SNR-based Subtraction

April 29, 2019

In [1]:
```
#Load necessary libraries
import numpy as np
import matplotlib.pyplot as plt
from scipy.io.wavfile import read, write
from numpy.fft import fft, ifft
from IPython.display import Audio
from scipy import signal
from ipykernel import kernelapp as app
```

In [2]:
```
FRAME_SIZE = 1024
```

In [3]:
```
# An algorithm to detect whether a frame is voiced or not
def voice_decider(frame):
    isVoiced = 0
    energy = 0

    for i in range(len(frame)):
        energy = energy + frame[i]*frame[i]

    if(energy>990000000):
        isVoiced = 1

    return isVoiced
```

In [4]:
```
# Load contaminated signal by crowd noise
Fs, data = read('test_audio.wav')
numFrames = int(len(data) / 1024)
output = np.zeros(len(data))
crowd_filter = np.zeros(FRAME_SIZE)
window_num = 0
noise_spectrum = np.zeros(FRAME_SIZE)
```

In [5]:
```
for i in range(numFrames):
    frame = data[i * FRAME_SIZE : ((i+1) * FRAME_SIZE)]

    if(voice_decider(frame.astype(float)) == 0):
        curFft = np.abs(np.fft.fft(frame))
        # For unvoiced frames, collect statistics about noise spectrum
```

```
            noise_spectrum = (noise_spectrum * window_num + curFft)/(window_num + 1)
            window_num = window_num + 1

            # Attenuate noise
            output[i * FRAME_SIZE : ((i+1) * FRAME_SIZE)] = frame/10000
            continue

        DFT = np.fft.fft(frame)

        noise_energy = 0
        speech_energy = 0

        # Calculating speech and noise energy
        for j in range(FRAME_SIZE):
            noise_energy = noise_energy + noise_spectrum[j]*noise_spectrum[j]
            speech_energy = speech_energy + np.abs(DFT[j])*np.abs(DFT[j])

        # Calculating SNR
        SNR = 20*np.log10(speech_energy/noise_energy)

        # Determine alpha
        if (SNR<-5):
            alpha = 4.75
        elif(SNR>20):
            alpha = 1
        else:
            alpha = 4 - 0.15*SNR

        # Determine beta
        if (SNR<0):
            beta = 0.04
        else:
            beta = 0.01

        for t in range(FRAME_SIZE):
            # Spectrum subtraction
            D = np.abs(DFT[t]) - alpha*noise_spectrum[t]

            if (np.abs(DFT[t]) !=0 and D>beta*noise_spectrum[t]):
                DFT[t] = (D)*DFT[t]/np.abs(DFT[t])
            elif(np.abs(DFT[t]) !=0):
                DFT[t] = (beta*noise_spectrum[t])*DFT[t]/np.abs(DFT[t])

        output[i * FRAME_SIZE : ((i+1) * FRAME_SIZE)] = np.fft.ifft(DFT)
```

```
/Users/lihongyi/anaconda3/lib/python3.6/site-packages/ipykernel_launcher.py:50: ComplexWarning
```

```
In [6]: # Play out the original audio
```

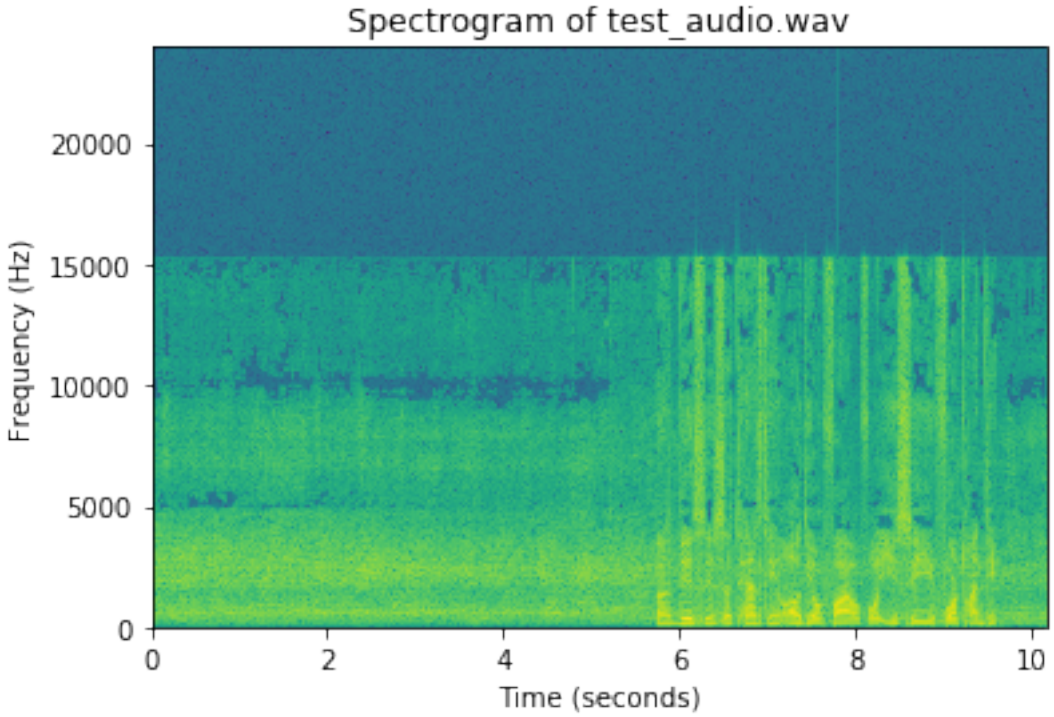
```
        Audio(data, rate=Fs)
```

Out[6]: <IPython.lib.display.Audio object>

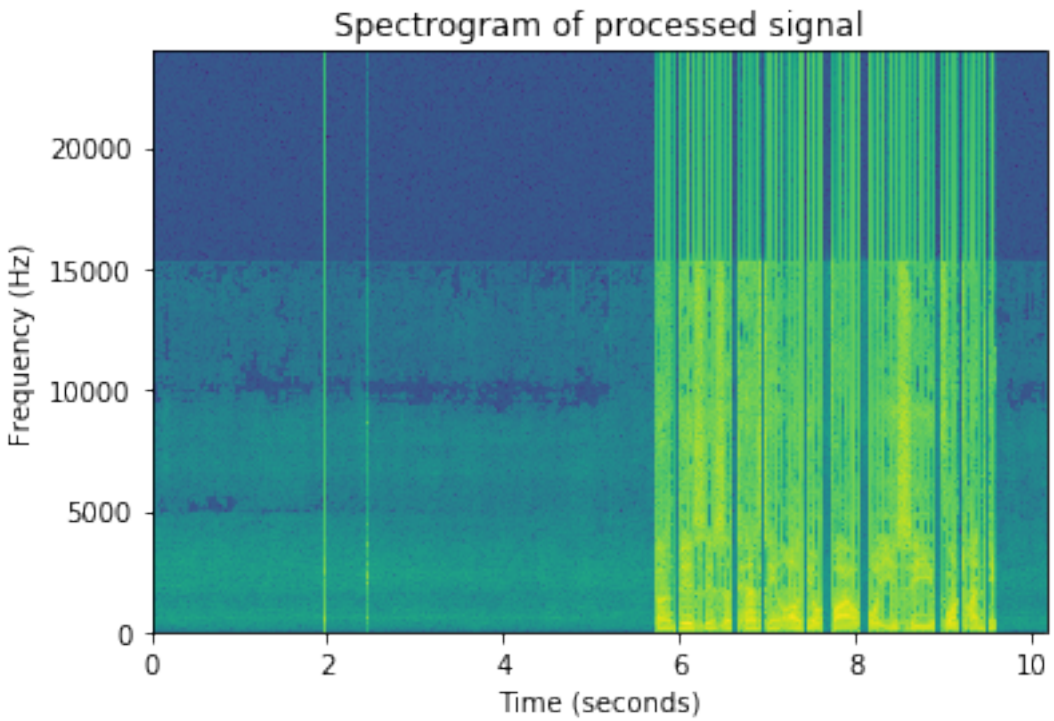
```
In [7]: # Play out the processed audio
        Audio(output, rate=Fs)
```

Out[7]: <IPython.lib.display.Audio object>

```
In [8]: # Plot the spectrogram for original signal
        Pxx, freqs, bins, im = plt.specgram(data, NFFT=1024, Fs=48000, noverlap=512)
        plt.xlabel('Time (seconds)')
        plt.ylabel('Frequency (Hz)')
        plt.title('Spectrogram of test_audio.wav')
        plt.show()
```




```
In [9]: # Plot the spectrogram for processed signal
        Pxx, freqs, bins, im = plt.specgram(output, NFFT=1024, Fs=48000, noverlap=512)
        plt.xlabel('Time (seconds)')
        plt.ylabel('Frequency (Hz)')
        plt.title('Spectrogram of processed signal')
        plt.show()
```


Spectrogram of processed signal
Frequency (Hz)
20000
15000
10000
5000
0
0
2
4
6
8
10
Time (seconds)