數位濾波器

Github 網址(內有程式碼):

https://github.com/ChristopherChen070535/Digital-Filter

撰寫人:

國立台北大學通訊工程學系三年級陳品鈞

- **專題介紹**
  - imple_filter.c :主要功能包括讀取和儲存 WAV 檔案(此程式的輸入檔為 blue_giant_fragment.wav)、將左聲道通過通濾波器和並將右聲道通過阻波器，進行摺積濾波，最後計算並保存濾波後信號的離散傅立葉變換結果。程式透過命令行參數接收濾波器的階數(M)、檔案名稱等設定，並輸出處理後的 WAV 檔案以及傅立葉變換和濾波器係數的結果。
    以下為此次專題主要探討 M 的三種情況 :
    ./simple_filter 1024 hL1024.txt hR1024.txt YL1024.txt YR1024.txt blue_giant_fragment.wav output1024.wav
    ./simple_filter 32 hL32.txt hR32.txt YL32.txt YR32.txt blue_giant_fragment.wav output32.wav
    ./simple_filter 8 hL8.txt hR8.txt YL8.txt YR8.txt blue_giant_fragment.wav output8.wav
  - show_data.py : 此程式利用 Matplotlib 庫，對來自.txt 的脈衝響應和對數頻譜數據進行視覺化。首先，我定義了一個函數，可以從文本文件讀取數據，並以莖圖形式繪製脈衝響應，然後將這些脈衝響應保存為相應的 PNG 文件。接著，我再定義了另外兩個函數，用於讀取頻譜數據，並以對數形式繪製頻譜。
  - run.sh : 用來將上述程式一鍵執行以得到結果
- **成果討論**
  1. M 的意義: 濾波器的長度 (M)是一個代表濾波器次數的長度，可以由 window method 求出，決定了濾波器的複雜度和頻率響應的銳利度。在 simple_filter.c 中，M 影響了 band_pass 函數和 high_pass、

low_pass 函數的計算以及左聲道和右聲道的濾波器係數。其特性為較大的 M 通常會產生較複雜的濾波器，其頻率響應可能更銳利，但也可能導致更多的計算和運算成本。

2. impulse responses 的繪製以及改變 M 對其之影響: M 對於頻率響應之影響:較大的 M 通常導致頻率響應更為銳利，故濾波器在特定頻率範圍內的通過或阻擋特性更加明確，我們可以從下圖 1-6 觀察，我發現如果 M 越大的頻率響應的 sample index 部分越多且越密集，這也就證實了前面的理論。

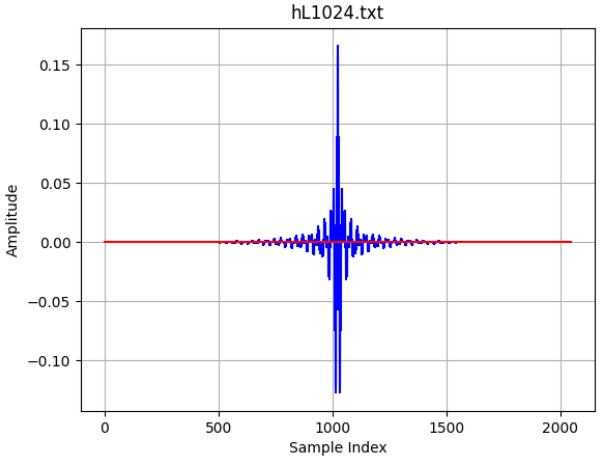

圖 1:M 為 1024 左聲道之脈衝響應

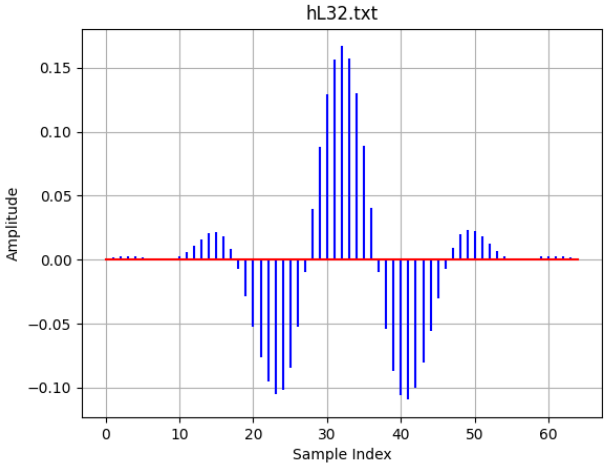

圖 2:M 為 32 左聲道之脈衝響應

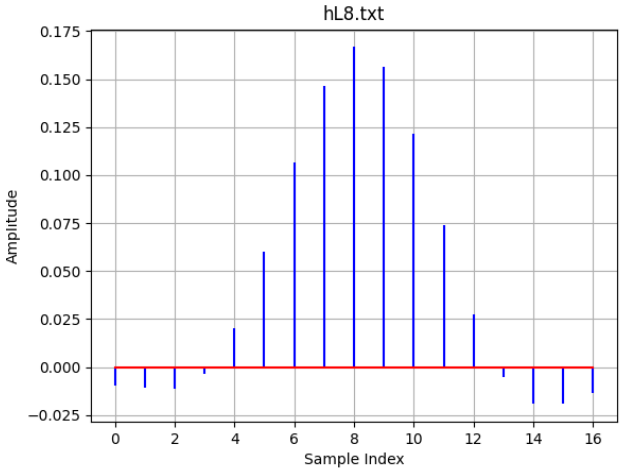

圖 3:M 為 8 左聲道之脈衝響應

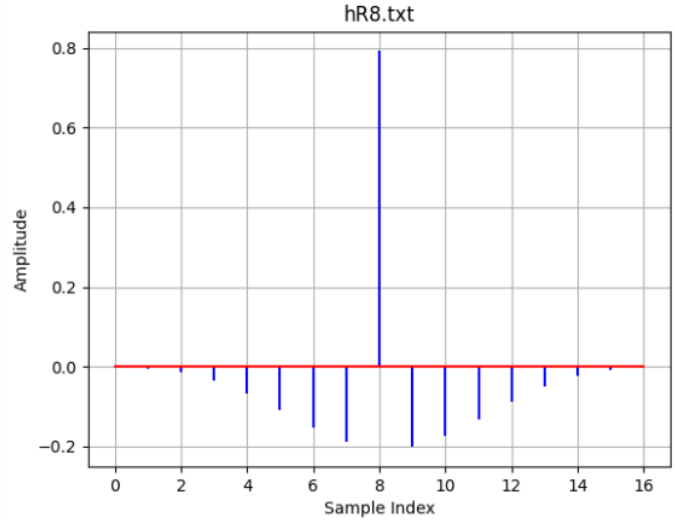

圖 4:M 為 8 右聲道之脈衝響應

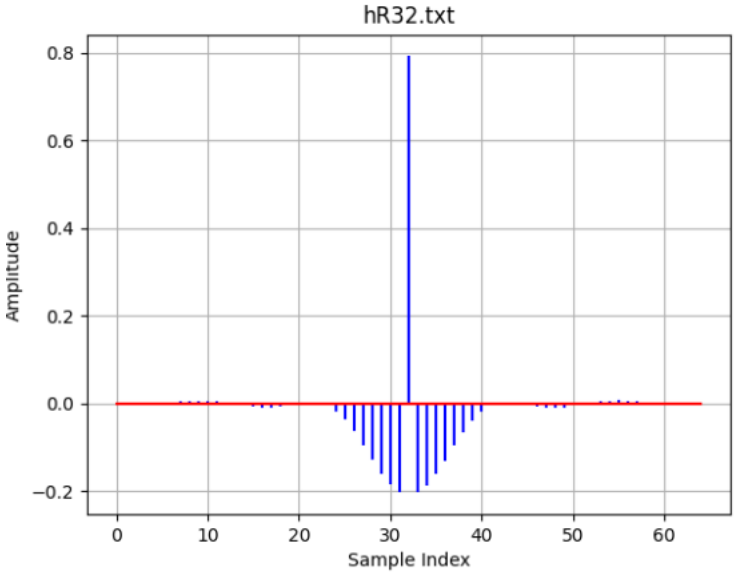

圖 5:M 為 32 右聲道之脈衝響應

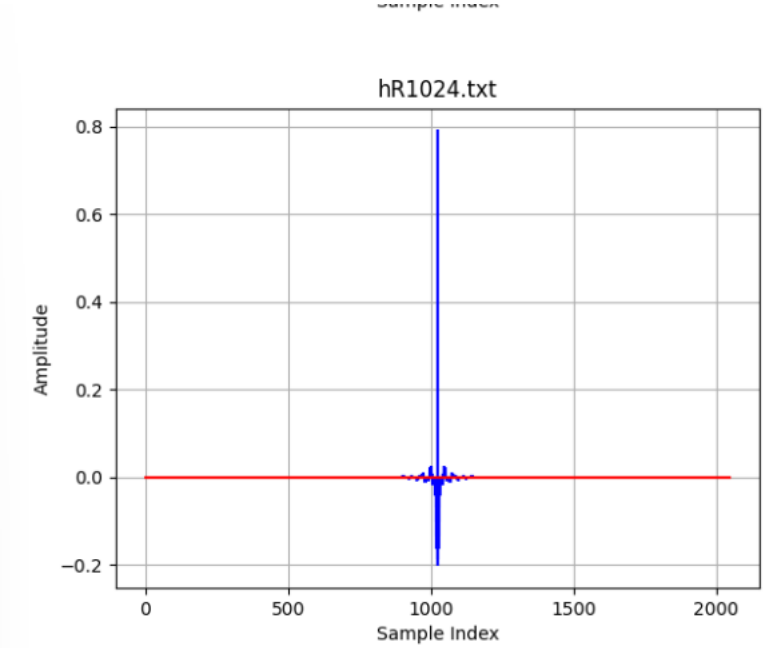

圖 6:M 為 1024 右聲道之脈衝響應

3. log spectrum 的繪製以及改變 M 對其之影響:

在這個程式中，M 影響了 log spectrum 的計算，我們對 convolution 後的資料進行傅立葉轉換，再將每個頻率成分的振幅轉換為對數（log）後再乘以 20，而 M 又會根據上述所示影響到濾波器的寬度，所以 M 也會因此影響到 log spectrum，所以從下結果圖 7-9 中可以發現當 M 愈大時，呈現出來的濾波效果愈接近理想的 bandpass 以及 bandstop 濾波器(如下圖 10 濾波器示意圖)

在觀察 log spectrum 時，可以發現對數操作可以將幅度的大範圍變化轉換為較小的範圍，使得小信號的變化更容易觀察，因此和頻譜圖(如下圖 11oceaudio 顯示的頻譜圖)進行比較時，我們可以發現其結果會相似，在波型上的形狀相似尤其是某些突出點，這也代表濾波及阻波效果相似，但不會完全相同，因此也可以根據 oceaudio 顯示的頻譜圖推論我們產生之 log spectrum 正確(下圖 7-9)。

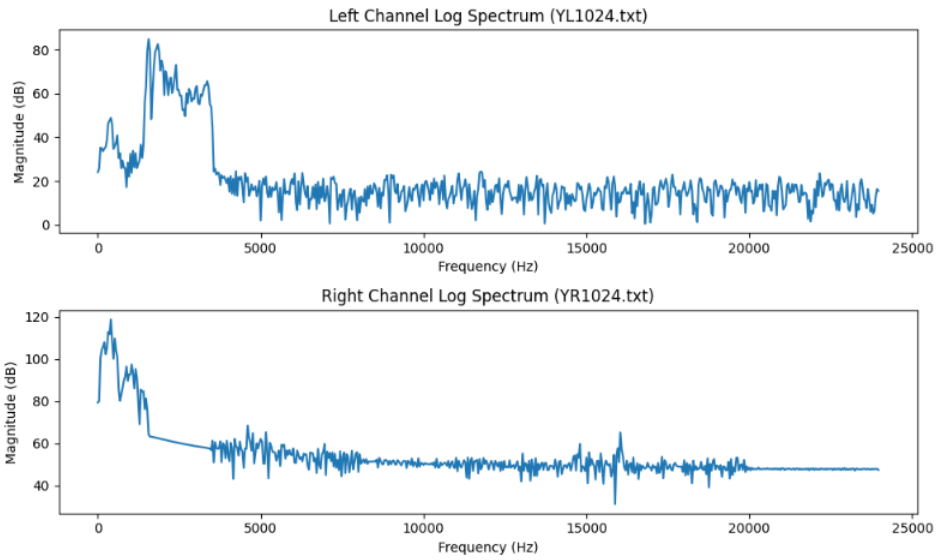

圖 7:M 為 1024 時之 logspectrum

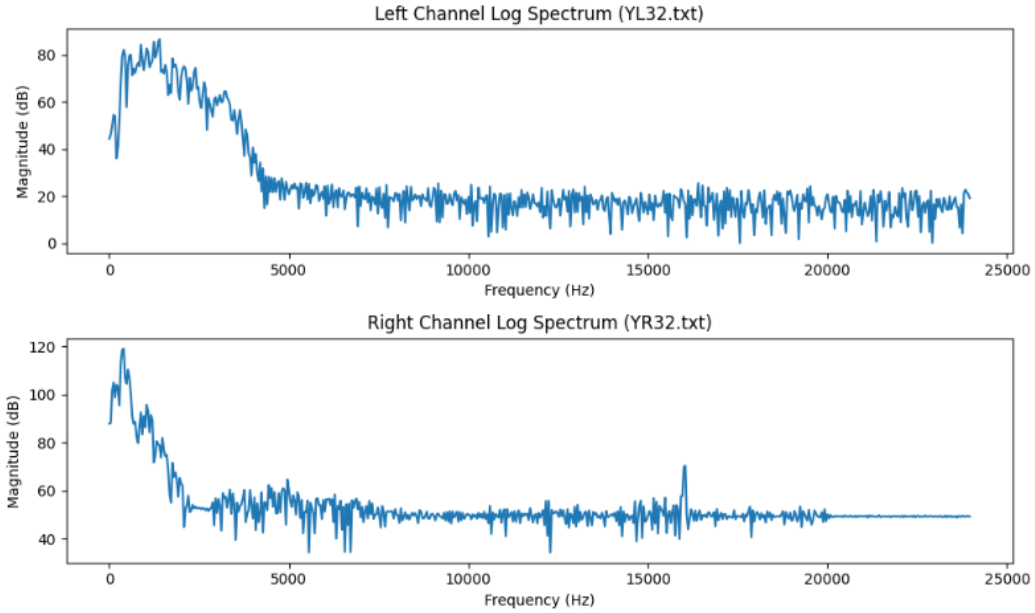

圖 8:M 為 32 時之 logspectrum

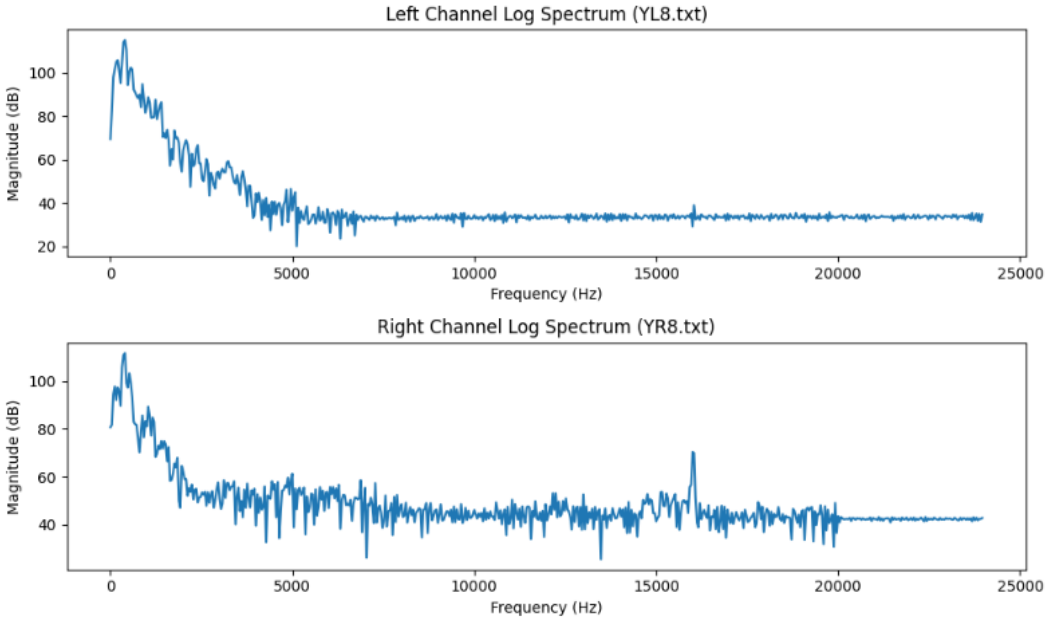

圖 9:M 為 8 時之 logspectrum

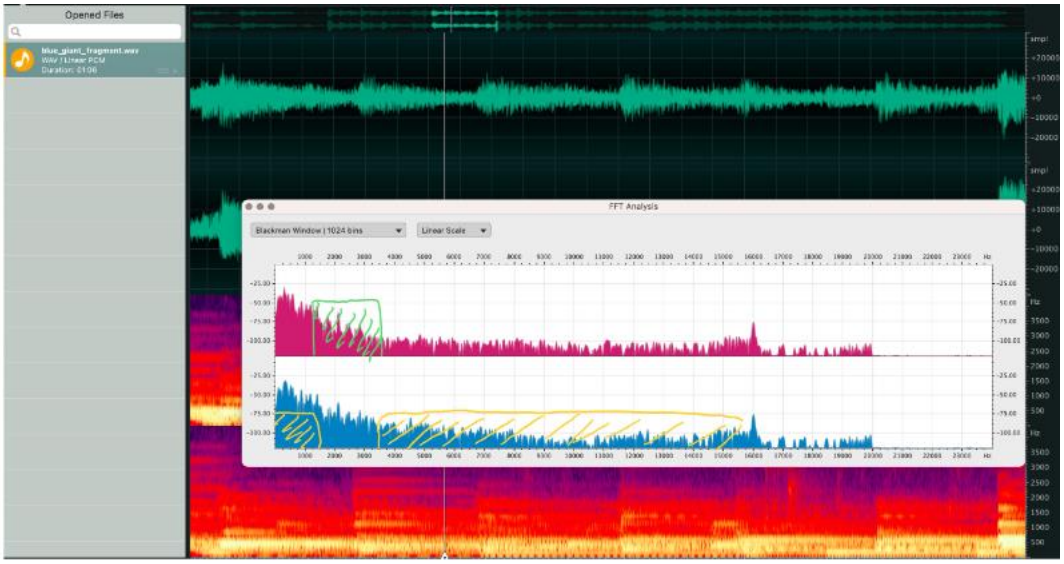

圖 10: 濾波器示意圖(綠色部分表示經過 bandpass 所剩之頻譜，橘色表示經過 bandstop 所剩之頻譜)

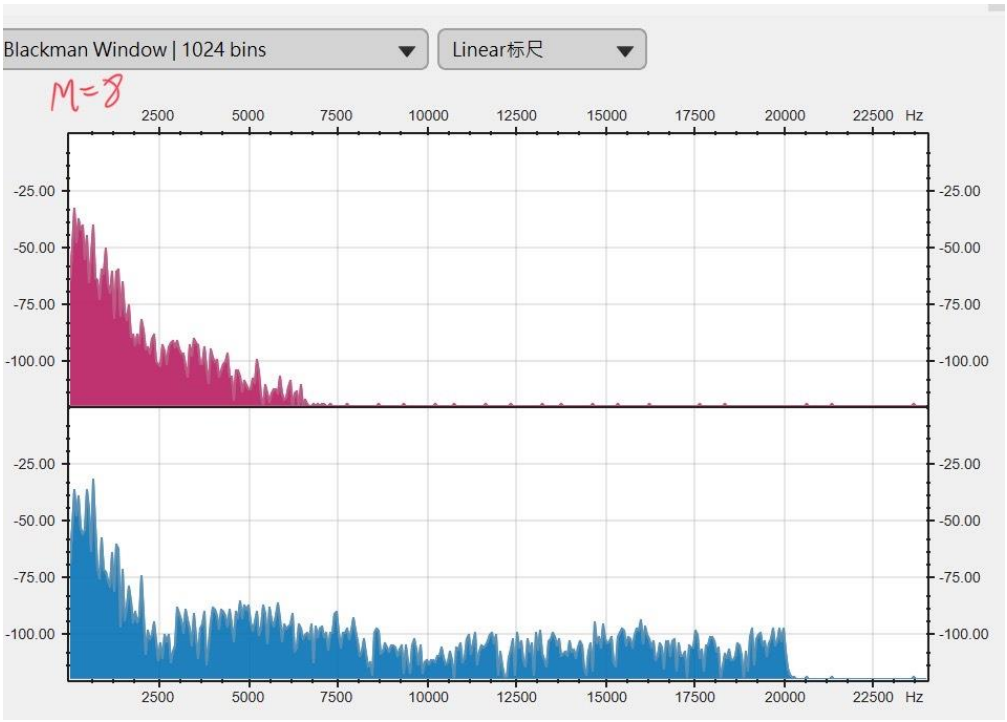

圖 11: oceaudio 顯示的頻譜圖

- **程式內容:**

1. 低通濾波器:如下圖 12， m 和 n 是代表濾波器的範圍以及當前的頻率。 FL 是低通濾波器的截止頻率。 FS 是音訊的取樣率。 PI 是圓周率。 首先，程式計算了截止頻率 wc，這是以弧度表示的截止頻率。 如果 n 等於 m，表示該係數位於濾波器的中心，則回傳 wc / PI。 如果 n 不等於 m，程式乘上 Hamming 窗函式的值（hamming 函式計算），為了將濾波器的頻率響應限制在截止頻率以內。

```
// 低通濾波器的頻率響應
float low_pass(int m, int n)
{
    float wc = 2 * PI * FL / FS;
    if (n == m)
    {
        return wc / PI;
    }
    else
    {
        return sinf(wc * ((float)(n - m))) / PI / ((float)(n - m)) * hamming(2 * m + 1, n);
    }
}
```

圖 12:低通濾波器的函式

2. 高通濾波器:FH 是高通濾波器的截止頻率。 其他變數和計算方式與低通濾波器相似，不同之處在於迴傳值的判定和低通濾波器相反。

```
// 高通濾波器的頻率響應
float high_pass(int m, int n)
{
    float wc = 2 * PI * FH / FS;
    if (n == m)
    {
        return 1.0 - wc / PI;
    }
    else
    {
        return (sinf(PI * ((float)(n - m))) - sinf(wc * ((float)(n - m)))) / PI / ((float)(n - m)) * hamming(2 * m +
    }
}
```

圖 13:高通濾波器的函式

3. 帶通濾波器: 帶通濾波器通常用於從一個信號中選擇特定的頻率帶。它允許通過一個特定的頻率範圍內的信號，而阻擋其他頻率範圍的信號。

```
// 帶通濾波器的頻率響應
float band_pass(int m, int n)
{
    float wh = (float)(0.19634955F)
    float wl = 2 * PI * FL / FS;
    if (n == m)
    {
        return 2.0 * (wh / PI - wl / PI);
    }
    else
    {
        return 2.0 * (sinf(wh * ((float)(n - m))) - sinf(wl * ((float)(n - m)))) / PI / ((float)(n - m)) * hamming(2
    }
}
```

圖 14:帶通濾波器的函式

4. 阻通濾波器:我將高通減去低通濾波器以實現阻帶濾波器

```
    // 建立右聲道的帶阻濾波器
    for (n = 0; n < (2 * M + 1); n++)
    {
        h_R[n] = high_pass(M, n) - low_pass(M, n);
        fprintf(file_hR, "%.15e\n", h_R[n]);
    }
```

圖 15:阻通濾波器的函式

5. 傅立葉轉換:根據下圖 16，此函示主要是對通郭濾波器後的數值進行傅立葉轉換，計算的方式是將數值轉為實部以及虛部在進行轉換如下圖 17

```
// 傅立葉轉換函數
void DFT(short int *xn, int len, FILE *file_Y)
{
    int a, n;
    float Xr[N_FFT];
    float Xi[N_FFT];

    for (a = 0; a < N_FFT; a++)
    {
        Xr[a] = 0;
        Xi[a] = 0;

        for (n = START; n <= END; n++)
        {
            Xr[a] += xn[n] * hamming(N_FFT - 1, n - START) * cosf(2 * PI * a * (n - START) / N_FFT);
            Xi[a] -= xn[n] * hamming(N_FFT - 1, n - START) * sinf(2 * PI * a * (n - START) / N_FFT);
        }

        // 取log後乘以20
        float magnitude = 20 * log10f(sqrtf(Xr[a] * Xr[a] + Xi[a] * Xi[a]));
        fprintf(file_Y, "%.15e\n", fabs(magnitude));
    }
}
```

圖 17:離散傅立葉轉換的函式

$$e^{-j2\pi kn/N} = cos(\frac{2\pi kn}{N}) - jsin(\frac{2\pi kn}{N})$$

圖 17:傅立葉轉換之計算

- **心得與反思**:此專題的製作我學到了如何使用不同類型的濾波器（低通、高通和帶通）來處理音訊。這使我對信號處理中的數學原理有了更深入的理解。此外，通過實作傅立葉轉換函式，我學會了如何將時域信號轉換為頻域信號，並且能夠將其應用於音訊數據，從而更好地了解信號的頻譜特性。