Compiled Question Texts

1a) ----------

```
% 1. elementary signals
clc; clear; close all;

n = -10 : 10;
x = (n == 0);
figure;
stem(n, x);
xlabel('n'); ylabel('\delta(n)');
title('Unit Impulse Sequence');
grid on;

x = (n >= 0);
figure;
stem(n, x);
xlabel('n'); ylabel('u(n)');
title('Unit Step Sequence');
grid on;

x = n .* (n >= 0);
figure;
stem(n, x);
xlabel('n'); ylabel('r(n)');
title('Unit Ramp Sequence');
grid on;

x = sin(0.2*pi*n);
figure;
stem(n, x);
xlabel('n'); ylabel('x(n)');
title('Sinusoidal Sequence');
grid on;

x = (0.8).^n .* (n >= 0);
figure;
stem(n, x, 'filled');
xlabel('n'); ylabel('x(n)');
title('Exponential Sequence');
grid on;

x = sign(sin(0.2*pi*n));
figure;
stem(n, x, 'filled');
xlabel('n'); ylabel('x(n)');
title('Square Wave Sequence');
grid on;
```

Compiled Question Texts

1b) ----------

```
# 2qn.. Continuous time sinusoid
F = 3;
A = 2;
th = 0;

Fs = 20*F;
Ts = 1/Fs;
T = 1/F;
t = 0:Ts:3*T;
x = A * sin(2*pi*F*t + th);
plot(t,x);
title('Continuous time sinusoid');
xlabel('t');
ylabel('x(t)');
grid on;
```

1c) ----------

```
# 3qn. Discrete time sinusoid
clc;
clear;

n = 0:50;
A = 1;
f = 0.1;
theta = 0;

x = A*cos(2*pi*f*n + theta);

stem(n, x, 'filled');
xlabel('n');
ylabel('x[n]');
title('Discrete-Time Sinusoidal Signal (f = 0.1)');
grid on;
```

1d) ----------

```
# 4qn. Fourier series
clc;
clear;

t = 0:0.001:1;     % time axis

% ---------- (a) N = 9 ----------
N1 = 9;
x1 = zeros(size(t));
```

```
for n = 1:2:N1     % odd harmonics only
    x1 = x1 + sin(2*pi*n*t);
end

% ---------- (b) N = 99 ----------
N2 = 99;
x2 = zeros(size(t));

for n = 1:2:N2
    x2 = x2 + sin(2*pi*n*t);
end

% ---------- (b) N = 999 ----------
N3 = 999;
x3 = zeros(size(t));

for n = 1:2:N3
    x3 = x3 + sin(2*pi*n*t);
end

% ---------- Plot Fourier Series Approximations ----------
figure;

subplot(3,1,1);
plot(t, x1);
title('Fourier Series Approximation (N = 9)');
xlabel('t'); ylabel('x(t)');
grid on;

subplot(3,1,2);
plot(t, x2);
title('Fourier Series Approximation (N = 99)');
xlabel('t'); ylabel('x(t)');
grid on;

subplot(3,1,3);
plot(t, x3);
title('Fourier Series Approximation (N = 999)');
xlabel('t'); ylabel('x(t)');
grid on;

% ---------- (d) Square Wave ----------
figure;
square_wave = square(2*pi*t);
plot(t, square_wave);
title('Square Wave using MATLAB square() function');
xlabel('t'); ylabel('x(t)');
grid on;
```

1e) ----------

```
# 5qn. Random signals
clc;
clear;

% -------------------------------
% (a) Normally Distributed Random Signal
% -------------------------------
N = 100;                    % length of signal
n = 0:N-1;

random_signal = randn(1, N);

figure;
stem(n, random_signal, 'filled');
title('Normally Distributed Random Signal (Length = 100)');
xlabel('n');
ylabel('x[n]');
grid on;

% -------------------------------
% (b) Noise Sinusoidal Signal
% -------------------------------
A = 1;
f = 0.05;                   % frequency
noise = 0.3 * randn(1, N);  % Gaussian noise

x = A*cos(2*pi*f*n) + noise;

figure;
stem(n, x, 'filled');
title('Noise Sinusoidal Signal');
xlabel('n');
ylabel('x[n]');
grid on;

% -------------------------------
% (c) Even and Odd Parts of Signal
% -------------------------------

% Create time-reversed signal
x_rev = fliplr(x);

% Even and Odd components
x_even = (x + x_rev)/2;
x_odd  = (x - x_rev)/2;

figure;

subplot(3,1,1);
```

```
stem(n, x, 'filled');
title('Original Noise Sinusoidal Signal');
xlabel('n'); ylabel('x[n]');
grid on;

subplot(3,1,2);
stem(n, x_even, 'filled');
title('Even Part of Signal');
xlabel('n'); ylabel('x_e[n]');
grid on;

subplot(3,1,3);
stem(n, x_odd, 'filled');
title('Odd Part of Signal');
xlabel('n'); ylabel('x_o[n]');
grid on;
```

2a) ----------

```
# 1qn. Convolution of two sequences
clc;
clear;

x1 = [1 2 3 4];
x2 = [2 3 4 5];

n1 = 0:length(x1)-1;
n2 = 0:length(x2)-1;

y = conv(x1, x2);

n = 0:length(y)-1;

figure;

subplot(3,1,1);
stem(n1, x1, 'filled');
title('Input Signal x_1(n)');
xlabel('n'); ylabel('x_1(n)');
grid on;

subplot(3,1,2);
stem(n2, x2, 'filled');
title('Input Signal x_2(n)');
xlabel('n'); ylabel('x_2(n)');
grid on;

subplot(3,1,3);
stem(n, y, 'filled');
```

```
title('Convolution Result y(n) = x_1(n) * x_2(n)');
xlabel('n'); ylabel('y(n)');
grid on;
```

2b) ----------

```
% 2qn.convolution ko dt_convolution file
function [y, ny] = dt_convolution(x1, n1, x2, n2)
L1 = length(x1);
L2 = length(x2);

L = L1 + L2 - 1;

y = zeros(1, L);
for i = 1:L1
    for j = 1:L2
        y(i + j - 1) = y(i + j - 1) + x1(i) * x2(j);
    end
end

ny = n1(1) + n2(1) : n1(end) + n2(end);
end
%-------
clc;
clear;


x1 = [1 2 3];
n1 = -1:1;

x2 = [2 1 4];
n2 = -2:0;

[y, ny] = dt_convolution(x1, n1, x2, n2);

figure;

subplot(3,1,1);
stem(n1, x1, 'filled');
title('Signal x_1[n]');
xlabel('n'); ylabel('x_1[n]');
grid on;

subplot(3,1,2);
stem(n2, x2, 'filled');
title('Signal x_2[n]');
xlabel('n'); ylabel('x_2[n]');
grid on;
```

```
subplot(3,1,3);
stem(ny, y, 'filled');
title('Convolution y[n] = x_1[n] * x_2[n]');
xlabel('n'); ylabel('y[n]');
grid on;
```

2c) ----------

```
# LCCD ko filters haru ko implementation
clc;
clear;

n = 0:20;

x = [ones(1,5) zeros(1,16)];

% --------------------------------
% 1. Three-Point Smoothing Filter
% y[n] = (1/3){x[n] + x[n-1] + x[n-2]}
% --------------------------------

y1 = zeros(size(x));

for i = 3:length(x)
    y1(i) = (1/3)*(x(i) + x(i-1) + x(i-2));
end

% --------------------------------
% 2. Differentiator
% y[n] = x[n] - x[n-1]
% --------------------------------

y2 = zeros(size(x));

for i = 2:length(x)
    y2(i) = x(i) - x(i-1);
end

% --------------------------------
% 3. Accumulator
% y[n] = x[n] + y[n-1]
% --------------------------------

y3 = zeros(size(x));

for i = 2:length(x)
    y3(i) = x(i) + y3(i-1);
end
```

```
figure;

subplot(4,1,1);
stem(n, x, 'filled');
title('Input Signal x[n]');
xlabel('n'); ylabel('x[n]');
grid on;

subplot(4,1,2);
stem(n, y1, 'filled');
title('Three-Point Smoothing Filter Output');
xlabel('n'); ylabel('y_1[n]');
grid on;

subplot(4,1,3);
stem(n, y2, 'filled');
title('Differentiator Output');
xlabel('n'); ylabel('y_2[n]');
grid on;

subplot(4,1,4);
stem(n, y3, 'filled');
title('Accumulator Output');
xlabel('n'); ylabel('y_3[n]');
grid on;
```

2d) ----------

```
# LCCD mai, 3 point smoothing filter , sinusoidal signals ko combine garna
clc;clear;
n = 0:100;

f1 = 0.05;
f2 = 0.47;

x1 = cos(2*pi*f1*n);
x2 = cos(2*pi*f2*n);

x = x1 + x2;

% Three-point smoothing filter
% y[n] = (1/3){x[n] + x[n-1] + x[n-2]}
y = zeros(size(x));

for i = 3:length(x)
    y(i) = (1/3)*(x(i) + x(i-1) + x(i-2));
end
```

```
figure;

subplot(4,1,1);
stem(n, x1);
title('Signal x_1[n] (f_1 = 0.05)');
xlabel('n'); ylabel('x_1[n]');
grid on;

subplot(4,1,2);
stem(n, x2);
title('Signal x_2[n] (f_2 = 0.47)');
xlabel('n'); ylabel('x_2[n]');
grid on;

subplot(4,1,3);
stem(n, x);
title('Combined Signal x[n] = x_1[n] + x_2[n]');
xlabel('n'); ylabel('x[n]');
grid on;

subplot(4,1,4);
stem(n, y);
title('Output y[n] of Three-Point Smoothing Filter');
xlabel('n'); ylabel('y[n]');
grid on;
```

2e) ----------

```
# Impulse response ko
# Generate a unit impulse sequence and pass it through the three point smoothing filter, the differentiator and the accumulator. Plot the impulse responses of the systems.
clc;
clear;

n = 0:20;

x = (n == 0);

y1 = zeros(size(x));
for i = 3:length(x)
    y1(i) = (1/3)*(x(i) + x(i-1) + x(i-2));
end

y2 = zeros(size(x));
for i = 2:length(x)
    y2(i) = x(i) - x(i-1);
end
```

```
y3 = zeros(size(x));
for i = 2:length(x)
    y3(i) = x(i) + y3(i-1);
end


figure;

subplot(4,1,1);
stem(n, x);
title('Unit Impulse Input \delta[n]');
xlabel('n'); ylabel('x[n]');
grid on;

subplot(4,1,2);
stem(n, y1);
title('Impulse Response: Three-Point Smoothing Filter');
xlabel('n'); ylabel('h_1[n]');
grid on;

subplot(4,1,3);
stem(n, y2);
title('Impulse Response: Differentiator');
xlabel('n'); ylabel('h_2[n]');
grid on;

subplot(4,1,4);
stem(n, y3);
title('Impulse Response: Accumulator');
xlabel('n'); ylabel('h_3[n]');
grid on;
```

2f) ----------

```
#Impulse response using impz
clc;
clear;

b1 = (1/3)*[1 1 1];
a1 = 1;

b2 = [1 -1];
a2 = 1;


b3 = 1;
a3 = [1 -1];
```

```
figure;

subplot(3,1,1);
impz(b1, a1);
title('Impulse Response using impz: Smoothing Filter');

subplot(3,1,2);
impz(b2, a2);
title('Impulse Response using impz: Differentiator');

subplot(3,1,3);
impz(b3, a3);
title('Impulse Response using impz: Accumulator');
```

2g) ----------

Check the stability test for the Accumulator, the Differentiator and Three-point smoothing filter. Also write a user defined MATLAB function to check whether the given LTI system is stable or not.

```
% ------------------------------------------------
% Stability Test of Discrete-Time LTI Systems
% Accumulator, Differentiator, Three-Point Smoothing
% ------------------------------------------------

clc;
clear;
close all;

% ---------- Impulse responses ----------

% Differentiator
h_diff = [1 -1];

% Three-point smoothing filter
h_smooth = [1/3 1/3 1/3];

% Accumulator (truncated impulse response)
N = 1000;               % approximation length
h_acc = ones(1, N);     % u[n]

% ---------- Stability Checks ----------
disp('STABILITY TEST RESULTS');
disp('----------------------');

disp('Differentiator:');
check_stability(h_diff);

disp('Three-point Smoothing Filter:');
check_stability(h_smooth);
```

```
disp('Accumulator:');
check_stability(h_acc);

% ---------- User-defined function ----------
function isStable = check_stability(h)
% CHECK_STABILITY checks BIBO stability of an LTI system
% using sum(|h[n]|) < infinity

sum_abs_h = sum(abs(h));

if isfinite(sum_abs_h)
   isStable = true;
   disp('  --> System is STABLE');
else
   isStable = false;
   disp('  --> System is UNSTABLE');
end

end
```

3a) ----------

1qn. Write a MATLAB program that calculate and plot the poles and zeros of this digital filter. Hint use the tf2zp & zplane Matlab command.

```
clc;
clear;

a = [1  -0.5  0.25  -0.1  0.05];
b = [0.2 0.4 0.3 0.1 0.05];


[z, p, k] = tf2zp(b, a);

disp('Zeros of the system:');
disp(z);

disp('Poles of the system:');
disp(p);

disp('Gain of the system:');
disp(k);
```

3b) ----------

2qn. Write a MATLAB program that calculates the transfer function of the digital filter that has the poles and zeros as z = {±1, 1 ±1i}, p = {±0.5, 0.1 ±0.2i}. Hint use the zp2tf MATLAB command. Let the gain of this digital filter k = 1. Also poles and zeros of this system.

```
% CRN: 021-395 - Transfer Function using zp2tf
```

```
z = [1; -1; 1+1i; 1-1i];
p = [0.5; -0.5; 0.1+0.2i; 0.1-0.2i];
k = 1;


[b, a] = zp2tf(z, p, k);

disp('Numerator Coefficients (b):'); disp(b);
disp('Denominator Coefficients (a):'); disp(a);
zplane(z, p);
title('Transfer function of the digitalfilter (021-389)');
```

3c) ----------

3qn. Write a MATLAB program that calculates and plot the poles and zeros of the accumulator circuit whose difference equation is

y[n] = x[n]+ y[n -1].

```
% CRN: 021-389 - Pole-Zero Plot for Accumulator

b = 1;
a = [1 -1];

[z, p, k] = tf2zp(b, a);

zplane(z, p);
title('CRN: 021-389 - Accumulator Pole-Zero Plot');

disp('Zeros:'); disp(z);
disp('Poles:'); disp(p);
disp('Gain:'); disp(k);
```

3d) ----------

4qn. CRN: 021-389 - Cascaded Form using tf2sos

```
z = [1; -1; 1+1i; 1-1i];
p = [0.5; -0.5; 0.1+0.2i; 0.1-0.2i];
k = 1;

[b, a] = zp2tf(z, p, k);
sos = tf2sos(b, a);

disp('Second Order Sections:');
disp(sos);
zplane(z, p);
```

```
title('Factorizing the 4th-Order Filter (021-389)');
```

3e) ----------

```
# 5qn. CRN: 021-389 - Partial Fraction Expansion using residuez
clc; clear; close all;
b = conv([1 0 -1], [1 -2 2]);
a = conv([1 -0.25], [1 -0.2 0.05]);

disp('Numerator b:'),
disp(b)
disp('Denominator a:'),
disp(a)

[r, p, k] = residuez(b, a);
disp('Residues r:');disp(r);
disp('Poles p:');disp(p);
disp('Direct terms k:');disp(k);

zplane(b, a);
title('Partial Fraction Expansion (021-389)');
```

3f) ----------

```
% CRN: 021-389 - Frequency and Phase Response
clc; clear; close all;
b = conv([1 0 -1], [1 -2 2]);
a = conv([1 -0.25], [1 -0.2 0.05]);
omega = linspace(0, pi, 1024);
z = exp(1j*omega);

Hz = polyval(b, z.^-1) ./ polyval(a, z.^-1);

subplot(2,1,1);
plot(omega, abs(Hz));
grid on;
title('Magnitude Response (021-389) |H(e^{j\omega})|');
xlabel('\omega (rad/sample)');
ylabel('Magnitude');

subplot(2,1,2);
plot(omega, angle(Hz));
grid on;
title('Phase Response arg (021-389)(H(e^{j\omega}))');
xlabel('\omega (rad/sample)');
ylabel('Phase (radians)');
```

4a) ----------

% Write a MATLAB program to compute discrete Fourier transform (DFT) of the signal.

```
clc; clear all; close all;
x = [1,2,3,4,5,6,7,8];
N = length(x);
X = zeros(1,N);
for k = 0:N-1
   for n = 0:N-1
      X(k+1) = X(k+1) + x(n+1)*exp(-1i*2*pi*k*n/N);
   end
end
disp('021-395 DFT X(k):');
disp(X);
```

4b) ----------

Also write a user defined function to calculate the direct DFT of given signal x[n]. The function may have the following syntax:

```
function y = dft_f(x)
   N = length(x);
   y = zeros(1, N);
   for k = 0:N-1
      for n = 0:N-1
         y(k+1) = y(k+1) + x(n+1) * exp(-1i*2*pi*k*n/N);
      end
   end
end
```

4c) ----------

Now plot graphs for real, imaginary, absolute value and phase of the calculated DFT.

```
clc; clear all; close all;
x = [1,2,3,4,5,6,7,8];
X = dft_f(x);
k = 0:length(X)-1;
subplot(2,2,1); stem(k, real(X)); title('Real Part (021-395)');
subplot(2,2,2); stem(k, imag(X)); title('Imaginary Part (021-395)');
subplot(2,2,3); stem(k, abs(X)); title('Magnitude (021-395)');
subplot(2,2,4); s= stem(k, angle(X)); title('Phase (021-395)');
```

4d) ----------

% Repeat the question no. 1 using DFT as linear transformation (XN

```
clc; clear all; close all;
```

```
x = [1,2,3,4,5,6,7,8];
N = length(x);
n = 0:N-1;
k = n';
WN = exp(-1i*2*pi/N);
WN_matrix = WN .^ (k*n);
X = WN_matrix * x.';
disp('DFT as linear transformation (021-395):');
disp(X);
```

4e) ----------

Check the answers of questions 1 & 2 using the built-in MATLAB function “fft”.

```
clc; clear all; close all;
x = [1,2,3,4,5,6,7,8];
X = fft(x);
disp('DFT using fft (021-395):');
disp(X);
```

4f) ----------

A MATLAB function ‘tic’ and ‘toc’ are used to measure the elapsed time. Measure the elapsed time to calculate
the DFT by three different ways.

```
clc; clear all; close all;
x = [1,2,3,4,5,6,7,8];

tic
X1 = dft_f(x);
t1 = toc;

tic
N = length(x);
n = 0:N-1; k = n';
WN = exp(-1i*2*pi/N); WN_matrix = WN .^ (k*n);
X2 = WN_matrix * x.';
t2 = toc;

tic
X3 = fft(x);
t3 = toc;

fprintf('Time (function): %f ms\nTime (linear): %f ms\nTime (fft): %f ms\n', t1*1000, t2*1000, t3*1000);
```

4g) ----------

```
% Repeat steps 1-4, to calculate the inverse DFT.
clc; clear all; close all;
X = fft([1,2,3,4,5,6,7,8]);
N = length(X);
n = 0:N-1; k = n';
WN = exp(1i*2*pi/N);
WN_matrix = WN .^ (k*n);
x_reconstructed = (1/N) * (WN_matrix * X.');
disp('Reconstructed x[n] using IDFT matrix (021-395):');
disp(x_reconstructed);

x_ifft = ifft(X);
disp('Reconstructed x[n] using ifft(021-395):');
disp(x_ifft);
```

4h) ----------

```
% Write a function MATLAB program to compute circular convolution of two unequal sequences.
function y = circular_conv(x1, x2)
    N = max(length(x1), length(x2));
    x1 = [x1, zeros(1, N - length(x1))];
    x2 = [x2, zeros(1, N - length(x2))];
    y = ifft(fft(x1) .* fft(x2));
end
```

4i) ----------

```
% Check the answer using the built-in function “cconv”.
clc;  clear all; close all;
x1 = [1,2,3]; x2 = [4,5];
y = cconv(x1, x2, max(length(x1), length(x2)));
disp('Circular convolution using cconv (021-395):');
disp(y);
```

4k) ----------

```
clc; clear; close all;
[x, Fs] = audioread('sample_audio.mp3');
if size(x,2) == 2
    x = x(:,1);
end
N = length(x);
X = fft(x);
```

```
f = (-N/2+1:N/2)*Fs/N;
plot(f, fftshift(abs(X)/N));
title('Amplitude Spectrum of Voice Signal (021-395)');
xlabel('Frequency (Hz)');
ylabel('|X(f)|');
grid on;
```

5a) ----------

Write a program that plots the signal x(t) = sum n = 1 to N ((- 1) ^ ((n - 1)/2))/(n ^ 2) * sin((n pie t)/2) Plot for N = 10 and 100+a and determine the period of signal x(t) assume a = 66

```
clc; close all; clear all;
a = 66;
t = -10:0.001:10;
N1 = 10;
x1 = zeros(size(t));
for n = 1:2:N1
x1 = x1 + ((-1)^((n-1)/2) / n^2) .* sin(n*pi*t/2);
end
subplot(2,1,1);
plot(t, x1);
title('x(t) for N = 10');
xlabel('t');
ylabel('x(t)');
grid on;
N2 = 100 + a;
x2 = zeros(size(t));
for n = 1:2:N2
x2 = x2 + ((-1)^((n-1)/2) / n^2) .* sin(n*pi*t/2);
endsubplot(2,1,2);
plot(t, x2);
title('x(t) for N = 166');
xlabel('t');
ylabel('x(t)');
grid on;
```

5b) ----------

2. Write a MATLAB program to plot x(t) = sin(20pi*t) + cos(50pi*t) - sin(a*pi*t) From figure obtained, determine the period of this signal.

```
clc; clear; close all;
a = 66;
t = 0:0.0001:2; % time axis
x = sin(20*pi*t) + cos(50*pi*t) - sin(a*pi*t);
plot(t, x);
grid on;
```

```
xlabel('t');
ylabel('x(t)');
title('x(t) = sin(20\pit) + cos(50\pit) - sin(66\pit)');
%sin(20 pie t) frequency = 10 Hz
%cos(50 pie t) frequency = 25 Hz
%sin(66 pie t) frequency = 33 Hz
%These frequencies are all integer multiples of 1 Hz.
% So the whole signal repeats after 1 second
```

5c) ----------

Write a MATLAB program that generates three sinusoidal signals with relative frequencies a/400, a/300 and a/200. The signals should then be added and the resultant signal (x) is applied to the four-point smoothing digital filter. Plot all the signals.

```
clc; clear; close all;
a = 66;
t = 0:0.001:2;
x1 = sin(2*pi*(a/400)*t);
x2 = sin(2*pi*(a/300)*t);
x3 = sin(2*pi*(a/200)*t);
x = x1 + x2 + x3;
h = (1/4) * ones(1,4);
y = filter(h, 1, x);subplot(5,1,1);
plot(t, x1); grid on;
title('Signal 1 : frequency = a/400');
subplot(5,1,2);
plot(t, x2); grid on;
title('Signal 2 : frequency = a/300');
subplot(5,1,3);
plot(t, x3); grid on;
title('Signal 3 : frequency = a/200');
subplot(5,1,4);
plot(t, x); grid on;
title('Resultant Signal (x)');
subplot(5,1,5);
plot(t, y); grid on;
title('Output after 4-point Smoothing Filter');
xlabel('Time (t)');
```

5d) ----------

4. Write a user defined MATLAB function to calculate convolution of two non-causal discrete-time signals. Find the convolution of x[n]={1, a, 3, -1,5,2] and h[n] = {- 1, 6, 5, a, 2} Plot all the signals.

```
% Create conv.m file to make function
function y = conv(x, h)
Nx = length(x);
Nh = length(h);
y = zeros(1, Nx + Nh - 1);
```

```
for n = 1:Nx
for k = 1:Nh
y(n+k-1) = y(n+k-1) + x(n)*h(k);
end
end
end
%main program in another .m file
clc; clear; close all;
a = 66;
x = [1 a 3 -1 5 2]; % 3 is at n = 0
h = [-1 6 5 a 2]; % 6 is at n = 0
nx = -2:3; % since 3 is at position 3nh = -1:3; % since 6 is at position 2
% Convolution
y = conv(x, h);
ny = (nx(1)+nh(1)):(nx(end)+nh(end));
subplot(3,1,1);
stem(nx, x, 'filled');
grid on;
title('Input Signal x[n]');
xlabel('n'); ylabel('x[n]');
subplot(3,1,2);
stem(nh, h, 'filled');
grid on;
title('Input Signal h[n]');
xlabel('n'); ylabel('h[n]');
subplot(3,1,3);
stem(ny, y, 'filled');
grid on;
title('Convolution Output y[n]');
xlabel('n'); ylabel('y[n]');
```

5e) ----------

Write a MATLAB program that plots the impulse response of the three-point smoothing filter. Plot also the frequency response of this system.

```
clc; clear; close all;
% Impulse response of 3-point smoothing filter
h = (1/3)*ones(1,3);
n = 0:2;
subplot(2,1,1);
stem(n, h, 'filled');
grid on;
xlabel('n');
ylabel('h[n]');
title('Impulse Response of 3-Point Smoothing Filter');
% Frequency response
[H, w] = freqz(h, 1, 512);
subplot(2,1,2);
plot(w/pi, abs(H));
grid on;
```

```
xlabel('Normalized Frequency (\times\pi rad/sample)');
ylabel('|H(e^{j\omega})|');
title('Magnitude Response of 3-Point Smoothing Filter');
```

5f) ----------

Write a user defined MATLAB function to calculate unit step sequence. Use this function to calculate unit impulse sequence and unit ramp sequence for values of n ranging from -5 to 15.

```
% create unitstep.m file to make function
function u = unitstep(n)
u = zeros(size(n));
u(n >= 0) = 1;
end
% main program
clc; clear; close all;
n = -5:15;
% Unit step sequence
u = unitstep(n);
% Unit impulse sequence
delta = unitstep(n) - unitstep(n-1);
% Unit ramp sequence
r = n .* unitstep(n);subplot(3,1,1);
stem(n, u, 'filled');
grid on;
title('Unit Step Sequence u[n]');
xlabel('n'); ylabel('u[n]');
subplot(3,1,2);
stem(n, delta, 'filled');
grid on;
title('Unit Impulse Sequence \delta[n]');
xlabel('n'); ylabel('\delta[n]');
subplot(3,1,3);
stem(n, r, 'filled');
grid on;
title('Unit Ramp Sequence r[n]');
xlabel('n'); ylabel('r[n]');
```

5g) ----------

7. Write a user defined function to calculate the direct DFT of given signal x[n] . Find the DFT of following sequence x[n] = {1, 2, 6, 5, 3, 2, 1, 2}

```
% create dft.m file to make function
function X = dft(x)
N = length(x);
X = zeros(1, N);
for k = 0:N-1
for n = 0:N-1
X(k+1) = X(k+1) + x(n+1) * exp(-1i*2*pi*k*n/N);
```

```
end
end
end
%main program
clc; clear; close all;
x = [1,2,6,5,3,2,1,2];
X = dft(x);
disp('DFT of the given sequence x[n]:');
disp(X);
```

5h) ----------

Write a user defined function to calculate the DFT of given signal x[n] using DFT as linear transformation. Find the DFT of following sequence x[n] = {1, 2, 6, 5, 3, 2, 1, 2}

```
% create dft_linear.m file to make function
function X = dft_linear(x)
N = length(x);
n = 0:N-1;
k = n';
WN = exp(-1i*2*pi/N); % Twiddle factor
WN_matrix = WN .^ (k*n); % DFT matrix
X = WN_matrix * x.'; % Linear transformation
end
% main program
clc; clear; close all;
x = [1 2 6 5 3 2 1 2];
X = dft_linear(x);
disp('DFT of the given sequence using linear transformation:');
disp(X)
```

5i) ----------

A third order digital filter has the difference equation of the form: y[n] = x[n]+2x[n-1]+3x[n-2]-0.5 y[n-1] -0.6 y[n-2]- 0.1y[n-3] Write a MATLAB program that calculates the poles and zeros of this digital filter.Also plot the poles and zeros of this filter.

```
clc; clear; close all;
b = [1 2 3];
a = [1 0.5 0.6 0.1];
% Calculate zeros and poles
zeros_f = roots(b);
poles_f = roots(a);
disp('Zeros of the digital filter:');
disp(zeros_f);
disp('Poles of the digital filter:');
disp(poles_f);
% Plot poles and zeros
figure;
zplane(b, a);
grid on;
```

```
title('Pole-Zero Plot of the Digital Filter');
```

5j) ----------

10. Write a MATLAB program that calculates the transfer function of the digital filter that has the following poles and zeros. Let the gain of this digital filter k = 1. {zk} = {- 3, - 1, 2 + i, 2 - i} and {pk} = {0.5, - 0.5, 0.1 + 0.4i, 0.1 - 0.4i}

```
clc; clear; close all;
% Given zeros and poles
z = [-3, -1, 2+1i, 2-1i];
p = [0.5, -0.5, 0.1+0.4i, 0.1-0.4i];
k = 1; % Gain
b = k * poly(z);
a = poly(p);
disp('Numerator coefficients (b):');
disp(b);
disp('Denominator coefficients (a):');
disp(a);
H = tf(b, a, 1);
disp('Transfer Function H(z):');
H
figure;
zplane(b, a);
grid on;
title('Pole-Zero Plot of the Digital Filter');
```