In the Name of God

Signals and Systems

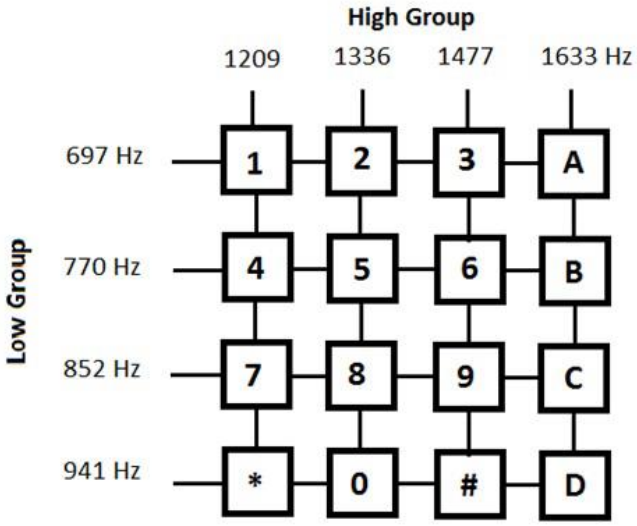
Matlab_Homework_3 Report

Armin Panjehpour – 98101288

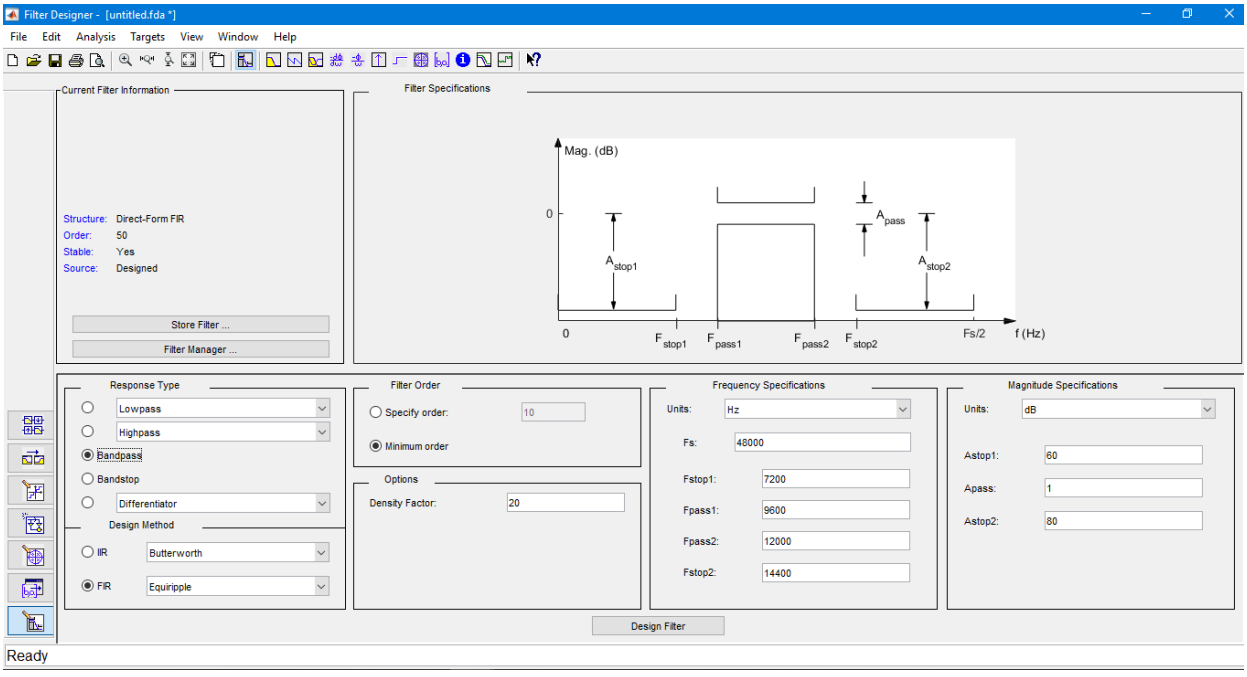
1- DTMF:

1.1- DTMF_Decoder:

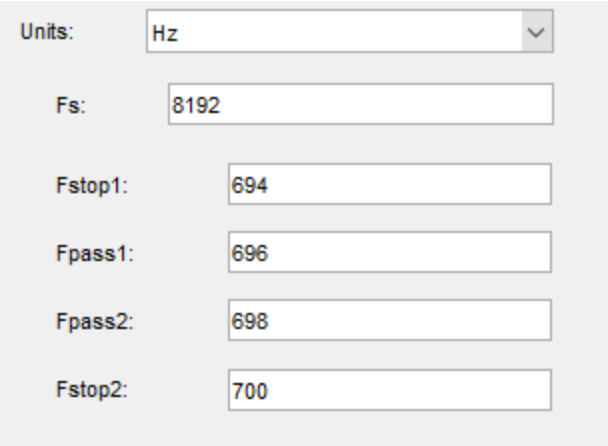
At first, we design 8 bandpass filters using **MATLAB filterDesigner**.



For designing such this filters, type filterDesigner in Matlab command window and a window like the one attached here will be opened and we choose bandpass mode:

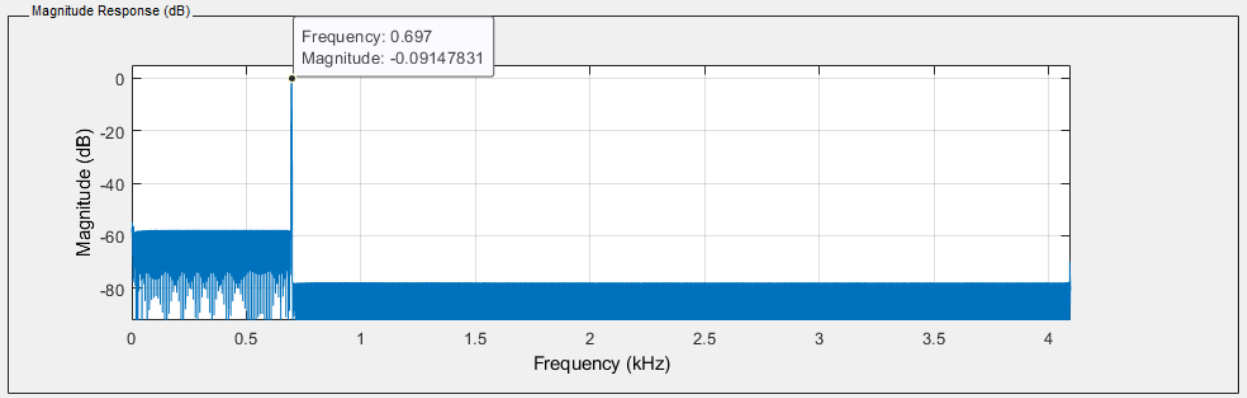


Now we have to set the parameters for our bandpass filter. For designing a 697Hz bandpass filter, we set these frequency values:

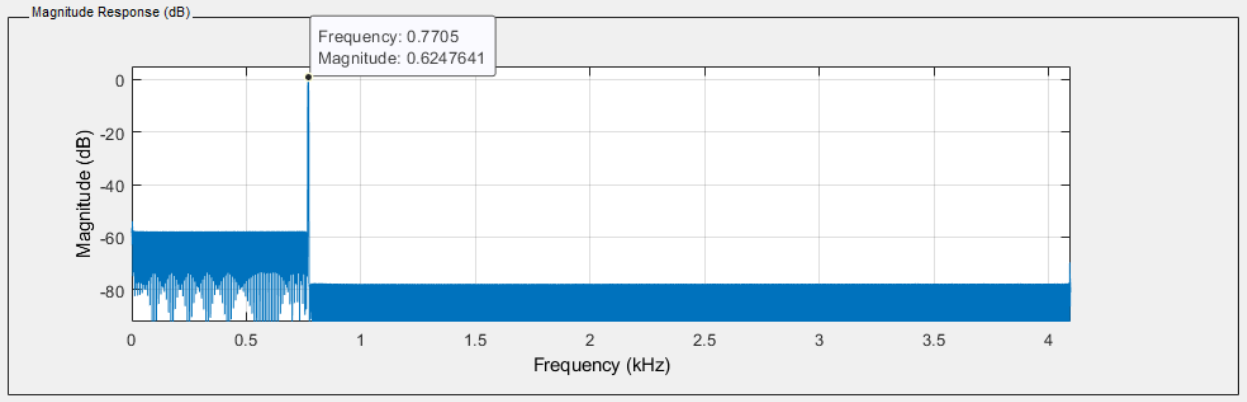


This filter will pass the frequency 697Hz which we wanted. For other filters, we can do the same.

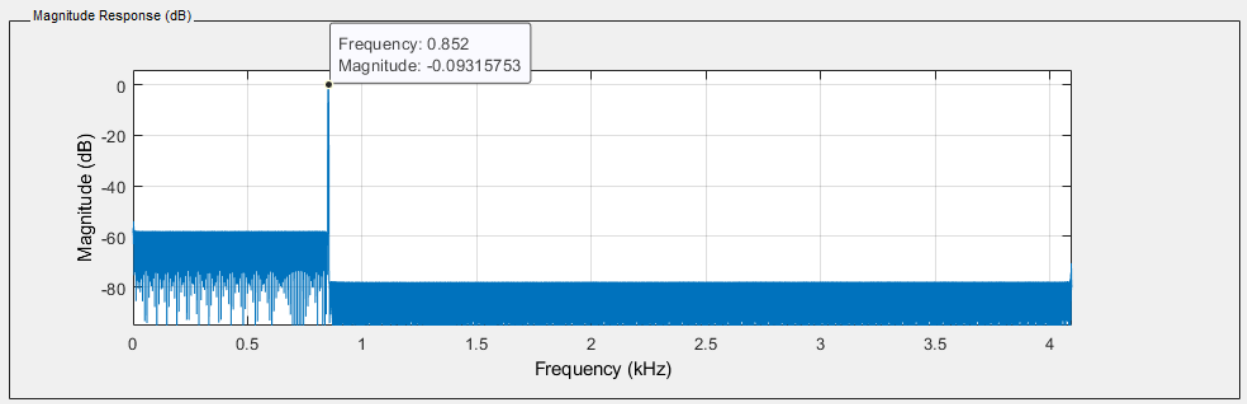
- Frequency response of filters:
  - 697 Hz:



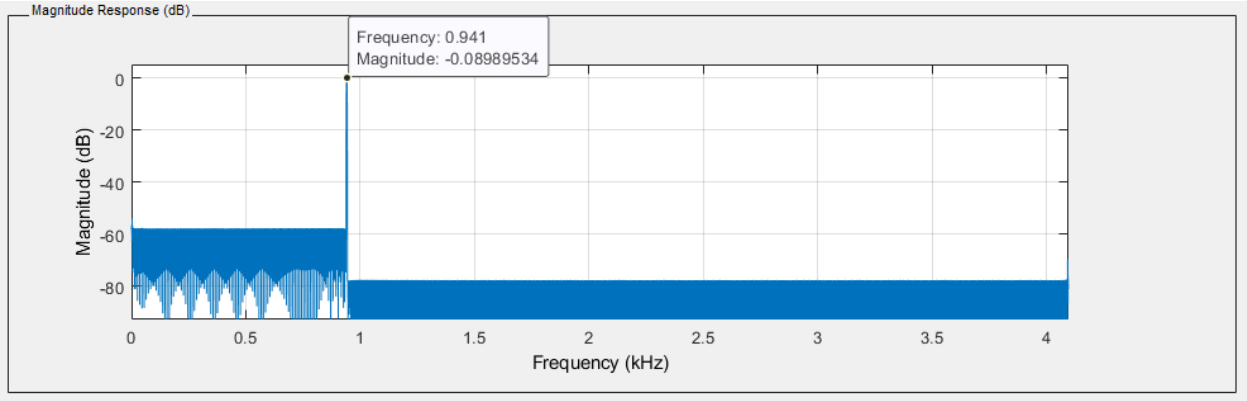
  - 770 Hz:



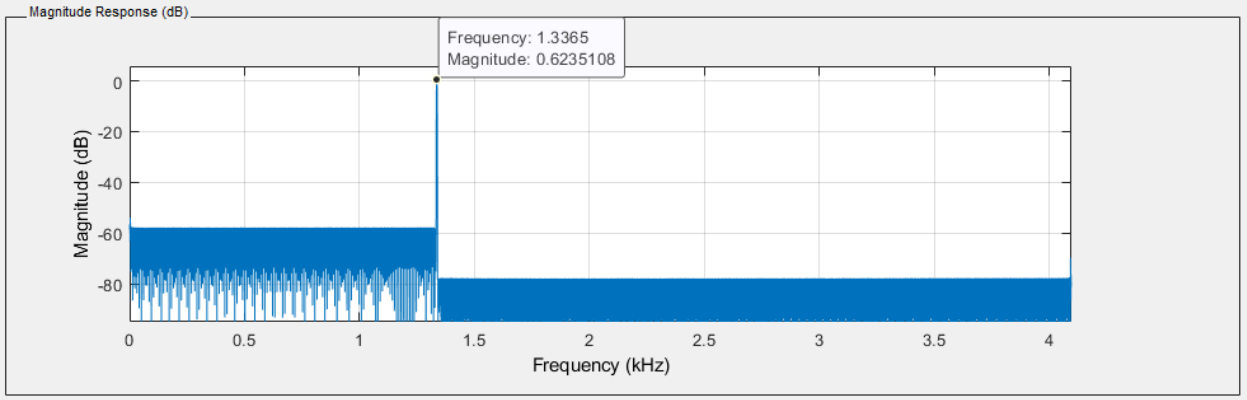
- 852 Hz:



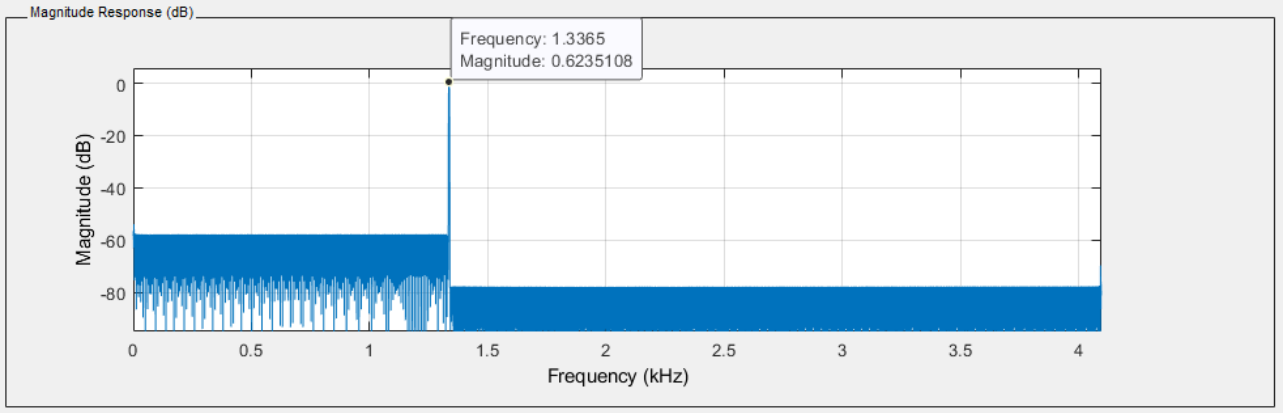
- 941 Hz:



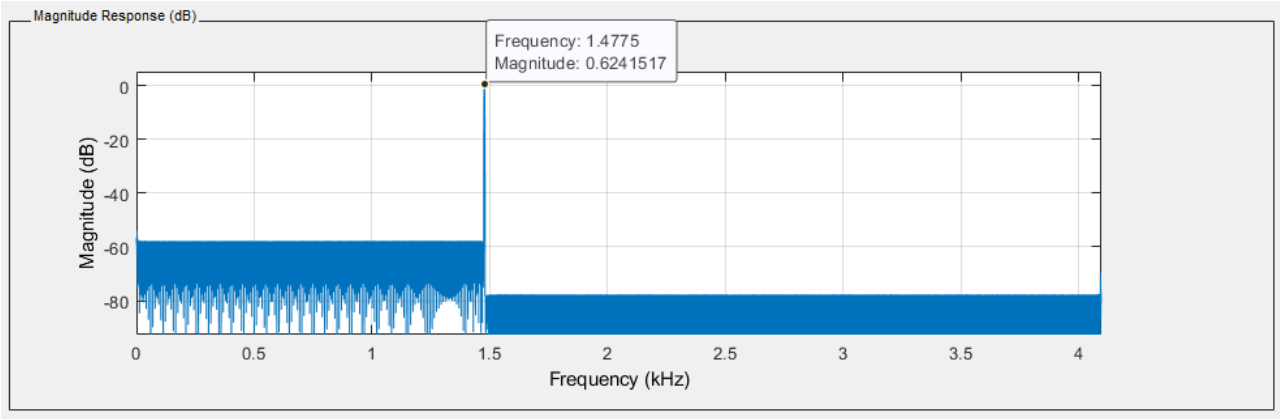
- 1209 Hz:



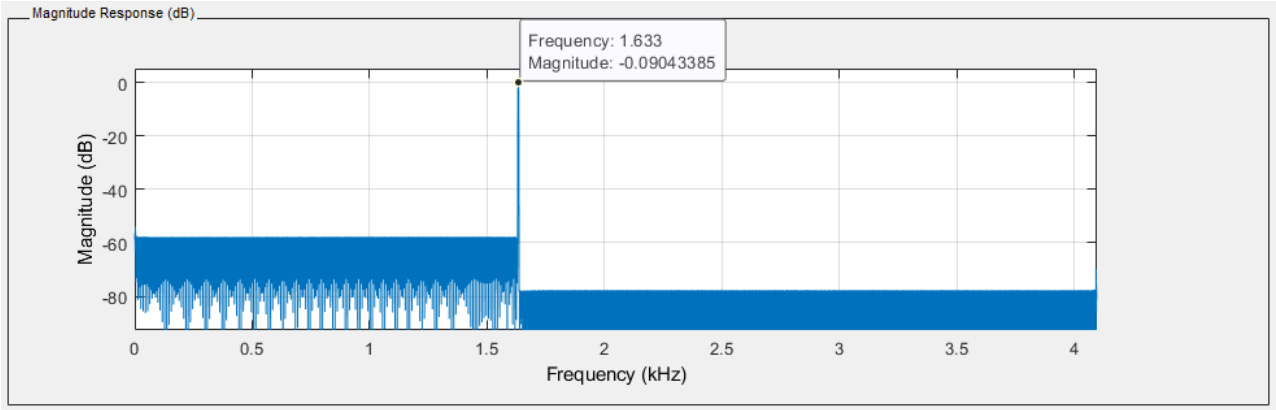
- 1336 Hz:



- 1447 Hz:



- 1633 Hz:



- Signals before and after applying the filters:

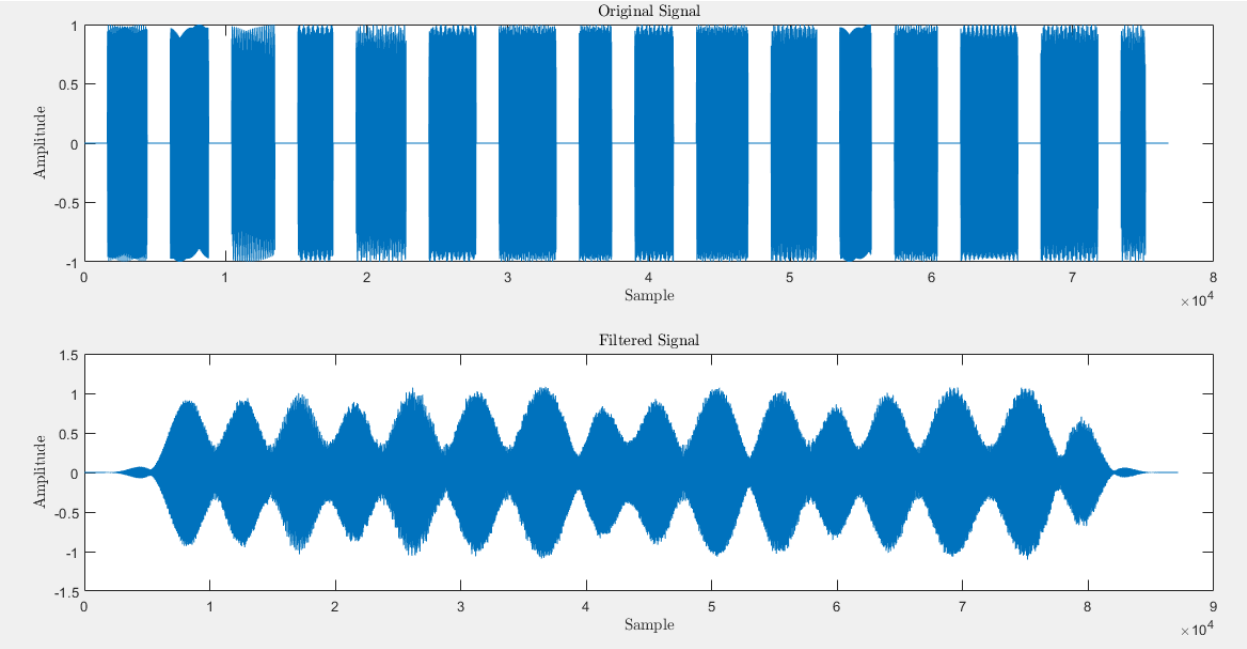


Figure 1: No Noise audio before and after the bandpass filters

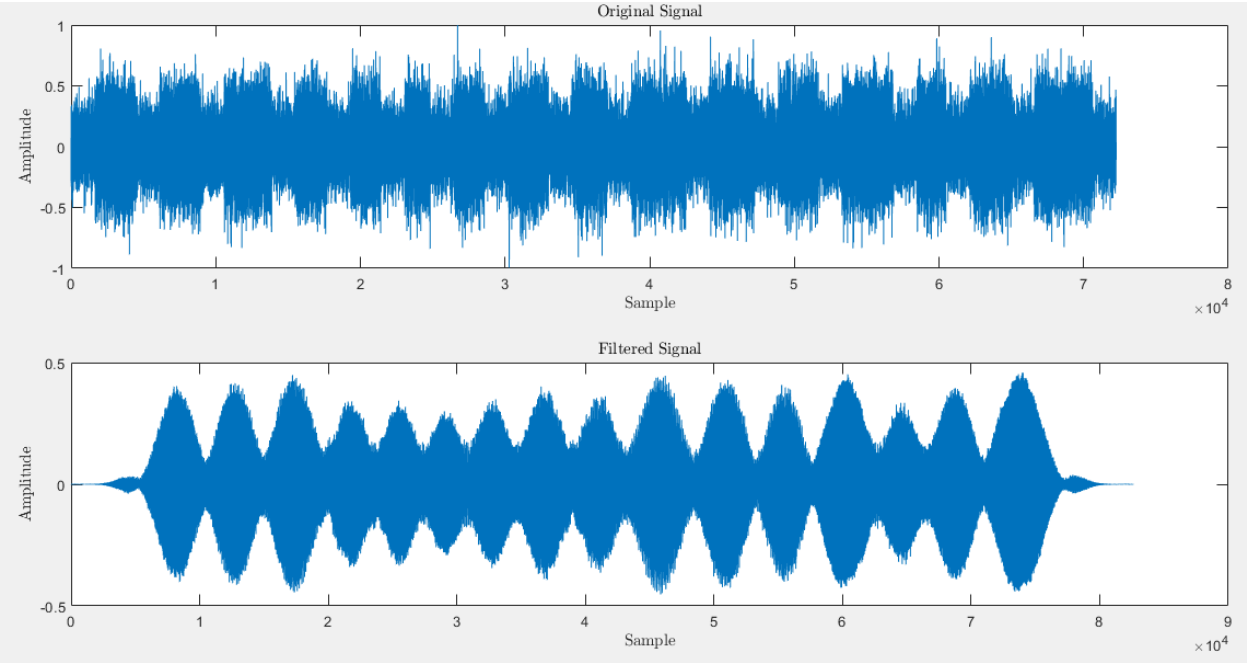


Figure 2: 00db Noise audio before and after the bandpass filters

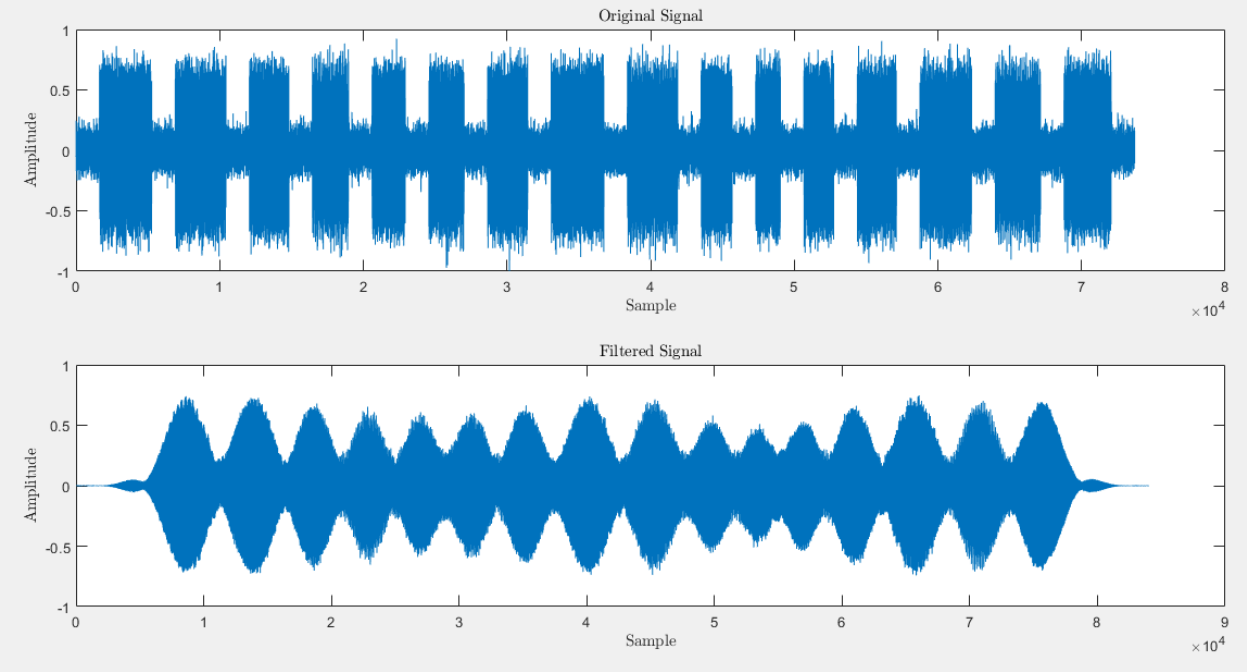


Figure 3: 10db Noise audio before and after the bandpass filters

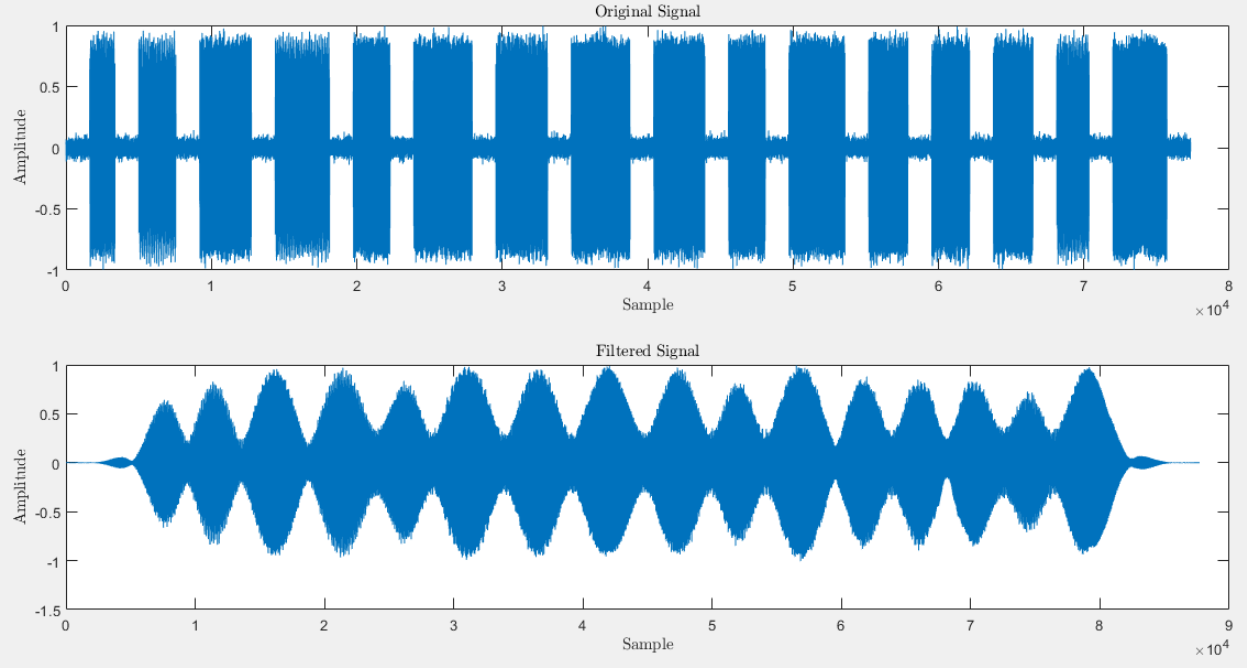


Figure 4: 20db Noise audio before and after the bandpass filters

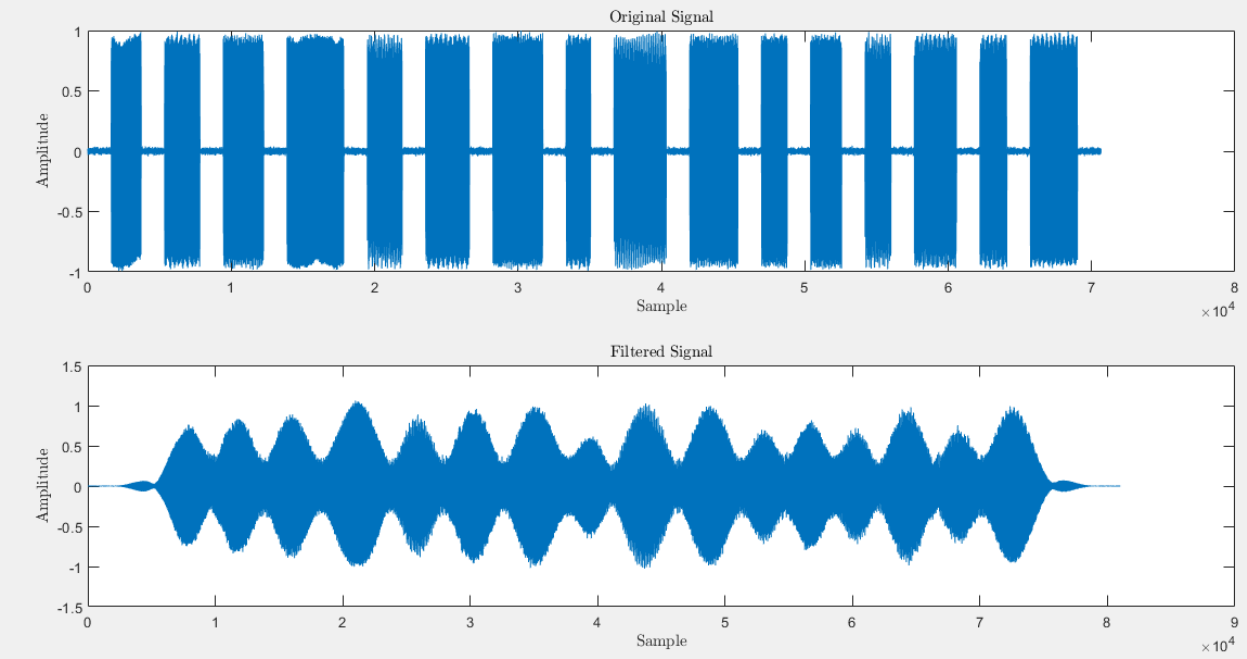


Figure 5: 30db Noise audio before and after the bandpass filters

- Algorithm:
  After applying the filters to the signal, we have 8 different signals which each of them has a specific frequency.

```
% apply the filters
bp697 = conv(BP_697, audio);
bp770 = conv(BP_770, audio);
bp852 = conv(BP_852, audio);
bp941 = conv(BP_941, audio);
bp1209 = conv(BP_1209, audio);
bp1336 = conv(BP_1336, audio);
bp1477 = conv(BP_1477, audio);
bp1633 = conv(BP_1633, audio);
```

Now we set 8 thresholds for each of this signals and check whether a key is pressed or not and if yes, save time of the pressing. The point is you have to change the threshold for diffrenet audio inputs to get the exact correct output. For No Noise audio, 10db noise, 20db noise and 30db noise thresholds are:

```
bp697_pressed = find(bp697 > 0.15);
bp770_pressed = find(bp770 > 0.15);
bp852_pressed = find(bp852 > 0.15);
bp941_pressed = find(bp941 > 0.15);
bp1209_pressed = find(bp1209 > 0.15);
bp1336_pressed = find(bp1336 > 0.15);
bp1477_pressed = find(bp1477 > 0.15);
bp1663_pressed = find(bp1633 > 0.15);
```

And for 00db noise:

```
bp697_pressed = find(bp697 > 0.15);
bp770_pressed = find(bp770 > 0.15);
bp852_pressed = find(bp852 > 0.1);
bp941_pressed = find(bp941 > 0.15);
bp1209_pressed = find(bp1209 > 0.1);
bp1336_pressed = find(bp1336 > 0.15);
bp1477_pressed = find(bp1477 > 0.15);
bp1663_pressed = find(bp1633 > 0.15);
```

This thresholds can be changed and they`re different for different inputs. You can estimate the threshold by just checking the plots of the filtered signals.
After this we have to find intersections of the times which each frequency has a high amplitude for the keys. Then get the minimum of each of the vectors and that would be start of where the specified frequency has a high amplitude and that would be independent to the time of pressing key. After this we will sort the times and now we have the ordered sequence of the pressed keys.

```
% find intersection of the sets for each key
one_pressed = min(intersect(bp697_pressed, bp1209_pressed));
two_pressed = min(intersect(bp697_pressed, bp1336_pressed));
three_pressed = min(intersect(bp697_pressed, bp1477_pressed));
A_pressed = min(intersect(bp697_pressed, bp1663_pressed));

four_pressed = min(intersect(bp770_pressed, bp1209_pressed));
five_pressed = min(intersect(bp770_pressed, bp1336_pressed));
six_pressed = min(intersect(bp770_pressed, bp1477_pressed));
B_pressed = min(intersect(bp770_pressed, bp1663_pressed));

seven_pressed = min(intersect(bp852_pressed, bp1209_pressed));
eight_pressed = min(intersect(bp852_pressed, bp1336_pressed));
nine_pressed = min(intersect(bp852_pressed, bp1477_pressed));
C_pressed = min(intersect(bp852_pressed, bp1663_pressed));

star_pressed = min(intersect(bp941_pressed, bp1209_pressed));
zero_pressed = min(intersect(bp941_pressed, bp1336_pressed));
hashtag_pressed = min(intersect(bp941_pressed, bp1477_pressed));
D_pressed = min(intersect(bp941_pressed, bp1663_pressed));
% order of pressed keys
Order = sort([one_pressed, two_pressed, three_pressed, A_pressed, ...
    four_pressed, five_pressed, six_pressed, B_pressed, seven_pressed, ...
    eight_pressed, nine_pressed, C_pressed, star_pressed, zero_pressed, ...
    hashtag_pressed, D_pressed]);
```

Decoded outputs:

No Noise

Command Window

4#206A78513*9DBC>>

00db

Command Window

A13D475BC296#80*>>

10db

Command Window

D176A354C90#8B2*>>

20db

Command Window

325640B*9187ADC#>>

30db

Command Window

#A7*60452918C3BD>>

1.2- Visualized output:

This is a very easy part. We just have to pass the outputs to a function which can recognize each character and with some if s we will put the photos together using **montage** function in MATLAB.

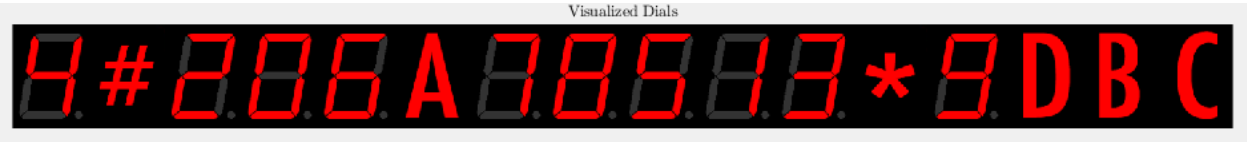


Figure 1: No Noise audio

Figure 2: 00db Noise audio

Figure 3: 10db Noise audio

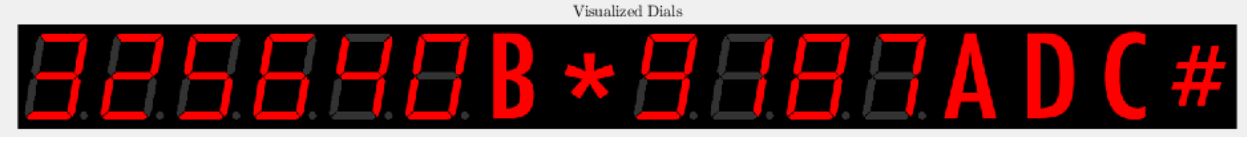


Figure 4: 20db Noise audio

Figure 5: 30db Noise audio

1.3- DTMF_Encoder:

What we have to do is just summing two sin functions with 2 different frequencies depends on the key which is pressed and add these signals together and at the end save the music with the **audiowriter** function:

$$Sound = \sin(2\pi f_i t/fs)$$

$$t = [(i-1) \times time\_pressed \times fs+1: i \times time\_pressed \times fs]$$

By putting these signals together, we`ll have our encoded audio. For example I`ve tried this for the input, 1234567890*#ABCD, and the 'output_dialing.wav' is the result. After that I gave it to my decoder to check and the output is:(set new thresholds)

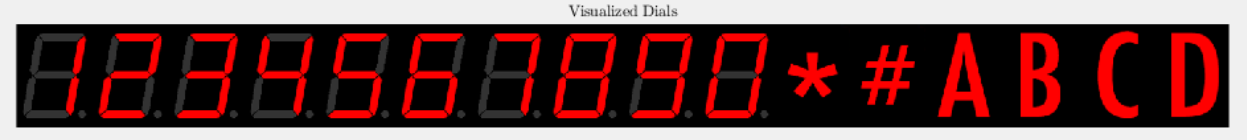


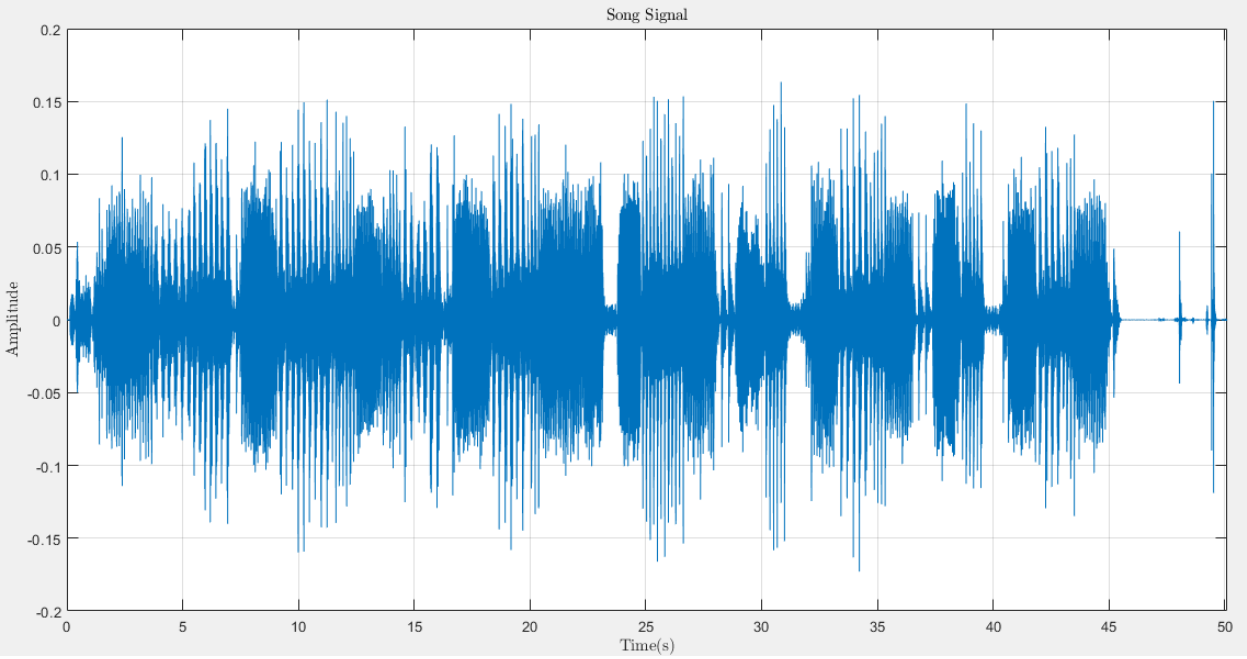
Figure 1: my audio

```
bp697_pressed = find(bp697 > 0.2);
bp770_pressed = find(bp770 > 0.2);
bp852_pressed = find(bp852 > 0.25);
bp941_pressed = find(bp941 > 0.2);
bp1209_pressed = find(bp1209 > 0.2);
bp1336_pressed = find(bp1336 > 0.2);
bp1477_pressed = find(bp1477 > 0.25);
bp1663_pressed = find(bp1633 > 0.2);
```

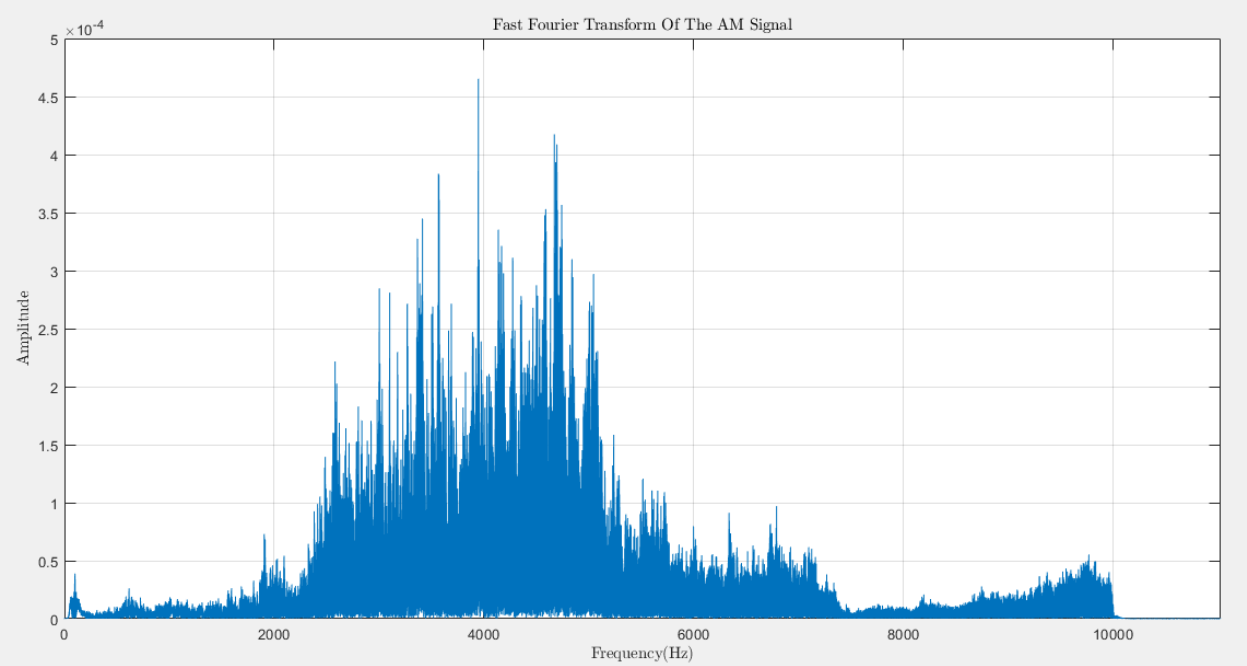
Figure 2: thresholds for this input

٢- Bird Song:

٢.١- Time domain plot:



٢.٢- Fourier Transform using FFT:

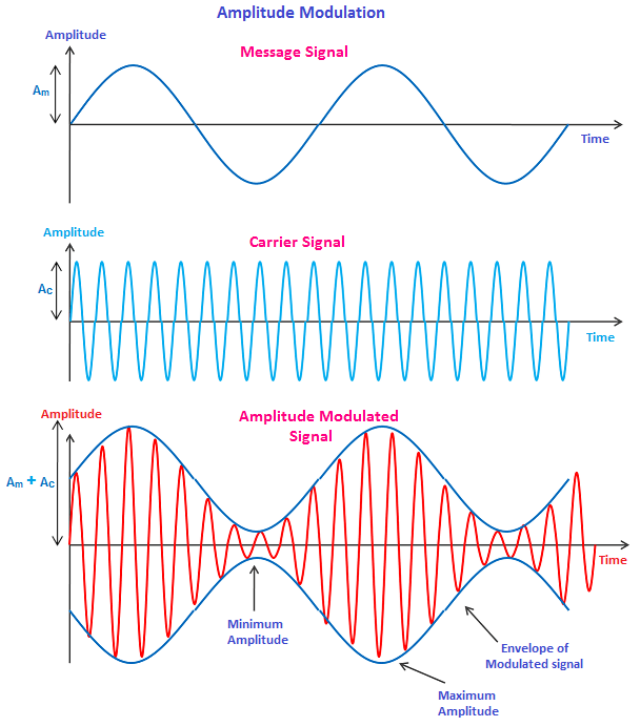


If we look at the power spectrum, it can be realized that most energy of the signal is in the range of about 2000Hz to 7000Hz. So we calculate the power of that range and the total power of the signal using **bandpower** function in MATLAB and we have:

```
SelectedRangePower =

   4.9435e-04


TotalPower =

   5.0600e-04


Percentage =

   97.6980
```

As we predicted, 97.7% of the signal`s energy is in this frequency range.

۲.۳ – Remove AM modulation:



A radio signal can carry audio or other information for broadcasting or other things, only if the it is modulated or changed in another way. One of the easiest methods is Amplitude modulation. In this method, the amplitude of the radio signal will change during the time and it represents the message signal in it. Some of AM modulation applications are:

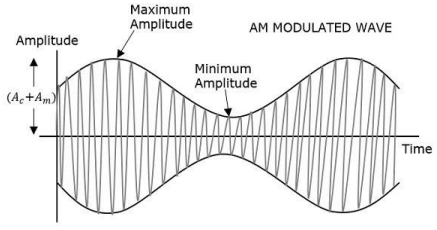
- Broadcast transmissions, Air band radio, Quadrature amplitude modulation, etc.

This method has advantages and disadvantages:

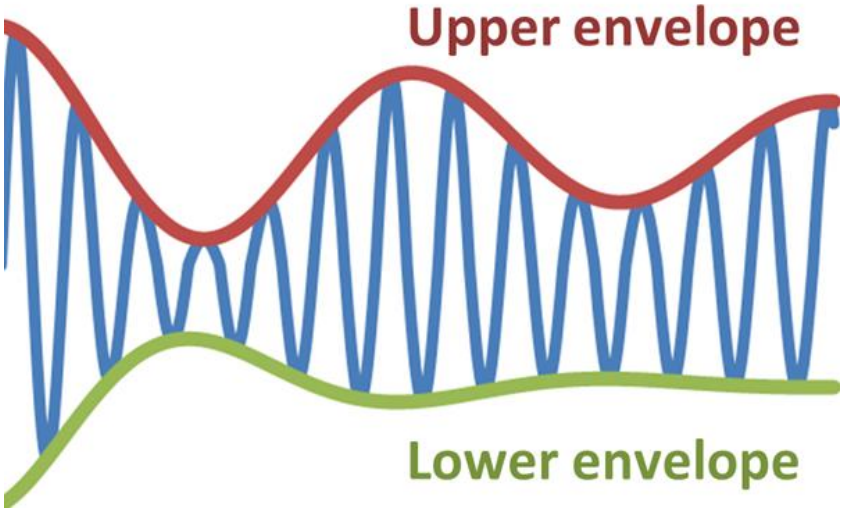
- Advantages:
  - Easy implementation
  - We can demodulate the modulated signal
  - AM receivers are very cheap since nothing specialized is needed
- Disadvantages:
  - Not efficient because of its power usage
  - It needs a band width equals to twice of highest audio frequency
  - Sensitive to noise

How it will be applied to the signals?

Usually, A low frequency sine wave which is our message signal will be manipulated by a higher frequency sine wave which is called carrier signal. Carrier signal doesn't contain any information. After summing these signals, we have our AM modulated signal with a domain of Amessage+Acarrer and Amessage-Acareer.

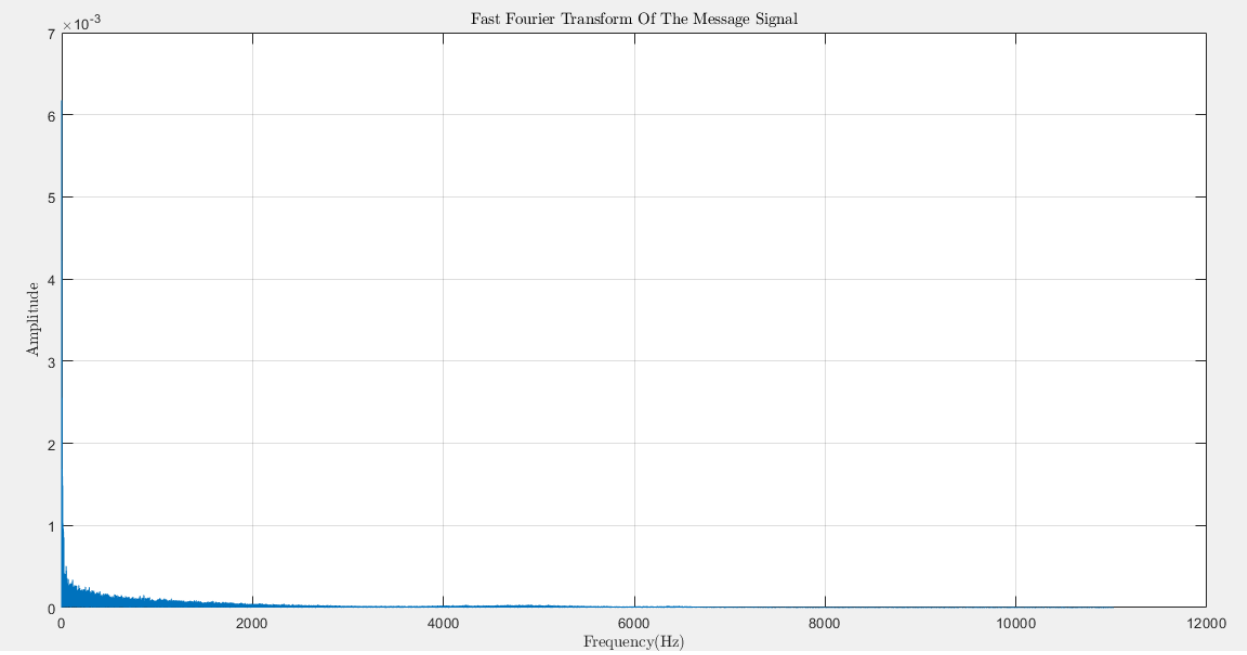


We use **envelope** function in MATLAB to get the envelopes of the modulated signal:



The message signal would be the lower_envelope - mean(lower_envelope) or higher_envelope - mean(higher_envelope).

Here we have the FFT of the unmodulated signal:



We can see that most of the energy is between something about what we expected, 6 to 40 Hz.