ECE 438 - Laboratory 9b
Speech Processing (Week 2)

Last Updated on April 5, 2022

Date:4/13/2023
Section:

Name	Signature	Time spent outside lab
Student Name #1 [Ruixiang Wang]		
Student Name #2 [---%]		

	Below expectations	Lacks in some respect	Meets all expectations
Completeness of the report			
Organization of the report			
Quality of figures: *Correctly labeled with title, x-axis, y-axis, and name(s)*			
Understanding of linear predictive coding (55 pts): *Synthesis of voiced speech: Python plots, questions, Coefficient computation: Python code (* `mylpc` *)*			
Understanding of speech coding and synthesis (45 pts): *Python plots and code, questions*			

In [86]:
```
import sys
import json
import numpy as np
import scipy as scp
import scipy.signal
import matplotlib.pyplot as plt
import soundfile as sf
import IPython.display as ipd
```

In [87]:
```
# make sure the plot is displayed in this notebook
%matplotlib inline
# specify the size of the plot
plt.rcParams['figure.figsize'] = (16, 6)

# for auto-reloading externnal modules
%load_ext autoreload
%autoreload 2
```

```
The autoreload extension is already loaded. To reload it, use:
  %reload_ext autoreload
```

Exercise 1.2: Synthesis of Voiced Speech

1. Use the following code to load three sets of filter coefficients: `A1` , `A2` , and `A3` , respectively, for the vocal tract model in equations (1) and (2). Each vector contains coefficients $\{a_1, a_2, \ldots, a_{15}\}$ for an all-pole filter of order 15.

In [3]:
```
coeff = json.load(open("coeff.json", 'r'))
A1 = np.array(coeff["A1"])
A2 = np.array(coeff["A2"])
A3 = np.array(coeff["A3"])
```

2. Complete the function below to create a length `N` excitation for voiced speech, with a pitch period of `Np` samples. The output vector `x` should contain a discrete-time impulse train with period `Np` (e.g., $[1, 0, 0, \cdots, 0, 1, 0, 0, \cdots]$).

In [8]:
```
def exciteV(N, Np):
    """
    Parameters
    ---
    N: the length of excitation
    Np: pitch period in number of samples

    Returns
    ---
    x: a discrete-time impulse train with period Np
    """
    ctr = Np
    x = np.zeros(N)
    for i in range(N):
        if ctr == Np:
            x[i] = 1
            ctr = 1
        else:
            x[i] = 0
            ctr+= 1
    return x
```

3. Assuming a sampling frequency of 8 kHz (0.125 ms/sample), create a 40 millisecond-long excitation with a pitch period of 8 ms, and filter it using equation (2) for each set of coefficients.

You may use the command:

```
    s = scp.signal.lfilter(np.array([1]), np.insert(-A, 0, 1), x)
```

where `A` is the row vector of filter coefficients.

[`scp.signal.lfilter()` (https://docs.scipy.org/doc/scipy/reference/generated/scipy.signal.lfilter.html)](https://docs.scipy.org/doc/scipy/reference/generated/scipy.signal.lfilter.html) filter data along one-dimension with an IIR or FIR filter.

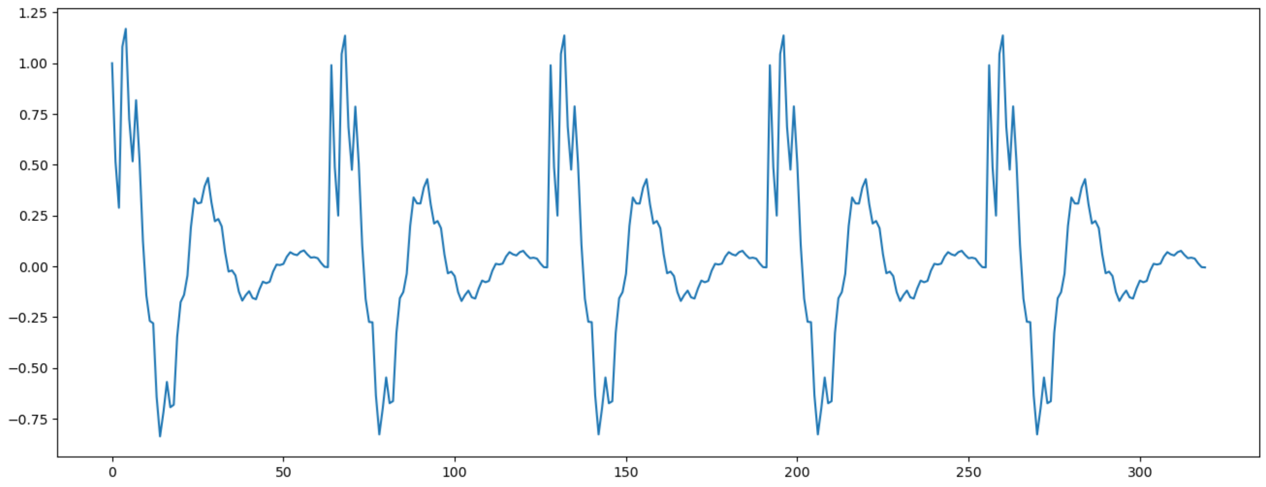
[`np.insert(arr, 0, 1)` (https://numpy.org/doc/stable/reference/generated/numpy.insert.html)](https://numpy.org/doc/stable/reference/generated/numpy.insert.html) insert the value 1 at the beginning of `arr` .

In [40]:
```
x = exciteV(int(40/0.125), int(8/0.125))
s1 = scp.signal.lfilter(np.array([1]), np.insert(-A1, 0, 1), x)
s2 = scp.signal.lfilter(np.array([1]), np.insert(-A2, 0, 1), x)
s3 = scp.signal.lfilter(np.array([1]), np.insert(-A3, 0, 1), x)
```

4. Plot each of the three filtered signals.

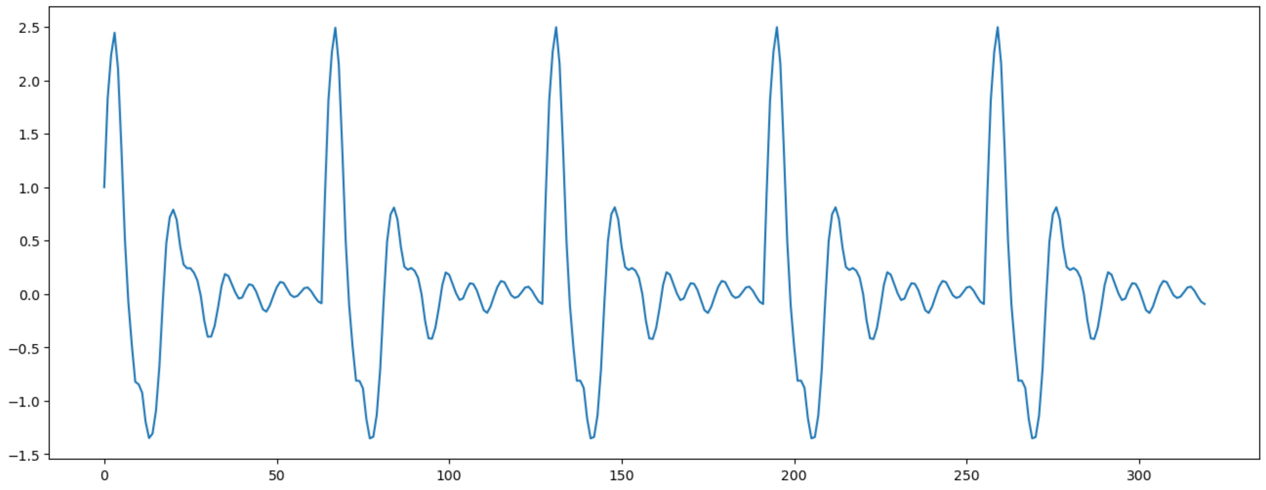
In [41]:
```
plt.plot(s1)
```

Out[41]:
```
[<matplotlib.lines.Line2D at 0x21971aa51f0>]
```



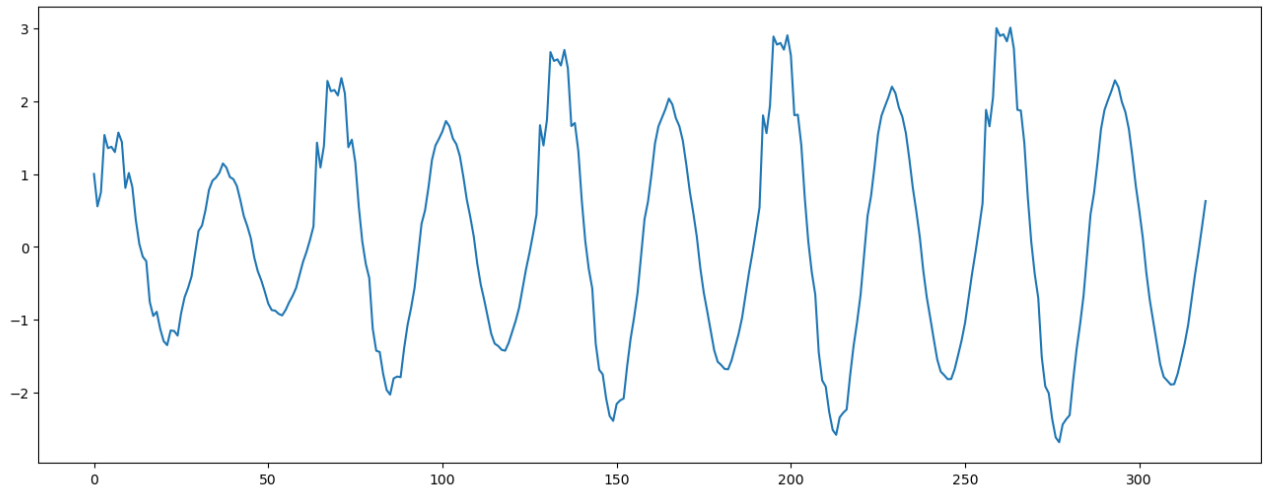
In [42]:
```
plt.plot(s2)
```

Out[42]:
```
[<matplotlib.lines.Line2D at 0x21972066700>]
```



In [43]:
```
plt.plot(s3)
```

Out[43]:
```
[<matplotlib.lines.Line2D at 0x219720c3d30>]
```



5. Use the following command to obtain the frequency response of these filters.

```
w, h = scp.signal.freqz(np.array([1]), np.insert(-A, 0, 1), 512)
```

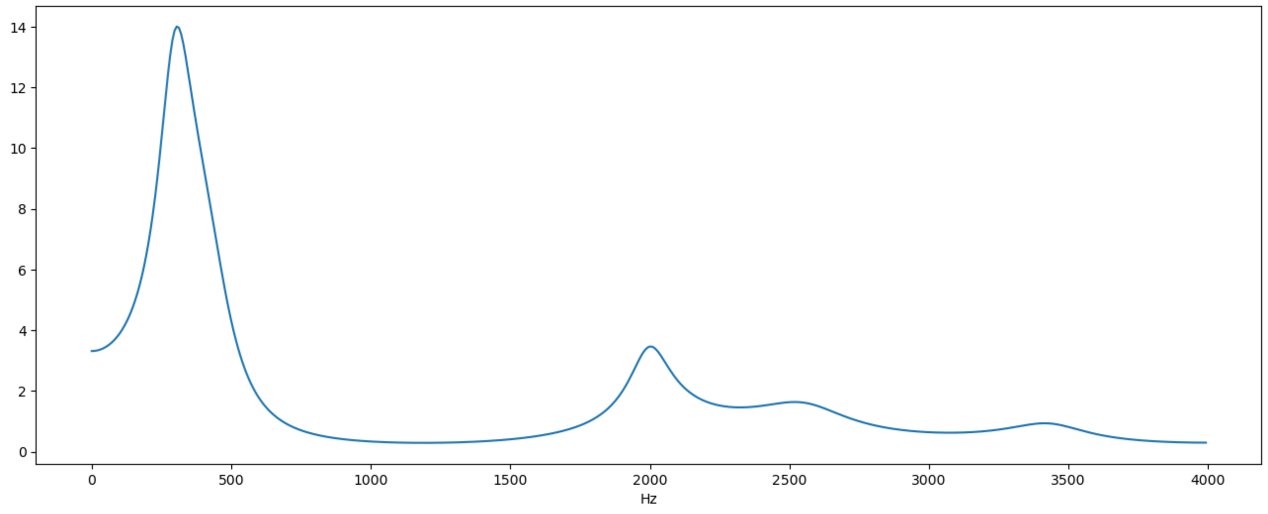
where `A` is the vector of coefficients.

In [44]:
```
w, h1 = scp.signal.freqz(np.array([1]), np.insert(-A1, 0, 1), 512)
w, h2 = scp.signal.freqz(np.array([1]), np.insert(-A2, 0, 1), 512)
w, h3 = scp.signal.freqz(np.array([1]), np.insert(-A3, 0, 1), 512)
```

6. Plot the magnitude of each response versus frequency in Hertz. Make sure to label the frequency axis in units of Hertz.

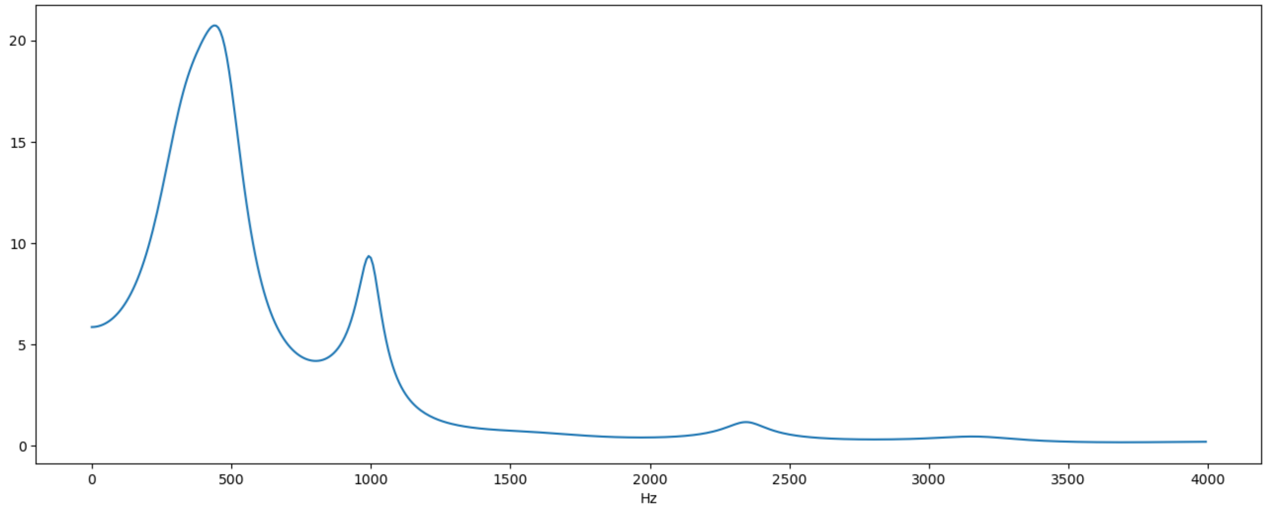
In [45]:
```
f = w/(2*np.pi)*8000
H1 = (np.abs(h1))
plt.plot(f,H1)
plt.xlabel("Hz")
```

Out[45]:
```
Text(0.5, 0, 'Hz')
```



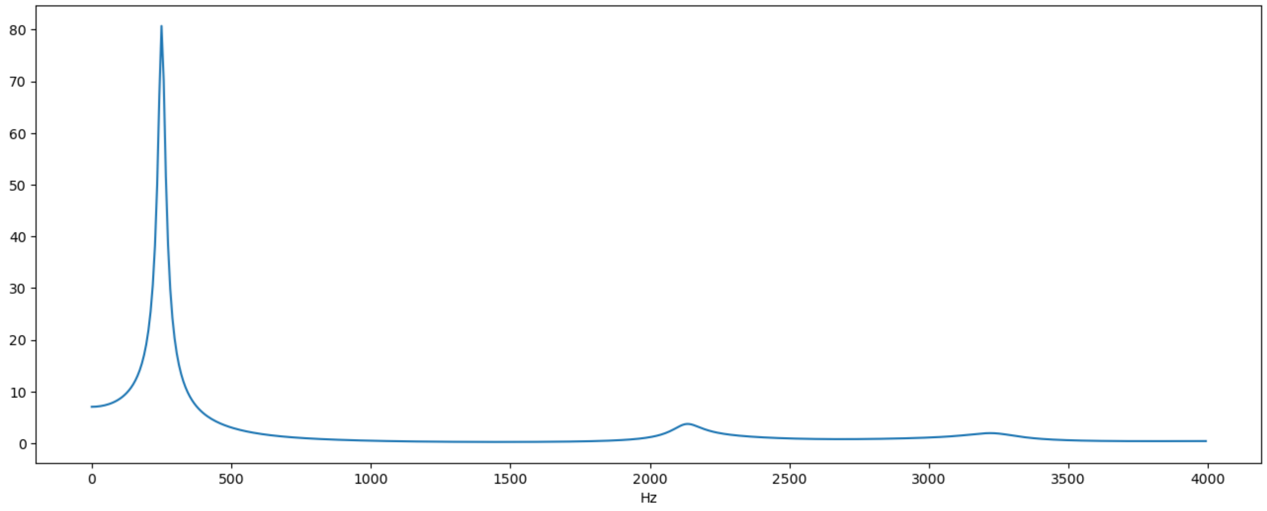
In [46]:
```
H2 = (np.abs(h2))
plt.plot(f,H2)
plt.xlabel("Hz")
```

Out[46]:
```
Text(0.5, 0, 'Hz')
```



In [47]:
```
H3 = (np.abs(h3))
plt.plot(f,H3)
plt.xlabel("Hz")
```

Out[47]:
```
Text(0.5, 0, 'Hz')
```



7. The location of the peaks in the spectrum correspond to the formant frequencies. For each vowel signal, estimate the first three formants (in Hz) and list them.

For A1: [350, 2000, 2500]
For A2: [480, 1000, 2400]
For A3: [250, 2100, 3250]

8. Generate the three signals again, but use an excitation which is 1-2 seconds long. Listen to the filtered signals.

In [55]:
```
x = exciteV(int(1000/0.125), int(8/0.125))
s1 = scp.signal.lfilter(np.array([1]), np.insert(-A1, 0, 1), x)
s2 = scp.signal.lfilter(np.array([1]), np.insert(-A2, 0, 1), x)
s3 = scp.signal.lfilter(np.array([1]), np.insert(-A3, 0, 1), x)

ipd.Audio(s1,rate = 8000)
```

Out[55]:

0:01 / 0:01

In [56]:
```
ipd.Audio(s2,rate = 8000)
```

Out[56]:

0:01 / 0:01

In [57]:
```
ipd.Audio(s3,rate = 8000)
```

Out[57]:

0:01 / 0:01

9. Can you hear qualitative differences in the signals generated in Q8? Can you identify the vowel sounds?

Yes, these signals are different. I suppose they are: a, o ,e

Exercise 2.3: LPC

1. Complete the function below to compute the order-P LPC coefficients for the vector x, using the autocorrelation method. Consider the input vector `x` as a speech segment, in other words do not divide it up into pieces. The output vector `coef` should be a vector containing the `P` coefficients $\{\hat{a}_1, \hat{a}_2, \ldots, \hat{a}_P\}$. In your function you should do the following:

- Compute the biased autocorrelation estimate of equation (17) for the lag values $0 \leq m \leq P$.
- Form the $\mathbf{r}_S$ and $\mathbf{R}_S$ vectors as in (15) and (16). Hint: Use the `scp.linalg.toeplitz()` (https://docs.scipy.org/doc/scipy/reference/generated/scipy.linalg.toeplitz.html) function to form $\mathbf{R}_S$.
- Solve the matrix equation (14) for $\hat{\mathbf{a}}$, which can be solved by inverting the matrix $\mathbf{R}_S$ using `np.linalg.inv()` (https://numpy.org/doc/stable/reference/generated/numpy.linalg.inv.html).

In [79]:
```python
def mylpc(x, P):
    """
    Parameters
    ---
    x: the speech segment
    P: the order

    Returns
    ---
    coef: the order-P LPC coefficients for x
    """
    N = len(x)
    rss = []
    for m in range(P+1):
        rss.append(np.sum(x[n]*x[n+m]/N for n in range(N-m)))

    rs = np.transpose(rss[1:P+1])
    Rs = scp.linalg.toeplitz(rss[:P])

    coef = np.linalg.inv(Rs)@rs
    return coef
```

2. Load `test.json` using the code below. This file contains two vectors: a signal `x` and its order-15 LPC coefficients `a`. Use your function to compute the order-15 LPC coefficients of `x`, and compare the result to the vector `a`.

Note: To check if two vectors are close, use the command below. This function will raise error if the two arrays are not equal up to desired tolerance. In the case below, both the absolute tolerance and the relative tolerance are 10^{-10}.

```python
np.testing.assert_allclose(np.array(coef), np.array(a), atol=1e-10, rtol=1e-10)
```

In [80]:
```python
test = json.load(open("test.json", 'r'))
x = np.array(test['x'])
a = np.array(test['a'])
print(a)
```

```
[ 0.51521099  0.02289244  0.92088278  0.13270476 -0.23130043 -0.79304343
 -0.20397227 -0.14077621  0.52269631  0.04471782  0.12478248 -0.04644767
 -0.06510713 -0.06699434 -0.0288174 ]
```

In [81]:
```
coef = mylpc(x, 15)
np.testing.assert_allclose(np.array(coef), np.array(a), atol=1e-10, rtol=1e-10)
```

```
C:\Users\rxw14\AppData\Local\Temp\ipykernel_20436\3377930730.py:15: DeprecationWarning: Calling np.s
um(generator) is deprecated, and in the future will give a different result. Use np.sum(np.fromiter
(generator)) or the python sum builtin instead.
  rss.append(np.sum(x[n]*x[n+m]/N for n in range(N-m)))
```

Exercise 3.1

1. Load and play the audio file `phrase.au` . This speeech signal is sampled at a rate of 8000 Hz.

In [96]:
```
phrase, fs = sf.read("phrase.au")
ipd.Audio(phrase,rate=8000)
print(len(phrase))
```

```
20001
```

2. Divide the original speech signal into 30ms non-overlapping frames. Place the frames into L consecutive rows of a matrix S (use `np.reshape()` (https://numpy.org/doc/stable/reference/generated/numpy.reshape.html)). If the samples at the tail end of the signal do not fill an entire column, you may disregard these samples.

Hint: Say the original signal is of length `N` , and only the first `M` (is divisible by 10 and is as large as possible) samples are needed, we can calculate `M` easily by

```
M = N // 10 * 10
```

where `//` is the floor division operator in Python.

In [97]:
```
N = int(30/0.125)
L = len(phrase)//N
phrase_new = phrase[:len(phrase)//N * N]

S = np.reshape(phrase_new, (L,N))
```

```
240 83
```

3. For each frame of the original word (i.e., each row of `S`), do the following:

- Compute the energy of each frame of the original word, and place these values in a length L vector called `energy` .
- Determine whether each frame is voiced or unvoiced. Use your zero cross function from the first week to compute the number of zero-crossings in each frame. For length N segments with less than $\frac{N}{2}$ zero-crossings, classify the segment as voiced, otherwise unvoiced. Save the results in a vector `VU` which takes the value of `1` for voiced and `0` for unvoiced.
- Use your `mylpc(x, P)` function to compute order-15 LPC coefficients for each frame. Place each set of coefficients into a column of a $L \times 15$ matrix `A` .

In [137]:
```python
energy = []

for i in range(L):
    energy.append(np.sum(np.multiply(S[i,:],S[i,:]))/N)

cnt = 0
VU = []
for ctr in range(L):
    for i in range(1,N):
        if (S[ctr,i] > 0)and(S[ctr,i-1]<0)or(S[ctr,i] < 0)and(S[ctr,i-1]>0):
            cnt += 1
    if cnt >= N/2:
        VU.append([1])
    else:
        VU.append([0])
    cnt = 0

A = []
print(S[1,:].shape)
for i in range(L):
    A.append(mylpc(S[i,:],15))
A = np.reshape(A, (L,15))
```

```
(240,)

C:\Users\rxw14\AppData\Local\Temp\ipykernel_20436\3377930730.py:15: DeprecationWarning: Calling np.s
um(generator) is deprecated, and in the future will give a different result. Use np.sum(np.fromiter
(generator)) or the python sum builtin instead.
  rss.append(np.sum(x[n]*x[n+m]/N for n in range(N-m)))
```

4. To see the reduction in data, add up the total number of bytes Python uses to store the encoded speech in the arrays `A` , `VU` , and `energy` (use the `sys.getsizeof()` function). Compute the compression ratio by dividing this by the number of bytes Python uses to store the original speech signal. Note that the compression ratio can be further improved by using a technique called vector quantization on the LPC coefficients, and also by using fewer bits to represent the gain and voiced/unvoiced indicator.

In [138]:
```python
byte_sum = sys.getsizeof(A) + sys.getsizeof(VU) + sys.getsizeof(energy)
print(byte_sum)

comp_ratio = byte_sum/sys.getsizeof(phrase)

print(comp_ratio)
```

```
1704
0.010642550214849606
```

5. Now the computed parameters will be used to re-synthesize the phrase using the model in Figure 1. Similar to your `exciteV()` function from Section 1.2, complete the function below that returns a length N excitation for unvoiced speech (generate a `np.random.normal(0, 1)` sequence).

In [139]:
```
def exciteUV(N):
    """
    Parameters
    ---
    N: the length of excitation

    Returns
    ---
    x: the excitation of length N
    """

    x = np.random.normal(0,1,size=(1,N))
    return x
```

6. Initialize an empty NumPy array `output`. Then, for each encoded frame, do the following:

- **Check if current frame is voiced or unvoiced.**
- **Generate the frame of speech by using the appropriate excitation into the filter specified by the LPC coefficients (you did this in Section 1.2). For voiced speech, use a pitch period of 7.5 ms. Make sure your synthesized segment is the same length as the original frame.**
- **Scale the amplitude of the segment so that the synthesized segment has the same energy as the original.**
- **Append `frame` to the end of the `output` array by using `output = np.append(output, frame)`.**

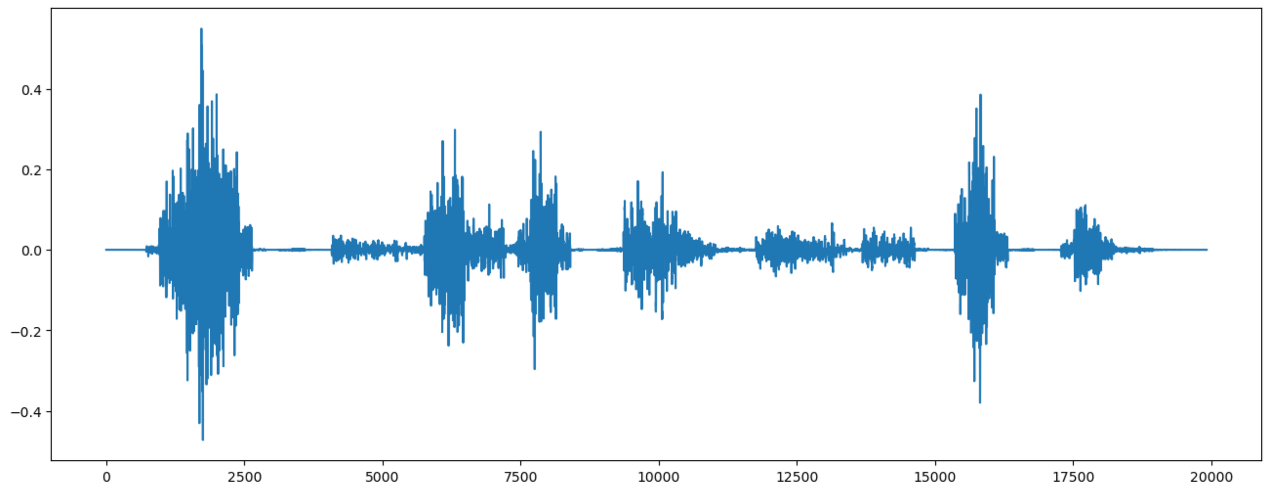
In [140]:
```
output = []

for i in range(L):
    if VU[i] == 1:
        frame = scp.signal.lfilter(np.array([1]), np.insert(-A[i,:], 0, 1), exciteV(N,int(7.5/0.125))
        frame = energy[i]*frame
    else:
        frame = scp.signal.lfilter(np.array([1]), np.insert(-A[i,:], 0, 1), exciteUV(N))
        frame = energy[i]*frame
    output = np.append(output, frame)
```

7. Plot both the original and synthesizsed words.

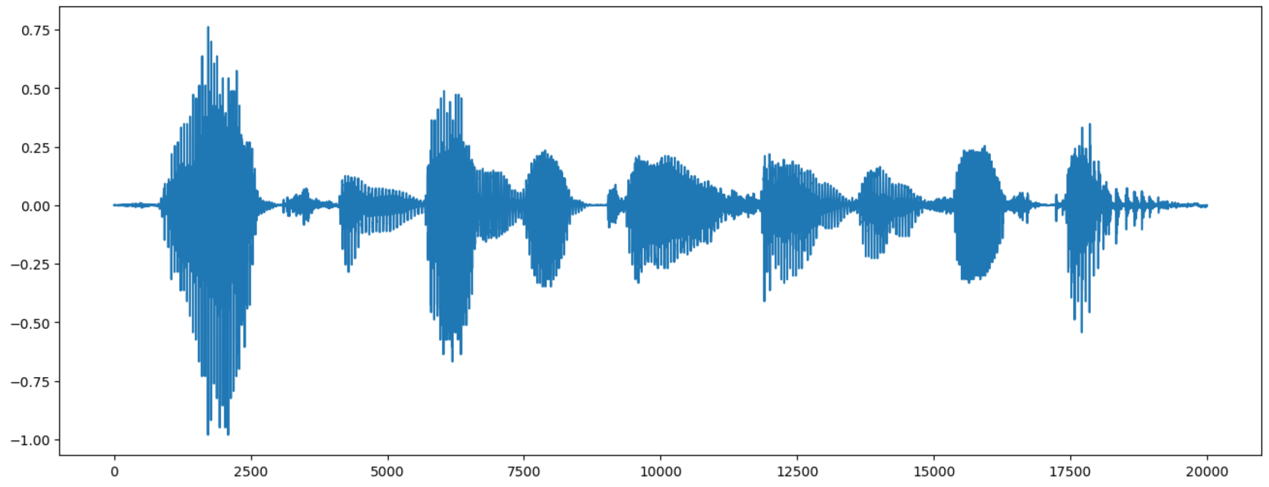
In [141]:
```
plt.plot(output)
```

Out[141]:
```
[<matplotlib.lines.Line2D at 0x21900b9e2e0>]
```



In [142]:
```
plt.plot(phrase)
```

Out[142]:
```
[<matplotlib.lines.Line2D at 0x21900bf8e80>]
```



8. Listen to the original and synthesized phrase. Comment on the quality of your synthesized signal. How might the quality be improved?

In [144]:
```
ipd.Audio(phrase,rate = 8000)
```

Out[144]:

0:02 / 0:02

In [143]:
```
ipd.Audio(output,rate = 8000)
```

Out[143]:

0:02 / 0:02

The synthesized signal sounds demonic. Unvoiced part is not represented well. We can improve the quality by increase the order P or represent unvoiced part using a different method.