Project2
Spectrogram

程式撰寫與結果報告

姓名：蘇家駒
學號：410786004

一、程式碼撰寫與解說

(1) 本程式使用之struct

```
42  typedef struct WAVE_INFO{ // Used to record the sample, frequency, file name of that 8 cos waves
43      int sampleRate;
44      int freq;
45      char fn[3];
46  }WaveInfo;
47
48  typedef struct spectroSetting{ // Used to record analize window size, window type, DFT window size and frame interval
49      int anaWindowSize;
50      char windowType;
51      int DFTWindowSize; //N
52      int frameInterval; //M
53  }Setting;
54
55  typedef struct ImagineNumber{ // A struct to store the real and imagine part of imagine number
56      double real;
57      double imagine;
58  }ImgNum;
```

(2) cos波型產生

首先，依照題目的要求需要先產生8個cos波，其振福皆為10000，頻率為50Hz、55Hz、200Hz、220Hz，且對其各取8K與16K的取樣率，其程式碼如下，使用Project1的程式碼產生，一開始的WaveInfo型態的陣列cos8Info紀錄8個cos波所需的資訊，第一個為取樣率再來是頻率最後是表示頻率轉字串用以檔案產生時的黨名使用。

接下來的程式宣告一個Wave型態的變數用以當作產生8個cos波的模板，第一個for迴圈用以讀取cos8Info裡的設定值並使用makeWave去產生對應的header，然後算出總共的sample點數並用giveWaveDuration去產生可以儲存波型資料的陣列；第二個for迴圈用來算cos的值，之後的程式碼為儲存.wav用，檔名產生適是用makeFileName來針對各個WaveInfo製作對應的檔名。

```
273     // Store infomation of 8 cos waves
274     WaveInfo cos8Info[8] = {
275         {16000, 50, "050"},
276         {16000, 200, "200"},
277         {16000, 55, "055"},
278         {16000, 220, "220"},
279         {8000, 50, "050"},
280         {8000, 200, "200"},
281         {8000, 55, "055"},
282         {8000, 220, "220"}
283     };
284
285     //Generate 8 waves
286     Wave wave;
287     double x, sec = 1.0;
288     int A = 10000;
289     for(int i = 0; i < 8; i++){
290         int f = cos8Info[i].freq;
291         int sr = cos8Info[i].sampleRate;
292         wave = makeWave(cos8Info[i].sampleRate, 16);
293         int sampleNum = sec * sr;
294         giveWaveDuration(&wave, sec);
295         for(int j = 0; j < sampleNum; j++){
296             x = A * cos(2*PI*f*j/sampleNum);
297             waveAddASample(&wave, x);
298         }
299         // Store them by frequency and smaple rate
300         char fName[20] = {};
301         makeFileName(fName, cos8Info[i].fn, sr);
302         FILE *file;
303         file = fopen(fName, "wb");
304         fwrite(&(wave.header), sizeof(WaveHeader), 1, file);
305         fwrite((void*)(wave.data),sizeof(char), wave.size, file);
306         fclose(file);
307         free(wave.data);
308     }
```

```
162 void makeFileName(char tmp[20], char fn[3], int sr){ // To easily make file name of the 8 cos waves
163     if(sr == 16000){
164         sprintf(tmp, "cos_%sHz-16k.wav", fn);
165     }else{
166         sprintf(tmp, "cos_%sHz-8k.wav", fn);
167     }
168 }
```

（3）產生Spectrogram

Setting型態用以紀錄要對波做的DFT的參數，包括Analize Winddow Size、Analize Window Type、DFT Window Size、Frame Interval，這裡宣告一個該型態的陣列紀錄題目給的設定值，在使用for迴圈讀取使得做DFT更加直觀，最後在for迴圈裡使用makeSpectrogram去生成.txt檔，須包的的參數：檔名、Setting、取樣率、第幾個Setting。

```
308     // Setting for this project
309     Setting setting[4] = {
310         {5, 'r', 8, 5},
311         {5, 'm', 8, 5},
312         {20, 'r', 32, 10},
313         {20, 'm', 32, 10}
314     };
315 
316     //Make spectrogram
317     int N, P;
318     for(int i = 0; i < 4; i++){
319         N = setting[i].DFTWindowSize * 8000 / 1000;
320         P = setting[i].anaWindowSize * 8000 / 1000;
321         initCosNSinTable(N, N);
322         windowFunction(setting[i].windowType, 0, P, N);
323         makeSpectrogram("cos_050Hz-8k.wav", setting[i], 8000, i+1);
324         makeSpectrogram("cos_055Hz-8k.wav", setting[i], 8000, i+1);
325         makeSpectrogram("cos_200Hz-8k.wav", setting[i], 8000, i+1);
326         makeSpectrogram("cos_220Hz-8k.wav", setting[i], 8000, i+1);
327         makeSpectrogram("vowel-8k.wav", setting[i], 8000, i+1);
328         free(cosTable);
329         free(sinTable);
330         free(window);
331 
332         N = setting[i].DFTWindowSize * 16000 / 1000;
333         P = setting[i].anaWindowSize * 16000 / 1000;
334         initCosNSinTable(N, N);
335         windowFunction(setting[i].windowType, 0, P, N);
336         makeSpectrogram("cos_200Hz-16k.wav", setting[i], 16000, i+1);
337         makeSpectrogram("cos_050Hz-16k.wav", setting[i], 16000, i+1);
338         makeSpectrogram("cos_055Hz-16k.wav", setting[i], 16000, i+1);
339         makeSpectrogram("cos_220Hz-16k.wav", setting[i], 16000, i+1);
340         makeSpectrogram("vowel-16k.wav", setting[i], 16000, i+1);
341         free(cosTable);
342         free(sinTable);
343         free(window);
344     }
345 
346     printf("finish");
347 }
```

其中的initCosNSin與windowFunction是用於計算在不同DFT Window Size、取樣率、Analize Window Size下其DFT算式中展開的尤拉表示式中的cos和sin的對應值與Window時對應的cos值，該作法可以明顯降低程式的執行時間，因為不用每次做轉換時都算一次cos和sin值，而用查表的方式可以大幅減少運算時間。

下圖表示其製作表的程式碼

```
11    double **cosTable;
12    double **sinTable;
13    double *window;
```

```
148  void initCosNSinTable(int K, int N){
149      // Because call cos, sin function any interate is too slow,
150      // so I make two Table to record correspond cos and sin value by k and n
151      // It make the proformance better
152      cosTable = (double **) malloc(K * sizeof(void *));
153      sinTable = (double **) malloc(K * sizeof(void *));
154      for(int i = 0; i < N; i++){
155          cosTable[i] = (double *) malloc(N * sizeof(double));
156          sinTable[i] = (double *) malloc(N * sizeof(double));
157      }          (int)0
158      for(int k = 0; k < K; k++){
159          for(int n = 0; n < N; n++){
160              cosTable[k][n] = cos((2*PI*k*n)/N);
161              sinTable[k][n] = sin((2*PI*k*n)/N);
162          }
163      }
164  }
```

```
167  void windowFunction(char type, int lBound, int hBound, int N){
168      //'r' for rectangular, 'm' for Hamming, 'n' for Hanning
169      // Make window be a table to reduce the times to execute cos function
170      window = (double *) malloc(N * sizeof(double));
171      for(int  i = 0; i < hBound; i++){
172          if((i >= lBound) && (i < hBound)){
173              if(type == 'r'){
174                  window[i] = 1.0;
175              }else if(type == 'm'){
176                  window[i] = 0.54 - 0.46*cos((2*PI*i)/(hBound-1));
177              }else if(type == 'n'){
178                  window[i] = 0.5-0.5*cos((2*PI*i)/(hBound-1));
179              }
180          }else{
181              window[i] = 0.0;
182          }
183      }
184  }
```

(4) getFileSize

開啟檔案後用fseek將讀取指標移到最後在使用ftell取得讀取指標共移動多少並回傳，由於指標開頭的位移是0，而尾巴則為該檔案的bytes數，因此用該方法所求之值即為檔案大小。

```c
int getFileSize(char fileName[]){ // To calculate the number of bytes of file, below will use this function
    FILE *file;
    file = fopen(fileName, "rb");
    // Use fseek to point out the end of file so that can tell how many bytes that the file has
    fseek(file, 0, SEEK_END);
    int size = ftell(file);
    printf("%d", size);
    return size;
}
```

（5）makeSpectrogram

首先將給定的檔名打開並將讀取值標移到44，因為前44是header，而我們要的是data，size是算總共有多少個點，透過之前寫的getFileSize可以取得檔案大小，除以2在減掉22即為所有資料點數，宣告data陣列儲存讀取的資料，用fread一次讀2bytes，讀取長度為size個，儲存進data，最後執行DFT Function。

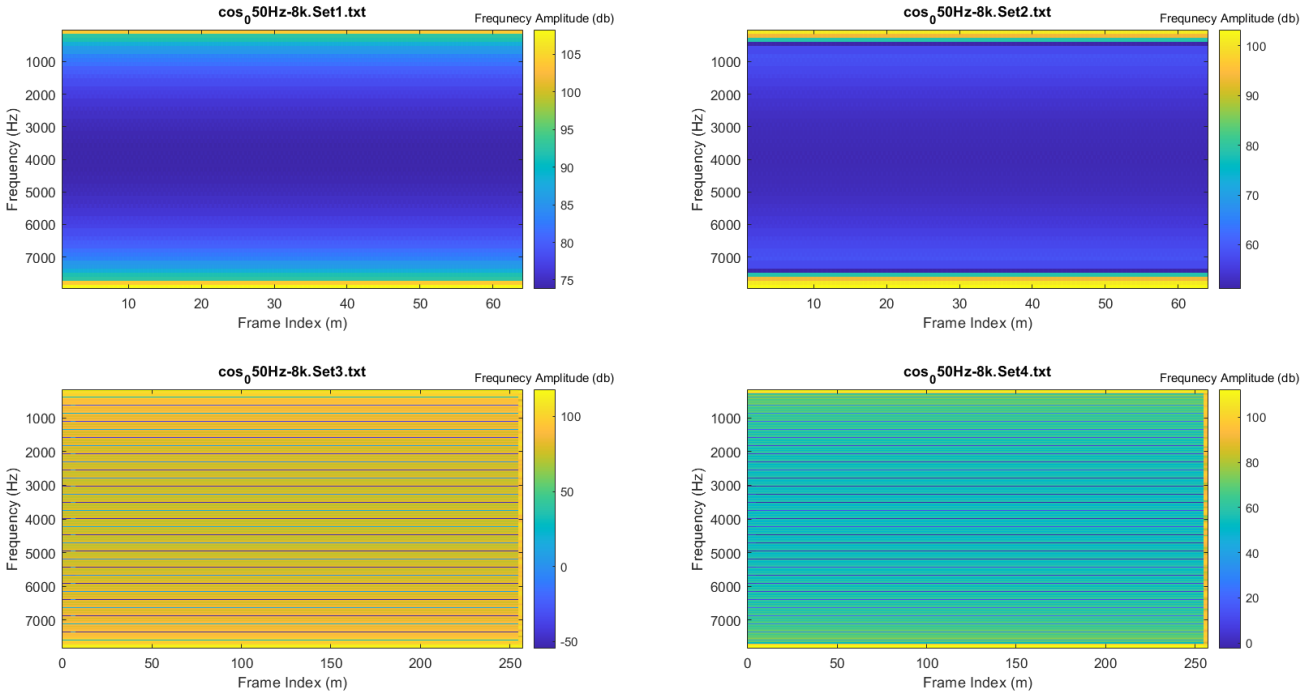
```c
void makeSpectrogram(char fileName[], Setting setting, int sampleRate, int setIndex){ //Combine read file and DFT
    FILE *file;
    file = fopen(fileName, "rb");
    fseek(file, 44 , SEEK_CUR); // Move file reading index to 44 aka begin of wave data

    int size = getFileSize(fileName) / 2 - 22; // Calculate total file points
    short int data[size];
    fread(data, sizeof(short int), size, file); // Read data and store them in short int type array 'data'
    fclose(file);

    DFT(setting, sampleRate, data, size, setIndex, fileName);
}
```

最後做DFT，該Function程式碼如下，先將題目給定的各種設定由毫秒轉成點並算出共有幾個音框需要轉換。接著生成記錄轉出來的值的檔案的檔名並打開該黨準備寫入。最後做傅立葉轉換，第一個for迴圈決定轉換哪個音框，第二個決定要算哪個頻率，第三個則是執行DFT的公式，全部算除來後取exponential的絕對值後取分貝值並存進檔案。DFT公式裡的Zero Padding程式碼如下，cos&sin的查表為提升程式速度很重要的一部份。

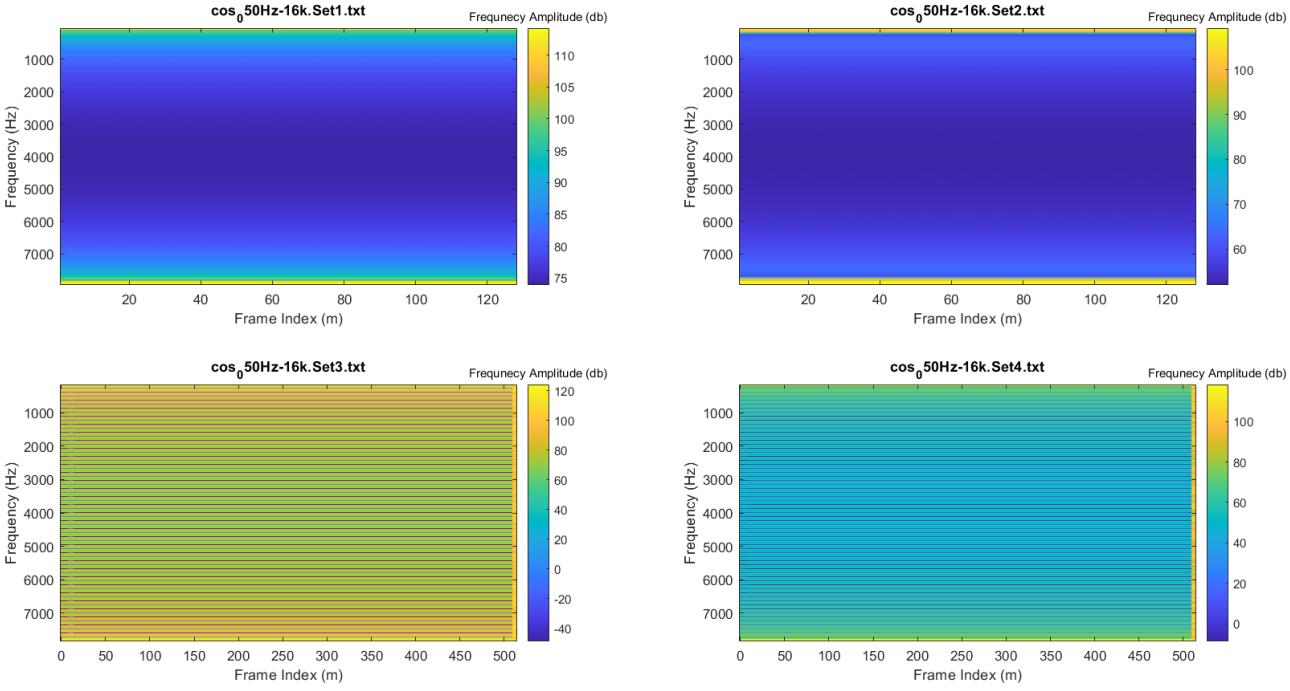
```
200  void DFT(Setting setting, int sampleRate, short int x[], int size, int setIndex, char fileName[]){
201      int N, M, P,frameNum;
202      //Transform below window or interval from ms to point
203      N = setting.DFTWindowSize * sampleRate / 1000;
204      M = setting.frameInterval * sampleRate / 1000;
205      P = setting.anaWindowSize * sampleRate / 1000;
206
207      //Calculate how many frame need to transform
208      if(size % M == 0){
209          frameNum = size / M;
210      }else{
211          frameNum = size / M + 1;
212      }
213
214      double res = 0;
215
216      //Generate spectrogram file(.txt) name.
217      char buf[30] = {'\0'};
218      strncpy(buf, fileName, strlen(fileName)-3);
219      sprintf(buf, "%s{Set%d}.txt", buf, setIndex);
220
221      //Open file.
222      FILE *fp;
223      fp = fopen(buf, "w");
224
225      int remain = 0;
226
227      printf("Transforming %s.\n", buf);
```

```
232      // DFT, first, need to decide which frame to transform
233      // Second, give which frequency bin to calculate its db
234      // Third, implement the Fourier transform formula
235      ImgNum sum = {0, 0};
236      for(int m = 0; m < frameNum; m++){
237
238          for(int k = 0; k < N; k++){
239              sum.real = 0;
240              sum.imagine = 0;
241              for(int n = 0; n < N; n++){
242                  // Transform exponential to Euler form
243                  sum.real += x[m*M+n] * window[n] * cosTable[k][n];
244                  sum.imagine += x[m*M+n] * window[n] * sinTable[k][n] * -1;
245              }
246              res = sqrt(pow(sum.real, 2) + pow(sum.imagine, 2)); // Get absolute value of imagine number
247              res = 20 * log10(res); // Calculate db value
248              fprintf(fp,"%lf ", res); // Write in txt file
249          }
250          fprintf(fp, "\n");
251      }
252      fclose(fp);
253  }
```

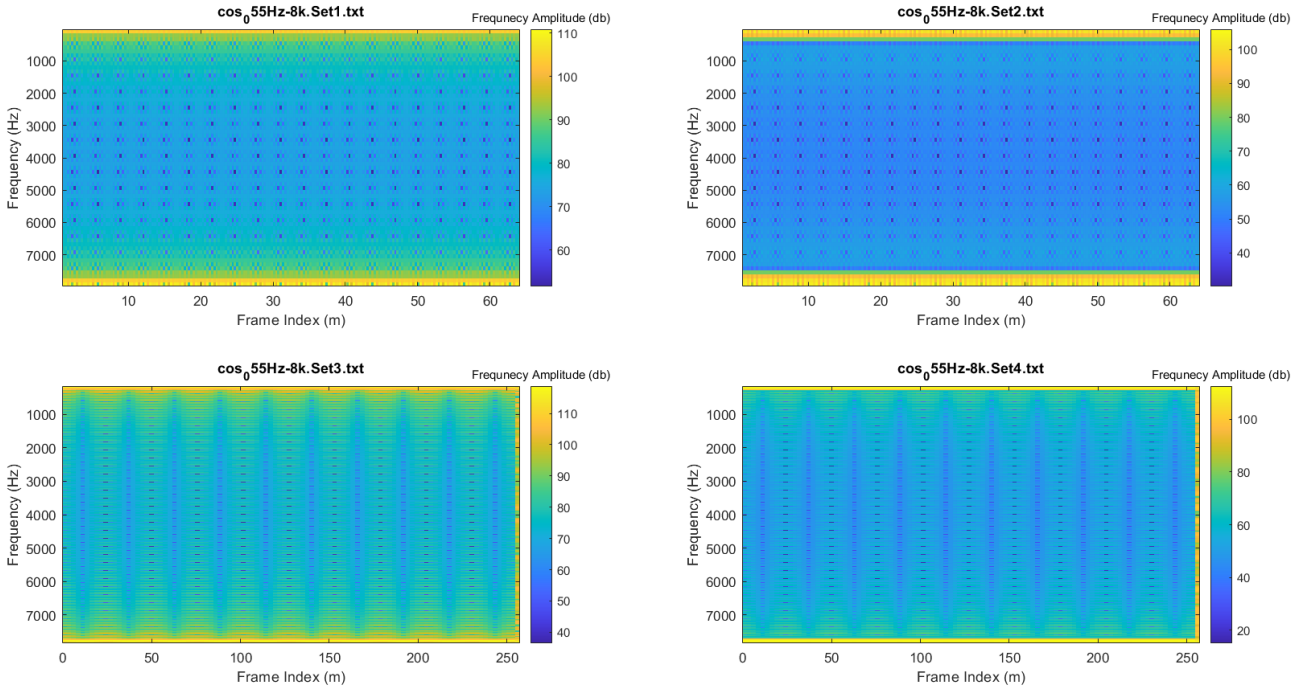
二、程式執行結果



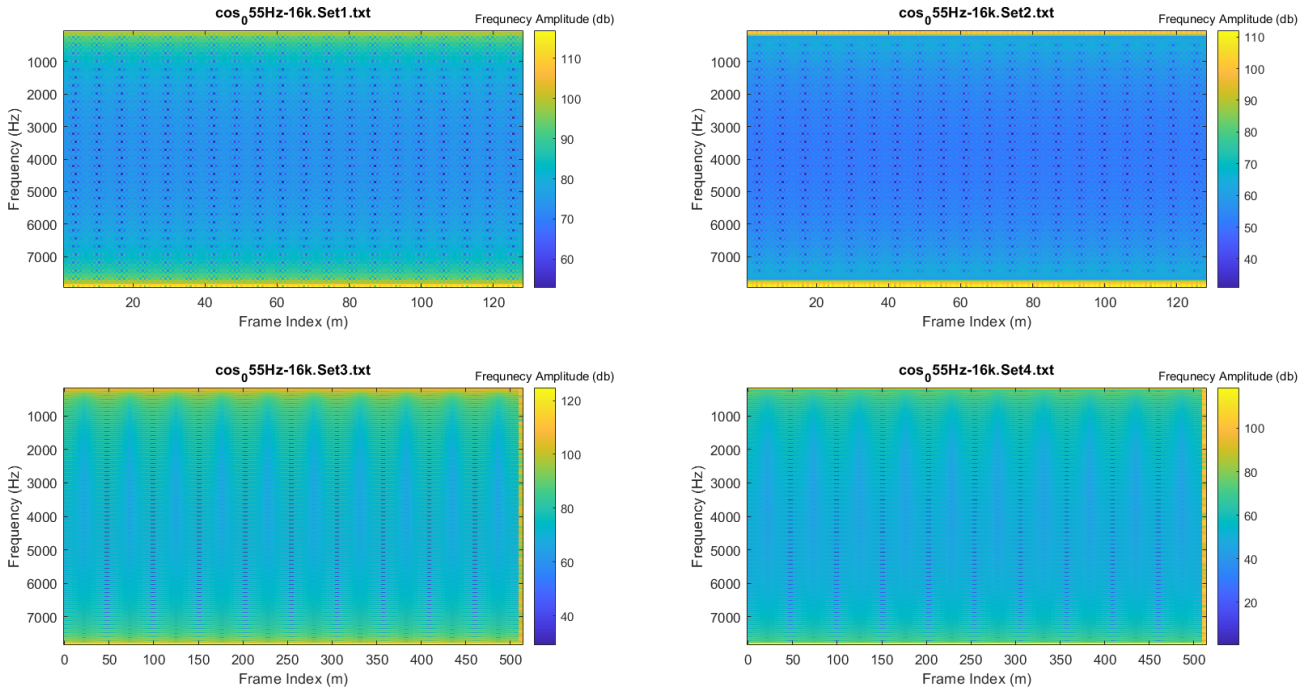
圖一 cos_50hz-8k



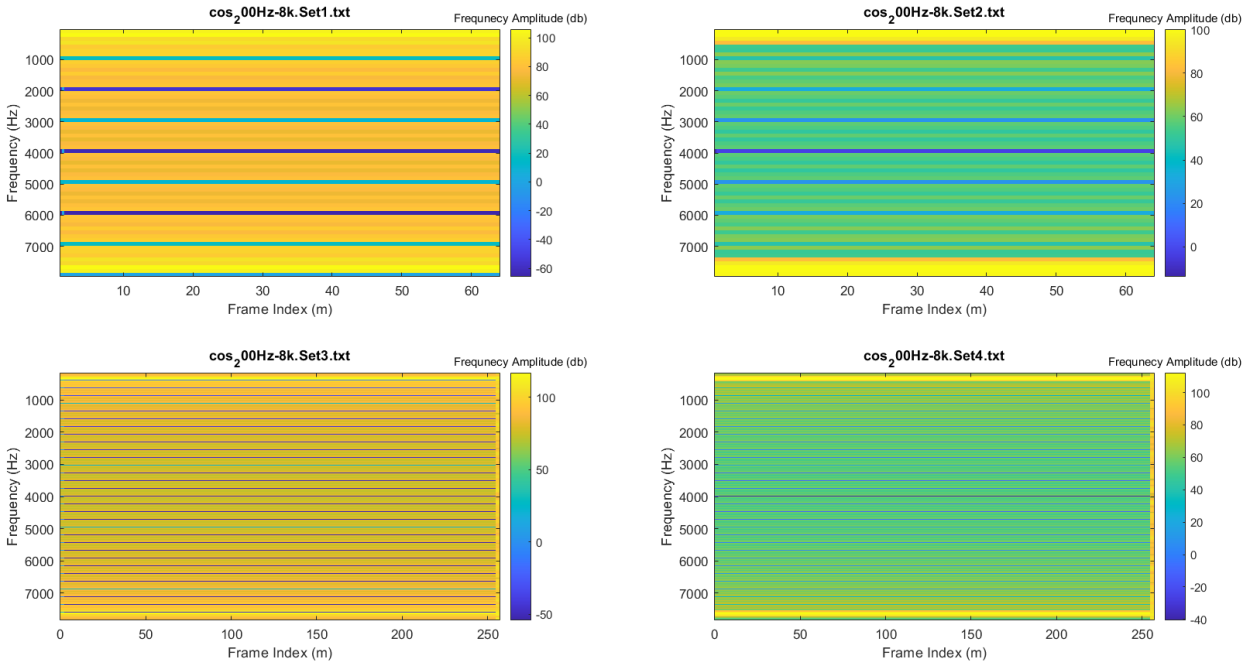
圖二 cos_50hz-16k



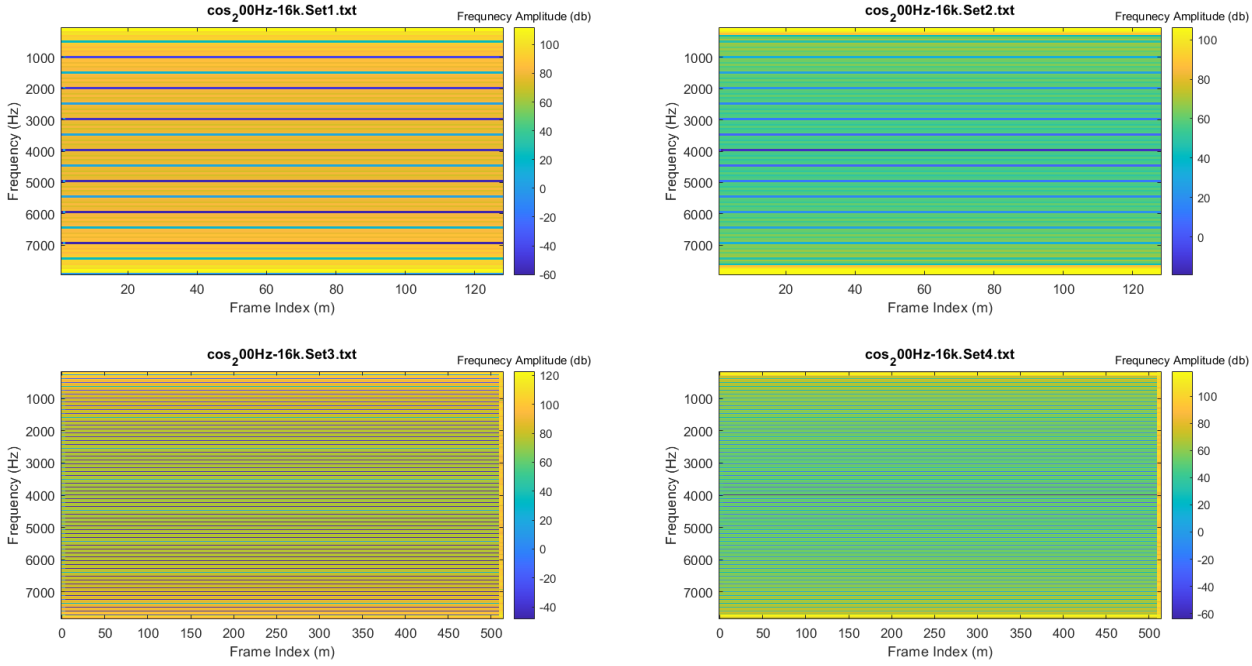
圖三 cos_55hz-8k



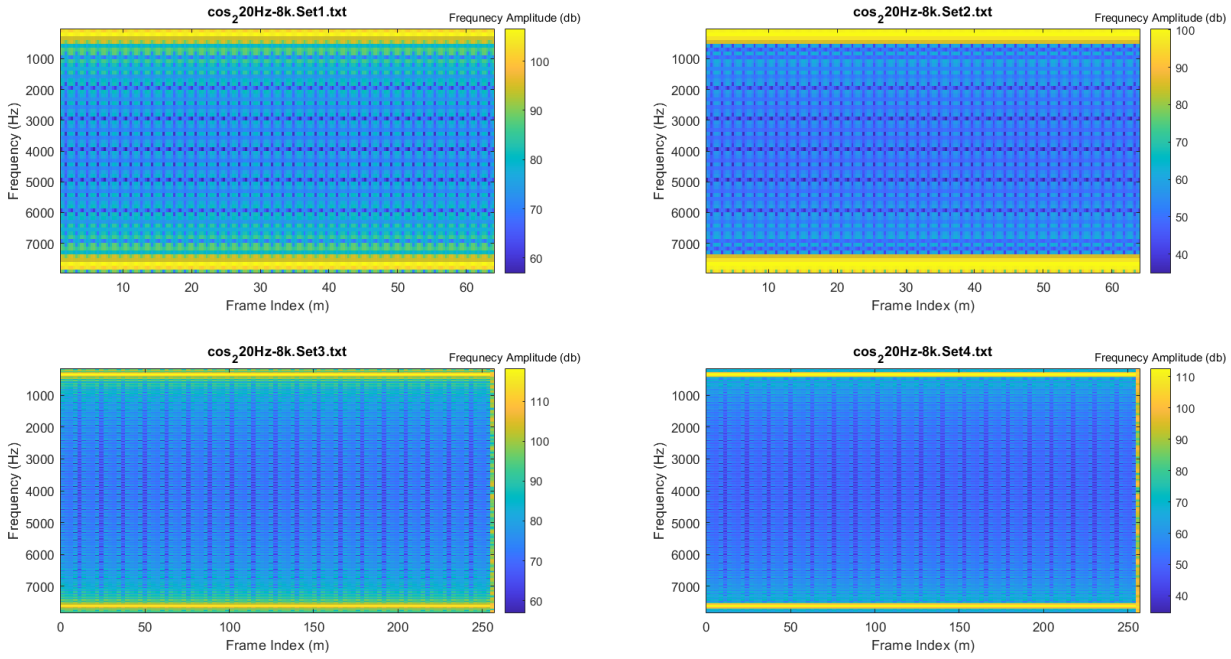
圖四 cos_55hz-16k



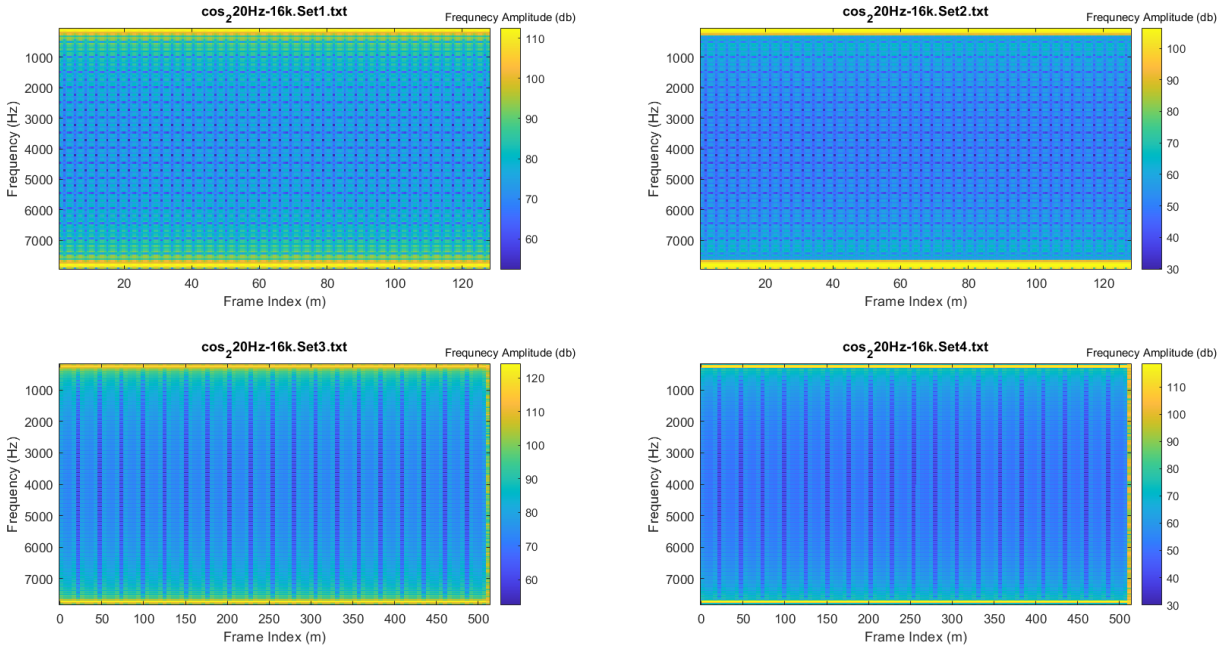
圖五 cos_200hz-8k



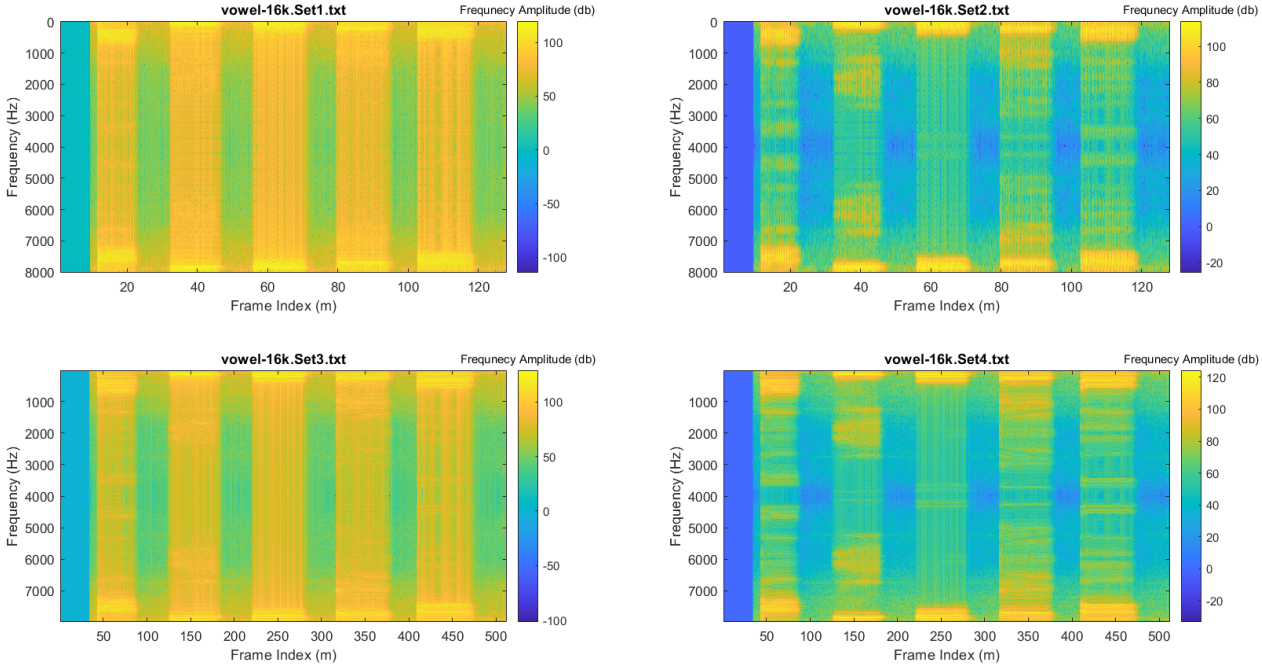
圖六 cos_200hz-16k



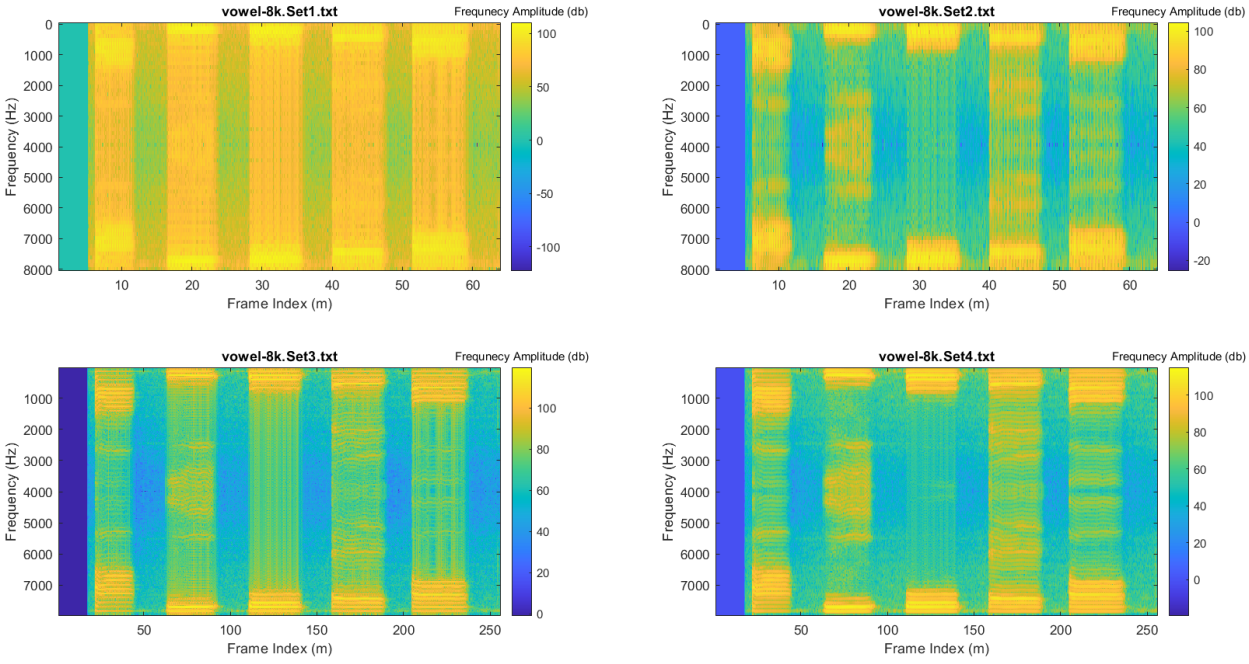
圖七 cos_220hz-8k



圖八 cos_220hz-16k



圖九 vowel-16k



圖十 vowel-8k

音框長度

執行後發現越長的音框分析出來的頻率越準確，以題目給的8ms與32ms，分析出來的頻譜以較長音框可更清楚顯示頻率成分，用單頻cos波可以很明顯的看出來，32ms的成分與非其成分有很明顯的分界，而8ms的分界則有點模糊，以220Hz來做說明，在其成分頻率處有非常明顯的線。

使用的Window Function

比較使用Rectangular和Hamming，使用Rectangular會參雜較多非其成分，以單頻cos波來看，中間原本應為深藍色的部分再使用Rectangular時靠近主要頻率的頻率會微微變淺，以自己錄的音檔觀察更為明顯，在沒有說話的音框會參雜很多不應該有的頻率成分，而使用Hamming時，這一情況明顯降低；在有說話的音框，其成分在使用Hamming也更為清晰。

音檔的取樣率

在同樣的頻率之下，不同的取樣率分析出來的頻譜解析度也不同，在本次作業中16k音檔分析出來的頻譜解析高於8k音檔分析出來的解析。

分析後遇到的問題

執行完程式後得到的頻譜圖中有很多圖會變成很多線，推測是因為使用Window Function進行Zero Padding之後，使得原本訊號的頻率成分在做DFT時不會被分析到導致出來的圖產生很多條線，而證明此種現象，可以觀察使用200Hz與220Hz時，200Hz會產生此種錯誤而220Hz不會。

錄音轉出來的頻譜由於前面沒有說話而沒有訊號因此轉換時該處的能量很低以至於乘是在計算時無法計算趨近於0的數字而導致溢位，以至於圖片一開始呈現一片色塊。

三、程式複雜度

以1秒的音檔來說

Set1執行(8k)：乘法 4134400 次、加法 3289600 次
Set1執行(16k)：乘法 16460800 次、加法 13132800 次

Set2執行(8k)：乘法 4134400 次、加法 3289600 次
Set2執行(16k)：乘法 16460800 次、加法 13132800 次

Set3執行(8k)：乘法 32844800 次、加法 26240000 次
Set3執行(16k)：乘法 131225600 次、加法 104908800 次

Set4執行(8k)：乘法 32844800 次、加法 26240000 次
Set4執行(16k)：乘法 131225600 次、加法 104908800次