DCA_Assignment3

September 28, 2016

```
In [43]: # Imports
         import numpy as np
         from numpy import cos, sin, pi, absolute, arange, ndarray, dtype
         from scipy.signal import kaiserord, lfilter, firwin, freqz
         from pylab import figure, clf, plot, xlabel, ylabel, xlim, ylim, title, gr
         from random import uniform
         from decimal import *
         import FixedPoint as FP
         import matplotlib.patches as mpatches
         import matplotlib.pyplot as plt
```

0.1 Setting Cases

1 - NoAS - No altered storage
2 - Fl16 - 16 bit Floating Point storage
3 - Fl32 - 32 bit Floating Point storage
4 - Fi16 - 16 bit Fixed Point storage
5 - Fi32 - 32 bit Fixed Point storage

```
In [45]: #Define class with multiple definitions
         class Filter:
             #Define variables used in following definitions (def)
             def __init__(self,signal,t,Instance,plotinstance,ripple_db,cutoff_hz):
                 self.signal = signal
                 self.time = t
                 self.Instance = Instance
                 self.ripple_db = ripple_db
                 self.cutoff_hz = cutoff_hz
                 self.plotinstance = plotinstance
                 self.taps = 0
                 self.nyq_rate = 0
                 self.filtersignal = 0
                 self.N = 0
                 c = getcontext() #return the current context for the active threa

             #set functions applied to self, will be detailed later
             def apply(self):
```

```
        self.createFIR() #create FIR filter
        self.setInstance() #choose filter instance
        self.filtersignal = lfilter(self.taps,1.0,self.signal) #applying f
        self.plot(self.plotinstance)

    #defining what actions to be taken depending on type
    #altering signal (self.signal) and taps (self.taps) of FIR filter
    #taps: filter order + 1
    def setInstance(self):
        if(self.Instance=='NoAS'):
            self.signal = self.signal
            self.taps = self.taps
        if(self.Instance=='Fl16'): #using Numpy float16 to alter to 16 bit
            self.signal = np.float16(self.signal)
            self.taps = np.float16(self.taps)
        if(self.Instance=='Fl32'): #using Numpy float16 to alter to 32 bit
            self.signal = np.float32(self.signal)
            self.taps = np.float32(self.taps)
        if(self.Instance=='Fi16'): #using FixedPoint to alter to 16 bit Fi
            for n, line in enumerate(self.signal):
                self.signal[n] = FP.FXnum(self.signal[n], FP.FXfamily(7,8)
            for n, line in enumerate(self.taps):
                self.taps[n] = FP.FXnum(self.taps[n], FP.FXfamily(7,8))
        if(self.Instance=='Fi32'): #using FixedPoint to alter to 16 bit Fi
            for n, line in enumerate(self.signal):
                self.signal[n] = FP.FXnum(self.signal[n], FP.FXfamily(15,1
            for n, line in enumerate(self.taps):
                self.taps[n] = FP.FXnum(self.taps[n], FP.FXfamily(15,16))

    #define FIR filter
    def createFIR(self):
        self.nyq_rate = sample_rate / 2.0 #Nyquist rate
        width = 5.0 / self.nyq_rate #transition from pass to stop
        self.N, beta = kaiserord(ripple_db, width) #order and Kaiser param
        self.taps = firwin(self.N, self.cutoff_hz/self.nyq_rate, window=('

    #defining plot init per instance
    def plot(self,Instance):
        if(Instance == 1): #No action
            None
        if(Instance == 2): #Show plot for signal
            self.PlotSignal()
        if(Instance == 3): #Show plots for filter coefficients, magnitude
            self.PlotFirCoefficient()
            self.PlotFirMagnitude()
            self.PlotSignal()
        if(Instance == 4): #show plot for filter coefficients
            self.PlotFirCoefficient()
```

```
        if(Instance == 5): #show plot for filter magnitude
            self.PlotFirMagnitude()
        show()

    #Plot for filter coefficients
    def PlotFirCoefficient(self):
        figure(1)
        plot(self.taps, 'bo-', linewidth=2)
        title('Filter Coefficients (%d taps)' % self.N)
        grid(True)

    #Plot for filter magnitude
    def PlotFirMagnitude(self):
        figure(2)
        clf()
        w, h = freqz(self.taps, worN=8000)
        plot((w/pi)*self.nyq_rate, absolute(h), linewidth=2)
        xlabel('Frequency (Hz)')
        ylabel('Gain')
        title('Frequency Response')
        ylim(-0.05, 1.05)
        grid(True)

        # Upper inset plot.
        ax1 = axes([0.42, 0.6, .45, .25])
        plot((w/pi)*self.nyq_rate, absolute(h), linewidth=2)
        xlim(0,8.0)
        ylim(0.9985, 1.001)
        grid(True)

        # Lower inset plot
        ax2 = axes([0.42, 0.25, .45, .25])
        plot((w/pi)*self.nyq_rate, absolute(h), linewidth=2)
        xlim(12.0, 20.0)
        ylim(0.0, 0.0025)
        grid(True)

    #Plot for original signal
    def PlotSignal(self):
        # The phase delay of the filtered signal.
        delay = 0.5 * (self.N-1) / sample_rate

        figure(3)

        #Plot original signal
        plot(self.time, self.signal)

        #Plot filtered signal, shifted to compensate for phase delay
```

```
                plot(self.time-delay, self.filtersignal, 'r-')

                #Remove first N-1 samples which are corrupted due to initial condi
                plot(self.time[self.N-1:]-delay, self.filtersignal[self.N-1:], 'g'
                xlabel('t')
                title('Signal')
                grid(True)

            #Returns taps when requested
            def getTaps(self):
                return self.taps

            #Returns signal when requested
            def getSignal(self):
                return self.signal

            #Returns filtered signal when requested
            def getFilterSignal(self):
                return self.filtersignal
```

0.2 Creating Random Signal

```
In [46]: sample_rate = 100.0
         nsamples = 300
         A1 = uniform(0,1); B1 = uniform(0,1); C1 = uniform(0,1); D1 = uniform(0,1)
         t = arange(nsamples) / sample_rate
         signal = cos(2*pi*0.5*t) + A1*sin(2*pi*2.5*t+0.1) + \
                 B1*sin(2*pi*15.3*t) + C1*sin(2*pi*16.7*t + 0.1) + \
                     D1*sin(2*pi*23.45*t+.8)
```

0.3 Filtering

S/F/T1 - NoAS - No altered storage
S/F/T2 - Fl16 - 16 bit Floating Point storage
S/F/T3 - Fl32 - 32 bit Floating Point storage
S/F/T4 - Fi16 - 16 bit Fixed Point storage
S/F/T5 - Fi32 - 32 bit Fixed Point storage
S = signal, F = filter, T = taps

```
In [47]: #Desired attenuation in stop band, dB
         ripple_db = 60.0
         #Cutoff frequency of filter, Hz
         cutoff_hz = 10.0

         #Setup filters per instance as set in class __init__
         #For alternative plots use: 2 for signal, 3 for filter coefficients, magi
                                    #4 for filter coefficients, 5 or magnitude
         #Signal, time, instance, plotinstance, attenuation, cutoff frequency
```

```
F1 = Filter(signal,t,'NoAS',1,ripple_db,cutoff_hz)
F2 = Filter(signal,t,'Fl16',1,ripple_db,cutoff_hz)
F3 = Filter(signal,t,'Fl32',1,ripple_db,cutoff_hz)
F4 = Filter(signal,t,'Fi16',1,ripple_db,cutoff_hz)
F5 = Filter(signal,t,'Fi32',1,ripple_db,cutoff_hz)

#Unfiltered signal
F1.apply()
T1 = F1.getTaps()
S1 = F1.getSignal()
SF1 = F1.getFilterSignal()

#Floating point 16 bit
F2.apply()
T2 = F2.getTaps()
S2 = F2.getSignal()
SF2 = F2.getFilterSignal()

#Floating point 32 bit
F3.apply()
T3 = F3.getTaps()
S3 = F3.getSignal()
SF3 = F3.getFilterSignal()

#Fixed point 16 bit
F4.apply()
T4 = F4.getTaps()
S4 = F4.getSignal()
SF4 = F4.getFilterSignal()

#Fixed point 32 bit
F5.apply()
T5 = F5.getTaps()
S5 = F5.getSignal()
SF5 = F5.getFilterSignal()
```

0.4 Calculating differences compared to original signal (no change in storage)

```
In [48]: #Differences between taps
tapsdifference2 = T1 - T2
tapsdifference3 = T1 - T3
tapsdifference4 = T1 - T4
tapsdifference5 = T1 - T5

#Differences between signals
signaldifference2 = S1 - S2
signaldifference3 = S1 - S3
signaldifference4 = S1 - S4
```

```
signaldifference5 = S1 - S5

#Differences between filtered signals
filtersignaldifference2 = SF1 - SF2
filtersignaldifference3 = SF1 - SF3
filtersignaldifference4 = SF1 - SF4
filtersignaldifference5 = SF1 - SF5
```

0.5 Plotting differences

1 - NoAS - No altered storage
2 - Fl16 - 16 bit Floating Point storage
3 - Fl32 - 32 bit Floating Point storage
4 - Fi16 - 16 bit Fixed Point storage
5 - Fi32 - 32 bit Fixed Point storage

```
In [55]: figure(5)
         #State what to plot
         plot(t,filtersignaldifference2,'r')

         #Define legend
         floatLegend = mpatches.Patch(color='red', label='16 bit Floating Point')
         legend(handles=[floatLegend])

         #Define title and axes
         title('Error of 16 bit Floating Point compared to original signal')
         xlabel('Time')
         ylabel('Error')

         figure(6)
         #State what to plot
         plot(t,filtersignaldifference3,'r')

         #Define legend
         floatLegend = mpatches.Patch(color='red', label='32 bit Floating Point')
         legend(handles=[floatLegend])

         #Define title and axes
         title('Error of 32 bit Floating Point compared to original signal')
         xlabel('Time')
         ylabel('Error')

         figure(7)
         #State what to plot
         plot(t,filtersignaldifference4,'r')

         #Define legend
         fixedLegend = mpatches.Patch(color='red', label='16 bit Fixed Point')
```

```
legend(handles=[fixedLegend])

#Define title and axes
title('Error of 16 bit Fixed Point compared to original signal')
xlabel('Time')
ylabel('Error')

figure(8)
#State what to plot
plot(t,filtersignaldifference5,'r')

#Define legend
fixedLegend = mpatches.Patch(color='red', label='32 bit Fixed Point')
legend(handles=[fixedLegend])

#Define title and axes
title('Error of 32 bit Fixed Point compared to original signal')
xlabel('Time')
ylabel('Error')

show()
```

```
In [ ]:
```