Lab Assignment

Imports, Functions and Variables

lab3main.py M

lab3 > lab3main.py > ...

You, 5 minutes ago | 1 author (You)

```python
#Lab3 ELE532
#Jahmil Ally (501045419)

#Imports
import os
import time
import numpy as np
import scipy.io as sci
import matplotlib.pyplot as plt
import matlab.engine
from scipy.integrate import quad
import cmath
from sympy import symbols, cos, pi, exp, integrate

#Variable Declaration
color="g"
eng = matlab.engine.start_matlab()

#Defining a generic function for plotting
def plot(f_t, t, newGraph=True, figsize=(12.0, 6.0), title="", functionLabel="", xLabel="t", yLabel="f(t)"):
    global color
    if newGraph:
        plt.figure(figsize=figsize)
        color="g"

    #Configure Colour
    if  color == "g" and  newGraph==False:
        color="r"
    elif color == "r" and  newGraph==False:
        color="y"
    elif color == "y" and  newGraph==False:
        color="b"
    elif color == "b" and  newGraph==False:
        color="o"
    elif color == "o" and  newGraph==False:
        color="p"

    #Configurable titles/ Labels
    plt.plot(t, f_t, color=color, label=functionLabel)
    if title !="":
        plt.title(title)
    if xLabel !="":
        plt.xlabel(xLabel)
    if yLabel !="":
        plt.ylabel(yLabel)

    #Visuals
    plt.grid(True)
    plt.legend()
    plt.tight_layout()
```

```python
def Dn(x, n, period=20):
    t = symbols("t")
    w0_1 = 2 * np.pi / period

    if x < 0 or x > 2:
        return None

    n_array = np.array(n)
    Dn_array = np.zeros_like(n_array, dtype=complex)

    if x == 0:
        Dn_array[n_array == 1] = 1/4
        Dn_array[n_array == -1] = 1/4
        Dn_array[n_array == 3] = 1/2
        Dn_array[n_array == -3] = 1/2

        return Dn_array

    elif x == 1 or x == 2:
        # Handle dividing by zero
        n_non_zero = np.where(n_array != 0, n_array, np.nan)  # Replace 0 with nan
        Dn_array = 1 / (n_non_zero * np.pi)

        if x == 1:
            Dn_array *= np.sin((n_non_zero * np.pi) / 2)
        elif x == 2:
            Dn_array *= np.sin((n_non_zero * np.pi) / 4)

        # Handle the case when n = 0 (replace with the correct value if needed)
        Dn_array[np.isnan(Dn_array)] = 0  # Replace nan with 0
        return Dn_array

def plot_spectra(D_n, n_range, title):
    plt.figure(figsize=(12, 7))

    #Magnitude spectrum
    plt.subplot(1, 2, 1)
    plt.stem(n_range, np.abs(D_n), "k", markerfmt="ok")
    plt.xlabel("n")
    plt.ylabel("|D_n|")
    plt.title(f"Magnitude Spectrum of {title}")

    #Phase spectrum
    plt.subplot(1, 2, 2)
    plt.stem(n_range, np.angle(D_n), "k", markerfmt="ok")
    plt.xlabel("n")
    plt.ylabel("∠ D_n [rad]")
    plt.title(f"Phase Spectrum of {title}")

    plt.tight_layout()
```

Part A

- **Problem A.1**
  Code:

Results:

- **Problem A.2**
  Code:

  Results:

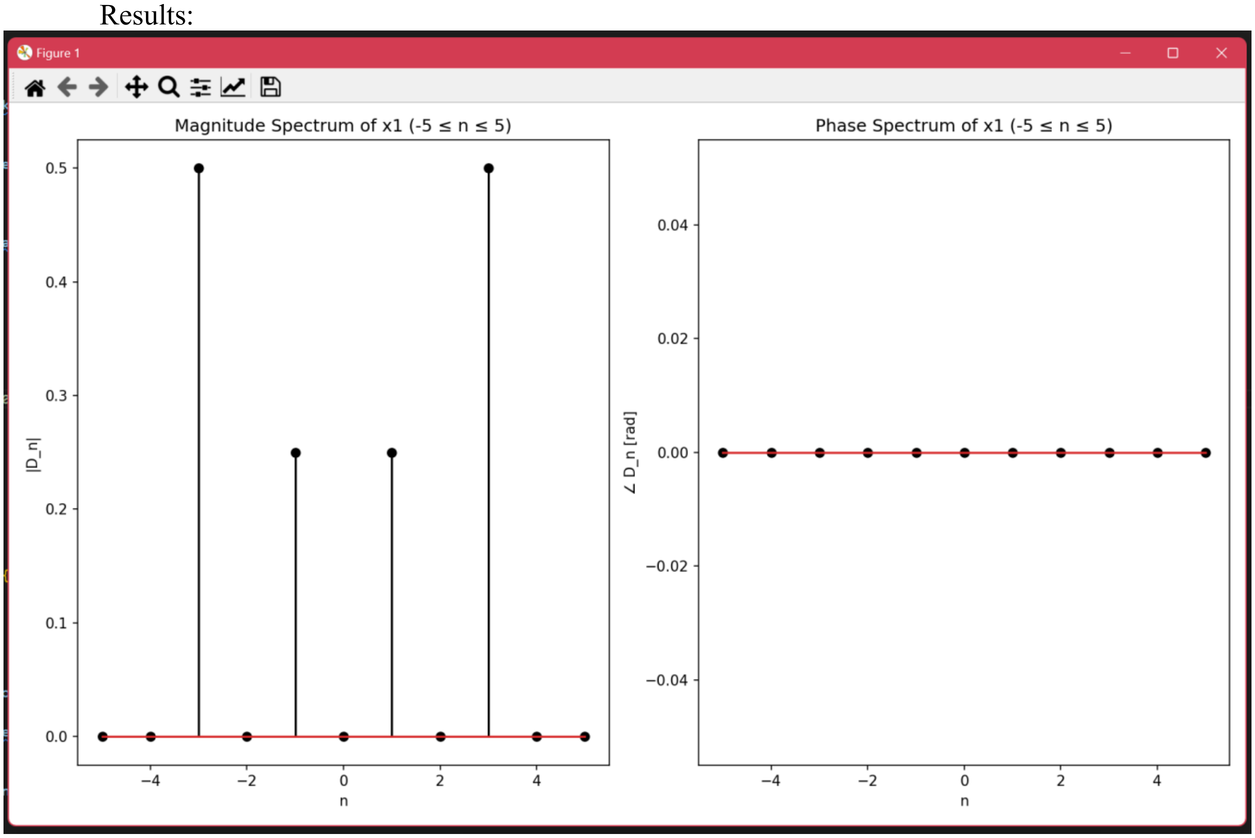
- **Problem A.4**
  Code:

```python
# Part A.4
ranges = [list(range(-5, 6)), list(range(-20, 21)), list(range(-50, 51)), list(range(-500, 501))]

for i in range(0, 4):
    currentRange = ranges[i]
    for j in range(0, 3):
        Dn_x = Dn(j, currentRange)

        if Dn_x is not None:
            n = np.array(currentRange)
            title = f"x{j+1} ({n[0]} ≤ n ≤ {n[-1]})"
            plot_spectra(Dn_x, n, title)
            plt.show()
```

  Results:



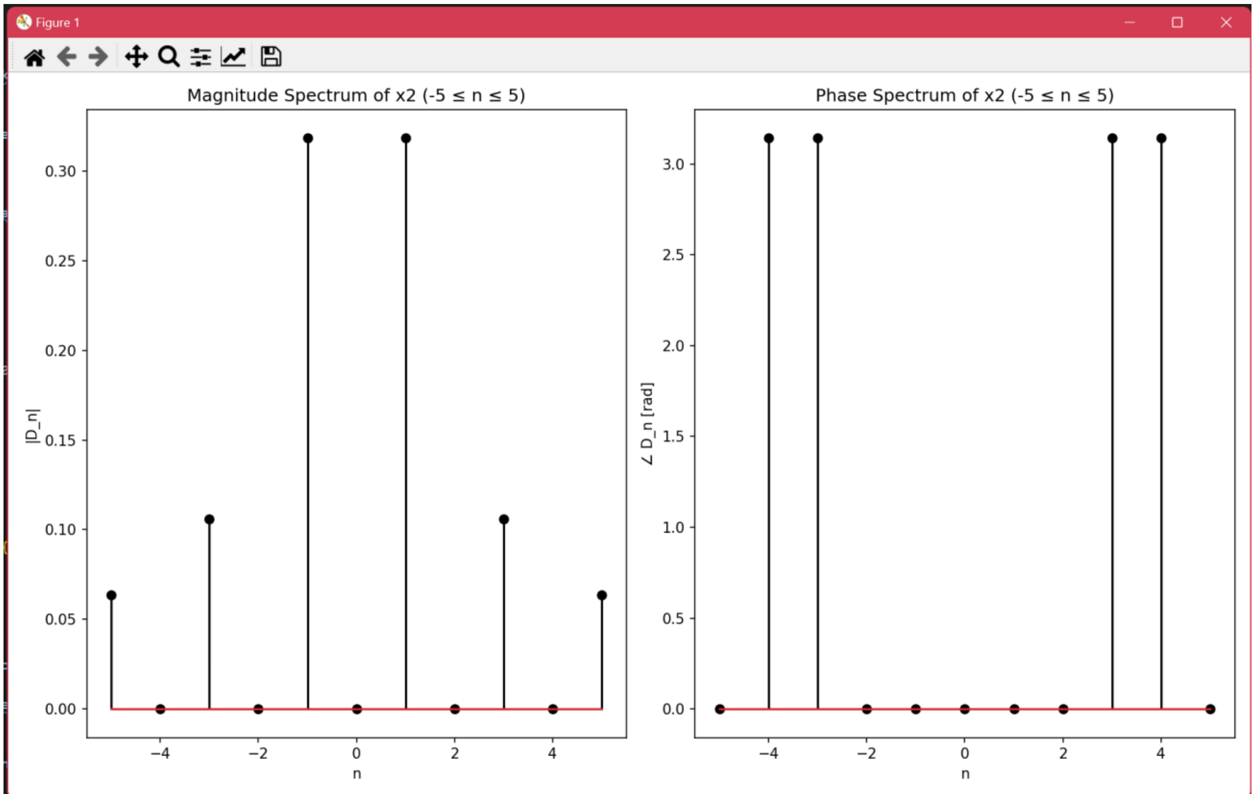


Figure 1
Magnitude Spectrum of x2 (-5 ≤ n ≤ 5)
|D_n|
n
Phase Spectrum of x2 (-5 ≤ n ≤ 5)
∠ D_n [rad]
n

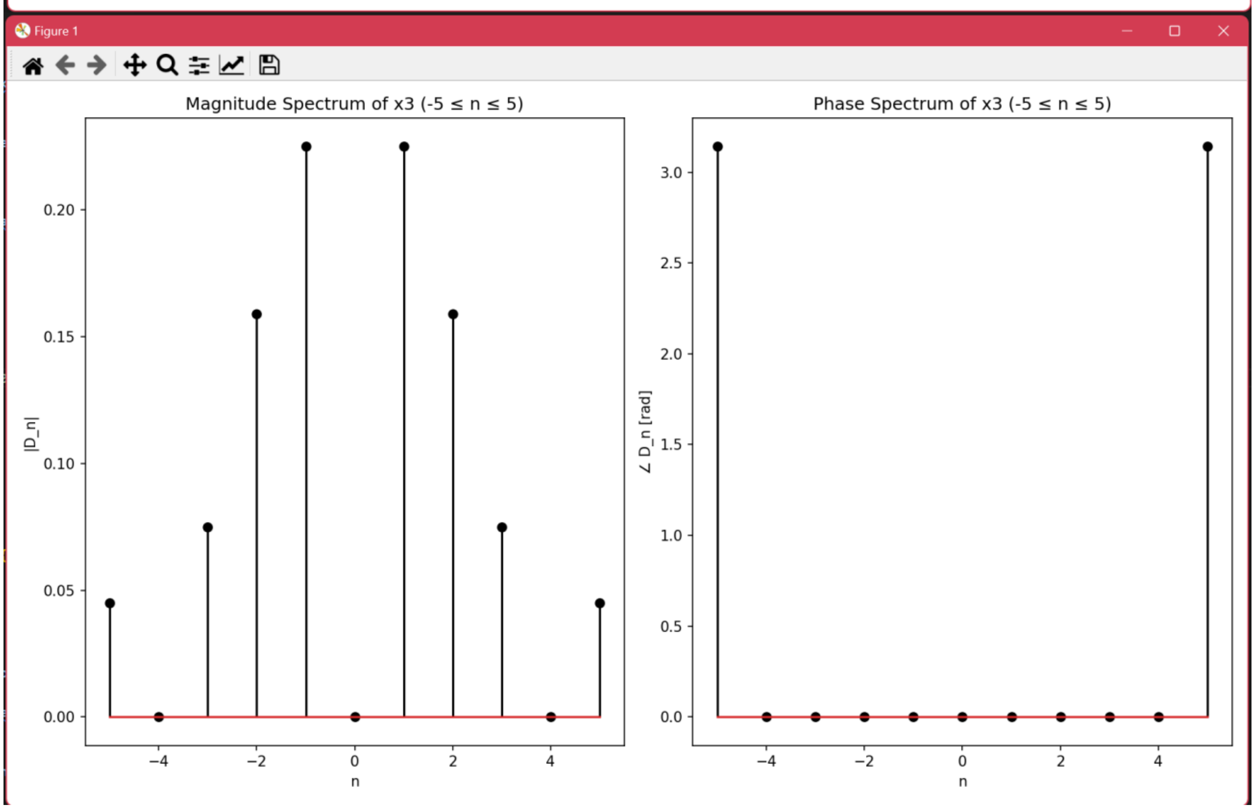


Figure 1
Magnitude Spectrum of x3 (-5 ≤ n ≤ 5)
|D_n|
n
Phase Spectrum of x3 (-5 ≤ n ≤ 5)
∠ D_n [rad]
n

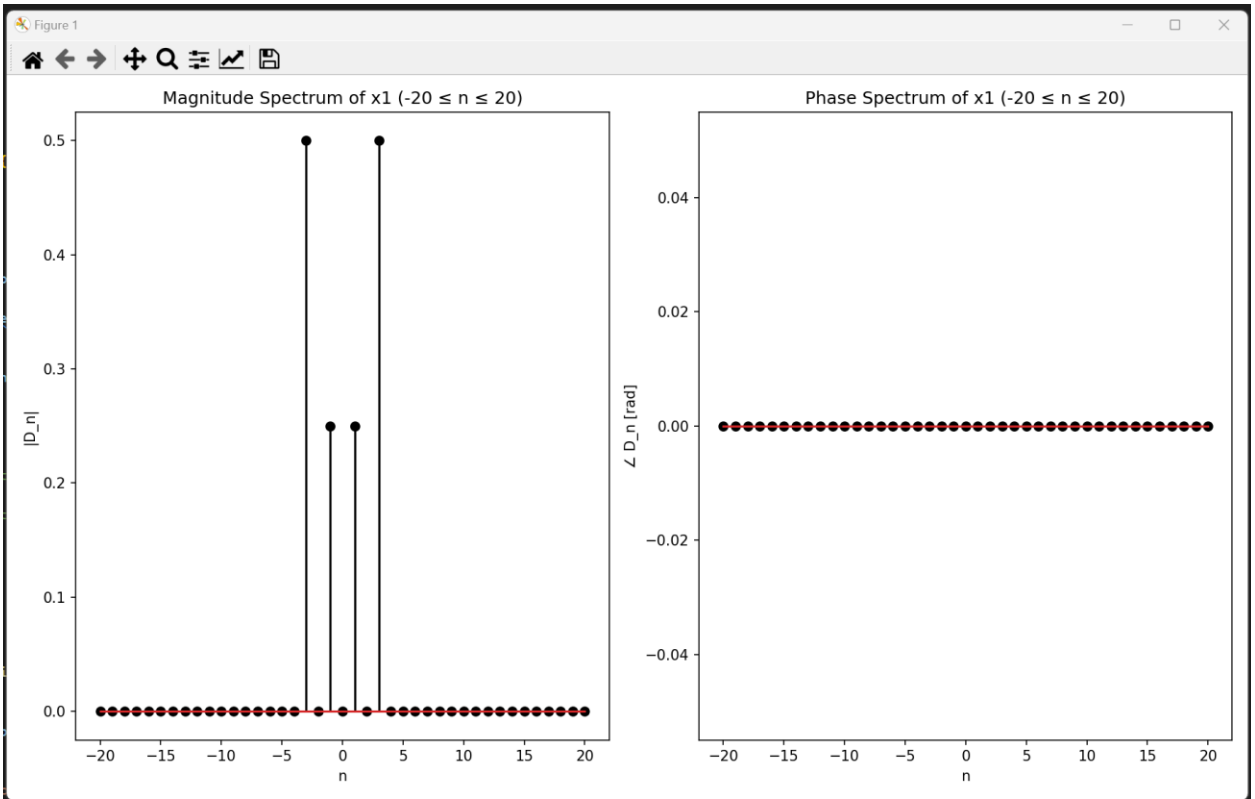


Figure 1
Magnitude Spectrum of x1 (-20 ≤ n ≤ 20)
|D_n|
n
Phase Spectrum of x1 (-20 ≤ n ≤ 20)
∠ D_n [rad]
n

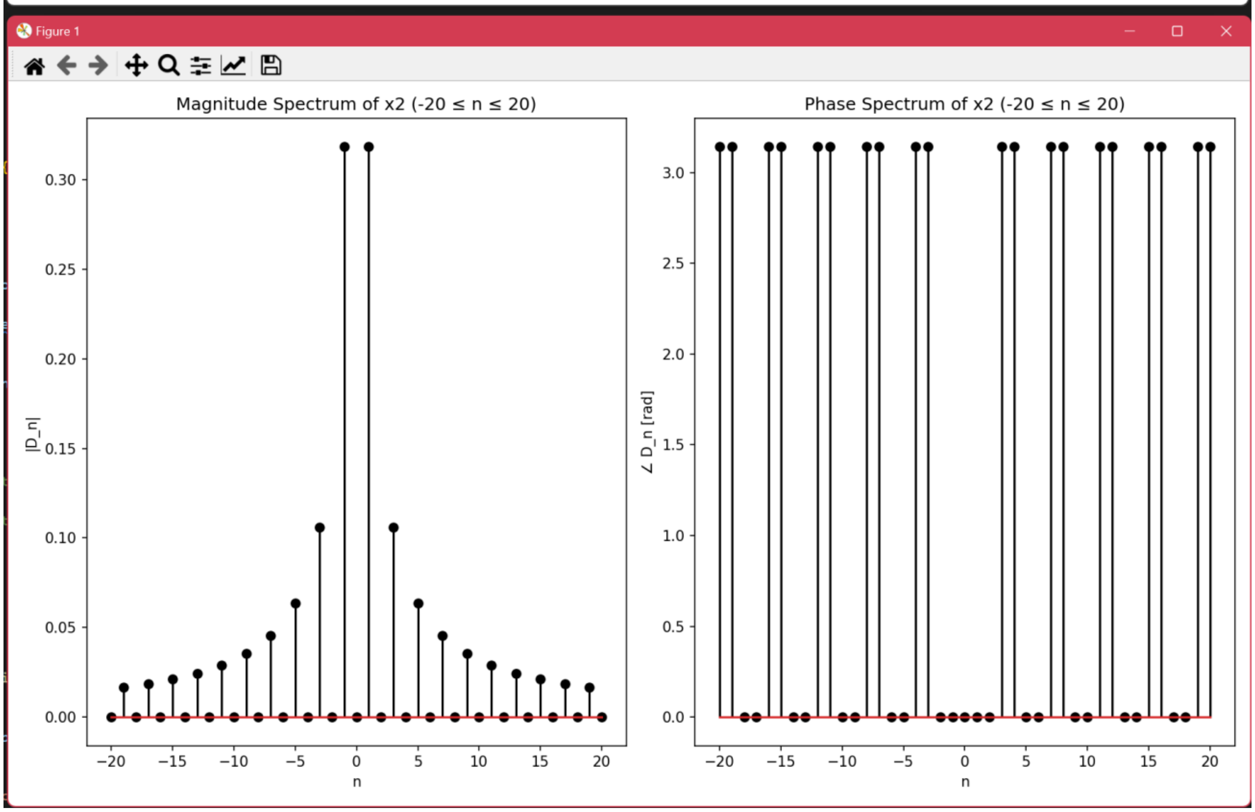


Figure 1
Magnitude Spectrum of x2 (-20 ≤ n ≤ 20)
|D_n|
n
Phase Spectrum of x2 (-20 ≤ n ≤ 20)
∠ D_n [rad]
n

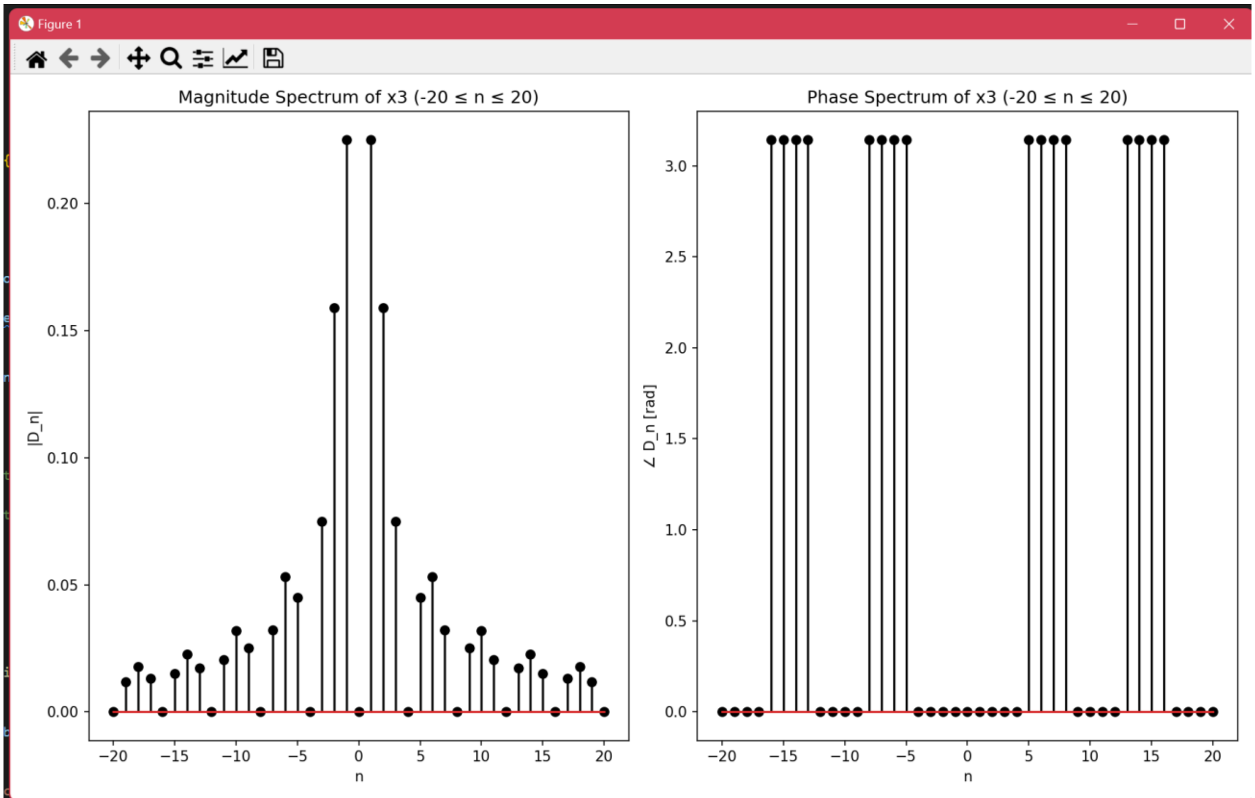


Figure 1
Magnitude Spectrum of x3 (-20 ≤ n ≤ 20)
|D_n|
n
Phase Spectrum of x3 (-20 ≤ n ≤ 20)
∠ D_n [rad]
n

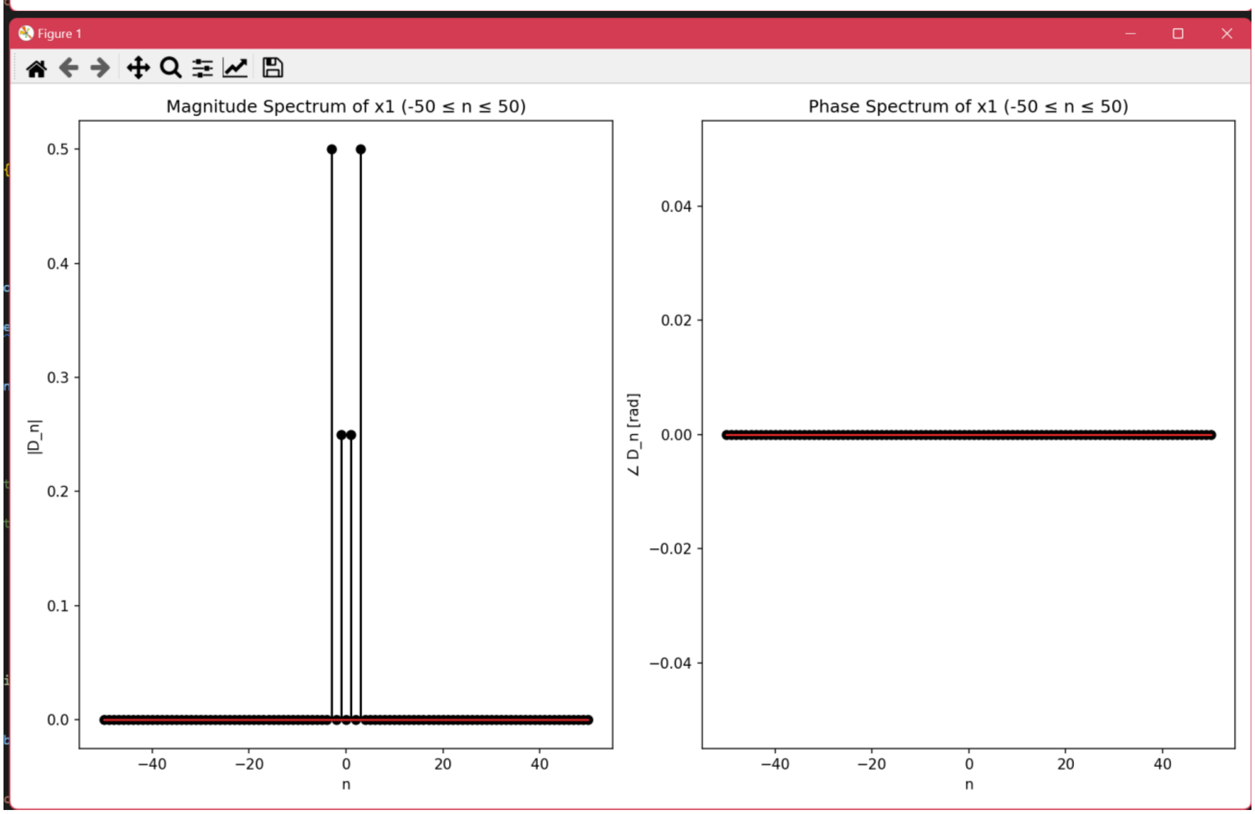


Figure 1
Magnitude Spectrum of x1 (-50 ≤ n ≤ 50)
|D_n|
n
Phase Spectrum of x1 (-50 ≤ n ≤ 50)
∠ D_n [rad]
n

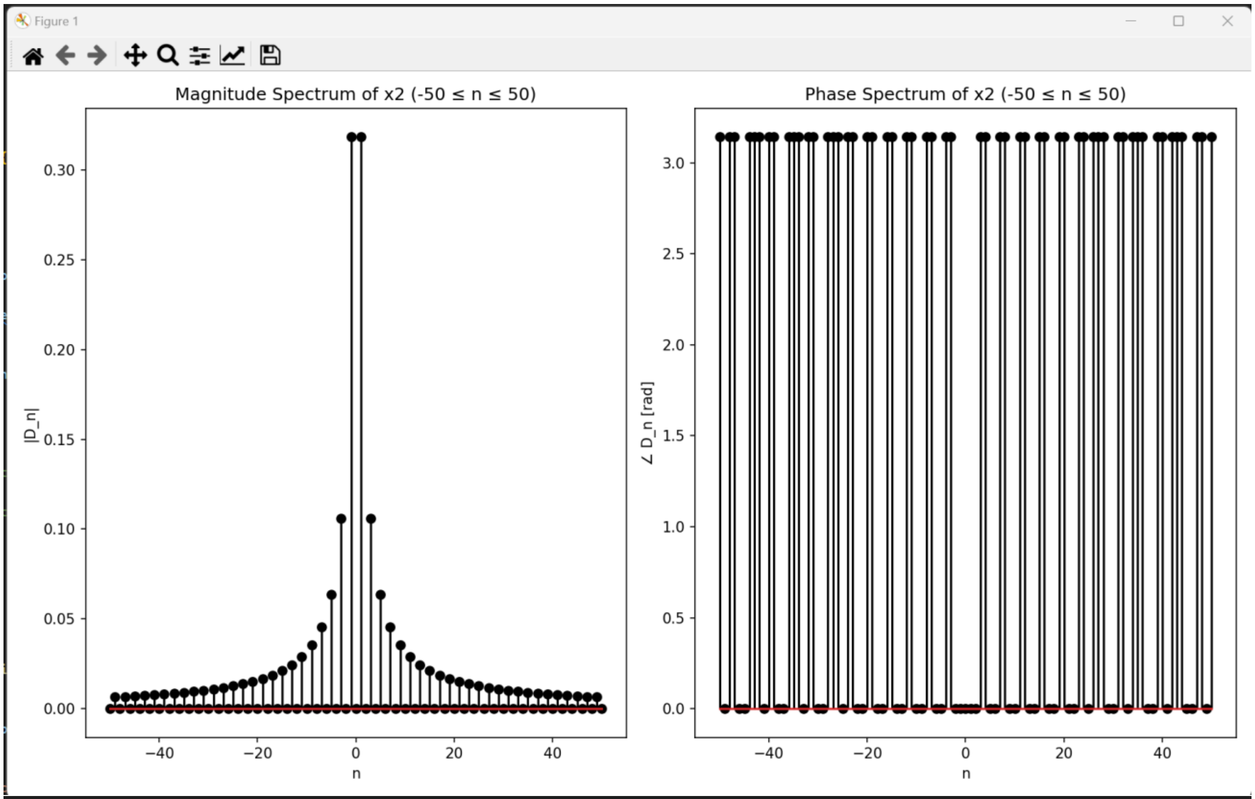


Figure 1
Magnitude Spectrum of x2 (-50 ≤ n ≤ 50)
Phase Spectrum of x2 (-50 ≤ n ≤ 50)
|D_n|
∠ D_n [rad]
n
n

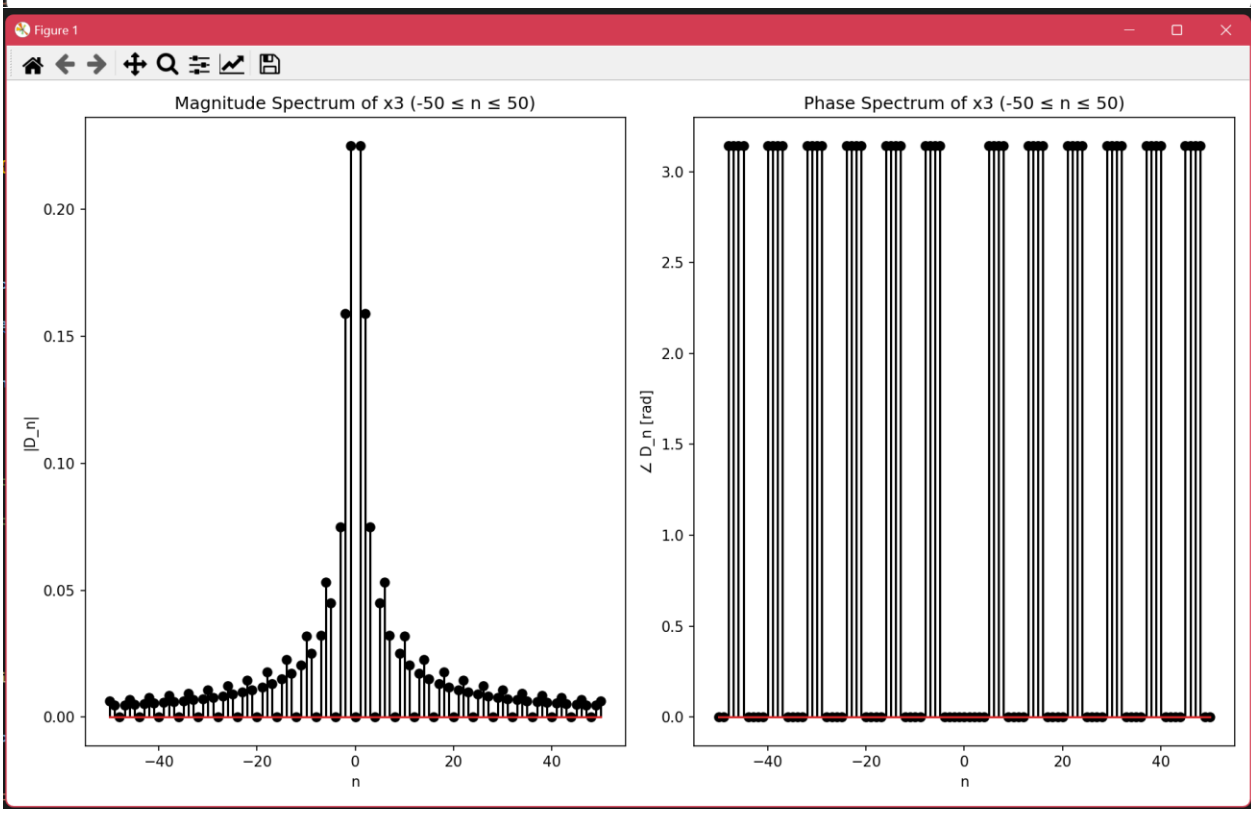


Figure 1
Magnitude Spectrum of x3 (-50 ≤ n ≤ 50)
Phase Spectrum of x3 (-50 ≤ n ≤ 50)
|D_n|
∠ D_n [rad]
n
n

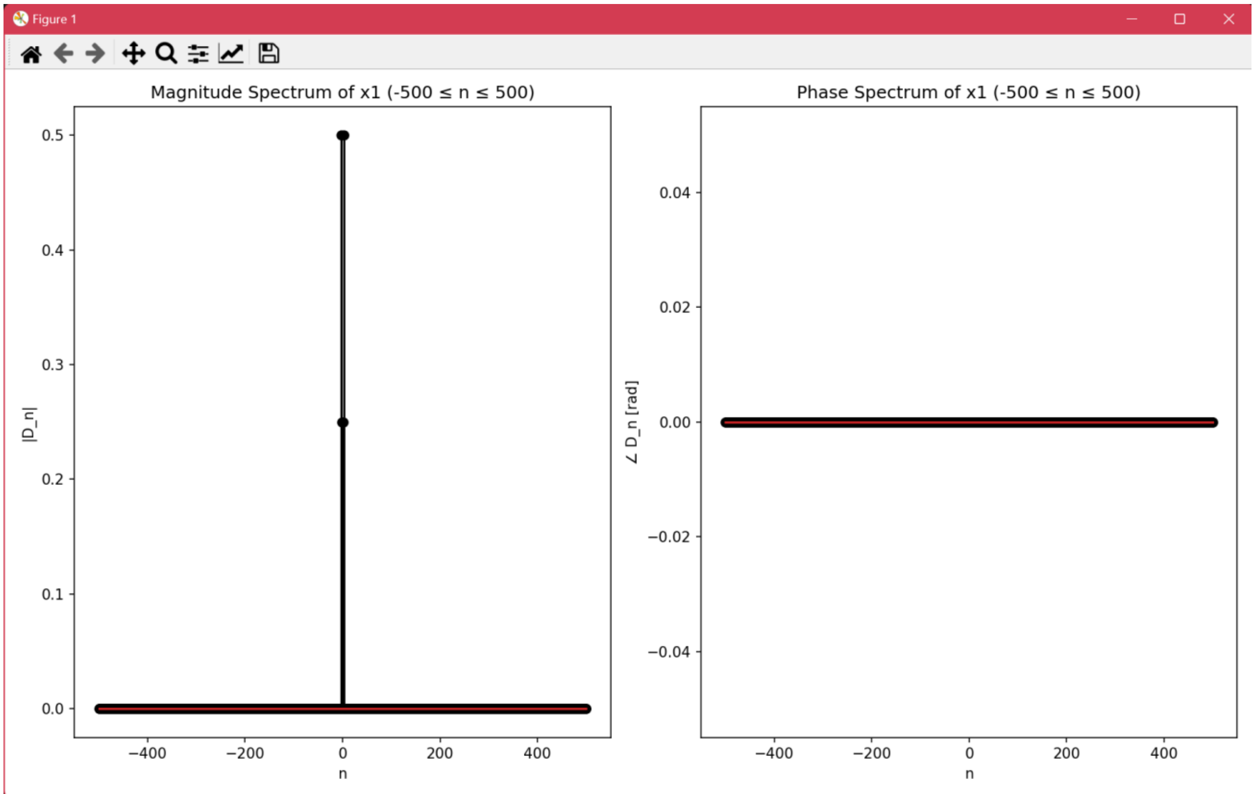


Figure 1
Magnitude Spectrum of x1 (-500 ≤ n ≤ 500)
|D_n|
n
Phase Spectrum of x1 (-500 ≤ n ≤ 500)
∠ D_n [rad]
n

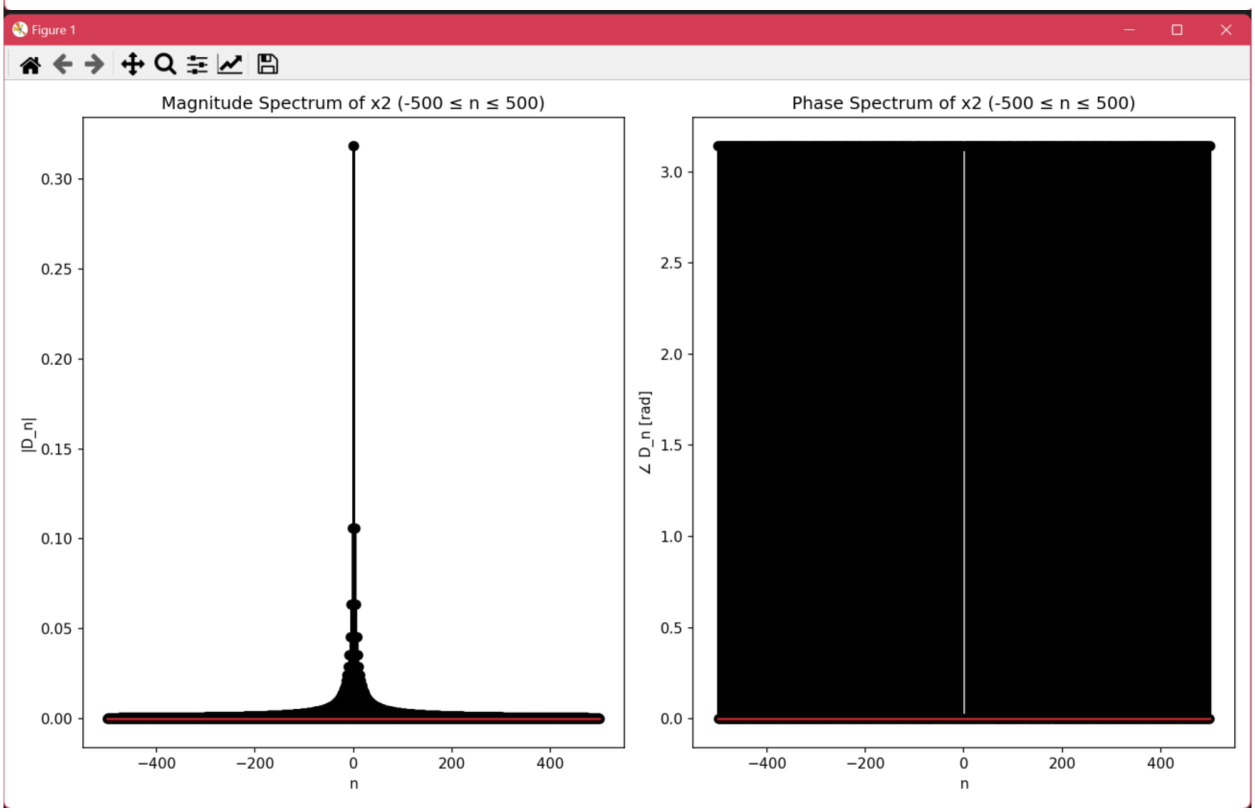
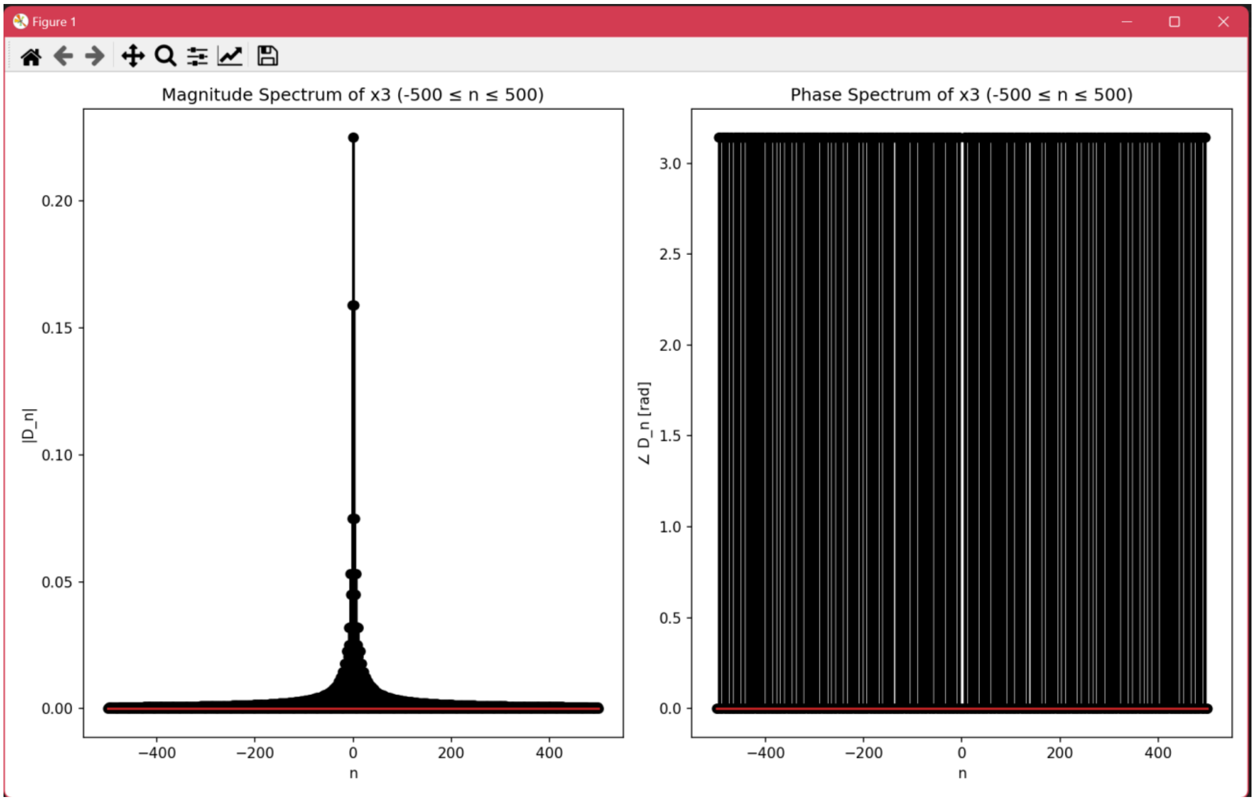


Figure 1
Magnitude Spectrum of x2 (-500 ≤ n ≤ 500)
|D_n|
n
Phase Spectrum of x2 (-500 ≤ n ≤ 500)
∠ D_n [rad]
n



- **Problem A.5 – A.6**

Code:

```python
# Part A.5 & A.6

def reconstruct_signal(Dn, n_range, t, period= 20 ):# Assuming T0 is 20 for x1(t), adjust as needed for other signals
    w0 = 2 * np.pi / period
    x_reconstructed = np.zeros_like(t, dtype=complex)

    for n, D in zip(n_range, Dn):
        x_reconstructed += D * np.exp(1j * n * w0 * t)

    return x_reconstructed.real

# Define the time vector for reconstruction
t = np.arange(-300, 301, 1)  # t from -300 to 300

# Reconstruct and plot signals for different ranges of n
for i, n_range in enumerate(ranges):
    plt.figure(figsize=(16, 12))

    for j in range(3):
        Dn_x = Dn(j, currentRange)

        if Dn_x is not None:
            x_reconstructed = reconstruct_signal(Dn_x, n_range, t)
            print(x_reconstructed)
            plt.subplot(3, 1, j+1)
            plt.plot(t, x_reconstructed, label=f"x"+str(j+1)+"(t) Reconstructed")
            plt.xlabel("t (sec)")
            plt.ylabel(f"x{j+1}(t)")
            plt.title(f"x{j+1}(t) Reconstructed from Fourier Coefficients n={n_range[0]} to {n_range[-1]})")
            plt.grid()
            plt.legend()

    plt.subplots_adjust(hspace=0.5)
    plt.show()
    plt.tight_layout()
```

Results:

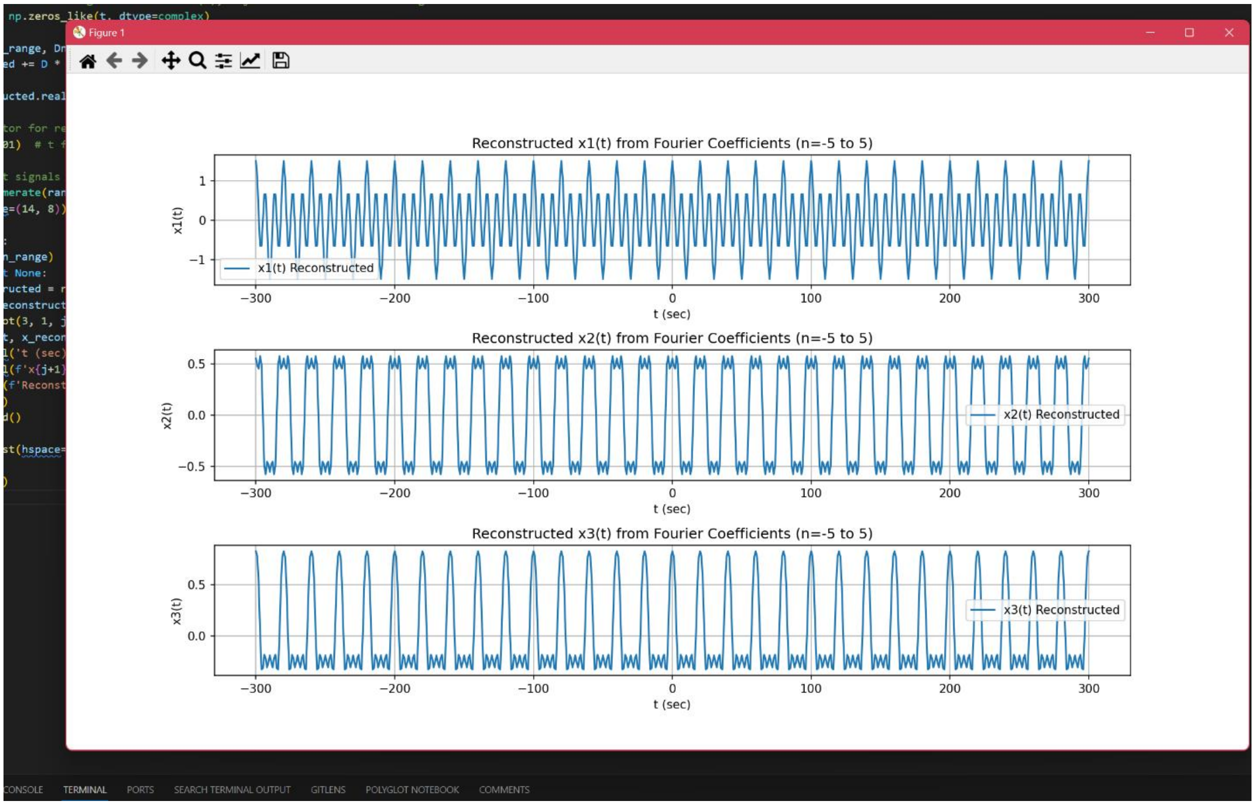


Figure 1
Reconstructed x1(t) from Fourier Coefficients (n=-5 to 5)
x1(t)
x1(t) Reconstructed
t (sec)
Reconstructed x2(t) from Fourier Coefficients (n=-5 to 5)
x2(t)
x2(t) Reconstructed
t (sec)
Reconstructed x3(t) from Fourier Coefficients (n=-5 to 5)
x3(t)
x3(t) Reconstructed
t (sec)
CONSOLE
TERMINAL
PORTS
SEARCH TERMINAL OUTPUT
GITLENS
POLYGLOT NOTEBOOK
COMMENTS

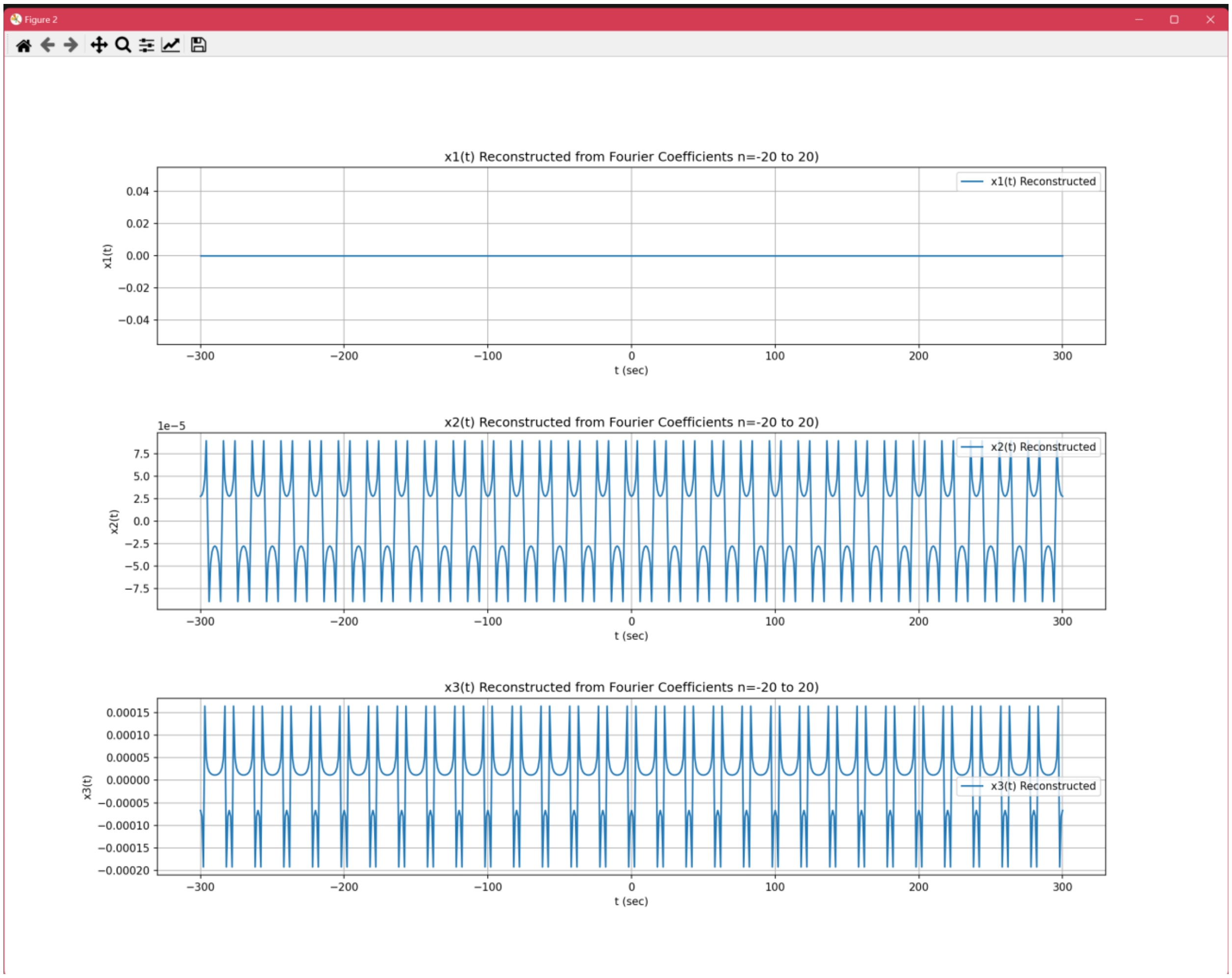


Figure 2
x1(t) Reconstructed from Fourier Coefficients n=-20 to 20)
x1(t) Reconstructed
0.04
0.02
0.00
−0.02
−0.04
x1(t)
−300
−200
−100
0
100
200
300
t (sec)
x2(t) Reconstructed from Fourier Coefficients n=-20 to 20)
1e−5
x2(t) Reconstructed
7.5
5.0
2.5
0.0
−2.5
−5.0
−7.5
x2(t)
−300
−200
−100
0
100
200
300
t (sec)
x3(t) Reconstructed from Fourier Coefficients n=-20 to 20)
x3(t) Reconstructed
0.00015
0.00010
0.00005
0.00000
−0.00005
−0.00010
−0.00015
−0.00020
x3(t)
−300
−200
−100
0
100
200
300
t (sec)

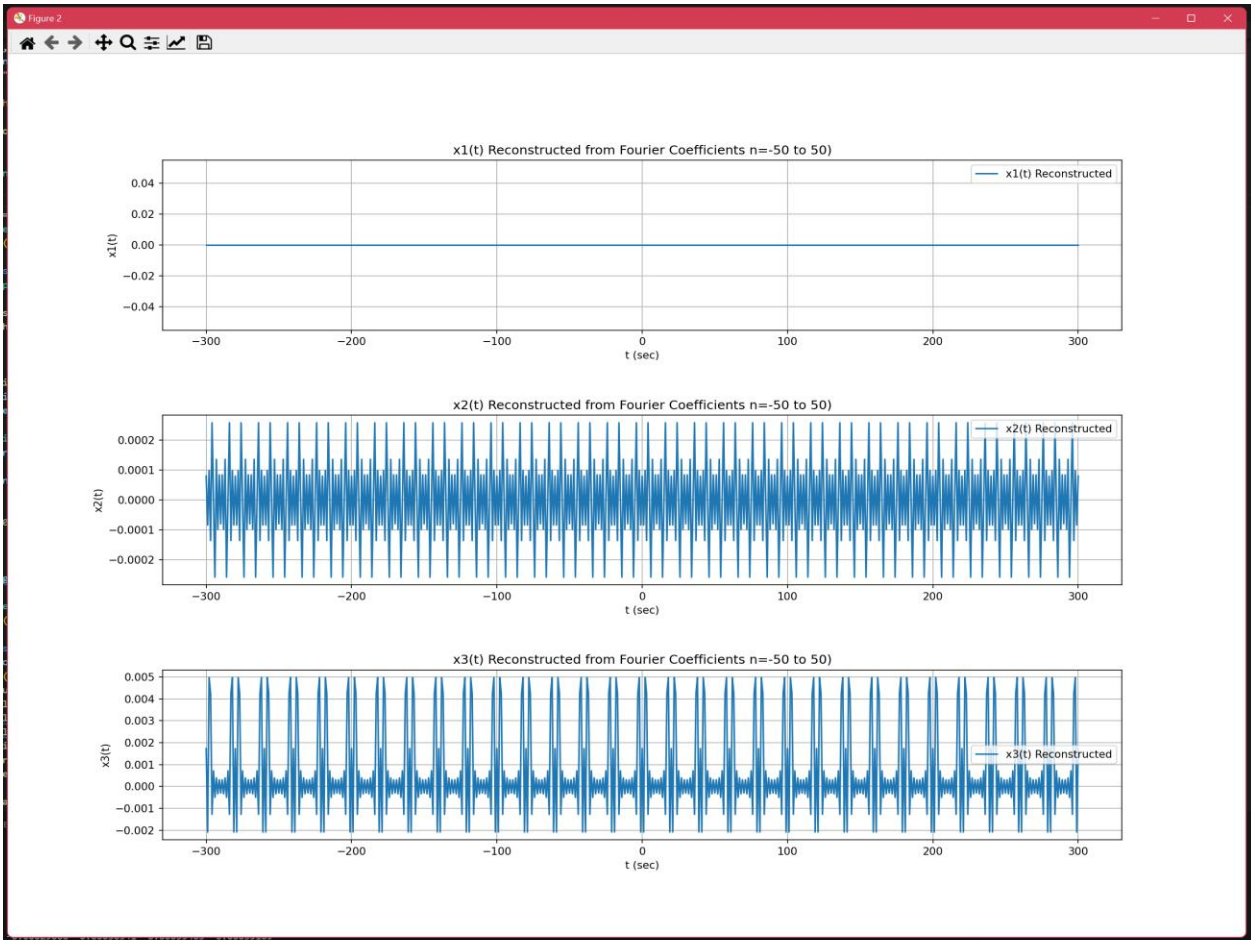
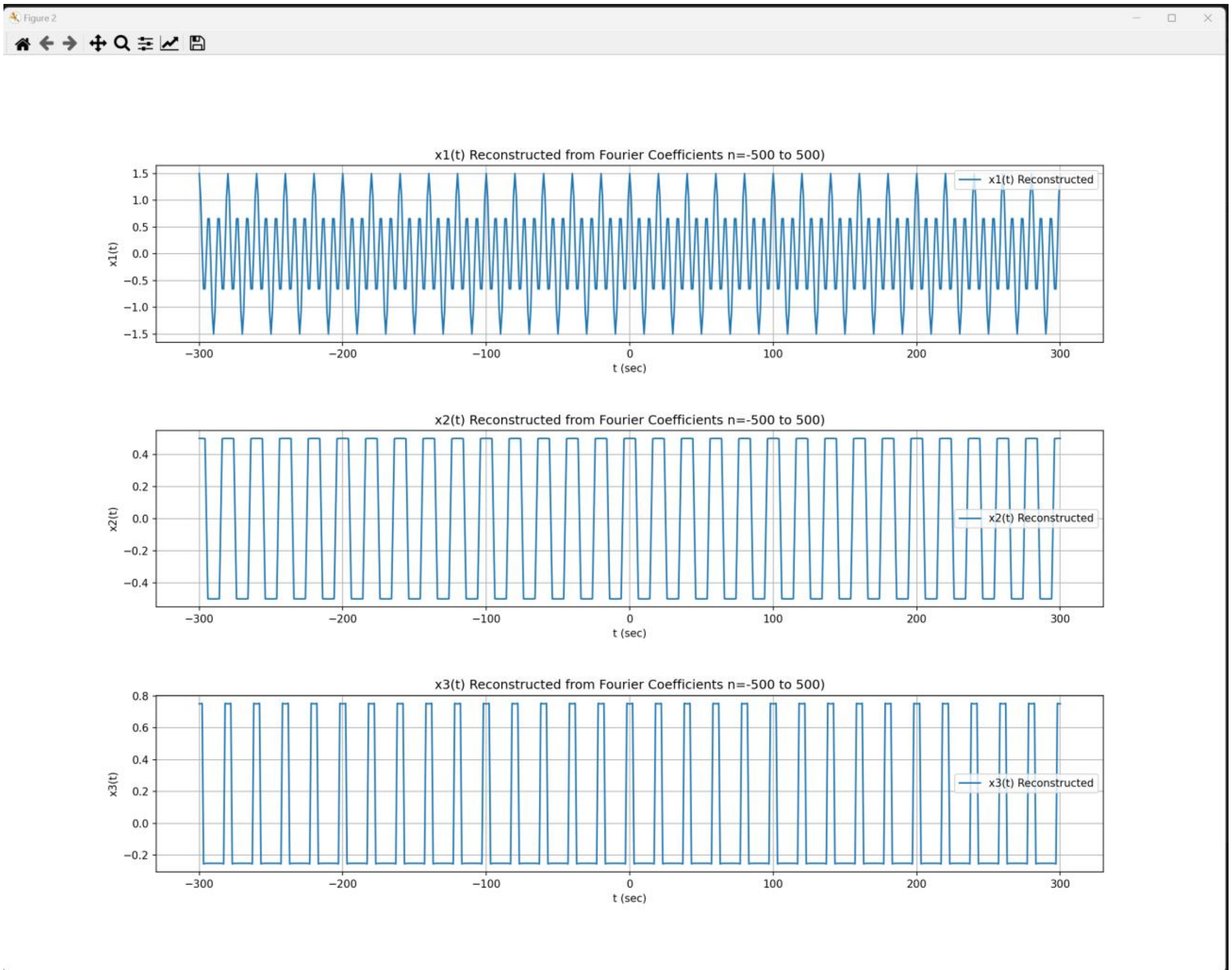


Figure 2
x1(t) Reconstructed from Fourier Coefficients n=-50 to 50)
x1(t) Reconstructed
x1(t)
t (sec)
x2(t) Reconstructed from Fourier Coefficients n=-50 to 50)
x2(t) Reconstructed
x2(t)
t (sec)
x3(t) Reconstructed from Fourier Coefficients n=-50 to 50)
x3(t) Reconstructed
x3(t)
t (sec)



Part B. Discussion

- **Problem B.1**

  Both x1(t) and x2(t) share a fundamental frequency of π/10, reflecting their periodic nature. In contrast, x3(t) has a different periodic characteristic, with a fundamental frequency of π/20, defining their behavior in the frequency domain.

- **Problem B.2**

  x1(t) exhibits a simpler harmonic structure with a limited set of Fourier coefficients, while x2(t) possesses a potentially infinite set, highlighting its more complex harmonic content in the frequency domain.

- **Problem B.3**

  Despite a shared rectangular pulse shape, x2(t) and x3(t) have differing Fourier coefficients due to distinct periods. Varied periods directly influence their fundamental frequencies, leading to different distributions of non-zero Fourier coefficients.

- **Problem B.4**

In x4(t), similar to x2(t) but shifted downward by 0.5 units, the DC component is -0.5, altered by the downward shift from x2(t)'s zero DC component.

- **Problem B.5**

  Augmenting the number of Fourier coefficients for x1(t) and x2(t) enhances the accuracy of signal reconstruction, minimizing approximation errors inherent in Fourier series estimations.

- **Problem B.6**

  While ideal perfect reconstruction necessitates an infinite number of Fourier coefficients, signals like x1(t) with limited bandwidth require only a finite number. More complex waveforms such as x2(t) and x3(t) benefit from increased coefficients for accurate reconstruction.

- **Problem B.7**

  Storing Fourier coefficients is an efficient means of signal representation, especially for large datasets or limited storage space. This method captures crucial frequency components, enabling accurate reconstruction when needed, commonly used in signal processing applications for data compression and storage. Feasibility depends on signal characteristics and application requirements.