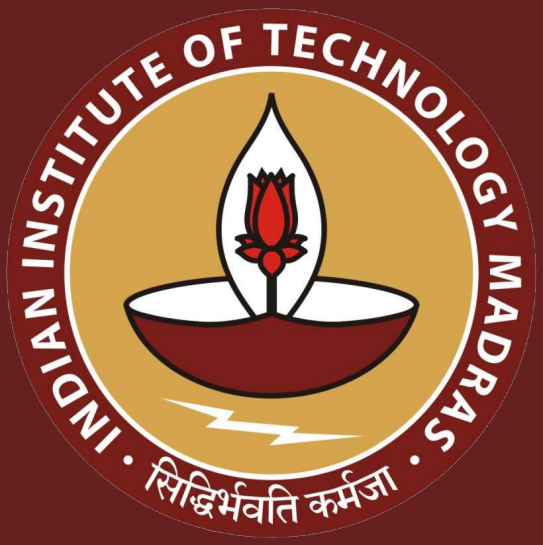


Assignment II

Kaustabh Ganguly

Roll : ch23m514

Time Series Analysis

1. **Answer the following:**

(a) Determine and sketch the magnitude and phase spectra of the following periodic signals:
(i) $x[n] = 4\sin(\frac{\pi(n-2)}{3}n)$, (ii) $x[n] = \cos(\frac{2\pi}{3}n) + \sin(\frac{2\pi}{5}n)$ and (iii) $x[n] = \cos(\frac{2\pi}{3}n)\sin(\frac{2\pi}{5}n)$

(b) Determine the periodic signal $x[n]$ with period $N = 8$ if its Fourier coefficients are given by $c_k = \cos(\frac{\pi k}{4}) + \sin(\frac{3\pi k}{4})$.

1) a) Discrete time signals are given. we need to find out discrete fourier transform (DFT) to analyse the discrete time signals.

$$X[k] = \sum_{n=0}^{N-1} x[n] e^{-j2\pi kn/N}$$

$X[k] \rightarrow$ DFT of $x[n]$
N is number of sample
k is frequency index from 0 to N-1

Magnitude:

$$|X[k]| = \sqrt{Re^2(X[k]) + Im^2(X[k])}.$$

Phase:

$$\angle X[k] = \tan^{-1}\left(\frac{Im(X[k])}{Re(X[k])}\right).$$

As these signal is infinite in length we assume N=100

Using DFT directly is computationally expensive, it involves $O(N^2)$ operations.

We will use Fast Fourier transform (FFT) which computes in $O(N \log N)$. I will use numpy library's fft. (The most well known version of FFT is the Radix-2 – Cooley Tukey Algorithm, which recursively breaks $N = N_1 N_2$ into smaller DFT of N_1 & N_2 size.)

i) $x[n] = 4 \sin\left(\frac{\pi(n-2)}{3} n\right)$

ii) $x[n] = \cos\left(\frac{2\pi}{3} n\right) + \sin\left(\frac{2\pi}{3} n\right)$

iii) $x[n] = \cos\left(\frac{2\pi n}{3}\right) \sin\left(\frac{2\pi n}{3}\right)$

3s

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

We will first compute DFT using FFT

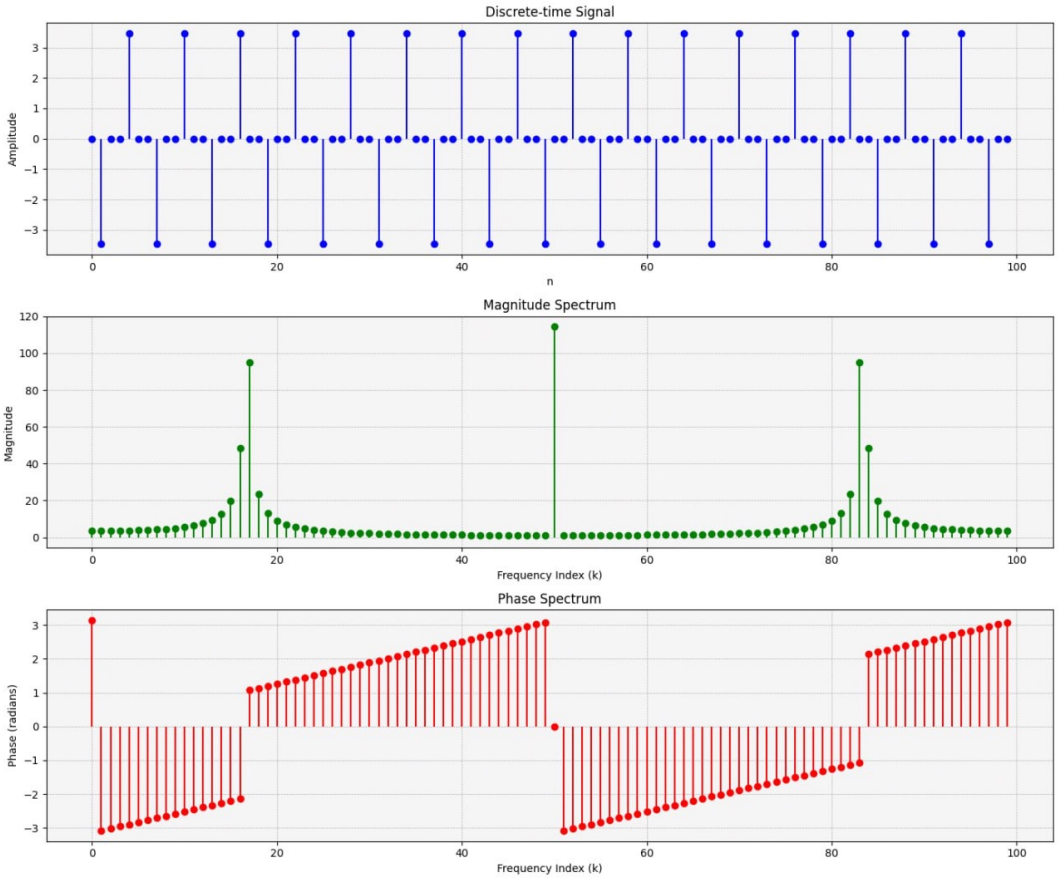
```python
def compute_dft(signal):
    """
    Compute the Discrete Fourier Transform (DFT) of the input signal.

    Parameters:
    - signal: numpy array, the input discrete-time signal

    Returns:
    - magnitude_spectrum: Magnitude spectrum of the signal
    - phase_spectrum: Phase spectrum of the signal
    """
    # Compute the DFT of the signal
    spectrum = np.fft.fft(signal)

    # Extract magnitude and phase spectra
    magnitude_spectrum = np.abs(spectrum)
    phase_spectrum = np.angle(spectrum)

    return magnitude_spectrum, phase_spectrum
```

```
def plot_signal_and_spectra(n, signal, magnitude_spectrum, phase_spectrum, title_prefix):
    """
    Plot the input signal, its magnitude spectrum, and phase spectrum

    Parameters:
    - n: numpy array, time indices for the discrete-time signal
    - signal: numpy array, the input discrete-time signal
    - magnitude_spectrum: Magnitude spectrum of the signal
    - phase_spectrum: Phase spectrum of the signal
    - title_prefix: str, prefix for the title of the plots
    """
    fig, ax = plt.subplots(3, 1, figsize=(14, 12))
    fig.suptitle(title_prefix, fontsize=16, y=1.05)

    # Plot the discrete-time signal
    ax[0].stem(n, signal, basefmt=" ", linefmt='-b', markerfmt='ob')
    ax[0].set_title("Discrete-time Signal")
    ax[0].set_xlabel("n")
    ax[0].set_ylabel("Amplitude")
    ax[0].grid(True, which='both', linestyle='--', linewidth=0.5)
    ax[0].set_facecolor('#f5f5f5')

    # Plot the magnitude spectrum
    ax[1].stem(n, magnitude_spectrum, basefmt=" ", linefmt='-g', markerfmt='og')
    ax[1].set_title("Magnitude Spectrum")
    ax[1].set_xlabel("Frequency Index (k)")
    ax[1].set_ylabel("Magnitude")
    ax[1].grid(True, which='both', linestyle='--', linewidth=0.5)
    ax[1].set_facecolor('#f5f5f5')

    # Plot the phase spectrum
    ax[2].stem(n, phase_spectrum, basefmt=" ", linefmt='-r', markerfmt='or')
    ax[2].set_title("Phase Spectrum")
    ax[2].set_xlabel("Frequency Index (k)")
    ax[2].set_ylabel("Phase (radians)")
    ax[2].grid(True, which='both', linestyle='--', linewidth=0.5)
    ax[2].set_facecolor('#f5f5f5')

    # Adjust layout
    plt.tight_layout()
```

```
# Generate signals and their spectra and then plot

# Define the length of the signal and time indices
N = 100
n_values = np.arange(N)

# Function 1: x[n] = 4 * sin(pi * (n - 2) * n / 3)
signal_1 = 4 * np.sin(np.pi * (n_values - 2) * n_values / 3)
mag_1, phase_1 = compute_dft(signal_1)
plot_signal_and_spectra(n_values, signal_1, mag_1, phase_1, title_prefix="Function 1:")

# Function 2: x[n] = cos(2*pi*n/3) + sin(2*pi*n/5)
signal_2 = np.cos(2 * np.pi * n_values / 3) + np.sin(2 * np.pi * n_values / 5)
mag_2, phase_2 = compute_dft(signal_2)
plot_signal_and_spectra(n_values, signal_2, mag_2, phase_2, title_prefix="Function 2:")

# Function 3: x[n] = cos(2*pi*n/3) * sin(2*pi*n/5)
signal_3 = np.cos(2 * np.pi * n_values / 3) * np.sin(2 * np.pi * n_values / 5)
mag_3, phase_3 = compute_dft(signal_3)
plot_signal_and_spectra(n_values, signal_3, mag_3, phase_3, title_prefix="Function 3:")

plt.show()
```

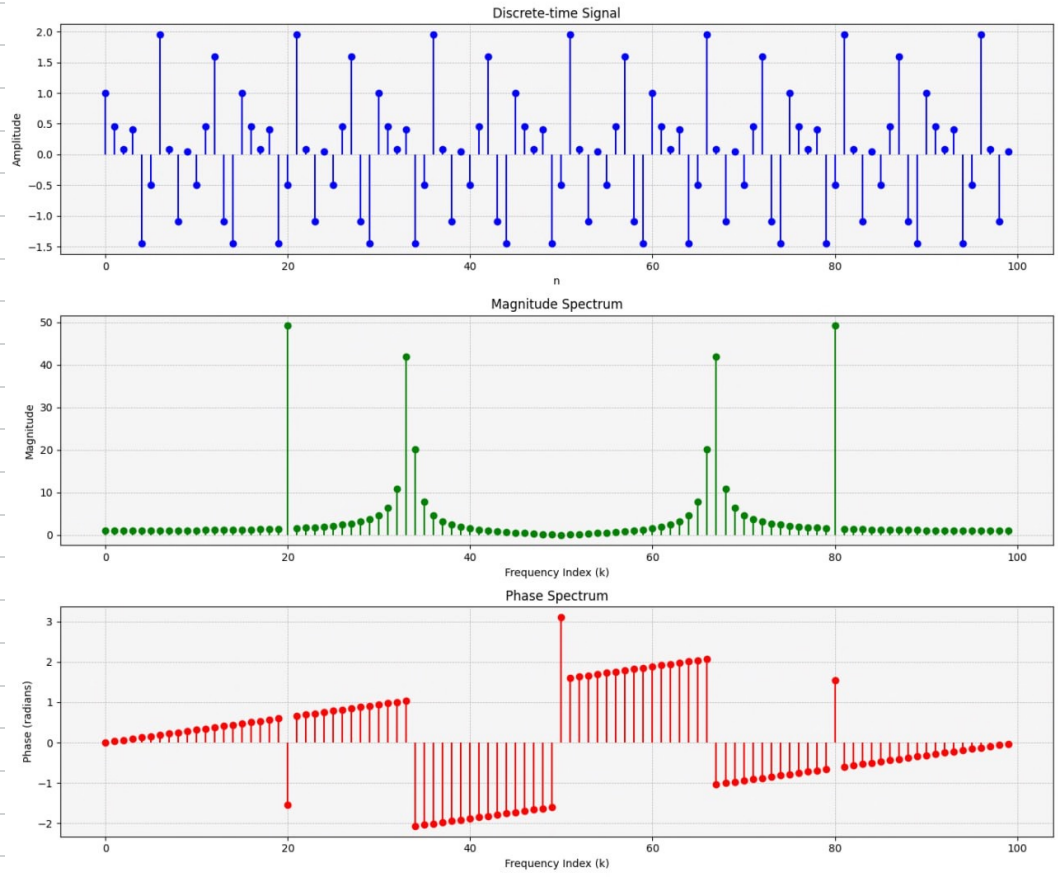
$$x[n] = 4\sin\left(\frac{\pi(n-2)}{3}n\right)$$

Function 1:

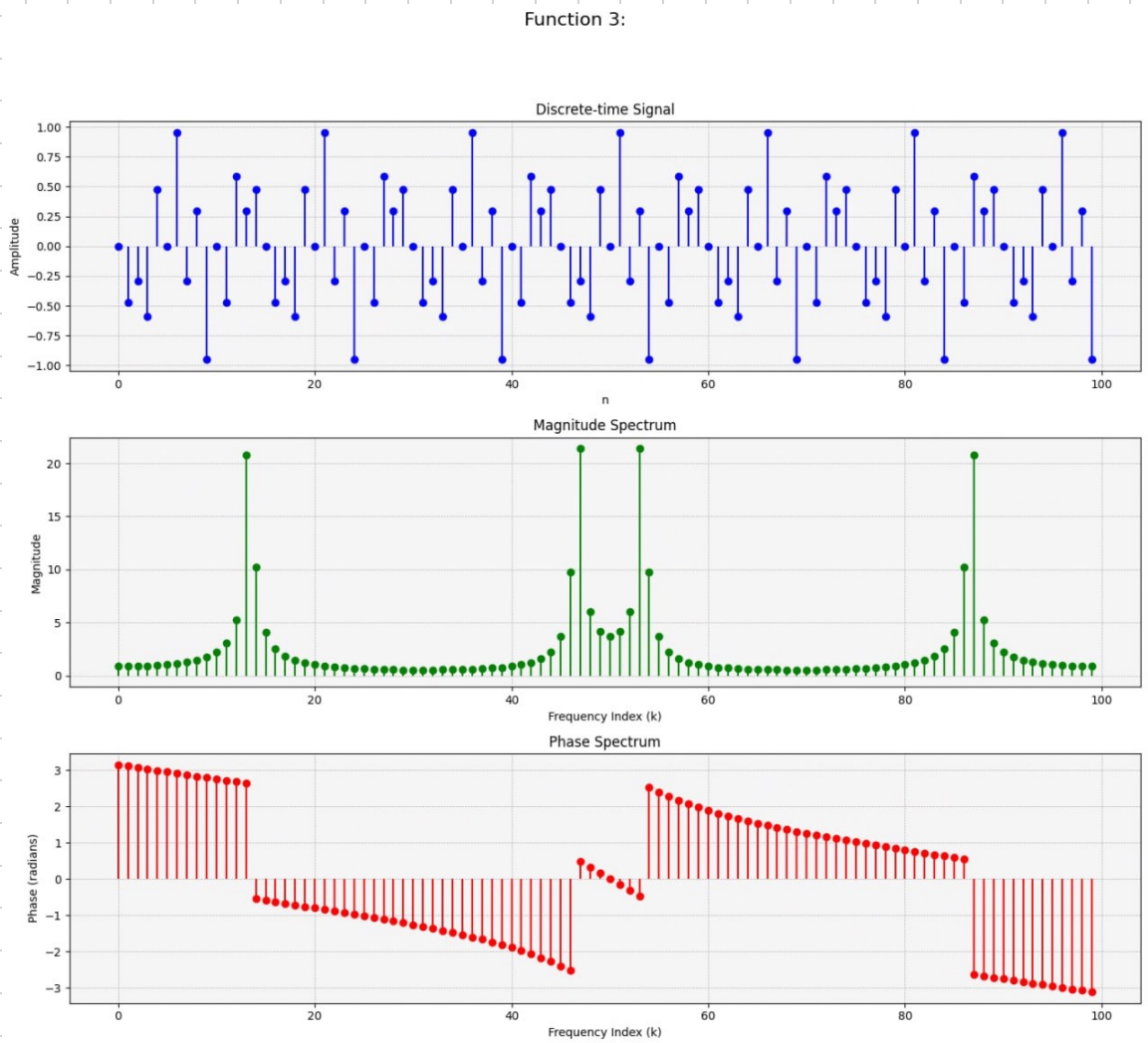


$$x[n] = \cos\frac{2\pi}{3}n + \sin\frac{2\pi}{5}n$$

Function 2:



$$x[n] = \cos\left(\frac{2\pi}{3}n\right)\sin\left(\frac{2\pi}{5}n\right)$$



b) $C_k = \cos\frac{\pi k}{4} + \sin\frac{3\pi k}{4}$; $N = 8$ $x[n] = ?$

We have to employ inverse discrete fourier transform.

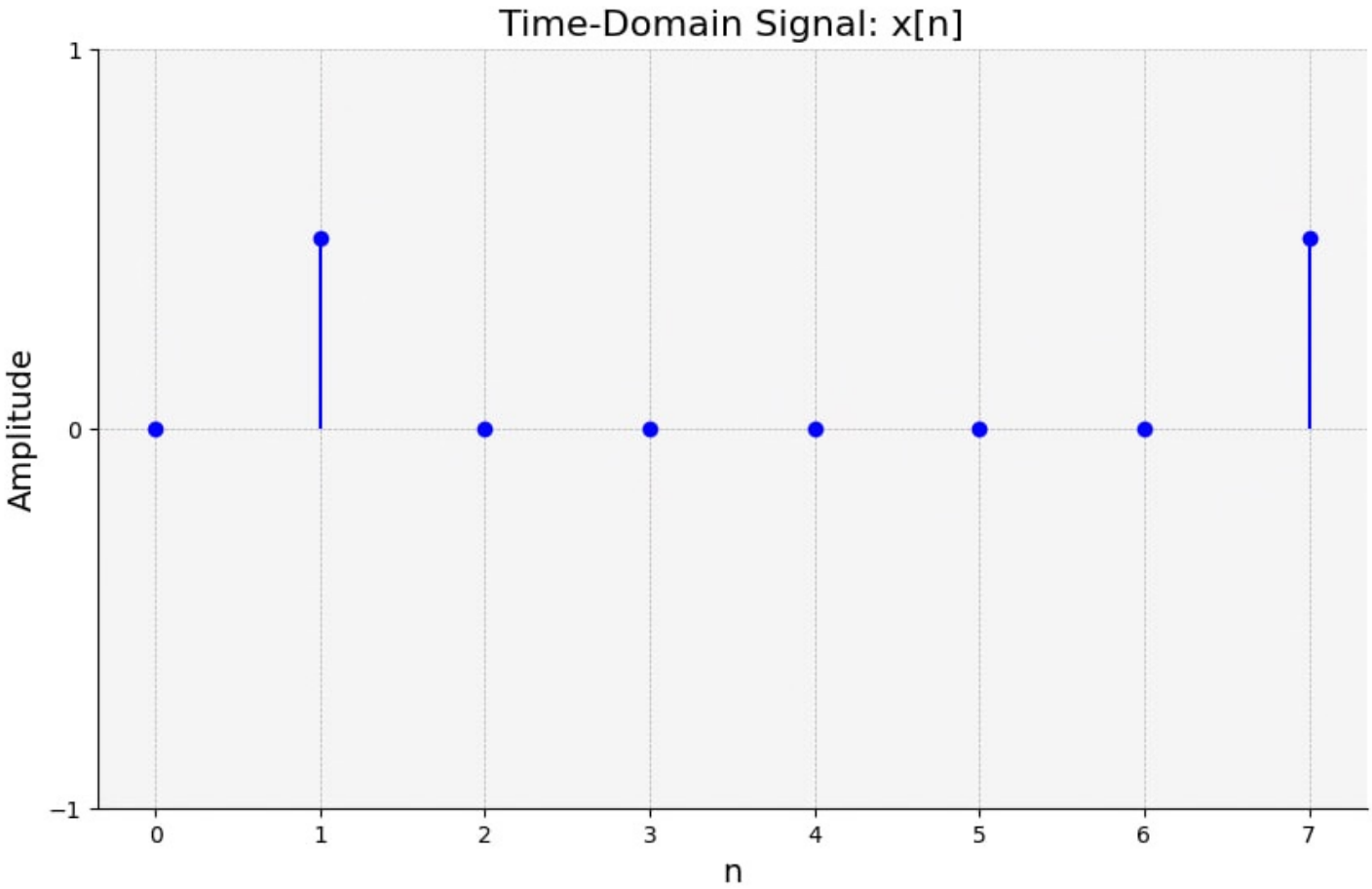
$$x[n] = \frac{1}{N}\sum_{k=0}^{N-1} C_k\, e^{j\left(\frac{2\pi}{N}\right)kn}$$

$x[n]$ is time domain signal
C_k are fourier coefficients
N is period of signal
n is time index 0 to N-1
k is frequency index 0 to N-1

```python
# Importing necessary libraries
import numpy as np
import matplotlib.pyplot as plt

# Define the Fourier coefficients c_k
def compute_fourier_coefficients(k_values):
    """
    Compute the Fourier coefficients c_k for given k values.

    Parameters:
    - k_values: numpy array, frequency indices

    Returns:
    - Fourier coefficients c_k
    """
    return np.cos(np.pi * k_values / 4) + np.sin(3 * np.pi * k_values / 4)
```

```python
# Compute the signal x[n] using IDFT
def compute_idft(c_k, N):
    """
    Compute the Inverse Discrete Fourier Transform (IDFT) for given Fourier coefficients.

    Parameters:
    - c_k: numpy array, Fourier coefficients
    - N: int, period of the signal

    Returns:
    - Time-domain signal x[n]
    """
    x_n = np.zeros(N, dtype=complex)
    for n in range(N):
        for k in range(N):
            x_n[n] += c_k[k] * np.exp(1j * (2 * np.pi / N) * k * n)
    x_n /= N
    return np.real(x_n)
```

```python
# Visualization
def plot_signal(x_n):
    """
    Plot the time-domain signal x[n]

    Parameters:
    - x_n: numpy array, time-domain signal
    """
    plt.figure(figsize=(10, 6))

    # Plot x[n]
    plt.stem(x_n, basefmt=" ", linefmt='-b', markerfmt='ob')
    plt.title("Time-Domain Signal: x[n]", fontsize=16)
    plt.xlabel("n", fontsize=14)
    plt.ylabel("Amplitude", fontsize=14)
    plt.grid(True, which='both', linestyle='--', linewidth=0.5)
    plt.xticks(np.arange(len(x_n)))
    plt.yticks(np.arange(int(min(x_n))-1, int(max(x_n))+2))
    plt.gca().set_facecolor('#f5f5f5')
    plt.gca().spines['top'].set_visible(False)
    plt.gca().spines['right'].set_visible(False)

    plt.show()

# Given values and computations
N = 8
k_values = np.arange(N)
c_k = compute_fourier_coefficients(k_values)
x_n = compute_idft(c_k, N)

# Plot the signal
plot_signal(x_n)
```



$$x[n] = \frac{1}{8}\sum_{k=0}^{7}\left\{\cos\frac{\pi k}{4} + \sin\frac{3\pi k}{4}\right\}e^{j\frac{\pi}{4}kn}$$

---

2. Answer the following:

(a) A signal $x[n]$ has the following Fourier Transform: $X(\omega) = \frac{1}{1-ae^{-j\omega}}$. Determine the Fourier Transform of the following signals:
(i) $x[2n+1]$ (ii) $e^{\pi n/2}x[n+2]$ (iii) $x[n]\cos[0.3\pi n]$ and (iv) $x[n] \star x[n-1]$

(b) Consider the periodic signal $x[n] = 1, 0, 1, 2, 3, 2$ starting from $n = 0$. Verify Parseval's theorem for this case.

a) DTFT is :

$$X(\omega) = \sum_{n=-\infty}^{\infty} x[n]e^{-j\omega n}$$

i) $x[2n+1]$ has 2 operation

Time scaling $x[2n]$

Time shifting $x[n+1]$

DTFT of time scaled signal $x[an]$ is $X\left(\frac{\omega}{a}\right)$

DTFT " " shifted " $x[n+m]$ is $X(\omega)e^{j\omega m}$

so, $X_1(\omega) = X\left(\frac{\omega}{2}\right)e^{j\omega}$ $\quad \{x[-n] \to X(-\omega)\}$

$$X(\omega) = \frac{1}{1-ae^{-j\omega}}$$

$$X\left(\frac{\omega}{2}\right) = \frac{1}{1-ae^{-\frac{j\omega}{2}}}$$

$$X_1(\omega) = \frac{e^{j\omega}}{1-ae^{-\frac{j\omega}{2}}} \cdot \text{(Ans)}$$

ii) $e^{\frac{\pi n}{2}}x[n+2]$

Time shift: $x[n+2]$

Multiply by exponential: $e^{\frac{\pi n}{2}}$ [A real term means growth or decay and not oscillation]

$x[n+2] \to X(\omega)e^{2j\omega}$

Multiplication in time domain means convolution in frequency domain

A signal $x[n]$ multiplied by $e^{j\omega_0 n}$ in time domain, it's frequency domain is:

$$X(\omega - \omega_0)$$

but here $e^{\frac{\pi}{2}n}$ is missing 'j'.

$$e^{\frac{\pi}{2}n} = \cos\frac{\pi}{2}n + j\sin\frac{\pi}{2}n$$

$$e^{\frac{\pi}{2}n}u[n+2] = \cos\left(\frac{\pi}{2}n\right)u[n+2] + j\sin\left(\frac{\pi}{2}n\right)u[n+2]$$

fourier transform $X_2(\omega)$

$$\cos\left(\frac{\pi}{2}n\right)u[n+2] + j\sin\left(\frac{\pi}{2}n\right)u[n+2]$$

$$= 0 + jX(\omega)e^{j2\omega}$$

$$X_2(\omega) = \frac{je^{2j\omega}}{1 - ae^{-j\omega}}.$$

```python
from sympy import symbols, cos, sin, pi, exp, simplify

# Define the symbols
omega, a = symbols('omega a', real=True, positive=True)
j = symbols('j', imaginary=True)

# Given Fourier transform X(omega)
def X_omega_expression(omega, a):
    return 1 / (1 - a * exp(-j * omega))

# Compute the Fourier transform for the term cos(pi*n/2) x[n + 2]
def compute_X_cos(omega, a):
    X_omg = X_omega_expression(omega, a)
    return X_omg * cos(pi/2) * exp(j * 2 * omega)

# Compute the Fourier transform for the term j*sin(pi*n/2) x[n + 2]
def compute_X_sin(omega, a):
    X_omg = X_omega_expression(omega, a)
    return X_omg * sin(pi/2) * exp(j * 2 * omega)

# Sum the two results to get the Fourier transform for e^(pi*n/2) x[n + 2]
def compute_X2_alternative(omega, a):
    X_cos_res = compute_X_cos(omega, a)
    X_sin_res = compute_X_sin(omega, a)
    return simplify(X_cos_res + j * X_sin_res)

# Display the result
result = compute_X2_alternative(omega, a)
print(result)
```

```
-j*exp(3*j*omega)/(a - exp(j*omega))
```

$$X_2(\omega) = \frac{-je^{3j\omega}}{a - e^{j\omega}}.$$ (Ans) Validated.

Assuming it's a typo $e^{j\frac{\pi}{2}n}$ 'j' is there

$$X(\omega - \omega_0) \quad \omega_0 = \frac{\pi}{2}$$

$$X_2(\omega) = X(\omega - \pi/2)e^{2j\omega}$$

$$= \frac{e^{2j\omega}}{1 - ae^{-j(\omega - \pi/2)}}$$

iii) $x[n]\cos[0.3\pi n]$

A signal $x[n]$ when multiplied by a sinusoidal signal $\cos(\omega_0 n)$ in the time domain, this operation corresponds to a modulation or frequency mixing in the frequency domain. (amp modulation)

$$x[n]\cos(\omega_0 n) = \frac{1}{2}\left(X(\omega - \omega_0) + X(\omega + \omega_0)\right)$$

$$X_3(\omega) = \frac{1}{2}\left(X(\omega - 0.3\pi) + X(\omega + 0.3\pi)\right)$$

$$X_3(\omega) = -\frac{1}{2}\left\{\frac{1}{ae^{-j(\omega+0.3\pi)} - 1} + \frac{1}{ae^{-j(\omega-0.3\pi)} - 1}\right\}$$

```python
from sympy import symbols, cos, pi, exp, simplify

# Define the symbols
omega, a = symbols('omega a', real=True, positive=True)
j = symbols('j', imaginary=True)

# Given Fourier transform X(omega)
def X_omega_expression(omega, a):
    return 1 / (1 - a * exp(-j * omega))

# Compute the Fourier transform for x[n] cos(0.3*pi n)
def compute_X3(omega, a):
    omega_0 = 0.3 * pi
    X_omg_minus = X_omega_expression(omega - omega_0, a)
    X_omg_plus = X_omega_expression(omega + omega_0, a)
    return (X_omg_minus + X_omg_plus) / 2

# Compute and simplify X3(omega)
X3_omega_simplified = simplify(compute_X3(omega, a))
print(X3_omega_simplified)
```

```
-1/(2*(a*exp(-j*(omega + 0.3*pi)) - 1)) - 1/(2*(a*exp(-j*(omega - 0.3*pi)) - 1))
```

Validated.

iv) $n[n] * u[n-1]$

convolution in time domain is multiplication in the frequency domain.

$$n[n] \rightarrow X(\omega)$$
$$n[n-1] \rightarrow X(\omega)e^{-j\omega}$$
$$X_4(\omega) = X(\omega)X(\omega)e^{-j\omega}$$
$$= \frac{e^{-j\omega}}{(1-ae^{-j\omega})^2} \quad (\text{Ans})$$

b) # Parseval's Theorem:

Parseval's theorem relates the energy of a signal in the time domain to the energy of it's Fourier series representation in the frequency domain

$$\sum_{n=-\infty}^{\infty} |x[n]|^2 = \sum_{k=-\infty}^{\infty} |c_k|^2$$

$x[n]$ is the signal in the class domain

c_k is the fourier series coefficient of the signal, given by:

$$c_k = \frac{1}{T}\int_T x(t) e^{-jw_0 kt}\, dt$$

for discrete integral is summation.

Discrete signal $x[n]$ with DFT $X[k]$

$$\sum_{n=0}^{N-1} |x[n]|^2 = \frac{1}{N}\sum_{k=0}^{N-1} |X[k]|^2$$

N is number of sample.

$x[n] = 1, 0, 1, 2, 3, 2 \quad N = 6$

Energy in time domain:

$$E_t = \sum_{n=0}^{N-1} |x[n]|^2$$

Energy in frequency domain after DFT

$$E_f = \frac{1}{N} \sum_{k=0}^{N-1} |X[k]|^2$$

```python
import numpy as np

def time_domain_energy(signal):
    """
    Compute the energy of a signal in the time domain.

    Parameters:
    - signal (list or np.array): Input discrete-time signal.

    Returns:
    - float: Energy of the signal in the time domain.
    """
    return sum([val**2 for val in signal])

def frequency_domain_energy(signal):
    """
    Compute the energy of a signal in the frequency domain using DFT.

    Parameters:
    - signal (list or np.array): Input discrete-time signal.

    Returns:
    - float: Energy of the signal in the frequency domain.
    """
    # Compute the Discrete Fourier Transform (DFT) of the signal
    X = np.fft.fft(signal)

    # Calculate the energy in the frequency domain
    N = len(signal)
    return (1/N) * sum([abs(val)**2 for val in X])
```

```python
def verify_parsevals_theorem(signal):
    """
    Verify Parseval's theorem for a given signal.

    Parameters:
    - signal (list or np.array): Input discrete-time signal.

    Returns:
    - bool: True if Parseval's theorem holds for the signal, False otherwise.
    """
    # Calculate energies in both domains
    E_t = time_domain_energy(signal)
    E_f = frequency_domain_energy(signal)
    print("Time domain energy is ",E_t)
    print("Frequency domain energy is ",E_f)

    # Check if the energies are approximately equal (accounting for potential floating-point inaccuracies)
    return np.isclose(E_t, E_f)

# Given signal
x = [1, 0, 1, 2, 3, 2]

# Verify Parseval's theorem for the signal
result = verify_parsevals_theorem(x)
print(f"Parseval's theorem verification for x[n]: {result}")
```

```
Time domain energy is  19
Frequency domain energy is  19.0
Parseval's theorem verification for x[n]: True
```

Parseval's theorem is proved.

E_t & E_f both have same energy.

---

3. **Answer the following:**

An FIR filter is described by the difference equation: $y[n] = x[n] + x[n-4]$.

(a) Compute and sketch its magnitude and phase response.

(b) Compute its response to the input $x[n] = \cos(\frac{\pi}{2}n) + \cos(\frac{\pi}{4}n)$

(c) Explain the results obtained in part (b) using those from part (a)

→ $y[n] = x[n] + x[n-4]$

For FIR filters, the impulse response is simply the coefficients of the difference equation.

$h[0] = 1 \quad h[1] = 0 \quad h[2] = 0 \quad h[3] = 0$

$h[4] = 1$

frequency response given by

$$H(e^{j\omega}) = \sum_{n=-\infty}^{\infty} h[n] e^{-j\omega n}$$

$$\Rightarrow H(e^{j\omega}) = h[0] + h[4] e^{-j4\omega}$$

$$\Rightarrow H(e^{j\omega}) = 1 + e^{-j4\omega}$$

Magnitude

$$|H(e^{j\omega})| = \sqrt{Re\{H(e^{j\omega})\}^2 + Im\{H(e^{j\omega})\}^2}$$

Phase response

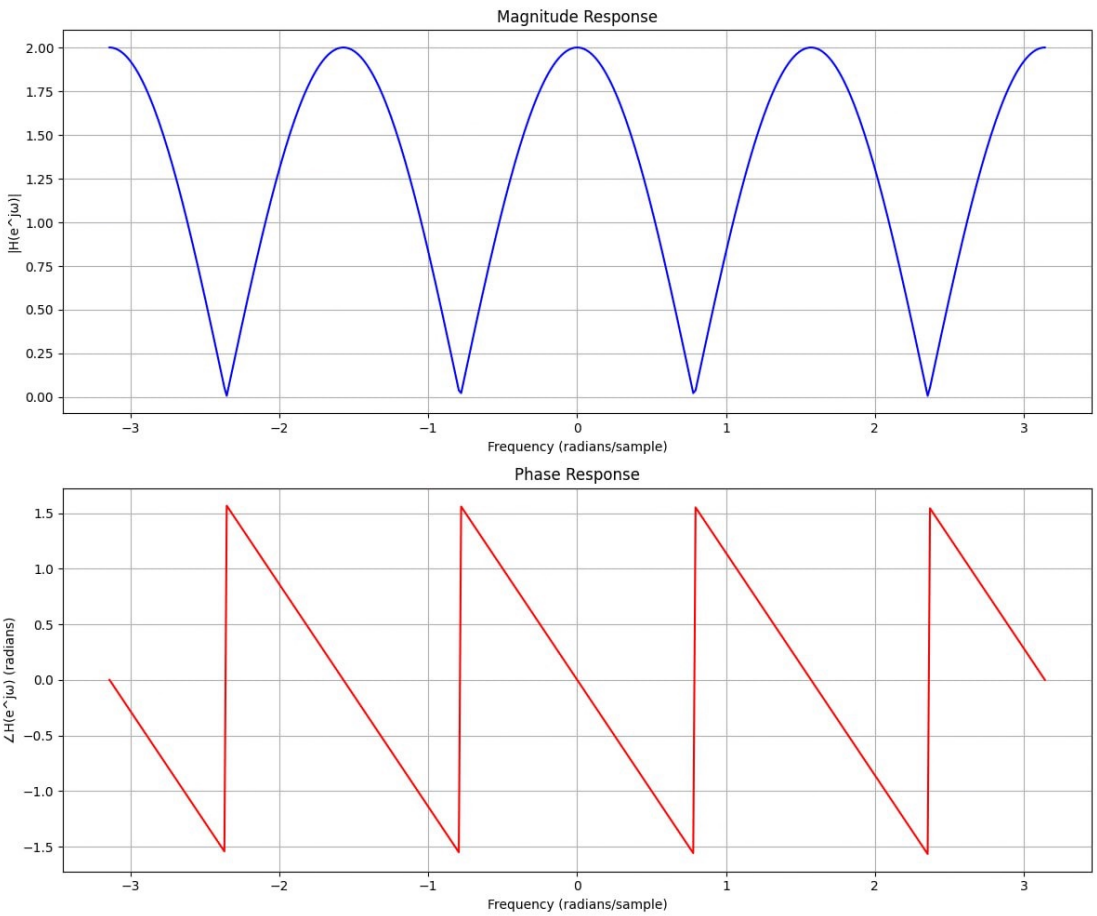
$$\angle H(e^{j\omega}) = \tan^{-1} \frac{Im(H(e^{j\omega}))}{Re(H(e^{j\omega}))}$$

$$H(e^{j\omega}) = 1 + \cos(4\omega) - j\sin(4\omega)$$

$$Re\{H(e^{j\omega})\} = 1 + \cos(4\omega)$$

$$\text{Im}\{H(e^{j\omega})\} = -j\sin(4\omega)$$

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

# Define omega values for plotting
omega = np.linspace(-np.pi, np.pi, 400)

# Compute frequency response using the given equation
H_omega = 1 + np.cos(4 * omega) - 1j * np.sin(4 * omega)

# Compute magnitude response
magnitude_response = np.abs(H_omega)

# Compute phase response (unwrapped to avoid discontinuities)
phase_response = np.unwrap(np.angle(H_omega))

# Plotting the magnitude and phase response
fig, ax = plt.subplots(2, 1, figsize=(12, 10))

# Magnitude Response
ax[0].plot(omega, magnitude_response, color='b')
ax[0].set_title('Magnitude Response')
ax[0].set_xlabel('Frequency (radians/sample)')
ax[0].set_ylabel('|H(e^jω)|')
ax[0].grid(True)

# Phase Response
ax[1].plot(omega, phase_response, color='r')
ax[1].set_title('Phase Response')
ax[1].set_xlabel('Frequency (radians/sample)')
ax[1].set_ylabel('∠H(e^jω) (radians)')
ax[1].grid(True)

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



b) $x[n] = \cos\left(\frac{\pi}{2}n\right) + \cos\left(\frac{\pi}{4}n\right)$

When a Linear time invariant (LTI) system is subjected to a sinusoidal input, the output will be sinusoidal, with same frequency as input. The amplitude & phase of the output sinusoid might differ from those of the input, and these differences are determined by the system's frequency response.

The input $x[n]$ is a sum of 2 sinusoids with frequency $\pi/2$ & $\pi/4$. Given the linearity of the system, we can determine the response of each sinusoid individually & then sum the results to get the overall response.

General form:

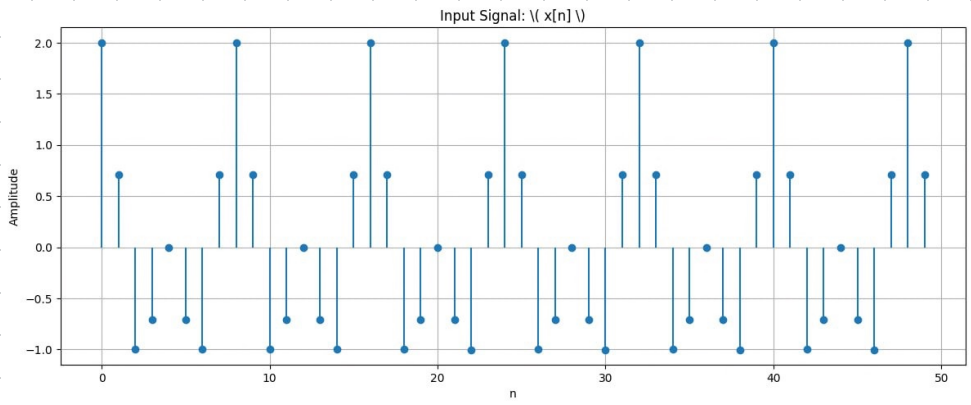
$$x[n] = A\cos(\omega_0 n + \phi)$$

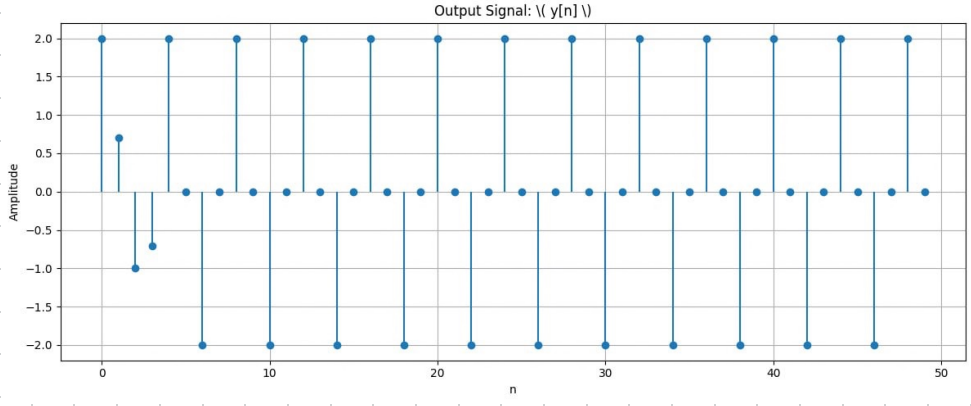
When such a sinusoid is passed through a LTI system with frequency response $H(e^{j\omega})$, the output sinusoid will have an amplitude scaled by the magnitude of the frequency response at ω_0 & a phase shift equal to the phase response at ω_0.

$$A_{out} = A \times |H(e^{j\omega_0})|$$

$$\phi_{out} = \phi + \angle H(e^{j\omega_0})$$

```
n = np.arange(0, 50)

# Given input signals
x1 = np.cos(np.pi/2 * n)
x2 = np.cos(np.pi/4 * n)
x = x1 + x2

# Compute the system's response to each sinusoid separately
# Using the difference equation: y[n] = x[n] + x[n - 4]
y1 = x1 + np.pad(x1, (4, 0), 'constant')[:-4]
y2 = x2 + np.pad(x2, (4, 0), 'constant')[:-4]

# Sum the results to get the overall response
y = y1 + y2

# Plotting the input and output signals
fig, ax = plt.subplots(2, 1, figsize=(12, 10))

# Input Signal
ax[0].stem(n, x, basefmt=" ", use_line_collection=True)
ax[0].set_title('Input Signal: \( x[n] \)')
ax[0].set_xlabel('n')
ax[0].set_ylabel('Amplitude')
ax[0].grid(True)

# Output Signal
ax[1].stem(n, y, basefmt=" ", use_line_collection=True)
ax[1].set_title('Output Signal: \( y[n] \)')
ax[1].set_xlabel('n')
ax[1].set_ylabel('Amplitude')
ax[1].grid(True)

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





Let's first apply filter to $\cos(\frac{\pi}{2}n)$

$y_1[n] = \cos(\frac{\pi}{2}n) + \cos(\frac{\pi}{2}(n-4))$

$\cos(\theta - 2\pi) = \cos\theta$

$y_1[n] = \cos(\pi/2\, n) + \cos(\pi/2)n$

$= 2\cos(\pi/2\, n)$

for $\cos(\pi/4\, n)$:

$y_2[n] = \cos(\pi/4\, n) + \cos(\pi/4(n-4))$

$\cos(\theta - \pi) = -\cos\theta$

$y_2[n] = \cos(\pi/4\, n) - \cos(\pi/4\, n) = 0$

$\therefore y[n] = y_1[n] + y_2[n]$

$= 2\cos(\pi/2\, n)$.

c) $H(e^{j\omega}) = 1 + e^{-j\omega}$

$y[n] = 2\cos(\frac{\pi}{2}n)$

at $\omega = \pi/2$ amplitude doubles for FIR filter response, so $y[n]$ has a component at $\omega = \frac{\pi}{2}$ with twice the amplitude of corresponding input component At $\omega = \pi/4$ filter's magnitude is 0 leading to attenuation of this component in the output.

4. **Answer the following:**

If $w_1[k] = (1 + c_1 q^{-1})e_1[k]$ and $w_2[k] = (1 + c_2 q^{-1})e_2[k]$, show that $w_3[k] = w_1[k] + w_2[k]$ may be written as $w_3[k] = (1 + c_3 q^{-1})e_3[k]$, and derive an expression for c_3 and $\sigma^2_{e_3}$ in terms of the other two processes.

→ $w_1[k] = (1 + c_1 q^{-1}) e_1[k]$

$w_2[k] = (1 + c_2 q^{-1}) e_2[k]$

$w_3[k] = (1 + c_1 q^{-1}) e_1[k] + (1 + c_2 q^{-1}) e_2[k]$

Assuming $e_1[k]$ & $e_2[k]$ are uncorrelated, we can define a new process $e_3[k]$ such that $e_3[k] = e_1[k] + e_2[k]$. The variance of $e_3[k]$ is given by the sum of variances of $e_1[k]$ & $e_2[k]$ due to their uncorrelated nature.

$$w_3[k] = e_3[k] + (c_1 q^{-1} e_1[k] + c_2 q^{-1} e_2[k])$$

or, $c_3 q^{-1} e_1[k] + c_3 q^{-1} e_2[k] = c_1 q^{-1} e_1[k] + c_2 q^{-1} e_2[k]$

$$\therefore c_3 = \frac{c_1 \sigma_{e_1}^2 + c_2 \sigma_{e_2}^2}{\sigma_{e_1}^2 + \sigma_{e_2}^2}$$

```
# Let's define the variances and coefficients as symbols
from sympy import symbols, Eq, solve

# Define the symbols for the variances and coefficients
sigma_e1_squared, sigma_e2_squared, sigma_e3_squared = symbols('sigma_e1_squared sigma_e2_squared sigma_e3_squared')
c1, c2, c3 = symbols('c1 c2 c3')

# Expression for c3 as a function of c1, c2, sigma_e1_squared, and sigma_e2_squared
c3_expression = (c1 * sigma_e1_squared + c2 * sigma_e2_squared) / (sigma_e1_squared + sigma_e2_squared)

# Expression for sigma_e3_squared as a function of sigma_e1_squared and sigma_e2_squared
sigma_e3_squared_expression = sigma_e1_squared + sigma_e2_squared

# Display the expressions for c3 and sigma_e3_squared
c3_expression, sigma_e3_squared_expression
```

```
((c1*sigma_e1_squared + c2*sigma_e2_squared)/(sigma_e1_squared + sigma_e2_squared),
 sigma_e1_squared + sigma_e2_squared)
```

} output

For $c_3 = \frac{c_1 \sigma_{e_1}^2 + c_2 \sigma_{e_2}^2}{\sigma_{e_1}^2 + \sigma_{e_2}^2}$

For $\sigma^2_{e_3}$ $\quad \sigma^2_{e_3} = \sigma^2_{e_1} + \sigma^2_{e_3}$

$$w_3[k] = (1 + c_1 q^{-1}) e_1[k] + (1 + c_2 q^{-1}) e_2[k]$$

$$w_3[k] = e_1[k] + c_1 q^{-1} e_1[k] + e_2[k] + c_2 q^{-1} e_2[k]$$

grouping:

$$w_3[k] = e_3[k] + (c_1 e_1[k-1] + c_2 e_2[k-1]) q^{-1}$$

{as q^{-1} is delay operator so
$q^{-1} e_1[k] = e_1[k-1]$ & $q^{-1} e_2[k] = e_2[k-1]$}

$$\therefore w_3[k] = e_3[k] + c_3 e_3[k-1] q^{-1}$$

$$w_3[k] = (1 + c_3 q^{-1}) e_3[k]$$
