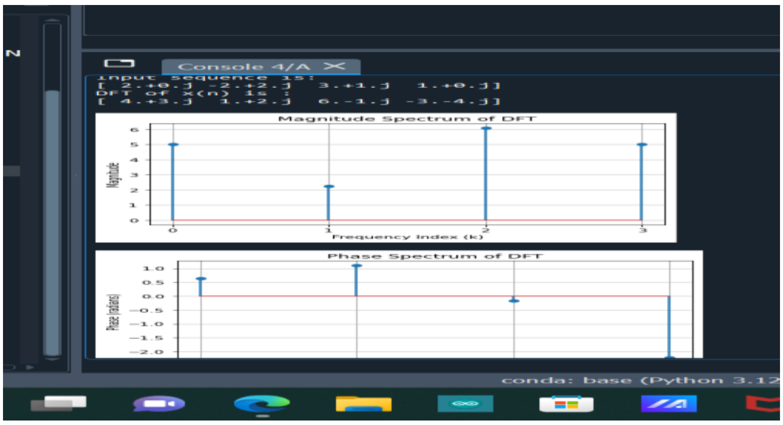
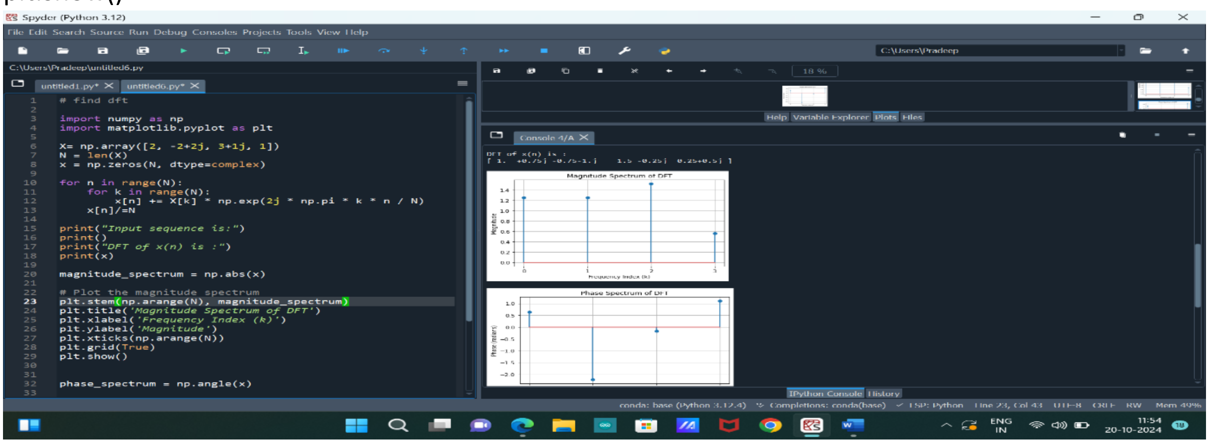
#DFT OF GIVEN SEQUENCE EXP-1

```
import numpy as np
import matplotlib.pyplot as plt
x = np.array([2, -2+2j, 3+1j, 1])
N = len(x)
X = np.zeros(N, dtype=complex)

for k in range(N):
    for n in range(N):
        X[k] += x[n] * np.exp(-2j * np.pi * k * n / N)
print("Input sequence is:")
print(x)
print("DFT of x(n) is :")
print(X)

magnitude_spectrum = np.abs(X)
# Plot the magnitude spectrum
plt.stem(np.arange(N), magnitude_spectrum)
plt.title('Magnitude Spectrum of DFT')
plt.xlabel('Frequency Index (k)')
plt.ylabel('Magnitude')
plt.xticks(np.arange(N))
plt.grid(True)
plt.show()


phase_spectrum = np.angle(X)
# Plot the phase spectrum
plt.stem(np.arange(N), phase_spectrum)
plt.title('Phase Spectrum of DFT')
plt.xlabel('Frequency Index (k)')
plt.ylabel('Phase (radians)')
plt.xticks(np.arange(N))
plt.grid(True)
plt.show()
```



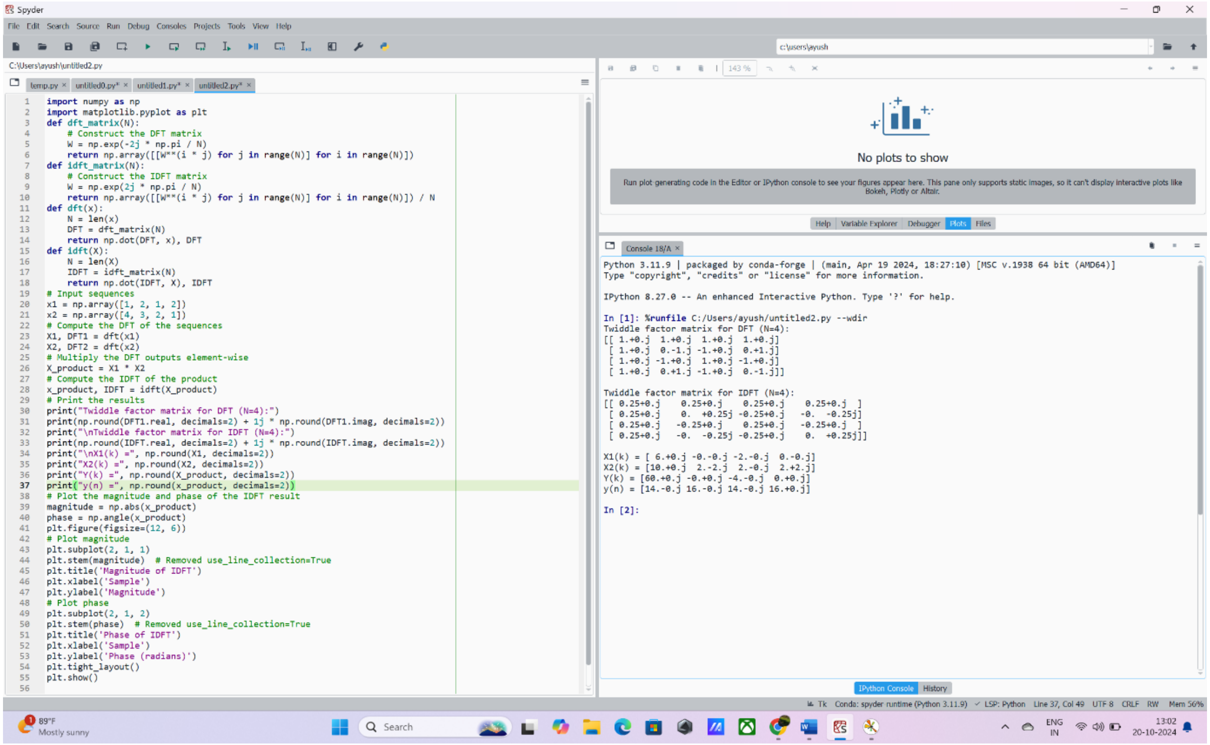
```
# IDFT OF GIVEN SEQUENCE EXP-1

import numpy as np
import matplotlib.pyplot as plt
X= np.array([2, -2+2j, 3+1j, 1])
N = len(X)
x = np.zeros(N, dtype=complex)

for n in range(N):
   for k in range(N):
      x[n] += X[k] * np.exp(2j * np.pi * k * n / N)
   x[n]/=N
print("Input sequence is:")
print()
print("DFT of x(n) is :")
print(x)

magnitude_spectrum = np.abs(x)
# Plot the magnitude spectrum
plt.stem(np.arange(N), magnitude_spectrum)
plt.title('Magnitude Spectrum of DFT')
plt.xlabel('Frequency Index (k)')
plt.ylabel('Magnitude')
plt.xticks(np.arange(N))
plt.grid(True)
plt.show()

phase_spectrum = np.angle(x)
# Plot the phase spectrum
plt.stem(np.arange(N), phase_spectrum)
plt.title('Phase Spectrum of DFT')
plt.xlabel('Frequency Index (k)')
plt.ylabel('Phase (radians)')
plt.xticks(np.arange(N))
plt.grid(True)
plt.show()
```

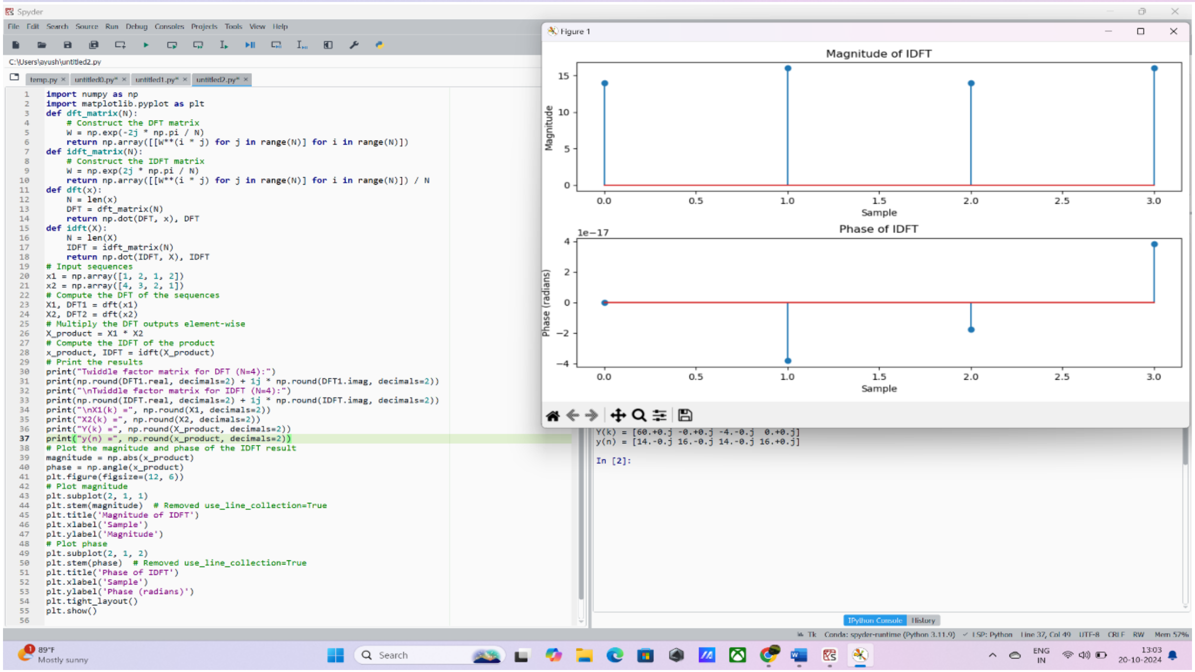


 # Perform circular convolution EXP-2

```
import numpy as np
import matplotlib.pyplot as plt
def dft_matrix(N):
   # Construct the DFT matrix
   W = np.exp(-2j * np.pi / N)
   return np.array([[W**(i * j) for j in range(N)] for i in range(N)])
def idft_matrix(N):
   # Construct the IDFT matrix
   W = np.exp(2j * np.pi / N)
   return np.array([[W**(i * j) for j in range(N)] for i in range(N)]) / N
def dft(x):
   N = len(x)
   DFT = dft_matrix(N)
   return np.dot(DFT, x), DFT
def idft(X):
   N = len(X)
   IDFT = idft_matrix(N)
   return np.dot(IDFT, X), IDFT
# Input sequences
x1 = np.array([1, 2, 1, 2])
x2 = np.array([4, 3, 2, 1])
# Compute the DFT of the sequences
X1, DFT1 = dft(x1)
X2, DFT2 = dft(x2)
# Multiply the DFT outputs element-wise
X_product = X1 * X2
# Compute the IDFT of the product
x_product, IDFT = idft(X_product)
# Print the results
print("Twiddle factor matrix for DFT (N=4):")
print(np.round(DFT1.real, decimals=2) + 1j * np.round(DFT1.imag, decimals=2))
print("\nTwiddle factor matrix for IDFT (N=4):")
print(np.round(IDFT.real, decimals=2) + 1j * np.round(IDFT.imag, decimals=2))
print("\nX1(k) =", np.round(X1, decimals=2))
print("X2(k) =", np.round(X2, decimals=2))
print("Y(k) =", np.round(X_product, decimals=2))
print("y(n) =", np.round(x_product, decimals=2))
# Plot the magnitude and phase of the IDFT result
magnitude = np.abs(x_product)
phase = np.angle(x_product)
plt.figure(figsize=(12, 6))
# Plot magnitude
plt.subplot(2, 1, 1)
plt.stem(magnitude)  # Removed use_line_collection=True
plt.title('Magnitude of IDFT')
plt.xlabel('Sample')
```

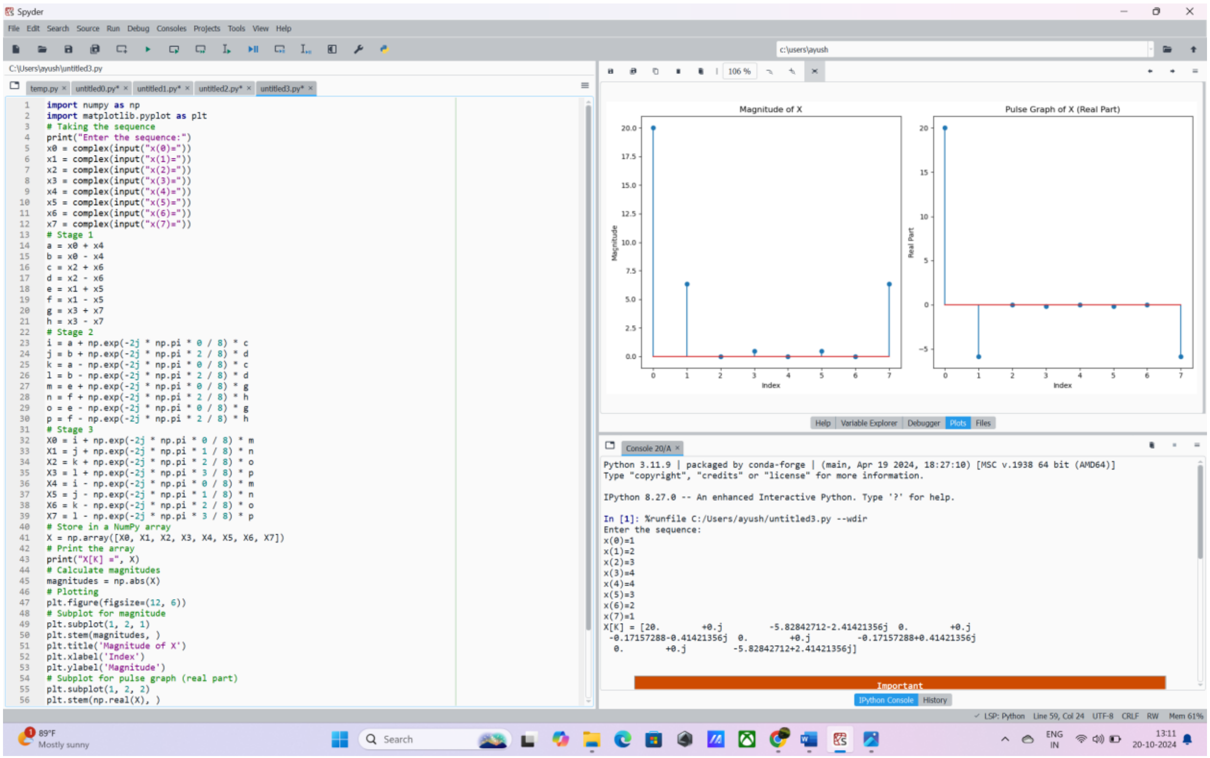
```python
plt.ylabel('Magnitude')
# Plot phase
plt.subplot(2, 1, 2)
plt.stem(phase)  # Removed use_line_collection=True
plt.title('Phase of IDFT')
plt.xlabel('Sample')
plt.ylabel('Phase (radians)')
plt.tight_layout()
plt.show()
```





```
# DFT Using DIT FFT EXP-3
import numpy as np
import matplotlib.pyplot as plt

# Taking the sequence
print("Enter the sequence:")
x0 = complex(input("x(0)="))
x1 = complex(input("x(1)="))
x2 = complex(input("x(2)="))
x3 = complex(input("x(3)="))
x4 = complex(input("x(4)="))
x5 = complex(input("x(5)="))
x6 = complex(input("x(6)="))
x7 = complex(input("x(7)="))

# Stage 1
a = x0 + x4
b = x0 - x4
c = x2 + x6
d = x2 - x6
e = x1 + x5
f = x1 - x5
g = x3 + x7
h = x3 - x7

# Stage 2
i = a + np.exp(-2j * np.pi * 0 / 8) * c
j = b + np.exp(-2j * np.pi * 2 / 8) * d
k = a - np.exp(-2j * np.pi * 0 / 8) * c
l = b - np.exp(-2j * np.pi * 2 / 8) * d
m = e + np.exp(-2j * np.pi * 0 / 8) * g
n = f + np.exp(-2j * np.pi * 2 / 8) * h
o = e - np.exp(-2j * np.pi * 0 / 8) * g
p = f - np.exp(-2j * np.pi * 2 / 8) * h

# Stage 3
X0 = i + np.exp(-2j * np.pi * 0 / 8) * m
X1 = j + np.exp(-2j * np.pi * 1 / 8) * n
X2 = k + np.exp(-2j * np.pi * 2 / 8) * o
X3 = l + np.exp(-2j * np.pi * 3 / 8) * p
X4 = i - np.exp(-2j * np.pi * 0 / 8) * m
X5 = j - np.exp(-2j * np.pi * 1 / 8) * n
X6 = k - np.exp(-2j * np.pi * 2 / 8) * o
X7 = l - np.exp(-2j * np.pi * 3 / 8) * p

# Store in a NumPy array
X = np.array([X0, X1, X2, X3, X4, X5, X6, X7])
```

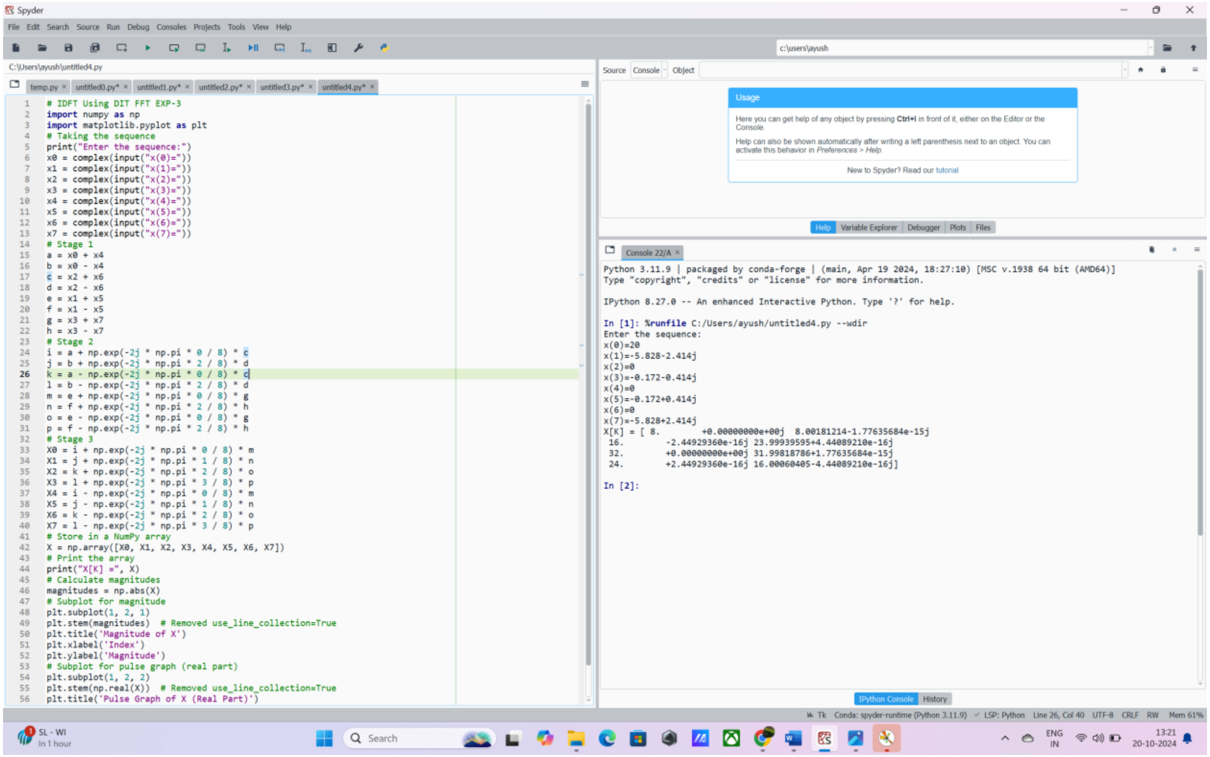
```
# Print the array
print("X[K] =", X)

# Calculate magnitudes
magnitudes = np.abs(X)

# Plotting
plt.figure(figsize=(12, 6))

# Subplot for magnitude
plt.subplot(1, 2, 1)
plt.stem(magnitudes, )
plt.title('Magnitude of X')
plt.xlabel('Index')
plt.ylabel('Magnitude')

# Subplot for pulse graph (real part)
plt.subplot(1, 2, 2)
plt.stem(np.real(X), )
plt.title('Pulse Graph of X (Real Part)')
plt.xlabel('Index')
plt.ylabel('Real Part')

plt.tight_layout()
plt.show()
```

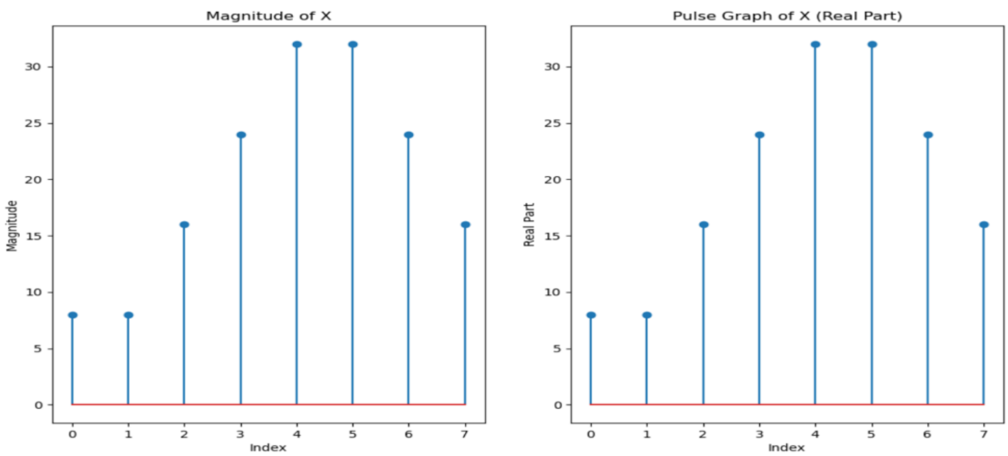


IDFT Using DIT FFT EXP-3

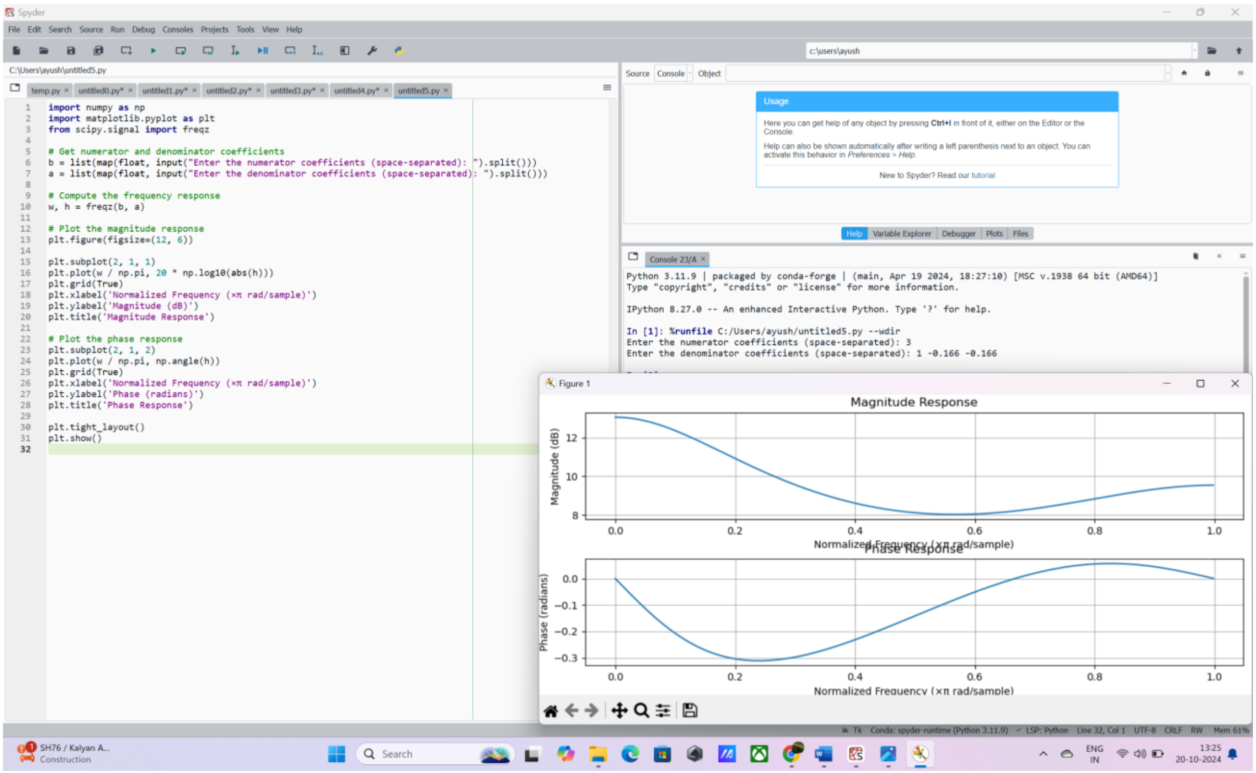
```
# IDFT Using DIT FFT EXP-3
import numpy as np
import matplotlib.pyplot as plt
# Taking the sequence
print("Enter the sequence:")
x0 = complex(input("x(0)="))
x1 = complex(input("x(1)="))
x2 = complex(input("x(2)="))
x3 = complex(input("x(3)="))
x4 = complex(input("x(4)="))
x5 = complex(input("x(5)="))
x6 = complex(input("x(6)="))
x7 = complex(input("x(7)="))
# Stage 1
a = x0 + x4
b = x0 - x4
c = x2 + x6
d = x2 - x6
e = x1 + x5
f = x1 - x5
g = x3 + x7
h = x3 - x7
# Stage 2
i = a + np.exp(-2j * np.pi * 0 / 8) * c
j = b + np.exp(-2j * np.pi * 2 / 8) * d
k = a - np.exp(-2j * np.pi * 0 / 8) * c
l = b - np.exp(-2j * np.pi * 2 / 8) * d
m = e + np.exp(-2j * np.pi * 0 / 8) * g
n = f + np.exp(-2j * np.pi * 2 / 8) * h
o = e - np.exp(-2j * np.pi * 0 / 8) * g
p = f - np.exp(-2j * np.pi * 2 / 8) * h
# Stage 3
X0 = i + np.exp(-2j * np.pi * 0 / 8) * m
X1 = j + np.exp(-2j * np.pi * 1 / 8) * n
X2 = k + np.exp(-2j * np.pi * 2 / 8) * o
X3 = l + np.exp(-2j * np.pi * 3 / 8) * p
X4 = i - np.exp(-2j * np.pi * 0 / 8) * m
X5 = j - np.exp(-2j * np.pi * 1 / 8) * n
X6 = k - np.exp(-2j * np.pi * 2 / 8) * o
X7 = l - np.exp(-2j * np.pi * 3 / 8) * p
# Store in a NumPy array
X = np.array([X0, X1, X2, X3, X4, X5, X6, X7])
# Print the array
print("X[K] =", X)
# Calculate magnitudes
magnitudes = np.abs(X)
```

```
# Subplot for magnitude
plt.subplot(1, 2, 1)
plt.stem(magnitudes)  # Removed use_line_collection=True
plt.title('Magnitude of X')
plt.xlabel('Index')
plt.ylabel('Magnitude')
# Subplot for pulse graph (real part)
plt.subplot(1, 2, 2)
plt.stem(np.real(X))  # Removed use_line_collection=True
plt.title('Pulse Graph of X (Real Part)')
plt.xlabel('Index')
plt.ylabel('Real Part')
```



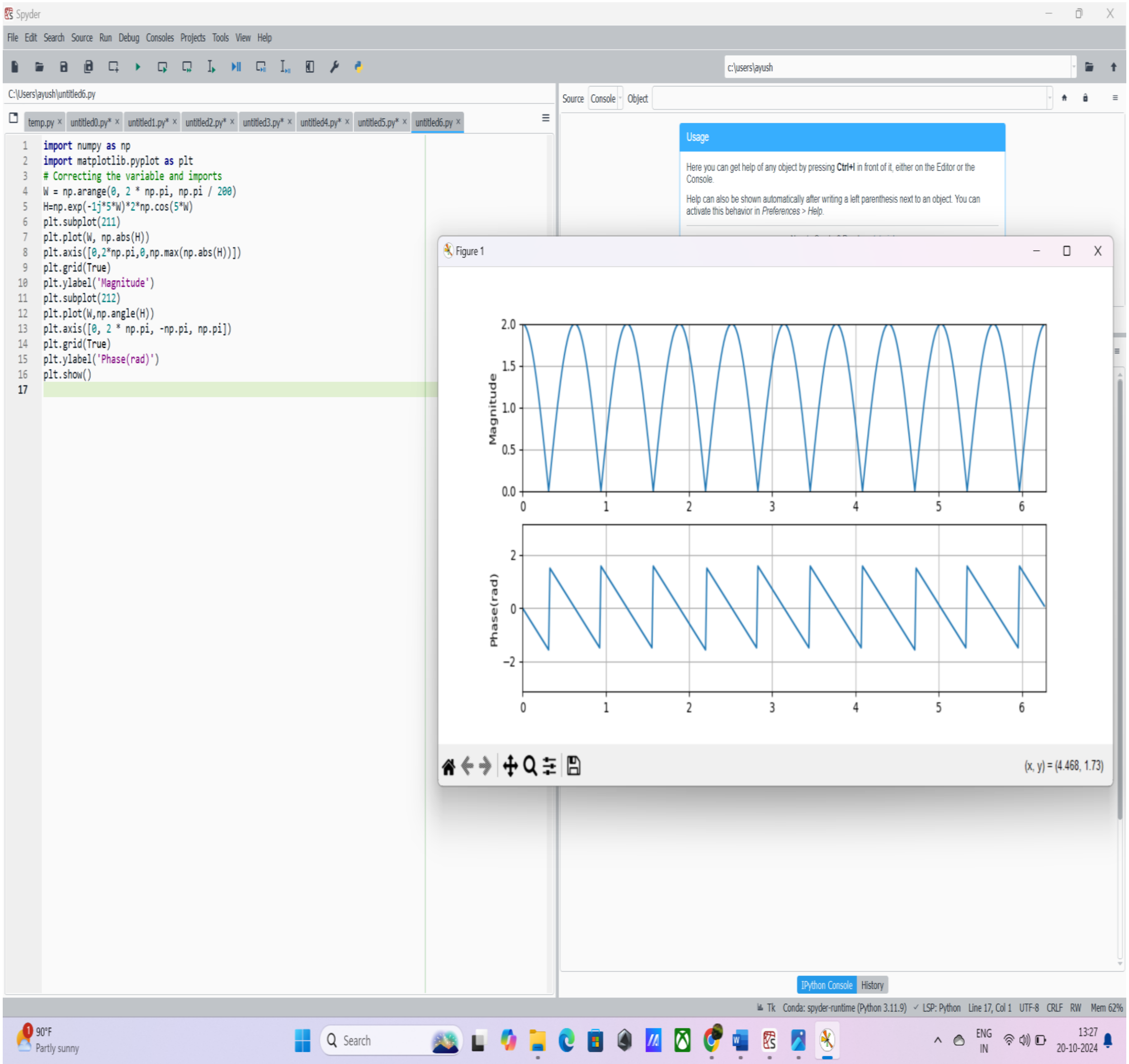


```
# Take numerator and denominator from user Exp-4
import numpy as np
import matplotlib.pyplot as plt
from scipy.signal import freqz
# Get numerator and denominator coefficients
b = list(map(float, input("Enter the numerator coefficients (space-separated): ").split()))
a = list(map(float, input("Enter the denominator coefficients (space-separated): ").split()))
# Compute the frequency response
w, h = freqz(b, a)
# Plot the magnitude response
plt.figure(figsize=(12, 6))
plt.subplot(2, 1, 1)
plt.plot(w / np.pi, 20 * np.log10(abs(h)))
plt.grid(True)
plt.xlabel('Normalized Frequency (×π rad/sample)')
plt.ylabel('Magnitude (dB)')
plt.title('Magnitude Response')
# Plot the phase response
plt.subplot(2, 1, 2)
plt.plot(w / np.pi, np.angle(h))
plt.grid(True)
plt.xlabel('Normalized Frequency (×π rad/sample)')
plt.ylabel('Phase (radians)')
plt.title('Phase Response')
plt.tight_layout()
plt.show()
```



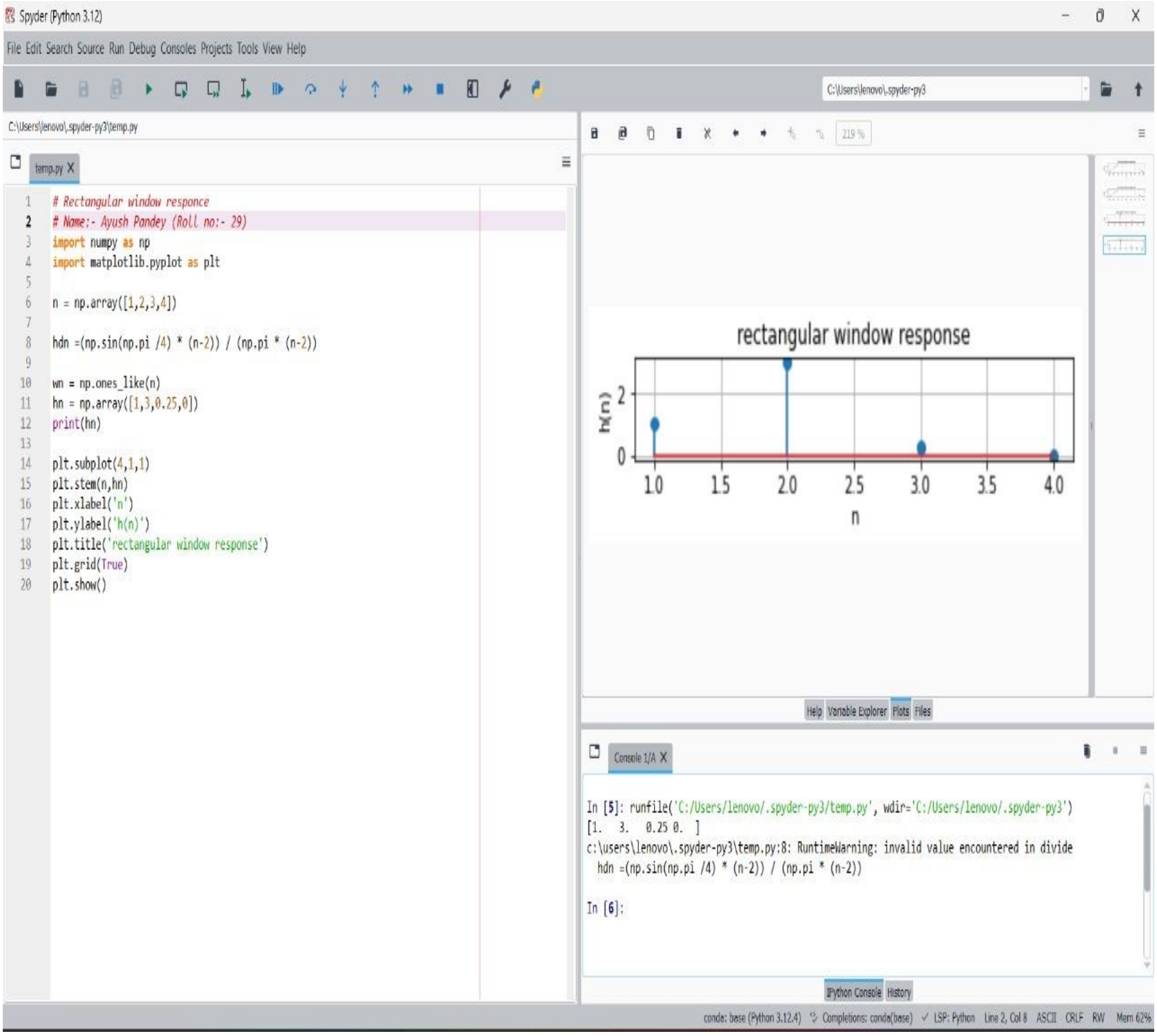
finding /compute the sequence and sketch amplitude/phase

```
import numpy as np
import matplotlib.pyplot as plt
# Correcting the variable and imports
W = np.arange(0, 2 * np.pi, np.pi / 200)
H=np.exp(-1j*5*W)*2*np.cos(5*W)
plt.subplot(211)
plt.plot(W, np.abs(H))
plt.axis([0,2*np.pi,0,np.max(np.abs(H))])
plt.grid(True)
plt.ylabel('Magnitude')
plt.subplot(212)
plt.plot(W,np.angle(H))
plt.axis([0, 2 * np.pi, -np.pi, np.pi])
plt.grid(True)
plt.ylabel('Phase(rad)')
plt.show()
```

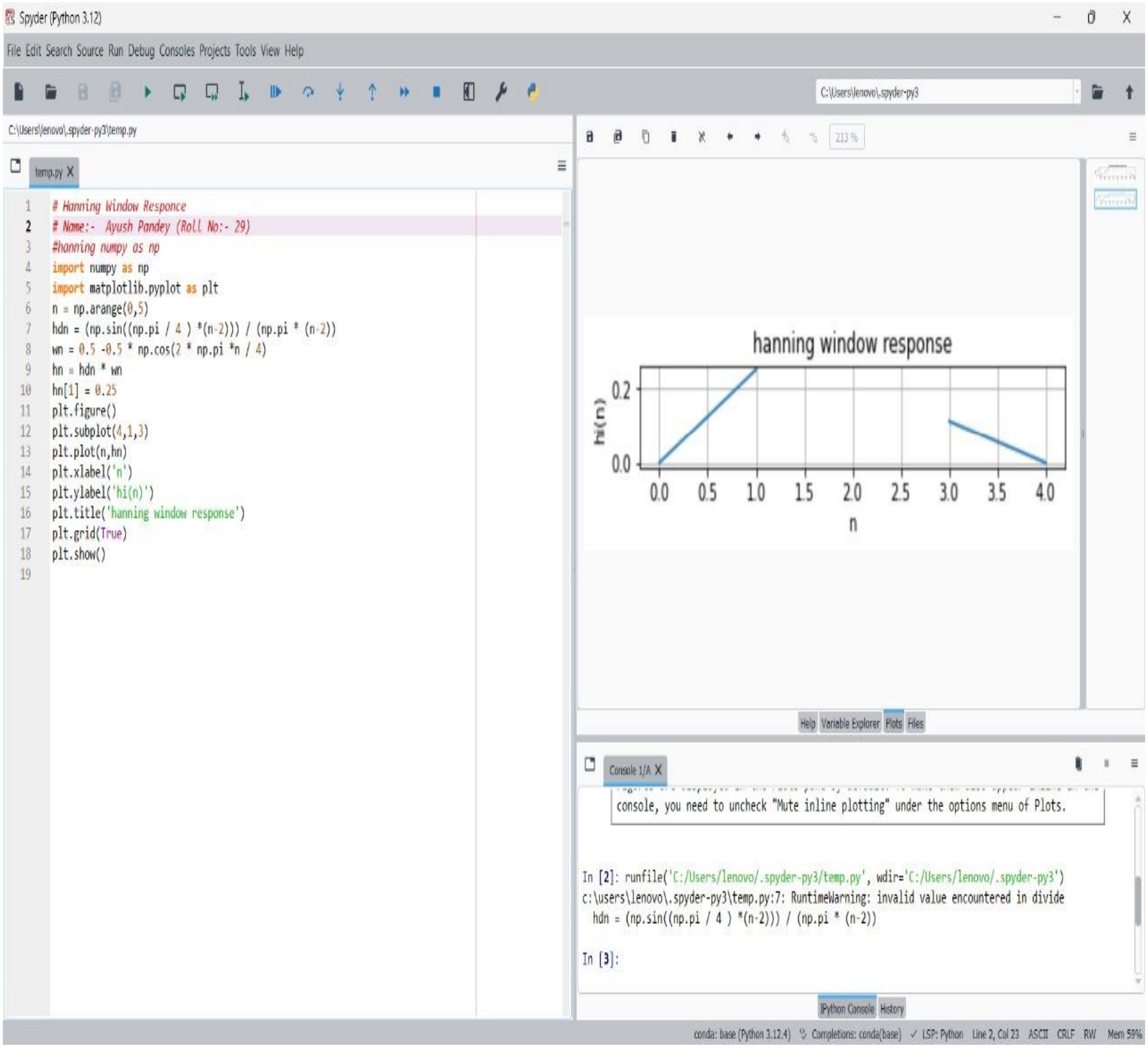


EXP-05

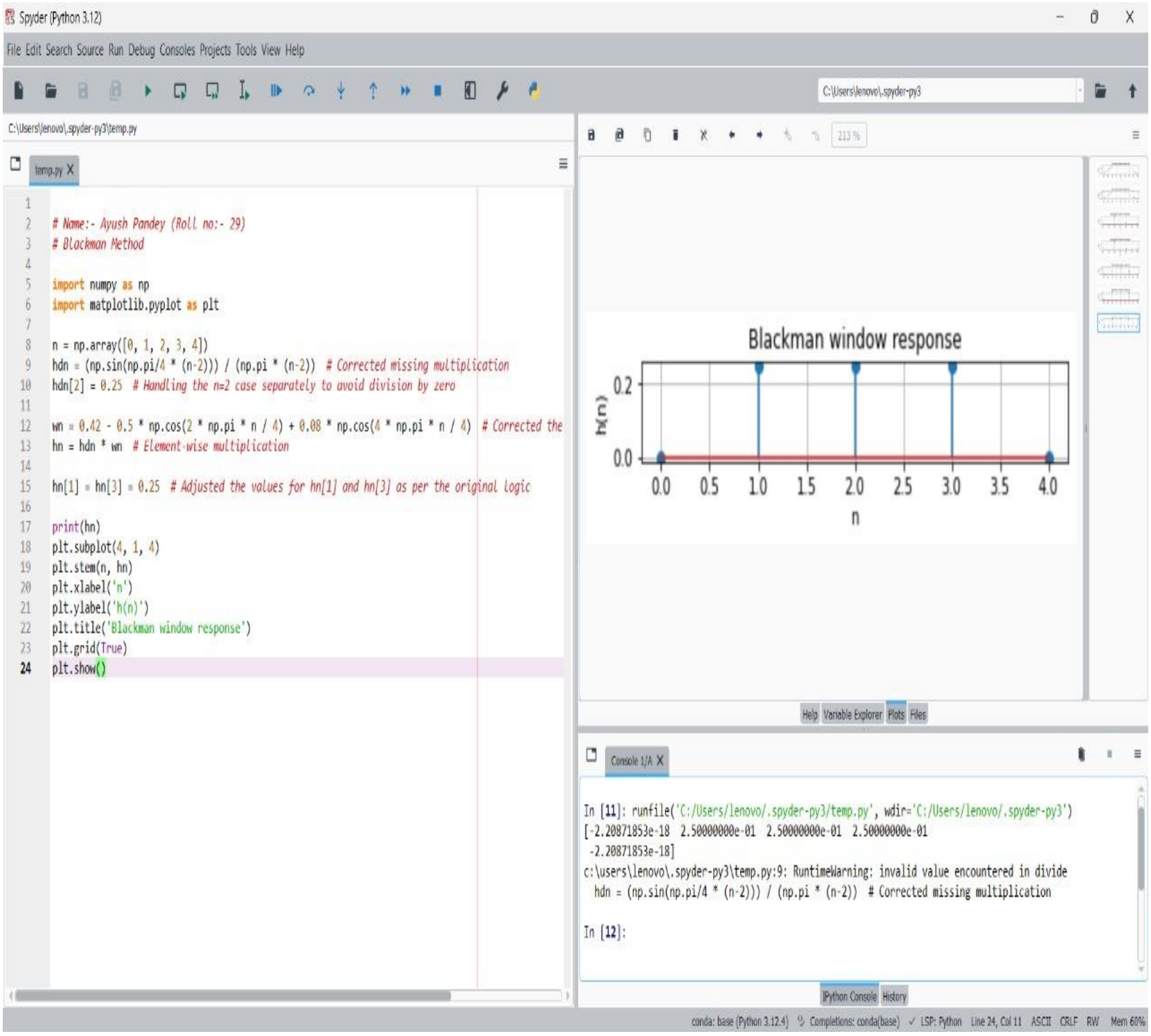
```
import numpy as np  #Rectangular window
import matplotlib.pyplot as plt
n= np.array([1,2,3,4])
hdn = (np.sin(np.pi/4) *(n-2)) / (np.pi * ( n-2))
wn = np.ones_like(n)
hn = np.array([1,3,0.25,0])
print (hn)
plt.subplot(4,1,1)
plt.stem(n,hn)
plt.xlabel('n')
plt.ylabel('h(n)')
plt.title('Rectangular  window response')
plt.grid(True)

plt.show()
```



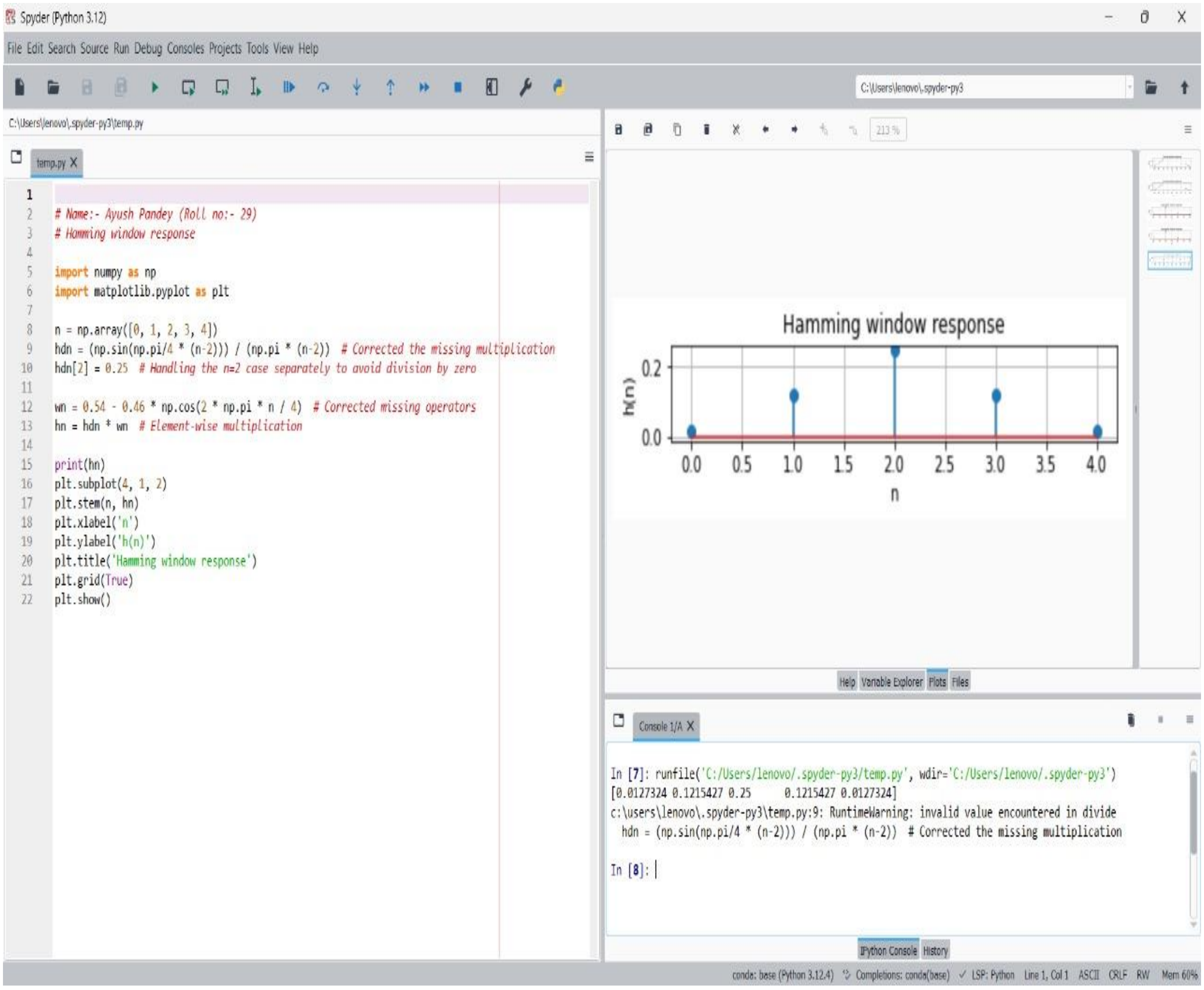
```
import numpy as np
import numpy as np  #Hanning window
import matplotlib.pyplot as plt
n = np.arange(0 , 5)
hdn = (np.sin((np.pi / 4) * (n - 2))) / (np.pi * (n - 2))
wn = 0.5 - 0.5 * np.cos(2 * np.pi * n / 4)
hn = hdn * wn
hn[1] = 0.25
plt.figure()
plt.subplot(4,1,3)
plt.plot(n, hn)
plt.xlabel('n')
plt.ylabel('hi(n)')
plt.title('Hanning Window Response')
plt.grid(True)
plt.show()
```



```
import numpy as np #Blackam window
import matplotlib.pyplot as plt
n = np.array([0,1,2,3,4])
hdn = (np.sin(np.pi / 4) * (n - 2)) / (np.pi * (n - 2))
hdn[2]=0.25
wn = 0.42-0.5*np.cos(2* np.pi*n/ 4)+0.08*np.cos(4*np.pi*n/4)
hn = hdn* wn
hn[1]=hn[3]=0.25

print (hn)
plt.subplot(4,1,2)
plt.stem(n,hn)
plt.xlabel('n')
plt.ylabel('h(n)')
plt.title('Hamming  window response')
plt.grid(True)
plt.show()
```



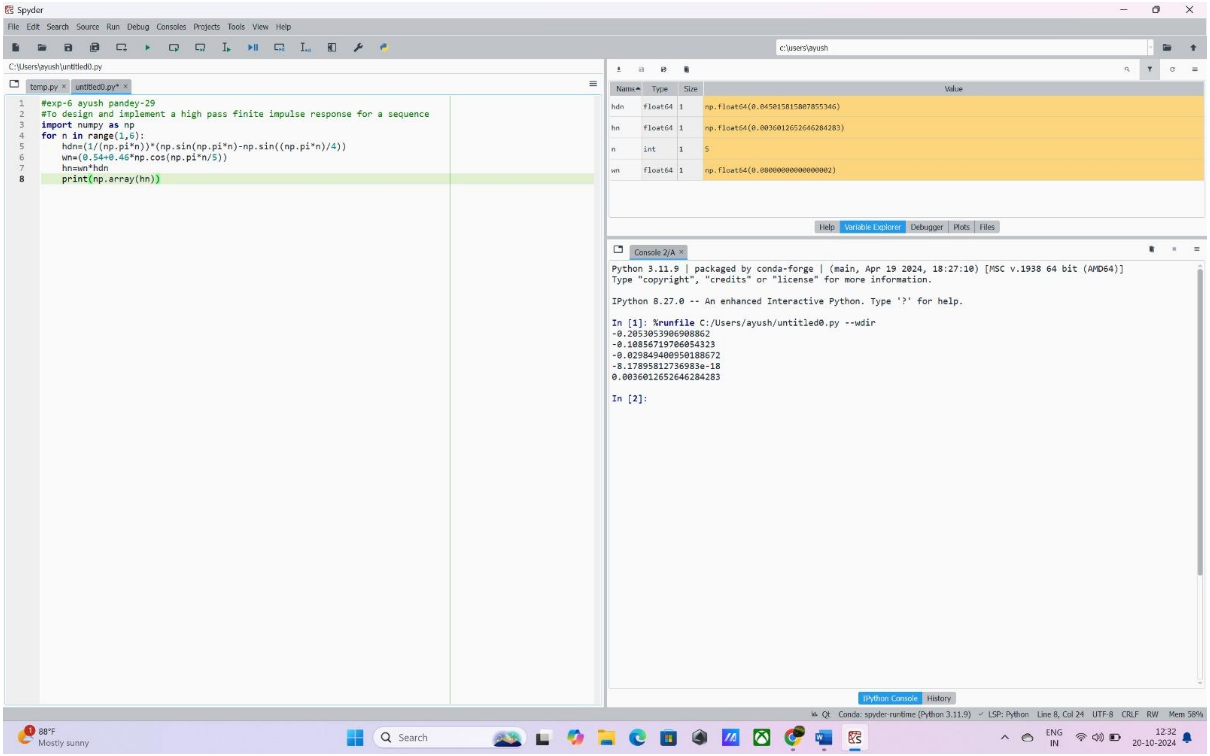
```
import numpy as np #Hamming Window
import matplotlib.pyplot as plt
n = np.array([0,1,2,3,4])
hdn = (np.sin(np.pi / 4) * (n - 2)) / (np.pi * (n - 2))
hdn[2]=0.25
wn = 0.54 -0.46*np.cos(2* np.pi*n/ 4)
hn = hdn* wn

print (hn)
plt.subplot(4,1,2)
plt.stem(n,hn)
plt.xlabel('n')
plt.ylabel('h(n)')
plt.title('Hamming  window response')
plt.grid(True)
plt.show()
```



Experiment no 6

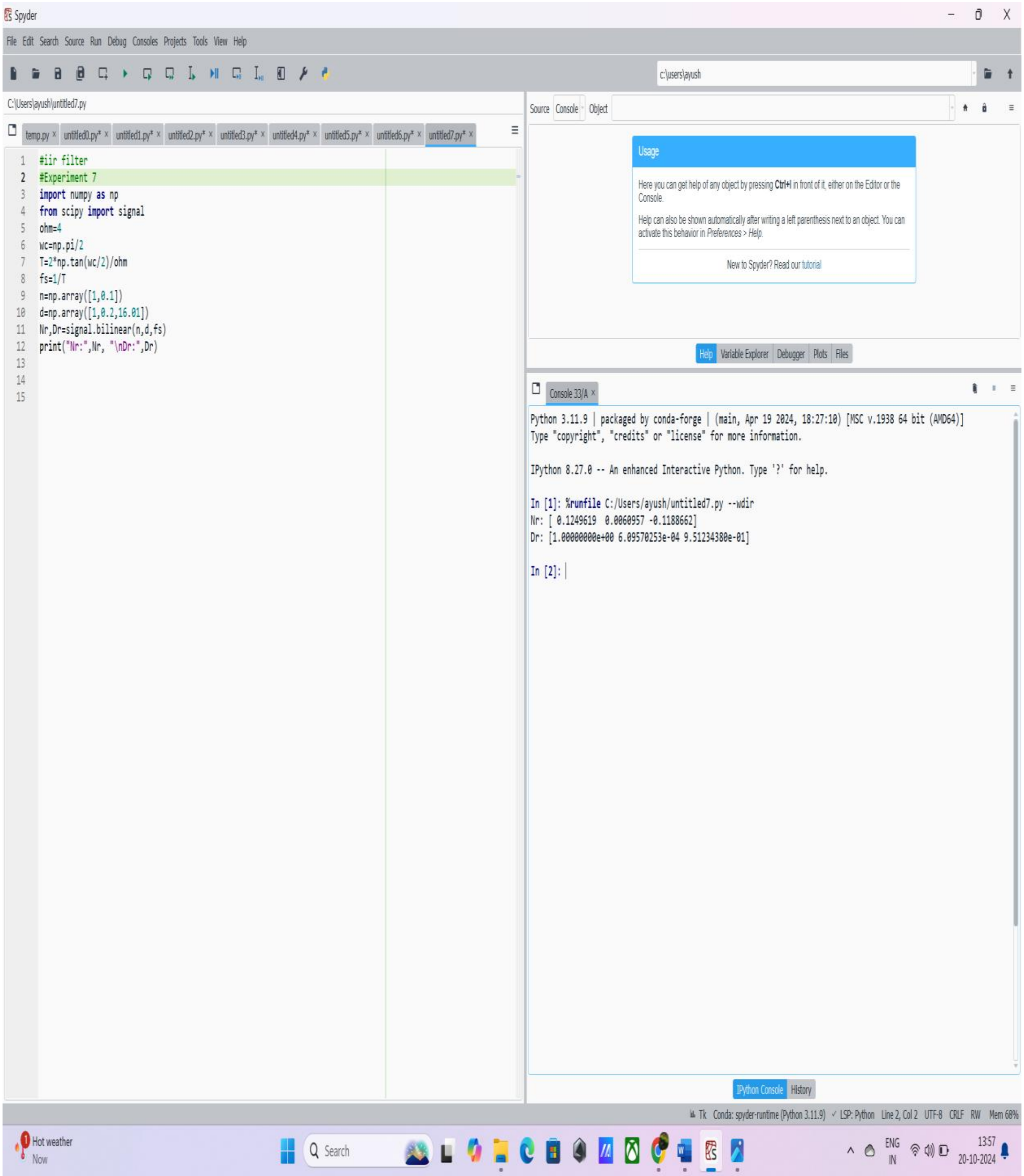
#To design and implement a high pass finite impulse response for a sequence

```
import numpy as np
for n in range(1,6):
   hdn=(1/(np.pi*n))*(np.sin(np.pi*n)-np.sin((np.pi*n)/4))
   wn=(0.54+0.46*np.cos(np.pi*n/5))
   hn=wn*hdn
   print(np.array(hn))
```



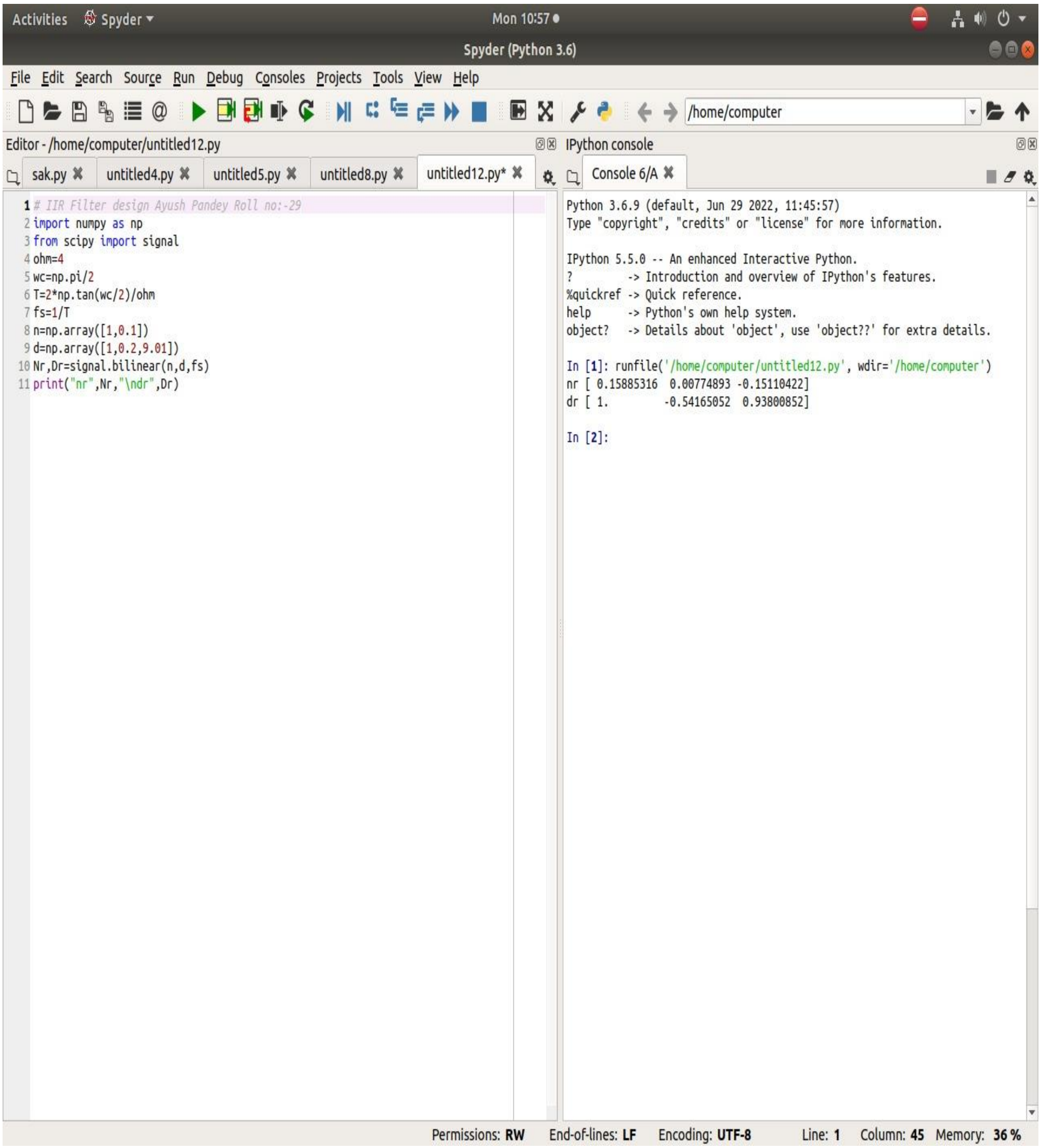
#IIR filter Experiment 7

```
import numpy as np
from scipy import signal
ohm=4
wc=np.pi/2
T=2*np.tan(wc/2)/ohm
fs=1/T
n=np.array([1,0.1])
d=np.array([1,0.2,16.01])
Nr,Dr=signal.bilinear(n,d,fs)
print("Nr:",Nr, "\nDr:",Dr)
```



#part 2

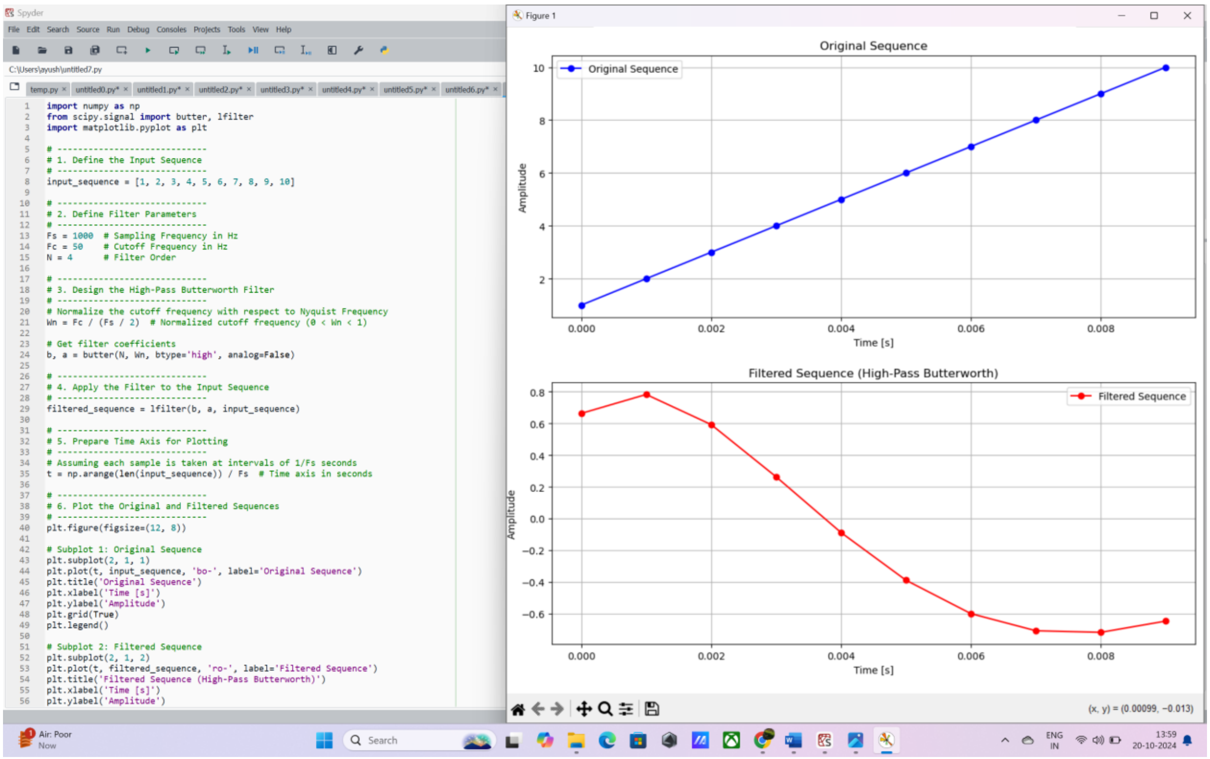
```
import numpy as np
from scipy import signal
ohm=4
wc=np.pi/2
T=2*np.tan(wc/2)/ohm
fs=1/T
n=np.array([1,0.1])
d=np.array([1,0.2,9.01])
Nr,Dr=signal.bilinear(n,d,fs)
print("Nr:",Nr, "\nDr:",Dr)
```



#Experiment 8 To design & Implemented High pass IIR filter

```
import numpy as np
from scipy.signal import butter, lfilter
import matplotlib.pyplot as plt

# -----------------------------
# 1. Define the Input Sequence
# -----------------------------
input_sequence = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]

# -----------------------------
# 2. Define Filter Parameters
# -----------------------------
Fs = 1000  # Sampling Frequency in Hz
Fc = 50    # Cutoff Frequency in Hz
N = 4      # Filter Order

# -----------------------------
# 3. Design the High-Pass Butterworth Filter
# -----------------------------
# Normalize the cutoff frequency with respect to Nyquist Frequency
Wn = Fc / (Fs / 2)  # Normalized cutoff frequency (0 < Wn < 1)

# Get filter coefficients
b, a = butter(N, Wn, btype='high', analog=False)

# -----------------------------
# 4. Apply the Filter to the Input Sequence
# -----------------------------
filtered_sequence = lfilter(b, a, input_sequence)

# -----------------------------
# 5. Prepare Time Axis for Plotting
# -----------------------------
# Assuming each sample is taken at intervals of 1/Fs seconds
t = np.arange(len(input_sequence)) / Fs  # Time axis in seconds

# -----------------------------
# 6. Plot the Original and Filtered Sequences
# -----------------------------
plt.figure(figsize=(12, 8))

# Subplot 1: Original Sequence
plt.subplot(2, 1, 1)
plt.plot(t, input_sequence, 'bo-', label='Original Sequence')
plt.title('Original Sequence')
plt.xlabel('Time [s]')
plt.ylabel('Amplitude')
```

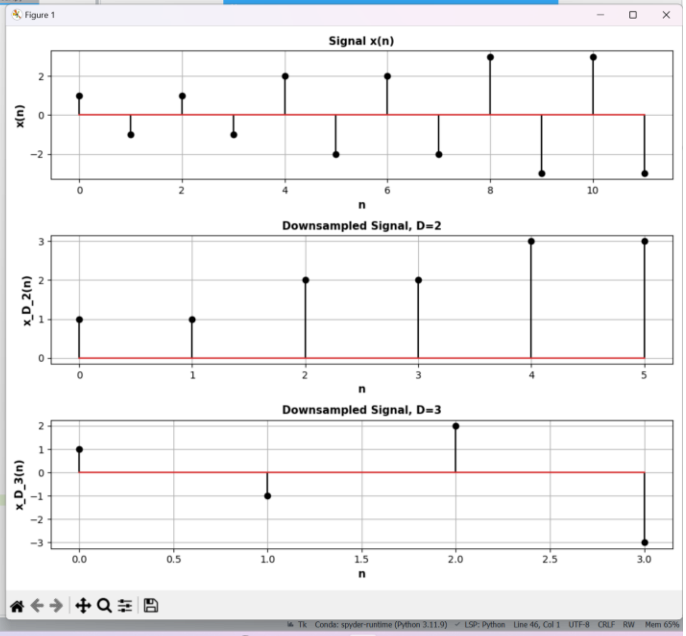
```
plt.grid(True)
plt.legend()

# Subplot 2: Filtered Sequence
plt.subplot(2, 1, 2)
plt.plot(t, filtered_sequence, 'ro-', label='Filtered Sequence')
plt.title('Filtered Sequence (High-Pass Butterworth)')
plt.xlabel('Time [s]')
plt.ylabel('Amplitude')
plt.grid(True)
plt.legend()

plt.tight_layout()
plt.show()
```



Experiment 9 Decimation Process

```
import numpy as np
import matplotlib.pyplot as plt

# Define the original signal
xn = np.array([1, -1, 1, -1, 2, -2, 2, -2, 3, -3, 3, -3])
N = len(xn)
n = np.arange(N)

# Downsample by factor of 2
xD2n = xn[::2]
n1 = np.arange(len(xD2n))

# Downsample by factor of 3
xD3n = xn[::3]
n2 = np.arange(len(xD3n))

# Plot the original signal
plt.figure(figsize=(10, 8))

plt.subplot(3, 1, 1)
plt.stem(n, xn, 'k')  # Original signal
plt.xlabel('n', fontsize=11, fontweight='bold')
plt.ylabel('x(n)', fontsize=11, fontweight='bold')
plt.title('Signal x(n)', fontsize=11, fontweight='bold')
plt.grid(True)

# Plot the downsampled signal (factor 2)
plt.subplot(3, 1, 2)
plt.stem(n1, xD2n, 'k')  # Downsampled by factor 2
plt.xlabel('n', fontsize=11, fontweight='bold')
plt.ylabel('x_D_2(n)', fontsize=11, fontweight='bold')
plt.title('Downsampled Signal, D=2', fontsize=11, fontweight='bold')
plt.grid(True)

# Plot the downsampled signal (factor 3)
plt.subplot(3, 1, 3)
plt.stem(n2, xD3n, 'k')  # Downsampled by factor 3
plt.xlabel('n', fontsize=11, fontweight='bold')
plt.ylabel('x_D_3(n)', fontsize=11, fontweight='bold')
plt.title('Downsampled Signal, D=3', fontsize=11, fontweight='bold')
plt.grid(True)

# Adjust layout and show the plot
plt.tight_layout()
plt.show()
```

```python
import numpy as np
import matplotlib.pyplot as plt

# Define the original signal
xn = np.array([1, -1, 1, -1, 2, -2, 2, -2, 3, -3, 3, -3])
N = len(xn)
n = np.arange(N)

# Downsample by factor of 2
xD2n = xn[::2]
n1 = np.arange(len(xD2n))

# Downsample by factor of 3
xD3n = xn[::3]
n2 = np.arange(len(xD3n))

# Plot the original signal
plt.figure(figsize=(10, 8))

plt.subplot(3, 1, 1)
plt.stem(n, xn, 'k')  # Original signal
plt.xlabel('n', fontsize=11, fontweight='bold')
plt.ylabel('x(n)', fontsize=11, fontweight='bold')
plt.title('Signal x(n)', fontsize=11, fontweight='bold')
plt.grid(True)

# Plot the downsampled signal (factor 2)
plt.subplot(3, 1, 2)
plt.stem(n1, xD2n, 'k')  # Downsampled by factor 2
plt.xlabel('n', fontsize=11, fontweight='bold')
plt.ylabel('x_D_2(n)', fontsize=11, fontweight='bold')
plt.title('Downsampled Signal, D=2', fontsize=11, fontweight='bold')
plt.grid(True)

# Plot the downsampled signal (factor 3)
plt.subplot(3, 1, 3)
plt.stem(n2, xD3n, 'k')  # Downsampled by factor 3
plt.xlabel('n', fontsize=11, fontweight='bold')
plt.ylabel('x_D_3(n)', fontsize=11, fontweight='bold')
plt.title('Downsampled Signal, D=3', fontsize=11, fontweight='bold')
plt.grid(True)

# Adjust layout and show the plot
plt.tight_layout()
plt.show()
```



Experiment 10 understanding how to read sound file