ECE 351 - 52

Signals and Systems 1

Lab 9

Submitted By :
Owen Blair

Contents

1 Important Notes

It is important to note that plt.show() is needed at the end of the python code to properly show the plots of each function. It was not included in every section of code that plots a function(s) because it is assumed that its included at the end the code. It is also assumed that the following is included at the begining of the program:

```
import numpy as np
import matplotlib.pyplot as plt
import scipy.signal as sig
import scipy.fftpack as fft
import math as m
```

The web address to the GitHub where LaTeX code is stored is here:
https://github.com/Blairis123/ECE351_Reports

The web address to the GitHub where the Python code is here:
https://github.com/Blairis123/ECE351_Code

2 Part 1 Deliverables

Part 1 consists of 5 tasks. The first task is plotting the fast Fourier transform of the function $f(t) = cos(2\pi t)$ on the interval of $0 \leq t \leq 2$ seconds. This needs to have a sampling frequency of 100 and should have the plot of the phase and signal. The x-axis values should also be limited to clearly show the relevant values. Keep in mind that some of the axis used is in the time domain and some is in the frequency domain. This should also be done for the following two functions:

$$f(t) = 5sin(2\pi t)$$

$$f(t) = 2cos((2\pi * 2t) - 2) + sin^2((2\pi * 6t) + 3)$$

Next, modify the the fast Fourier transform so that the if an element $X_{mag} < 1e-10$ then the corresponding angle $X_{phi} = 0$. After this, use the fast Fourier transform on the N=15 Fourier approximation signal from lab 8. Use T=8 on the time-domain of $0 \leq t \leq 16$ seconds.

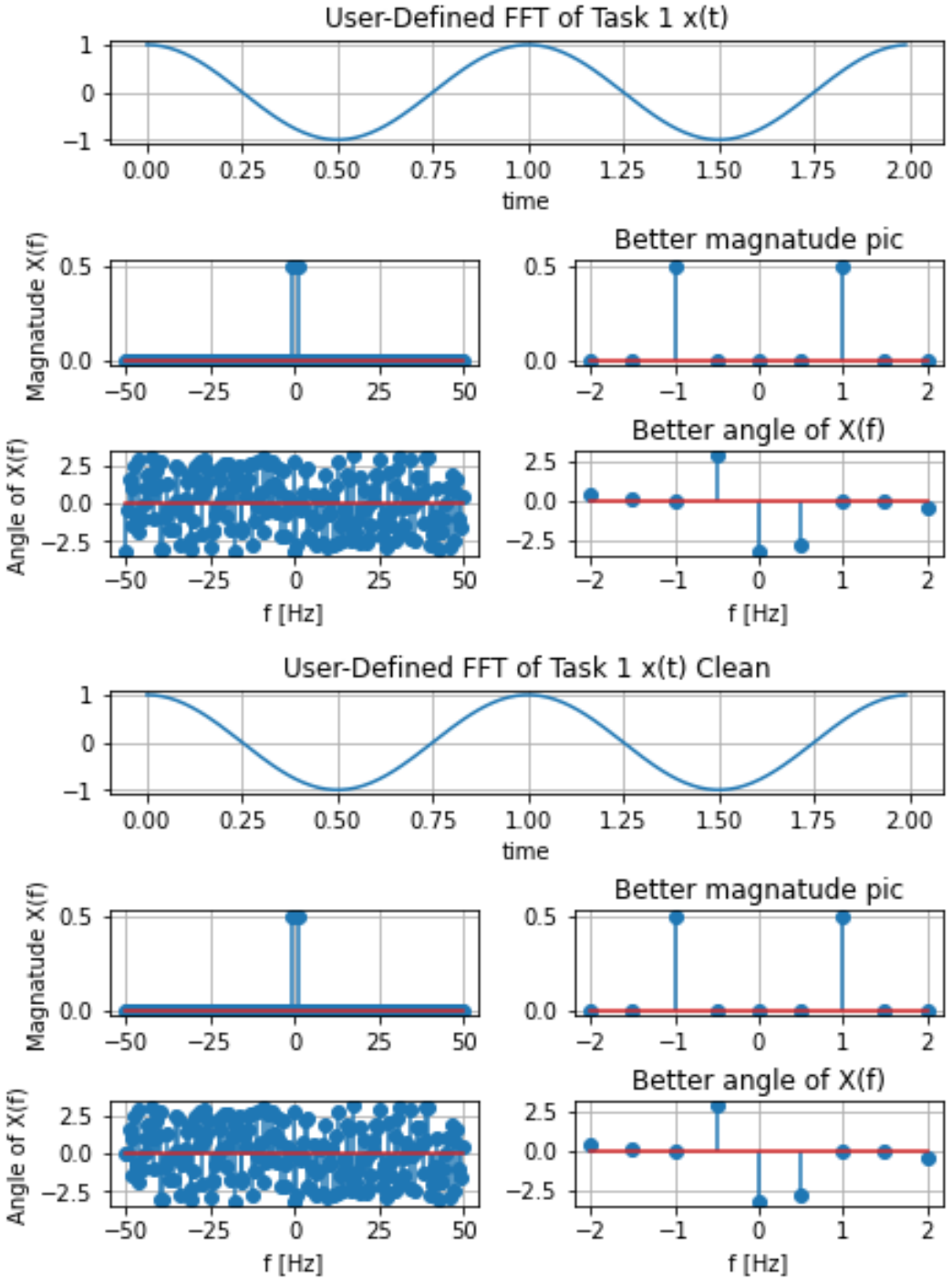
Listing 1: FFT Code is the code used to compute the fast Fourier transform. It takes two inputs, X, the function array to apply the Fourier transform and fs, the sampling frequency. The outputs array is a multi dimensional array that contains various information. Index 0 is the Fourier transform of the input, index 1 is the

same array but centered at the zero frequency, index 2 is the frequency array corresponding to the Fourier output, index 3 is an array of the Fourier magnitudes, and index 4 is an array of the Fourier angles. The following *Listing 2: FFTClean Code* is the same as *Listing 1* but it cleans the output of the angle array.

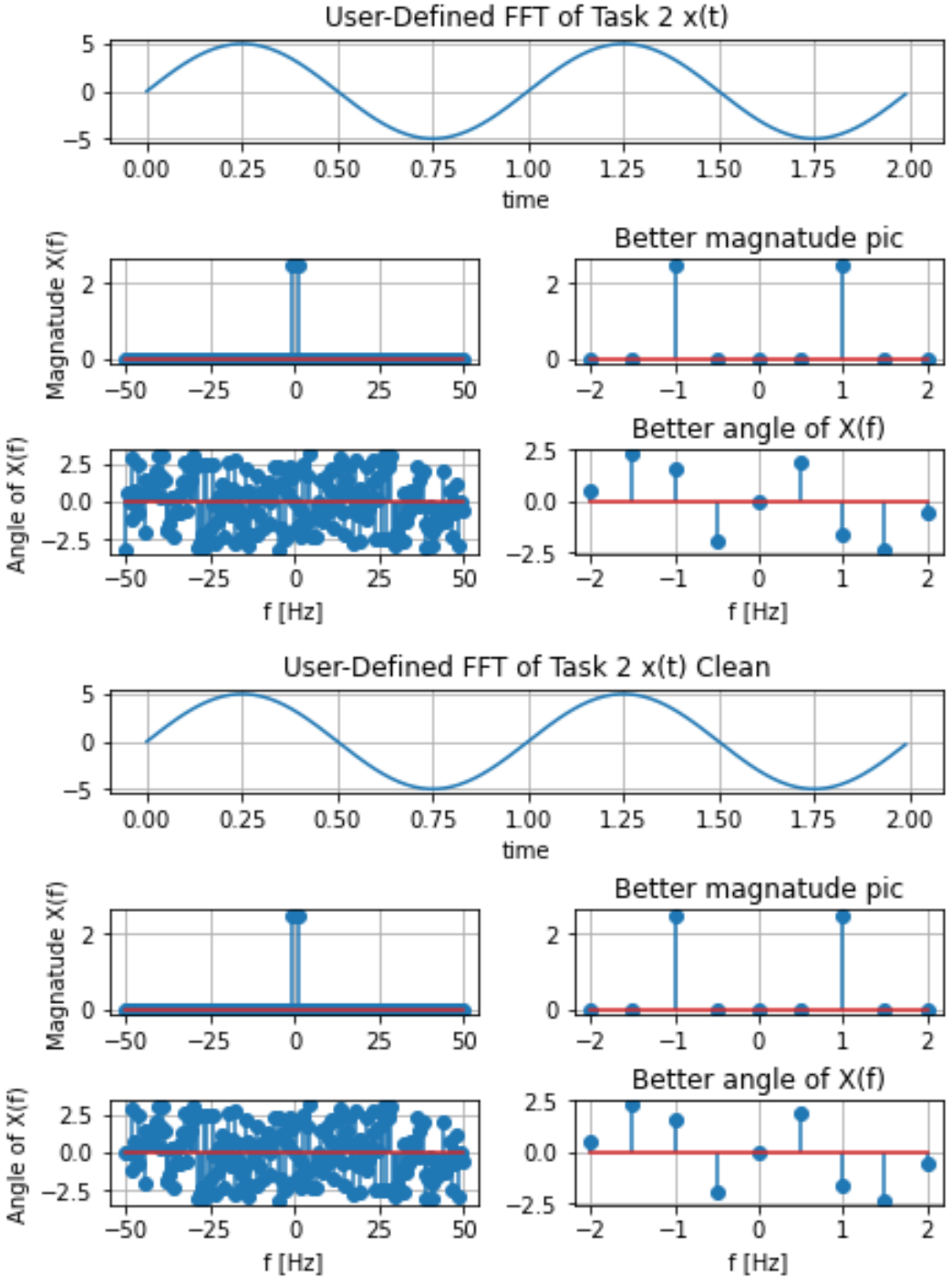
Listing 3: plot_fft Code is the code that was used to create the format of the plots in the figures section. The inputs are *title*, the title of the plot, x, the original function, X_{mag}, the magnitude array of the Fourier transform, X_{phi}, the angle array of the Fourier transform, *freq*, the frequency array of the Fourier transform, and t, the time array that corresponds to the original function. This function has no data to return the program but it will output the plots to the plots tab in Spyder5. *Listing 4: Implementation Code* is the code used to impalement the previous listed functions.

3 Figures

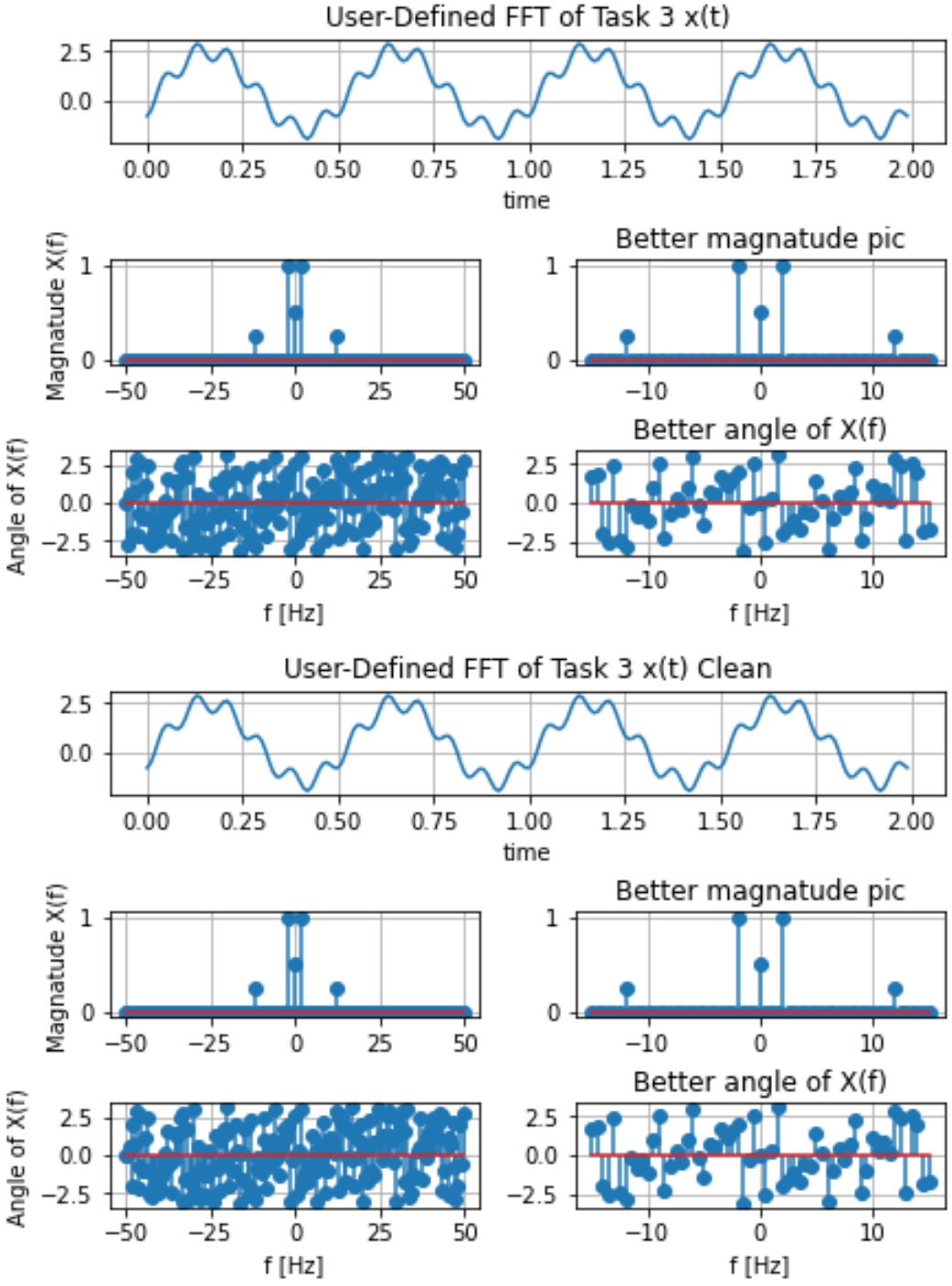
3.1 Task 1 Plot



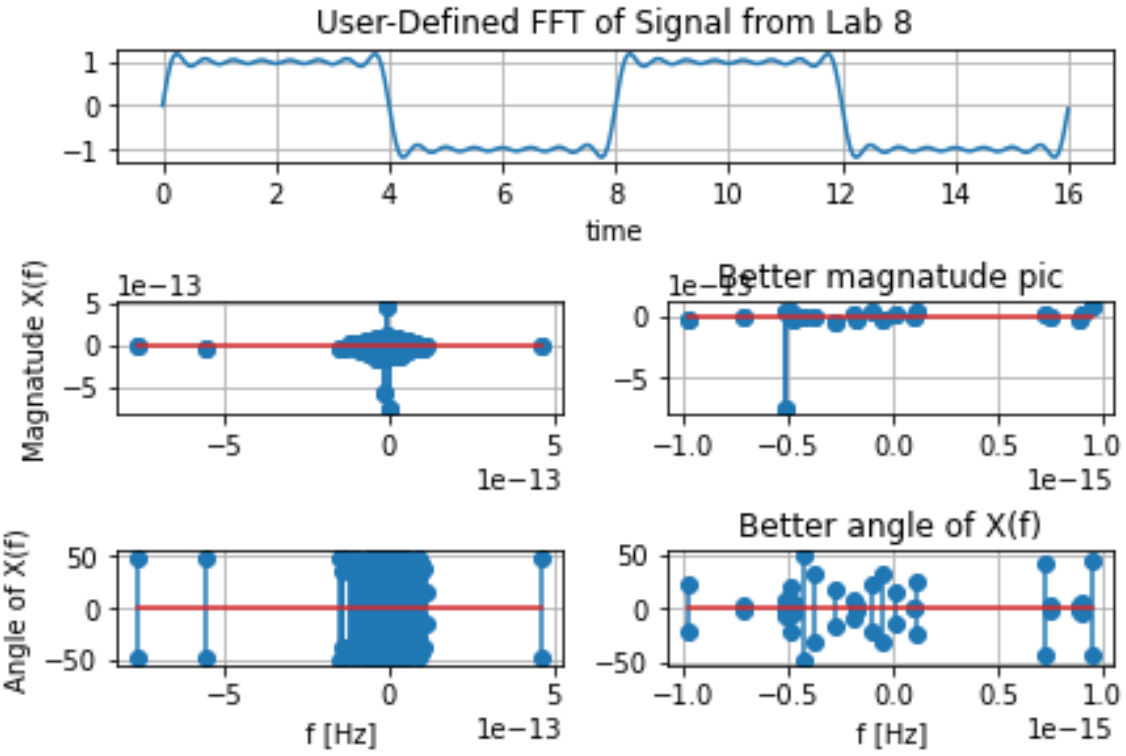
3.2 Task 2 Plot



3.3 Task 3 Plot



3.4 Lab 8 Signal Plot of N=15



4 Questions

4.1 Question 1

When the sampling rate of the fast Fourier transform is lowered the number of data points calculated decreases. This in turn decreases the resolution of the plots. If **fs** is increase the number of points calculated for the fast Fourier transform increases and the resolution of the plots also increase.

4.2 Question 2

Eliminating the small phase magnitudes should make a significant difference in the plots in that only the important phase angles are non-zero values. Unfortunately, the cleaned version of my code did not work and only eliminated the first and last points that were $\leq 1e-10$. The result should be a phase angle plot that has a lot less noise than what is present.

4.3 Question 3

The results from task 1 and task 2 can be verified by hand calculating the Fourier transforms of cosine and sign function. The Fourier transform of the $cos(2\pi t)$ is

$$\mathcal{F}\{cos(2\pi t)\} = \frac{1}{2}(\delta(\omega - \omega_0) + \delta(\omega + \omega_0))$$

The Fourier transform if the function $5sin(2\pi t)$ is:

$$\mathcal{F}\{5sin(2\pi t)\} = \frac{5j}{2}(\delta(\omega - \omega_0) + \delta(\omega + \omega_0))$$

Both of these transforms will produce a series of 'spikes' that will repeat for every $\omega_0 * n$ frequencies. This tracks with the 'spikes' generated from the fast Fourier transform plots/

5 Listings

5.1 Listing 1: FFT Code

```
"""User defined fast fourier transform function.
INPUTS:
    X, a function array
    fs, frequency of sampling rate
OUTPUTS:
    output, an array containing various information
    output[0], a fourier transform of the input function
    output[1], same as output[0] but zero frequency is the center
   of spectrum
    output[2], frequency array corresponding to fourier output
    output[3], array of fourier magnitudes
    output[4], array of fourier angles
"""
def FFT(X, fs):

    #Length of input array
    n = len(X)

    #Preform fast fourier transform
    X_fft = fft.fft(X)

    #shift zero frequency to center of the spectrum
    X_fft_shift = fft.fftshift(X_fft)

    # Calculate frequencies for output. fa is sampling frequency
    X_freq = np.arange(-n/2, n/2) * fs / n
```

```
    #zm_freq = np.arrange(-n/4, n/4) * fs/(n/2)

    #Calculate magnitude and phase
    X_mag = np.abs(X_fft_shift)/n
    X_phi = np.angle(X_fft_shift)

    output = [X_fft, X_fft_shift, X_freq, X_mag, X_phi]
    return output
```

5.2 Listing 2: FFTClean Code

```
"""CleanFFT
CleanFFT is like the user defined FFT function, but if a given magnitude is
    less than 1e-10 the corresponding phase angle will be set to zero.
DEPENDENCIES:
    FFT(X, fs), Function that returns an array containing the fast fourier
    transform of an array X with a given sample rate fs.
INPUTS:
    X, an array to apply the fourier transform on
    fs, sampling frequency for fourier transform
OUTPUTS:
    output, an array containing cleaned up arrays of X_mag, X_phi and freq
"""
def FFTClean(X,fs):
    XArry = FFT(X, fs)
    useableArry = [XArry[2], XArry[3], XArry[4]]
    for i in range(0, len(useableArry)-1):
        if (useableArry[1][i] <= 0.000000001):
            useableArry[2][i] = 0

    return useableArry
```

5.3 Listing 3: plot_fft Code

```
"""
plot_fft is for ploting stuff!
"""
def plot_fft(title, x, X_mag, X_phi, freq, t, zmInt):

    #Calculate the zoomed in data for magnatude and frequency
    zm_mag = [];
    zm_mag_freq = [];
    for i in range(0, len(freq)-1):
```

```
        if ((freq[i]>=-zmInt) and (freq[i]<=zmInt)):
            zm_mag.append(X_mag[i])
            zm_mag_freq.append(freq[i])

    zm_phi = [];
    zm_phi_freq = [];
    for i in range(0, len(freq)-1):
        if ((freq[i]>=-zmInt) and (freq[i]<=zmInt)):
            zm_phi.append(X_phi[i])
            zm_phi_freq.append(freq[i])

    fig3 = plt.figure(constrained_layout=True)
    gs = fig3.add_gridspec(3, 2)
    f3_ax1 = fig3.add_subplot(gs[0, :])
    f3_ax1.set_title('User-Defined FFT of '+ title)
    f3_ax1.set_xlabel('time')
    f3_ax1.plot(t,x)
    plt.grid()


    f3_ax2 = fig3.add_subplot(gs[1, 0])
    f3_ax2.set_ylabel('Magnatude X(f)')
    f3_ax2.stem(freq, X_mag)
    plt.grid()

    f3_ax3 = fig3.add_subplot(gs[1, 1])
    f3_ax3.set_title('Better magnatude pic')
    f3_ax3.stem(zm_mag_freq, zm_mag)
    plt.grid()

    f3_ax4 = fig3.add_subplot(gs[2, 0])
    f3_ax4.set_ylabel('Angle of X(f)')
    f3_ax4.set_xlabel('f [Hz]')
    f3_ax4.stem(freq, X_phi)
    plt.grid()

    f3_ax5 = fig3.add_subplot(gs[2, 1])
    f3_ax5.set_title('Better angle of X(f)')
    f3_ax5.set_xlabel('f [Hz]')
    f3_ax5.stem(zm_phi_freq, zm_phi)
    plt.grid()
```

5.4 Listing 4: Implementation Code

```
    #Define step size
steps = 1e-2

    #t for part 1
```

```
start = 0
stop = 2
    #Define a range of t_pt1. Start @ 0 go to 20 (+a step) w/
    #a stepsize of step
t = np.arange(start, stop, steps)

# Sampling frquency for lab
fs = 100

# Task 1 input function, FFT, FFTClean
cos_2_pi = np.cos(2* np.pi * t)
FFTcos_2_pi = FFT(cos_2_pi, fs)
FFTCleanTask1 = FFTClean(cos_2_pi, fs)

# Task 2 input function, FFT, FFTClean
sin_2_pi_5 = 5 * np.sin(2 * np.pi * t)
FFTsin_2_pi_5 = FFT(sin_2_pi_5, fs)
FFTCleanTask2 = FFTClean(sin_2_pi_5, fs)

# Task 3  input function, FFT, FFTClean
task3Func = 2* np.cos((4*np.pi*t) - 2) + (np.sin((12*np.pi*t) + 3)
    )**2
FFT_task3Func = FFT(task3Func, fs)
FFTCleanTask3 = FFTClean(task3Func, fs)

#Fourier plot of the previous signal from lab 8
t2 = np.arange(0, 16, steps)
x_15 = xFourier(t2, 8, 15)

FFT_Lab8 = FFT(x_15, fs)

#Make the plots using the function!!!
plot_fft("Task 1 x(t)", cos_2_pi, FFTcos_2_pi[3], FFTcos_2_pi[4],
    FFTcos_2_pi[2], t, 2)
plot_fft("Task 2 x(t)", sin_2_pi_5, FFTsin_2_pi_5[3], FFTsin_2_pi_5
    [4], FFTsin_2_pi_5[2], t, 2)
plot_fft("Task 3 x(t)", task3Func, FFT_task3Func[3], FFT_task3Func
    [4], FFT_task3Func[2], t, 15)

# Plot the noise reduced versions of the functions
plot_fft("Task 1 x(t) Clean", cos_2_pi, FFTCleanTask1[1],
    FFTCleanTask1[2], FFTCleanTask1[0], t, 2)
plot_fft("Task 2 x(t) Clean", sin_2_pi_5, FFTCleanTask2[1],
    FFTCleanTask2[2], FFTCleanTask2[0], t, 2)
plot_fft("Task 3 x(t) Clean", task3Func, FFTCleanTask3[1],
    FFTCleanTask3[2], FFTCleanTask3[0], t, 15)

plot_fft("Signal from Lab 8", x_15, FFT_Lab8[1], FFT_Lab8[2],
    FFT_Lab8[0], t2, 1e-15)
```