LAB CODE AND TITLE : 19CCE284 - SOFTWARE DEFINED RADIO LAB
EXPERIMENT NUMBER : 6
DATE : 06/06/2022

WAVELETS :

* AIM :

Develop a code for signal denoising using Haar wavelets. Analyse the effectiveness of denoising for varying levels of decomposition. Repeat the analysis for varying degrees of noise added to the clean sinusoidal signal.

* SOFTWARE REQUIRED :

Spyder IDE - Python 3.9

* CODE :

```
import matplotlib.pyplot as plt # Provides an implicit way of plotting
import numpy as np # support for large, multi-dimensional arrays and matrices
import scipy.fftpack as sf # module to calculate discrete fast Fourier transform

# 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.

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 Declaration
fourier_transform_l_t = [] # Convert Multidimensional Array to 1D
for i in range(r1):
    for j in range(c2):
        fourier_transfer_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("Amplitude (in Volts)")
plt.title("" + 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()

# Down-sample the given signal by a factor of "m" :-
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])
    return down-sampler

# Up-sample the given sample by a factor of "l":-
def up-sample (signal, l):
    up-sampler = []
    for i in range (len (signal)):
        up-sampler. append (signal[i]) # Copy the element value
from original array to up sampler array
        if i != len (signal):
            for k in range (l-1):
                up-sampler. append (0) # Insert "l-1" zeros between
the elements of the array
        else:
            break
    return up-sampler

# Perform convolution, x[n] * h[n]:-
def convolution (x, h):

    # string padding refers to adding, usually, non-informative
characters to a string to one or both ends of it. This is most often
done for output formatting and alignment purposes, but it can
have useful practical applications. numpy. pad () function is used
to pad the numpy arrays.

    size = (len(x) + len (h)) - 1 # Compute the size of system response
    x = np. pad (x, (0, size - len (x)), 'constant')
    h = np. pad (h, (0, size - len (h)), 'constant')
    y = np. zeros (size, dtype = float) # Returns a new array of
given shape and type, with zeros.
```

```
    for i in range(size):
        for j in range(size):
            if i>=j:
                y[i] = float(y[i] + (float(x[i-j]) * float(h[j])))
    return y # system response y[n]

# Driver Code : main(); Execution starts here.

# 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): "))
original_signal = np.sin(2 * np.pi * sine_frequency * time_axis)

title = "original signal"
plt.xlabel("Time in Seconds")
plt.ylabel("A(t) in Volts")
plt.title("" + str(title) + " in Time Domain")
plt.stem(time_axis, original_signal)
plt.grid(True)
plt.show()

DFT(original_signal, title)
print("\n The spectral analysis of the original signal in time and
frequency domain is shown in the above plots.")

# Generate a Noise :- (Draw random samples from a normal
distribution)
y = np.random.normal(0, 0.2, np.size(original_signal)); # Additive
mixed Gaussian Noise
original_signal = np.add(original_signal, y)
```

```
plt.title("Histogram Representation - Gaussian Naise")
plt.hist(y) # Plot a histogram
plt.grid(True)
plt.show()


#Plot the Noisy signal:-
plt.ylabel('Amplitude (in Volts)')
plt.title('Noisy Sinusoidal Wave')
plt.stem(original_signal)
plt.grid(True)
plt.show()


#Perform spectral Analysis:-
x_f = abs(sf.fft(original_signal)) # Return the absolute value of
the fast Fourier transform
l= np.size(original_signal)
fr= (fs/2) * np.linspace(0, 1, int(l/2))
xl_m = (2/l) * abs(x_f[0:np.size(fr)]); # Compute Magnitude Spectrum


# Plot Magnitude Spectrum:-
plt.title("Spectrum of Noisy Signal")
plt.xlabel("Frequency (in Hertz)")
plt.ylabel('Magnitude (in dB)')
plt.stem(fr, 20* np.log10(xl_m))
plt.grid(True)
plt.show()


print("\n\n The spectral analysis of the noisy signal in time and
frequency domain is shown in the above plots.")

h1= [1/np.sqrt(2), 1/np.sqrt(2)]; h2 = [1/np.sqrt(2), 1/np.sqrt(2)]
# same coefficients
```

```
g1 = [1/np.sqrt(2), -1/np.sqrt(2)]; g2 = [-1/np.sqrt(2), 1/np.sqrt(2)]
# complementary coefficients
l_m_factor_expander = 2

decomposition = int(input("\n Enter the number of levels of
decomposition : "))
scaling_level = int(input("Enter the scaling level, the wavelet
noise should be removed (< level of decomposition): "))
for i in range(decomposition):

    # Analyze:-
    a = convolution(original_signal, h1); p = convolution(original_
signal, g1)
    b = down_sample(a, l_m_expander); q = down_sample(p,
l_m_expander)

    # synthesize:-
    c = up_sample(b, l_m_factor_expander); r = up_sample(q,
l_m_factor_expander)
    d = convolution(c, h2); s = convolution(r, g2)

    #Add:-
    denoised_signal = np.add(d,s)

    if i == (decomposition - scaling_level):
        denoised_signal = d

title = 'Denoised signal'
plt.ylabel('Amplitude (in Volts)')
plt.title(title);
plt.stem(denoised_signal)
plt.grid(True)
plt.show()
```

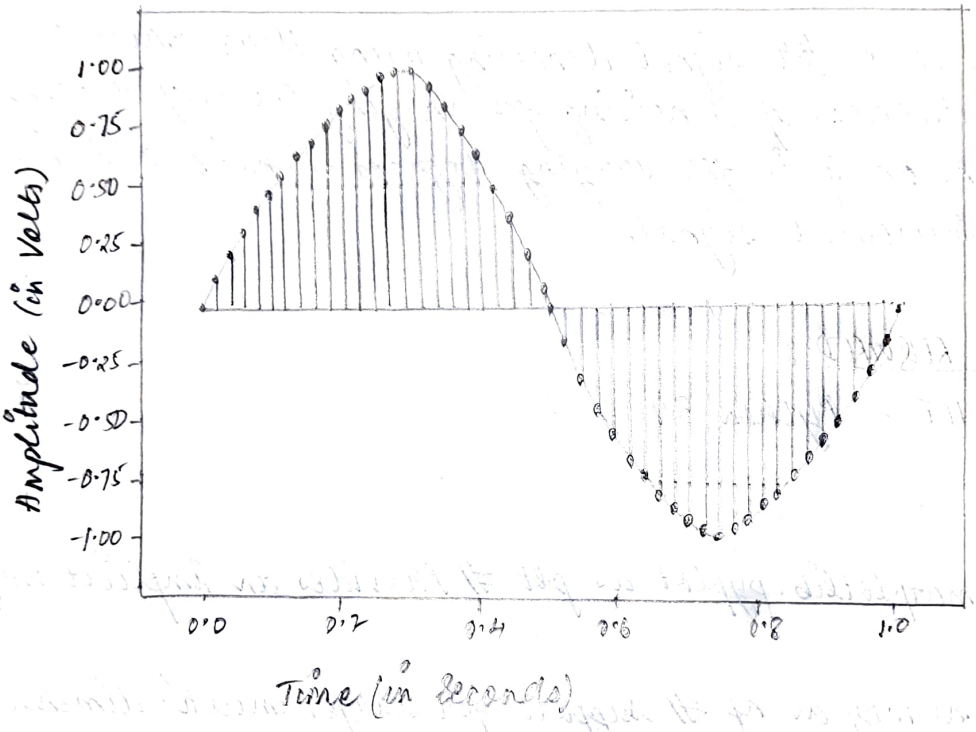
DFT (denoised_signal, title)
print ("In the spectral analysis of the denoised signal in time and frequency domain is shown in the above plots.")

* RESULT:
For the given sinusoidal signal, performed wavelet decomposition and reconstruction using Haar wavelet system and verified the perfect reconstruction property of wavelets. [one level decomposition using filter bank concept] All simulation results were verified successfully.

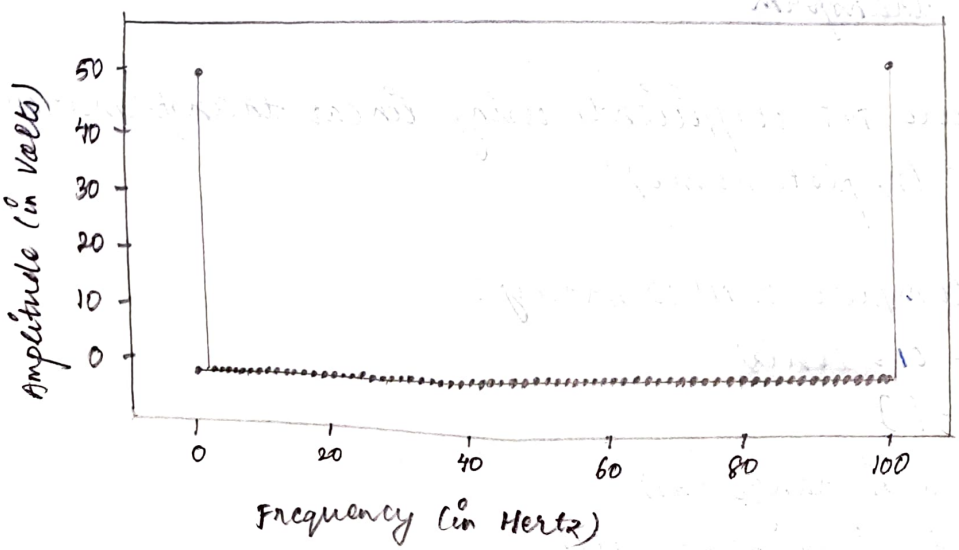
OUTPUT – CONSOLE WINDOW & PLOTS

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

Original signal in Time Domain

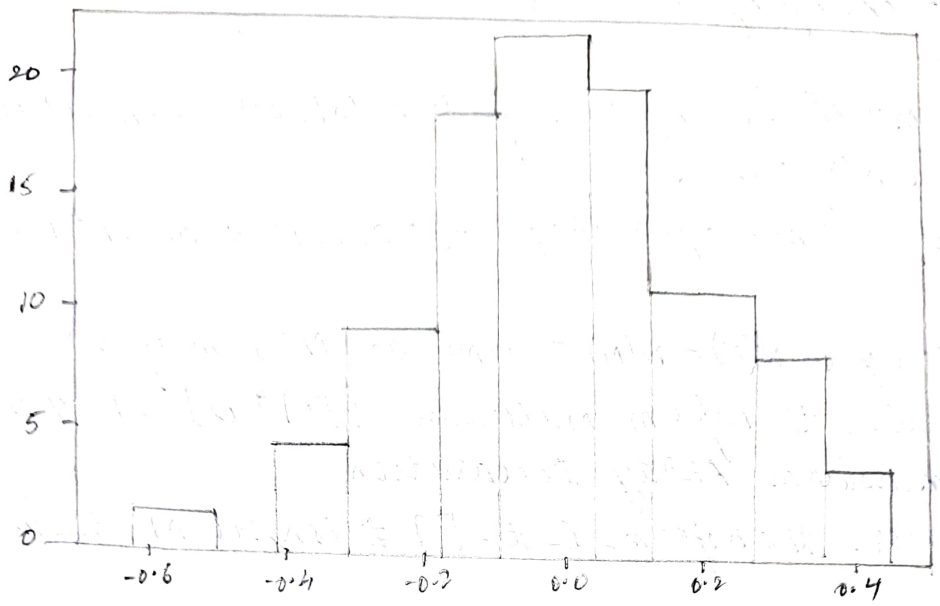


Original signal in Frequency Domain

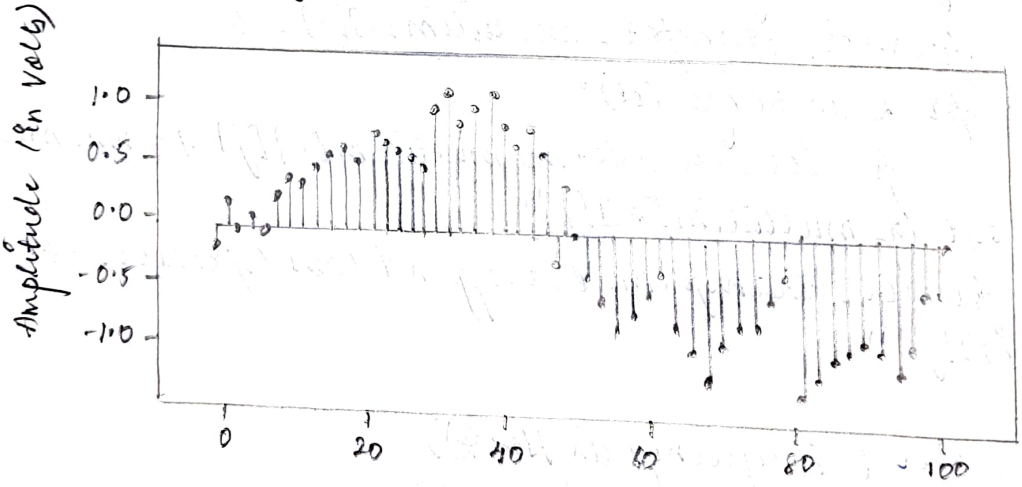


The spectral analysis of the original signal in time and frequency domain is shown in the above plots.

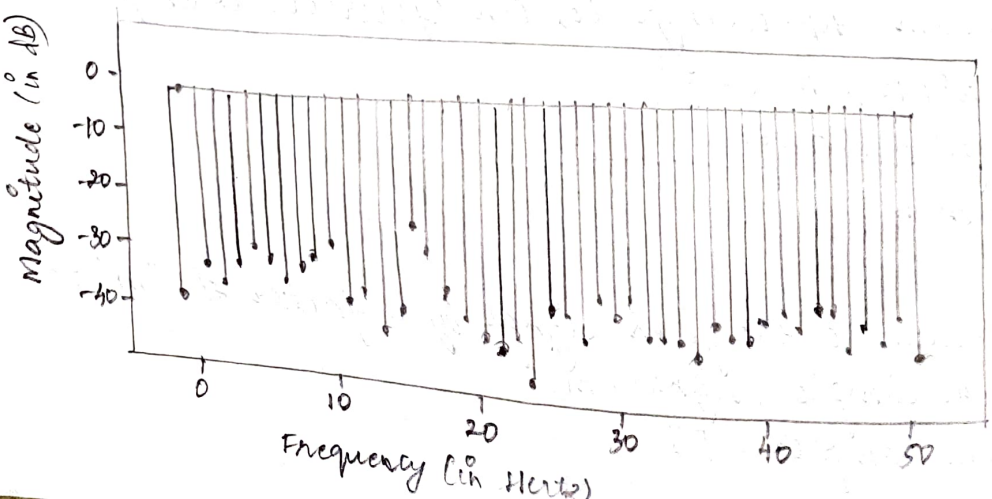
Histogram Representation - Gaussian Noise



Noisy Sinusoidal wave

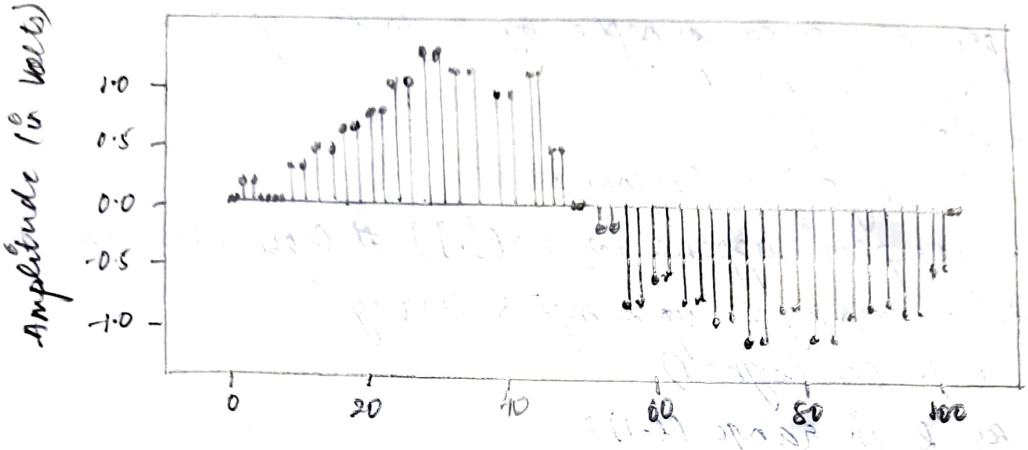


Spectrum of Noisy Signal

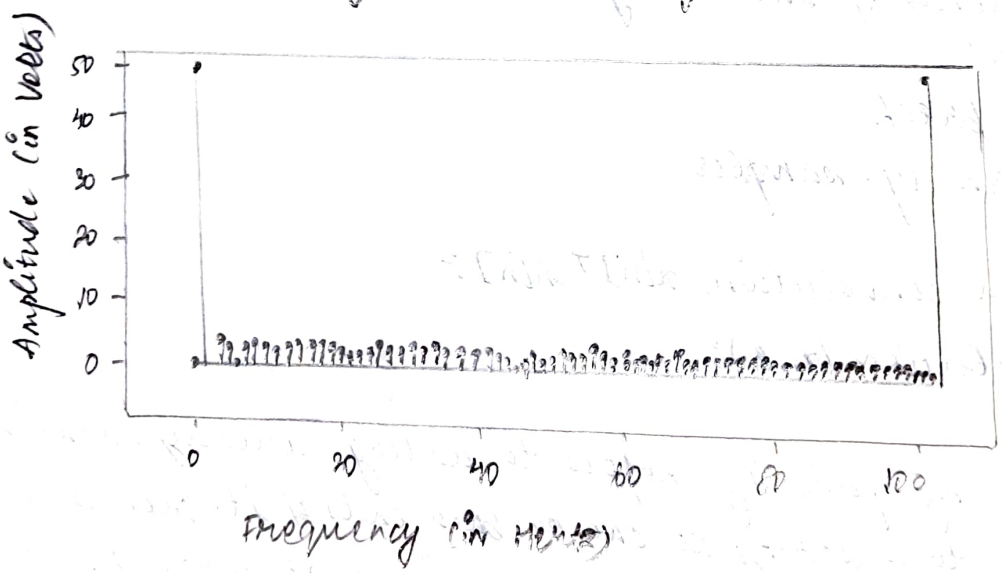


Enter the number of levels of decomposition : 100
Enter at which scaling level, the wavelet noise should be removed (< level of decomposition) : 1

Denoised Sinusoidal Signal



Denoised signal in Frequency Domain



The spectral analysis of the denoised signal in time and frequency domain is shown in the above plots.