DSP Final Project Team 16

鋼琴單音之分析與粗略的去除雜訊

Member: 曾雋卿

**摘要**

對於鋼琴的一些單音，觀察其DFT，找出一些特徵，再以這些特徵為基礎，試著實作出一套幫單音去除雜訊的方法。

**工具**

用來分析的音樂檔，是我用手機，把我彈奏的鋼琴單音錄製下來，所得到的。錄製了2種格式: .aac 和 .flac。

使用的軟體: Matlab以及相關toolbox。

**分析內容**

說明:

我對於錄音檔案的命名方式

第一個英文字母是音名，從左數到右的2個數字--

1st number:音高 0:跟中央C在一起的； 1:比中央的再高8度

2nd number:當時錄製的順序，從0開始排

(例如: d0-0指的是中央C隔壁那個D，而且是我第1個錄的該音)

下文對於各個音的稱呼，也大致同上(例如: D0指的是中央C隔壁那個D)

錄音檔涵蓋的範圍:從中央C，到F1

DFT的部分我只關注magnitude，而選擇不去看phase. 下文中，任何關於DFT domain的圖，如未特別標註，都是畫magnitude

Time domain的部分，我關注兩點:

(1) summation of array, which I think should be 0 for an ideal, noise-free wave.

(2) 肉眼能觀察並比較的平滑度

一些初步的結論

**I) 捨棄使用.flac檔**

.flac在高頻與DC的地方，有著不一定存在於.aac檔，且特徵明顯的雜訊。因此，我決定採用.aac作為其餘分析與討論的對象。

|  |  |
| --- | --- |
|  |  |
|  |  |
| 以下為用於比較之.aac | |
|  |  |
|  |  |

**II) 泛音、脈波、和傅立葉級數**

觀察time domain，能發現有些檔案有相當明顯的脈波，而其他的，有些是較不明顯，有些則是根本不像脈波。

|  |  |
| --- | --- |
|  |  |
|  |  |

從DFT domain來看:

|  |  |
| --- | --- |
|  |  |
|  |  |

透過比較兩邊的圖，我認為:時域的脈波，是頻率上的泛音的結果；

而泛音又來自於傅立葉展式:如果一個訊號有週期(如聲波)，又不是單純的sinusoidal，那麼，by Fourier series，就能展成無限多項的sinusoidal。不過現實中，因為後面的項係數太小，所以只會看到前幾項。

III)

凡本次專題中分析到的鋼琴單音，皆有泛音(至少有第一泛音)，只是泛音的數量多寡，和泛音的強弱分布，各自不同。例如: C1(比中央C高1個八度的C)的泛音，量多而明顯，甚至有第一泛音強過基音的情形發生。

|  |  |  |
| --- | --- | --- |
|  |  |  |

IV)

Either fundamental frequency or the 1st overtone has the largest magnitude. 最高能量可能出現在第一泛音的音，都是那些擁有較多泛音，泛音能量也較強的音，像是C1、G0、D0。

基於以上4點，我想出了一個粗略的，幫鋼琴單音的錄音檔去除雜訊的方法。概念:先在頻譜上找出基音與其泛音，之後就可以把其他部分通通砍掉了。

Algorithm: denoise based on property of overtone.

Let's call it "overtone-based denoise". Assumption:

1. noise do not overflow signal everywhere

2. (two times fundamental frequency) < N/2, where N is the total # of points

3. Highest peak is at either fundamental frequency or the 1st overtone

Conceptual Code: (I won't call it pseudo code since it actually does not look like one)

**procedure** OBD(F) # input F is an array of DFT

A = magnitude of F;

N = length of A;

L = ;

Mark = array of zeroes with length N;

ave = sum(A) / N;

count = 5;

Find = 0;

# if after 5 search, no suitable peak is found: exit and report exception

while ((there is no 1 in Mark) && (count > 0)):

{

count--;

# circular-even -> only have to do left hand side

p = searchPeak(A, Mark, [1, L]);

# in reality, overtones are not precisely at integer multiples

range = interval[p-10, p+10];

# check this peak is 1st overtone or fundamental frequency

pf = searchPeak(A, Mark, 0.5\*range);

if A(pf) > sqrt(ave \* A(p)):

# that peak is the 1st overtone

range = interval[pf-10, pf+10];

if 3\*p < L: # see if we can search on

# those whose peak at 1st overtone usually has many overtones

# so we should be able to get its 2nd overtone

p2 = searchPeak(A, Mark, 3\*range);

if A(p2) > sqrt(A(p) \* ave): # still, use amplitude at p to check

# mark fundamental frequency & all overtones

for (i = 1; pf \* i < L; i++) :

Mark(searchPeak(A, Mark, i\*range));

Find = 1;

ff = pf; # fundamental frequency

else: # this is not funamental fequency

Mark(pf) = -1; # disqualify this peak

Mark(p) = -1;

else: # if we can't, then guess we find it.

Mark(pf) = 1;

Mark(p) = 1;

Find = 1;

ff = pf;

else: # the peak is at the fundamental frequency

# check if its 1st overtone exists

p1 = searchPeak(A, Mark, 2\*range);

if A(p1) >= sqrt(ave \* A(p)): # if so

if 3\*p < N/2:

p2 = searchPeak(A, 3\*range);

if A(p2) > sqrt(A(p1) \* ave):

for (i = 1; p \* i < L; i++) :

Mark(searchPeak(A, i\*range));

Find = 1;

ff = p;

else:

Mark(p) = -1;

else:

Mark(p) = 1;

Mark(p1) = 1;

Find = 1;

ff = p;

else: # this is not funamental fequency

Mark(p) = -1; # disqualify this peak

}

if (Find == 0):

output("No legal fundamental frequency found.");

return;

# Find all corresponding peaks on the other side

# In matlab, due to the way matlab index (1 to length(A)),

# circular-even is A[k] = A[length +2 - k]

for (i = 2; i <= L; i++)

if (Mark(i) == 1)

Mark(N + 2 - i) = 1;

keep = array of 0 with size N;

For all peaks P:

eps = findSpread(P, A, ff);

for (i = P - eps; i <= P + eps; i++):

keep(i) = 1;

For i = 1 to N:

if (keep(i) ! = 1)

F(i) = 0;

output("Procedure complete.")

return;

**end**

**procedure** searchPeak(A, Mark, [init, end])

# only for array with all elements nonnegative

# suppose no 2 elements have same value

max = 0; # value

maxI = -1; # index

for (i = init; i < end; i++):

if (Mark(i) != -1): # only if it is not disqualified

if (A(i) > max):

max = A(i);

maxI = i;

return maxI;

**end**

**procedure** findSpread(p, A, ff)

# each peak is at least ff's indices away from each other

ave = geometric mean of all points in A;

totalA = sum(A( : - ave);

# area under curve centered at peak with average deducted

eps = 1; # epsilon, the range of spread

AP = 0; # area we have so far with spread of epsilon

for (eps = 1; AP < 0.94 \* totalA; eps++):

# find spread by comparing the area ratio

AP = sum(A(p-eps : p+eps) - ave);

return eps;

**end**