SIGNAL PROCESSING SYSTEM DESIGN LAB

YASH ANAND
(23EE65R22)

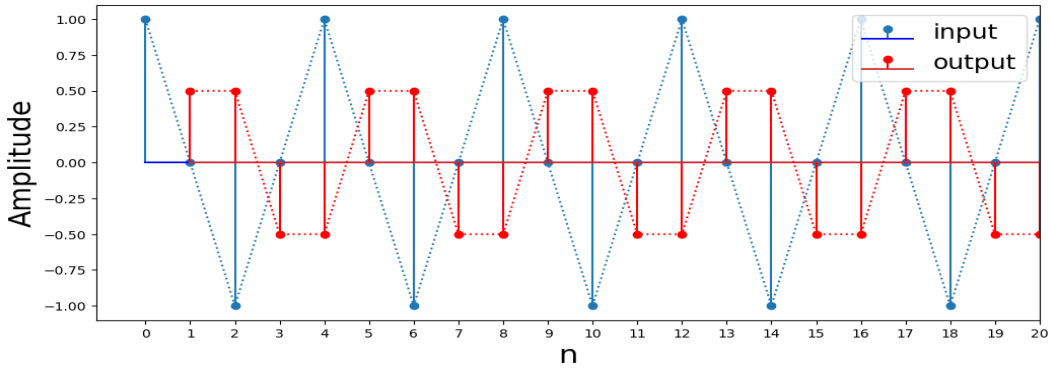
5. Finite Impulse Response Filters

In this segment, we delved into the Parks-McClellan algorithm, a tool for generating coefficients of Finite Impulse Response (FIR) filters. It achieves this by solving an optimal equiripple Chebyshev approximation, aligning with the ideal filter stipulations provided in the specifications. Utilizing the signal.remez function, this design technique empowers designers to define passband and stopband frequencies, the targeted response within the passband, and optionally assign varying significance to the desired response within the passband compared to the stopband.

5.1
AIM:
The aim of the given code is to demonstrate the impact of linear filtering on a cosine input signal using the SciPy and Matplotlib libraries. It shows how applying a linear filter to the input signal affects its amplitude and shape..
FUNCTION IMPORTED:
signal from scipy for signal processing.
arange()- arranging the sequence in ascending order.
cos(np.arange())- used to generate an input signal x by applying a cosine function to the sample index.
signal.lfilter()- Apply a linear filter to the input signal using signal.lfilter

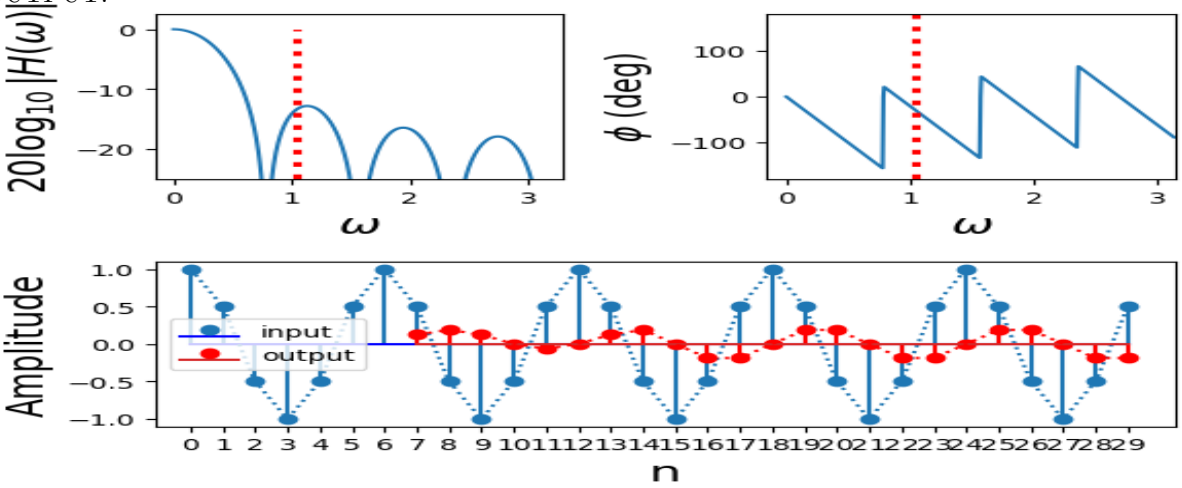
OBSERVATION:
The code generates a plot illustrating the effect of linear filtering on an input cosine signal. It showcases the input signal, the filter output, and various plot settings to enhance the visualization. The stem plot and line plot combined provide a clear depiction of how the filtering operation alters the signal's amplitude.

OUTPUT:

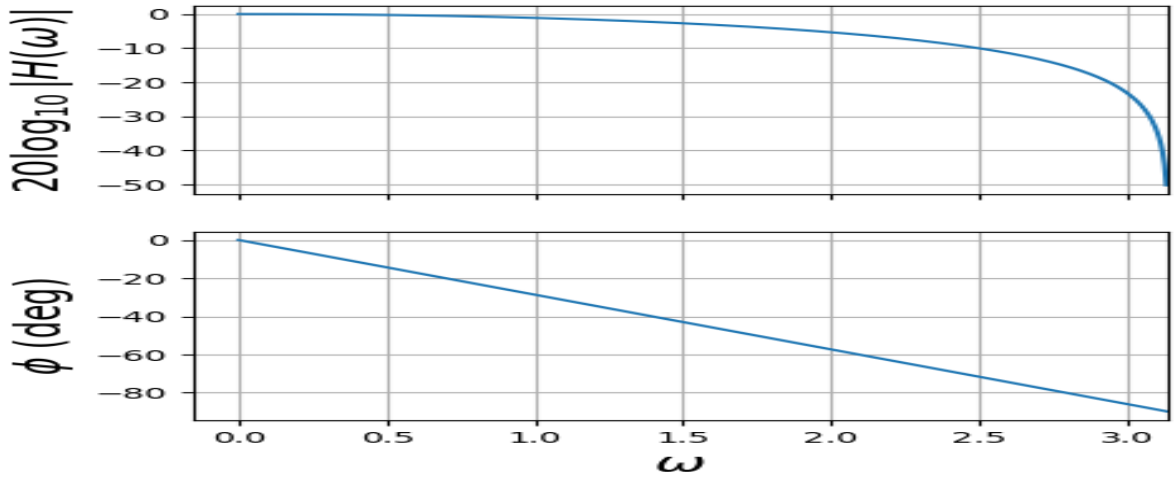


5.2
AIM:The aim of the provided code is to visually demonstrate and explain the concepts related to filtering and its frequency response. The code showcases how to generate and visualize the frequency response of a moving average filter, and it also demonstrates the effect of this filter on a cosine input signal.
FUNCTION IMPORTED:
gs = gridspec.GridSpec()-Create a grid of subplots ,adjust the spacing between subplots.
gs.update(wspace, hspace)-wspace: This parameter controls the horizontal spacing between subplots.hspace: This parameter controls the vertical spacing between subplots, allowing you to control the gap between rows of subplots.
fig.add subplot(subplot(gs[]))- creates a subplot at the position corresponding to the top-left cell of the grid.
ax.stem()- method provided by the matplotlib.pyplot library in Python for creating stem plots in a subplot (axes) of a figure.
OBSERVATION:
The output is the signal.frequency function computes the filter's magnitude and phase response given the filter co-

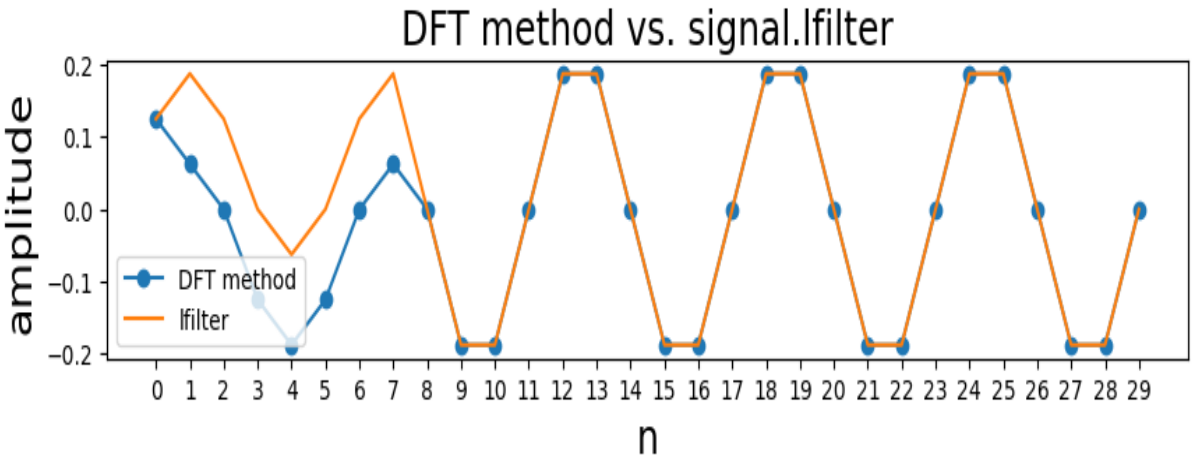
efficents.
The code generates a figure with three subplots. The first subplot shows the frequency response of the moving average filter, the second subplot shows the phase response, and the third subplot shows the input and output signals after applying the filter.
OUTPUT:



5.3
AIM-To shows the magnitude response of the filter in decibels.
FUNCTION IMPORTED:
Import the pyplot module from the matplotlib library for creating plots.
subplots(sharex=True)-Create a figure with two vertically stacked subplots.
OBSERVATION
This code generates a figure with two vertically aligned subplots. The top subplot shows the magnitude response of the filter in decibels, and the bottom subplot shows the phase response of the filter in degrees. The x-axis in both subplots represents angular frequency.
OUTPUT:



5.4
AIM- To compare the output of a FIR filter using two different methods.
FUNCTION IMPORTED:
yc = np.fft.ifft(...): Calculate the convolution of the filter and input signal using the frequency domain approach. It involves the element-wise product of the frequency domain representations of h and x, followed by an inverse FFT to return to the time domain.
OBSERVATION:
The purpose of this code is to visually compare the output of a FIR filter obtained using the DFT method and the signal.lfilter function. The yc output is calculated through convolution in the frequency domain, and it is compared with the output obtained using the built-in signal.lfilter function. The plot allows you to see the similarities and differences between these two methods.
OUTPUT:



5.5

AIM - To visualize the frequency response of a low-pass filter .

FUNCTION IMPORTED:

The scipy.signal and numpy.fft libraries.

w,Hh = signal.freqz() - get entire frequency domain.

wx = fft.fftfreq(len()) - shift to center for plotting.

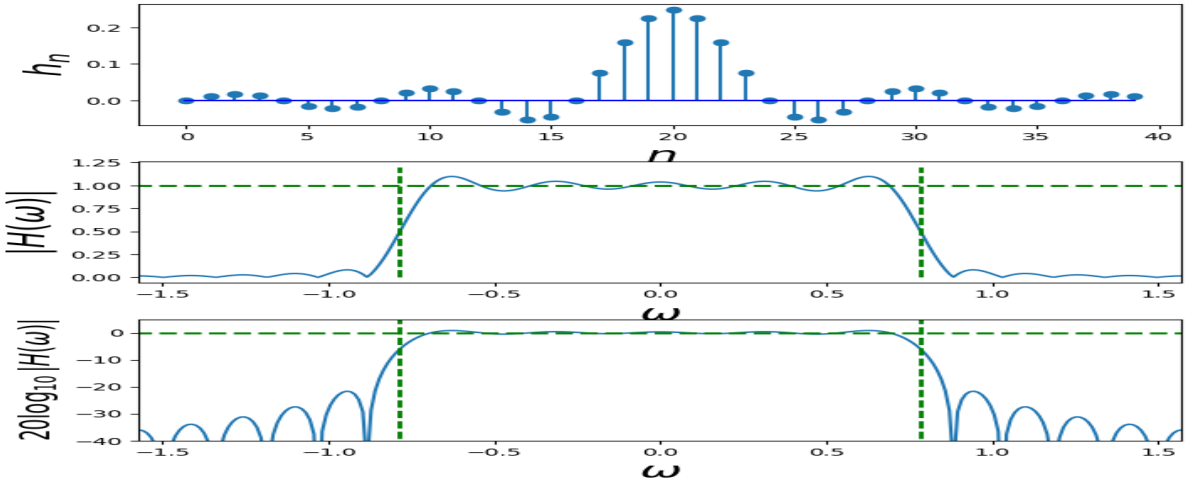
fig.set size inches()-This function is called on the figure object to set its size.

subplots adjust(hspace)-used to adjust the spacing between subplots.

OBSERVATION:

The code visually represents the characteristics of a low-pass filter.

OUTPUT:



5.6

AIM:

The aim of the code is to plot the real part of the Fourier transform of a square wave, and to observe the Gibbs phenomenon.

FUNCTION IMPORTED:

fig.set size inches():-Sets the dimensions of the figure.

omega = np.linspace(): Generates an array of angular frequencies ranging.

ax.annotate(...): Adds an annotation indicating the Gibbs phenomenon.

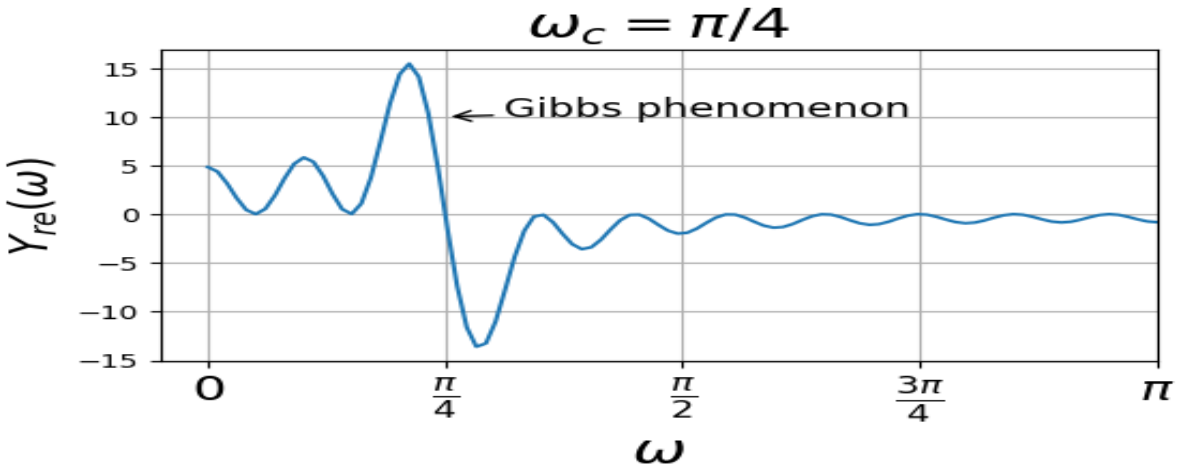
textcoords-This parameter helps you control where the annotation text is placed relative to the point you're annotating.

arrowprops- This parameter is a dictionary that contains various settings for customizing the arrow's appearance.

OBSERVATION:

The observation of the code is that the Gibbs phenomenon is more pronounced near the center frequency. This is because the Fourier transform of a square wave has a sharper peak at the center frequency.

OUTPUT:



5.7

AIM:

The aim of the code is to plot the frequency response of a finite impulse response (FIR) filter, and to observe the effects of the Gibbs phenomenon.

FUNCTION IMPORTED:

signal.hamming(len(n)):- This function call generates a Hamming window of the same length as the array n, which represents the filter taps.

The len(n) argument specifies the length of the window, which corresponds to the number of filter taps.

OBSERVATION:

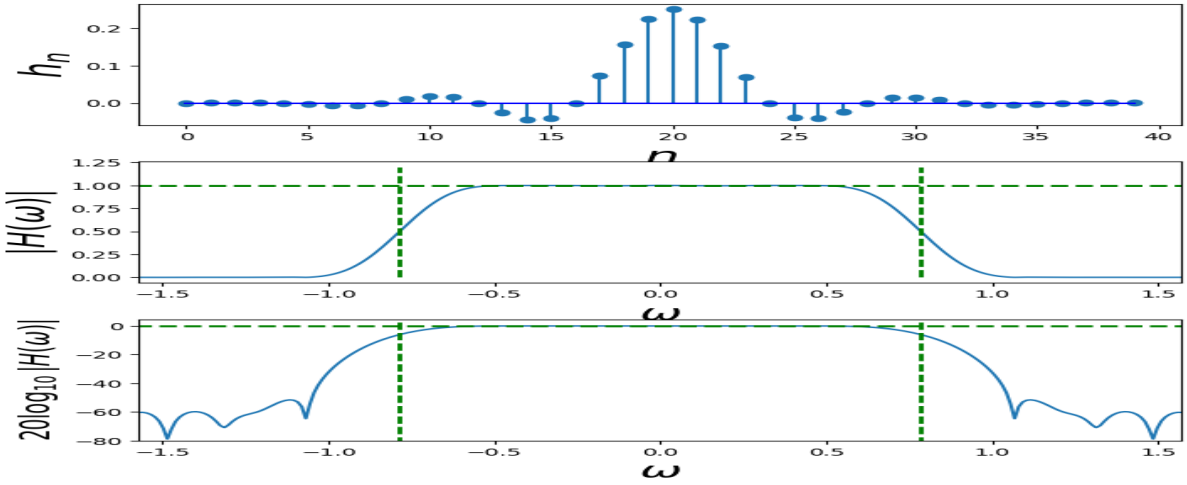
The code also plots the logarithmic frequency response of the filter. The logarithmic frequency response is more sensitive to the Gibbs phenomenon than the linear frequency response. The code also shows that the Gibbs phenomenon can be reduced by using a wider window. A wider window smooths out the discontinuities in the impulse response, and this reduces the Gibbs phenomenon.

OUTPUT:

Text(0, 0.5, ’20

$log_{10}|H($

$omega)|$’)



5.8

AIM:

The aim of the code is to design a finite impulse response (FIR) filter using the Kaiser window, and to observe the frequency response of the filter.

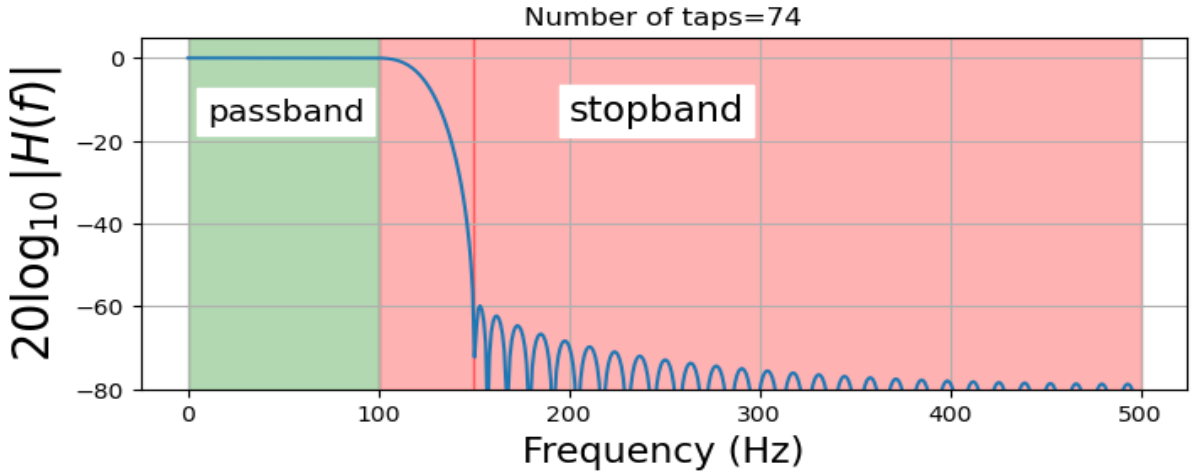
FUNCTION IMPORTED:

from matplotlib.patches import Rectangle

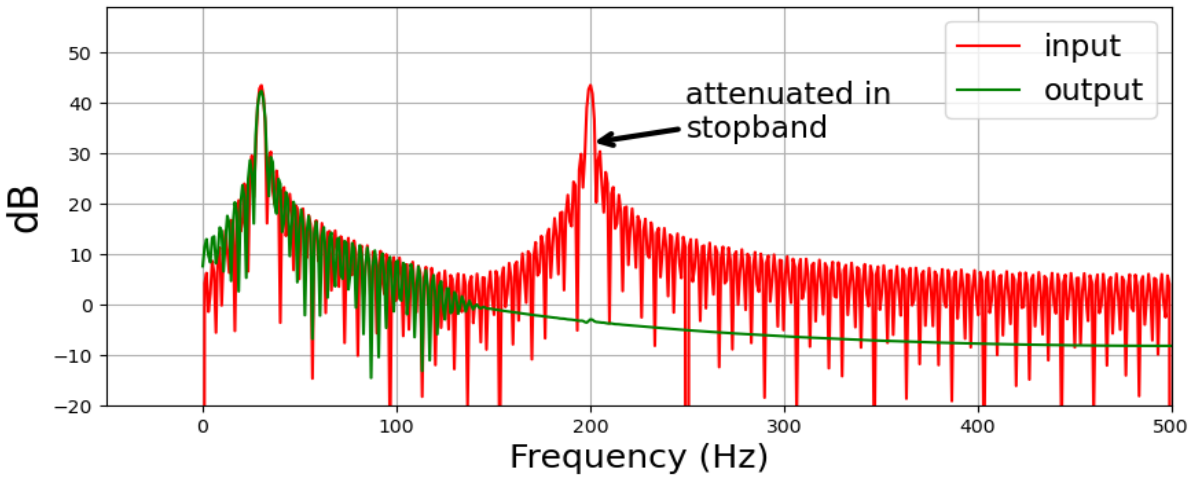
M, beta = signal.fir filter design.kaiserord(): Calculate the number of filter taps (M) and the beta parameter for the Kaiser window function using the desired attenuation and transition bandwidth.

hn = signal.firwin(): Design the FIR filter using the firwin function with the Kaiser window. The filter taps are stored in the array hn.

w, H = signal.freqz(hn): Calculate the frequency response of the filter.
ax.add patch(...): This function is used to add a graphical patch, such as a rectangle, to the plot.
Rectangle(...): This is a constructor for creating a rectangle patch. It takes several parameters to define the position, size, color, and other properties of the rectangle.
width=fpass: This parameter sets the width of the rectangle to fpass, which corresponds to the width of the passband.
alpha=0.3: This parameter sets the transparency level of the rectangle. A value of 0.3 makes the rectangle slightly transparent.
OBSERVATION:
The observation of the code is that the Kaiser window is a good choice for designing FIR filters with sharp transitions between the passband and the stopband. The Kaiser window is a non-rectangular window that has a smooth shape. This smooth shape helps to reduce the Gibbs phenomenon in the frequency response of the filter.
OUTPUT:

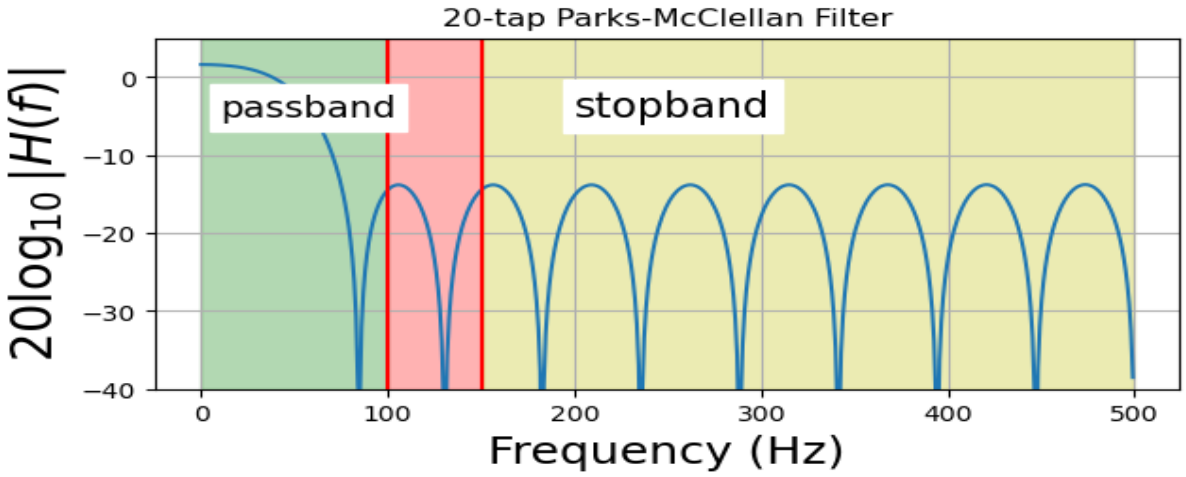


5.9
AIM:
The aim of the code is to filter out the signal at 200 Hz using the FIR filter designed in the previous code, and to observe the effect of the filter.
FUNCTION IMPORTED:
from numpy import fft
ax.legend()- function takes a list of labels as its argument.
OBSERVATION:
The observation of the code is that the FIR filter is able to filter out the signal at 200 Hz with good attenuation. The attenuation in the stopband is more than 60 dB.
The code also shows that the Fourier transform of the output signal has a sharper transition between the passband and the stopband than the Fourier transform of the input signal. This is because the FIR filter has a sharp transition between the passband and the stopband.
OUTPUT:

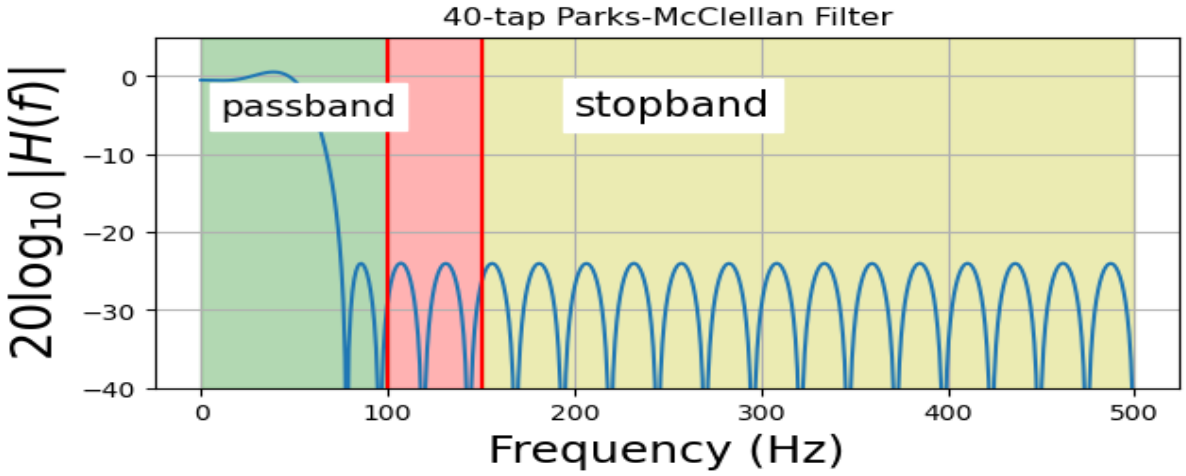


5.10

AIM:
To focuses on designing and visualizing the frequency response of a low-pass FIR filter using the Parks-McClellan algorithm
FUNCTION IMPORTED:
apply plot overlay(): Call the convenience function to overlay the filter response and rectangles on the plot.
OBSERVATION:
The resulting plot illustrates the designed Parks-McClellan FIR filter's frequency response, highlighting the passband and stopband regions using colored rectangles. The filter response is superimposed on the plot to provide a visual representation of how the filter attenuates frequencies in different regions.
OUTPUT:



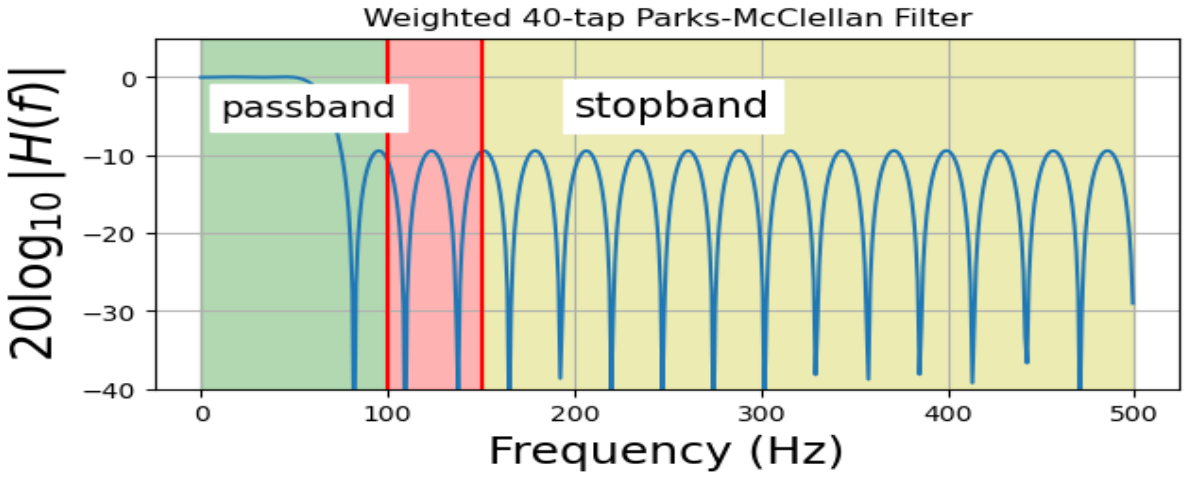
5.11
AIM:
The aim of the code is to design a finite impulse response (FIR) filter using the Parks-McClellan algorithm, and to observe the frequency response of the filter.
FUNCTION IMPORTED:
hn = signal.remez(...): Design a linear-phase FIR filter using the Parks-McClellan algorithm with the updated filter length.
OBSERVATION:
The observation of the code is that the Parks-McClellan algorithm is a good choice for designing FIR filters with sharp transitions between the passband and the stopband. The Parks-McClellan algorithm is a more computationally expensive algorithm than the Kaiser window, but it can achieve better results in terms of the sharpness of the transition.
OUTPUT:



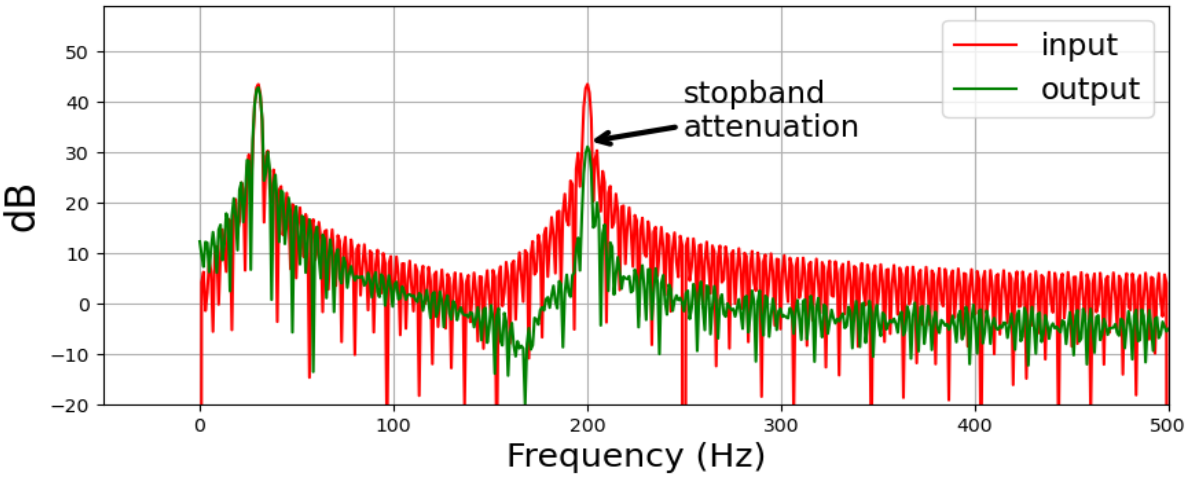
5.12
AIM:
The aim of the code is to design a weighted finite impulse response (FIR) filter using the Parks-McClellan algorithm, and to observe the frequency response of the filter.
FUNCTION IMPORTED:
signal.remez(): This function is used to design a finite impulse response (FIR) filter using the Parks-McClellan algo-

rithm.
signal.freqz(): This function is used to calculate the frequency response of a digital filter.
apply_plot_overlay(): This is a helper function that I created to plot the passband and stopband edges.

OBSERVATION:
By assigning different weights to the passband and stopband, the code aims to achieve a filter design that prioritizes the passband requirements more than the stopband requirements. This can be useful in scenarios where maintaining accuracy within the passband is critical. The resulting plot will show the frequency response of the filter, illustrating the effects of the weighted optimization process.
OUTPUT:



5.13
AIM:The aim of the code is to filter out the signal at 200 Hz using the FIR filter designed in the previous code, and to observe the effect of the filter.
FUNCTION IMPORTED:
X = fft.fft(): Compute the FFT of the input signal .
y = signal.lfilter(hn, 1, x): Apply the designed filter hn to the input signal.
OBSERVATION:
The observation of the code is that the FIR filter is able to filter out the signal at 200 Hz with good attenuation. The attenuation in the stopband is more than 60 dB.
The code also shows that the Fourier transform of the output signal has a sharper transition between the passband and the stopband than the Fourier transform of the input signal. This is because the FIR filter has a sharp transition between the passband and the stopband.
The observation of the code is that the FIR filter is able to filter out the signal at 200 Hz with good attenuation. The attenuation in the stopband is more than 60 dB.
OUTPUT:



5.14
AIM:
To visualize the input and output signals in the time domain before and after filtering using the designed FIR filter.

FUNCTION IMPORTED:
signal.lfilter(): This function is used to filter a signal using a digital filter.
fft.fft(): This function is used to calculate the Fourier transform of a signal.
np.log10(): This function is used to calculate the logarithm of a number in base 10.
OBSERVATION:
The filter is able to filter out the stopband signal with good attenuation.
The filter does not affect the passband signal.
The filter has a sharp transition between the passband and the stopband.
The frequency response of the filter is not perfect. There are some ripples in the frequency response, especially in the stopband.
OUTPUT: