Norwegian University of Science and Technology
Department of Electronics and Telecommunications

TTT4120 Digital Signal Processing - Suggested solution for problem set 5 (Python)

In [1]:

```
import numpy as np
from matplotlib import pyplot as plt
from scipy import signal
from scipy import io
from scipy.io.wavfile import read
import sounddevice as sd

%matplotlib inline
plt.rcParams['figure.figsize'] = [10, 5]
```

Problem 1

(a).

We have that $S_{xx}(\omega) = |X(\omega)|^2$. The DTFT of $x(n)$ is found as follows.

$$X(\omega) = \sum_{n=-\infty}^{\infty} x(n)e^{-j\omega n} = \sum_{n=0}^{\infty} a^n e^{-j\omega n} = \sum_{n=0}^{\infty}(ae^{-j\omega})^n = \frac{1}{1-ae^{-j\omega}}$$

Thus, we get

$$S_{xx}(\omega) = |X(\omega)|^2 = X(\omega)X^*(\omega) = \frac{1}{1-ae^{-j\omega}}\frac{1}{1-ae^{j\omega}} = \frac{1}{1+a^2-2a\cos\omega}$$

$$S_{xx}(f) = S_{xx}(\omega)|_{\omega=2\pi f} = \frac{1}{1+a^2-2a\cos(2\pi f)}$$

(b).

Let $l \geq 0$. We get

$$r_{xx}(l) = \sum_{n=-\infty}^{\infty} x(n+l)x(n) = \sum_{n=-\infty}^{\infty} a^{n+l}u(n+l)a^n u(n) = a^l \sum_{n=-\infty}^{\infty} a^{2n} = \frac{a^l}{1-a^2}.$$

Let now $l < 0$. Using the symmetry of the autocorrelation function and the fact that $-l > 0$ we get

$$r_{xx}(l) = r_{xx}(-l) = \frac{a^{-l}}{1-a^2},$$

It follows that for any value of l we have

$$r_{xx}(l) = \frac{a^{|l|}}{1-a^2},$$

We can now calculate $S_{xx}(\omega)$ by taking the DTFT of $r_{xx}(l)$

$$S_{xx}(\omega) = \sum_{l=-\infty}^{\infty} r_{xx}(l)e^{-j\omega l} = \sum_{l=-\infty}^{\infty} \frac{a^{|l|}}{1-a^2}e^{-j\omega l} = \frac{1}{1-a^2}\left(\sum_{l=-\infty}^{-1} a^{-l}e^{-j\omega l} + \sum_{l=0}^{\infty} a^l e^{-j\omega}\right.$$

which is the same as in 1a.

(c).

Plots of $x(n), r_{xx}(l)$, and $S_{xx}(f)$ for different values of a are shown in figure 1-3.

In [2]:

```
aa = [0.5, 0.9, -0.9]
N = 50

x_axis_xn = np.linspace(0, N-1, N)
plt.rcParams['figure.figsize'] = [12, 4]

for i in range(len(aa)):
    a = aa[i]
    fig, (ax1, ax2, ax3) = plt.subplots(1, 3)
    fig.suptitle('Figure '+ str(i+1) + ': $x(n), r_{xx}(l)$, and $S_{xx}(f
    xn = a**x_axis_xn
    x_axis_rxx = np.linspace(-N, N, N+1)
    rxx = a**(abs(x_axis_rxx))/(1-a**2)
    x_axis_Sxx = np.linspace(-0.5, 0.5, 2*N)
    Sxx = 1/(1+a**2-2*a*np.cos(2*np.pi*x_axis_Sxx))
    ax1.stem(x_axis_xn, xn)
    ax1.set_xlabel('$n$')
    ax1.set_ylabel('$x(n)$')
    ax2.stem(x_axis_rxx, rxx)
    ax2.set_xlabel('$l$')
    ax2.set_ylabel('$r_{xx}(l)$)')
    ax3.plot(x_axis_Sxx, Sxx)
    ax2.set_xlabel('$f$')
    ax2.set_ylabel('$S_{xx}(f)$)')
    plt.show()
```

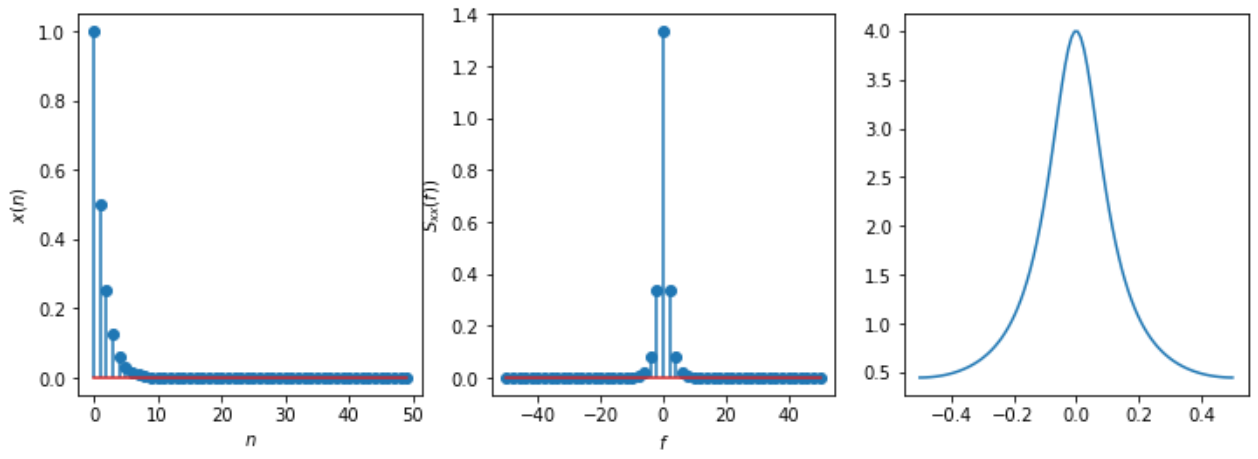


Figure 1: $x(n), r_{xx}(l)$, and $S_{xx}(f)$ when a = 0.5

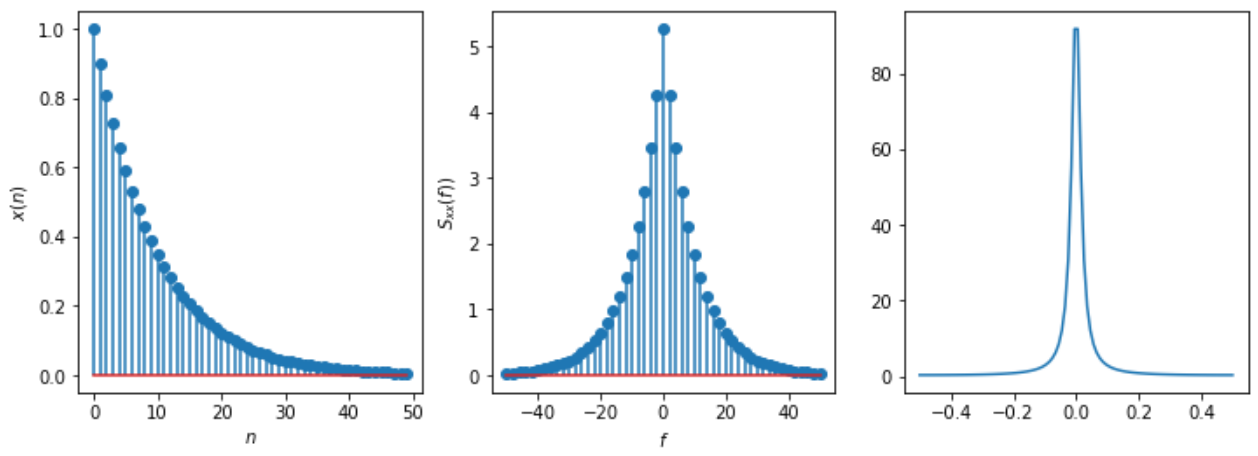


Figure 2: $x(n), r_{xx}(l)$, and $S_{xx}(f)$ when a = 0.9

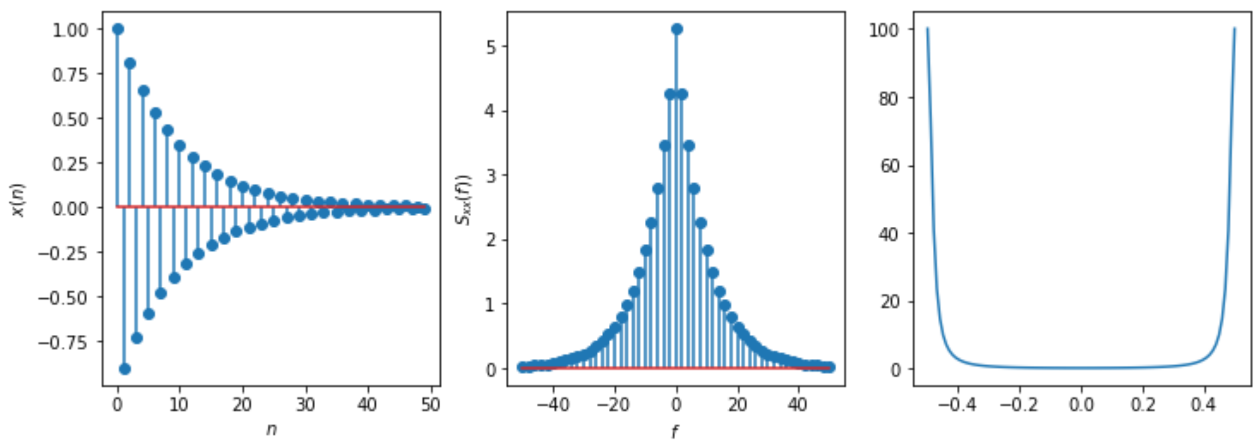


Figure 3: $x(n), r_{xx}(l)$, and $S_{xx}(f)$ when a = -0.9

Let us first compare the plots for $a = 0.5$ and $a = 0.9$. We observe that the signal $x(n)$ varies much faster for $a = 0.5$. The correlation between two samples at distance l from each other is thus much smaller in this case. This can be verified by comparing the autocorrelation functions at some lag l (e.g. $l = 10$). Furthermore, since the signal $x(n)$ varies much faster for $a = 0.5$, it means that it contains higher frequency components. This can be verifyed by comparing the energy spectral densities. It is easily seen that the energy is more concentrated at low frequencies for $a = 0.9$ than for $a = 0.5$.

Let us now compare the plots for $a = 0.9$ and $a = -0.9$. We see that for $a = -0.9$ the signal changes very rapidly for adjacent samples $(x(n+1) \approx -x(n))$, but the signal values at distance 2 from each other are rather similar. The same observation can be made by looking at the autocorrelation functions: the correlation between samples separated by a lag of 2,4,6,... is as high as for the $a = 0.9$, while the correlation between samples at distance 1,3,5,... from each other has a large negative value (it is large because the absolute value of the samples is similar, and negative because of the change in sign). The changes in sign between adjacent samples are quick variations in the signal, which means that the signal contains high frequency components. This is verified by looking at the energy spectral density, where it can be seen that all signal energy is concentrated around high frequencies. The two properties of the autocorrelation function that can be observed from the plots are that $r_{xx}(l)$ is an even function (i.e. $r_{xx}(-l) = r_{xx}(l)$), and that it attains its maximum at $l = 0$.

(d).

The energy of the signal is simply given by $r_{xx}(0)$

$$E_x = \sum_{n=-\infty}^{\infty} x^2(n) = r_{xx}(0) = \frac{1}{1-a^2}$$

(e).

When a signal is filtered, we have the following relation.

$$S_{yy}(f) = |H(f)|^2 S_{xx}(f)$$

The frequency response to the first filter is obtained as follows,

$$H_1(f) = \text{DTFT}\{\delta(n) - a\delta(n-1)\} = 1 - ae^{-j\omega}$$

and

$$|H_1(f)|^2 = (1 - ae^{-j\omega})(1 - ae^{j\omega}) = 1 + a^2 - 2a\cos(2\pi f)$$

Note that $h_1(n)$ is actually the inverse of $x(n)$. If we denote the result of the first filtering stage as $y_1(n)$, then

$$S_{y_1y_1}(f) = |H_1 f|^2 S_{xx}(f) = 1$$

The second filter is lowpass filter with cut-off at $f = 1/4$. Thus, we get

$$S_{yy}(f) = |H_2(f)|^2 S_{y_1y_1}(f) = \begin{cases} \cos^2(2\pi f), & |f| \leq \frac{1}{4} \\ 0, & \frac{1}{4} \leq |f| \leq \frac{1}{2}. \end{cases}$$

Filtering by the lowpass filter removes all frequency components higher than the cut-off frequency. The entire energy of the output signal is thus contained in the frequency range $|f| ≤ \frac{1}{4}. The energy of the output signal can be found using

$$E_y = \int_{-\frac{1}{2}}^{\frac{1}{2}} S_{yy}(f)df = \int_{-\frac{1}{2}}^{\frac{1}{2}} \cos^2(2\pi f)df = \frac{1}{4}$$

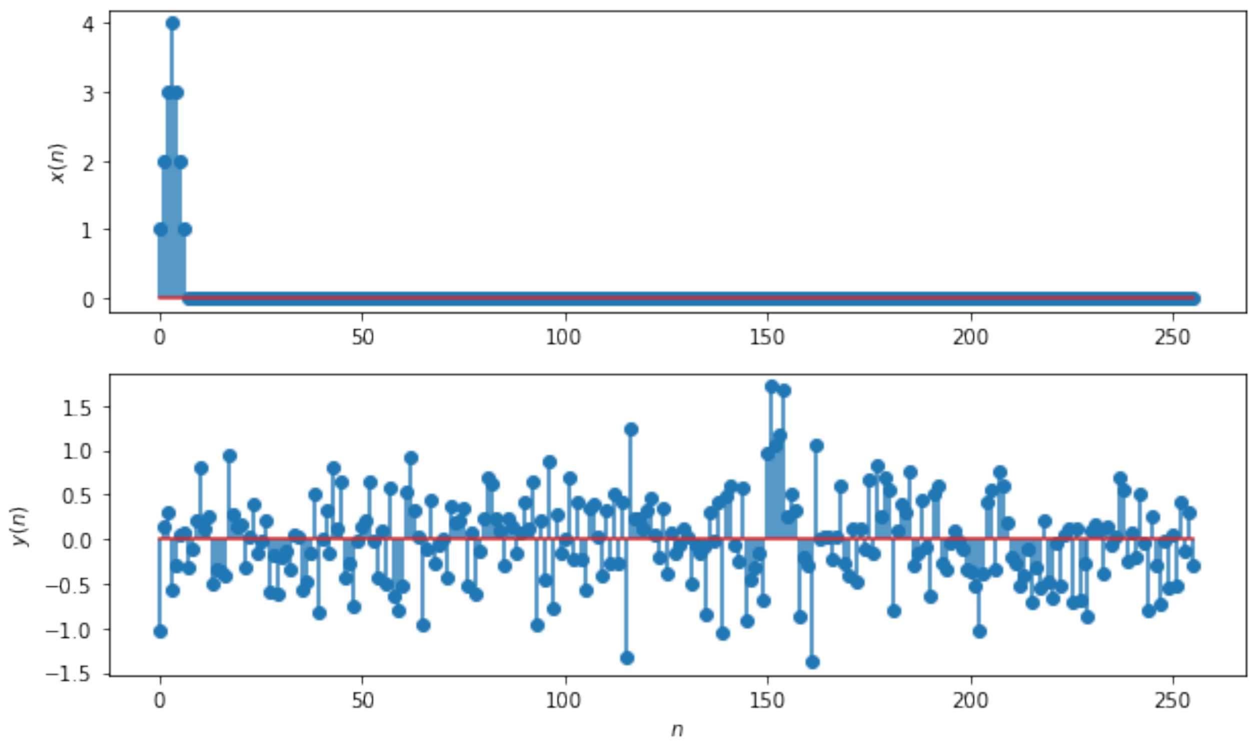
Problem 2

(a).

Plots of the signals $x(n)$ and $y(n)$ are shown in figure 4. Due to the presense of a high noise component in the received signal $y(n)$, it is not easy to conclude wether $y(n)$ also contains a reflection of the signal $x(n)$.

In [4]:
```python
signals = io.loadmat('signals.mat')

x = signals["x"][0]
y = signals["y"][0]

N = len(x)
x_axis = np.linspace(0, N-1, N)

# Plot of the magnitude responses of the two branches
plt.rcParams['figure.figsize'] = [10, 6]
fig, axs = plt.subplots(2)
fig.suptitle('Figure 4: The signals $x(n)$ and $y(n)$. ', fontsize=15)

axs[0].stem(x)
axs[0].set(ylabel = '$x(n)$')
axs[1].set(xlabel = '$n$', ylabel = '$y(n)$')
axs[1].stem(y)
plt.show()
```

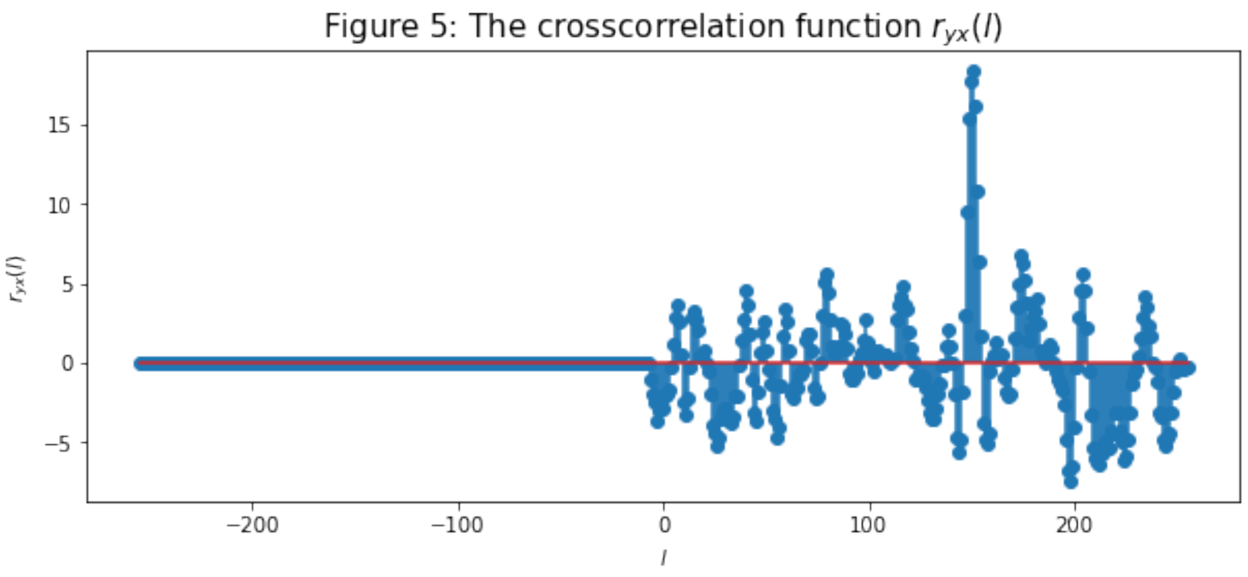
Figure 4: The signals $x(n)$ and $y(n)$.



(b).

Code to generate and plot $r_{yx}(l)$ is given below.

In [5]:
```
corr = signal.correlate(y, x)
len_corr_symmery = int((len(corr)-1)/2)
x_axis_corr = np.linspace(-len_corr_symmery, len_corr_symmery, len(corr))

plt.rcParams['figure.figsize'] = [10, 4]
plt.stem(x_axis_corr, corr)
plt.title('Figure 5: The crosscorrelation function $r_{yx}(l)$', fontsize =
plt.xlabel('$l$')
plt.ylabel('$r_{yx}(l)$')
plt.show()
```



(c).

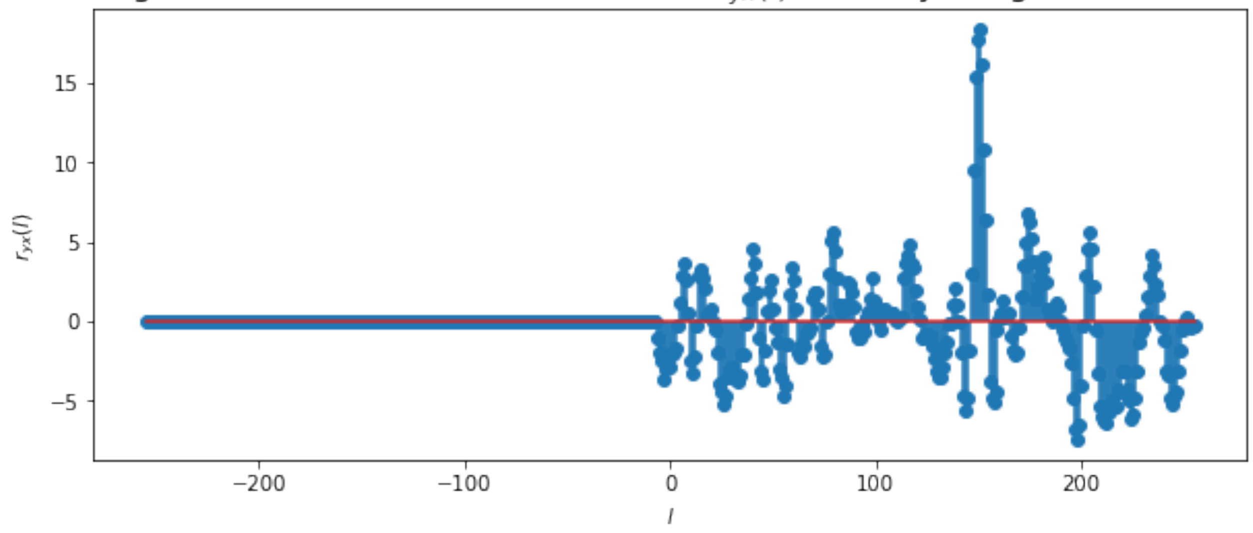
The crosscorrelation function is given by

$$r_{yx}(l) = y(l) * x(-l)$$

The following code can be used to calculate the crosscorrelation function.

In [6]:
```
x_flipped = np.flip(x)
corr = np.convolve(x_flipped, y)

plt.stem(x_axis_corr,corr)
plt.title('Figure 6: The crosscorrelation function $r_{yx}(l)$ found by us
plt.xlabel('$l$')
plt.ylabel('$r_{yx}(l)$')
plt.show()
```

Figure 6: The crosscorrelation function $r_{yx}(l)$ found by using Convolution.



It produces the identical result as the one in 2b).

(d).

The crosscorrelation function has a distinct peak, which suggests that the signal $y(n)$ contain a reflected component of the emitted signal $x(n)$. The crosscorrelation function attains its maximum at $l = 150$, so this value is our best estimate for the delay D.

In [7]:
```
delay = x_axis_corr[np.argmax(corr)]
print('D = ' + str(delay))
```

```
D = 150.0
```

This method is much more reliable than the direct comparison of the plots of the emitted and the received signals.

Problem 3

(a).

We can find the transfer function as:

$$\begin{aligned} Y(z) &= X(z) + \alpha z^{-R} X(z) \\ &= X(z)(1 + \alpha z^{-R}) \end{aligned}$$

$$H(z) = \frac{Y(z)}{X(z)} = 1 + \alpha z^{-R}$$

(b).

The relationship between the delay in seconds, D_s, and the delay R is $D_s = \frac{R}{f_s} = \frac{R}{22050\text{Hz}}$.

(c).

The filter can be implemented the following way:

In [8]:

```
R = 10
alpha = 0.9

b = np.zeros(R)
a = np.zeros(R)
b[0] = 1
b[R-1] = alpha
a[0] = 1

# find the frequency response
w, h = signal.freqz(b, a)

# Plot of the frequency response
plt.rcParams['figure.figsize'] = [10, 6]
fig, axs = plt.subplots(2)
fig.suptitle('Figure 7: Frequency response for single-echo filter, $ R $ =

axs[0].plot(w, abs(h))
axs[0].set(ylabel = 'Magnitude response$')
axs[1].set(xlabel = 'Angular frequency', ylabel = 'Phase response')
axs[1].plot(w, np.angle(h))
plt.show()


# find and plot the impulse response
dlti = signal.dlti(b, a)
t, d_impulse = signal.dimpulse(dlti, n=25)

plt.rcParams['figure.figsize'] = [10, 4]
plt.stem(t, np.squeeze(d_impulse))
plt.title('Figure 8: Impulse response for single-echo filter, $ R $ = ' +
plt.show()
```

Figure 7: Frequency response for single-echo filter, R = 10 and $alpha =$ 0.9

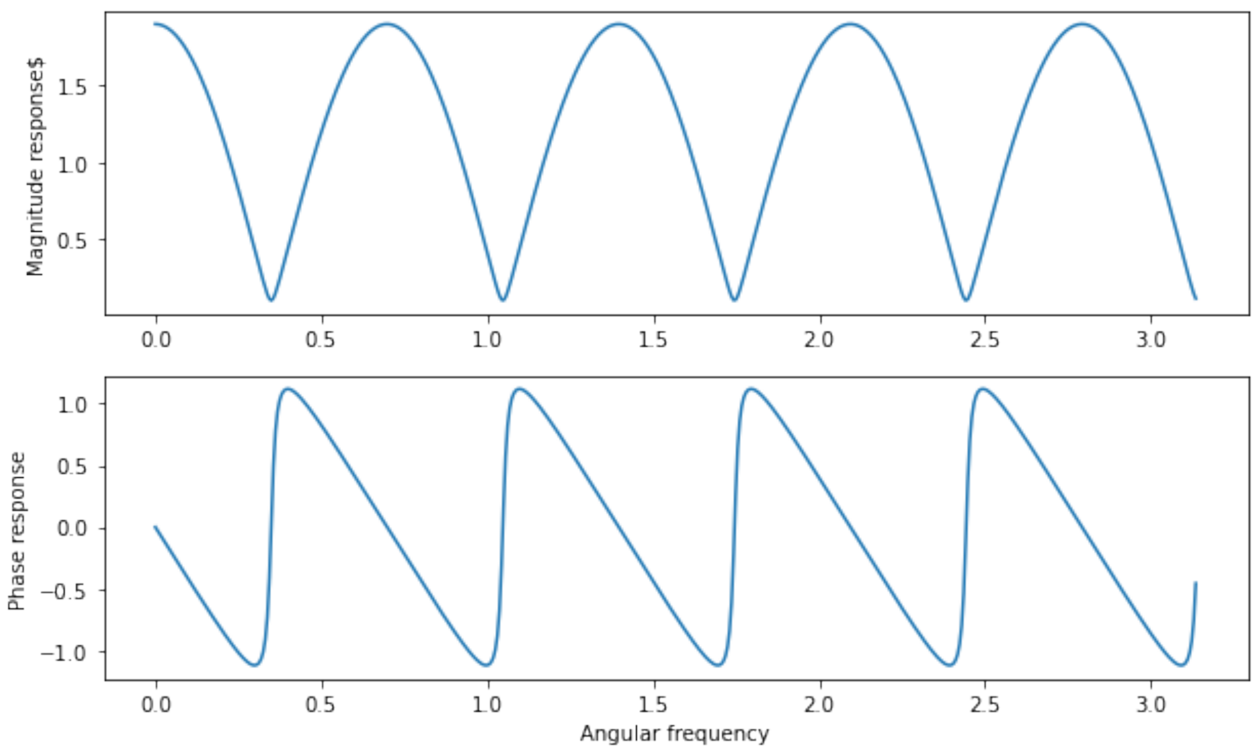
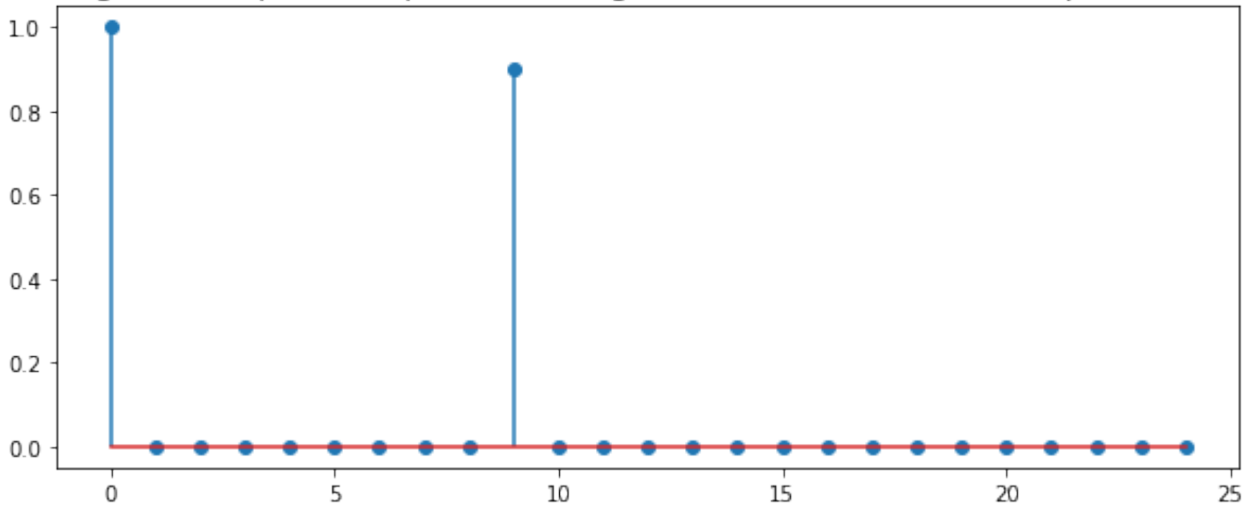


Figure 8: Impulse response for single-echo filter, R = 10 and $alpha =$ 0.9



(d).

We use the filter from (c) to filter the signal. Change the parameters in (c) and run both of the cells to listen to the effect.

In [10]:
```python
fs, x = read('piano.wav')

xscaled = x/np.max(x)
sd.play(xscaled,fs)

print("Playing orginal... ")
input()

y = signal.lfilter(b, a, x)
yscaled = y/np.max(y)
sd.play(yscaled,fs)

print('Playing filtered...')
input()
```

```
Playing orginal...

Playing filtered...
```

Out[10]:
```
''
```

We hear that a larger R and α gives a more pronounced echo effect.

(e).

We find the transfer function as:

$$Y(z) = E(z) - E(z)z^{-NR}\alpha^N$$
$$E(z) = X(z) + E(z)z^{-R}\alpha$$
$$E(z)(1 - z^{-R}\alpha) = X(z) \implies E(z) = \frac{X(z)}{1 - z^{-R}\alpha}$$
$$Y(z) = E(z)(1 - \alpha^N z^{-NR})$$
$$= X(z)\frac{1 - \alpha^N z^{-NR}}{1 - z^{-R}\alpha}$$
$$H(z) = \frac{1 - \alpha^N z^{-NR}}{1 - z^{-R}\alpha}$$

This can be implemented in python as:

In [11]:

```
R = 16
alpha = 0.8
N = 6

b = np.zeros(R*N+1)
a = np.zeros(R*N+1)
b[0] = 1
b[R*N] = -(alpha**N)
a[0] = 1
a[R] = - alpha

# play original sound
sd.play(xscaled,fs)
print("PLaying original... ")
input()

# filter signal, and play the output
y = signal.lfilter(b, a, x)
yscaled = y/np.max(y)
sd.play(yscaled,fs)
print('Playing filtered...')
input()

# frequency response of the filter
w, h = signal.freqz(b, a)
plt.plot(w, abs(h))
plt.title('Figure 9: Magnitude response for multiple-echo filter with $ R 
plt.xlabel('Angular frequency')
plt.ylabel('Magnitude')
plt.show()

plt.plot(w, np.angle(h))
plt.title('Figure 10: Phase response for multiple-echo filter with $ R $ =
plt.xlabel('Angular frequency')
plt.ylabel('Phase')
plt.show()

# impulseresponse
dlti = signal.dlti(b, a)
t, d_impulse = signal.dimpulse(dlti, n=R*N + 1)
plt.stem(t, np.squeeze(d_impulse))
plt.title('Figure 11: Impulse response for multiple-echo filter with $ R $
plt.xlabel('$n$')
plt.ylabel('Amplitude')
plt.show()
```

```
PLaying original...

Playing filtered...
```

Figure 9: Magnitude response for multiple-echo filter with R = 16, N = 6 and $alpha$ = 0.8

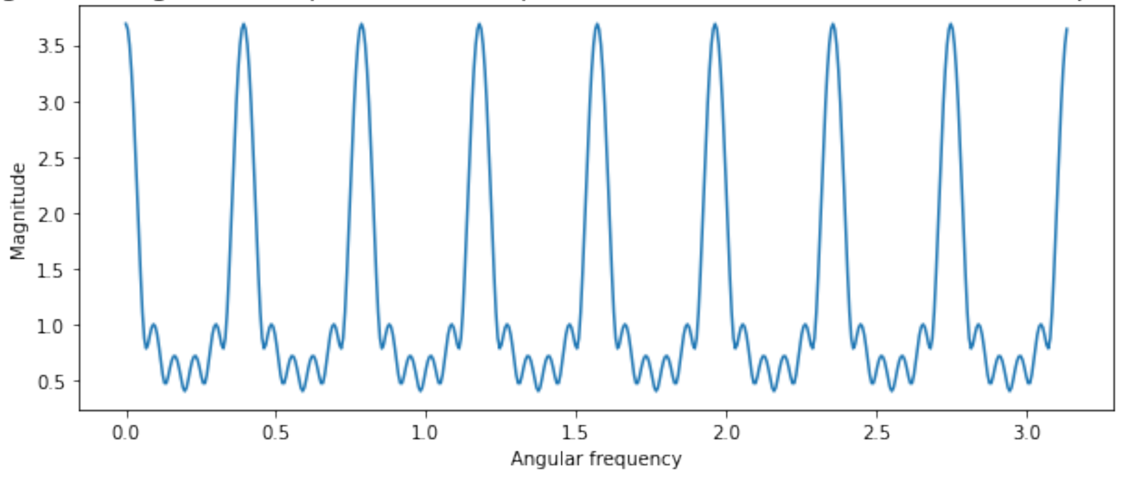


Figure 10: Phase response for multiple-echo filter with R = 16, N = 6 and $alpha$ = 0.8

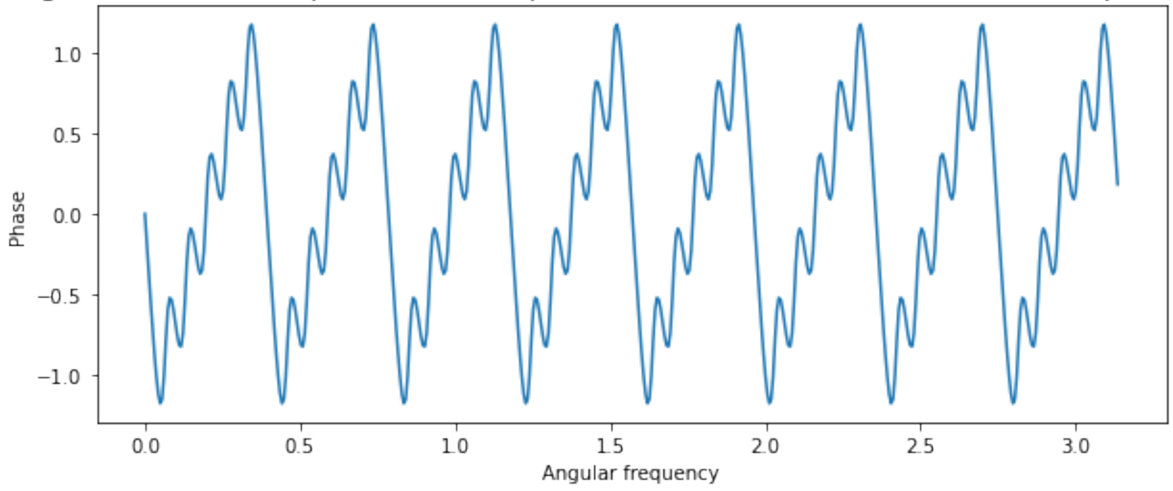
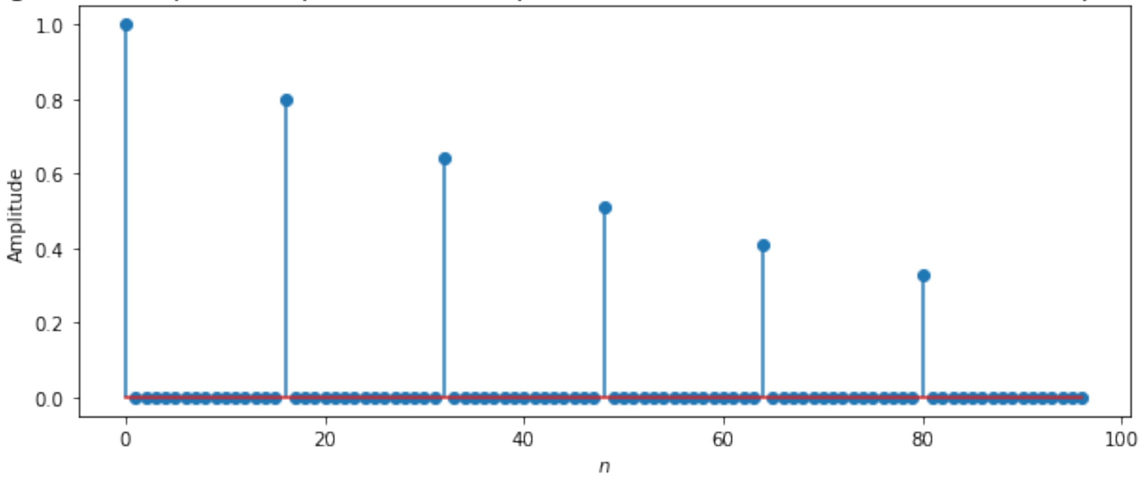


Figure 11: Impulse response for multiple-echo filter with R = 16, N = 6 and $alpha$ = 0.8



(f).

We can in the multiple filter observe that we have multiple spikes in the impulse response. Each spike indicates an echo.

(g).

After filtering the sound will have multiple echoes. This will give the sound a more spacious sound when compared to the original sound file, which only consist of the direct sound.

(h).

In a real room the sound will have multiple echoes from all the walls. Hence the filter with multiple echoes will give the most "natural" sound.