Signal and System Computer homework

Dr.Hamid Behroozi

department : Electrical Engineering

Amirreza Velae 400102222

Computer homework 1
Report

April 8, 2023

Signal and System Computer homework

Computer homework 1

Amirreza Velae 400102222

Plot some signals

Discrete signals

To plot given functions in Matlab, I defined a delta and heaviside functions for discrete signals and used it to plot the signals. The code is as follows:

```
n0 = 0;
n1 = 1;
n = -10:10;
delta = @(n) ((n-n0)==0);
myheaviside = @(n) ((n>=n0)==1);
```

Source Code 1: Delta and Heaviside function

Given signals are as follows:

$$x_1[n] = 4\delta[n+5] + 2\delta[n] - \delta[n-3] \tag{1}$$

Matlab code for plotting $x_1[n]$ is as follows:

```
x1 = 4*delta(n+5) + 2*delta(n)-delta(n-3);
stem(n,x1)
```

Source Code 2: First Signal Codes

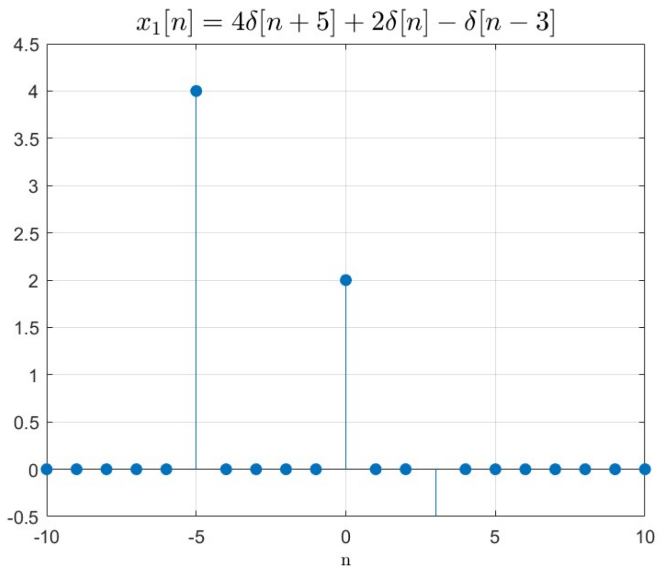

Figure 1: First Signal Graph

$$x_2[n] = (-0.8)^n u[n] \tag{2}$$

Matlab code for plotting $x_2[n]$ is as follows:

```
    x2 = (-0.8).^n .* myheaviside(n);
    stem(n,x2)
```

Source Code 3: Second Signal Code

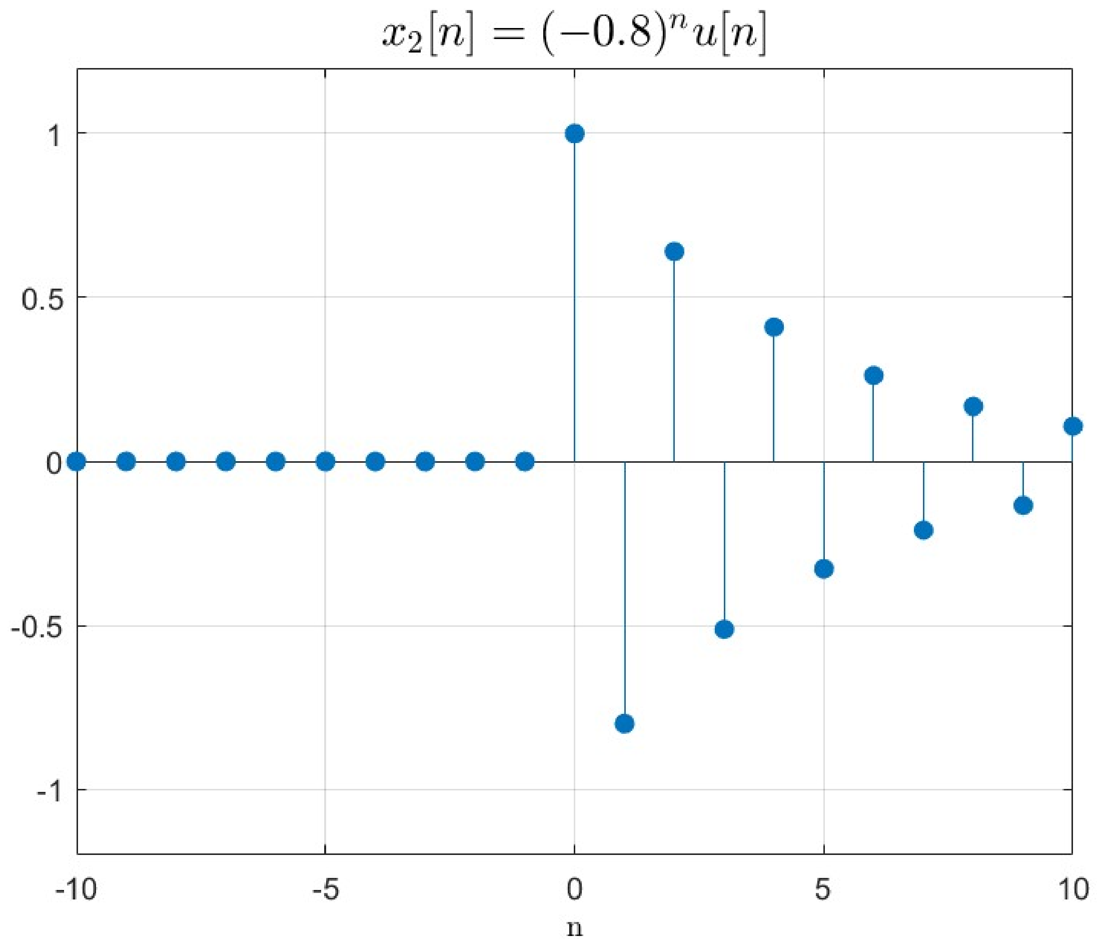

Figure 2: Second Signal Graph

$$x_3[n] = u[n-5] - u[n+5] \tag{3}$$

Matlab code for plotting $x_3[n]$ is as follows:

```matlab
    x3 = myheaviside(n-5) - myheaviside(n+5);
stem(n,x3)
```

Source Code 4: Third Signal Code

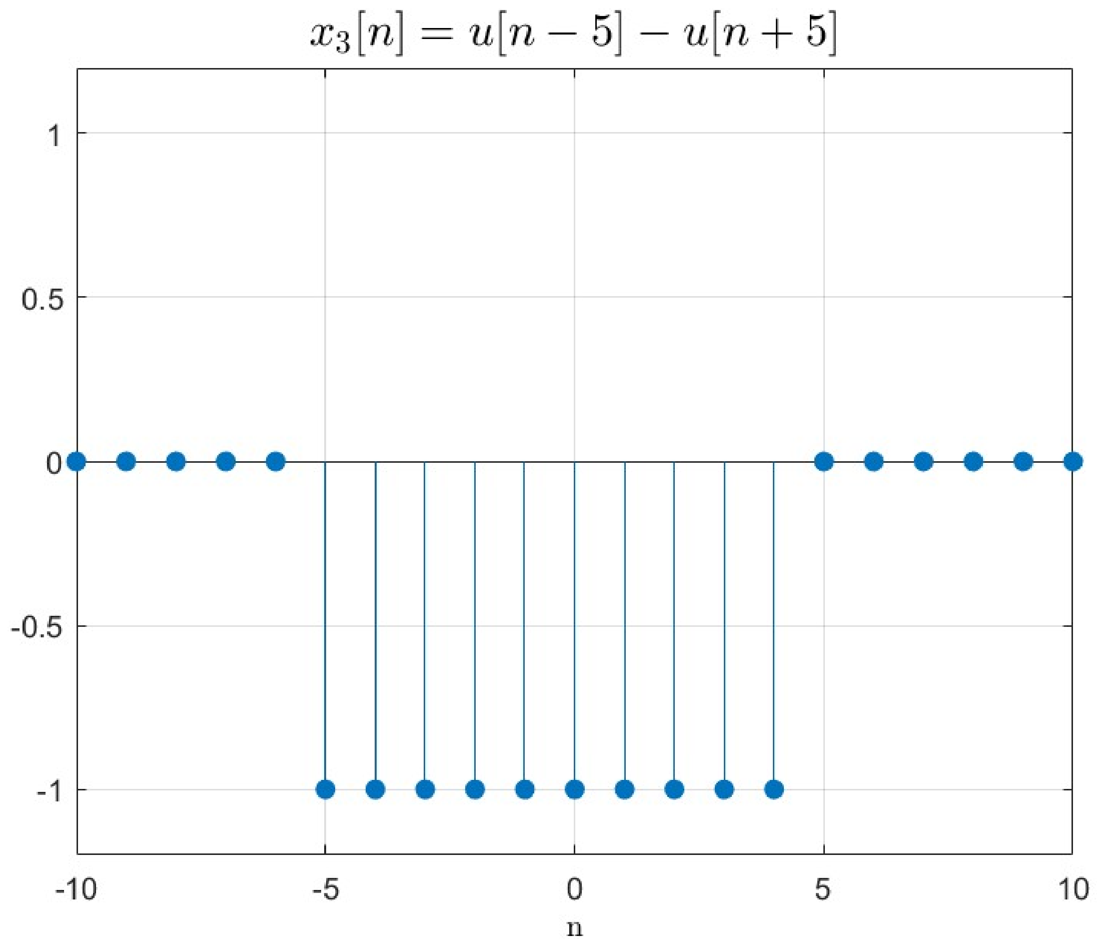

Figure 3: Third Signal Graph

$$x_4 = cos(0.5\pi n) + 3sin(\pi n) \tag{4}$$

Matlab code for plotting $x_4[n]$ is as follows:

```
x4 = cos(0.5*pi*n) + 3*sin(pi*n);
stem(n,x4)
```

Source Code 5: Fourth Signal Code

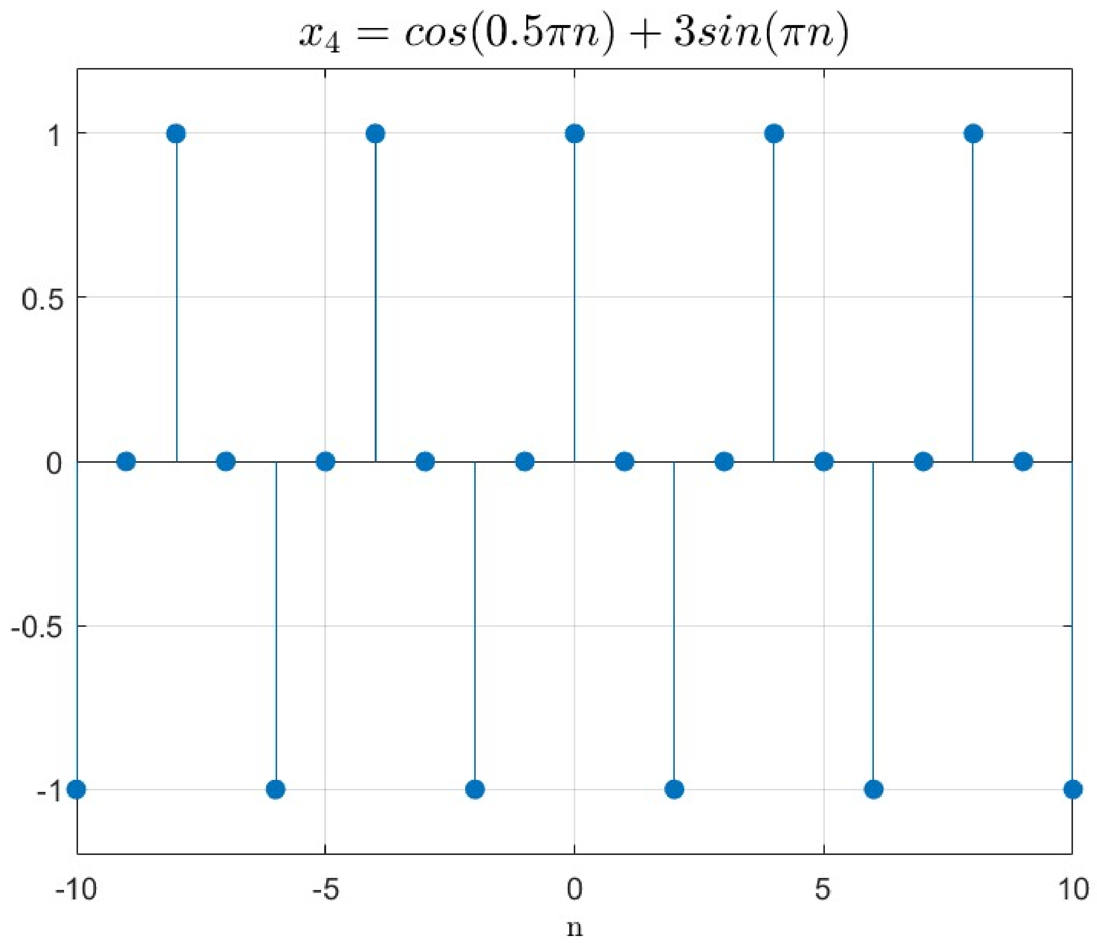

Figure 4: Fourth Signal Graph

$$x_5 = \sum_{i=-\infty}^{n} (x_1 + x_2)[i] \tag{5}$$

Matlab code for plotting $x_5[n]$ is as follows:

```
    x5 = 1.*n;
    x5(1) = x1(1)+ x2(1);
    for i = 2:21
        x5(i) = x5(i-1) + x1(i)+x2(i);
    end
    stem(n,x5)
```

Source Code 6: Fifth Signal Code

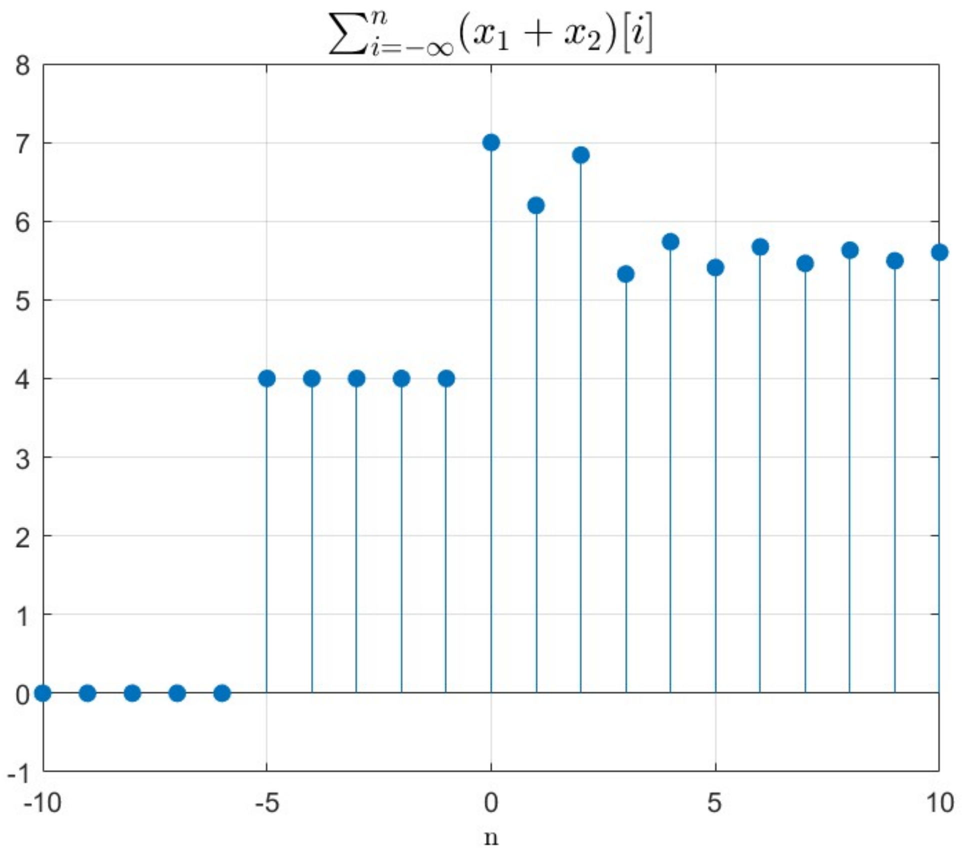

Figure 5: Fifth Signal Graph

▬ ***Is there any easier way to calculate $x_5[n]$ signal?***

Soloution

If we rewrite the $x_5[n]$ equation, it's obvious there is an easier way to calculate $x_5[n]$:

$$\begin{aligned}
x_5 &= \sum_{i=-\infty}^{n} (x_1 + x_2)[i] = \sum_{i=-\infty}^{n} (4\delta[i+5] + 2\delta[i] - \delta[i-3] + (-0.8)^i u[i]) \\
&= 4u[n+5] + 2u[n] - u[n-3] + \sum_{i=-\infty}^{n} (-0.8)^i u[i] \\
&\rightarrow x_5 = \begin{cases} 4u[n+5] + 2u[n] - u[n-3] & n < 0 \\ 4u[n+5] + 2u[n] - u[n-3] + \frac{1-(-0.8)^n}{1+0.8} & n > 0 \end{cases}
\end{aligned}$$

Continuous signals

To plot given functions in Matlab, I defined a ramp function for continuous signals and used it to plot the signals.Also I defined time interval. The code is as follows:

```
t =-5:0.01:5;
r = t.*heaviside(t);
```

Source Code 7: Time interval and Ramp function

For a bigger figure and visualization, I changed the size of figure in matlab. Following code illustrates this matter:

```
figure('Renderer', 'painters', 'Position', [500 60 700 800]);
```

Source Code 8: Figure function parametrs

$$x_1(t) = 1 - e^{-t}u(t) \tag{6}$$

Matlab code for plotting $x_1(t)$ is as follows:

```
x1 = (1-exp(-t)).*heaviside(t);
plot(t,x1t)
```

Source Code 9: First Signal Code

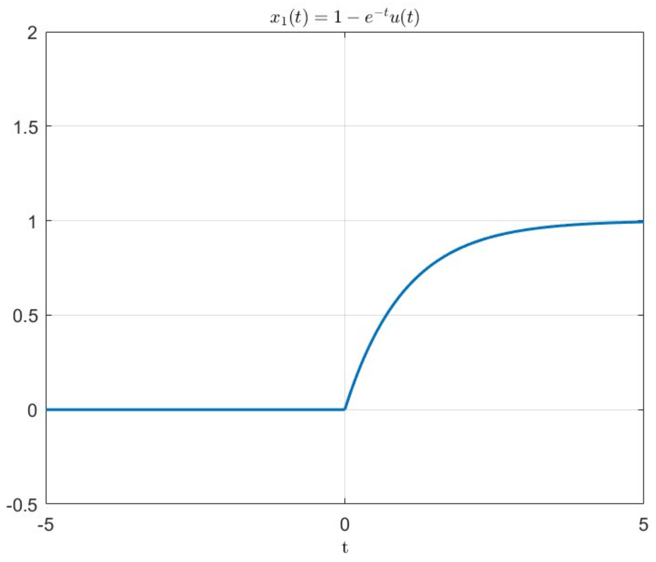

Figure 6: First Signal Graph

$$x_2(t) = 3r(t+3) + u(t) - u(t-2) \tag{7}$$

Matlab code for plotting $x_2(t)$ is as follows:

```
x2 = 3*r(t+3) +heaviside(t) - heaviside(t-2) ;
plot(t,x2)
```

Source Code 10: Second Signal Code

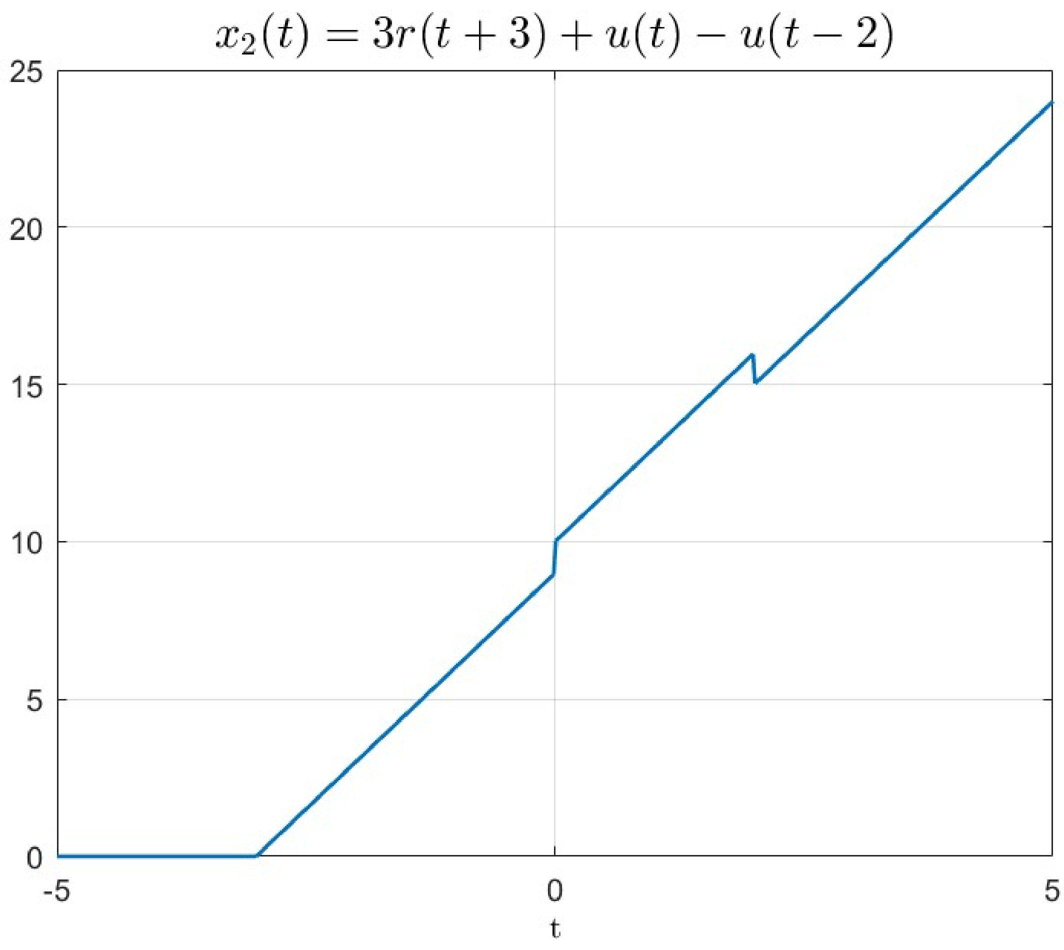

Figure 7: Second Signal Graph

$$x_3(t) = sinc(t)(u(t+3) - u(t-3)) \tag{8}$$

Matlab code for plotting $x_3(t)$ is as follows:

```
x3 = sinc(t).*(heaviside(t+3) - heaviside(t-3));
plot(t,x3)
```

Source Code 11: Third Signal Code

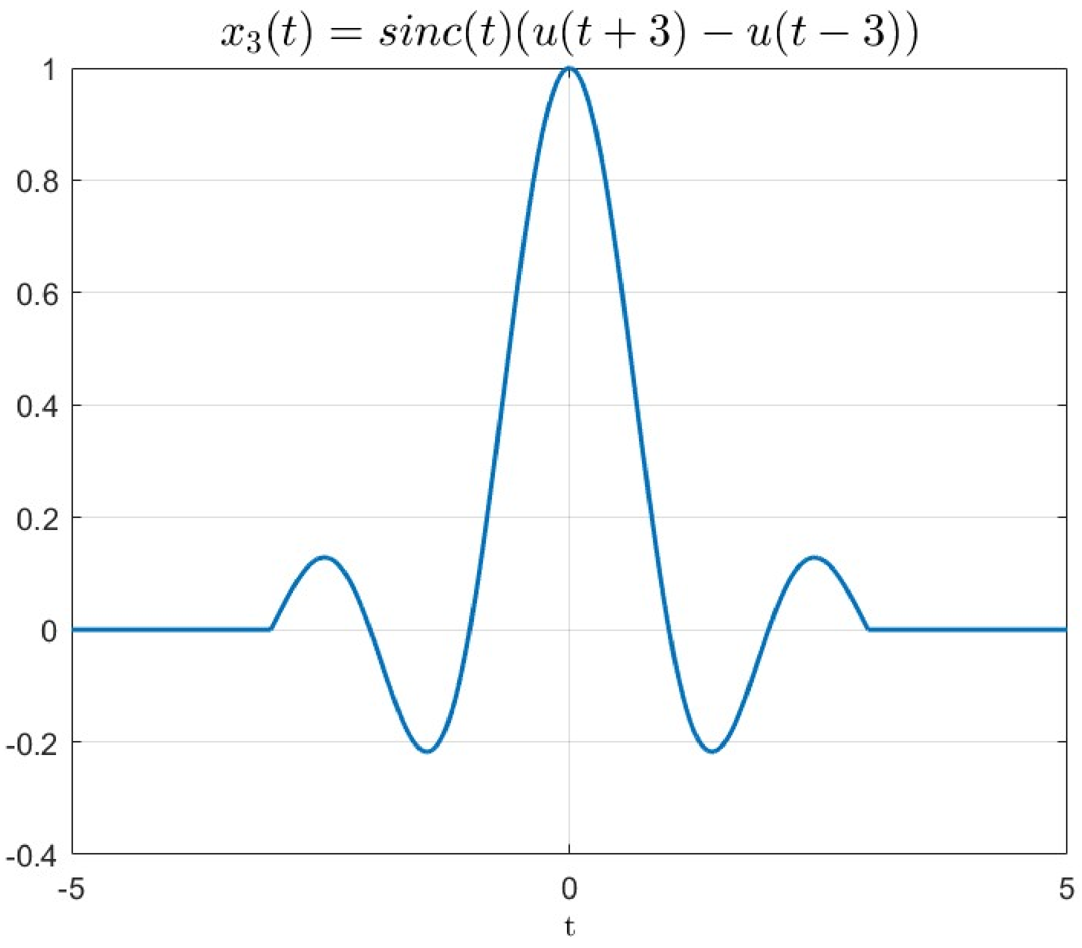

Figure 8: Third Signal Graph

$$x_4 = 0.4cos(\pi t) + 1x_5 = x_4cos(30\pi t) \tag{9}$$

Matlab code for plotting $x_4(t)$ and $x_5(t)$ is as follows:

```
syms t
x4 = 0.4*cos(pi*t)+1;
x5 = x4*cos(30*pi*t);
fplot(x4)
hold on
fplot(x5)
```

Source Code 12: Fourth and Fifth Signal Code

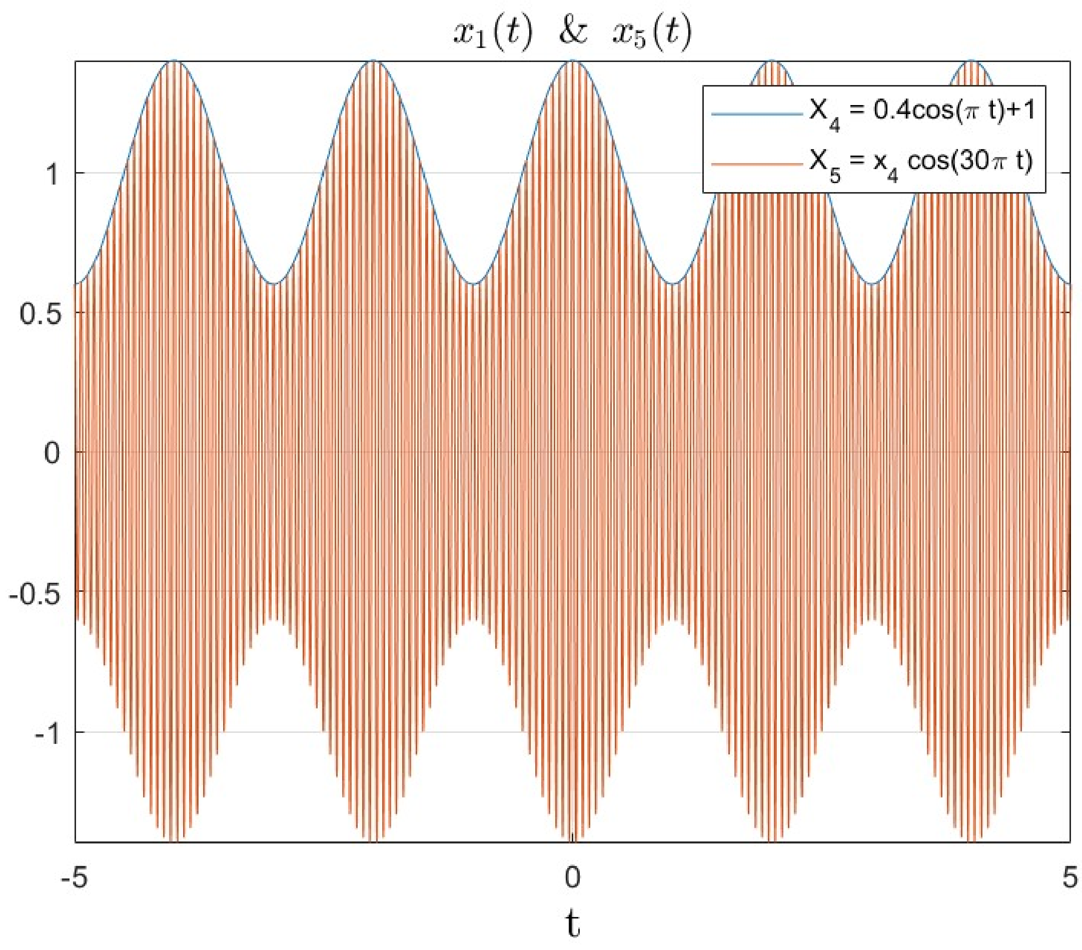

Figure 9: Fourth and Fifth Signal Graph

Convolution

Create a convolution function to convolve two signals together:

```
function y = myConv(x, h)
y = zeros(1, length(x) + length(h) - 1);
for i = 1:length(y)
    for j = 1:length(h)
        if i-j+1 > 0 && i-j+1 <= length(x)
            y(i) = y(i) + x(i-j+1) * h(j);
        end
    end
end
end
```

Source Code 13: myConv Code

Next, convolve these signals pairwise:

$$x_1[n] = u[n+7] - u[n-8] + 0.5\delta[n] - 4\delta[n-3] + 2\delta[n+4]$$
$$x_2[n] = (\frac{3}{5})^n(u[n+12] - u[n-12])$$
$$x_3[n] = n\,sin(0.5)(u[n+15] - u[n-15])$$

myConv results for x_1 & x_2:

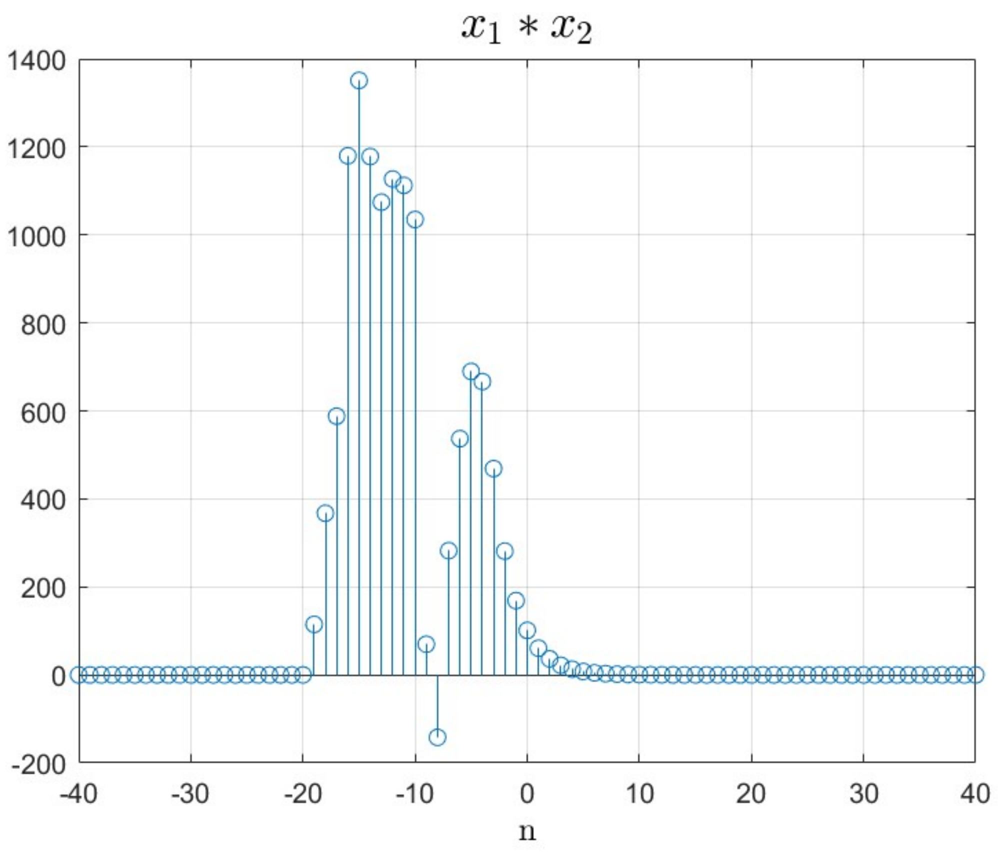

Figure 10: First and Second Signal Convolution Graph

▬ ***myConv results for*** x_2 & x_3:

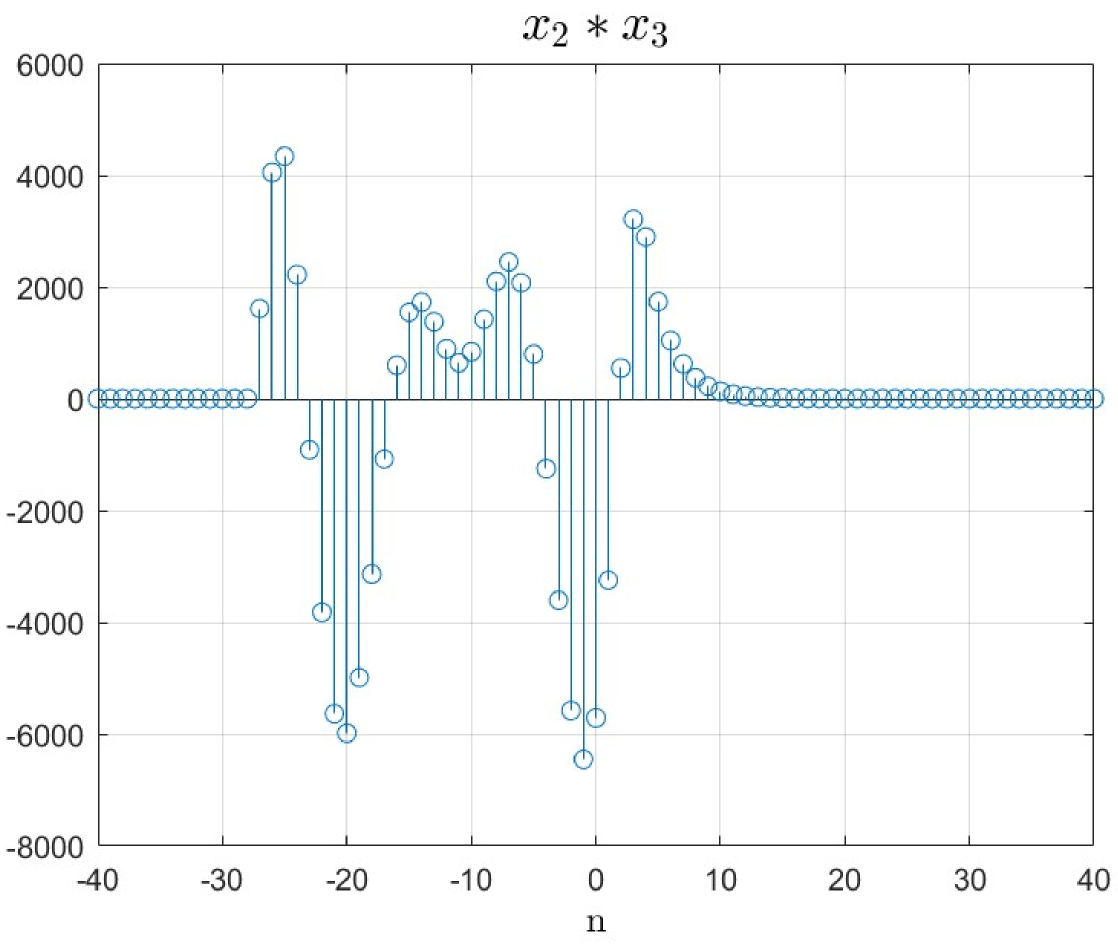

Figure 11: Second and Third Signal Convolution Graph

myConv results for x_1 & x_3***:***

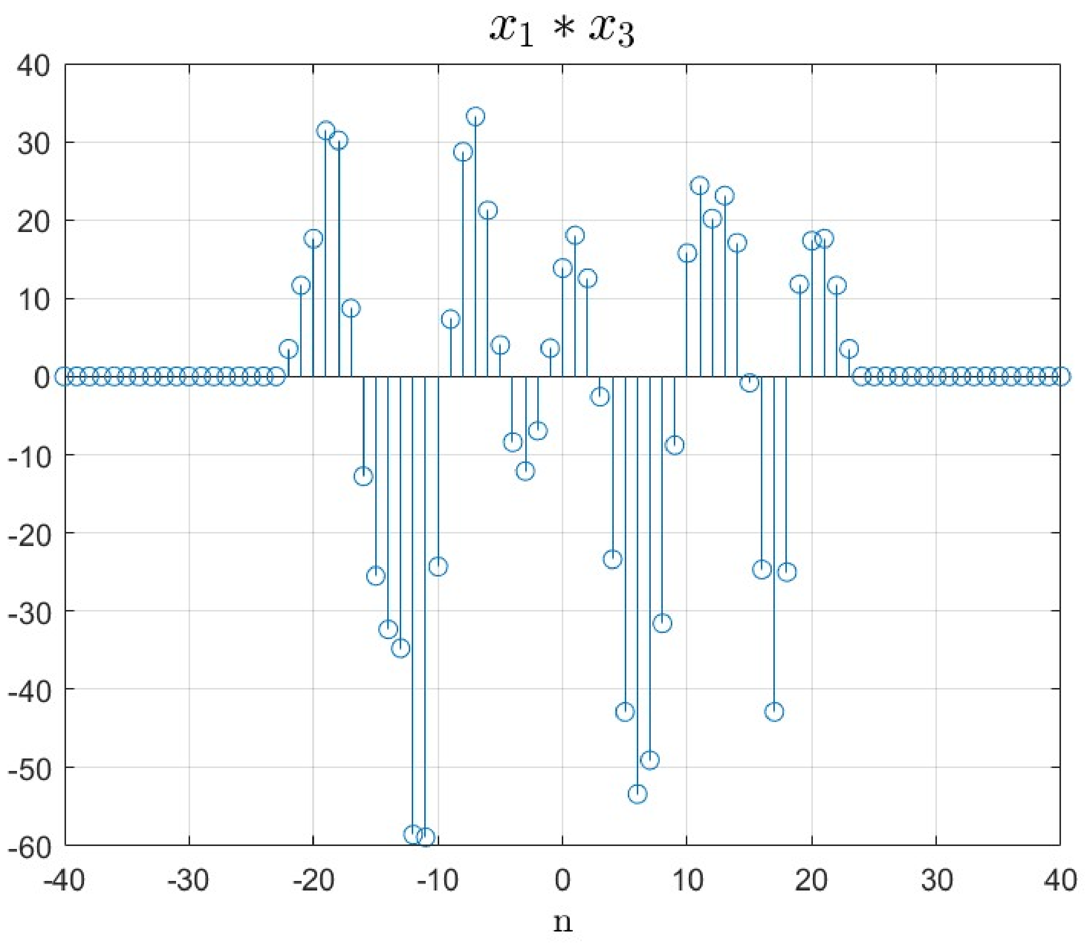

Figure 12: First and Third Signal Convolution Graph

Compare myConv and conv

Elapsed time for myConv function is as follows:

```
    Elapsed time is 0.001177 seconds.
    Elapsed time is 0.000495 seconds.
    Elapsed time is 0.000206 seconds.
```

Source Code 14: Elapsed time for myConv

Also elapsed time for Matlab's built-in conv function is as follows:

```
    Elapsed time is 0.001177 seconds.
    Elapsed time is 0.000073 seconds.
    Elapsed time is 0.000074 seconds.
```

Source Code 15: Elapsed time for conv

Also the results of myConv and conv are as follows:

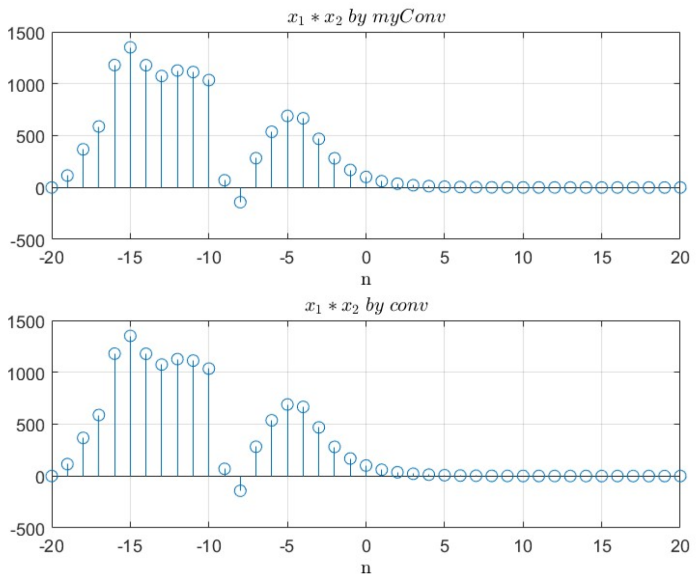

Figure 13: First and Second Signal Convolution compare Graph

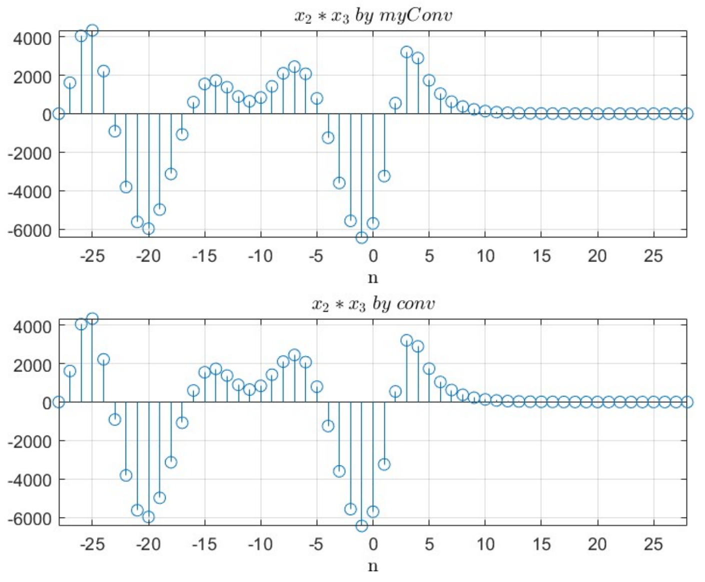

Figure 14: Second and Third Signal Convolution Graph

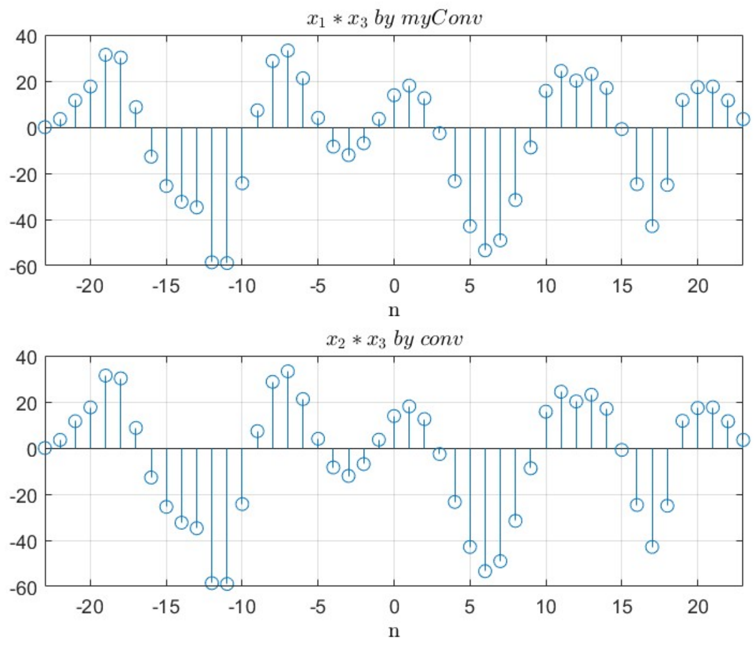

Figure 15: First and Third Signal Convolution Graph

■ *What does same do as a parameter in con2 in matlab?*

Soloution

According to Matlab Mathworks document, it computes the central part of the convolution Convsame, which is a submatrix of Convfull with the same size as first matrix.

Image Processing

decomposing image

This section is about image Processing. First of all, we load the image we want to process:

```
img = imread('pic.jpg');
imshow(img);
```

Source Code 16: Loading image

Then, I decomposed *img* to red, green and blue matrices which hold values of every pixel in 3 main colors as an uint8 variable:

```
R = img(:,:,1);
G = img(:,:,2);
B = img(:,:,3);
```

Source Code 17: decomposing image to RGB chan els

As illustrated in Mathworks document, Uint8 variable in Matlab can store values from 0 to 255; so it takes 8 bits to store an integer as uint8.
Then, Original image and 3 main channels of image are shown with *imshow* function in matlab:

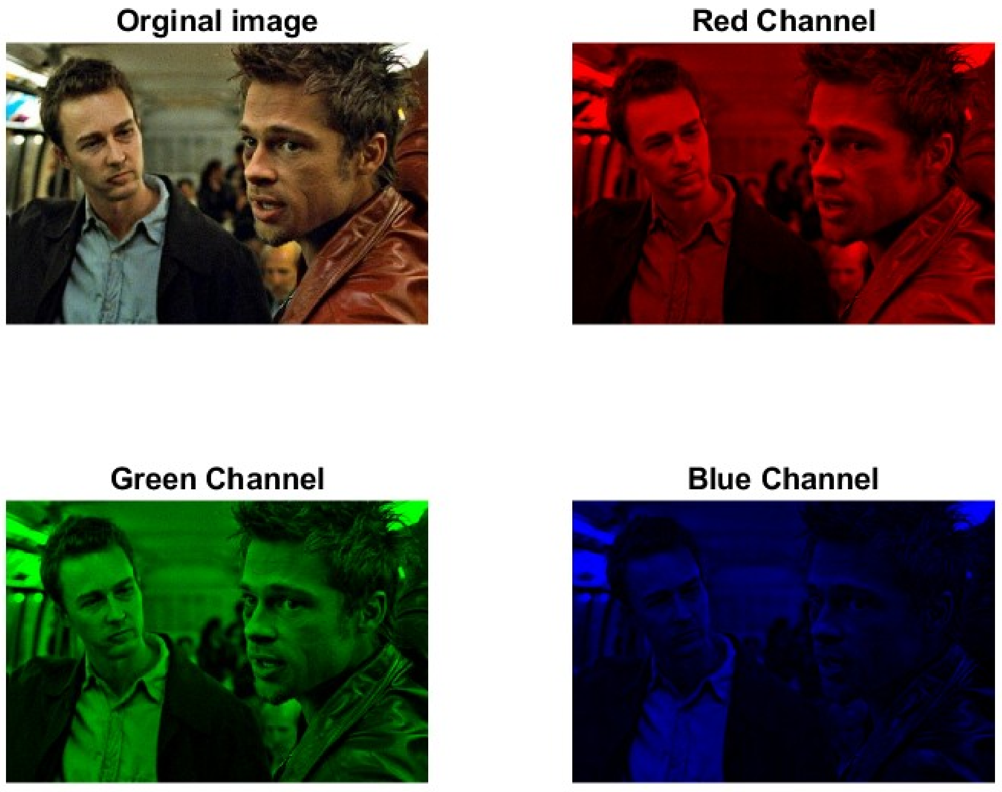

Figure 16: RGB image

Source code for decomposing image to RGB channels is as follows:

```
    subplot(2,2,2);
    Rimg = img;
    Rimg(:,:,2:3) = 0;
    imshow(Rimg)
    title('Red Channel');

    subplot(2,2,3);
    Gimg = img;
    Gimg(:,:,1) = 0;
    Gimg(:,:,3) = 0;
    imshow(Gimg)
    title('Green Channel');

    subplot(2,2,4);
    Bimg = img;
    Bimg(:,:,1:2) = 0;
    imshow(Bimg)
    title('Blue Channel');
```

Source Code 18: decomposing image to RGB channels

▬ *gray scale image*

Next step is to convert RGB image to grayscale image via mean of R, G, B matrices:

```
RGB_sum = uint8(mean(cat(3, R, G, B),3));
imshow(RGB_sum)
title('Mean of R,G,B matrices')
```

Source Code 19: converting RGB image to grayscale image

Grayscaling of image can also done by rgb2gray function in matlab:

```
I = rgb2gray(img);
imshow(I)
title('Grayscale of image done by rgb2gray')
```

Source Code 20: converting RGB image to grayscale image

Results of grayscaling are as follows:

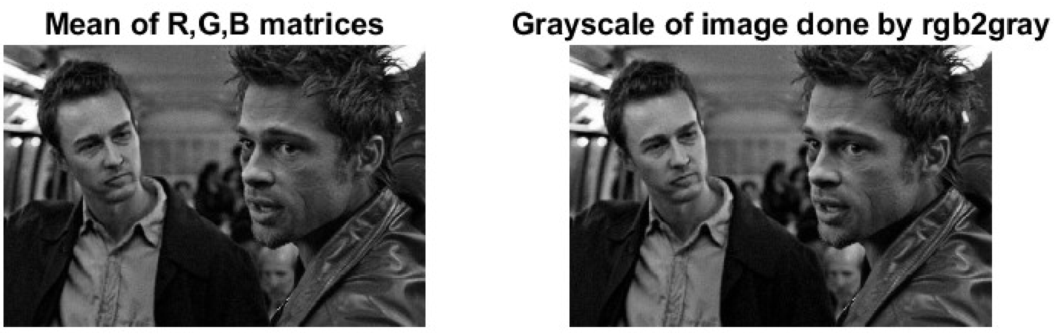

Figure 17: Grayscale image

As a matter of fact, the results of both methods are the same.This is because the mean of R, G, B matrices is the same as the result of rgb2gray function.

▬ *Convert to double*

In this section, we convert *RGB_sum* matrix to double type and compare it to *I* matrix which is the result of rgb2gray function:

```
    subplot(1,2,1)
    img_double = im2double(RGB_sum);
    imshow(img_double);
    subplot(1,2,2)
    imshow(I)
```

Source Code 21: converting RGB image to grayscale image

Results of converting *RGB_sum* matrix to double type are as follows:

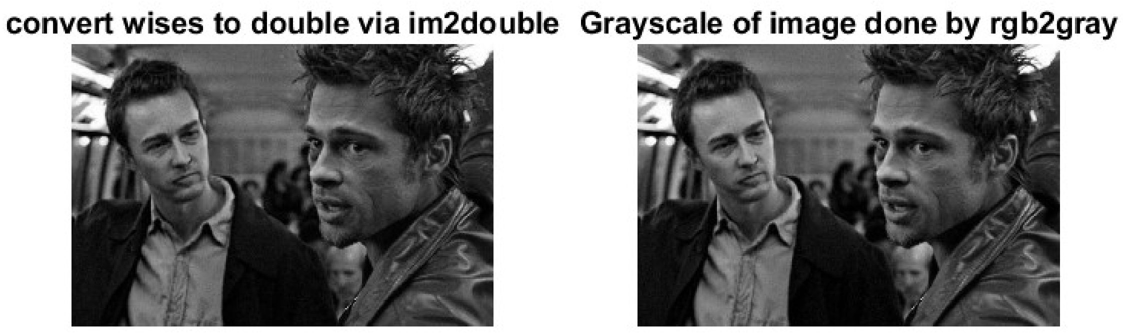
convert wises to double via im2double
Grayscale of image done by rgb2gray

Figure 18: Convert to double

Results are equal; because the *imshow* function in matlab is overloaded with double wises.

Edge detection

Grayscale edge detection

We can find vertical and horizontal edges of an image with EDK_x and EDK_y:

```
    EDK_x = [-1 0 1; -2 0 2; -1 0 1];
    EDK_y = [-1 -2 -1; 0 0 0; 1 2 1];
```

Source Code 22: Edge detection

Then, we convolve EDK_x and EDK_y with *img_double* matrix:

```
    img_x = conv2(img_double, EDK_x, 'same');
    img_y = conv2(img_double, EDK_y, 'same');
```

Source Code 23: Edge detection

To find the magnitude of edges, we use *sqrt* function:

```
    img_mag = sqrt(img_x.^2 + img_y.^2);
```

Source Code 24: Edge detection

How does EDK_x and EDK_y work?

The matrix mentioned is called a Sobel filter which is used to detect edges in an image by convolving it with the image. The Sobel filter is a 3x3 matrix that approximates the gradient of the image intensity function at each pixel. The Sobel filter can be used with the conv2 function in MATLAB to detect edges in an image.
Results of edge detection are as follows:

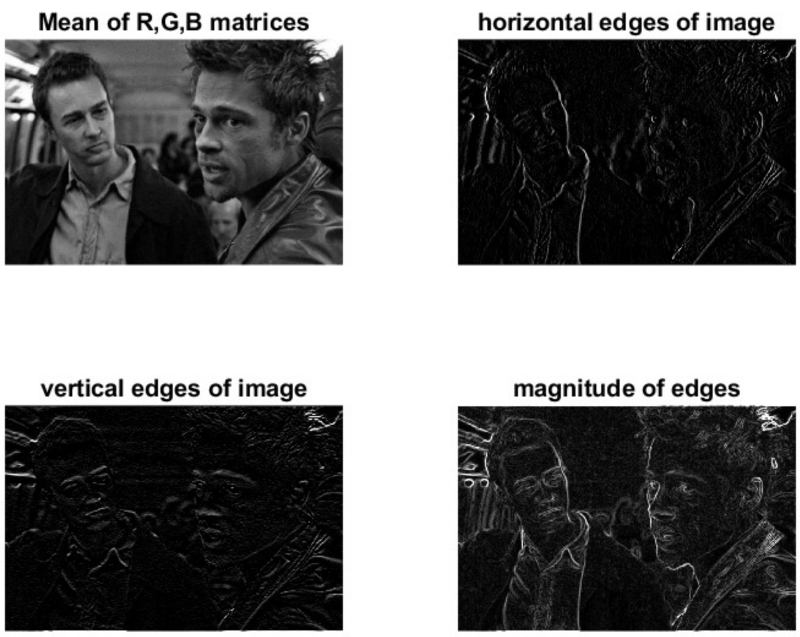

Figure 19: Edge detection

▬ *Color edge detection*

In this section, we want to detect edges in color image. First of all, we convert *img* to double type:

```
img_double = im2double(img);
```

Source Code 25: Edge detection

Then, we decompose *img_double* to R, G, B matrices:

```
Rdouble = img_double(:,:,1);
Gdouble = img_double(:,:,2);
Bdouble = img_double(:,:,3);
```

Source Code 26: Edge detection

We convolve EDK_x and EDK_y with $Rdouble, Gdouble, Bdouble$ matrices:

```
R_x = conv2(Rdouble, EDKx, 'same');
R_y = conv2(Rdouble, EDKy, 'same');
G_x = conv2(Gdouble, EDKx, 'same');
G_y = conv2(Gdouble, EDKy, 'same');
B_x = conv2(Bdouble, EDKx, 'same');
B_y = conv2(Bdouble, EDKy, 'same');
```

Source Code 27: Edge detection

To find the magnitude of edges, we use *sqrt* function:

```
R_mag = sqrt(R_x.^2 + R_y.^2);
G_mag = sqrt(G_x.^2 + G_y.^2);
B_mag = sqrt(B_x.^2 + B_y.^2);
```

Source Code 28: Edge detection

Finally, we combine R, G, B matrices to *img_double* matrix:

```
img_double(:,:,1) = R_mag;
img_double(:,:,2) = G_mag;
img_double(:,:,3) = B_mag;
```

Source Code 29: Edge detection

Results of edge detection are as follows:

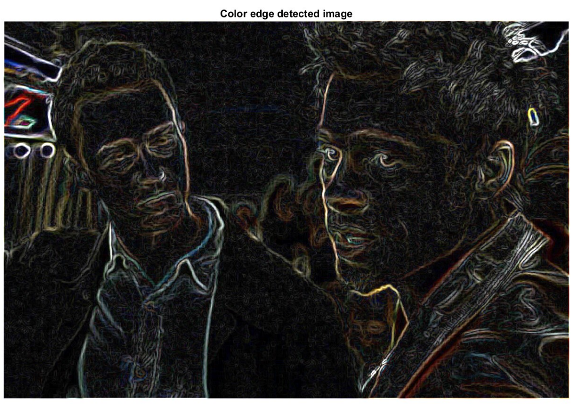

Figure 20: Color edge detection

Some other Kernels

Image blurring

In this section, we want to blur an image. To do this, we use BBK:

```
BBK = 1/9 * ones(3);
```

Source Code 30: Image blurring

Then, we convolve BBK with *img_double* matrix:

```
Bimg1 = conv2(img_double, BBK, 'same');
```

Source Code 31: Image blurring

Results of image blurring are as follows:

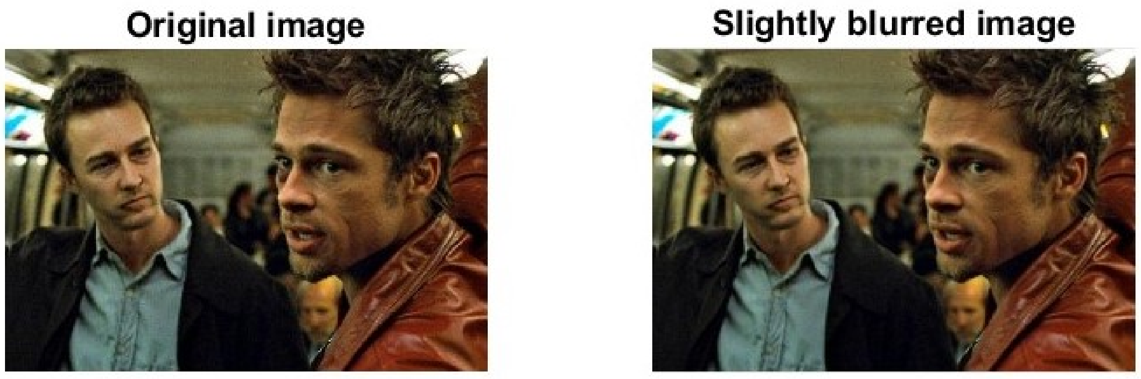

Figure 21: Image blurring

Why BBK blurs the image?

This matrix is called a box filter which is used to blur an image by taking the average of all neighboring pixels. The values in the matrix add up to 1 which ensures that the output image has the same brightness as the input image1. The box filter can be used with the conv2 function in MATLAB to blur an image.

It's obvious that the image is not blurred much. To solve this problem, I used BBK with different sizes:

```
kernel_size = 15;
kernel = ones(kernel_size) / kernel_size^2;
Bimg2 = conv2(img_double, kernel, 'same');
```

Source Code 32: Image blurring

Then I got the following results:
As you can see, the image is blurred more than the previous one.

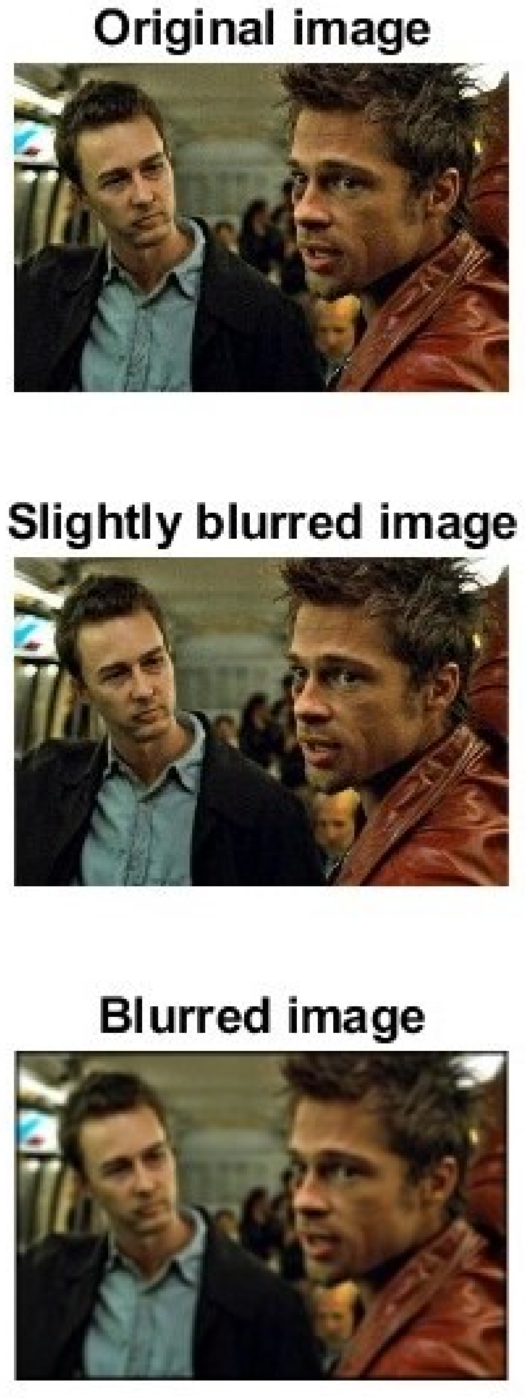

Figure 22: Image blurring

▬ *Image sharpening*

In this section, we want to sharpen an image. To do this, we use ISK:

```
ISK = [0 -1 0; -1 5 -1; 0 -1 0];
```

Source Code 33: Image sharpening

Then, we convolve ISK with *img_double* matrix:

```
SharpenImg = imconv(img,ISK);
```

Source Code 34: Image sharpening

Results of image sharpening are as follows:

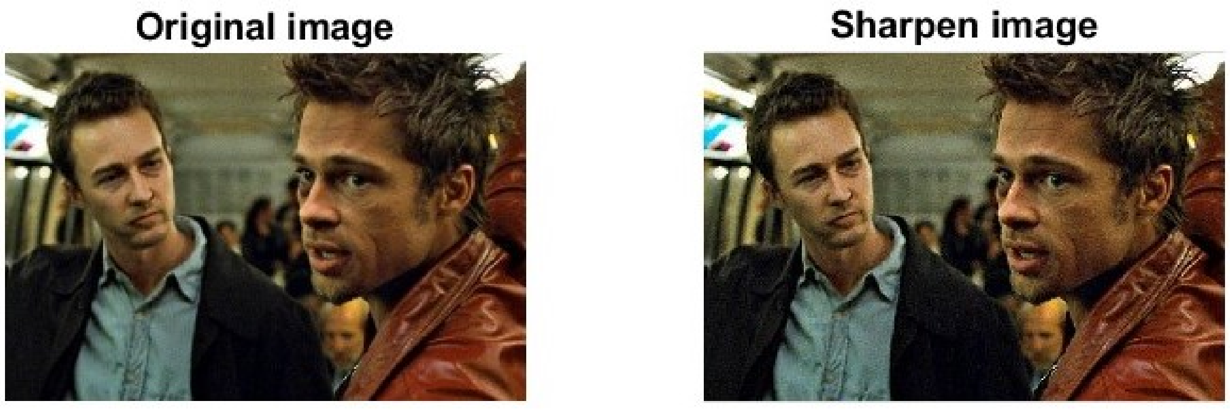

Figure 23: Image sharpening

Voice Processing

Mean of two channels

First of all, we should import the audio file:

```
y = audioread('Music.wav');
```

Source Code 35: Voice Processing

TO plot the audio file, we use the following code:

```
figure
stem(y)
```

Source Code 36: Voice Processing

The result is as follows: The plot shows that music is combine of two channels. As stated in

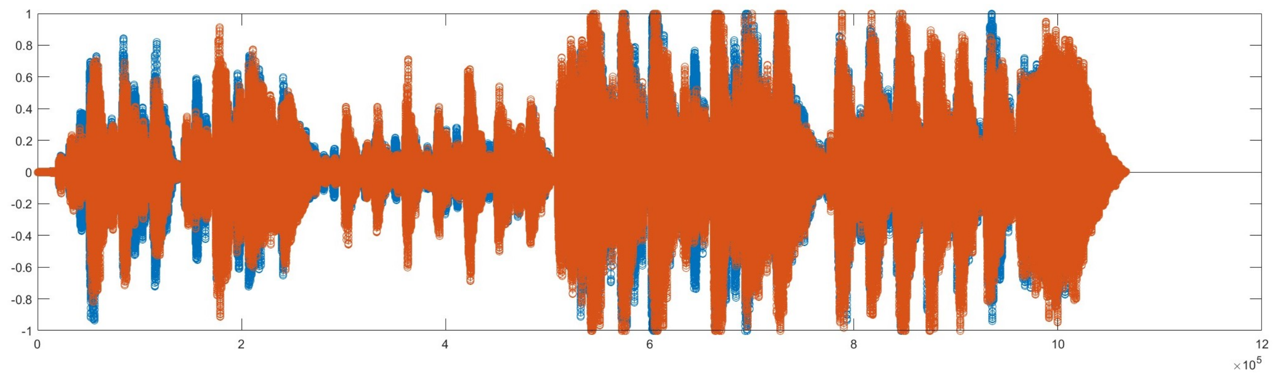

Figure 24: Voice Processing

the homework, we should take mean of the two channels. To do this, we use the following code:

```
y = mean(y,2);
```

Source Code 37: Voice Processing

Next, we should play the audio file. To do this,I defined a frequency variable called fs. Then, I used the following code:

```
fs = 48000;
soundsc(y,fs)
pause(22)
soundsc(mean_y,fs)
```

Source Code 38: Voice Processing

difference between the two voices from my point of view was marginal and ignorable.

Echo effect

In this section, we want to add echo effect to the audio file. To do this, we should follow these steps:

1. Define a delay variable called n_d and gain variable called α.
2. Define a new vector called y.
3. Define a for loop to add echo effect or use filter function.

Given formula is as follows:

$$y[n] = x[n] + \alpha x[n - n_d] \qquad \alpha = 0.8 \;\; \& \;\; n_d = 0.2fs \tag{10}$$

To define a delay variable called n_d and gain variable called α, we use the following code:

```
nd = 0.2*fs;
alpha = 0.8;
```

Source Code 39: Echo effect

To echo given audio file via filter, I used the following code:

```
echoMusix = filter([1 ,zeros(1,nd) , alpha] , 1 , mean_y);
soundsc(echoMusix,fs)
```

Source Code 40: Echo effect

To plot the audio file, we use the following code:

```
figure
stem(echoMusix)
```

Source Code 41: Echo effect

The result is as follows:

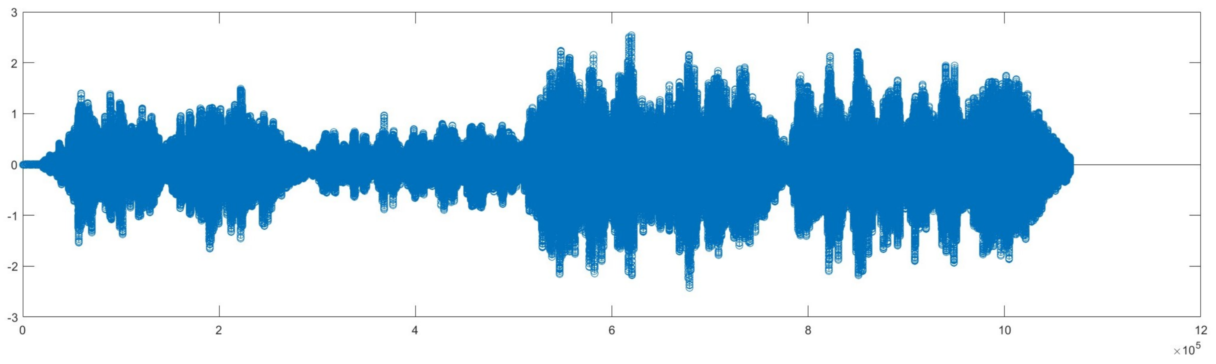

Figure 25: Echo effect

Recover the original audio file

In this section, we want to recover the original audio file with Z-transform. To do this, we should know that impulse response of inverse Z-transform is as follows:

$$h[n] = \frac{1}{z^N} = \sum_{k=0}^{N-1} \frac{1}{z^k} \tag{11}$$

So for $y[n] = x[n] + \alpha x[n - n_d]$ we have:

$$Y[z] = X[z] + \alpha X[z] z^{-n_d} \tag{12}$$

Also, inverse Z-transform of $Y[z]$ is as follows:

$$H^{-1}[z] = \frac{1}{H[z]} \tag{13}$$

So, we have:

$$H^{-1}[z] = \frac{1}{1 + \alpha z^{-n_d}} \tag{14}$$

To recover the original audio file, we use the following code:

```
    RecoveredMusic = filter(1,[1 ,zeros(1,nd) , alpha],echoMusix);
```

Source Code 42: Recover the original audio file

Add several echo effects and recover the original audio file

In this section, we want to add several echo effects and recover the original audio file. To do this, I defined a filter :

```
f = zeros(1,3*nd+1);
f(1)=1;
f(nd)=alpha;
f(2*nd)=alpha^2;
f(3*nd)=alpha^3;
```

Source Code 43: Add several echo effects and recover the original audio file

Then for echoed audio file, I used the following code:

```
echoMusix = filter(f,1,mean_y);
soundsc(echoMusix,fs)
```

Source Code 44: Add several echo effects and recover the original audio file

To recover the original audio file, I used the following code:

```
RecoveredMusic = filter(1,f,echoMusix);
```

Source Code 45: Add several echo effects and recover the original audio file

Results are as follows:

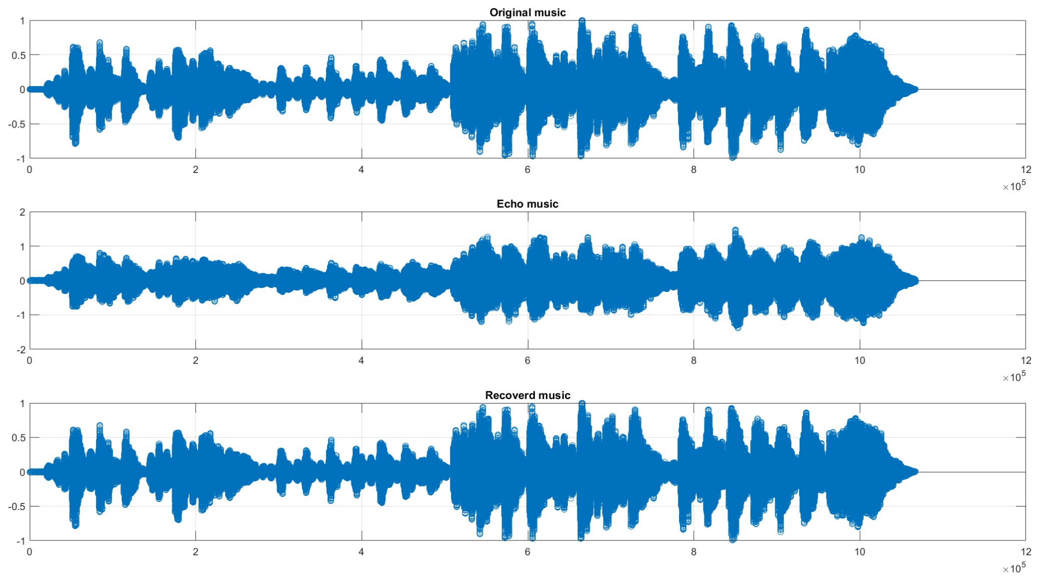

Figure 26: Add several echo effects and recover the original audio file

Add noise

In this section, we want to add noise to the audio file. To do this, we should follow these steps:

1. Define a random variable called *noise*.
2. Define a new vector called z.
3. Define a for loop to add echo effect or use filter function.

First, we add a normal distribution noise to the audio file. To do this, we use the following code:

```
Mean=0;
Var=0.01;
z1 = imnoise(mean_y,'Gaussian',Mean,Var);
```

Source Code 46: Build and add noise

Then, we play the audio file. To do this, we use the following code:

```
soundsc(z1,fs)
```

Source Code 47: Play noisy audio file

Second, we add a uniform distribution noise to the audio file. To do this, we use the following code:

```
u = -0.1 + rand(length(mean_y),1)*(0.2);
z2 = u+mean_y;
```

Source Code 48: Build and add noise

Then, we play the audio file. To do this, we use the following code:

```
soundsc(z2,fs)
```

Source Code 49: Play noisy audio file

To plot the audio file, we use the following code:

```
figure
stem(z1)
figure
stem(z2)
```

Source Code 50: Build and add noise

The result is as follows:

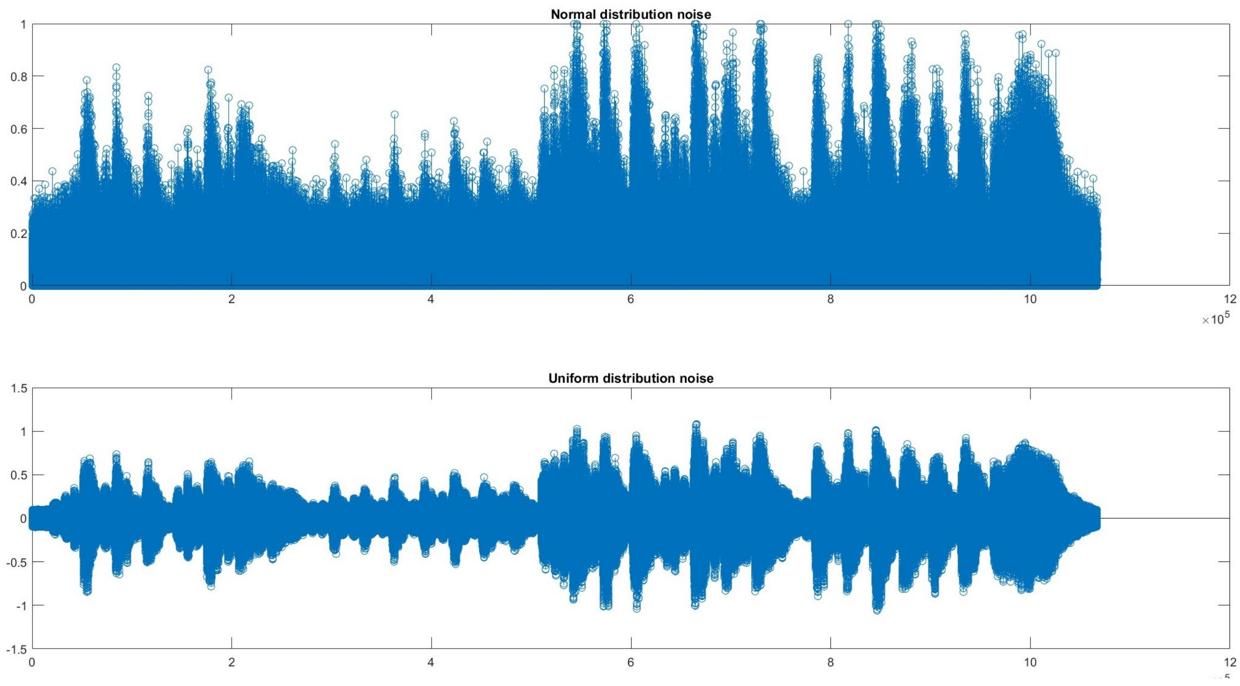

Figure 27: Build and add noise

Add variant sin wave

In this section, we want to add a variant sin wave to the audio file. To do this, we should follow these steps to build a variant frequency sin wave:

```
t = 0:1/fs:(length(mean_y)-1)/fs;
f = linspace(1000,2000,length(t));
y = 0.1*sin(2*pi.*f.*t);
```

Source Code 51: Add variant sin wave

This vector is a 1×1067147 vector. Then, we should add this vector to the audio file. To do this, I transposed the vector y and added it to the audio file. To do this, I used the following code:

```
y = transpose(y);
MusicSine = mean_y + y;
```

Source Code 52: Build variant frequency sin wave

To play the audio file, I used the following code:

```
soundsc(MusicSine,fs)
```

Source Code 53: Add variant sin wave

At last, to write the audio file, I used the following code:

```
    audiowrite('MusicSine.wav',MusicSine,fs)
```

Source Code 54: Save audio file

End of Computer homework 1