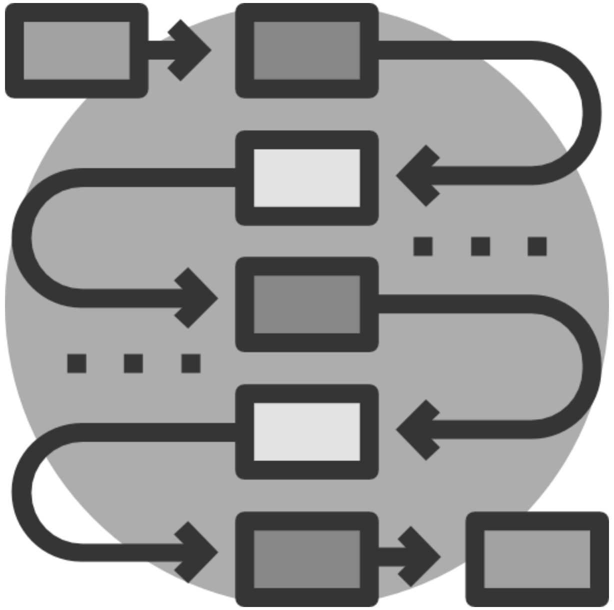
雜訊消除

組員：許家愷、楊志璿、涂哲誠

Introduction



Introduction

Decode

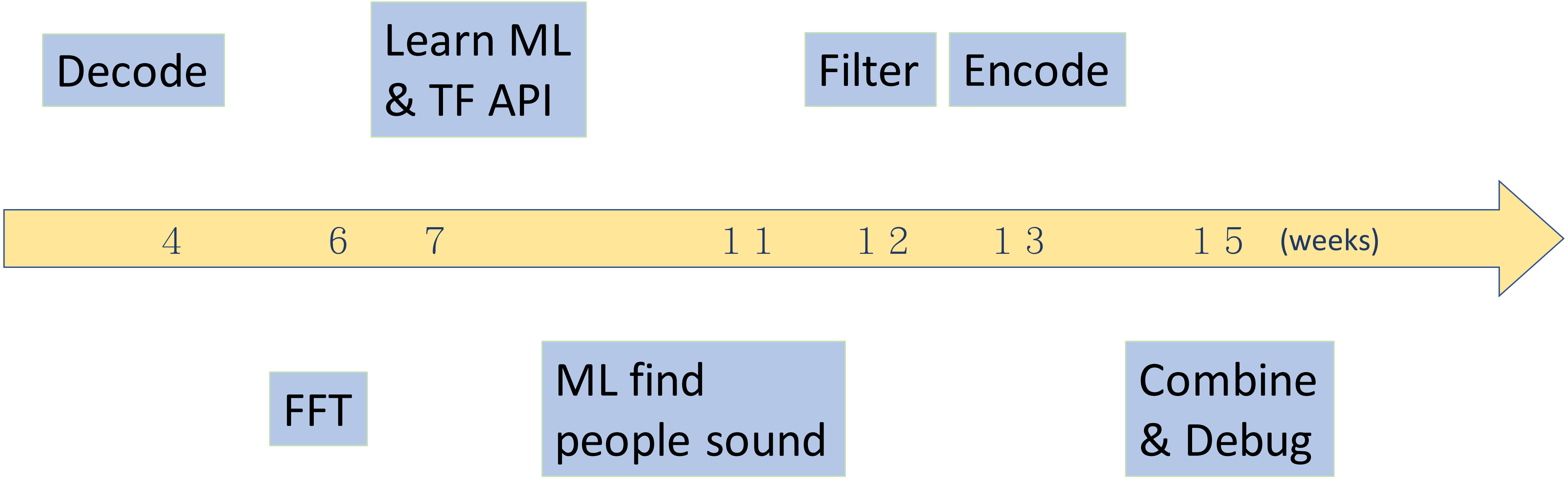
Readfile

FFT/IFFT

Noise

Encode

Schedule



Decode

Learn ML & TF API

Filter

Encode

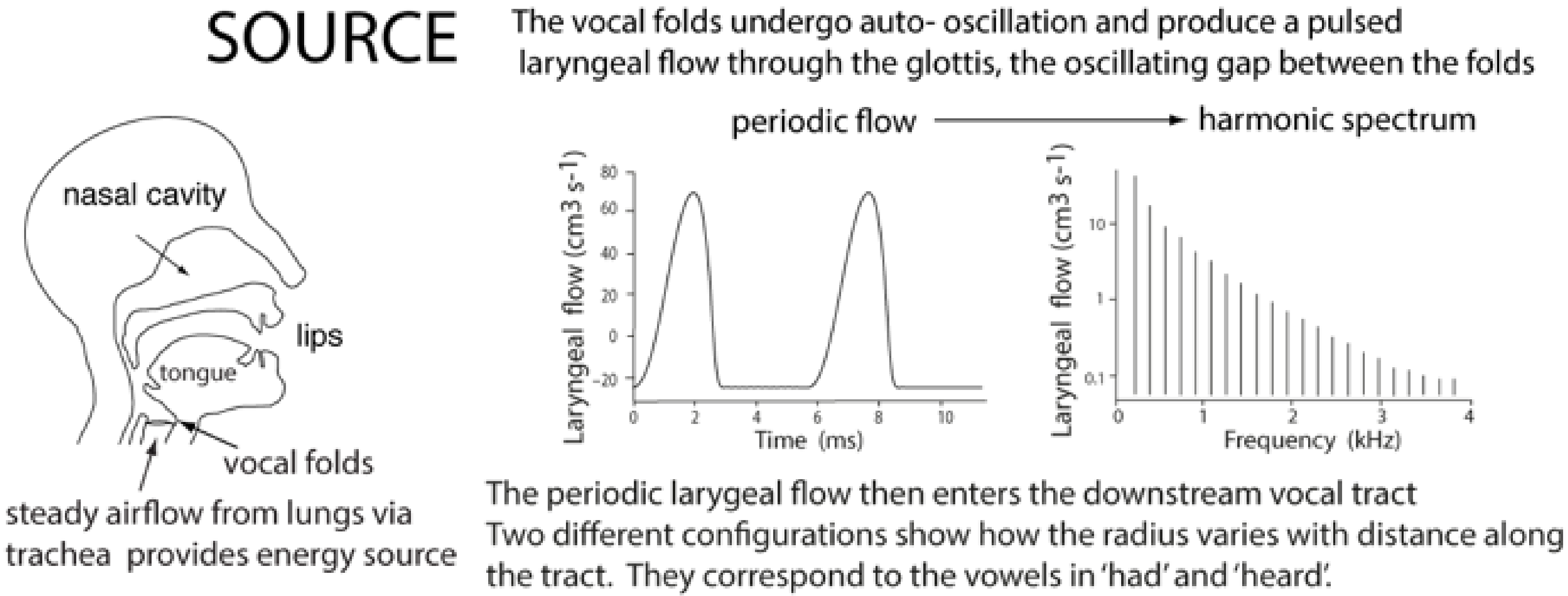
4 6 7 11 12 13 15 (weeks)

FFT

ML find people sound

Combine & Debug

Method



SOURCE

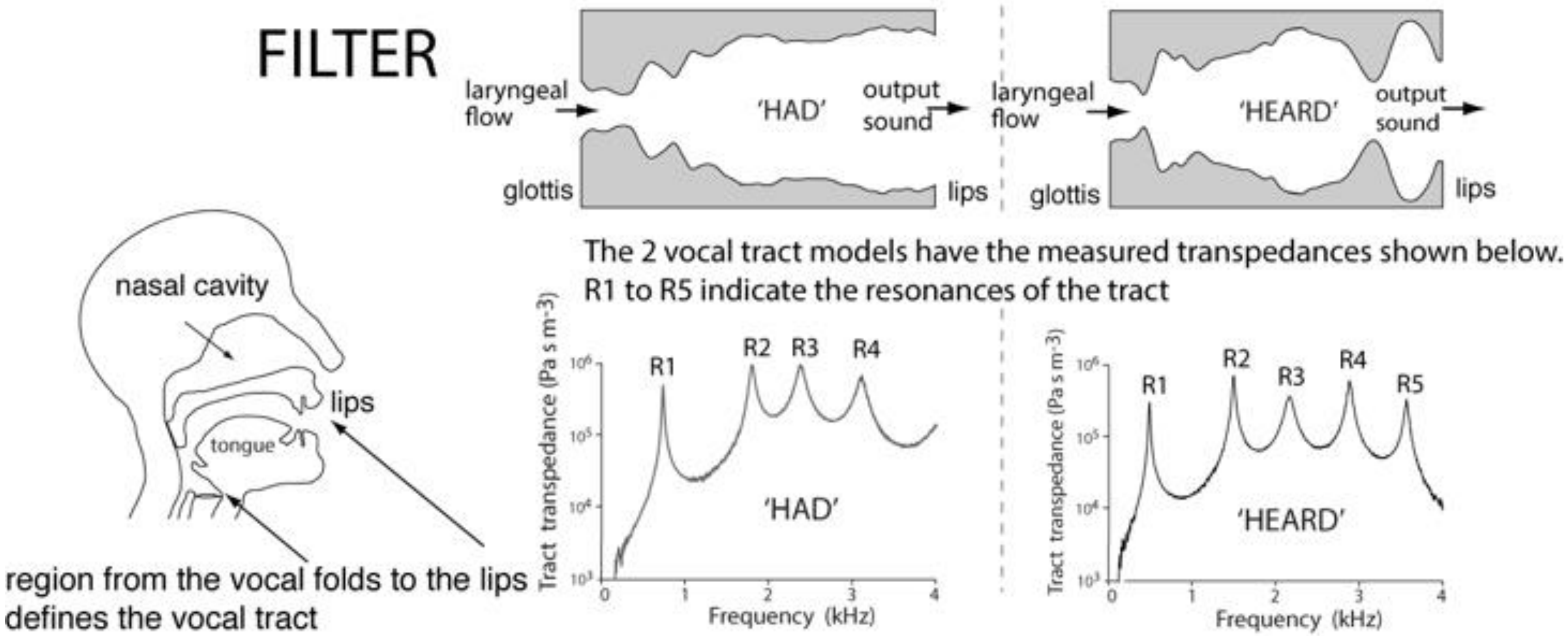
The vocal folds undergo auto- oscillation and produce a pulsed laryngeal flow through the glottis, the oscillating gap between the folds

steady airflow from lungs via trachea provides energy source

The periodic larygeal flow then enters the downstream vocal tract Two different configurations show how the radius varies with distance along the tract. They correspond to the vowels in 'had' and 'heard'.

https://newt.phys.unsw.edu.au/jw/voice.html

Method

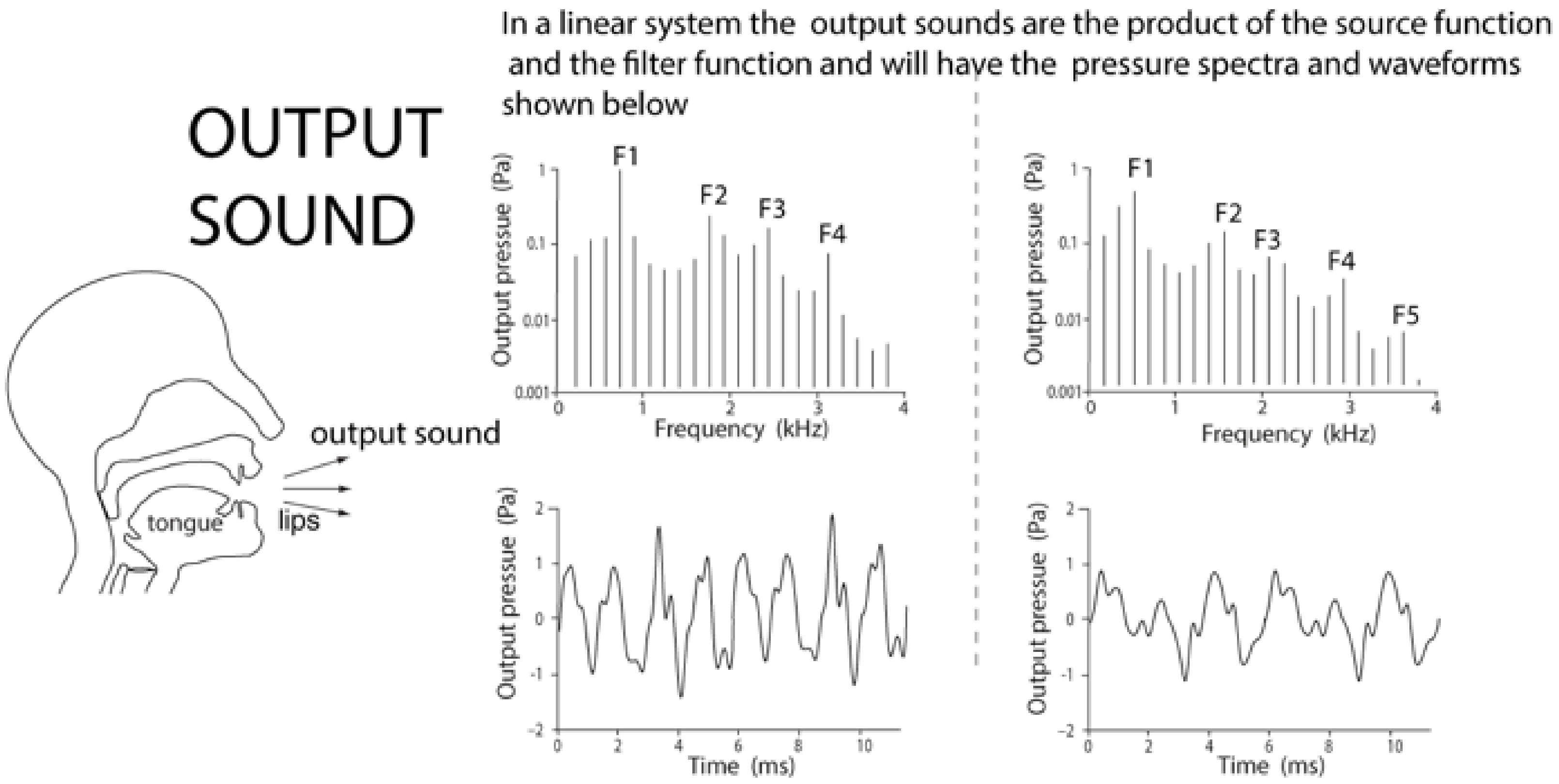


The 2 vocal tract models have the measured transpedances shown below. R1 to R5 indicate the resonances of the tract

region from the vocal folds to the lips defines the vocal tract

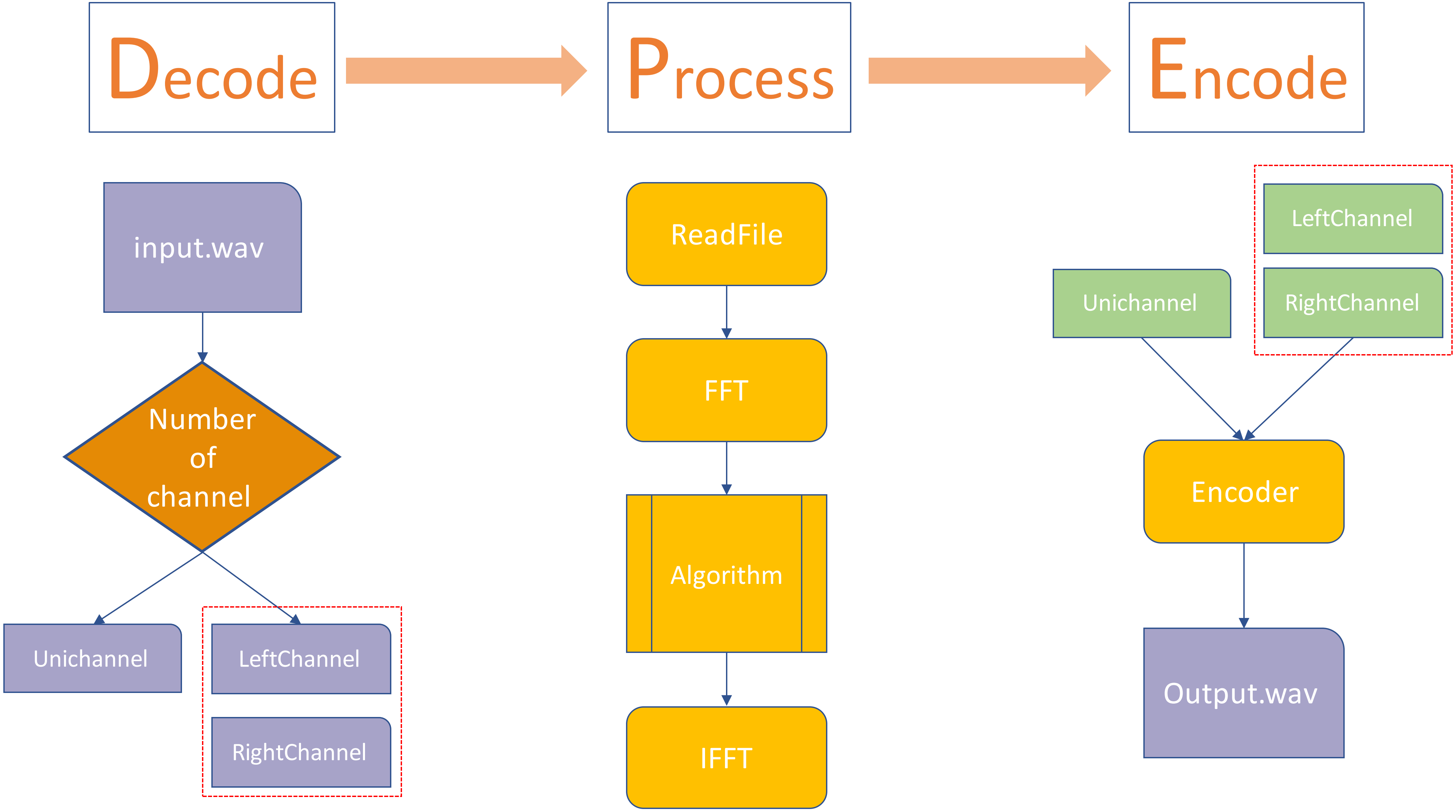
https://newt.phys.unsw.edu.au/jw/voice.html

Method



In a linear system the output sounds are the product of the source function and the filter function and will have the pressure spectra and waveforms shown below

https://newt.phys.unsw.edu.au/jw/voice.html



Decode

input.wav

Number of channel

Unichannel

LeftChannel

RightChannel

Process

ReadFile

FFT

Algorithm

IFFT

Encode

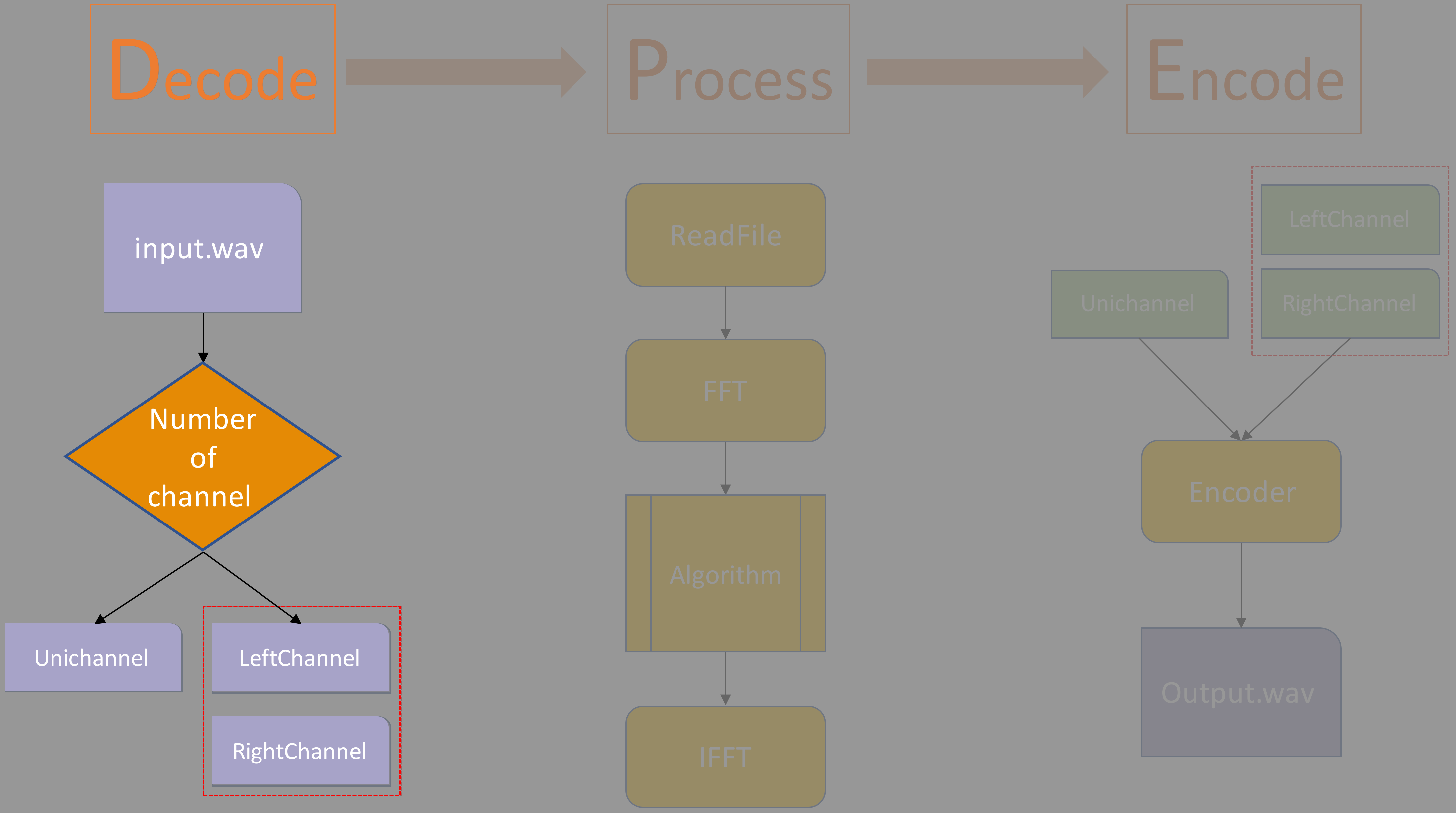
LeftChannel

Unichannel

RightChannel

Encoder

Output.wav



Decode

Process

Encode

input.wav

Number of channel

Unichannel

LeftChannel

RightChannel

ReadFile

FFT

Algorithm

IFFT

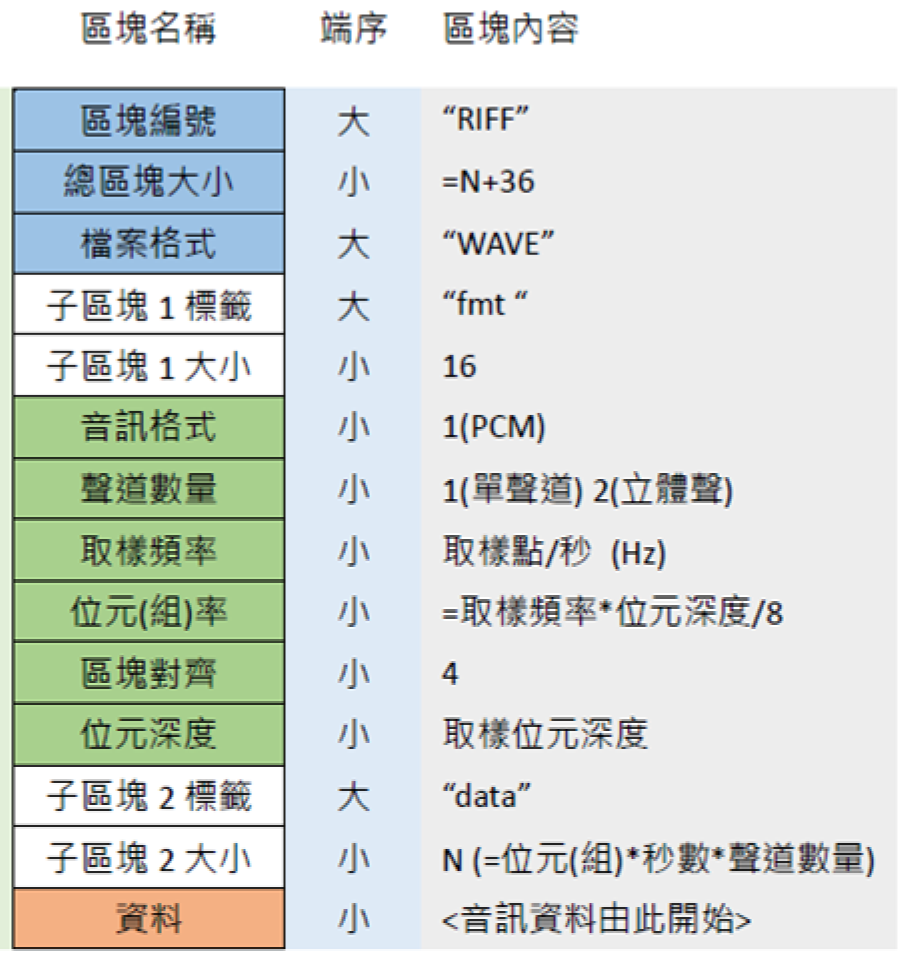
LeftChannel

Unichannel

RightChannel

Encoder

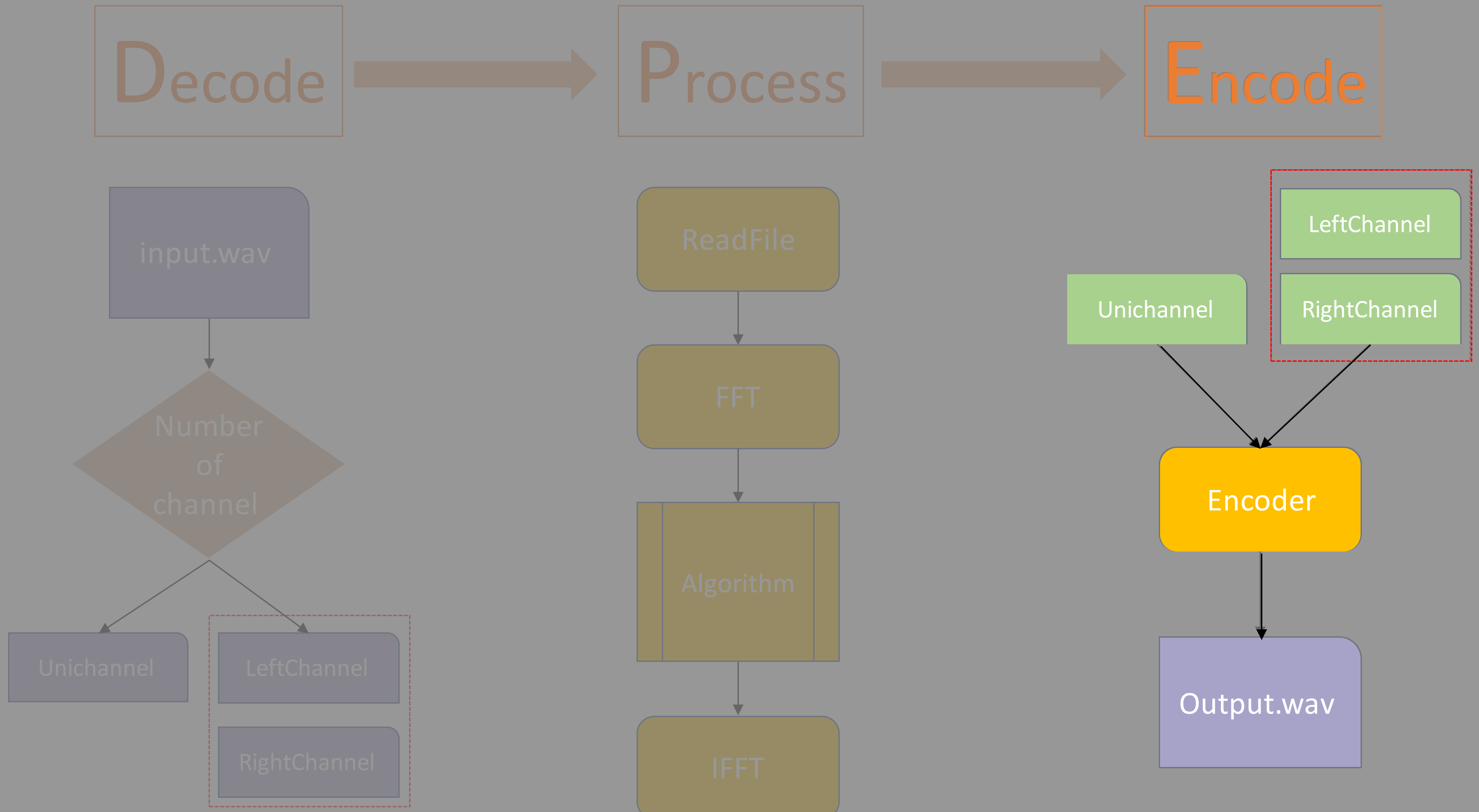
Output.wav



區塊名稱	端序	區塊內容
區塊編號	大	“RIFF”
總區塊大小	小	=N+36
檔案格式	大	“WAVE”
子區塊 1 標籤	大	“fmt “
子區塊 1 大小	小	16
音訊格式	小	1(PCM)
聲道數量	小	1(單聲道) 2(立體聲)
取樣頻率	小	取樣點/秒 (Hz)
位元(組)率	小	=取樣頻率*位元深度/8
區塊對齊	小	4
位元深度	小	取樣位元深度
子區塊 2 標籤	大	“data”
子區塊 2 大小	小	N (=位元(組)*秒數*聲道數量)
資料	小	<音訊資料由此開始>

Decode

```
import wave                                   # python module
f = wave.open(filename, 'rb’)                 # in binary mode
params = f.getparams()                        # 取得音檔前面的參數
nchannels, sampwidth, framerate, nframes = params[:4]
strData = f.readframes(nframes)               # 讀取整首歌
waveData = np.fromstring(strData, dtype=np.int16)
# np.fromstring 會自動把兩個bytes 翻轉過來 並轉成int
```



Decode

Process

Encode

input.wav

Number of channel

Unichannel

LeftChannel

RightChannel

ReadFile

FFT

Algorithm

IFFT

LeftChannel

Unichannel

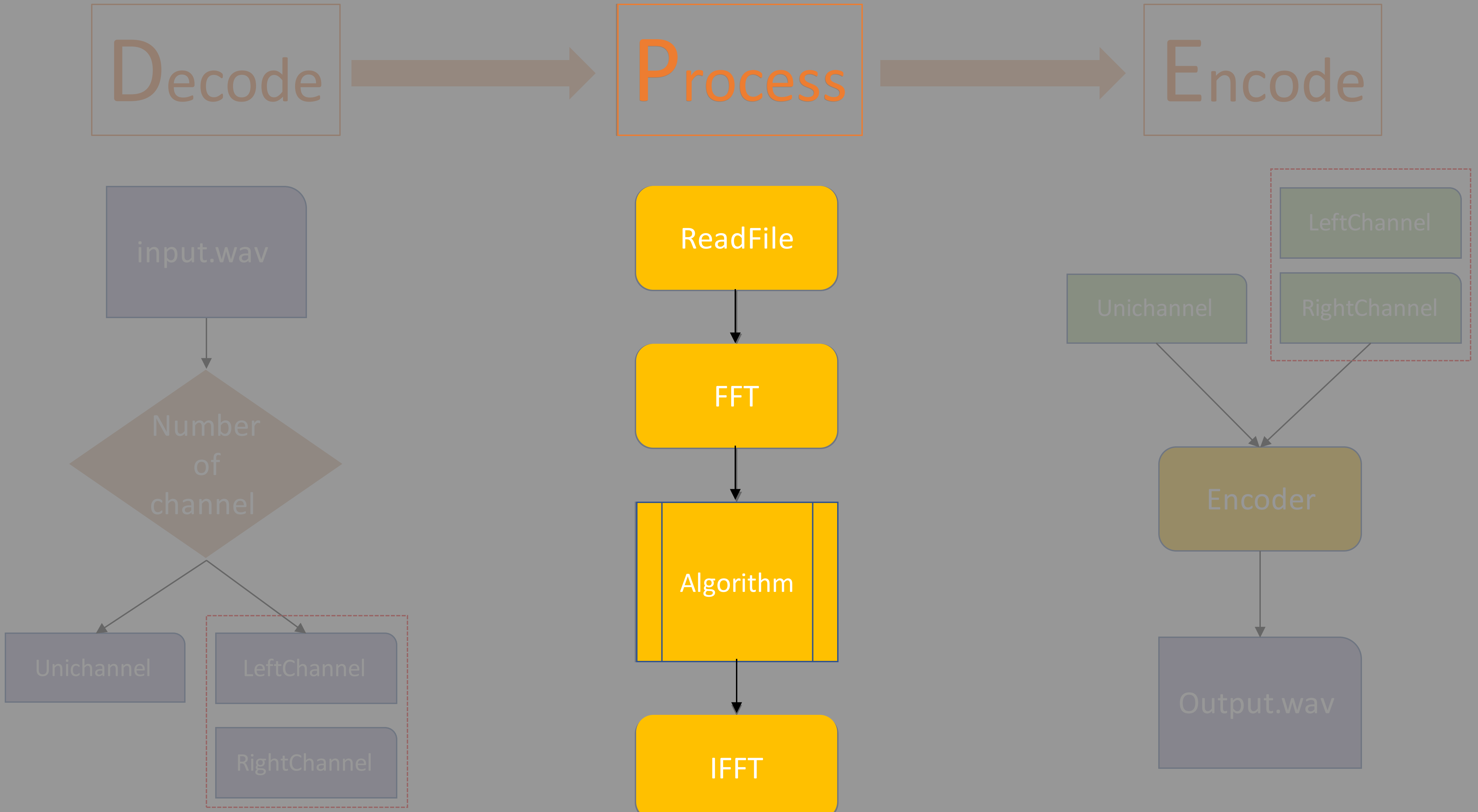
RightChannel

Encoder

Output.wav

Encode

```
for l,r in zip(L_IFFT, R_IFFT):                # 把兩個陣列合併
        change = int(l)
        if change < 0 :
            change = change+65536              # 補數關係，小於0要加65536
        lastE = change%128                     #切割兩個byte
        firstE = change>>8
        try:                                   #預防寫入的資料過大
            writeFile.writeframes((lastE).to_bytes(1, byteorder='big')) #先寫入後面的byte
            writeFile.writeframes((firstE).to_bytes(1, byteorder='big')) #再寫入前面的byte
        except:
            print(change) #把過大的數字印出
```



Decode

Process

Encode

input.wav

Number of channel

Unichannel

LeftChannel

RightChannel

ReadFile

FFT

Algorithm

IFFT

LeftChannel

Unichannel

RightChannel

Encoder

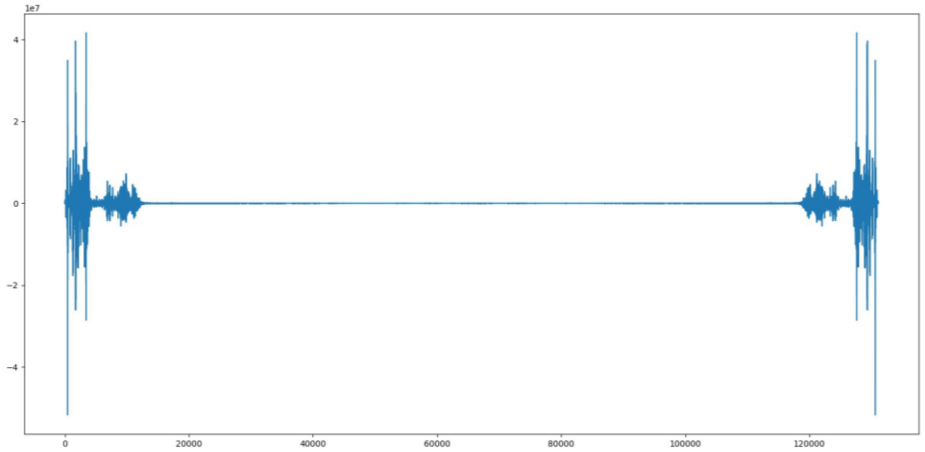
Output.wav

Algorithm – Filter & subtraction

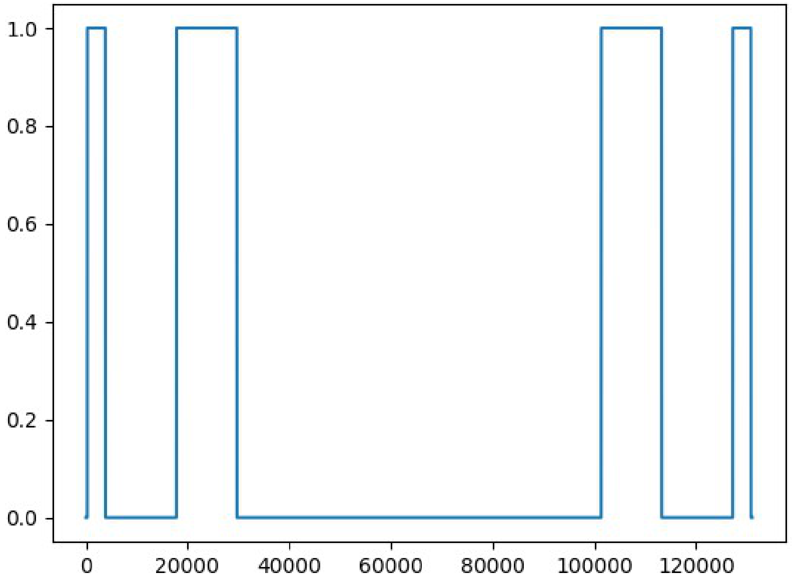
```
def freqFilter(freqTable):
    for i in range(1 << num):
        for j in len(lowerBound):
            if ((not (i > lowerBound[j]
                 and i < upperBound[j]))):
                freqTable[i] = 0
    return freqTable
```

```
def subtraction(rawFreqData, mode=0,
                toFindSuitableNoise=False):
    filtedData = freqFilter(rawFreqData)
    if toFindSuitableNoise == True:
        findSuitableNoise(filtedData, mode)
    return [a-b for a,b in zip(filtedData, SuitableNoise)]
```

Imagine Filter

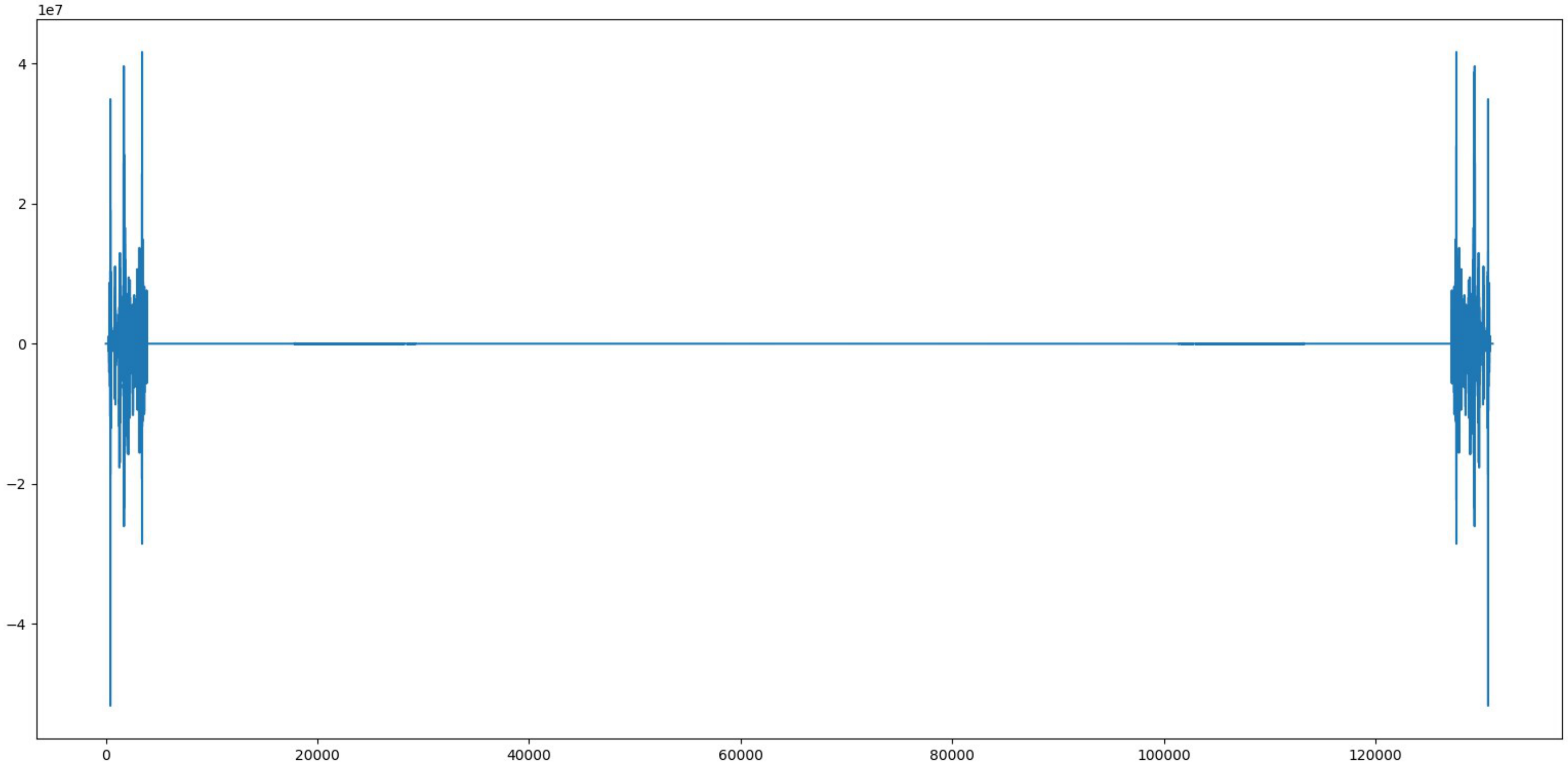


×

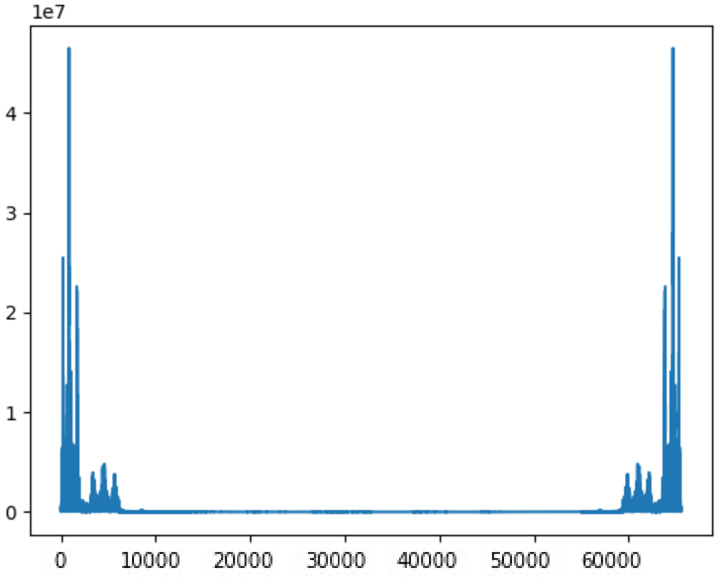


=

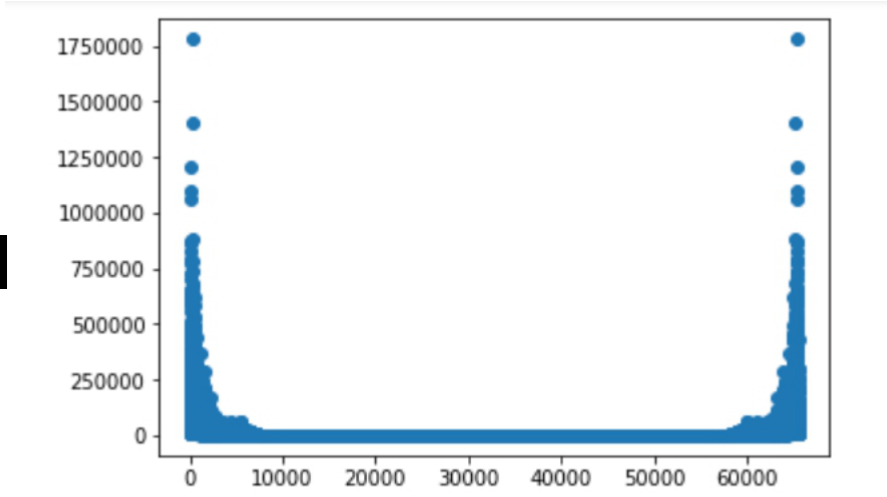
Imagine Filter



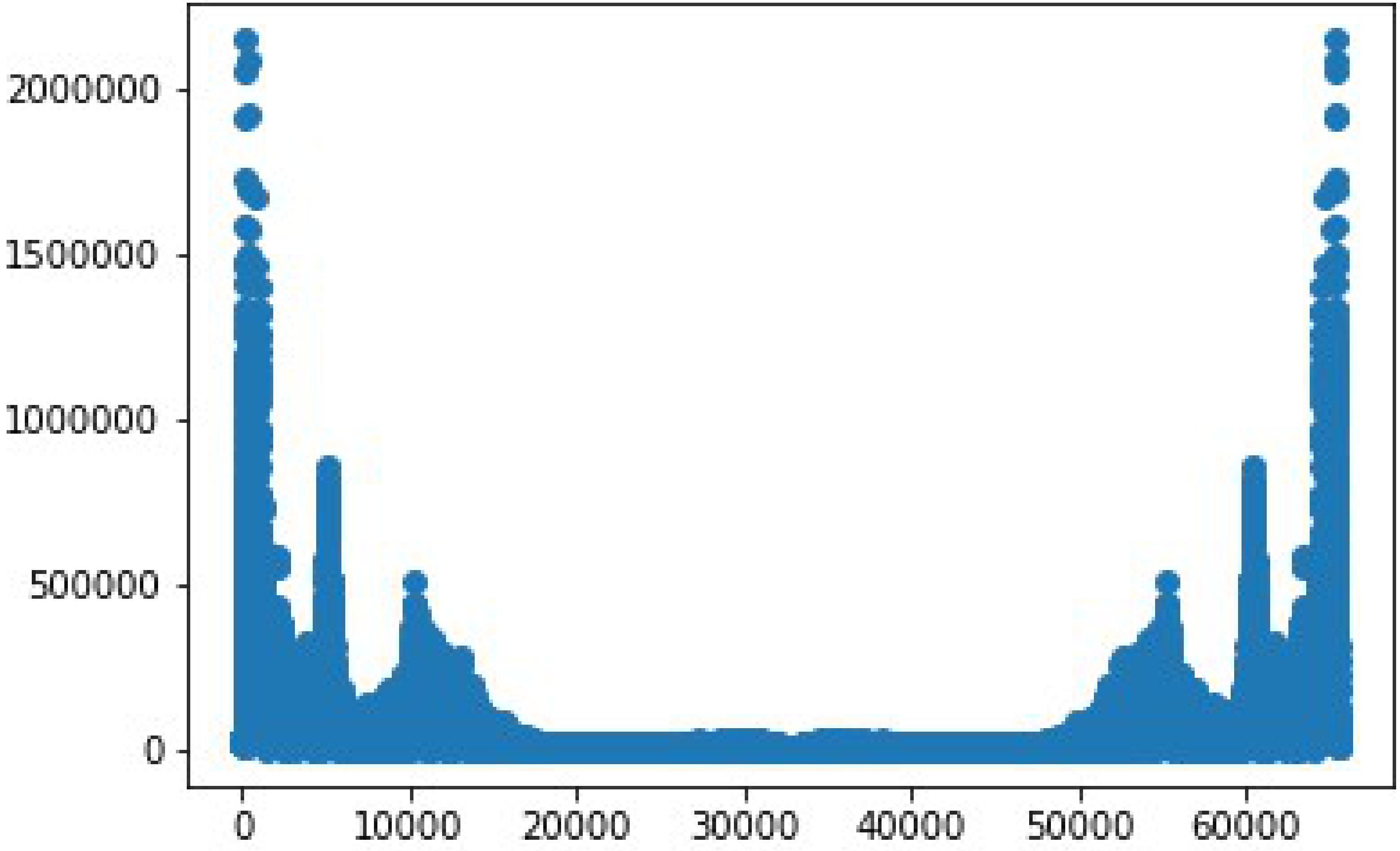
Imagine subtraction



−



→


2000000
1500000
1000000
500000
0
0
10000
20000
30000
40000
50000
60000

Algorithm – Find suitable noise

```
def findSuitableNoise(filtedData, mode=0):
    '''
    filtedData is the frequency graph which had been filtered
    mode = 0 is abslute
         = 1, 1.4, 2... etc. is power
            (Real number not complex)
    ‘’’
    processFunction = testingPower if mode else testingAbs
    success, times = processFunction(filtedData, mode)
    SuitableNoise = [x * times for x in success]
```

Algorithm – Find suitable noise

```
def testingAbs(rawData, mode):
    rawDataSum = 0
    result = []
    for low, high in zip(lowerBound, upperBound):
        rawDataSum += sum(rawData[low:high + 1])

    for i in range(len(noiseData)):
        subtract = 0
        print(len(noiseData[i]))
        print(len(rawData))
        for j in range(len(noiseData[i])):
            tmp = noiseData[i][j] * (rawDataSum / sN[i])
            subtract += abs(rawData[j] - tmp)
        result.append((subtract, i))

    bestNoise = min(result)
    return bestNoise, rawDataSum / sN[bestNoise[1]]
```

Algorithm – Find suitable noise

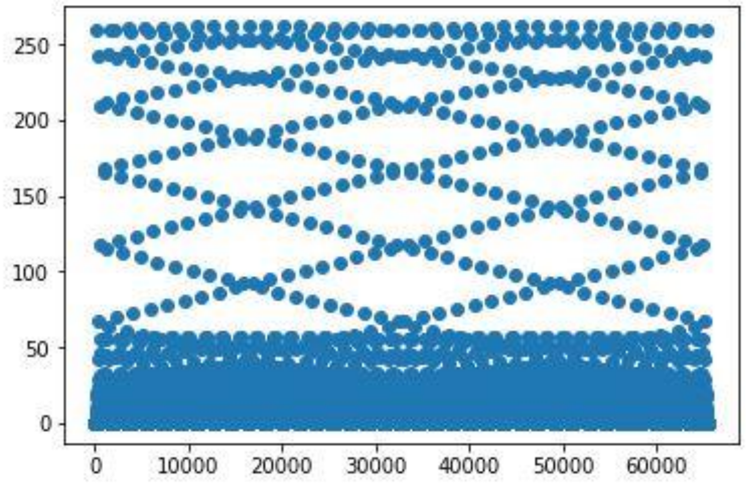
```
def findSuitableNoise(filtedData, mode=0):
    '''
    filtedData is the frequency graph which had been filtered
    mode = 0 is abslute
         = 1, 1.4, 2... etc. is power
            (Real number not complex)
    ‘‘’
    processFunction = testingPower if mode else testingAbs
    success, times = processFunction(filtedData, mode)
    SuitableNoise = [x * times for x in success]
```

The better way

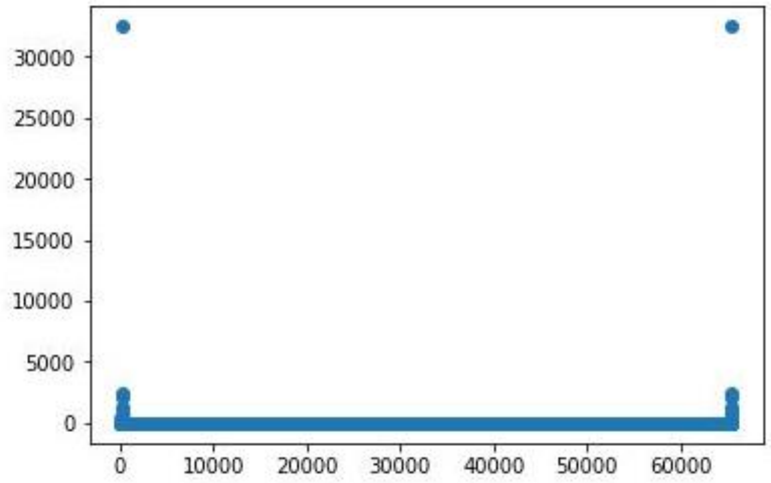
- Time domain subtraction
- Smoothly generate

Problems-1

Our fft in y=sin(pi/250x)



numpy fft in y=sin(pi/250x)

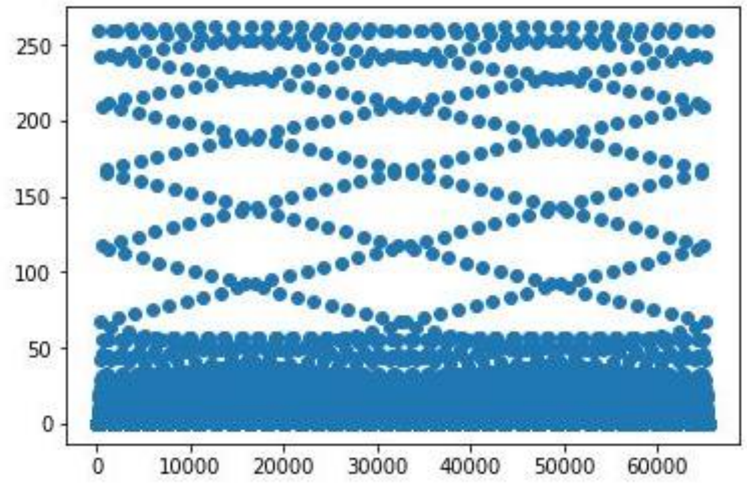


Problems-1

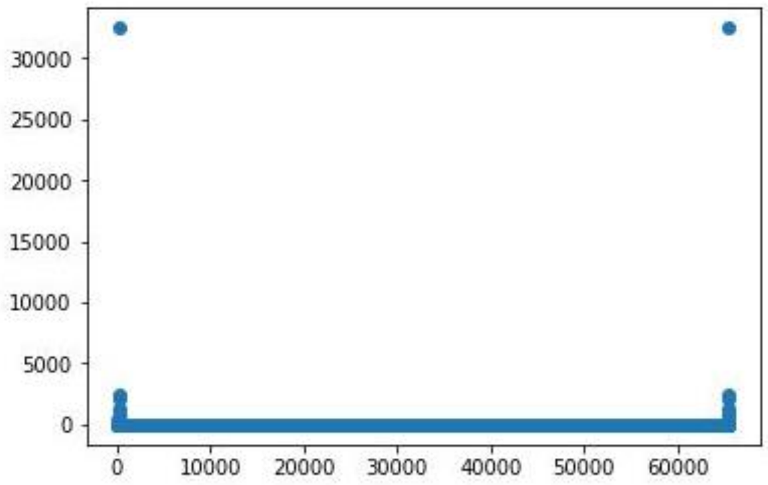
- FFT dtype=int64

```
def fft_T(x):
    # must x size must be 2^N
    x = np.asarray(x, dtype=np.complex64)
    n = x.shape[0]

    if n < 8:
        numberList = np.arange(n)
        i = numberList.reshape((n, 1))  # turn to column
        M = np.exp(-2j * pi * i * numberList / n)
        return np.dot(x, M)  # matrix dot product

    X_even = fft_T(x[::2])
    X_odd = fft_T(x[1::2])
    twiddleFactor = np.exp(-1j * pi * np.arange(n >> 1) / (n >> 1))  # half
    return np.concatenate([X_even + twiddleFactor * X_odd, X_even - twiddleFactor * X_odd])
```

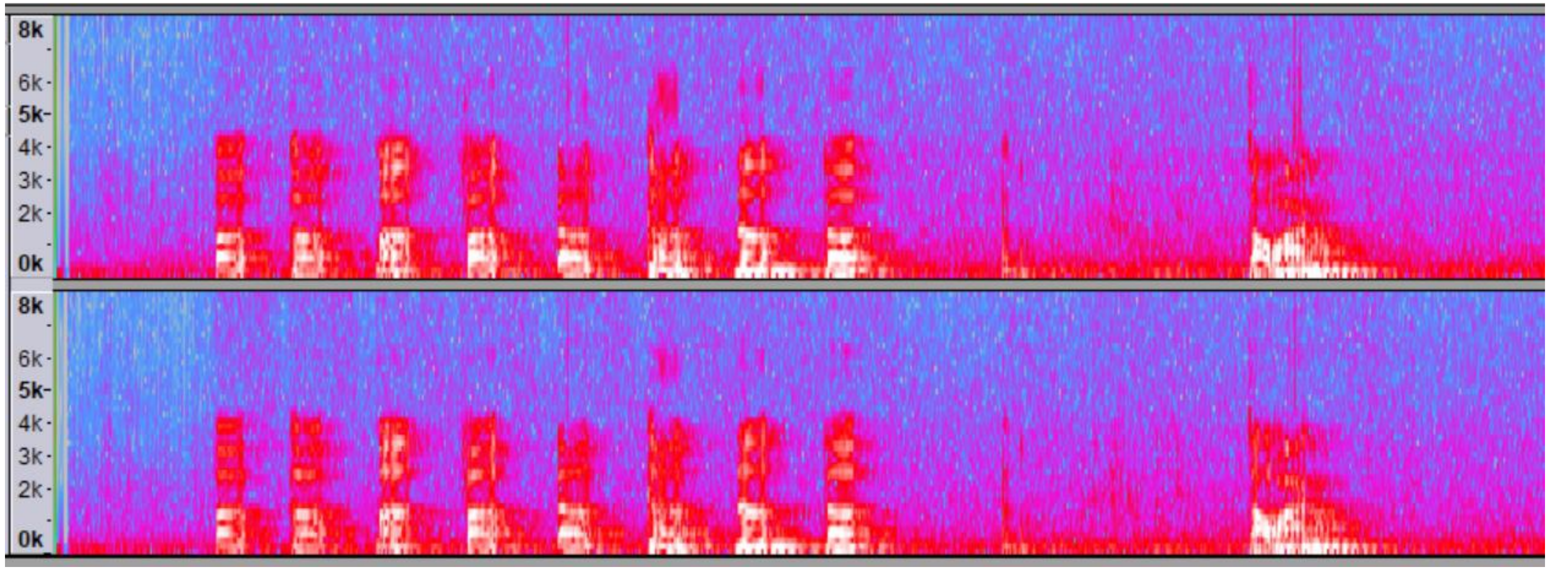
Our fft in y=sin(pi/250x)



numpy fft in y=sin(pi/250x)



Problems-2



Problems-2

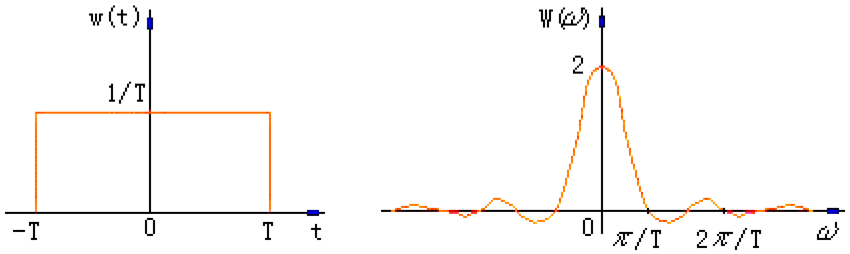
- Our filter is equivalent to multiply window functions
- $F(\omega) \cdot G(\omega) \;=\; f(t)\ \text{Convolution}\ g(t)$

```
def freqFilter(freqTable):
    # 1<<num is sample rate
    # so num is same as fft.py
    global num
    global filterTable
    After_filter = [0]*(1<<num)
    for run in range(1<<num):
        After_filter[run] = filterTable[run] * freqTable[run]
```

Problems-2

- Our filter is equivalent to multiply window functions
- $\mathrm{F}(\omega) \cdot \mathrm{G}(\omega) \ = \ \mathrm{f}(\mathrm{t})$ Convolution $\mathrm{g}(\mathrm{t})$

```
def freqFilter(freqTable):
    # 1<<num is sample rate
    # so num is same as fft.py
    global num
    global filterTable
    After_filter = [0]*(1<<num)
    for run in range(1<<num):
        After_filter[run] = filterTable[run] * freqTable[run]
```



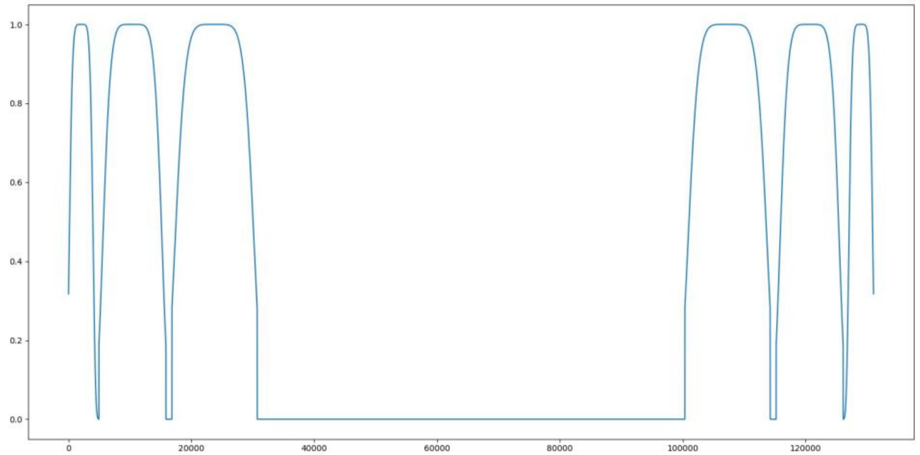
src: https://www.twblogs.net/a/5b8cf3582b71771883374f58

Problems-2

```
filterTable=[0]*(1<<num)
lowerBound = [90, 2000, 6000]                    #90, 2000, 6000
upperBound = [1300, 5000, 10000]                 #1300, 5000,10000
for l in range( len(lowerBound) ):
    lowerBound[l] = round( lowerBound[l]/( 44100/(1<<num) ) )
    upperBound += [ (1<<num) - lowerBound[l] ]
for u in range( len(upperBound) ):
    upperBound[u] = round( upperBound[u]/( 44100/(1<<num) ) )
    lowerBound += [ (1<<num) - upperBound[u] ]
lowerBound.sort()
upperBound.sort()
expand = 1000

def gaussGenerate():
    for l,u in zip(lowerBound, upperBound):
        sig = signal.general_gaussian( (u-l) + expand*2, sig = (u-l)//2, p = 3)
        for run in range( (u-l) + expand*2 ):
            cac = run – expand + l
            if(cac >= 0 and cac < 131072):
                filterTable[cac] = sig[run]

gaussGenerate()
```

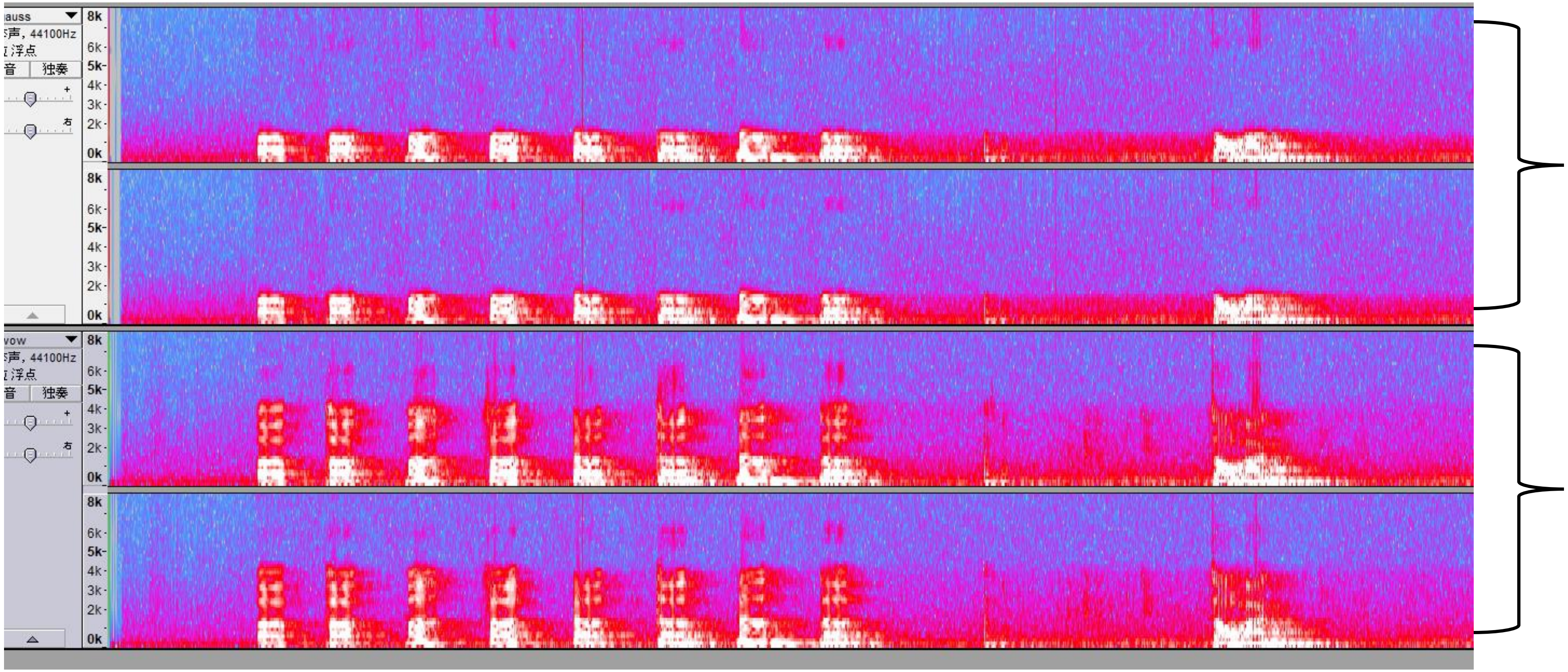
Problems-2

New filter



```
filterTable=[0]*(1<<num)
lowerBound = [90, 2000, 6000]                          #90, 2000, 6000
upperBound = [1300, 5000, 10000]                       #1300, 5000,10000
for l in range( len(lowerBound) ):
    lowerBound[l] = round( lowerBound[l]/( 44100/(1<<num) ) )
    upperBound += [ (1<<num) - lowerBound[l] ]
for u in range( len(upperBound) ):
    upperBound[u] = round( upperBound[u]/( 44100/(1<<num) ) )
    lowerBound += [ (1<<num) - upperBound[u] ]
lowerBound.sort()
upperBound.sort()
expand = 1000

def gaussGenerate():
    for l,u in zip(lowerBound, upperBound):
        sig = signal.general_gaussian( (u-l) + expand*2, sig = (u-l)//2, p = 3)
        for run in range( (u-l) + expand*2 ):
            cac = run – expand + l
            if(cac >= 0 and cac < 131072):
                filterTable[cac] = sig[run]

gaussGenerate()
```

Problems-2



Guass

Origin

Problems-3

```
#numpy.fft.ifft()
def ifft(a, n=None, axis=-1, norm=None):
    a = asarray(a)
    if n is None:
        n = a.shape[axis]
    fct = 1/n
    if norm is not None and _unitary(norm):
        fct = 1/sqrt(n)
    output = _raw_fft(a, n, axis, False, False, fct)
    return output
```

```
#Our ifft_T()

def ifft_T(x):
    x = np.array(x, dtype=complex)
    n = len(x)

    output = fft_T(x)

    return [element / n for element in output]
```

Problems-3

```
#numpy.fft.ifft()
def ifft(a, n=None, axis=-1, norm=None):
    a = asarray(a)
    if n is None:
        n = a.shape[axis]
    fct = 1/n
    if norm is not None and _unitary(norm):
        fct = 1/sqrt(n)
    output = _raw_fft(a, n, axis, False, False, fct)
    return output
```

```
#Our ifft_T()
def ifft_T(x):
    x = np.array(x, dtype=complex)
    n = len(x)

    output = fft_T(x)

    return [element / n for element in output]
```

```
>>> a = range(1<<5)
>>> b = fft_T(b)
>>> c = ifft_T(b)
>>> c
[0j
(31.00000000649298+1.9984014443252818e-15j)
(30.000000085901572+3.730689515421092e-15j)
(29.000000170765194+5.10702591327572e-15j)
…
(16.99999994318653+1.9984014443252818e-15j)
(16+0j)
(15.000000056813466-6.661338147750939e-16j)
…
(2.9999998292348096-8.881784197001252e-16j)
(1.9999999140984315-5.2850017498963115e-15j)
(0.9999999935070285-7.327471962526033e-15j)]
>>>
```

Problems-3

```
def ifft_T(x):
    x = np.array(x, dtype=complex)
    n = len(x)
    half = n//2
    x = np.concatenate((x[half:], x[:half]),  axis = None) #shifting
    output = fft_T(x)
    output = list(reversed(output))
    output.insert(0, output[-1])
    del output[-1]
    return [element / n for element in output]
```

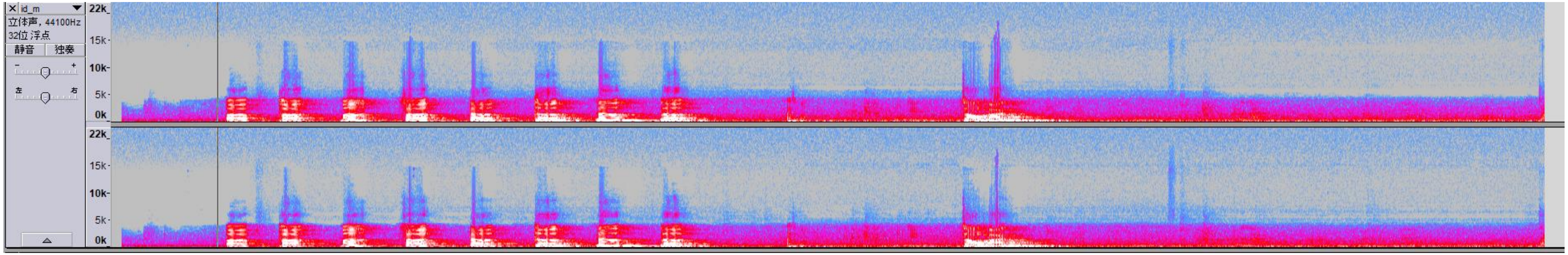
Problems-3

```
def ifft_T(x):
    x = np.array(x, dtype=complex)
    n = len(x)
    half = n//2
    x = np.concatenate((x[half:], x[:half]), axis = None) #shifting
    output = fft_T(x)
    output = list(reversed(output))
    output.insert(0, output[-1])
    del output[-1]
    return [element / n for element in output]
```

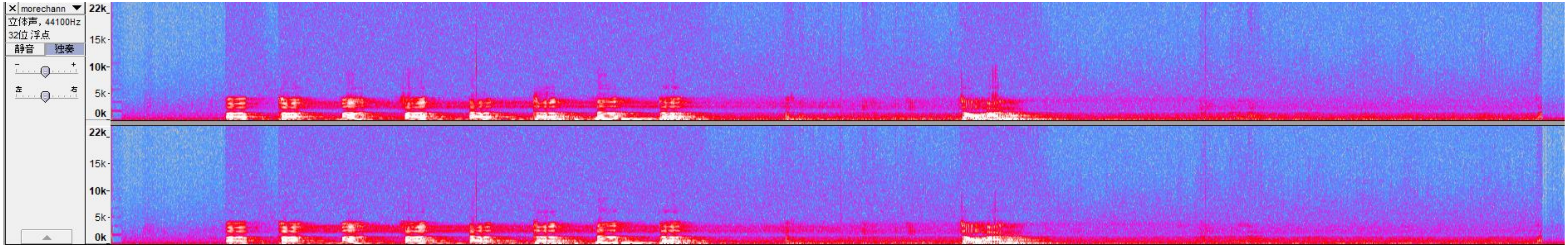
It works,
but I still do not know why!!

Problems-4

Before process



After process



Project result