Norwegian University of Science and Technology
Department of Electronics and Telecommunications

TTT4120 Digital Signal Processing - Suggested solution for problem set 4 (Python)

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

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

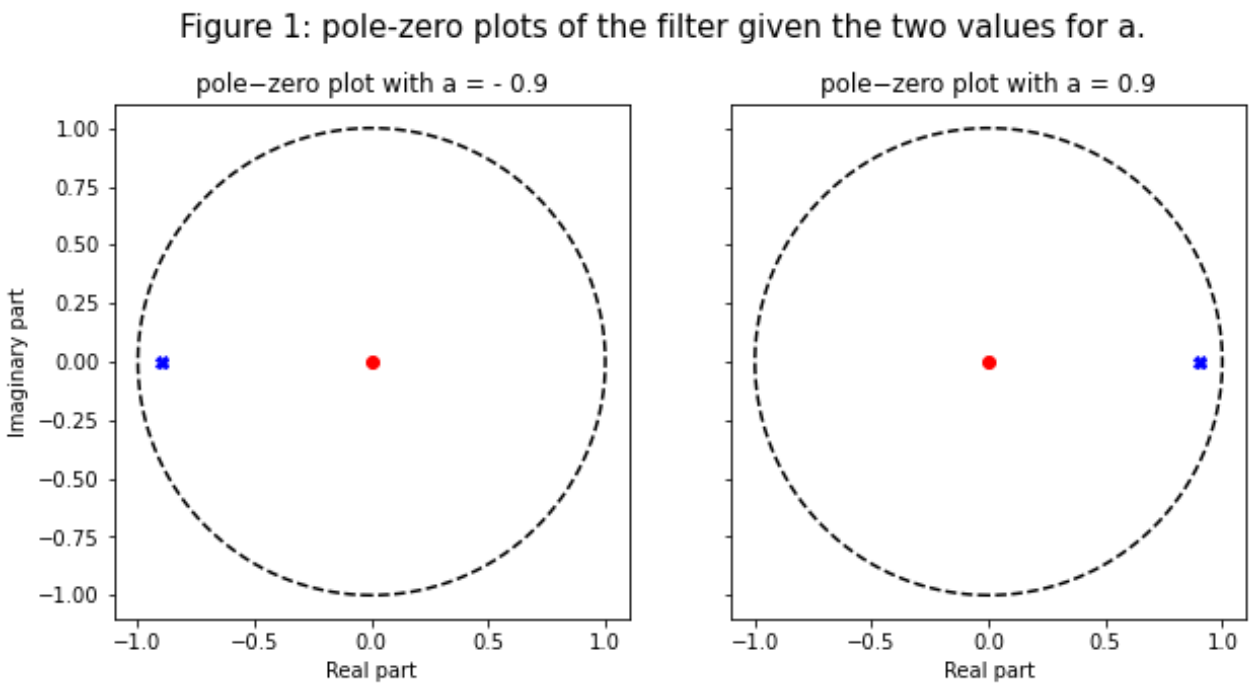
Problem 1

(a).

The pole-zero plots are shown in the figure below.

In [2]:

```
a = -0.9

# find the zeros and the poles for the two cases
poles1 = np.roots([1, -a])
poles2 = np.roots([1, a])
zeros = np.roots([1, 0])

fig, (ax1, ax2) = plt.subplots(1, 2)
fig.suptitle('Figure 1: pole-zero plots of the filter given the two values

# plot circle
theta = np.linspace(-np.pi, np.pi, 1000)
ax1.plot(np.sin(theta), np.cos(theta), '--k')
ax1.set_aspect(1)
ax2.plot(np.sin(theta), np.cos(theta), '--k')
ax2.set_aspect(1)

# plot poles and zeros
ax1.plot(np.real(poles1), np.imag(poles1), 'Xb', label = 'Poles')
ax1.plot(np.real(zeros), np.imag(zeros), 'or', label = 'Zeros')
ax1.set_xlabel('Real part')
ax1.set_ylabel('Imaginary part')
ax1.set_title('pole-zero plot with a = - 0.9')

ax2.plot(np.real(poles2), np.imag(poles2), 'Xb', label = 'Poles')
ax2.plot(np.real(zeros), np.imag(zeros), 'or', label = 'Zeros')
ax2.set_xlabel('Real part')
ax2.label_outer()
ax2.set_title('pole-zero plot with a = 0.9')
plt.show()
```



The frequency response is the z-transform evaluated along the unit circle. The magnitude of the response will be high at frequencies close to the pole, and lower at frequencies further away from the pole. We can then see from the plot that when $a = 0.9$ the filter is a lowpass filter, and when $a = -0.9$ the filter is a highpass filter.

(b).

The program is started by running the Pezdemo.m file (found on blackboard). This can be used to visualize the effect of pole-zero placements on a system's frequency and impulse response and it is only valid for causal filters. Note that the *Add with Conjugate*-button is activated as default.

Problem 2

(a).

The inverse filter is given by

$$H_1 = H^{-1} = (1 - \tfrac{1}{2}z^{-1})(1 + \tfrac{1}{2}z^{-1}) = 1 - \tfrac{1}{4}z^{-2}$$

(b).

Since this filter is an FIR filter, it is stable.

(c).

Since filter $H_1(z)$ has zeros at $z_1 = \frac{1}{2}$ and $z_1 = -\frac{1}{2}$, which is inside the unit circle, the filter is a minimum phase filter.

(d).

For a filter with a zero at z to be linear phase, the filter must also have a zero at $\frac{1}{z}$. The filter has zeros at $z_1 = \frac{1}{2}$ and $z_2 = -\frac{1}{2}$, but not at 2 or -2. Thus, the filter is <u>not</u> linear phase.

Problem 3

(a).

$A(z)$ is allpass because it has pole and zero respectivly in α and α^{-1}.

(b).

By examining the block diagram we see that, for the upper branch,

$$Y_{ub}(z) = \frac{1}{2}[1 + A(z)]X(z)$$

$$H_{ub}(z) = \frac{Y_{ub}(z)}{X(z)} = \frac{1}{2}[1 + A(z)] = \frac{1}{2}[1 + \frac{\alpha - z^{-1}}{1 - \alpha z^{-1}}] = \frac{1}{2}[\frac{1 - \alpha z^{-1} + \alpha - z^{-1}}{1 - \alpha z^{-1}}] = \frac{(1+\alpha) - (1+\alpha)z^{-1}}{2 - 2\alpha z^{-1}} =$$

and for the lower branch,

$$Y_{lb}(z) = \frac{K}{2}[1 - A(z)]X(z)$$

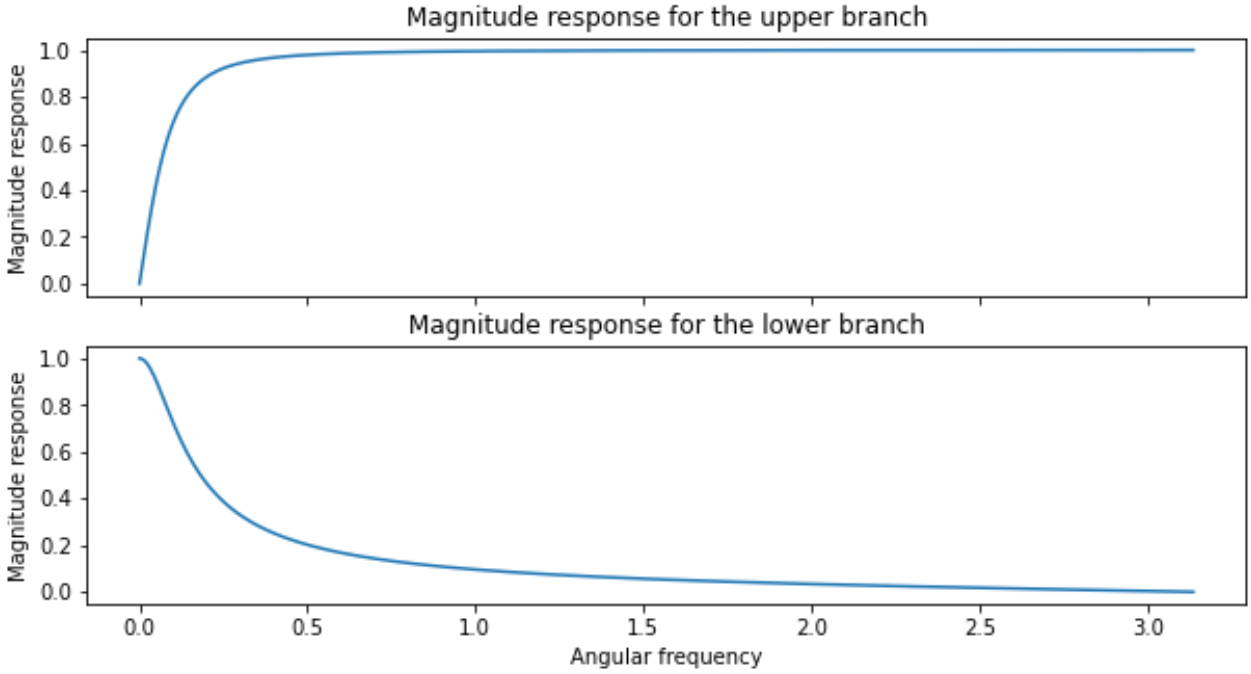
$$H_{lb}(z) = \frac{Y_{lb}(z)}{X(z)} = \frac{K}{2}[1 - A(z)] = \frac{K}{2}[1 - \frac{\alpha - z^{-1}}{1 - \alpha z^{-1}}] = \frac{K}{2}[\frac{1 - \alpha z^{-1} - \alpha + z^{-1}}{1 - \alpha z^{-1}}] = \frac{K(1+\alpha) - K(1+\alpha)}{2 - 2\alpha z^{-1}}$$

Thus, the magnitude response can be calculated and plotted with the following code.

In [3]:

```
# Upper branch
b_ub = [1.9, -1.9]
a_ub = [2, -1.8]
w, h1 = signal.freqz(b_ub, a_ub)

# Lower branch
b_lb = [0.1, 0.1]
a_lb = a_ub
w, h2 = signal.freqz(b_lb, a_lb)

# Plot of the magnitude responses of the two branches
fig, axs = plt.subplots(2)
fig.suptitle('Figure 2: magnutude responses for the two branches. ', fonts

axs[0].plot(w, np.abs(h1))
axs[0].set(ylabel = 'Magnitude response')
axs[0].set_title('Magnitude response for the upper branch')
axs[0].label_outer()
axs[1].set(xlabel = 'Angular frequency', ylabel = 'Magnitude response')
axs[1].set_title('Magnitude response for the lower branch')
axs[1].plot(w, np.abs(h2))
plt.show()
```

Figure 2: magnutude responses for the two branches.



We can see that the upper branch represents a highpass filter, while the lower branch represents a lowpass filter.

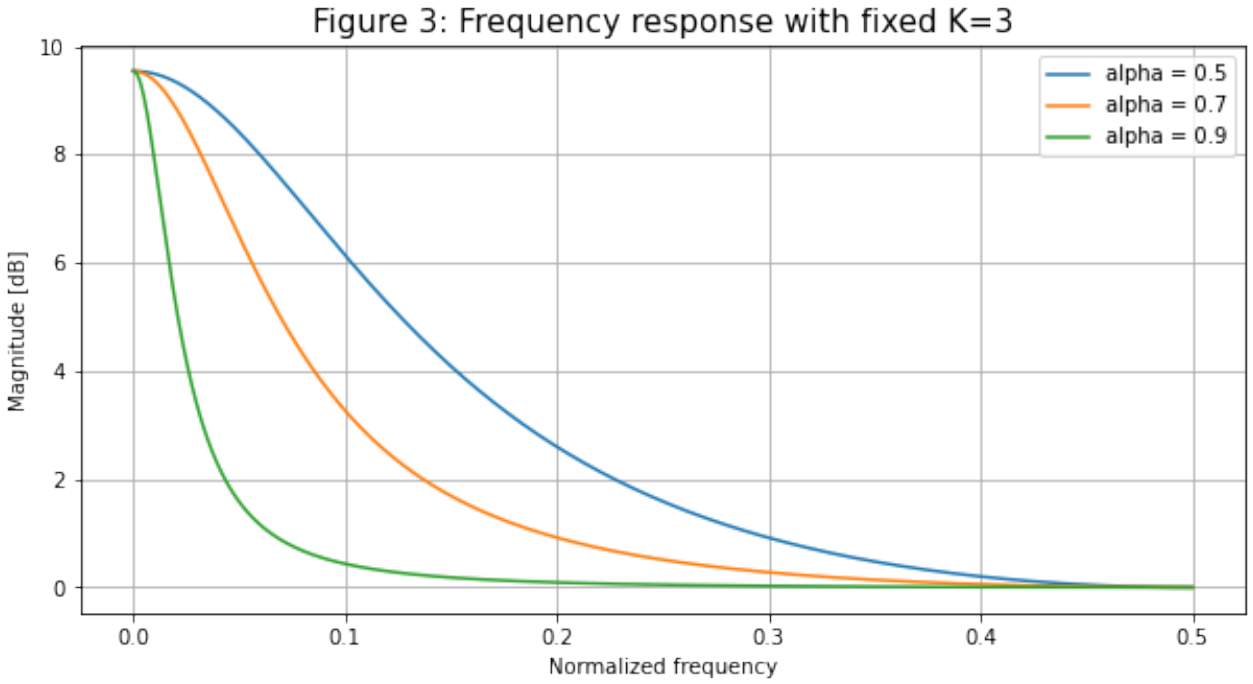
(c).

Figure 3 is a plot of the magnitude response with K fixed at 3 and different αs, while figure 4 is a plot of the magnitude response with α fixed at 0.7, and different values for K .

In [4]:

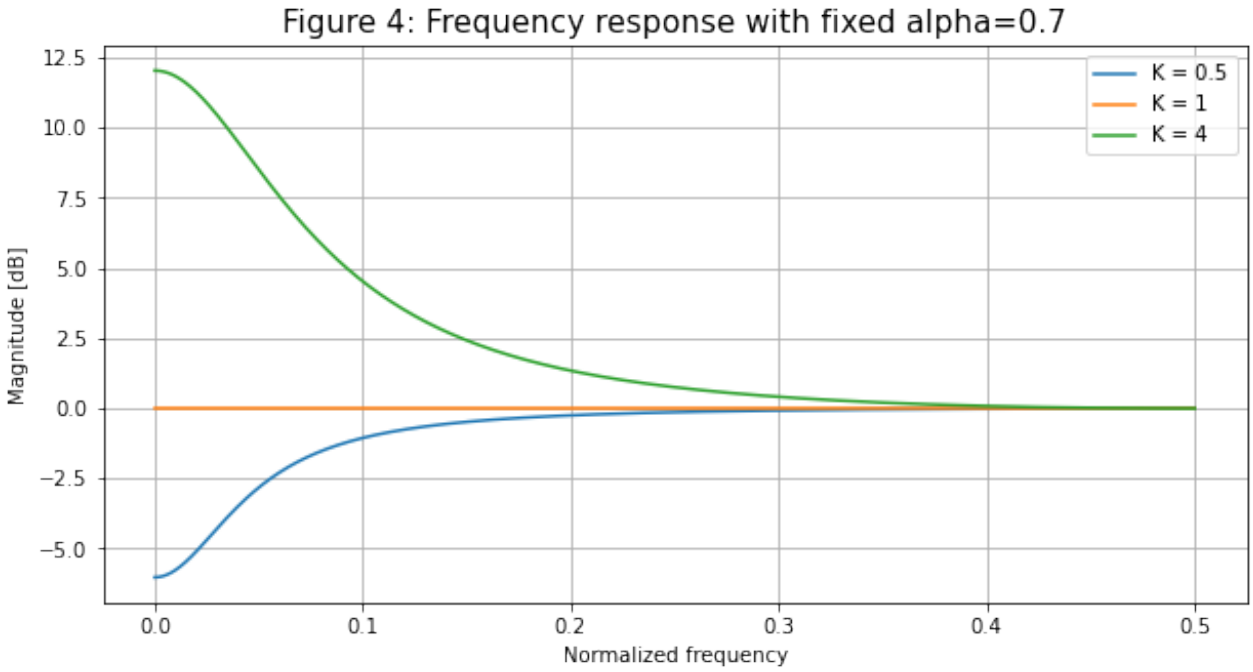
```
# import data
fs,x = read('pluto.wav')

# play original sound
print('Playing *original* sound clip')
xscaled = x/np.max(x)
sd.play(xscaled,fs)

# let K be fixed over three different alphas
K = 3
alphas = [0.5, 0.7, 0.9]

f = plt.figure()
for alpha in alphas:
    # coefficients
    b = [(K/2)*(1-alpha)+(1/2)*(1+alpha), (K/2)*(1-alpha)-(1/2)*(1+alpha)]
    a = [1, -alpha]

    # filtering data
    y = signal.lfilter(b,a,x)

    '''
    # play filtered sound
    print('Playing *filtered* sound clip, alpha = '+ str(alpha))
    yscaled = y/np.max(y)
    sd.play(yscaled,fs);
    input('Press a key to continue...')
    '''

    # plot
    w,h = signal.freqz(b, a, 1024)
    x_axis = np.linspace(0, 0.5, len(h))
    plot = plt.plot(x_axis, 20*np.log10(np.abs(np.real(h))), label = 'alpha

plt.legend()

plt.title('Figure 3: Frequency response with fixed K='+str(K), fontsize =
plt.xlabel('Normalized frequency');
plt.ylabel('Magnitude [dB]');
plt.grid()
plt.show()
```

```
Playing *original* sound clip
```



In [5]:

```
# now, let alpha be fixed over three different Ks
Ks = [0.5, 1, 4]
alpha = 0.7

f = plt.figure()
for K in Ks:
    # coefficients
    b = [(K/2)*(1-alpha)+(1/2)*(1+alpha), (K/2)*(1-alpha)-(1/2)*(1+alpha)]
    a = [1, -alpha]

    # filtering data
    y = signal.lfilter(b,a,x)

    '''
    # play filtered sound
    print('Playing *filtered* sound clip, K = '+ str(K))
    yscaled = y/np.max(y)
    sd.play(yscaled,fs);
    input('Press a key to continue...')
    '''

    # plot
    w,h = signal.freqz(b, a, 1024)
    x_axis = np.linspace(0, 0.5, len(h))
    plot = plt.plot(x_axis, 20*np.log10(np.abs(np.real(h))), label = 'K =

plt.legend()
plt.title('Figure 4: Frequency response with fixed alpha='+str(alpha), font
plt.xlabel('Normalized frequency');
plt.ylabel('Magnitude [dB]');
plt.grid()
plt.show()
```



As we can see from Figure 3 and 4, K is the gain of the lower branch (the lowpass filter). By adjusting K we regulate the contribution from the lowpass filter relative to the contribution from the highpass filter. Thus, K controls the boost or cut at low frequencies. The parameter α controls the boost or cut bandwidth.

Problem 4

(a).

Plot of the amplitude spectra for $G(f)$ and $D(f)$ are given below and shown in figure 5 and 6.

In [6]:

```
L = 500
fx = 0.04
fy = 0.1
A = 0.25

x_axis = np.linspace(0, L-1, L)

# generate the sequences
dn =  [A*(np.cos(2*fx*np.pi*x)+ np.cos(2*np.pi*fy*x)) for x in range(L)]
en = np.random.normal(size=L)
gn = dn + en


# plot dn and gn
fig, axs = plt.subplots(2)
fig.suptitle('Figure 5: plots of d[n] and d[n].', fontsize=15)

axs[0].plot(x_axis, dn)
axs[0].set( ylabel = '$d[n]$')

axs[1].plot(x_axis, gn)
axs[1].set(xlabel = 'n', ylabel = '$g[n]$')
plt.show()


# find the magnitudespectrums and plot them
fig, axs = plt.subplots(2)
fig.suptitle('Figure 6: plots of D(f) and G(f).', fontsize=15)

fft = np.fft.fft(dn)
fft_axis = np.linspace(0, 0.5, int(len(fft)/2))
axs[0].plot(fft_axis, np.abs(fft[:int(len(fft)/2)].real))
axs[0].set(ylabel = '$|D(f)|$')

fft = np.fft.fft(gn)
axs[1].plot(fft_axis, np.abs(fft[:int(len(fft)/2)].real))
axs[1].set(xlabel = 'f', ylabel = '$|G(f)|$')
plt.show()
```

Figure 5: plots of d[n] and d[n].

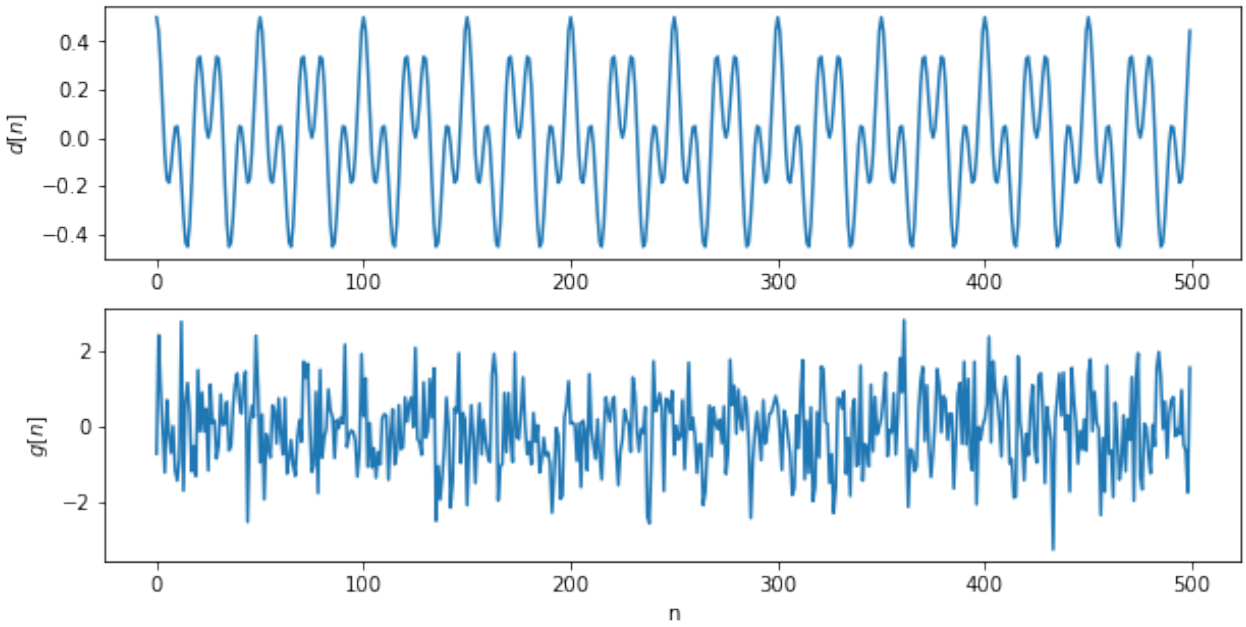
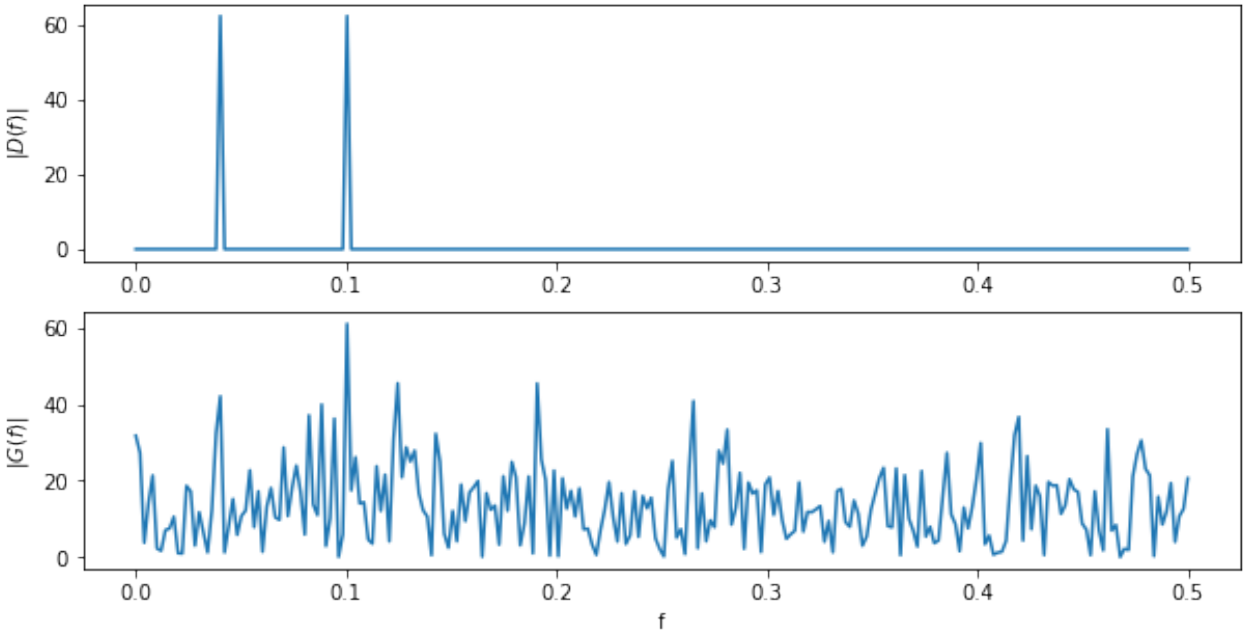


Figure 6: plots of D(f) and G(f).



(b).

We choose poles with a distance of 0.01 to the unit circle. This gives us the transfer functions of the digital resonators given by,

$$H_x(z) = \frac{(1 - z^{-1})(1 + z^{-1})}{(1 - 0.99e^{j2\pi f_x} z^{-1})(1 - 0.99e^{-j2\pi f_x} z^{-1})}$$

$$H_y(z) = \frac{(1 - z^{-1})(1 + z^{-1})}{(1 - 0.99e^{j2\pi f_y} z^{-1})(1 - 0.99e^{-j2\pi f_y} z^{-1})}$$

The following code generates plots of zeros and poles, and amplitude responses of the resonators. These plots are shown in figure 7 and 8. From $|H_x(f)|$ and $|H_y(f)|$ in figure 8, we can see peaks at f_x and f_y respectively. This means that the resonators will remove all frequency components from a signal except components close to f_x and f_y.

In [7]:

```
b_x = [1, 0, -1]
a_x = np.poly([0.99*np.e**(1j*2*np.pi*fx),0.99*np.e**(-(1j*2*np.pi*fx))])

b_y = [1, 0, -1]
a_y = np.poly([0.99*np.e**(1j*2*np.pi*fy),0.99*np.e**(-(1j*2*np.pi*fy))])

# find the zeros and the poles for the two cases
poles_x = np.roots(a_x)
zeros_x = np.roots(b_x)
poles_y = np.roots(a_y)
zeros_y = np.roots(b_y)


fig, (ax1, ax2) = plt.subplots(1, 2)
fig.suptitle('Figure 7: pole-zero plots ...', fontsize=15)

# plot circle
theta = np.linspace(-np.pi, np.pi, 1000)
ax1.plot(np.sin(theta), np.cos(theta), '--k')
ax1.set_aspect(1)
ax2.plot(np.sin(theta), np.cos(theta), '--k')
ax2.set_aspect(1)

# plot poles and zeros
ax1.plot(np.real(poles_x), np.imag(poles_x), 'Xb', label = 'Poles')
ax1.plot(np.real(zeros_x), np.imag(zeros_x), 'or', label = 'Zeros')
ax1.set_xlabel('Real part')
ax1.set_ylabel('Imaginary part')
ax1.set_title('Zeros and poles for $|H_x(z)|$.')

ax2.plot(np.real(poles_y), np.imag(poles_y), 'Xb', label = 'Poles')
ax2.plot(np.real(zeros_y), np.imag(zeros_y), 'or', label = 'Zeros')
ax2.set_xlabel('Real part')
ax2.label_outer()
ax2.set_title('Zeros and poles for $|H_y(z)|$.')
plt.show()


# plot of the magitude response of the two resonators
w_x, h_x = signal.freqz(b_x, a_x)
plot = plt.plot(w_x/(2*np.pi), np.abs(h_x), label = '$|H_x(f)|$')
w_y, h_y = signal.freqz(b_y, a_y)
plot = plt.plot(w_y/(2*np.pi), np.abs(h_y), label = '$|H_y(f)|$')
plt.legend()
plt.title('Figure 8: Plot of $|H_x(f)|$ and $|H_y(f)|$.', fontsize = 15)
plt.xlabel('$f$')
plt.show()
```

Figure 7: pole-zero plots ...

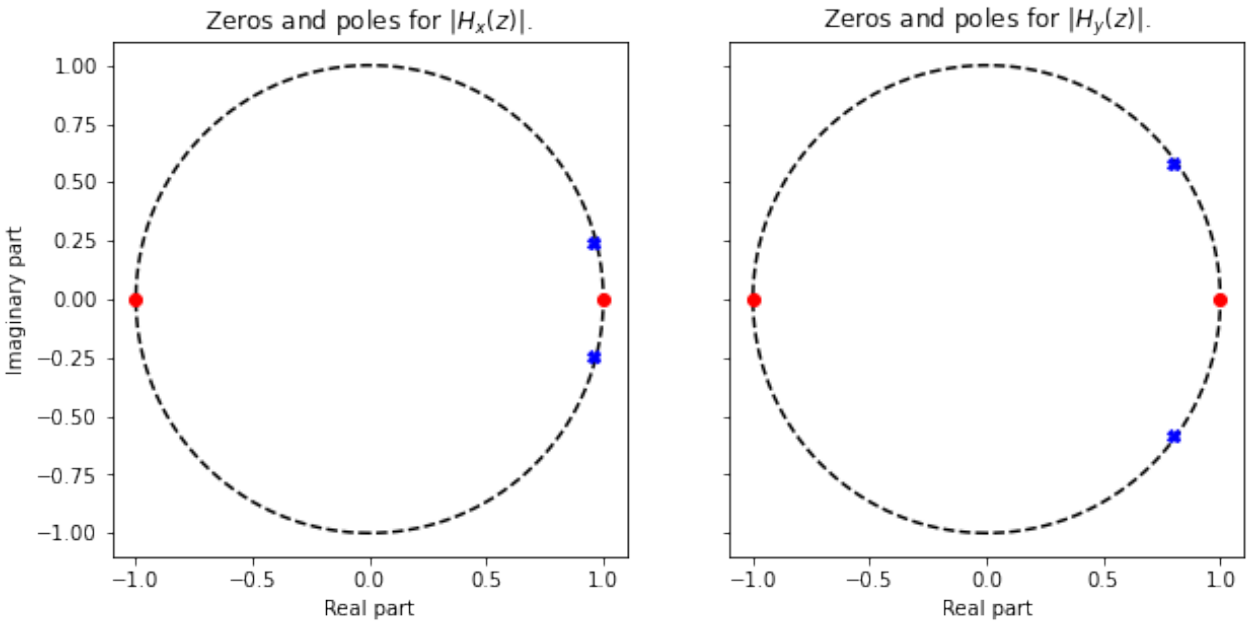
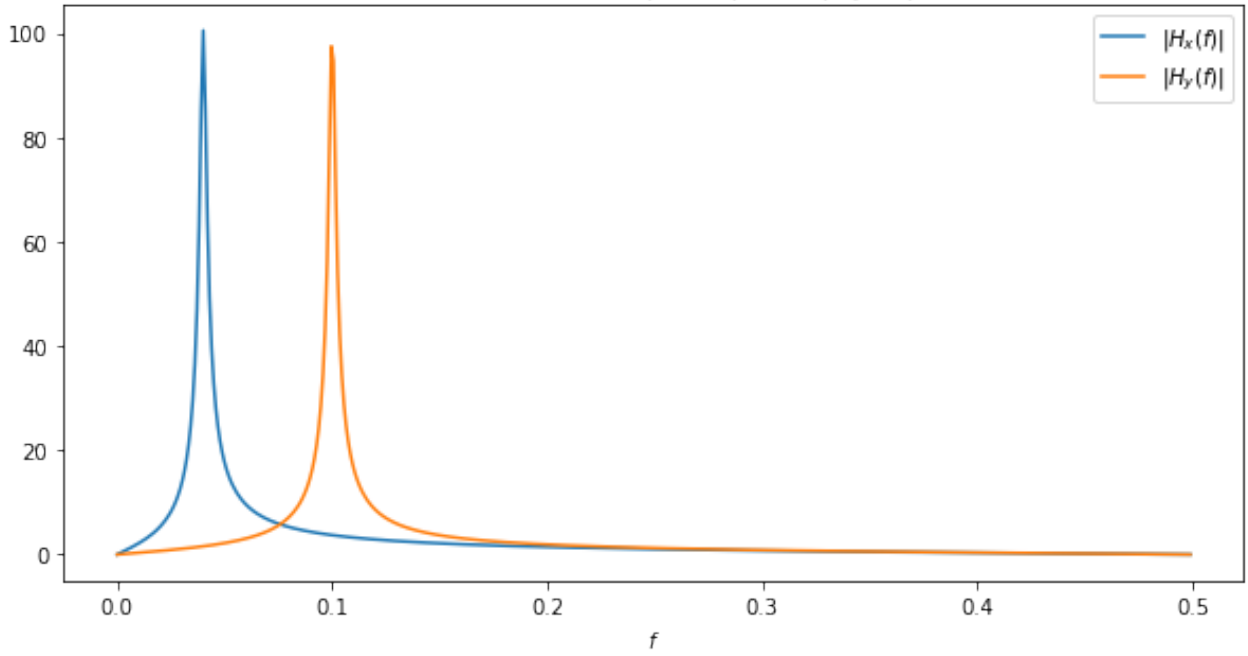


Figure 8: Plot of $|H_x(f)|$ and $|H_y(f)|$.



(c).

The code for filtering the noise contaminated signal with $H_x(z)$ and $H_y(z)$ are given below. Plots of the output from the filter as well as their amplitude spectra can be found in figure 9 and 10.

In [8]:

```
# filtering of the noise contaminated signal g[n]
qx = signal.lfilter(b_x, a_x, gn)
qy = signal.lfilter(b_y, a_y, gn)

plot = plt.plot(x_axis, qx, label = '$q_x[n]$')
plot = plt.plot(x_axis, qy, label = '$q_y[n]$')
plt.legend()
plt.title('Figure 9: Output signal $q_x[n]$ and $q_y[n]$.', fontsize = 15)
plt.show()

# find the ffts of the filtered signals
fftx = np.fft.fft(qx, n = 2048)
ffty = np.fft.fft(qy, n = 2048)
fft_axis = np.linspace(0, 0.5, 1024)

# plot of the ffts of the filtered signals
fig, axs = plt.subplots(2)
fig.suptitle('Figure 10: Amplitude spectra og the output signals,  $Q_x(f)$

axs[0].plot(fft_axis, np.abs(fftx[:int(len(fftx)/2)]))
axs[0].set(ylabel = '$|Q_x(f)|$')

fft = np.fft.fft(gn)
axs[1].plot(fft_axis, np.abs(ffty[:int(len(ffty)/2)]))
axs[1].set(xlabel = '$f$', ylabel = '$|Q_y(f)|$')
plt.show()
```

Figure 9: Output signal $q_x[n]$ and $q_y[n]$.

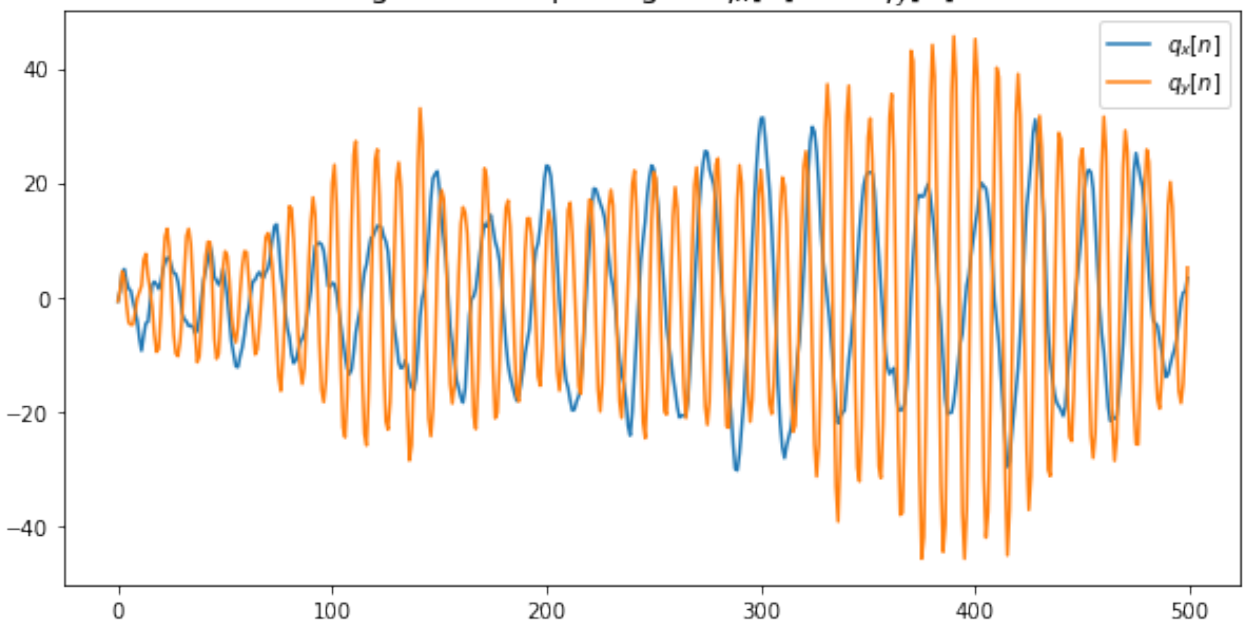
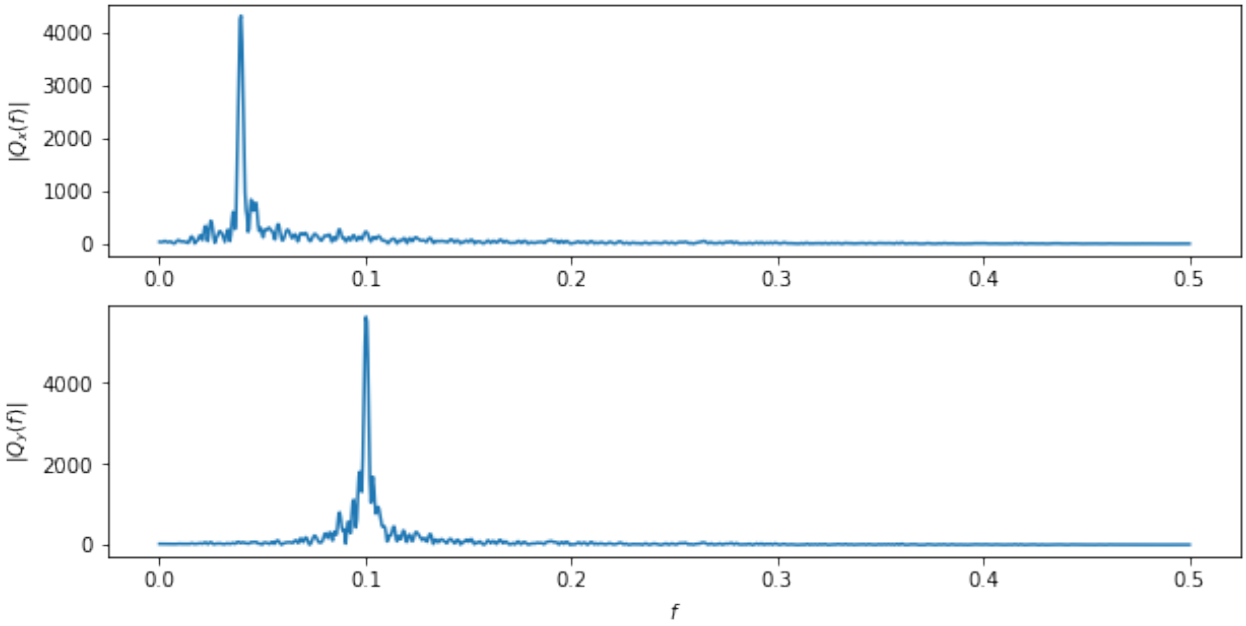


Figure 10: Amplitude spectra og the output signals, $Q_x(f)$ and $Q_y(f)$.



In figure 10 and 11, we see the output signals after filtering with $H_x(z)$ and $H_y(z)$ respectively. From $q_x[n]$ we can see that the frequency component f_x has been preserved, while f_y and most of the noise have been removed. In the signal $q_y[n]$, only f_y has been preserved. The same observations can be made from the corresponding spectra.

(d).

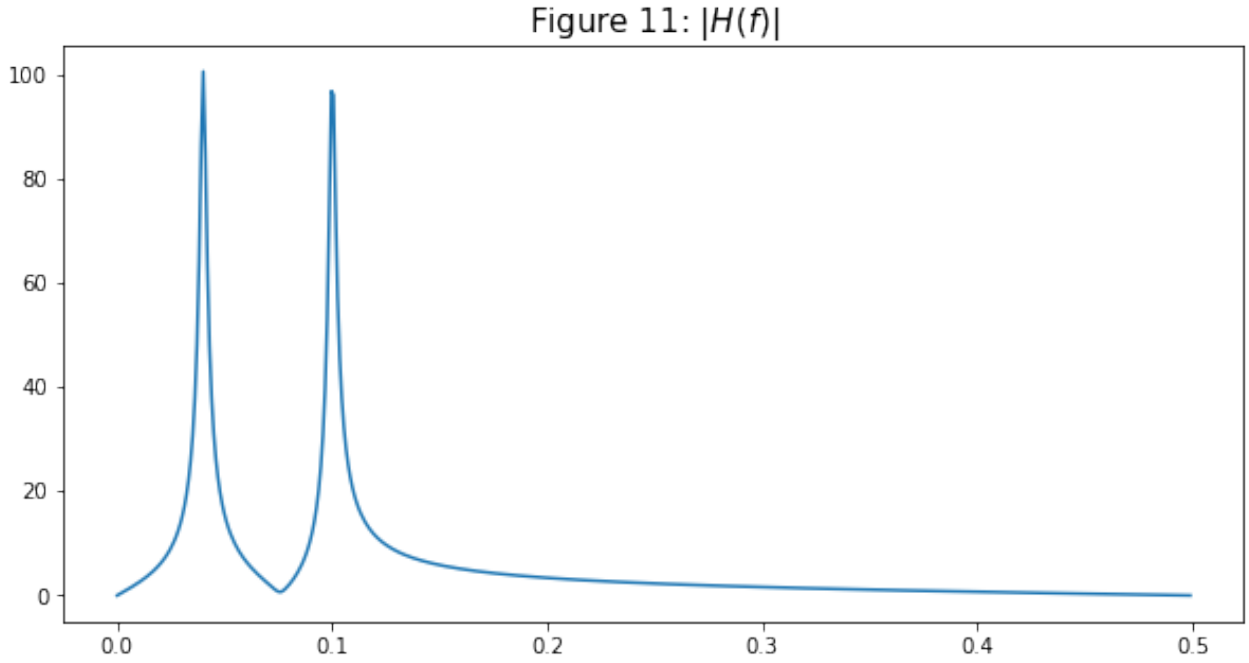
Since we want to isolate the two sinusoids with the resulting filter we need to combine then in parallel. The expression for $H(z)$ can then be written as

$$H(z) = Hx(z) + Hy(z)$$

.

The following code will plot the magnitude response of the resulting system.

In [9]:
```
H = h_x + h_y
plt.plot(w_y/(2*np.pi), np.abs(H))
plt.title('Figure 11: $|H(f)|$', fontsize = 15)
plt.show()
```



From $|H(f)|$ in figure 11 we can see that there are peaks at both f_x and f_y. In addition, the response is zero between the peaks.

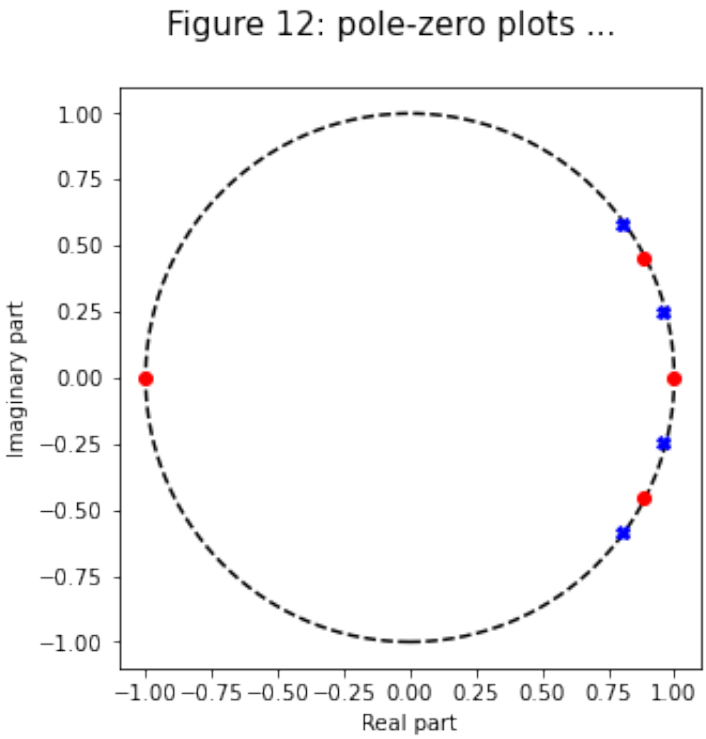
To find the poles and zeros we can write the expression for $H(z)$ as

$$H(z) = H_x(z) + H_y(z) = \frac{(1 - z^{-1})(1 + z^{-1})}{(1 - p_x z^{-1})(1 - p_x^* z^{-1})} + \frac{(1 - z^{-1})(1 + z^{-1})}{(1 - p_y z^{-1})(1 - p_y^* z^{-1})} = \frac{(1 -}$$

We can see from this equation that the filter will have all all the poles and zeros from H_x and H_y, in addition to two new zeros between the poles.

In [10]:

```
zeros = [*zeros_x, 0, 0]
zeros[2:4] = np.roots(np.poly(poles_x) + np.poly(poles_y))
poles = [*poles_x, *poles_y]

fig, ax = plt.subplots()
fig.suptitle('Figure 12: pole-zero plots ...', fontsize=15)

# plot circle
theta = np.linspace(-np.pi, np.pi, 1000)
ax.plot(np.sin(theta), np.cos(theta), '--k')
ax.set_aspect(1)

# plot poles and zeros
ax.plot(np.real(poles), np.imag(poles), 'Xb', label = 'Poles')
ax.plot(np.real(zeros), np.imag(zeros), 'or', label = 'Zeros')
ax.set_xlabel('Real part')
ax.set_ylabel('Imaginary part')
plt.show()
```



We can see from the plot in figure 12 that we have all of the poles and zeros from both $H_x(z)$ and $H_y(z)$, in addition to two new zeros between the poles.

Code for filtering the noise contaminated signal with $H(z)$ is given below, and resulting plot is shown in figure 13. We can see the result after filtering with $H(z)$. Although $q[n]$ is a somewhat distorted version of $d[n]$, we can observe similarity. The spectrum $|Q(f)|$ shows that most of the noise has been removed.

In [11]:

```
# filtering gn with the comined filter H(z)
qn = signal.lfilter(np.poly(zeros), np.poly(poles), gn)

# plot of the output signal and its fft
fig, axs = plt.subplots(2)
fig.suptitle('Figure 13: output signal after g[n] is filtered with $H(z)$.
axs[0].plot(qn, label = '$q[n]$')
axs[0].set(ylabel = '$q[n]$')

# find the ffts of the filtered signals
fftq = np.fft.fft(qn, n = 2048)
fft_axis = np.linspace(0, 0.5, 1024)

axs[1].plot(fft_axis, np.abs(fftq[:int(len(fftq)/2)]))
axs[1].set(xlabel = '$f$', ylabel = '$|Q(f)|$')
plt.show()
```

Figure 13: output signal after g[n] is filtered with $H(z)$.