```
'''
authour: Mayur Kamat
affiliation: 201104032, TE-E&TC Engg. Sem V, 2021-22, GEC
last updated: 15/10/2022
'''

#importing necessary functions from libraries
from matplotlib import pyplot as plt
from matplotlib.widgets import Slider
from numpy import cos, abs, array, zeros
from math import pi
from scipy.fft import fft, ifft
from plotconfig import *

#global (fig, ax) tuple, making it global makes it easier to update values and use
GUI
fig1, ax = plt.subplots()

#keeps track of the currently displayed plot
CurrentGraph = 0

#plots, calculates and updates the signals using the global variables from plotconfig
#which are updated in the update functions below
def plotSingals():
    global fig1, ax

    #calculating the message signals
    vm = amp*cos(2*pi*fm*time) + amp

    #quantizing the message signal
    q_signal = zeros(vm.size)
    for i in range(vm.size):
        for k in q_levels:
            if ((vm[i] >= k ) and (vm[i]< k+ step_size)):
                q_signal[i] = k

    #encoding the signal
    encoded_levels =[bin(int(q))[2:] for q in range(q_levels.size)]
    mapped_levels = dict(zip(encoded_levels,q_levels))
    print(mapped_levels)

    #calculating the FFT
    spectrum = fft(q_signal)

    #designing the ideal lowpass filter
    filter = array([0]*(frequency.size))
    for f in range(frequency.size):
        if frequency[f] > -(fm+10) and frequency[f] < fm+10:
            filter[f] = 1


    #multiplying the filters spectrum with the FDM spectrum to recover the message
    spectrum_filtered = spectrum * filter

    #taking the inverse of the filtered spectrum to get the signal back
    vr = ifft(spectrum_filtered)

    #functions below plot the singals
    def plot_q_signal():
        ax.clear()
        ax.set_xlabel('time - (sec)')
        ax.set_ylabel('amplitude - (volts)')
        ax.set_title('message and quantized signal')
        ax.plot(time[:100], vm[:100], 'b', label='Message')
        ax.step(time[:100], q_signal[:100], 'r', label='Quantized signal')
        ax.yaxis.set_ticks(q_levels)

    def plot_spectrum():
        ax.clear()
        ax.set_xlabel('freq - (Hz)')
        ax.set_ylabel('amplitude - (volts)')
        ax.set_title('spectrum')
        ax.plot(frequency, abs(spectrum)/N, 'b', label='Quantized signal spectrum')

    def plot_recovered():
        ax.clear()
        ax.set_xlabel('time - (sec)')
        ax.set_ylabel('amplitude - (volts)')
```

```
        ax.set_title('Recovered signal')
        ax.step(time[:100], q_signal[:100], 'b', label='Quantized siganl')
        ax.plot(time[:100], vr[:100], 'r', label='Recovered signal')
        ax.yaxis.set_ticks(q_levels)

    def plot_messageAndRecovered():
        ax.clear()
        ax.set_xlabel('frequency - (hertz)')
        ax.set_ylabel('Amplitude - (volts)')
        ax.set_title('message and recovered signal')
        ax.plot(time[:100], vm[:100], 'b', label='Message')
        ax.plot(time[:100], vr[:100], 'r', label='Recovered signal')


    #dictionary to call the plotting functins as and when the graph slider value changes
    GraphSelector = {
        0 : plot_q_signal,
        1 : plot_spectrum,
        2 : plot_recovered,
        3 : plot_messageAndRecovered,
        }

    GraphSelector.get(CurrentGraph)()

    #plot adjustments
    fig1.tight_layout(h_pad=2)
    fig1.set_size_inches(14, 7)
    plt.subplots_adjust(bottom=0.4)

    #draws the plot
    ax.grid(True)
    ax.legend()
    plt.draw()

def update_graph(val):
    global CurrentGraph
    CurrentGraph = val
    plotSingals()

#slider widgets
ax_graph = plt.axes([0.17, 0.27, 0.65, 0.03])
graph_Slider = Slider(ax_graph, 'Graph Select', valmin=0, valmax=3, valstep=1, valinit=0)

#plots the signal on run
plotSingals()

#handles updates on the sliders widgets
graph_Slider.on_changed(update_graph)

#needed in vscode to plot the fig in a new window...can be ignored in spyder
plt.show()

```