Heaven’s light is our guide

Rajshahi University of Engineering & Technology



**Department of Computer Science & Engineering**

**Assignment**

Course Name: Digital Signal Processing

Course Code: CSE 3209

|  |  |
| --- | --- |
| Submitted by  Name:Kapil Deb Barman  Roll:2003181  Section: C  Series: 20 | Submitted to  Md. Farukuzzaman Faruk  Assistant Professor,  Department of CSE,  RUET |

**1.Introduction:**

This assignment focuses on fundamental DSP concepts, including

Discrete Fourier Transform (DFT), Discrete-Time Fourier Transform (DTFT), and Filtering. You will implement and analyze these concepts using Python.

**2.Basic Signal Operations:**

**2.1 Signal Generation and Transformations**

**Code:**

fs = 100

f = 5

N = 100

k = 10

a = 2

n = np.arange(N)

x = np.sin(2 \* np.pi \* f \* n / fs)

x\_shifted = np.roll(x, k)

x\_reversed = x[::-1]

n\_scaled = np.arange(0, N, a)

x\_scaled = x[n\_scaled]

plt.figure(figsize=(12, 8))

plt.subplot(3, 1, 1)

plt.stem(n, x, linefmt='b-', markerfmt='bo', basefmt='k', label='Original x[n]')

plt.stem(n, x\_shifted, linefmt='r-', markerfmt='ro', basefmt='k', label=f'Shifted x[n-{k}]')

plt.xlabel('n')

plt.ylabel('Amplitude')

plt.title('Time Shifting')

plt.legend()

plt.grid()

plt.subplot(3, 1, 2)

plt.stem(n, x, linefmt='b-', markerfmt='bo', basefmt='k', label='Original x[n]')

plt.stem(n, x\_reversed, linefmt='g-', markerfmt='go', basefmt='k', label='Reversed x[-n]')

plt.xlabel('n')

plt.ylabel('Amplitude')

plt.title('Time Reversal')

plt.legend()

plt.grid()

plt.subplot(3, 1, 3)

plt.stem(n, x, linefmt='b-', markerfmt='bo', basefmt='k', label='Original x[n]')

plt.stem(n\_scaled, x\_scaled, linefmt='m-', markerfmt='mo', basefmt='k', label=f'Scaled x[{a}n]')

plt.xlabel('n')

plt.ylabel('Amplitude')

plt.title('Time Scaling')

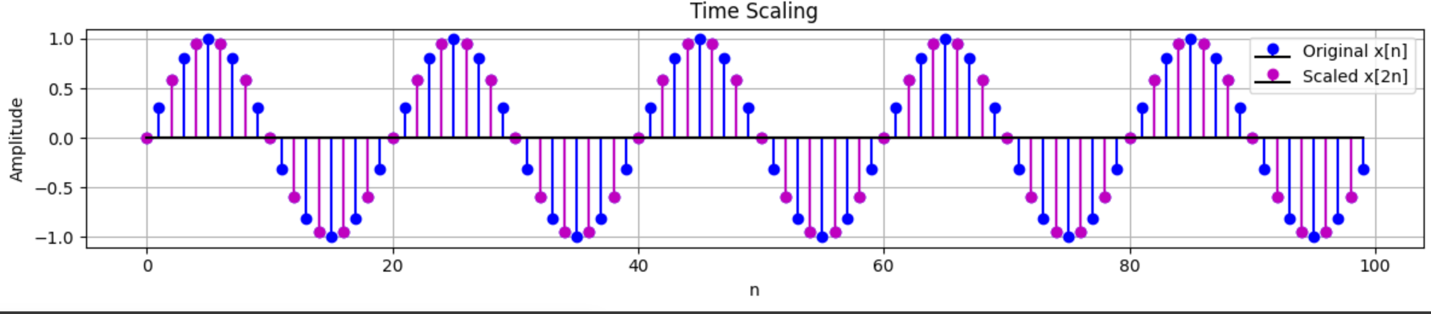
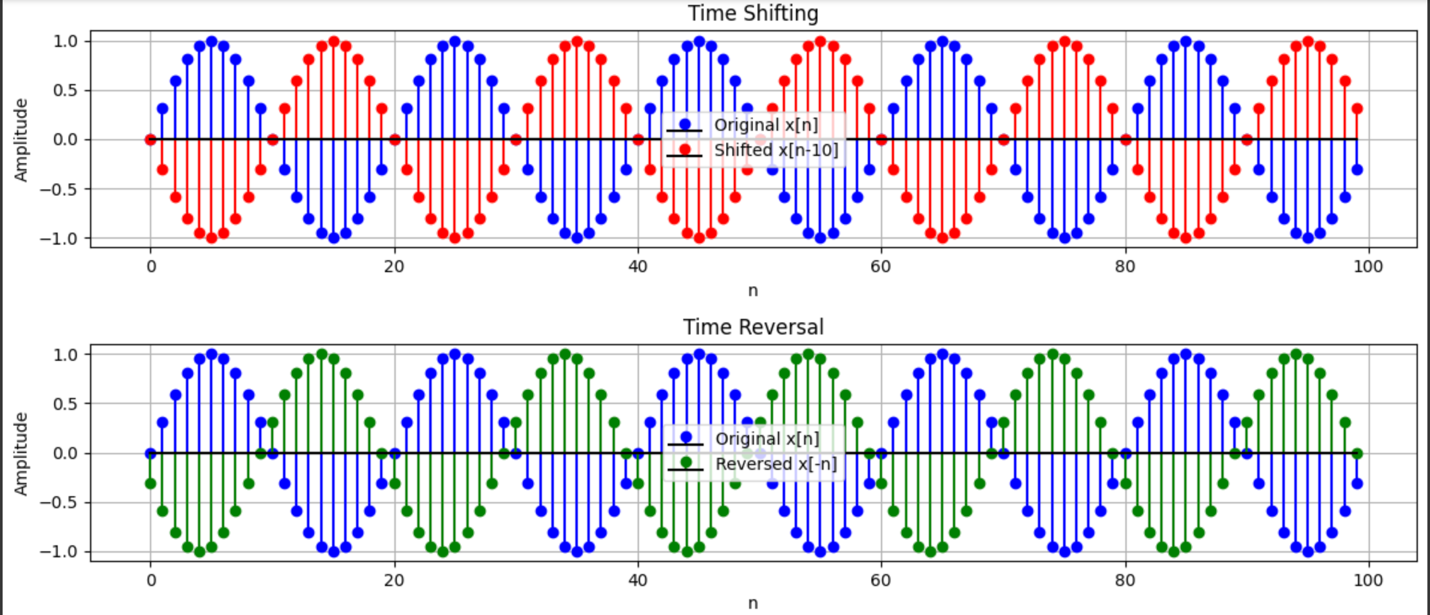
plt.legend()

plt.grid()

plt.tight\_layout()

plt.show()

**Result:**



**3.Discrete Fourier Transform (DFT)**

**Code:**

def DFT(x):

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)

return X

X\_manual = DFT(x)

X\_numpy = np.fft.fft(x)

plt.figure(figsize=(10, 4))

#plt.subplot(3, 1, 1)

plt.plot(np.abs(X\_manual), 'bo-', label='Manual DFT')

plt.plot(np.abs(X\_numpy), 'r--', label='Numpy FFT')

plt.xlabel('Frequency Index')

plt.ylabel('Magnitude')

plt.title('DFT vs Numpy FFT')

plt.legend()

plt.grid()

plt.show()

**Result:**

A graph with numbers and a line

AI-generated content may be incorrect.

**4.Discrete-Time Fourier Transform (DTFT)**

**Code:**

omega = np.linspace(-np.pi, np.pi, 1000)

X\_dtft = np.array([np.sum(x \* np.exp(-1j \* w \* n)) for w in omega])

plt.figure(figsize=(10, 4))

plt.plot(omega, np.abs(X\_dtft), 'g-', label='DTFT')

plt.xlabel('Frequency (rad/sample)')

plt.ylabel('Magnitude')

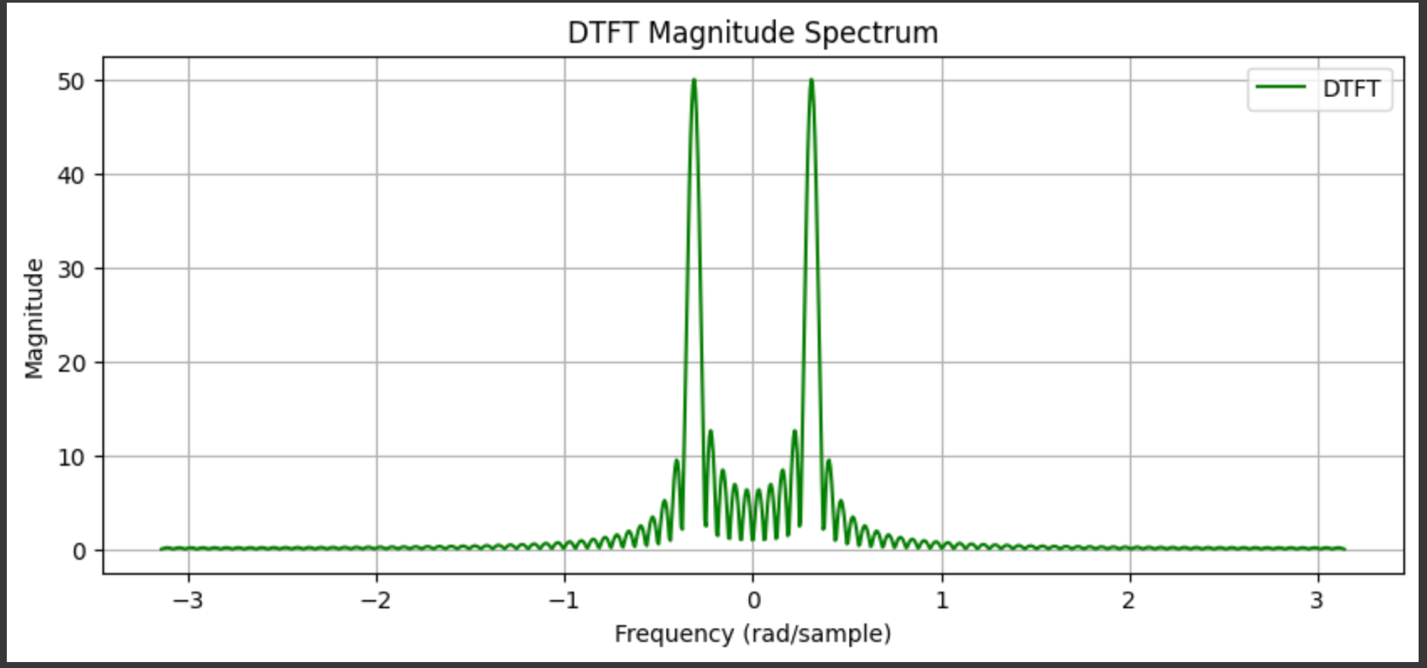
plt.title('DTFT Magnitude Spectrum')

plt.legend()

plt.grid()

plt.show()

**Result:**



**5.Fast Fourier Transform (FFT) Efficiency**

**Code:**

sizes = [64, 128, 256, 512, 1024]

dft\_times = []

fft\_times = []

for N in sizes:

x = np.random.rand(N)

start\_time = time.time()

X\_manual = DFT(x)

dft\_times.append(time.time() - start\_time)

start\_time = time.time()

X\_numpy = np.fft.fft(x)

fft\_times.append(time.time() - start\_time)

plt.figure(figsize=(8, 6))

plt.plot(sizes, dft\_times, label='DFT ', marker='o', linestyle='--', color='r')

plt.plot(sizes, fft\_times, label='FFT ', marker='s', linestyle='-', color='b')

plt.yscale('log')

plt.xlabel('Input Size (N)')

plt.ylabel('Execution Time (seconds)')

plt.title('DFT vs FFT Execution Time')

plt.legend()

plt.grid()

plt.show()

**Result:**

A graph with a line and a red dotted line

AI-generated content may be incorrect.

**6.Filtering in the Frequency Domain**

**Code:**

f\_c = 10 #cutoff

def low\_pass\_filter(X, fs, cutoff\_freq):

N = len(X)

freqs = np.fft.fftfreq(N, 1/fs)

filter\_mask = np.abs(freqs) <= cutoff\_freq

X\_filtered = X \* filter\_mask

return X\_filtered

X = DFT(x)

X\_filtered = low\_pass\_filter(X, fs, f\_c)

def IDFT(X\_filtered):

N = len(X\_filtered)

x\_filtered = np.zeros(N, dtype=complex)

for n in range(N):

for k in range(N):

x\_filtered[n] += X\_filtered[k] \* np.exp(2j \* np.pi \* k \* n / N)

return np.real(x\_filtered)

x\_filtered = IDFT(X\_filtered)

plt.figure(figsize=(12, 8))

plt.subplot(3, 1, 1)

plt.stem(n, x, linefmt='b-', markerfmt='bo', basefmt='k', label='Original x[n]')

plt.xlabel('n')

plt.ylabel('Amplitude')

plt.title('Original Signal')

plt.legend()

plt.grid()

plt.subplot(3, 1, 2)

plt.stem(n, x\_filtered, linefmt='r-', markerfmt='ro', basefmt='k', label='Filtered x[n]')

plt.xlabel('n')

plt.ylabel('Amplitude')

plt.title('Low-pass Filtered Signal')

plt.legend()

plt.grid()

plt.tight\_layout()

plt.show()

**Result:**

