CSE 220: Fourier Transform Assignment

November 9, 2024

Task 1:

The purpose of this assignment is to understand how Fourier Transform (FT) can approximate various functions. You will implement custom Fourier Transform and Inverse Fourier Transform (IFT) algorithms, then use these to approximate parabolic, triangular, sawtooth, and rectangular wave functions.

Function creation

- **Parabolic Function**: Create a parabolic function $y = x^2$ within the interval $[-2, 2]$ and 0 elsewhere.
- **Triangular Function**: Create a triangular wave within the interval $[-2, 2]$ with height 1 and 0 elsewhere.
- **Sawtooth Function**: Create a sawtooth wave within the interval $[-2, 2]$ with a slope of 1 and 0 elsewhere.
- **Rectangular Function**: Create a rectangular pulse within the interval $[-2, 2]$ with a height of 1 and 0 elsewhere.

Fourier Transform

Go to `FT_basic.py`. Complete the function `fourier_transform(signal, frequencies, sampled_times)` where `signal` is the y values of the function, `frequencies` are the frequency values for which you want to store the transform and `sampled_times` are the corresponding x values for y. Also complete the `inverse_fourier_transform(ft_signal, frequencies, sampled_times)` where `ft_signal` is the fourier transformed signal containing two parts - real and imaginary. Read the comments for better understanding and edit other parts according to the comments. Remember, you can only use `trapz` for integration. Any built-in libraries for FT, FFT, DFT, IFT are not allowed.

Experiment with Frequency Limits

Try different frequency ranges, starting with low frequencies and gradually increasing. Observe how the approximation improves as higher frequencies are included. But at some point higher frequencies will distort the results. Why?

Plotting

For each function:

- Plot the original function.
- Plot the frequency spectrum.
- Plot the Fourier approximated function.

Sample Output

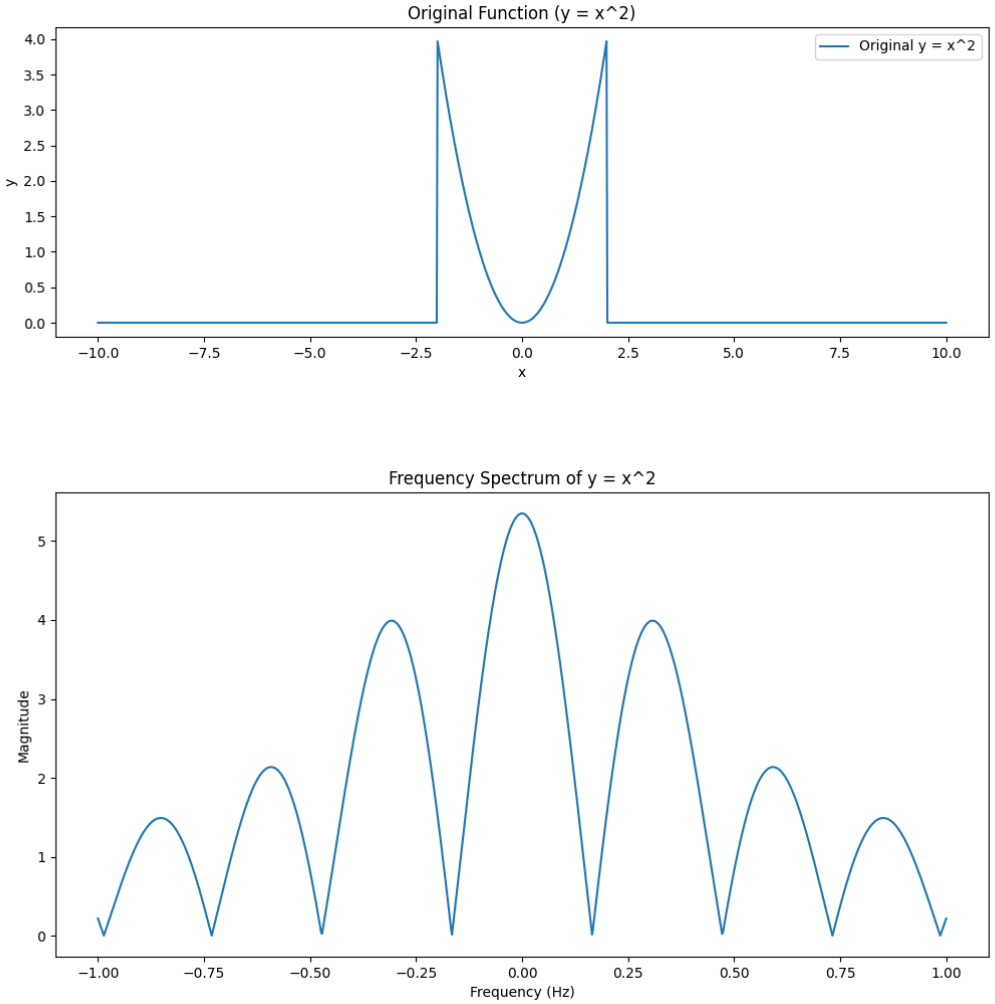

Figure 1: Frequency range from -1 to 1

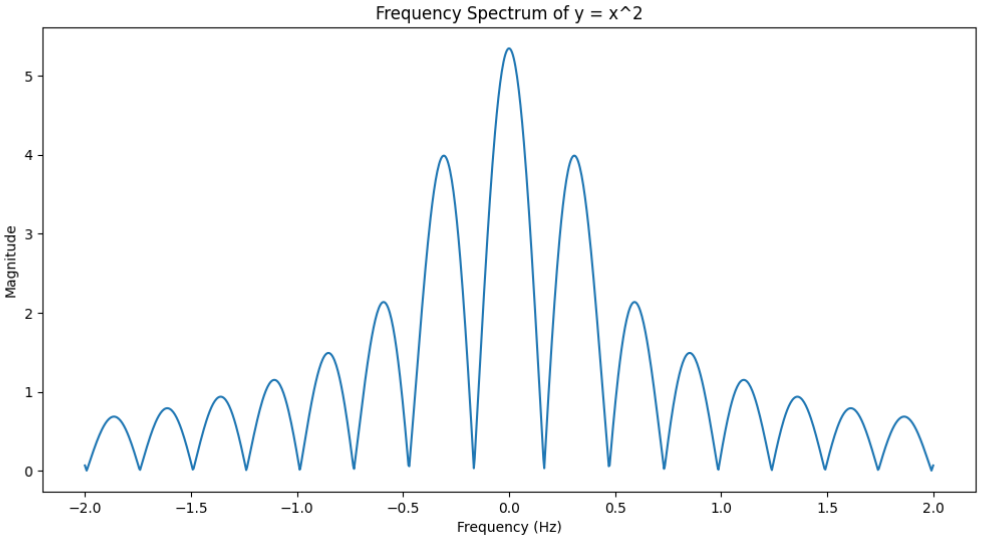

Figure 2: Frequency range from -2 to 2

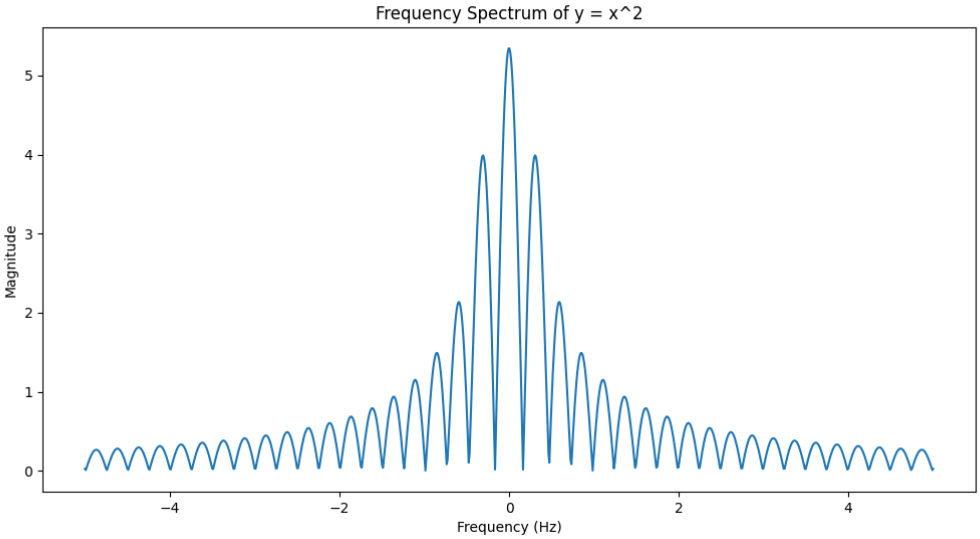

Figure 3: Frequency range from -5 to 5

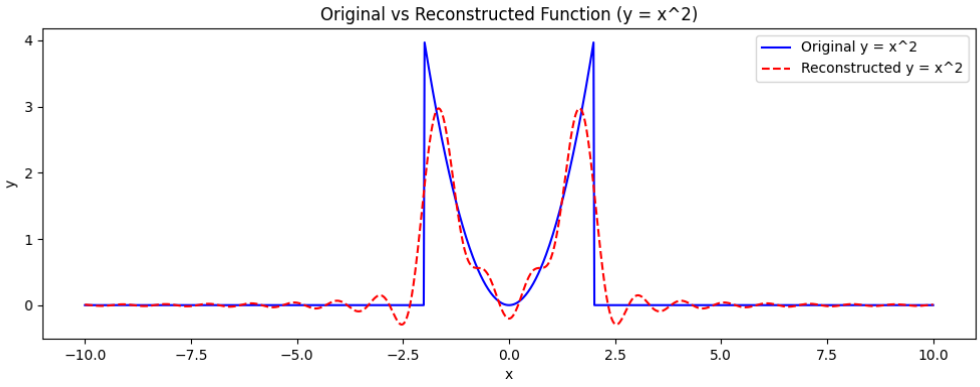

Figure 4: Reconstructed signal for frequency range -1 to 1

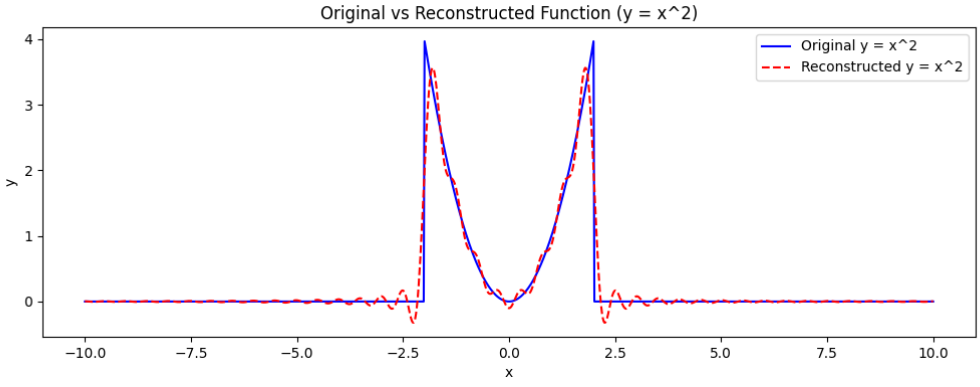

Figure 5: Reconstructed signal for frequency range -2 to 2

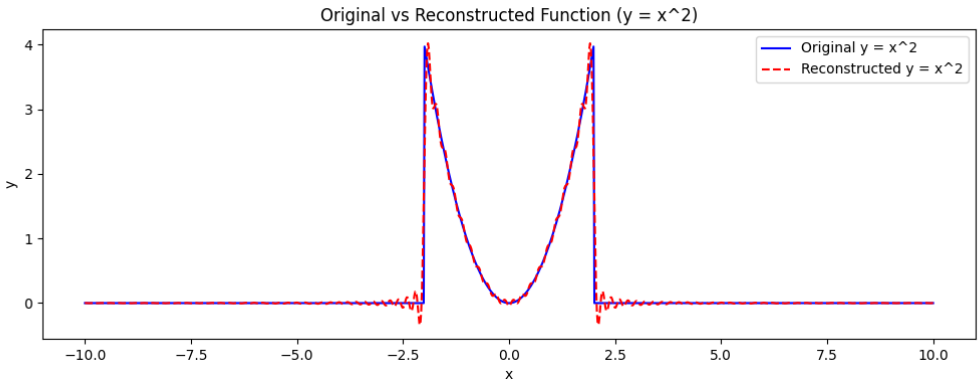

Figure 6: Reconstructed signal for frequency range -5 to 5

Task 2

This is a very easy and interesting task to understand how Fourier Transform can be used in audio processing. You have been given a `buzzjc.wav` file, with an audio clip of someone saying something, but it has been corrupted by adding a buzzer signal consisting of a few tones. Use fourier transform to figure out the tone frequencies (at least approximately). Next remove these tones (how?) and then listen to the clip (note: you can only play a real signal, so make sure you only take the real part). Expected denoised signal is given in `denoised_audio.wav`.

You do not have to think about taking music as a input or how to save a music file. Most of the codes have been given in `audio_processing.py`. You also implemented fourier transform and inverse transform from first task. Just think about how to reduce the noise.

The code may take some time (∼30-40 seconds) to run. Do not worry about it. It is highly recommended not to change the parameters given in skeleton code like `interval_step`,sample_rate,max_time,max_freq etc. Otherwise you will not get desired output or may take more time. You can play the sound in Windows Media Player or in VS code too. If you can not play the .wav files in your device in any way, you can use this online tool.

Marks Distribution

Task	**Marks**
Task 1	60
Function creation	10
Fourier Transform	20
Inverse Fourier Transform	20
Plotting	10
Task 2	40
Fourier Transform and Inverse Fourier Transform	20
Denoising	20
Total	100

Submission Guideline

- Create a folder named by your student ID 2105XXX. Put `FT_basic.py`, `audio_processing.py` inside the folder. You do not have to submit any .wav files. Zip the folder and submit the 2105XXX.zip.