q1

March 12, 2018

1 Implement 2D Fast Fourier Transform

- Implemented using FFT in 1D
- Using the implemented 1D fft appy it on all rows
- Implement FFT on the output from all the rows for the desired output

1.0.1 FFT in one 1D

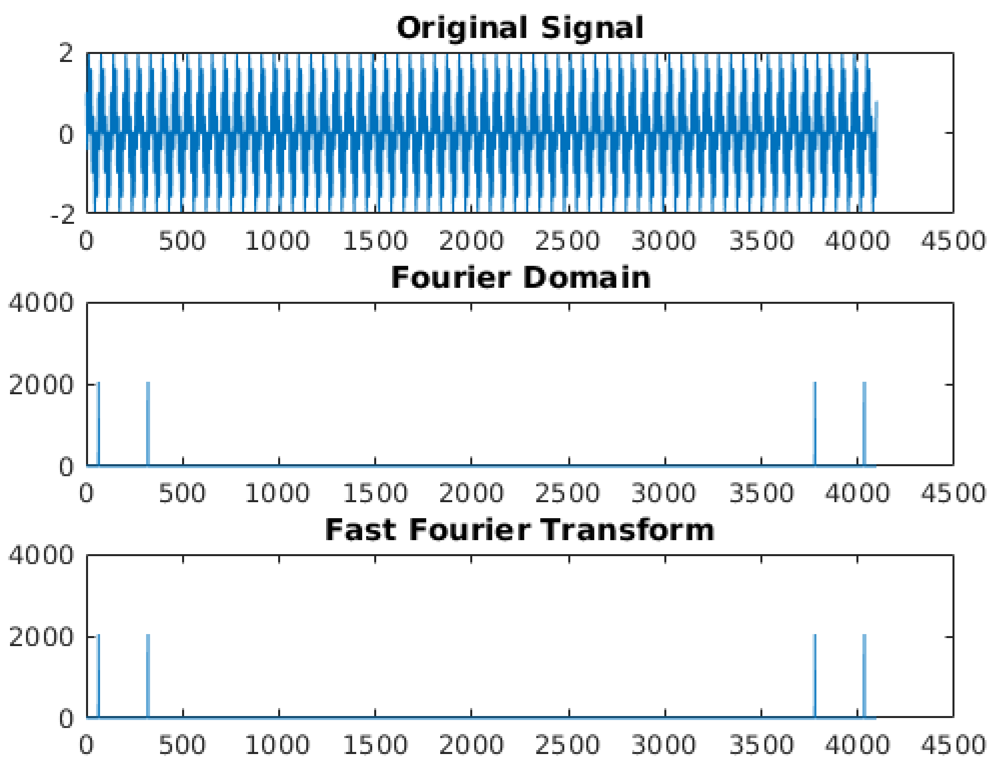
```
function [out_sig] = NEWFFT(in_sig,k,N)
    len = size(in_sig,2)
    out_sig = zeros(1,len);
    if len == 1
        out_sig = in_sig;

    else
        Ok = in_sig(1:2:len);
        Ek = in_sig(2:2:len);

        FOk = NEWFFT(Ok,k,N/2);
        FEk = NEWFFT(Ek,k,N/2);
        size(FOk)
        size(FEk)
        f1 = FEk + exp(-1i*2*pi*k/N).*FOk;
        f2 = FEk - exp(-1i*2*pi*k/N).*FOk;

        out_sig = cat(2,f1,f2);
    end
end
```

1.0.2 Using FFT in 1D example

```
In [7]: % Sampling Frequency
        fs = 64;
        % Frequency
        N = 64;
        n = 0:N*fs -1/fs;
```

```
t = 2*pi*[0:1/fs:N-1/fs];
y = sin(t) + cos(5*t);

% Hence the formula of Fourier transform is
ft_mat = exp(-1i*2*pi*n'*n/(N*fs));
y_ft = y*ft_mat;
```

In [11]:
```
figure;
subplot(3,1,1);
plot(y);
title('Original Signal');

subplot(3,1,2);
plot(abs(y_ft));
title('Fourier Domain');

% Using FFT
y_fft = NEWFFT(y,n,N*fs);
subplot(3,1,3);
plot(abs(y_fft));
title('Fast Fourier Transform');
```



2 Implementing FFT for 2D

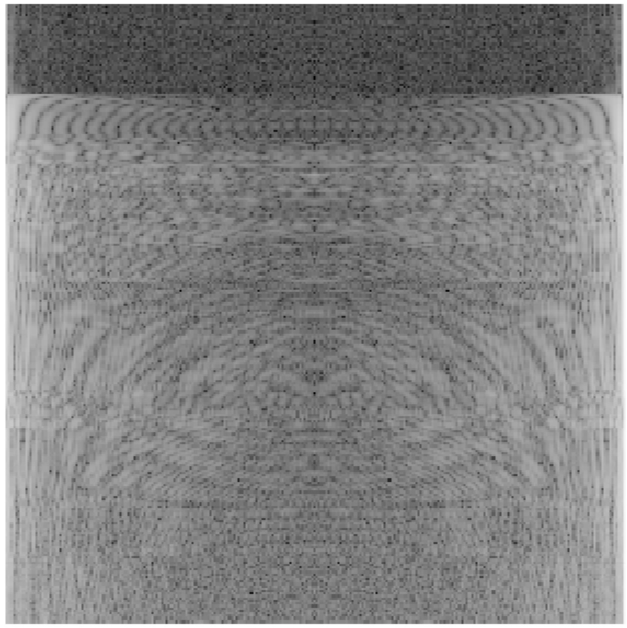
2.0.1 Function

```
function [out_image] = NEW_FFT2(in_image)
    N = size(in_image,1);
    M = size(in_image,2);
    out_image = zeros(N,M);
    for n=N
        size(out_image(n,:))
        out_image(n,:) = NEWFFT(in_image(n,:),[1:M],N);
    end

    out_image = abs(out_image);
    out_image = log(out_image + 1);
    out_image = mat2gray(out_image);
    imshow(out_image)



end
```

2.0.2 Test our function on multiple images

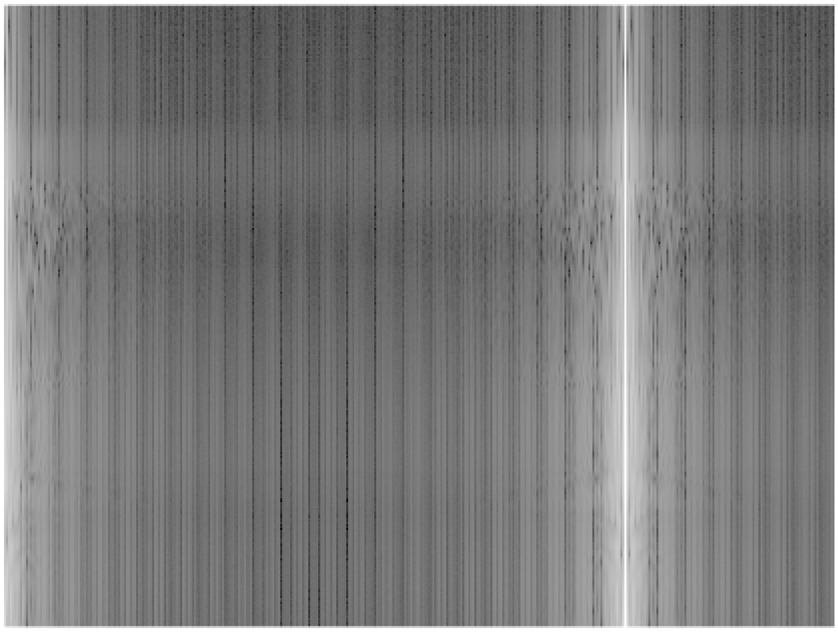
```
In [1]: img = imread('./cameraman.png');
        imshow(img);
        img = double(img);
```

```
In [2]: % Transform the image
        new_img = NEW_FFT2(img);
```

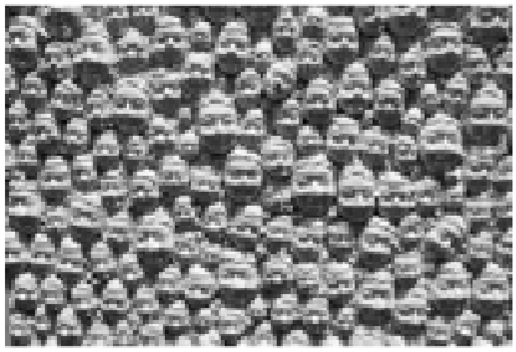


```
In [2]: img = imread('./blur.jpg');
        img = rgb2gray(img);
        img = imresize(img,1/3);
        imshow(img);
        img = double(img);
```

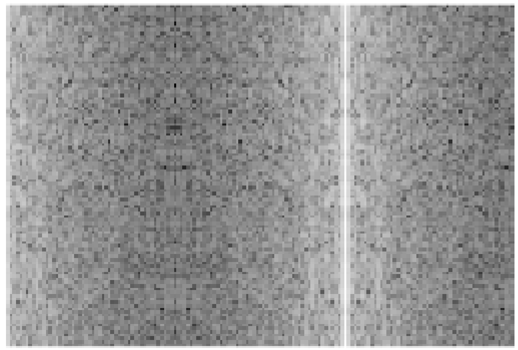
```
In [3]: new_img = NEW_FFT2(img);
```



```
In [3]: img = imread('./Faces.jpg');
        img = rgb2gray(img);
        img = imresize(img,1/4);
        imshow(img);
        img = double(img);
```



```
In [5]: new_img = NEW_FFT2(img);
```



q2

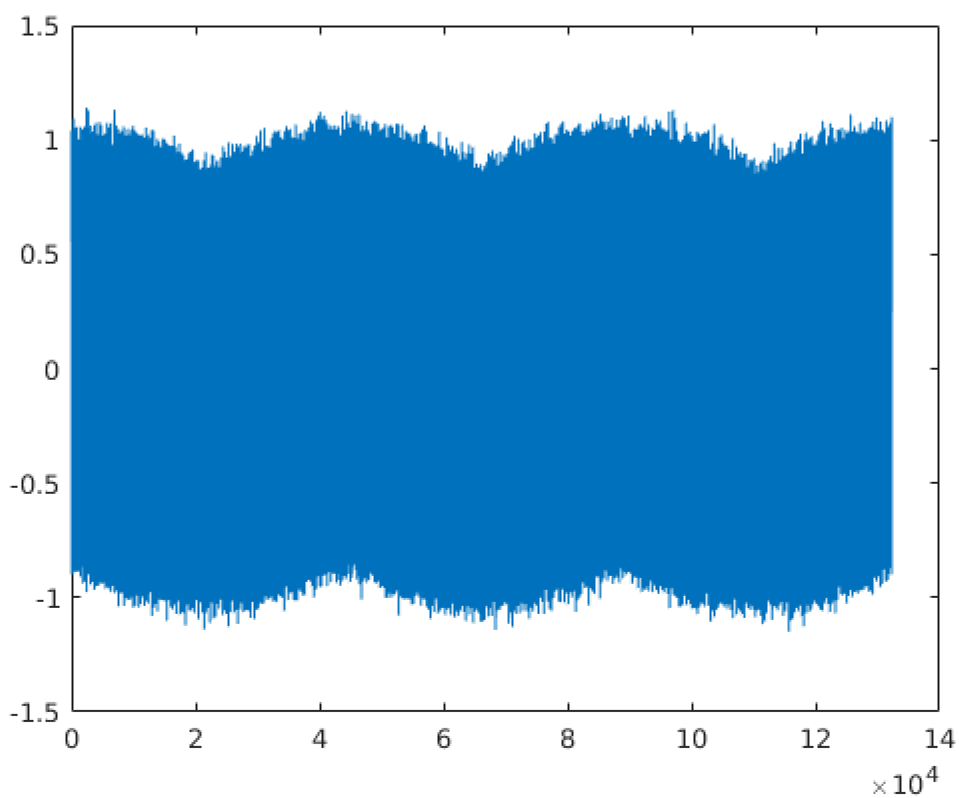
March 12, 2018

1 Removing Noise from a signal

- Our goal is to remove the noise added to the two tone telephones sinusoids
- By taking the fourier transform of the signal
- Find the main frequencies
- Use a rect filter and take an ifft of the smooth signal

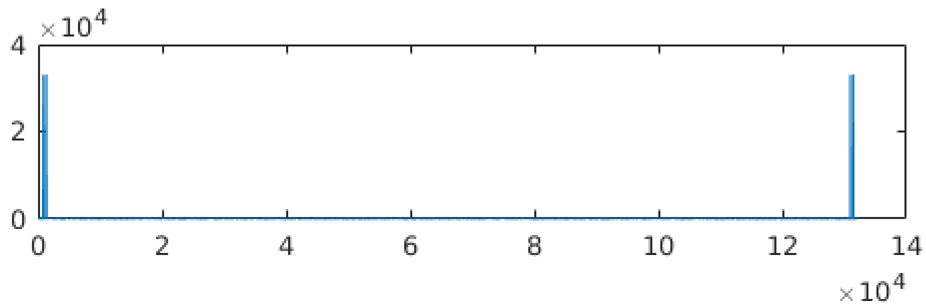
1.0.1 Load and play the original audio

```
In [11]: load('./q2.mat')
         audio_x = audioplayer(X,41400);
         play(audio_x);
         plot(X);
```



1.0.2 Compute the fourier transform and plot it.

```
In [2]: f_x = fft(X);
        plot(abs(f_x));
        pbaspect([5 1 1]);
```



1.0.3 We notice 4 peaks in the plot

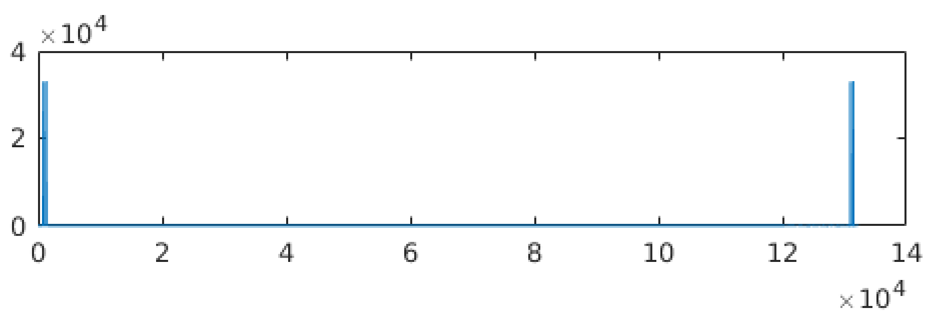
```
In [3]: [val,f] = sort(abs(f_x));
        val(end-4:end),f(end-4:end)
```

```
ans =

   1.0e+04 *

    0.0339
    3.3068
```

```
    3.3068
    3.3082
    3.3082


ans =

      130981
        1321
      130982
         882
      131421
```

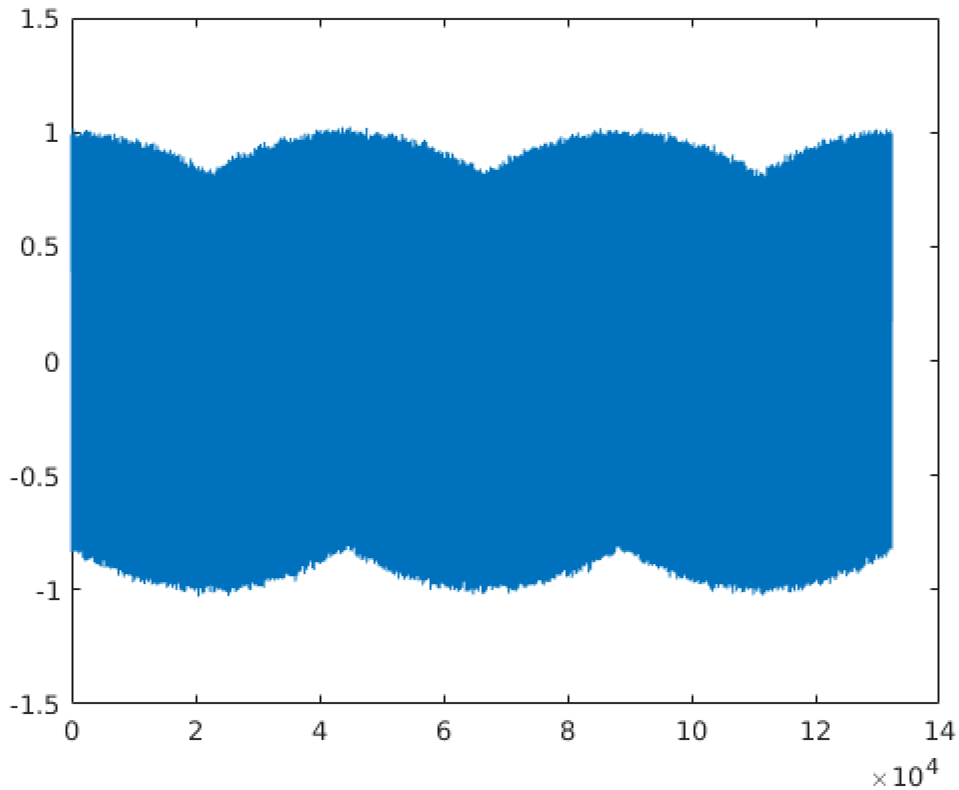
1.0.4 Multiply with a rect function for bandwidth 700-1400 , 11000-end to remove noise

```
In [5]: rect = zeros(size(f_x));
        rect(700:1400) = 1;
        rect(end-12000:end) = 1;
        plot(abs(f_x.*rect));
        pbaspect([5 1 1])
```



1.0.5 Do inverse fourier transform

```
In [6]: y = ifft(f_x.*rect);
        plot(real(y));
```



1.0.6 Convert to audio

```
In [12]: audio_r = audioplayer(real(y),Fs);
         audiowrite('result.wav',real(y),Fs);
         play(audio_r);
```

```
Warning: Data clipped when writing file.
> In audiowrite>clipInputData (line 396)
  In audiowrite (line 176)
```

q3

March 12, 2018

1 Create a spectrogram

1.0.1 Function to calculate spectrogram

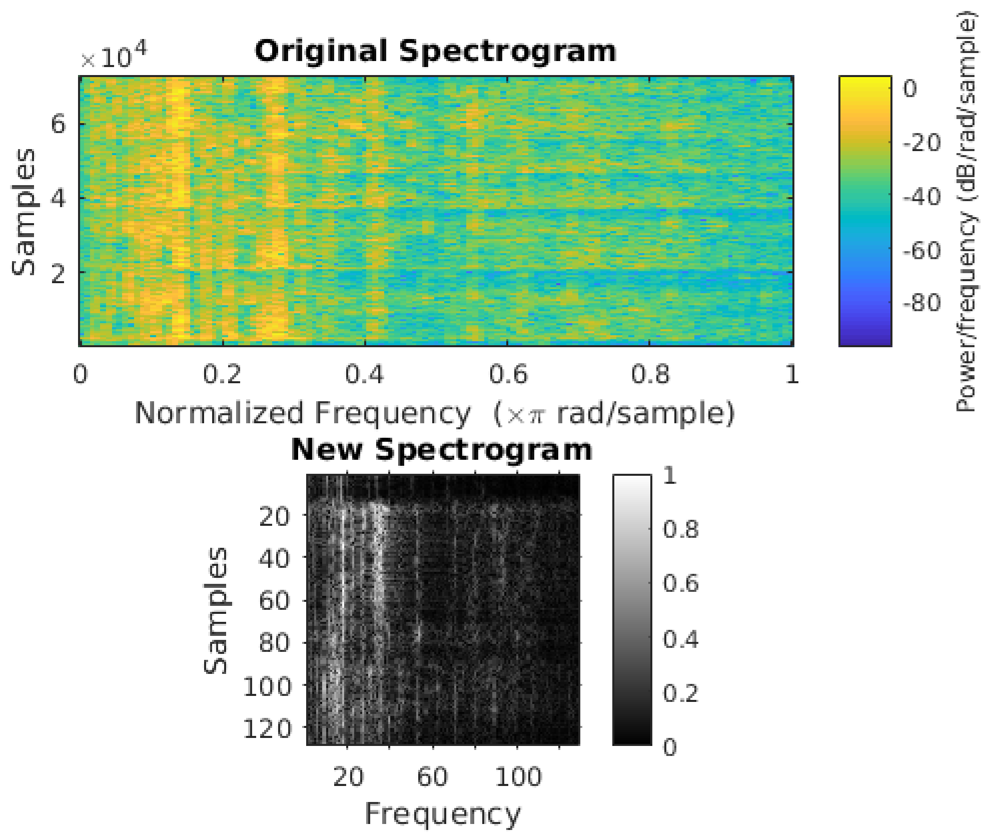
```
% Create a simple spectrogram
% Input
%   X - initial signal
%   W - window length
%   s - stride
% Output
%   result - image produced by the spectrogram

function [result] = SPECTROGRAM(y,W,s)
    num_iter = int16((size(y,1) - W)/s) + 1;
    fin_img = zeros(num_iter,W+1);
    result = zeros(num_iter,int16(size(fin_img,2)/2));
    for i = [1:num_iter-2]
        f_y = fft(y(i*s:W + i*s));
        f_y = fftshift(f_y);
        fin_img(i,1:size(f_y)) = abs(f_y);
    end
    fin_img = log(fin_img +1);
    fin_img = mat2gray(fin_img);
    result = fin_img(1:min(size(result,1),int16(W/2)) ,int16(end/2) + 1:end);
    imshow(result);
    axis on;
    ylabel('Samples');
    xlabel('Frequency');
    colorbar;
    title('New Spectrogram');
end
```

1.0.2 Run on different sounds and compare image produced

1.handel

```
In [6]: load 'handel.mat';
        sound(y);
```

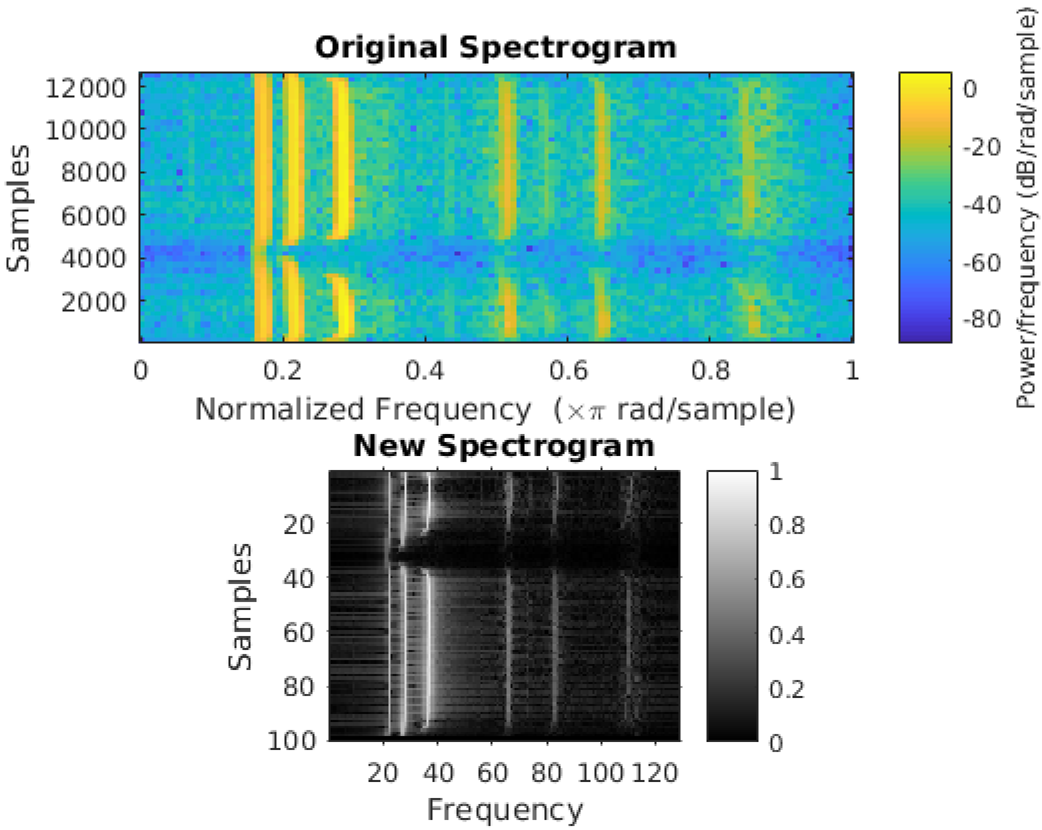
```
W = 256;
s = 128;

% Using inbuilt spectrogram
figure;
subplot(2,1,1);
spectrogram(y,W,W/s);
title('Original Spectrogram');

% Using my spectrogram
subplot(2,1,2);
fin_img = SPECTROGRAM(y,W,s);
```



2.Train

In [2]:
```
load train;
sound(y);

W = 256;
s = 128;
```

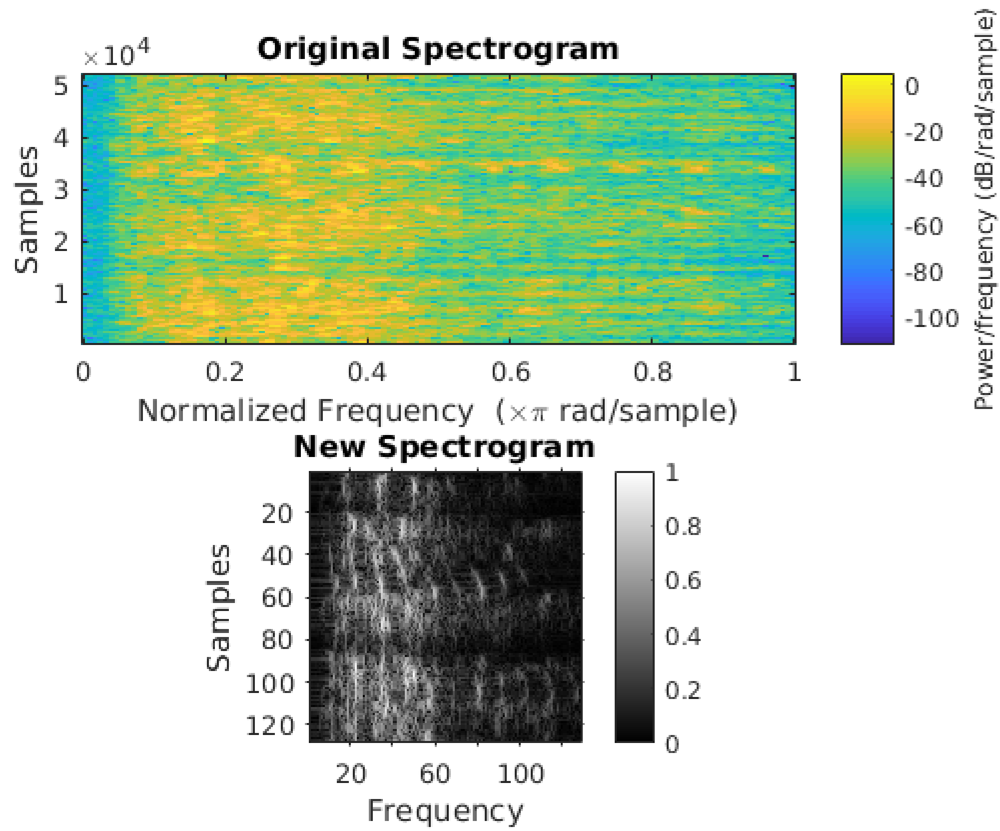
```
% Using inbuilt spectrogram
figure;
subplot(2,1,1);
spectrogram(y,W,W/s);
title('Original Spectrogram');

% Using my spectrogram
subplot(2,1,2);
fin_img = SPECTROGRAM(y,W,s);
```



3.Laughter

In [1]:
```
load laughter;
sound(y);

W = 256;
s = 64;

% Using inbuilt spectrogram
figure;
```

```
subplot(2,1,1);
spectrogram(y,W,W/s);
title('Original Spectrogram');

% Using my spectrogram
subplot(2,1,2);
fin_img = SPECTROGRAM(y,W,s);
```



1.0.3 Results

1.0.4 Window size

- Lesser window size => coarse image
- Wider window size => fine image
- because precision of fft reduces with decrease in N(width)

1.0.5 Stride

- More stride less width of spectrogram
- Less stride more time to compute

q4

March 12, 2018

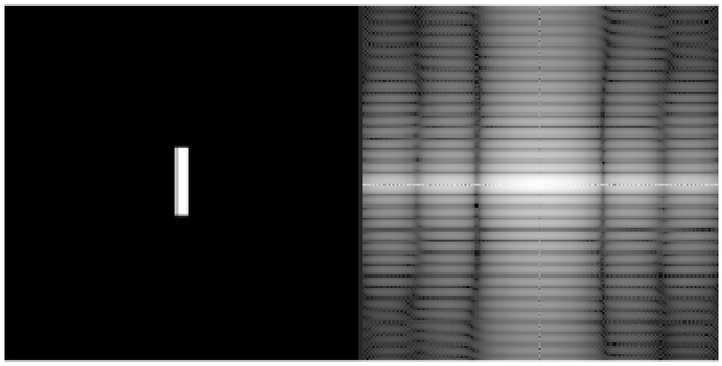
0.1 Apply Fourier transform on multiple images

0.1.1 For cleaner fft we do some transformations

```
function [f] = fourier_transform(img)
    f = fft2(img);
    f = fftshift(f);
    f = abs(f);
    f = log(f+1);
    f = mat2gray(f);
    imshowpair(img,f,'montage');

end
```

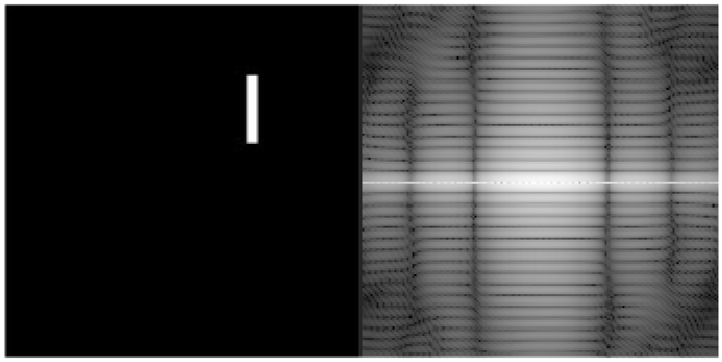
0.1.2 Img1a

```
In [2]: img = imread('./Img1a.png');
        fourier_transform(img);
```



0.1.3 Img1b

```
In [1]: img = imread('./Img1b.png');
        fourier_transform(img);
```

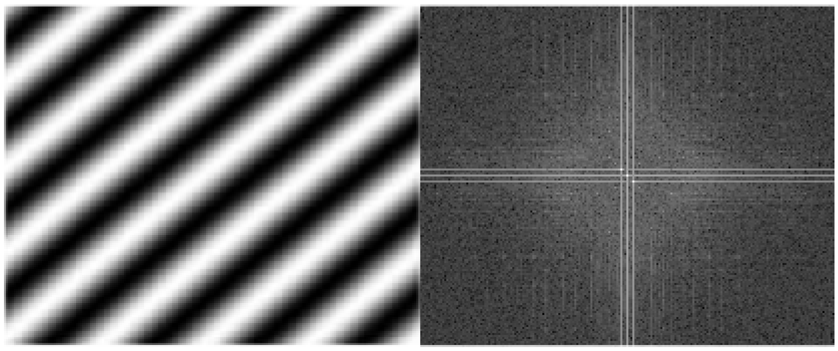


0.1.4 Intresting result

- We can notice that the rectangle shifted but the fourier domain of the image doesn't change
- Because fourier domain doesnt depend on the location of the wave

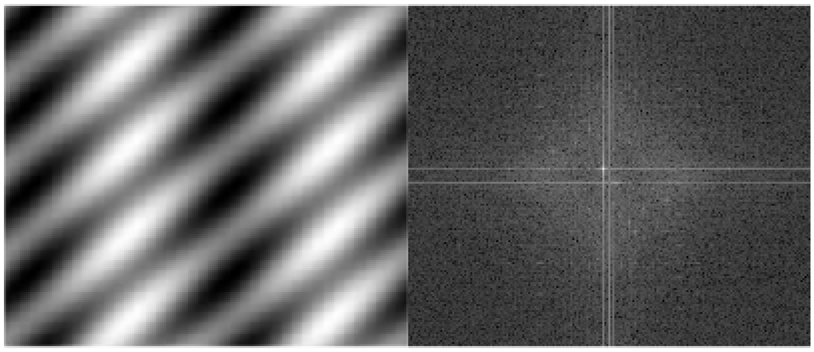
0.1.5 Img2a'

```
In [2]: img = imread('./Img2a.png');
        fourier_transform(img);
```



0.1.6 Img2b

```
In [3]: img = imread('./Img2b.png');
        fourier_transform(img);
```

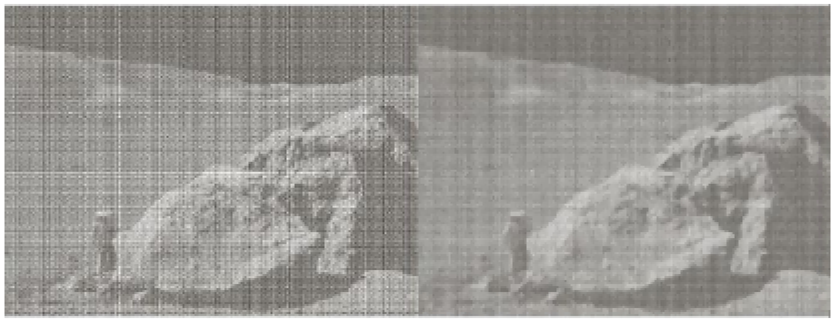


0.1.7 Result

- Bluring affects both time and fourier domain

0.2 Removing salt and pepper noise using median filter

In [1]:
```
img = imread('./Img3.png');
for i = 1:3
new_img(:,:,i) = medfilt2(img(:,:,i));
end
imgaussfilt(new_img,1.5);
imshowpair(img,new_img,'montage');
```



q5

March 12, 2018

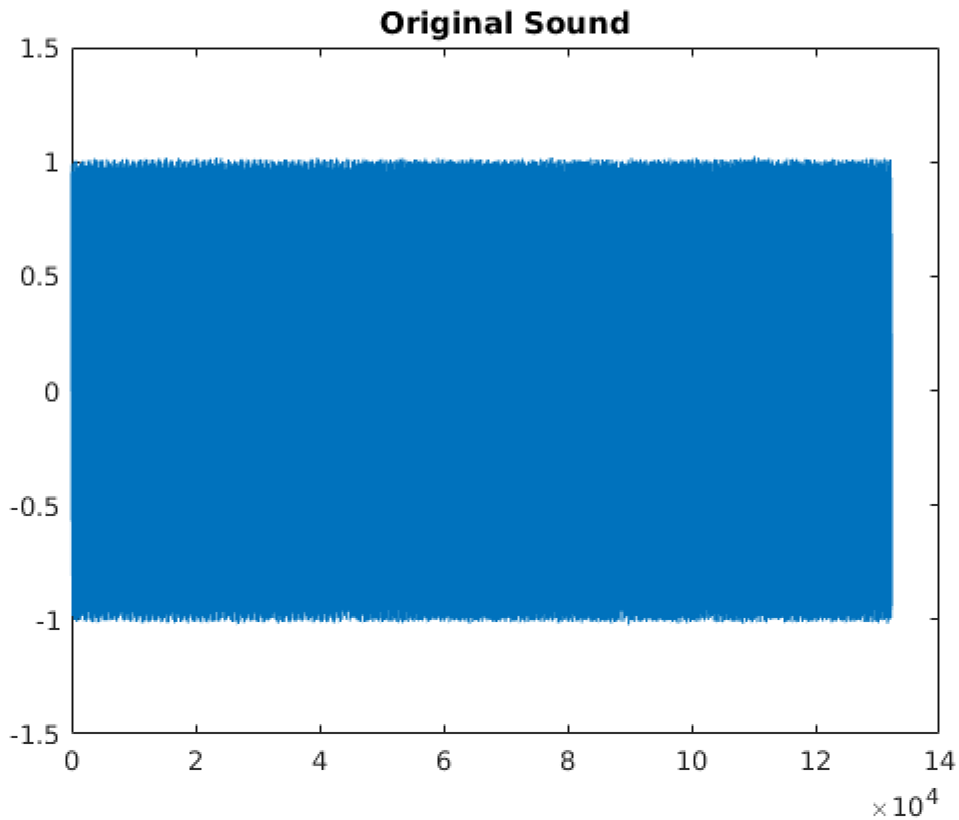
1 Detect number using dialtone

1.0.1 Function to Eavesdrop

```
function [result] = Eavesdrop(audio_file)

    info = audioinfo(audio_file);

%   Each number is assumed to be pressed for 1 sec hence num=length of number
    num =  info.Duration;


    X = audioread(audio_file);
    plot(X);
    title('Original Sound');
    step_length = info.SampleRate;

%     Get the audio files of each number and create a vector of their signal

    s_num = zeros(10,step_length);
    for i = [1:10]
    s = strcat('./',num2str(mod(i,10) ,'%2d'),'.ogg');
    a = audioread(s);
    s_num(i,:) = a(1:step_length);
    end

%   For every num check the dot product with each num, then take the maximum as the output
    result = 0;
    for i = [1:num]
        x = X((i-1)*step_length +1 : i*step_length);
        [val,max_num] = max(s_num*x);
        result = 10*result + mod(max_num,10);
    end
end
```

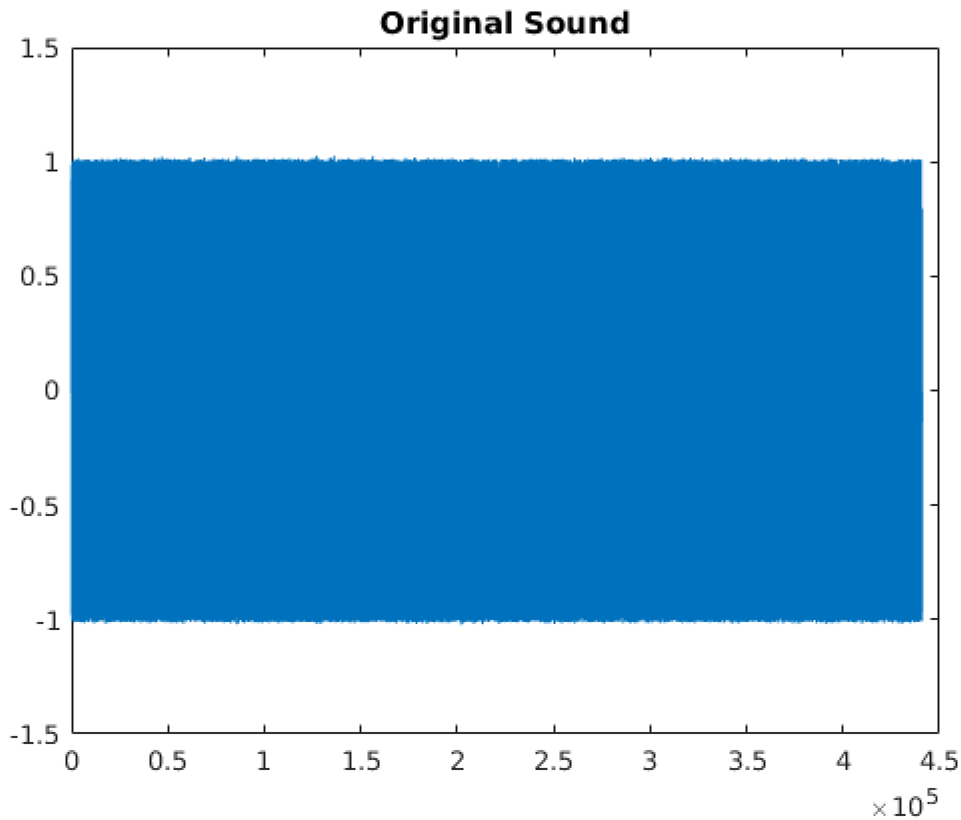
```
In [1]: result = Eavesdrop('./Police.ogg')
```

```
result =

   100
```




```
In [2]: result = Eavesdrop('./9515733002.ogg');
        sprintf('%10d\n',result )
```

```
ans =

    '9515733002
     '
```


Original Sound
1.5
1
0.5
0
-0.5
-1
-1.5
0
0.5
1
1.5
2
2.5
3
3.5
4
4.5
×10^5

q6

March 12, 2018

0.1 Decript signal using fourier transform

- Each message is shuffled in the fourier domain.
- We have to take the fft, permute and listen to each n

0.1.1 Function to compute permutation for each signal and check if its correct

```
function [right_perm] = Listen_perm(audiofile)

x = audioread(audiofile);

% Plot the orignal signal and its fft
figure;
subplot(2,1,1);
plot(x);
title('Original Sound');

% Only work in fourier domain
f_y = fft(x);
subplot(2,1,2);
plot(abs(f_y));
title('Ft of original');


% We do not need the conjugate
f_y = f_y(1:end/2);

% Divide the whole into 4 parts
pause
d = size(f_y,1)/4;
f_y = reshape(f_y,d,4);

or = perms([1,2,3,4]);
    for i = [1:24]
        or(i,:)
        % Permute and reshape to original shape
        new_x = perm_ord(f_y,or(i,:),d);
```

```
        subplot(2,1,2);
        plot(new_x);
        title('New Sound');

        sound(new_x,41400);
        pause
    end

    right_perm = [2,3,1,4]
end
```

0.1.2 Funtion to print the corect order

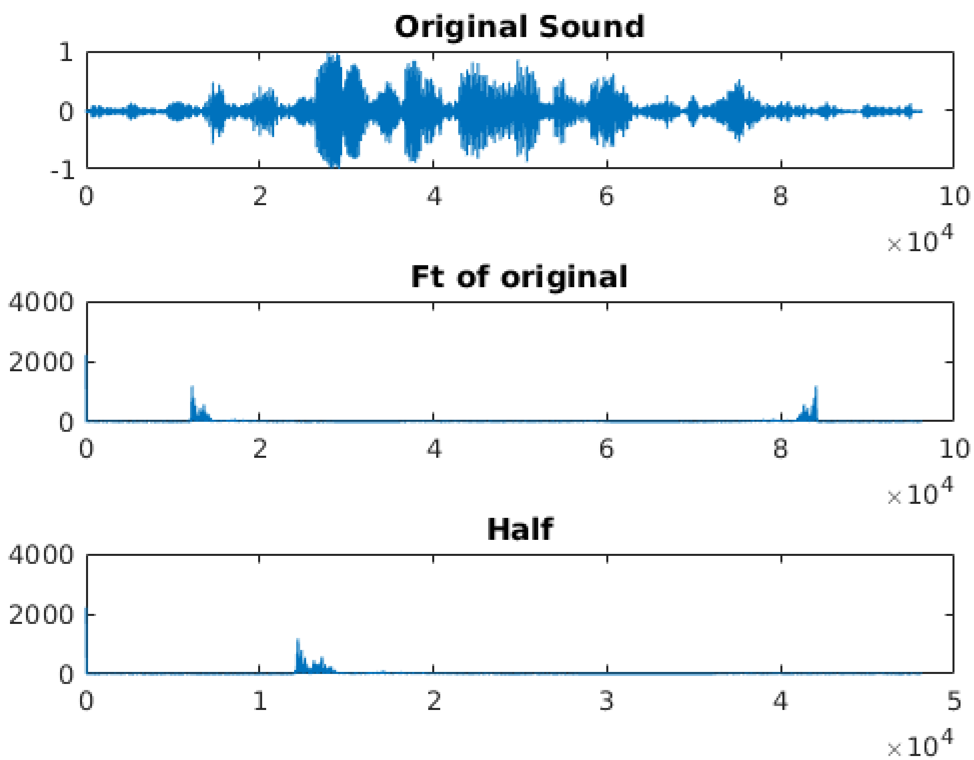
```
function [new_x] = perm_ord(f_y,ord,d)


        new_y = [f_y(:,ord(1)), f_y(:,ord(2)), f_y(:,ord(3)), f_y(:,ord(4))];
        new_y = reshape(new_y,d*4,1);

        % add the flip version
        y = zeros(2*size(new_y,1),1);
        y(1:end/2) = new_y;
        y(end/2 + 1:end) = conj(flipud(new_y(:)));

        figure;
        subplot(2,1,1);
        plot(abs(y));
        title('Frequency domain');

        new_x = real(ifft(y));
end
```

0.1.3 Message 1

=> If you are good at something never do it for free

```
In [39]: x = audioread('./message1.wav');

         figure;
         subplot(3,1,1);
         plot(x);
         title('Original Sound');
         % Only work in fourier domain
         f_y = fft(x);
         %
         subplot(3,1,2);
         plot(abs(f_y));
         title('Ft of original');
```

```
% We do not need the conjugate
f_y = f_y(1:end/2);
subplot(3,1,3);
plot(abs(f_y));
title('Half');
% Divide the whole into 4 parts
d = size(f_y,1)/4;
f_y = reshape(f_y,d,4);
```



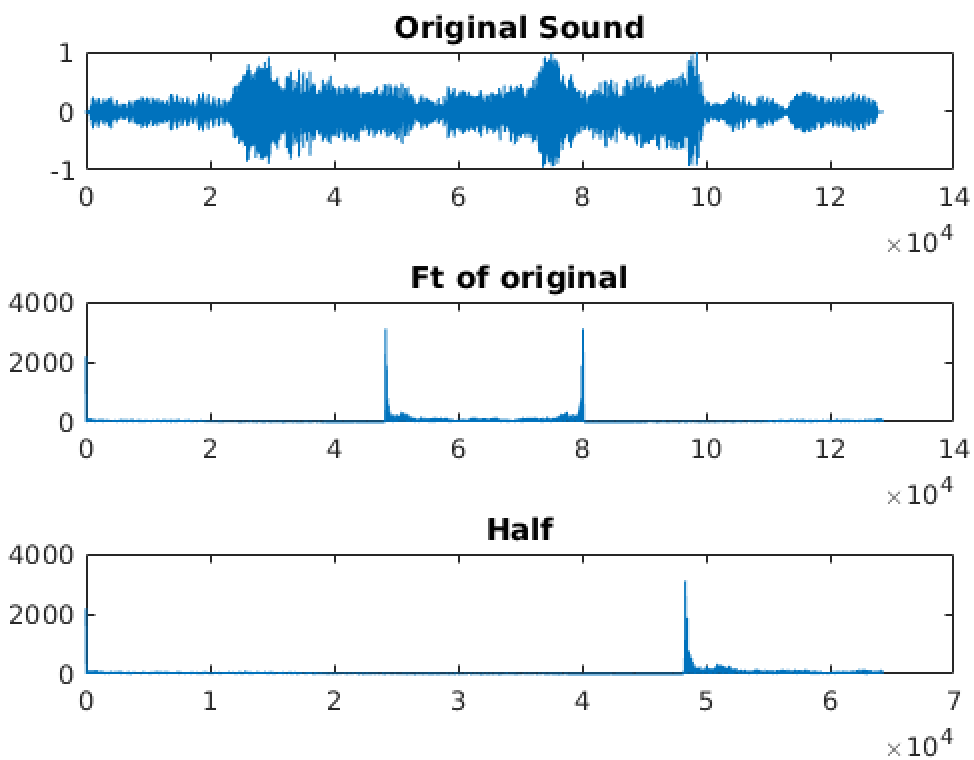

```
In [40]: ord = [2,3,3,4];
         new_x = perm_ord(f_y,ord,d);
         sound(new_x,43400);
```

0.1.4 Message 2

=> Why so serious

```
In [42]: x = audioread('./message2.wav');

         figure;
         subplot(3,1,1);
```

```
plot(x);
title('Original Sound');
% Only work in fourier domain
f_y = fft(x);
%
subplot(3,1,2);
plot(abs(f_y));
title('Ft of original');
% We do not need the conjugate
f_y = f_y(1:end/2);
subplot(3,1,3);
plot(abs(f_y));
title('Half');
% Divide the whole into 4 parts
d = size(f_y,1)/4;
f_y = reshape(f_y,d,4);
```

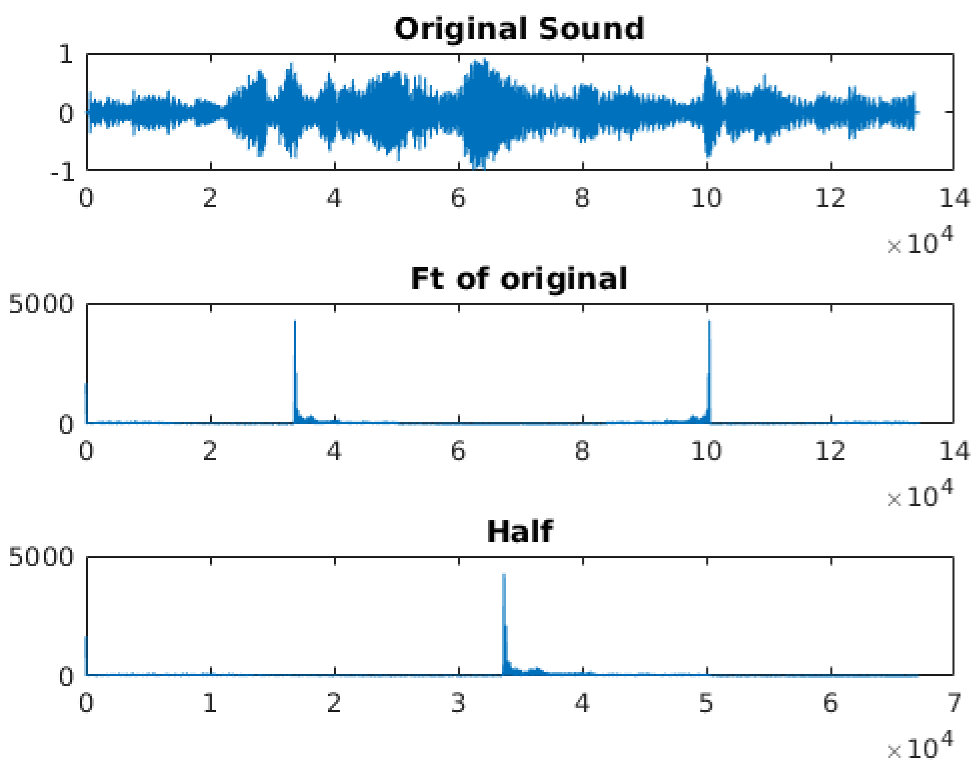



```
In [41]: ord = [4,3,2,3];
         new_x = perm_ord(f_y,ord,d);
         sound(new_x,43400);
```

0.1.5 Message 3

=> Lets put a smile on that face

In [56]:
```
x = audioread('./message3.wav');

figure;
subplot(3,1,1);
plot(x);
title('Original Sound');
% Only work in fourier domain
f_y = fft(x);
%
subplot(3,1,2);
plot(abs(f_y));
title('Ft of original');
% We do not need the conjugate
f_y = f_y(1:end/2);
subplot(3,1,3);
plot(abs(f_y));
title('Half');
% Divide the whole into 4 parts
d = size(f_y,1)/4;
f_y = reshape(f_y,d,4);
```



```
In [57]: ord = [3,2,2,2];
         new_x = perm_ord(f_y,ord,d);
         sound(new_x,43400);
         plot(new_x);
```