An Intuitive Guide to the Discrete Fourier Transform in Audio

Andrew Davis

September 9, 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

$$A \sin(2\pi f t + \phi)$$

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.

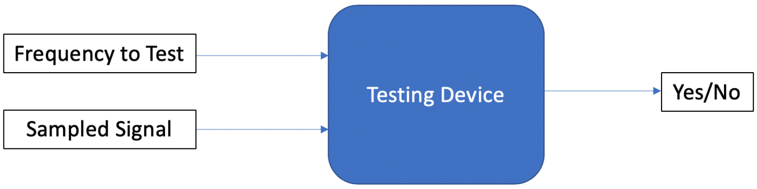
So why do we care about the Discrete Fourier Transform in particular? Because it converts a finite discrete signal (i.e., a finite number of samples) into a finite discrete frequency domain (i.e., a finite number of sinusoids with particular phases and amplitudes). Some of these other tools like the 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. Figure 1 shows a diagram of such a testing device. It should be able to take our signal and some frequency and output one of two things: yes, the frequency is part of the signal or no, the frequency is not part of the signal. 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. In essence, this is the approach of the DFT, and the testing device we use to detect frequencies is called the “inner product”.

Figure 1: A sketch of the desired procedure for determining frequency in a signal.



Orthogonality and the Inner Product

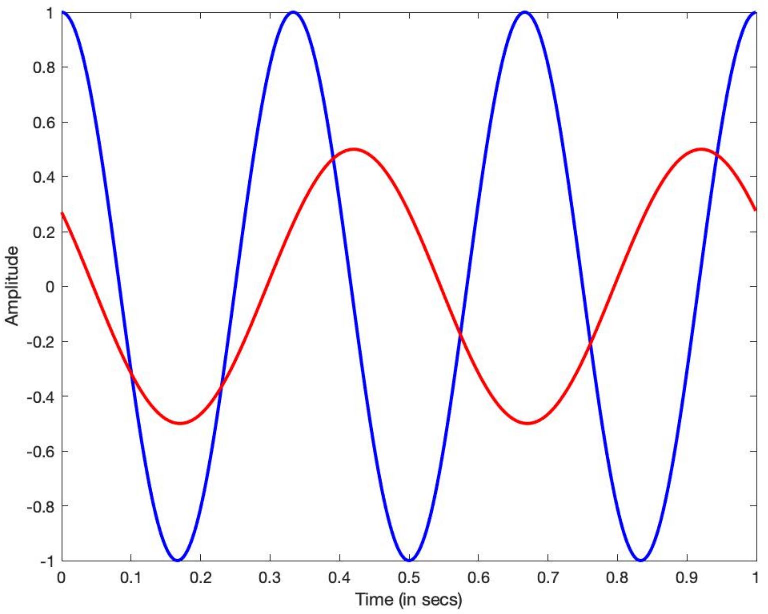
The inner product is an operation just like addition or multiplication and has several definitions depending upon the application. Unlike addition and multiplication, the inputs to the inner product are not numbers. They are **sequences** of numbers. But what exactly is a sequence of numbers? A sequence is simply an ordered collection. In digital signal processing, a sequence of numbers is defined with square brackets. For example, we could have a sequence $[9, -2, 7]$. Because sequences are ordered, the sequence $[9, -2, 7]$ is distinct from $[-2, 7, 9]$. It matters that the initial element is 9 as opposed to -2.

Additionally, we like to give sequences names so that we can easily refer to them. We could assign the sequence $[9, -2, 7]$ the name x with an index variable n such that $x[n] = [9, -2, 7]$. The index variable is used to refer to any element in the sequence. For example, the notation $x[0]$ is equivalent to the zeroth element in the sequence, namely 9. In digital signal processing and nearly all computer languages, we always number sequences starting with index zero. It can be a confusing convention at first. For example, $x[1]$ refers to -2 and not 9. In digital signal processing, sequences generally hold samples of audio (i.e., amplitudes). So a sequence like $x[n] = [0.5, 0.2, 0]$ is a sound file with three samples of amplitude 0.5, 0.2, and 0.

Sequences are the inputs to the inner product. **The inner product is calculated by multiplying the values at each index of the two sequences and then summing those products**. As an example, let us take two sequences: $x[n] = [1, 2, 3]$ and $y[n] = [1, -1, 0]$. The inner product of $x[n]$ and $y[n]$ is $(1 * 1) + (2 * -1) + (3 * 0) = -1$. We take the first numbers of each sequence 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 inner product will always yield a single number. If the result is zero, we say that the two sequences are orthogonal. In this case, $x[n]$ and $y[n]$ are not orthogonal. In math, we often use angle brackets to denote the inner product. For example, the inner product of sequences x and y is notated as $\langle x, y \rangle$.

We can also take the inner product of two functions as well. The inner product for two functions is also defined as the sum of pointwise products. Consider the two sinusoids shown in Figure 2. The red wave can be expressed as $0.5\cos(2\pi(2)t + 1)$, and the blue wave can be expressed as $\sin(2\pi(3)t + \pi/2)$.

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



In this example, we will take the inner product over a specific range of time from $t = 0$ to $t = 1$. To begin, we start with the first points from the red and blue sinusoids at $t = 0$. At that moment, the red 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 1. That's an infinite number of points! Fortunately, if we can express these sinusoids in the form $A \sin(2\pi f t + \phi)$, then the calculation becomes relatively trivial with calculus. The inner product for these two functions over this time period is zero, and therefore the functions are orthogonal over that time period.

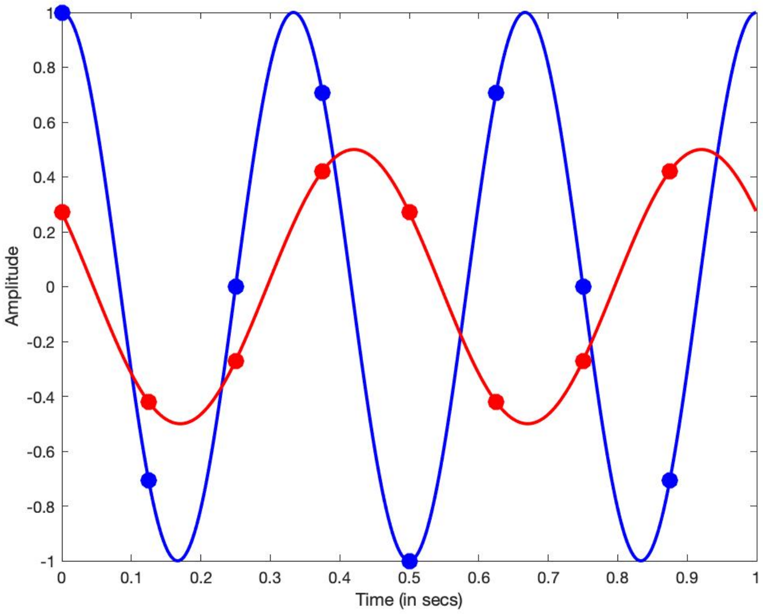
It is important to note that we have not proven anything or ascribed any significance yet to the inner product or orthogonality. We have simply defined an operation (i.e, the inner product) that can be performed on any two sequences or functions, nothing more. Similarly, orthogonality is simply a label attached to the result of the inner product if it is zero. So why do we care about orthogonality and the inner product? 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. Note that there are some caveats to this claim that we will discuss soon.

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 interval, but we must provide the extra condition that the waves are **periodic** along that interval. 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 2, we will see that both the blue and the red 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. To illustrate and provide a little proof of concept, let us sample the two sinusoids from Figure 2. If you look at Figure 2, we can see that the red 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 should expect that when we take the inner product of these two sequences of samples that we get a result of zero because the frequencies are different. To start, we need to choose a sampling rate that will prevent aliasing. Anything above 6Hz will suffice, so let us choose 8 Hz. Figure 3 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 red sinusoid are $r[n] = [0.27, -0.421, -0.27, 0.421, 0.27, -0.421, -0.27, 0.421]$. The inner product of $b[n]$ and $r[n]$ 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 which is expected because the two sinusoids have different frequencies.

Figure 3: Two sinusoids that are periodic along the interval 0 to $L = 1$.



As a side note, samples for a sinusoid can be generated by replacing time t with Tn where T is the sampling period and n is the sample number or index. T is related to the sampling rate f_s. T specifies the amount of time between each sample and can be calculated as $1/f_s$. The two sequences $b[n]$ and $r[n]$ were generated by sampling the sinusoids $\sin(6\pi t + \pi/2)$ and $0.5\cos(4\pi t + 1)$. Therefore, $b[n] = \sin(6\pi Tn + \pi/2)$ and $r[n] = 0.5\cos(4\pi Tn + 1)$.

Let us 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. **Moving forward unless otherwise indicated, our claims will always be based on these two assumptions even when not explicitly indicated.**

Probing for Sinusoids

As stated at the beginning of this section, the inner product will be our tool for detecting frequencies in an audio signal. To this point, we have seen how the inner product of two sinusoids can determine the similarity of their frequencies. If the result of the inner product is non-zero, we know the sinusoids must have the same frequency. If the result is zero, the sinusoids must have different frequencies. But most audio signals are not a single sinusoid. So how can the inner product detect the presence of sinusoids in a more complex audio signal?

Recall that any song, speech, soundscape, or any other kind of audio can be broken down into a sum of sinusoids as shown below.

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

Suppose that we want to know if some audio signal has a particular frequency. Let us call that frequency f_{test}. One strategy would be to separate the audio signal into its sinusoidal components and take the inner product of a sinusoid with frequency f_{test} against each component. Any non-zero result would indicate that f_{test} was present in the audio signal. Unfortunately, we cannot just dissect an audio signal and separate it into its individual components for testing. So how can we test each component?

Here, math comes to the rescue. The inner product has a special property called the **distributive property**. 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. The same is true for the inner product![2] 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 $\langle x, y + z\rangle = \langle x, y\rangle + \langle x, z\rangle$.

This is a wonderful result because we do not have to separate the frequency components of our audio signal before testing. 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 means the two sinusoids are of different frequency. Therefore, a non-zero result indicates that the frequency is a part of that signal. A result of zero means that the frequency is not a part of the signal. So the inner product is our tool to implement Figure 1.

To solidify this concept, let us take eight samples from some signal and test that signal for certain frequencies. We will name the signal $x[n]$ and we will assume that the sampling rate is 8Hz. Here are those samples:

$$x[n] = [0.1728, 0.2574, -0.0445, -0.0432, -0.1728, -0.2574, 0.0445, 0.0432]$$

Let us first test to see whether 2Hz is present in the signal. To do so, let us generate samples from a 2Hz sinusoid. Here are those samples below set to the variable t:

$$t[n] = [0, 1, 0, -1, 0, 1, 0, -1]$$

Let us now take the inner product of $x[n]$ with $t[n]$, notated as $\langle x, t\rangle$.

$$\langle x, t\rangle = (0.1728 \cdot 0) + (0.2574 \cdot 1) + (-0.0445 \cdot 0) + ... = 0$$

The result of the inner product of x and t is zero, meaning that x and t are orthogonal. Therefore, the frequency 2Hz is not part of the audio signal x.

Let us test another frequency. Let us generate samples for 3Hz to see if that frequency is present in x. Below are samples from a 3Hz sine wave.

$$u[n] = [0, 0.7071, -1, 0.7071, 0, -0.7071, 1, -0.7071]$$

Again let us take the inner product of $x[n]$ with $u[n]$. This time the result is 0.392. A non-zero result indicates that 3Hz is present in x. We could test x against other frequencies to see if they exist. The samples for x were generated from the following function $0.2\sin(2\pi t + 1.3) + 0.1 * \sin(6\pi t - 0.2)$. As you can see 3Hz is represented by $0.1 * \sin(6\pi t - 0.2)$. If we tested x against a frequency of 1Hz, we would also get a non-zero result.

As an important reminder, orthogonality of sinusoids is predicated on the fact that all sinsuoids are periodic along the same interval. In this somewhat contrived example, I was careful to assemble a signal x whose frequency components were periodic along one second of time. This limited the creation of x to only integer frequencies. Similarly, I made sure to test x only with integer frequencies as well. Suppose I tested x against the samples of a frequency like 1.5Hz. We would get a non-zero inner product! This would seem to indicate the presence of 1.5Hz in the signal but we know that x was derived only from a sinusoid of 1Hz and 3Hz. The issue is that 1.5Hz is **not** periodic across a time period of one second. It is paramount to remember that all of the claims made so far are based upon the critical restriction that both the testing sinusoid and the components of our audio signal are all periodic along the same distance. Later in this document, we will return to the issue of aperiodicity but for now our claims will be based on this assumption.

[2] 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.

The Pesky $\pi/2$ Problem

It seems as though we have solved the problem stated at the outset of this section. We have discovered a tool that can detect the presence of a single frequency in any audio signal by taking the inner product of the signal and a sinusoid with some testing frequency. And essentially we have, but there is one special caveat that we need to address. **It is not true that only sinusoids of different frequencies are orthogonal. Two sinusoids of the same frequency are also orthogonal when their phase differs by $\pi/2$.**

Why is this problematic? The inner product was supposed to be our perfect tool that could definitively determine the presence of a sinusoid. If the result was non-zero, the sinusoids have the same frequency. If the result was zero, the sinsusoids have different frequencies. The former is still true but the latter may not always be. It is possible that we could take the inner product of some audio signal against a testing sinusoid and get a result of zero even when the testing frequency is part of the audio signal.

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

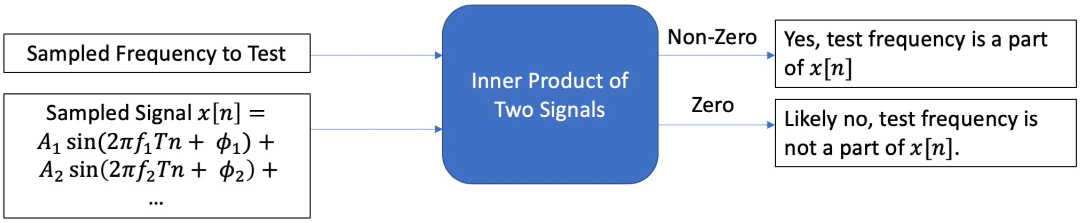

Sampled Frequency to Test
Sampled Signal $x[n] = A_1 \sin(2\pi f_1 Tn + \phi_1) + A_2 \sin(2\pi f_2 Tn + \phi_2) +$ ...
Inner Product of Two Signals
Non-Zero
Zero
Yes, test frequency is a part of $x[n]$
Likely no, test frequency is not a part of $x[n]$.

Figure 4 shows an updated version of the behavior of the inner product and what it means. We can see that a result of zero, likely means that the testing frequency is not part of the audio signal but not definitively.

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 1. 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 when the audio signal has that frequency.

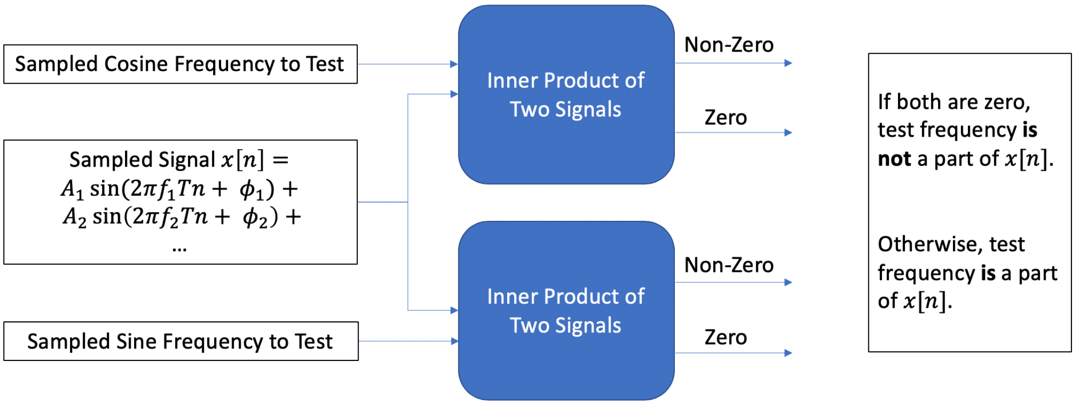
It may not be readibly apparent why using two sinusoids makes any difference. For simplicity, let us say that we will test our audio signal using a sine wave and a cosine wave, both of frequency f_{test}. Sine and cosine differ in phase by $\pi/2$. Let us assume that our audio signal also has a component of frequency f_{test} but we know nothing about that component's phase. In the original scenario, we would just take the inner product of our audio signal against a sine wave with frequency f_{test}. Most likely, the result would be non-zero, successfully indicating the presence of f_{test} in the audio signal. However, if the component in the audio signal differs in phase from a sine wave by exactly $\pi/2$, our result would be zero. Again, we wish that were not the case. Let us assume that the result is zero. A difference in $\pi/2$ from a sine wave is either a cosine wave or a negative cosine wave (i.e., $-\cos$). If we take the inner product again, now with a cosine wave, the result will be non-zero because a cosine wave is not $\pi/2$ out of phase with a cosine wave or a negative cosine wave.

In essence, there does not exist any phase that is both $\pi/2$ out of phase with both a cosine wave and a sine wave. Therefore, taking two inner products, one with sine and one with cosine, ensures that at least one of the results will be non-zero in the event the frequency is part of the audio signal. Figure 5 shows our completed implementation of the testing mechanism outlined from Figure 1. Note how we now need two sampled sinusoids: one cosine and one sine. While this requires extra processing, we can now definitively conclude the presence of a particularly frequency in an audio signal.

The DFT also takes two inner products to determine the presence of a particular frequency. However, it uses a cosine wave and a **negative** sine wave. The same principles still apply. There does not exist any phase that is both $\pi/2$ out of phase with both a cosine and a negative sine wave. So we will still be able to determine the presence of any frequency with certainty.

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.


Sampled Cosine Frequency to Test
Sampled Signal $x[n] =$ $A_1 \sin(2\pi f_1 Tn + \phi_1) +$ $A_2 \sin(2\pi f_2 Tn + \phi_2) +$ ...
Sampled Sine Frequency to Test
Inner Product of Two Signals
Non-Zero
Zero
Inner Product of Two Signals
Non-Zero
Zero
If both are zero, test frequency **is not** a part of $x[n]$.
Otherwise, test frequency **is** a part of $x[n]$.

Imaginary Numbers

Figure 5 shows that we will generate two results, one for each inner product. In digital signal processing, it is common to hold two separate values in a complex number. 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 tools for solving problems in mathematics, engineering and the physical sciences. This includes signal processing. In fact, complex numbers are just as "real" as say negative numbers. It is 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.

When we have a number like $a + bi$, we separate the real component a from the imaginary component b. The plus sign can be confusing because we cannot actually add a and b. They are treated as separate and distinct units. It is akin to how coefficients of distinct variables such as $3y + 4z$ cannot be added.

Because a complex number contains two distinct quantities, the real and imaginary components can be used to store the results of each inner product. We can use the real and imaginary components to represent any two distinct quantities. For example, we can use complex numbers to represent 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 3 apples and 4 oranges, we can write a single number $3 + 4i$. Complex numbers are a convenient device to represent any two values.

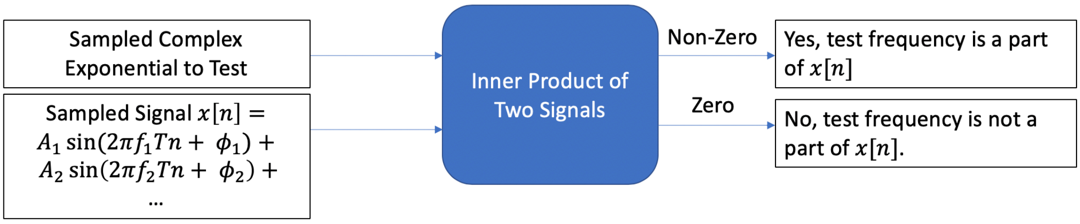
Complex numbers have another property that makes them the appropriate choice. The great mathematician, Leonhard Euler, discovered one of the fundamental mathematical equations that relates numbers such as i, e as well as sin and cos. It is called Euler's formula:

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

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 express 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 $\langle x[n], e^{2\pi f_{test} Tni} \rangle$. 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.

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

Figure 6: Complete solution using complex exponentials.


Sampled Complex Exponential to Test
Sampled Signal $x[n] = A_1 \sin(2\pi f_1 Tn + \phi_1) + A_2 \sin(2\pi f_2 Tn + \phi_2) + \ldots$
Inner Product of Two Signals
Non-Zero
Zero
Yes, test frequency is a part of $x[n]$
No, test frequency is not a part of $x[n]$.

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 $x[n]$. face 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.

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 just a mathematical description of the procedure described in Figure 6. Importantly, **the equation for the DFT tests only one frequency.** X_k is the result of just one test. To build the entire frequency spectrum for an audio signal, we would need to use the DFT equation multiple times.

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 the frequency without knowing the sampling rate. This is true if we look at any sequence of samples including those operated on by the DFT. Take a sequence of samples from a simple sine wave. We actually 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 notated 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 orthogonality for finite sinusoids only holds for periodic sinusoids. A sinusoid is periodic if it completes an integer number of cycles across some period. In the DFT, that period is the number of samples of our audio signal $x[n]$ denoted by N. 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 $f_s = 1000\text{Hz}$ 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 $k \cdot f_s/N$ cycles per second. This is the frequency of the testing sinusoid. Therefore, we can state that

$$f = \frac{k}{N} f_s \tag{3}$$

Once we have the sampling rate, we know the frequency represented by X_k. In fact, you can see in the complex sinusoid all the components for determing frequency (i.e., $\frac{k}{N}$) except the sampling rate.

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 by rearranging Equation 3 and substituting it into the DFT equation from Equation 2.

$$X_f = \sum_{n=0}^{N-1} x[n] \cdot e^{-i2\pi \frac{f}{f_s} n} \tag{4}$$

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 4, you would need to be sure that the frequencies you tested were in fact periodic along the interval N. While Equation 4 may translate the standard DFT equation to the variables of f and f_s with which you are likely more comfortable, it becomes much less obvious what frequencies actually do complete an 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 us use a simple example to illustrate the DFT equation. 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, respectively, 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 $x[n]$ contains the frequency 2Hz. We of course know that it does but let us ensure that the DFT returns a non-zero value. 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 into Equation 2, 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. 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 zero in this notation so $x[1] = -0.1432$ and not -0.3649. Using the same procedure, we can calculate the imaginary summation as well. That 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 $x[n]$. But as this small example illustrates, the DFT can let us test to see whether a periodic sinusoid is part of our signal.

Reconstructing Amplitude 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 amplitude and phase of the sinusoid as well. Let us assume that the signal $x[n]$ has a sinusoid of the form $A\cos(2\pi fTn + \phi)$ that is periodic along the interval N.[3] We can also write the sinusoid in the form $A\cos(2\pi\frac{k}{N}n + \phi)$ using Equation 3. 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 5 if the sinusoid exists in $x[n]$. Recall it will be zero otherwise.

$$X_k = \frac{AN\cos(\phi)}{2} + \frac{AN\sin(\phi)}{2}i \tag{5}$$

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 the imaginary number $z = 3 + 4i$, then $\mathrm{Re}(z) = 3$ and $\mathrm{Im}(z) = 4$. The symbols themselve may seem cumbersome but they are simply a convenience for referring to the components of an imaginary number. Therefore, we can say based on Equation 5 that $\mathrm{Re}(X_k) = \frac{AN\cos(\phi)}{2}$ and $\mathrm{Im}(X_k) = \frac{AN\sin(\phi)}{2}$. Given Equation 5 and our new notation, the amplitude A can be dervied from X_k as follows:

$$A = \frac{2}{N}\sqrt{(\mathrm{Re}(X_k))^2 + (\mathrm{Im}(X_k))^2} \tag{6}$$

Recalling our example above, let us reconstruct the amplitude from $X_2 = 0.5404 - 0.8415i$. Plug in the real and imaginary components to Equation 6. We then 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. For those familiar with complex numbers, Equation 6 looks remarkably close to the formula for the magnitude of a complex number. The magnitude of complex number z is $\sqrt{(\mathrm{Re}(z))^2 + (\mathrm{Im}(z))^2}$. The only difference then is a scaling factor of $2/N$. Equation 6 is sometimes referred to as the “normalized magnitude”.

Similarly, we can also use X_k to reconstruct the phase ϕ of the sinusoid. Equation 7 shows the formula for doing so.

[3] 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.

$$\phi = \tan^{-1}\left(\frac{\mathrm{Im}(X_k)}{\mathrm{Re}(X_k)}\right) \tag{7}$$

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 wrap the phase between $-\frac{\pi}{2}$ and $\frac{\pi}{2}$. Some software like Matlab and others offer a special function that accounts for the sign of the numerator and denominator. It is sometimes referred to as "atan2". It is fine to use either but you may need to correct some phases by hand if your calculator wraps the phase between $-\frac{\pi}{2}$ and $\frac{\pi}{2}$.

Equations 5, 6, and 7 apply to nearly all X_k. The exceptions is when k/N is a multiple of $\frac{1}{2}$, corresponding to $k = N/2$ (i.e., the Nyquist frequency) and $k = 0$ (sometimes termed the DC offset in electronics). In these instances, the complex number X_k cannot be used to recover the original amplitude and phase. In such instances, $X_k = AN\cos(\phi) + 0i$. Because the imaginary component is zero, we do not know what proportion of A and ϕ contribute to the real component of X_k. This is usually not an issue. We do not expect to have any frequency component at the Nyquist frequency if we have properly filtered our signal to prevent aliasing. The DC offset is also rather inconsequential. DC 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 $x[n]$ and one frequency. However, we want to calculate the DFT for numerous frequencies. Remember that frequency is not explicitly stated in the DFT equation but rather implied by k in conjunction with the sampling rate. To flesh out the full frequency spectrum, X_k is calculated for every value of k from 0 to $N-1$. Each calculation of X_k is sometimes referred to as a frequency bin. A value of $k = 0$ calculates how much DC offset is in the signal (i.e., if the averages of the samples is something other than zero). A value of $k = \frac{N}{2}$ tests for presence of the Nyquist frequency defined as half the sampling rate. We do not need to test for values of $k \geq N$ because the values of the DFT repeat. You can test this yourself as $X_0 = X_N$ 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	Amplitude	Phase
0	0	$0 + 0i$	0	0
1	1	$-2 + 0i$	0.5	π
2	2	$0.5403 - 0.8415i$	0.25	-1
3	3	$0 + 0i$	0	0
4	4	$0 + 0i$	0	0
5	5	$0 + 0i$	0	0
6	6	$0.5403 + 0.8145i$	0.25	1
7	7	$-2 + 0i$	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, k is equivalent to frequency of the kth bin, 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.

Figure 7 also has columns for amplitude and phase as calculated using Equations 6 and 7, respectively. We can see that for a frequency of 1Hz there exists a sinusoid of amplitude 0.5 and phase π and for a frequency of 2Hz there exists a sinusoid of amplitude 0.25 and phase -1. As expected the samples for $x[n]$ were taken from a signal matching the specifications of those two sinsusoids, namely $0.5\cos(2\pi t + \pi) + 0.25\cos(4\pi t - 1)$.

We can also see that we seemingly have extra sinusoids in our frequency domain at 6Hz and 7Hz. We can ignore these frequencies. The DFT is symmetric about the Nyquist frequency so any frequencies below the Nyquist frequency will appear above the Nyquist frequency. This is simply a byproduct of the math used for the DFT. We could have simply just calculated the frequency for bins $k = 0$ up to the Nyquist frequency. However, the standard frequency domain computed by the DFT ranges from $k = 0$ to $N - 1$. For audio signals, the upper half of the frequency domain is unimportant but for different signals it is.

Interpreting The Magnitude and Phase Spectra

When we calculate the DFT, we often display the magnitude and phase of each X_k 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 us 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 $N = 128$

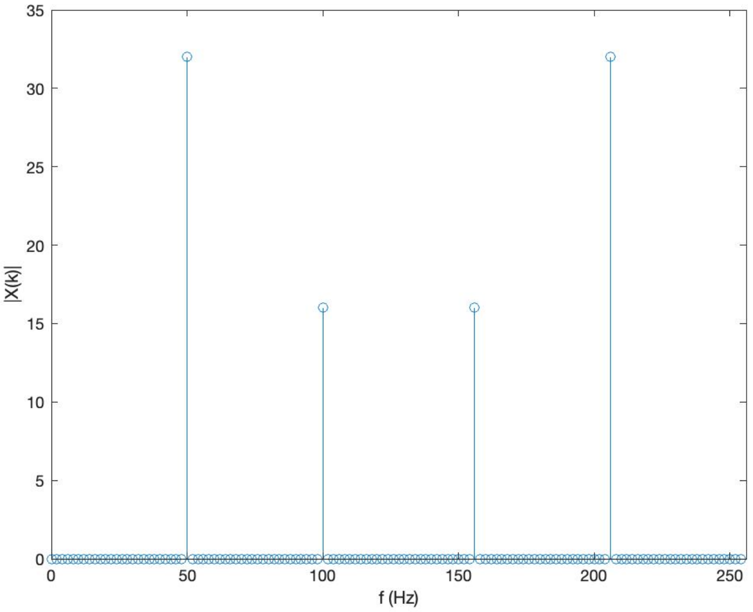


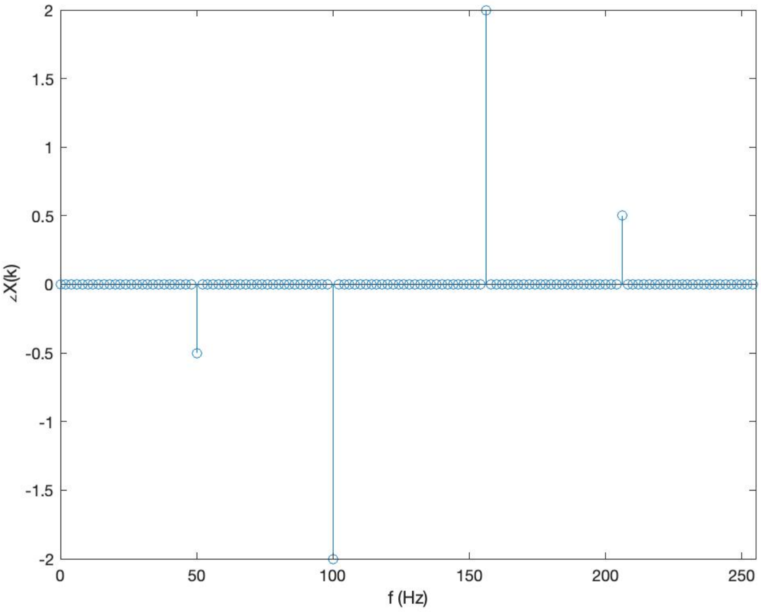
Figure 8 shows the magnitude spectrum of $x[n]$. Here we plot the magnitude of each X_k. Recall that the magnitude of a complex number z is defined as $\sqrt{(\text{Re}(z))^2 + (\text{Im}(z))^2}$. It is quite common to see plots of the magnitude spectrum using simply the magnitude of each X_k. To derive the amplitude of the original frequencies, we must multiply each value in the figure by $2/N$ to match Equation 6. Alternatively, some magnitude spectrum are plotted on a decibel scale. It is always important to check how the magnitude spectrum is plotted as there are several variations.

In Figure 8, we see four peaks in the graph. We can conclude that there are four frequency components 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 magnitude spectrum 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 plots the argument of the complex number X_k. Those familiar with complex numbers will note that the argument of a complex number is also calculated using Equation 7. Thus, the argument of each X_k is the same as the phase of the frequency component, assuming we model the frequency component as a cosine wave. As expected

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



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, and that the inner product is distributive. Therefore taking the inner product of some signal $x[n]$ with a sinusoid s is the same as taking the inner product of s 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.

In the real world, we will rarely, if ever, be that lucky. If we were to take a series of N samples from a song or a speech, it is very unlikely that the slice of audio 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 all the frequencies and their respective amplitudes and phases with complete precision for any signal. The DFT, however, can give us a good **estimate** of what frequencies are part of some signal.

For a cosine wave that completes exactly k periods, we were able to show precisely what the value of X_k would be as shown in Equation 5. Though we did not show the derivation for that equation, it is similarly based on the assumption that $x[n]$ is periodic along N. Therefore, we will not be able to use Equation 5 for aperiodic $x[n]$. As a consequence, we must also recognize that Equation 6 and Equation 7 will not be able to recover the original amplitudes and phases exactly as these are derived from Equation 5. 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.

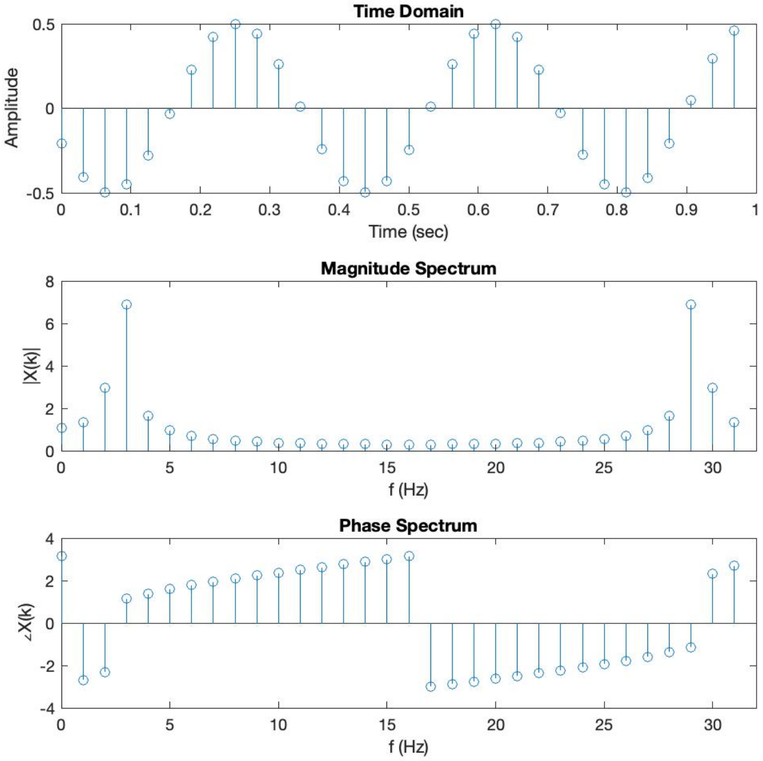
Some of our conclusions will remain the same regardless of the periodicity of $x[n]$. The symmetry we saw in the magnitude and phase spectrum still applies. We shall also see that we can perfectly recover the original $x[n]$ regardless of its periodicity using a technique called the Inverse Discrete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

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



Plotted in Figure 10 is the magnitude and phase spectra of the original signal as well as its time domain representation because 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 magnitude and phase spectra. The magnitude spectrum was calculated by taking the magnitude of each complex number. 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 spectrum due to the peak at that point. 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 spectrum at the frequencies of $x[n]$ provided those frequencies have sufficient amplitude. Notice though that each frequency bin registers some level of magnitude. Farther away from the peak, the magnitudes decrease but are still non-zero. This is called "spectral leakage". It is important to understand that we are not seeing multiple frequencies. Rather we are seeing the effect of how one aperiodic component of $x[n]$ spreads out to all frequency bins.

Can we recover the original amplitude of 0.5 from the magnitude spectrum? The answer, in short, is no. For one, we would need to guess about where the apex of the peak was. The peak is always centered at the frequency of each component of $x[n]$. It is hard to tell based on the magnitude spectrum though where exactly that is. We can use a process called interpolation to estimate the peak. Even so, each frequency component of $x[n]$ spreads to each adjacent bin. Therefore, even if we could determine the peak, the magnitude at that location could be the product of the spread of several different peaks. Therefore, it will be impossible to reconstruct the original amplitude. Nevertheless we can get a good sense of the relative strength of each component. Higher peaks represent stronger frequency components; lower peaks represent weaker frequency components.

How can we interpret the phase spectrum? Again we see a sudden change around 2Hz. The phase spectrum 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 spectrum of $x[n]$ around 2.7Hz and the original phase. In general, the phase spectrum of the DFT is more abstruse and difficult to parse. Sometimes sudden shifts in the phase spectrum 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 spectrum. We will not be spending much time examining the phase spectrum as it usually is less important for musical applications. We shall see soon though that there is a better way to interpret the phase spectrum.

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. Importantly, the DFT is parsing the frequencies of this signal and not the smoothly oscillating sinusoid from where the samples originated. 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 nearly all periodic signals can be represented as a sum of cosine waves where each cosine wave is a harmonic of the fundamental period N.[4] 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

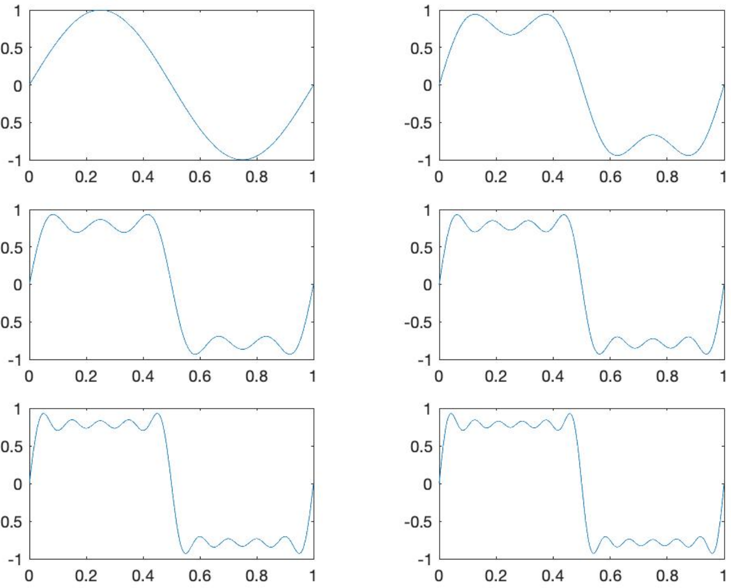
[4] I mention "nearly all" because such a signal must satisfy the Dirichlet conditions. But for all pratical purposes, any real audio signal will satisfy the Dirichlet conditions.

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 the period 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 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.

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

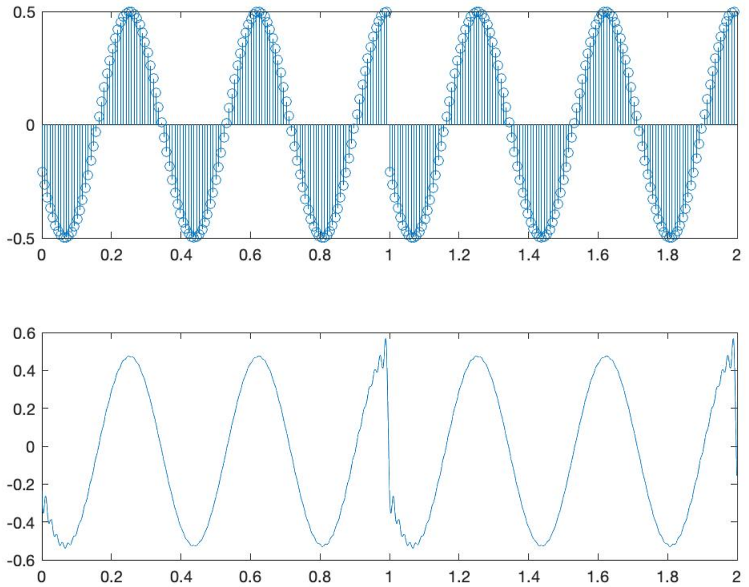


Let us look back at the example we used to discuss how the DFT reacted to an aperiodic sinusoid. In that example, we took a slice of $N = 32$ samples from $x[n] = 0.5\cos(2\pi(2.7)Tn + 2)$ that did not capture a complete period from $x[n]$. Nevertheless, the DFT still considered those N samples to be periodic. Again, the top plot in Figure 12 shows how the DFT perceives the periodicity of that waveform. Because that signal is periodic, we can represent it using a sum of harmonic cosine waves. The bottom plot from Figure 12 shows an approximation of that signal with a sum of harmonic cosine waves. How did I know which cosine waves to sum to achieve this approximation? From the DFT, of course! **Another interpretation of the DFT is that it encodes the information to recreate the original signal using a sum of cosine waves.**

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 approximation shown in the bottom plot from Figure 12.[5] 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

[5] Note that we will be using a normalized magnitude here where the magnitude of the complex number X_k is multiplied by a factor of $2/N$.

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



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 cosine waves also form a basis. It is 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 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.

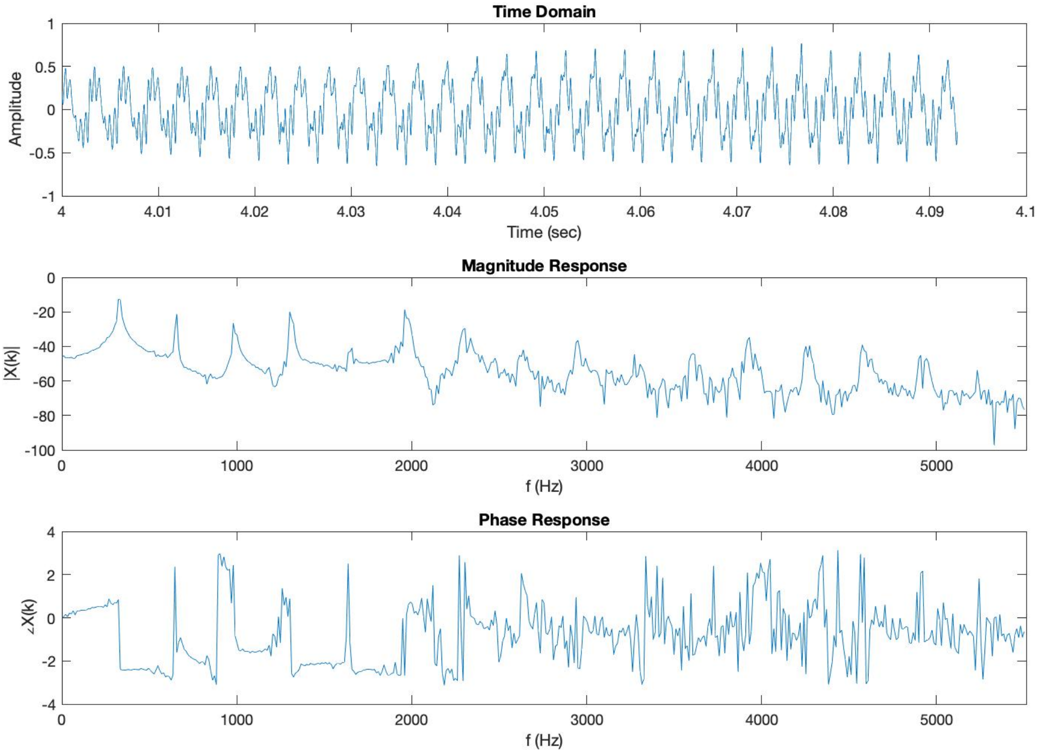
The DFT on Real-World Sounds

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 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

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



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.

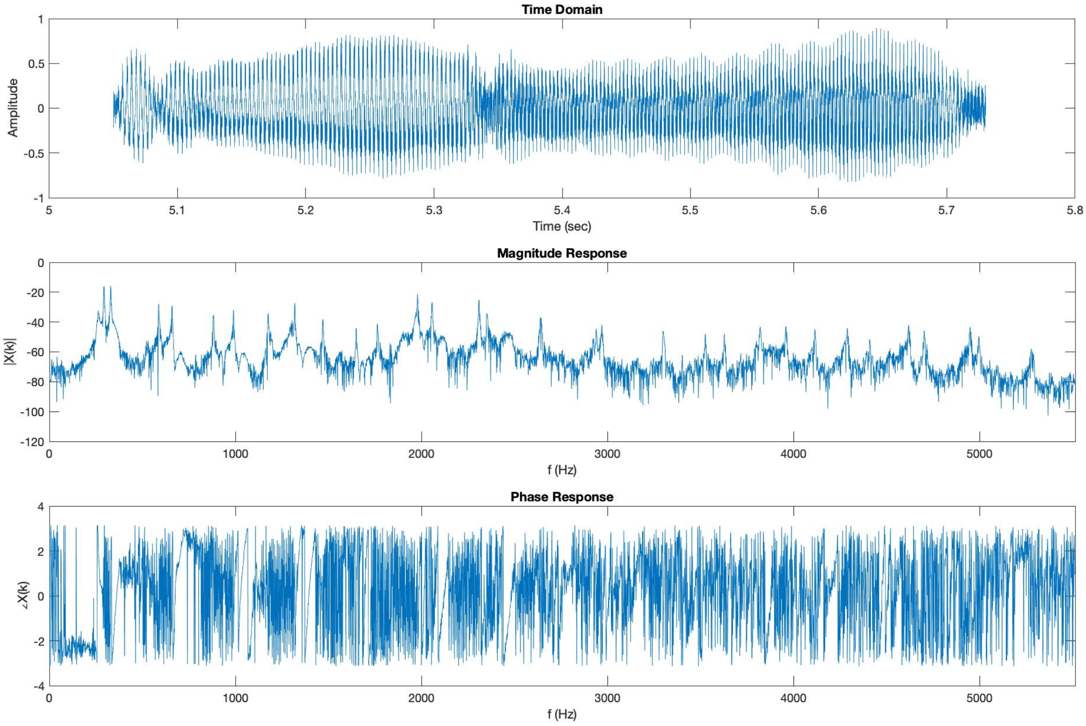
Time Resolution vs. Frequency Resolution

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 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.

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



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 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 8 shows the equation for the IDFT.

$$x[n] = \frac{1}{N}\sum_{k=0}^{N-1} X_k \cdot e^{i\frac{2\pi}{N}kn} \tag{8}$$

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.

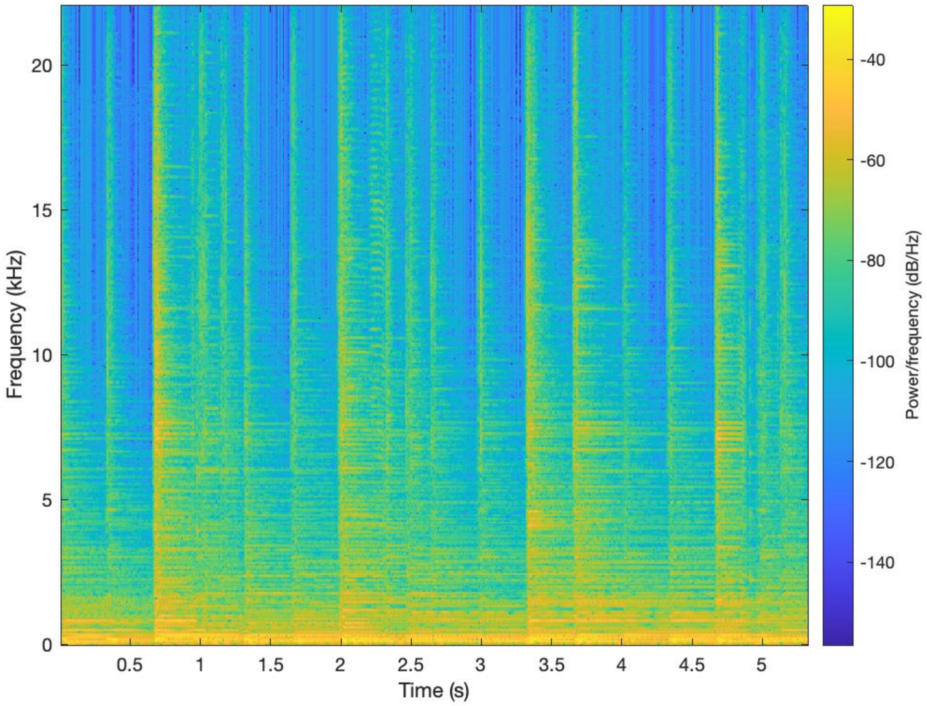
Unfortunately, when these two main conditions are not obeyed, the DFT cannot accurately parse the frequency components of $x[n]$. Nearly all $x[n]$ in practice violate these two conditions. Nevertheless we can look at peaks in the magnitude response to get a good estimate of where the original frequencies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