1. **Program to generate the following discrete time signals. a) Unit sample sequence, b)Unit step sequence, c) Exponential sequence, d)Sinusoidal sequence, e) Random sequence**

```
clc; clear all; close all;
N = 10;
n = -N:1:N;
% unit Impulse
x1=[zeros(1,N) 1 zeros(1,N)];
subplot(4,2,1); stem(n,x1); title(' Impulse Signal');
%unit Step signal
x2=[zeros(1,N) 1 ones(1,N)];
subplot(4,2,2); stem(n,x2); title('Unit step signal');
%Exponential growing/decaying signal
n2=0:0.1:N;
a2=2;
x4=a2.^n2;
subplot(4,2,3); stem(n2,x4); title( 'Exponential growing
signal');
a3=0.5;
x5=a3.^n2;
subplot(4,2,4); stem(n2,x5); title( 'Exponential decaying
signal');
%cosine signal
x6=cos(n2);
subplot(4,2,5); stem(n2,x6); title('Cosine signal');
%sine signal
x7=sin(n2);
subplot(4,2,6); stem(n2,x7); title('sine signal');
% random sequence
x8=rand(1, 10)
subplot(4,2,7);
hist(x8);title('random sequence');
```

2. **Program to perform the following operations on signals. a) Signal addition, b) Signal multiplication, c)Scaling, d) Shifting, e)Folding**

```
x=[1 2 1 2];
subplot (3,2,1); stem(x);title('input x');
y=[2 2 1 1];
subplot (3,2,2); stem(y);title('input y');
%addition
a=x+y;
subplot (3,2,3); stem(a);title('addition');
%multiplication
b=x.*y;
subplot (3,2,4); stem(b);title('multilplication');
clc;close all; clear all;

%folding a signal
clc;close all; clear all;
n=-1:2;
x=[3 -1 0 -4];
subplot(2,1,1);stem(n,x);axis([-3 3 -5 5]);
title('Signal x(n)');
c=fliplr(x);
y=fliplr(-n);
subplot(2,1,2);stem(y,c);axis([-3 3 -5 5]);
title('Reversed Signal x(-n)') ;


%shifting signal
n1=input('Enter the amount to be delayed');
n2=input('Enter the amount to be advanced');
n=-2:2;
x=[-2 3 0 1 5];
subplot(3,1,1);stem(n,x);title('Signal x(n)');
m=n+n1;
y=x;
subplot(3,1,2);stem(m,y);title('Delayed signal x(n-n1)');
```

```
t=n-n2;
z=x;
subplot(3,1,3);stem(t,z);title('Advanced signal x(n+n2)');
clc;close all; clear all;

%scaling
n=[-2 -1 0 1 2];
x=[3 1 0 4 6];
subplot(3,1,1);stem(n,x);title('original');hold on; %original
n2 = 4*n;
subplot(3,1,2);stem(n2,x);title('scaled by 4') %x4
n3 = 2*n;
subplot(3,1,3);stem(n3,x);title ('scaled by 2')%x2
```

3. **Program to perform convolution of two given sequences (without using built-in function) and display the signals.**

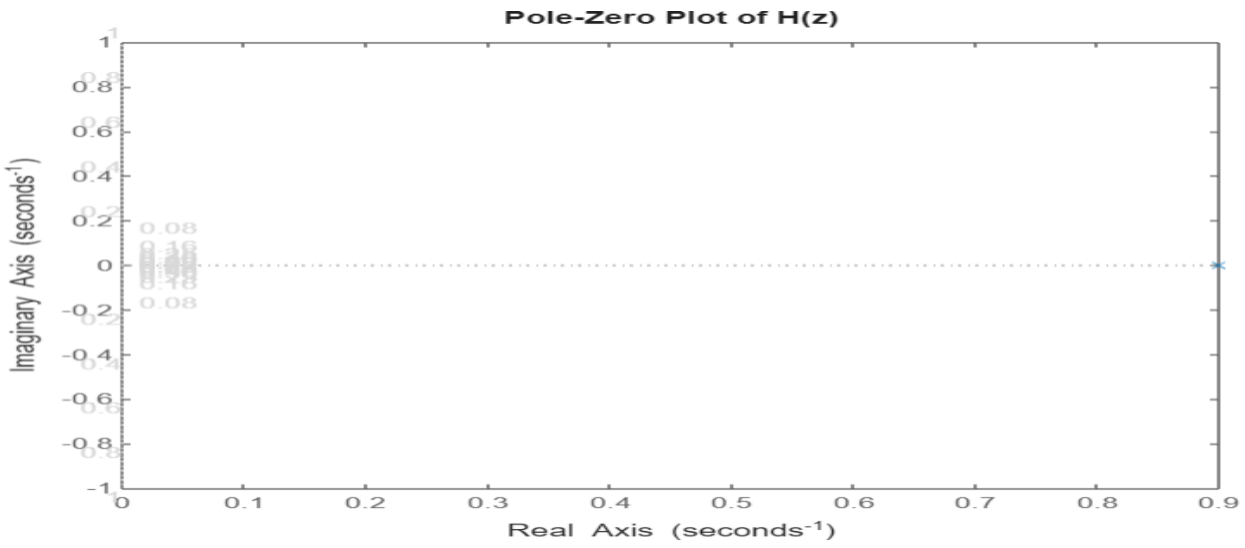
```
clc;close all;
%inputs
x=input('enter the input sequence of x(n)');
h=input('enter the input sequence of h(n)');
%find the length of signal
n1=length(x);
n2=length(h);
%find length of y(n)
N=n1+n2-1;
%zero padding to make the length=N
x=[x,zeros(1,N-n1)];
h=[h,zeros(1,N-n2)];
%initialse the output with zero
y=zeros(1,N);
```

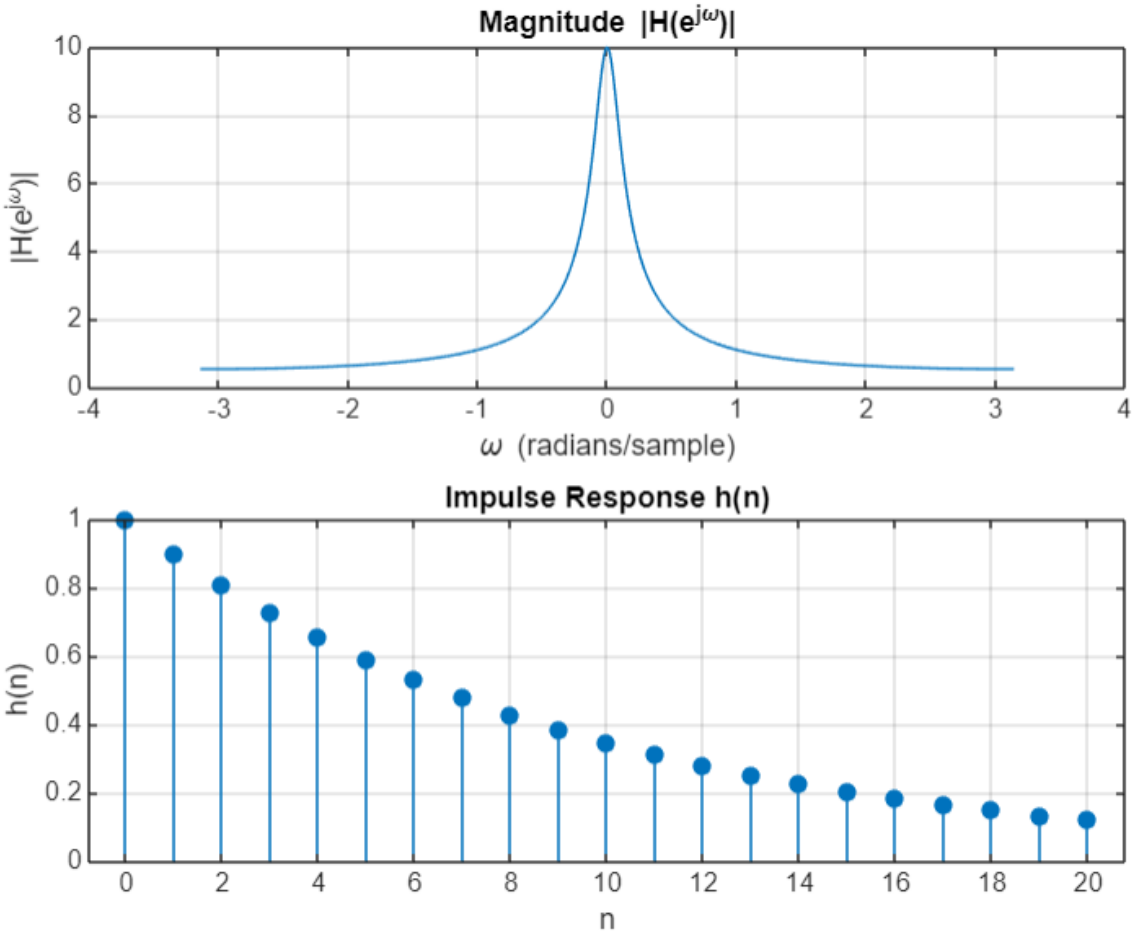
```
%perform convolution
for n=1:N
    for k=1:n
        y(n)=y(n)+x(k).*h(n-k+1);
    end
end
disp(y);
%plot the inputs & outputs
ny=0:N-1;
subplot(3,1,1);stem(ny,x);title('first sequence');
subplot(3,1,2);stem(ny,h);title('second sequence');
subplot(3,1,3);stem(ny,y);title('Convoluted sequence');
```

4. Consider a causal system y(n) = 0.9y(n-1)+x(n).
a) Determine H(z) and sketch its pole zero plot.
b) Plot |H(ejω)| and ∠ H(ejω)
c) Determine the impulse response h(n).

```
% Part A: Transfer Function and Pole-Zero Plot
numerator=1;   % z
denominator=[1,-0.9]; % z - 0.9
H=tf(numerator,denominator);
% Pole-Zero plot
figure;
pzmap(H);
title('Pole-Zero Plot of H(z)');
grid on;
% Part B: Frequency Response
omega = linspace(-pi, pi, 512); % Frequency from -pi to pi
H_freq = freqz(numerator, denominator, omega);
```

```
magnitude = abs(H_freq);
phase = angle(H_freq);
% Plot Magnitude
figure;
subplot(2,1,1);
plot(omega, magnitude);
title('Magnitude |H(e^{j\omega})|');
xlabel('\omega (radians/sample)');
ylabel('|H(e^{j\omega})|');
grid on;
% Plot Phase
subplot(2,1,2); plot(omega, phase);
title('Phase \angle H(e^{j\omega})');
xlabel('\omega (radians/sample)');
ylabel('\angle H(e^{j\omega}) (radians)');
grid on;

% Part C: Impulse Response
n= 0:20; % Sample range
h = (0.9).^n; % Impulse response for n >= 0
% Plot Impulse Response figure;
stem(n,h,'filled');
title('Impulse Response h(n)');
xlabel('n');
ylabel('h(n)');
grid on;
```

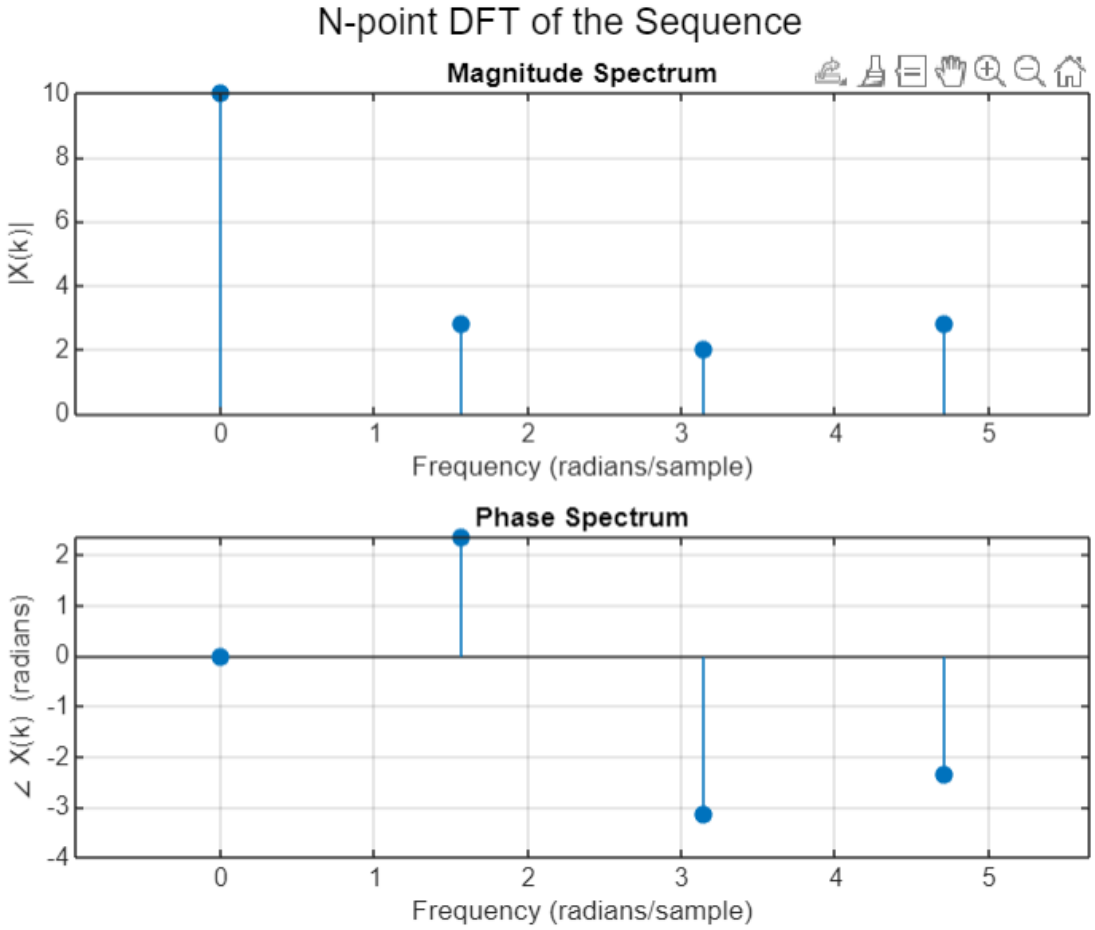




5. Computation of N point DFT of a given sequence (without using built-in function) and to plot the magnitude and phase spectrum.

```
% Define the input sequence
x = [1, 2, 3, 4]; % Example sequence
N = length(x); % Number of points in DFT

% Initialize the DFT output
X = zeros(1, N); % DFT output array

% Compute the N-point DFT for k = 0:N-1
for k = 0:N-1
for n = 0:N-1
X(k+1) = X(k+1) + x(n+1) * exp(-1j * 2 * pi * k * n / N);
```

```
end
end
% Magnitude and Phase Spectrum
magnitude = abs(X);
phase = angle(X);
% Frequency bins for plotting
frequencies = (0:N-1) * (2 * pi / N); % Frequency range
% Plotting the Magnitude Spectrum figure;
subplot(2, 1, 1);
stem(frequencies, magnitude, 'filled'); title('Magnitude
Spectrum'); xlabel('Frequency (radians/sample)');
ylabel('|X(k)|');
grid on;
% Plotting the Phase Spectrum
subplot(2, 1, 2); stem(frequencies, phase, 'filled');
title('Phase Spectrum');
xlabel('Frequency (radians/sample)'); ylabel('\angle X(k)
(radians)');
grid on;
% Adjust layout
sgtitle('N-point DFT of the Sequence');
```


N-point DFT of the Sequence
Magnitude Spectrum
|X(k)|
Frequency (radians/sample)
Phase Spectrum
∠ X(k) (radians)
Frequency (radians/sample)

6. Using the DFT and IDFT, compute the following for any two given sequences a)Circular convolution b) Linear convolution

```
% Define two example sequences

x = [1, 2, 3];  % First sequence

h = [0.5, 1]; % Second sequence

% Lengths of the sequences

N = max(length(x), length(h)); % Length for DFT (circular
convolution)

L = length(x) + length(h) - 1; % Length for linear convolution

% Circular Convolution using DFT and IDFT

X = fft(x, N); % DFT of x

H = fft(h, N); % DFT of h

% Circular convolution result

Y_circular = X .* H; % Element-wise multiplication in
frequency domain

y_circular = ifft(Y_circular); % IDFT to get the circular
convolution result

% Linear Convolution using DFT and IDFT

% Zero-pad the sequences to length L

x_padded = [x,zeros(1,L-length(x))]; % Zero-pad x

h_padded = [h,zeros(1,L-length(h))]; % Zero-pad h

% DFT of the zero-padded sequences

X_linear = fft(x_padded, L); % DFT of x padded

H_linear = fft(h_padded, L); % DFT of h padded

% Linear convolution result

Y_linear = X_linear .* H_linear; % Element-wise multiplication
in frequency domain

y_linear = ifft(Y_linear); % IDFT to get the linear
convolution result

% Display Results

disp('Circular Convolution Result:'); disp(y_circular);

disp('Linear Convolution Result:'); disp(y_linear);
```

Circular Convolution Result:

3.5000 2.0000 3.5000

Linear Convolution Result:

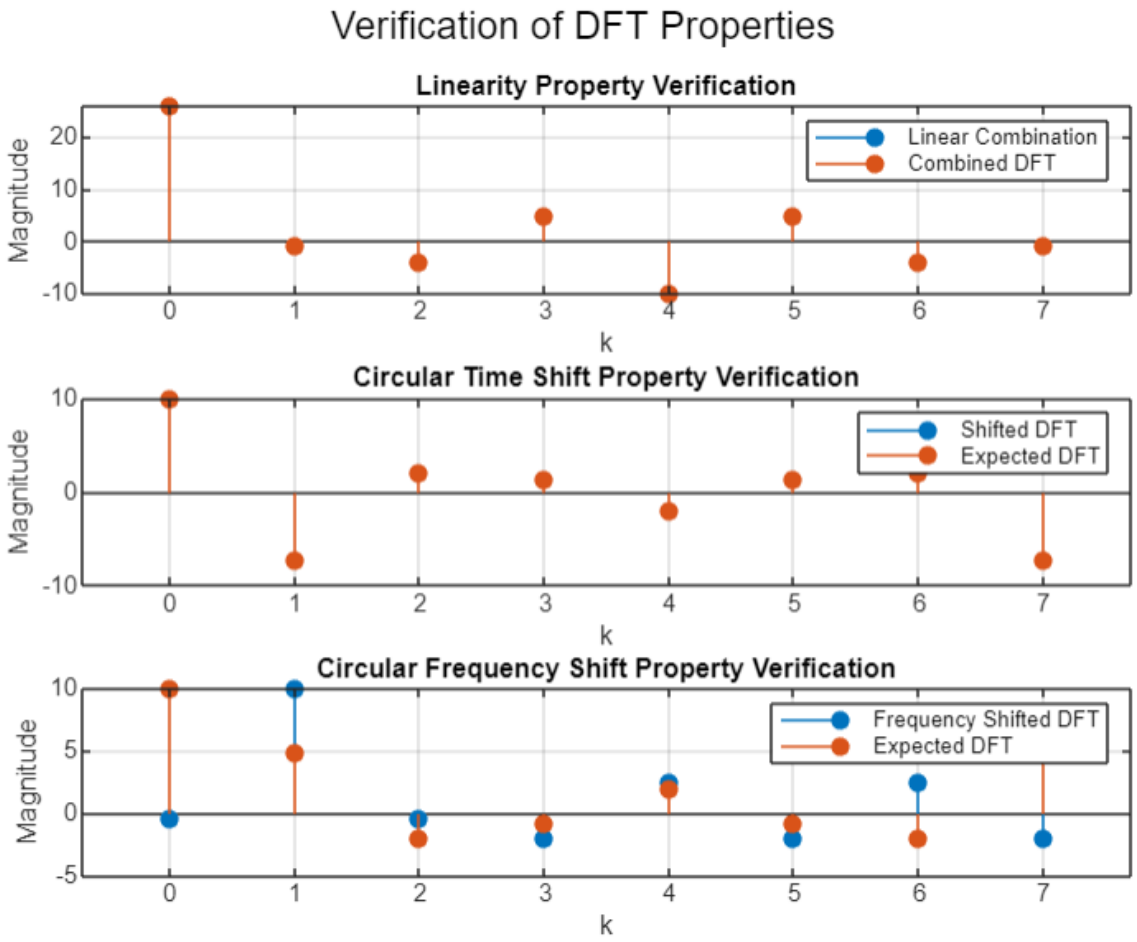
0.5000 2.0000 3.5000 3.0000

7. Verification of Linearity property, circular time shift property & circular frequency shift property of DFT.

```
% Define parameters

N = 8; % Length of DFT

n = 0:N-1; % Sample indices

% Define two sequences for testing

x1 = [1, 2, 3, 4, 0, 0, 0, 0]; % First sequence

x2 = [0, 1, 0, 1, 0, 0, 0, 0]; % Second sequence

% Define coefficients for linearity

alpha = 2; % Coefficient for x1

beta = 3; % Coefficient for x2

% 1. Linearity Property

% DFT of individual sequences

X1 = fft(x1);

X2 = fft(x2);

% DFT of the linear combination

X_linear_combination = fft(alpha * x1 + beta * x2);

% Verify linearity

X_combined = alpha * X1 + beta * X2;

% Display results

disp('Linearity Verification:');

disp('DFT of linear combination:');
disp(X_linear_combination); disp('Linear combination of
DFTs:'); disp(X_combined);

% 2. Circular Time Shift Property

% Time shift by 2 samples

shift = 2;

x_shifted = circshift(x1, shift); % Circularly shift x1 by 2

X_shifted = fft(x_shifted); % DFT of the shifted sequence

X_original = fft(x1); % DFT of the original sequence

% Verify circular time shift property

% Expected: X_shifted = X_original * e^(-j*2*pi*shift/N*n)

% Display results

disp('Circular Time Shift Verification:'); disp('DFT of
shifted sequence:'); disp(X_shifted);

disp('Expected result:');

expected_shifted = X_original .* exp(-1j*2*pi*shift/N * n);
disp(expected_shifted);
```

```
% 3. Circular Frequency Shift Property

% Frequency shift by 1 sample

freq_shift = 1;

X_freq_shifted = circshift(X_original,freq_shift);

% Circularly shift DFT

% Verify circular frequency shift property

% Expected: x[n] * e^(j*2*pi*freq_shift*n/N)

% Display results

disp('Circular Frequency Shift Verification:'); disp('DFT of
frequency shifted sequence:'); disp(X_freq_shifted);

disp('Expected result:');

expected_freq_shifted = X_original .* exp(1j * 2 * pi *
freq_shift * n / N); disp(expected_freq_shifted);

% Plotting results for verification figure;

% Linearity Property

subplot(3, 1, 1);

stem(n, real(X_linear_combination), 'filled', 'DisplayName',
'Linear Combination'); hold on;

stem(n, real(X_combined), 'filled', 'DisplayName', 'Combined
DFT'); title('Linearity Property Verification');

xlabel('k'); ylabel('Magnitude'); legend;

grid on;

% Circular Time Shift Property

subplot(3, 1, 2);

stem(n, real(X_shifted), 'filled', 'DisplayName', 'Shifted
DFT'); hold on;

stem(n, real(expected_shifted), 'filled', 'DisplayName',
'Expected DFT'); title('Circular Time Shift Property
Verification');

xlabel('k');

ylabel('Magnitude'); legend;

grid on;

% Circular Frequency Shift Property

subplot(3, 1, 3);

stem(n, real(X_freq_shifted), 'filled', 'DisplayName',
'Frequency Shifted DFT'); hold on;

stem(n, real(expected_freq_shifted), 'filled', 'DisplayName',
'Expected DFT'); title('Circular Frequency Shift Property
Verification');

xlabel('k'); ylabel('Magnitude'); legend;
```

```
grid on;
sgtitle('Verification of DFT Properties');
```



8. **Develop decimation in time radix-2 FFT algorithm without using built-in functions.**

```
function X = radix2_fft(x)

% Check if the length of x is a power of 2 N = length(x);

if mod(N, 2) ~= 0 || N == 0 || (N & (N - 1)) ~= 0
error('Length of input must be a power of 2');

end

% Recursive FFT computation if N == 1

X = x; % Base case return;

end

% Split into even and odd parts

X_even = radix2_fft(x(1:2:N)); % FFT of even-indexed elements
X_odd = radix2_fft(x(2:2:N)); % FFT of odd-indexed elements

% Combine results X = zeros(1, N); for k = 0:N/2-1

twiddle_factor = exp(-1j * 2 * pi * k / N) .* X_odd(k + 1); %
Twiddle factor X(k + 1) = X_even(k + 1) + twiddle_factor; %
Combine even and odd

X(k + N/2) = X_even(k + 1) - twiddle_factor; % Combine even
and odd end

end

% Example Usage

N = 8; % Length of the input signal (must be a power of 2) x =
[1, 2, 3, 4, 0, 0, 0, 0]; % Example input sequence

% Compute the FFT X = radix2_fft(x);

% Display results disp('FFT Result:'); disp(X);

% Plot the magnitude and phase of the FFT figure;

subplot(2, 1, 1);

stem(0:N-1, abs(X), 'filled'); title('Magnitude of FFT');
xlabel('Frequency Index');

ylabel('|X(k)|'); grid on;

subplot(2, 1, 2);

stem(0:N-1, angle(X), 'filled'); title('Phase of FFT');
xlabel('Frequency Index'); ylabel('\angle X(k) (radians)');
grid on;

sgtitle('Radix-2 FFT Result');
```

9. Design and implementation of digital low pass FIR filter using a window to meet the given specifications.

```
% Filter specifications
 Fs = 1000;
          % Sampling frequency (Hz)
 Fc = 150;
 N = 21;
           % Cutoff frequency (Hz)
             % Filter order (must be odd)
 window_type = 'hamming'; % Type of window ('hamming',
'hanning', 'rectangular')
 % Normalize the cutoff frequency
 Wn = Fc / (Fs / 2);  % Normalized cutoff frequency (0 to 1)
 % Design the ideal low-pass filter (sinc function)
 n = 0:N-1;
          % Time index
 h_ideal = 2 * Wn * sinc(2 * Wn * (n - (N-1)/2)); % Ideal
filter impulse response
 % Apply window to the ideal filter
 if strcmp(window_type, 'hamming')
    w = hamming(N)';
 elseif strcmp(window_type, 'hanning')
    w = hanning(N)';
 elseif strcmp(window_type, 'rectangular')
    w = ones(1, N);
 else
    error('Unsupported window type. Use hamming, hanning, or
rectangular.');
 end
 h = h_ideal .* w;  % Apply window
 % Normalize the filter coefficients
 h = h / sum(h);
     % Normalize to ensure unity gain at DC
 % Frequency response of the filter
```

```
[H, f] = freqz(h, 1, 1024, 'whole');  % Frequency response
% Plot the filter coefficients
figure;
subplot(2, 1, 1);
stem(0:N-1, h, 'filled');
title('FIR Filter Coefficients');
xlabel('n (samples)');
ylabel('h[n]');
grid on;
% Plot the frequency response
subplot(2, 1, 2);
plot(f * Fs / (2 * pi), 20*log10(abs(H))); % Convert to dB
title('Frequency Response of the FIR Filter');
 xlabel('Frequency (Hz)');
ylabel('Magnitude (dB)');
grid on;
% Add a vertical line at the cutoff frequency
hold on;
xline(Fc, '--r', 'Cutoff Frequency');
hold off;
sgtitle('Low-Pass FIR Filter Design Using Window Method');
% Optional: Test the filter with a sample signal
t = 0:1/Fs:1;
    % Time vector
signal = sin(2*pi*50*t) + sin(2*pi*200*t); % Example signal
(50 Hz + 200 Hz)
filtered_signal = filter(h, 1, signal);    % Apply filter
% Plot the original and filtered signal
figure;
subplot(2, 1, 1);
plot(t, signal);
title('Original Signal');
xlabel('Time (s)');
ylabel('Amplitude');
grid on;
subplot(2, 1, 2);
plot(t, filtered_signal);
```

```
title('Filtered Signal (Low-Pass)');
xlabel('Time (s)');
ylabel('Amplitude');
grid on;
 sgtitle('Signal Filtering using FIR Low-Pass Filter');
```

10. Design and implementation of digital High pass FIR filter using a window to meet the given specifications.

```
% Filter specifications
Fs = 1000;
         % Sampling frequency (Hz)
Fc = 150;
N = 21;
          % Cutoff frequency (Hz)
            % Filter order (must be odd)
window_type = 'hamming'; % Type of window ('hamming',
'hanning', 'rectangular')
% Normalize the cutoff frequency
Wn = Fc / (Fs / 2);  % Normalized cutoff frequency (0 to 1)
% Design the ideal high-pass filter (using sinc function)
n = 0:N-1;
         % Time index
h_ideal = -2 * Wn * sinc(2 * Wn * (n - (N-1)/2)); % Ideal
filter impulse response
h_ideal((N-1)/2 + 1) = h_ideal((N-1)/2 + 1) + 1; % Correct
the DC gain
% Apply window to the ideal filter
if strcmp(window_type, 'hamming')
   w = hamming(N)';
elseif strcmp(window_type, 'hanning')
   w = hanning(N)';
elseif strcmp(window_type, 'rectangular')
   w = ones(1, N);
else
   error('Unsupported window type. Use hamming, hanning, or
rectangular.');
end
```

```
h = h_ideal .* w;  % Apply window
% Normalize the filter coefficients
h = h / sum(w);
    % Normalize to ensure unity gain at high frequencies
% Frequency response of the filter
[H, f] = freqz(h, 1, 1024, 'whole');  % Frequency response
 % Plot the filter coefficients
figure;
subplot(2, 1, 1);
stem(0:N-1, h, 'filled');
title('FIR High-Pass Filter Coefficients');
xlabel('n (samples)');
ylabel('h[n]');
grid on;
% Plot the frequency response
subplot(2, 1, 2);
plot(f * Fs / (2 * pi), 20*log10(abs(H))); % Convert to dB
title('Frequency Response of the FIR High-Pass Filter');
xlabel('Frequency (Hz)');
 ylabel('Magnitude (dB)');
grid on;
% Add a vertical line at the cutoff frequency
hold on;
xline(Fc, '--r', 'Cutoff Frequency');
hold off;
sgtitle('High-Pass FIR Filter Design Using Window Method');
% Optional: Test the filter with a sample signal
t = 0:1/Fs:1;
    % Time vector
signal = sin(2*pi*50*t) + sin(2*pi*200*t); % Example signal
(50 Hz + 200 Hz)
filtered_signal = filter(h, 1, signal);    % Apply filter
% Plot the original and filtered signal
figure;
subplot(2, 1, 1);
plot(t, signal);
title('Original Signal');
```

```
xlabel('Time (s)');
ylabel('Amplitude');
grid on;
subplot(2, 1, 2);
plot(t, filtered_signal);
title('Filtered Signal (High-Pass)');
xlabel('Time (s)');
ylabel('Amplitude');
grid on;
sgtitle('Signal Filtering using FIR High-Pass Filter');
```

11. Design and implementation of digital IIR Butterworth low pass filter to meet the given specifications.

```
% Filter specifications
Fs = 1000;
          % Sampling frequency (Hz)
Fc = 150;
           % Cutoff frequency (Hz)
            order = 4;
          % Order of the Butterworth filter
% Normalize the cutoff frequency
Wn = Fc / (Fs / 2);  % Normalized cutoff frequency (0 to 1)
% Design the Butterworth low-pass filter
[b, a] = butter(order, Wn, 'low'); % Get filter coefficients
% Frequency response of the filter
[H, f] = freqz(b, a, 1024, 'whole');  % Frequency response
% Plot the filter coefficients
figure;
subplot(2, 1, 1);
zplane(b, a);  % Zero-pole plot
title('Zero-Pole Plot of Butterworth Low-Pass Filter');
% Plot the frequency response
subplot(2, 1, 2);
plot(f * Fs / (2 * pi), 20*log10(abs(H))); % Convert to dB
title('Frequency Response of Butterworth Low-Pass Filter');
xlabel('Frequency (Hz)');
```

```
ylabel('Magnitude (dB)');
grid on;
% Add a vertical line at the cutoff frequency
hold on;
xline(Fc, '--r', 'Cutoff Frequency');
hold off;
sgtitle('Butterworth Low-Pass Filter Design');
 % Optional: Test the filter with a sample signal
t = 0:1/Fs:1;
    % Time vector
signal = sin(2*pi*50*t) + sin(2*pi*200*t); % Example signal
(50 Hz + 200 Hz)
filtered_signal = filter(b, a, signal);    % Apply filter
% Plot the original and filtered signal
figure;
subplot(2, 1, 1);
plot(t, signal);
title('Original Signal');
xlabel('Time (s)');
ylabel('Amplitude');
grid on;
subplot(2, 1, 2);
 plot(t, filtered_signal);
title('Filtered Signal (Butterworth Low-Pass)');
xlabel('Time (s)');
ylabel('Amplitude');
grid on;
sgtitle('Signal Filtering using Butterworth Low-Pass
Filter');
```

12. Design and implementation of digital IIR Butterworth High pass filter to meet the given specifications.

```
% Filter specifications
Fs = 1000;
         % Sampling frequency (Hz)
```

```
Fc = 150;
order = 4;
          % Cutoff frequency (Hz)
         % Order of the Butterworth filter
          % Normalize the cutoff frequency
Wn = Fc / (Fs / 2);  % Normalized cutoff frequency (0 to 1)
% Design the Butterworth high-pass filter
[b, a] = butter(order, Wn, 'high'); % Get filter coefficients
% Frequency response of the filter
[H, f] = freqz(b, a, 1024, 'whole');  % Frequency response
% Plot the filter coefficients
figure;
subplot(2, 1, 1);
zplane(b, a);  % Zero-pole plot
title('Zero-Pole Plot of Butterworth High-Pass Filter');
% Plot the frequency response
subplot(2, 1, 2);
plot(f * Fs / (2 * pi), 20*log10(abs(H))); % Convert to dB
title('Frequency Response of Butterworth High-Pass Filter');
xlabel('Frequency (Hz)');
ylabel('Magnitude (dB)');
grid on;
% Add a vertical line at the cutoff frequency
hold on;
xline(Fc, '--r', 'Cutoff Frequency');
hold off;
sgtitle('Butterworth High-Pass Filter Design');
% Optional: Test the filter with a sample signal
t = 0:1/Fs:1;
    % Time vector
signal = sin(2*pi*50*t) + sin(2*pi*200*t); % Example signal
(50 Hz + 200 Hz)
filtered_signal = filter(b, a, signal);    % Apply filter
% Plot the original and filtered signal
figure;
subplot(2, 1, 1);
plot(t, signal);
```

```
 title('Original Signal');
 xlabel('Time (s)');
 ylabel('Amplitude');
 grid on;
 subplot(2, 1, 2);
 plot(t, filtered_signal);
 title('Filtered Signal (Butterworth High-Pass)');
 xlabel('Time (s)');
 ylabel('Amplitude');
 grid on;
 sgtitle('Signal Filtering using Butterworth High-Pass
Filter');
```