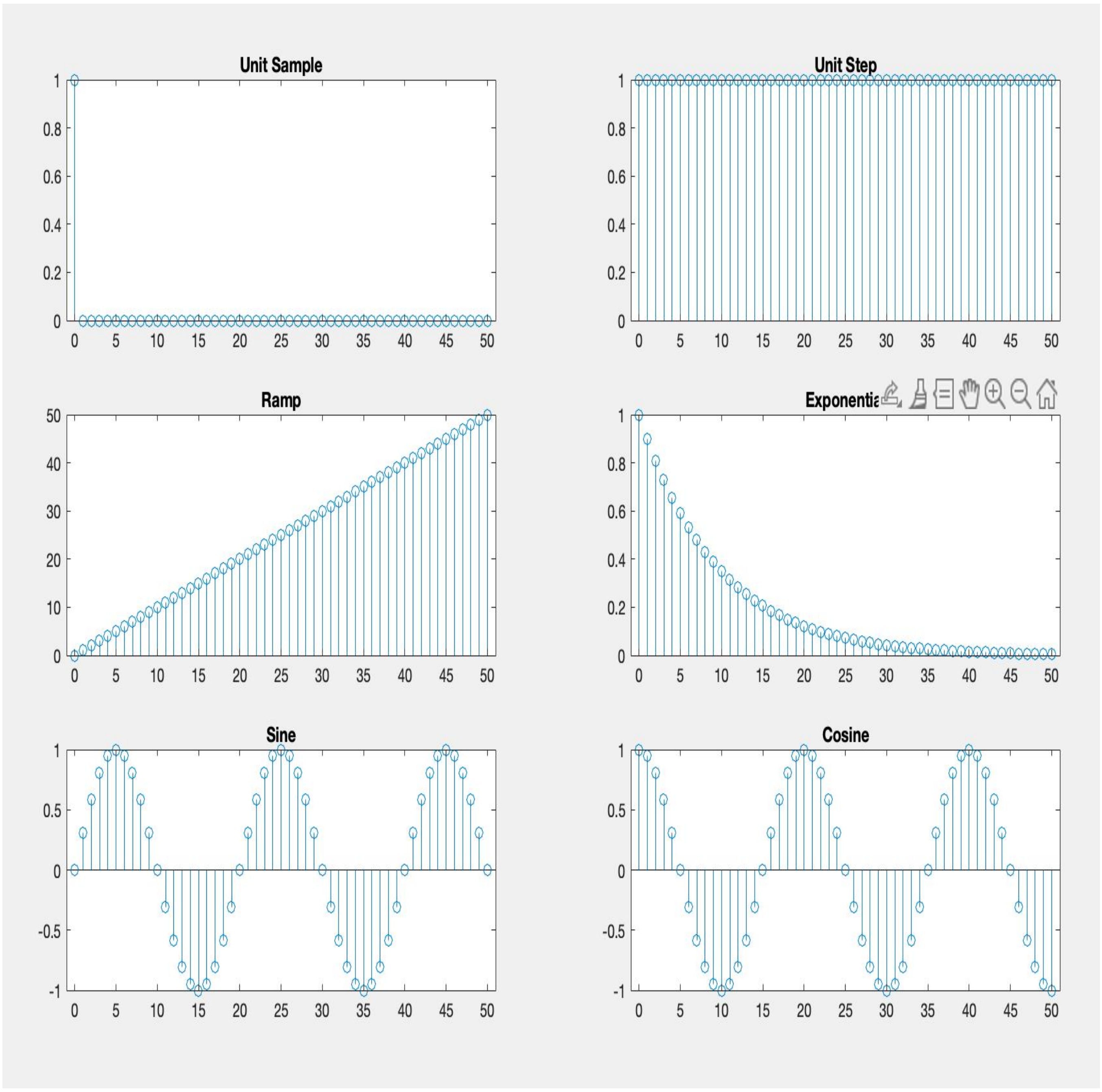
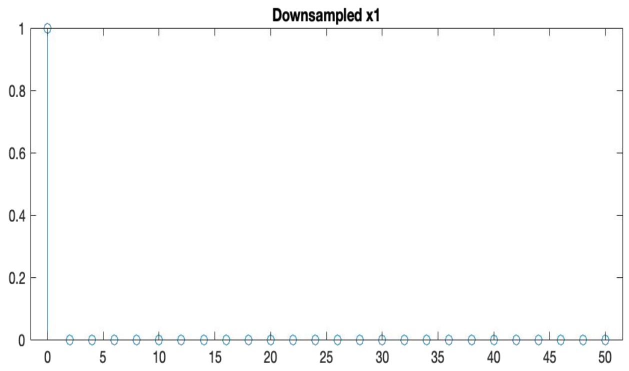
1. Plot the following sequences (i) Unit sample sequence, (ii) Unit step sequence, (iii) Ramp sequence (iv) Exponential sequence (v) sine sequence (vi) cosine sequence. Also, down sample each of the above sequences and plot.

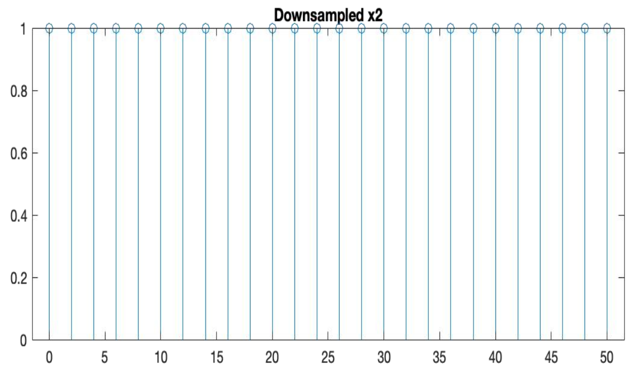
CODE:

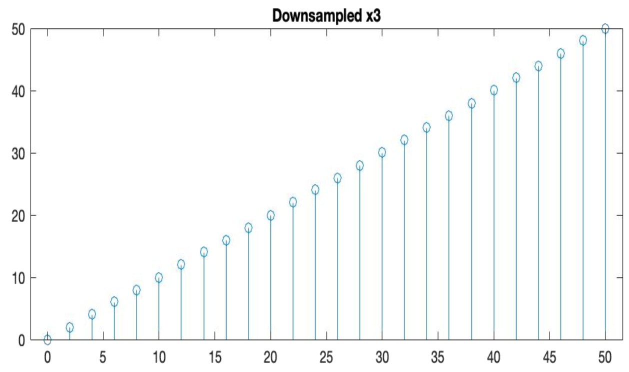
```
n = 0:50;
% (i) Unit sample
x1 = zeros(size(n)); x1(n==0) = 1;
% (ii) Unit step
x2 = double(n>=0);
% (iii) Ramp
x3 = n;
% (iv) Exponential (e.g., a^n with a=0.9)
a = 0.9; x4 = a.^n;
% (v) Sine
f = 0.05; x5 = sin(2*pi*f*n);
% (vi) Cosine
x6 = cos(2*pi*f*n);

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
```

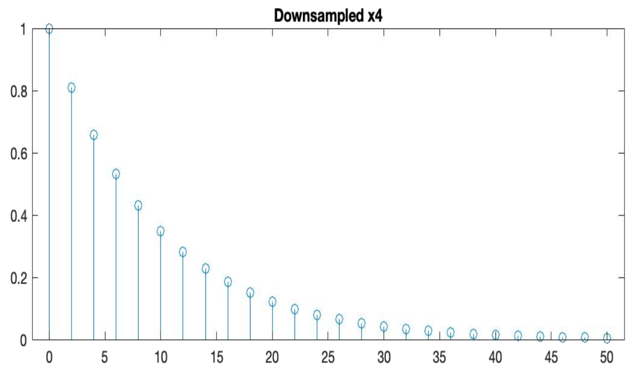
OUTPUT:

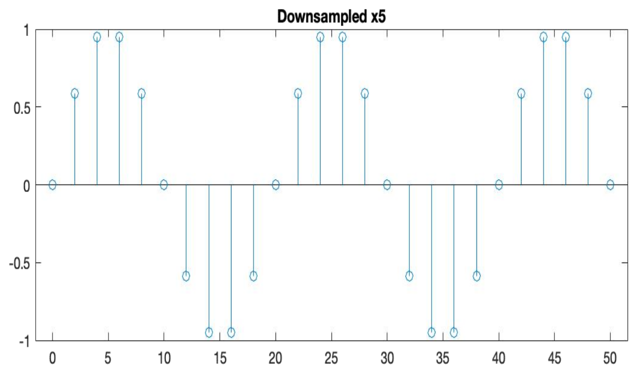

Unit Sample
Unit Step
Ramp
Exponentia
Sine
Cosine

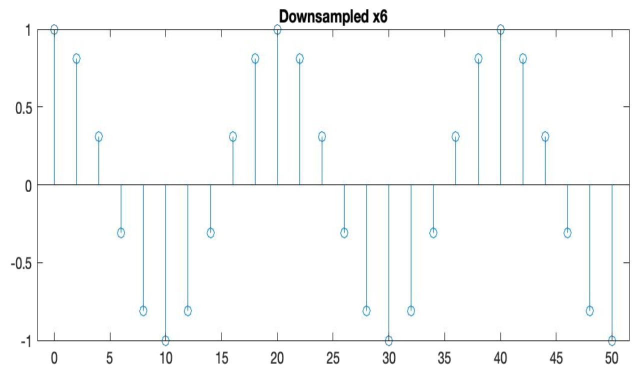

Downsampled x1


Downsampled x2


Downsampled x3


Downsampled x4


Downsampled x5

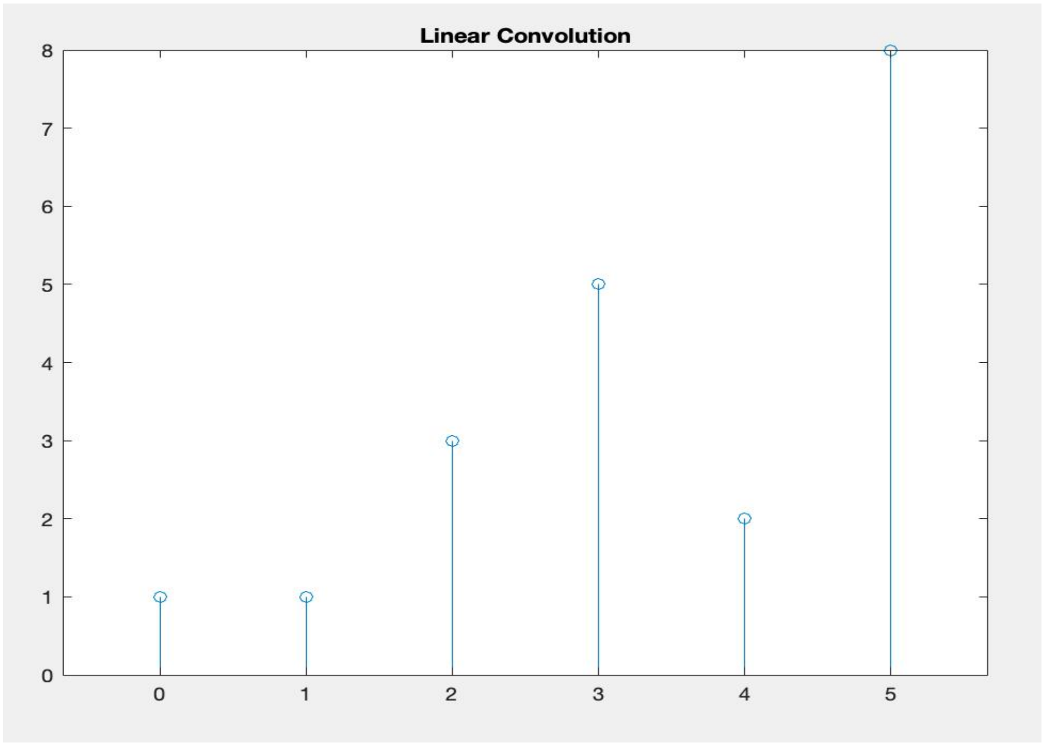

Downsampled x6

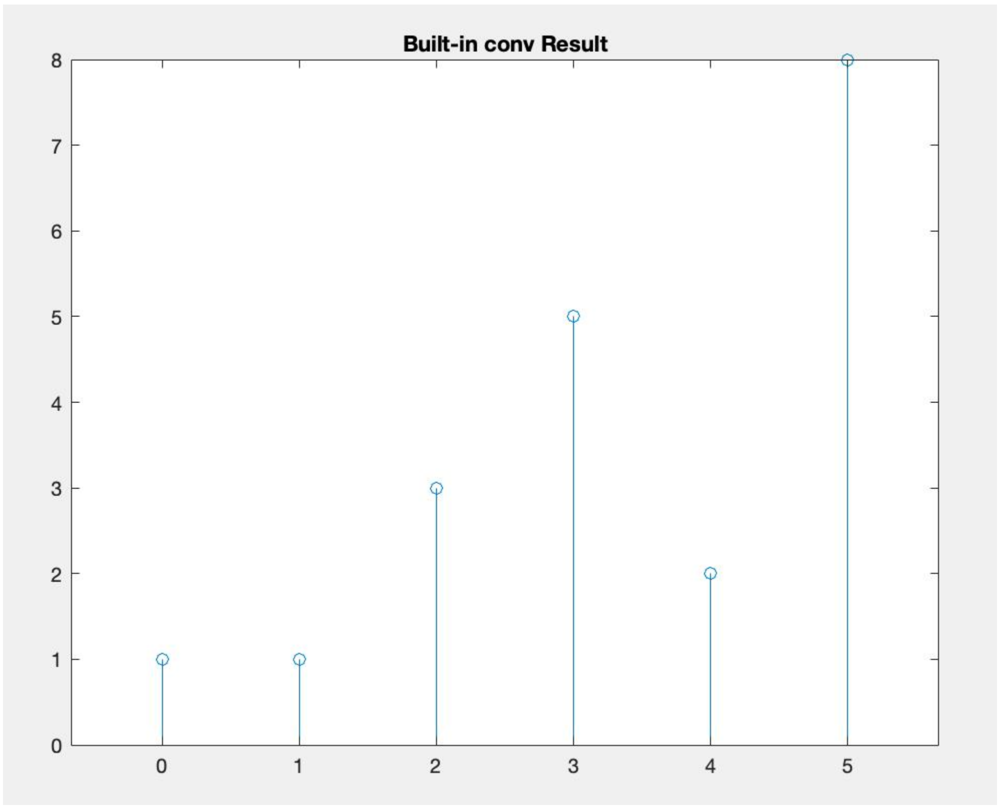
2. Write a MATLAB program to perform linear convolution of two sequences x(n) and h(n). Also verify the result using inbuilt functions.

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:





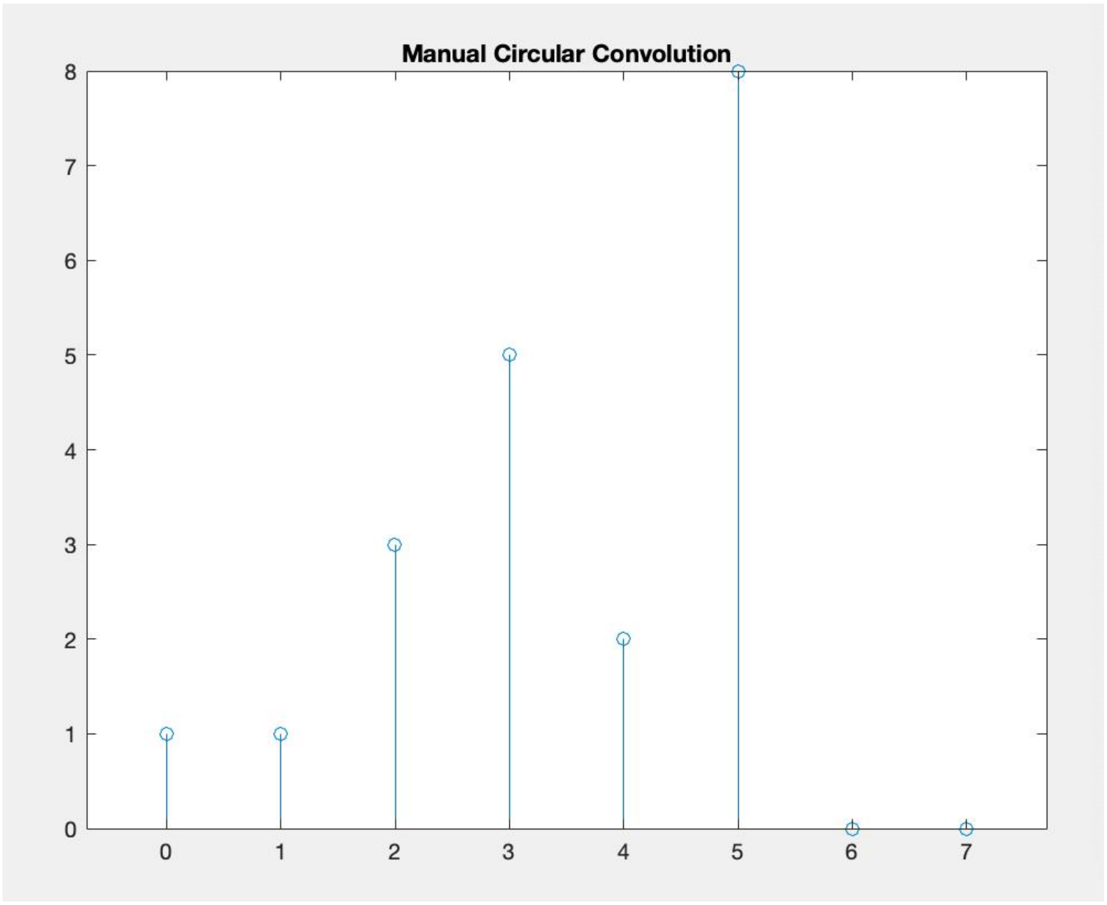
```
Manual vs Built-in comparison:
     1     1     3     5     2     8
     1     1     3     5     2     8
```

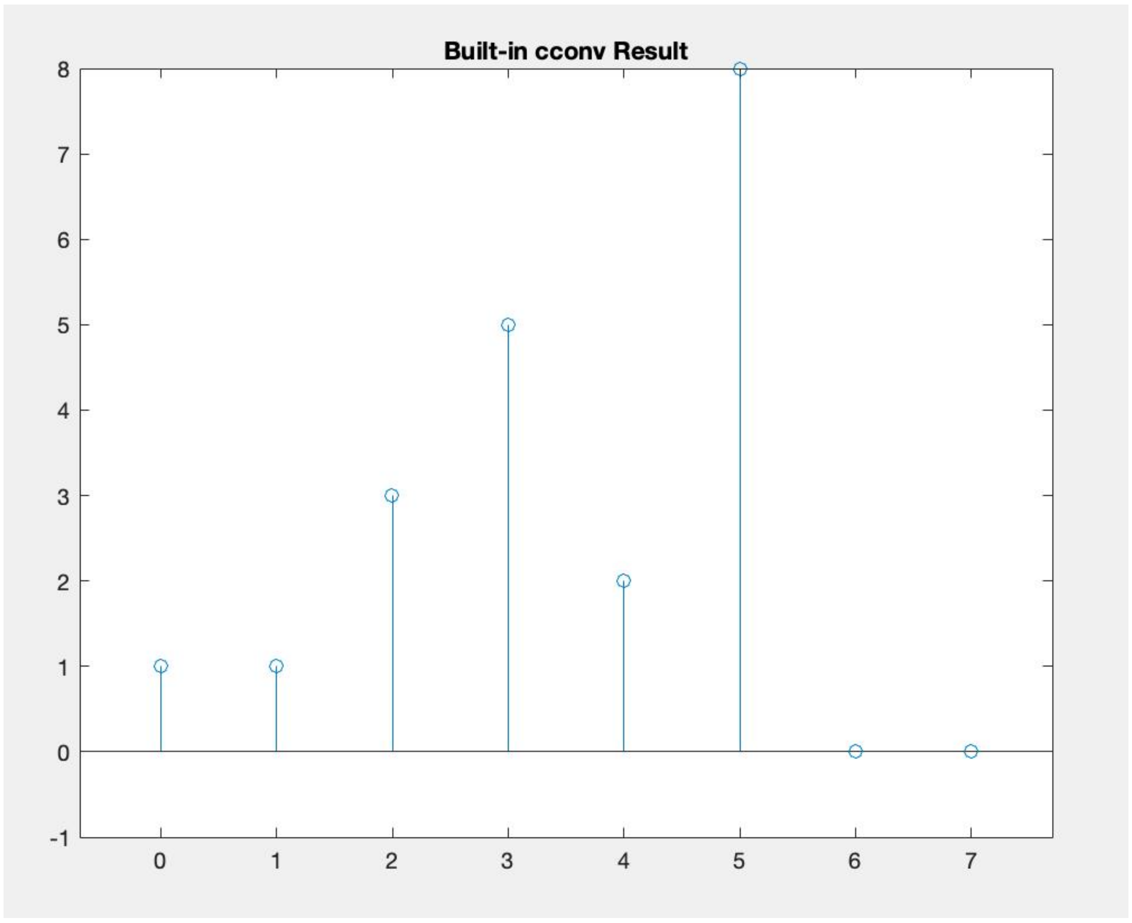
3. Write a MATLAB program to perform circular convolution of two sequences x(n) and h(n). Also verify the result using inbuilt functions.

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:





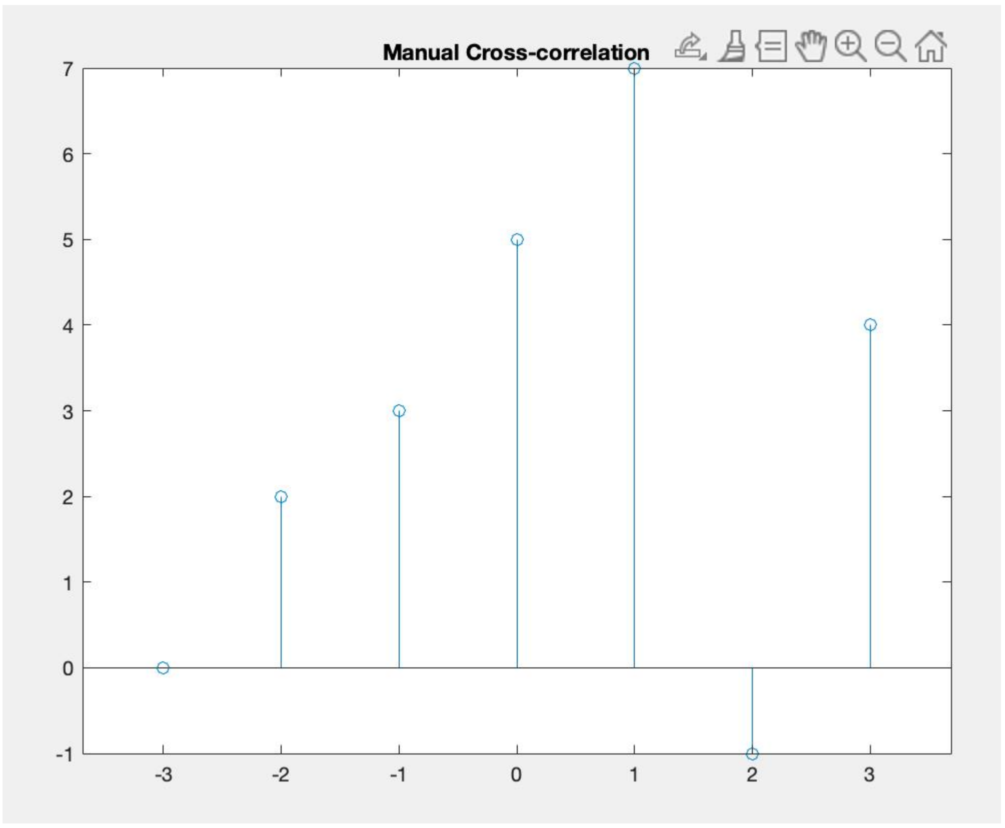
```
Manual vs Built-in Circular Convolution:
  Columns 1 through 6

    1.0000    1.0000    3.0000    5.0000    2.0000    8.0000
    1.0000    1.0000    3.0000    5.0000    2.0000    8.0000

  Columns 7 through 8

         0         0
    0.0000   -0.0000
```

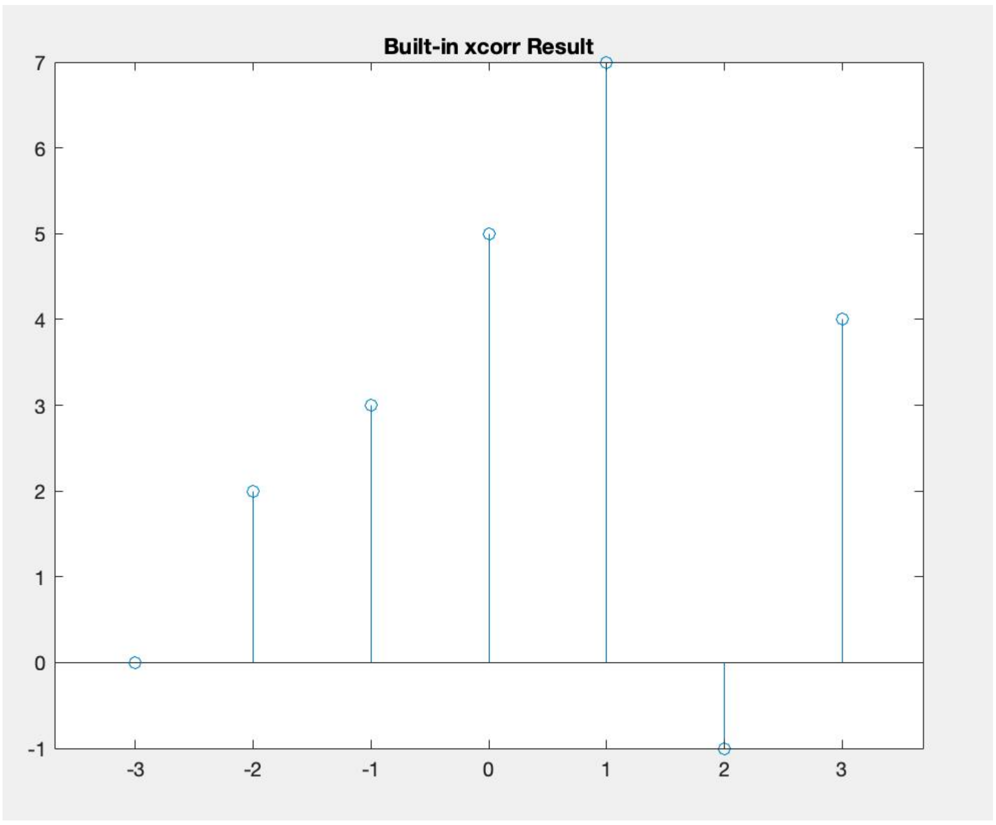
4. Write a MATLAB program to perform cross correlation between two sequences x(n) and h(n). Also verify the result using inbuilt functions.

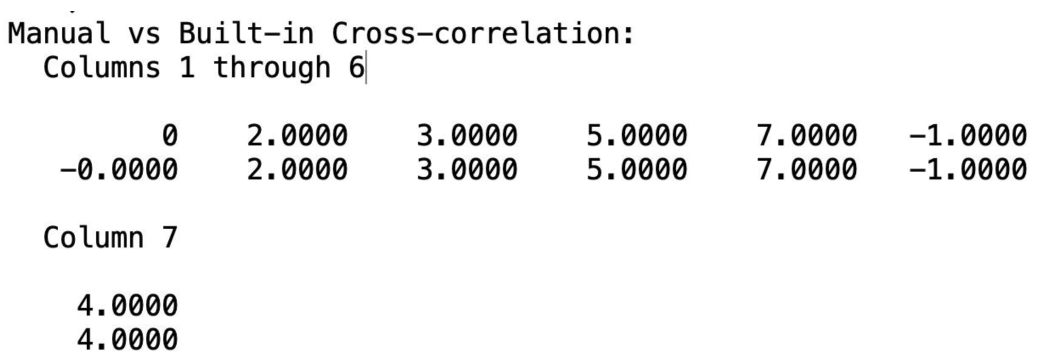
CODE:

```
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:


Manual Cross-correlation
7
6
5
4
3
2
1
0
-1
-3
-2
-1
0
1
2
3


Built-in xcorr Result
7
6
5
4
3
2
1
0
-1
-3
-2
-1
0
1
2
3



```
Manual vs Built-in Cross-correlation:
  Columns 1 through 6

         0    2.0000    3.0000    5.0000    7.0000   -1.0000
   -0.0000    2.0000    3.0000    5.0000    7.0000   -1.0000

  Column 7

    4.0000
    4.0000
```

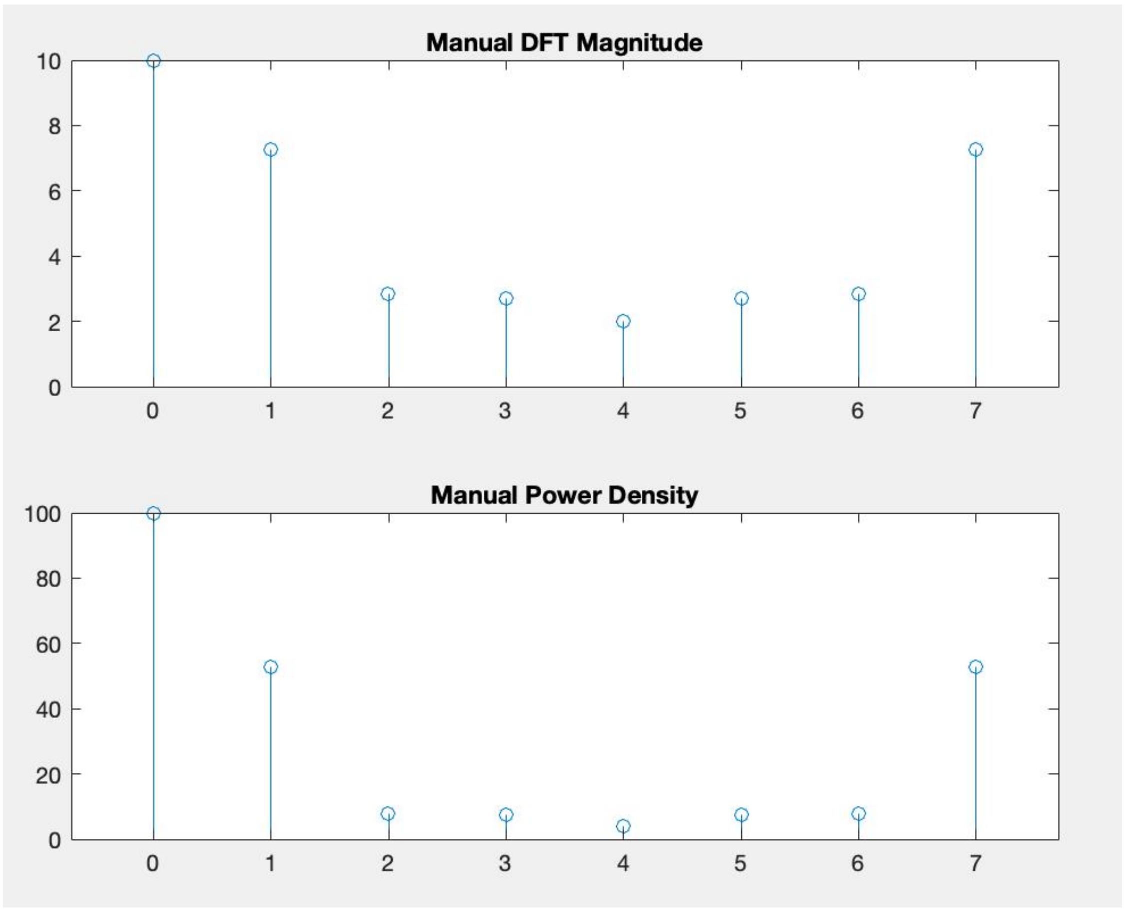
5. Compute and implement the N-point DFT of a given sequence and compute the power density spectrum of the sequence.

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:

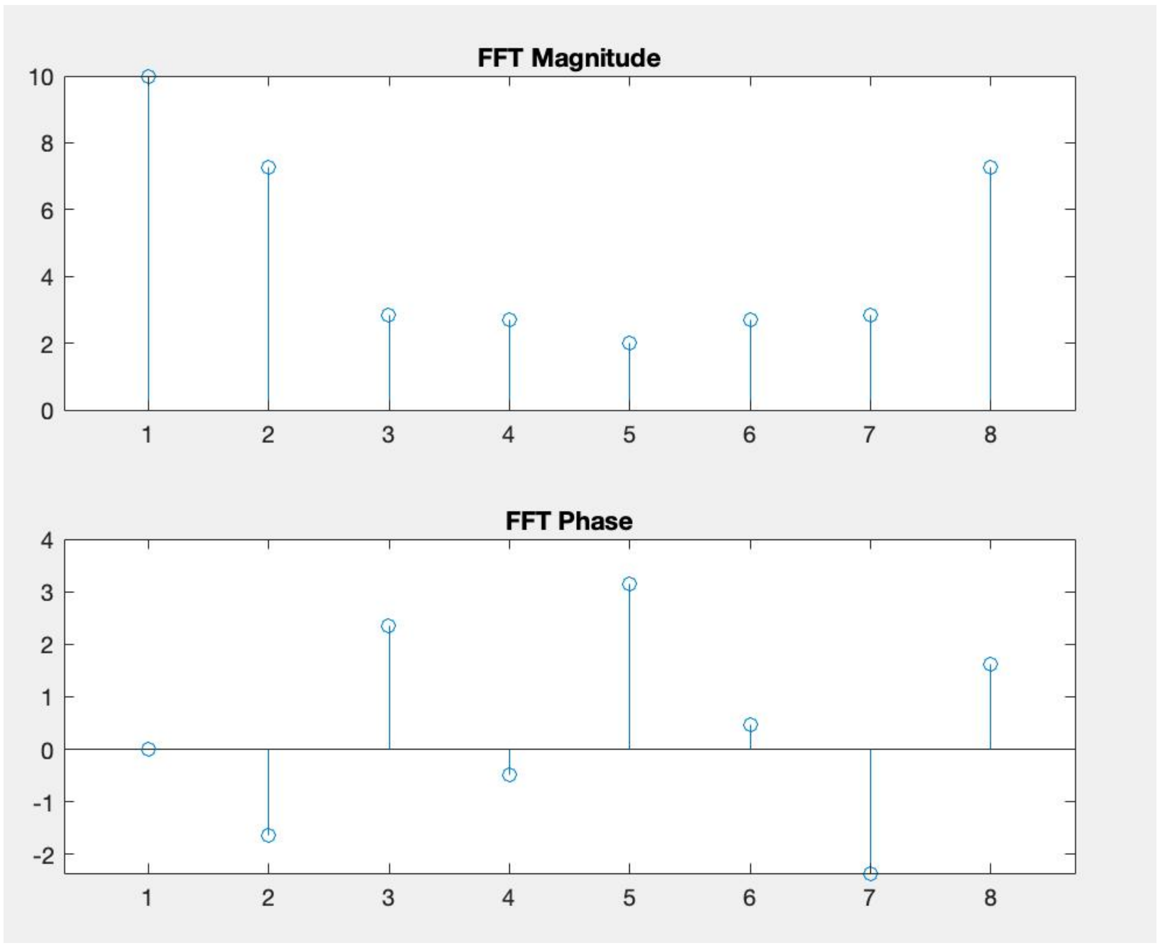


6. Implement and verify N-point DIT-FFT of a given sequence and find the frequency response(magnitude and phase).

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:



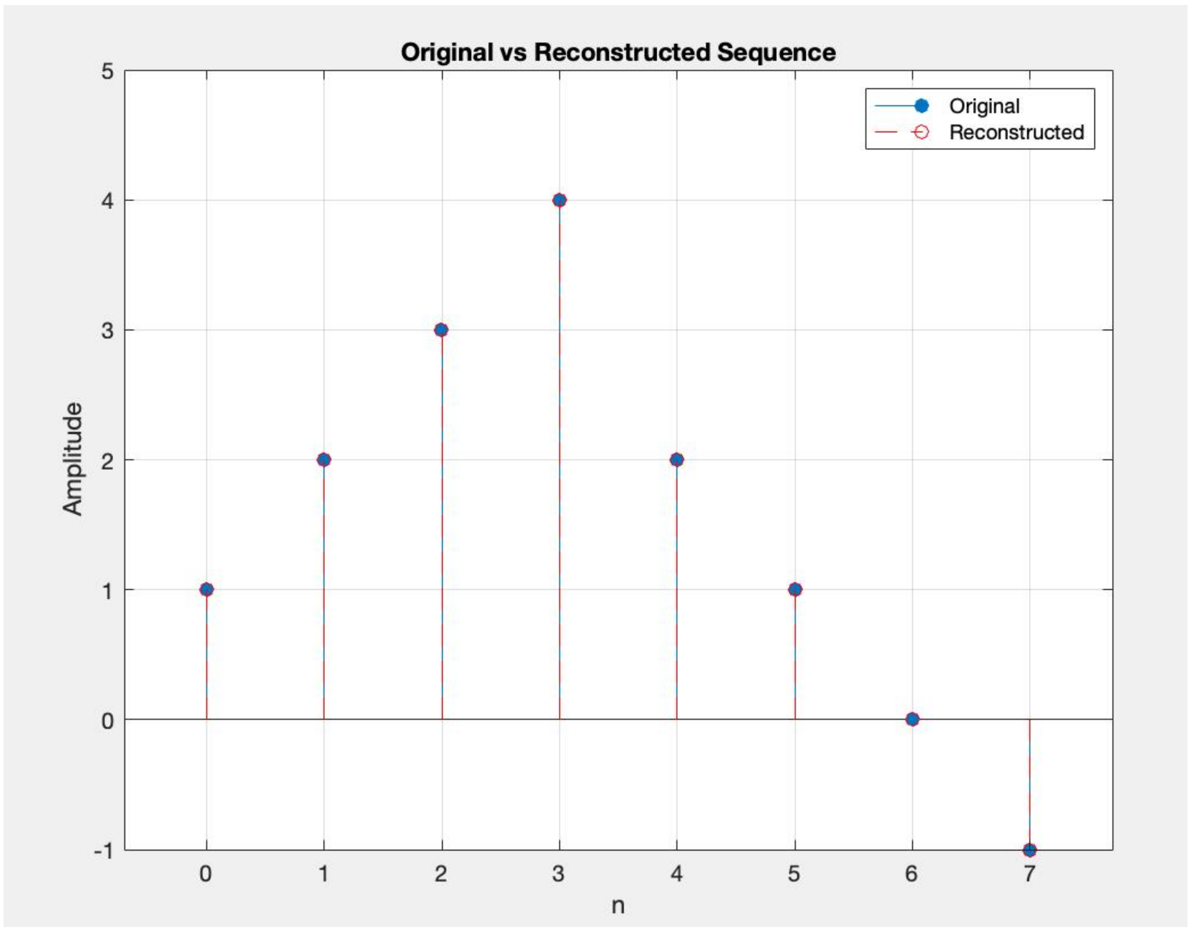
7. Implement and verify N-point IFFT of a given sequence.

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:



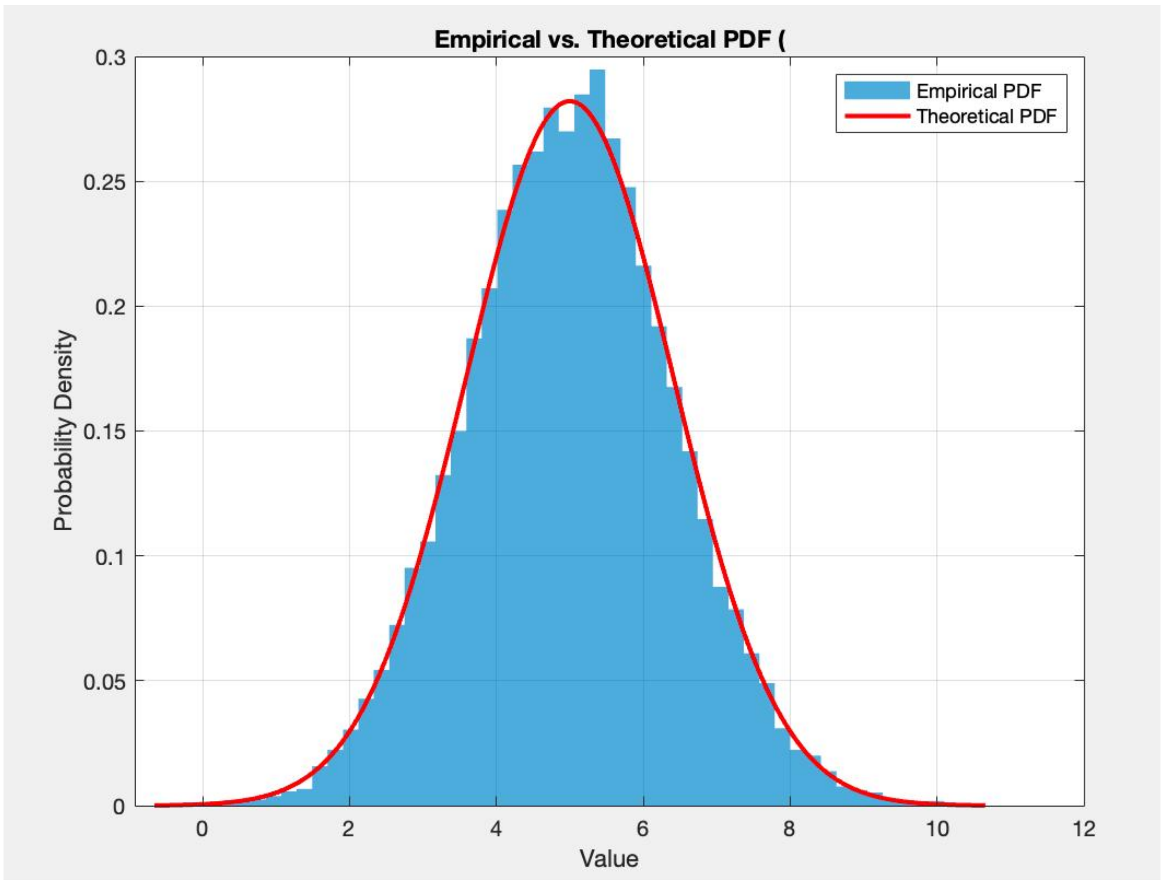
8. Write a MATLAB program to generate Gaussian numbers with given mean and variance. Plot the PDF of the generated numbers.

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:



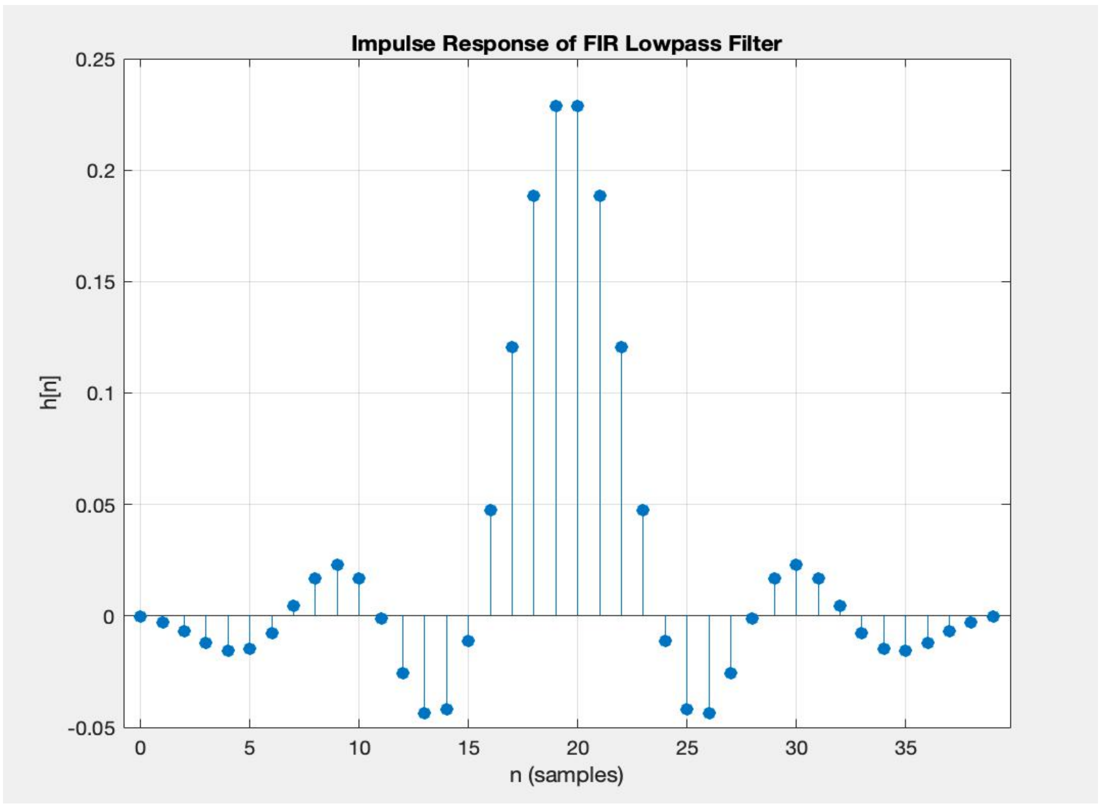
9. Design a FIR lowpass filter with given specification and verify the magnitude, phase and impulse response using FDA toolbox.

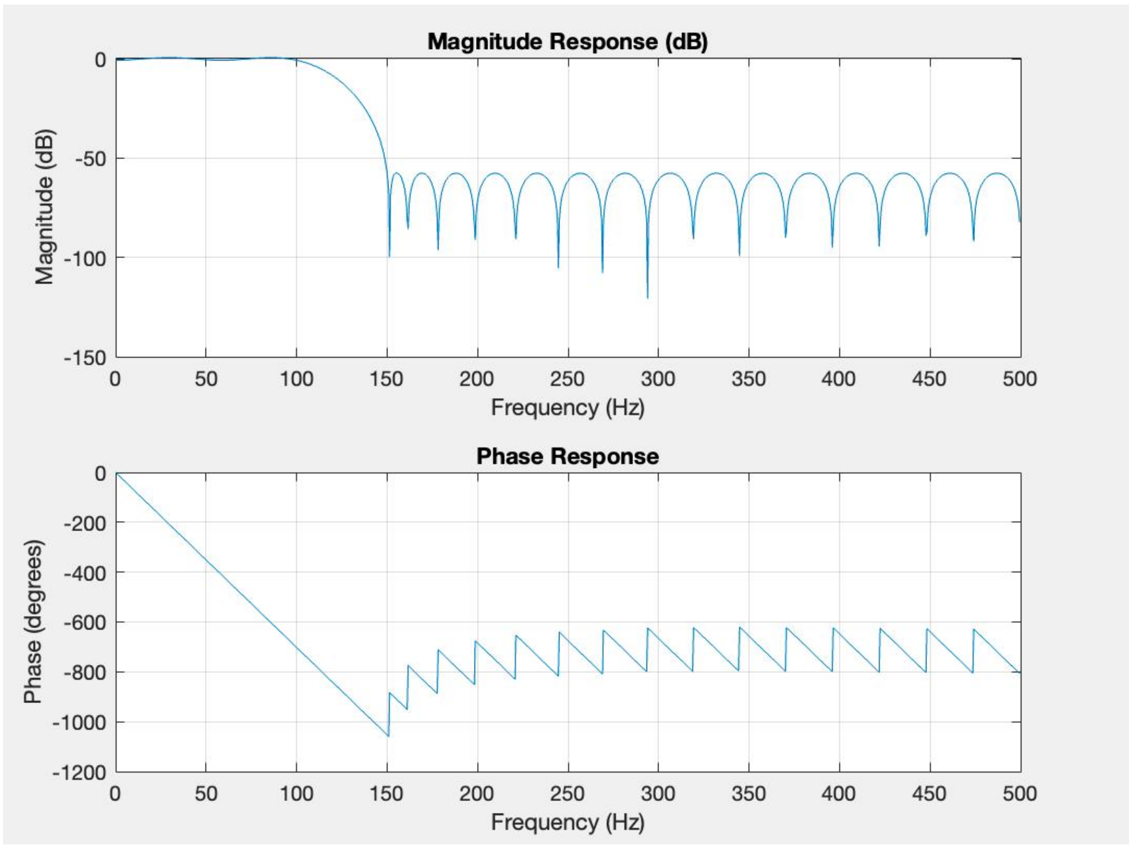
CODE:

```
%% 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:


Impulse Response of FIR Lowpass Filter
h[n]
n (samples)
0.25
0.2
0.15
0.1
0.05
0
-0.05
0
5
10
15
20
25
30
35



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

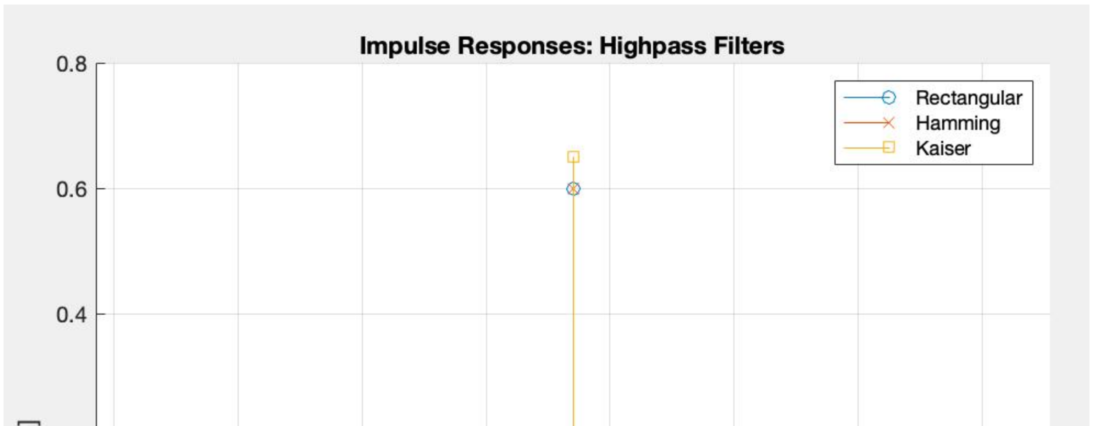
CODE:

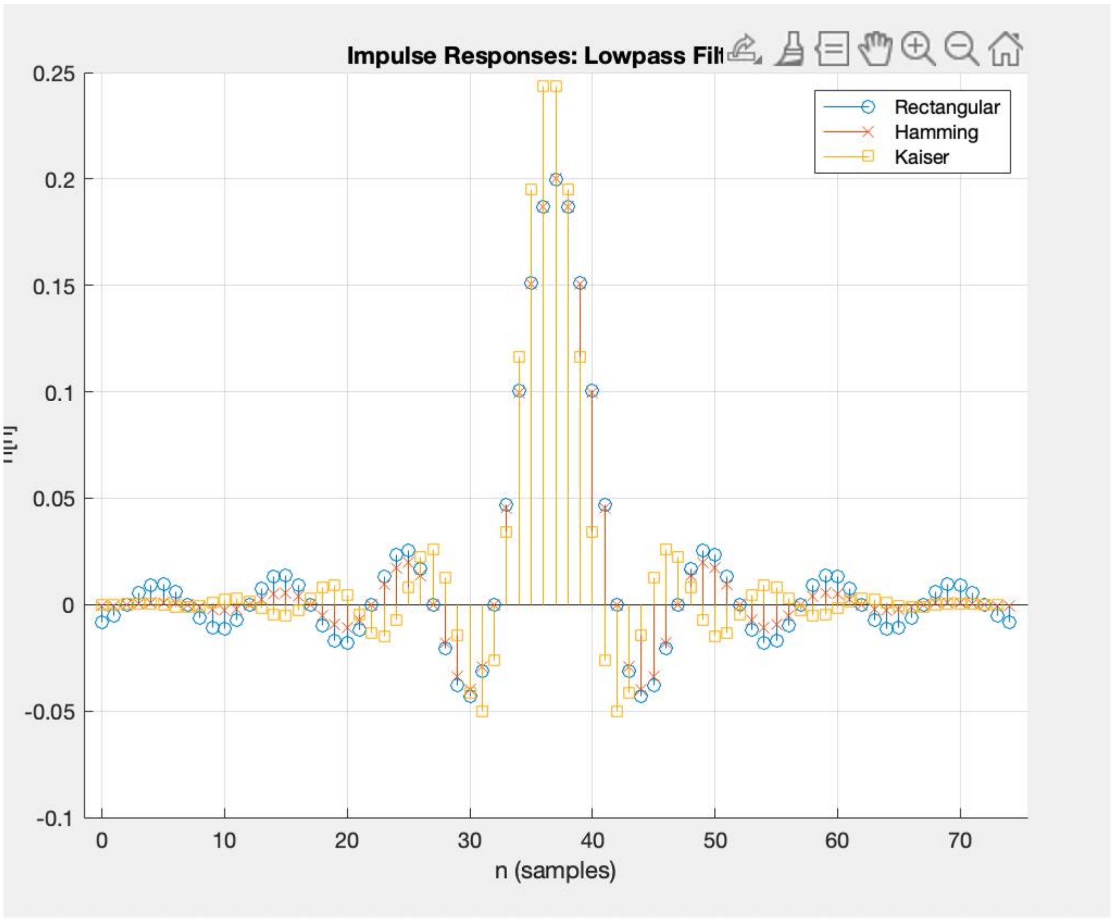
```
%% Specifications (modify as needed)
Fs = 1000; % Sampling frequency (Hz)
Fp_lp = 100; % Lowpass passband edge (Hz)
Fs_lp = 150; % Lowpass stopband edge (Hz)
Fp_hp = 200; % Highpass passband edge (Hz)
Fs_hp = 150; % Highpass stopband edge (Hz)
Rp = 1; % Passband ripple (dB)
Rs = 60; % Stopband attenuation (dB)

%% Estimate filter orders (using default Kaiser
approximation)
```

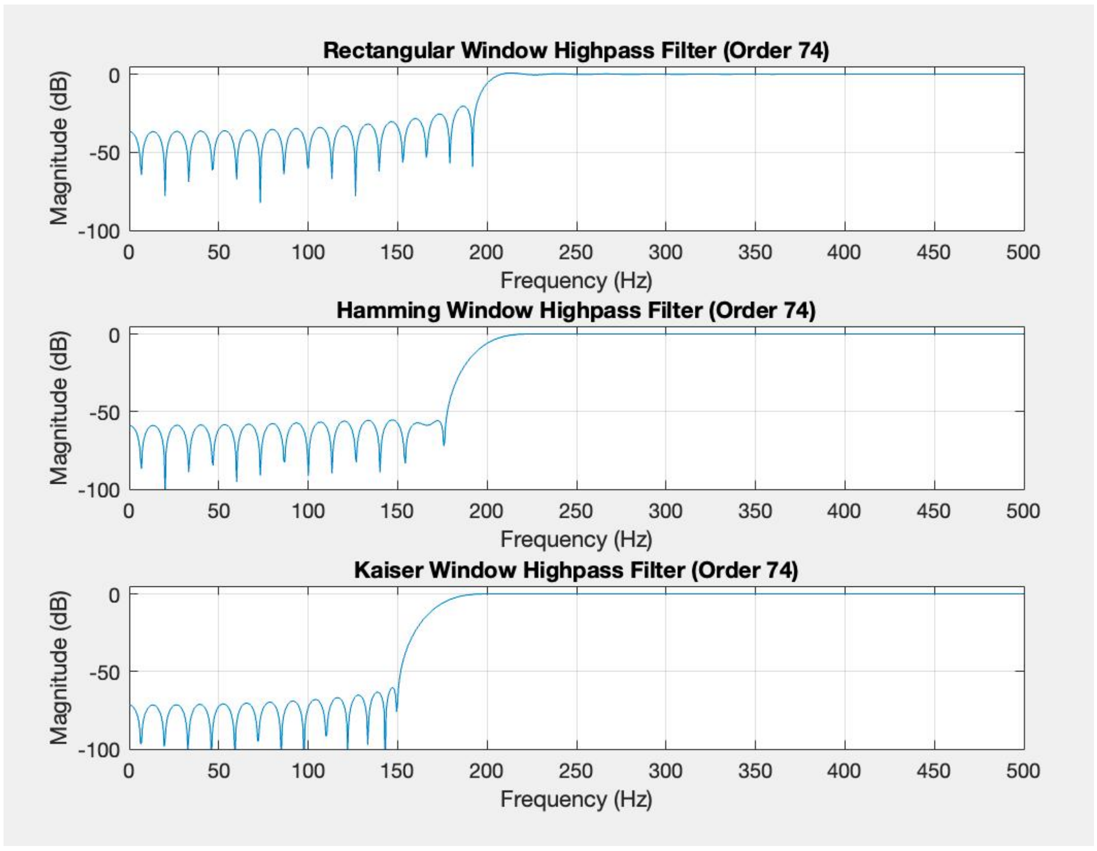
```
% For rectangular & Hamming, choose N large enough
manually or via fir1
% For Kaiser, use kaiserord
[n_kaiser_lp, Wn_lp, beta_lp, ftype_lp] =
kaiserord([Fp_lp Fs_lp], [1 0], [ (10^(Rp/20)-
1)/(10^(Rp/20)+1) 10^(-Rs/20) ], Fs);
N_kaiser_lp = n_kaiser_lp;
[n_kaiser_hp, Wn_hp, beta_hp, ftype_hp] =
kaiserord([Fs_hp 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);
```

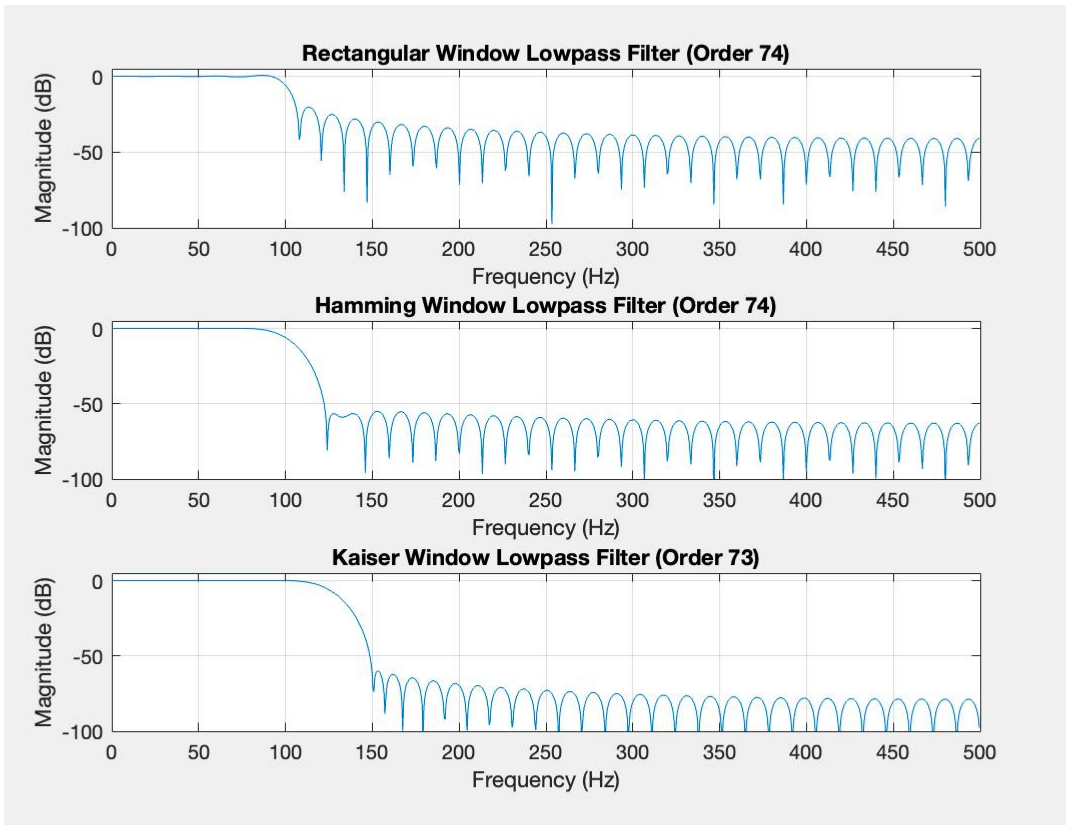
OUTPUT:




Impulse Responses: Lowpass Fil
0.25
0.2
0.15
0.1
0.05
0
-0.05
-0.1
h[n]
0
10
20
30
40
50
60
70
n (samples)
Rectangular
Hamming
Kaiser



Rectangular Window Highpass Filter (Order 74)
Magnitude (dB)
Frequency (Hz)
Hamming Window Highpass Filter (Order 74)
Magnitude (dB)
Frequency (Hz)
Kaiser Window Highpass Filter (Order 74)
Magnitude (dB)
Frequency (Hz)



Rectangular Window Lowpass Filter (Order 74)
Magnitude (dB)
Frequency (Hz)
Hamming Window Lowpass Filter (Order 74)
Magnitude (dB)
Frequency (Hz)
Kaiser Window Lowpass Filter (Order 73)
Magnitude (dB)
Frequency (Hz)

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

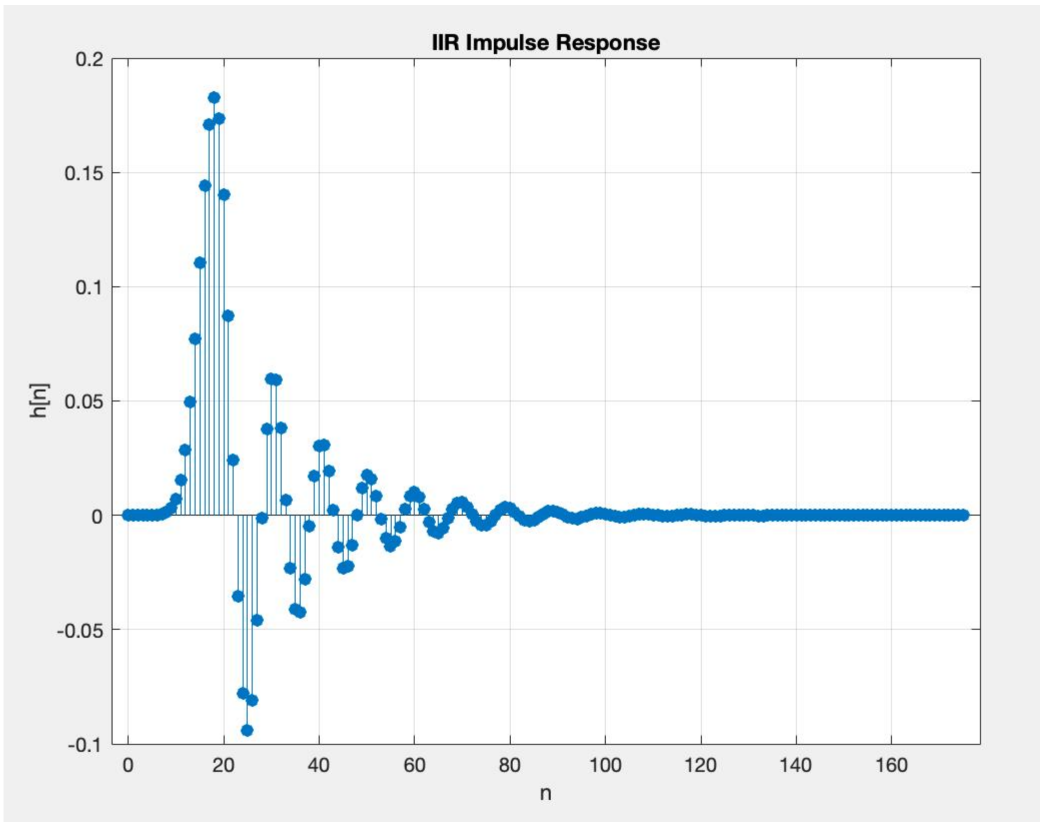
CODE:

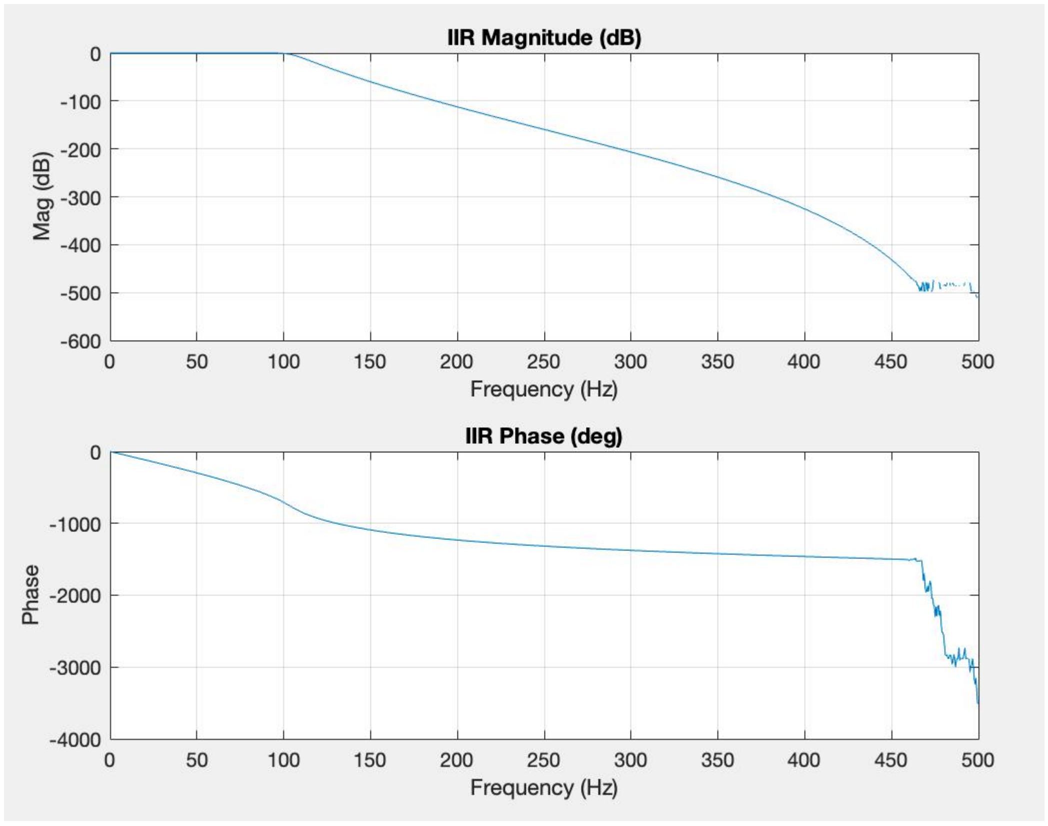
```
%% Specifications (modify as needed)
Fs = 1000; % Sampling frequency (Hz)
Fp_lp = 100; % Lowpass passband edge (Hz)
Fs_lp = 150; % Lowpass stopband edge (Hz)
Fp_hp = 200; % Highpass passband edge (Hz)
Fs_hp = 150; % Highpass stopband edge (Hz)
Rp = 1; % Passband ripple (dB)
Rs = 60; % Stopband attenuation (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)
```

```matlab
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:


IIR Impulse Response
h[n]
n
0.2
0.15
0.1
0.05
0
-0.05
-0.1
0
20
40
60
80
100
120
140
160


IIR Magnitude (dB)
Mag (dB)
Frequency (Hz)
0
-100
-200
-300
-400
-500
-600
0
50
100
150
200
250
300
350
400
450
500
IIR Phase (deg)
Phase
Frequency (Hz)
0
-1000
-2000
-3000
-4000

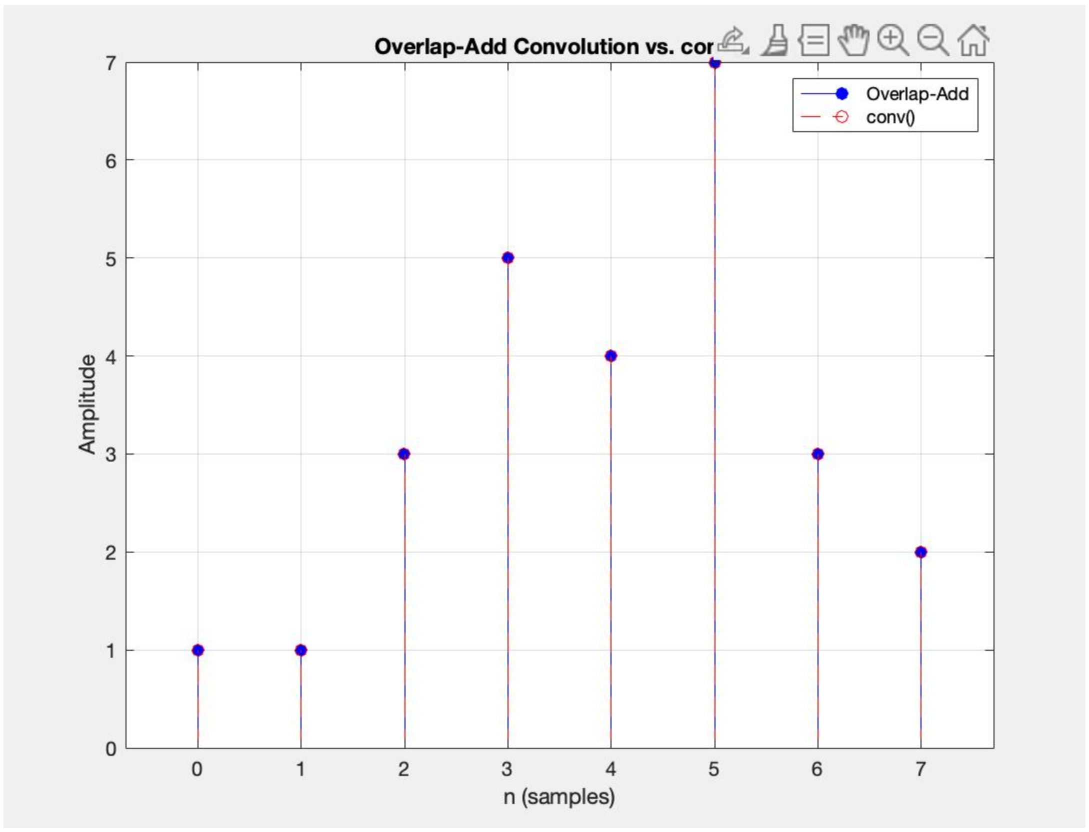
12. Write a MATLAB program to perform linear convolution of two sequences using overlap and add method.

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:



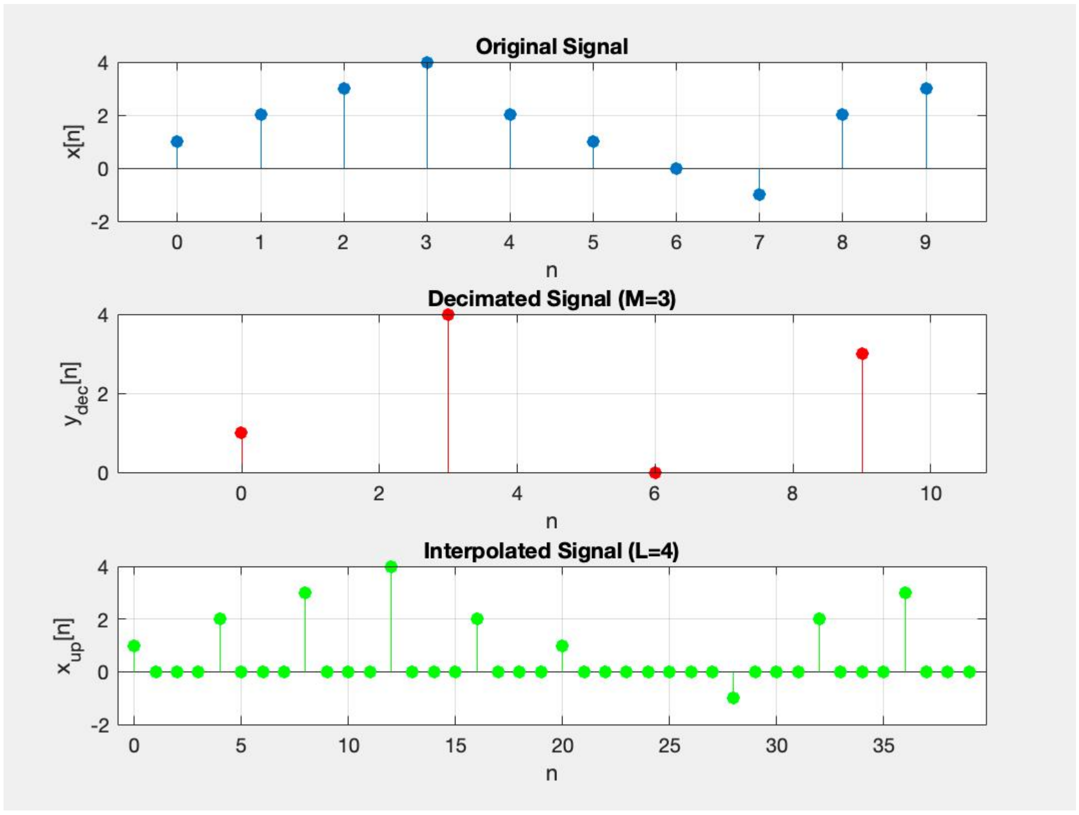
13. Compute the decimation and interpolation for the given signal

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.

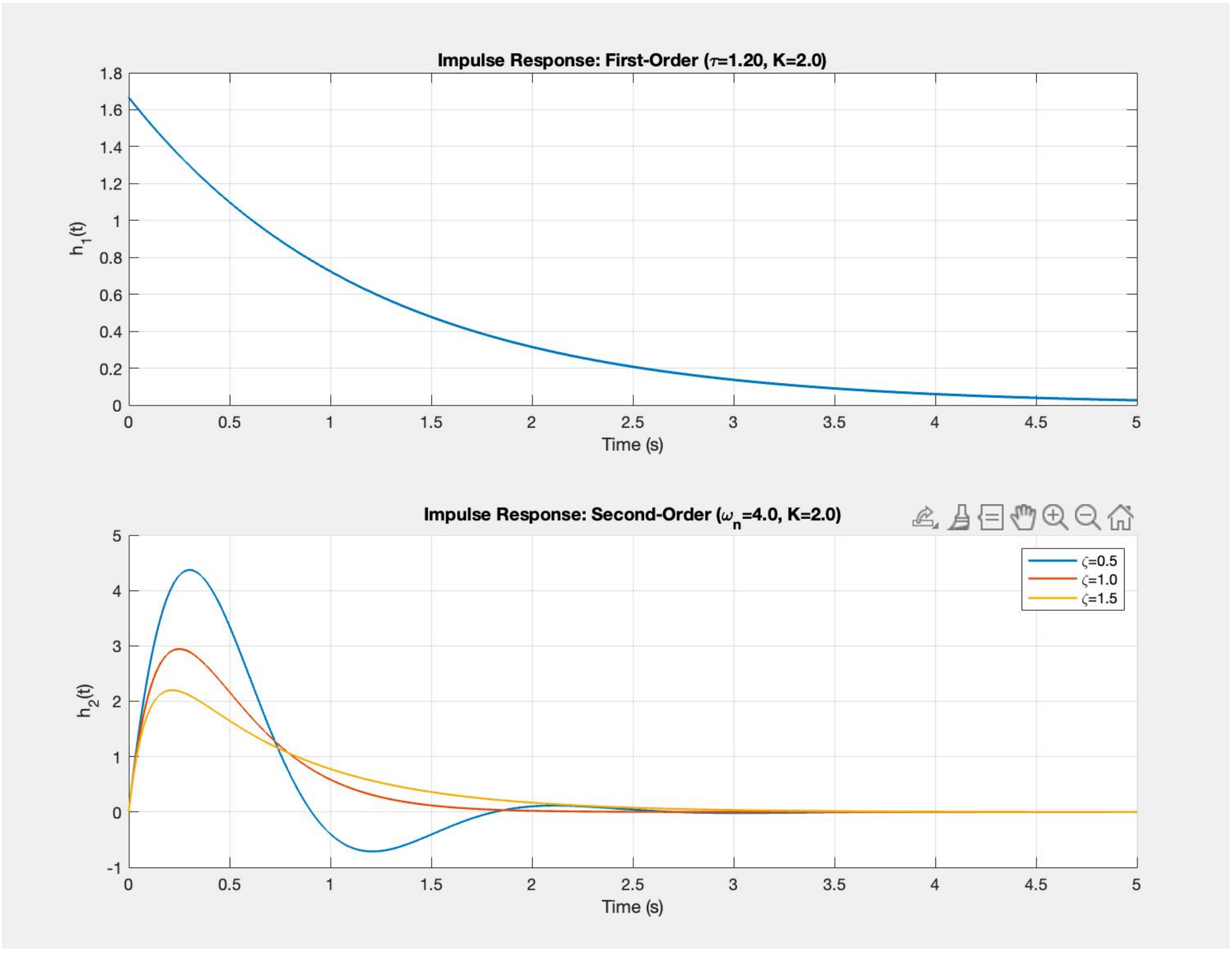
CODE:

```
% 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}
```

```matlab
%% 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:


Impulse Response: First-Order (τ=1.20, K=2.0)
h_1(t)
Time (s)
Impulse Response: Second-Order (ω_n=4.0, K=2.0)
h_2(t)
ζ=0.5
ζ=1.0
ζ=1.5
Time (s)