FREQUENCY DICTIONARY WITH DISCRETE FOURIER TRANSFORM AND FILTERING

- Atanda Abdullahi Adewale MSc. Computer Vision

1. **Objective**

This project provides an understanding of how to perform various operations on a discrete signal, such as computing its Discrete Fourier Transform DFT, analyzing the frequency spectrum, adjusting the signal frequency, canceling a spurious frequency, and denoising the signal. These skills can be applied to other signals in various domains, such as audio or image processing.

The Discrete Fourier Transform (DFT) of a sinusoidal signal s[k]= sin(2.pi.fs.k/fg) is given by the equation:

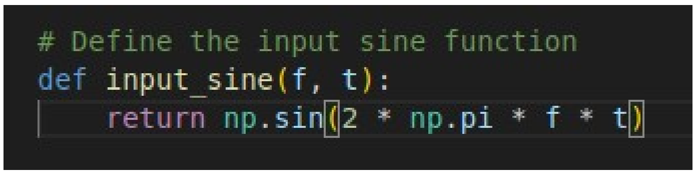
X[n] = 1/N ∑_k=0^N-1 s[k] * e^(-j.2.pi.n.k/N)

$$X(k) = \sum_{n=0}^{N-1} x(n) e^{-j2\pi kn/N}$$

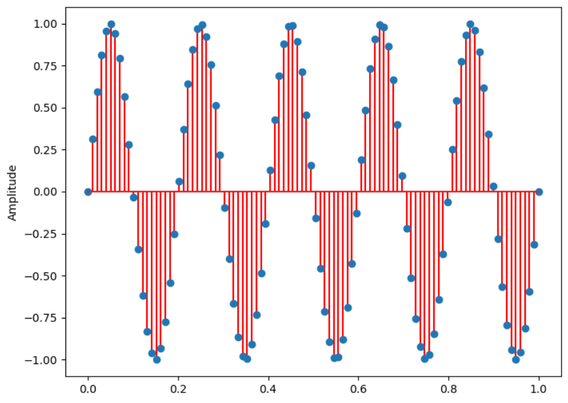
where N = number of samples,
fs = sampling frequency, and fg = signal frequency. The amplitude can be computed as the magnitude of X[n]: A[n] = |X[n]|

2. **Methods**

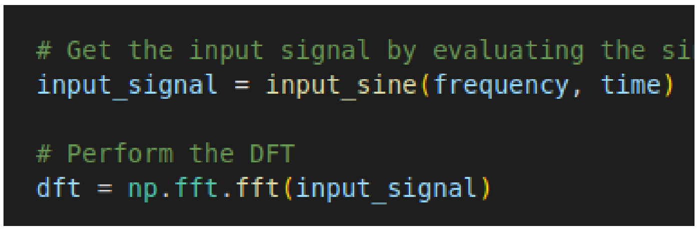
- Generate a sinusoidal signal using the equation: s[k] = sin(2.PI. fs. k/fg)



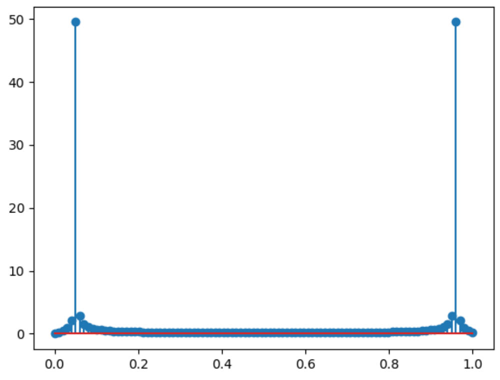
```
# Define the input sine function
def input_sine(f, t):
    return np.sin(2 * np.pi * f * t)
```



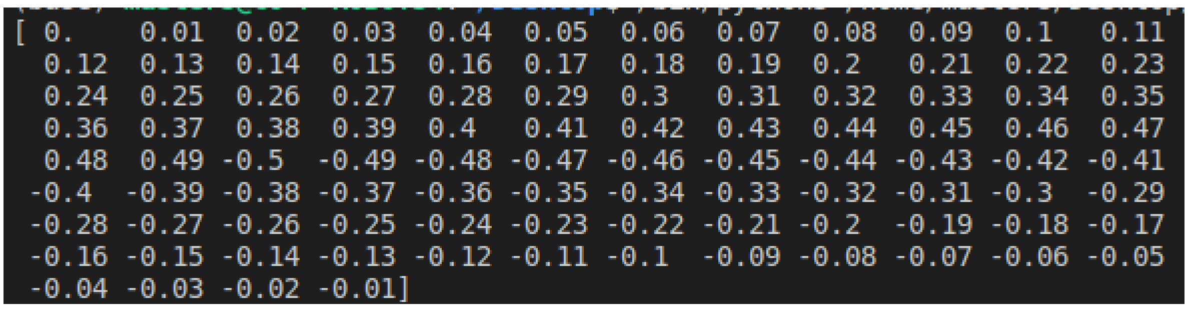
- Taking sampling rate of 100 and Frequency of 5hz, compute the DFT of the signal



```
# Get the input signal by evaluating the si
input_signal = input_sine(frequency, time)

# Perform the DFT
dft = np.fft.fft(input_signal)
```



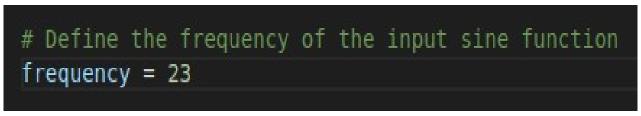
- The **frequency dictionary** can be computed by scaling the x-axis with the frequencies: f[n] = n.fs/N



```
[ 0.    0.01  0.02  0.03  0.04  0.05  0.06  0.07  0.08  0.09  0.1   0.11
  0.12  0.13  0.14  0.15  0.16  0.17  0.18  0.19  0.2   0.21  0.22  0.23
  0.24  0.25  0.26  0.27  0.28  0.29  0.3   0.31  0.32  0.33  0.34  0.35
  0.36  0.37  0.38  0.39  0.4   0.41  0.42  0.43  0.44  0.45  0.46  0.47
  0.48  0.49 -0.5  -0.49 -0.48 -0.47 -0.46 -0.45 -0.44 -0.43 -0.42 -0.41
 -0.4  -0.39 -0.38 -0.37 -0.36 -0.35 -0.34 -0.33 -0.32 -0.31 -0.3  -0.29
 -0.28 -0.27 -0.26 -0.25 -0.24 -0.23 -0.22 -0.21 -0.2  -0.19 -0.18 -0.17
 -0.16 -0.15 -0.14 -0.13 -0.12 -0.11 -0.1  -0.09 -0.08 -0.07 -0.06 -0.05
 -0.04 -0.03 -0.02 -0.01]
```

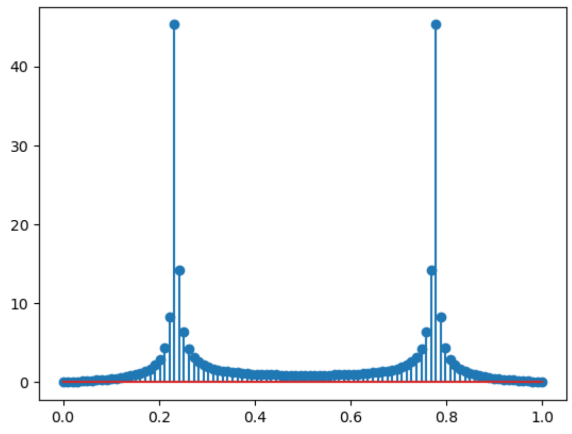
- Change the signal frequency so that it does not match the dictionary. What is the result of the DFT?

Set f = 23;

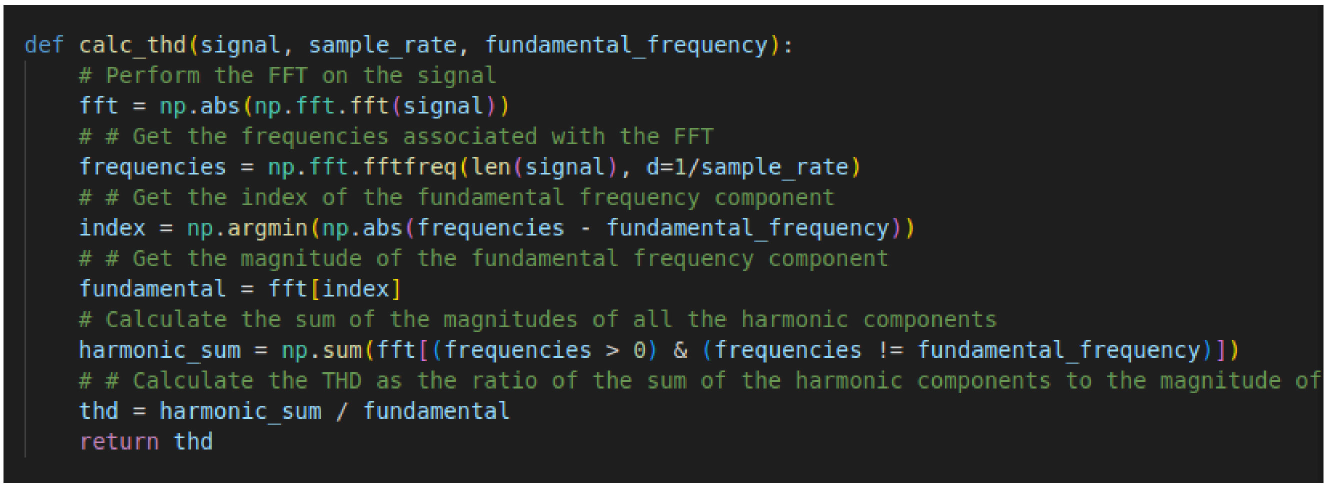


```
# Define the frequency of the input sine function
frequency = 23
```

The result of the DFT is a distorted version of the original signal with a mixture of other frequencies

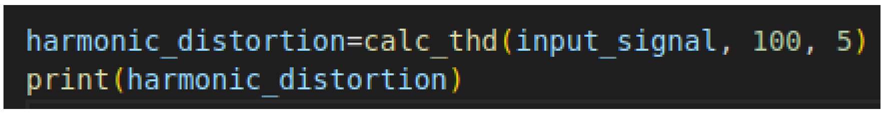


Adjust the number N to obtain the best Total harmonic distortion (THD=sqrt(A2^2+ A3^2…)/A1). Discard the mean component.



```
def calc_thd(signal, sample_rate, fundamental_frequency):
    # Perform the FFT on the signal
    fft = np.abs(np.fft.fft(signal))
    # # Get the frequencies associated with the FFT
    frequencies = np.fft.fftfreq(len(signal), d=1/sample_rate)
    # # Get the index of the fundamental frequency component
    index = np.argmin(np.abs(frequencies - fundamental_frequency))
    # # Get the magnitude of the fundamental frequency component
    fundamental = fft[index]
    # Calculate the sum of the magnitudes of all the harmonic components
    harmonic_sum = np.sum(fft[(frequencies > 0) & (frequencies != fundamental_frequency)])
    # # Calculate the THD as the ratio of the sum of the harmonic components to the magnitude of
    thd = harmonic_sum / fundamental
    return thd
```

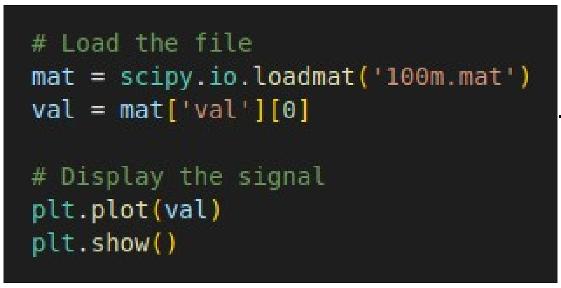
Set N to 100,



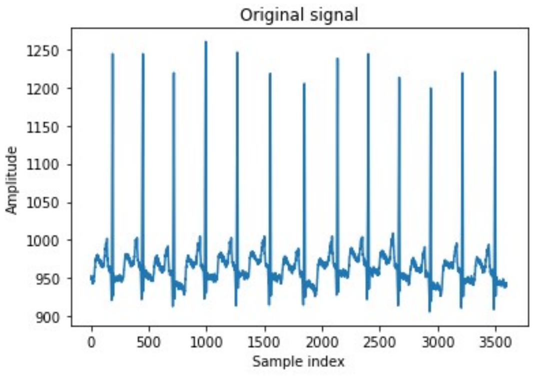
```
harmonic_distortion=calc_thd(input_signal, 100, 5)
print(harmonic_distortion)
```

THD: 6.195424857396873

- Load the file 100m.mat and display the signal (variable val)

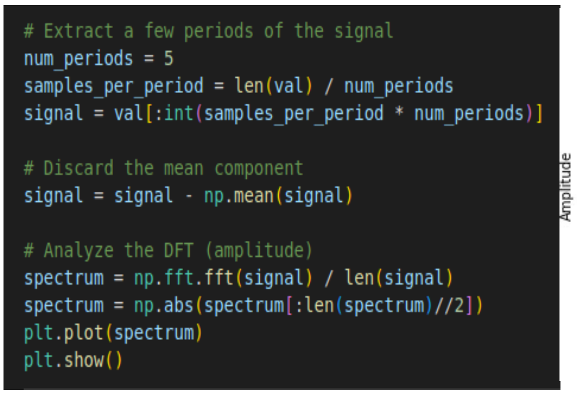


```
# Load the file
mat = scipy.io.loadmat('100m.mat')
val = mat['val'][0]

# Display the signal
plt.plot(val)
plt.show()
```

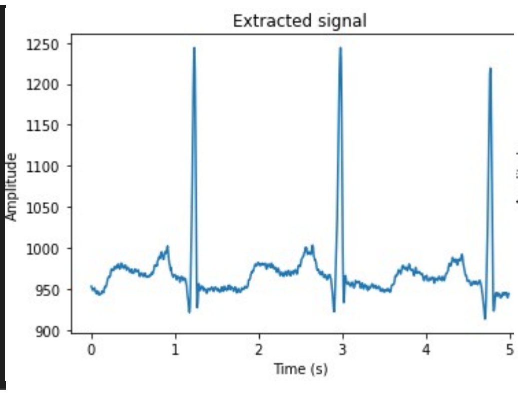


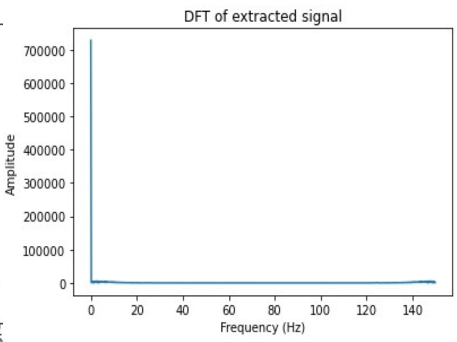
-Extract few periods and analyse the DFT (amplitude). Rk : in python use val[Ø] instead of val. Discard the mean component.

Taking fs = 150, fg = 50, start = 0 * fs and end = 5 * fs



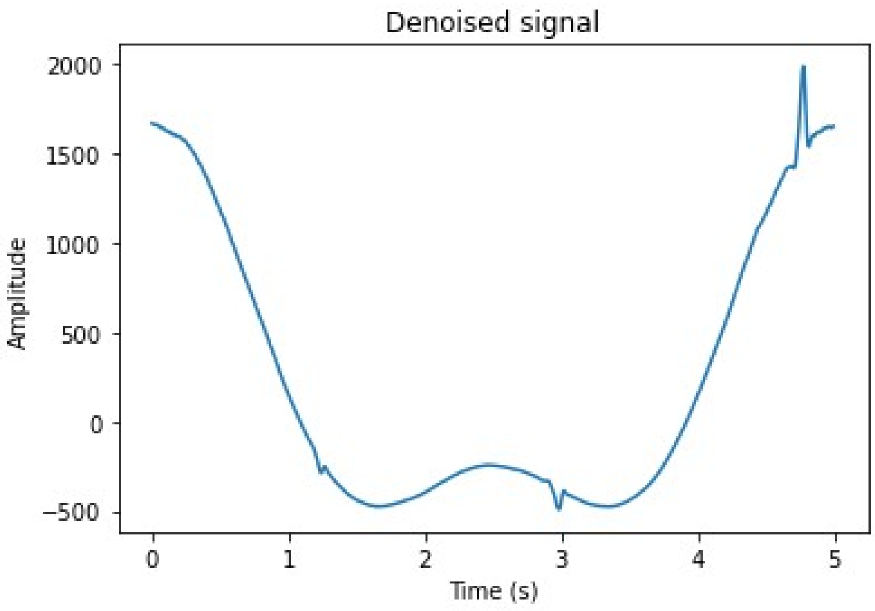
```
# Extract a few periods of the signal
num_periods = 5
samples_per_period = len(val) / num_periods
signal = val[:int(samples_per_period * num_periods)]

# Discard the mean component
signal = signal - np.mean(signal)

# Analyze the DFT (amplitude)
spectrum = np.fft.fft(signal) / len(signal)
spectrum = np.abs(spectrum[:len(spectrum)//2])
plt.plot(spectrum)
plt.show()
```





Removing the mean component shows no difference and Spike are still present after canceling the 60HZ Spurious frequencies.

Lastly, with a 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.



The Butterworth low-pass filter is designed to filter out high-frequency components in the input signal. The cutoff frequency, sample rate, and filter order are specified as inputs to the filter design function, 'butter_lowpass'.

Total Harmonic Distortion Calculation: DFT is first applied to the filtered signal to determine the magnitude of the frequency components. The index of the fundamental frequency component is then determined and the sum of the magnitudes of all the harmonic components is calculated. The THD is then determined as the ratio of the sum of the harmonic components to the magnitude of the fundamental component.

3. **Conclusion**

Future perspectives may include the study of other signal processing techniques, such as filtering, compression, and quantization, and their applications in various domains.

II.

```
import numpy as np
import matplotlib.pyplot as plt
import scipy.io as sio
from scipy.signal import butter, filtfilt

def DFT(signal):
    N = len(signal)
    X = []
    for n in range(N):
        x = 0
        for k in range(N):
            x += signal[k] * np.exp(-2j * np.pi * n * k / N)
        X.append(x)
    return np.array(X)

# Load the signal from the file
data = sio.loadmat('100m.mat')
signal = data['val'].flatten()

# Plot the signal
plt.figure()
plt.plot(signal)
plt.xlabel('Sample index')
plt.ylabel('Amplitude')
plt.title('Original signal')

# Extract few periods from the signal
fs = 150
fg = 50
start = 0 * fs
end = 5 * fs
signal = signal[start:end]
t = np.arange(len(signal)) / fs
```

```
# Plot the extracted signal
plt.figure()
plt.plot(t, signal)
plt.xlabel('Time (s)')
plt.ylabel('Amplitude')
plt.title('Extracted signal')

# Analyze the DFT of the extracted signal
X = DFT(signal)
A = np.abs(X)
f = np.arange(len(signal)) * fs / len(signal)

# Plot the DFT
plt.figure()
plt.plot(f, A)
plt.xlabel('Frequency (Hz)')
plt.ylabel('Amplitude')
plt.title('DFT of extracted signal')

# Discard the mean component
signal = signal - np.mean(signal)

# Plot the resulting signal
plt.figure()
plt.plot(t, signal)
plt.xlabel('Time (s)')
plt.ylabel('Amplitude')
plt.title('Signal after removing mean component')

# Cancel the 60Hz spurious frequency
f_spurious = 60
index = np.argmin(np.abs(f - f_spurious))
X[index] = 0
signal = np.real(np.fft.ifft(X))

# Plot the resulting signal
plt.figure()
plt.plot(t, signal)
plt.xlabel('Time (s)')
plt.ylabel('Amplitude')
plt.title('Signal after cancelling 60Hz frequency')

# Apply Butterworth filter in the Fourier space
order = 2
cutoff = 70
b, a = butter(order, cutoff / (fs / 2), 'low')
X = np.fft.fft(signal)
X_filtered = np.zeros_like(X)
for i in range(len(X)):
    if i < len(X) / 2:
        X_filtered[i] = X[i] * b[0]
        for j in range(1, order + 1):
            X_filtered[i] += X[i - j] * b[j] + X[i + j] * b[j]
    else:
        X_filtered[i] = X[i] * b[0]
        for j in range(1, order + 1):
            X_filtered[i] += X[i - j] * b[j] + X[i + j - len(X)] * b[j]
X_filtered /= np.sum(b)

# Reconstruct the denoised signal
signal_filtered = np.real(np.fft.ifft(X_filtered))

# Plot the denoised signal
plt.figure()
plt.plot(t, signal_filtered)
plt.xlabel('Time (s)')
plt.ylabel('Amplitude')
plt.title('Denoised signal')
plt.show()
```

I.

```
import numpy as np
import scipy.io
from scipy.signal import butter, lfilter
import matplotlib.pyplot as plt
from scipy.signal import freqs

def calc_thd(signal, sample_rate, fundamental_frequency):
    # Perform the FFT on the signal
    fft = np.abs(np.fft.fft(signal))
    # # Get the frequencies associated with the FFT
    frequencies = np.fft.fftfreq(len(signal), d=1/sample_rate)
    # # Get the index of the fundamental frequency component
    index = np.argmin(np.abs(frequencies - fundamental_frequency))
    # # Get the magnitude of the fundamental frequency component
    fundamental = fft[index]
    # Calculate the sum of the magnitudes of all the harmonic components
    harmonic_sum = np.sum(fft[(frequencies > 0) & (frequencies != fundamental_frequency)])
    # # Calculate the THD as the ratio of the sum of the harmonic components to the magnitude of
the fundamental component
    thd = harmonic_sum / fundamental
    return thd

# Define the input sine function
def input_sine(f, t):
    return np.sin(2 * np.pi * f * t)

# Define the sampling rate and time range
sampling_rate = 100
time = np.linspace(0, 1, sampling_rate)

# Define the frequency of the input sine function
frequency = 23

# Get the input signal by evaluating the sine function over the time range
input_signal = input_sine(frequency, time)
```

```
# Perform the DFT
dft = np.fft.fft(input_signal)

# Get the magnitude of the DFT
dft_magnitude = np.abs(dft)

freqs = np.fft.fftfreq(len(input_signal))
print(freqs)

for coef, freq in zip(dft, freqs):
    if coef:
        print('{c:>6} * exp(2 pi i t * {f})'.format(c=coef,
                                                    f=freq))

y = butter_lowpass_filter(input_signal, 2, 5)
plt.plot(y)

harmonic_distortion=calc_thd(input_signal, 100, 5)
print(harmonic_distortion)

# Plot the input signal and the DFT magnitude
plt.figure(figsize = (8, 6))
plt.stem(time, input_signal, 'r')
plt.ylabel('Amplitude')
plt.show()

plt.stem(time, dft_magnitude)
plt.show()

# Load the file
mat = scipy.io.loadmat('100m.mat')
val = mat['val'][0]

# Display the signal
plt.plot(val)
plt.show()

# Extract a few periods of the signal
num_periods = 5
samples_per_period = len(val) / num_periods
signal = val[:int(samples_per_period * num_periods)]

# Discard the mean component
signal = signal - np.mean(signal)
# Analyze the DFT (amplitude)
spectrum = np.fft.fft(signal) / len(signal)
spectrum = np.abs(spectrum[:len(spectrum)//2])
plt.plot(spectrum)
plt.show()
```