Lab 9: An Introduction to Using the FFT Function in MATLAB

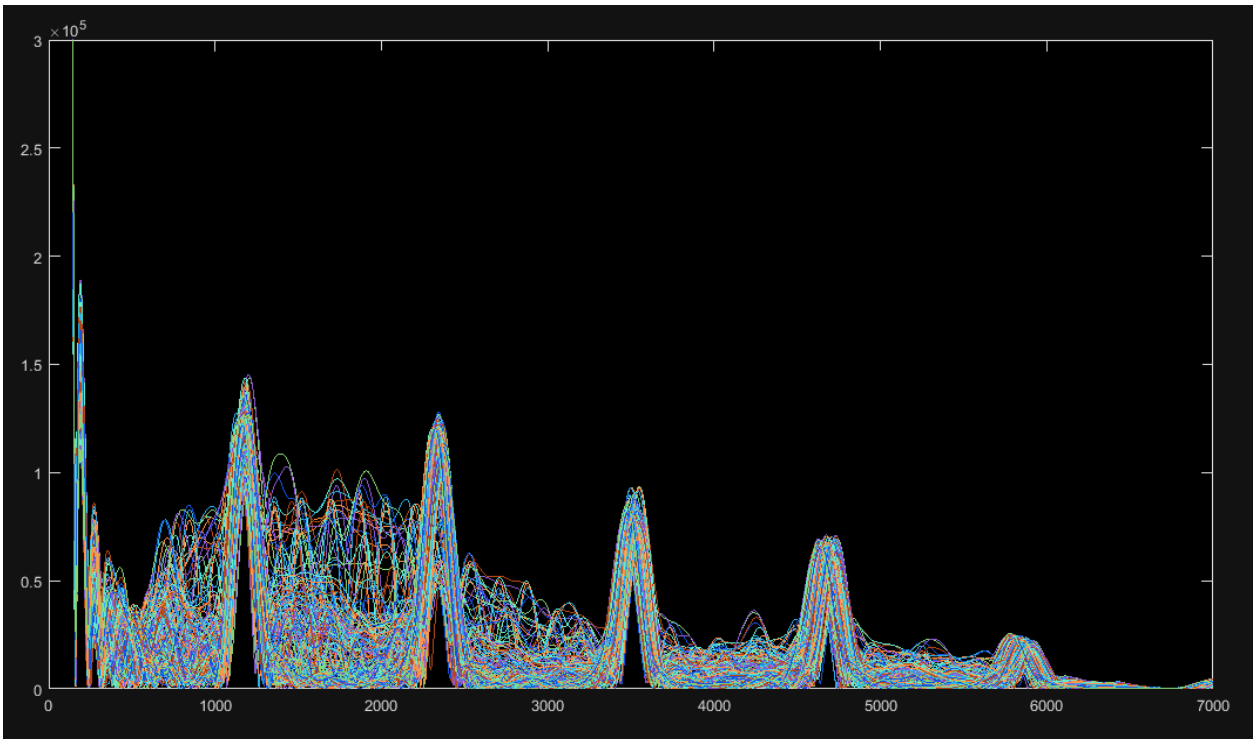


Figure 1 – This image depicts the overlay of hundreds of FFT plots of heart-rate samples, taken 1 second at a time. The spikes indicate the frequency components of the heart rates signal, where the first harmonic (at around the 1000th index) stores information about the average heart rate in beats/minute. Our ultimate goal in this lab will be to apply MATLAB's FFT function to analyze the average heart rate of provided ECG signals.

PART 1

The goal of this lab is to understand the use of MATLAB's implementation of the Fast Fourier Transform (FFT) and to apply it in order to find the heart rate in some real-world electrocardiogram (ECG) signals.

In this pre-lab, you will be introduced to the concept of the FFT. A basic definition is provided, as well as some general background and terminology that will be used throughout this lab. It is important that you watch the lecture videos that are provided in section 1.3.

1.1 Definition: What is the FFT?

FFT is an acronym for the *Fast Fourier Transform*, an algorithm that computes the discrete Fourier transform (DFT) of a discrete-time signal. In other words, the FFT takes a *discrete-time* signal and extracts important *frequency domain* information from that signal. It is considered

"fast" because its computational complexity is far less than that of the DFT, the Discrete Fourier Transform. In this lab, we won't care about the mechanics of the algorithm (hurrah!); we are simply concerned with how to use the FFT function in MATLAB to determine frequency domain information of signals.

1.2 Background: How does the FFT "Read" Frequency Information?

Before starting the lab, it may be useful to gain some insight as to how the discrete Fourier transform extracts frequency information from a discrete input signal. The FFT utilizes a series of *sinusoidal basis signals* of varying frequency. The FFT determines how closely the input signal "matches" these basis signals, and outputs a vector of complex numbers - one for each iteration of the algorithm. In this lab, we will call these complex measurements *frequency bins*, or simply *bins*.

Because the outputs of the FFT are complex, we need an easier way to visualize them. Simply plotting the vector of frequency bins will result in a complicated jumble of vectors in the complex plane. Therefore, we "read" the results of the FFT by plotting the *magnitude spectrum* of the frequency bins. The higher the magnitude of the frequency bin, the higher the *correlation* between the input signal and a sinusoid of that frequency. Therefore, the magnitude spectrum should spike at regions where the frequency of the sinusoidal basis signals match the input signal. Recall that any signal can be expressed as the sum of sinusoids of different frequencies. Consequently, the FFT is an extremely powerful tool because it can help us decompose a complicated discrete-time signal into its sinusoidal components.

1.3 Useful Lecture Series:

For more information on the DFT and MATLAB FFT function, refer to the following series of lecture videos by David Dorran. They are very useful for gaining an intuitive understanding of the DFT and FFT, and will prove useful in completing this lab.

[1] **Using MATLAB's FFT Function**(17:40)

https://www.youtube.com/watch?v=dM1y6ZfQkDU

[2] **Using MATLAB's FFT Function (2) – Zero Padding and Windowing**(23:06)

https://www.youtube.com/watch?v=V_dxWuWw8yM

[3] **How the Discrete Fourier Transform (DFT) Works - An Overview**(4:23)

https://www.youtube.com/watch?v=h6QJLx22zrE

1.4 Exercise: Decomposing Signals into Sinusoids

Rewrite the following signals, by hand, as a sum of sinusoids (that is, expand them as much as possible). Identify the **amplitude, frequency, and phase** of each sinusoidal component. In your write-up, provide your answers with explanations or scans of your work. (*Note:* this should be a trivial task that serves essentially as a review of the beginning of the course)

$$x_1(t) = 10\sin(2\pi(100)t + \pi)\sin(2\pi(100)t - \pi)$$

$$x_2(t) = \cos(2\pi(30)t + 0.25\pi)\cos\left(2\pi(20)t - (1/3)\pi\right)$$

1.5 FFT of a Single Sinusoid with Integer Frequency

In this section, we will use MATLAB's FFT function to analyze basic signals – ones that can be expressed mathematically. We will then apply what we know from this warmup to apply the FFT to a real-world example, where the frequency components of the signals (ECG heart rate data) are unknown.

To start, we'll use the FFT function for a discrete-time signal that is a single sinusoid. This way it will be clearer as to what the FFT is really doing. Let's consider the sinusoid defined below:

$$x(t) = \cos(2\pi(50)t + 0.25\pi)$$

Plot the continuous-time frequency spectrum of this signal by hand.

We can see that the frequency is 50 Hz and the phase is π/4 radians. When we run the FFT algorithm, we should expect to see the plot of the magnitude spike at the index corresponding to the frequency 50 Hz. There should only be one peak, since the signal is composed of a single sinusoid.

1) Define the variable `fs` for the sampling frequency, and assign it 1000 samples/second.
2) Construct a time vector using this sampling frequency that is exactly 1500 samples long, *starting at zero, not at 1/fs*.
3) Define $x(t)$, the sinusoid described above, as the vector `xx`.

4) **Run the FFT function on xx.** The function `fft()` only takes one argument: the input signal `xx`. We will use a semicolon to suppress the output (otherwise the command window will fill with complex numbers). By convention, frequency domain variables are denoted with capital letters, so we will use `XX` to store the outputs.

```
>> XX = fft(xx);
```

5) You should notice that the output `XX` is defined as a "*1x1500 complex double*." This is again because the outputs of the FFT are complex numbers. The output, in the frequency domain, is divided into a number of complex amplitudes in frequency 'bins'; the number of frequency bins is equal to the number of samples. In this case, there are 1500 complex amplitudes, in 1500 frequency bins, because there are 1500 samples in our signal. To make sense of the outputs, we need to take the magnitudes and phases of the complex amplitudes in this complex vector.

a. **Create a new vector that stores the values of `abs(XX)`** and **plot it** with the `plot(abs(XX))` command. This plot is called the *magnitude spectrum*, where the x-values are numbered frequency bins and the y-values are their corresponding magnitudes.

b. Compare this plot to your continuous-time spectrum. **What's different about it?**

c. In the plot of the magnitude spectrum, you should notice two congruent peaks on either side. For the signals we will be dealing with in this lab (real signals), the magnitude spectrum will be two sided. That is, any peak will be a mirror image of the peak on the other side. Because of this redundancy, we can simply evaluate the peak on the left.

   What is the vector index corresponding to the left peak? Hint: use the `find` function

   What is the value of the magnitude spectrum everywhere else?

d. The result of the Fourier transform of a sampled signal goes into frequency bins that correspond to the normalized radian frequency, ˆω. Our convention thus far has been to draw magnitude spectra from –pi to pi. MATLAB's FFT function, however, returns frequency bins ranging from 0 to 2pi.
   Plot the MATLAB FFT output against normalized radian frequency. Here's the command for the normalized radian frequency output:

```
ww = 0:(2*pi/length(XX)):(2*pi-1/length(XX));
plot(ww,abs(XX));
```

e. MATLAB's `fftshift` function is a way to "fix" the output of the MATLAB FFT function in order to make it range from –pi to pi. **Plot the shifted MATLAB FFT output against normalized radian frequency**:

```
ww = -pi:(2*pi/length(XX)):(pi-1/length(XX));
plot(ww,abs(fftshift(XX)));
```

f. The normalized radian frequency is related to the frequency in Hz by the sampling frequency and a factor of 2pi. **Plot the FFT of XX against the Hertz frequencies of the bins. What are the frequencies matching the peaks?**

6) The FFT can also be used to extract *phase* information from the signal. From the complex amplitude of the MATLAB FFT function, you can use the `angle()` or `phase()` function to find the phase in each bin. **Plot the phase spectrum of the signal against Hertz frequency. What are the phases of the peaks? Does this match what you would expect, and why?**

7) In parts (5) and (6), we identified the frequency and phase of $x(t)$. It turns out that we can also use the FFT to determine the exact magnitude of $x(t)$, by finding the value of the magnitude spectrum in that frequency bin. **Determine the exact magnitude of the peak.** Note that the two peaks will have the same magnitude, being complex conjugates of each other.

1.6 FFT of Multiple Sinusoids with Integer Frequency

Now we will use the FFT function to analyze more complicated signals. In part 1.4, we were given the functions:

$$x_1(t) = 10\sin(2\pi(100)t + \pi)\sin(2\pi(100)t - \pi)$$

$$x_2(t) = \cos(2\pi(30)t + 0.25\pi)\cos(2\pi(20)t - (1/3)\pi)$$

These are clearly not sinusoids, but can be decomposed into a sum of finitely many sinusoids. MATLAB's FFT function will be useful in determining whether your work from section 1.4 was correct. Complete the following steps:

1) Using the same sampling frequency and duration from part 1.5, **create** the vectors `x1` and `x2`, storing 1500 samples of the signals $x_1(t)$ and $x_2(t)$.
2) **Plot the *magnitude spectra* of `x1` and `x2` against Hertzian frequency**. Unlike in 1.5, we should observe multiple peaks, each corresponding to a single sinusoidal component. Comment on the appearance of each spectrum. **Do you notice any differences in the number of peaks for these signals?**

3) **Using MATLAB, find the frequencies, amplitudes, and phases of the sinusoidal components of x1 and x2. Compare** these results to what you found by hand.

Lab 9 PART 2

2.1 FFT of a Sinusoid with Non-Integer Frequency

In the previous sections, we dealt with the FFT of sinusoids and signals composed of sinusoids with integer frequencies. In other words, these sinusoids exhibit an integer number of cycles per period. In this section, we will be analyzing the magnitude spectrum of the sinusoid:

$$x(t) = \cos(2\pi(50.5)t + 0.25\pi)$$

Notice that this sinusoid is the same as that of an earlier section, only the frequency in Hz has changed from 50 Hz to 50.5 Hz. We want to see how the appearance of the magnitude spectrum is influenced by taking the FFT of a sinusoid with non-integer frequency.

1) Using 1000 samples/second for `fs` and a time vector of 1.5 seconds, **construct** the vector `xx` to represent the given sinusoid.

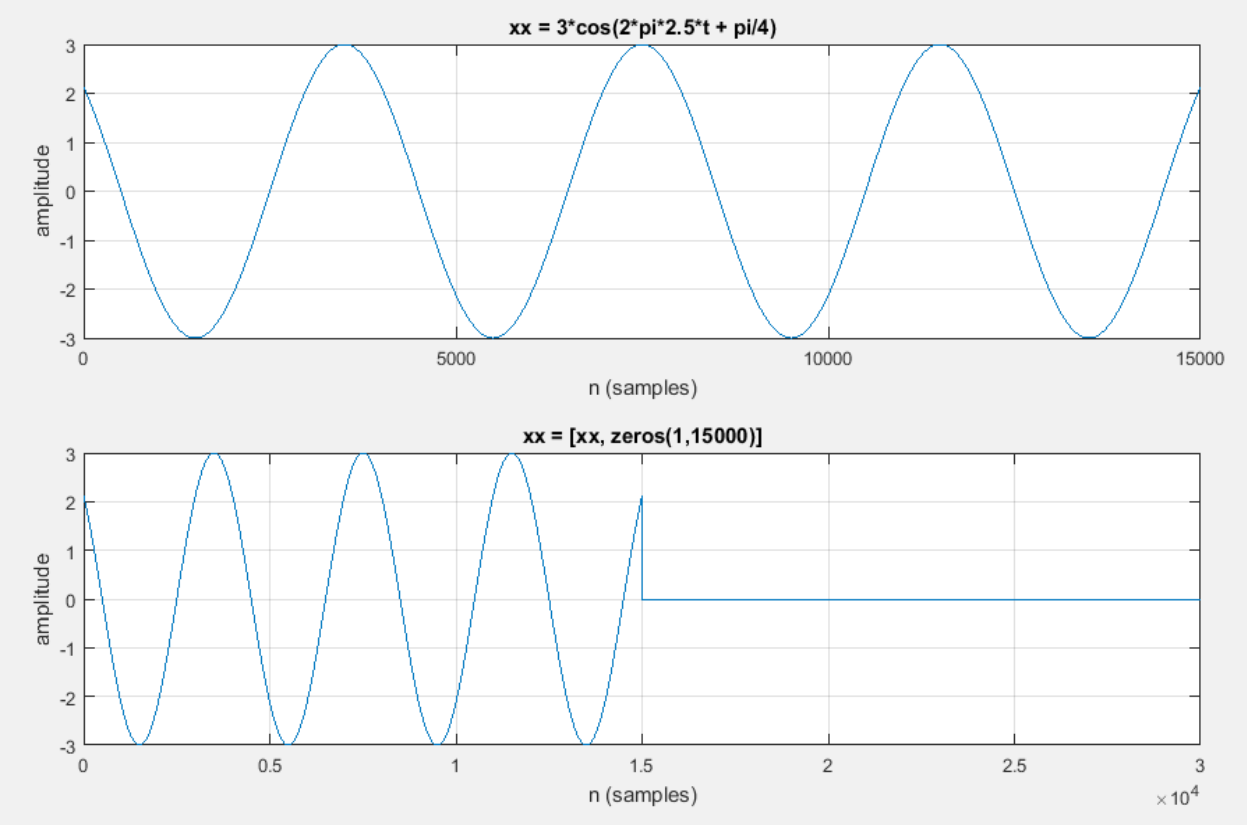
2) Use `fft()` and assign the output to `XX`. Plot the magnitude spectrum. Zoom on the left-most peak. You should notice that the spectrum is nonzero at multiple values surrounding this peak. By introducing a non-integer frequency, we see a *spread of energy* across the spectrum. **Find the frequency bins where this maximum occurs.** Does this exactly match the frequency 50.5 Hz?

2.2 Zero Padding

As you saw in the last section, the default resolution of the magnitude spectrum is not always what we need it to be. At the bin value you identified, the frequency may not have perfectly matched the frequency 50.5 Hz. To correct this, one might think to increase the sampling frequency, but this does not correct the error: it only expands the frequency range. To more accurately measure the frequency of our signal using the FFT, we can use a process called *zero padding*.

Zero padding works by concatenating a large sample of zeros (using the `zeros(1,N)` function, where `N` is the number of zeros appended) at the end of the signal being analyzed. For example, suppose our signal is $x(t)=3\cos(2\pi(2.5)t+\pi/4)$. The following script uses vector notation to add 15,000 zeros to the end of `xx`. Here the sampling frequency is 1000 samples/second and `t` is a time vector of 1500 samples. The results are plotted below:

```
xx = 3*cos(2*pi*2.5*t+ pi/4);
xx = [xx zeros(1,15000)];
```



Recall that the DFT works by correlating the input signal with sinusoidal basis functions (i.e. sines and cosines with an integer numbers of cycles over the duration of the input signal). Zero padding increases the likelihood of the DFT identifying a sinusoidal basis function that perfectly matches a sinusoidal component of the input signal. For more information on the mechanics of zero padding, refer to the following video:

[4] **How DFT Zero Padding Works**

https://www.youtube.com/watch?v=7yYJZfc5q-I

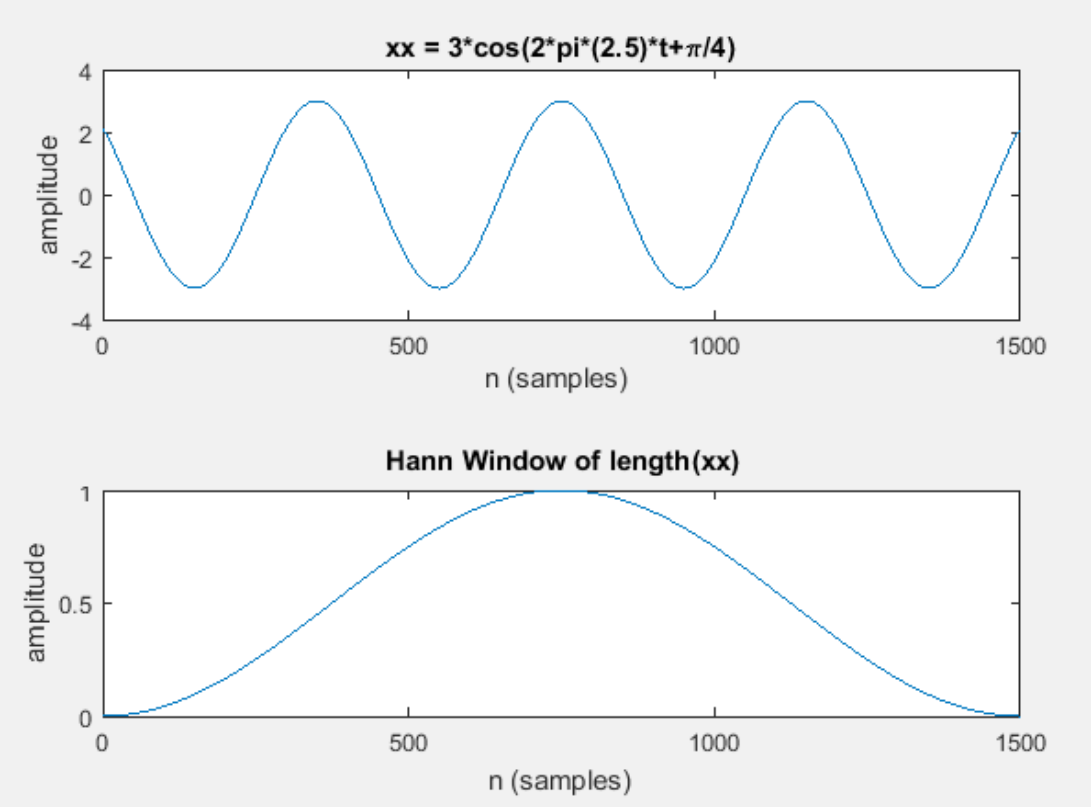
Now that you have some background information, complete the following steps:

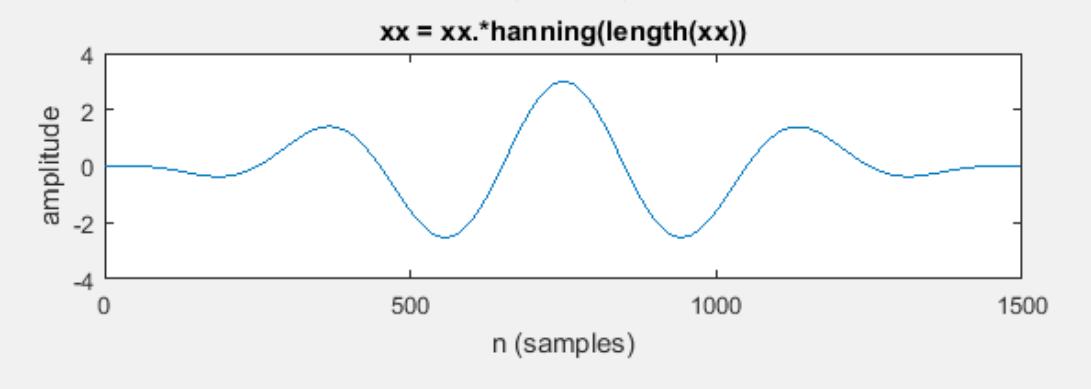
1) **Append 15,000 zeros** to our signal from section 1.7 (the one with frequency 50.5 Hz), and **run the FFT again. What is the effect of the zero padding on the magnitude spectrum of the signal? What is the frequency of the bin for which the magnitude is maximized?**

2) The appearance of the magnitude spectrum may be somewhat odd compared to what we have seen before. There should appear to be a main "lobe" and several smaller "side lobes" on the sides. **How does the width of these side lobes compare to the main lobe?**
3) Try zero padding the signal with 25,000 zeros, 50,000 zeros, and 100,000 zeros. Using the subplot function, **compare the appearances of the magnitude spectra**. **What is the effect of increasing the number of zeros in the accuracy of the frequency calculation? Does the error increase or decrease?** In the case of our signal, **how many zeros would need to be added in order to obtain an exact value for the frequency of our input signal?**

2.3 Windowing

When analyzing signals with sinusoidal components of non-integer frequencies, we will generally see magnitude spectra with "side lobes" surrounding the main peaks where frequency components occur. You have seen in the previous example that regardless of how many zeros we append at the end of our signal, we will still observe side lobes. When analyzing the FFT of ECG heart rate data in Part 2 of this lab, these side lobes may actually *interfere* with our calculations of the unknown heart rate. One method for reducing the size of side lobes is called *windowing*.





Suppose we have the signal $x(t)=3\cos(2\pi(2.5)t+\pi/4)$ which we padded with zeros in the previous section. If we multiply our signal with a bell-shaped function (the *window*) to reduce the amplitude of the endpoints of the function, we obtain a signal that fades in and out. In this lab we will use the Hann Window, similar to the Hamming window from previous labs, only differing by a constant. The following code implements the Hann Window:

```
xx = 3*cos(2*pi*2.5*t+ pi/4);
xx = xx.*hanning(length(xx))';
```

The results of implementing this code snippet are shown in the image on the previous page. If you read the documentation of the `hanning()` function in MATLAB, you will notice that it takes only one argument (the length of the input vector) and outputs a column vector. Hence, in order to multiply the values of the Hann Window by the values of our input signal, they need to be of the same dimension. Hence, we may need to transpose the Hann vector (i.e. `hanning(length(xx))'`). For a more in-depth explanation on windowing, refer to the following:

[5]DFT Windowing Explanation and Demo

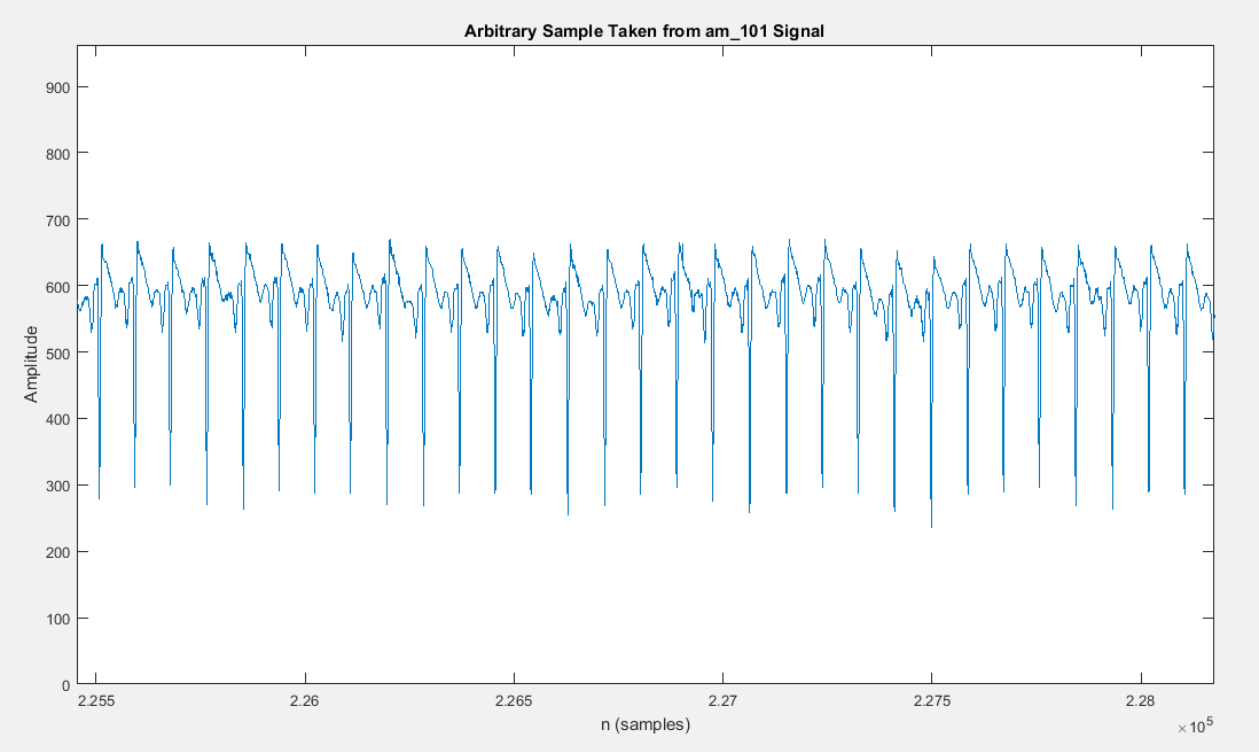
https://www.youtube.com/watch?v=PYd-pzaZpYE

Now, based on that background information, complete the following steps:

1) Returning to our signal $x(t)=\cos(2\pi(50.5)t+0.25\pi)$ from the past two sections, we want to reduce the size of the "side lobes" in the plot of the magnitude spectrum. Using the script above, **window** $x(t)$ **with a Hanning window**. *Note: this needs to be done before zero padding!* As before, use a sampling frequency of 1000 samples/second and a time vector of 1.5 seconds.

2) Padding with an appropriately large number of zeros, plot the magnitude spectrum of $x(t)$ after windowing and zero padding. **Make a side-by-side plot** of the magnitude spectra, zoomed in at the peak value, for `XX` with windowing and `XX` without windowing. **What effect does windowing have on the FFT? How does the magnitude of the maximum peak change with windowing? Is it higher or lower than that of the maximum peak without windowing? Has the width changed at all?**

2.4 Determining Average Heart Rate Using FFT



A plot showing a sample of heart beat waveforms from one of the ECG measurements in the zip file.

In this exercise, you will employ the `fft()` function in MATLAB to automatically extract the heart rate from electrocardiogram (ECG) measurements. The data file on Canvas, `heartratelabdata`, contains four ECG signals (obtained from http://physionet.org/cgi-bin/atm/ATM). The sampling rate for all data files is 125 Hz. Your goal will be to use the Fourier transform to produce four plots, showing the average heart rate changing over each signal's duration.

Key Steps to a Solution

Note: You are allowed and encouraged to use anything you have learned to help the FFT identify the heart rate. This may include zero padding, windowing, filters, etc. In addition, you can choose whatever window length or overlap length you feel will give the best result.

1. Load each signal in MATLAB: am101, am105, am107, and cm110.
2. In this step, you will program a function or write a MATLAB script to read each signal and determine the average heart rate. Since each signal is *one hour* long, you will need to divide the signal into subsamples (or 'windows') of equal width. It is suggested you start by using a window about 2 minutes long. DO THE MATH in order to condense it to 2 minutes with even windows. (*Note:* different window sizes may be needed for a better result)
3. In order to plot the average heart rate over time, you will need to iterate through the windows and take the FFT for each window. It is suggested that there be an *overlap* between each window as the function iterates. This overlap is suggested to start to be about 10 seconds. (*Note:* different overlap sizes may be needed for a better result)
4. Take the absolute value of the FFT of the sample (`abs(XX)`) and identify the peak closest to bin zero, but not at zero. You are looking for the *first harmonic*. Your algorithm should automatically identify the first harmonic each time the function iterates, because it is this peak that stores the frequency information of the heart rate. **Note that this may not be the highest peak in your spectrum**; often, in the real world, the second or third harmonic, with frequency two or three times the frequency of the first harmonic, may be louder.
5. Record the index/bin of the peak you identified. Convert this bin number to frequency in Hz, or beats per second. Note that this may be near one beat per second. Multiply this frequency by 60 to obtain the heart rate, in beats per minute, for each window.
6. Store these average heart rates into a vector, and then produce a plot showing the change in the average heart rate over time.
7. Write the above into a function that takes a data vector (am101, am105, am107 or cm110) as input, calculates the heart rate for each two-minute window, and plots heart rate over time.
8. Use that function to produce the four plots, one for each signal.