DEPARTMENT OF ELECTRONIC AND TELECOMMUNICATION
UNIVERSITY OF MORATUWA

EN3551 Digital Signal Processing

This is offered as a "EN3551 Digital Signal Processing" module's partial completion.

Assignment 01 : Detecting Harmonics in Noisy Data and Signal Interpolation using DFT

200686J : Vishagar A.

9^{th} of september, 2023

Abstract

This report deals with the explanation of the solutions for the given questions in the assignment 01 of the EN3150 module. The solutions are explained in a way that it is easy to understand and follow. The solutions are explained with the help of the code snippets and the results. We are mainly focussed on some real world problems like "Detecting harmonics from a noisy data" and "Interpolation using DFT". ***And I have used python as the programming language to find the solutions for the given problems.***

Contents

1 Detecting Harmonics in Noisy Data

In this section we were given with a signal with four harmonics added with a heavy noise to find the harmonic frequency of the signal using Discrete Fourier Transform. Therefore to find these harmonics we were asked to use **DFT Averaging** technique.

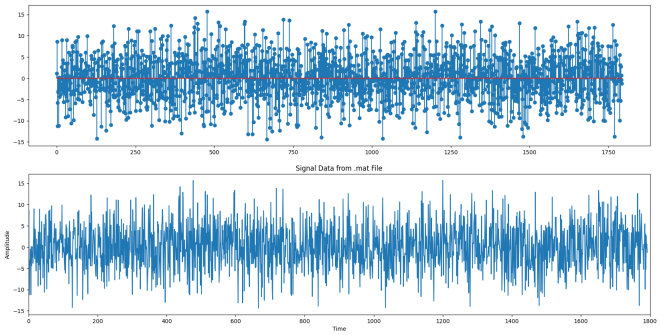

Figure 1: Provided sample file

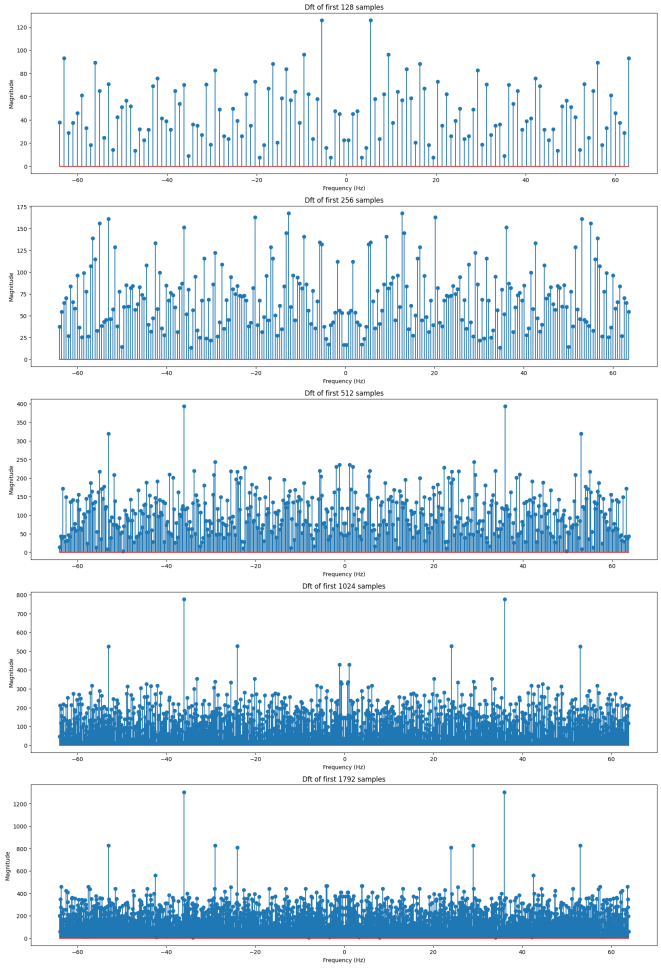

Figure 2: DFT for first 128,256,512,1024,1792 samples

Above showed images are the results for the first part of the question, where we asked to extract the first 128,256,512,1024,1792 samples from the given signal and find the DFT of those samples. As we can clearly see from the above images, **the DFT of the first 128 samples is not enough** to find the harmonic frequencies of the signal. The reason for this , **we are sampling at different points which may not capture the harmonic frequencies.** And the sampling points increases with the lenght of the signal. Therefore by this **may capture any missed frequency components.** And with the **increase of the signal length** we can see that the **DFT is getting more and more comprehensible** where we can see the harmonics distinctively.

And for the second part of the question we were asked to find the harmonic frequencies of the signal using DFT averaging technique. Following to this these are the procedures of **DFT Averaging**

- Partition the input signal {x[n], n = 0, 1, 2, ..., N - 1} into L equal subsets, with each subset having K samples. (N = LK)
- Apply DFT to each subset of the samples.
- Calculate the arithmetic mean of the gained sets of DFT sequences.
- Plot the average DFT sequence.
- Find the indices with higher frequencies.

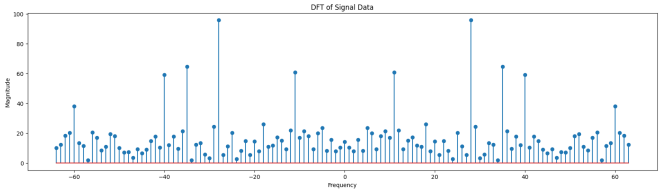

Figure 3: DFT after averaging, L = 14

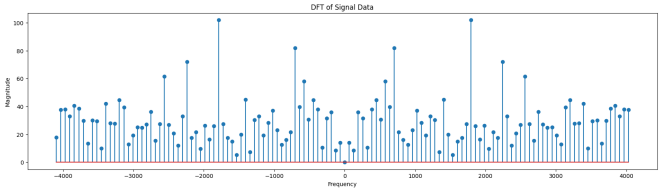

Figure 4: DFT after averaging, L = 4

For the selection of the 'L' value with keeping the 'K' value as 128. I have tried with different values and found that the **If we decrease the value less than 4 signal get uncomprehensible to find harmonics.**

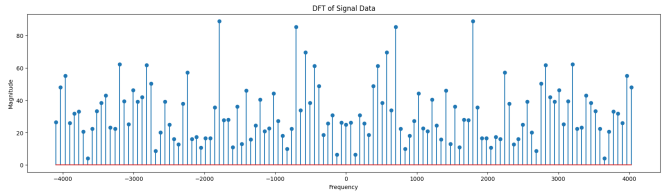

Figure 5: DFT after averaging, L = 3

Following were the harmonic frequencies that I have found from the DFT averaging technique.

- Selected harmony 1 : 11 Hz
- Selected harmony 2 : 28 Hz
- Selected harmony 3 : 35 Hz
- Selected harmony 4 : 40 Hz

But if we change the value of **'K' to 100** we cannot see the harmonics more clearly. And on the other hand if we increase the values of 'L' we can see that the **harmonics are getting more and more worse** with along the **computational complexity is getting higher and higher**.

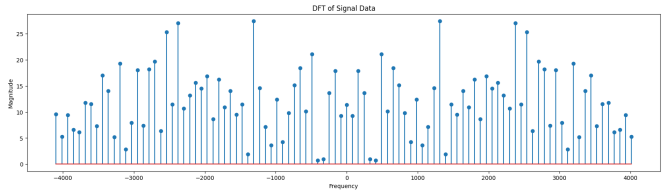

Figure 6: DFT after averaging, K =100

K represents the subset length, remaining constant across all subsets. It dictates how many samples are in each subset. When the sampling frequency is 128 Hz and K is set to 128, **each subset encompasses a full signal cycle, with all samples being identical in content.** However, using different K values like 100 or 135 leads to issues. With K is not equal to 128, subsets may not capture complete signal cycles, causing sample misalignment. This misalignment disrupts harmonic contributions across subsets. For example, when K = 100, the first sample in one subset may not align with the same point in the signal cycle as the first sample in another subset, making harmonic extraction challenging during averaging in methods like DFT.

2 Signal Interpolation Using DFT

In this task we were given with a audio signal. We were asked to get the first 20000 samples from that audio signal and create further more small sub sample arrays. These arrays are similar to the original array where there are some data loss in the produced sub-arrays.

Our task was simple. We need to re-construct the original signal from the corrupted or else data-missed signal. for that we need to take the discrete fourier transform of the subsets, Interpolate it with zeroes and find the inverse DFT and compare with our original singal.

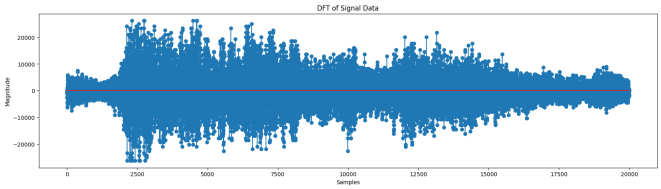

Figure 7: Extracted Samples from Audio

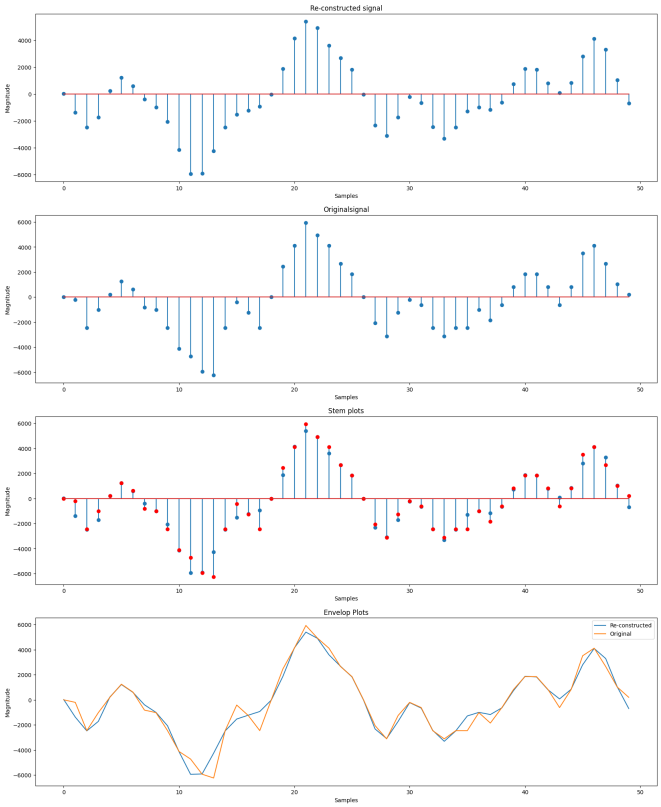

Figure 8: Reconstructed Signal by Interpolating x_2[n] where k = 1

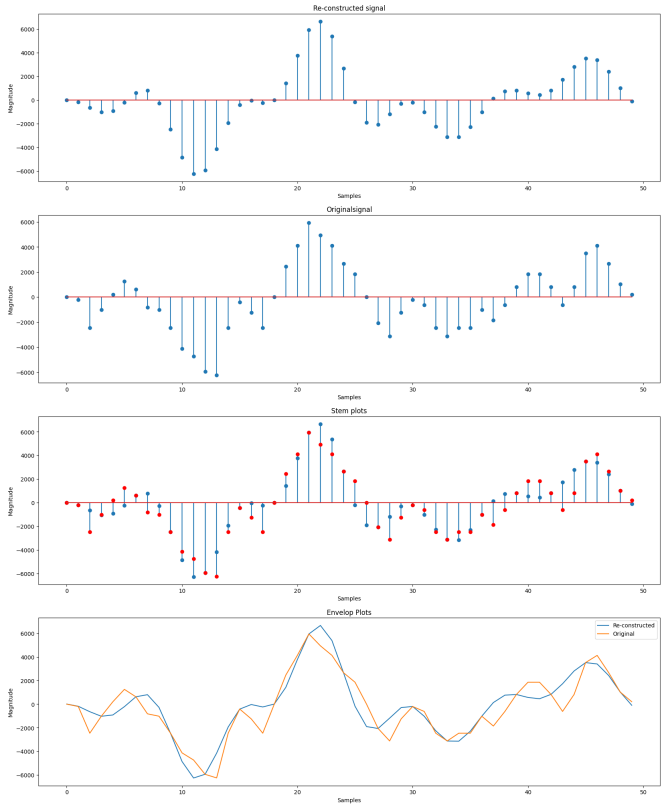

Figure 9: Reconstructed Signal by Interpolating x_3[n] where k = 2

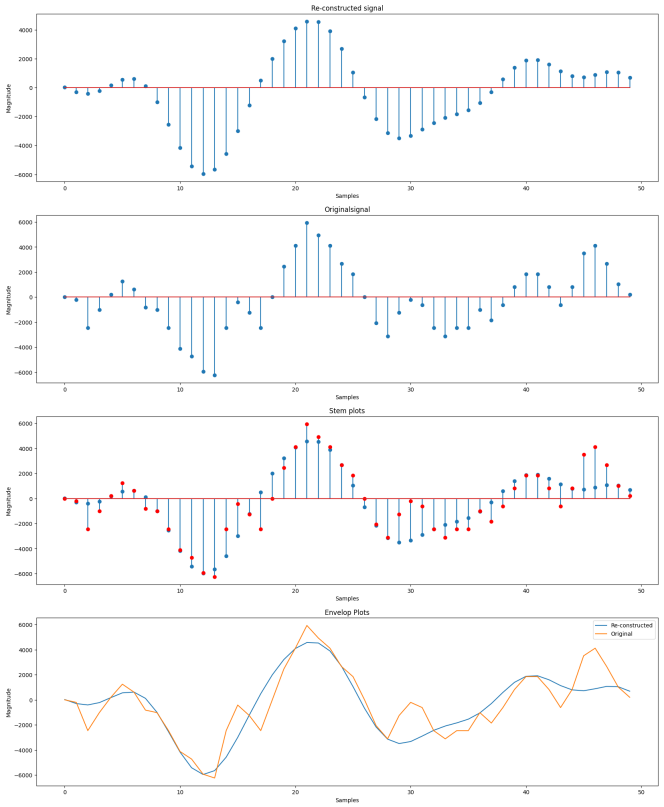

Figure 10: Reconstructed Signal by Interpolating x_4[n] where k = 3

2-Norm values of the differences

- between x and x_2 : 4.065
- between x and x_3 : 7.883
- between x and x_4 : 9.267

From the above results we can see that we have a **lower norm value for the x_2 signal.** Therefore we can say that the difference between the values of the original signal and x_2 signal is less compared to other signals which we gained from the interpolation. **Which also shows that x_2 is the best match for the original signal.**

Numerical Precision and Data Truncation may be the cause for the mismatches between the original and re-constructed signal using interpolation method in frequency doamin.

3 References

- Numpy Documentation
- Preferred Text Book

4 Github Repository

Following is the link to my Github repository for this assignment.

Github/EN3551_Assignment_01

5 Appendix

Code for Task 01

```
import numpy as np
import matplotlib.pyplot as plt
import scipy.io

# Load the reuired data
mat_data = scipy.io.loadmat('signals/signal686.mat')

# Access the variable from the loaded data
data = mat_data['xn_test']
data = np.squeeze(data)

# Visualize the data as a plot
fig,ax = plt.subplots(2,1,figsize=(20,10))
ax[0].stem(data)
ax[1].plot(data)
plt.xlabel('Time')
plt.ylabel('Amplitude')
plt.xlim(0,1800)
plt.title('Signal Data from .mat File')
plt.show()

# initialize constants
sample_frequency = 128
period = 14

# Constructing several subsets of the data

s_1 = data[0:129]
s_2 = data[0:257]
s_3 = data[0:513]
s_4 = data[0:1025]
s_5 = data[0:1793]

# Finding DFT
dft_result1 = np.fft.fft(s_1)
dft_result2 = np.fft.fft(s_2)
dft_result3 = np.fft.fft(s_3)
dft_result4 = np.fft.fft(s_4)
dft_result5 = np.fft.fft(s_5)


# Initialize Frequency Array
frequency_bins = [
    np.arange((-sample_frequency/2), (sample_frequency/2), sample_frequency/len(dft_result))
    for dft_result in [dft_result1, dft_result2, dft_result3, dft_result4, dft_result5]
]
# Visualize the magnitude of the DFT result
fig,ax = plt.subplots(5,1,figsize=(20,30))
ax[0].stem(frequency_bins[0], np.abs(dft_result1))
ax[1].stem(frequency_bins[1], np.abs(dft_result2))
ax[2].stem(frequency_bins[2], np.abs(dft_result3))
ax[3].stem(frequency_bins[3], np.abs(dft_result4))
ax[4].stem(frequency_bins[4], np.abs(dft_result5))

ax[0].set_title('Dft of first 128 samples')
ax[1].set_title('Dft of first 256 samples')
ax[2].set_title('Dft of first 512 samples')
ax[3].set_title('Dft of first 1024 samples')
ax[4].set_title('Dft of first 1792 samples')

ax[0].set_xlabel('Frequency (Hz)')
ax[0].set_ylabel('Magnitude')
ax[1].set_xlabel('Frequency (Hz)')
ax[1].set_ylabel('Magnitude')
ax[2].set_xlabel('Frequency (Hz)')
ax[2].set_ylabel('Magnitude')
ax[3].set_xlabel('Frequency (Hz)')
ax[3].set_ylabel('Magnitude')
ax[4].set_xlabel('Frequency (Hz)')
ax[4].set_ylabel('Magnitude')

plt.show()
# DFT Averaging Method
# Initialize variables
original_array = data
L = 14

subsample_size = int(len(data)/L)
num_subsamples = len(original_array) // subsample_size

# Create Subsets with necessary samples
subsamples = [original_array[i * subsample_size:(i + 1) * subsample_size] for i in range(num_subsamples)]

# Calculation of DFT
dft_result_sub_samples = np.fft.fft(subsamples)
frequency_bins_sub_samples = np.arange((-sample_frequency/2),(sample_frequency/2),sample_frequency/subsample_size)

# Averaging DFT
average_dft = np.mean(dft_result_sub_samples, axis=0)
average_dft = np.fft.fftshift(average_dft)

# Finding the Harmonic Frequencies
sorted_array = np.argsort(np.abs(average_dft))[::-1]
top_indices = sorted_array[:8]
top_frequencies = top_indices - int(sample_frequency/2)
top_frequencies.sort()

for i in range(len(top_frequencies[4:])):
    print("Selected harmony",i+1,":", top_frequencies[4:][i],"Hz")

# Visualize the magnitude of the DFT result
fig,ax = plt.subplots(figsize=(20,5))
ax.stem(frequency_bins_sub_samples, np.abs(average_dft))
plt.xlabel('Frequency')
plt.ylabel('Magnitude')
plt.title('DFT of Signal Data')
plt.show()
```

Code for the Task 02

```
import matplotlib.pyplot as plt
import wave
import numpy as np

# Read the audio file
audio_file = wave.open('handel_audio.wav', 'r')

sample_frequency = audio_file.getframerate()
num_samples = audio_file.getnframes()
audio_data = audio_file.readframes(num_samples)
audio_array = np.frombuffer(audio_data, dtype=np.int16)

# Initailize the sample array
N = 20000
y_n = audio_array[:N]

# Creating Subsets
x_1 = y_n[0:N]
x_2 = y_n[0:N:2]
x_3 = y_n[0:N:3]
x_4 = y_n[0:N:4]

# Verifyng sets
print("shape of array : ",x_1.shape)
print("shape of array : ",x_2.shape)
print("shape of array : ",x_3.shape)
print("shape of array : ",x_4.shape)

# Visualizing the Imported signal
fig,ax = plt.subplots(figsize=(20,5))
ax.stem(y_n)
plt.xlabel('Samples')
plt.ylabel('Magnitude')
plt.title('DFT of Signal Data')
plt.show()

# Calculating the DFT
dft_result_x_1 = np.fft.fft(x_1)
dft_result_x_2 = np.fft.fft(x_2)
dft_result_x_3 = np.fft.fft(x_3)
dft_result_x_4 = np.fft.fft(x_4)
dft_result_y_n = np.fft.fft(y_n)


# Defining the Interpolation Procedure
def interpolation(array,k):
    N = len(array)
    N1 = int((N+1)/2)
    N2 = int(N/2)
    if N % 2 != 0:
        array_part_1 = np.array(array[:N1-1])
        array_part_2 = np.zeros(k*N)
        array_part_3 = np.array(array[N1:N])
        interpolated_array = np.concatenate((array_part_1,array_part_2,array_part_3))
    else:
        array_part_1 = np.array(array[:N2])
        array_part_2 = np.zeros(k*N-1)
        array_part_3 = np.array(array[N2+1:N])
        term = array[N2+1] / 2
        term = np.reshape(term,1)
        interpolated_array = np.concatenate((array_part_1,term,array_part_2,term,array_part_3))
    return interpolated_array

# Interpoating the Signals in Frequency Domain
interpolated_x_2 = interpolation(dft_result_x_2,1)
interpolated_x_3 = interpolation(dft_result_x_3,2)
interpolated_x_4 = interpolation(dft_result_x_4,3)

# Re-Constructinf the Original Signal
redefined_x_2 = np.fft.ifft(interpolated_x_2)
redefined_x_2 *= 2
redefined_x_3 = np.fft.ifft(interpolated_x_3)
redefined_x_3 *= 3
redefined_x_4 = np.fft.ifft(interpolated_x_4)
redefined_x_4 *= 4


redefined_x_n = np.fft.ifft(y_n)

# Visualizing the Signals
fig,ax = plt.subplots(4,1,figsize=(20,25))
ax[0].stem(redefined_x_2[:50])
ax[1].stem(y_n[:50])
ax[2].stem(redefined_x_2[:50])
ax[2].stem(y_n[:50],markerfmt='red')
ax[3].plot(redefined_x_2[:50] , label = "Re-constructed")
ax[3].plot(y_n[:50], label = "Original")

ax[0].set_xlabel('Samples')
ax[0].set_ylabel('Magnitude')
ax[0].set_title("Re-constructed signal")

ax[1].set_xlabel('Samples')
ax[1].set_ylabel('Magnitude')
ax[1].set_title("Originalsignal")

ax[2].set_xlabel('Samples')
ax[2].set_ylabel('Magnitude')
ax[2].set_title("Stem plots")

ax[3].set_xlabel('Samples')
ax[3].set_ylabel('Magnitude')
ax[3].set_title("Envelop Plots")
ax[3].axis('on')

plt.legend()
plt.show()

fig,ax = plt.subplots(4,1,figsize=(20,25))
ax[0].stem(redefined_x_3[:50])
ax[1].stem(y_n[:50])
ax[2].stem(redefined_x_3[:50])
ax[2].stem(y_n[:50],markerfmt='red')
```

```
ax[3].plot(redefined_x_3[:50] , label = "Re
    -constructed")
ax[3].plot(y_n[:50], label = "Original")


ax[0].set_xlabel('Samples')
ax[0].set_ylabel('Magnitude')
ax[0].set_title("Re-constructed signal")

ax[1].set_xlabel('Samples')
ax[1].set_ylabel('Magnitude')
ax[1].set_title("Originalsignal")

ax[2].set_xlabel('Samples')
ax[2].set_ylabel('Magnitude')
ax[2].set_title("Stem plots")

ax[3].set_xlabel('Samples')
ax[3].set_ylabel('Magnitude')
ax[3].set_title("Envelop Plots")
ax[3].axis('on')

plt.legend()
plt.show()

fig,ax = plt.subplots(4,1,figsize=(20,25))
ax[0].stem(redefined_x_4[:50])
ax[1].stem(y_n[:50])
ax[2].stem(redefined_x_4[:50])
ax[2].stem(y_n[:50],markerfmt='red')
ax[3].plot(redefined_x_4[:50] , label = "Re
    -constructed")
ax[3].plot(y_n[:50], label = "Original")


ax[0].set_xlabel('Samples')
ax[0].set_ylabel('Magnitude')
ax[0].set_title("Re-constructed signal")

ax[1].set_xlabel('Samples')
ax[1].set_ylabel('Magnitude')
ax[1].set_title("Originalsignal")

ax[2].set_xlabel('Samples')
ax[2].set_ylabel('Magnitude')
ax[2].set_title("Stem plots")

ax[3].set_xlabel('Samples')
ax[3].set_ylabel('Magnitude')
ax[3].set_title("Envelop Plots")
ax[3].axis('on')

plt.legend()
plt.show()

difference = y_n[:50] - redefined_x_2[:50]
norm_2 = np.linalg.norm(difference, ord=2)
print("2-norm of the difference:", norm_2)

difference = y_n[:50] - redefined_x_3[:50]
norm_2 = np.linalg.norm(difference, ord=2)
print("2-norm of the difference:", norm_2)

difference = y_n[:50] - redefined_x_4[:50]
norm_2 = np.linalg.norm(difference, ord=2)
print("2-norm of the difference:", norm_2)
```