Michael Hickey

Professor Tom

Digital Signals Processing

Lab 10 Filtering Using FFT and IFFT

August 1st, 2018

Code:

import numpy as np

import matplotlib.pyplot as plt

from scipy.fftpack import fft, ifft

from scipy.signal import lfilter,freqz

def signal\_function(t):

# t is a list of numbers.

return np.cos(2\*np.pi\*10\*t) + np.cos(2\*np.pi\*100\*t) + np.cos(2\*np.pi\*200\*t)

def fft\_plots(t, v, signal\_xt, abs\_fft\_out, ifft\_out):

###########################

### FFT and IFTTT PLOTS ###

###########################

# Set up three subplots.

f, xarr = plt.subplots(3)

# Change the color of the plots.

plt.style.use('seaborn-dark') #print(plt.style.available)

# Add space between the subplots.

plt.subplots\_adjust(hspace = 1.25)

# NOISY SIGNAL

xarr[0].set\_title("Noisy Signal") # subplot[index]

xarr[0].set\_xlabel("Time t")

xarr[0].set\_ylabel("Volts V")

xarr[0].plot(t, signal\_xt)

# FFT SIGNAL

xarr[1].set\_title("FFT Amplitude Spectrum")

xarr[1].set\_xlabel("Frequency")

xarr[1].set\_ylabel("Amplitude")

xarr[1].plot(v[0:499], abs\_fft\_out[0:499])

# IFFT SIGNAL

xarr[2].set\_title("Inverse FFT Amplitude Spectrum")

xarr[2].set\_xlabel("Time t")

xarr[2].set\_ylabel("Volts V")

xarr[2].plot(t, ifft\_out)

return 0

def time\_domain\_filtering\_plots(t, signal\_xt, b, bn, filter\_out):

###################################

### TIME DOMAIN FILTERING PLOTS ###

###################################

# TDFP = Time Domain Filtering Plots

# Set up a column of subplots.

f, tdfp = plt.subplots(3)

# Change the style of the plots.

plt.style.use("seaborn-dark")

# Add space between the subplots.

plt.subplots\_adjust(hspace = 1.25)

# NOISY SIGNAL

tdfp[0].set\_title("Noisy Signal") # subplot[index]

tdfp[0].set\_xlabel("Time t")

tdfp[0].set\_ylabel("Volts V")

tdfp[0].plot(t, signal\_xt)

# FILTER IMPULSE

tdfp[1].set\_title("Filter Impulse (YOU CAN IGNORE THIS SUBPLOT)") # subplot[index]

tdfp[1].set\_xlabel("Time t")

tdfp[1].set\_ylabel("Volts V")

tdfp[1].stem(bn, b)

# FILTER OUTPUT (CONVOLUTION)

tdfp[2].set\_title("Filter Output (Convolution)") # subplot[index]

tdfp[2].set\_xlabel("Time t")

tdfp[2].set\_ylabel("Volts V")

tdfp[2].plot(t, filter\_out)

return 0

def frequency\_domain\_filtering\_plots(v, abs\_fft\_out, hzTrue, H, w\_filter\_out):

########################################

### FREQUENCY DOMAIN FILTERING PLOTS ###

########################################

# fdf = Frequency Domain Filtering

f, fdf = plt.subplots(3)

# Change the style of the plots.

plt.style.use("seaborn-dark")

# Add space between the subplots.

plt.subplots\_adjust(hspace = 1.25)

# Amplitude Spectrum (Real Half)

fdf[0].set\_title("Amplitude Spectrum (Real Half)") # subplot[index]

fdf[0].set\_xlabel("Frequency Hz")

fdf[0].set\_ylabel("Amplitude")

fdf[0].plot(v[0:499], abs\_fft\_out[0:499])

# Frequency Response of Filter

fdf[1].set\_title("Frequency Response of Filter") # subplot[index]

fdf[1].set\_xlabel("Frequency Hz")

fdf[1].set\_ylabel("Amplitude")

fdf[1].plot(hzTrue[0:499], H[0:499])

# Filter Output (Fast Fourier Transform)

fdf[2].set\_title("Filter Output (fft)") # subplot[index]

fdf[2].set\_xlabel("Frequency Hz")

fdf[2].set\_ylabel("Amplitude")

fdf[2].plot(v[0:499], w\_filter\_out[0:499])

return 0

def final\_plot(t, v, filter\_out, filter\_out\_fft\_out, w\_filter\_out, ifft\_tdf):

# fp = Final Plot

f, fp = plt.subplots(4)

# Change the style of the plots.

plt.style.use("seaborn-dark")

# Add space between the subplots.

plt.subplots\_adjust(hspace = 1.25)

# FILTER OUTPUT (CONVOLUTION)

fp[0].set\_title("Filter Output (Convolution)") # subplot[index]

fp[0].set\_xlabel("Time t")

fp[0].set\_ylabel("Volts V")

fp[0].plot(t, filter\_out)

# FFT of Convolution

fp[1].set\_title("FFT of Time Domain Filtering (Convolution)") # subplot[index]

fp[1].set\_xlabel("Time t")

fp[1].set\_ylabel("Amplitude")

fp[1].plot(v[0:499], filter\_out\_fft\_out[0:499])

# Filter Output of Frequency Domain Filtering (Fast Fourier Transform)

fp[2].set\_title("Filter Output of Frequency Domain Filter (fft)") # subplot[index]

fp[2].set\_xlabel("Frequency Hz")

fp[2].set\_ylabel("Amplitude")

fp[2].plot(v[0:499], w\_filter\_out[0:499])

# The Inverse Fast Fourier Transform of the output of the filtered waveform from FFT.

fp[3].set\_title("IFFT of FFT of Frequency Domain Filtering") # subplot[index]

fp[3].set\_xlabel("Time t")

fp[3].set\_ylabel("Amplitude")

#fp[3].plot(v[0:499], ifft\_tdf[0:499])

fp[3].plot(t, ifft\_tdf)

return 0

if \_\_name\_\_ == "\_\_main\_\_":

#####################

### FFT and IFTTT ###

#####################

# t is of length 1000. Numbers range from 0 to 2.

t = np.linspace(0, 2, 1000)

# Making a list of numbers which will make the list of frequencies.

n = np.linspace(0, 1, 1000) # n goes from 0 to 1 with 1000 points in-between.

# Create the signal xt

signal\_xt = signal\_function(t)

# T is the sample period

# T = (endTime - startTime)/(signalLength)

T = (2-0)/len(signal\_xt) # T = 0.002

# v is the sample frequency multiplies by the length of the function to make an array of frequencies.

v = n/T # v ranges from 0 to 500 with 1000 samples in between.

# Obtain the absolute value of the fast Fourier transform of the signal xt

fft\_out = fft(signal\_xt)

abs\_fft\_out = abs(fft\_out)

# Get the inverse Fourier transform from the fast Fourier transform.

ifft\_out = ifft(fft\_out)

#############################

### TIME DOMAIN FILTERING ###

#############################

# 15 point moving average filter

N = 15

# Make list of 15 ones, and divide every single one by 15.

b = (1/N)\*np.ones(N)

# Make 15 different points for graphing.

bn = np.arange(15)

# Generate a filtered output of the signal\_xt, This convolves the sequence b, with the noisy data.

filter\_out = lfilter(b, 1, signal\_xt)

##################################

### FREQUENCY DOMAIN FILTERING ###

##################################

# The rate at which the frequency is sampled.

sample\_rate = 1/T # sample\_rate = 500

# H is a representation of our moving average filter called the frequency response.

hz, H = freqz(b, 1, worN = v\*2\*np.pi/sample\_rate)

hzTrue = hz\*sample\_rate/(2\*np.pi)

# Filter the fast Fourier transform of the signal xt

w\_filter\_out = H\*fft\_out

##################

### FINAL PLOT ###

##################

# From the Time Domain Filtering.

filter\_out\_fft\_out = fft(filter\_out) # Plot amplitude spectrum.

# Take the Inverse Fast Fourier Transform of the FFT of the Time Domain Filtering.

ifft\_tdf = ifft(w\_filter\_out)

#################

### ALL PLOTS ###

#################

# FFT and IFFT Filtering Plots

fft\_plots(t, v, signal\_xt, abs\_fft\_out, ifft\_out)

# Time Domain Filtering Plots

time\_domain\_filtering\_plots(t, signal\_xt, b, bn, filter\_out)

# Frequency Domain Filtering Plots

frequency\_domain\_filtering\_plots(v, abs\_fft\_out, hzTrue, H, w\_filter\_out)

# Final Plots

final\_plot(t, v, filter\_out, filter\_out\_fft\_out, w\_filter\_out, ifft\_tdf)

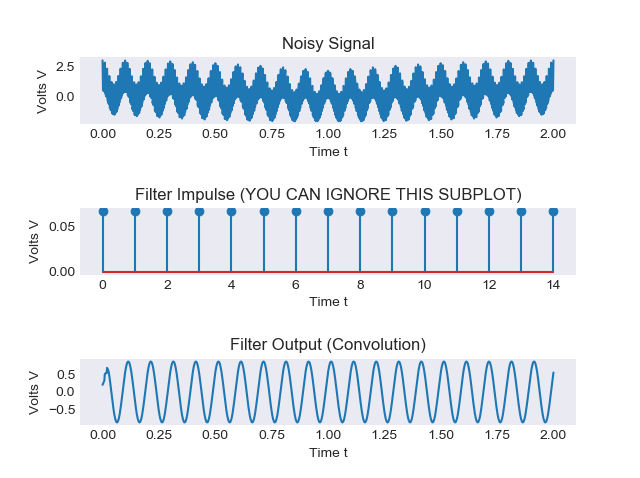
# Show all the plots.

plt.show()

A close up of a logo

Description generated with very high confidence

The Noisy Signal is presented in subplot one of the figure. The FFT (Fast Fourier Transform) is taken of the noisy signal. The FFT gives us the frequency of the noisy signal; which is presented in subplot two of the figure. Next, the IFFT (Inverse Fast Fourier Transform) is taken of the FFT Noisy Signal. The IFFT gives the user a similar waveform in comparison to the original Noisy Signal.



This figure is about taking a fifteen-point moving average filter convolved with the Noisy Signal. The result of convolving the filter with the Noisy Signal will produce a convolved filter output. The result of convolving the two first subplots together results in the third subplot.

A picture containing screenshot

Description generated with very high confidence

This figure shows how the filter will react with the amplitude spectrum of the Noisy Signal. First off, we want only the spike at 10 Hz to be allowed through the filter. Notice how the filter’s graph shows a high amplitude, somewhere between 0.5 and 1 at frequency 10Hz. This high amplitude of the filter at 10Hz allows for the frequency we want to capture pass through the filter. Notice that the frequency response of the of the filter is low for everything else after, in comparison to the high point at 10Hz. Because the rest of the filter is low, the last two spikes of the amplitude spectrum will go to zero. This is great because we do not want to capture those last two spikes. This shows that the filter is allowing a specific frequency through the filter, and blocking all other frequencies from passing through.

A screenshot of a cell phone

Description generated with very high confidence

The final plot shows the Filtered Output of the Noisy Signal, performed by the convolution method we made. It also shows what frequency was allowed through the filter in the second subplot. The third subplot is the frequency output from the FFT method. The fourth plot shows the IFFT of the frequency output from the Frequency Domain Filter FFT method. This shows that similar results and objectives can be reached from multiple approaches.