**Fourier Transform and Voice Recognition**

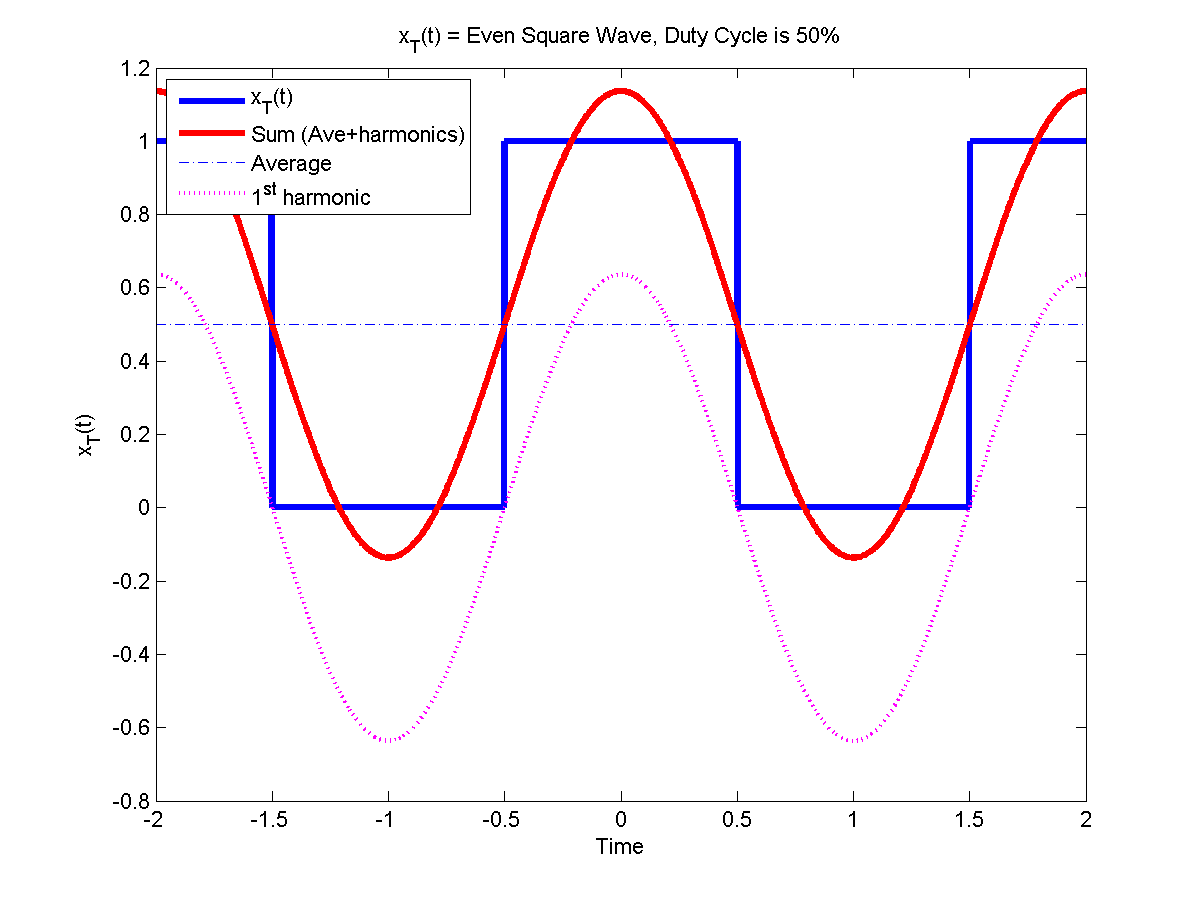
**Introduction**

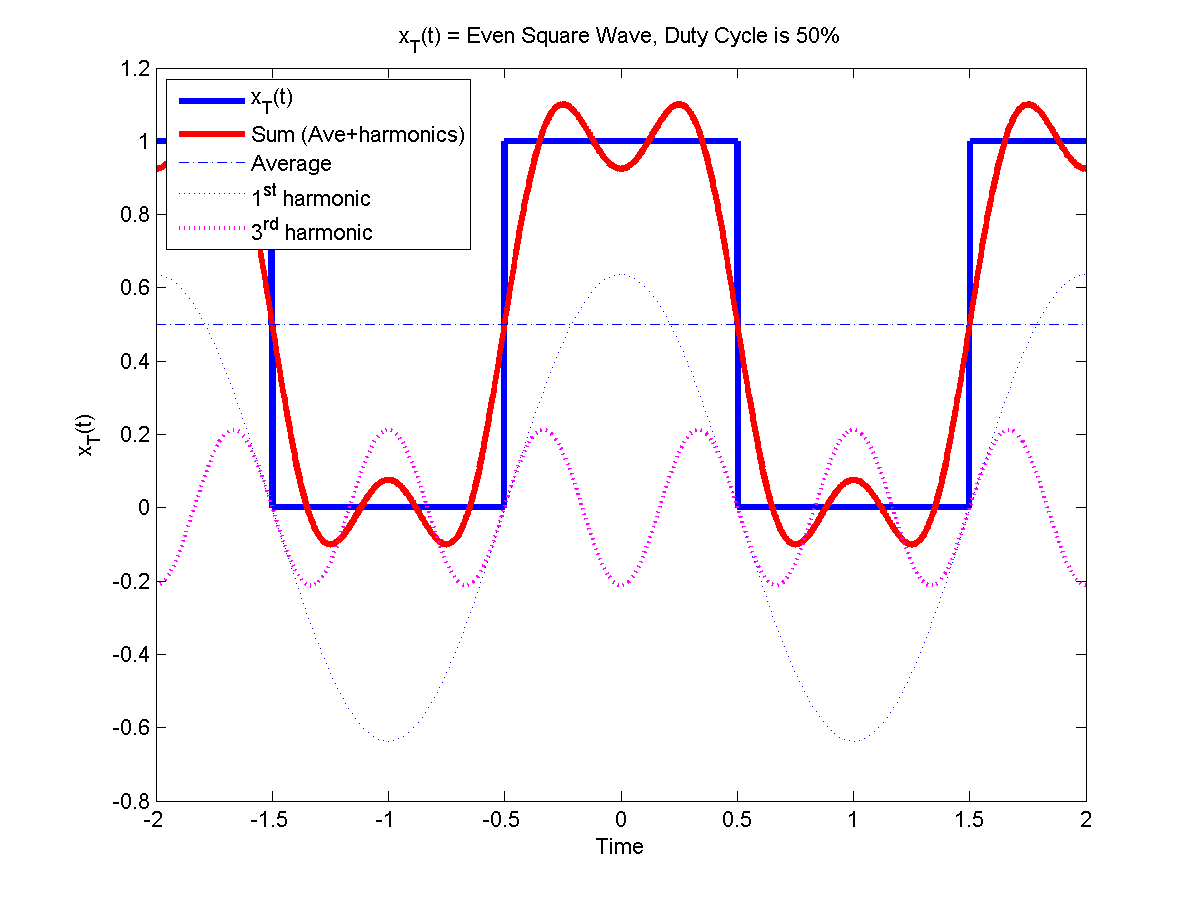
Voice recognition has long been an active field of research due to its great potential in human-machine interactions. Since the invention of computers, we have constantly aimed for better and more direct way of sending computers instructions. From originally having no interface, we have come to a long way and developed graphical user interface and more recently, touch user interface. Looking forward, voice user interface is the next major user interface to come, as it frees up people’s hands and eyes. In fact, people have implemented voice control function much earlier than the touch user interface. However, voice control struggled to become a stand-alone interface due to the difficulties in language processing and voice recognition. This paper is going to look into one of the ways of voice recognition—Fourier transform, examine its specific implementation in Matlab and discuss factors that influence recognition accuracy.

**Fourier Series**

The main idea of Fourier transform is to transform a function from its time domain to its frequency domain. In order to better understand this concept, I will first talk about Fourier series. Fourier series is an infinite linear combination of sine and cosine waves with different period. The expression is the following:

*f* (*t*) = *A*0 + *A*1cos(ω0*t*) + *B*1sin(ω0*t*) + *A*2cos(2ω0*t*) + *B*2sin(2ω0*t*) +···+ *Am*cos(*m*ω0*t*)+ *Bm*sin(*m*ω0*t*)

This series can simulate any periodic function, whether it is square wave or triangle wave. As it is shown in Figure 1,

1.  (b)

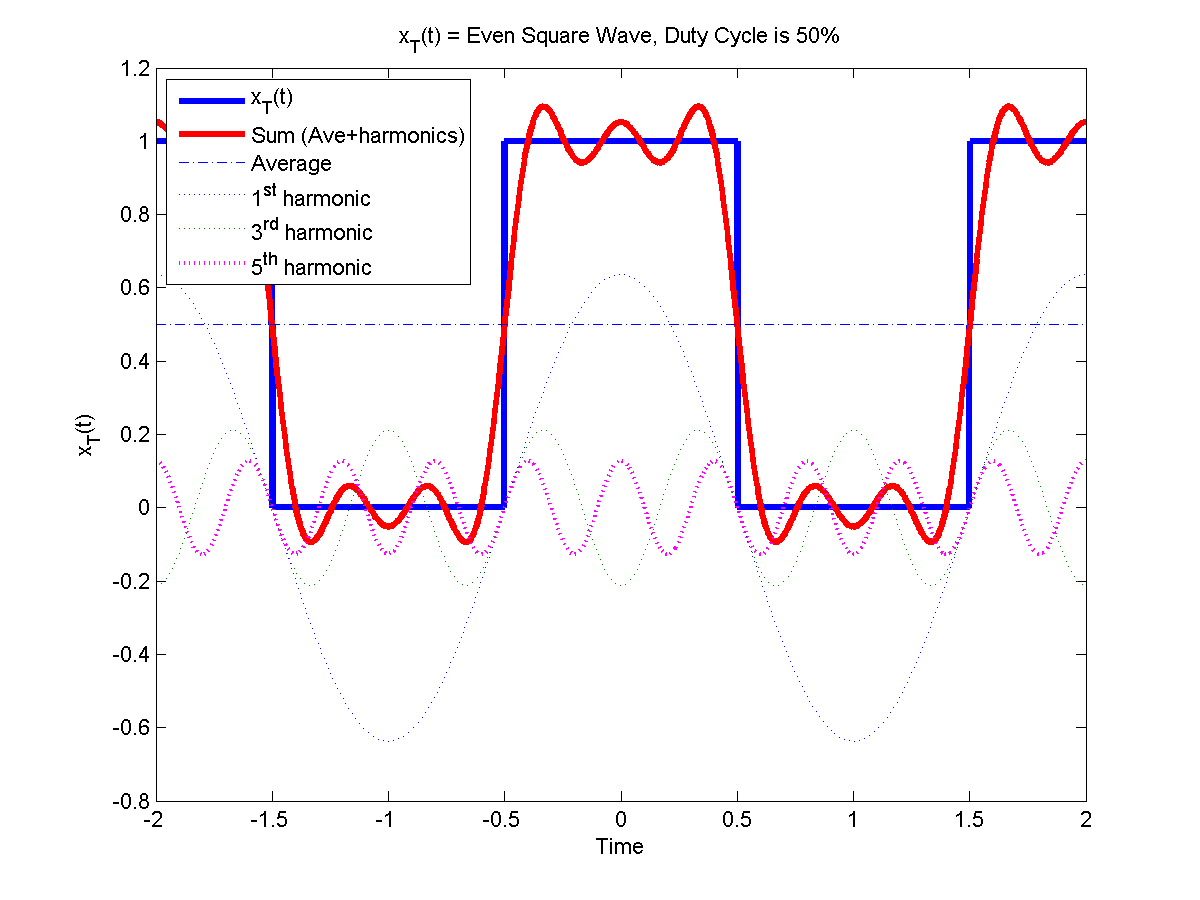
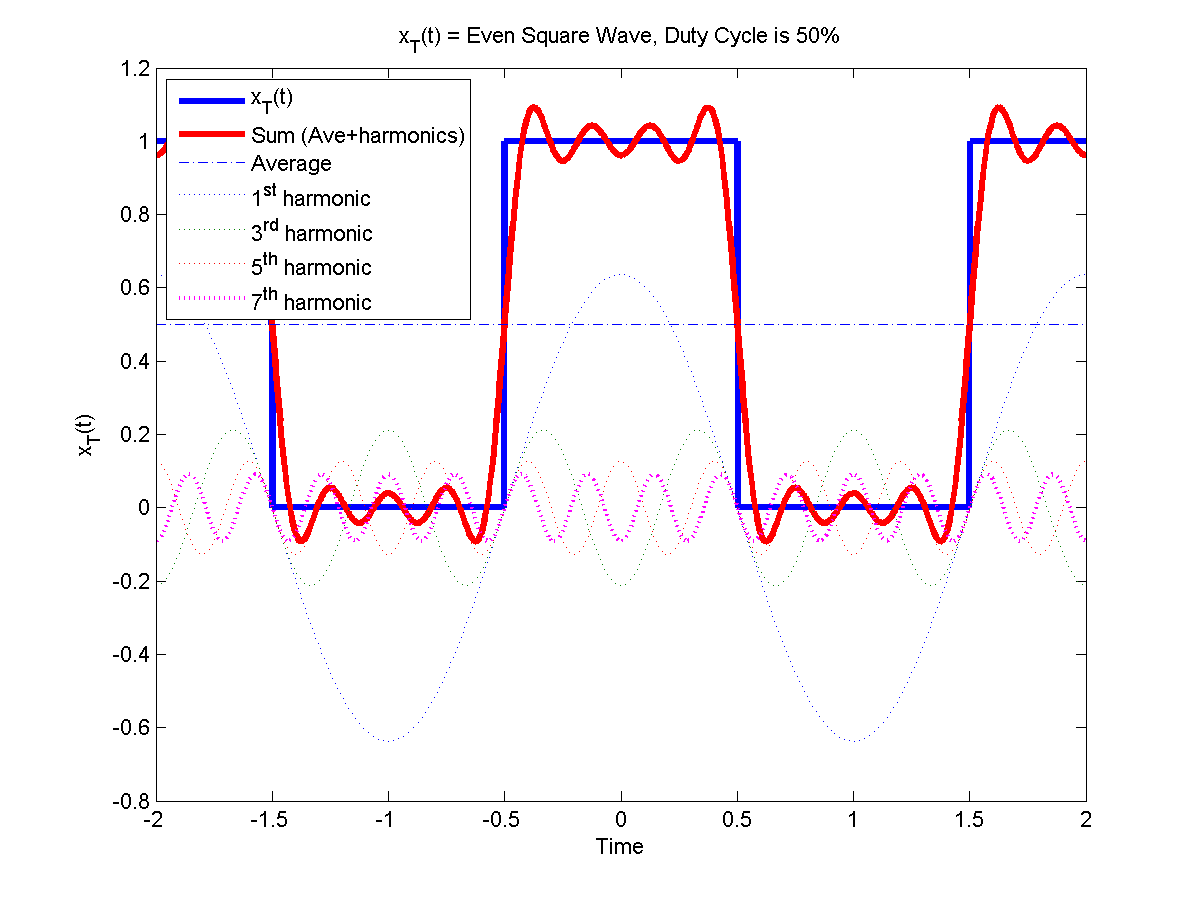
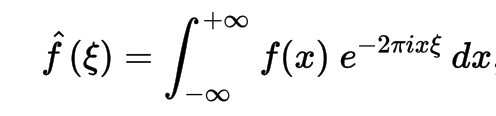
(c)  (d)

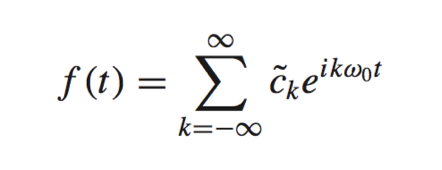
Figure 1. (a) square wave fitted by a1cos(ω0t), (b) square wave fitted by a1cos(ω0t) + a3cos(3ω0t), (c) square wave fitted by a1cos(ω0t) + a3cos(3ω0t) + a5cos(5ω0t), (d) square wave fitted by a1cos(ω0t) + a3cos(3ω0t) + a5cos(5ω0t) + a7cos(7ω0t)

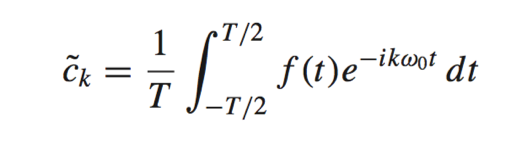
As we can see, the sum of cosine waves becomes closer to the square wave when more terms are added. Each term has different coefficient, which can be interpreted as the amplitude of each sinusoid wave. Any periodic functions can be decomposed to sinusoid waves with different frequency and amplitude, and this decomposition is Fourier series.

**From Fourier Series to Fourier Transform**

Fourier transform behaves similarly to Fourier series but decomposes aperiodic waves by allowing the period of sinusoid waves go to infinity. The formula of Fourier transform is the following:



The Fourier series can be written using Euler formula as the following:

And *Ck* is the integral:

If the aperiodic function has value 0 except in the interval [T/2, -T/2], the Fourier transform equation can be rewritten as

../../../../../../Downloads/CodeCogsEqn-2

And *Ck* can be expressed using Fourier transformation of that particular interval:

../../../../../../Downloads/CodeCogsEqn-3

Therefore, Fourier series and Fourier transform are two mathematically related concepts.

**Fourier Transform in Matlab**

Matlab has essentially two ways of computing Fourier transform. One is for continuous Fourier transform and one is for discrete Fourier transform. The function for the former is fourier(). Given a function f = exp(-3\*abs(x)) + 3\*sin(x^2+2), fourier(f) outputs the following result:

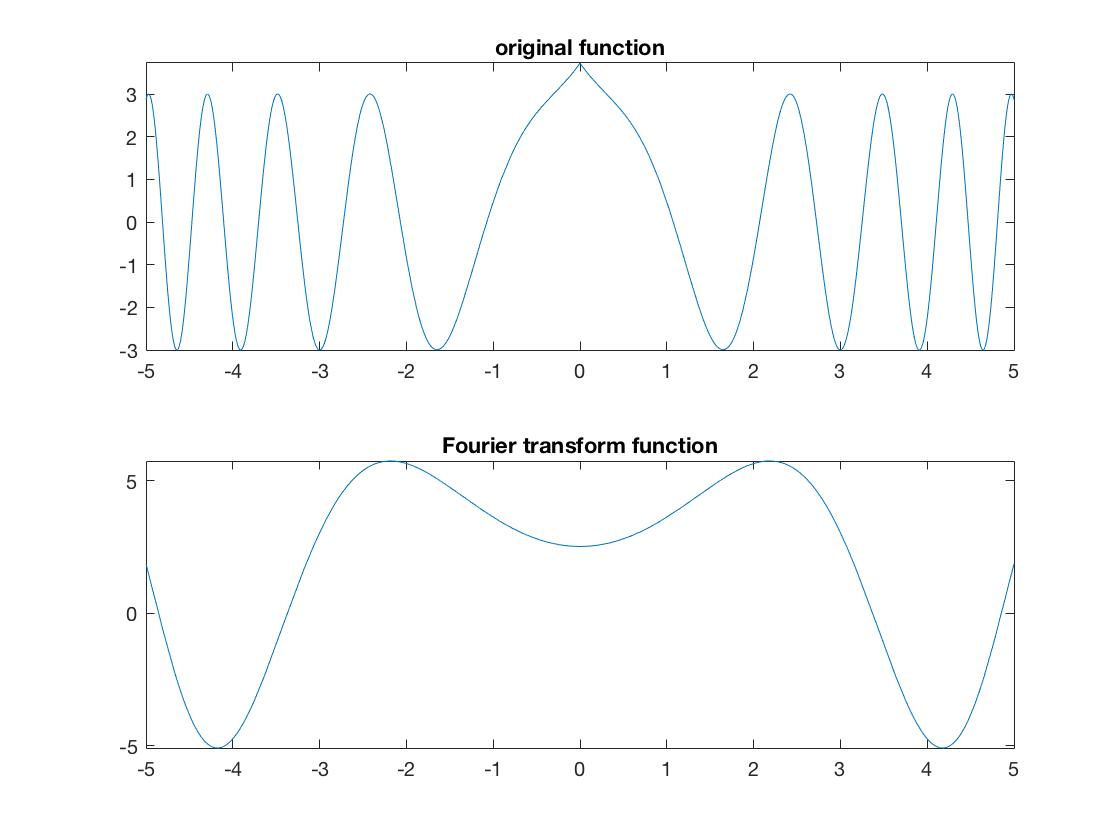


Figure2. The upper graph of the function is the original function and the lower graph is the Fourier transform

However, digital signals are all discrete and there is no way to know the function of a signal. Discrete Fourier Transform (DFT) becomes useful in this situation.

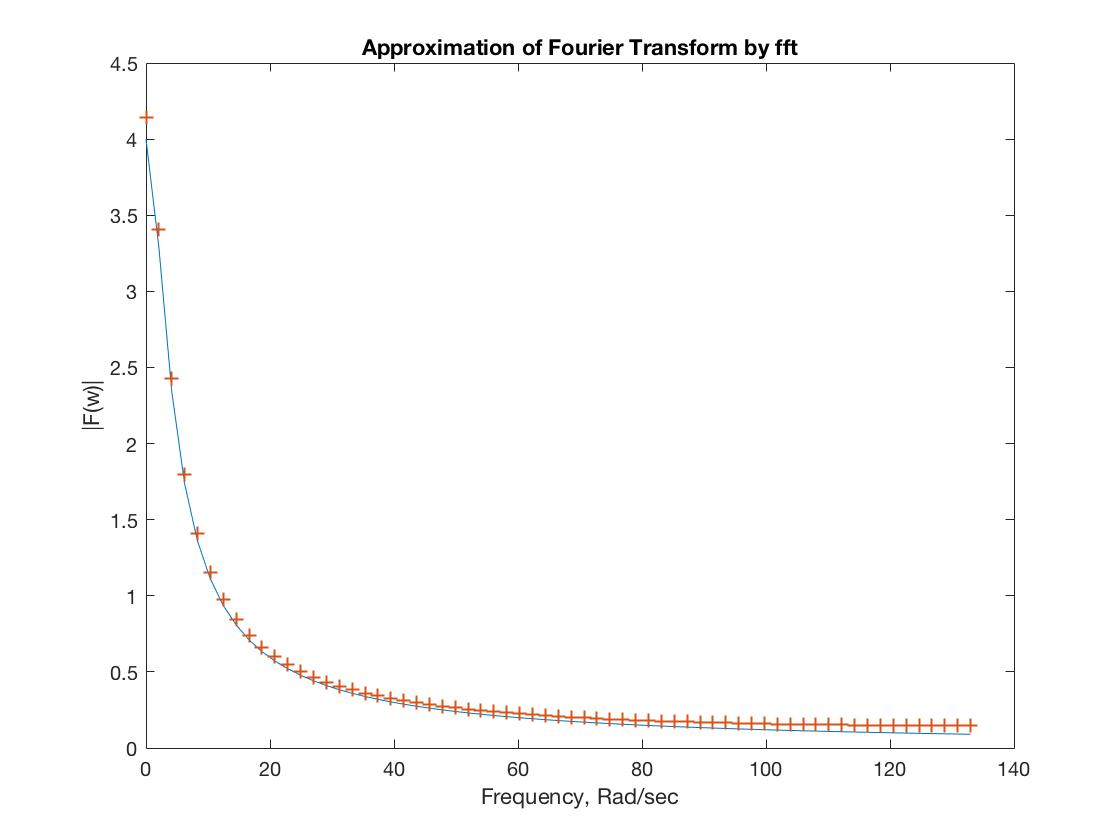
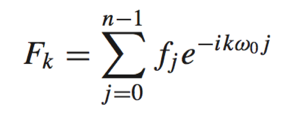


Figure 3. Approximating Fourier transform of f(t) = 12\*exp(-3\*t) by DFT. The blue line is the Fourier transform and the yellow dots represent the output of DFT function.

The function of DTF by definition is the following:



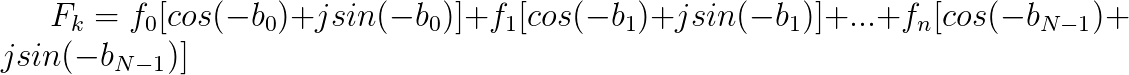
If we group ikw0 together and call it bn, the function becomes:

../../../../../../Downloads/CodeCogsEqn-6

Then we can expand the sum:

../../../../../../Downloads/CodeCogsEqn-7

To further expand the formula, we use Euler’s formula to expand the exponential:



And if we group all the cosine terms as A and sine terms as B, the function becomes:

../../../../../../Downloads/CodeCogsEqn

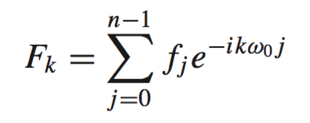
And this formula tells about the amplitude and phase about the kth frequency components.

The complexity of the DFT is O(n2), because for each *Fk* one needs to go through *n* terms. That’s in total (k x n) operations, giving the complexity of O(n2)

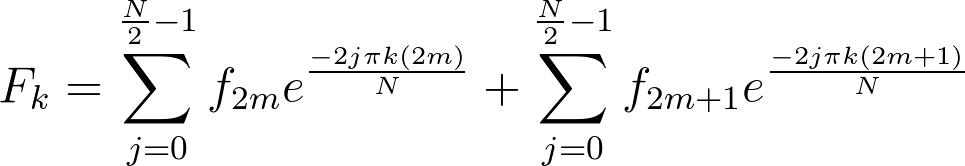
This extremely big complexity makes naïve DFT unpractical in real life applications, raising the need to have another way of computing DFT.

**Fast Fourier Transform**

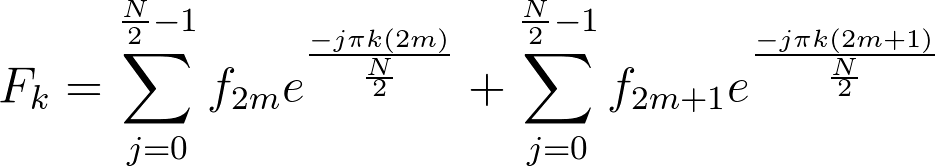
Fast Fourier transform uses divide and conquer to drastically reduce the complexity to O(nlog2n). Starting from the same equation for DFT:



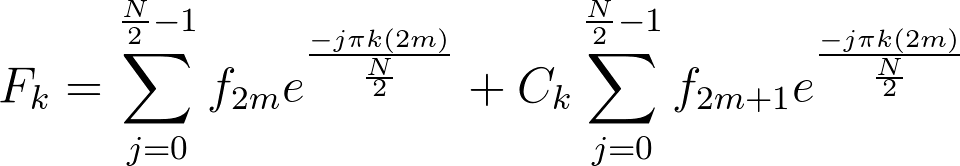
We can divide it into two terms, one for odd and one for even:



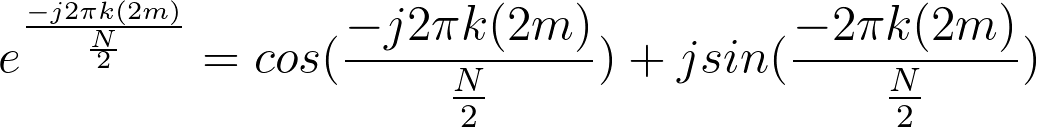
Move the 2 from the nominator of the exponential to the denominator we have:



To make the exponential terms of each part the same, we can factor out a constant Ck:



The exponential term can be expanded using Euler’s formula:



Because k ranges from 0 to N, the cosine and sine terms repeat themselves after k passing N/2 due to the symmetry of cosine and sine function. This property effectively reduces the operation by a half. N terms allow such division happen log­2N times and this gives us O(Nlog2N) complexity, making it a much faster algorithm than O(N2) naïve DFT.

**Fast Fourier Transform of periodic Signals**

Fast Fourier Transform decomposes seemingly complicated wave function in its time domain into frequency domain.

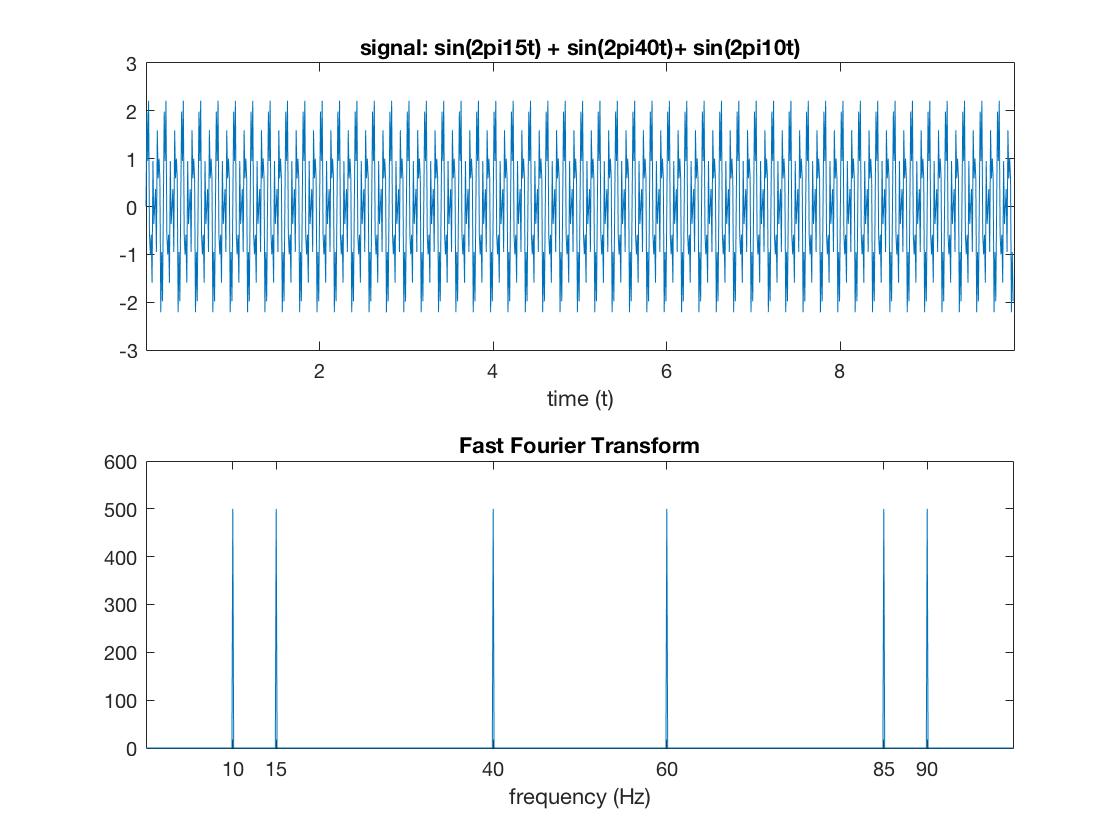
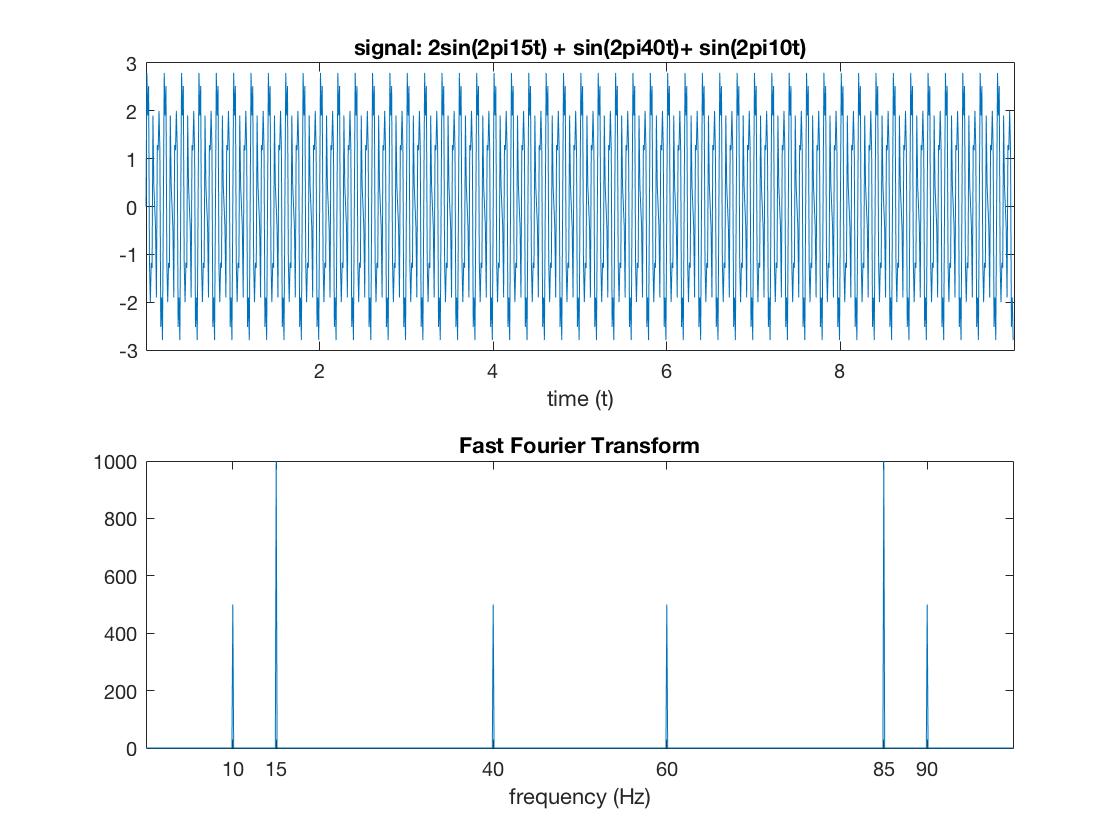
For example, the wave function ../../../../../../Downloads/CodeCogsEqn-13 has the following results:

Figure 4. The upper half is the graph of the signal and the lower half is the graph of Fourier transform

The Fourier transform accurately extracts the three distinct frequencies: 10 Hz, 15 Hz and 40 Hz. And all three peaks are of the same height because the three sine waves have the same magnitude.

If we add some coefficient before the sine wave, the result of FFT will give peaks of different height (Figure 5.)



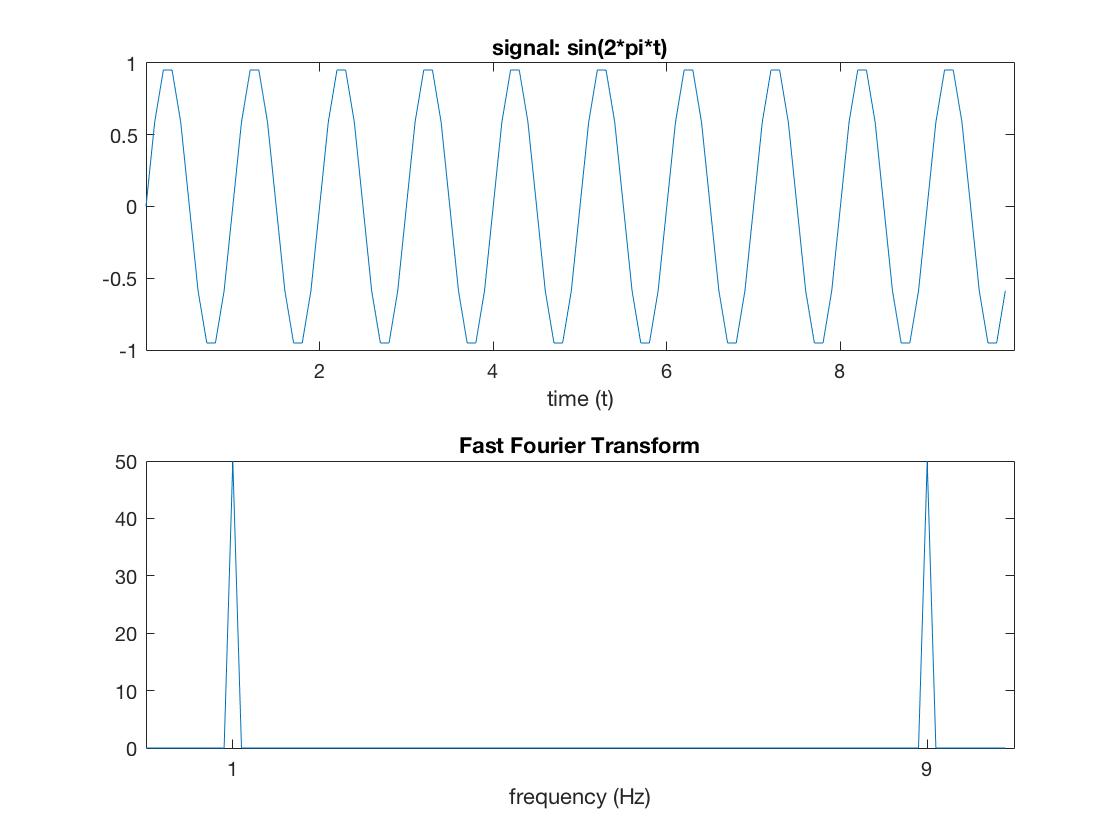
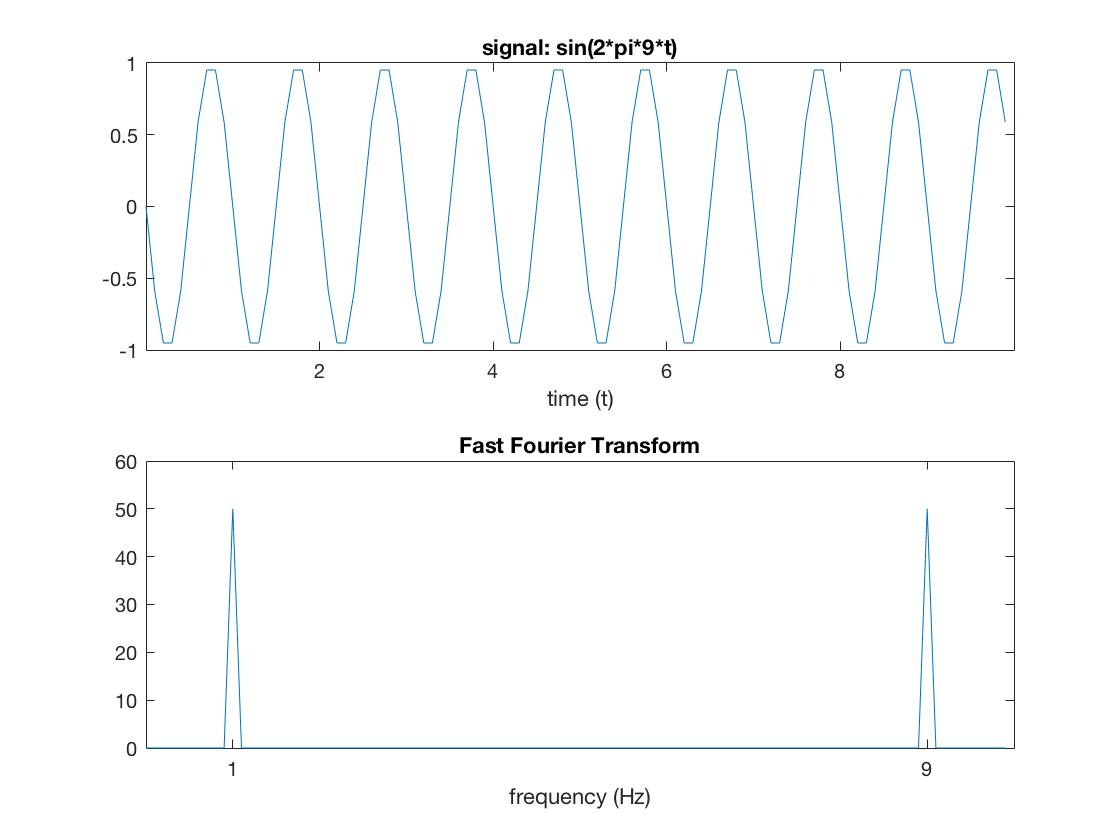
../../../../../../Downloads/CodeCogsEqn-14Figure 5. The signal wave is and the peak on 15 Hz and 85 Hz are exactly twice as much as the other peaks.

**Nyquist Frequency and Aliasing**

One thing worth pointing out from the above two figures (Figure 4 & 5) is that each sine wave has two corresponding frequency peaks, which is counter intuitive. This is a result of aliasing. It means that two or more wave functions could yield the same result under one sampling frequency.

For example, the wave sin(2πt) and sin(2π9t) give the same signal when the sampling frequency is 10 Hz (Figure 6).

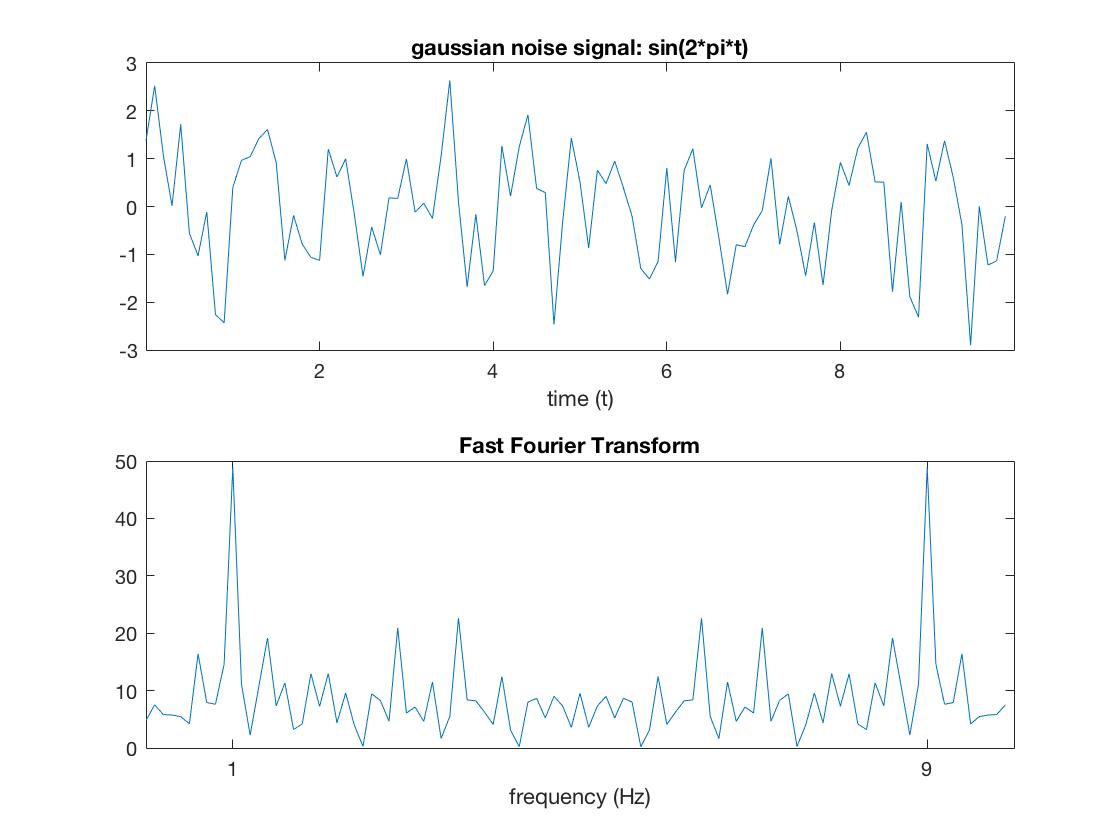
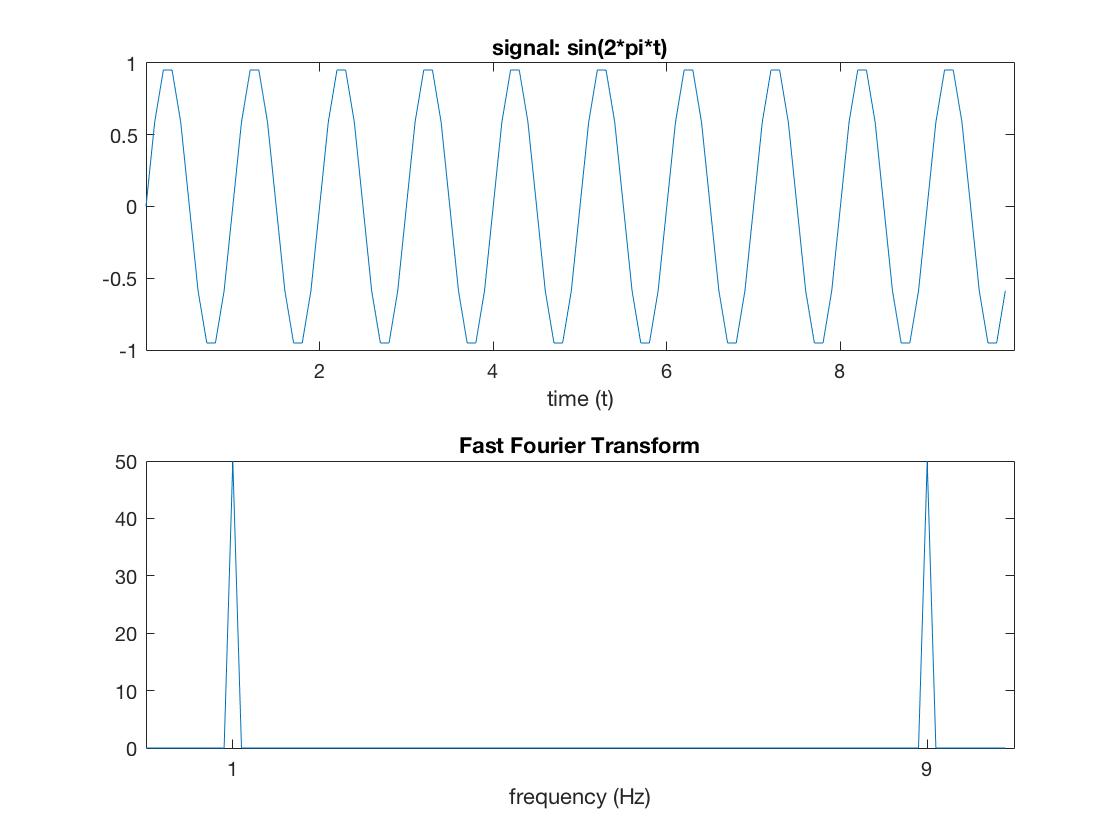
The maximum frequency a 10 Hz sampling could effectively capture is 5 Hz and that’s called Nyquist Frequency. Any frequency x Hz above 5 Hz will look the same as the (10 - x) Hz. Therefore, when calculating the magnitude of frequencies yielded by the FFT, one needs to count the magnitude of the aliasing frequency, which is the same as the real frequency. Thus, the real magnitude is always twice what’s yielded by the Fast Fourier transform.

Figure 6. Example of aliasing. The signal wave and FFT result of sin(2πt) and sin(2π9t) appear the same on the figure but are different in reality.

**Fast Fourier Transform of periodic Signals with noise**

The Fourier Transform is useful for signal processing because it’s robust to noise and good at extracting the feature of the sound.

For example, when adding Gaussian noise to sin(2πt), the original wave becomes unidentifiable (Figure 7). The periodicity and amplitude change drastically. However, looking at the Fourier Transform diagram, the two frequency peak at 1 Hz and 9 Hz are salient.



(a) (b)

Figure 7. (a) sin(2πt) with Gaussian noise added (b) original sin(2πt) wave.

To prove that Fourier Transform is resistant to noise in general, uniformly distributed noise is added. And the result appears to be more robust than when Gaussian noise is added, because the Fourier Transform peaks are still visible when the noise amplitude is twice the signal amplitude (Figure 8).

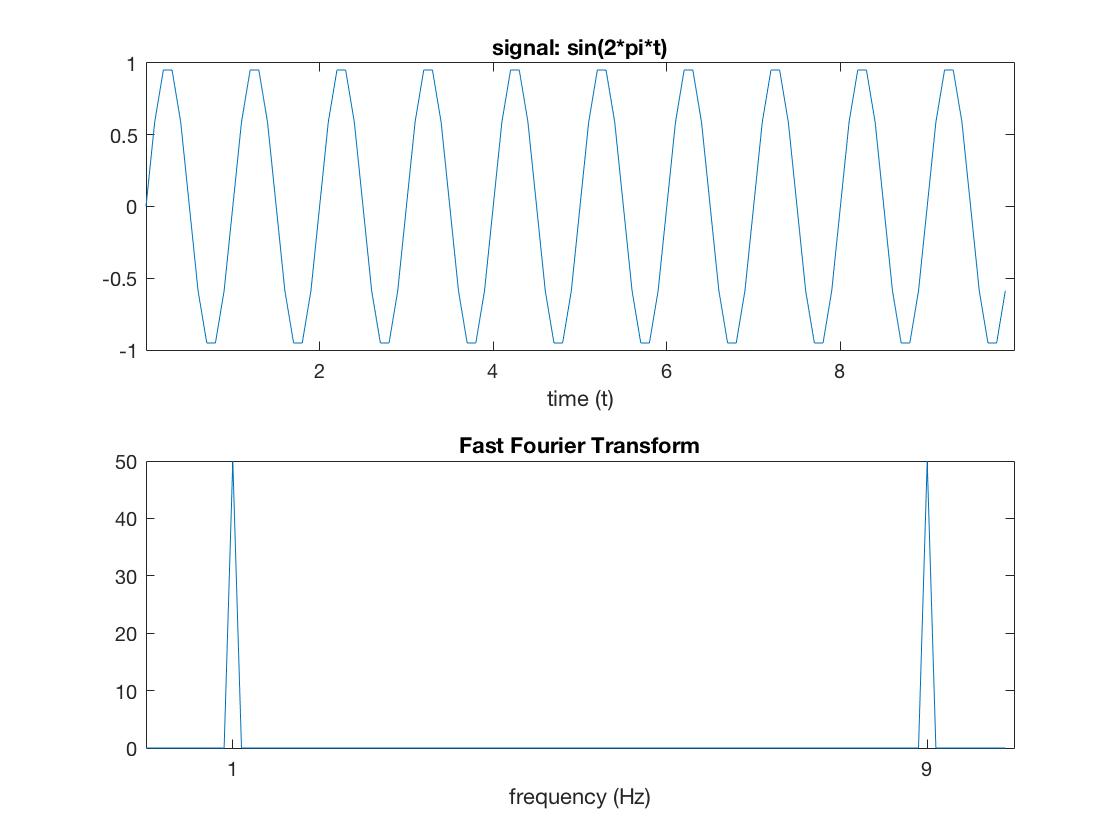
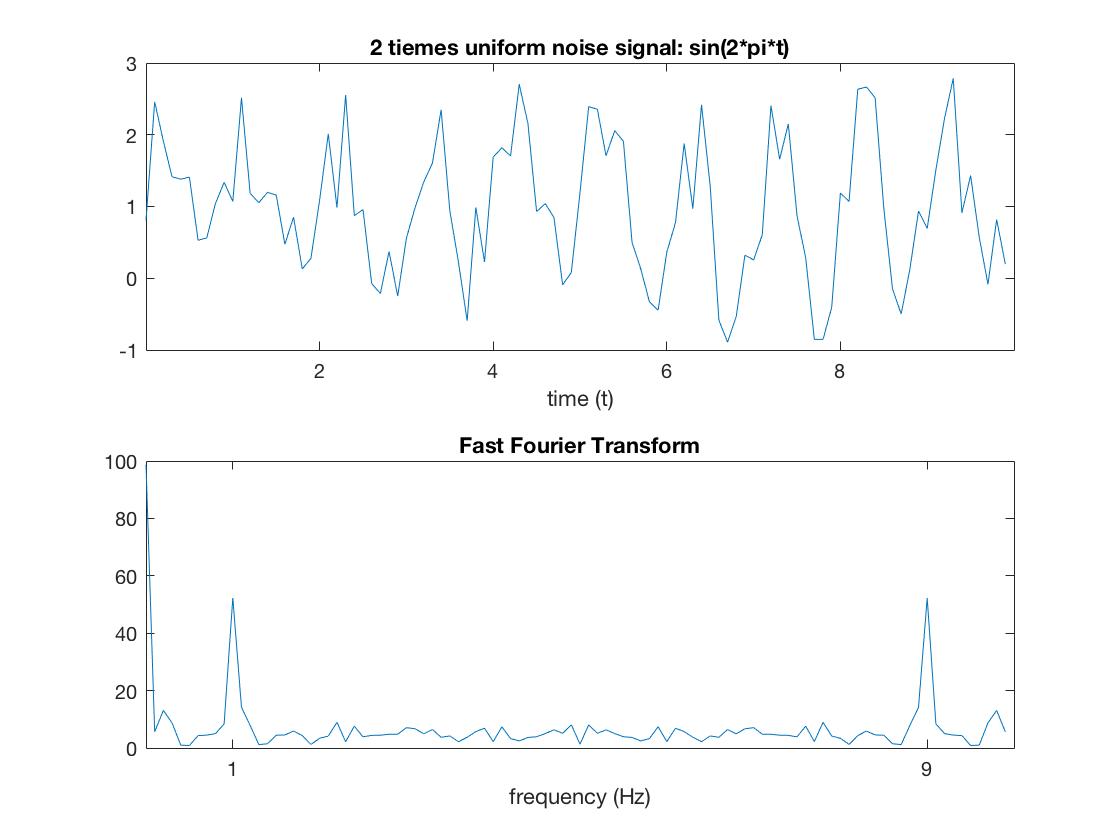


Figure 8. (a) sin(2πt) with 2x uniform noise added (b) original sin(2πt) wave.

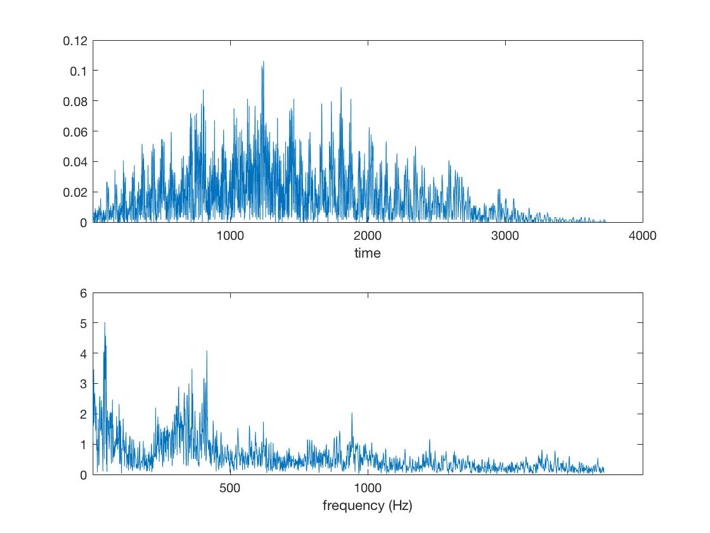
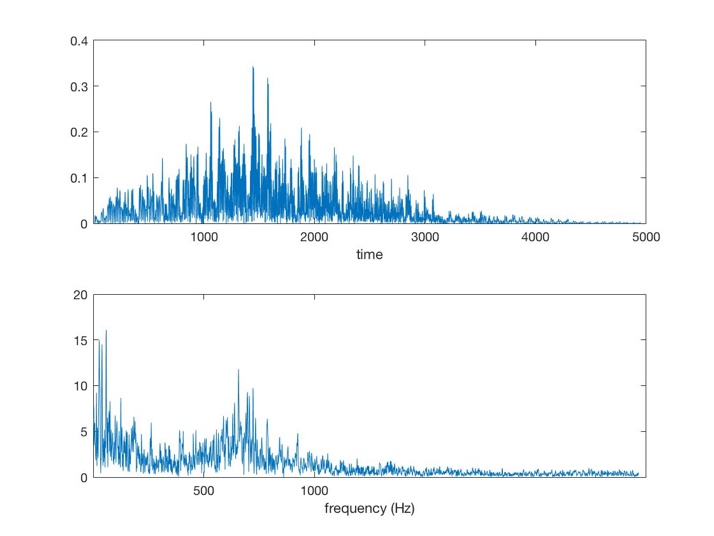
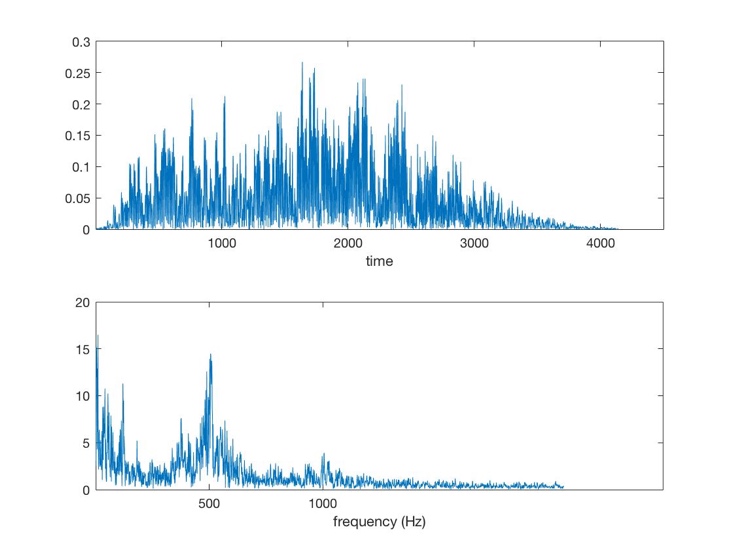
**Voice Recognition with FFT**

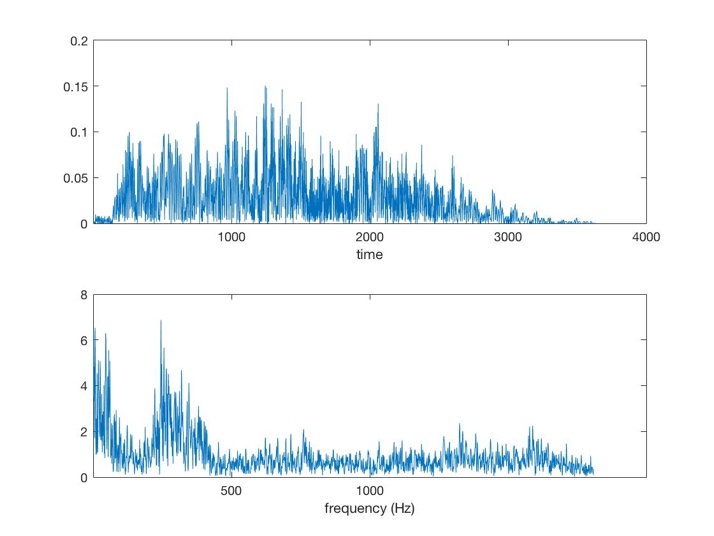
We have so far demonstrated that Fourier transform is capable of decomposing a wave from the time domain into the frequency domain, and is robust to background noise. Since all digital signals are discrete, we need to use DFT to conduct conduct Fourier transform on the signals. We proved that the DFT has a high fidelity to continuous Fourier transform. However, the problem of normal DFT is that it has complexity of O(n2), which is unpractical to implement. FFT, as a fast algorithm of DFT, solves the problem by lowering the complexity to O(nlog2n).

Because FFT has above characteristics, it can be used to differentiate different voices. Figure 9 shows the spectrum of the sound wave and FFT of pronunciation from 1 to 4. Although the wave in the time domain look quite similar, the FFT of each word is different. They all have two major peaks in the frequency map, but they occur at different frequencies. One has the second peak around 700 Hz; two has the second peak around 400 Hz; three has the second peak around 250 Hz and four has the second peak around 500 Hz.

**Comparing FFT spectra**

The sound of words is compared with every pronunciation in the database. Let’s denote the input signal X and the voice in the database Y. ifft(fft(X) \* conj(fft(Y)) ) is calculated as an evaluation of the similarity between two spectrum. The pair with the highest score is output as the recognized word.

 (a) (b)



(c) (d)

Figure 9. (a) sound wave and FFT of “one” (b) sound wave and FFT of “two” (c) sound wave and FFT of “three” (d) “four”

**Discussion**

Voice recognition by FFT alone struggles to reach a high accuracy in practice, especially when the database is big. The underlying reason might be that human voice is confined in a relatively small range, which results in similar FFT spectra of different pronunciation. More measure needs to be implemented to reach a better accuracy in voice recognition.