LAB CODE AND TITLE : SOFTWARE DEFINED RADIO LAB 19CCE284
EXPERIMENT NUMBER : 3
DATE : 25/04/2022

FRACTIONAL RATE CONVERSION

* AIM :

Using the concept of interpolation and decimation, perform the fractional rate conversion for achieving the given sampling rate. study the time domain and frequency domain characteristics.

* SOFTWARE REQUIRED :

Spyder IDE (3.9.7)

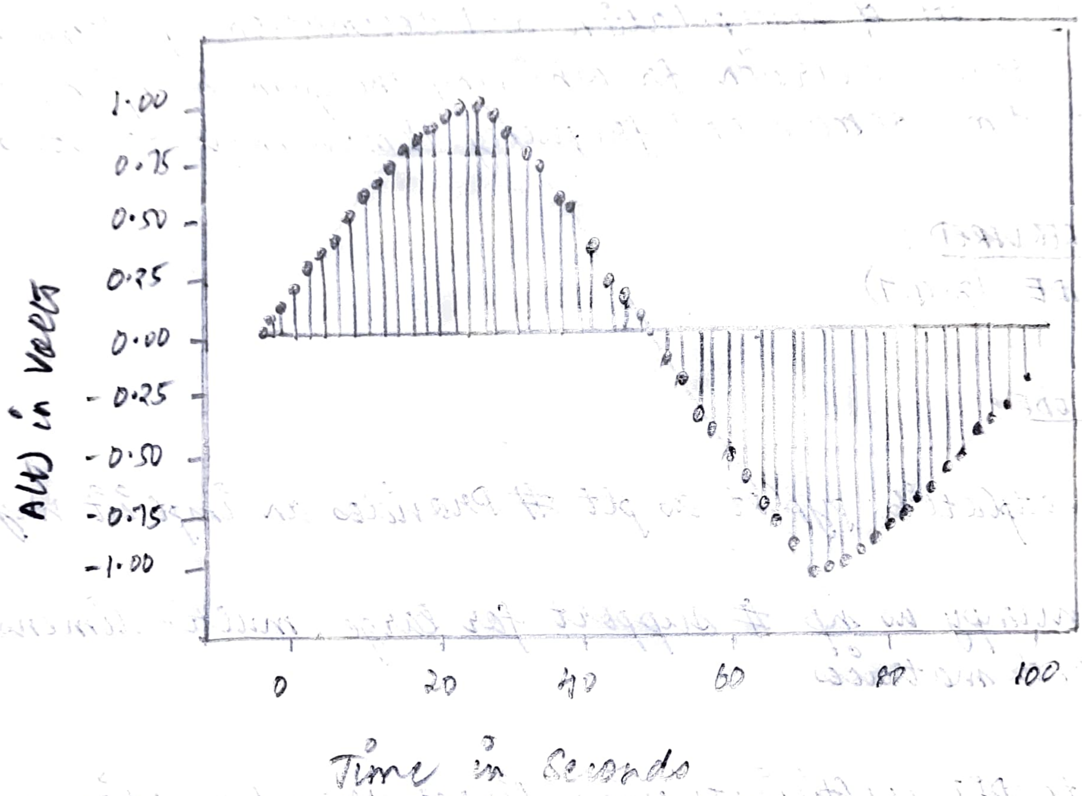
* PYTHON CODE :

```
import matplotlib.pyplot as plt # Provides an implicit way of
plotting
import numpy as np # support for large, multi-dimensional
array and matrices

# Compute DFT coefficients using linear transformation method:
def DFT (x, plot_name) :

    # Compute W(N) 1D Array :
    rl = cl = len (x)
    wn = []
    for i in range (rl) :
        for j in range (cl) :
            wn.append (np.exp (-2j * np.pi * i * j / len(x)))

    # numpy.reshape () is used to give a new shape to an array
without changing its data.
```

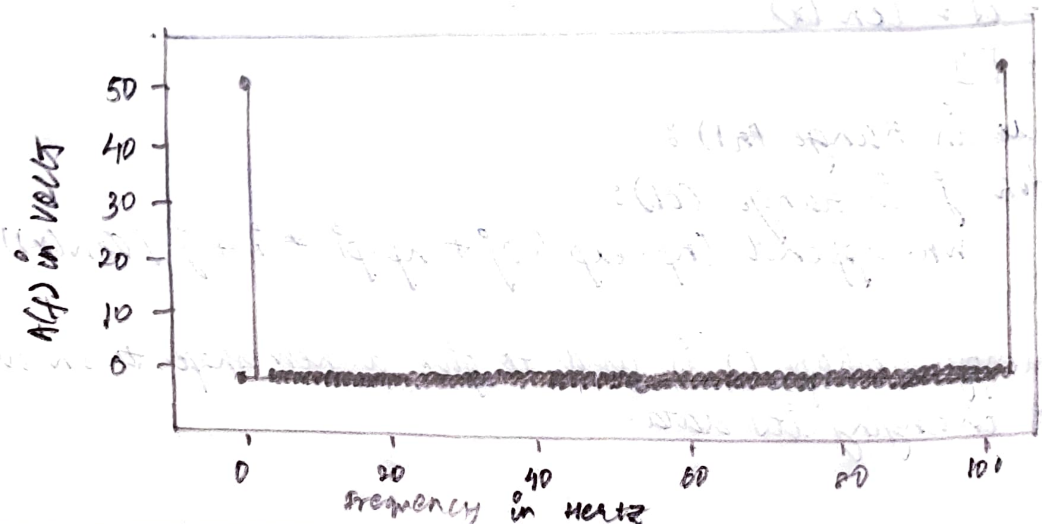
+ OUTPUT :

Enter the desired sampling frequency (in Hertz) : 100
Enter the frequency of the sine signal (in Hertz) : 1

Spectral Analysis of Original Signal
in Time Domain



Spectral Analysis of Original Signal
in Frequency Domain



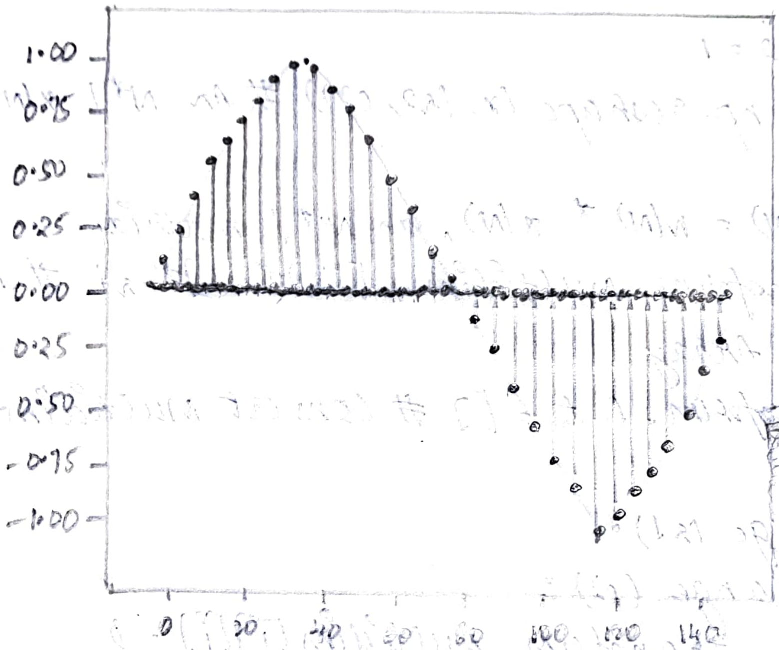
~~wn_multidim = np.reshape() is used to give a new shape to an array without changing its data.~~

```
wn_multidim = np.reshape(wn, (r1, c1)) # An N*N W(N) matrix
r2 = len(x); c2 = 1
x_multidim = np.reshape(x, (r2, c2)) # An N*1 x(N) matrix

# Compute X(N) = W(N) * x(N), an N*1 matrix
fourier_transform_multidim = [[0]*c2]* r1 # Null Multidimensional Array
fourier_transform_l_t = [] # Convert Multidimensional Array to 1D
for i in range(r1):
    for j in range(c2):
        fourier_transform_multidim[i][j] = 0
        for k in range(c1):
            fourier_transform_multidim[i][j] += wn_multidim[i][k] * float(x_multidim[k][j])
        fourier_transform_l_t.append(abs(fourier_transform_multidim[i][j]))

plt.xlabel("Frequency in Hertz")
plt.ylabel("A(f) in volts")
plt.title("Spectral Analysis of " + str(plot_name) + " in Frequency Domain")
plt.stem(np.arange(0, len(fourier_transform_l_t)), fourier_transform_l_t)
plt.grid(True)
plt.show()
```

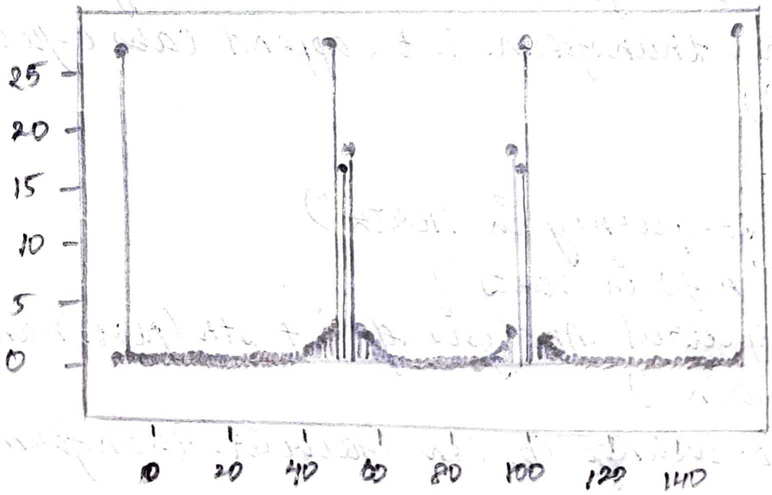
PTO

Enter the value of fractional rate conversion : 1.5

Spectral Analysis of Final Plot in Time Domain



Spectral Analysis of Final Plot in Frequency Domain



The original signal has been up-sampled by a factor of 3 and down-sampled by a factor of 2.

```
# sketch the spectrum for given sampling rate in time domain:
def time_domain (signal, plot_name):

    plt.xlabel ("Time in seconds")
    plt.ylabel ("A(t) in volts")
    plt.title ("Spectral Analysis of " + str(plot_name)+ "in
Time Domain")
    plt.stem (np.arange (0, len(signal)), signal)
    plt.grid ()
    plt.show ()

    DFT (signal, plot_name) # Frequency domain analysis of the
given signal

def down_sample (signal, m):
    down_sampler = []
    # Copy the element value from original signal to down
sampler array with the increment value set to "m" -
    for i in range (0, len(signal), m):
        down_sampler.append (signal[i])
    title = "Final Plot"
    time_domain (down_sampler, title)

def up_sample (signal, m, l):
    up_sampler = []
    for i in range (len (signal)):
        up_sampler.append (signal[i]) # Copy the element value
from original signal to up sampler array
        If i != len(signal) - 1:
            for k in range (l - 1):
                up_sampler.append (0) # Insert "l-1" zeros
between the elements of the array
```

```
        else :
            break
    down_sample (up_sampler, m)

# Driver code - main ()

# Generate the sine signal :-
fs = int (input ("Enter the desired sampling frequency (in Hertz): "))
time_axis = np. arange (0, 1, 1/fs) # Define the time axis
sine_frequency = int (input ("Enter the frequency of the sine
signal (in Hertz): "))

title = "Original Signal"
original_signal = []
original_signal = np. sin (2 * np.pi * sine_frequency *
time_axis)
time_domain (original_signal, title)

fractional_rate = float (input ("\n Enter the value of fractional
rate conversion : "))
for x in range (1, 100, 1) :
    y = fractional_rate * x
    if y - int (y) == 0 :
        up_sample (original_signal, x, int (y))
        break

print ("\n The original signal has been up-sampled by a factor
of " + str(int(y)) + " and down-sampled by a factor of " +
str (x) + ".")
```

* **RESULT :**

Performed the fractional rate conversion for achieving the given sampling rate using the concept of interpolation and decimation. Also, studied the time domain and frequency domain characteristics and all simulation results were verified successfully.

9/5/24