«به نام خدا»

نام: فاطمه زهرا بخشنده استاد: دکتر احمد اکبری ازیرانی

شماره دانشجویی: 98522157

گزارش پروژه پایانی:

این تمرین دارای سه بخش است. کد هر سه بخش در فایل متلب code.m، موجود است و فایل صوتی اصلی و فایل صوتی کد شده با هافمن، و فایل های صوتی نویزی و فیلتر شده نیز همگی در فایل zip قرار دارند.

توضیحات هر بخش:

بخش اول:

- در این بخش، ابتدا يك فايل صوتی با فرمت WAV را که شامل نام و نام خانوادگی خودم هست بارگذاري مي كنيم.

```
%% Section 1
fprintf("Section 1:\n\n");

%% Loading audio wav file

voice = 'my name.wav';
% Read the data back into MATLAB
[y, Fs] = audioread(voice);

% create audioplayer object for wav file
player = audioplayer(y, Fs);
```

- اندازه صوت را با دستور size بدست می آوریم. ابتدا شکل صوت را با size(y) و سپس اندازه آن که همان تعداد سمبل هاست را با size(y, 1) بدست می آوریم که مقدار آن 157696 است.

- به تعداد 157696 سمبل داريم كه هر يك از جنس double است كه دقت آن تا ۷ رقم اعشار هست. براي ذخيره سازي اين فرمت نياز به ۸ بايت داريم .

- نرخ سمبل در ثانيه برابر 44100 شده است. يعني در هر ثانيه اين تعداد سمبل داريم. از تقسيم تعداد سمبل در نرخ سمبل بر ثانيه مي توانيم به زمان فايل صوتي برسيم. كه اين عدد 3.575873 شده است. كه اگر زمان فايل صوتی موجود در زيپ را نگاه كنيم ميبينيم كه همين مقدار است.

```
%% Questions

%m = audioinfo(voice);
%fprintf("Symbols: %d \n", m.TotalSamples);

% Number of symbols
disp("Shape of audio file:");
disp(size(y));
n = size(y, 1);
ch = size(y, 2);
fprintf("Number of symbols: %d\n", n); %Total Samples
fprintf("Each symbol has %d channels.\n", ch);

% Symbol rate = Fs
fprintf("Symbol rate: %d [symbols/second]\n", Fs);

% Time of audio file
t = Fs \ n;
fprintf("Time: %d\n", t);
fprintf("-----------------------------------------\n");

ys = y(:, 1);

% middle symbol type and value
middle = ys(floor(n/2), 1);
fprintf("Middle symbol type: %s\n", class(middle));
fprintf("Middle symbol value: %f\n", middle);
fprintf("-----------------------------------------\n");

fprintf("Min symbol value: %f\n", min(ys)); % min symbol value
fprintf("Max symbol value: %f\n", max(ys)); % max symbol value
fprintf("-----------------------------------------\n\n");
```

خروجی این بخش در متلب به صورت زیر است:

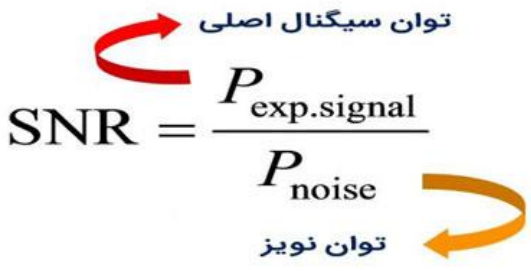
```
Section 1:


Shape of audio file:
      157696           1


Number of symbols: 157696
Each symbol has 1 channels.
Symbol rate: 44100 [symbols/second]
Time: 3.575873e+00
-------------------------------------------
Middle symbol type: double
Middle symbol value: 0.000549
-------------------------------------------
Min symbol value: -0.169983
Max symbol value: 0.204407
-------------------------------------------
```

بخش دوم:

- نسبت توان سیگنال به توان نویز SNR نامیده مي شود. هرچه این مقدار بیشتر باشد یعني نویز کمتر مي باشد و قطعا کیفیت لینک ارتباطی که از آن استفاده می کنیم هم بهتر خواهد بود.



SNR با فرمول بالا بدست می آید. برای نشان دادن آن در واحد دسی بل، از لگاریتم استفاده می کنیم و 10*log(snr) را بدست می آوریم.

در واقع می توانیم snr را با تابع زیر محاسبه کنیم:

```
function snr = snr(x, y)
  noisy = cast(x, 'double');
  original = cast(y, 'double');

  noise = noisy - original;

  squared_signal = original .^ 2;
  squared_noise = noise .^ 2;

  sum_squared_signal = sum(squared_signal(:));
  sum_squared_noise = sum(squared_noise(:));

  snr = sum_squared_signal / sum_squared_noise;

  snr = 10 * log10(snr);
end
```

در این تابع خروجی میزان نسبت سیگنال به نویز با واحد دسی بل dB نمایش داده می شود.

تابع snr در متلب نیز کاری مشابه این تابع انجام می دهد. منتها تابع snr متلب صوت اصلی و مقدار نویز را به ترتیب به عنوان ورودی اول و دوم میگیرد. در تابع بالا x صوت نویزی و y صوت اصلی است.

- ابتدا با دستور awgn در متلب، یک بار با SNR=0 و یک بار با SNR=10 به صوت خود نویز اضافه می کنیم.

```
%% Section 2
fprintf("Section 2:\n\n");

%% add White Noise

% Add white Gaussian noise to audio
SNR = 0;
yn0 = awgn(y, SNR, 'measured');
player1 = audioplayer(yn0, Fs);

SNR = 10;
yn10 = awgn(y, SNR, 'measured');
player2 = audioplayer(yn10, Fs);
```

حالا yn0 یک صوت نویزی با SNR=0 و yn10 یک صوت نویزی با SNR=10 می باشد.

- صوت های نویزی را فیلتر می کنیم تا رفع نویز شوند.

برای این منظور از دو تابع filter و wiener2 استفاده می کنیم و عملکرد آن ها را در رفع نویز مقایسه می کنیم.

```
%% filter

[b,a] = butter(1,1000/(Fs/2),'low');

% filter noisy audio with snr=0
filtered0 = filter(b,a,yn0);

% Remove noise - wiener
[a_wiener0, noise_out] = wiener2(yn0);
player3 = audioplayer(a_wiener0, Fs);
audiowrite('my name noisy snr=0 wiener filtered.wav', a_wiener0,
Fs);

% filter noisy audio with snr=10
filtered10 = filter(b, a,yn10);

% Remove noise - wiener
[a_wiener10, noise_out] = wiener2(yn10);
player4 = audioplayer(a_wiener10, Fs);
audiowrite('my name noisy snr=10 wiener filtered.wav',
a_wiener10, Fs);

% print SNRs
fprintf("noisy audio with snr=0 SNR: %.1f\n", snr(y, yn0-y));
fprintf("noisy audio with snr=0 filtered with filter function
SNR: %.4f\n", snr(y, filtered0 - y));
fprintf("noisy audio with snr=0 filtered with wiener function
SNR: %.4f\n\n", snr(y, a_wiener0 - y));

fprintf("noisy audio with snr=10  SNR: %.1f\n", snr(y, yn10-y));
fprintf("noisy audio with snr=10 filtered with filter function
SNR: %.4f\n", snr(y, filtered10 - y));
fprintf("noisy audio with snr=10 filtered with wiener function
SNR: %.4f\n\n", snr(y, a_wiener10 - y));
fprintf("this section has 2 figures and 5 voices.\nwait for all
sections to complete to see them.\n");
fprintf("-----------------------------------------\n\n");
```

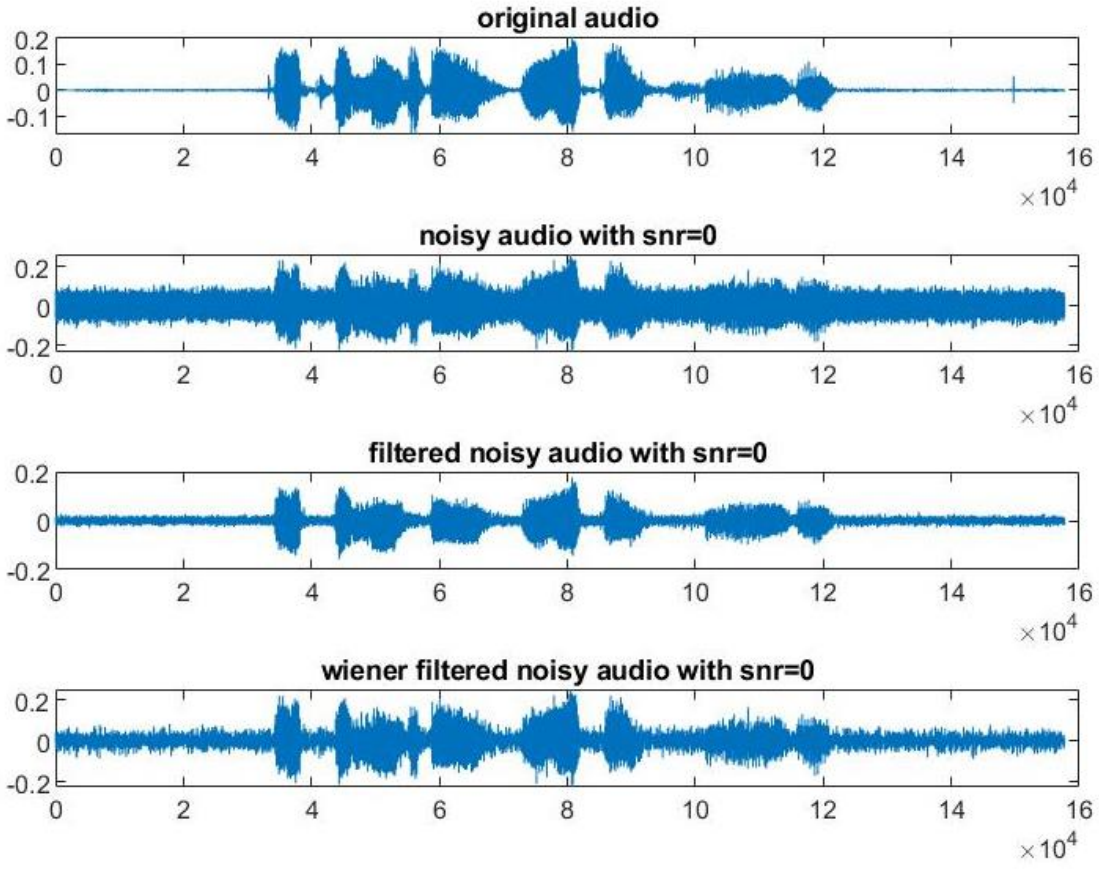
خروجی این بخش به صورت زیر است:

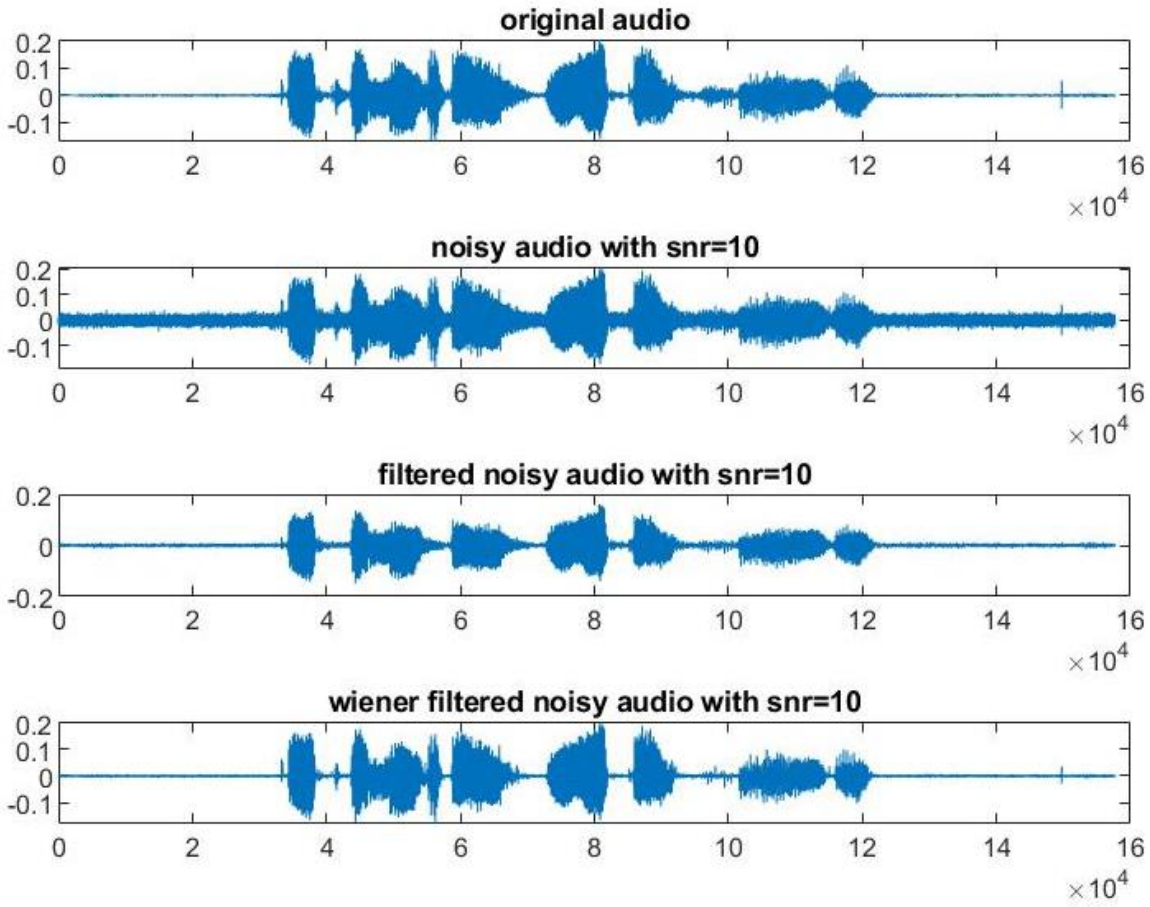
```
Section 2:


noisy audio with snr=0 SNR: -0.0
noisy audio with snr=0 filtered with filter function SNR: 5.3357
noisy audio with snr=0 filtered with wiener function SNR: 5.2643


noisy audio with snr=10  SNR: 10.0
noisy audio with snr=10 filtered with filter function SNR: 6.2659
noisy audio with snr=10 filtered with wiener function SNR: 10.1499


this section has 2 figures and 5 voices.
wait for all sections to complete to see them.
```





صوت اصلی، صوت های نویزی، و صوت های فیلتر شده را رسم کردیم تا باهم مقایسه کنیم. کد رسم آن ها در فایل متلب code.m موجود است.

طبق نتایج میبینیم که وقتی SNR مقدار کمتری دارد مقدار نویز کمتر و صوت نویزی بیشتر شبیه به صوت اصلی است.

برای رفع نویز از دو فیلتر استفاده کردم. تابع filetr که فیلتر پایین‌گذر انجام میدهد و تابع wiener2.

در مورد فیلتر wiener، می توان گفت این فیلتر در پردازش سیگنال برای تولید تخمینی از سیگنال مورد نظر با فیلتر کردن خطی با زمان ثابت (LTI) یک فرآیند نویز مشاهده‌شده، با فرض سیگنال ثابت و طیف نویز و نویز افزایشی استفاده می‌شود.

در مورد این دو فیلتر با مقایسه نسبت های snr و دیدن شکل سیگنال های فیلتر شده نتیجه میگیریم:

- ✓ هر دو نوع فیلتر در سیگنالی که خیلی نویزی بود و SNR آن 0 بود عملکرد مشابهی داشته و سیگنالی با SNR تقریبا برابر 5 به ما می دهند.

- ✓ اما در سیگنالی که نویز کمی داشت و SNR آن 10 بود low pass filter عملکرد خوبی نداشته و SNR آن نسبت به حالت نویزی افت کرده است. زیرا در این حالت نویز ما زیاد نبوده و با فیلتر پایین‌گذر باعث می شویم صوت فیلتر شده از صوت اصلی دور شود.

- ✓ اما فیلتر wiener در این حالت نیز عملکرد خوبی داشته و SNR سیگنال فیلتر شده نسبت به سیگنال نویزی افزایش یافته است و برابر 10.1499 شده است.

پس در کل فیلتر wiener عملکرد خوبی داشته است. و از روی شکل نیز مشخص است که با استفاده از این فیلتر مقدار زیادی رفع نویز کرده ایم و به صوت اصلی نزدیک شده ایم.

5 فایل صوتی در انتهای اجرای کد متلب، پشت سر هم play می شوند که فایل اول، صوت اصلی، دومی صوت نویزی با SNR=0، سومی صوت نویزی با SNR=0 پس از فیلتر با wiener، چهارمی صوت نویزی با SNR=10 و پنجمی صوت نویزی با SNR=10 پس از فیلتر با wiener می باشد.

این صوت ها در فایل zip نیز قرار دارند.

بخش سوم:

- در این بخش، ابتدا صوت خود را با کد کننده هافمن کد می کنیم. و سپس آن را در سیستم در کنار فایل های دیگر، ذخیره می کنیم.
  ابتدا با استفاده از دستور huffmandict که يك الفبا و يك توزيع احتمال مي گيرد، ديكشنري هافمن را مي سازيم. پس از توليد الفبا، با استفاده از دستور huffmanenco رشته كد شده را توليد و درون فايل صوتي جديد ذخيره مي كنيم.

```
[h, alphabet]= hist(y, unique(y));
p = h/sum(h);
[dict, len] = huffmandict(alphabet, p);
encoded = huffmanenco(y, dict);

% Save
audiowrite('my name compressed.wav', encoded, Fs);
```

چون با unique(y) تعداد سمبل های زیادی انتخاب می شوند و در نهایت کیفیت ما دقیقتر خواهد بود، کد کردن آن کمی طول می کشد.

- سایز سیگنال اصلی و سیگنال کد شده را به صورت زیر مقایسه می کنیم.

```
% Compare
File = dir(voice);
fprintf("Original file size: %d\n", 8 * File.bytes);
fprintf("Compressed file size: %d\n", length(encoded));
```

که خروجی آن به صورت زیر است:

```
Section 3:


Wait for encoding to complete to see the result.

Original file size: 2523760

Compressed file size: 1741297
```

میبینیم که مقدار سایز صوت پس از فشرده سازي کمتر شده است.

اما پس از ذخیره سازي فایل فشرده شده، متوجه مي شویم حجم فایل در سیستم بیشتر شده است. دلیل آن این است که فایل اولیه خود مکانیزمي براي فشرده سازي خود داشته است که ما با باز کردن آن و تبدیل آن به بیت در متلب، فشرده سازي اولیه را از بین برده ایم.

همچنین چون کد کننده هافمن یک نوع فشرده سازی بدون اتلاف است، مستلزم این است که داده ها دور ریخته نشوند، که به نوبه خود از فضا یا پهنای باند بیشتری استفاده می کند .بر خلاف فشرده‌سازی با اتلاف، فشرده‌سازی بدون تلفات منجر به تخریب داده‌ها نمی‌شود و داده‌های فشرده‌شده مشابه نسخه اصلی غیرفشرده است. در واقع فشرده سازی بدون اتلاف کیفیت را بیش از اندازه افزایش می دهد.

- زمان لازم براي انتقال اين فايل با لينك توصيف شده به صورت زير به دست مي آيد:

$$T = \frac{\text{size}}{\text{speed rate}}$$

$$\text{Original T} = \frac{2523760}{(64 * 1000)} = 39.4338\ \text{s}$$

$$\text{Compressed T} = \frac{1741297}{(64 * 1000)} = 27.2077\ \text{s}$$

نسبت زمان انتقال صوت فشرده شده به زمان انتقال صوت اصلی تقریبا برابر 70% است.

- همانطور که دیدیم با کد کننده هافمن، تقریبا 30% از زمان انتقال را کم کردیم. اما برای انتقال اطلاعات معمولا نیازمند فشرده سازی بیشتری هستیم. پس مجبور هستیم که به سراغ کدکننده های با اتلاف برویم.
همانطور که در بخش قبل گفتیم، فشرده سازی بدون اتلاف، مستلزم این است که داده ها دور ریخته نشوند، که به نوبه خود از فضا یا پهنای باند بیشتری استفاده می کند .بر خلاف فشرده‌سازی با اتلاف، فشرده‌سازی بدون تلفات منجر به تخریب داده‌ها نمی‌شود و داده‌های فشرده‌شده مشابه نسخه اصلی غیرفشرده است. در واقع فشرده سازی بدون اتلاف کیفیت را بیش از اندازه افزایش می دهد. به همین دلیل اگر اندازه یا پهنای باند فایل نگران کننده باشد، فشرده سازی با تلفات بسیار منطقی تر است .