Ödev7_Ornek4

May 7, 2021

[1]:
```
# As usual, we begin with importing necessary libraries and functions
import numpy as np               # for using basic array functions
import matplotlib.pyplot as plt # for this example, it may not be necessary

# the main package for signal processing is called "scipy" and we will use␣
 ↪"signal" sub-package
import scipy.signal as sgnl
from scipy.fftpack import fft, ifft
# alternative syntax: from scipy import signal as sgnl
%matplotlib notebook

# to read .csv data file
from scipy.io import loadmat
```

We will use *scipy.io* library for loading .mat files. The following piece of code shows how to access the filter coefficients properly. Note that, if you import an IIR filter, both numerator and denominator coefficients exist. But, if you import an FIR filter, only numerator coefficients exist.

[2]:
```
# importing an IIR filter
filter_data = loadmat('butterworth.mat')      # load the filter obtained from␣
 ↪pyfda
Coeffs = filter_data['ba'].astype(np.float)   # get the coefficients and␣
 ↪convert them to float (from string)

b = Coeffs[:,0]       # first column is b
a = Coeffs[:,1]       # second column is a (only if the filter is IIR)
```

Now that we obtained the coefficients, we can use them for any purpose. We will demonstrate frequency domain plots and the impulse response for now.

[3]:
```
w, Hw = sgnl.freqz(b,a)

Hw_mag = abs(Hw)
Hw_phs = np.unwrap(np.angle(Hw))

fig, (ax1, ax2) = plt.subplots(2)
ax1.plot(w/np.pi, Hw_mag)
ax2.plot(w/np.pi, Hw_phs), plt.ylabel('degrees')
```

```
plt.xlabel('frequency x$\pi$ rad/sample')
```

```
<IPython.core.display.Javascript object>

<IPython.core.display.HTML object>
```

```
[3]: Text(0.5, 0, 'frequency x$\\pi$ rad/sample')
```

```
[7]: n, hn = sgnl.dimpulse((b,a,1), n=100)          # compute the impulse response␣
      ↪for 100 points
     plt.figure()
     plt.plot(n, np.squeeze(hn))
```

```
<IPython.core.display.Javascript object>

<IPython.core.display.HTML object>
```

```
[7]: [<matplotlib.lines.Line2D at 0x270272f1e10>]
```

Now we will use this filter with an arbitrary signal to filter out a frequency component. We will use the same signal as in the *Ornek2*. We will use **lfilter** function (recall from the z-transform prelab) for filtering using the coefficients.

```
[8]: # Generate the individual components and the input signal
     N = 400      # total number of samples in the input signal
     M = 60       # size of the window function (and sample length of the frequency␣
      ↪components)
     n1 = np.arange(0, M+1, 1)    # index vector of M+1 points
     wn = 0.54 - 0.46*np.cos(2*np.pi*n1/M)       # window function (Hamming window)

     # the components
     x1 = wn*np.cos(0.2*np.pi*n1)                # component with w1 = 0.2*pi␣
      ↪frequency
     x2 = wn*np.cos(0.4*np.pi*n1 - np.pi/2)      # component with w1 = 0.4*pi␣
      ↪frequency
     x3 = wn*np.cos(0.8*np.pi*n1 - np.pi/5)      # component with w1 = 0.8*pi␣
      ↪frequency

     xn = np.concatenate((x3, x1, x2, np.zeros(N-3*len(n1))), axis=None)

     plt.figure()
     plt.plot(xn)
```

```
<IPython.core.display.Javascript object>

<IPython.core.display.HTML object>
```

```
[8]: [<matplotlib.lines.Line2D at 0x270275cc2b0>]
```

[6]:
```
yn = sgnl.lfilter(b, a, xn)          # using the coefficients, we applied the␣
 ↪signal to the input of the filter.

plt.figure()
plt.plot(yn)
```

```
<IPython.core.display.Javascript object>

<IPython.core.display.HTML object>
```

[6]:
```
[<matplotlib.lines.Line2D at 0x2702702cfd0>]
```

[ ]:
```
```

[ ]:
```
```