Digital Signal Processing Lab (ECL333)

Experiment 2: Spectral Analysis of ECG Signal using DFT and FFT

Department of Electronics and Communication Engineering

Faculty: Jinu Jayachandran, Viji R

Student Name: ________________________ **Roll Number:** ____________________

Instructions

1. Complete the Pre-lab work before the lab session.
2. Write all answers neatly within the space provided. Do not attach additional sheets.
3. Submit your lab sheet before leaving the lab. Late submissions receive zero marks.
4. Download an ECG signal sample from 'ecg_signal.csv' file from ETLAB and save it in the same folder as your python code.
5. Ensure to import Python libraries 'numpy', 'matplotlib', 'pandas' and 'time'.
6. The post-lab questions are indicated by **PT_Q**.

Objective

- To perform frequency-domain analysis of a real-world ECG signal using DFT.
- To compare computation time and output of DFT and FFT.
- To verify Parseval's theorem for energy conservation between time and frequency domains.
- To implement frequency-domain filtering using FFT and IFFT to remove high-frequency noise components from the ECG signal.

1 Theoretical Background

The Discrete Fourier Transform (DFT) converts a finite-length discrete-time signal from the time domain to the frequency domain. For a sequence $x[n]$, the DFT is given by:

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

Computing DFT directly requires $O(N^2)$ operations, whereas the Fast Fourier Transform (FFT) reduces it to $O(N \log N)$.

Parseval's theorem relates time-domain and frequency-domain energies:

$$\sum_{n=0}^{N-1} |x[n]|^2 = \frac{1}{N} \sum_{k=0}^{N-1} |X[k]|^2$$

Twiddle Factor Matrix and the Role of N

The DFT can be expressed as a matrix-vector multiplication:

$$\mathbf{X}_N = \mathbf{W}_N \cdot \mathbf{x}_N$$

where $\mathbf{W}_N$ is the Twiddle Factor Matrix with elements with (k, n)-th element as:

$$[\mathbf{W}_N]_{k,n} = e^{-j\frac{2\pi}{N}kn}$$

This matrix captures the frequency-domain transformation and defines how each time-domain sample contributes to each frequency bin.

Key Insight: Higher N provides finer frequency analysis but increases processing cost and memory use.

Pre-lab Work

Q1. Find the DFT of $x[n] = \{1, j, 1, j\}$ using matrix method.[2 Marks]

Let $x = [1, j, 1, j]^T$, and $N = 4$. The DFT matrix $\mathbf{W}_4$ is:

$$\mathbf{W}_4 = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & -j & -1 & j \\ 1 & -1 & 1 & -1 \\ 1 & j & -1 & -j \end{bmatrix}$$

$$X = \mathbf{W}_4 \cdot x = \begin{bmatrix} 1 + j + 1 + j \\ 1(-j) + j(-1) + 1(j) + j(1) \\ 1 + 1 + 1 + 1 \\ 1(j) + j(-1) + 1(-j) + j(-j) \end{bmatrix} = \begin{bmatrix} 2 + 2j \\ 0 \\ 2 + 2j \\ 0 \end{bmatrix}$$

Q2. If a 16-point DFT is taken what are the frequency values at which sampling is done ? [1 Mark]

$$\omega_k = \frac{2\pi k}{16} \quad \text{for } k = 0, 1, ..., 15 \Rightarrow \left\{0, \frac{2\pi}{16}, \frac{4\pi}{16}, \dots, \frac{30\pi}{16}\right\} \text{ radians/sample}$$

Q2. Frequency resolution for N=1024:

$$\Delta f = \frac{F_s}{N} = \frac{250}{1024} \approx 0.244 \text{ Hz}$$

Q3. Consider a signal that contains frequency components at 0.010π, 0.1π, and 0.25π radians/sample. If we compute an 16-point DFT of this signal, which of these frequency components will be resolvable (identifiable)? Justify your answer based on the frequency bins represented by the 16-point DFT.(*Hint: Understand the concept of frequency resolution.*) [2 Marks]

Resolving 0.010π, 0.1π, 0.25π radians/sample in a 16-point DFT:

Resolution $\Delta\omega = \frac{2\pi}{16} = \frac{\pi}{8} \approx 0.393$ rad/sample

Only 0.25π is close to a DFT bin (3rd bin). Others are below resolution. **Q3. Resolvable frequencies:** 8-point DFT gives resolution of $\Delta\omega = \frac{2\pi}{8} = \frac{\pi}{4}$ rad/sample. Only 0.25π falls on a DFT bin and is resolvable.

Q4. Write a python function `manual_dft(x, N)` to find the `N`-point DFT of signal `x` using DFT equation. (*Note:* If `N` is not passed as argument during the function call then the length of `x` shall be taken as `N`). [5 Marks]

```
def manual_dft(x, N=None):
import numpy as np
x = np.asarray(x)
N = len(x) if N is None else N
X = np.zeros(N, dtype=complex)
for k in range(N):
       for n in range(len(x)):
              X[k] += x[n] * np.exp(-2j * np.pi * k * n / N)
return X
```

Q5. Python Function to generate DFT Matrix:

```
def dft_matrix(N):
import numpy as np
W = np.zeros((N, N), dtype=complex)
for k in range(N):
       for n in range(N):
              W[k, n] = np.exp(-2j * np.pi * k * n / N)
return W
```

2 Procedure

In many signal processing applications, especially biomedical signals like the electrocardiogram (ECG), signals are often corrupted by unwanted noise components. These noises can arise from various sources such as muscle movement (EMG), power line interference, or baseline wandering. We are going to use one such noisy ECG signal for our experiment.

2.1 Plotting the ECG signal

1. Load an ECG signal from CSV file.

   (a) The sampled ECG signals are given in '.csv' file format. The first column represents the time at which sample is taken and the second column represents the sample value in mV.

   The following code using `pandas` module will help you load the csv file and get the data.

```
import pandas as pd
# ------------------------------
# 1. Load ECG Signal from CSV
# ------------------------------
df = pd.read_csv('ecg_signal.csv')
t = df['Time (s)'].values
ecg = df['ECG (mV)'].values
Fs = int(1 / (t[1] - t[0]))
```

PT_Q. From the ‘csv‘ file, what is the sampling frequency of the ECG signal. Explain From the CSV, using $Fs = \frac{1}{t[1]-t[0]}$, assuming $t[1] - t[0] = 0.004$ s, we get $Fs = 250$ Hz.

PT_Q. From the CSV file, the total duration of the ECG signal is 10s.

If $F_s = 250$ Hz, total samples $= 250 \times 4 = 1000$.

2. Plot the first 4 seconds of the ECG signal (*The x and y axis should be properly labelled and a title should be given*). Draw the plot below.

2.2 Comparison of computational complexity of DFT and FFT algorithm

In this section we are going to validate the efficiency of FFT algorithms. Will go step by step.

1. Consider

   ```
   N_values = [2, 4, 8, 16, 32, 64, 128, 256, 512, 1024]
   ```

2. For each value in `N_values`, extract the first `N` samples from the `ecg` and measure the time taken to compute its DFT using the following two methods:

   (a) `manual_dft(x)` – direct implementation of the DFT equation,

   (b) `np.fft.fft(x)` – the built-in Fast Fourier Transform (FFT) function from the `numpy` library.

   Use the `time()` function from the `time` module to measure execution time. Refer to the code snippet below for guidance.

   ```
   import numpy as np
   for N in N_values:
   x = #Get the first N samples of ecg


   start = time.time()

   #Find the DFT using manual_dft(x)

   # Append the difference of time.time() and start to a list (let us name
       it manual_times). Make sure to initialize the list before appending.

   start = time.time()

   #Find the DFT using fft() function in numpy module.

   #Append the difference of time.time() and start to another list (let us
       name it fft_times). Make sure to initialize the list before appending.
   ```

3. Print the execution time for different values of `N_values` for each of the three cases. Basically, print the list `manual_times` and `fft_times`.

4. Plot the execution time vs `N_values` for each of two cases. The x-axis should be the values of N and y-axis should be the execution time. Give proper labels, titles and markers.

5. Draw the graph here.

PT_Q. The time difference using `manual_dft()` and `np.fft.fft()` for $N = 512$ (or $N = 1024$) is between 0.5s and 10s depending on the system used.

PT_Q. Why is FFT executing faster than that of by using equation?. What are your thoughts?

FFT uses divide-and-conquer strategy to reduce computational complexity from $O(N^2)$ to $O(N \log N)$.

2.3 Verification of Parseval's theorem

1. Using Parseval's theorem we are going to verify if the energy of the signal in time domain and frequency domain are same.

   Definition: For a finite-length discrete-time signal $x[n]$ of length N, and its Discrete Fourier Transform (DFT) $X[k]$, Parseval's theorem states that the total energy of the signal in the time domain is equal to the total energy in the frequency domain, scaled by a factor of $\frac{1}{N}$:

$$\sum_{n=0}^{N-1} |x[n]|^2 = \frac{1}{N} \sum_{k=0}^{N-1} |X[k]|^2$$

   Interpretation:

   - The left-hand side represents the energy of the signal in the time domain.
   - The right-hand side represents the energy in the frequency domain.
   - The theorem confirms that the DFT preserves signal energy, except for the $\frac{1}{N}$ scaling factor.

   The energy of a discrete signal is the sum of the squares of the absolute values of each value of the discrete signal.

```
# ------------------------------
# 5. Parseval's Theorem Verification
# ------------------------------
E_time = #Find the sum of squares of absolute values of ecg
E_freq = #Find (1/N) multiplied by the sum of squares of absolute values
    of ecg
print(f"Time Domain Energy: {E_time:.4f}")
print(f"Freq Domain Energy: {E_freq:.4f}")
```

PT_Q. Why is there a $(1/N)$ scaling factor for energy calculation in frequency domain equation but is absent in time domain equation?

Because DFT spreads signal energy over N frequency bins, a $\frac{1}{N}$ is required to ensure energy equivalence.

2.4 Frequency Components and Heart Rate

We are going to find the frequencies present in the noisy ECG signal and then calculate the heart rate.

1. Use the `np.fft.fft()` function to find DFT of the signal. This function gives an N-point FFT where `N=length of input argument` by default.
2. Plot the FFT output. **You will observe peak values corresponding to the ECG signal frequency in the spectrum.**

```
X = #Find the FFT of ecg using np.fft.fft()
#Stem plot of abs(X). The x-axis is frequency in Hz and y-axis is the
   magnitude.
```

3. Draw the plot here.

PT_Q. The number of frequency domain samples in the plot is 2500 and the `N` value in the N-point FFT is 2500 (if you haven't explicitly specified the value of `N` in the code)

4. **Why are there peaks on both sides of the FFT Spectrum?**

   When computing the FFT of a real-valued signal like an ECG trace, the output exhibits symmetry due to the properties of the Discrete Fourier Transform (DFT).

   (a) **Hermitian Symmetry:** For a real-valued time-domain signal $x[n]$, its DFT $X[k]$ satisfies:
   $$X[N-k] = X^*[k]$$
   This is known as **Hermitian symmetry**, which means the second half of the FFT spectrum is the complex conjugate mirror of the first half.

   (b) **Interpretation:** The first half (from 0 to $\frac{N}{2}$) represents the positive frequencies. The second half (from $\frac{N}{2}+1$ to $N-1$) represents negative frequencies. The magnitudes at corresponding positive and negative frequencies are identical:
   $$|X[k]| = |X[N-k]|$$

   (c) **Practical Note: Correct Way to Plot Spectrum** To visualize the frequency content meaningfully for real-valued signals:

   - Use function `np.fft.fftfreq(N, d=1/Fs)` for frequency axis. The function returns the DFT sample frequencies.
   - Plot only the first half: `freqs[:N//2]` vs `np.abs(X[:N//2])`.

   This avoids redundant symmetric information and focuses on the meaningful positive frequencies.

```
freqs = np.fft.fftfreq(N, d=1/Fs)
#Stem Plot abs(X) with values at indices from 0 to N/2 in Y-axis and
    freqs with values at indices from 0 to N/2 in X-axis
```

PT_Q. The first three values of `freqs` are $[0, 0.1, 0.2]$ and the last three are $[-0.3, -0.2, -0.1]$ (depends on the value of `N`)

PT_Q. What does the values of `freqs` corresponds to?. Explain

The `freqs` array specifies the actual frequency.

5. To find the heart rate, find the index corresponding to the highest peak value in the samples of `X` from 0 to $N/2$. (*You can use `np.argmax()` function*). Then find the frequency (in `freqs`) corresponding to this index value.
6. Product of this frequency with 60 gives the heart rate in beats per minute (bpm).

PT_Q. Apart from this frequency you might have observed another peak but lower magnitude than the highest one. What do you think this frequency represent?

This could be due harmonics of heart rate or because of the frequency leakage due to sinc function.

2.5 Frequency Domain Filtering of ECG signal

Frequency-domain filtering involves transforming a time-domain signal into the frequency domain using the **Fast Fourier Transform (FFT)**. In this representation, the signal is decomposed into its frequency components. Noise typically resides in specific frequency bands that differ from those of the desired signal. For example, high-frequency noise can be attenuated using a **low-pass filter**.

2.5.1 Theory

- **Transform to Frequency Domain:** Apply FFT to convert the time-domain signal $x(t)$ to its frequency representation:

$$X(f) = \text{FFT}\{x(t)\}$$

- **Apply Frequency Mask:** Set all frequency components beyond a chosen cutoff frequency f_c to zero. This can be expressed as:

$$X_{\text{filtered}}(f) = \begin{cases} X(f), & |f| \leq f_c \\ 0, & |f| > f_c \end{cases}$$

- **Inverse Transform:** Convert the filtered spectrum back to the time domain using the inverse FFT:

$$x_{\text{filtered}}(t) = \Re\left\{\text{IFFT}(X_{\text{filtered}}(f))\right\}$$

2.5.2 Experiment

1. A cutoff frequency of 40 Hz is used to eliminate high-frequency noise from an ECG signal (our ECG signal without noise has frequency in and around $1-2$Hz). The spectrum beyond ± 40 Hz is zeroed out in the frequency domain. The filtered signal, obtained via inverse FFT, shows reduced noise while preserving key ECG features.

   Note: The inverse IFFT can be found using the function `np.fft.ifft()`.

```
cutoff = 40 # Hz
#Make a copy of X (let us call as X_filtered). See code snipet in section
     2.4 for the definition of X.
#Find all values of X_filtered whose corresponding frequency is greater
    than cutoff and make those values zero. To find the 'corresonding
    frequency' use freqs variable.
#Find the IFFT of the updated X_filtered to get ecg_filtered.
#Plot the real(ecg_filtered) and ecg in the same figure.
```

2. Draw the plot you obtained here.

PT_Q. In this experiment, frequencies above 40 Hz were eliminated. What kind of filter does this correspond to?

This is a **Low-pass filter**, preserving components below 40 Hz and removing high-frequency noise.

Work Done During Lab

(Record your major observations, frequency peaks, and comparisons here.)

- ECG signal was loaded using pandas from a CSV file.
- Initial 4 seconds of ECG plotted to identify key waveforms.
- Computed DFT manually and via FFT for different N values.
- Measured execution time and verified FFT is significantly faster.
- Applied Parseval's theorem – observed near equality in time and frequency energy.
- Identified main frequency peaks using FFT magnitude spectrum.
- Calculated heart rate from dominant peak in FFT.
- Applied a frequency-domain low-pass filter with 40 Hz cutoff to denoise ECG.
- Compared filtered vs. original signal in time domain.

Analysis and Inference

(Discuss FFT vs. DFT performance, dominant ECG frequencies, and the significance of Parseval's theorem in this context.)

- FFT dramatically reduces computation time vs. direct DFT. For large N, FFT is indispensable in real-time applications.
- ECG spectrum showed low-frequency dominance, consistent with typical heart rates (around 1–2 Hz).
- Parseval's theorem verified energy preservation; minor numerical discrepancies observed.
- Frequency-domain filtering efficiently removed noise while preserving vital ECG features.
- Overall, the lab demonstrates key advantages of FFT-based analysis in biomedical signal processing.

Code Attachment

(Attach your final Python code here.)

Student Signature: ______________________ **Date:** ______________