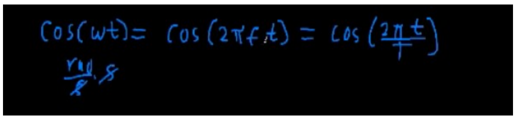
****************Draft****************

FFT Octave

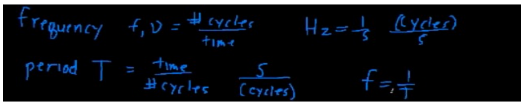
09/08/20

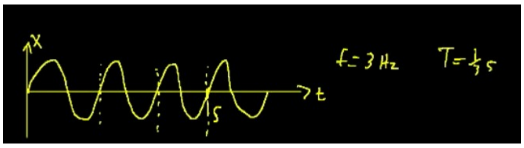
****************Draft****************

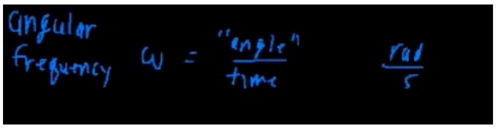
Background:
https://youtu.be/67Uaxh-4EIQ



freq & period

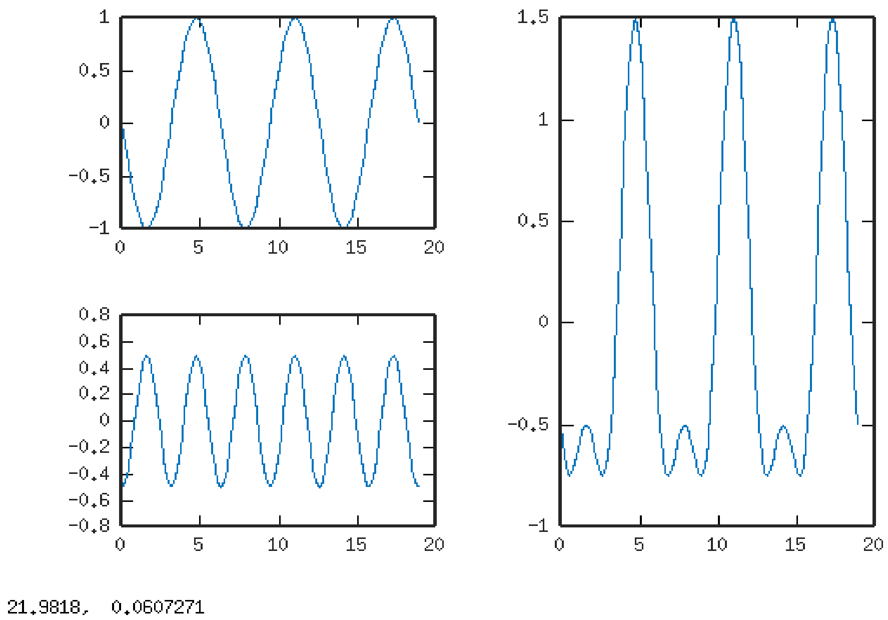






http://xoctave.com/blog/signal-processing-with-xoctave/

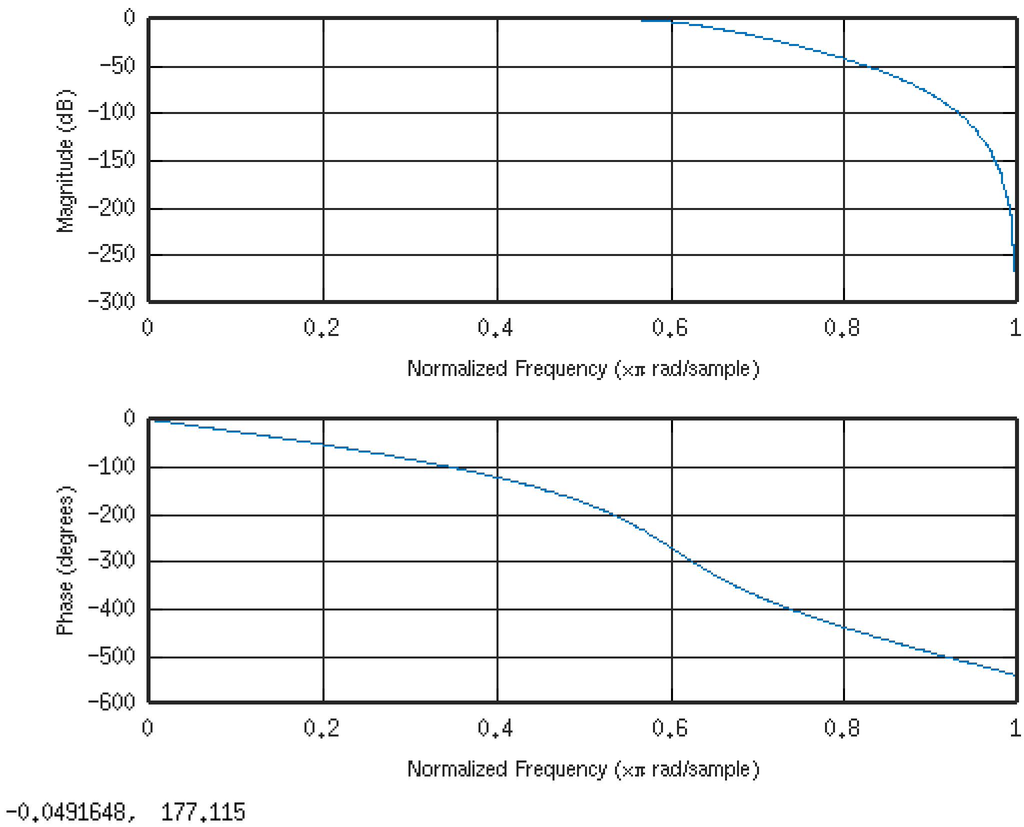
```
clear
close all
pkg load signal
t = linspace(0,6*pi,100); % time
y1 =sin(-t);  % first signal
y2=-0.5*cos(2*t);  % second signal
% plotting
figure,
subplot(2,2,1),plot(t,y1); % top chart
subplot(2,2,3),plot(t,y2), % bottom left chart
subplot(1,2,2),plot(t,y1+y2) % resulting right chart
```



https://www.mathworks.com/help/signal/ref/butter.html

%Design a 6th-order lowpass Butterworth filter with a cutoff frequency of 300 Hz,
%which, for data sampled at 1000 Hz, corresponds to 0.6π rad/sample.
%Plot its magnitude and phase responses. Use it to filter a 1000-sample random signal.

```
clear
close all
pkg load signal
fc = 300;
fs = 1000;

[b,a] = butter(6,fc/(fs/2));
freqz(b,a);
```



dataIn = randn(1000,1);
dataOut = filter(b,a,dataIn);
//github.com/develone/Butterworth_filter_coefficients-MATLAB-in-C.git
modifing bwlp.c by including liir.c
git diff

```
diff --git a/bwlp.c b/bwlp.c
index 018cde2..e91ef6b 100644
--- a/bwlp.c
+++ b/bwlp.c
@@ -41,6 +41,7 @@
 #include <string.h>
 #include <math.h>
 #include "iir.h"
+#include "liir.c"
```

Use the following command to compile “gcc bwlp.c -lm -o lp”

Use the following command to create “./lp 6 .6 1 lpcoeffsc” B & A in the file lpcoeffsc.

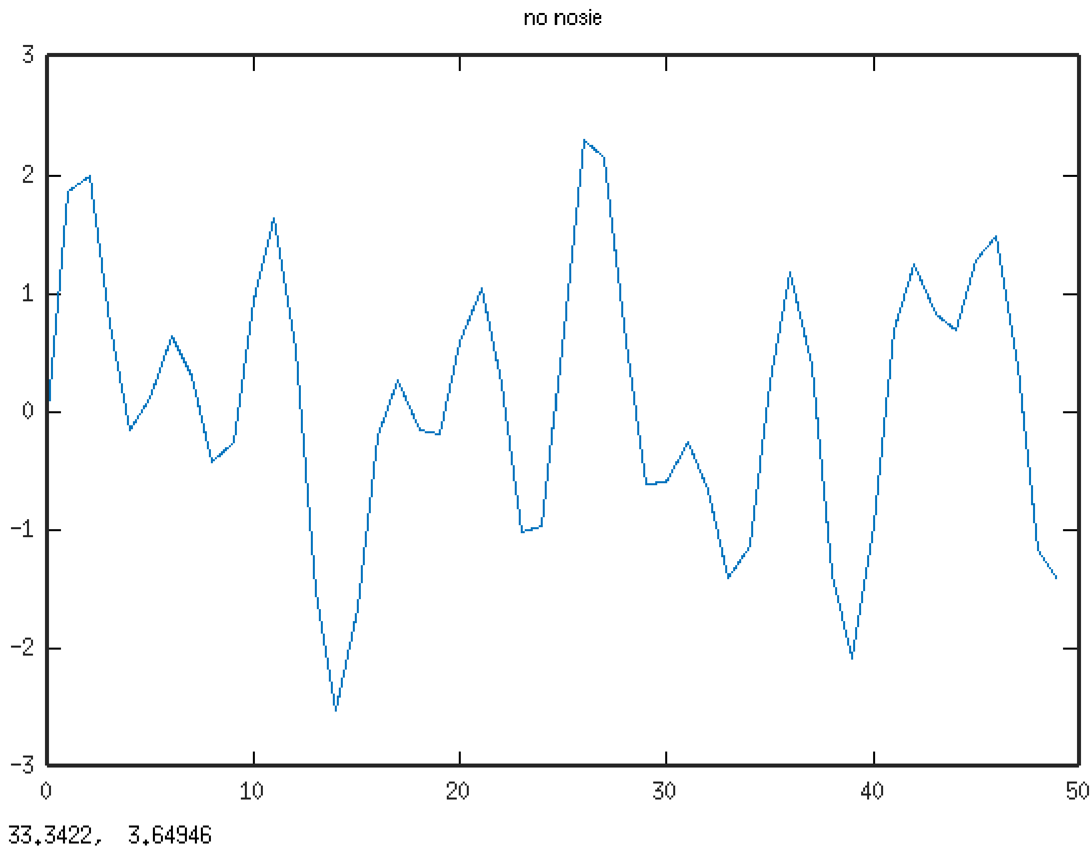
7
0.070115413492454
0.420692480954722
1.051731202386805
1.402308269849073
1.051731202386805
0.420692480954722
0.070115413492454
7

1.000000000000
1.187600680176
1.305213349289
0.674327525298
0.263469348280
0.051753033880
0.005022526595

Starting with an example in Matlab at “https://www.mathworks.com/help/matlab/ref/fft.html”

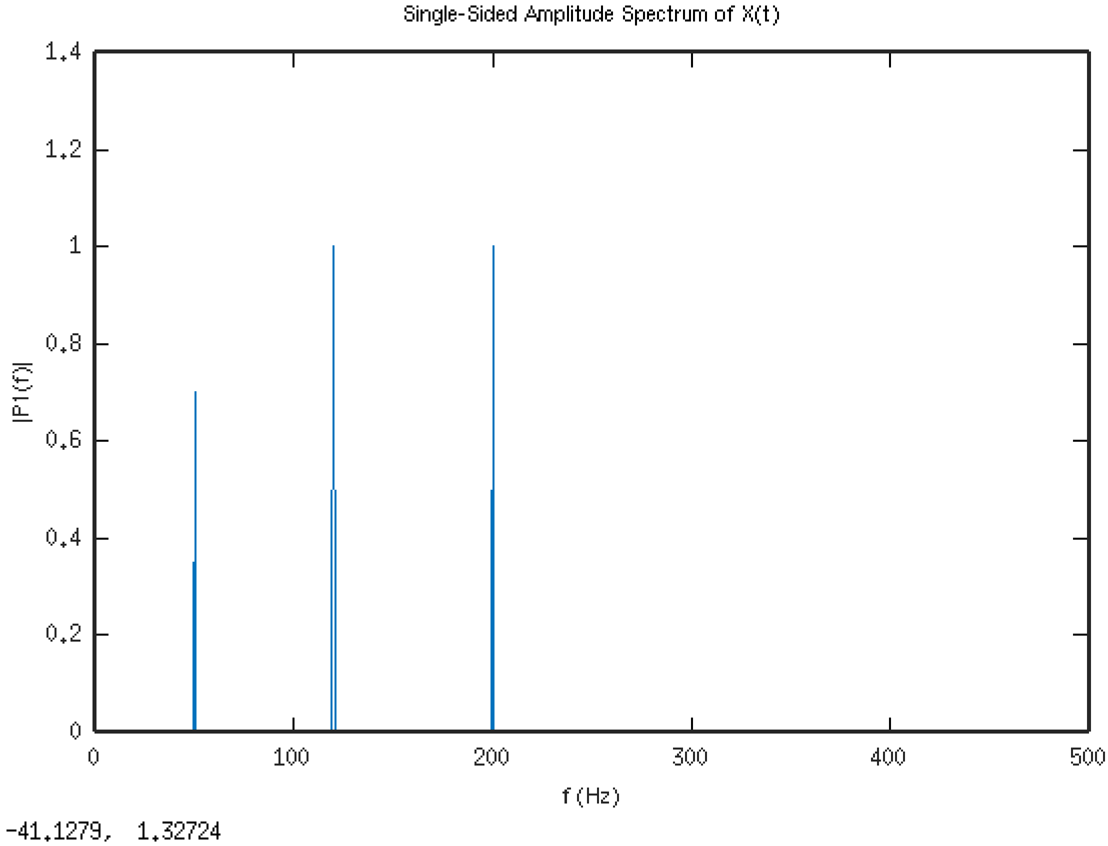
The signal below is combination of 3 frequencies.   The 3 frequencies are
50, 120, and 200 Hz.

Figure 1



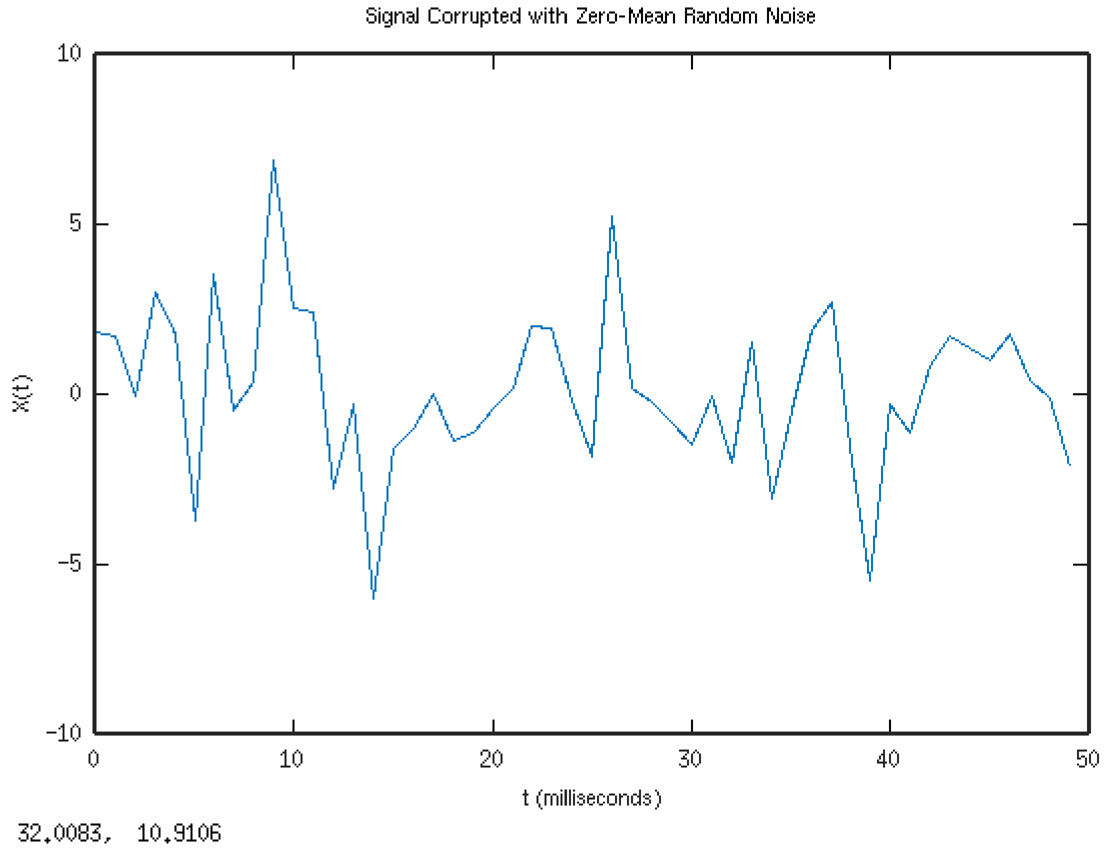
Taking the FFT demonstrates what frequencies make up the signal above.

Figure 2



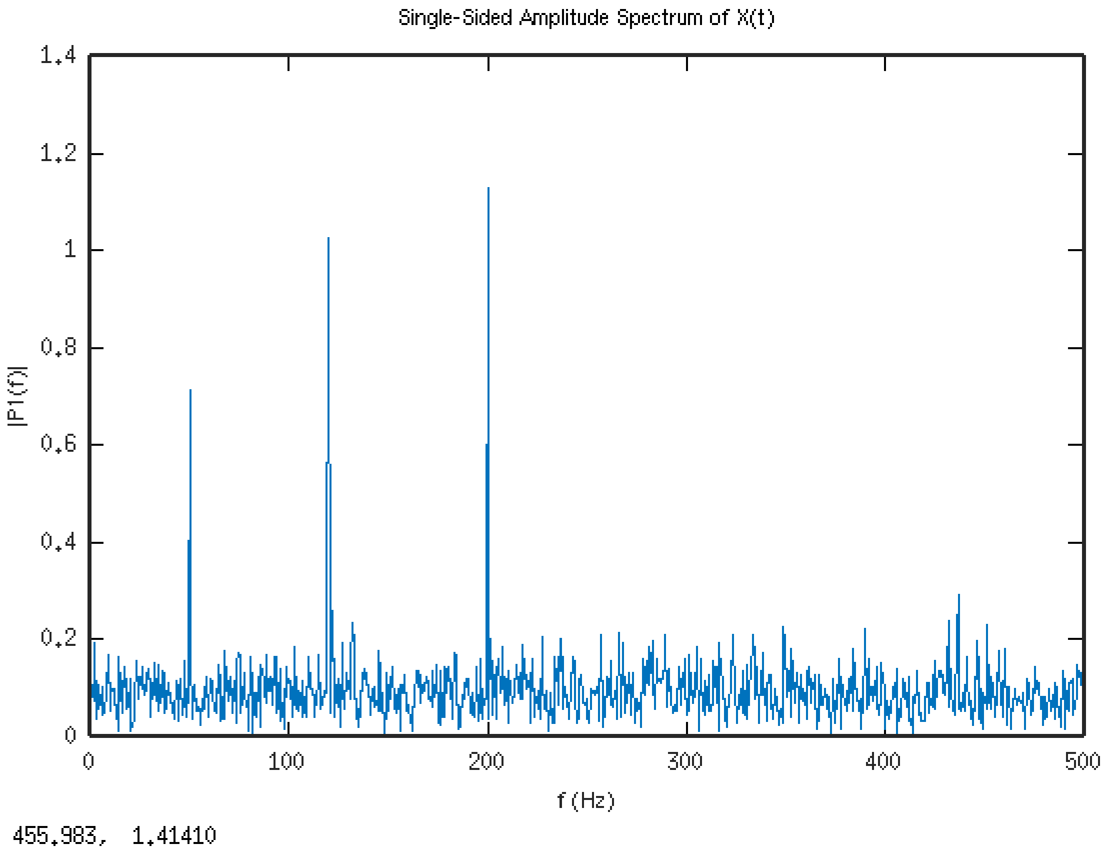
The signal below is combination of 3 frequencies witgh random noise. The 3 frequencies are 50, 120, and 200 Hz.

Figure 3



Taking the FFT demonstrates what frequencies make up the signal above.

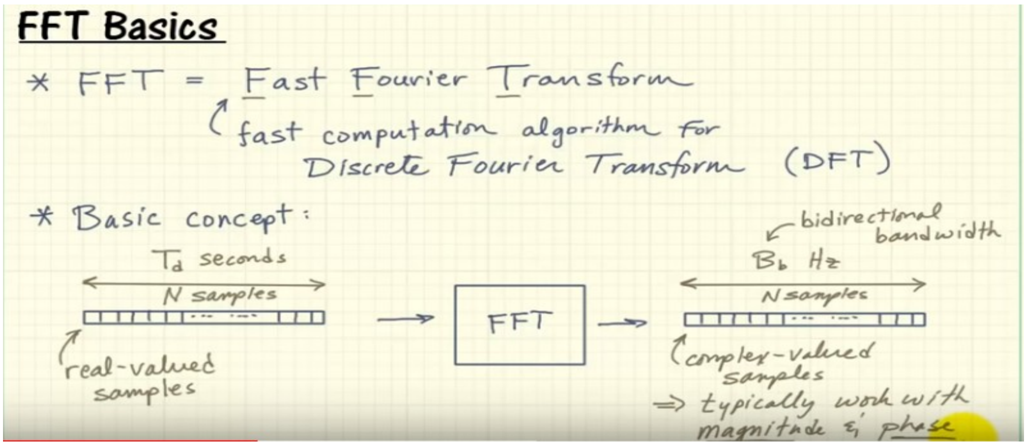
Figure 4



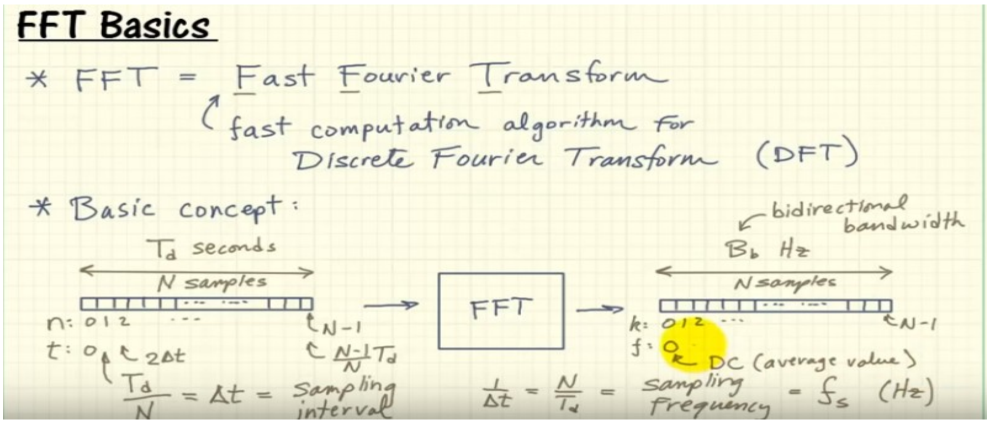
https://www.youtube.com/watch?v=z7X6jgFnB6Y&feature=youtu.be

fft.png

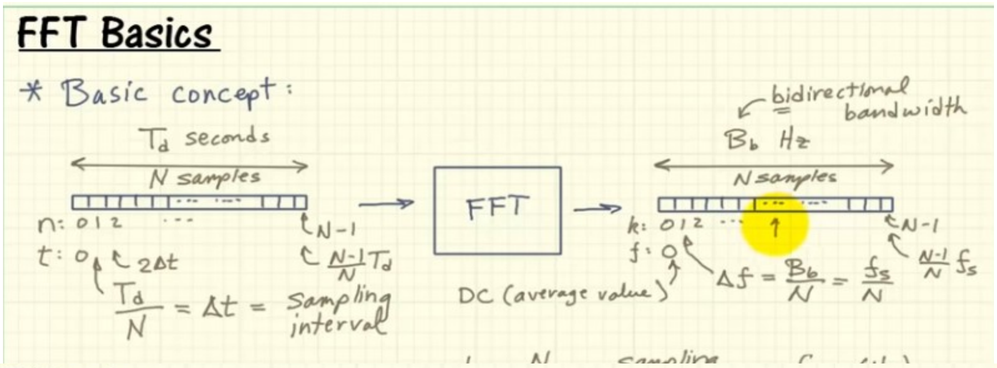
FFT basic concepts

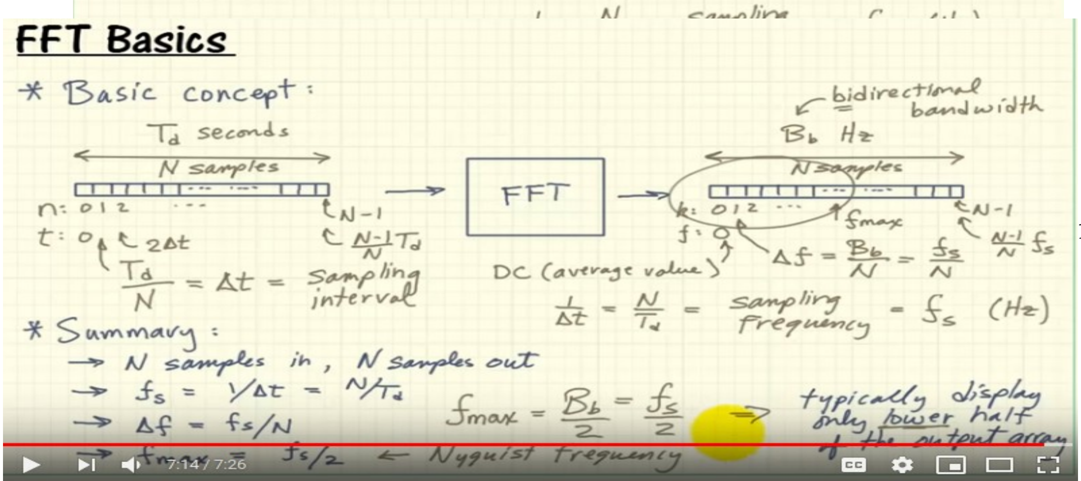


fft1.png



fft2.png



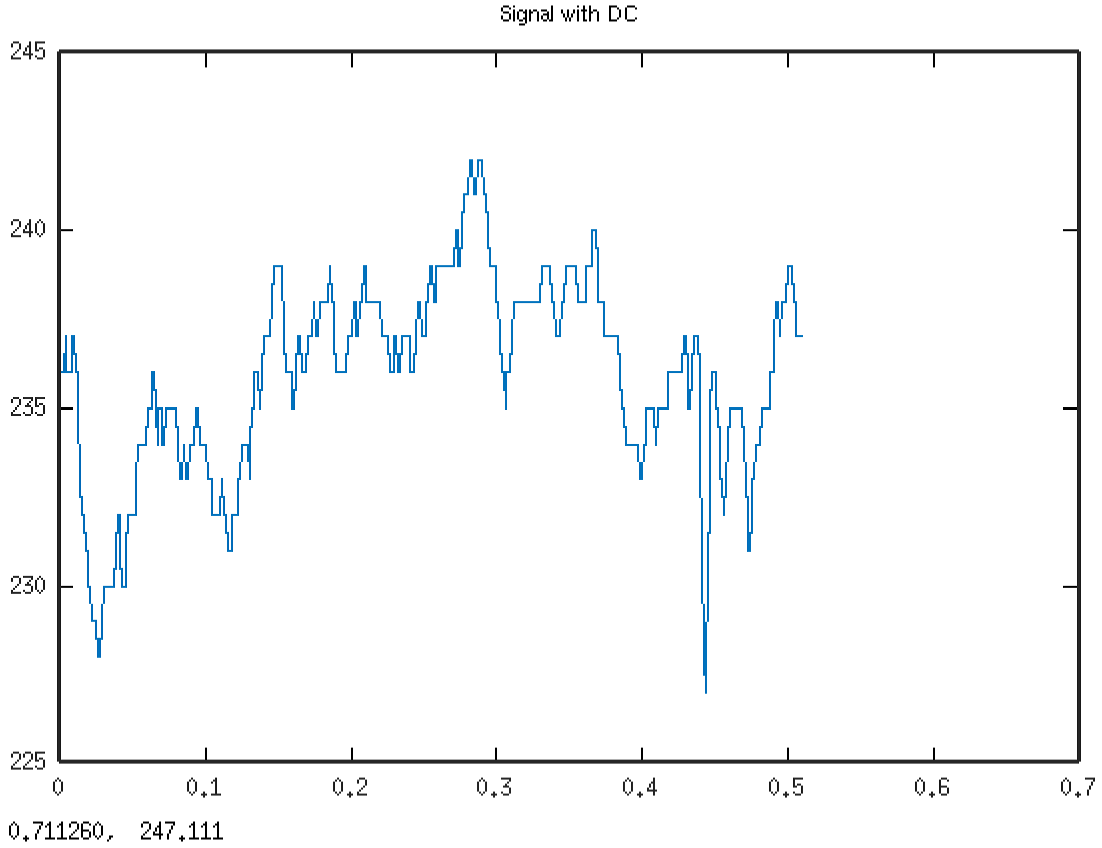


fft3.png

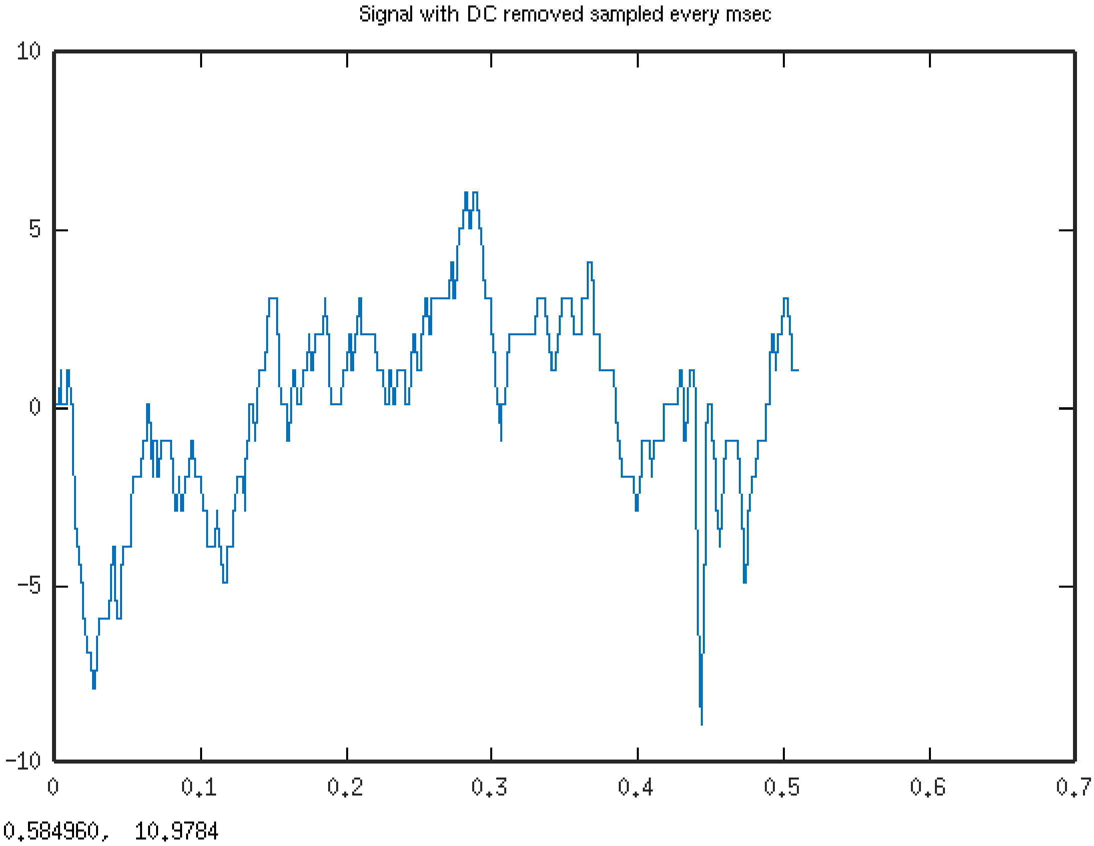
Octave code that produce the plots above.

```
clear
close all

Fs = 1000;            % Sampling frequency
T = 1/Fs;             % Sampling period
L = 1500;             % Length of signal
t = (0:L-1)*T;        % Time vectorc

%S = 0.7*sin(2*pi*50*t) + sin(2*pi*120*t);
S = 0.7*sin(2*pi*50*t) + sin(2*pi*120*t) + sin(2*pi*200*t);

X = S + 2*randn(size(t));

figure

plot(1000*t(1:50),X(1:50))
title('Signal Corrupted with Zero-Mean Random Noise')
xlabel('t (milliseconds)')
ylabel('X(t)')

Y = fft(X);

P2 = abs(Y/L);
P1 = P2(1:L/2+1);
P1(2:end-1) = 2*P1(2:end-1);

figure
f = Fs*(0:(L/2))/L;
plot(f,P1)
title('Single-Sided Amplitude Spectrum of X(t)')
xlabel('f (Hz)')
ylabel('|P1(f)|')
```

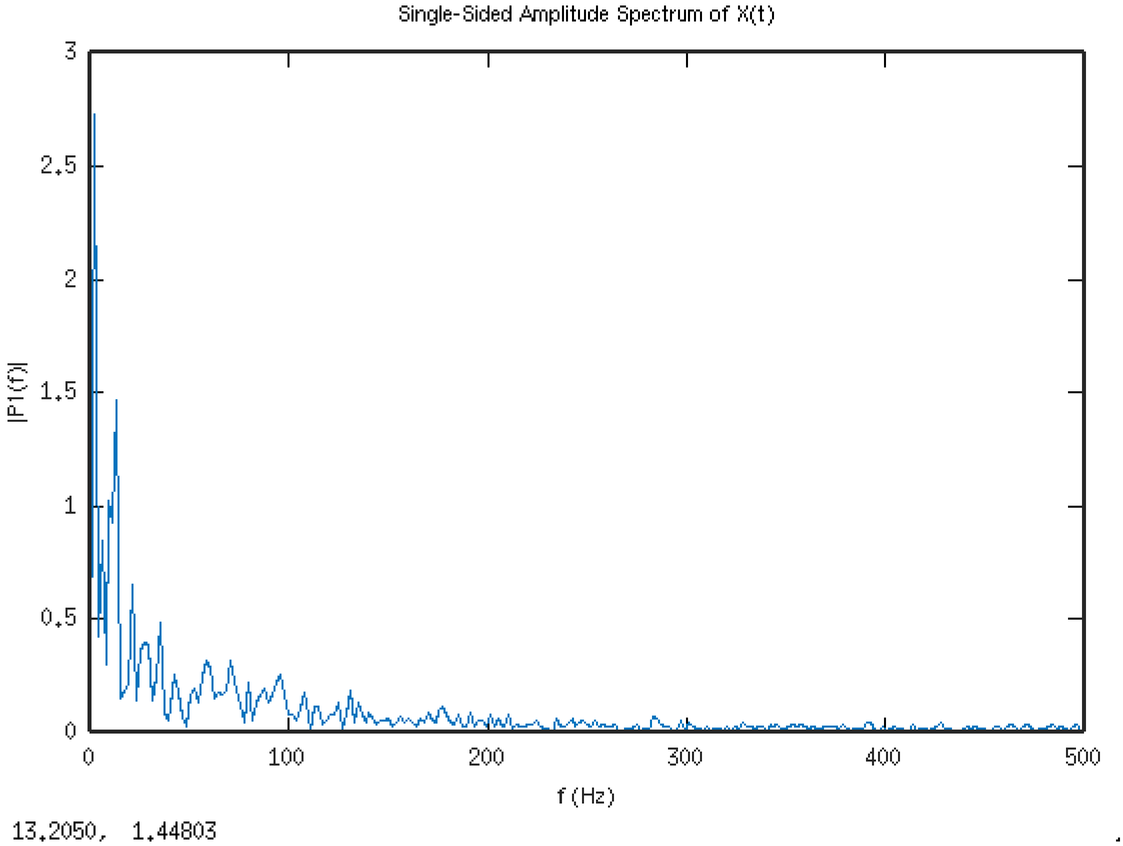
red.pgm 50th row sampled at 1KHz



red.pgm 50th row sampled at 1KHz with DC removed.
xbar = 235.90



FFT red.pgm 50th row.




```
clear
close all
%https://www.mathworks.com/matlabcentral/answers/1828-remove-dc-component-from-eeg-signals
Fs = 1000;            % Sampling frequency
T = 1/Fs;             % Sampling period
L = 512;              % Length of signal
t = (0:L-1)*T;        % Time vector

A = imread('red.pgm');
figure
x = A(1:512,50);
xbar = mean(x)

plot(t,x)

title("Signal with DC")

f = fft(x);
f(1) = 0;
x_ac = real(ifft(f));
[x_ac, x - mean(x)];

figure
plot(t,x_ac)
title("Signal with DC removed sampled every msec")
```

```
Ys = fft(x_ac);

P2 = abs(Ys/L);
P1 = P2(1:L/2+1);
P1(2:end-1) = 2*P1(2:end-1);

figure
f = Fs*(0:(L/2))/L;
plot(f,P1)
title('Single-Sided Amplitude Spectrum of X(t)')
xlabel('f (Hz)')
ylabel('|P1(f)|')
```

Located an example of real fft using kissftt https://github.com/heavyii/kissfft-example
Extracted the 2048 samples of real data from kissfft-example and read it into octave.
100 FFTs using kiss-fft

```
time ./real-fft
hz =  1044.921875
hz =  1044.921875
hz =  1044.921875
hz =  1044.921875
hz =  1044.921875
          .
          .
          .
hz =  1044.921875
hz =  1044.921875
hz =  1044.921875
hz =  1044.921875

real	0m0.026s
user	0m0.008s
sys	0m0.018s
```

Octave finds the hz at 1089.26 which is 4.24 % higher than kiss-fft.

******************Draft*******************

Testing kissfft using kissfft-example
08/27/20

******************Draft*******************

forked the kissfft-example

```
git clone https://github.com/heavyii/kissfft-example.git
```

git clone https://github.com/develone/kissfft-example.git

cd kissfft-example/

make

```
kiss_fftr.c: In function ‘kiss_fftr_alloc’:
kiss_fftr.c:53:18: warning: cast increases required alignment of target type [-Wcast-align]
     st->tmpbuf = (kiss_fft_cpx *) (((char *) st->substate) + subsize);
```

cp ../kissfft/kiss_fft.c .

cp ../kissfft/*.h .

cp ../kissfft/*.hh .

make

sudo pip install matplotlib

sudo apt-get install python-gi-cairo

./real-fft dump-raw-3.txt test-output.txt

image1

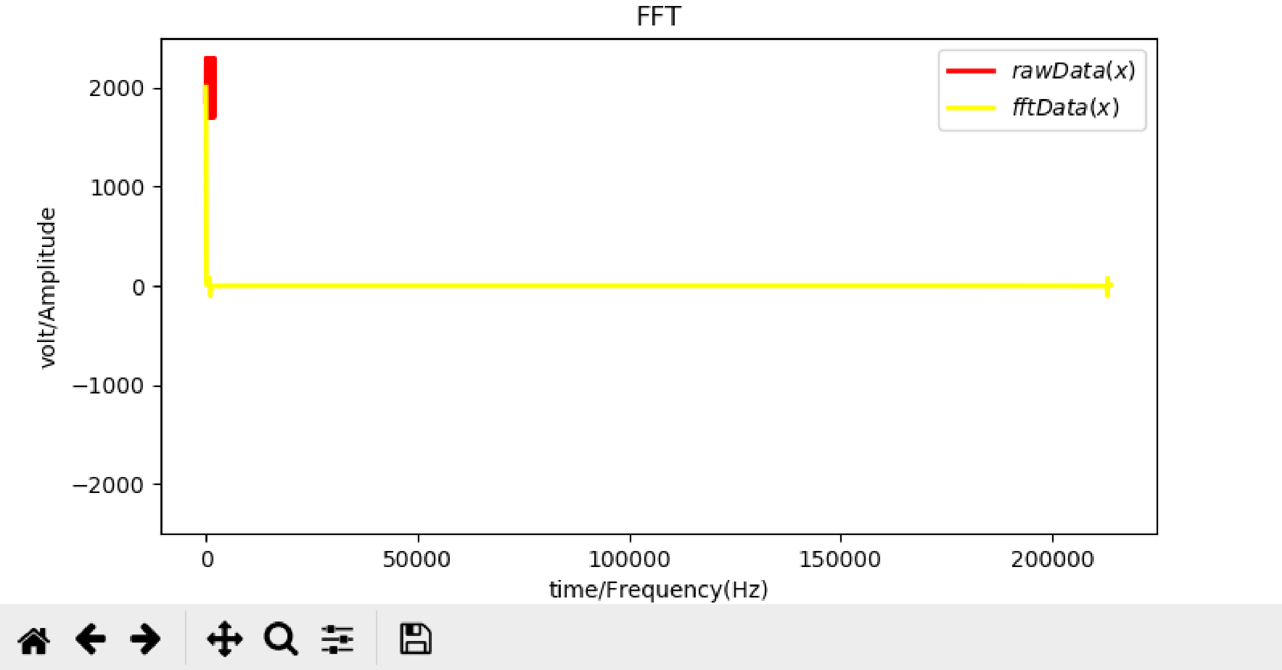

FFT
rawData(x)
fftData(x)
2000
1000
0
−1000
−2000
volt/Amplitude
0
50000
100000
150000
200000
time/Frequency(Hz)

image2

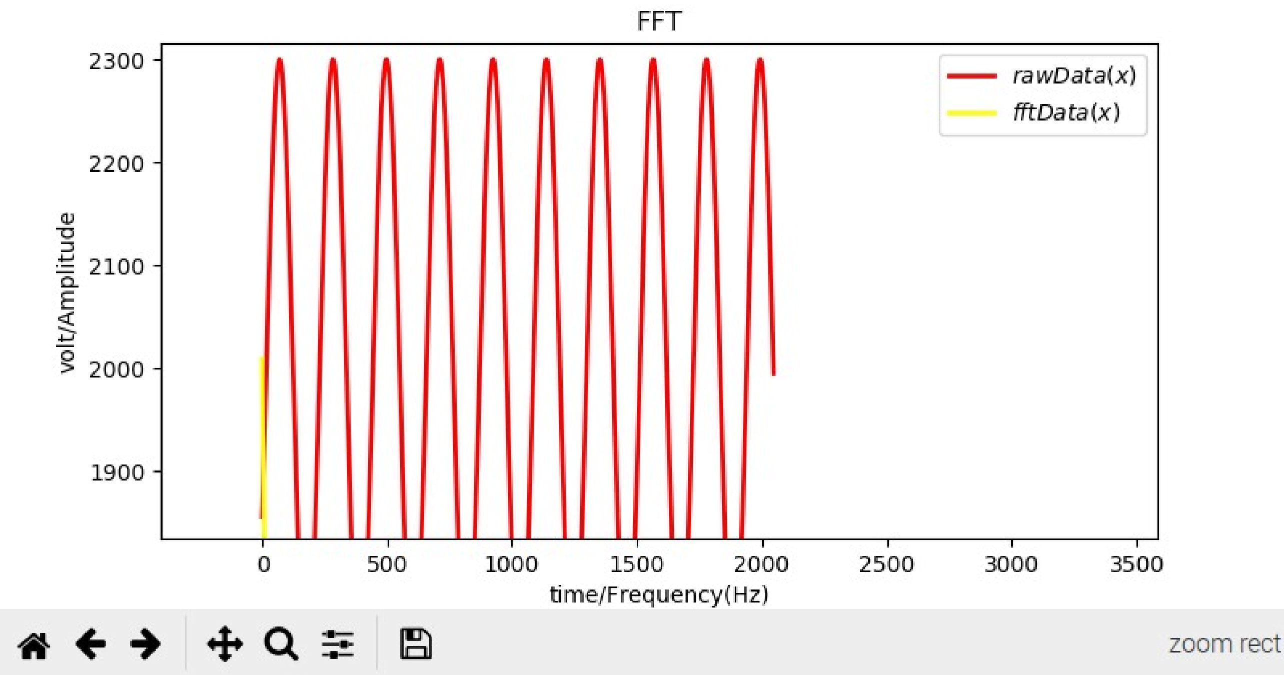


image3

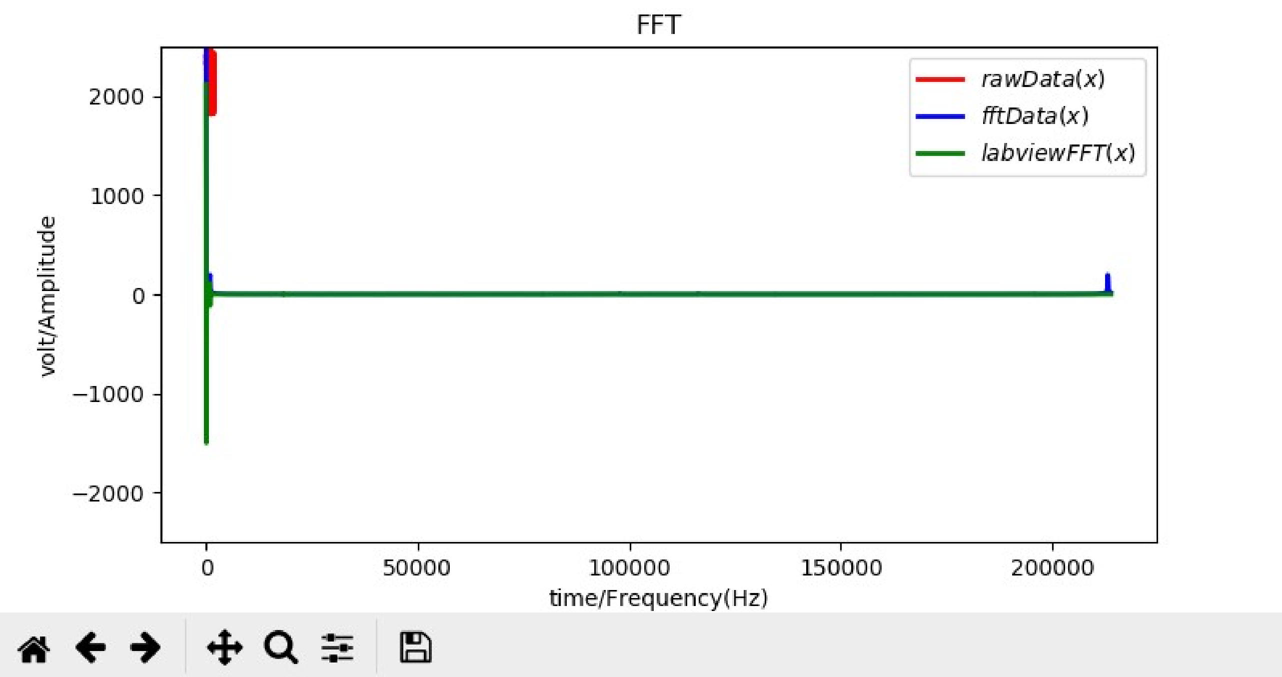


image4

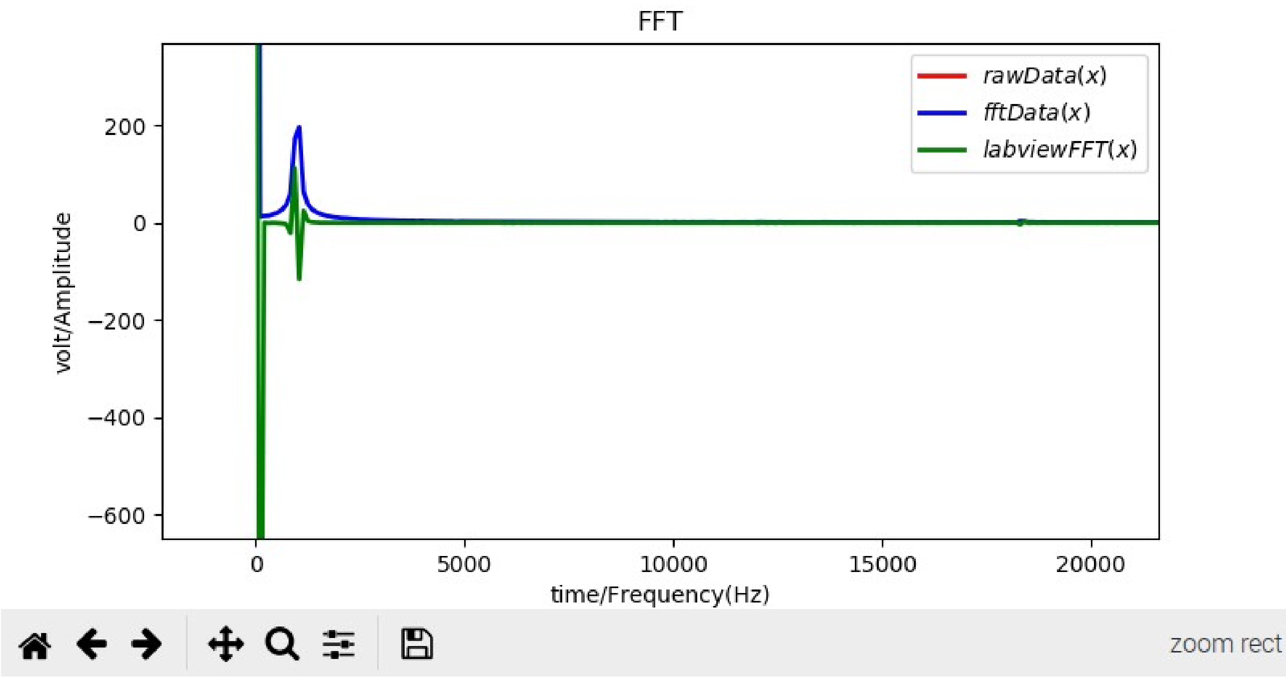


image5

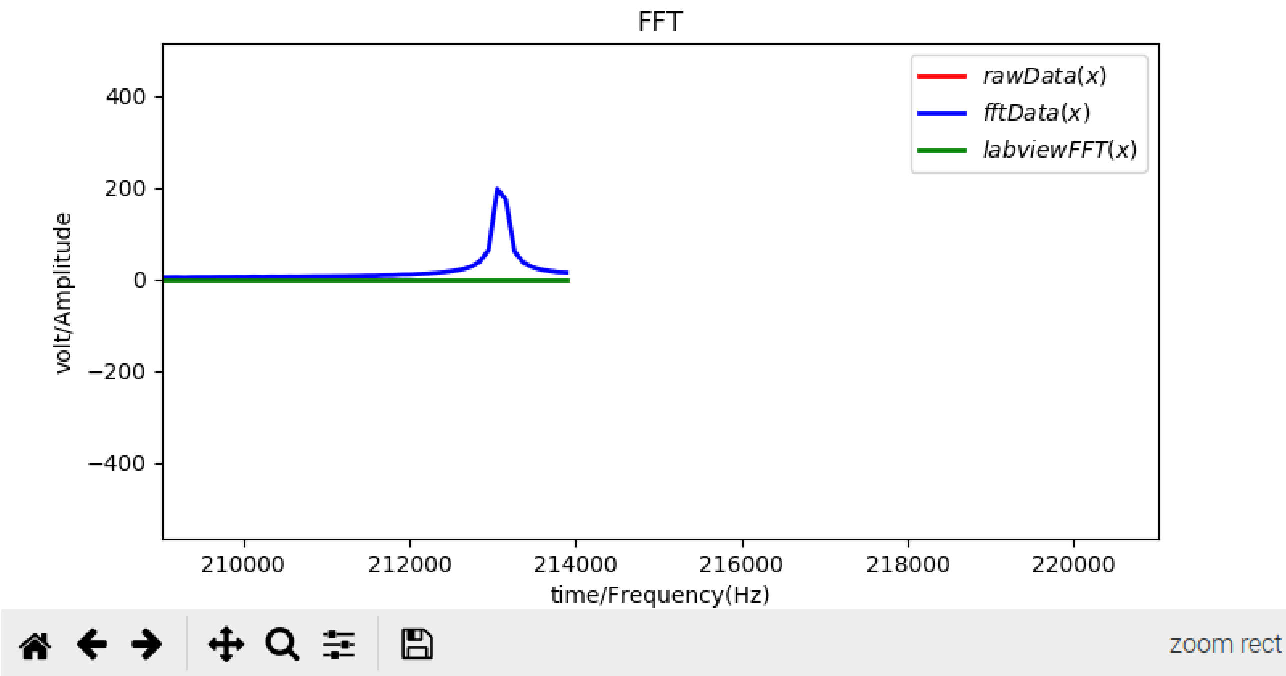
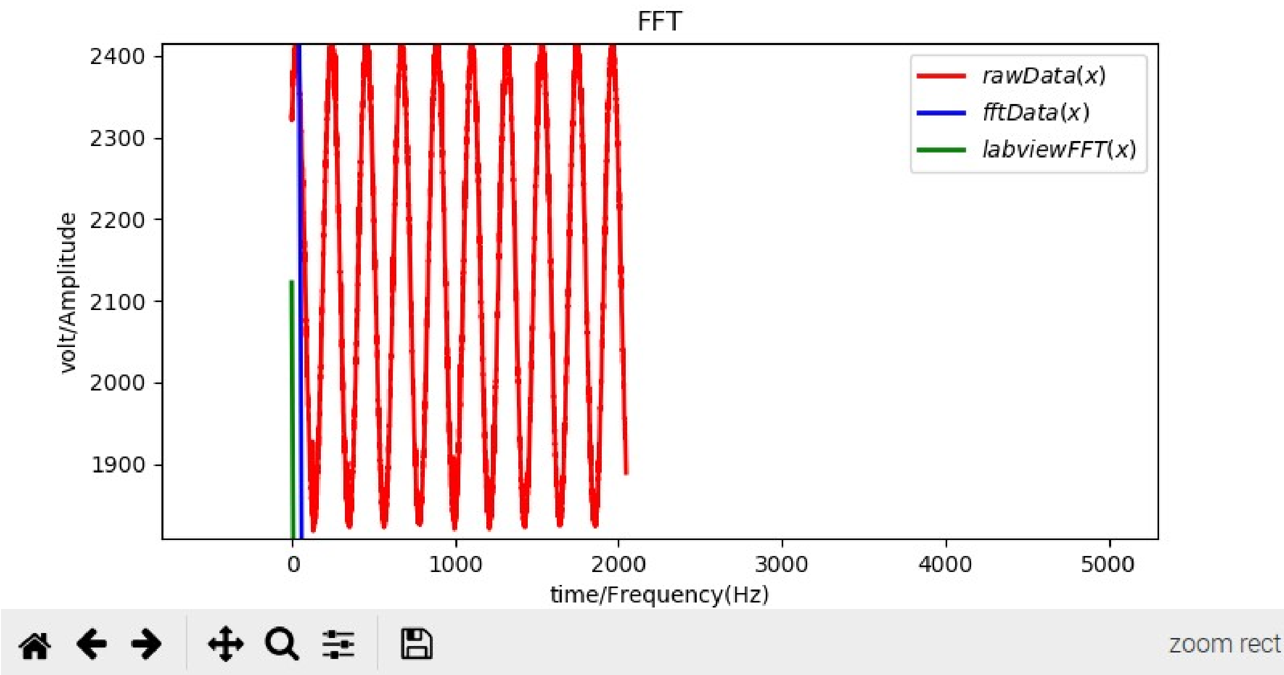
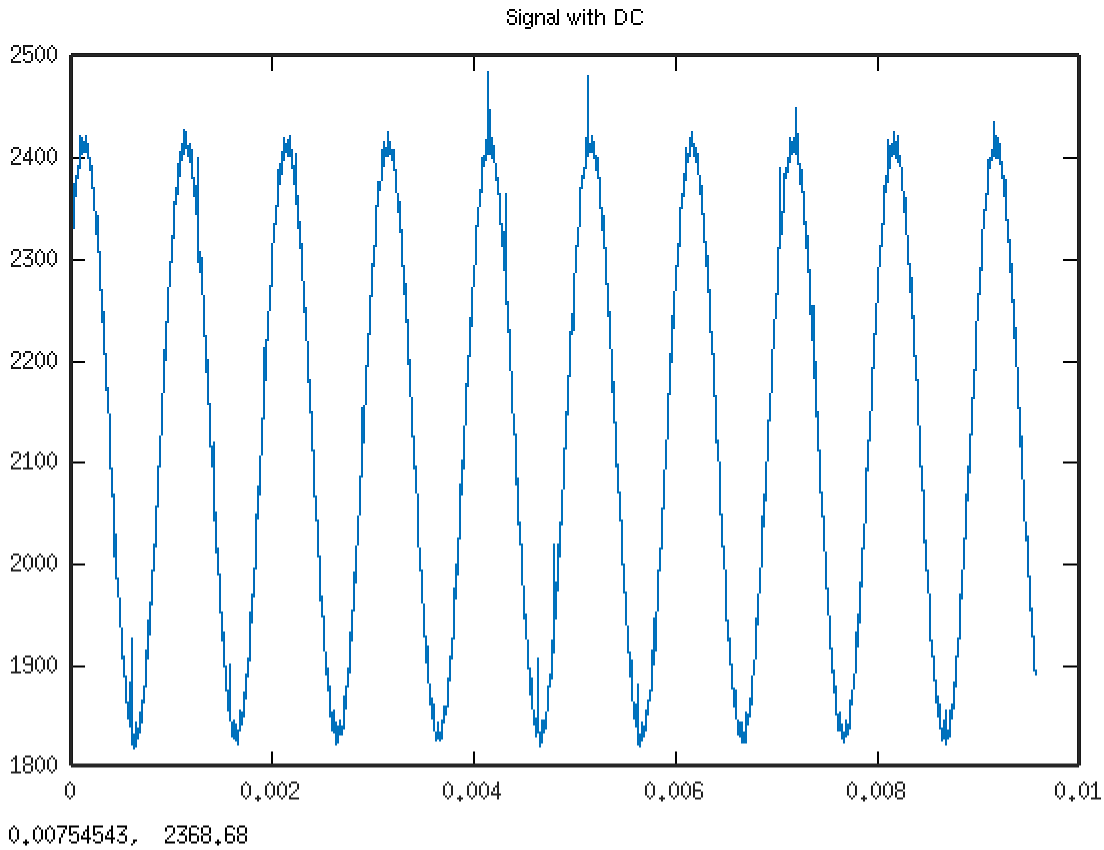
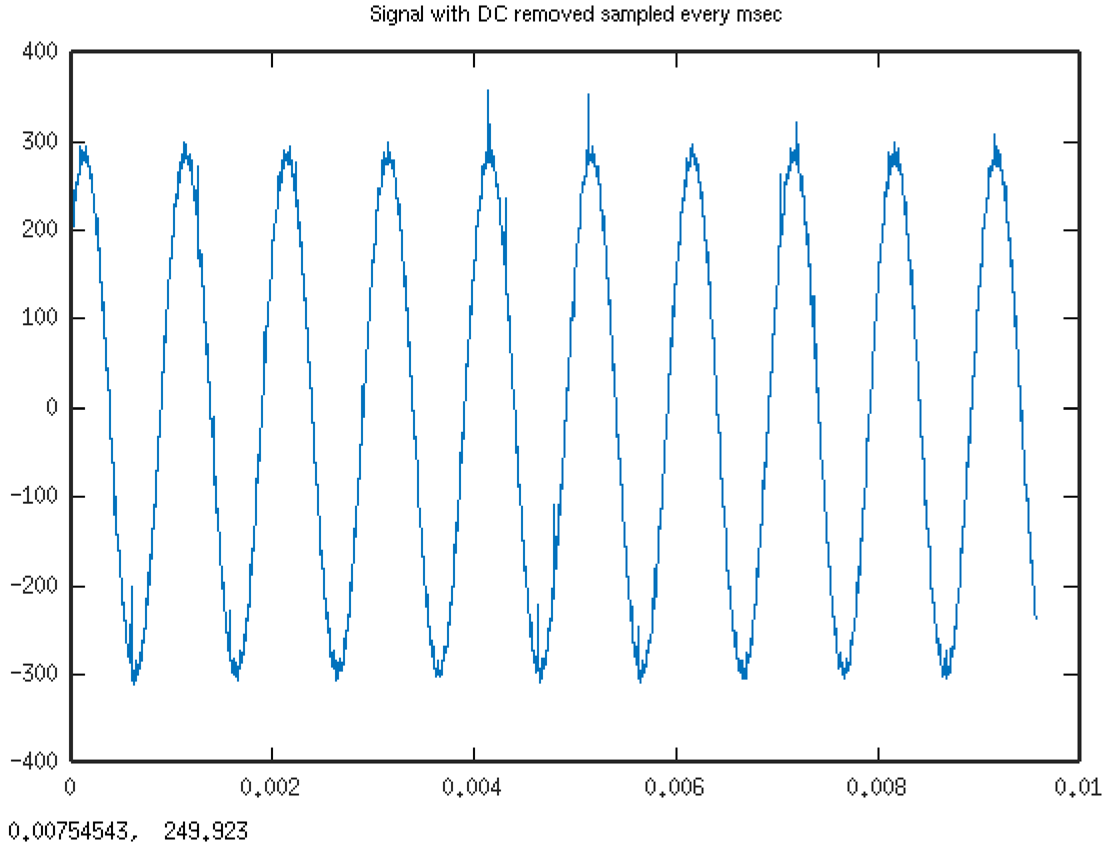


image6


FFT
rawData(x)
fftData(x)
labviewFFT(x)
volt/Amplitude
2400
2300
2200
2100
2000
1900
0
1000
2000
3000
4000
5000
time/Frequency(Hz)
zoom rect


Signal with DC
2500
2400
2300
2200
2100
2000
1900
1800
0
0.002
0.004
0.006
0.008
0.01
0.00754543, 2368.68



Removed the DC from extracted data.

Compute the FFT.