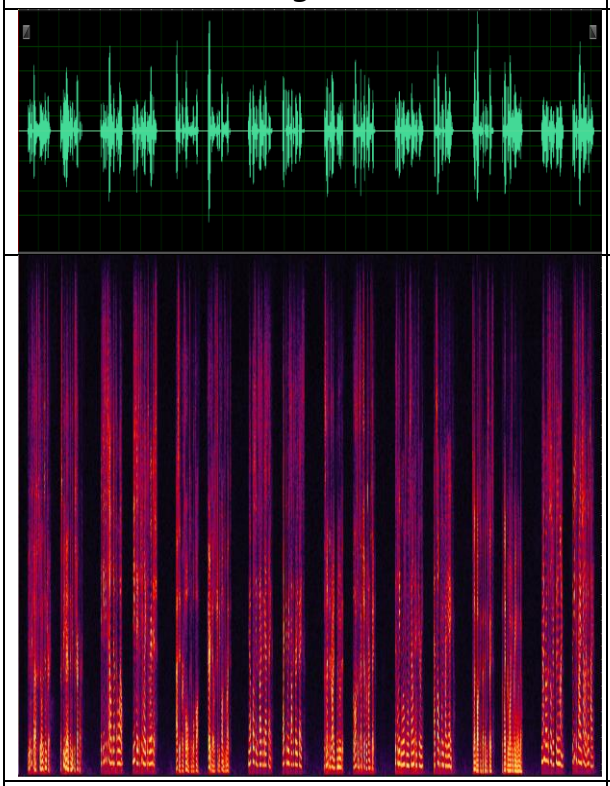
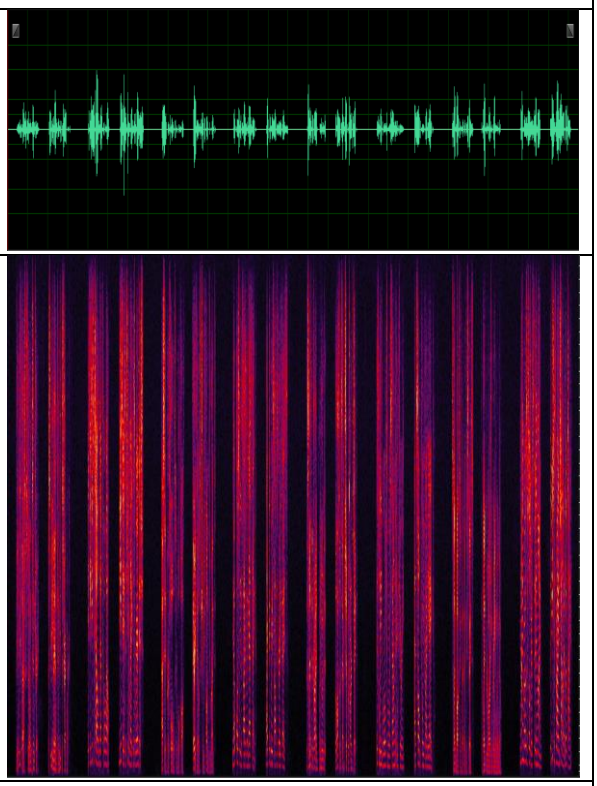
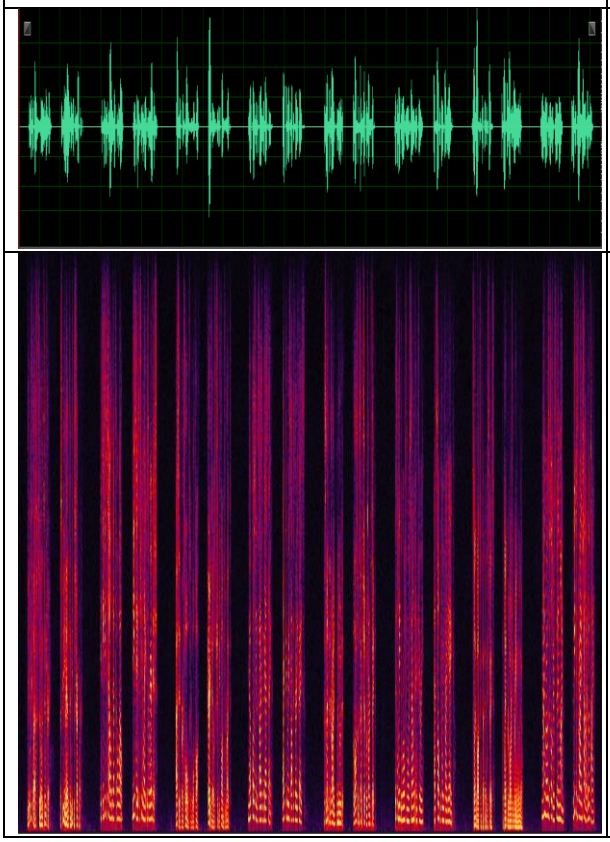
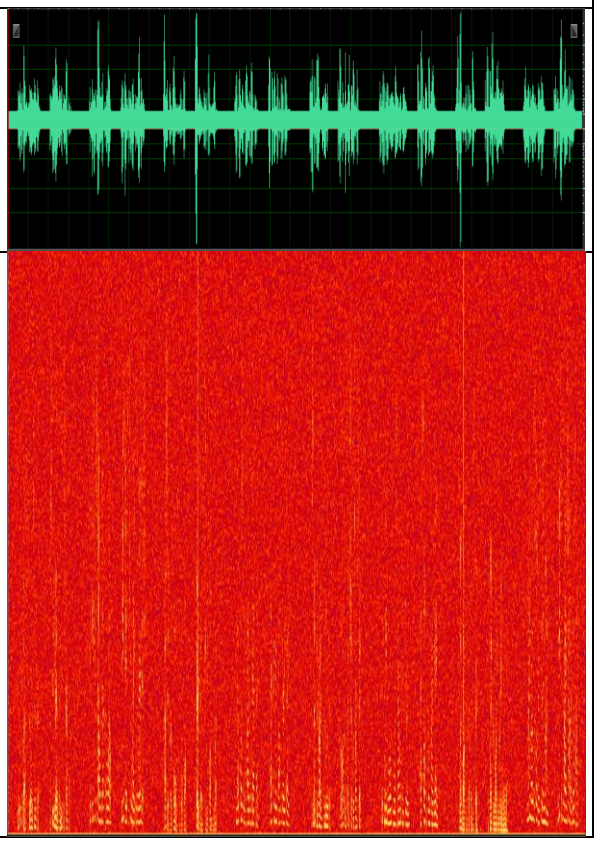
Assignment #3

REPORT

교과명	Digital Speech Processing
교수명	신종원 교수님
제출일	2023.04.26
제출자	AI 대학원 20231050 채종욱

1. Wav 파일 비교

Original	f_10
	
Lattice	**White_noise_lattice**
	

2. 결과분석

1) Original과 f_10만든 wav파일의 결과를 비교하자면, f_10이 Original의 pitch나 harmonic을 어느정도 담아낸 것을 확인할 수 있다. 이는 f_10은 forward prediction error로 원래 신호의 오차 값이기 때문에, 원래 신호의 모습을 닮은 듯 한 것으로 보인다.

2) Original과 Lattice의 wav파일의 결과를 비교하면, 거의 원 신호와 가깝게 복구된 것을 확인할 수 있다. f_10의 파라미터를 이용해서 복원한 것이기 때문에, f_10의 입력 신호를 들어갔던 신호를 복원한 것을 볼 수 있다.

3) White_noise_lattice의 wav파일을 확인해보면 신호 전체에 noise가 깔려있는 것을 볼 수 있다. 이는 입력을 white noise로 주었기 때문에 어쩔 수 없이 생긴 현상으로 보여진다. 하지만 전체적으로 noise가 깔려 있긴 하지만 원래 신호의 pitch나 harmonic성분은 잘 복원한 것으로 보여진다. 실제로 wav파일을 들어보면 전체적으로 noise가 깔려있지만 음성도 들을 수 있는 것을 확인하였다.

3. 코드분석

1) Levinson Durbin & PARCOR

$$
\begin{aligned}
e^{(0)}[n] &= b^{(0)}[n] = s[n], && 0 \le n \le L-1, \\
e^{(i)}[n] &= e^{(i-1)}[n] - k_i b^{(i-1)}[n-1], && 1 \le i \le p, \\
& && 0 \le n \le L-1+i, \\
b^{(i)}[n] &= b^{(i-1)}[n-1] - k_i e^{(i-1)}[n], && 1 \le i \le p, \\
& && 0 \le n \le L-1+i, \\
e[n] &= e^{(p)}[n], && 0 \le n \le L-1+p.
\end{aligned}
\qquad
k_i = \frac{\sum_{m=0}^{L-1+i} e^{(i-1)}[m] \cdot b^{(i-1)}[m-1]}{\left\{ \sum_{m=0}^{L-1+i} \left[e^{(i-1)}[m] \right]^2 \sum_{m=0}^{L-1+i} \left[b^{(i-1)}[m-1] \right]^2 \right\}^{\frac{1}{2}}}.
$$

```
def levinson_durbin(input_data, order):

    e = np.zeros((order+1, len(input_data) + order))
    b = np.zeros((order+1, len(input_data) + order))

    kapa = np.zeros(order+1)

    e[0] = np.concatenate([input_data, np.zeros(order)])
    b[0] = np.concatenate([input_data, np.zeros(order)])

    for m in range(1, order+1):

        # get kapa
        x = np.concatenate([e[m-1, :len(input_data) + m]])
        y = np.concatenate([np.zeros(1), b[m-1, :len(input_data) + m-1]])

        k = np.dot(x, y) / (LA.norm(x) * LA.norm(y))
        kapa[m] = k

        # n = 0, n - 1 < 0 , b[n-1] = 0
        e[m, 0] = e[m-1, 0]
        b[m, 0] = -(kapa[m] * e[m-1, 0])

        # n >= 1
        e[m, 1:len(input_data) + m] = e[m-1, 1:len(input_data) + m] - (kapa[m] * b[m-1][:len(input_data) + m-1])
        b[m, 1:len(input_data) + m] = b[m-1, len(input_data) + m-1] - (kapa[m] * e[m-1,1:len(input_data) + m])

#     print('kapa :', kapa)
#     print('e[10] :', e[10])
#     print('b[10] :' , b[10])

    return e, b, kapa
```

이미지에 나와있는 알고리즘을 반영하여 구현하였다. b같은 경우는 n-1번째 인덱스부터 시작이기 때문에 n보다 작은 경우가 있었는데, 이때는 0으로 처리해주었다. Kapa는 PARCOR를 사용해서 구하였다. 분자의 경우는 내적, 분모의 경우는 L1_norm을 이용해서 구해주었다.

2) Lattice Filter

$$
\begin{aligned}
e^{(p)}[n] &= e[n], && 0 \le n \le L-1+p,\\
e^{(i-1)}[n] &= e^{(i)}[n] + k_i b^{(i-1)}[n-1], && i = p, p-1, \ldots, 1,\\
& && 0 \le n \le L-1+i-1,\\
b^{(i)}[n] &= b^{(i-1)}[n-1] - k_i e^{(i-1)}[n], && i = p, p-1, \ldots, 1,\\
& && 0 \le n \le L-1+i,\\
s[n] &= e^{(0)}[n] = b^{(0)}[n], && 0 \le n \le L-1.
\end{aligned}
$$

```
def lattice_filter(f, b, kapa, order, input_data):
    lattice_e = np.zeros((order+1, len(input_data) + order))
    lattice_b = np.zeros((order+1, len(input_data) + order))

    lattice_e[order] = f.copy()  # 초기값 설정
    lattice_b = b.copy()  # 초기값 설정


    for i in range(order, 0, -1):
        lattice_e[i-1,0] = lattice_e[i,0]
        lattice_b[i,:0] =  - (kapa[i] * lattice_e[i-1,0])

        lattice_e[i-1,1:len(input_data) + i] = lattice_e[i,1:len(input_data) + i] + (kapa[i] * lattice_b[i-1,:len(input_
        lattice_b[i,1:len(input_data) + i] = lattice_b[i-1,len(input_data) + i-1] - (kapa[i] * lattice_e[i-1,1:len(input

    |
#     print('e0', lattice_e[0])
#     print('b0', lattice_b[0])
    s = lattice_e[0][:-order]
    return s
```

Lattice Filter도 이미지에 나와있는 알고리즘을 반영하여 구현하였다. e, b, kapa를 levinson durbin에서 구한 값을 이용해서 처리해주었다. 매개변수 f는 forward error와 white noise를 입력으로 넣어줘서 결과 wav파일을 출력하였다.