{k}{N} f_s$. However, if you were to use the version of the DFT from Equation 3, you would need to be sure that the frequencies you tested were in fact periodic along the interval N. While Equation 3 may translate the standard DFT equation to the variables of f and f_s that you are likely more comfortable with, it becomes much less obvious what frequencies actually do complete integer number of cycles along N. That is the purpose of the variable k and why Equation 2 is actually the easier way to view the computation of the DFT. It is also very little work to figure out the frequency with $f = \frac{k}{N} f_s$.

Below summarizes the meaning of each variable in the DFT equation.

- $x[n]$: the signal
- n: the indexing variable
- X_k: a complex number that is the result of the inner product of $x[n]$ and a complex sinusoid
- k: the number of complete cycles our testing complex sinusoid completes
- N: the number of samples of our signal $x[n]$
- Frequency can be determined if the sampling rate is known by using $f = \frac{k}{N} f_s$

A Simple Example

Let's use a simple example to illustrate how to use the DFT equation. Let us take a very small example. Suppose we have samples drawn from the following signal $x = 0.5\cos(2\pi t + \pi) + 0.25\cos(4\pi t - 1)$. The signal x contains two sinusoids of frequency 1Hz and 2Hz with differing phases and amplitudes. Let us say we were to draw eight samples ($N = 8$) from x at a sampling rate of 8Hz for ease. Here are those samples below:

$$x[n] = [-0.3649, -0.1432, -0.1351, 0.1432, 0.6351, 0.5639, -0.1351, -0.5639]$$

Now let us use the DFT equation to check whether to see our sampled signal contains the frequency 2Hz. We of course know that it does but the DFT should give us a non-zero value. Let us verify that is in fact true. We need to first figure out what k should be. We can simply solve using $f = \frac{k}{N} f_s$ and determine that for $f = 2$, we should use $k = 2$. Plugging in our values for k and N, we now need to compute the following:

$$X_2 = \sum_{n=0}^{7} x[n] \cdot e^{-i\frac{\pi}{2}n}$$

Unfortunately, we cannot just simply plug this into our normal calculators and figure out what X_2 is. The complex exponential needs to be converted into its sinusoidal form using Euler's formula as shown in Equation 1 in order to be calculated. So let us rewrite that now:

$$X_2 = \sum_{n=0}^{7} x[n] \cdot (\cos(-\frac{\pi}{2}n) + i\sin(-\frac{\pi}{2}n)) = \sum_{n=0}^{7} x[n]\cos(-\frac{\pi}{2}n) + i\sum_{n=0}^{7} x[n]\sin(-\frac{\pi}{2}n))$$

Let us calculate each summation independently. The left summation is the real part of the complex number X_2 and the right summation is the imaginary part.

$$\begin{aligned}\sum_{n=0}^{7} x[n]\cos(-\frac{\pi}{2}n) =&(x[0]\cdot 1) + (x[1]\cdot 0) + (x[2]\cdot -1) + (x[3]\cdot 0)\\ &+ (x[4]\cdot 1) + (x[5]\cdot 0) + (x[6]\cdot -1) + (x[7]\cdot 0)\\ =&x[0] - x[2] + x[4] - x[6]\\ =&-0.3649 - (-0.1351) + 0.6351 - (-0.1351)\\ =&0.5404\end{aligned}$$

Half of the result trivially goes away because many of those samples are multiplied by zero. The bracket notation $x[1]$, for example, simply means to use the 1st sample from $x[n]$. Note that we count starting from index 0 in this notation so $x[1] = -0.1432$ and not -0.3649. Using the same procedure, we can calculate the imaginary summation as well and we will see that the result is $-0.8415i$. Therefore, the value of X_2 is $0.5404 - 0.8415i$. Note that X_2 is non-zero! Therefore we can see that the frequency 2Hz is indeed part of our signal as it should be.

If we want to test whether 3Hz is a part of $x[n]$, then we can calculate the DFT with the appropriate k for 3Hz which also happens to be 3. Using the same procedure, we would find that $X_3 = 0 + 0i$. X_3 is zero, meaning that 3Hz is not part of $x[n]$ as expected. Of course, we will almost always be using the DFT on unknown signals for $x[n]$. The DFT can let us test to see whether a periodic sinusoid is part of our signal.

Reconstructing Magnitude and Phase

While the DFT can tell us whether a certain frequency is part of some signal, it can do even better. The complex number X_k can be used to reconstruct the magnitude and phase of the sinusoid as well. Let us assume that the signal $x[n]$ has a sinusoid of the form $A\cos(2\pi fTn + \phi)$ [2] that is periodic along the interval N. If it is periodic along N, we can also write the sinusoid in the form $A\cos(2\pi\frac{k}{N}n + \phi)$. The latter form matches nicely with X_k because the subscript k from X_k is the same k in $A\cos(2\pi\frac{k}{N}n + \phi)$. When we take the DFT for $x[n]$, the complex number will be of the form shown in Equation 4 if the sinusoid exists in $x[n]$. Remember it will be 0 otherwise.

$$X_k = \frac{AN\cos(\phi)}{2} + \frac{AN\sin(\phi)}{2}i \tag{4}$$

In some notations for complex numbers, the real part and imaginary parts of a complex number is given with the symbols: Re() and Im(), respectively. If I had an imaginary number $z = 3 + 4i$, then $\mathrm{Re}(z) = 3$ and $\mathrm{Im}(z) = 4$. The symbols themselve may seem complex but they are simply a convenience for referring to the two components of an imaginary number. Therefore, I could say that in our particular example with $x[n]$ that $\mathrm{Re}(X_k) = \frac{AN\cos(\phi)}{2}$ and $\mathrm{Im}(X_k) = \frac{AN\sin(\phi)}{2}$. Given Equation 4 and our handy tools for referring to the imaginary and real parts of a complex number, we can derive the amplitude A from X_k as follows:

[2] We could very well assumed a sinusoid of the form $A\sin(2\pi fTN + \phi)$ instead of using a cosine. It certainly makes no mathematical difference as a sine wave and cosine wave are only differentiated by their phase. It will be nicer mathematically if we think about it as a cosine wave. The corresponding X_k would have a complex number of the form $X_k = \frac{AN\sin(\phi)}{2} - \frac{AN\cos(\phi)}{2}i$. While this is perfectly valid, the careful reader will note that the angle or argument of the complex number does not match ϕ as it does in the cosine form. The cosine form presents a nice convenience.

$$A = \frac{2}{N}\sqrt{(\text{Re}(X_k))^2 + (\text{Im}(X_k))^2} \tag{5}$$

Recalling our example above, let's reconstruct the magnitude from $X_2 = 0.5404 - 0.8415$. Plugging in the real and imaginary component to Equation 5, we get $A = \frac{2}{8}\sqrt{(0.5404)^2 + (-0.8415)^2} = 0.25$. If we look back at the original signal $x = 0.5\cos(2\pi t + \pi) + 0.25\sin(4\pi t - 1)$, you will see that the sinusoid with frequency 2Hz does indeed have an amplitude of 0.25.

Similarly, we can also use X_k to get back the phase ϕ of the sinusoid as well. Equation 6 shows the formula for doing so.

$$\phi = \tan^{-1}\left(\frac{\text{Im}(X_k)}{\text{Re}(X_k)}\right) \tag{6}$$

From the same example, if we compute $\tan^{-1}(\frac{-0.8415}{0.5404})$, we get -1 which is indeed the phase of $0.25\sin(4\pi t - 1)$. One needs to be careful when using the inverse tan function with computers or calculators. The sign of the real and imaginary part indicates which quadrant the phase will be located. Most computers and calculators however keep the phase between $-\frac{\pi}{2}$ and $\frac{\pi}{2}$. Some software like Matlab and others offer a special function that takes into the account the sign of the numerator and denominator. It is sometimes referred to as "atan2" or "arctan2". It is fine to use either but you simply need to be aware of the limitations of your software and may need to correct some phases by hand if your calculator keeps the phase between $-\frac{\pi}{2}$ and $\frac{\pi}{2}$.

These two equations will work for nearly all the frequency bins X_k. The exceptions are when k/N is a multiple of $\frac{1}{2}$ which corresponds to $k = N/2$ (i.e., the Nyquist frequency) and $k = 0$ (sometimes termed the D.C. offset). In these instances, the complex number X_k cannot be used to recover the original amplitude and phase because $X_k = AN\cos(\phi) + 0i$. Because the imaginary component is zero, we do not have a way to determine what proportion of A and ϕ contribute to the real component of X_k. We simply have too many variables and not enough information. Generally though, this is not a big deal. We do not expect to have any frequency component at the Nyquist frequency if we have properly filtered our signal to prevent aliasing. The D.C. offset is also rather inconsequential. D.C. offset refers to how far away the average of a signal deviates from zero. It is not a component of the signal that contributes to pitch.

Frequency Bins

The DFT equation from Equation 2 only calculates the inner product of our signal against one frequency. Typically though we want to calculate the DFT for numerous frequencies. Remember that frequency is not explicitly stated in the DFT equation. k is the variable we use to test our signal against different periodicities. The DFT calculates all values of k from 0 all the way up to $N - 1$. A value of $k = 0$ calculates how much D.C. bias is in the signal (i.e., if the averages of the samples is something other than zero). A value of $k = \frac{1}{2}N$ tests for presence of the Nyquist frequency defined as half the sampling rate. A value of $k = N$ would test for the presence of the frequency equal to the sampling rate f_s but it turns out that the DFT starts repeating itself starting at $k = N$. You can test this yourself but $X_0 = X_N$ and $X_1 = X_{N+1}$ and so on.

Figure 7: A table of the DFT results from samples of $0.5\cos(2\pi t + \pi) + 0.25\cos(4\pi t - 1)$

k	Frequency (in Hz)	X_k	X_k Magnitude	X_k Phase	Magnitude	Phase
0	0	$0 + 0i$	0	0	0	0
1	1	$-2 + 0i$	2	π	0.5	π
2	2	$0.5403 - 0.8415i$	1	-1	0.25	-1
3	3	$0 + 0i$	0	0	0	0
4	4	$0 + 0i$	0	0	0	0
5	5	$0 + 0i$	0	0	0	0
6	6	$0.5403 + 0.8145i$	1	2.57	0.25	1
7	7	$-2 + 0i$	2	$-\pi$	0.5	$-\pi$

Figure 7 shows a table of all the calculations from the DFT in the column labeled X_k. For convenience, the frequency of each bin is also given. Because the number of samples equals the sampling rate, the

value of k is in fact the frequency but that will rarely be the case. We generally deal with very high sampling rates like 44.1kHz and smaller numbers of samples for the DFT. To calculate the frequency of each X_k, sometimes referred to as frequency bins, we simply compute $k\frac{f_s}{N}$. You can see that in our trivial example that $\frac{f_s}{N} = \frac{8}{8} = 1$ so the frequency of each bin is just k.

The magnitude and phase for each complex number X_k are given. See Appendix A for more information about how the magnitude and phase are calculated for a complex number. Generally, when we examines plots of the DFT, we are looking at graphs of the magnitude and phase of the complex number. The magnitude of X_k tells us how much of that frequency is present in the original signal. The magnitude of X_k is proportional to the amplitude of the original frequency, but it does not give the exact amplitude of the original frequency. For that we would need to use Equation 5. The original phase and the phase of the complex number are also given as well. For music applications, the phase is usually unimportant information. We generally care most about the magnitude because small differences in phase tend to not impact the way we perceive sound. There are exceptions though and in some applications we need to be careful with the way we treat phase.

Interpreting Magnitude and Phase Graphs

When we perform the DFT on a particular signal, we often display the magnitude and phase graphically. Below are two examples of the magnitude and phase spectrum of some signal $x[n]$ with sample rate $f_s = 256$Hz and $N = 128$. Let's see what we can deduce about the time domain form of the signal from these two spectrums.

Figure 8: The magnitude spectrum of $x[n]$ at $f_s = 256$ and $N = 128$

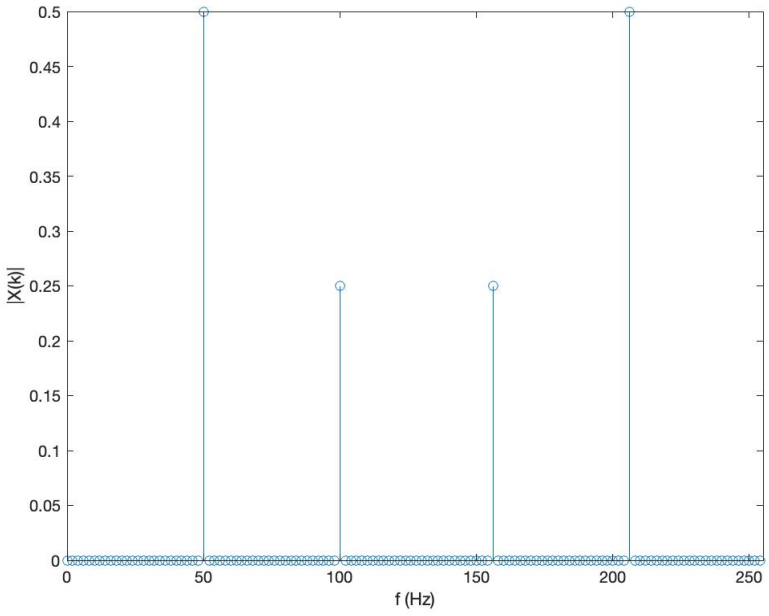
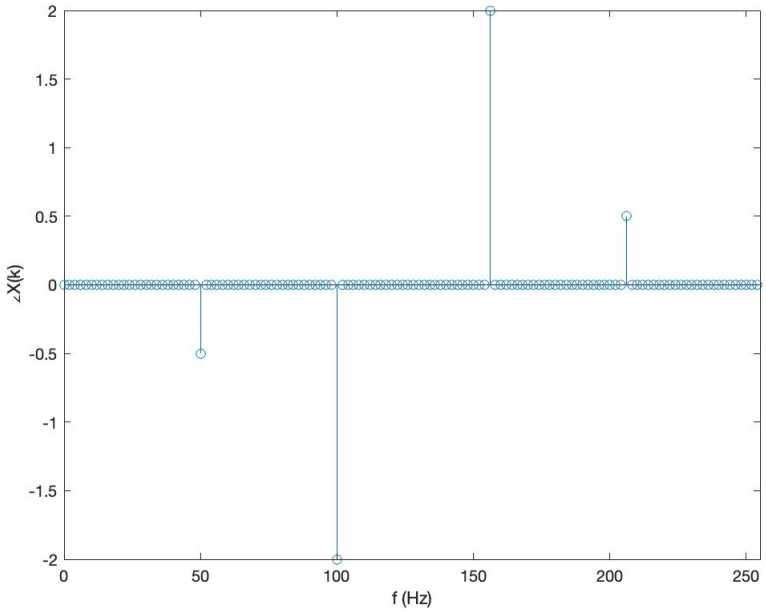


Figure 8 shows the magnitude spectrum of $x[n]$. When looking at the magnitude spectrum of a sound it is important to know how the magnitude is scaled. The magnitude of a complex number is always calculated as $\sqrt{(\text{Re}(z))^2 + (\text{Im}(z))^2}$; however, that magnitude is sometimes multiplied by a constant. An unnormalized magnitude simply computes $\sqrt{(\text{Re}(z))^2 + (\text{Im}(z))^2}$. Plotting unnormalized magnitudes is less common. Why? The magnitude spectrum is proportional to the size of N. A larger number of samples N will proportionally increase the magnitude spectrum of the signal even if there is no change in the sinusoidal components of the signal. Look back at Equation 4. You will see that X_k can increase or decrease without change to A or ϕ simply by changing N. An altenative is to simply divide out the factor of N, and sometimes the factor of $1/2$, from the real and imaginary component so that the value of each frequency bin is related solely by the phase and amplitude of the original sinusoidal component. This is

Figure 9: The phase spectrum of $x[n]$ at $f_s = 256$ and $N = 128$



referred to as a normalized magnitude spectrum. The normalized magnitude spectrum is calculated as $\frac{1}{N}\sqrt{(\text{Re}(z))^2 + (\text{Im}(z))^2}$.

Figure 8 shows the normalized magnitude spectrum as $\frac{2}{N}\sqrt{(\text{Re}(z))^2 + (\text{Im}(z))^2}$ to remove the factor of 1/2. This is exactly equivalent to Equation 5. Therefore, the advantage here is that the magnitude spectrum of Figure 8 depicts the exact amplitude of each sinusoidal component of the original signal. In general though, the magnitude is usually calculated as $\frac{1}{N}\sqrt{(\text{Re}(z))^2 + (\text{Im}(z))^2}$. The form of $\frac{2}{N}\sqrt{(\text{Re}(z))^2 + (\text{Im}(z))^2}$ is ideal if we know that all frequency components of $x[n]$ are periodic along N. This assumption about periodicity is the key for asserting Equation 4. Most signals, however, will not be perfectly periodic along N. In fact, the DFT will not allow us to perfectly reconstruct the exact frequencies, amplitudes, and phases using the complex frequency bins if the signal has aperiodic frequency components along N. Therefore, we usually just see the normalized magnitude spectrum in the form of $\frac{1}{N}\sqrt{(\text{Re}(z))^2 + (\text{Im}(z))^2}$ or the unnormalized form.

Let us examine Figure 8. We see four peaks in the graph. The magnitude spectrum here is calculated using $\frac{2}{N}\sqrt{(\text{Re}(z))^2 + (\text{Im}(z))^2}$, equivalent to the amplitude of each sinusoidal component. Therefore, we see four amplitudes at frequencies 50Hz, 100Hz, 156Hz, and 206Hz. We can ignore the components of the spectrum above the Nyquist frequency of 128Hz. Therefore, we have two sinusoidal components in $x[n]$: 50Hz at an amplitude of 0.5 and 100Hz at an amplitude of 0.25.

Note the symmetry in the magnitude spectrum. If we were to draw a straight line down the center of the magnitude spectrum at the Nyquist frequency, the left half and right half would be perfectly symmetrical. A fundamental property of the DFT is that when $x[n]$ is a real signal and does not include any complex components, the two halves of the DFT are symmetrical. Because we are dealing solely with audio signals, we always use the DFT with real signals and therefore will always have symmetry.

Figure 9 shows the phase spectrum of $x[n]$. The phase spectrum of a signal depicts the argument or phase of the complex number X_k. Recall that the argument of the complex number is the angle between the imaginary and real components if we were to plot them on a 2D graph. Why plot the argument of the complex number? The phase of the original sinusoidal component is exactly the argument of the complex number, assuming we model the sinusoidal component as a cosine wave. As expected we see peaks at exactly 50Hz, 100Hz, 156Hz, and 206Hz. Again, we can ignore the two frequencies above the Nyquist frequency. We can see phases of -0.5 and -2 at 50Hz and 100Hz, respectively. Putting together the information from the magnitude and phase spectrum, we can perfectly recreate the original signal: $x[n] = 0.5\cos(2\pi(50)t - 0.5) + 0.25\cos(2\pi(100)t - 2)$.

There is also symmetry in the phase spectrum as well. Again if we were to draw a straight line

vertically at the Nyquist frequency and flip the signs of one of the halves, we would have two symmetrical halves. Again these symmetrical properties are true for all real signals $x[n]$ such as audio.

Periodicity

DFT Assumptions

We have discussed at length how two sinusoids of different frequencies are orthogonal, assuming they are both periodic along some interval. This was a key assumption that allowed us to parse a signal into its various constituent frequencies. Recall that any signal can be broken down into a summation of sinusoids. When we take the inner product of some signal $x[n]$ with a sinusoid (let's call it f), that is the same as taking the inner product of f with each sinusoid that constitutes $x[n]$. Everything that we have learned so far has hinged on the assumption that all sinusoids, including the ones that make up $x[n]$, are periodic along that same interval of samples N. If you look back at the small example from earlier, the signal $x[n]$ was composed of two sinusoids $0.5\sin(2\pi t + \pi)$ and $0.25\sin(4\pi t - 1)$, both of which are periodic over one second. When we took the DFT of that signal, we tested x for frequencies of 0Hz, 1Hz, 2Hz, etc., all of which are also periodic over one second. This example was contrived to work perfectly within the assumptions of the DFT, namely that both $x[n]$ and our testing frequencies were periodic over the number of samples from the signal. Remember that we used 8 samples at a sampling rate of 8Hz which consitutes one second of time.

In the real world, we will rarely, if ever, be that lucky. If I were to take a series of N samples from a song or from a speech, it is very unlikely that the samples would be periodic along N. So what are we to do? Is the DFT somehow now unusable? The answer is no. But we do need to recalibrate our assumptions of what it can and cannot do. For one, the DFT will not be able to tell us with complete precision all the frequencies and their respective amplitudes and phases that constitute any signal. But perhaps, we should not have been misled to begin with. The DFT produces a finite number of frequency bins and its possible that our signal could have any number of sinusoidal components. So already we can see that it is not the perfect solution to our problem. The DFT, however, can give us a good **estimate** of what frequencies are part of some signal.

When we attempted to reconstruct the magnitude and phase of the original signal in the previous section, we saw that the value of X_k was derived from the phase and amplitude of a cosine wave as shown in Equation 4. Though we did not show the derivation for that equation, it is similarly based on the assumption that all frequency components of $x[n]$ are periodic along N. Moving forward, we will not be able to use Equation 4 unless we know with certainty that $x[n]$ is periodic along N. In turn, we must also jettison Equation 5 and Equation 6 as they are dervied from Equation 4. Not all hope is lost though. We can still use the DFT to say plenty about the frequency spectrum of $x[n]$. The magnitude of X_k will give us a good estimate of the relative strength of the frequency components in the original signal as we will see shortly. The phase of X_k will also give us a good estimate of the original phase as well.

Some of our conclusions will remain the same regardless of $x[n]$. The symmetry we saw in the magnitude and phase spectrum still applies. We shall also see that we can perfectly recover the original $x[n]$ regardless of its periodicity exactly using a technique called the Inverse Discrete Fourier Transform.

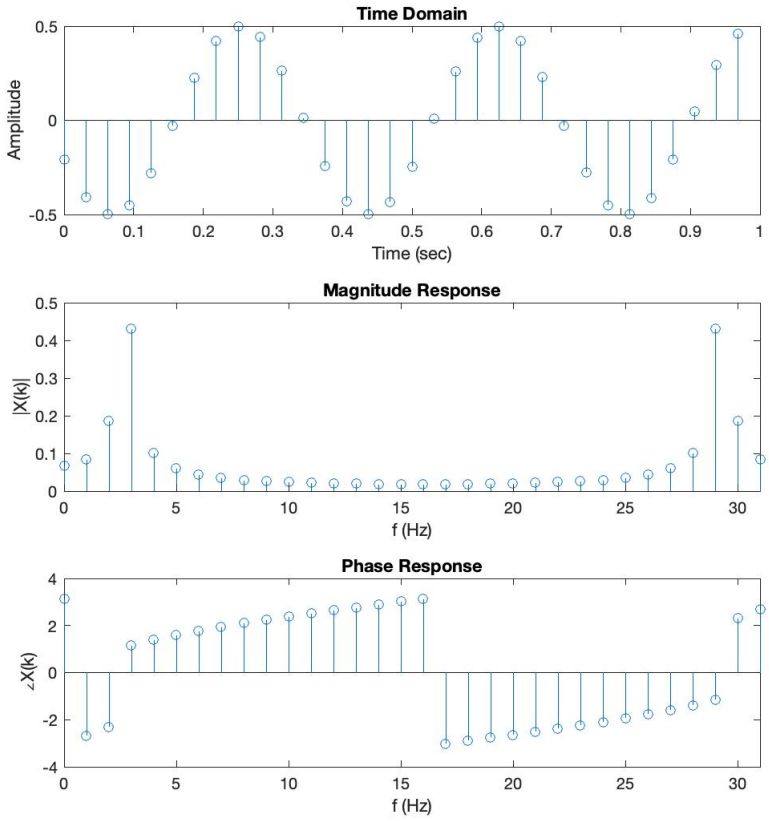
Though the DFT has shortcomings and does not give the complete picture of the frequency domain for all $x[n]$, it is still incredibly powerful and useful. We will still be able to gain a tremendous amount of information about our signal and do some interesting spectral processing to create engaging audio effects.

DFT of an Aperiodic Sinusoid

Let us consider what happens when we take the DFT of a signal that is not periodic along N. For ease, let us start with the simplest of signals, a sinusoid. We will take $N = 32$ samples from a cosine wave that is not periodic along N. The frequency of the cosine wave will be 2.7Hz with an amplitude of 0.5 and a phase of 2. To get the samples, we can use $x[n] = 0.5\cos(2\pi(2.7)Tn + 2)$ and compute the values for $n = 0, 1, 2, ...$ etc. The sampling rate is 32Hz (and thus $T = 1/32$) which means we will produce frequency bins of 0Hz, 1Hz, 2Hz, etc. Importantly, note that there is no frequency bin for 2.7Hz. It might be a fair assumption to guess that all the frequency bins will be zero since we only have one frequency in the signal and it does not match any of the frequency bins.

Plotted in Figure 10 is the magnitude and phase spectrum of the original signal as well as the time domain representation of the signal.

Figure 10: The time domain, magnitude response and phase response of $x[n] = 0.5\cos(2\pi(2.7)Tn + 2)$ for $N = 32$ and $f_s = 32$Hz.



We can see in the time domain representation that the signal is aperiodic along N. We can observe that the the signal does not start and end in the same place. Sample 0, notated as $x[0]$, has an amplitude of roughly -0.2 and the final sample $x[31]$ at time 0.96875 seconds is nearly at 0.5. A periodic signal should begin and end roughly near the same amplitude. Alternatively, we can see that the signal does not complete an even number of cycles. It completes two complete cycles plus some fractional number of cycles.

After computing the DFT of $x[n]$, Figure 10 shows the normalized magnitude and phase response. The normalized magnitude response was calculated using a factor of $2/N$ meaning that the magnitude response mirrors the amplitude of the original signal. If we look at the graph, we can see two peaks roughly between 2 and 3Hz and between 29 and 30Hz. We can also see the symmetry between the two halves if we split the magnitude spectrum down the middle. We can see that every frequency bin has some magnitude in it which might suggest that our original signal was composed of 32 sinusoids of varying amplitudes. But we know that is not the case. There is just one at 2.7Hz. We can, however, see "activity" at 2.7Hz in the magnitude response. That is one of the peaks in the magnitude spectrum. In fact, the original amplitude of 0.5 is pretty close to height of that peak around 2.7Hz. The other peak is simply the mirror image of 2.7Hz above the Nyquist frequency. We saw this symmetry occurred even with periodic sinusoids. So many of the same instincts that we derived in the context or periodic $x[n]$ hold true with aperiodic $x[n]$.

How do we account though for positive magnitudes in all the bins? This is a phenomenon called **spectral leakage**. We will always see peaks in the magnitude response at the frequencies of $x[n]$ provided those frequencies have sufficient amplitude. If the frequency perfectly matches one of the frequency bins (i.e., periodic), then we will see one spike at that bin. If not, we will see a peak at

the bins closest to the frequency and a smooth spreading out around that peak. This is the "leakage". Thus, when we look at the magnitude response for this simple $x[n]$, we are not seeing multiple different frequencies. We are seeing one peak, indicating we have just one frequency component.

How can we interpret the phase response? Again we see a sudden change around 2Hz. The phase response shifts drastically at the location moving from negative to positive. The original phase of the sinusoid from $x[n]$ was $+2$, and it is hard to see any relationship between the phase response of $x[n$ around 2.7Hz and the original phase. In general, the phase response of the DFT is more abstruse. It is difficult to parse. Sometimes sudden shifts in the phase response are the result of the periodic nature of sinusoids. Phases outside the range $+\pi$ to $-\pi$ get wrapped back down to that range. This can cause abrupt shifts in the phase response. We will not be spending much time examining the phase response as it usually is less important for musical applications.

The Periodicity of the DFT

We have seen how the DFT responds to periodic and aperiodic sinusoids. In this section, I would like to offer another perspective on the DFT. We have learned that the DFT can accurately parse the frequencies of $x[n]$ if $x[n]$ is periodic along those N samples. I should also state explicitly that the DFT **assumes** that those N samples are from a periodic function regardless of whether they are or not and that those N samples constitute a period from that periodic signal $x[n]$. If we look back at the time domain representation of Figure 10, we can see a plot of the samples from the sinusoid of 2.7Hz. The DFT assumes those N samples make up one period of the **periodic** signal $x[n]$. $x[n]$ is indeed periodic but those N samples are not a period of $x[n]$. One way to see what the DFT "sees" is to simply repeat those N samples and look at the time domain representation of the sound. The top plot in Figure 12 shows a plot of 2.7Hz with more samples to highlight the disjointed transition when the samples repeat. The DFT is parsing the frequencies of this signal and not the smoothly oscillating sinusoid from where the samples originated. Notice how this signal shown in the top plot of Figure 12 is **not** the same as a smooth sinusoid. The abrupt transition that happens at each repeat creates an entirely new signal. That is the signal that the DFT converts to the frequency domain, and accurately I might add. Another way, then, to think about spectral leakage is that it is the distortion of the original waveform that occurs when we fail to capture a complete period. An important intuition to take away is that sharp discontinuities in the time domain create rich spectrum in the frequency domain.

In math, we can show that any periodic signal can be represented as a sum of cosine waves where each cosine wave is a harmonic of the fundamental period N. This is called the Fourier Series, and it turns out we can derive the DFT from the Fourier Series. We will not do that here. But the main point is that the DFT assumes the N samples are a period from periodic $x[n]$ and attempts to represent the N samples as a sum of cosine waves, each of which is a harmonic of time period spanned by N.

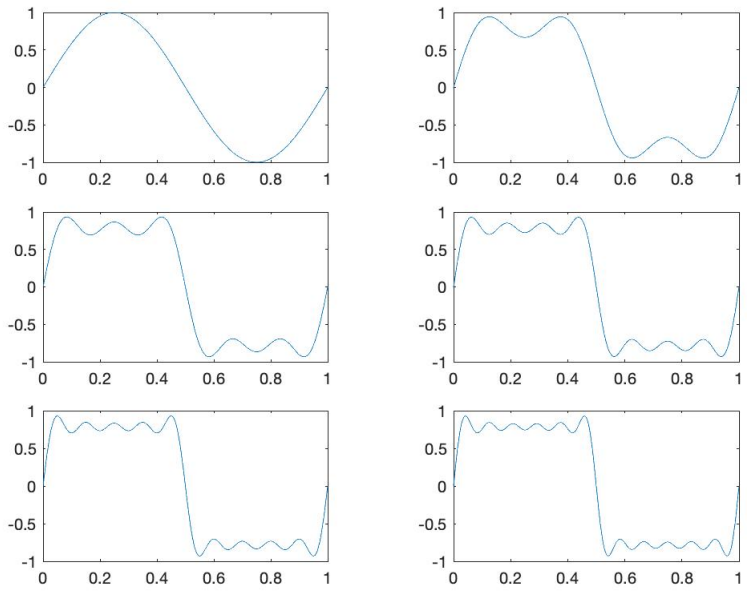
To demonstrate the Fourier Series, we will use a classic example: the square wave. The Fourier Series tells us we can represent the square wave as a sum of cosine waves. This should feel wrong on some intuitive level. The square wave has discontinuities and lines of zero slope. It would seem impossible that we could take a series of cosine waves – let alone cosine waves from a harmonic series – and sum them to create a square wave. Yet it is true! A square wave can indeed be created by summing the odd harmonics where each harmonic's amplitude is one over the harmonic number.

Figure 11 shows an iterative process where each successive chart shows the next harmonic being added. As more and more harmonics get added, the graph begins to move closer to the shape of a square wave. A square wave is composed of an infinite number of harmonics. So we will not be able to display the sum of an infinite number of sinusoids using a computer. This example though illustrates a key point of signal processing, namely that any periodic waveform can be represented as a sum of harmonic sinusoids.

Let us look back at the example we used to discuss how the DFT reacted to an aperiodic sinusoid. In that example, we took a slice of $N = 32$ samples from $x[n] = 0.5\cos(2\pi(2.7)Tn + 2)$ that did not capture a complete period from $x[n]$. Nevertheless, the DFT still considered those N samples to be periodic. Again, the top plot in Figure 12 shows how the DFT perceives the periodicity of that waveform. The bottom plot from Figure 12 shows an approximation of that signal using a sum of harmonic cosine waves. Remember that any periodic signal can be represented as a sum of infinite cosine waves. It is an approximation though because we simply have not summed an infinite number of cosine waves.

How did I know which cosine waves to sum to achieve this approximation? From the DFT, of course! The magnitude and phases we calculate for each frequency bin can be used in conjunction with the frequency of each bin to construct a series of cosines waves that, when summed, produce the

Figure 11: Successive addition of harmonics to approximate a square wave



approximation shown in the bottom plot from Figure 12. For example, the first frequency bin has a frequency of 0, magnitude of 0.0523, and a phase of π. If we plug those values into a cosine wave, we get $0.0523 * \cos(2 * \pi * 0 * t + \pi) = 0.0523 * (-1) = -0.0523$. The first frequency bin of every DFT is always how much DC offset is in the signal. We can see that -0.0523 is relatively small which makes sense because our original signal, even lopped off at the end, has relatively little DC offset. The second frequency bin has a frequency of 1Hz, a magnitude of 0.0696, and a phase of -2.6026. If we plug those values into a cosine wave, we get $0.0696 * \cos(2 * \pi * (1) * t - 2.6026) = 0.0696 \cos(2\pi t - 2.6026)$. We can add this to -0.0523 to get the first two sinusoidal approximation. If we continued in this fashion with every frequency bin up to $k = N/2$, the Nyquist frequency, we get the approximation shown in the bottom plot from Figure 12.

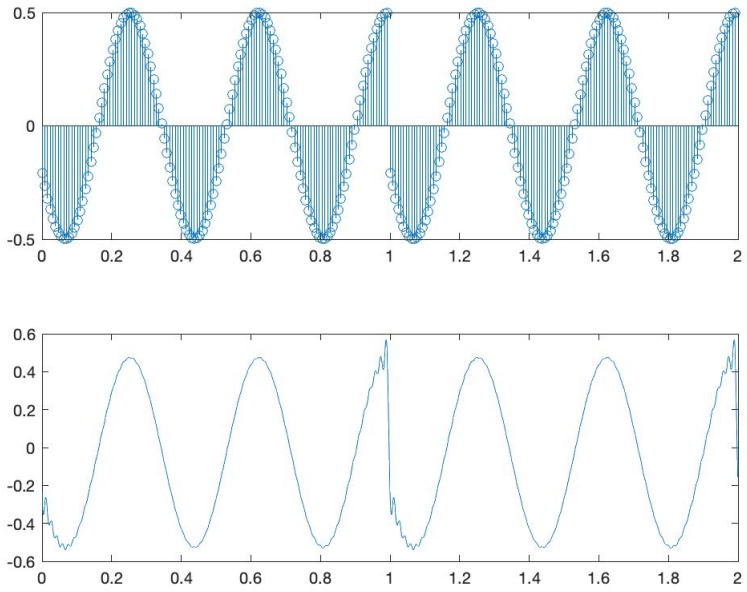
In this interpretation of the DFT, we can see how both the phase and magnitude derive meaning. We can think of the DFT as producing a series of harmonic cosine waves that get their specific parameters from the values in each frequency bin. We call these harmonic cosine waves a **basis**, and we **project** our original signal $x[n]$ onto this basis. This is the geometric interpretation of the DFT. The terms "projection" and "basis" are mathematical terms. One common "basis" we all know is the coordinate axis system. It is a space that allows us to plot any 2D shape. A series of harmonic sine and cosine waves also form a basis. It's a basis that allows us to plot or describe any periodic signal. We can think of projection as a means to transform our original signal in terms of our basis (i.e., a sum of harmonic sine and cosine waves). But how exactly do we do this projection or transformation? With the inner product. The inner product, with which we started our duscussion of the DFT, is a means of projecting a signal onto a different basis such as the frequency domain. Therefore, another way to view the DFT is that it takes a period of N samples and projects them onto a set of harmonic sinusoids that approximate the waveform described by those N samples.

The DFT on Real-World Sounds

The DFT on Audio

So far we have used contrived examples to get a better understanding of how the DFT works on both periodic and aperiodic samples. Let us put it all together and examine an unknown piece of audio. We will look at the magnitude and phase response of the samples and draw some conclusions about the original audio file. Later in this section, I will reveal the origin of the audio file and we can reexamine our

Figure 12: Periodicity of the DFT as shown through the time domain representation in the samples and the cosine basis



conclusions. Figure 13 shows the time domain representation, magnitude response and phase response of this signal.

If we look at the time domain representation of the sound, we can see a repeating peak of the same shape. This strongly suggests strong harmonic content because repetitions in the time domain produce a fundamental whose frequency is the number of repetitions per second. The period of the shape is roughly 0.003 seconds or a third of a hundredth of a second. With a period of 0.003 seconds, we would assume that we would have a fundamental roughly around 330Hz. Though its a little difficult to tell, the magnitude response also has a peak near 330Hz which confirms are intuition about the presence of 330Hz in the sound. Observe as well how the time domain waveform maintains the same period but has slight differences in each repetition and particularly over time. We can see the amplitude of the time domain waveform grows over the span of that tenth of a second. These slight variations suggest a real instrument as opposed to an electronic waveform like a sawtooth wave which would have identical repetitions. Those slight differences as the sound progresses through time give a real instrument that "naturalness" that distinguishes it from electronic instruments.

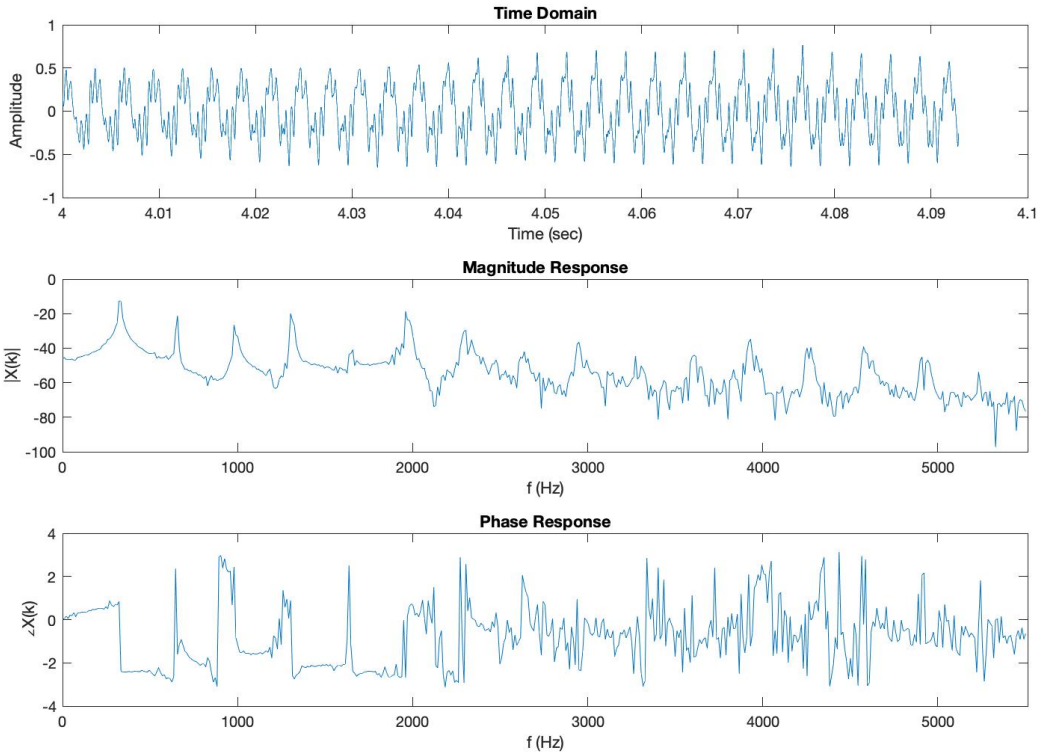
We will ignore the phase response for this analysis; however, the magnitude response contains important information. Notice the succession of peaks equidistant apart. We can see them at roughly 330Hz, 660Hz, 990Hz, etc. All these subsequent peaks after 330Hz represent partials of the fundamental 330Hz which roughly corresponds to an "E4". Though the whole magnitude spectrum is not shown, we can see a relatively rich harmonic spectrum where many of the harmonics are present. Interestingly though, the fifth harmonic at 1650Hz is not as prominent. Based on this magnitude spectrum, it is likely that this is a real instrument that produces lots of harmonics.

It turns out that this is a violin playing an "E4". Both the time domain and the magnitude spectrum suggested an instrument playing a note at "E4". Violins are instruments with rich spectrum. The variation in the peak height help give the violin its unique sound. If we were to take the DFT of the next series of samples, we would see slight changes in the peak height. The slight evolution of harmonics as well as several inharmonic partials give the violin the wonderful sound we have come to love.

Time Resolution vs. Frequency Resolution

The DFT provides a straightforward way to convert audio from the time domain to the frequency domain. Many programming languages and audio applications have optomized algorithms that handle

Figure 13: The time domain, magnitude response and phase response plots of an unknown audio signal



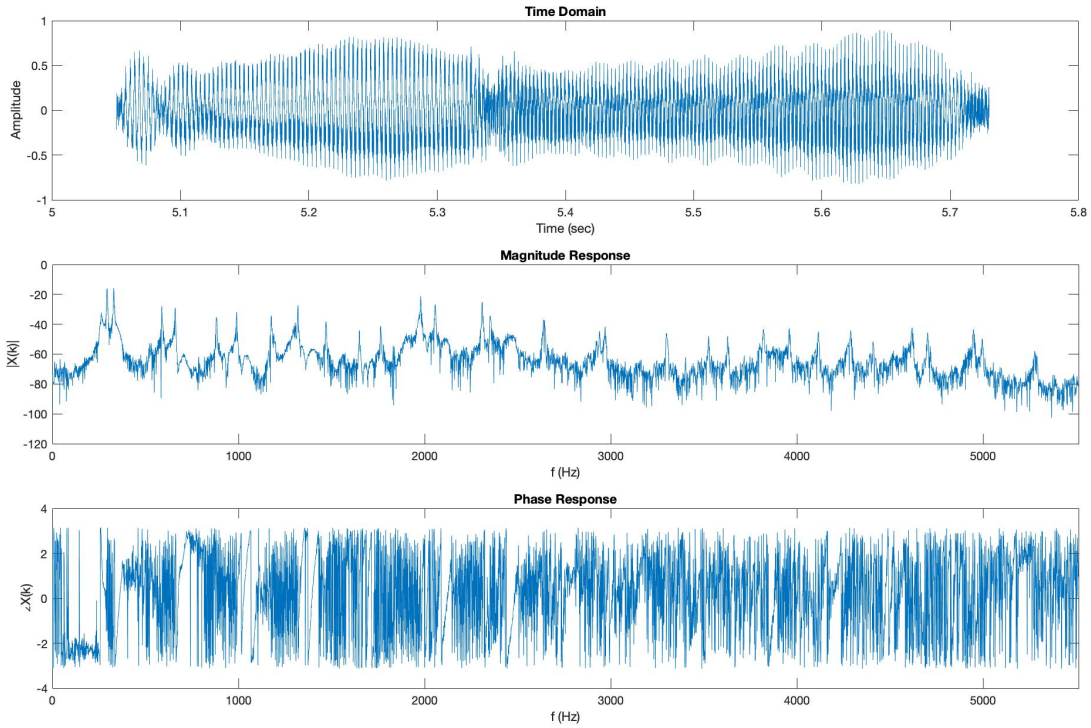
all of the computation. The user typically decides what samples will be processed and how many. The number of samples N is an important consideration. Recall that the number of frequency bins is equivalent to the number of samples taken and that the distance between each bin is f_s/N. The larger N is, the more frequency bins and the more finely parsed the frequency spectrum is. N plays an important role in separating frequencies, particularly lower frequencies. Our perception of intervals is based on the ratio betweendistinct frequencies. For example, a ratio of 2:1 expresses an octave and holds true whether the frequencies are 40Hz and 20Hz or 20000Hz and 10000Hz. Lower notes are clustered closer together in frequency while higher notes are spaced farther. By contrast, the frequency bins of the DFT are linear. The same distance separates each bin. Consider a standard sampling rate of $f_s = 44100$Hz and a $N = 1024$. The distance between each bin is approximately 43Hz, leading to bins of 0Hz, 43Hz, 86Hz, etc. The frequency 43Hz corresponds roughly to the note F1 and 86Hz corresponds roughly to F2. That is a full octave separating those two frequency bins!

To increase the frequency resolution, we can increase the number of samples we process. This is an easy solution, but it comes at a cost. The more samples N we take, the poorer our time resolution becomes. We generally use the DFT on short slices of audio so we can get a snapshot of the frequency content. The larger N becomes, the wider that snapshot is. For audio with a slow changing spectra, this is not a problem. For audio with fast moving spectra, the N samples can contain many musical moments, making it harder to interpret the magnitude response.

As an example, consider a larger N for the violin E4 excerpt we saw earlier. Let's set N to 30000, significantly larger than the $N = 4096$ samples we took to plot Figure 13. This moment comes from a later portion in the audio file and is plotted in Figure 14.

If we examine the magnitude response, we can see a set of two repeating peaks starting around 290Hz. Using the same intuitions from earlier, it would be logical to surmise that we have two notes playing with fundamentals around 290Hz and slightly higher at, say, 330Hz. Though it is difficult to tell from Figure 14, those peaks occur at exactly frequency bins 294Hz and 330.8Hz, both very close to the notes D4 and E4 respectively. Indeed, it is true that the violin plays those two notes from 5.05 to

Figure 14: The time domain, magnitude response and phase response plots of a violin



5.75 seconds. However, the magnitude response fails to indicate whether those notes are successive or chordal. The time domain, nevertheless, provides some clues. We can see two distinct notes starting at 5.1 seconds and 5.35 seconds suggesting that the notes are played successively. Choosing a large N like 30000 in this example unfortunately captured two distinct musical moments and merged the frequency content of both notes into one magnitude spectrum. This is an example of where poor time resolution creates confusing magnitude spectra. A better solution to this problem would be to use the DFT twice to analyze each note separately. This would require smaller N, and in turn our frequency resolution would be poorer. One nice thing about the magnitude response shown in Figure 14 is that the peak frequency bins are almost exactly the frequency of the original note, making it easy to determine the fundamentals. Reducing N would potentially move the frequency bins farther away from those fundamentals. This is tradeoff we must make when choosing a size for N.

As a general rule, if the audio file has a slow changing spectrum like ambient music, choose a larger N. The issues related to time resolution will be mitigated if the audio experiences relatively little change in the time domain. If the audio file has fast music, then use a smaller N. Though the frequency resolution will be poorer, a larger N will simply capture too many musical events to parse.

The Inverse Fourier Transform and Spectral Processing

The Inverse Discrete Fourier Transform

We have seen how a time domain signal can be projected onto a basis of cosine and sine waves to give the frequency domain representation of the sound. Is it possible to inverse that process? Can we take a signal in the frequency domain and convert it to the time domain? Fortunately, the answer is yes! This technique is called the Inverse Discrete Fourier Transform (abbreviated as IDFT). Equation 7 shows the equation for the IDFT.

$$x[n] = \frac{1}{N} \sum_{k=0}^{N-1} X_k \cdot e^{i\frac{2\pi}{N}kn} \tag{7}$$

If we examine the IDFT, we can see that we sum the inner product of a complex sinusoids against all of our complex frequency bins X_k to restore the original signal $x[n]$. The equation for the IDFT is very similar to the one for the DFT shown in Equation 2. The only differences are $x[n]$ and X_k are swapped, the complex exponential is now positive, and the IDFT has a scaling constant of $\frac{1}{N}$. The appendix shows a proof that the IDFT and DFT are truly inverse operations. The beauty of the IDFT is that it allows us to convert back and forth between the frequency domain and the time domain. We can convert a signal to the frequency domain, perform some spectral manipulations and then convert back to the time domain, creating a processed version of the original. Though less frequently done, we could also build a signal in the frequency domain and then use the IDFT to get the time domain representation of the sound. At the end of the day, all signals need to be converted back to the time domain because all digital-to-analog converters (ADCs) require time domain representations of sound.

Spectral Processing

The DFT and IDFT provide the means to convert an audio signal to the frequency domain for processing and then convert back to the time domain. Spectral manipulation involves changing the complex numbers for each X_k. Simple operations include zeroing out bins, scaling the magnitude, changing the phase, rearranging bins, etc. There is a host of different techniques that one could apply. One seemingly obvious example is to create a brickwall filter by zeroing out frequency bins to create a high-pass, low-pass, or bandpass filter. Zeroing out bins only creates a brickwall filter if the signal $x[n]$ is periodic along N. Otherwise, nasty artifacts will be present in the time domain once the IDFT is complete. Discussing and analyzing various spectral techniques is beyond the scope of this guide; however, experimenting with simple spectral operations is good practice for developing intuitions.

Beyond the DFT

We have seen how the DFT is a way to transform an audio signal into the frequency domain. We have also seen that it is not a perfect tool. The DFT assumes two things:

1. The signal $x[n]$ is periodic
2. The N samples constitute one period from $x[n]$

We could add a third condition, namely that we should be careful to sample from a bandlimited signal so that we do not produce aliasing in the frequency representation. But generally this is assumed.

Unfortunately, when these two main conditions are not obeyed, the DFT cannot accurately parse the frequency components of $x[n]$. Nearly all $x[n]$ in practice violate these two conditions. Nevertheless we can look at peaks in the magnitude response to get a good estimate of where the original frequencies lie. In general, the magnitude response contains the most important information we need.

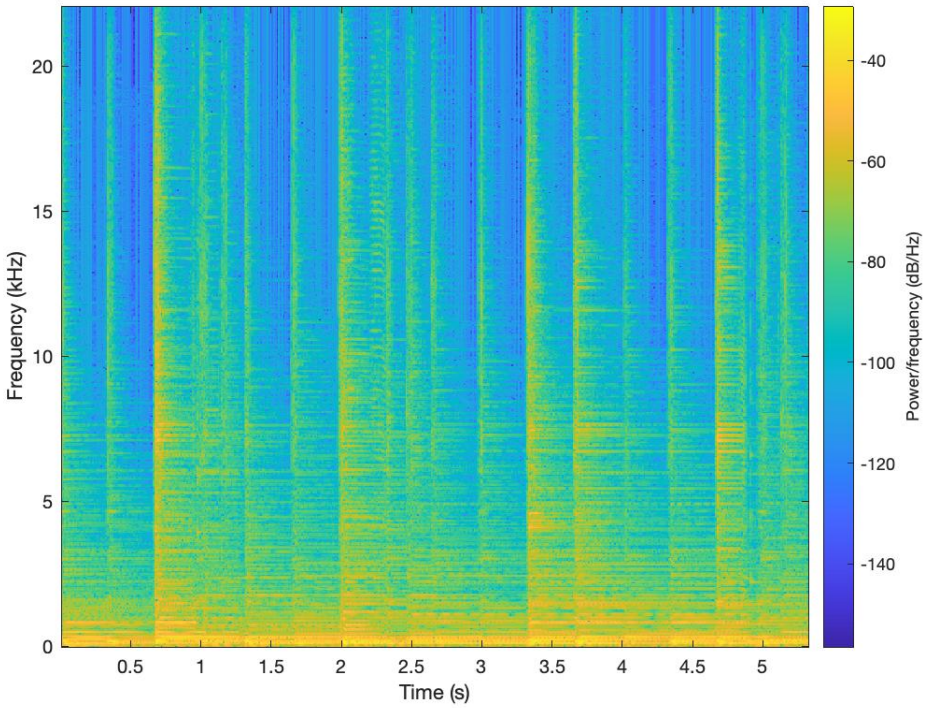
Short-Time Fourier Transform

As we saw in the section on Time Resolution vs Frequency Resolution, too many samples starts to blur different time components of the sound. Generally we want to get a snapshot of each moment in a song. The Short-Time Fourier Transform takes successive DFTs on a longer stretch of audio to get a sense of how the frequency components of sound change over time. In audio platforms like Max/MSP or SuperCollider or underneath the hood in Digital Audio Workstations like Logic or ProTools, the Short-Time Fourier Transform (abbreviated STFT) is the main tool we use for spectral processing. We divide up an audio segment into a series of small moments in time, take the DFT of each of those moments, and apply processing in the frequency domain to create some particular audio effect. The STFT is also used for audio analysis. We often plot the results of the STFT as a spectrogram. Figure 15 shows the spectrogram of an electric guitar.

The x-axis represents the time and the y-axis represents frequency. Each vertical bar in the spectrogram represents one DFT of one segment of N samples from the audio file. The spectrogram plots the magnitude response. Higher magnitudes are distinguished from lower magnitudes using a color map. In

this spectrogram, magnitude is charted on a decibel scale where higher magnitudes are brighter and lower magnitudes are darker. The spectrogram provides a nice visualization for how the frequency content of a sound evolves over time.

Figure 15: Spectrogram of strummed guitar chords



In this spectrogram, it becomes easy to tell where each strum lies. Like many instruments, the attack portion of the guitar is percussive and noiser in sound. Noise tends to have equal energy throughout the entire frequency spectrum. In the spectrogram, those strums are the solid yellow and green vertical lines, representing equal energy across that portion of audio. The harmonic parts are the comb-like lines that protrude to the right of those solid vertical lines. In harmonic sounds, the frequency spectrum only has energy at the harmonics. Those horizontal lines articulate the energy of each harmonic present in the sound. Taken together, we can see how the percussive attacks followed by the harmonic resonance gives a complete picture of the frequency content for these guitar chords.

More DSP

Understanding the tools of digital signal processing like the Discrete Fourier Transform is imperative for audio programming and developing industry tools in audio. For musicians, we care mostly about the tools that are created and how to use them. The actual implementation and mathematical framework necessary to understand DSP theory is left to the engineers and mathematicians. Nevertheless, having a solid foundation in digital signal processing, and in particular the techniques to convert between the time and frequency domains, will invariably make us better musicians and quicker to capitalize on digital audio tools. Hopefully, this guide has demystified the DFT to some degree and given you better insight into this important tool for audio signal processing.

Appendix C: Inverse DFT Stuff

In this appendix, we will show that the IDFT can be used to recover the original signal $x[n]$ from the frequency domain. Recall Equations 2 and 7 for the DFT and IDFT, respectively. Let's start the equation for the IDFT shown below:

$$x[n] = \frac{1}{N} \sum_{k=0}^{N-1} X_k e^{i\frac{2\pi}{N}kn}$$

We want to show that the right side is also $x[n]$ if X_k was calculated using the DFT on $x[n]$. Here, we can substitute in the DFT equation for X_k on the sequence $x[n]$. For clarity, we will use a different index variable m for the DFT equation to distinguish it from the IDFT index variable n.

$$x[n] = \frac{1}{N} \sum_{k=0}^{N-1} \left(\sum_{m=0}^{N-1} x[m] \cdot e^{-i\frac{2\pi}{N}km} \right) e^{i\frac{2\pi}{N}kn}$$

Here we will do some clever rearranging.

$$x[n] = \frac{1}{N} \sum_{k=0}^{N-1} \sum_{m=0}^{N-1} x[m] \cdot e^{i\frac{2\pi}{N}k(n-m)}$$

$$x[n] = \frac{1}{N} \sum_{m=0}^{N-1} \sum_{k=0}^{N-1} x[m] \cdot e^{i\frac{2\pi}{N}k(n-m)}$$

Note that the switching of summations is a clever trick that occurs all the time in DFT proofs.

$$x[n] = \sum_{m=0}^{N-1} \frac{1}{N} \sum_{k=0}^{N-1} x[m] \cdot e^{i\frac{2\pi}{N}k(n-m)}$$

$$x[n] = \sum_{m=0}^{N-1} x[m] \left(\frac{1}{N} \sum_{k=0}^{N-1} e^{i\frac{2\pi}{N}k(n-m)} \right)$$

Let us now analyze the portion in parentheses. If $n \neq m$, then $\frac{1}{N} \sum_{k=0}^{N-1} e^{i\frac{2\pi}{N}k(n-m)} = \frac{1-e^{2\pi i(n-m)}}{N(1-e^{2\pi i(n-m)/N})} = 0$. The trick here is solving the summation using the formula for a finite geometric sum. We can reason that the number is always zero because $n-m$ is always an integer and therefore the complex exponential has a phase that is always a multiple of 2π. If $n = m$, then $\frac{1}{N} \sum_{k=0}^{N-1} e^{i\frac{2\pi}{N}k(n-m)} = \frac{1}{N} \sum_{k=0}^{N-1} 1 = \frac{N}{N} = 1$.

If we analyze the outer summation, then we will see that each term in the summation will be 0 except when $n = m$. That term has a value of $x[m]$ or $x[n]$. Therefore, we have showed that the right side does indeed reduce to $x[n]$.