Signal and System Computer homework

Dr.Hamid Behroozi

department : Electrical Engineering

Amirreza Velae 400102222

Computer homework 2

Report

May 22, 2023

Signal and System Computer homework

Computer homework 2

Amirreza Velae 400102222

Fourier series

Fourier series coefficients calculator and recover the signal

To calculate the Fourier series coefficients, I wrote the following function:

```
function ak = FSC_calculator(xc, Fs, T, k_min, k_max)
    N = floor(Fs * T);
    n = linspace(0,N-1,N)/Fs;
    xd = arrayfun(xc,n);
    ak0 = fftshift(fft(xd,N)/N);
    ak = ak0(N/2+k_min+1:N/2+k_max+1);
end
```

Source Code 1: Fourier series coefficients calculater

This functions works if and only if demanded **k's** are less than $T \times Fs$.

Find Fourier section coefficients fo given functions

Demanded functions are as follows

- Triangle wave with period $T = 2$ and amplitude $A = 1$ and duty cycle $D = 50\%$.
- Sawtooth wave with period $T = 1$ and amplitude $A = 2$.
- $x_{c3}(t) = e^{2t} + 2t^3$ with period $T = 3$.

First I computed a_k's for each function and then I plotted them in a bar chart.Then, I recoverd the signal using the following function:

```
function y = FS_calculator(ak, k_min, k_max, T, t)
    y=0;
    for i=1:k_max-k_min+1
        y = y + ak(i)*exp(((2*pi/T)*(i+k_min-1)*1i).*t);
    end
end
```

Source Code 2: Fourier series calculater results are as follows:

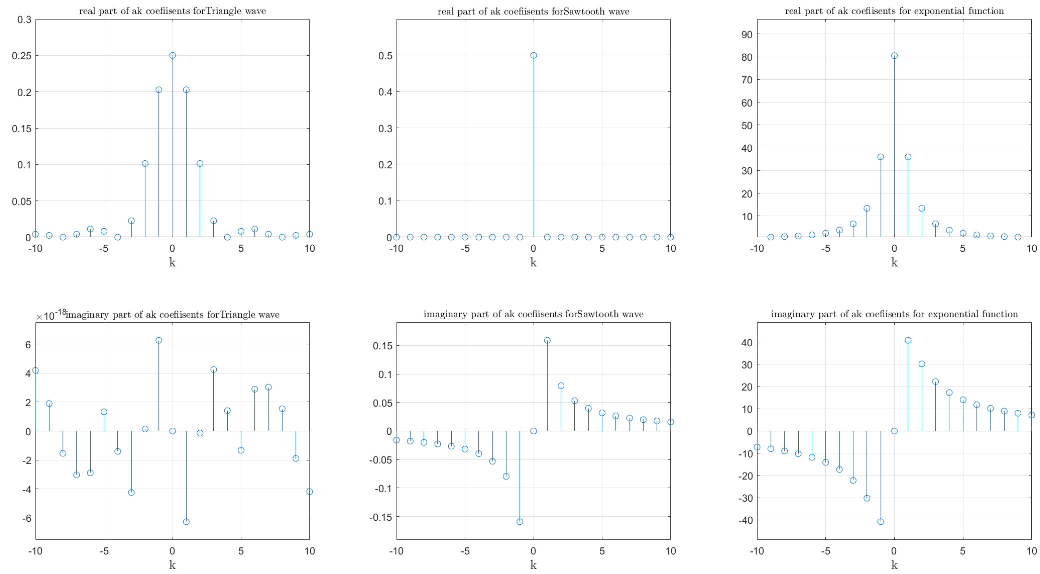

Figure 1: Fourier series coefficients for waves

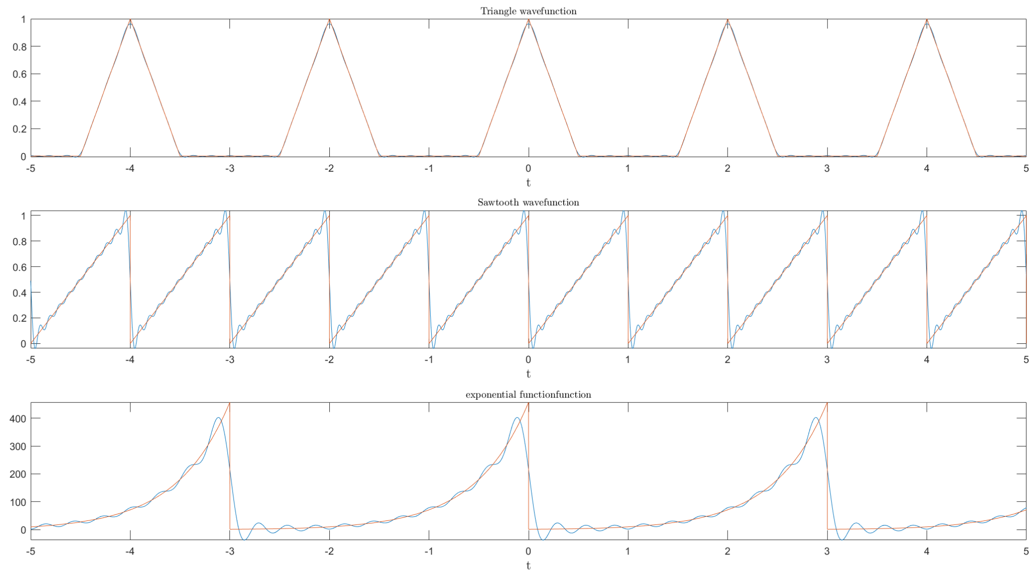

Figure 2: Recovery of waves

As you can see in figure 2, the recovery of the signal is not perfect. This is because of the fact that we have used a finite number of a_k's. If we use more a_k's, the recovery will be more accurate.
It's also obvious that Gibbs phenomenon is present in the recovery of functions.Gibs phenomenon is a phenomenon in which there are oscillations in the recovery of a function near discontinuities. This phenomenon has nothing to do with the number of a_k's used in the recovery of the function.

Create gif of recovery of rectangular wave

Given function is shifted and scaled rectangular wave with period $T = 8$ and amplitude $A = 1$:

$$x_c(t) = \Pi(\frac{t+1}{4})$$

It's demanded to create a gif of recovery of this function with different number of a_k's starting from $k = 1$ to $k = 30$.
To create the gif, I used the following function:

```
function save_Square_Wave_gif(func , K , t)
    filename = 'FS_Simulation.gif';
    for k = 1:K
        clf
        fplot(func)
        ylim([-0.3 , 1.3])
        grid on
        hold on
        title('Square Wave function for k =' + string(k),'Interpreter','latex','FontSize',10)
        xlabel('t','Interpreter','latex','FontSize',13)
        ak = FSC_calculator(func, 1000, 8, -k, k);
        plot(t,FS_calculator(ak , -k , k ,8 ,t))
        pause(0.4)
        frame = getframe(gcf);
        im = frame2im(frame);
        [imind,cm] = rgb2ind(im,256);
        if k == 1
            imwrite(imind,cm,filename,'gif','Loopcount',inf);
        else
            imwrite(imind,cm,filename,'gif','WriteMode','append');
        end
    end
end
```

Source Code 3: Effect of number of a_k's on recovery of rectangular wave

result is given in report outputs folder named FS_Simulation.gif.
Something interesting about this gif is that the recovery of the function changes when k is odd. This is because of the fact that the function is odd and when k is odd, the recovery is also odd and when k is even, the recovery is even. So, when k is odd, the recovery is more accurate.

Dual-tone multi frequency

Create DTMF signal of your student number

It's demanded that put 1000 samples of frequency of every digit within 100 null signals.Given frequencies are as follows:

Freq.	1209 Hz	1336 Hz	1477 Hz
697 Hz	1	2	3
770 Hz	4	5	6
852 Hz	7	8	9
941 Hz	*	0	#

Figure 3: frequency of every digit

My student ID is 400102222. I wrote this function to turn this number to dual-tone multi frequency:

```
function call = number_to_sound(x,Fs,space)
    t1 = 0:1/Fs:999/Fs;
    spaces = zeros(1, space);
    call = zeros(1,0);
    for i=1:length(x)
        y = digit_to_sound(x(i));
        y = arrayfun(y,t1);
        call = cat(2 , call , y);
        if(i<length(x))
            call = cat(2 , call , spaces);
        end
    end
end
```

Source Code 4: Create DTMF of a number

The function I used to create every digits special wave is as follows:

```
function y = digit_to_sound(x)
    y = @(t) 0;
    a = @(t) 0;
    b = @(t) 0;
    if(mod(x,3) ==1)
        a = @(t) cos(2*pi*1209*t);
    end

    if(mod(x,3) == 2)
        a = @(t) cos(2*pi*1336*t);
    end
    if(mod(x,3) == 0)
        a = @(t) cos(2*pi*1477*t);
    end

```

```
    if(x<=3)
        b = @(t) cos(2*pi*697*t);
    end

    if(x<=6 && x>3)
        b = @(t) cos(2*pi*770*t);
    end

    if(x<=9 && x>6 )
        b = @(t) cos(2*pi*852*t);
    end
    y = @(t) a(t) + b(t);
    if(x==0)
        y = @(t) cos(2*pi*1336*t) + cos(2*pi*941*t);
    end
end
```

Source Code 5: Create DTMF of every digit

and the code to do so is as follows:

```
num = 400102222;
digits = int32(cell2mat(cellfun(@str2num, split(num2str(num),""),UniformOutput
    =false)));
call = number_to_sound(digits,8192,100);
filename = 'StudentID.wav';
audiowrite(filename, call, 8192);
```

Source Code 6: Create DTMF of Student ID

Result is saved in report outputs folder named StudentID.wav

calculate DFFT of every digit

I wrote this function to calculate discrete fourier transform of every number between 0 & 9:

```
function plot_every_digit_fourier_trnasform(fs,N)
    x=fs*(-N/2:N/2-1)/N;
    figure
    for i=0:9
        subplot(4,3,i+1);
        if(i==9)
            subplot(4,3,[10,11,12])
        end
        plot_fourier_digits(x,i,N,fs)
        title('dfft for'+string(i),'Interpreter','latex','FontSize',10)
        xlabel('f','Interpreter','latex','FontSize',13)
    end
end
```

Source Code 7: Function for calculating DFFT of numbers between 0 and 9

Also, to calculate DFFT of a digit, I used this function:

```
function plot_fourier_digits(x,number,N,fs)
    func = digit_to_sound(number);
    n=0:1/fs:(N-1)/fs;
    xd = arrayfun(func,n);
    dfft = fourier_transform(xd , N);
    plot(x,dfft)
end
```

Source Code 8: Function for calculating DFFT of a digit

results are as follows:

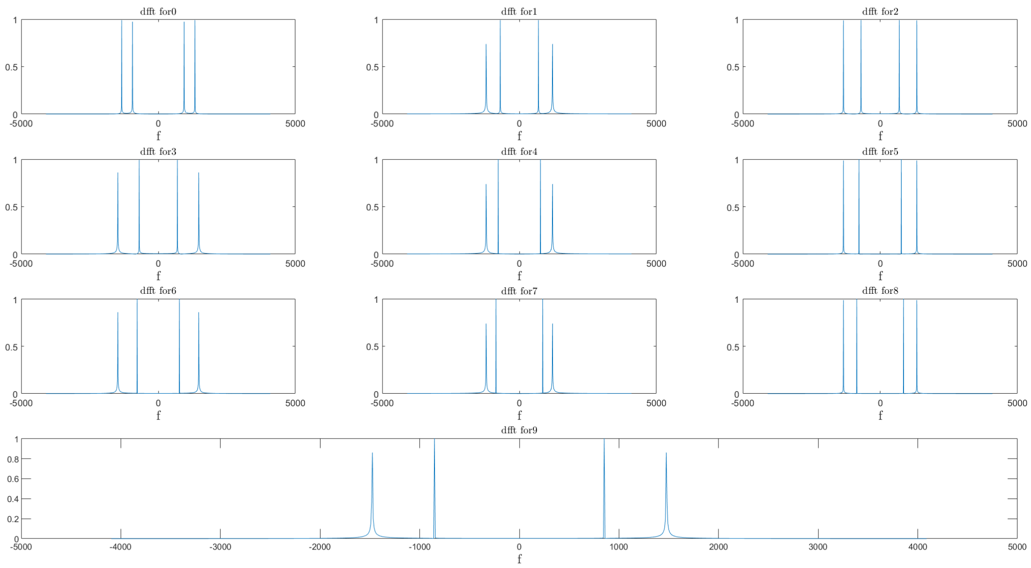

Figure 4: DFFT of every digit

It's also common sense; cause every digit is combine of two different frequency of cosine wave, It should have two delta functions at $-fs$ and fs.

Derive numbers from their sound

First, I removed every zero sample from data by **zero_remover** function :

```
function y = zero_remover(x)
    transitions = diff([0 ;x == 0; 0]); %find where the array goes from non-zero to zero and vice versa
    runstarts = find(transitions == 1);
    runends = find(transitions == -1); %one past the end
    runlengths = runends - runstarts;
    % keep only those runs of length 100 or less:
    runstarts(runlengths < 100) = [];
    runends(runlengths < 100) = [];
    %expand each run into a list indices:
    indices = arrayfun(@(s, e) s:e-1, runstarts, runends, 'UniformOutput', false);
    indices = [indices{:}];  %concatenate the list of indices into one vector
    % Remove those zeros which are consecuitve 3 in number
    x(indices) = [] ;
    y = x;
    %credit : https://www.mathworks.com/matlabcentral/answers/399882-removing-consecutive-zeros-of-a-certain-length
end
```

Source Code 9: Function for removing zero samples

Then, I used a function to facture every number to digits and then another function to calculate DFFT of the separated parts of dialings and then another function to match them to digits frequencies:

```
function numbers = number_dector(str)
    [y,fs]=audioread("Audio/"+str+".wav");
    N = 1000;
    y = zero_remover(y);
    y = y';
    n = size(y);
    n = n(2)/1000;
    numbers = [];
    for i = 1 :n
        my_fft=fftshift(fft(y((i-1)*1000+1:1000*i)));
        fft_oneside=my_fft(1:N);
        dfft=abs(fft_oneside)/(N/2);
        [maxVals, maxIdxs] = maxk(dfft, 4);
        number = match_number_frequency(maxIdxs,N);
        numbers = [numbers number];
    end
    numbers = int8(numbers);
end
```

Source Code 10: Factorize number and get DFFT's max indices

This function removes the frequencies below 0 **HZ** and then shifts back the positives from the records frequency:

```
function number = match_number_frequency(maxIdxs,N)
    maxIdxs = maxIdxs - N/2;
    cindition = find(maxIdxs < 0);
    c = setdiff(1:length(maxIdxs), cindition);
    positiveMaxIdxs = maxIdxs(c);
    frequencies = positiveMaxIdxs*(8192/N);
    number = match_frequency_to_number(frequencies);
end
```

Source Code 11: Remove frequencies below 0 HZ

At last, this function gets two max frequencies and match them with digits:

```
function number = match_frequency_to_number(frequencies)
    row_frequencies = [697 770 852 941];
    column_frequencies = [1209 1336 1477];
    for i=1:2
        row_frequencies1 = row_frequencies - frequencies(1);
        row_frequencies2 = row_frequencies - frequencies(2);
    end
    [~, idx] = min(abs([row_frequencies1 row_frequencies2]));
    match_row_frequency = mod(idx,4);
    for i=1:2
        column_frequencies1 = column_frequencies - frequencies(1);
        column_frequencies2 = column_frequencies - frequencies(2);
    end
    [~, idx2] = min(abs([column_frequencies1 column_frequencies2]));
    match_column_frequency = mod(idx2,3);
    if(match_column_frequency==0)
        match_column_frequency=3;
    end
    number = (match_row_frequency-1)*3 + match_column_frequency;


```

```
    if(match_column_frequency == 2 && match_row_frequency ==0)
        number = 0;
    end
end
```

Source Code 12: get two max frequencies

Results of deriving numbers from records of 8192 **HZ**:

Variables - x1

x1

1x11 int8

	1	2	3	4	5	6	7	8	9	10	11
1	0	9	1	2	1	2	3	4	5	6	7

Figure 5: digits of Dialing1

Variables - x2

x2

1x11 int8

	1	2	3	4	5	6	7	8	9	10	11
1	0	9	1	0	0	4	1	3	1	3	1

Figure 6: digits of Dialing2

Variables - x3

x3

1x11 int8

	1	2	3	4	5	6	7	8	9	10	11
1	0	9	1	6	7	6	8	8	6	7	1

Figure 7: digits of Dialing3

Show numbers on UI

I used this function to do so:

```
function show_number(images,x1,name)
    h = figure;
    set(h,'Position',[450 360 800 420]);
    for i=1:length(x1)
        h = subplot(1,length(x1),i);
        imshow(images{x1(i)+1})
        pos = get(h,'OuterPosition');
        pos(3) = 0.103; % set width to 100% of figure
        set(h,'OuterPosition',pos);
    end
    pos = get(h,'OuterPosition');
    pos(3) = 0.113; % set width to 100% of figure
    set(h,'OuterPosition',pos);
    sgtitle(name)

end
```

Source Code 13: show numbers on UI

Results are as follows: Also, I used this function to load images:

```
function images = load_numbers_image()
    number0 = imread("Numbers\0.png");
    number1 = imread("Numbers\1.png");
    number2 = imread("Numbers\2.png");
    number3 = imread("Numbers\3.png");
    number4 = imread("Numbers\4.png");
    number5 = imread("Numbers\5.png");
    number6 = imread("Numbers\6.png");
    number7 = imread("Numbers\7.png");
    number8 = imread("Numbers\8.png");
    number9 = imread("Numbers\9.png");
    images = {number0 ,number1 ,number2, number3, number4, number5, number6,
   number7, number8, number9};
end
```

Source Code 14: show numbers on UI

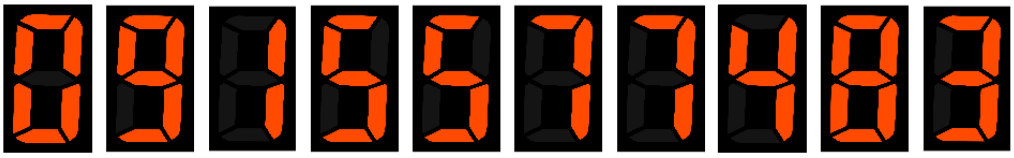

Figure 8: Dialing1

Figure 9: Dialing2

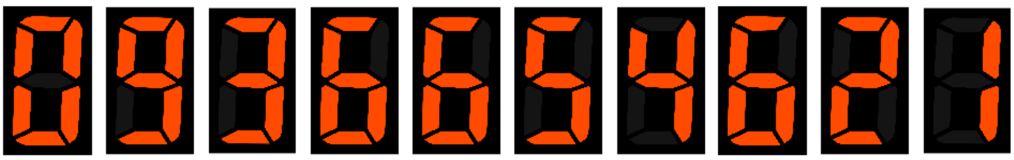

Figure 10: Dialing3

To get real dialings, I just another zero remover function and yet another function to deal with it, but the general method stays the same.

```
function numbers = real_number_dector(str)
    [y,fs]=audioread("Audio/"+str+".wav");
    [runstarts,runends] = real_zero_remover(y);
    y = y';
    runends = runends';
    runstarts = runstarts';
    runends= [1  runends];
    runstarts = [runstarts length(y)];
    numbers = [];
    n = length(runstarts);
    for i = 1 : n-1
        my_fft=fftshift(fft(y(runends(i):runstarts(i))));
        N = runstarts(i) - runends(i);
        fft_oneside=my_fft(1:N);
        dfft=abs(fft_oneside)/(N/2);
        [maxVals, maxIdxs] = maxk(dfft, 4);
        number = match_number_frequency(maxIdxs,N);
        numbers = [numbers number];
    end
    numbers = int8(numbers);
end
```

Source Code 15: Factorize number and get DFFT's max indices

and the real zero remover returns starts and ends of dialings:

```
function [runstarts,runends] = real_zero_remover(x)
    transitions = diff([0 ;x == 0; 0]);
    runstarts = find(transitions == 1);
    runends = find(transitions == -1);
    runlengths = runends - runstarts;
    runstarts(runlengths < 100) = [];
    runends(runlengths < 100) = [];
end
```

Source Code 16: Function for removing zero samples results for real dialings are as follows:

Figure 11: Real Dialing1

Figure 12: Real Dialing2

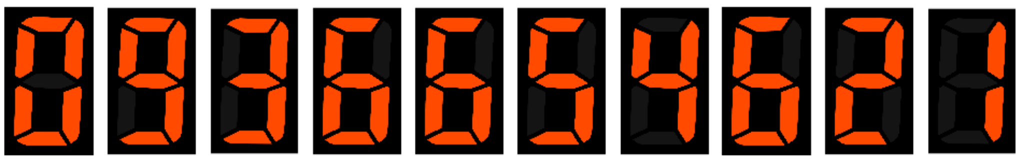

Figure 13: Real Dialing3

Introduction to Audio Processing(2)

load sound.wav and show it in frequency domain

I used this code to load it and show it in frequency domain:

```
[y,Fs]=audioread("Audio/sound.wav");

y=(y(:,1)+y(:,2))/2;
N=length(y);
x=Fs*(-N/2:N/2-1)/N;
dfft = fourier_transform(y , N);

figure
plot(x,dfft)
title('dfft for given sound','Interpreter','latex','FontSize',15)
xlabel('f','Interpreter','latex','FontSize',13)
```

Source Code 17: load sound and plot it in frequency domain

Given sound in frequency domain looks like this:

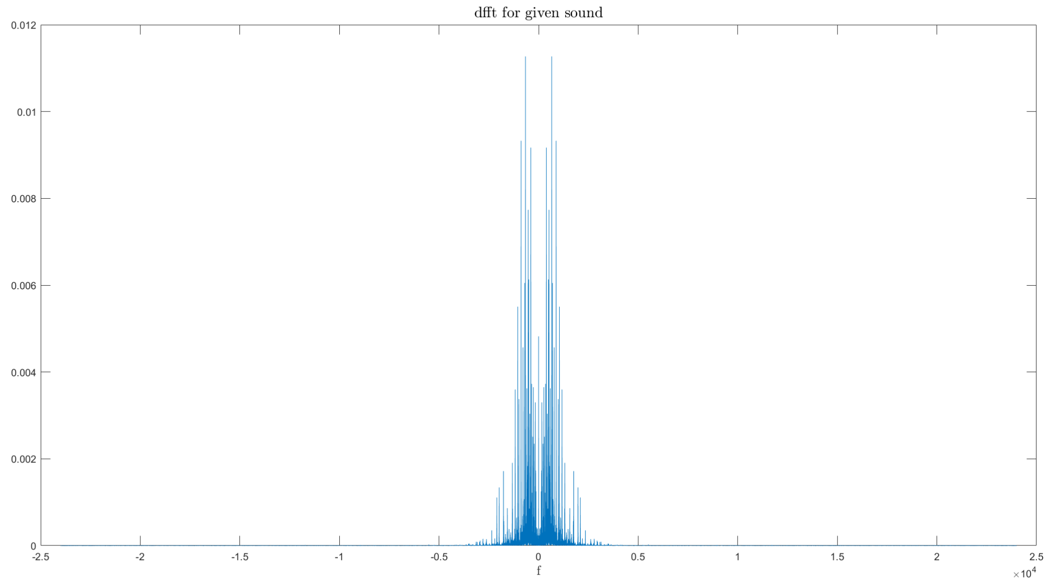

Figure 14: sound in frequency domain

Add normal noise to sound and listen to it

The code looks like this:

```
mu = 0;
sigma = sqrt(0.0025);
r = normrnd(mu,sigma,[N,1]);
noisySound = y + r;
audiowrite("noisySound.wav",y,Fs);
```

Source Code 18: add normal noiseto sound

The noisy sound is saved in results folder.

Filter sound by given filter

The given Filters code is as follows:

```
function Hd = Filt()
    %FILT Returns a discrete-time filter object.

    % Butterworth Bandpass filter

    % All frequency values are in Hz.
    Fs = 48000;  % Sampling Frequency

    N   = 10;
    Fc1 = 50;
    Fc2 = 1200;

    h  = fdesign.bandpass('N,F3dB1,F3dB2', N, Fc1, Fc2, Fs);
    Hd = design(h, 'butter');
end
```

Source Code 19: Filters function

to filter sound via given filter, I used this code:

```
my_filter=Filt();
recoverd_music=filter(my_filter,noisySound);

audiowrite("recoverdmusic.wav",recoverd_music,Fs);
```

Source Code 20: Filtering sound

The recovered sound is saved in results folder.

Plot sounds DFFT

I used this code to plot the original and noisy sound and the filtered one:

```
figure
subplot(3,1,1)
dfft = fourier_transform(noisySound , N);
plot(x,dfft)
title('Discrete fourier transform of of noisy music')

subplot(3,1,2)
dfft = fourier_transform(y , N);
plot(x,dfft)
title('Discrete fourier transform of of music')


subplot(3,1,3)
dfft = fourier_transform(recoverd_music, N);
plot(x,dfft)
title('fft Discrete fourier transform of of recoverd music')
```

Source Code 21: Plot sounds

The results are as follows:

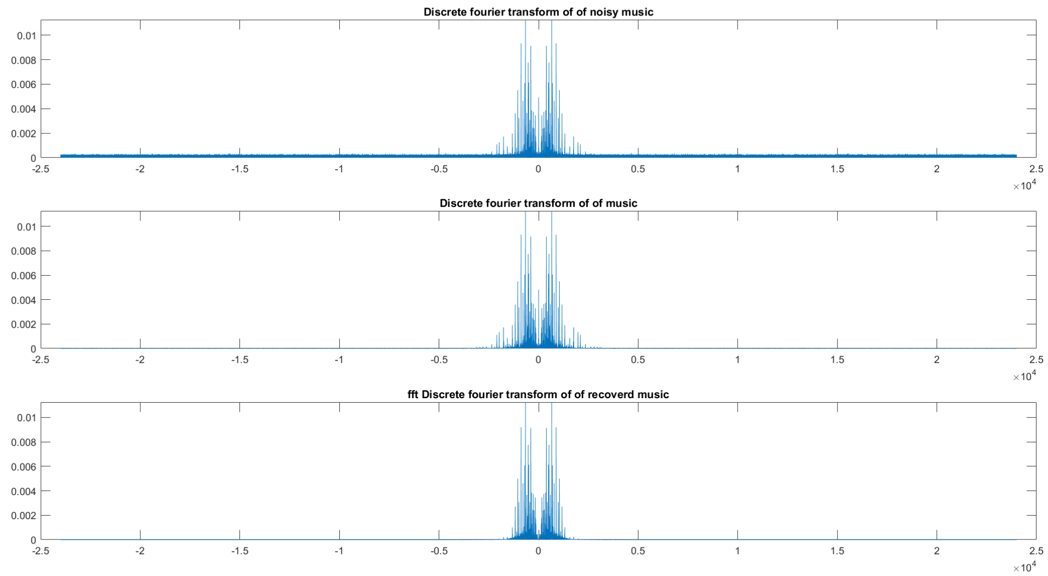

Figure 15: Plots

Create a new filter

I created a new filter via this code:

```
band_width=2300;

func_IdealFilter=@(w) (abs(w)<band_width);

arr_IdealFilter = arrayfun(func_IdealFilter,x);
arr_IdealFilter = arr_IdealFilter';

filteredMusic=arr_IdealFilter.*fftshift(fft(noisySound));
time_domain_filteredMusic=(ifft(ifftshift(filteredMusic)));
audiowrite("filteredNoisyMusic1.wav",time_domain_filteredMusic,Fs);
```

Source Code 22: Filters function

Plot recoverd signals via convolution and fft

I used this code to recover signal via convolution and fft

```
filteredMusic2=conv(ifft(arr_IdealFilter),noisySound,'same');
audiowrite("filteredNoisyMusic2.wav",filteredMusic2,Fs);

figure
subplot(2,1,1)
N=length(time_domain_filteredMusic);
x=Fs*(-N/2:N/2-1)/N;
dfft = fourier_transform(time_domain_filteredMusic , N);
plot(x,dfft)
title('Discrete fourier transform of of recoverd music via dfft')

subplot(2,1,2)
```

```
N=length(filteredMusic2);
x=Fs*(-N/2:N/2-1)/N;
dfft = fourier_transform(filteredMusic2 , N);
plot(x,dfft)
title('Discrete fourier transform of of recoverd music via convolution')
```

Source Code 23: Recover signals via convolution and fft and plot them

results are as follows: As its clear via plots, recoverd signal via fft is much clea.

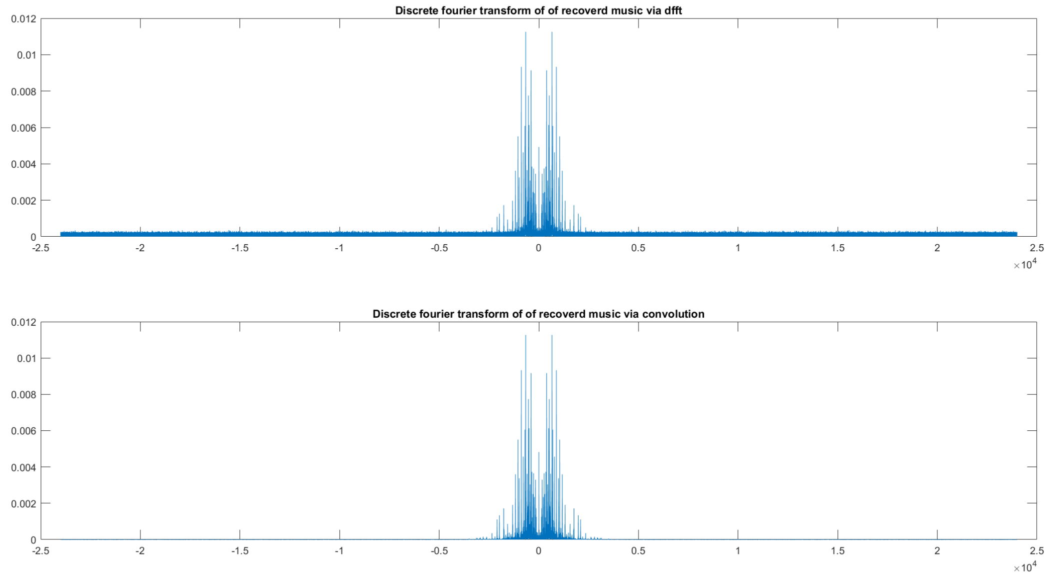

Figure 16: recoverd signals via convolution and fft

End of Computer homework 2