Q1 ) part 1) % Define the system function

n = 0:30;

y = 2\*((0.4).^n).\*(n>=0) - ((0.2).^n).\*(n>=0);

% Find the impulse response

impulse = (y==1);

% Find the step response

step = cumsum(y);

% Plot the impulse response

figure

stem(n,impulse);

xlabel('n');

ylabel('Impulse Response');

title('Impulse Response of the System');

% Plot the step response

figure

stem(n,step);

xlabel('n');

ylabel('Step Response');

title('Step Response of the System');

%In the above code, n is the index of the system,

% y is the system function, impulse is the impulse response of the system,

% and step is the step response of the system.

% The stem() function is used to plot the impulse and step response,

% and the xlabel(), ylabel(), and title() functions are used to add

% labels to the plots.

%The function cumsum() is used to find the step response of the system

% by taking the cumulative sum of the system function over the index n.

\

Q1 part 2

% Define the system function

n = 0:40;

y = 0.447\*((1.618).^n).\*(n>=0) - 0.447\*((-0.618).^n).\*(n>=0);

% Find the impulse response

impulse = (y==1);

% Find the step response

step = cumsum(y);

% Plot the impulse response

figure

stem(n,impulse);

xlabel('n');

ylabel('Impulse Response');

title('Impulse Response of the System');

% Plot the step response

figure

stem(n,step);

xlabel('n');

ylabel('Step Response');

title('Step Response of the System');

Q2

t=-10:.01:10;

T=4;

fm=1/T;

x=cos(2\*pi\*fm\*t);

subplot(2,2,1);

plot(t,x);

xlabel('time');ylabel('x(t)')

title('continous time signal')

grid;

n1=-4:1:4

fs1=1.6\*fm;

fs2=2\*fm;

fs3=8\*fm;

x1=cos(2\*pi\*fm/fs1\*n1);

subplot(2,2,2);

stem(n1,x1);

xlabel('time');ylabel('x(n)')

title('discrete time signal with fs<2fm')

hold on

subplot(2,2,2);

plot(n1,x1)

grid;

n2=-5:1:5;

x2=cos(2\*pi\*fm/fs2\*n2);

subplot(2,2,3);

stem(n2,x2);

xlabel('time');ylabel('x(n)')

title('discrete time signal with fs=2fm')

hold on

subplot(2,2,3);

plot(n2,x2)

grid;

n3=-20:1:20;

x3=cos(2\*pi\*fm/fs3\*n3);

subplot(2,2,4);

stem(n3,x3);

xlabel('time');ylabel('x(n)')

title('discrete time signal with fs>2fm')

hold on

subplot(2,2,4);

plot(n3,x3)

grid;

Q3

clc;

clear all;

close all;

Ac=2; %carrier amplitude

fc=0.5; %carrier frequency

Am=.5; %message signal amplitude

fm=.05; %message signal frequency

Fs=100; %sampling rate/frequency

ka=1;%Amplitude Sensitivity

t=[0:0.1:50];%defining the time range & disseminating it into samples

ct=Ac\*cos(2\*pi\*fc\*t); %defining the carrier signal wave

mt=Am\*cos(2\*pi\*fm\*t); %defining the message signal

AM=ct.\*(1+ka\*mt); %Amplitude Modulated wave, according to the standard definition

subplot(3,1,1);%plotting the message signal wave

plot(mt);

ylabel('Message signal');

subplot(3,1,2); %plotting the carrier signal wave

plot(ct);

ylabel('carrier');

subplot(3,1,3); %plotting the amplitude modulated wave

plot(AM);

ylabel('AM signal');

Q4

clc

clear all

close all

t = 0:0.001:1; %upto 1000 samples

vm = input('Enter Amplitude (Message) = ');

vc = input('Enter Amplitude (Carrier) = ');

fM = input('Enter Message frequency = ');

fc = input('Enter Carrier frequency = ');

m = input('Enter Modulation Index = ');

msg = vm\*sin(2\*pi\*fM\*t);

subplot(3,1,1); %plotting message signal

plot(t,msg);

xlabel('Time');

ylabel('Amplitude');

title('Message ');

carrier = vc\*sin(2\*pi\*fc\*t);

subplot(3,1,2); %plotting carrier signal

plot(t,carrier);

xlabel('Time');

ylabel('Amplitude');

title('Carrier Signal');

y = vc\*sin(2\*pi\*fc\*t+m.\*cos(2\*pi\*fM\*t));

subplot(3,1,3);%plotting FM (Frequency Modulated) signal

plot(t,y);

xlabel('Time');

ylabel('Amplitude');

title('FM Signal');

Q5

% Set the parameters

Fs = 1000; % Sampling frequency

T = 1/Fs; % Sampling period

t = 0:T:1-T; % Time vector

f1 = 20; % Frequency of sinusoidal signal 1

f2 = 50; % Frequency of sinusoidal signal 2

A1 = 1; % Amplitude of sinusoidal signal 1

A2 = 0.5; % Amplitude of sinusoidal signal 2

M = 10; % Filter length

% Generate the signal composed of several sinusoidal signals

x = A1\*sin(2\*pi\*f1\*t) + A2\*sin(2\*pi\*f2\*t);

% Add noise to the signal

noise = 0.5\*randn(size(x));

y = x + noise;

% Design the M-point Moving Average Filter

b = ones(1,M)/M;

a = 1;

% Apply the filter to the signal

filtered\_signal = filter(b, a, y);

% Plot the original signal, the noisy signal, and the filtered signal

figure;

subplot(3,1,1);

plot(t, x);

title('Original Signal');

xlabel('Time (s)');

ylabel('Amplitude');

ylim([-1.5 1.5]);

subplot(3,1,2);

plot(t, y);

title('Noisy Signal');

xlabel('Time (s)');

ylabel('Amplitude');

ylim([-1.5 1.5]);

subplot(3,1,3);

plot(t, filtered\_signal);

title('Filtered Signal');

xlabel('Time (s)');

ylabel('Amplitude');

ylim([-1.5 1.5]);

Q6

% Generate input sequence x[n]

n = 0:99;

x = cos(2\*pi\*0.1\*n) + cos(2\*pi\*0.3\*n) + cos(2\*pi\*0.5\*n);

% Define fourth-order system coefficients

b = [0.0002 0.0008 0.0012 0.0008 0.0002];

a = [1.0000 -2.9843 3.8840 -2.4419 0.5421];

% Apply input sequence to fourth-order system

y = filter(b, a, x);

% Define first-stage system coefficients

b1 = [0.0201 0.0201];

a1 = [1.0000 -0.9598];

% Apply input sequence to first-stage system

y1 = filter(b1, a1, x);

% Define second-stage system coefficients

b2 = [0.0392 0.0392];

a2 = [1.0000 -0.9216];

% Apply first-stage output to second-stage system

y2 = filter(b2, a2, y1);

% Plot input and output sequences

subplot(3,1,1);

stem(n, x);

title('Input sequence x[n]');

xlabel('n');

ylabel('x[n]');

subplot(3,1,2);

stem(n, y);

title('Output sequence y[n] (fourth-order system)');

xlabel('n');

ylabel('y[n]');

subplot(3,1,3);

stem(n, y2);

title('Output sequence y2[n] (two-stage system)');

xlabel('n');

ylabel('y2[n]');

Q8

% Define the filter order and cutoff frequency

n = 2;

Wn = 0.2;

% Create the Butterworth filter

[b,a] = butter(n, Wn);

% Define the square signal

t = 0:0.01:1;

x = square(2\*pi\*t);

% Apply the Butterworth filter to the square signal

y = filter(b, a, x);

% Plot the filtered signal

plot(t, y);

xlabel('Time (s)');

ylabel('Amplitude');

title('Response of Butterworth Filter to Square Signal');

%In the above code, butter() is a built-in MATLAB function that

% generates the coefficients of a Butterworth filter with a given

% order and cutoff frequency. The filter() function is used to

% apply the filter to the square signal, and the plot() function is

% used to plot the filtered signal. The xlabel(), ylabel(), and title()

% functions are used to add labels to the plot.

Q9

% Set the parameters of the raised cosine filter

Fs = 1000; % Sampling frequency in Hz

T = 1; % Symbol period in seconds

beta = 0.5; % Roll-off factor

span = 10; % Filter span in symbols

t = linspace(-span/2, span/2, span\*Fs\*T+1); % Time vector

% Design the raised cosine filter

h = rcosdesign(beta, span, Fs, 'sqrt');

% Compute the frequency response of the filter

f = linspace(-Fs/2, Fs/2, length(t)); % Frequency vector

H = fftshift(fft(h));

% Plot the frequency response of the filter

plot(f, abs(H));

title('Frequency Response of Raised Cosine Filter');

xlabel('Frequency (Hz)');

ylabel('Magnitude');

Q10

% Set the parameters of the raised cosine filter

Fs = 1000; % Sampling frequency in Hz

T = 1; % Symbol period in seconds

beta = 0.5; % Roll-off factor

span = 10; % Filter span in symbols

% Design the raised cosine filter

h = rcosdesign(beta, span, Fs, 'sqrt');

% Compute the impulse response of the filter

impulse = [1 zeros(1, span\*Fs\*T)]; % Impulse signal

h\_impulse = filter(h, 1, impulse); % Filter the impulse signal

% Plot the impulse response of the filter

t = linspace(0, span\*T, length(h\_impulse)); % Time vector

plot(t, h\_impulse);

title('Impulse Response of Raised Cosine Filter');

xlabel('Time (s)');

ylabel('Amplitude');

Q11

% Define the signal parameters

fs = 1000; % sampling frequency

t = 0:1/fs:1; % time vector

f = 50; % frequency of the signal

A = 1; % amplitude of the signal

phi = pi/4; % phase of the signal

% Generate the sinusoidal signal

x = A\*cos(2\*pi\*f\*t + phi);

% Add noise to the signal

noise = randn(size(x))\*0.2; % Gaussian noise with standard deviation of 0.2

x\_noisy = x + noise;

% Apply a low-pass filter to the noisy signal to remove high-frequency noise

[b, a] = butter(3, 0.1); % Butterworth filter of order 3 and

%cutoff frequency of 0.1

x\_filtered = filter(b, a, x\_noisy);

% Plot the original, noisy and filtered signals

figure;

subplot(3,1,1);

plot(t,x);

xlabel('Time (s)');

ylabel('Amplitude');

title('Original Signal');

subplot(3,1,2);

plot(t,x\_noisy);

xlabel('Time (s)');

ylabel('Amplitude');

title('Noisy Signal');

subplot(3,1,3);

plot(t,x\_filtered);

xlabel('Time (s)');

ylabel('Amplitude');

title('Filtered Signal');

%In the above code, the randn() function is used to generate

% Gaussian noise with a standard deviation of 0.2, which is added

% to the sinusoidal signal. The butter() function is used to generate

% the coefficients of a Butterworth low-pass filter of order 3 and

% cutoff frequency of 0.1. The filter() function is used to apply the

% filter to the noisy signal. The subplot() and plot() functions are used

% to plot the original, noisy and filtered signals.

Q12

% Define the signal parameters

fs = 1000; % sampling frequency

t = 0:1/fs:1; % time vector

f = 50; % frequency of the signal

A = 1; % amplitude of the signal

phi = pi/4; % phase of the signal

% Generate the sine signal

x = A\*cos(2\*pi\*f\*t + phi);

% Amplitude Shift Keying (ASK) modulation

% Define the modulation index

m = 2;

ask = m\*x;

% Phase Shift Keying (PSK) modulation

% Define the modulation index

m = pi/4;

psk = x.\*cos(m\*t);

% Frequency Shift Keying (FSK) modulation

% Define the modulation index

m = 50;

fsk = x.\*cos(2\*pi\*m\*t);

% Plot the original, ASK, PSK and FSK signals

figure;

subplot(4,1,1);

plot(t,x);

xlabel('Time (s)');

ylabel('Amplitude');

title('Original Signal');

subplot(4,1,2);

plot(t,ask);

xlabel('Time (s)');

ylabel('Amplitude');

title('ASK Modulation');

subplot(4,1,3);

plot(t,psk);

xlabel('Time (s)');

ylabel('Amplitude');

title('PSK Modulation');

subplot(4,1,4);

plot(t,fsk);

xlabel('Time (s)');

ylabel('Amplitude');

title('FSK Modulation');

%In the above code, x is the original sine signal and ASK, PSK

% and FSK are the modulated signals. The plot() function is used to

% plot the original, ASK, PSK and FSK signals.

% The xlabel(), ylabel(), and title() functions are used to add

% labels to the plots.

Q13

% Define the numerator and denominator of the system function

num = [2 1];

den = [1 -0.6];

% Compute the frequency response of the system

[H, w] = freqz(num, den);

% Plot the magnitude and phase response

figure;

subplot(2,1,1);

plot(w, abs(H));

xlabel('Frequency (rad/s)');

ylabel('Magnitude');

title('Magnitude Response');

subplot(2,1,2);

plot(w, angle(H));

xlabel('Frequency (rad/s)');

ylabel('Phase (rad)');

title('Phase Response');

%In the above code, num and den are the numerator and

% denominator of the system function. The freqz() function

% is used to compute the frequency response of the system.

% The plot() function is used to plot the magnitude and phase

% response of the system. The xlabel(), ylabel(), and title()

% functions are used to add labels to the plots.

Q14

% Define the numerator and denominator of the transfer function

num = [1 -0.9];

den = [1 -0.8];

% Find the impulse response of the system

impulse\_response = impz(num, den);

% Plot the impulse response

figure;

stem(impulse\_response);

xlabel('n');

ylabel('Impulse Response');

title('Impulse Response of the System');

%Here is an example of how we might write a MATLAB script to find the

% impulse response of a discrete system described by the transfer function

% H(z) = (1-0.9z^-1)/(1-0.8z^-1):

%In the above code, num and den are the numerator and

% denominator of the transfer function of the system.

% The impz() function is used to find the impulse response of the system,

% this function generates the impulse response of the system based on the

% given transfer function. The stem() function is used to plot the impulse

% response, and the xlabel(), ylabel(), and title() functions are used to

% add labels to the plot.

Q15

% Define the two signals

x = randn(1,100);

y = randn(1,100);

% Find the cross-correlation of the signals

rxy = xcorr(x, y);

% Plot the cross-correlation

figure;

plot(rxy);

xlabel('Lag');

ylabel('Cross-correlation');

title('Cross-correlation of Signals x and y');

%In the above code, x and y are the two signals for which

% the cross-correlation is to be found. The xcorr() function

% is used to find the cross-correlation of the signals. The plot()

% function is used to plot the cross-correlation, and the xlabel(),

% ylabel(), and title() functions are used to add labels to the plot.

Q16

% Define the original signal

fs = 1000; % sampling frequency

t = 0:1/fs:1; % time vector

x = cos(2\*pi\*100\*t) + cos(2\*pi\*200\*t); % original signal

% Define the decimation factor

M = 4;

% Perform decimation

y = decimate(x, M);

% Plot the original and decimated signals

figure;

subplot(2,1,1);

plot(t,x);

xlabel('Time (s)');

ylabel('Amplitude');

title('Original Signal');

subplot(2,1,2);

plot(t(1:M:end),y);

xlabel('Time (s)');

ylabel('Amplitude');

title('Decimated Signal');

%In the above code, fs is the sampling frequency of the original signal,

% t is the time vector, and x is the original signal.

% The decimate() function is used to perform the decimation process on

% the signal. The decimation factor M is defined which is used to reduce

% the number of samples in the signal by a factor of M. The plot()

% function is used to plot the original and decimated signals, and

% the xlabel(), ylabel(), and title() functions are used to add

% labels to the plots.

%Also, note that the decimate() function uses a low-pass filter

% to remove high-frequency components before downsampling to reduce

% the aliasing effect. If you want to use a different

Q17

% Define the original signal

fs = 1000; % sampling frequency

t = 0:1/fs:1; % time vector

x = cos(2\*pi\*100\*t) + cos(2\*pi\*200\*t); % original signal

% Define the interpolation factor

L = 4;

% Perform interpolation

y = interp(x, L);

% Plot the original and interpolated signals

fs\_new = L\*fs;

t\_new = (0:1/fs\_new:(length(y)-1)/fs\_new);

figure;

subplot(2,1,1);

plot(t,x);

xlabel('Time (s)');

ylabel('Amplitude');

title('Original Signal');

subplot(2,1,2);

plot(t\_new,y);

xlabel('Time (s)');

ylabel('Amplitude');

title('Interpolated Signal');

%In the above code, fs is the sampling frequency of the original signal,

% t is the time vector, and x is the original signal.

% The interp() function is used to perform the interpolation process

% on the signal. The interpolation factor L is defined which is used

% to increase the number of samples in the signal by a factor of L.

% The plot() function is used to plot the original and interpolated

% signals, and the xlabel(), ylabel(), and title() functions are used

% to add labels to the plots.

%Also, note that the interp() function uses zero-order hold interpolation,

% which simply inserts L-1 zeros between each sample.

Q18

% Define the original signal

fs = 1000; % sampling frequency

t = 0:1/fs:1; % time vector

x = cos(2\*pi\*100\*t) + cos(2\*pi\*200\*t); % original signal

% Define the interpolation factor

L = 4;

% Perform interpolation

y = resample(x, L, 1);

% Plot the original and interpolated signals

fs\_new = L\*fs;

t\_new = (0:1/fs\_new:(length(y)-1)/fs\_new);

figure;

subplot(2,1,1);

plot(t,x);

xlabel('Time (s)');

ylabel('Amplitude');

title('Original Signal');

subplot(2,1,2);

plot(t\_new,y);

xlabel('Time (s)');

ylabel('Amplitude');

title('Interpolated Signal');

%In this code, the resample() function is used to perform the

% sampling rate conversion. The first argument is the original signal x,

% the second argument is the interpolation factor L and the third argument

% is the decimation factor 1.

%The resample() function uses a polyphase implementation of the

% sample-rate conversion which is an efficient and accurate method

% for converting the sample rate of a signal.

% The resample() function also includes a low-pass filter

% that removes high-frequency components before upsampling to

% reduce the aliasing effect.

%Note that the output signal's time vector is also

% adjusted according to the new sampling frequency fs\_new.

Q19

fs=1;

N = 1000; %number of samples

n = 0:N-1; %sample indices

x = zeros(1,N); %initialize signal

x(round(0.7\*N) + 1) = 1; %set value at n=0.7

X = fft(x); %calculate Fourier transform

f = (0:N-1)\*(fs/N); %frequency axis

figure;

plot(f, abs(X)); %plot magnitude spectrum

xlabel('Frequency (Hz)');

ylabel('Magnitude');

title('Magnitude Spectrum of x[n]');

xlim([0 1])

Q20

% Define the signal

n = 1:16;

x = (n<=8).\*n;

% Compute the DFT of the signal

X = fft(x);

% Time-delay property

k = 0:15;

delay = 5;

X\_delayed = X.\*exp(-1j\*2\*pi\*k\*delay/16);

x\_delayed = ifft(X\_delayed);

% Frequency shifting property

k\_shifted = k-8;

X\_shifted = X.\*exp(-1j\*2\*pi\*k\_shifted/16);

x\_shifted = ifft(X\_shifted);

% Modulation property

k\_modulated = k-8;

fc = 2;

X\_modulated = X.\*exp(1j\*2\*pi\*fc\*k\_modulated/16);

x\_modulated = ifft(X\_modulated);

% Time inversion property

k\_inverted = k;

X\_inverted = flip(X);

x\_inverted = ifft(X\_inverted);

% Plot the original and transformed signals

subplot(2,3,1);

stem(n,x);

title('Original signal');

subplot(2,3,2);

stem(k,abs(X));

title('DFT of original signal');

subplot(2,3,3);

stem(n,x\_delayed);

title('Time-delayed signal');

subplot(2,3,4);

stem(k,abs(X\_delayed));

title('DFT of time-delayed signal');

subplot(2,3,5);

stem(n,x\_shifted);

title('Frequency-shifted signal');

subplot(2,3,6);

stem(k,abs(X\_shifted));

title('DFT of frequency-shifted signal');

figure;

subplot(2,2,1);

stem(n,x);

title('Original signal');

subplot(2,2,2);

stem(k\_modulated,abs(X\_modulated));

title('DFT of modulated signal');

subplot(2,2,3);

stem(n,x\_inverted);

title('Time-inverted signal');

subplot(2,2,4);

stem(k\_inverted,abs(X\_inverted));

title('DFT of time-inverted signal');