TECHIN 513 – Lab 3

DTFT, FFT, and Their Applications

Lab Instructions

1. Please approach the course instructor or grader for assistance if you need help.
2. We highly recommend working in groups of **2 members**. When collaborating with students in other groups, please do not share code but only discuss the relevant concepts and implementation methods.
3. Please document your code well by **using appropriate comments, variable names, spacing, indentation, docstrings**, etc. Please refer to TECHIN 509 for more information.
4. The starter code is not binding on you. Feel free to modify it as you wish. Everything is fine so long as you are getting the correct results.
5. Please upload the .ipynb file to canvas. Use Markdown cells appropriately to answer the questions. Each team only needs to submit one report.
6. Please enter the names of all the team members in your Jupyter notebook so that you do not lose credit for your work.

Lab Submission Requirements

In your notebook, briefly describe your results and discuss the problems you encountered, the solutions that you came up with. Also, answer the questions asked in each section. Make sure to include the code of all tasks in code cells.

Lab Objectives

The goal of this lab is to visualize frequency domain analysis. We will next apply Discrete Fourier Transform (DFT) and Fast Fourier Transform (FFT) in three real-world applications, including **feature extraction**, **image compression**, and **data denoising**.

Task 1: Decompose Square Wave into Harmonic Components (5 points)

In this task, we will analyze a discrete-time square wave by decomposing it into harmonic components using DFT. You need to complete the following tasks:

1. Complete the function generate_square_wave(). Refer to the docstring for additional requirements.
2. Let the sampling frequency be 1000Hz (i.e., sample 1000 times per second). Use the generate_square_wave() function to produce a periodic square wave with fundamental frequency 5Hz and duration 5 seconds.
3. Read the sample code to understand how to perform DFT on the square wave.
4. Complete the function reconstruct_wave(). Refer to the docstring for additional requirements. The signal can be reconstructed as $\sum_k C e^{2\pi f_k t + \phi_k}$, where f_k and ϕ_k are the frequency and phase of the k-the harmonic component.
5. Call your reconstruct_wave() function with the harmonic components indexed from -5 to 5. Note that the frequency components returned by np.fft.fftfreq are centered around zero.

6. Repeat Step 4 with the harmonic components indexed from -10 to 10 and indexed from -20 to 20, respectively.
7. Choose appropriate visualization to compare the original square wave and all reconstructed waveforms. Make sure your plot is clearly labeled. Explain your observations.

Task 2: Image Compression using FFT (5 points)

In this task, we will explore features extracted from image data. We will first use FFT to extract features from a given image. Then we will discard "less important" features, i.e., those FFT coefficients with small magnitudes. Note that by discarding the less important features, we set the corresponding FFT coefficients to zero. We will finally reconstruct an image based on the remaining features. The reconstruction can be achieved via inverse FFT (you may find np.fft.ifft2() useful).

1. Load the lion.jpg figure. Perform FFT on the loaded image, and display the FFT result.
2. Complete the function compress_img(). The detailed requirements of this function can be found in code template.
3. Call the compress_img() with the result in Step 1 and ratio as 0.99, 0.05, 0.01, and 0.002, respectively.
4. Visualize the output qualities.

Hint:

1. Depending on the choice of ratio, the threshold magnitude used to discard less important FFT coefficients varies. To find the threshold, you may first sort the FFT result, and search for the threshold based on the given ratio.
2. You may find masking useful when zeroing out less important FFT coefficients. Multiplying an array with a Boolean mask will zero out the entries corresponding to False entries in the mask.

Discussion (2 points)

Discuss how FFT can be leveraged to save storage and communication cost when transmitting data. Support your claim with results **AND** numbers in your implementation.

Task 3: Denoising a Signal using FFT (8 points)

In this task, we will investigate how to use FFT to denoise a signal as shown in our in-class motivating example. Use the generate_sine() function to produce two sine waves: $\sin(2\pi \times 50t)$ and $\sin(2\pi \times 150t)$ of duration 1 second. Let $g(t) = \sin(2\pi \times 50t) + \sin(2\pi \times 120t)$ and the sampling frequency be 1000Hz. Add a random noise following a standard normal distribution to the sampled waveform. We can generate the noise using np.random.randn().

We will apply a concept named **power spectrum density (PSD)** to denoise the signal. The PSD describes how the power of a signal is distributed over its frequency spectrum. It provides a

quantitative way to analyze the energy or variance of a signal across different frequency components. In practice, the PSD is particularly useful for:

- **Signal Characterization**: Identifying dominant frequencies or patterns in a signal.
- **Noise Analysis**: Distinguishing between signal and noise components, as noise typically spreads its power evenly across frequencies.
- **System Analysis**: Understanding the frequency response of systems or environments.

For discrete signals, the PSD is estimated as $PSD(f) = \frac{|X(f)|^2}{N}$, where $X(f)$ is the DFT of a signal $x[n]$, f is frequency, and N is the number of samples. The unit of PSD is power per frequency (W/Hz). Alternatively, you can call the Welch's method using welch(signal, fs) from scipy.signal to estimate PSD.

Follow the steps below to denoise the signal.

1. Compute the FFT of the noisy signal.
2. Find frequency components with high power. We can find these components by first sorting the components by PSD, and then use the threshold_ratio to identify those need to be kept.
3. Zero out frequency components with low power.
4. Use the remaining frequency components to reconstruct the signal. You can reconstruct the signal via inverse FFT. Note that the reconstructed signal is complex, and we only need the real part. To extract the real part, we can use np.real().
5. Plot $f(t)$ and the reconstructed signal for comparison. Repeat this step with threshold_ratio 0.1, 0.2, and 0.3. Explain your observations under different threshold_ratio.

Optional Task: FFT for Feature Engineering

FFT can also be used to process time series data, making features embedded in time series data easier to be learned by machine learning models. Please refer to this link for additional readings and implementation: FFT for human activity recognition.