Expt 1

```
% Generalised code for Linear and Circular Convolution

% --- Input Sequences (you can change these)

x = input('Enter the First sequence x[n] as a vector (e.g., [1 2 3]): ');

h = input('Enter the Second sequence h[n] as a vector (e.g., [4 5 6]): ');


%% Linear Convolution

y_linear = conv(x, h); % Linear convolution

disp('Linear Convolution Output:');

disp(y_linear);


% Plot Linear Convolution

figure;

stem(0:length(y_linear)-1, y_linear, 'filled');

title('Linear Convolution Output');

xlabel('n');

ylabel('Amplitude');

grid on;


%% Circular Convolution

% Determine length for circular convolution (use length of longer sequence)

N = max(length(x), length(h));


% Zero-pad both sequences to length N

x_circ = [x, zeros(1, N - length(x))];

h_circ = [h, zeros(1, N - length(h))];


% Perform Circular Convolution

y_circ = cconv(x_circ, h_circ, N);

disp('Circular Convolution Output:');

disp(y_circ);


% Plot Circular Convolution
```

```
figure;
stem(0:N-1, y_circ, 'filled');
title('Circular Convolution Output');
```

expt 2

CODE : DFT

```
N = input('Enter the value of N: ');
x = input('Enter the sequence: ');
x = x(:).';
n = 0:N-1;
k = 0:N-1;
WN = exp(-1j * 2 * pi / N);
nk = n' * k;
WNnk = WN .^ nk;
Xk = x * WNnk;
disp('N point DFT is X[k] = ');
disp(Xk);
```

OUTPUT :

```
Enter the value of N: 4
Enter the sequence: [1 2 3 4]
N point DFT is X[k] =
10.0000 + 0.0000i -2.0000 + 2.0000i -2.0000 - 0.0000i -2.0000 - 2.0000i
```

CODE : IDFT

```
WN_inv = exp(1j * 2 * pi / N);
WNnk_inv = WN_inv .^ nk;
x_reconstructed = (1/N) * (Xk * WNnk_inv);
disp('Reconstructed sequence after IDFT x[n] = ');
disp(x_reconstructed);
```

OUTPUT

```
Reconstructed sequence after IDFT x[n] =
1.0      - 0.0000i 2.0000 - 0.0000i 3.0000 - 0.0000i 4.0000 + 0.0000i
```

Expt 3

MATLAB code : Overlap – Add and Overlap – Save Method

```
x = [1 2 3 4 5 6 7 8];
h = [1 2];
M = length(h);
L = 4; %Block length
N = M + L - 1;
X_blocks = buffer(x,L);
y_total = zeros(1, size(X_blocks,2)*L + M - 1);
for i = 1:size(X_blocks,2)
x_block = [X_blocks(:,i)' zeros(1, N - L)];
h_pad = [h zeros(1, N - M)];
y_block = ifft(fft(x_block).*fft(h_pad));
start = (i-1)*L + 1;
y_total(start:start+N-1) = y_total(start:start+N-1) + y_block;
end
disp('Output using Overlap-Add Method:');
disp(y_total);
stem(y_total);
title('Overlap-Add Output');
xlabel('n');
ylabel('Amplitude');
grid on;
x_pad=[zeros (1,M-1)x];
num_blocks=ceil((length(x_pad)-)/L);
y_ola=[];
for i =1: num_blocks
start=(i-1)*L+1;
stop = start +N-1;
if stop > length (x_pad)
x_seg = [x_pad (start:end)zeros(1,stop-length(x_pad))];
else
x_seg =x_pad(start:stop);
end
h_pad =[h zeros(1,N-M)];
y_block= ifft(fft(x_seg).*fft( h_pad));
y_ols =[y_ols y_block(M:end)];
end
```

expt 4

```
clc; clear; close all;

fs = 1000;
t = 0:1/fs:1-1/fs;

x = sin(2*pi*50*t) + 0.5 * sin(2*pi*150*t) + 0.25 * sin(2*pi*300*t);

[bl, al] = butter(4, 100/(fs/2), 'low');
yl = filter(bl, al, x);

[bh, ah] = butter(4, 100/(fs/2), 'high');
yh = filter(bh, ah, x);

[bb, ab] = butter(4, [100 200]/(fs/2), 'bandpass');
yb = filter(bb, ab, x);

w0 = 150/(fs/2);
Q = 30;
[bn, an] = iirnotch(w0, w0/Q);
yn = filter(bn, an, x);

delay = 50;
g_comb = 0.7;
bc = zeros(1, delay+1); bc(1) = 1; bc(end) = g_comb;
ac = 1;
yc = filter(bc, ac, x);

g_ap = 0.8;
bap = [g_ap, 1];
aap = [1, g_ap];
yap = filter(bap, aap, x);

f0 = 150;
r = 0.95;
omega = 2*pi*f0/fs;
br = [1];
ar = [1, -2*r*cos(omega), r^2];
yr = filter(br, ar, x);

filters = {
    'Low-pass Filter', yl, bl, al;
    'High-pass Filter', yh, bh, ah;
    'Band-pass Filter', yb, bb, ab;
    'Notch Filter', yn, bn, an;
    'Comb Filter', yc, bc, ac;
    'All-pass Filter', yap, bap, aap;
    'Digital Resonator', yr, br, ar;
};

for i = 1:size(filters,1)
    name = filters{i, 1};
    y = filters{i, 2};
```

```
    b = filters{i, 3};
    a = filters{i, 4};

    figure('Name', [name ' - Time Domain']);
    plot(t, x, 'k--', 'DisplayName', 'Original Signal'); hold on;
    plot(t, y, 'DisplayName', name);
    title([name ' - Time Domain']);
    xlabel('Time (s)');
    ylabel('Amplitude');
    legend('show');
    grid on;

    figure('Name', [name ' - Frequency Response']);
    [h, w] = freqz(b, a, 1024, fs);
    plot(w, 20*log10(abs(h)), 'LineWidth', 1.2);
    xlabel('Frequency');
    ylabel('Magnitude');
    grid on;
end
```

expt 5

```
clc;
clear;
close all;

Ap = 1;
fp = 1000;
As = 60;
fs = 6000;
Fs = 15000;

Wp = 2 * Fs * tan(pi * fp / Fs);
Ws = 2 * Fs * tan(pi * fs / Fs);

[N, Wc] = buttord(Wp, Ws, Ap, As, 's');

[ba, aa] = butter(N, Wc, 's');

[bd, ad] = bilinear(ba, aa, Fs);

fprintf("Butterworth LPF Design Using BLT Method\n");
fprintf("--------------------------------------------------\n");
fprintf("Filter Order: %d\n", N);
fprintf("Analog Cutoff Frequency: %.2f rad/sec\n", Wc);
fprintf("Digital Cutoff Frequency: %.2f Hz\n", (Wc/(2*pi)));

[H, f] = freqz(bd, ad, 1024, Fs);
figure;
plot(f, 20*log10(abs(H)), 'LineWidth', 1.5);
grid on;
xlabel('Frequency (Hz)');
ylabel('Magnitude (dB)');
title(sprintf('Low-pass Butterworth Filter (Order = %d)', N));
ylim([-80, 5]);

fvtool(bd, ad);
title('Butterworth IIR Filter using BLT')
```

expt 6

FIR Design using Rectangular, Bartlett, Hann, Hamming, and Blackman Windows

```
clear, clc, close all;

% --- Filter Specifications ---
Fs = 1;        % Normalized Sampling Frequency (Nyquist = 1)
N = 63;        % Filter Order
Wc = 0.3;      % Normalized Cutoff Frequency (0 to 1)
filtType = 'low'; % 'low' | 'high' | 'bandpass' | 'stop'

% --- Define Windows ---
windowNames = {'Rectangular', 'Bartlett', 'Hann', 'Hamming', 'Blackman'};
windowFuncs = {@rectwin, @bartlett, @hann, @hamming, @blackman};
numWins = numel(windowNames);

Wins = cell(numWins,1);
Bs = cell(numWins,1);
Hs = cell(numWins,1);

% --- Design Filters for each window ---
for k = 1:numWins
   W = windowFuncs{k}(N+1); % Window of length N+1
   Wins{k} = W;

   % FIR filter design
   B = fir1(N, Wc, filtType, W, 'noscale');
   Bs{k} = B;

   % Frequency response
   [H,f] = freqz(B, 1, 1024, Fs);
   Hs{k} = struct('H', H, 'f', f);
end

% Plot Frequency Response
for k = 1:numWins
   figure;
   plot(Hs{k}.f, 20*log10(abs(Hs{k}.H)+1e-6), 'LineWidth', 1.5);
   title([windowNames{k} ' Window Frequency Response']);
   xlabel('Normalized Frequency');
   ylabel('Magnitude (dB)');
   grid on;
   ylim([-100 5]);
end
```

expt 7

```
tol = 1e-6;

%---Define Transfer Function---
systems = {};
systems{1} = struct('b', [1, -0.5], 'a', [1], 'name', 'System 1');
systems{2} = struct('b', [1, -2], 'a', [1], 'name', 'System 2');
systems{3} = struct('b', conv([1, -0.5], [1, -2]), 'a', [1], 'name', 'System 3');

%---Loop Through Each System---
for k = 1:length(systems)
  b = systems{k}.b;
  a = systems{k}.a;
  name = systems{k}.name;

  %---Find Zeros of the system---
  z = roots(b);

  %---Classification based on zero locations---
  inside = sum(abs(z) < 1-tol);
  outside = sum(abs(z) > 1+tol);
  on_unit = sum(abs(abs(z)-1) <= tol);

  % Decide Type of System
  if outside == 0 && on_unit == 0
    type = 'Minimum - Phase';
  elseif inside == 0 && on_unit == 0
    type = 'Maximum - Phase';
  else
    type = 'Special Case (zeros on unit circle)';
  end

  %---Print classification result in command window---
  fprintf('%s = %s\n', name, type);

  %---Plot pole - zero diagram for this system---
  subplot(1,3,k); zplane(b,a); title(name);
end
```

expt 8

```
% N = 4; Wp = 0.4;
[b, a] = butter(4, 0.4);
[z, p, k] = butter(4, 0.4);

% Equivalent discrete-time transfer function
[b, a] = butter(N, Wp);
sys = tf(b, a, -1); % -1 implies discrete-time with Ts = 1
display(sys);

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

% Display transfer function in Direct Form and Cascaded (Second-Order) Form
disp('Direct Form:'); disp(tf(b, a, -1));
[sos, g] = tf2sos(b, a);
disp('Second-Order Sections (Cascaded):'); disp(zpk(sos(:, 1:2), sos(:, 4:5), sos(:, 6), -1));
disp('Gain:'); disp(g);

% Display transfer function in Parallel Form Sections
[r, p, k] = residuez(b, a);
disp('Parallel Form Sections:');
for i = 1:length(r)
   disp(['Section ' num2str(i) ': ' num2str(r(i)) '/(1 - ' num2str(p(i)) 'z^{-1})']);
end

% Interactive Filter Overview
figure;
h = fvtool(b, a);
set(h, 'Fs', 1, 'NormalizedFrequency', 'on', 'Color', [1 1 1]);
title('Butterworth IIR Filter Overview', 'NumberTitle', 'off');

% Plot Frequency Response, Pole-Zero Plot, Impulse Response, Step Response
figure(2);
subplot(2, 2, 1);
freqz(b, a, 512);
title('Frequency Response');
grid on;
subplot(2, 2, 2);
zplane(b, a);
title('Pole-Zero Plot');
subplot(2, 2, 3);
impz(b, a);
title('Impulse Response');
subplot(2, 2, 4);
stepz(b, a);
title('Step Response');

% Display Direct Form Sections (Coefficient values are approximate for display)
fprintf('\nDirect Form:\n');
fprintf('Numerator: '); disp(b);
```

```
fprintf('Denominator: '); disp(a);

% Display Parallel Form Sections (Coefficient values from residuez)
fprintf('\nParallel Form Sections (residuez):\n');
fprintf('Residues (r): '); disp(r');
fprintf('Poles (p): '); disp(p');
fprintf('Direct Term (k): '); disp(k);

% Output values from MATLAB window for verification (example of values shown in image)
% Direct Form:
% Numerator: 0.0266 0.0766 0.0766 0.0266
% Denominator: 1.0000 -0.7021 0.4800 -0.1827 0.0301

% Second-Order Sections (Cascaded):
% Section 1: 0.6409 (1 - 0.2266z^{-1} + 0.4663z^{-2})
% Section 2: 0.6409 (1 - 0.6531z^{-1} + 0.4663z^{-2})
% Gain: 0.0646

% Parallel Form Sections:
% Section 1: 0.1866 / (1 - 0.7996z^{-1})
% Section 2: 0.0863 / (1 - 0.1863z^{-1})
% Section 3: 0.3801 / (1 - 0.3861z^{-1})
% Section 4: 0.1861 / (1 - 0.1863z^{-1})
% Direct Term: 1.9666
```