Norwegian University of Science and Technology
Department of Electronics and Telecommunications

# TTT4120 Digital Signal Processing - Suggested solution for problem set 8 (Python)

In [1]:
```
import numpy as np
from matplotlib import pyplot as plt
from scipy import signal
import sounddevice as sd
from scipy.io.wavfile import read
```

## Problem 1

### (a).

The filter $H(z)$ is a first order all-pole filter. The output of the filter is therefore an AR[1] process.

## (b)

The first order predictor is defined as

$$\hat{x}(n) = -a_1 x(n-1)$$

The prediction coefficient $-a_1$ can be found by minimizing the prediction error power $\sigma_f^2 = E[f^2(n)]$, where $f(n) = x(n) - \hat{x}(n)$ is the prediction error.

We have that

$$\sigma_f^2 = E[(x(n) - \hat{x}(n))^2] = E[(x(n) + a_1 x(n-1))^2]$$

The optimal value for $a_1$ can be found by minimizing the prediction error power, i.e.

$$\frac{\partial \sigma_f^2}{\partial a_1} = 0$$

Thus, we get

$$E[2(x(n) + a_1 x(n-1))x(n-1)] = 0$$

$$\gamma_{xx}(1) + a_1 \gamma_{xx}(0) = 0 \implies a_1 = -\frac{\gamma_{xx}(1)}{\gamma_{xx}(0)}$$

In the previous problem set the autocorrelation function of the signal $x(n)$ was found to be

$$\gamma_{xx}(m) = \left(-\frac{1}{2}\right)^{|m|}$$

Thus, we get

$$a_1 = \frac{1}{2}$$

By repeating the procedure for the second order predictor, we get

$$a_1 = \frac{1}{2} \text{ and } a_2 = 0$$

This means thet we can not obtain further reduction of the prediction error by using a higher order peredictor.

The above results could be expected, since $x(n)$ is an AR[1] process. The optimal predictor is thus the first order predictor with prediction coefficient equal to the filter coefficient.

# Problem 2

## (a).

This is a MA(1)-process, as only the current and the former value of the input signal are used in forming the output signal.

## (b).

We have

$$x(n)x(n-l) = (w(n) - 0.5w(n-1))(w(n-l) - 0.5w(n-l-1))$$
$$= w(n)w(n-l) - 0.5w(n)w(n-l-1) - 0.5w(n-1)w(n-l) + 0.25w(n-1)w(n$$

Taking the expectation of this leads to

$$\begin{aligned}
\gamma_{xx}(l) &= \gamma_{ww}(l) - 0.5\gamma_{ww}(l+1) - 0.5\gamma_{ww}(l-1) + 0.25\gamma_{ww}(l) \\
&= 1.25\gamma_{ww}(l) - 0.5\left(\gamma_{ww}(l+1) + \gamma_{ww}(l-1)\right) \\
&= 1.25\sigma_w^2\delta(l) - 0.5\sigma_w^2\left(\delta(l+1) + \delta(l-1)\right) \\
&= \begin{cases} 1.25 & l = 0 \\ -0.5 & l = \pm 1 \\ 0 & \text{otherwise.} \end{cases}
\end{aligned}$$

The expression for the power density spectrum can be found as follows

$$\begin{aligned}
\Gamma_{xx}(f) = \sum_{l=-1}^{l} \gamma_{xx}(l)e^{-j\omega l} &= -\frac{1}{2}e^{j\omega} + 1.25 - \frac{1}{2}e^{-j\omega} \\
&= 1.25 - \cos(2\pi f).
\end{aligned}$$

## (c).

The optimal predictor of order $p$ is given by:

$$\hat{x}(n) = -\sum_{k=1}^{p} a_k x(n-k).$$

To obtain simple matrix equations that we can solve using python, we use the version of the Yule-Walker that does not contain \sigma^2_f, i.e.

$$\gamma_{xx}(0)a_1 = -\gamma_{xx}(-1)$$

$$\begin{bmatrix} \gamma_{xx}(0) & \gamma_{xx}(1) \\ \gamma_{xx}(-1) & \gamma_{xx}(0) \end{bmatrix} \begin{bmatrix} a_1 \\ a_2 \end{bmatrix} = \begin{bmatrix} -\gamma_{xx}(-1) \\ -\gamma_{xx}(-2) \end{bmatrix}$$

$$\begin{bmatrix} \gamma_{xx}(0) & \gamma_{xx}(1) & \gamma_{xx}(2) \\ \gamma_{xx}(-1) & \gamma_{xx}(0) & \gamma_{xx}(1) \\ \gamma_{xx}(-2) & \gamma_{xx}(-1) & \gamma_{xx}(0) \end{bmatrix} \begin{bmatrix} a_1 \\ a_2 \\ a_3 \end{bmatrix} = \begin{bmatrix} -\gamma_{xx}(-1) \\ -\gamma_{xx}(-2) \\ -\gamma_{xx}(-3) \end{bmatrix}$$

for order one, two, and three respectively. Then we calculate $\sigma_f^2$ as

$$\sigma_f^2 = \sum_{k=0}^{p} a_k \gamma_{xx}(k)$$

The following code can be used to find the coefficients, and $\sigma_f^2$ for each AR order.

In [2]:

```
Gamma_xx_f = np.array([1.25, -0.5, 0, 0])
R1 = np.array([1.25])
R2 = np.array([[1.25, -0.5], [-0.5, 1.25]])
R3 = np.array([[1.25, -0.5, 0], [-0.5, 1.25, -0.5], [0, -0.5, 1.25]])

a_1 = R1**(-1)*(-Gamma_xx_f[1])
std_1 = np.sum(np.multiply(np.append(1, a_1), Gamma_xx_f[0:2]))
print('----------------------------\nAR(1)')
print(f'Variance: {std_1}')
print(a_1, '\n')

print('----------------------------\nAR(2)')
a_2 = np.linalg.inv(R2)*(-Gamma_xx_f[1:3])
std_2 = np.sum(np.multiply(np.append(1, a_2.T[0]), Gamma_xx_f[0:3]))
print(f'Variance: {std_2}')
print(a_2.T[0], '\n')

print('----------------------------\nAR(3)')
a_3 = np.linalg.inv(R3)*(-Gamma_xx_f[1:4])
std_3 = np.sum(np.multiply(np.append(1, a_3.T[0]), Gamma_xx_f[0:4]))
print(f'Variance: {std_3}')
print(a_3.T[0], '\n')
```

```
----------------------------
AR(1)
Variance: 1.05
[0.4]

----------------------------
AR(2)
Variance: 1.0119047619047619
[0.47619048 0.19047619]

----------------------------
AR(3)
Variance: 1.0029411764705882
[0.49411765 0.23529412 0.09411765]
```

Approximated values are given above. We can see that the mean square error decreases as we increase the model order. This means that the AR model is a better approximation of the MA[1] process.

## (d).

The power density spectrum estimate obtained by using an AR[p] model is given by

$$\hat{\Gamma}_{ff}(f) = \Gamma_{ff}(f)|H(f)|^2 = \sigma_f^2 \left| \frac{1}{A(f)} \right|^2 = \frac{\sigma_f^2}{\left| 1 + \sum_{k=1}^{p} a_k e^{-j2\pi fk} \right|^2}$$

Figure 1 shows the power density spectrum estimates based on models of different order compared to the power density spectrum of the process. It can be seen that the estimates become closer to the correct value as the model order increases. Thus, the model is the best approximation of these three

In [3]:

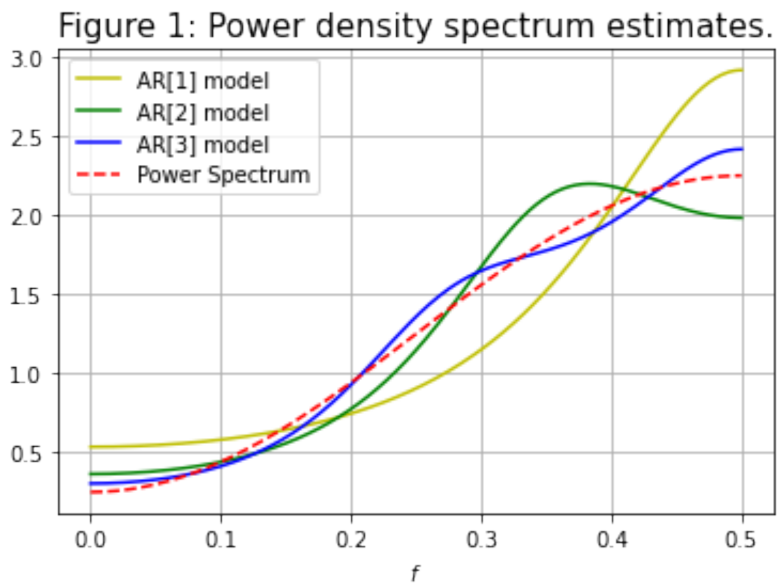
```
#p=1
freq_axis = np.linspace(0, 0.5, 201)
AR1 = a_1*np.exp(-1j*2*np.pi*freq_axis)
power_dens = std_1/np.abs(1+AR1)**2
plt.plot(freq_axis, power_dens, 'y', label = 'AR[1] model')


#p=2
AR2 = a_2.T[0][0]*np.exp(-1j*2*np.pi*freq_axis)
AR2 += a_2.T[0][1]*np.exp(-1j*2*np.pi*freq_axis*2)
power_dens = std_2/np.abs(1+AR2)**2
plt.plot(freq_axis, power_dens, 'g', label = 'AR[2] model')

#p=3
AR3 = a_3.T[0][0]*np.exp(-1j*2*np.pi*freq_axis)
AR3 += a_3.T[0][1]*np.exp(-1j*2*np.pi*freq_axis*2)
AR3 += a_3.T[0][2]*np.exp(-1j*2*np.pi*freq_axis*3)
power_dens = std_2/np.abs(1+AR3)**2
plt.plot(freq_axis, power_dens, 'b', label = 'AR[3] model')

#powerspec calculated
Gamma = 1.25 - np.cos(2*np.pi*freq_axis)
plt.plot(freq_axis, Gamma, 'r--', label = 'Power Spectrum')

plt.legend()
plt.rcParams['figure.figsize'] = [12, 7]
plt.title('Figure 1: Power density spectrum estimates.', fontsize = 15)
plt.grid()
plt.xlabel('$f$')
plt.show()
```



# Problem 3

Here is a recipie on how to solve this task

- Firstly, the vowel sample files are loaded with scipy.io.wavfile.read()
- Then each vowel is modelled as an AR[10] process. The AR coefficients can be found with pysptk.sptk.lpc.
- To transform a vowel $v_i$ into another vowel $v_j$ we need the prediction error signal from $v_i$. For that use signal.lfilter() and the AR coefficients of that vowel. This is the first part of Figure 2:

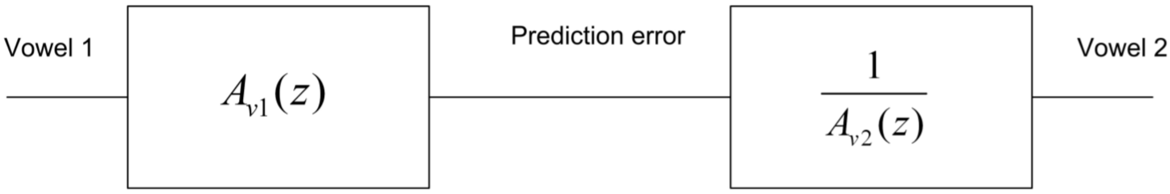



**Figure 2:** Vowel transformation system

- Generate the new vowel $v_j$ by filtering the prediction error using the inverse prediction-error filter (with the coefficients that you found for $v_j$). This is the second part of Figure 2.

- The output signal can be played using sd.play().

The code below is a implementation of this recipie.

In [4]:

```
import time
import pysptk

# firstly, the vowel sample files are loaded
fs, a =read('a.wav')
fs, e =read('e.wav')
fs, i =read('i.wav')
fs, o =read('o.wav')
fs, u =read('u.wav')
fs, y =read('y.wav')
fs, ae =read('ae.wav')
fs, oe =read('oe.wav')
fs, aa =read('aa.wav')

#set the deafult sampleing rate to the same as the recording
sd.default.samplerate = fs


# define which vowels we want to change from and to.
from_vowel = a
to_vowel = i

# Then each vowel is modelled as an AR[10] process. The AR coefficients ca
ak_from = pysptk.sptk.lpc(np.array(from_vowel/max(from_vowel)), order=10)
ak0_from = ak_from[0]
ak_from[0] = 1

ak_to = pysptk.sptk.lpc(np.array(to_vowel/max(to_vowel)), order=10)
ak0_to = ak_to[0]
ak_to[0] = 1

# filtering as described above.
filtered = signal.lfilter(ak_from, [ak0_from], x=from_vowel)
changed = signal.lfilter([ak0_to], ak_to, x=filtered)

# play the result
sd.play(changed/max(changed))
```

## Full proposal of solution

An example of a program that transforms a recording of a vowel into another vowel is given below.

In [10]:

```
# keep the vowels in lists so it is easier to keep track of
vowel_vec_string = ['a', 'e', 'i', 'o', 'u', 'y', 'ae', 'oe', 'aa']
vowel_vec = [a, e, i, o, u, y, ae, oe, aa]


# defining all the functions
def normalize(vec):
    return vec/max(vec)

def find_index_and_model(vok):
    i = vowel_vec_string.index(str(vok))
    coefficients = pysptk.sptk.lpc(np.array(normalize(vowel_vec[i])), order
    return coefficients

def change_vowel(recording):
    # find the AR[10] model of the vowel we are changeing from
    ak = pysptk.sptk.lpc(np.array(normalize(recording)), order=10)
    ak0 = ak[0]
    ak[0] = 1

    # find the AR[10] model of the vowel we are changeing to
    ak_x = change_to_vowel()
    ak_x0 = ak_x[0]
    ak_x[0] = 1

    # filter vowel 1 with its own coeffisient, to get the prediction error
    filtered = signal.lfilter(ak, [ak0], x=recording)

    # filter the prediction error with the inverse model of vowel 2
    changed = signal.lfilter([ak_x0], ak_x, x=filtered)

    return changed

def record(duration):
    print('Recording')
    rec = sd.rec(int(duration*fs), channels=2)
    time.sleep(duration/4)
    print('.')
    time.sleep(duration/4)
    print('.')
    time.sleep(duration/4)
    print('.')
    time.sleep(duration/4)
    print('Done.\n')
    return normalize(rec.T[0])

# ask the user what vowel we shoud shange to
def change_to_vowel():
    print('Choose from this list:', vowel_vec_string)
    vowel = input('Which vowel do you want to change to?\n')
    while (vowel not in vowel_vec_string):
        print('\nERROR: Not in list, please select from the list given abov
        vowel = input('Which vowel do you want to change to?\n')
    return find_index_and_model(vowel)
```

In [11]:

```
recording = record(1.5)
changed_vowel = change_vowel(aa)
sd.play(normalize(changed_vowel))
```

```
Recording
.
.
.
Done.

Choose from this list: ['a', 'e', 'i', 'o', 'u', 'y', 'ae', 'oe', 'aa']
Which vowel do you want to change to?
i
```