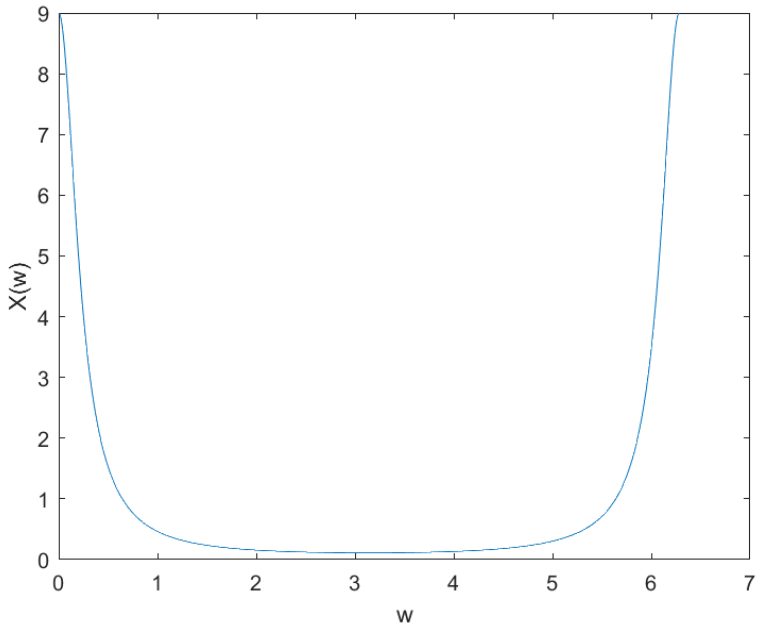
ELEC421 – HW3 - Or Bahari 51277200

1.a.

So we got the DTFT of a signal x(n) which is:

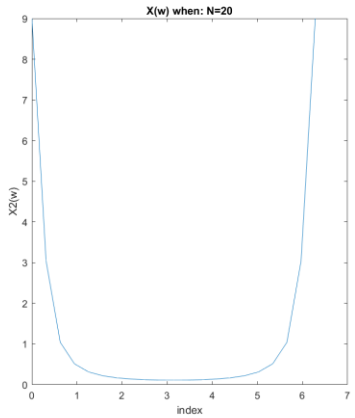
$$X(w) = \frac{1-a^2}{1-2\,a\cos(w)+a^2}\,, a = 0.8, 0 \le w \le 2\pi$$

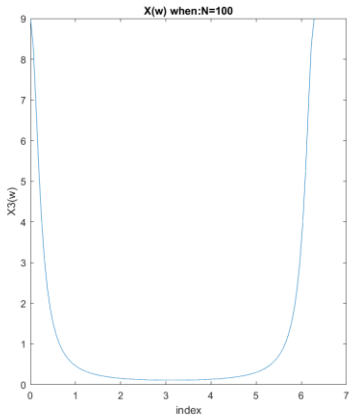


Code:

```
%1.a
a=0.8;
w=0:0.01:2*pi;
X=(1-a^2)./(1-2*a.*cos(w)+a^2);
plot (w,X);
xlabel("w");
ylabel("X(w)");
```

1.b+c.



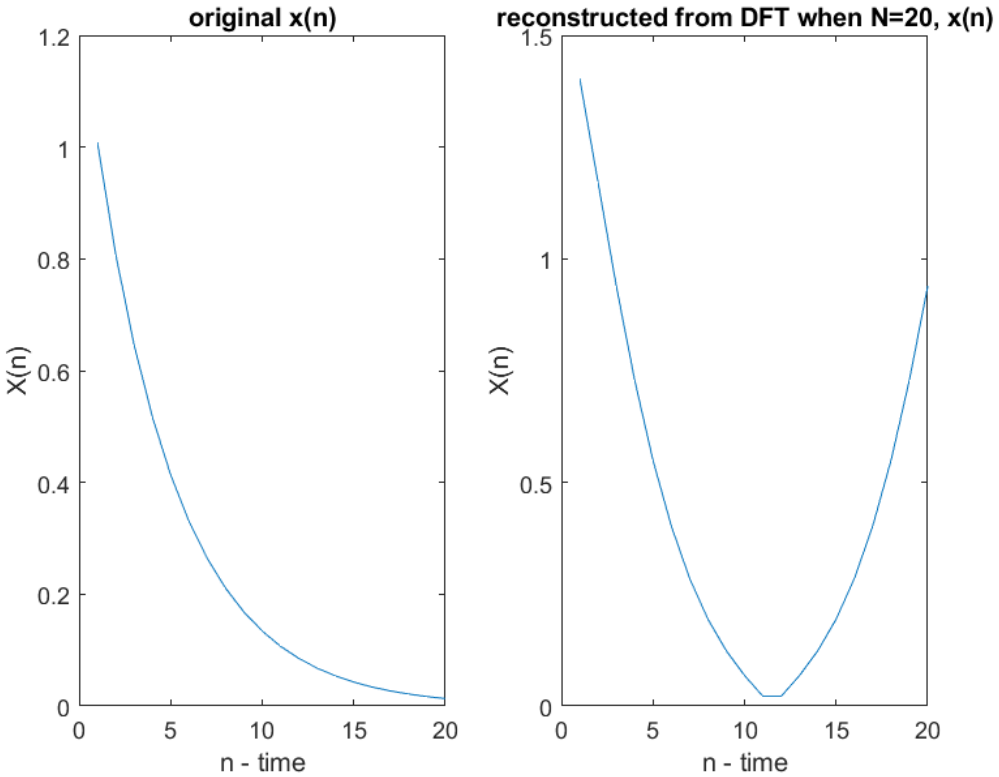


Code:

```
%1.b+c
N=20;
w=0:2*pi/N:2*pi;
X2=(1-a^2)./(1-2*a.*cos(w)+a^2);
subplot (1,2,1);
plot(w,X2);
xlabel('frequancy');
ylabel('X2(w)');
title("X(w) when: N=20");
subplot (1,2,2);
N=100;
w=0:2*pi/N:2*pi;
X3=(1-a^2)./(1-2*a.*cos(w)+a^2);
plot(w,X3);
xlabel('frequancy');
ylabel('X3(w)');
title("X(w) when:N=100");
```

1.d. with N=100 we get a plot which is more like the original continuous function. N=20 is similar but not the same. X(n) is an infinite signal (in the time domain). Therefore, the larger N is in the DFT the more accurate the reconstruction is (it will never be completely the same).

1.e.



We can see that the reconstructed signal (right plot) is different, that is because of aliasing in the time domain.

Code:

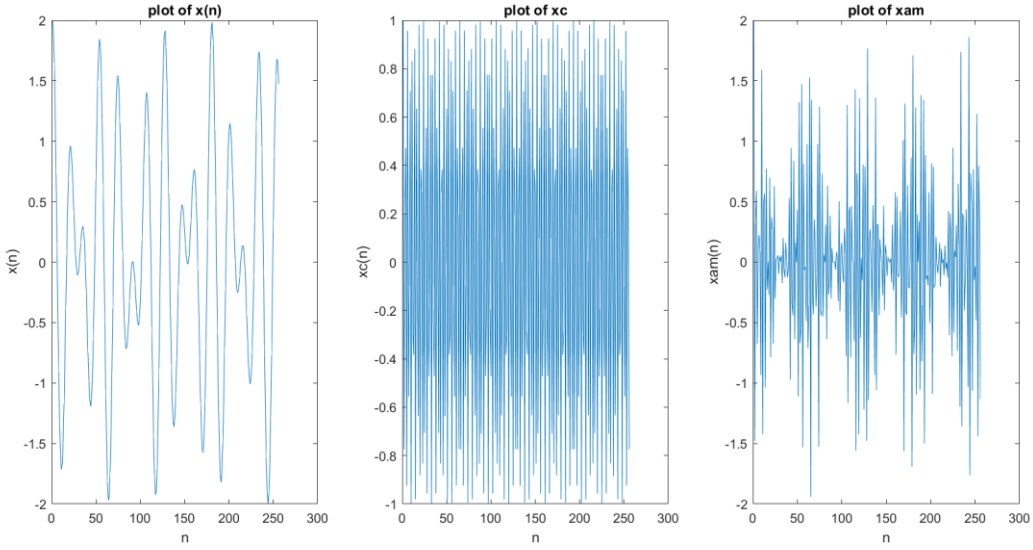
```
%1.e.
origin=abs(ifft(X));
n20=abs(ifft(X2));
figure(15);
subplot(1,2,1);
plot(origin);
title('original x(n)');
xlabel('n - time');
ylabel("X(n)");
xlim([0 20]);
subplot(1,2,2);
plot (n20);
xlim([0 20]);
title('reconstructed from DFT when N=20, x(n)');
xlabel('n - time');
ylabel("X(n)");
```

2.a.

$$x(n) = \cos(2\pi f_1) + \cos(2\pi f_2), f1 = \frac{1}{18}, f2 = \frac{5}{128}$$

$$x_c(\mathrm{n}) = \cos(2\pi f_c), fc = 50/128$$

$$x_{am}(\mathrm{n}) = x_c(\mathrm{n}) \cdot x(n)$$



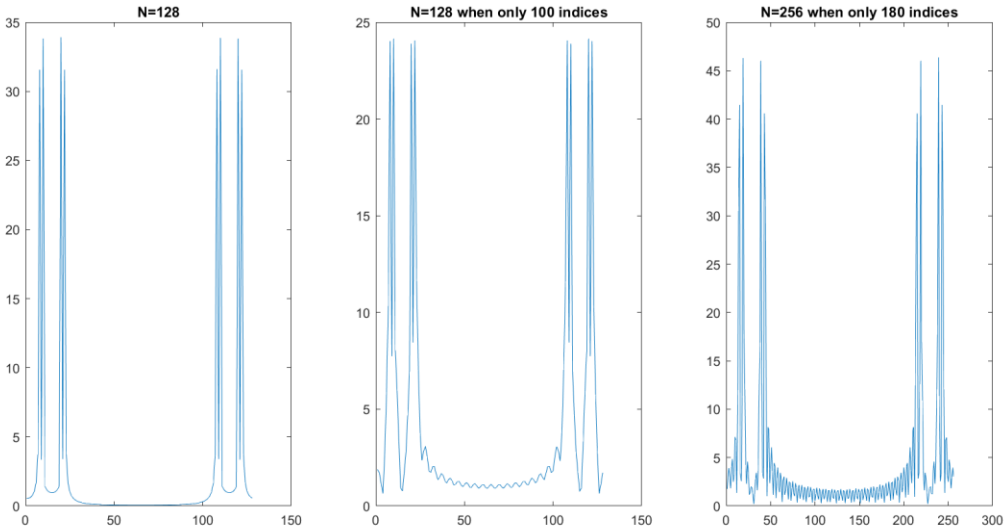
Code:

```
%2.a
f1=1/18;
f2=5/128;
fc=50/128;
n=0:1:255;
xn=cos(2*pi*f1*n)+cos(2*pi*f2*n);
xc=cos(2*pi*fc*n);
xam=xn.*xc;
figure(2);
subplot (1,3,1);
plot(xn);
xlabel("n");
ylabel("x(n)");
title("plot of x(n)");
subplot (1,3,2);
```

```
plot(xc);
xlabel("n");
ylabel("xc(n)");
title("plot of xc");
subplot (1,3,3);
plot(xam);
xlabel("n");
ylabel("xam(n)");
title("plot of xam");
```

2.b.+c.+e.

Note – These signals are Discrete, but I chose to use 'plot' instead of 'stem' just so the reader could have a better understanding.



Code:

```
%2.b+c+d
figure(21);
subplot(1,3,1);
Xam=fft(xam,128);
plot(fftshift(abs(Xam)));
title("N=128");
subplot(1,3,2);
n=0:1:99;
xn=cos(2*pi*f1*n)+cos(2*pi*f2*n);
xc=cos(2*pi*fc*n);
xam=xn.*xc;
Xam=fft(xam,128);
plot(fftshift(abs(Xam)));
title("N=128 when only 100 indices");
subplot(1,3,3);
n=0:1:179;
xn=cos(2*pi*f1*n)+cos(2*pi*f2*n);
xc=cos(2*pi*fc*n);
xam=xn.*xc;
Xam=fft(xam,256);
plot(fftshift(abs(Xam)));
title("N=256 when only 180 indices");
```

2.e.

Let's play with xam(n):

$x_{am}(\mathrm{n}) = x_c(\mathrm{n}) \cdot x(n) = \cos(2\pi f_1) \cdot \cos(2\pi f_c) + \cos(2\pi f_2) \cdot \cos(2\pi f_c)$

According to trigonometric identity the frequencies we will get are:

F1=f1+fc, F2=fc-f1, F3=f2+fc, F4=fc-f2
So we get the frequencies:0.43,0.35,0.44,0.33. So we have 4 frequencies, meaning 8 spikes in the Fourier transform – which is indeed what we get in all of them. There is not aliasing, and the resolution is good enough in all of them as the smallest |f1-f2|=0.02 and 1/L=1/128=0.07. which means that our resolution is good enough to show all the spikes (also 1/256 which is even better). So, D has a better resolution in frequency, but B has the lowest leakage.

3.a+b.

MSE of 3.a:

mse =

45.5993

MSE of 3.b:

mse_cb =

11.1698

mse_cr =

12.2273

Code:

```
%3
figure(3);
img=imread('peppers.png');
subplot (1,3,1);
converted_img=rgb2ycbcr(img);
imshow(img);
title("origin");
title("converted");
d_sam=imresize(converted_img,0.25,'bilinear');
mod_Y=imresize(d_sam(:,:,1),4,'bilinear');
mse = immse(mod_Y,converted_img(:,:,1));
mse; %report mse
mod_img=cat(3,imresize(d_sam(:,:,1),4,'bilinear'),converted_img(:,:,2),converted_img(:,:,3));
mod_rgb=ycbcr2rgb(mod_img);
subplot(1,3,2);
imshow(mod_rgb);
title('Luma resample');

mod_cb=imresize(d_sam(:,:,2),4,'bilinear');
mod_cr=imresize(d_sam(:,:,3),4,'bilinear');
mse_cb = immse(mod_cb,converted_img(:,:,2));
mse_cr= immse(mod_cr,converted_img(:,:,3));
```

```
mse_cb;mse_cr; %reporting!
mod_img=cat(3,converted_img(:,:,1),mod_cb,mod_cr);
mod_rgb=ycbcr2rgb(mod_img);
subplot(1,3,3);
imshow(mod_rgb);
title('Cb+Cr resample');
```

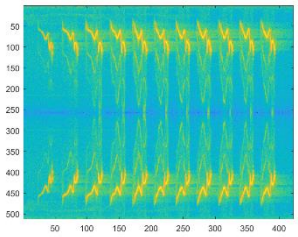
3.c.

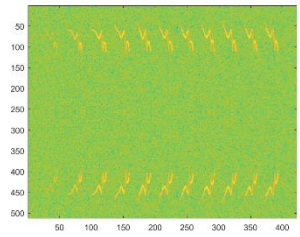
The LUMA resample image is blurry and the Cr/Cb resampled image is pretty much as the original.

The smaller MSE in 3.b. also shows that there is less difference to the original image when resampling Cb and Cr.

Therefore, when compressing, we should compress the Cb and Cr channels as they don't; really effect the image. the Y channel down sampling effects the sharpness of the image and we rather keep it as it is.

4.a.
w – window size.
The larger the window is the more resolution we have in frequency and the less we have in time and vice versa).
q - the distance between windows.
This is the result of running the flie:





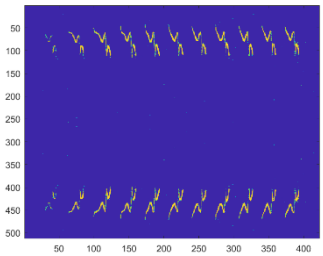
4.b.

We indeed can hear that the denoising filter got rid of the quiet background noise. And also we can see it from the spectrogram.

Code:

```
%% Denoise
stnt = perform_thresholding(stn, 2*sigma, 'hard');
%figure; imagesc(log(abs(stnt)));

%my code
stnt_time=perform_stft(stnt,w,q,options);
sound(stnt_time,fs);
%4.b - We can hear less background noises
```

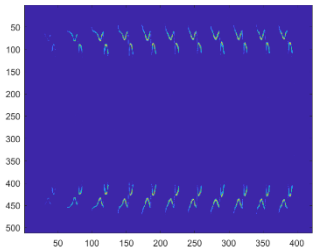


4.c.

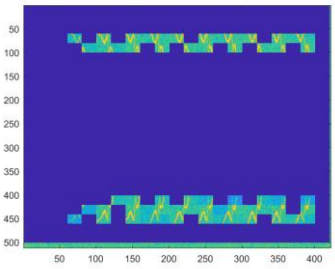
The results – also a signal with less background noise.

I played with the block_size and found that 1 does well. (there are more numbers that do the denoise well, for example 20 is not a good number, it keeps the noise).

Block_size=1: (good)



Block_size=20: (not good!)




```
%4.c
options.block_size=1;
stnt = perform_thresholding(stn, 2*sigma, 'block',options);
stnt_time=perform_stft(stnt,w,q,options);
figure; imagesc(log(abs(stnt)));

sound(stnt_time,fs);
```

5.a.

So in this function: homomorphic(Img, 2, .25, 2, 0, 5)

Img - The greyscale image.

2 - ratio between high frequency and low frequency. Boosting the higher frequency values more and decreasing the lower frequency values (reducing the visibility

of the smaller details).

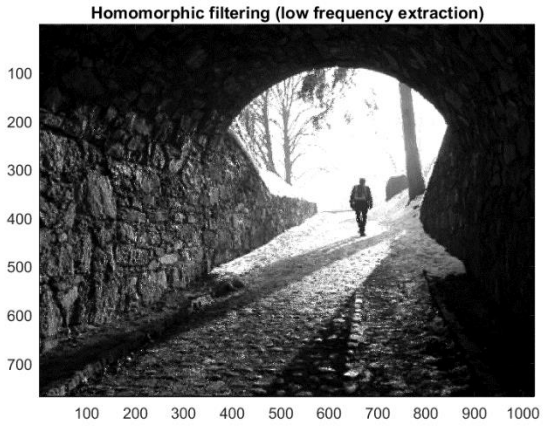
.25 - Cutoff frequency value of the filter we are going to create and use.

2 - Order of the filter, determines the steepness of the filter.

0 - The lower intensity level of the image. Removing the extreme minimal values.

5 - The higher intensity level of the image. Removing the extreme maximal values.

5.b.



Code:

```
%5.b
newim = homomorphic(Img, -2, .25, 2, 2, 10);
figure(52);
imagesc(newim);
title('Homomorphic filtering (low frequency extraction)');
axis image; colormap gray;
```

5.c.

Highboostfilter.m - creates a filter that will be convolved with our signal. the function checks for validity which is validity of n (filter's order) and for cut-off frequency and then if boost>1 it means we are amplifying this frequencies otherwise it's means we are lowering these frequencies (that's why for boost>1 we use highpassfilter and otherwise lowpassfilter).