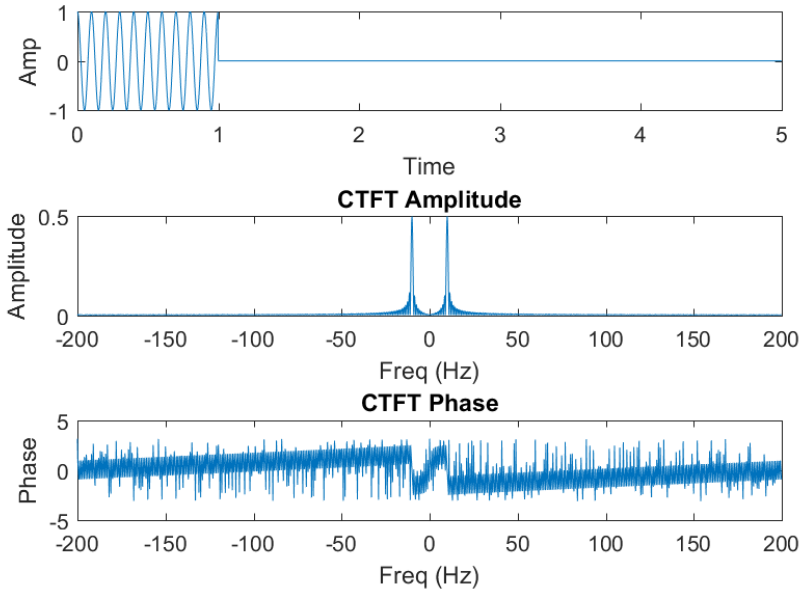
電機三 B05502145 林禹丞

Report

1. **Short-time Fourier transform:** Using the following sample $x(t)$ that is composed of a set of four sinusoidal waveforms joined together in sequence. Each waveform is only composed of one of four frequencies {10, 25, 50, 100 Hz}. Then it is sampled at 400 Hz.

$$x(t) = \begin{cases} \cos(2\pi \times 10t) & ,0 \leq t < 1\text{s} \\ \cos(2\pi \times 25t) & ,1 \leq t < 2\text{s} \\ \cos(2\pi \times 50t) & ,2 \leq t < 3\text{s} \\ \cos(2\pi \times 100t) & ,3 \leq t < 4\text{s} \\ 0 & ,\text{otherwise.} \end{cases}$$

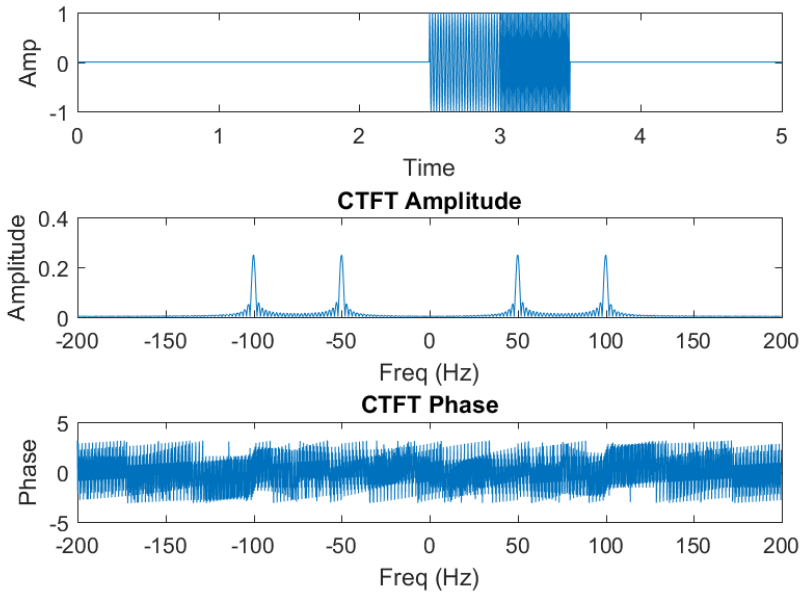
(a) Using a **rectangular window** with window size is 1s, capture $x(t)$ in $0 \leq t < 1s$. We call this section $x_1(t)$. Plot $x_1(t)$ with respect to the time t in seconds. Plot the magnitude of **CTFT** of $x_1(t)$, with the horizontal axis is the frequency f in the range of $[f_s/2; f_s/2]$. Please **add axis labels**, and **describe what you observe.**



因為0到1秒是10Hz的cos波，所以可以觀察這段區間的頻率集中在10Hz附近。

(b) Using the same window, slide the window and capture $x(t)$ in $2.5 \leq t < 3.5s$. We call this section $x_2(t)$. Plot $x_2(t)$ with respect to the time t in seconds. Plot the magnitude of **CTFT** of $x_2(t)$, with the horizontal axis is the frequency f in

the range of [f_s/2; f_s/2]. Please **add axis labels**, and **describe what you observe.**

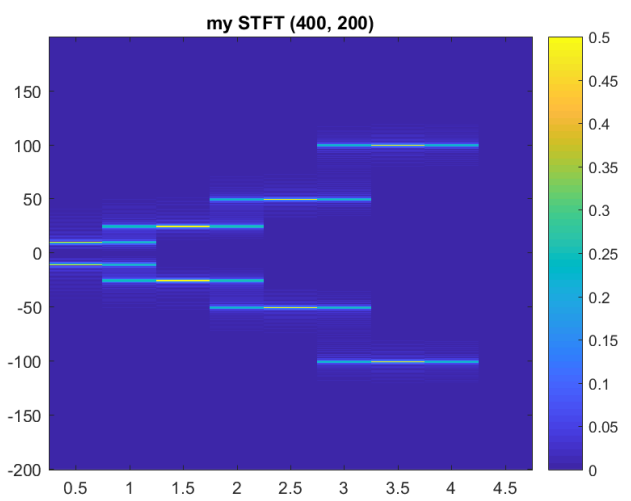


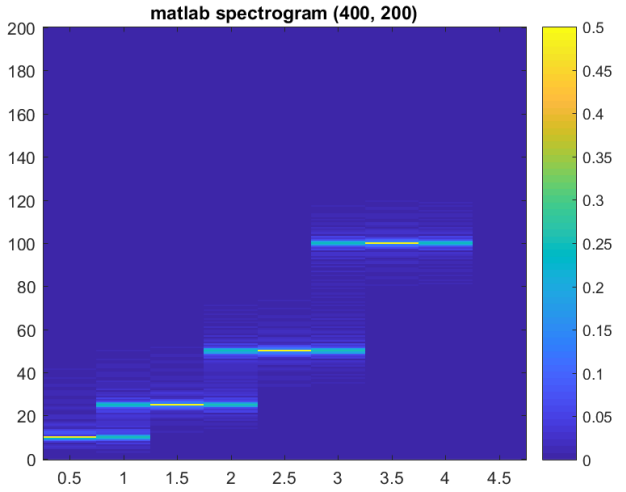
因為2.5到3秒是50Hz的cos波、3到3.5秒是100Hz的cos波，所以可以觀察這段區間的頻率集中在50和100Hz附近。

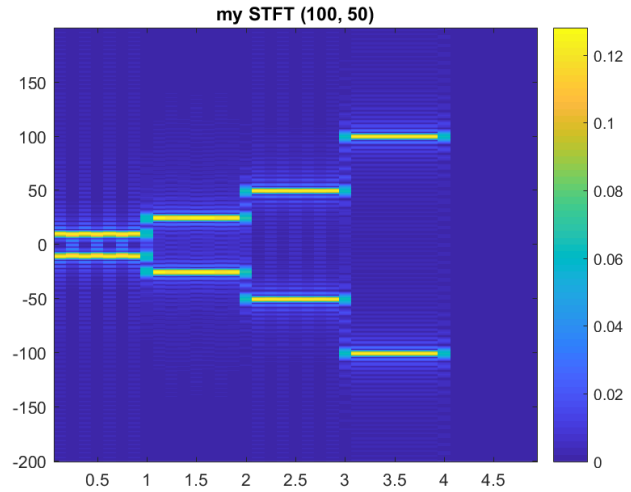
(c) Now your job is to write your own short-time Fourier transform function to implement time-frequency analysis over the *x*(*t*).

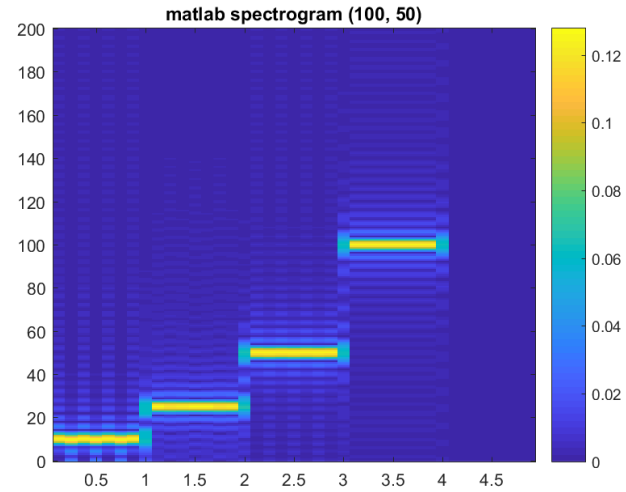
```
function [S, f, t] = STFT(x, window, Noverlap, Nfft, fs)
    % If x is a signal of length Nx, then s has k columns, where
    % k = ⌊(Nx - noverlap)/(length(window) - noverlap)⌋ if window is a vector.
    % If x is real and nfft is even, then s has (nfft/2 + 1) rows.
    % If x is real and nfft is odd, then s has (nfft + 1)/2 rows.
    % If x is complex, then s has nfft rows.
    assert(length(window) > Noverlap, 'size of window must be larger than Noverlap')
    col_size = Nfft/2+1;
    row_size = floor( (length(x) - Noverlap) / (length(window) - Noverlap) );
    S = zeros(col_size, row_size);
    t = zeros(1, row_size);
    from = 1;
    to = length(window);
    for i = 1 : row_size
        X = x(1,from:to);
        [f, A] = CTFT(X, fs, Nfft);
        S(1:end,i) = A(1:col_size);
        t(1, i) = ( from + floor(length(window) / 2) - 1)/ fs;
        from = from + (length(window) - Noverlap);
        to = to + (length(window) - Noverlap);
    end
    f = f(1:col_size);
end
```

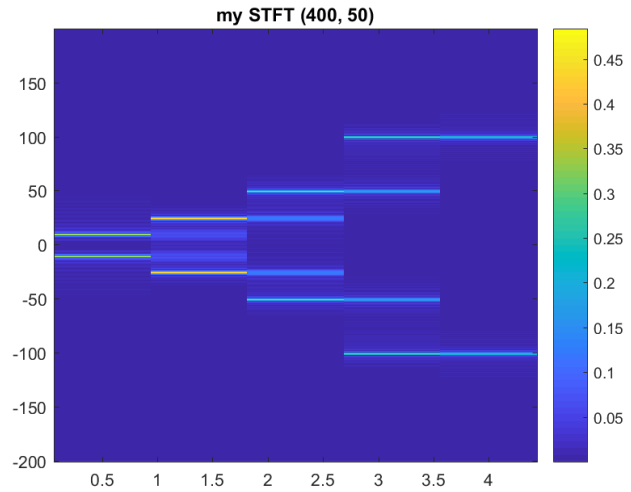
(d) Adjust the window size, and Noverlap to see how the S changes. You can use Matlab functions, surf and colorbar, to show S. Please **add axis labels** and describe what you observe.

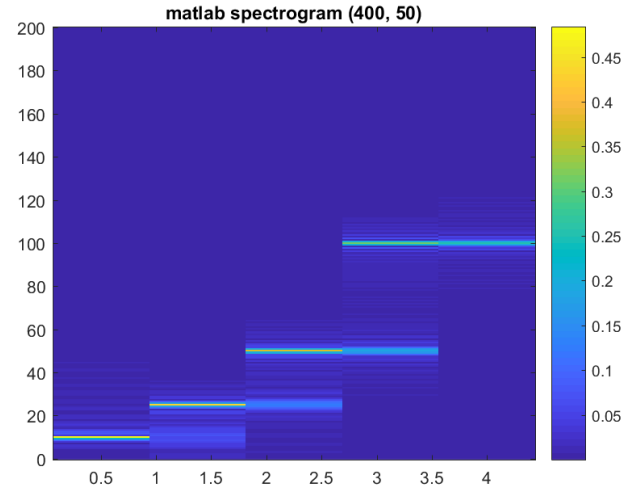








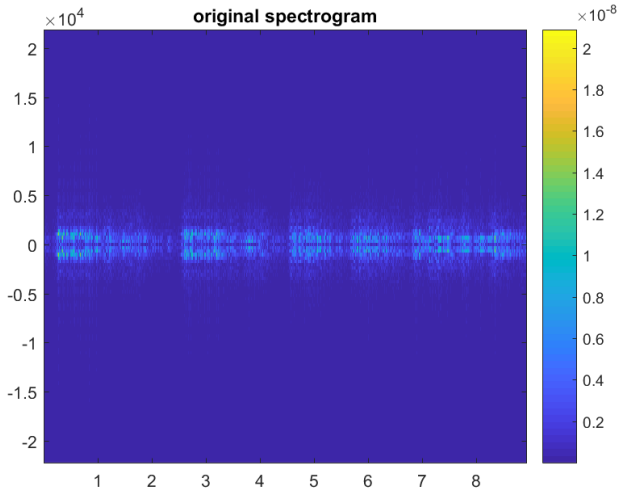


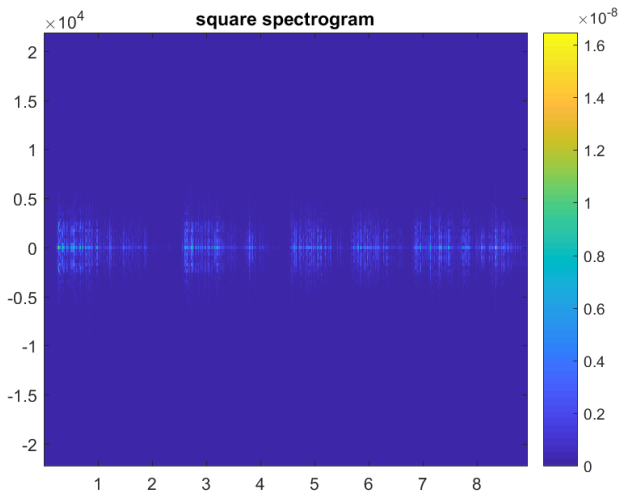


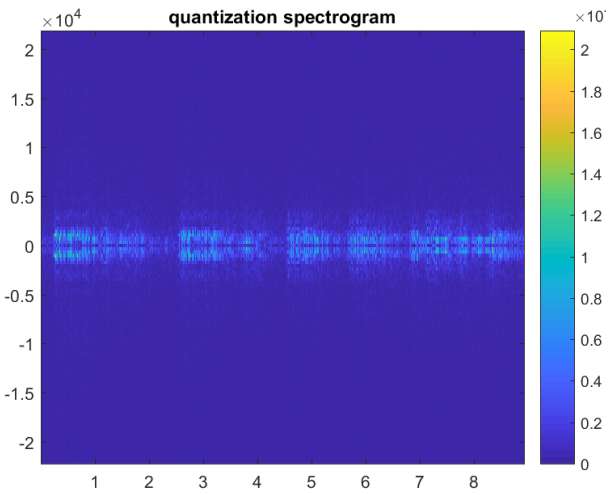
左邊是(c)中我自己實作的STFT函數，右邊是MATLAB的spectrogram函數，可以看到我們寫的STFT正頻率部分和matlab的function大致相似。圖片上

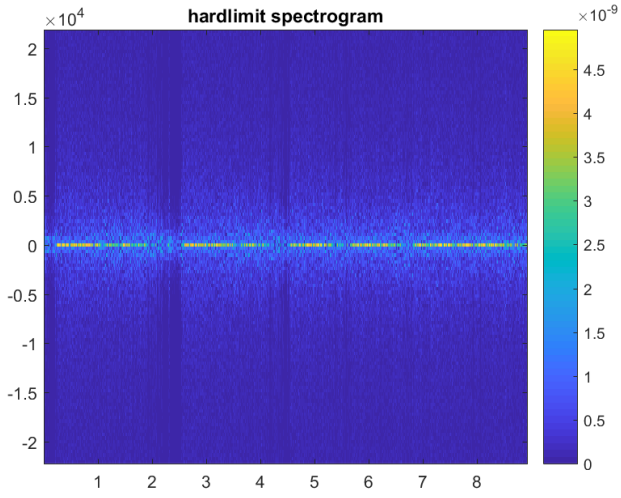
標的數字代表(windows有值的長度,Noverlap的大小)。可以觀察到當window有值的長度越長，我們可以得到的頻率就越準確，但是時間就相對地不精確；例如第一排的圖可以看到3秒右邊應該要只有100Hz的單頻，但是3秒左右還是可以看到50和100Hz的成分，但是在頻率上可以看到長條狀非常細，可以清楚的看出頻率的位置；而第二排的圖片我們可以看到時間上3秒的左邊和右邊是分得很清楚的，但是頻率的寬度就很寬，不太能判斷頻率的準確值。而Novelap參數則影響不大，主要是會影響時間上取的次數，可以看到第一排圖片有7段、第三排圖片只有5段。

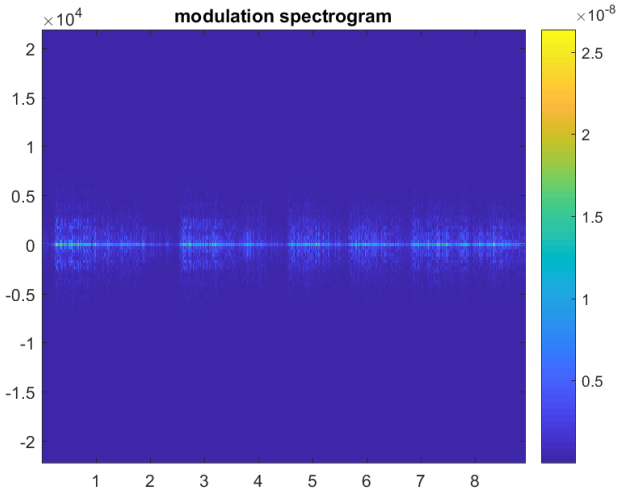
(e) Perform short-time Fourier transform over handel.ogg and processed handel.ogg in Lab1 (i.e., hard-limiting, squaring, modulation, quantization). **Show the spectrograms** and **describe what you observe**.









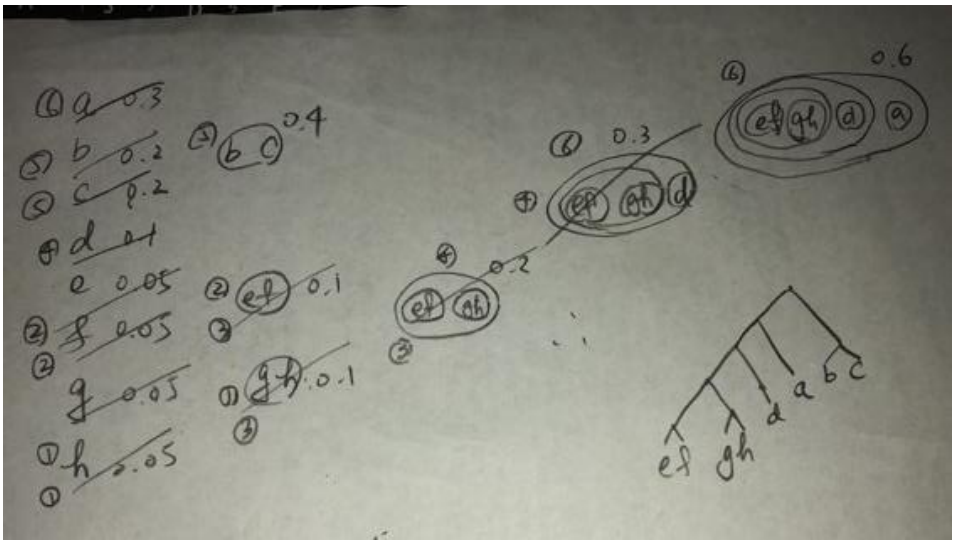


可以發現squaring後會產生些微的高頻信號；quantizing後可以發現和原信號的spectrogram相差不大；hardlimit後可以發現頻率集中在低頻區；modulation後可以發現頻率變高了，但是原本負頻區的部分變成正低頻，所以聽起來會有音高變低的感覺。

2. **Huffman coding**

   (a) Construct a Huffman tree by using these symbol. Everyone's method of implementing Huffman coding may be different, so please write on your calculation process.

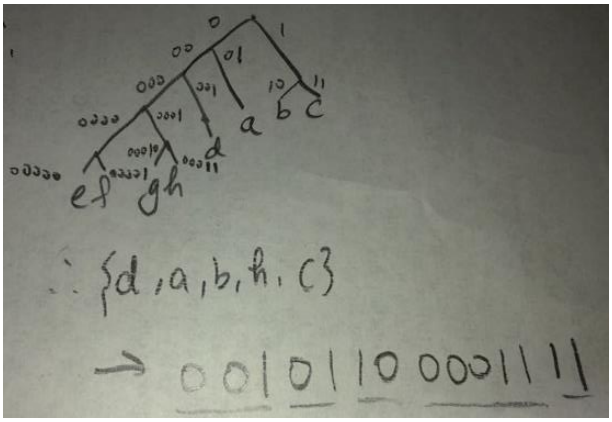
Symbol	Prob.
a	0.3
b	0.2
c	0.2
d	0.1
e	0.05
f	0.05
g	0.05
h	0.05



每次都把最小的兩個機率併成一個node，並把機率加起來放進原本的set中，持續步驟到全部都併成一棵樹。(圖中的數字代表步驟順序)

(b) Following (a), encode the sequence of symbols using the Huffman tree constructed in (a)

{d, a, b, h, c}



(c) Encode the sequence of symbols in (b) using **PCM**.

因為總共有8個symbol，所以我們月3bit還表示各個symbol，從a到z分別為000、001、010、011、100、101、110、111。所以{d, a, b, h, c }會被encode成011000001111010。

(d) Compare the data size between (b) and (d).

PCM中，因為每個symbol有3個bit，所以總共15個bit。而(b)中huffmann encoding的方法則只需要14bit。這是因為越常出現的symbol，所們可以

用比較短的bit來encode，以達到縮短bit stream的效果。

(e) In Lab1, you have written a Matlab function quantizer_L_level. Here you should write a Matlab PCM encoder function with the following auguments y = pcm_enc(x, numBits) where x is output signal of quantizer_L_level, numBits is the number of bits of each codeword, and y is PCM bit stream.

```
function y = pcm_enc(x, numBits)
    level = 2^numBits;
    delta = 2 / level;
    x = (x + (level-1)*delta/2) / delta + 1;
    y = reshape(de2bi(x)',1,[]);
end
```

(f) Write a Matlab function to construct Huffman code dictionary with following arguments: dict = huffman_dict(symbols, p) where symbols are distinct signal values of source, p is a probability vector whose kth element is the probability of kth symbol, and dict is a two-column cell array in which the first column lists the distinct signal values from symbols and the second column lists the corresponding Huffman codewords. The form of dict is same as Matlab function huffmandict

```
function dict = huffman_dict(symbols, p)
    dict = cell(length(symbols), 2);
    for i = 1 : length(symbols)
        dict{i,1} = symbols(i);
    end
    % node is a structure array
    for i = 1 : length(symbols)
        s.array = [i];
        s.prob = p(i);
        node(i) = s;
    end
    nodeb = node
    while length(nodeb) > 1
        min = 1.1;
        minmin = 1.1;
        minp = -1;
        minminp = -1;
        for i = 1:length(nodeb)
            if nodeb(i).prob < minmin
                min = minmin;
```

```
                minp = minminp;
                minmin = nodeb(i).prob;
                minminp = i;
                continue
            end
            if nodeb(i).prob < min
                min = nodeb(i).prob;
                minp = i;
            end
        end
        if minp > minminp
            temp = minp;
            minp = minminp;
            minminp = temp;
        end
        for i = 1:length(nodeb(minp).array)
            dict{nodeb(minp).array(i),2} = [0
dict{nodeb(minp).array(i),2}];
        end
        for i = 1:length(nodeb(minminp).array)
            dict{nodeb(minminp).array(i),2} = [1
dict{nodeb(minminp).array(i),2}];
        end
        s.array = [nodeb(minp).array nodeb(minminp).array];
        s.prob = nodeb(minp).prob + nodeb(minminp).prob;
        nodeb = [nodeb(1:minp-1), nodeb(minp+1:minminp-1),
nodeb(minminp+1:end)];
        nodeb = [nodeb s];
        nodeb_length = length(nodeb);
    end
end
```

(g) Write a Huffman encoder function which has the following arguments: y = huffman_enc(x, dict); where x is output signal of quantizer_L_level.

```
function y = huffman_enc(x, dict)
    C = cell(length(x),1);
    for i = 1 : length(x)
        C{i, 1} = dict{find( [dict{:,1}] == x(i)),2};
    end
```

```
    y = [C{:,1}]';
end
```

(h) Write a Huffman decoder function which has the following arguments: x_dec = huffman_dec(y, dict); where y is a bit stream, and x_dec represents the decoded x.

```
function x_dec = huffman_dec(y, dict)
    s.array = [1:length(dict)];
    s.level = 1;
    % 0 : left 1 : right
    s = split_tree(s, dict);
    st = s;
    x_dec = [];
    for i = 1 : length(y)
        if y(i) == 1
            st = st.right;
            if length(st.array) == 1
                x_dec = [x_dec; st.array(1)];
                st = s;
            end
        else
            st = st.left;
            if length(st.array) == 1
                x_dec = [x_dec; st.array(1)];
                st = s;
            end
        end
    end
    for i = 1 : length(x_dec)
        x_dec(i) = dict{x_dec(i),1}
    end
end

function st = split_tree(s, dict)
    if length(s.array) <= 1
        st = s;
        return
    end
    x.array = [];
```

```
    y.array = [];
    for i = 1 : length(s.array)
        if dict{s.array(i),2}(s.level) == 0
            x.array = [x.array s.array(i)];
        else
            y.array = [y.array s.array(i)];
        end
    end
    x.level = s.level + 1;
    y.level = s.level + 1;
    st.array = s.array;
    st.left = split_tree(x, dict);
    st.right = split_tree(y, dict);
end
```

3. **Converting Signals into Bits:** We want you to convert analog signal to digital signal via the techniques containing **sampling & quantization & mapping**. Please turn handel.ogg into bits.

(a) After sampling (i.e., audioread) and quantization (i.e., quantizer_L_level), Huffman coding needs the probability distributions of samples (symbols). Count the number of occurrences of samples, and convert it into probabilities.

(b) Following (a), map samples to bits using Huffman coding.

(c) After sampling and quantization, map samples to bits using PCM.

(d) Compare the data size of (b) and (c).

(e) Please decode the digital bits in (b) and (c) to digital signals. Compare two recovered handel.ogg.

(c)
(a)
(b)
(e)

```
% assume amp is in [-1, 1]
[x, fs] = audioread('handel.ogg');
xmax = 1;
bit = 4;
level = 2^bit;
xt = quantizer_L_level(x, xmax, level)';

% (e) PCM
y = pcm_enc(xt, 4);
length_y = length(y)

% (f) huffman dict
delta = 2 * xmax / level;
symbols = [-(level-1)*delta/2:delta:(level-1)*delta/2];
p = histc(xt, symbols);
p = p / sum(p);
dict = huffman_dict(symbols, p);
dict2 = huffmandict(symbols, p);

% (g) huffman encode
y = huffmanenco(xt, dict);
length_y = length(y);
y1 = huffman_enc(xt, dict);
length_y = length(y1);
check_enc = sum(y ~= y1)          % 0 for correct

% (h) huffman decode
xd = huffmandeco(y, dict);
length_x = length(xd);
xd2 = huffman_dec(y, dict);
length_x = length(xd2);
check_dec = sum(xd ~= xd2)        % 0 for correct
```

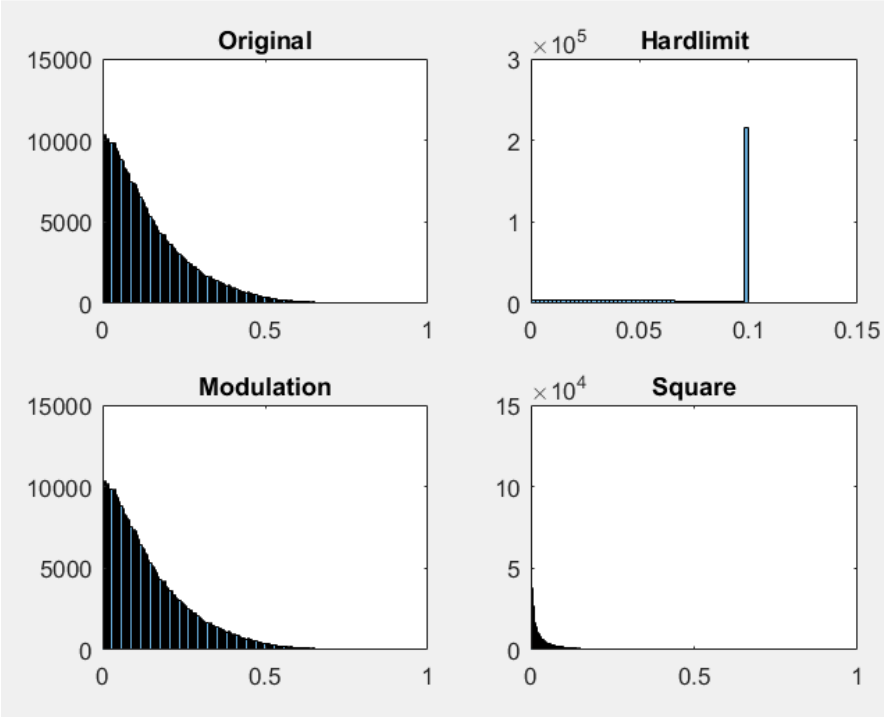
我們用自己寫的huffmax_dict、encode和decode與matlab的{huffmandict, huffmanenco, huffmandeco}比較，發現完全結果完全一樣。另外可以得到**PCM編碼有1574344個bit**、而**Huffman編碼有1070899個bit**。最後我們也比較原信號和**encode再decode**的信號，發現完全一樣。

(f) Encode the processed handel.ogg (i.e., hard-limiting, squaring, modulation) using Huffman coding and PCM, and compare the data size.

```
check_enc =

     0


check_dec =

     0
```

可以從左邊看到encode後的長度{原信號、hardlimit0.1、modulation100、square}分別由上到下。下圖中我們對他們的信號做histogram，我們可以發現**越compact(類似信號的集中度)的信號他的encode長度就越短**。

```
>> prob3

length_y =

     1117803


length_y1 =

     1065563


length_y2 =

      726787


length_y3 =

      488268
```



4. **The Average Codeword Length of Huffman Codes:** In this problem, we will study the average codeword length of Huffman codes via Monte-Carlo simulation. We will consider the symbols $\mathscr{S} = \{\mathrm{a}, \mathrm{b}, \ldots, \mathrm{h}\}$ in Table 1.

   (a) Calculate the entropy of $\mathscr{S}$, i.e. $H(\mathscr{S})$.

   (b) Find the average codeword length $\overline{L}$ of the Huffman code.

   (c) Generate a sequence of $n = 100$ symbols according to the probability in Table 1. Find the length of encoded binary data (in bits) after Huffman encoding.

   (d) Repeat the experiment in Problem 4c multiple times (say $R = 1000$ times). Each experiment randomly generates a sequence of symbols. Assume that the length of the encoded binary data for the r-th experiment is $L_n^{(r)}$. Compute the average length of the binary data according to

$$\overline{L}_n = \frac{1}{R}\sum_{r=1}^{R} L_n^{(r)}. \tag{1}$$

   (e) Compare the quantity $\overline{L}_n/n$ in Problem 4d with the entropy $H(\mathscr{S})$ and the average codeword length $\overline{L}$ in Problems 4a and 4b. Comment on your results.

   (f) Repeat Problem 4e with $n = 1000$ and $R = 10000$.

```
>> prob4
entropy =
    2.6464

avg_length =
    2.7000

v_enc_length =
   267

avg_length =
  269.8280

avg_length_d100 =
    2.6983

avg_length =
  2.7269e+03

avg_length_d1000 =
    2.7269
```

Appendix

1. Code for problem1

```
%% problem 1
```

```
% (a)
%% problem 1

% (a)
fs = 400;
I = [0:1/fs:1-1/fs];
x1 = cos(2*pi*10*I);
x2 = cos(2*pi*25*I);
x3 = cos(2*pi*50*I);
x4 = cos(2*pi*100*I);
x5 = I * 0;
x = [x1 x2 x3 x4 x5];
It = [0:1/fs:5-1/fs];


O = ones(1,fs);
Z = zeros(1,fs);
w = [O Z Z Z Z];
CTFT_draw(x.*w, fs, It);
saveas(gcf, 'Q1a.png');

% (b)
CTFT_draw(x.*circshift(w,5*fs/2), fs, It);
saveas(gcf, 'Q1b.png');

% (c)
% STFT() freq start from negative, while spectrogram() start from 0
figure;
w = ones(1,400);
noverlap = 200;
Nfft = 512;
[s1, f, t] = STFT(x, w, noverlap, Nfft, fs);
imagesc(t, f, flipud(abs(s1)));
set(gca,'YDir','normal');
title('my STFT (400, 200)')
colorbar;
saveas(gcf, 'Q1d_1.png');
```

```
figure;
w = ones(1,400);
noverlap = 200;
Nfft = 512;
[s2, f2, t2] = spectrogram(x, w, noverlap, Nfft, fs);
imagesc(t2, f2, abs(s2)/fs);
set(gca,'YDir','normal');
title('matlab spectrogram (400, 200)')
colorbar;
saveas(gcf, 'Q1d_2.png');

% (d) adjust window size, noverlap
figure;
w = ones(1,100);
noverlap = 50;
Nfft = 512;
[s1, f, t] = STFT(x, w, noverlap, Nfft, fs);
imagesc(t, f, flipud(abs(s1)));
set(gca,'YDir','normal');
title('my STFT (100, 50)')
colorbar;
saveas(gcf, 'Q1d_3.png');

figure;
w = ones(1,100);
noverlap = 50;
Nfft = 512;
[s2, f2, t2] = spectrogram(x, w, noverlap, Nfft, fs);
imagesc(t2, f2, abs(s2)/fs);
set(gca,'YDir','normal');
title('matlab spectrogram (100, 50)')
colorbar;
saveas(gcf, 'Q1d_4.png');

figure;
w = ones(1,400);
noverlap = 50;
Nfft = 512;
```

```
[s1, f, t] = STFT(x, w, noverlap, Nfft, fs);
imagesc(t, f, flipud(abs(s1)));
set(gca,'YDir','normal');
title('my STFT (400, 50)')
colorbar;
saveas(gcf, 'Q1d_5.png');

figure;
w = ones(1,400);
noverlap = 50;
Nfft = 512;
[s2, f2, t2] = spectrogram(x, w, noverlap, Nfft, fs);
imagesc(t2, f2, abs(s2)/fs);
set(gca,'YDir','normal');
title('matlab spectrogram (400, 50)')
colorbar;
saveas(gcf, 'Q1d_6.png');

% (e)
[x, fs] = audioread('handel.ogg');
figure;
w = ones(1,100);
noverlap = 50;
Nfft = 128;
[s2, f2, t2] = STFT(x, w, noverlap, Nfft, fs);
imagesc(t2, f2, abs(s2)/fs);
set(gca,'YDir','normal');
title('original spectrogram')
colorbar;
saveas(gcf, 'Q1e_1.png');


[x, fs] = audioread('handel_square.wav');
figure;
w = ones(1,100);
noverlap = 50;
Nfft = 128;
[s2, f2, t2] = STFT(x, w, noverlap, Nfft, fs);
imagesc(t2, f2, abs(s2)/fs);
```

```
set(gca,'YDir','normal');
title('square spectrogram')
colorbar;
saveas(gcf, 'Q1e_2.png');


[x, fs] = audioread('handel_4-bit_Quantization.wav');
figure;
w = ones(1,100);
noverlap = 50;
Nfft = 128;
[s2, f2, t2] = STFT(x, w, noverlap, Nfft, fs);
imagesc(t2, f2, abs(s2)/fs);
set(gca,'YDir','normal');
title('quantization spectrogram')
colorbar;
saveas(gcf, 'Q1e_3.png');


[x, fs] = audioread('handel_hardlimit_0.1.wav');
figure;
w = ones(1,100);
noverlap = 50;
Nfft = 128;
[s2, f2, t2] = STFT(x, w, noverlap, Nfft, fs);
imagesc(t2, f2, abs(s2)/fs);
set(gca,'YDir','normal');
title('hardlimit spectrogram')
colorbar;
saveas(gcf, 'Q1e_4.png');

[x, fs] = audioread('handel_modulation_1000.wav');
figure;
w = ones(1,100);
noverlap = 50;
Nfft = 128;
[s2, f2, t2] = STFT(x, w, noverlap, Nfft, fs);
imagesc(t2, f2, abs(s2)/fs);
set(gca,'YDir','normal');
```

```
title('modulation spectrogram')
colorbar;
saveas(gcf, 'Q1e_5.png');
clear all;

function [W, A] = CTFT_draw(x, fs, It)
    figure;
    subplot(3,1,1);
    plot(It, x);
    ylabel('Amp');
    xlabel('Time');

    [col, row] = size(x);
    y = fft(x, row);

    N = row;
    w = 2*pi*(0:(N-1)) / N;
    w2 = fftshift(w);
    w3 = unwrap(w2 - 2*pi);

    y = y / fs;
    W = w3/pi*fs/2;
    A = abs(fftshift(y));
    subplot(3,1,2);
    plot(w3/pi*fs/2, abs(fftshift(y)));
    title('CTFT Amplitude');
    xlabel('Freq (Hz)');
    ylabel('Amplitude');
    subplot(3,1,3);
    plot(w3/pi*fs/2, angle(fftshift(y)));
    title('CTFT Phase');
    xlabel('Freq (Hz)');
    ylabel('Phase');
end

function [W, A] = CTFT(x, fs, Nfft)
    y = fft(x, Nfft);
    w = 2*pi*(0:(Nfft-1)) / Nfft;
```

```
    w2 = fftshift(w);
    w3 = unwrap(w2 - 2*pi);

    y = y / fs;
    W = w3/pi*fs/2;
    A = fftshift(y);
end

function [S, f, t] = STFT(x, window, Noverlap, Nfft, fs)
    % If x is a signal of length Nx, then s has k columns, where
    % k = |(Nx – noverlap)/(length(window) – noverlap)| if window is a
vector.
    % If x is real and nfft is even, then s has (nfft/2 + 1) rows.
    % If x is real and nfft is odd, then s has (nfft + 1)/2 rows.
    % If x is complex, then s has nfft rows.
    assert(length(window) > Noverlap, 'size of window must be larger
than Noverlap')
    col_size = Nfft;
    row_size = floor( (length(x) - Noverlap) / (length(window) -
Noverlap) );
    S = zeros(col_size, row_size);
    t = zeros(1, row_size);
    from = 1;
    to = length(window);
    for i = 1 : row_size
        X = x(from:to);
        [f, A] = CTFT(X, fs, Nfft);
        S(1:end,i) = A(1:col_size);
        t(1, i) = ( from + floor(length(window) / 2) - 1)/ fs;
        from = from + (length(window) - Noverlap);
        to = to + (length(window) - Noverlap);
    end
    f = f(1:col_size);
end
```

2. Code for problem2 & 3(exclude 3(f))

```
%% problem 2
```

```
% assume amp is in [-1, 1]
[x, fs] = audioread('handel.ogg');
xmax = 1;
bit = 4;
level = 2^bit;
xt = quantizer_L_level(x, xmax, level)';

% (e) PCM
y = pcm_enc(xt, 4);
length_y = length(y)

% (f) huffman dict
delta = 2 * xmax / level;
symbols = [-(level-1)*delta/2:delta:(level-1)*delta/2];
p = histc(xt, symbols);
p = p / sum(p);
dict = huffman_dict(symbols, p);
dict2 = huffmandict(symbols, p);

% (g) huffman encode
y = huffmanenco(xt, dict);
length_y = length(y);
y1 = huffman_enc(xt, dict);
length_y = length(y1);
check_enc = sum(y ~= y1)           % 0 for correct

% (h) huffman decode
xd = huffmandeco(y, dict);
length_x = length(xd);
xd2 = huffman_dec(y, dict);
length_x = length(xd2);
check_dec = sum(xd ~= xd2)         % 0 for correct

% clear all;

function y = quantizer_L_level(x, xmax, level)
    delta = 2 * xmax / level;
```

```
    partition = [-xmax:delta:xmax];
    codebook = [0,-(level-1)*delta/2:delta:(level-1)*delta/2,0];
    [I, y] = quantiz(x,partition,codebook);
end

function y = pcm_enc(x, numBits)
    level = 2^numBits;
    delta = 2 / level;
    x = (x + (level-1)*delta/2) / delta + 1;
    y = reshape(de2bi(x)',1,[]);
end

function dict = huffman_dict(symbols, p)
    dict = cell(length(symbols), 2);
    for i = 1 : length(symbols)
        dict{i,1} = symbols(i);
    end
    % node is a structure array
    for i = 1 : length(symbols)
        s.array = [i];
        s.prob = p(i);
        node(i) = s;
    end
    nodeb = node
    while length(nodeb) > 1
        min = 1.1;
        minmin = 1.1;
        minp = -1;
        minminp = -1;
        for i = 1:length(nodeb)
            if nodeb(i).prob < minmin
                min = minmin;
                minp = minminp;
                minmin = nodeb(i).prob;
                minminp = i;
                continue
            end
            if nodeb(i).prob < min
```

```
                min = nodeb(i).prob;
                minp = i;
            end
        end
        if minp > minminp
            temp = minp;
            minp = minminp;
            minminp = temp;
        end
        for i = 1:length(nodeb(minp).array)
            dict{nodeb(minp).array(i),2} = [0
dict{nodeb(minp).array(i),2}];
        end
        for i = 1:length(nodeb(minminp).array)
            dict{nodeb(minminp).array(i),2} = [1
dict{nodeb(minminp).array(i),2}];
        end
        s.array = [nodeb(minp).array nodeb(minminp).array];
        s.prob = nodeb(minp).prob + nodeb(minminp).prob;
        nodeb = [nodeb(1:minp-1), nodeb(minp+1:minminp-1),
nodeb(minminp+1:end)];
        nodeb = [nodeb s];
        nodeb_length = length(nodeb);
    end
end

function y = huffman_enc(x, dict)
    C = cell(length(x),1);
    for i = 1 : length(x)
        C{i, 1} = dict{find( [dict{:,1}] == x(i)),2};
    end
    y = [C{:,1}]';
end

function x_dec = huffman_dec(y, dict)
    s.array = [1:length(dict)];
    s.level = 1;
    % 0 : left 1 : right
```

```
    s = split_tree(s, dict);
    st = s;
    x_dec = [];
    for i = 1 : length(y)
        if y(i) == 1
            st = st.right;
            if length(st.array) == 1
                x_dec = [x_dec; st.array(1)];
                st = s;
            end
        else
            st = st.left;
            if length(st.array) == 1
                x_dec = [x_dec; st.array(1)];
                st = s;
            end
        end
    end
    for i = 1 : length(x_dec)
        x_dec(i) = dict{x_dec(i),1};
    end
end

function st = split_tree(s, dict)
    if length(s.array) <= 1
        st = s;
        return
    end
    x.array = [];
    y.array = [];
    for i = 1 : length(s.array)
        if dict{s.array(i),2}(s.level) == 0
            x.array = [x.array s.array(i)];
        else
            y.array = [y.array s.array(i)];
        end
    end
    x.level = s.level + 1;
```

```
    y.level = s.level + 1;
    st.array = s.array;
    st.left = split_tree(x, dict);
    st.right = split_tree(y, dict);
end

% function x_dec = huffman_dec(y, dict)
%     x_dec = []
%     yt = y;
%     pos = 1;
%     dict2.symbol = dict(:,1);
%     dict2.code = dict(:,2);
%     i = 1
%     while( ~isempty(yt) )
%         temp = yt(pos);                                 % Get first
bit(char).
%         dictb = dict2;                                  % Get a backup
of the input dictionary.
%         while(1)
%             dictb = found_match( temp, pos, dictb);     % Get a sub
dictionary.
%             if ( length(dictb.code) ~= 1 )              % Until one
codeword left.
%                 pos = pos + 1;                          % Match second
bit(char).
%                 temp = yt(pos);                         % Get first char
%             else
%                 pos = 1;                                % Reset position.
%                 yt = yt(length(dictb.code{1})+1:end);   % Update the
input signal.
%                 break;
%             end
%         end
%         i = i + 1
%         % x_dec = [x_dec dictb.symbol]                  % Append char
to decoded signal.
%         % size(x_dec)
%     end
```

```
% end

% function dictb = found_match( code, pos, dict )
%     dictb.symbol={}; dictb.code={};                    % Create a
dictionary structure.
%     j = 1;                                            % Iterator.
%     for i = 1:length(dict.code)                         % For each code
in dictionary.
%         if ( dict.code{i}(pos) == code )                % If inpute code
matches
%             dictb.symbol(j) = dict.symbol(i);          % Get the symbol
that matches.
%             dictb.code(j) = dict.code(i);              % Get the code
of the matched symbol.
%             j = j + 1;                                % Prepare for a
next symbol.
%         end
%     end
% end
```

3. Code for 3(f)

```
%% problem 3(f)

% assume amp is in [-1, 1]
[x, fs] = audioread('handel.ogg');
[x1, fs] = audioread('handel_hardlimit_0.1.wav');
[x2, fs] = audioread('handel_modulation_100.wav');
[x3, fs] = audioread('handel_square.wav');
xmax = max(abs(x))+1e-4;
xmax1 = max(abs(x1))+1e-4;
xmax2 = max(abs(x2))+1e-4;
xmax3 = max(abs(x3))+1e-4;

bit = 4;
level = 2^bit;
xt = quantizer_L_level(x, xmax, level)';
xt1 = quantizer_L_level(x1, xmax1, level)';
xt2 = quantizer_L_level(x2, xmax2, level)';
xt3 = quantizer_L_level(x3, xmax3, level)';
```

```
% huffman dict
delta = 2 * xmax / level;
symbols = [-(level-1)*delta/2:delta:(level-1)*delta/2];
p = histc(xt, symbols);
p = p / sum(p);
dict = huffmandict(symbols, p);


delta = 2 * xmax1 / level;
symbols = [-(level-1)*delta/2:delta:(level-1)*delta/2];
p1 = histc(xt1, symbols);
p1 = p1 / sum(p1);
dict1 = huffmandict(symbols, p1);


delta = 2 * xmax2 / level;
symbols = [-(level-1)*delta/2:delta:(level-1)*delta/2];
p2 = histc(xt2, symbols);
p2 = p2 / sum(p2);
dict2 = huffmandict(symbols, p2);


delta = 2 * xmax3 / level;
symbols = [-(level-1)*delta/2:delta:(level-1)*delta/2];
p3 = histc(xt3, symbols);
p3 = p3 / sum(p3);
dict3 = huffmandict(symbols, p3);

% huffman encode
% original
y = huffmanenco(xt, dict);
length_y = length(y)
% hardlimit
y1 = huffmanenco(xt1, dict1);
length_y1 = length(y1)
% modulation
y2 = huffmanenco(xt2, dict2);
length_y2 = length(y2)
```

```
% square
y3 = huffmanenco(xt3, dict3);
length_y3 = length(y3)
% compactness
figure;
subplot(2,2,1);
histogram(abs(x));
title('Original')
subplot(2,2,2);
histogram(abs(x1));
title('Hardlimit')
subplot(2,2,3);
histogram(abs(x2));
title('Modulation')
subplot(2,2,4);
histogram(abs(x3));
title('Square')

% clear all;

function y = quantizer_L_level(x, xmax, level)
    delta = 2 * xmax / level;
    partition = [-xmax:delta:xmax];
    codebook = [0,-(level-1)*delta/2:delta:(level-1)*delta/2,0];
    [I, y] = quantiz(x,partition,codebook);
end
```

4. Code for problem4

```
%% problem 4

% 1 a 0.3
% 2 b 0.2
% 3 c 0.2
% 4 d 0.1
% 5 e 0.05
% 6 f 0.05
% 7 g 0.05
% 8 h 0.05
```

```
n = 100;
symbols = [1:8];
prob = [0.3 0.2 0.2 0.1 0.05 0.05 0.05 0.05];
dict = huffmandict(symbols, prob);
len = [];
for i = 1 : length(dict)
    len = [len length(dict{i,2}) ];
end

entropy = -sum(prob.*log2(prob))

avg_length = sum(prob.*len)

v = randsrc(n, 1, [symbols; prob]);
v_enc = huffmanenco(v, dict);
v_enc_length = length(v_enc)

sum = 0;
R = 1000;
for i = 1 : R
    v = randsrc(n, 1, [symbols; prob]);
    v_enc = huffmanenco(v, dict);
    sum = sum + length(v_enc);
end
avg_length = sum / R

avg_length_d100 = avg_length / n

n = 1000;
R = 10000;
for i = 1 : R
    v = randsrc(n, 1, [symbols; prob]);
    v_enc = huffmanenco(v, dict);
    sum = sum + length(v_enc);
end
avg_length = sum / R

avg_length_d1000 = avg_length / n
```

```
clear all;
```