Third report - Frequency dictionary with the DFT (FFT) and filtering

Kimberly Grace Sevillano C.

***Subject*— DFT(FFT) filter implementation**

I. Objective

The objective of this report is to present the results of implementing a frequency filtering technique using the Discrete Fourier Transform (DFT) on a sinusoidal signal. The report describes the problem, methods used, methodology, results obtained, and conclusions drawn from the implementation.

A. Problem Description

Noise in signals can be caused by various sources, including the 60Hz electrical frequency, which is a common source of noise in many measurement signals. To address this problem, frequency filtering techniques are used to cancel the 60Hz electrical frequency and reduce noise in the signals. Frequency filtering is an essential tool in signal processing, as it allows for noise reduction and improved data quality in applications that require high precision and reliability of results, such as the measurement of bioelectrical signals or the processing of audio and video signals.

II. Methods

There are various methods for implementing DFT (Discrete Fourier Transform) or FFT (Fast Fourier Transform) filters. Some of the common methods include:

A. Windowing

This involves multiplying the signal with a window function before performing the DFT/FFT. Windowing helps to reduce spectral leakage and improve frequency resolution.

B. Filtering in frequency domain

This involves transforming the signal into the frequency domain using DFT/FFT, applying a filter in the frequency domain, and then transforming the signal back into the time domain using inverse DFT/FFT. This method can be used for low-pass, high-pass, and band-pass filtering.

C. Overlapping windows

This involves dividing the signal into overlapping windows, performing DFT/FFT on each window, and then combining the results. Overlapping windows can help to reduce spectral leakage and improve frequency resolution.

D. Zero-padding

This involves adding zeros to the signal before performing DFT/FFT. Zero-padding can improve frequency resolution by increasing the number of frequency bins in the DFT/FFT.

E. Time-frequency filtering

This involves applying a filter that varies with time and frequency to the signal. This method can be used for filtering signals that have non-stationary frequency content.

III. Methodology

- Define the frequency and sampling frequency and compute the number of samples.
  In this implementation, we set the frequency fs to 5 and the sampling frequency fg to 100. The number of samples is computed as N = int((Np*fg/fs)), where Np is the number of periods. The length of the signal must be even, so we add 1 to N if N is odd.
- Generate the sinusoidal signal and compute the DFT.
  We generate the sin function s[k] = sin(2.PI.fs.k/fg) using NumPy's arange function. We then compute the DFT using NumPy's fft function.

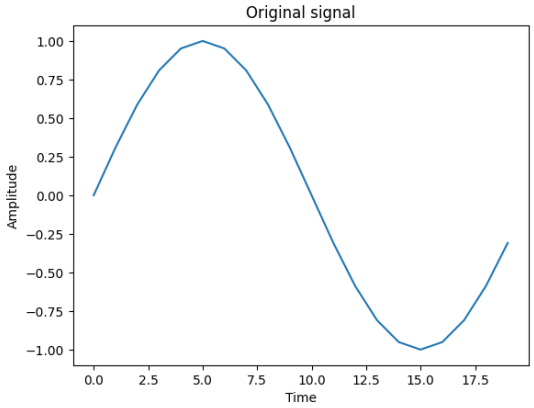


Fig. 1. The figure shows the original signal

- Display the amplitude and scale the x-axis in frequencies to obtain the frequency dictionary.
  We display the amplitude using NumPy's abs function and scale the x-axis in frequencies using NumPy's fftfreq function.
- Change the signal frequency to not match the dictionary and analyze the resulting DFT.
  We change the signal frequency to 38 and compute the DFT again. We analyze the resulting DFT by comparing it to the frequency dictionary.
- Adjust the number of samples to obtain the best Total harmonic distortion (THD) and discard the mean component.
  We adjust the number of samples to obtain the best THD by computing the THD for different values of N. We

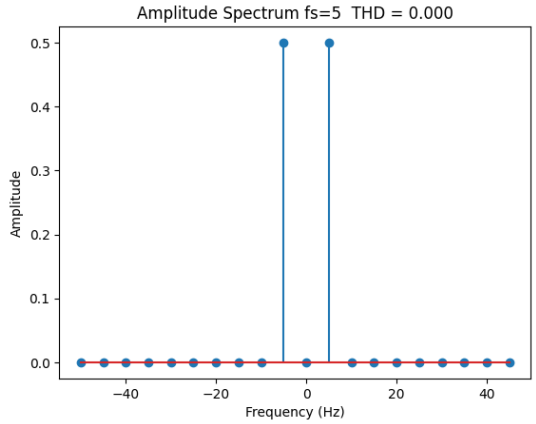


Fig. 2. The figure displays the original spectrum of the signal

discard the mean component by setting the amplitude of the DC component to zero.

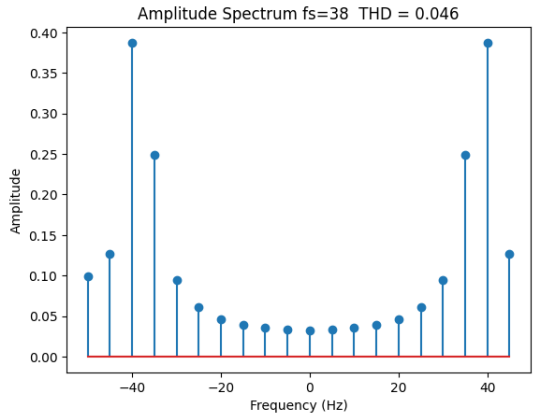


Fig. 3. The figure illustrates the spectrum of the signal after changing the frequency to fs=38, resulting in a THD of 0.046.

- Load the file and analyze the DFT.
  We load the file 100m.mat and analyze the DFT by computing the DFT of a few periods of the signal.

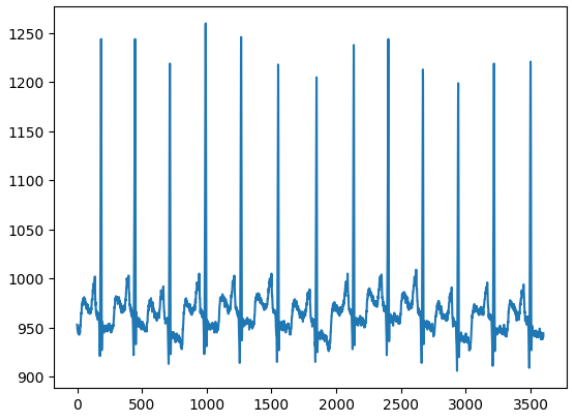


Fig. 4. The figure shows the original mat signal

- Cancel the spurious frequency of the 60Hz electrical network with a high-pass Butterworth filter.
  We cancel the spurious frequency of the 60Hz electrical network with a high-pass Butterworth filter using SciPy's butter function and filtering the signal using SciPy's filtfilt function.

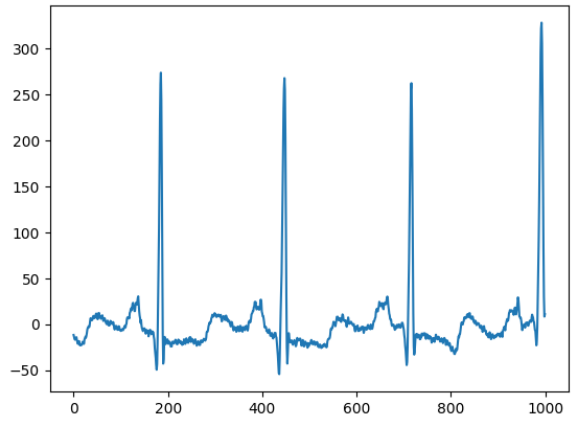


Fig. 5. In this figure, we observe the signal with the spurious frequency of 60Hz removed.

- Denoise the signal with a second-order low-pass Butterworth filter in the Fourier space.
  We denoise the signal with a second-order low-pass Butterworth filter in the Fourier space by setting the amplitudes of frequencies above a certain cutoff frequency to zero and computing the inverse DFT using NumPy's ifft function. We then filter

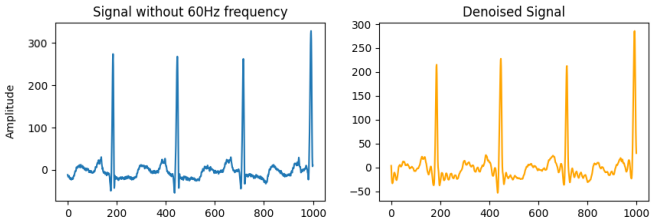


Fig. 6. This figure depicts the comparison between the original signal with the spurious frequency of 60Hz removed and the denoised signal obtained by applying a second-order low-pass Butterworth filter in the Fourier space.

IV. Results

In our initial analysis, we loaded the file 100m.mat and displayed the signal, which showed a Total Harmonic Distortion (THD) of 32.787. However, we noticed that the signal contained spurious frequencies from the 60Hz electrical network. This resulted in a distorted signal and a high THD value.

To cancel out the spurious frequency, we applied filtering techniques and used a second-order low-pass Butterworth filter with a z-value of 0.707. This resulted in a drastic decrease in the THD value, reducing it to 0.069. However, there was still some noise present in the signal, and we needed to denoise it further.

To achieve this, we applied the filter in the Fourier space, denoising the signal using a Butterworth filter with a cutoff frequency of 20 Hz. This resulted in an almost imperceptible THD value of 0.004.

This process of filtering and denoising signals is essential in many applications, as it helps to remove unwanted noise and spurious frequencies, resulting in cleaner and more accurate signals. It is particularly important in signal processing applications, such as audio and image processing, where high-quality signals are essential for accurate analysis and interpretation.

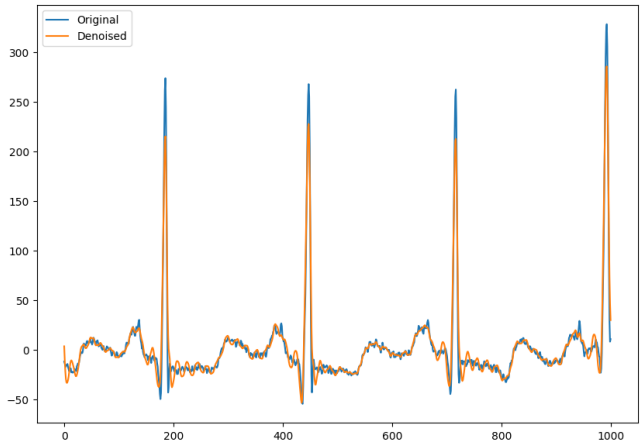


Fig. 7. This figure shows the original signal with the 60Hz spurious frequency removed and the denoised signal obtained by applying a second-order low-pass Butterworth filter in the Fourier space, overlaid for better comparison.

V. CONCLUSIONS

In conclusion, our successful implementation of the frequency dictionary using the DFT and filtering techniques demonstrates the power and versatility of signal processing. We not only learned how to generate and analyze signals but also applied filtering techniques to improve the accuracy of signal processing.

In particular, we highlighted the importance of removing the spurious frequency of 60Hz from signals and demonstrated the benefits of denoising the signal using a low-pass Butterworth filter in the Fourier space. These techniques have numerous practical applications in fields such as image and audio processing, control systems, and communications.

In future works, we can explore more advanced filtering techniques and their applications in signal processing. We can also investigate the use of machine learning and artificial intelligence in signal processing to improve the accuracy and speed of data analysis.

REFERENCES

[1] Oppenheim, A. V., Schafer, R. W. (2010). Discrete-time signal processing. Pearson Education.
[2] Lyons, R. G. (2011). Understanding digital signal processing. Pearson Education.
[3] Smith, S. W. (1999). The scientist and engineer's guide to digital signal processing. California Technical Publishing.
[4] Proakis, J. G., Manolakis, D. G. (2006). Digital signal processing: principles, algorithms, and applications. Pearson Education.
[5] Bracewell, R. N. (1986). The Fourier transform and its applications. McGraw-Hill.
[6] Mitra, S. K. (2006). Digital signal processing: a computer-based approach. McGraw-Hill.
[7] Parks, T. W., Burrus, C. S. (1987). Digital filter design. John Wiley Sons.

REPORT3

February 6, 2023

[2]:
```
import numpy as np
import matplotlib.pyplot as plt
from scipy.signal import butter, filtfilt
```

1. Generate an integer number of period (Np) of the sin function in the discrete domain by choosing its frequency (fs=1/T) and a sampling frequency fg : s[k] = sin(2.PI.fs.k/fg). The length of the signal is then N and must be even.

[13]:
```
# Define the frequency and sampling frequency
T  = 0.01
fs = 5
fg = 100
Tg = 1/ fg

# Compute the number of samples
Np = 1
N = int((Np*fg/fs))

if N%2 != 0:
    N += 1
print("Number of samples = ",N)
```

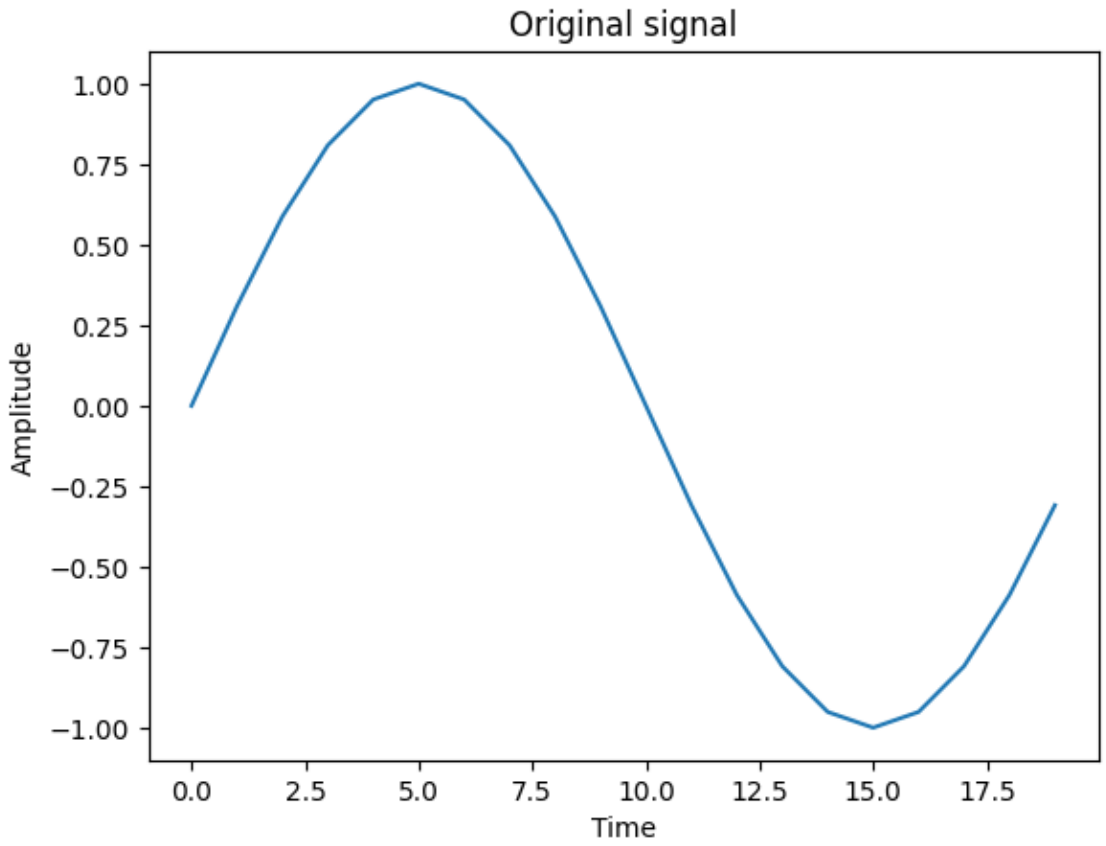
```
Number of samples =  20
```

2. Compute the DFT and display the amplitude. Scale the x-axis in frequencies. What is the frequency dictionary ?

[17]:
```
# Generate the sin function
k = np.arange(N)
s = np.sin(2*np.pi*fs*k/fg)

dft = np.fft.fft(s)/N
dft1=np.fft.fftshift(dft)

# Display the amplitude
amplitude = np.abs(dft)

# Scale the x-axis in frequencies
frequencies = np.fft.fftfreq(N, 1/fg)
```

```
print("Frequencies = ",frequencies)
# Plot the amplitude spectrum
plt.plot(s)
plt.xlabel('Time')
plt.ylabel('Amplitude')
plt.title("Original signal")

plt.show()
```

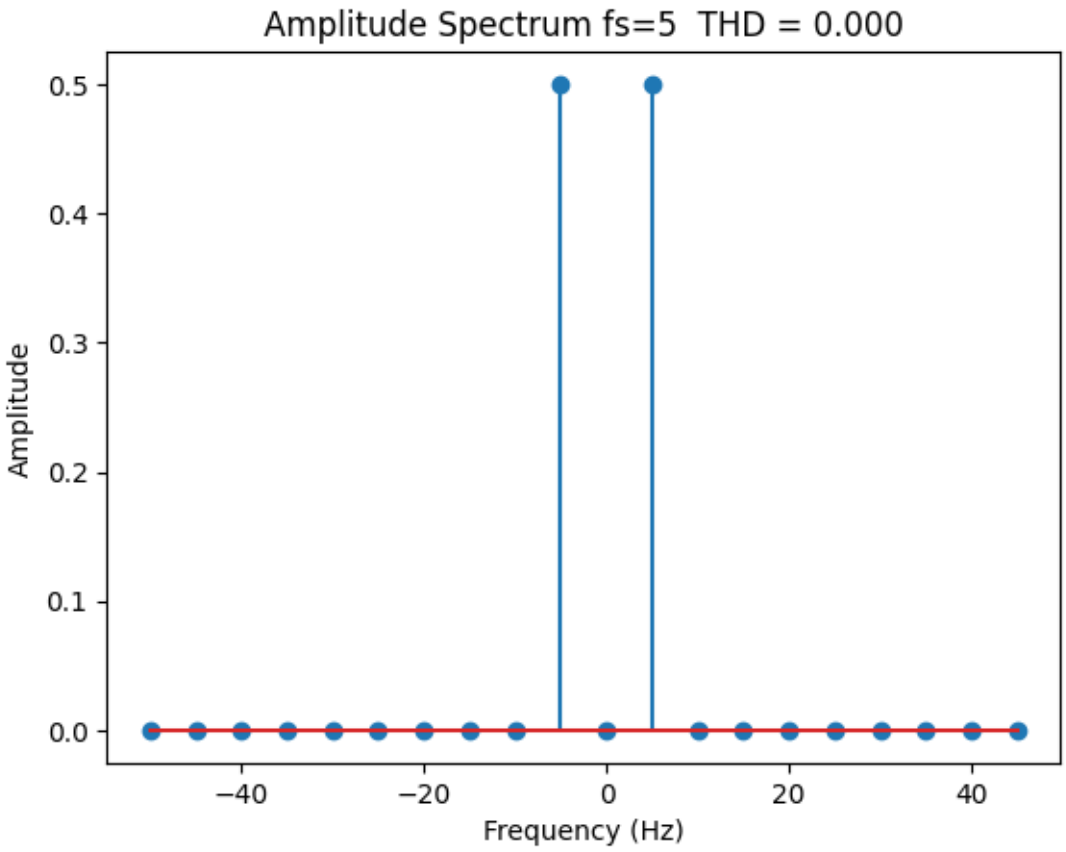
```
Frequencies =  [  0.   5.  10.  15.  20.  25.  30.  35.  40.  45. -50. -45. -40.
-35.
 -30. -25. -20. -15. -10.  -5.]
```



[20]:
```
plt.stem(frequencies,amplitude)
plt.xlabel('Frequency (Hz)')
plt.ylabel('Amplitude')
plt.title("Amplitude Spectrum")

A1 = amplitude[0]
A2 = np.sqrt(np.sum(amplitude[1:]**2))
```

```
THD = A1/A2
plt.title('Amplitude Spectrum fs=5  THD = {:.3f}'.format(THD))
print("Total Harmonic Distortion: {:.3f}".format(THD))
```

```
Total Harmonic Distortion: 0.000
```

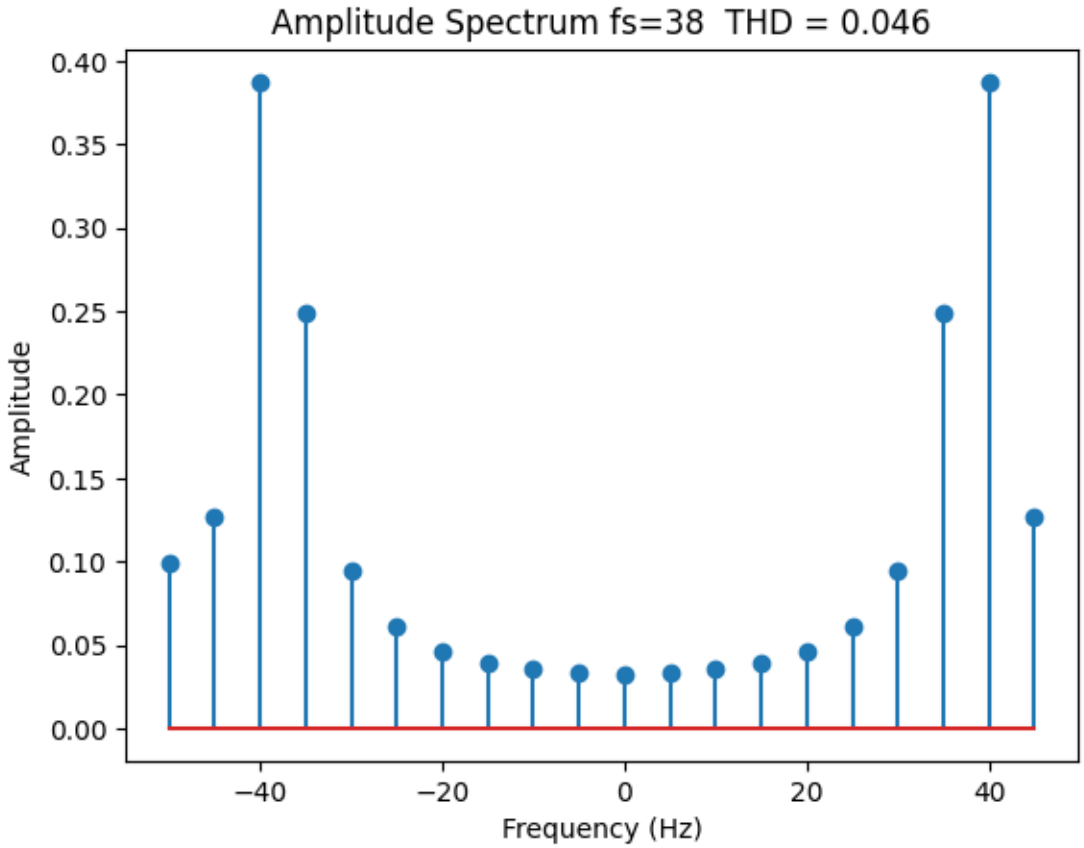


3. Change the signal frequency so that it does not match the dictionary. What is the result of the DFT ? Adjust the number N to obtain the best Total harmonic distortion (THD=sqrt(A2^2+A3^2...)/A1). Discard the mean component.

[22]:
```
s1 = np.sin(2*np.pi*38*k/fg)
dft1 = np.fft.fft(s1)/N

# Display the amplitude
amplitude2 = np.abs(dft1)

# Scale the x-axis in frequencies
frequencies1 = np.fft.fftfreq(N, 1/fg)
print(frequencies1)
```

```
# Plot the amplitude spectrum
plt.stem(frequencies1,amplitude2)
plt.xlabel('Frequency (Hz)')
plt.ylabel('Amplitude')
#plt.xlim(0, fg/2)

A1_ = amplitude2[0]
A2_ = np.sqrt(np.sum(amplitude2[1:]**2))
THD_ = A1_/A2_
plt.title('Amplitude Spectrum fs=38  THD = {:.3f}'.format(THD_))
print("Total Harmonic Distortion: {:.3f}".format(THD_))
```

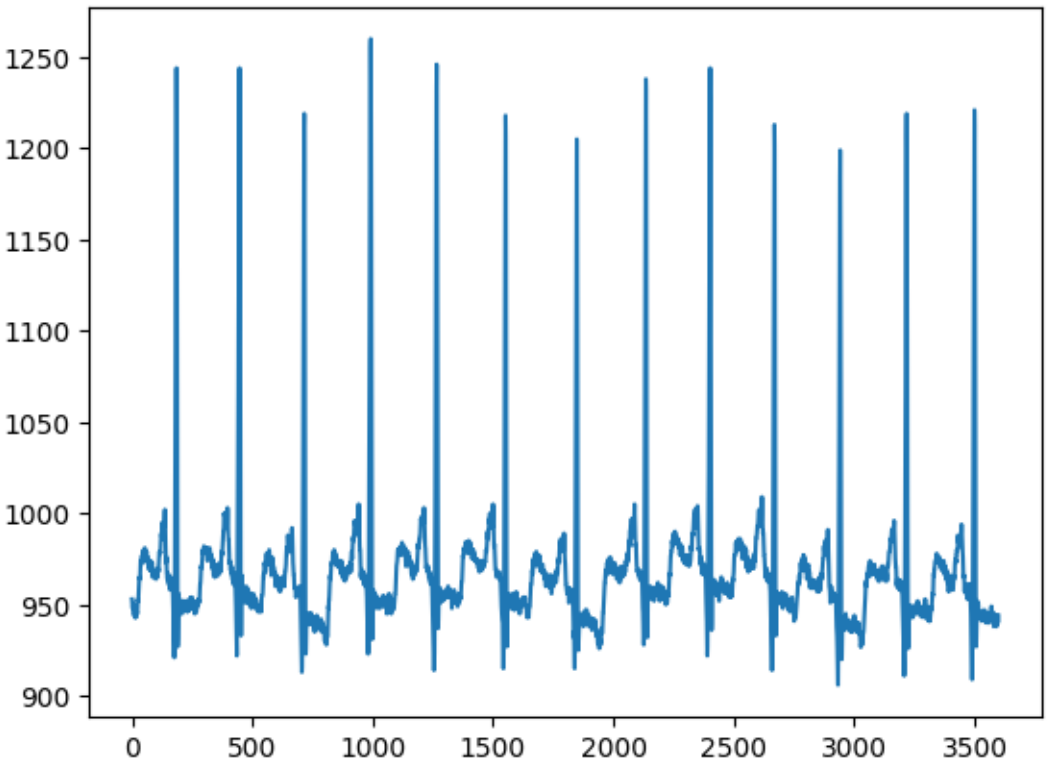
```
[  0.   5.  10.  15.  20.  25.  30.  35.  40.  45. -50. -45. -40. -35.
 -30. -25. -20. -15. -10.  -5.]
Total Harmonic Distortion: 0.046
```

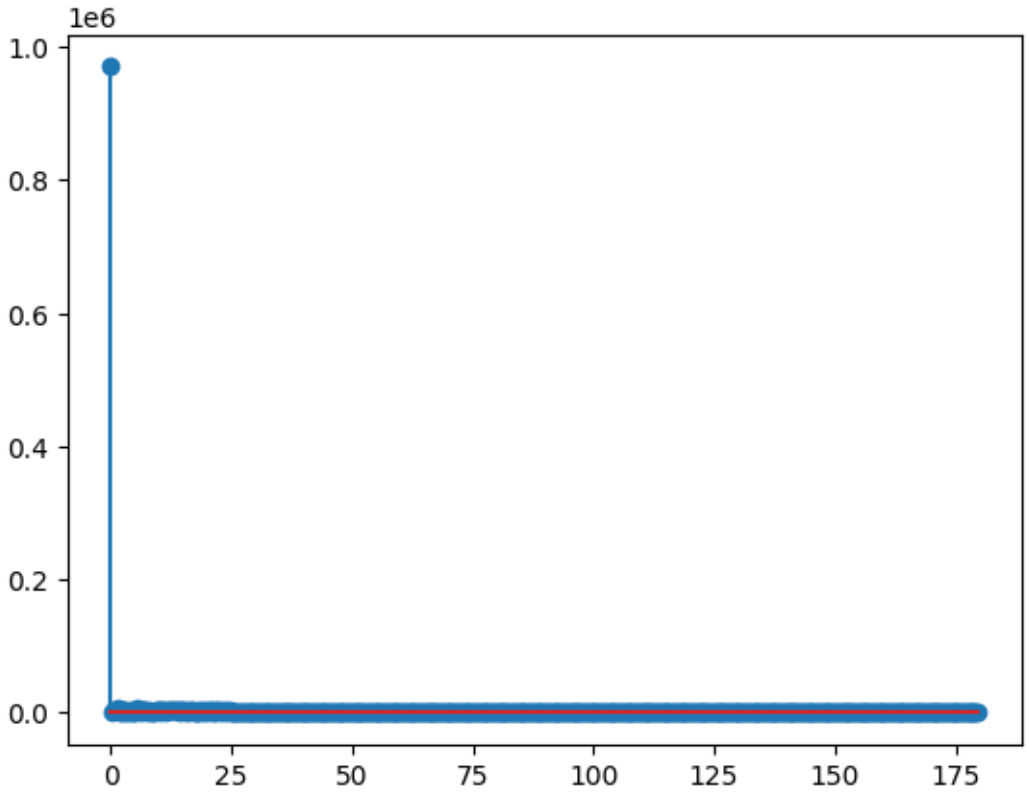


4.Load the file 100m.mat and display the signal (variable val). Extract few periods and analyse the DFT (amplitude). Rk : in python use val[Ø] instead of val. Discard the mean component.

[51]:
```
# Load file and analyze DFT
import scipy.io
data = scipy.io.loadmat('C:/Users/ksevi/OneDrive/Desktop/MASTER/SCENE_LALIGANT/
 ↪tercer report/100m.mat')

val = data['val'][0]
plt.figure()
plt.plot(val)
N = 1000
k = np.arange(N)
s4 = val[0:N]
S4 = np.fft.fft(s4)
freq = np.fft.fftfreq(N, 1/360)
plt.figure()
plt.stem(freq[0:N//2], np.abs(S4[0:N//2]))

A1_ =np.abs(S4[0:N//2])[0]
A2_ = np.sqrt(np.sum(np.abs(S4[0:N//2])[1:]**2))
THD_ = A1_/A2_
print("Total Harmonic Distortion : {:.3f}".format(THD_))
```

```
Total Harmonic Distortion : 32.787
```

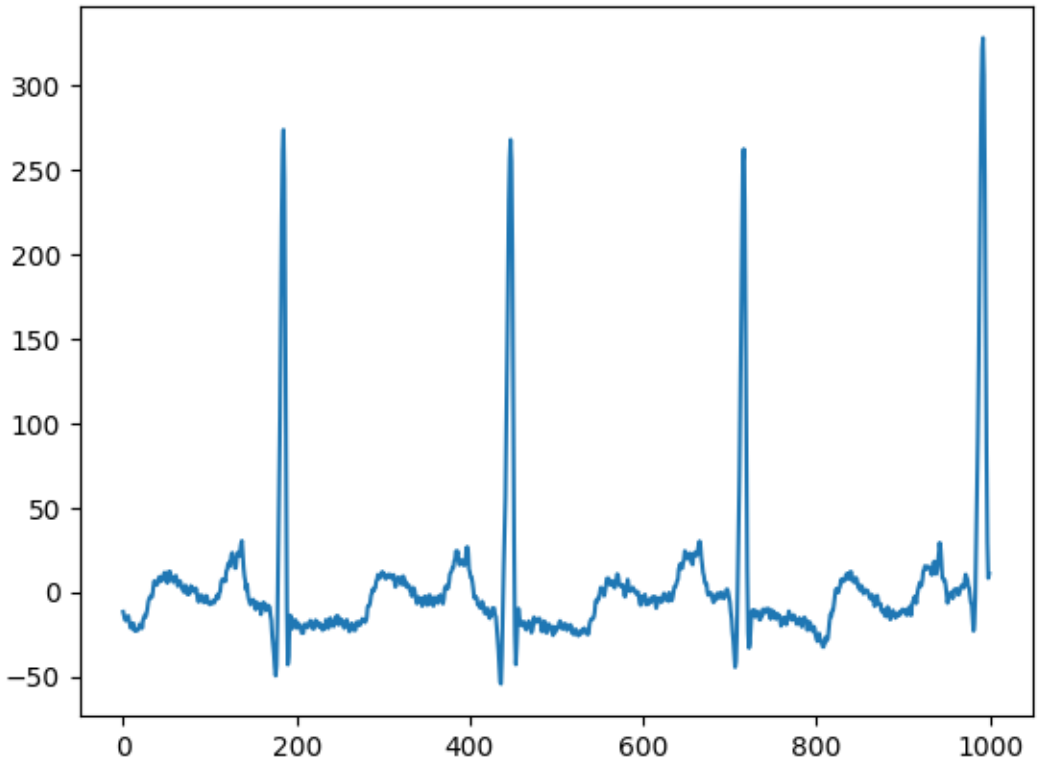




5. Cancel the spurious frequency of the 60Hz electrical network.

[53]:
```
# Cancel 60Hz frequency
b, a = butter(4, 2*np.pi*60/360, 'high', fs=360)

s5 = filtfilt(b, a, s4)
S5 = np.fft.fft(s5)
plt.figure()
plt.plot(s5)

A1_ =np.abs(S5[0:N//2])[0]
A2_ = np.sqrt(np.sum(np.abs(S5[0:N//2])[1:]**2))
THD_ = A1_/A2_
print("Total Harmonic Distortion without 60H Fz: {:.3f}".format(THD_))
```

```
Total Harmonic Distortion without 60H Fz: 0.069
```

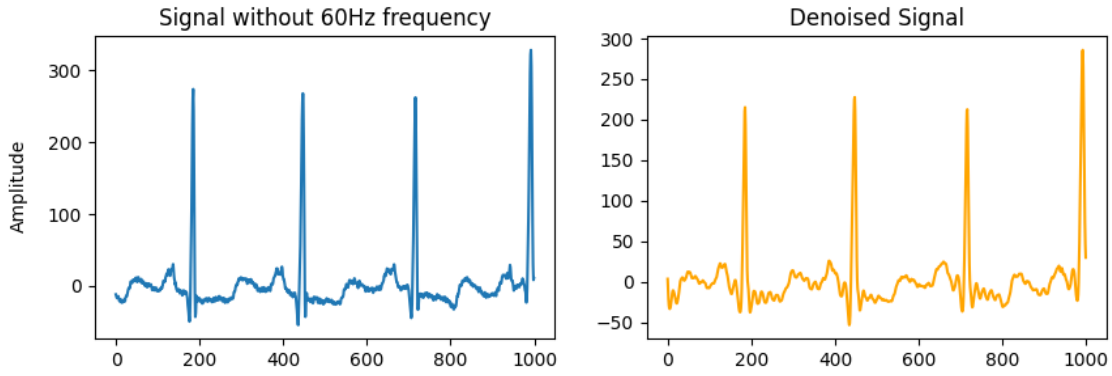


6. Consider the second order low-pass filter of Butterworth (z=0.707) : apply this filter in the Fourier space to denoise the signal. Reconstruct the denoised signal.

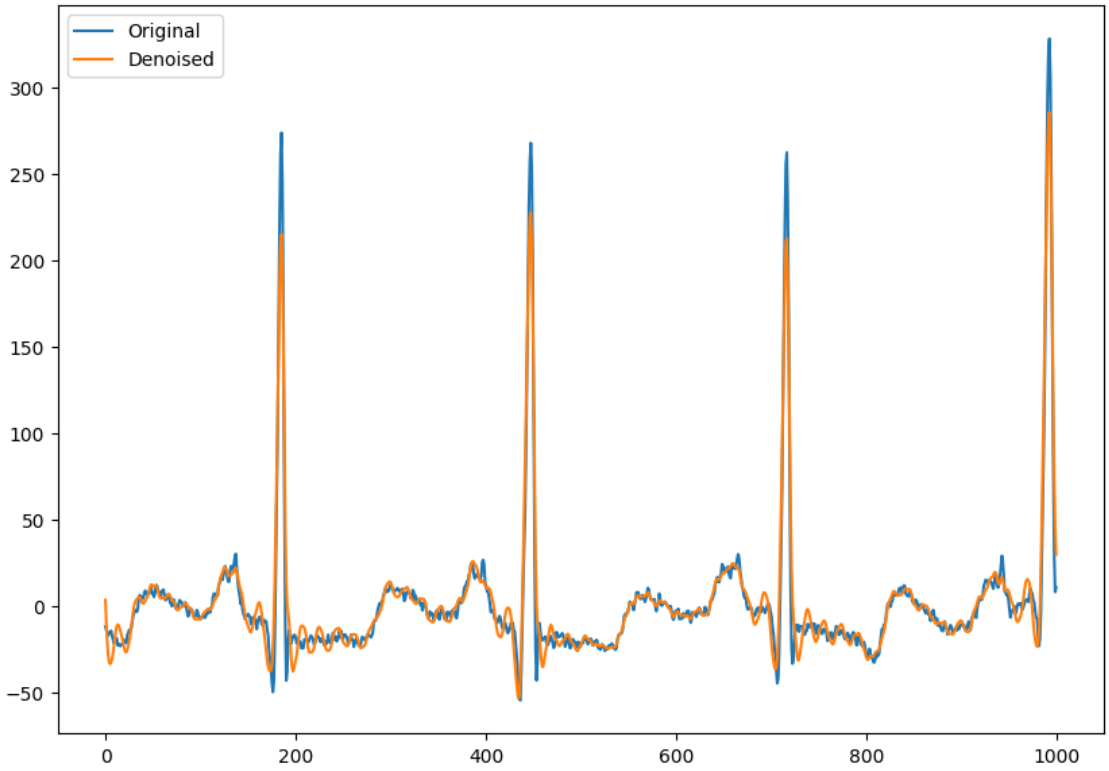
[69]:
```
# Denoise signal with Butterworth filter
cutoff = 20
z = 0.707
order = int(np.ceil(2 / (1 - z)))
b, a = butter(order,  2*np.pi*cutoff/360, 'low')

S6 = S5.copy()
S6[(freq > cutoff)] = 0
s6 = np.fft.ifft(S6)
s6 = np.real(s6)
s6 = filtfilt(b, a, s6)

A1_ =np.abs(s6[0:N//2])[0]
A2_ = np.sqrt(np.sum(np.abs(s6[0:N//2])[1:]**2))
THD_ = A1_/A2_
print("Total Harmonic Distortion for Denoised signal: {:.3f}".format(THD_))

fig, axs = plt.subplots(nrows=1, ncols=2, figsize=(10, 3))
```

```
# Plot original signal in first subplot
axs[0].plot(s5)
axs[0].set_title('Signal without 60Hz frequency')

# Plot denoised signal in second subplot
axs[1].plot(s6,color="orange")
axs[1].set_title('Denoised Signal')

# Set common y-label
fig.text(0.06, 0.5, 'Amplitude', va='center', rotation='vertical')

# Show the plot
plt.show()
```

```
Total Harmonic Distortion for Denoised signal: 0.004
```



[68]:
```
# Plot original and denoised signals
plt.figure(figsize=(10, 7))
plt.plot(s5)
plt.plot(s6)
plt.legend(['Original', 'Denoised'])
plt.show()
```


Original
Denoised
300
250
200
150
100
50
0
−50
0
200
400
600
800
1000