REPORT

Farbod Khodadadi 810102545

"CA2 Report"

1_2_1) using size function we clearly see that S is [697344,2] matrix. The sampling rate (sampling_Frequency) is also 48000 Hz.

After running the S=S(:,1); code S is a mono sound means it's just a [697344,1] matrix.

1_2_2) use sound function to play the normalized dataset.

1-2-3) The volume of our output has gone up due the normalization

$\Rightarrow$ before being normalized $\rightarrow$ Amplitud $\in \{a,b\} \rightarrow$ $0<b<1$ and $-1<a<0$

after normalization $\rightarrow$ Amplitude $\in \{0,1\}$

also the sound is no longer steno so it seems like coming from a single point.

```
%1-2-1)
[S ,Fs] = audioread('Recording.m4a');

S_dim=size(S)

S=S(:,1);

%1-2-2)

S = S/max(abs(S));
sound(S,Fs);

%1-2-3)
audiowrite('Recording_out.wav',S,Fs);
```

Part 2)

2) → in down sampling we use a diffrent Frequency or sampling rate to sample only specific datas from a dataset → we have a down sampling rate $\Rightarrow K \Leftarrow \frac{\text{new\_freq}}{\text{org\_freq}}$, $\frac{1}{K} = \left(\frac{\text{new\_freq}}{\text{org\_freq}}\right)^{-1}$

new_Freq	org_Freq	K	1/K
40 K	48K	0,83	1,2
20 K	"	0,4	2,5
10 K	"	0,2	5
5 K	"	0,104	9,6
2 K	"	0,041	24,4
1 K	"	0,02	50

*) note that 1/K for down sampling must be an int to work properly.

```
n=5000;
m=Fs;
k=n/m;

%For other desired frequencies:
%n=(desired frequency)
%m=Fs
%Fs_new = n

Fs_down=Fs * k;
S_down=S(1:round(1/k):end);
S_down_dim=size(S_down);
sound(S_down,Fs_down);
```

Part 3)

3) 3-1) when we use $2 \times F_s$ and use this sampling Frequency to play our original dataset. we approximatly send 96000 samples each time to the audioplaye device means we send our samples 2 time Faster to play so we hear it with 2x speed!

3-2) $S_{new}(t) = S(t/2)$ ← down-sample by factor of 2

$\hookrightarrow S_{new}(f) = \int_{-\infty}^{+\infty} S(t/2)\, e^{-j(2\pi)ft}\, dt \rightarrow f = f_{org}$

$\hookrightarrow t/2 = t' ; \quad \rightarrow S_{new}(f) = \int_{-\infty}^{+\infty} 2S(t')\, e^{-j(2\pi f)2t'}\, dt'$

$\hookrightarrow S_{new}(f) = 2\, S(2f)$ → conclusion is by down-sampling by a factor of 2 (half the samples), we compress our data in time domain which cause and expansion in frequency domain (2f).

frequency doubled so the sound we hear is high pitched and also 2x speed.

```
%3-1)

%remove comments to play sounds
sound(S,2*Fs);

%3_2)

S_2=S(1:2:end);
sound(S_2,Fs);
```

3_3)

3-3) we use buffer( ) to construct frames without for loop.

with higher fram lengths like 9000 or 1000 audio quality is better (we have more adjacent samples so less distortion) but the dropped frames also are larger so we loose more data. Best audios are 4000, 5000 and 3000. we get a balance between quality and data saved.

```
frame_len=2000;
frames=buffer(S,frame_len,0);
frames_new=frames(:,1:2:end);

s_new=frames_new(:);
sound(s_new,Fs)

for N = 1000:1000:10000
    frames = buffer(S, N, 0, 'nodelay');
    frames_new = frames(:,1:2:end);
    Frame_Dropped_Signal = frames_new(:);

    audiowrite(sprintf('Audio_For_N=%d.wav', N), Frame_Dropped_Signal, Fs);
    fprintf('Processed N = %d\n', N);
end
```

```
Processed N = 1000
Processed N = 2000
Processed N = 3000
Processed N = 4000
Processed N = 5000
Processed N = 6000
Processed N = 7000
Processed N = 8000
Processed N = 9000
Processed N = 10000
```

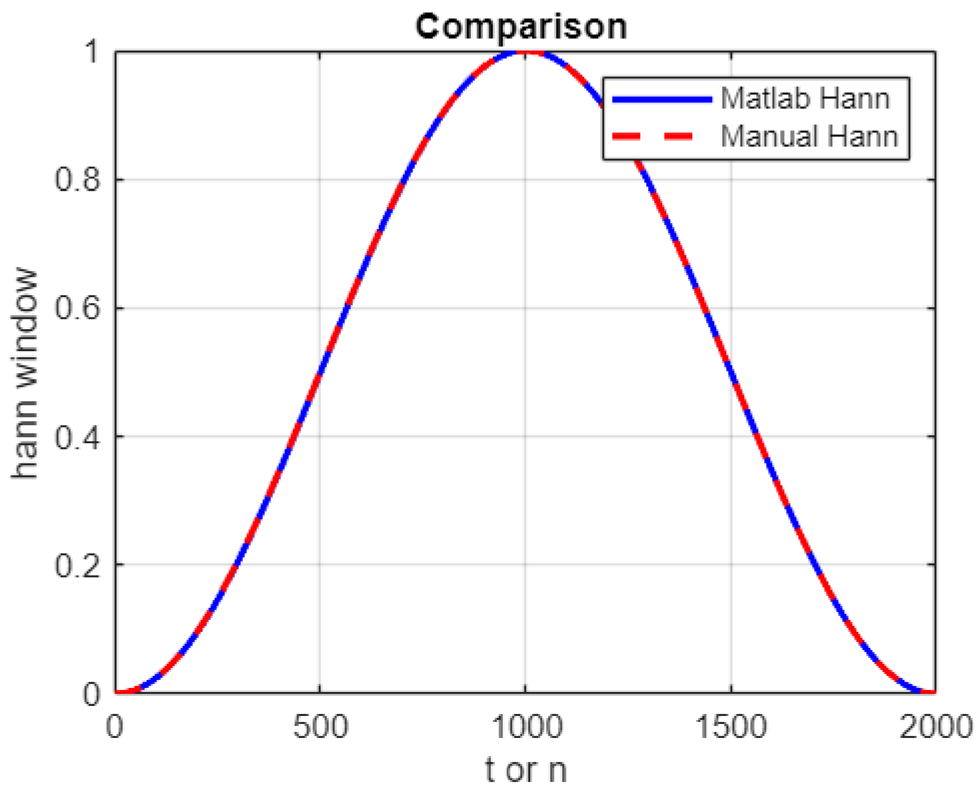
3_4)

3_4) $hann_N(t) = \sin^2\left(\frac{\pi t}{N}\right) \xrightarrow{window} hann_N[n] = \sin^2\left(\frac{\pi n}{N}\right) \quad n \in [0, N]$

$$S\_construct = \sum_{K=0}^{L} F_K(n-KN)\, h(n-KN)$$

$$\to S\_construct = \sum_{K=0}^{L} F_K(n-KN)\, \underbrace{\sin^2\left(\frac{\pi}{N}(n-KN)\right)}_{\sin^2\left(\frac{\pi n}{N} - K\pi\right)}$$

*)Comparison between Matlab built in Hann function and Manualy constructed :



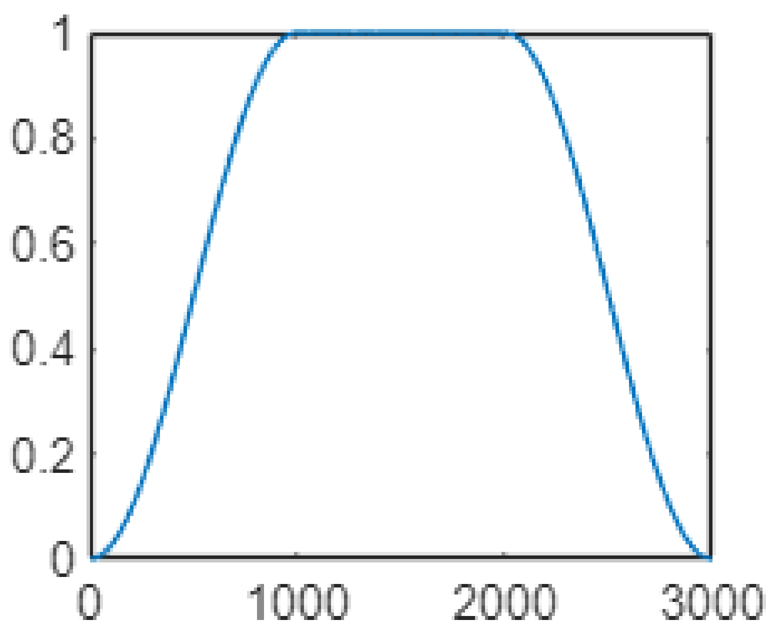

```
%constructed manualy
t=1:1:2000;
hann_win = sin(pi * t / N).^2;

win=hann(N);


figure;
plot(t,win, 'b-',t,hann_win, 'r--', 'LineWidth', 2);
xlabel('t or n');
ylabel('hann window');
title('Comparison ');
legend('Matlab Hann', 'Manual Hann');
grid on;
```

*)As seen in the Graph Hann function stisfies out condition:

For added window → $W[n] + W[n - N/2] = 1$ for $n \in [\frac{N}{2}, N]$

as seen below from $n=1000$ to $n=2000$

added winow is 1.



Last Part constructing our OverlapAdd signal and playing the sound:

For constructing our down-sampled Signal:

$$S\_d = \sum_{K=0}^{L} \underbrace{F_K(n-KN)\, h(n-KN)}_{\text{windowed\_frames}}$$

$$\text{So } S\_d = \sum_{K=0}^{[num\_frames]} frames\_win_K[n-KN] \rightarrow \text{implemented in matlab.}$$

```
win_frames =frames .* win;
win_frames_downed = win_frames(:,1:2:end);

num_frames_downed =size(win_frames_downed, 2);

S_4 = zeros(N/2 * num_frames_downed, 1);
for i = 1:num_frames_downed
    s_idx= (i-1) * N/2+ 1;
    end_idx = i *N/2;
    S_4(s_idx:end_idx) = win_frames_downed(1:N/2,i);
end

sound(S_4,Fs)
```

I also used a for loop to creat the audio output for N’s from 2000 to 7000 using Overlap and Add method:

```
for N = 2000:1000:7000
    frames=buffer(S,N,N/2);
    win=hann(N);

    win_frames =frames .* win;
    win_frames_downed = win_frames(:,1:2:end);

    num_frames_downed =size(win_frames_downed, 2);

    S_4 = zeros(N/2 * num_frames_downed, 1);
    for i = 1:num_frames_downed
        s_idx= (i-1) * N/2+ 1;
        end_idx = i *N/2;
        S_4(s_idx:end_idx) = win_frames_downed(1:N/2,i);
    end

    audiowrite(sprintf('Overlap_Add_Audio_For_N=%d.wav', N), S_4, Fs);

end
```

And last part Reapeating the process for triangular function that I manually created in matlab:

```
N=5000;
frames=buffer(S,N,N/2);

%constructed manualy
function y = tri(t)
    y = zeros(size(t));
    i= abs(t) < 1;
    y(i) = 1-abs(t(i));
end

n=1:1:N
win_tri =tri((n-(N/2))/(N/2))
win_tri=win_tri(:)


win_frames =frames .* win_tri;
win_frames_downed = win_frames(:,1:2:end);

num_frames_downed =size(win_frames_downed, 2);

S_4 = zeros(N/2 * num_frames_downed, 1);
for i = 1:num_frames_downed
    s_idx= (i-1) * N/2+ 1;
    end_idx = i *N/2;
    S_4(s_idx:end_idx) = win_frames_downed(1:N/2,i);
end

 audiowrite(sprintf('Triangular_Audio.wav'), S_4, Fs);
```

N=5000 worked slightly better than N=7000 because with shorter frame length we can easier track fast changing samples or better say frequencies and with longer frame lengths we usually have longer overlaps so some parts of audio that are not related might end up together after down sampling and shifting .

Longer length means longer shifts that causes discontinuity.

Also using Hann function is better than triangular since it’s a smoother function and triangular cant satisfy Overlap and Add condition , also its easier to reconstruct Hann windows since it’s a constant 1 from N/2 to N when shifted and multiplied.