

Lab 2: Source Coding: Turning Signals into Bits

B04901067 電機四 陳博彥

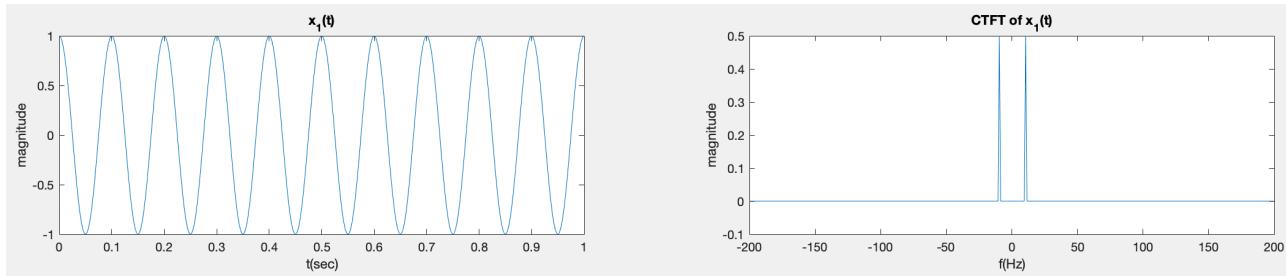
所有的程式碼都存放在 https://github.com/Andy19961017/Communication_System_Lab/tree/master/Lab2

第1(a)(b)題是用src_1ab.m寫的，1(c)(d)(e)題是用src_1cde.m寫的，依此類推。Function另外寫在各自的檔案中。

1.

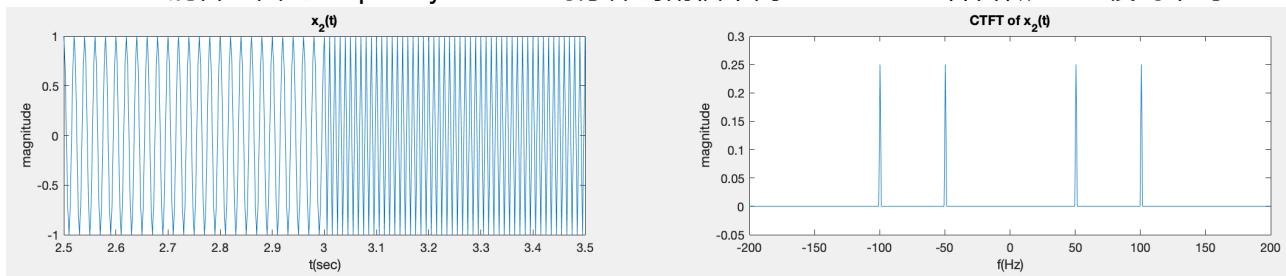
(a)

於在這段時間內，我們擷取的訊號就是 $\cos(2\pi \cdot 10 \cdot t)$ 。在time domain中，我們看到的是一個10Hz的cosine wave，而在frequency domain中，看到的是在正負10Hz的地方，各有一個高為0.5的delta function，與CTFT的理論值相符。



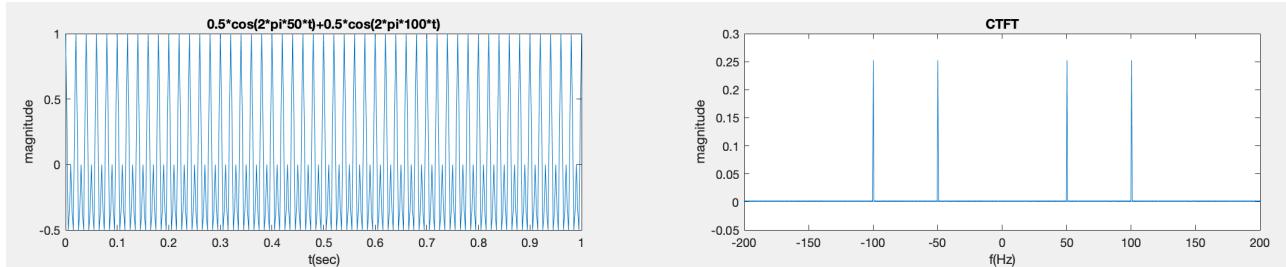
(b)

在time domain中，可看見在3秒的時候頻率有所轉變，在frequency domain中，結果為四根高0.25的delta function，位置分別在正負50與正負100Hz處，這恰好與 $0.5\cos(2\pi \cdot 50 \cdot t) + 0.5\cos(2\pi \cdot 100 \cdot t)$ 的CTFT一樣，我們可以把這個訊號解讀為兩個不同頻率的cosine wave混合，因此frequency domain的分佈為兩個不同cosine wave各自做CTFT後的平均。



(bonus)

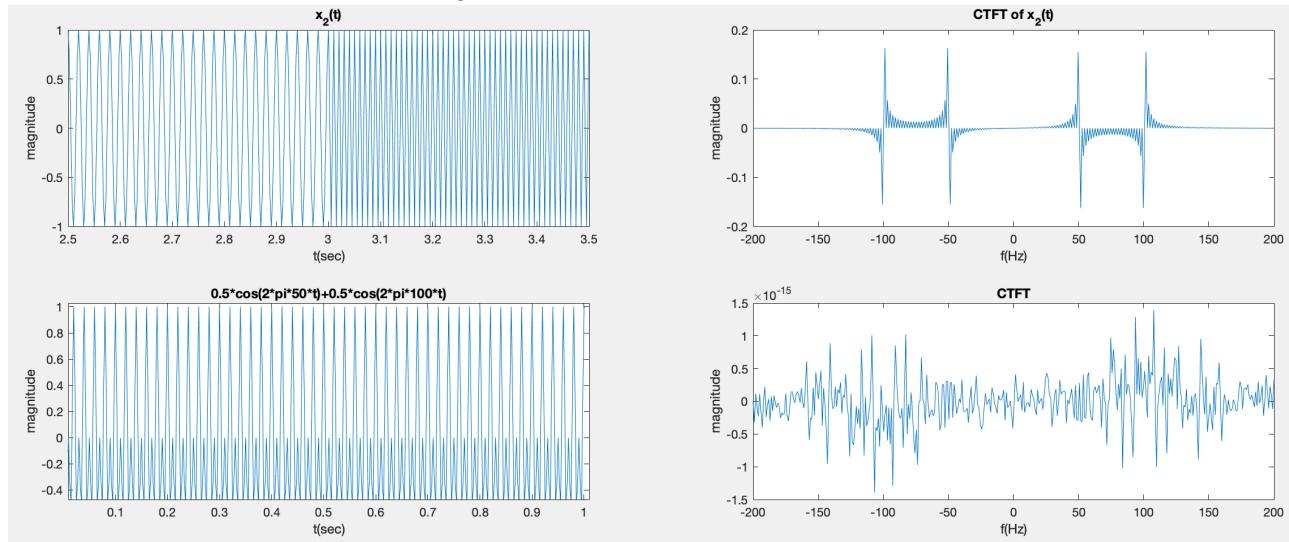
以下是 $0.5\cos(2\pi \cdot 50 \cdot t) + 0.5\cos(2\pi \cdot 100 \cdot t)$ 在time domain和frequency domain的作圖。可發現此信號與(b)中x₂(t)的CTFT一模一樣。



然而，CTFT pair應該是一對一對應的關係，兩個不同的訊號怎麼可能有一樣的CTFT呢？

我計算了一下理論值，發現，兩者CTFT的real part是一樣的，但imaginary part不一樣。
 $0.5\cos(2\pi \cdot 50 \cdot t) + 0.5\cos(2\pi \cdot 100 \cdot t)$ 的CTFT的imaginary part為零，而 $x_2(t)$ 的不為零。而在做上面的圖時，只有顯示real part，才會看起來一樣。

為了驗證，我改畫兩個CTFT的imaginary part，果真不一樣，如下圖：

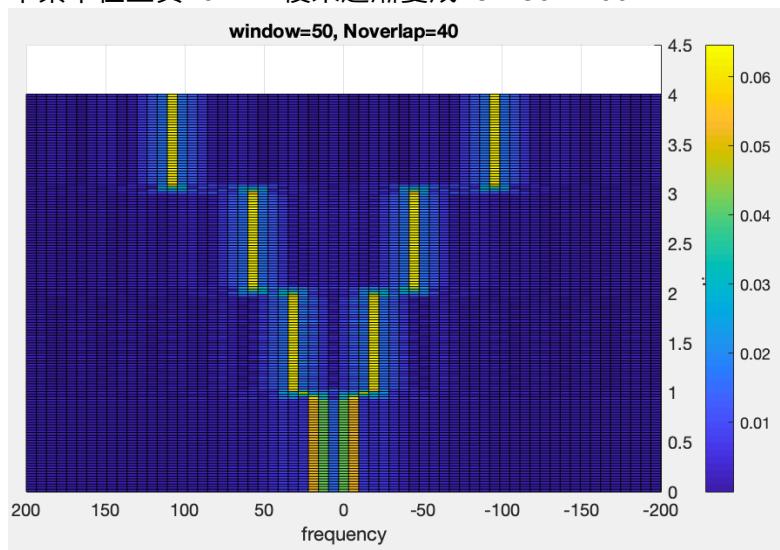


ps. $0.5\cos(2\pi \cdot 50 \cdot t) + 0.5\cos(2\pi \cdot 100 \cdot t)$ 的CTFT的imaginary part在圖中並不為零，但仔細一看，y軸上有乘一個 10^{-15} ，圖中的訊號只是電腦計算浮點數造成的誤差。

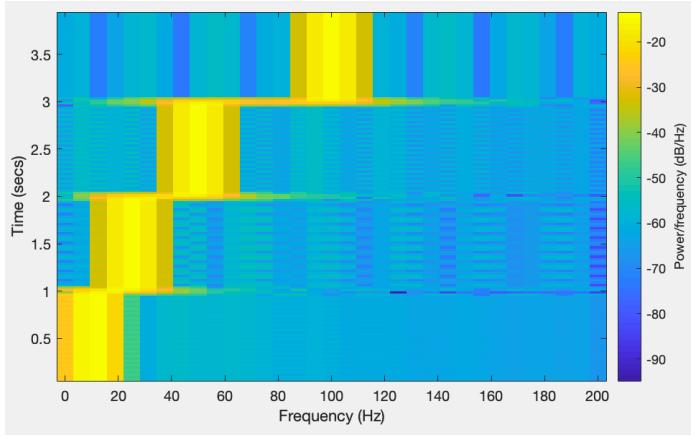
(c)

見STFT.m（在報告最後面付的github link中）。

將一開始定義的 $x(t)$ 丟到STFT中（ $y=STFT(x1, 50, 40, 64, 400)$ ）可得到下圖結果，可看見一開始頻率集中在正負10Hz，後來逐漸變成25、50、100Hz。

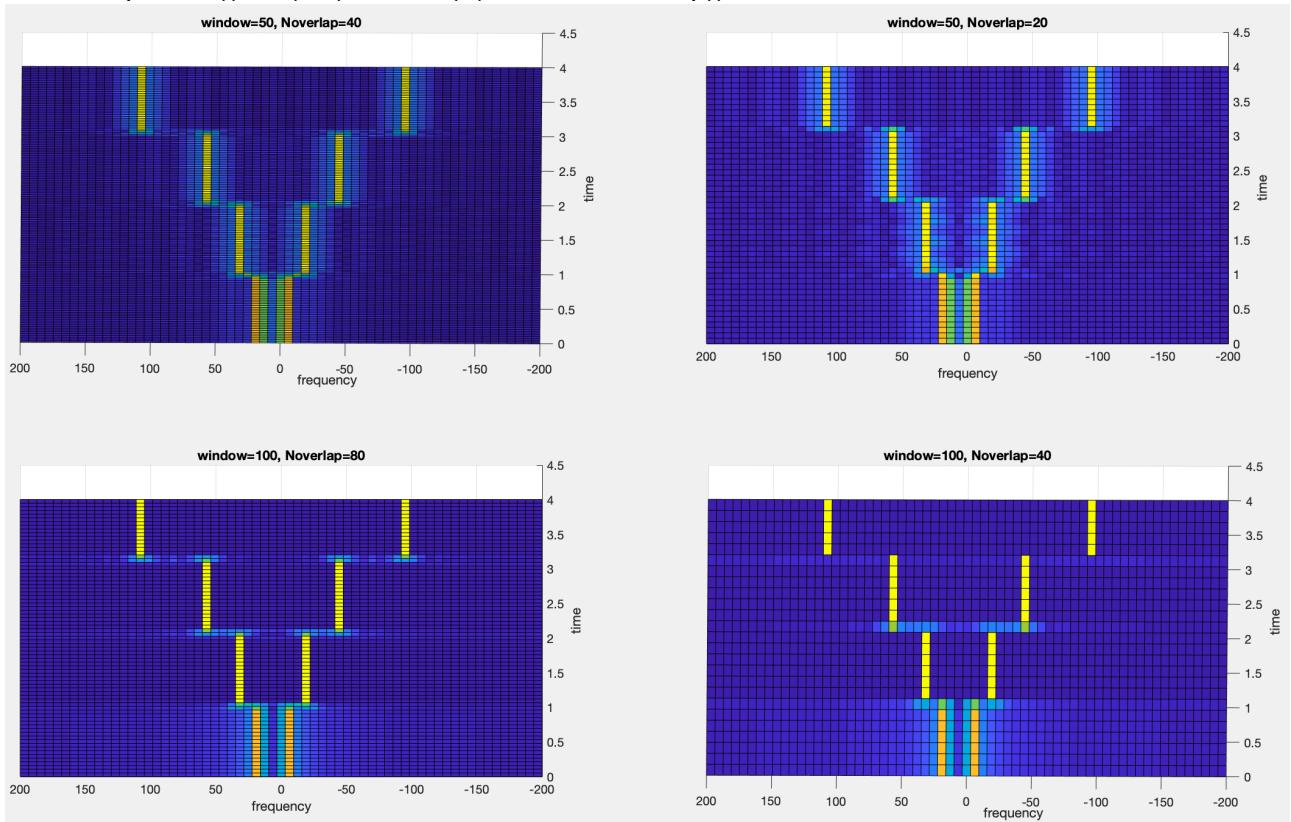


若用matlab內建函數spectrogram則得到下圖結果。spectrogram只顯示頻率為正的半邊，另一個差異在於z軸上的scale，我的STFT產生出的數值，最大值約在0.06左右，而spectrogram的約為20。理論上，input是震幅為1的cosine wave，對其做CTFT產生的delta function高度應該是0.5，這兩個函數所產生的數值為何是如此，我目前還沒弄懂原因。



(d)

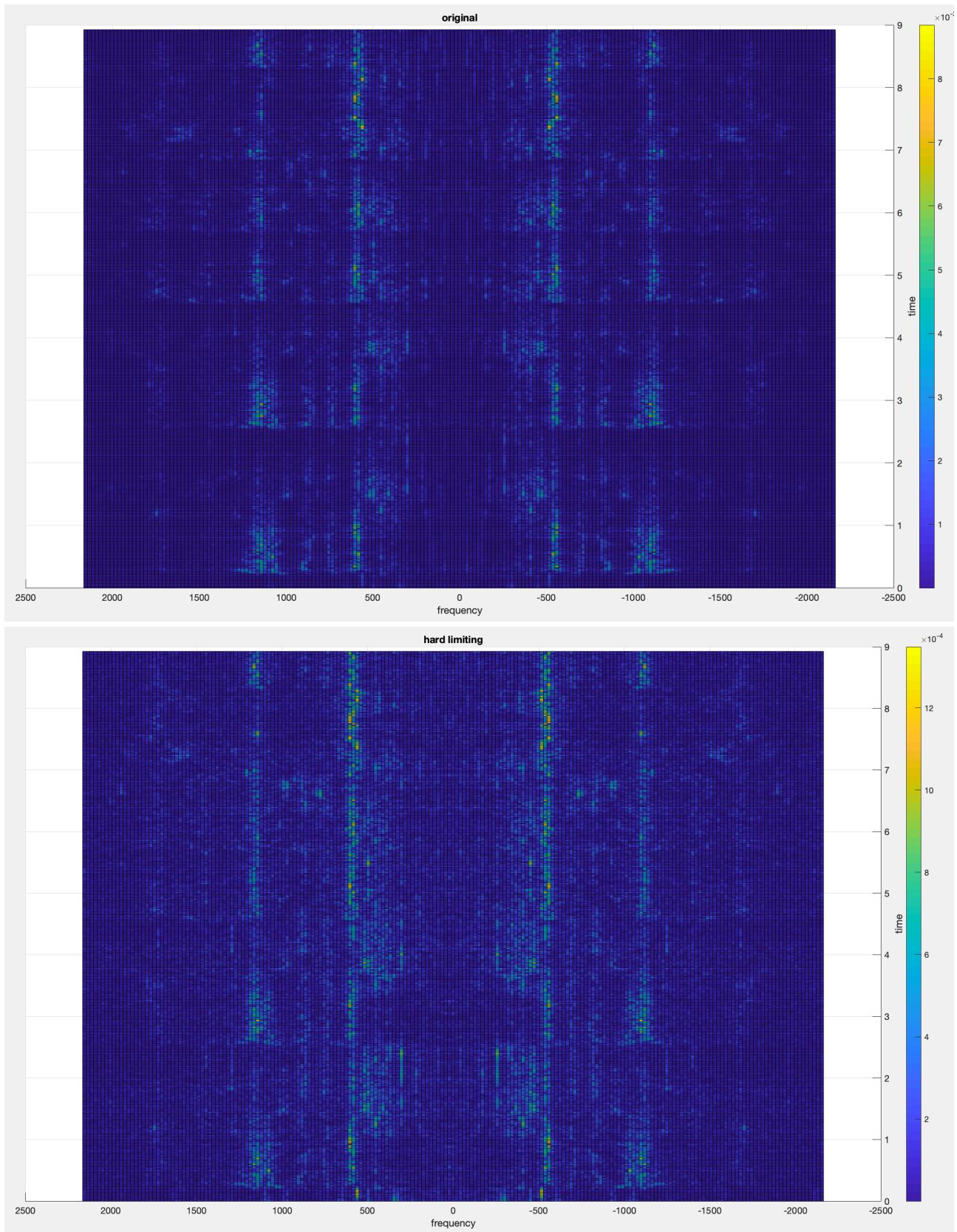
觀察以下四圖可發現，調整window跟Noverlap所產生的差異只有time軸上的解析度，window越小、Noverlap越大，則解析度越高。至於時間軸上timestep的數目為
 $N_{timeStep} = \text{floor}((\text{size}(x,2)-\text{window})/(\text{window}-\text{Noverlap}))+1$ 。

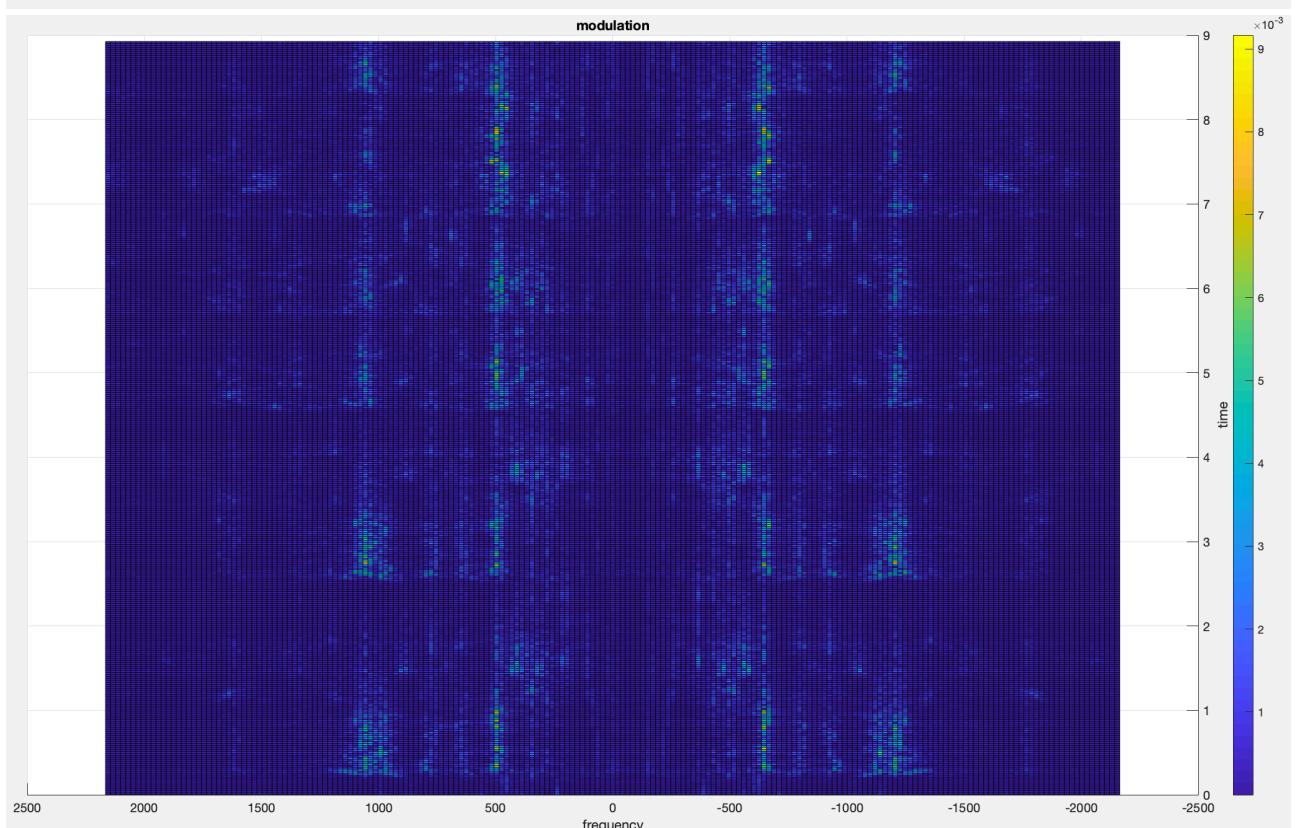
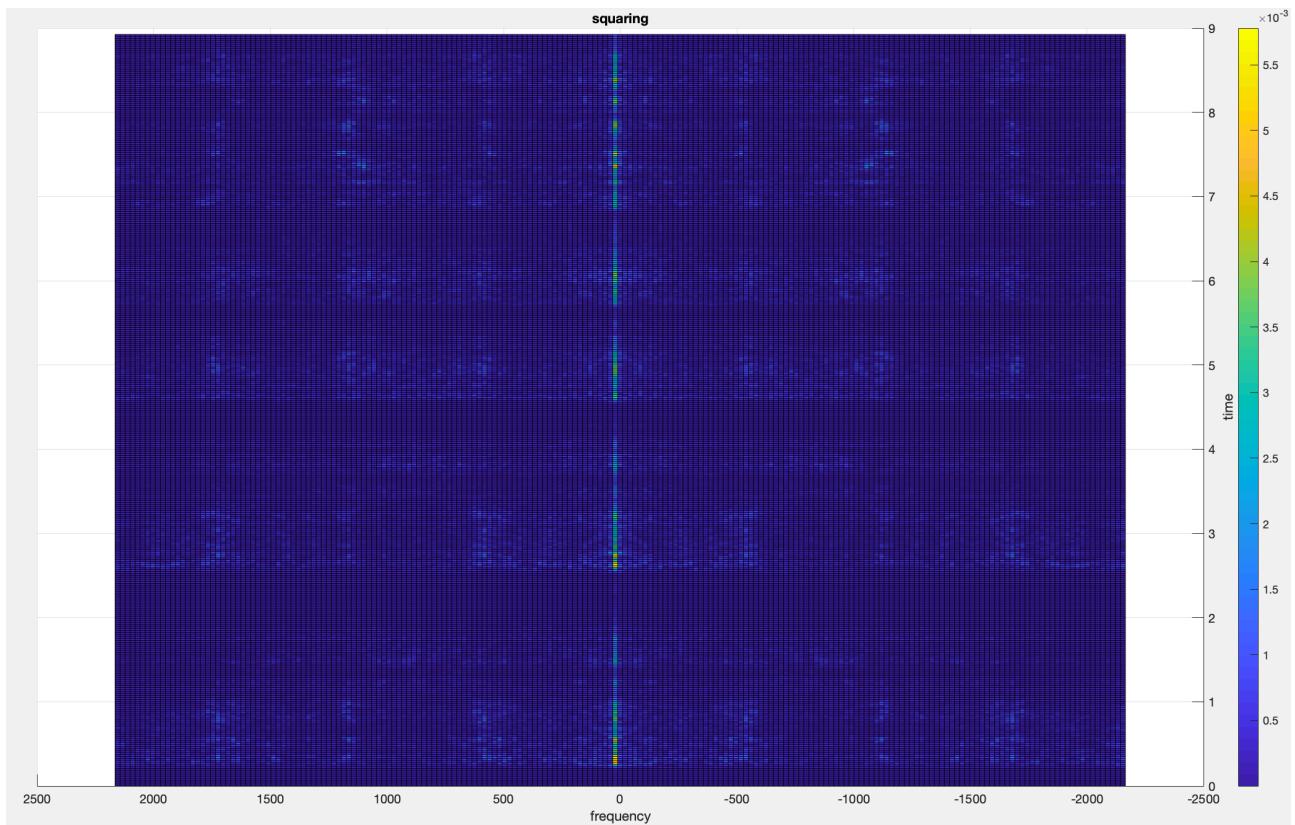


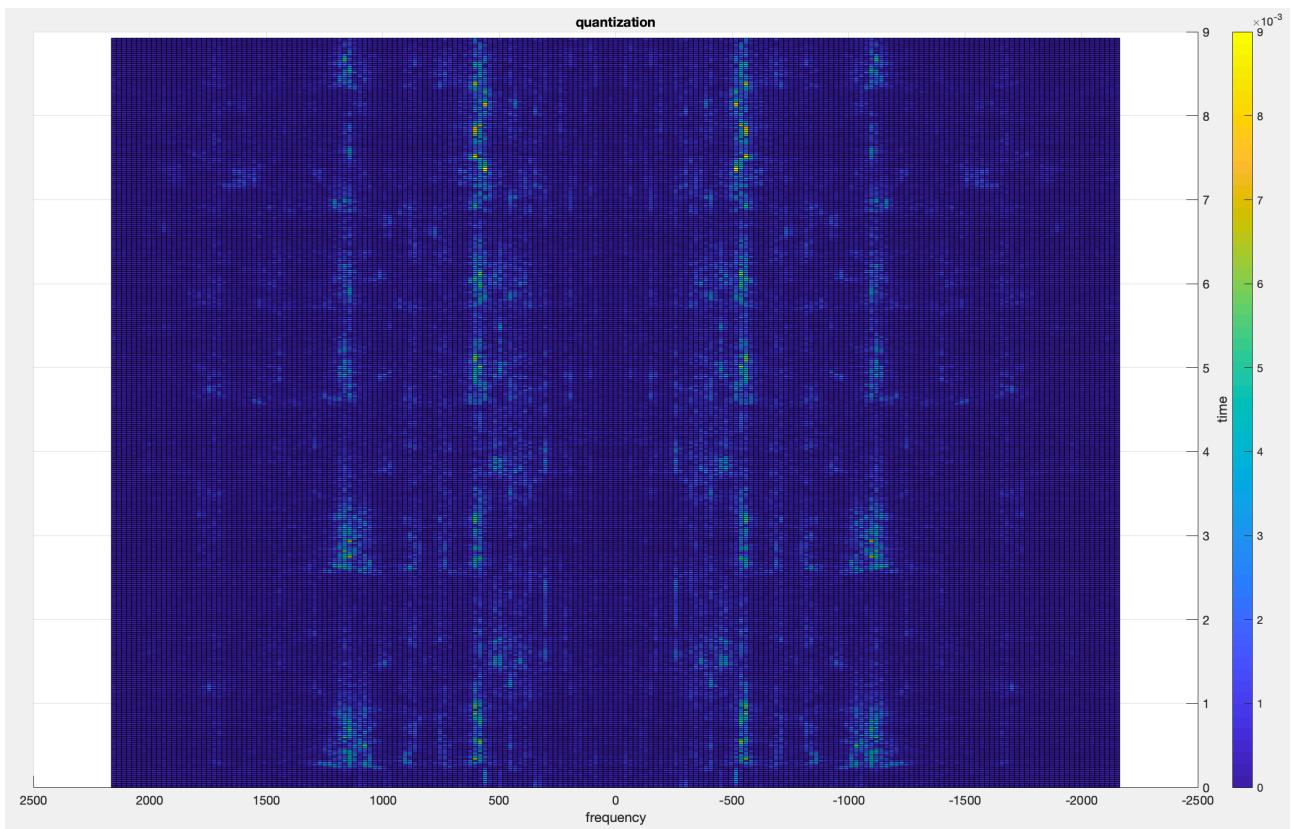
(e)

比較以下五張圖：

Hard limiting的spectrogram的頻譜比較「糊在一起」，相對的original比較有一陣一陣的樣子。
 Squaring的頻譜能量全部集中在低頻。
 至於其他的，用肉眼看來都跟original差不多。







2.

2.

(a)

Symbol	Code	Prob
a	00	0.300
b	10	0.210
c	11	0.211
d	011	0.1011
e	01010	0.0501010
f	01011	0.0501011
g	01000	0.0501000
h	01001	0.0501001

answer

(b) $d \ a \ b \ h \ c \Rightarrow \underbrace{011}_{d} \ \underbrace{00}_{a} \ \underbrace{10}_{b} \ \underbrace{01001}_{h} \ \underbrace{11}_{c} \quad 14 \text{ bits}$

(c) $a \Rightarrow 000, b \Rightarrow 001, c \Rightarrow 010, d \Rightarrow 011, e \Rightarrow 100, f \Rightarrow 101, g \Rightarrow 110, h \Rightarrow 111$

$d \ a \ b \ h \ c \Rightarrow \underbrace{011}_{d} \ \underbrace{000}_{a} \ \underbrace{001}_{b} \ \underbrace{111}_{h} \ \underbrace{010}_{c} \quad 15 \text{ bits}$

(d) Compare data size: (b) 要 14bits (c) 要 15bits, (b) 略好於(c)。

若考慮 bits per symbol 的期望值，(b) 是 $0.7 \times 2 + 0.1 \times 3 + 0.2 \times 5 = 2.7$ ，(c) 是 3，(b) 依然略好於(c)。

(e)(f)(g)(h)

function分別定義在quantizer_L_level.m、pcm_enc.m、huffman_dict.m、huffman_enc.m、huffman_dec.m中。

src_2efgh.m中有利用這些function做一些操作。

在src_2efgh.m的前半部，先產生一個 $x=\cos(t)$ ，其中 x 有20個sample，我先利用quantizer_L_level將 x quantize，再用pcm_enc將其encode成100 bits的向量 (numBits==2)。

在src_2efgh.m的後半部，我按照Table1定義的symbol還有機率，用huffman_dict產生一個dict，用huffman_enc encode ‘bca’這串資訊，在用huffman_dec decode回來。

3.

見src_3.m

此題中，我將numBits設為4。

(a)觀察可發現，越接近0的symbol越常出現。

(b)觀察可發現，越接近0的symbol的Huffman codeword length越短。

(c)

(d)

Data size of Huffman code of original signal (quantized): 1070890

Data size of PCM code of original signal (quantized): 1574344

由於numBits為4，PCM code長度剛好為原本sample數的4倍，Huffman code則壓縮了不少。

Huffman code在此題的壓縮效果還不錯，那是因為這個訊號大部分的數值點都在零附近，機率分布相當集中，entropy比較小，而Huffman code的壓縮效果會接近entropy。

(e)由於Huffman跟PCM都是無損的encode—decode，兩個recover的信號都跟原本的（經過quantization的）完全一樣，聽起來自然也一樣。

(f)

Data size of Huffman code of hard limiting signal (quantized): 393586

Data size of PCM code of hard limiting signal (quantized): 1574344

Data size of Huffman code of squaring signal (quantized): 435632

Data size of PCM code of squaring signal (quantized): 1574344

Data size of Huffman code of modulation signal (quantized): 888467

Data size of PCM code of modulation signal (quantized): 1574344

PCM一律是把每個symbol轉成4個bits，因此每個的data size都一樣。

至於Huffman code，我們可以發現他對hard limiting跟squaring的壓縮效果都特別顯著。hard limiting會把很多過大的數值都變一樣，squaring會把所有數值都變正的，而且也會往0附近集中（訊號數值在正負1之間），兩個操作都會讓訊號的entropy變小，因此Huffman code的壓縮效果才會特別顯著。

4.

(a)

$\text{entropy} = 0.3 \cdot \log(1/0.3) + 2 \cdot 0.2 \cdot \log(1/0.2) + 0.1 \cdot \log(1/0.1) + 4 \cdot 0.05 \cdot \log(1/0.05) = 2.646$ (bits),
where $\log(\cdot)$ have the base 2.

(b)

average codeword length of the Huffman code = $0.7 \cdot 2 + 0.1 \cdot 3 + 0.2 \cdot 5 = 2.7$ (bits)

(bonus)

average codeword length of Shannon code = $0.3 \cdot \text{roof}(\log(1/0.3)) + 2 \cdot 0.2 \cdot \text{roof}(\log(1/0.2)) + 0.1 \cdot \text{roof}(\log(1/0.1)) + 4 \cdot 0.05 \cdot \text{roof}(\log(1/0.05)) = 0.3 \cdot 2 + 2 \cdot 0.2 \cdot 3 + 0.1 \cdot 4 + 4 \cdot 0.05 \cdot 5 = 3.2$

(ps. Shannon code每個codeword的長度為 $l_i = \lceil -\log_2 p_i \rceil$)

資料壓縮的lower bound就是entropy，在此情況為2.646 bits/symbol，Huffman code是optimal的壓縮法，達到2.7 bits/symbol，Shannon code的平均長度是3.2 bits/symbol，甚至比PCM還差

(PCM==>3 bits/symbol)。

Shannon code的壓縮好壞隨著symbol的機率分佈會有所不同，當每個symbol的出現機率都是 $2^{(-n)}$ ，Shannon code會有最佳表現（此時平均長度恰等於entropy）。

(c)

length of encoded binary data : 260

(d)

average length of the binary data : 2.697740e+02

(e)

average codeword length : 2.697740e+00

可以看見，average codeword length十分接近期望值(2.7)，也比entropy (2.646) 大。由於Law of Large Number，這樣的實驗相當於執行了 $R \cdot n = 10^5$ 次抽樣，平均值才會很接近期望值。

(f)

average codeword length : 2.699955e+00

可以看見，average codeword length又更接近期望值(2.7)。這樣的實驗相當於執行了 $R \cdot n = 10^7$ 次抽樣，平均值才會如此接近期望值。