1. Plot the following sequences (i) Unit sample sequence, (ii) Unit step sequence, (iii) Ramp sequence (iv) Exponential sequence (v) sine sequence (vi) cosine sequence. Also, down sample each of the above sequences and plot.

**Code:**

n = 0:50;

% (i) Unit sample

x1 = zeros(size(n)); x1(n==0) = 1;

% (ii) Unit step

x2 = double(n>=0);

% (iii) Ramp

x3 = n;

% (iv) Exponential (e.g., a^n with a=0.9)

a = 0.9; x4 = a.^n;

% (v) Sine

f = 0.05; x5 = sin(2\*pi\*f\*n);

% (vi) Cosine

x6 = cos(2\*pi\*f\*n);

figure;

subplot(3,2,1); stem(n, x1); title('Unit Sample');

subplot(3,2,2); stem(n, x2); title('Unit Step');

subplot(3,2,3); stem(n, x3); title('Ramp');

subplot(3,2,4); stem(n, x4); title('Exponential');

subplot(3,2,5); stem(n, x5); title('Sine');

subplot(3,2,6); stem(n, x6); title('Cosine');

% Downsample by 2

ds = 2;

figure;

for k=1:6

xn = eval(['x',num2str(k)]);

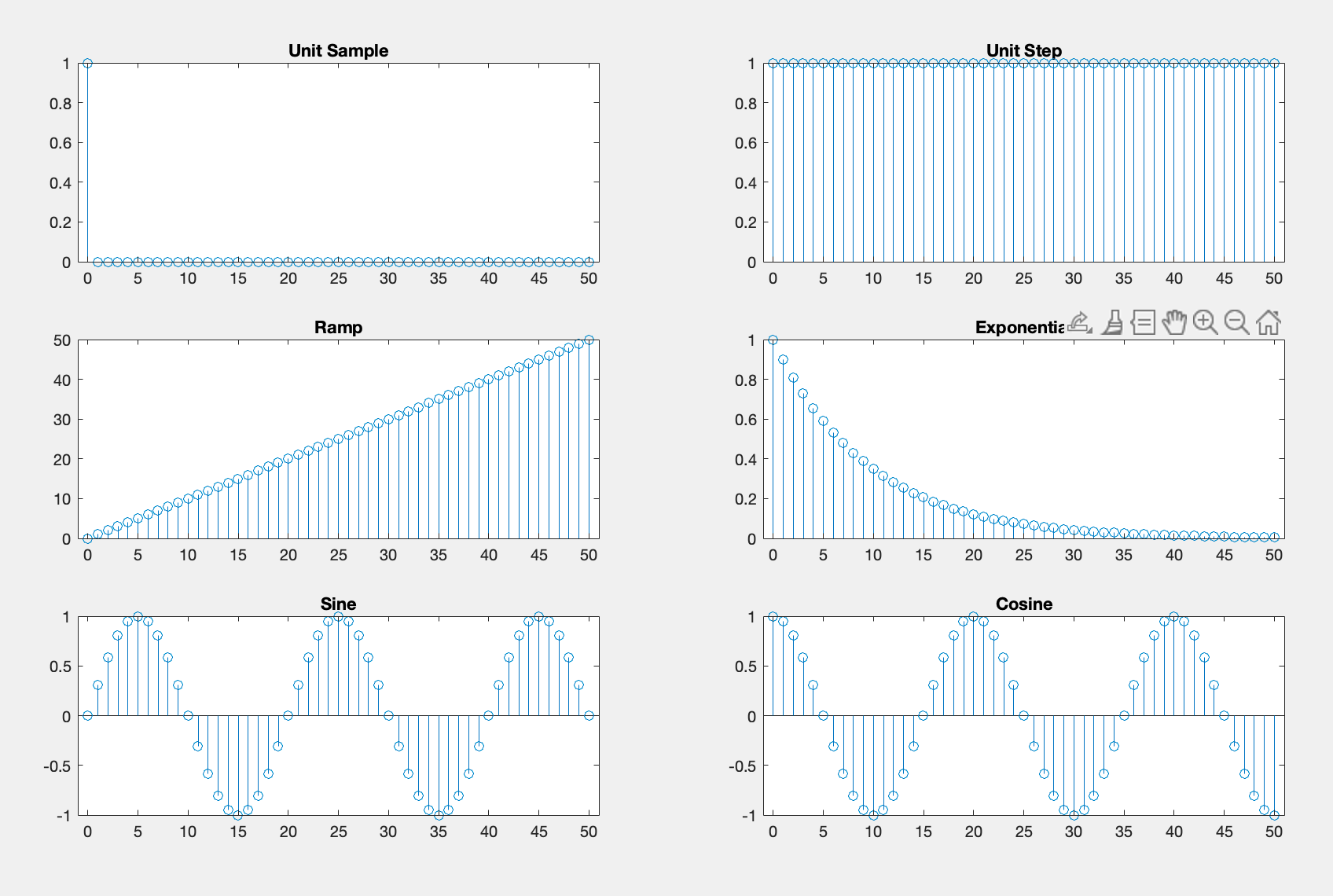
subplot(3,2,k);

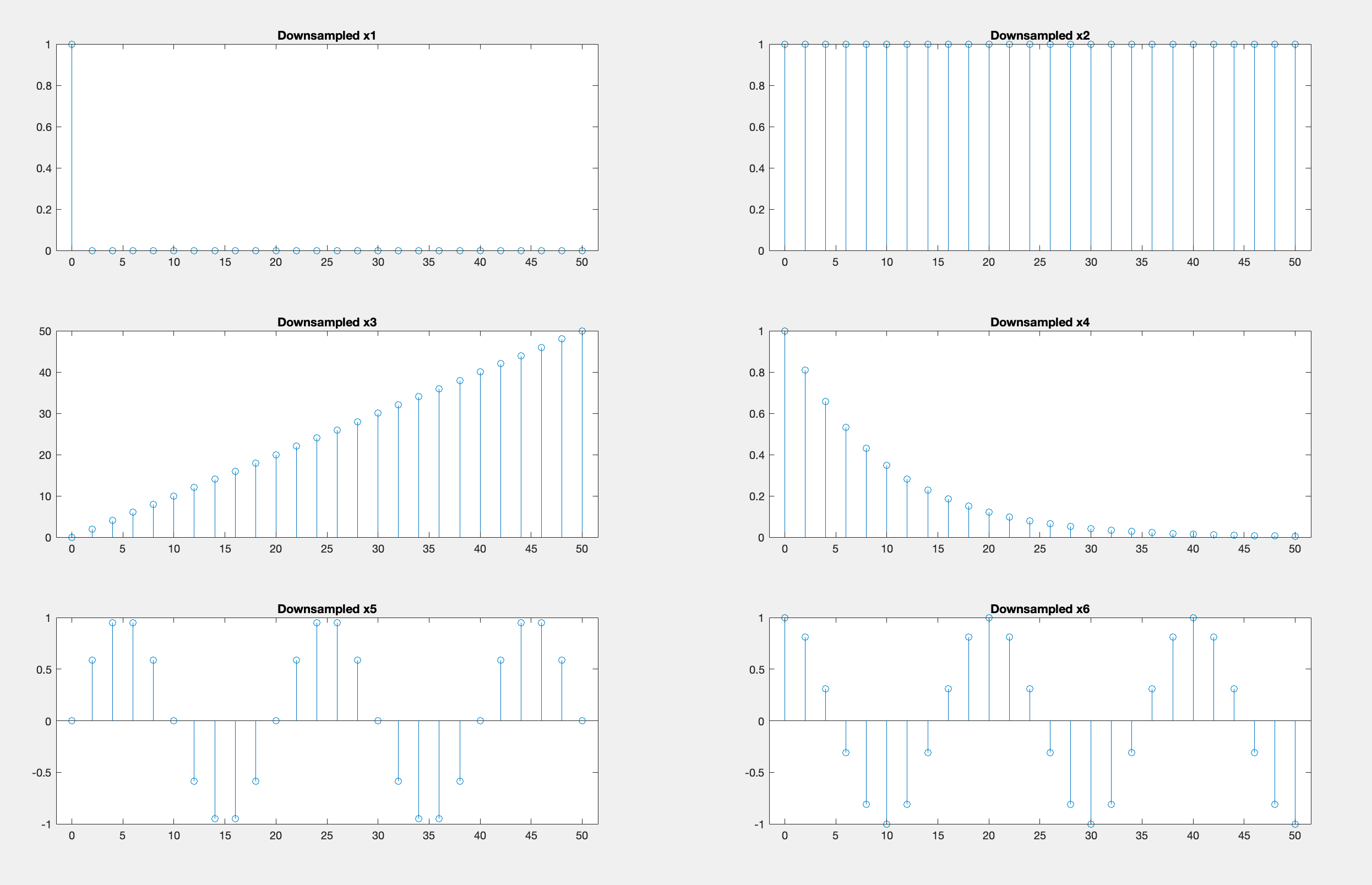
stem(n(1:ds:end), xn(1:ds:end));

title(['Downsampled x',num2str(k)]);

end

OUTPUT:





1. Write a MATLAB program to perform linear convolution of two sequences x(n) and h(n). Also verify the result using inbuilt functions.

CODE:

x = [1 2 3 4]; h = [1 -1 2];

N = length(x) + length(h) - 1;

y = zeros(1, N);

for i = 1:length(x)

for j = 1:length(h)

y(i + j - 1) = y(i + j - 1) + x(i) \* h(j);

end

end

% Plot manual result

figure; stem(0:N-1, y); title('Manual Linear Convolution');

% Verify using built-in conv()

y\_builtin = conv(x, h);

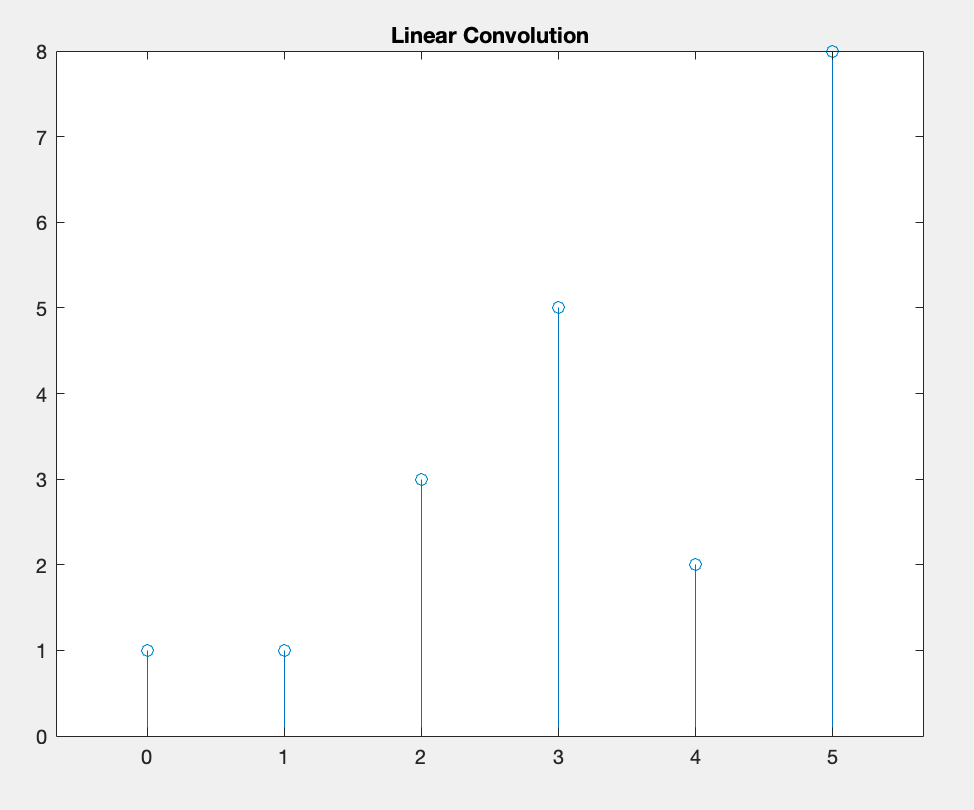
% Plot built-in result

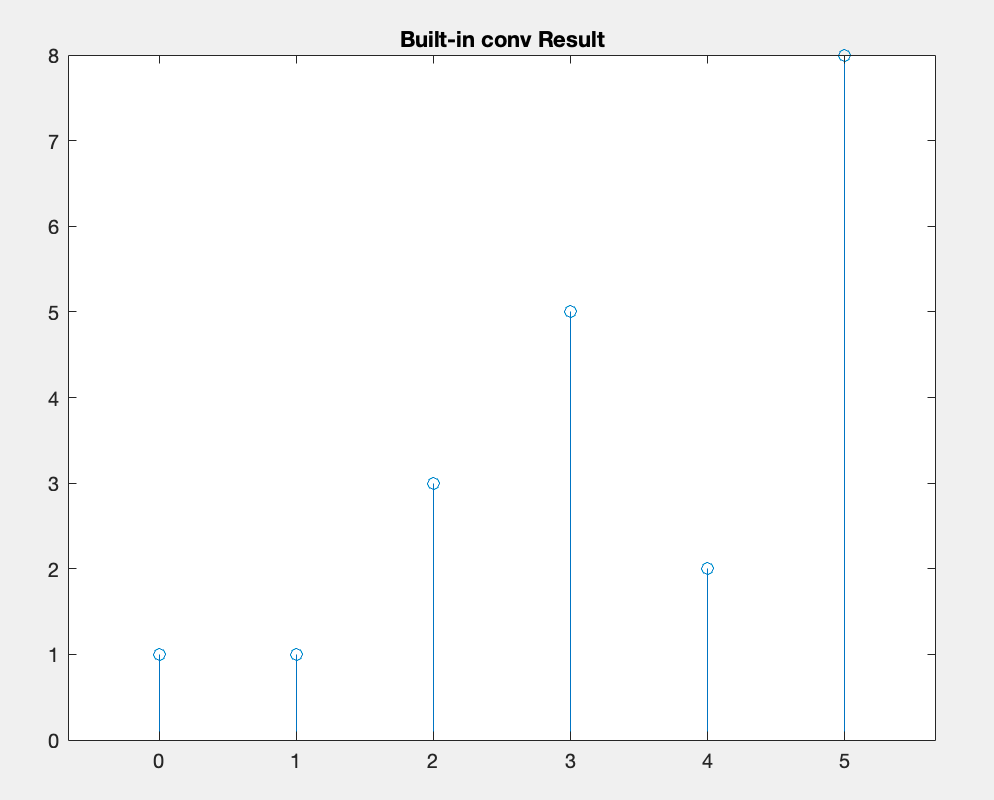
figure; stem(0:length(y\_builtin)-1, y\_builtin); title('Built-in conv Result');

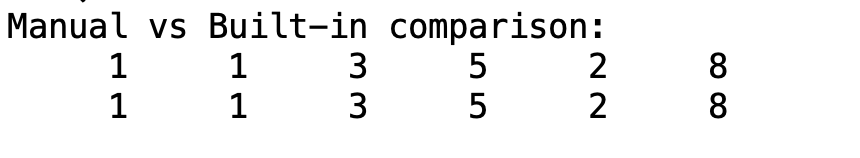
% Compare results

disp('Manual vs Built-in comparison:');

disp([y; y\_builtin]);

OUTPUT:





3. Write a MATLAB program to perform circular convolution of two sequences x(n) and h(n). Also verify the result using inbuilt functions.

CODE:

x = [1 2 3 4]; h = [1 -1 2];

Nfft = 8; % Circular convolution length

% Zero-pad sequences

x\_pad = [x zeros(1, Nfft-length(x))];

h\_pad = [h zeros(1, Nfft-length(h))];

% Manual circular convolution

circ = zeros(1, Nfft);

for n0 = 1:Nfft

for k = 1:Nfft

idx = mod(n0 - k, Nfft) + 1;

circ(n0) = circ(n0) + x\_pad(k) \* h\_pad(idx);

end

end

% Plot manual result

figure; stem(0:Nfft-1, circ); title('Manual Circular Convolution');

% Verify using built-in cconv()

circ\_builtin = cconv(x, h, Nfft);

% Plot built-in result

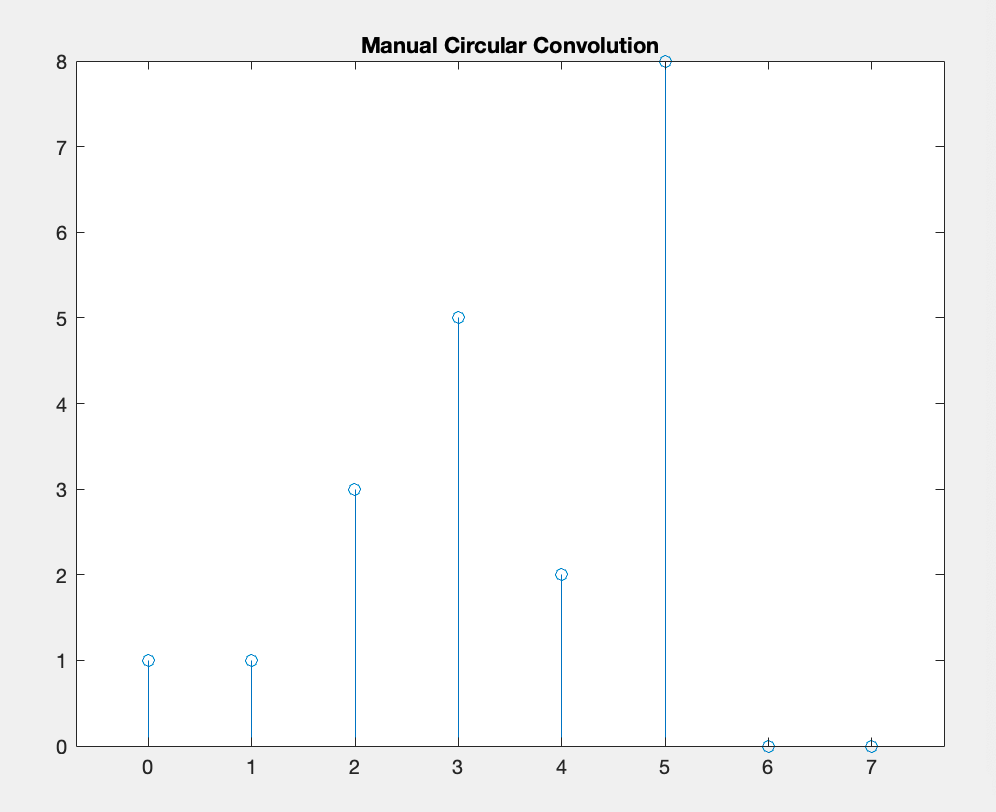
figure; stem(0:Nfft-1, circ\_builtin); title('Built-in cconv Result');

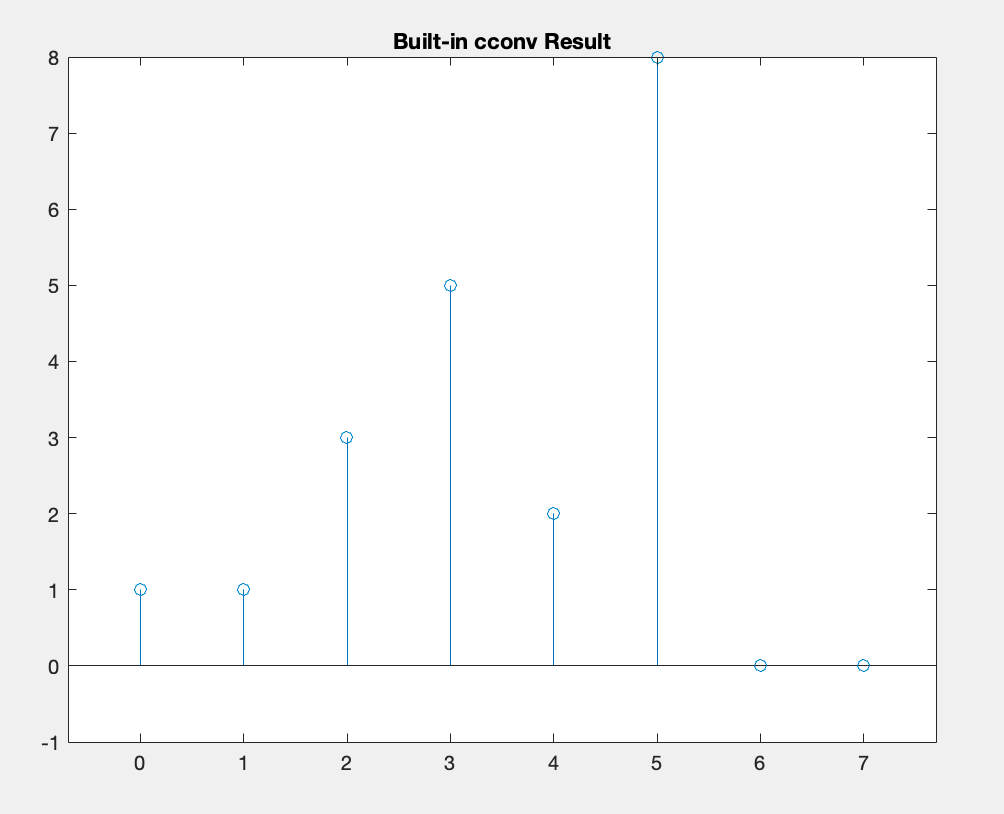
% Compare results

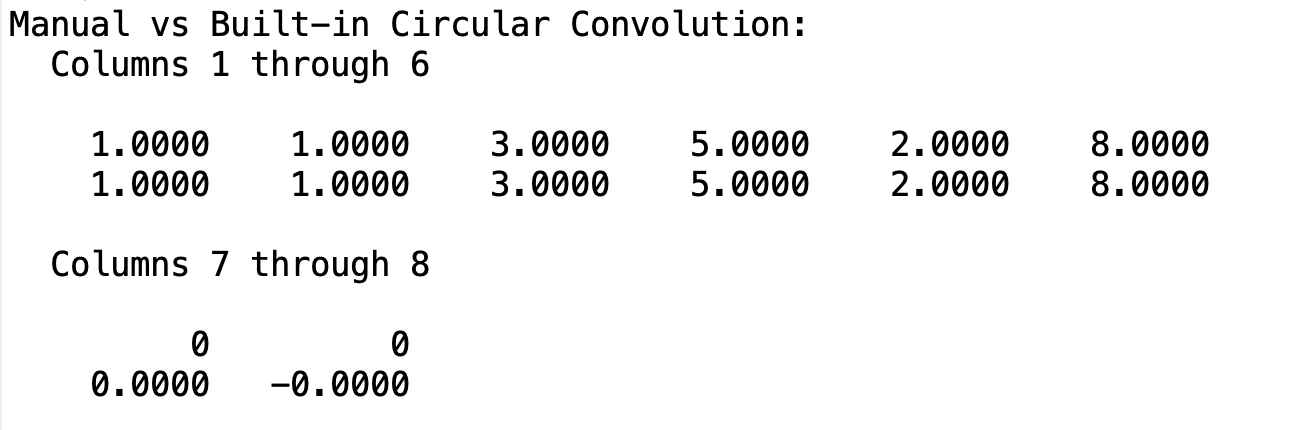
disp('Manual vs Built-in Circular Convolution:');

disp([circ; circ\_builtin]);

OUTPUT:







4. Write a MATLAB program to perform cross correlation between two sequences x(n) and h(n). Also verify the result using inbuilt functions.

CODE:

x = [1 2 3 4]; h = [1 -1 2];

% Define lag range to match built-in xcorr output

lags = -(length(x)-1):(length(x)-1);

% Manual cross-correlation over full lag range

r = zeros(1, length(lags));

for idx = 1:length(lags)

lag = lags(idx);

sumv = 0;

for n0 = 1:length(x)

m = n0 - lag;

if m >= 1 && m <= length(h)

sumv = sumv + x(n0) \* h(m);

end

end

r(idx) = sumv;

end

% Plot manual result

figure; stem(lags, r); title('Manual Cross-correlation');

% Verify using built-in xcorr()

r\_builtin = xcorr(x, h);

% Built-in lags match lags variable

% Plot built-in result

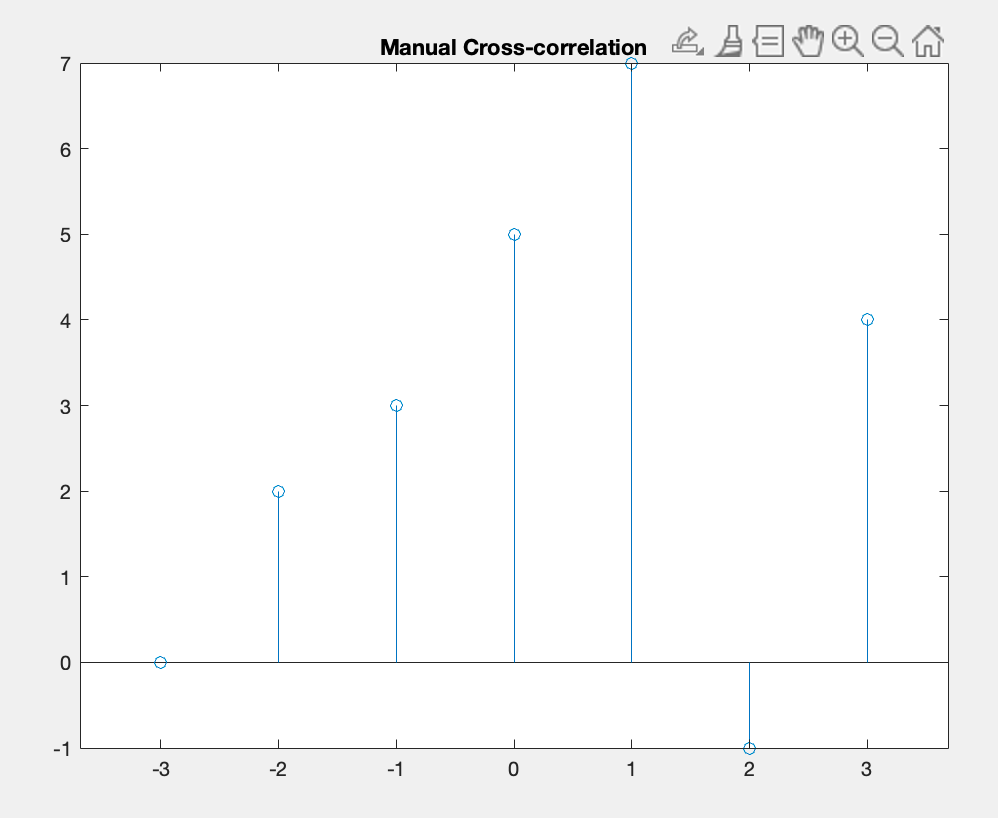
figure; stem(lags, r\_builtin); title('Built-in xcorr Result');

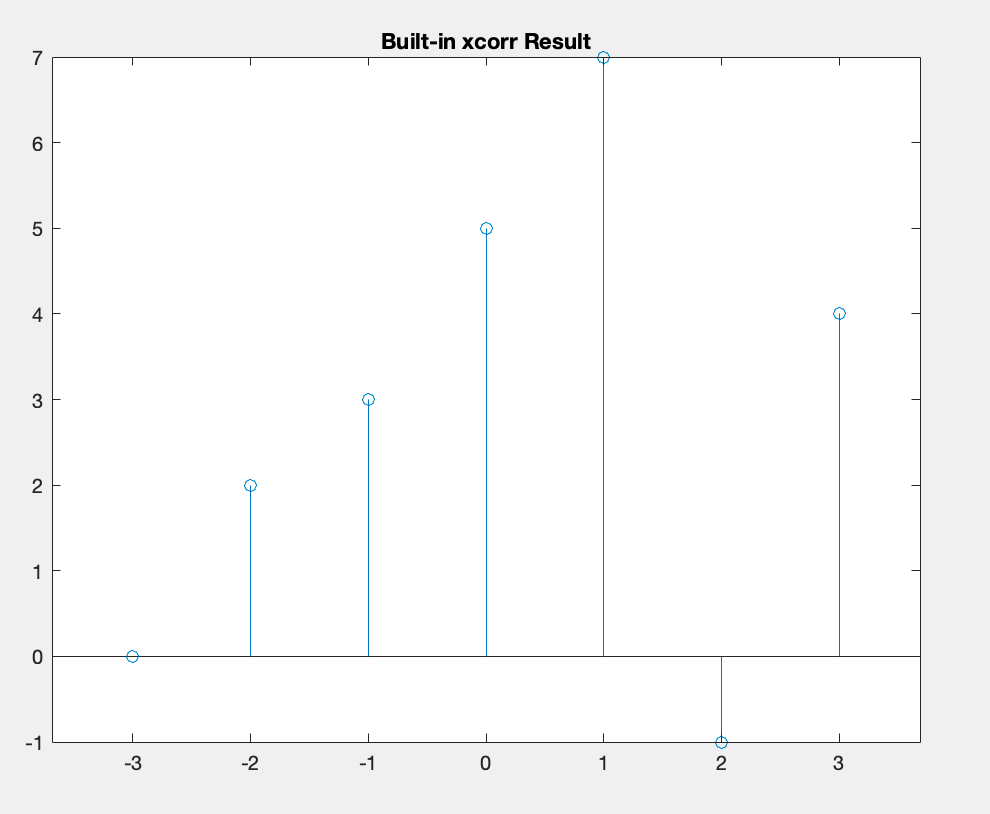
% Compare results

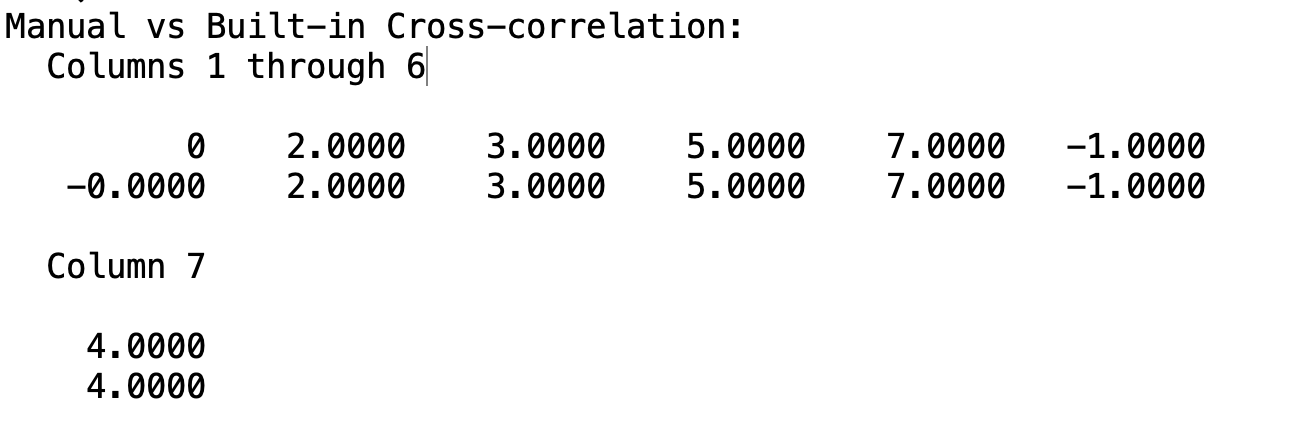
disp('Manual vs Built-in Cross-correlation:');

disp([r; r\_builtin]);

OUTPUT:







5. Compute and implement the N-point DFT of a given sequence and compute the power density spectrum of the sequence.

CODE:

% Given sequence and DFT length

x\_seq = [1 2 3 4]; N = 8;

% Manual DFT computation

X\_manual = zeros(1, N);

n = 0:N-1;

for k = 0:N-1

for nn = 0:length(x\_seq)-1

X\_manual(k+1) = X\_manual(k+1) + x\_seq(nn+1)\*exp(-1j\*2\*pi\*k\*nn/N);

end

end

P\_manual = abs(X\_manual).^2;

% Plot manual DFT magnitude and power

figure;

subplot(2,1,1); stem(0:N-1, abs(X\_manual)); title('Manual DFT Magnitude');

subplot(2,1,2); stem(0:N-1, P\_manual); title('Manual Power Density');

% Verify using built-in fft()

X\_fft = fft(x\_seq, N);

P\_fft = abs(X\_fft).^2;

figure;

subplot(2,1,1); stem(0:N-1, abs(X\_fft)); title('FFT Magnitude');

subplot(2,1,2); stem(0:N-1, P\_fft); title('FFT Power Density');

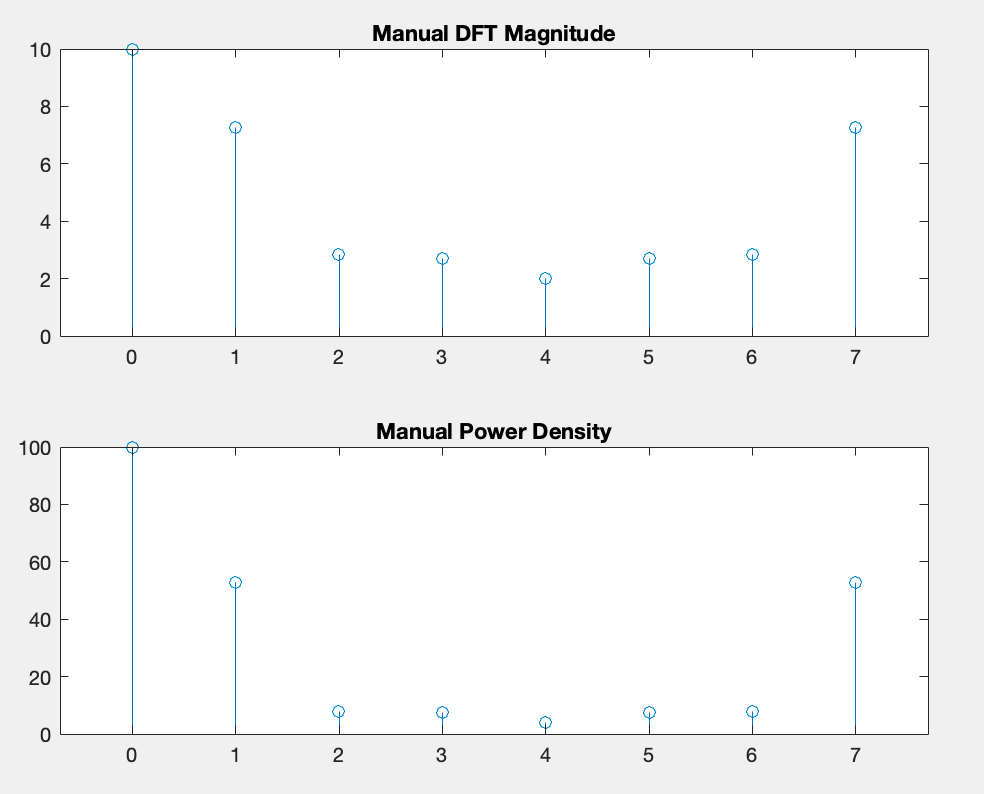
disp('Manual vs FFT magnitude and power (rows: manual; fft):');

disp([abs(X\_manual); abs(X\_fft)]);

disp([P\_manual; P\_fft]);

OUTPUT:

6. Implement and verify N-point DIT-FFT of a given sequence and find the frequency response(magnitude and phase).

CODE:

function X = my\_fft(x)

N = length(x);

if N == 1

X = x;

else

X\_even = my\_fft(x(1:2:end));

X\_odd = my\_fft(x(2:2:end));

X = zeros(1, N);

for k = 0:(N/2-1)

tw = exp(-1j \* 2\*pi \* k / N);

X(k+1) = X\_even(k+1) + tw \* X\_odd(k+1);

X(k+1+N/2) = X\_even(k+1) - tw \* X\_odd(k+1);

end

end

end

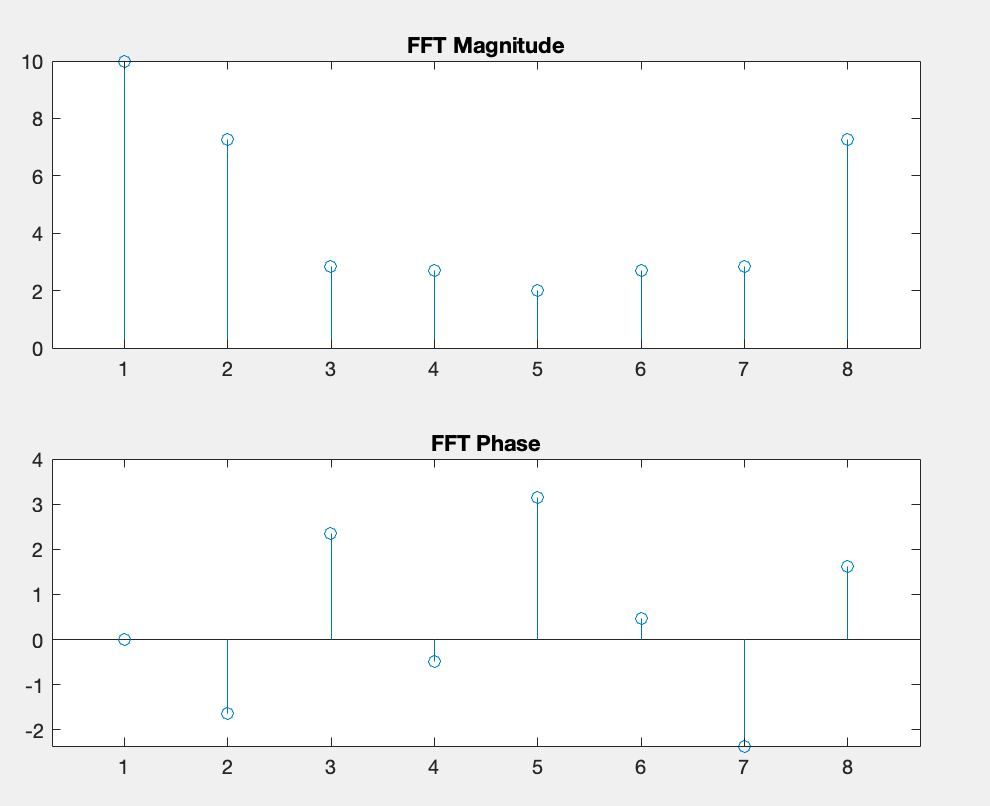
Xfft = my\_fft(x\_pad);

figure;

subplot(2,1,1); stem(abs(Xfft)); title('FFT Magnitude');

subplot(2,1,2); stem(angle(Xfft)); title('FFT Phase');

OUTPUT:



7. Implement and verify N-point IFFT of a given sequence.

CODE:

function x = my\_ifft(X)

N = length(X);

% Preallocate output

x = zeros(1, N);

% Compute inverse DFT manually

for n = 0:N-1

sum\_val = 0;

for k = 0:N-1

angle = 2\*pi\*k\*n/N;

sum\_val = sum\_val + X(k+1) \* exp(1j \* angle);

end

x(n+1) = sum\_val / N;

end

end

N = 8;

% Sample time-domain sequence

xt = [1, 2, 3, 4, 2, 1, 0, -1];

% Compute its FFT using built-in function for reference

X\_builtin = fft(xt, N);

% Compute IFFT using custom function

xt\_reconstructed = my\_ifft(X\_builtin);

% Display results

disp('Original sequence:');

disp(xt);

disp('Reconstructed sequence from custom IFFT:');

disp(real(xt\_reconstructed)); % should match original within numerical error

% Compute error

error = max(abs(xt - real(xt\_reconstructed)));

disp(['Maximum reconstruction error: ', num2str(error)]);

% Plot original vs reconstructed

figure;

stem(0:N-1, xt, 'filled'); hold on;

stem(0:N-1, real(xt\_reconstructed), 'r--');

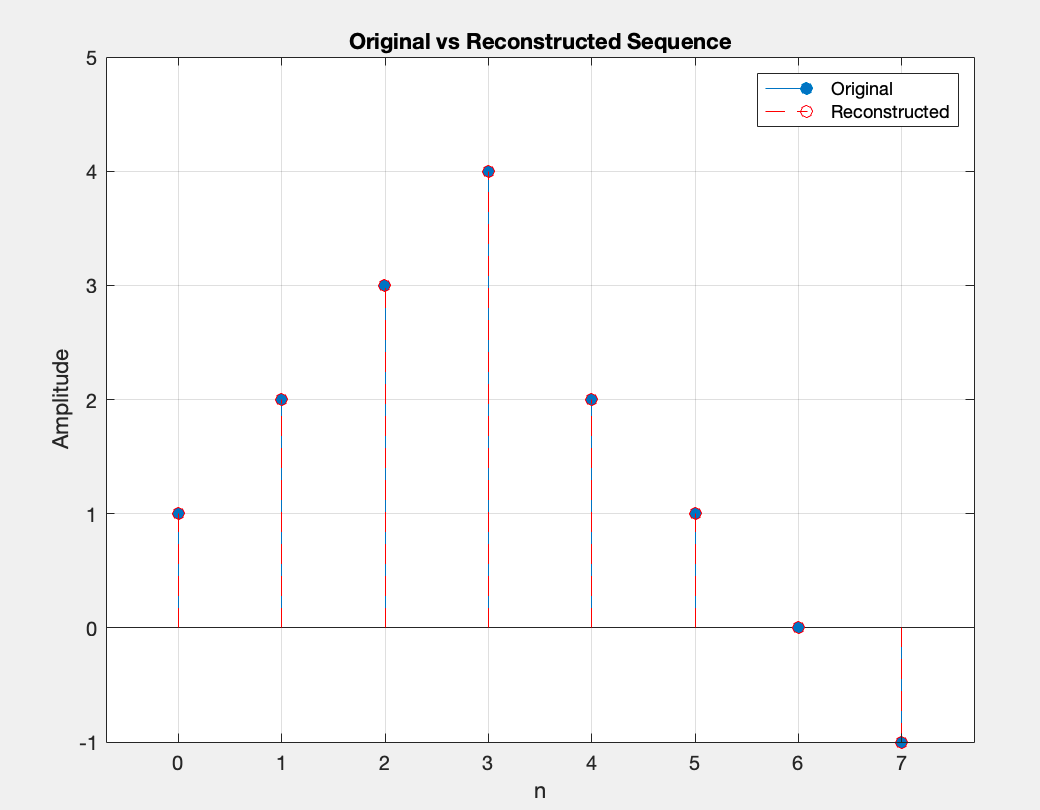
title('Original vs Reconstructed Sequence');

xlabel('n'); ylabel('Amplitude');

legend('Original', 'Reconstructed');

grid on;

OUTPUT:



8. Write a MATLAB program to generate Gaussian numbers with given mean and variance. Plot the PDF of the generated numbers.

CODE:

% Parameters (user can modify)

mu = 5; % Desired mean

sigma2 = 2; % Desired variance

N = 10000; % Number of samples to generate

numBins = 50; % Number of bins for histogram

% Generate Gaussian samples

sigma = sqrt(sigma2);

data = mu + sigma.\*randn(N,1);

% Plot empirical PDF (normalized histogram)

figure;

histogram(data, numBins, 'Normalization', 'pdf', 'EdgeColor', 'none');

hold on;

% Theoretical PDF

y = linspace(mu-4\*sigma, mu+4\*sigma, 200);

theoretical\_pdf = (1/(sqrt(2\*pi\*sigma2))) \* exp(-(y-mu).^2/(2\*sigma2));

plot(y, theoretical\_pdf, 'r', 'LineWidth', 2);

title(sprintf('Empirical vs. Theoretical PDF (\mu=%.2f, \sigma^2=%.2f)', mu, sigma2));

xlabel('Value');

ylabel('Probability Density');

legend('Empirical PDF', 'Theoretical PDF');

grid on;

% Compute and display sample statistics

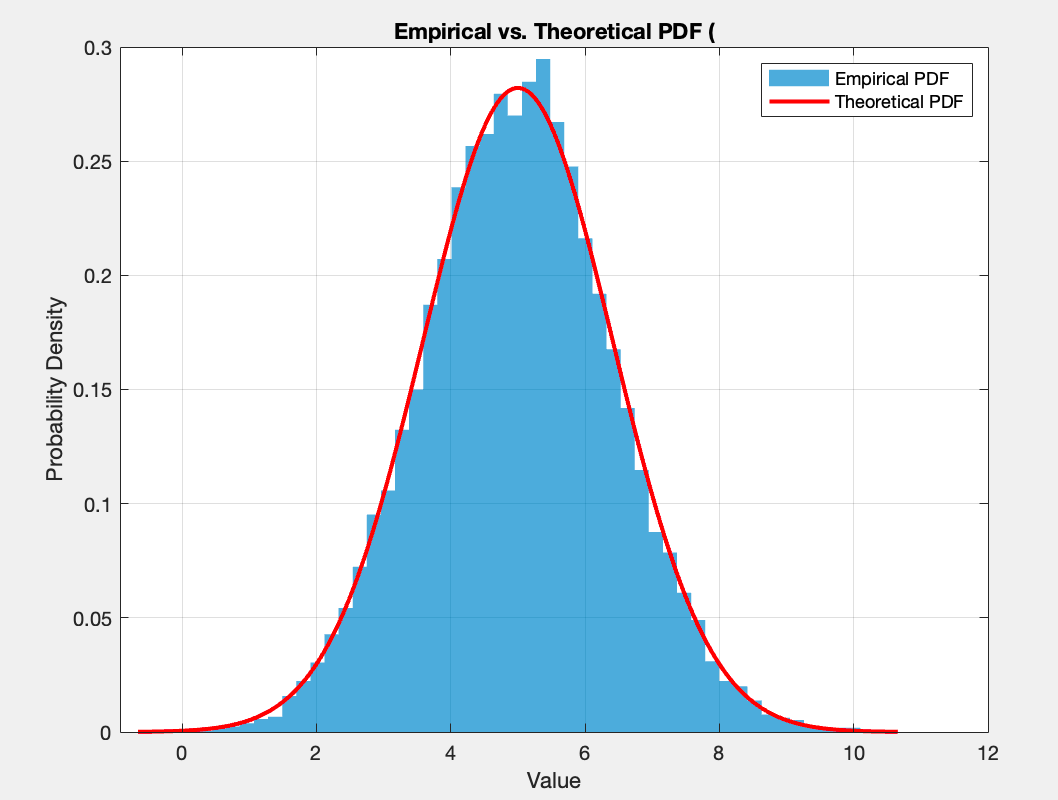
sample\_mean = mean(data);

sample\_variance = var(data);

fprintf('Desired mean: %.4f, Sample mean: %.4f\n', mu, sample\_mean);

fprintf('Desired variance: %.4f, Sample variance: %.4f\n', sigma2, sample\_variance);

OUTPUT:



9. Design a FIR lowpass filter with given specification and verify the magnitude, phase and impulse response using FDA toolbox.

CODE:

%% Specifications (modify as needed)

Fs = 1000; % Sampling frequency in Hz

Fp = 100; % Passband edge frequency in Hz

Fs2 = 150; % Stopband edge frequency in Hz

Rp = 1; % Passband ripple in dB

Rs = 60; % Stopband attenuation in dB

%% Normalize frequencies (0 to 1)

Wp = Fp/(Fs/2);

Ws = Fs2/(Fs/2);

%% Estimate filter order and design using Parks-McClellan (firpm)

% firpmord: [N, fo, ao, w] = firpmord(f, a, dev, fs)

[N, fo, ao, w] = firpmord([Fp, Fs2], [1, 0], [ (10^(Rp/20)-1)/(10^(Rp/20)+1), 10^(-Rs/20) ], Fs);

b = firpm(N, fo, ao, w);

%% Display designed filter coefficients and order

fprintf('Designed FIR lowpass filter order: %d\n', N);

disp('Filter coefficients b:'); disp(b');

%% Analyze using FDA Toolbox (fvtool)

% Magnitude response

fvtool(b,1, 'Fs', Fs, 'Analysis', 'magnitude');

% Phase response

fvtool(b,1, 'Fs', Fs, 'Analysis', 'phase');

% Impulse response

fvtool(b,1, 'Fs', Fs, 'Analysis', 'impulse');

%% Alternative visualization in figures

% Frequency response using freqz

figure;

[H, f] = freqz(b,1,1024, Fs);

subplot(2,1,1);

plot(f, 20\*log10(abs(H))); grid on;

title('Magnitude Response (dB)'); xlabel('Frequency (Hz)'); ylabel('Magnitude (dB)');

subplot(2,1,2);

plot(f, unwrap(angle(H))\*180/pi); grid on;

title('Phase Response'); xlabel('Frequency (Hz)'); ylabel('Phase (degrees)');

% Impulse response using impz

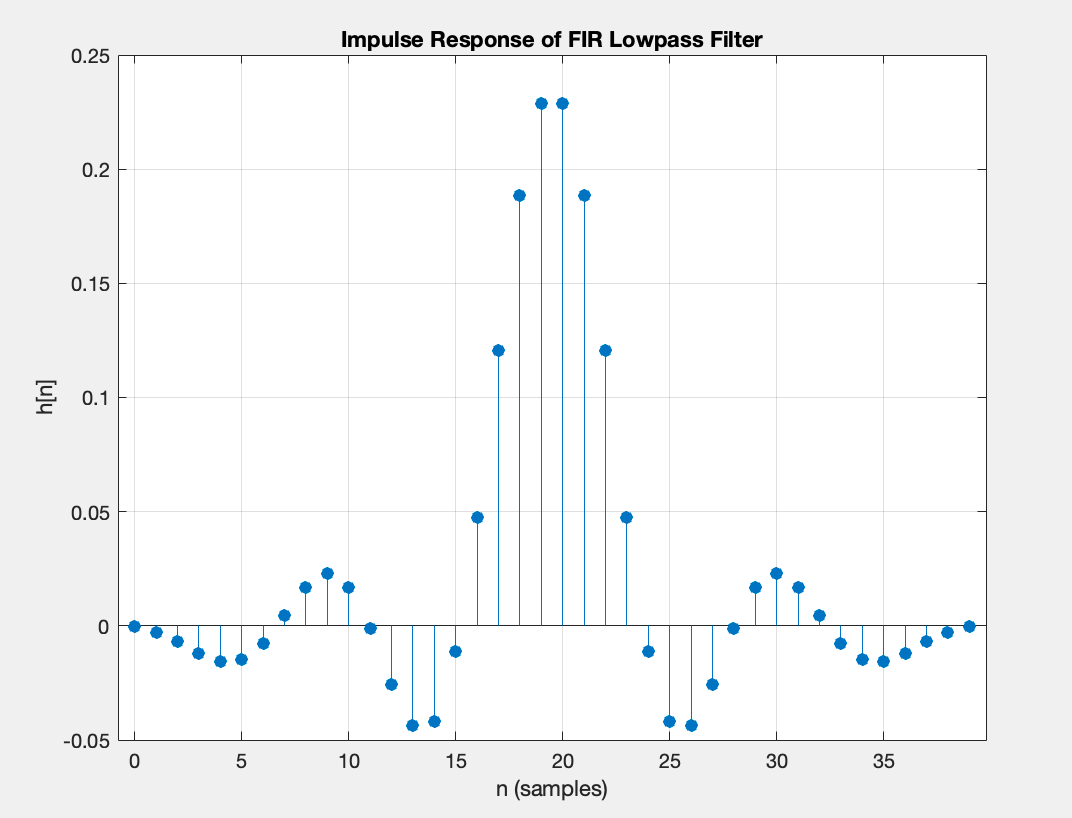
figure;

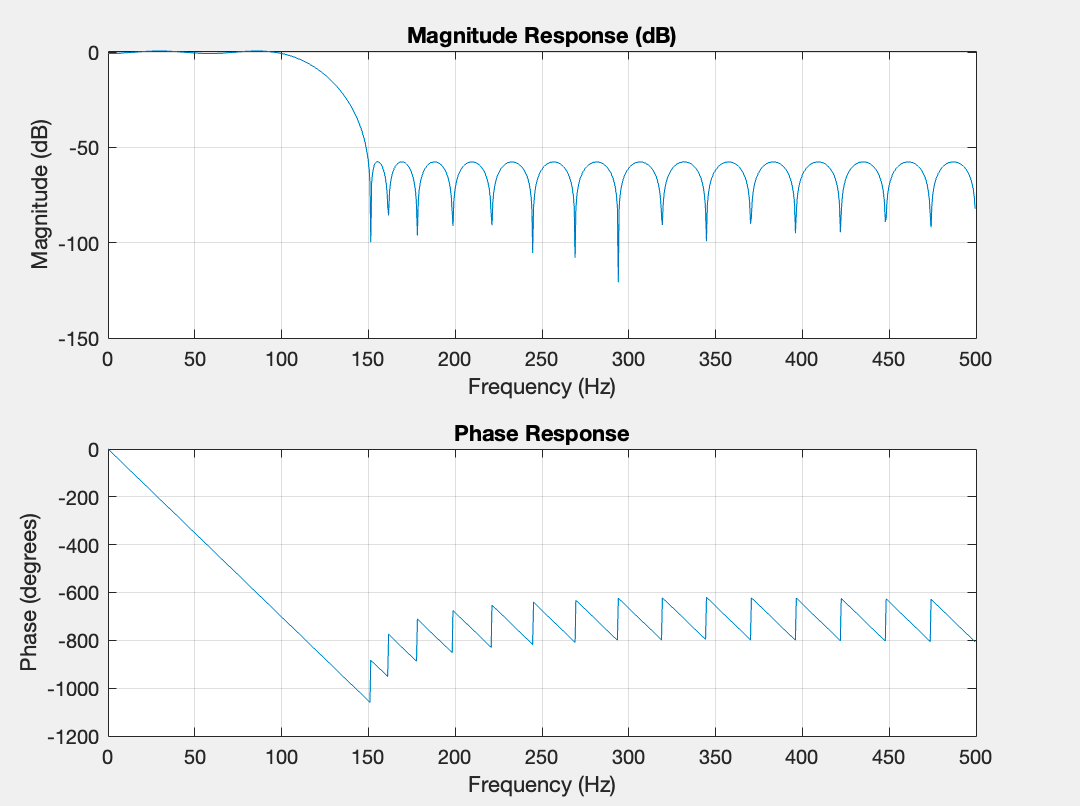
[ h\_n, n ] = impz(b,1, N+1);

stem(n, h\_n, 'filled'); grid on;

title('Impulse Response of FIR Lowpass Filter'); xlabel('n (samples)'); ylabel('h[n]');

OUTPUT:





10. Design FIR filter (Low Pass Filter /High Pass Filter) using windowing technique. Using (i) rectangular window (ii). Hamming window (iii). Kaiser window

CODE:

%% Specifications (modify as needed)

Fs = 1000; % Sampling frequency (Hz)

Fp\_lp = 100; % Lowpass passband edge (Hz)

Fs\_lp = 150; % Lowpass stopband edge (Hz)

Fp\_hp = 200; % Highpass passband edge (Hz)

Fs\_hp = 150; % Highpass stopband edge (Hz)

Rp = 1; % Passband ripple (dB)

Rs = 60; % Stopband attenuation (dB)

%% Estimate filter orders (using default Kaiser approximation)

% For rectangular & Hamming, choose N large enough manually or via fir1

% For Kaiser, use kaiserord

[n\_kaiser\_lp, Wn\_lp, beta\_lp, ftype\_lp] = kaiserord([Fp\_lp Fs\_lp], [1 0], [ (10^(Rp/20)-1)/(10^(Rp/20)+1) 10^(-Rs/20) ], Fs);

N\_kaiser\_lp = n\_kaiser\_lp;

[n\_kaiser\_hp, Wn\_hp, beta\_hp, ftype\_hp] = kaiserord([Fs\_hp Fp\_hp], [0 1], [10^(-Rs/20) (10^(Rp/20)-1)/(10^(Rp/20)+1)], Fs);

N\_kaiser\_hp = n\_kaiser\_hp;

%% Design filters using window methods

% 1. Rectangular window

N\_rect = max(N\_kaiser\_lp, N\_kaiser\_hp); % choose common order

b\_rect\_lp = fir1(N\_rect, Fp\_lp/(Fs/2), 'low', rectwin(N\_rect+1));

b\_rect\_hp = fir1(N\_rect, Fp\_hp/(Fs/2), 'high', rectwin(N\_rect+1));

% 2. Hamming window

b\_ham\_lp = fir1(N\_rect, Fp\_lp/(Fs/2), 'low', hamming(N\_rect+1));

b\_ham\_hp = fir1(N\_rect, Fp\_hp/(Fs/2), 'high', hamming(N\_rect+1));

% 3. Kaiser window

b\_kais\_lp = fir1(N\_kaiser\_lp, Wn\_lp, 'low', kaiser(N\_kaiser\_lp+1, beta\_lp));

b\_kais\_hp = fir1(N\_kaiser\_hp, Wn\_hp, 'high', kaiser(N\_kaiser\_hp+1, beta\_hp));

%% Plot responses

filters = {b\_rect\_lp, b\_ham\_lp, b\_kais\_lp; b\_rect\_hp, b\_ham\_hp, b\_kais\_hp};

types = {'Lowpass', 'Highpass'};

winNames= {'Rectangular', 'Hamming', 'Kaiser'};

for t = 1:2

figure('Name', [types{t} ' Filter Responses']);

for w = 1:3

b = filters{t, w};

[H, f] = freqz(b,1,1024, Fs);

subplot(3,1,w);

plot(f, 20\*log10(abs(H))); grid on;

title(sprintf('%s Window %s Filter (Order %d)', winNames{w}, types{t}, length(b)-1));

xlabel('Frequency (Hz)'); ylabel('Magnitude (dB)');

ylim([-100 5]);

end

xlabel('Frequency (Hz)');

end

%% Impulse responses comparison (Lowpass)

figure('Name', 'Impulse Responses - Lowpass'); hold on;

stem(0:length(b\_rect\_lp)-1, b\_rect\_lp, 'o');

stem(0:length(b\_ham\_lp)-1, b\_ham\_lp, 'x');

stem(0:length(b\_kais\_lp)-1, b\_kais\_lp, 's');

title('Impulse Responses: Lowpass Filters');

xlabel('n (samples)'); ylabel('h[n]');

legend(winNames); grid on;

%% Impulse responses comparison (Highpass)

figure('Name', 'Impulse Responses - Highpass'); hold on;

stem(0:length(b\_rect\_hp)-1, b\_rect\_hp, 'o');

stem(0:length(b\_ham\_hp)-1, b\_ham\_hp, 'x');

stem(0:length(b\_kais\_hp)-1, b\_kais\_hp, 's');

title('Impulse Responses: Highpass Filters');

xlabel('n (samples)'); ylabel('h[n]');

legend(winNames); grid on;

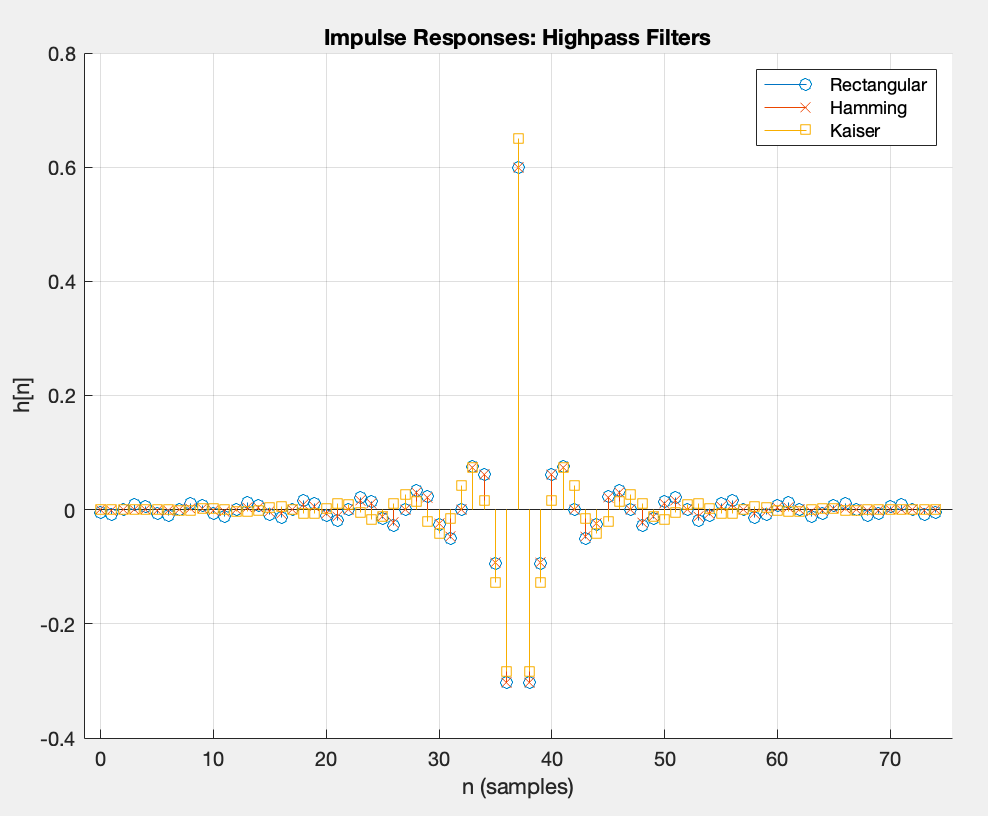
%% Display orders and beta values

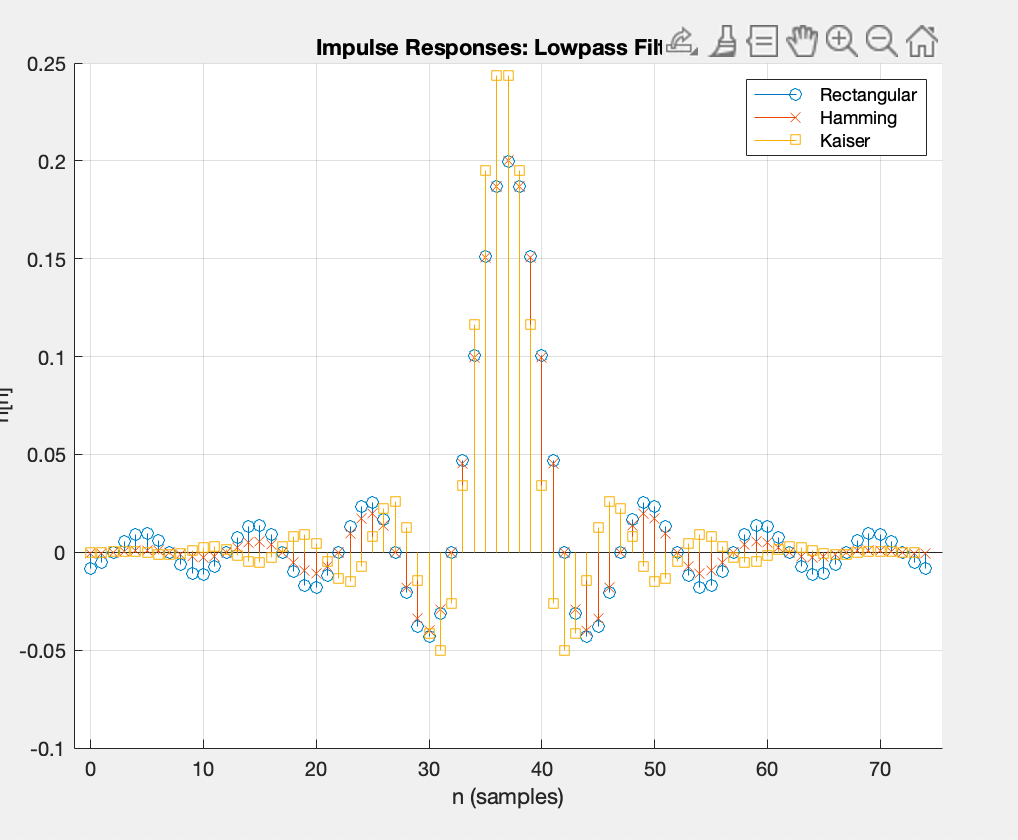
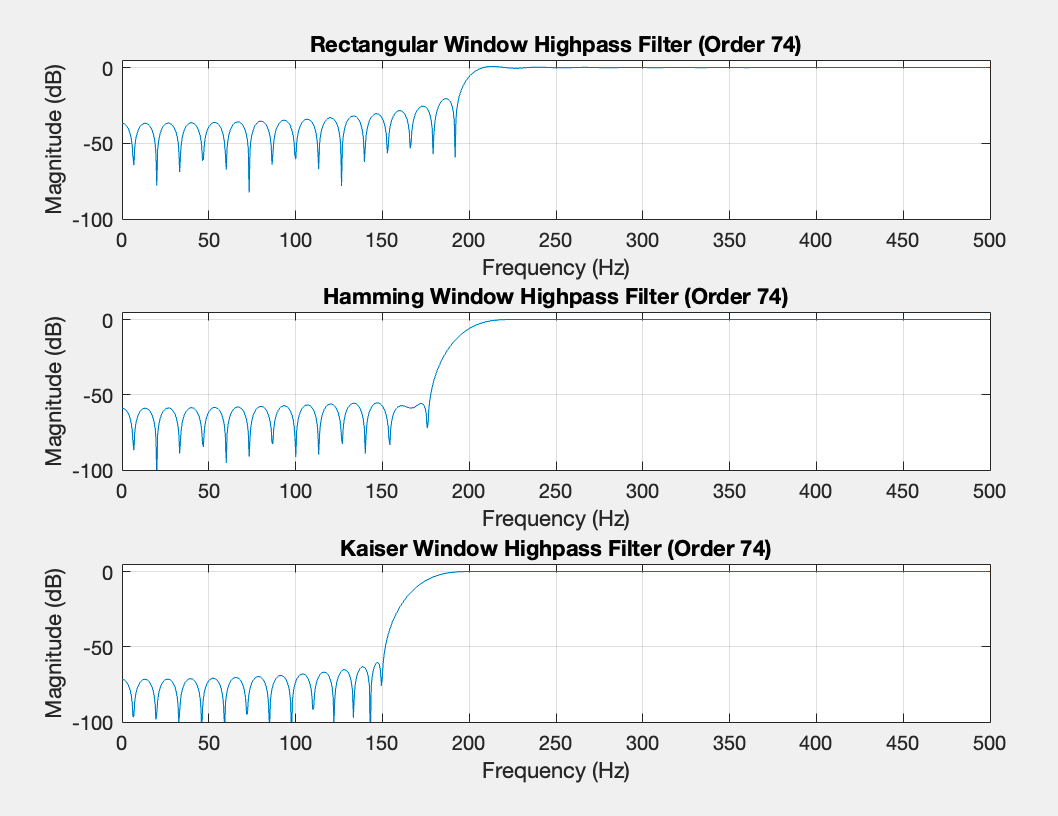
fprintf('Rectangular window filter order: %d\n', N\_rect);

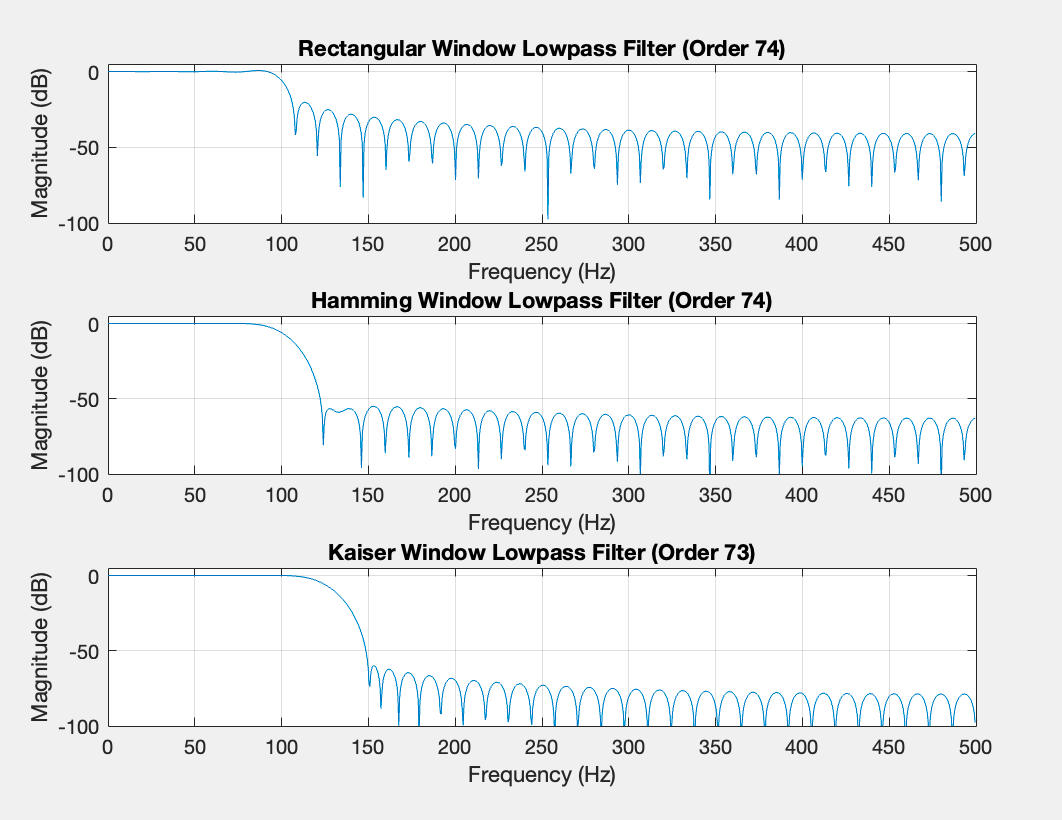
fprintf('Kaiser lowpass order: %d, beta=%.2f\n', N\_kaiser\_lp, beta\_lp);

fprintf('Kaiser highpass order: %d, beta=%.2f\n', N\_kaiser\_hp, beta\_hp);

OUTPUT:







11. Design a IIR lowpass Butterworth filter with following specification and verify magnitude, phase and impulse response using FDA tool.

CODE:

%% Specifications (modify as needed)

Fs = 1000; % Sampling frequency (Hz)

Fp\_lp = 100; % Lowpass passband edge (Hz)

Fs\_lp = 150; % Lowpass stopband edge (Hz)

Fp\_hp = 200; % Highpass passband edge (Hz)

Fs\_hp = 150; % Highpass stopband edge (Hz)

Rp = 1; % Passband ripple (dB)

Rs = 60; % Stopband attenuation (dB)

%% FIR: Estimate Kaiser filter orders

[n\_kaiser\_lp, Wn\_lp, beta\_lp, ftype\_lp] = kaiserord([Fp\_lp Fs\_lp], [1 0], [(10^(Rp/20)-1)/(10^(Rp/20)+1) 10^(-Rs/20)], Fs);

N\_kaiser\_lp = n\_kaiser\_lp;

[n\_kaiser\_hp, Wn\_hp, beta\_hp, ftype\_hp] = kaiserord([Fs\_hp Fp\_hp], [0 1], [10^(-Rs/20) (10^(Rp/20)-1)/(10^(Rp/20)+1)], Fs);

N\_kaiser\_hp = n\_kaiser\_hp;

%% FIR: Design filters using window methods

N\_rect = max(N\_kaiser\_lp, N\_kaiser\_hp);

% Rectangular

b\_rect\_lp = fir1(N\_rect, Fp\_lp/(Fs/2), 'low', rectwin(N\_rect+1));

b\_rect\_hp = fir1(N\_rect, Fp\_hp/(Fs/2), 'high', rectwin(N\_rect+1));

% Hamming

b\_ham\_lp = fir1(N\_rect, Fp\_lp/(Fs/2), 'low', hamming(N\_rect+1));

b\_ham\_hp = fir1(N\_rect, Fp\_hp/(Fs/2), 'high', hamming(N\_rect+1));

% Kaiser

b\_kais\_lp = fir1(N\_kaiser\_lp, Wn\_lp, 'low', kaiser(N\_kaiser\_lp+1, beta\_lp));

b\_kais\_hp = fir1(N\_kaiser\_hp, Wn\_hp, 'high', kaiser(N\_kaiser\_hp+1, beta\_hp));

%% FIR: Plot magnitude responses

filters = {b\_rect\_lp, b\_ham\_lp, b\_kais\_lp; b\_rect\_hp, b\_ham\_hp, b\_kais\_hp};

types = {'Lowpass', 'Highpass'};

winNames= {'Rectangular', 'Hamming', 'Kaiser'};

for t = 1:2

figure('Name', [types{t} ' FIR Responses']);

for w = 1:3

b = filters{t, w}; [H,f] = freqz(b,1,1024,Fs);

subplot(3,1,w);

plot(f,20\*log10(abs(H))); grid on;

title(sprintf('%s Window %s FIR (Order %d)',winNames{w},types{t},length(b)-1));

xlabel('Frequency (Hz)'); ylabel('Magnitude (dB)'); ylim([-100 5]);

end

end

%% FIR: Impulse responses comparison

figure('Name','Impulse Responses - FIR Lowpass'); hold on;

stem(0:length(b\_rect\_lp)-1,b\_rect\_lp,'o');

stem(0:length(b\_ham\_lp)-1,b\_ham\_lp,'x');

stem(0:length(b\_kais\_lp)-1,b\_kais\_lp,'s');

title('FIR Lowpass Impulse Responses'); xlabel('n'); ylabel('h[n]'); legend(winNames); grid on;

figure('Name','Impulse Responses - FIR Highpass'); hold on;

stem(0:length(b\_rect\_hp)-1,b\_rect\_hp,'o');

stem(0:length(b\_ham\_hp)-1,b\_ham\_hp,'x');

stem(0:length(b\_kais\_hp)-1,b\_kais\_hp,'s');

title('FIR Highpass Impulse Responses'); xlabel('n'); ylabel('h[n]'); legend(winNames); grid on;

%% FIR: Display orders and beta

fprintf('Rectangular FIR order: %d\n',N\_rect);

fprintf('Kaiser LP order: %d, beta=%.2f\n',N\_kaiser\_lp,beta\_lp);

fprintf('Kaiser HP order: %d, beta=%.2f\n',N\_kaiser\_hp,beta\_hp);

%% IIR: Butterworth Lowpass Filter Design

% Specifications (modify as needed)

Fp\_iir = 100; % Passband edge (Hz)

Fs\_iir = 150; % Stopband edge (Hz)

Rp\_iir = 1; % Passband ripple (dB)

Rs\_iir = 60; % Stopband attenuation (dB)

% Normalize

Wp = Fp\_iir/(Fs/2);

Ws = Fs\_iir/(Fs/2);

% Determine minimum order and cutoff

[n\_butt, Wn\_butt] = buttord(Wp, Ws, Rp\_iir, Rs\_iir);

[b\_iir, a\_iir] = butter(n\_butt, Wn\_butt, 'low');

% Display IIR order and coefficients

fprintf('\nButterworth IIR Lowpass order: %d\n', n\_butt);

fprintf('Numerator b\_iir: '); disp(b\_iir);

fprintf('Denominator a\_iir: '); disp(a\_iir);

%% IIR: FDA Toolbox analysis

fvtool(b\_iir, a\_iir, 'Fs', Fs, 'Analysis', 'magnitude');

fvtool(b\_iir, a\_iir, 'Fs', Fs, 'Analysis', 'phase');

fvtool(b\_iir, a\_iir, 'Fs', Fs, 'Analysis', 'impulse');

%% IIR: Alternative plots

figure('Name','IIR Magnitude & Phase');

[H\_iir,f\_iir] = freqz(b\_iir,a\_iir,1024,Fs);

subplot(2,1,1);

plot(f\_iir,20\*log10(abs(H\_iir))); grid on; title('IIR Magnitude (dB)'); xlabel('Frequency (Hz)'); ylabel('Mag (dB)');

subplot(2,1,2);

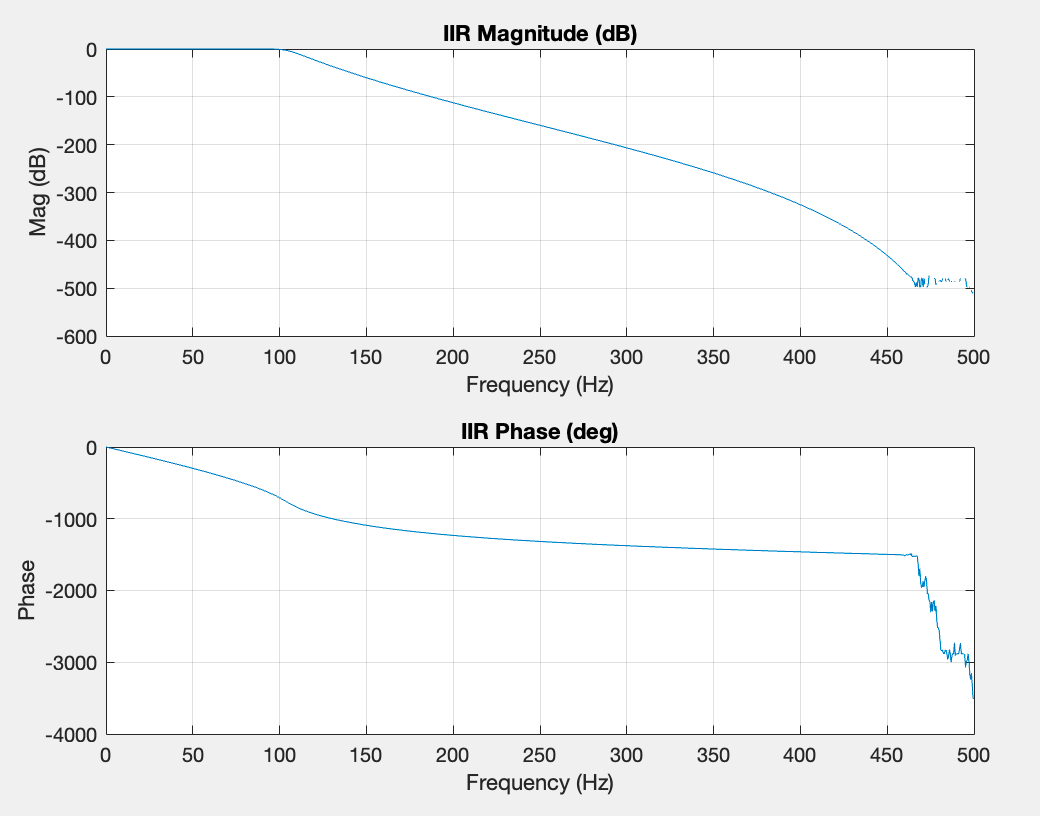
plot(f\_iir,unwrap(angle(H\_iir))\*180/pi); grid on; title('IIR Phase (deg)'); xlabel('Frequency (Hz)'); ylabel('Phase');

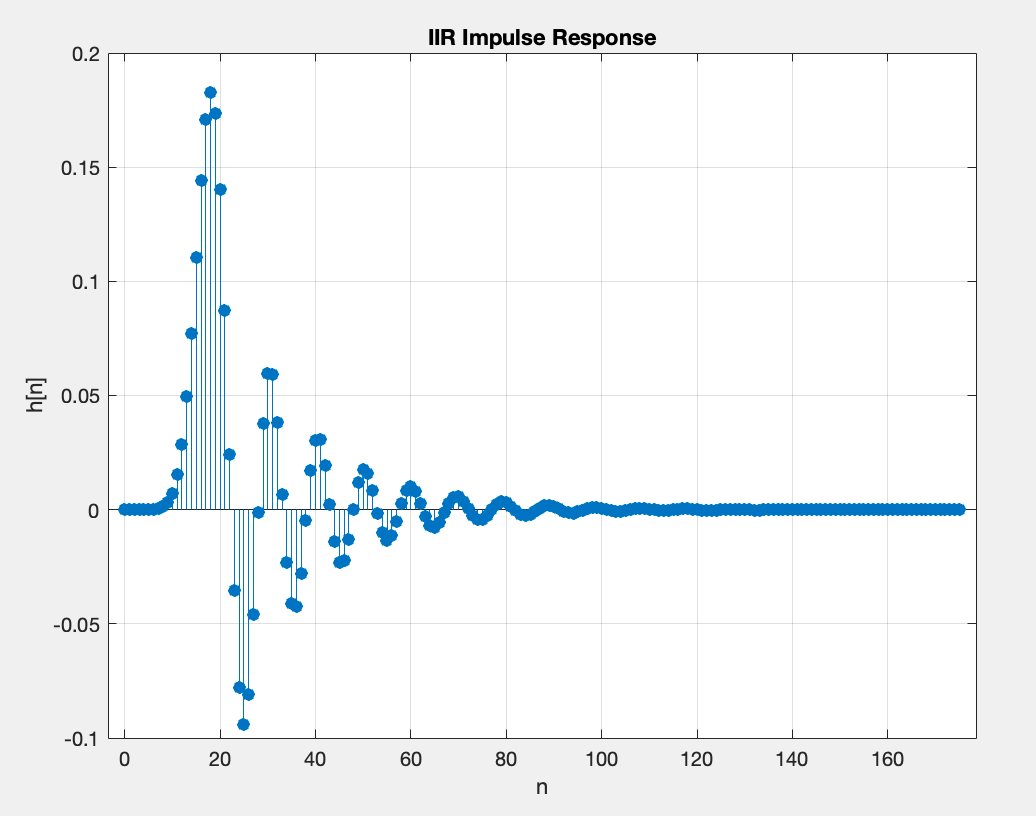
figure('Name','IIR Impulse Response');

[hiir,n\_iir] = impz(b\_iir,a\_iir);

stem(n\_iir, hiir, 'filled'); grid on; title('IIR Impulse Response'); xlabel('n'); ylabel('h[n]');

OUTPUT:





12. Write a MATLAB program to perform linear convolution of two sequences using overlap and add method.

CODE:

%% Input sequences (modify as needed)

x = [1 2 3 4 2 1]; % Input signal

h = [1 -1 2]; % Impulse response (filter)

%% Parameters

L = 4; % Block length for processing (choose L < length(x))

%% Compute lengths and FFT size

Nx = length(x); % Length of input signal

Nh = length(h); % Length of impulse response

N = L + Nh - 1; % FFT length for each block

%% Precompute FFT of impulse response

h\_pad = [h, zeros(1, N - Nh)]; % Zero-pad h to length N

H = fft(h\_pad); % FFT of padded impulse response

%% Initialize output

y = zeros(1, Nx + Nh - 1); % Output vector for full linear convolution

%% Overlap-Add processing

for start = 1:L:Nx

% Determine current block

stop = min(start + L - 1, Nx);

x\_block = x(start:stop);

M = length(x\_block); % Actual block length

% Zero-pad block to length N

x\_pad = [x\_block, zeros(1, N - M)];

% FFT-based convolution for this block

Y\_block = ifft(fft(x\_pad) .\* H);

% Only first M+Nh-1 samples are valid

Lblock = M + Nh - 1;

y(start:start + Lblock - 1) = y(start:start + Lblock - 1) + real(Y\_block(1:Lblock));

end

%% Reference using built-in conv()

y\_ref = conv(x, h);

%% Display and error check

disp('Output from Overlap-Add method:'); disp(y);

disp('Reference output using conv():'); disp(y\_ref);

fprintf('Maximum absolute error: %.2e\n', max(abs(y - y\_ref)));

%% Plot comparison

n = 0:length(y)-1;

figure;

stem(n, y, 'b', 'filled'); hold on;

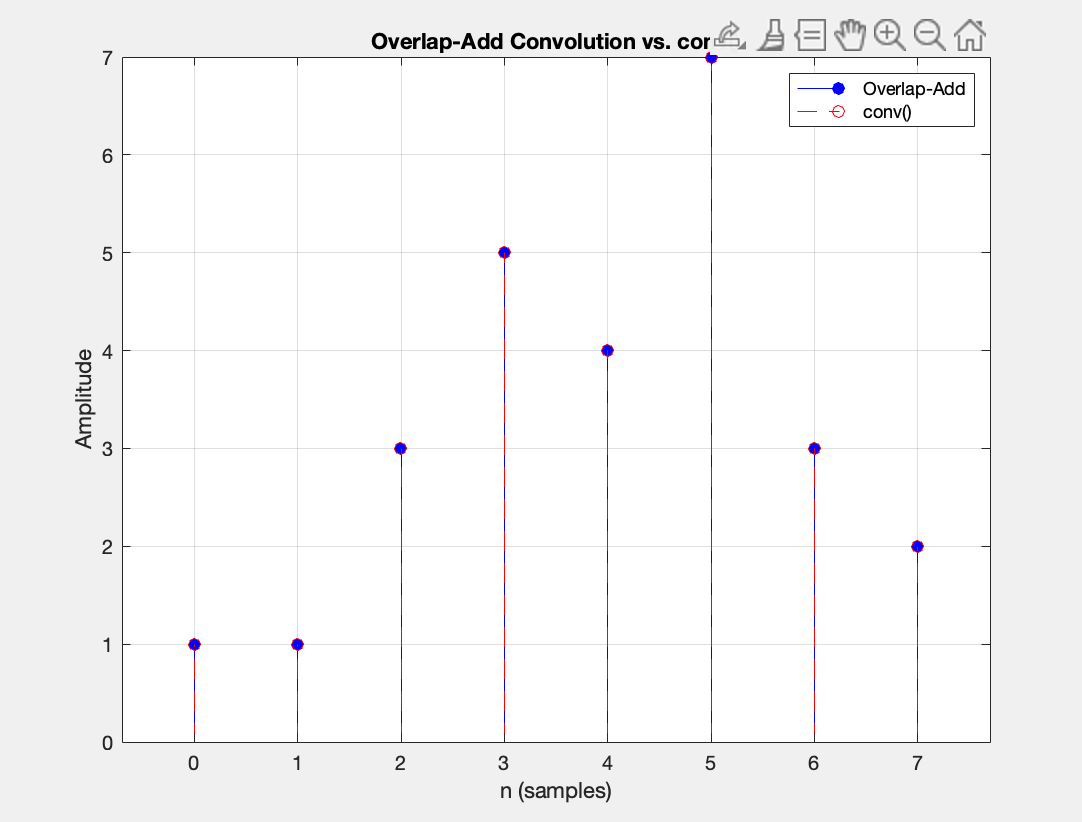
stem(n, y\_ref, 'r--');

title('Overlap-Add Convolution vs. conv()');

xlabel('n (samples)'); ylabel('Amplitude');

legend('Overlap-Add','conv()'); grid on;

OUTPUT:



13. Compute the decimation and interpolation for the given signal

CODE:

%% Input signal (modify as needed)

x = [1 2 3 4 2 1 0 -1 2 3]; % Example input signal

n = 0:length(x)-1;

%% Decimation parameters

M = 3; % Decimation factor (keep 1 out of M samples)

% Perform decimation (down-sampling)

y\_dec = x(1:M:end);

n\_dec = n(1:M:end);

%% Interpolation parameters

L = 4; % Interpolation factor (insert L-1 zeros between samples)

% Perform interpolation (up-sampling)

x\_up = zeros(1, L\*length(x));

x\_up(1:L:end) = x;

n\_up = 0:length(x\_up)-1;

%% Display results

fprintf('Original signal length: %d\n', length(x));

fprintf('Decimated signal length (M=%d): %d\n', M, length(y\_dec));

fprintf('Interpolated signal length (L=%d): %d\n', L, length(x\_up));

%% Plot signals

figure;

subplot(3,1,1);

stem(n, x, 'filled');

title('Original Signal'); xlabel('n'); ylabel('x[n]'); grid on;

subplot(3,1,2);

stem(n\_dec, y\_dec, 'r','filled');

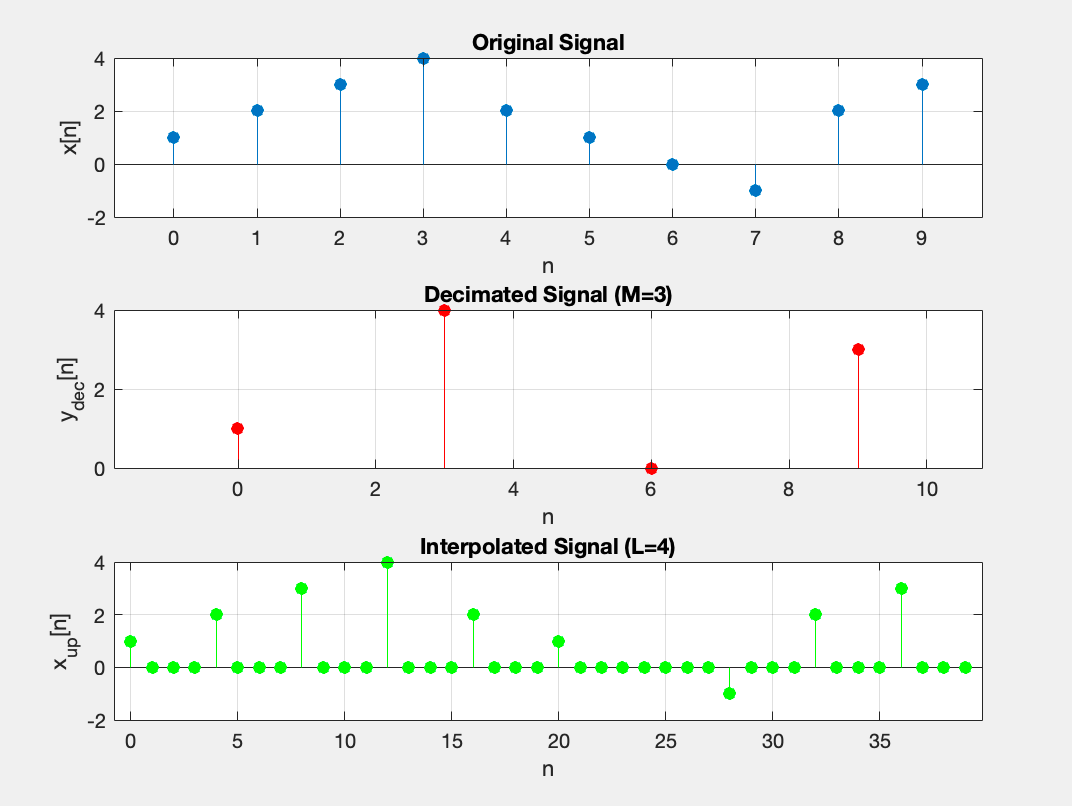
title(sprintf('Decimated Signal (M=%d)', M)); xlabel('n'); ylabel('y\_{dec}[n]'); grid on;

subplot(3,1,3);

stem(n\_up, x\_up, 'g','filled');

title(sprintf('Interpolated Signal (L=%d)', L)); xlabel('n'); ylabel('x\_{up}[n]'); grid on;

OUTPUT:



14. Impulse response of first order and second order systems.

CODE:

% Computes and plots impulse responses of 1st‑ and 2nd‑order LTI systems

clear; close all; clc;

%% Common settings

K = 2; % system gain

t\_end = 5; % end time for simulation (s)

dt = 0.001; % time step (s)

t = 0:dt:t\_end; % time vector

%% 1) First‑Order System

tau = 1.2; % time constant (s)

h1 = (K/tau) \* exp(-t/tau); % h1(t) = (K/τ)e^{-t/τ}

%% 2) Second‑Order System

wn = 4.0; % natural frequency (rad/s)

zeta\_vals = [0.5, 1, 1.5]; % [underdamped, critical, overdamped]

h2 = struct(); % will hold each case

for i = 1:length(zeta\_vals)

zeta = zeta\_vals(i);

if zeta < 1 % underdamped

wd = wn \* sqrt(1 - zeta^2);

h = (K\*wn / sqrt(1 - zeta^2)) \* exp(-zeta\*wn\*t) .\* sin(wd\*t);

elseif zeta == 1 % critically damped

h = K \* wn^2 \* t .\* exp(-wn\*t);

else % overdamped (zeta > 1)

s1 = -zeta\*wn + wn\*sqrt(zeta^2 - 1);

s2 = -zeta\*wn - wn\*sqrt(zeta^2 - 1);

h = K \* wn^2 \* (exp(s1\*t) - exp(s2\*t)) ./ (s1 - s2);

end

h2(i).zeta = zeta;

h2(i).h = h;

end

%% Plotting

figure('Position',[100 100 800 600]);

% First order

subplot(2,1,1)

plot(t, h1, 'LineWidth',1.5)

grid on

title(sprintf('Impulse Response: First‑Order (\\tau=%.2f, K=%.1f)', tau, K))

xlabel('Time (s)')

ylabel('h\_1(t)')

% Second order

subplot(2,1,2)

hold on

for i = 1:length(h2)

plot(t, h2(i).h, 'LineWidth',1.2, ...

'DisplayName', sprintf('\\zeta=%.1f', h2(i).zeta));

end

grid on

title(sprintf('Impulse Response: Second‑Order (\\omega\_n=%.1f, K=%.1f)', wn, K))

xlabel('Time (s)')

ylabel('h\_2(t)')

legend('Location','Northeast')

hold off

OUTPUT:

