1st exp: To verify removal of noise by Moving Mean/ Average

```
clear
close all
clc
load handel
y= y(1:256);
figure;
plot(y)
title('Audio File With Noise');
ysmm= movmean(y, 3, 'endpoints', 'discard' );
figure;
plot(ysmm)
title('Audio File After Filtering');
```

2nd exp: To remove the noise (salt pepper noise) from Image data using averaging filter and median filters

```
a = imread('flower.jpg');
figure;
imshow(a);
title('Original Image');
j = imnoise(a, 'salt & pepper', 0.02);
figure;
imshow(j);
title('Image with Salt and Pepper Noise');
k = zeros(size(j));
for c = 1:size(j, 3)
 k(:, :, c) = filter2(fspecial('average', 3), double(j(:, :, c)));
end
k = mat2gray(k);
figure;
imshow(k);
title('Filtered Image with Average Filter');
```

3rd exp: Design of FIR adaptive filter as an Equalization filter

```
% Define Parameters
fs = 44100; % Sample rate
worN = logspace(log10(20), log10(20^(3.3)), 1000); % Frequencies to plot

% Low-Pass Filter
lpFilt = designfilt('lowpassiir', 'FilterOrder', 2, ...
                    'HalfPowerFrequency', 10000/(fs/2), 'DesignMethod', 'butter');

% Low-Shelf Filter (Manual Design)
lowShelfFreq = 200; % Low-shelf frequency
lowShelfGain = 2;   % Gain (linear scale)
[bLowShelf, aLowShelf] = shelf(lowShelfFreq, lowShelfGain, fs);

% Notch Filter
notchFreq = 880; % Notch frequency
notchQ = 0.707;  % Quality factor
[bNotch, aNotch] = iirnotch(notchFreq/(fs/2), notchFreq/(fs/2) / notchQ);

% Frequency Responses
[H1, w1] = freqz(lpFilt, worN, fs);
[H2, w2] = freqz(bLowShelf, aLowShelf, worN, fs);
[H3, w3] = freqz(bNotch, aNotch, worN, fs);

% Combined Response
H_combined = H1 .* H2 .* H3;

% Plot Equalizer Magnitude Response
figure;
semilogx(worN, 20*log10(abs(H_combined))); % Plot magnitude response in dB
```

```
grid on;
title('Adaptive Equalization Filter');
xlabel('Frequency (Hz)');
ylabel('Magnitude (dB)');

% --- Custom Low-Shelf Filter Function ---
function [b, a] = shelf(f0, gain, fs)
    A = sqrt(gain); % Gain in amplitude
    omega = 2 * pi * f0 / fs;
    alpha = sin(omega) / (2 * sqrt(A)); % Bandwidth

    b0 = A * ((A + 1) - (A - 1) * cos(omega) + 2 * sqrt(A) * alpha);
    b1 = 2 * A * ((A - 1) - (A + 1) * cos(omega));
    b2 = A * ((A + 1) - (A - 1) * cos(omega) - 2 * sqrt(A) * alpha);
    a0 = (A + 1) + (A - 1) * cos(omega) + 2 * sqrt(A) * alpha;
    a1 = -2 * ((A - 1) + (A + 1) * cos(omega));
    a2 = (A + 1) + (A - 1) * cos(omega) - 2 * sqrt(A) * alpha;

    b = [b0, b1, b2] / a0; % Normalize coefficients
    a = [1, a1 / a0, a2 / a0];
end
```

4th exp: Develop OFDM standard complaint waveforms using FFT and other blocks

```
fs = 20;
fc = 1.5625;
n=28;

x1t=exp(j*2*pi*fc*[0:n-1]/fs);
x1f = fft(x1t,n);
subplot(2,1,1);
plot([-n/2:n/2-1]*fs/n, fftshift(abs(x1f)));
xlabel('frequency,mhz');
ylabel('amplitude');
title('frequency response of complex sine signal');

x2f = [zeros(1,n/2) 0 zeros(1,9) 1 zeros(1,n/2-10-1)];
% rest all zeros;
x2t = n*ifft(fftshift(x2f));
diff = x2t-x1t;
err = diff.*diff/length(diff);
ht=sinc([-20:20]/3);
hf=fft(ht,1024);
subplot(2,1,2);
plot([-512:511]*30/1024, fftshift(abs(hf)));
xlabel('frequency,mhz');
ylabel('amplitude');
title('frequency response of sinc filter');
```

5th exp: Digital modulation and demodulation in AWGN and other channel setting.

```
Signal = randi([0 3], 1000, 1);
BER = [];
n = 20;
for i = 1:n
    Errors = signal_awgn(Signal, i);
    BER = [BER Errors / 1000];
end
semilogy(1:n, BER, 'o-');
xlabel('SNR (dB)');
ylabel('Bit Error Rate (BER)');
grid on;
title('BER vs. SNR for 4-PSK modulation');

function Errors = signal_awgn(Signal, SNR)
```

```
    TxSignal = pskmod(Signal, 4, pi/4);
    RxSignal = awgn(TxSignal, SNR, 'measured');
    Signal2 = pskdemod(RxSignal, 4, pi/4);
    Errors = symerr(Signal, Signal2);
end
```

6th exp: Compress image using different standards such as JPEG, JPEG 2000 and SPIHT and EZW codes

6.a) JPEG Compression:

```
clc;
clear all;
close all;
original_img = imread('flower.jpg');
figure, imshow(original_img);
title('Original Image');
imwrite(original_img, 'new_img.jpeg', 'Quality', 10);
info_original = imfinfo('flower.jpg');
info_compressed = imfinfo('new_img.jpeg');
compressed_img = imread('new_img.jpeg');
figure, imshow(compressed_img);
title('Compressed Image');
Original_size = info_original.FileSize;
Compressed_size = info_compressed.FileSize;
Comp_ratio = Original_size / Compressed_size;
fprintf('Original Image Size: %d bytes\n', Original_size);
fprintf('Compressed Image Size: %d bytes\n', Compressed_size);
fprintf('Compression Ratio: %.2f\n', Comp_ratio);
```

6.b) SPIHT Compression:

```
clc;
close all;
clear all;
X = imread('flower.jpg');
X_resized = imresize(X, [128, 128]);
[cratio, bpp] = wcompress('c', X_resized, 'imag_c.wtc', 'spiht');
Xc = wcompress('u', 'imag_c.wtc');
delete('imag_c.wtc');
figure;
subplot(1, 2, 1);
imshow(X_resized);
title('Original Image');
subplot(1, 2, 2);
imshow(Xc);
title('Compressed Image');
fprintf('Compression Ratio: %.2f\n', cratio);
fprintf('Bits per Pixel (BPP): %.2f\n', bpp);
```

6.c) E2Z Compression:

```
clc
close all;
clear all;
X = imread("flower.jpg");
X_resized = imresize(X, [128, 128]);
[cratio, bpp] = wcompress('c', X_resized, 'imag.c.wtc', 'ezw');
Xc = wcompress('u', 'imag.c.wtc');
delete('imag.c.wtc');
figure;
subplot(1, 2, 1);
imshow(X_resized);
title('Original Image - 4k6');
subplot(1, 2, 2);
imshow(Xc);
title('Compressed Image - 4k6');
fprintf('Compression Ratio: %.2f\n', cratio);
fprintf('Bits per Pixel (BPP): %.2f\n', bpp);
```

7th exp: Compress video signal using standards H216, H263 and MPEG4 and H264 Codes

```
clc;
clear all;
close all;
v = VideoReader('ccbp_selfintro.mp4');
disp(['Number of Frames: ', num2str(v.NumFrames)]);
disp(['Frame Rate: ', num2str(v.FrameRate)]);
fo = [];
frameCount = 0;
while hasFrame(v) % Loop through all frames
 frame = readFrame(v); % Read current frame
 ch = rgb2gray(frame); % Convert frame to grayscale
 [n, m] = size(ch); % Get frame dimensions
 op = [];
 cnt = 1;
 temp = ch(1, 1);
 for i = 1:n
 for j = 1:m
 if ch(i, j) == temp
 cnt = cnt + 1;
 else
 op = [op, cnt, temp]; % Store run length and pixel value
 cnt = 1; % Reset count
 temp = ch(i, j); % Update pixel value
 end
 end
 end
 op = [op, cnt, temp];
 fo = [fo, op];
 frameCount = frameCount + 1;
end
[m, n, p] = size(frame);
originalSize = m * n * p * frameCount;
compressedSize = length(fo);
compressionRatio = originalSize / compressedSize;
% Display results
disp(['Compression Ratio: ', num2str(compressionRatio)]);
disp(['Total Frames Processed: ', num2str(frameCount)]);
```

8th exp: Evaluate different performance aspects required for lossy and lossless compression methods

```
clc;
clear all;
close all;
rgb = imread('flower.jpg');
imshow(rgb);
title('Original Image');
imwrite(rgb, 'mypeppers.tif');
imwrite(rgb, 'test_image.study_013.tif');
bytes_in_memory_peppers = numel(rgb);
info_png = imfinfo('peppers.png');
bytes_on_disk_peppers_png = info_png.FileSize;
compression_ratio_peppers_png = bytes_in_memory_peppers / bytes_on_disk_peppers_png;
peppers2 = imread('peppers.png');
is_equal_png = isequal(rgb, peppers2);
url = 'https://blogs.mathworks.com/images/steve/2012/plot_screen_shot.png';
rgb_plot = imread(url);
bytes_in_memory_plot = numel(rgb_plot);
info_plot = imfinfo(url);
bytes_on_disk_plot = info_plot.FileSize;
compression_ratio_plot = bytes_in_memory_plot / bytes_on_disk_plot;
imwrite(rgb_plot, 'plot_screen_shot.jpg');
info_jpg = imfinfo('plot_screen_shot.jpg');
```

```
bytes_on_disk_peppers_jpg = info_jpg.FileSize;
compression_ratio_peppers_jpg = bytes_in_memory_peppers / bytes_on_disk_peppers_jpg;
figure;
subplot(1,2,1);
imshow('flower.jpg');
title('Original PNG');
subplot(1,2,2);
imshow('flower.jpg');
title('JPEG Compressed');
rgb_plot_compressed = imread('plot_screen_shot.jpg');
figure;
subplot(2,2,1);
imshow(rgb_plot);
title('Original Image');
axis([0 800 0 600]);
subplot(2,2,2);
imshow(rgb_plot_compressed);
title('JPEG Compressed Image');
axis([0 800 0 600]);
disp(['Compression ratio for PNG: ', num2str(compression_ratio_peppers_png)]);
disp(['Compression ratio for JPEG: ', num2str(compression_ratio_peppers_jpg)]);
disp(['Compression ratio for Plot Image: ', num2str(compression_ratio_plot)]);
```

9th exp: Perform Automatic gain and frequency control for an Audio signal.

```
clear;
clc;
dt = 0.001;
t = 0:dt:20;
A = 10;
f = 0.25;
xin = A * sin(2 * pi * f .* t);
L = round(10 * f * (1 / dt));
powerTarget = 0.2;
yout = zeros(1, length(xin));
gainAGC = ones(1, length(xin));
for n = 1:length(t) - 1
 yout(n) = gainAGC(n) * xin(n);
 gainAGC(n + 1) = gainAGC(n) * (1 - (1 / L) * (yout(n)^2 - powerTarget));
end

figure;
subplot(2, 1, 1);
plot(t, xin, '-b', 'LineWidth', 1.5); hold on;
plot(t, yout, '-r', 'LineWidth', 1.5);
grid on; zoom on;
xlabel('Time (s)'); ylabel('Amplitude');
legend('Input Signal (x_{in})', 'Output Signal (y_{out})');
title('AGC Input and Output');

subplot(2, 1, 2);
plot(t(1:end-1), gainAGC(1:end-1), '-k', 'LineWidth', 1.5);
grid on; zoom on;
xlabel('Time (s)'); ylabel('Gain');
title('Gain of AGC');
```

10th exp: Audio compression algorithms (Using Sub band coding and Linear predictive code)

```
close all;
clear all;
clc;
%% Load Built-In Audio Data
load handel.mat; % Load built-in audio (it gives variables 'y' and 'Fs')
x = y'; % Transpose to row vector
fs = Fs; % Sampling frequency
lnx = length(x); % Length of input signal
L = 2; % Decimation/interpolation factor
len = 25; % Length of FIR filter
%% Sub-band Filters
wc = 1 / L; % Cut-off frequency (normalized)
freq = -pi : 2 * pi / (lnx - 1) : pi; % Frequency vector
% Low-pass filter
lp = fir1(len - 1, wc, 'low');
% High-pass filter
hp = fir1(len - 1, wc, 'high');
%% Filter the Input Signal
yl = conv(x, lp); % Low-pass filter the input
yh = conv(x, hp); % High-pass filter the input
%% Time-Domain Plot
figure(1);
subplot(3, 1, 1);
plot(x); axis([0 lnx min(x) max(x)]);
ylabel('Amplitude'); title('Speech Signal and Filters in Time Domain');
subplot(3, 1, 2);
stem(lp); axis([0 length(lp) min(lp) - 0.1 max(lp) + 0.1]);
ylabel('Low-pass Filter Coefficients');
subplot(3, 1, 3);
stem(hp); axis([0 length(hp) min(hp) - 0.1 max(hp) + 0.1]);

ylabel('High-pass Filter Coefficients');
%% Frequency-Domain Plot
X = fftshift(fft(x, lnx)); % FFT of input signal
Lp = fftshift(fft(lp, lnx)); % FFT of low-pass filter
Hp = fftshift(fft(hp, lnx)); % FFT of high-pass filter
YL = fftshift(fft(yl, lnx)); % FFT of low-pass filtered signal
YH = fftshift(fft(yh, lnx)); % FFT of high-pass filtered signal
figure(2);
subplot(3, 2, 1);
plot(freq / pi, abs(X)); ylabel('|X|');
title('Frequency-Domain Representation of Speech and Bands');
axis([-1 1 min(abs(X)) max(abs(X))]);
subplot(3, 2, 3);
plot(freq / pi, abs(Lp), 'g'); ylabel('|Lp|');
axis([-1 1 min(abs(Lp)) max(abs(Lp))]);
subplot(3, 2, 4);
plot(freq / pi, abs(Hp), 'g'); ylabel('|Hp|');
axis([-1 1 min(abs(Hp)) max(abs(Hp))]);
subplot(3, 2, 5);
plot(freq / pi, abs(YL), 'b'); ylabel('|YL|');
axis([-1 1 min(abs(YL)) max(abs(YL))]);
legend('Low Band');
subplot(3, 2, 6);
plot(freq / pi, abs(YH), 'r'); ylabel('|YH|');
axis([-1 1 min(abs(YH)) max(abs(YH))]);
legend('High Band');
%% Downsampling
ydl = yl(1:L:end); % Low-band decimation
ydh = yh(1:L:end); % High-band decimation
%% Reconstruct Bands
s0 = conv(ydl, lp); % Low-pass reconstruction
```

```
s1 = conv(ydl, hp); % High-pass reconstruction
s2 = conv(ydh, lp); % Low-pass reconstruction (high-band)
s3 = conv(ydh, hp); % High-pass reconstruction (high-band)
b0 = s0(1:L:end); % Decimate reconstructed low-pass
b1 = s1(1:L:end); % Decimate reconstructed high-pass
b2 = s2(1:L:end); % Decimate high-pass low-band
b3 = s3(1:L:end); % Decimate high-pass high-band
%% Frequency Plot of Four Bands
figure(3);
subplot(4, 1, 1);
plot(freq / pi, abs(fftshift(fft(b0, lnx)))); ylabel('|B0|');
title('Four Bands in Frequency Domain');
axis([-1 1 min(abs(fft(b0))) max(abs(fft(b0)))]);

subplot(4, 1, 2);
plot(freq / pi, abs(fftshift(fft(b1, lnx)))); ylabel('|B1|');
axis([-1 1 min(abs(fft(b1))) max(abs(fft(b1)))]);
subplot(4, 1, 3);
plot(freq / pi, abs(fftshift(fft(b2, lnx)))); ylabel('|B2|');
axis([-1 1 min(abs(fft(b2))) max(abs(fft(b2)))]);
subplot(4, 1, 4);
plot(freq / pi, abs(fftshift(fft(b3, lnx)))); ylabel('|B3|');
axis([-1 1 min(abs(fft(b3))) max(abs(fft(b3)))]);
%% Reconstruction Using Filters
reconst_fil = L * fir1(len - 1, 1 / L); % Reconstruction filter with cutoff 1/L
sb0 = conv(reconst_fil, b0);
sb1 = conv(reconst_fil, b1);
sb2 = conv(reconst_fil, b2);
sb3 = conv(reconst_fil, b3);
Slow = sb0 + sb1; % Combine low-band signals
Shigh = sb2 + sb3; % Combine high-band signals
subll = zeros(1, length(Slow) * 2); % Interpolation for low-band
subhh = zeros(1, length(Shigh) * 2); % Interpolation for high-band
subll(1:L:end) = Slow;
subhh(1:L:end) = Shigh;
sub_recon = conv(reconst_fil, subll) + conv(reconst_fil, subhh); % Final reconstruction
%% Final Comparison
figure(4);
subplot(3, 1, 1);
plot(freq / pi, abs(X)); ylabel('|X|');
title('Final Bands and Reconstruction');
axis([-1 1 min(abs(X)) max(abs(X))]);
subplot(3, 1, 2);
plot(freq / pi, abs(fftshift(fft(Slow, lnx)))); ylabel('|Low Band|');
axis([-1 1 min(abs(fft(Slow))) max(abs(fft(Slow)))]);
subplot(3, 1, 3);
plot(freq / pi, abs(fftshift(fft(sub_recon, lnx))), 'r'); ylabel('|Synthesized|');
axis([-1 1 min(abs(fft(sub_recon))) max(abs(fft(sub_recon)))]);
legend('Synthesized Band');
```