DSP Codes

1. Experiment 1: DFT

```
% Lab_1A_code.m - Frequency contents of sinusoidal signals

clear; % clear Matlab Workspace
close all; % close all figures
clc; % clear Command Window

A1=5; % amplitude of sine wave
fsig1=1000; % frequency, Hz, of the sine wave
T1=1/fsig1; % period, sec, of the sine wave
FS=8000; % sampling frequency, Hz
N=8; % number of sampling points for the sine wave and number of DFT points

%Plot CT signal
t=0:T1/100:T1; % time axis values
x=A1*sin(2*pi*fsig1*t); % sine wave (continuous time) values
figure('Name', 'FFT Analysis 1A'); % plot figure name
subplot(3,1,1);
plot(t,x); % plot of sine wave (continuous time) vs time
xlabel('time ms');
ylabel('x(t)');



%Plot DT signal

n=0:N-1; % sample numbers
xn=A1*sin(2*pi*fsig1/FS*n); % sine wave (discrete time) values
subplot(3,1,2);
stem(n,xn); % stem plot of sine wave (discrete time) vs sample number
xlabel('sample n');
ylabel('x(n)');

%Plot DFT signal


XFT=fft(xn,N); % N point DFT of xn using FFT function
disp(XFT); % display DFT values
XFTmag=abs(XFT); % magnitudes of DFT values
subplot(3,1,3);
stem(n,XFTmag); % stem plot of DFT values vs sample number
xlabel('discrete frequency k');
ylabel('XFTmag(k)');
```

2. Experiment 2: Convolution

2A. 4- Point linear and circular convolution

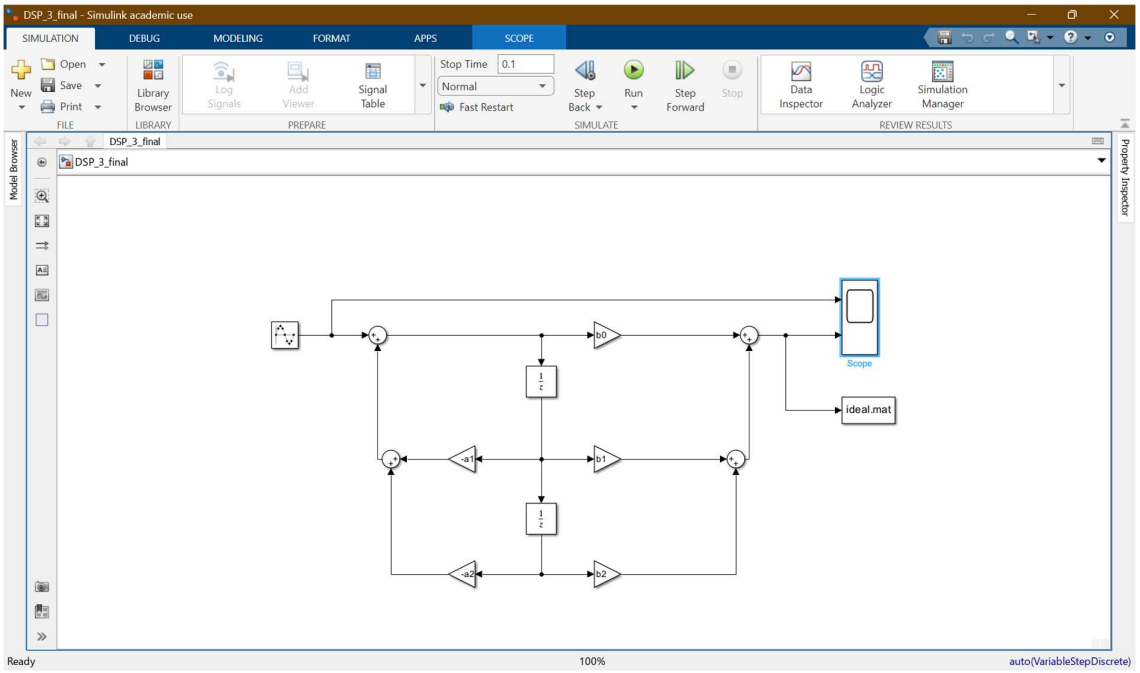
```
% 4-Point Linear Convolution


m = [ 1 2 0 0 ];
n = [ 1 2 4 0 ];


y = conv(m,n);


disp('Result of the convolution:');
disp(y);


% 4-Point Circular Convolution


x = [ 1 2 0 0 ];
h = [ 1 2 4 0 ];


N = 4; %Circular Convolution Length


y = zeros(1, N);


for k = 1:N
    for n = 1:N
        y(k) = y(k) + x(n) * h(mod(k-n, N) + 1);
    end
end


disp('Circular Convolution Result: ');
disp(y);
```

2B. 8-point linear and circular convolution

```
% 8-Point Linear Convolution


x = [ 1 2 0 0 ];
h = [ 1 2 4 0 ];
```

```
x_padded = [x, zeros(1, 8 - length(x))];
h_padded = [h, zeros(1, 8 - length(h))];


y = conv(x_padded, h_padded);


disp('Result of the convolution:');
disp(y);


% 8-Point Circular Convolution


x = [ 1 2 0 0 ];
h = [ 1 2 4 0 ];


x_padded = [x, zeros(1, 8 - length(x))];
h_padded = [h, zeros(1, 8 - length(h))]


N = 8; %Circular Convolution Length


y = zeros(1, N);


for k = 1:N
    for n = 1:N
        y(n) = y(k) + x_padded(n) * h_padded(mod(k-n, N) + 1);
    end
end


disp('Circular Convolution Result: ');
disp(y);
```

3. Simulink



Model post-load function:

```
a1 = -0.9;
a2 = 0.81;
b0 = 1/3;
b1 = 1/3;
b2 = 1/3;
w0 = pi/6;
yi1 = -3;
yi2 = 2;

%wi1 = yi1/b0-b1*yi2/b0^2;
%wi2 = yi2/b0;
wi1 = 0;
wi2 = 0;

Vs = 1;
%Vs = 0;
```

4. FIR Filter

```
 %% FIR Low Pass Filter Design for Speech Signal
clc; clear; close all;


% Read audio file
[x, Fs] = audioread('flute.wav');
x = x(:,1);  % Use one channel
disp(['Sampling Frequency: ', num2str(Fs), ' Hz']);


% Input LPF specifications
% Format: [fp, ApdB, fs, AsdB]
specs = input('Enter LPF specs [fp, ApdB, fs, AsdB]: ');
fp = specs(1);  ApdB = specs(2);  fs = specs(3);  AsdB = specs(4);


% Derived parameters
fc = (fp + fs) / 2;           % Cutoff frequency
ftran = (fs - fp) / 2;        % Transition bandwidth (normalized)
disp(['Normalized transition frequency = ', num2str(ftran)]);


% Select window type based on specs
if (ApdB >= 0.74) && (AsdB <= 21)
    win = 'rectwin';  wname = 'Rectangular';  N = ceil(0.9/ftran);
elseif (ApdB >= 0.0546) && (AsdB <= 44)
    win = 'hann';      wname = 'Hanning';     N = ceil(3.1/ftran);
elseif (ApdB >= 0.0194) && (AsdB <= 53)
    win = 'hamming';   wname = 'Hamming';     N = ceil(3.3/ftran);
elseif (ApdB >= 0.0017) && (AsdB <= 74)
    win = 'blackman';  wname = 'Blackman';    N = ceil(5.5/ftran);
else
    error('Invalid filter specifications.');
end


% Ensure odd filter length
if mod(N,2) == 0, N = N + 1; end
disp([wname, ' window selected, N = ', num2str(N)]);


% Design FIR LPF
b = fir1(N-1, fc, 'low', feval(win, N));
figure; freqz(b, 1, 1024);
title(['Frequency Response using ', wname, ' window']);


% Filter the audio
xlp = filter(b, 1, x);
```

```
% Time axis
t = (0:length(x)-1) / Fs;


% Plot original vs filtered signals
figure('Name', 'Original vs Filtered');
subplot(2,1,1);
plot(t, x);
title('Original Signal');
xlabel('Time (s)');
ylabel('Amplitude');


subplot(2,1,2);
plot(t, xlp);
title(['Filtered Signal (', wname, ' window)']);
xlabel('Time (s)');
ylabel('Amplitude');


% Play original sound
disp('Playing Original Audio...');
sound(x, Fs);
pause(length(x)/Fs + 1);  % Wait until playback finishes


% Play filtered sound
disp('Playing Filtered Audio...');
sound(xlp, Fs);
```

5. IIR Filter

```
 % Experiment 05: IIR LPF Design for Speech Signal using Bilinear
Transformation

clc; clear; close all;


% --- User Inputs ---
fp = input('Enter pass band cutoff freq (Hz): ');
fs = input('Enter stop band cutoff freq (Hz): ');
Ap = input('Enter pass band attenuation (dB): ');
As = input('Enter stop band attenuation (dB): ');
Fs = input('Enter sampling frequency (Hz): ');


% --- Prewarping ---
wp = tan(pi * fp / Fs);
ws = tan(pi * fs / Fs);


% --- Filter Order (Force N = 2 for this code) ---
N = ceil((log10((10^(As/10)-1)/(10^(Ap/10)-1))) / (2*log10(ws/wp)));
fprintf('Calculated filter order N = %d\n', N);
N = 2;  % Force to 2 since code supports only 2nd order


% --- Cutoff Frequency ---
wc = wp / ((10^(Ap/10)-1)^(1/(2*N)));
fprintf('Cutoff frequency wc = %.4f rad/s\n', wc);


% --- Digital Filter Coefficients (via Bilinear Transform) ---
b = wc * [1 2 1];
a = [1 + 2*wc + wc^2, 2*(wc^2 - 1), 1 - 2*wc + wc^2];


% --- Frequency Response ---
f = linspace(0, Fs/2, 1000);
w = 2 * pi * f / Fs;
H = polyval(b, exp(-1j*w)) ./ polyval(a, exp(-1j*w));


Hn = H / abs(H(1));


% --- Plotting ---
figure;
subplot(2,1,1);
semilogx(f, 20*log10(abs(Hn)), 'r', 'LineWidth', 1.5);
xlabel('Frequency (Hz)'); ylabel('Magnitude (dB)');
title('Magnitude Response of IIR Filter'); grid on;
```

```
subplot(2,1,2);
semilogx(f, angle(H)*180/pi, 'r', 'LineWidth', 1.5);
xlabel('Frequency (Hz)'); ylabel('Phase (degrees)');
title('Phase Response of IIR Filter'); grid on;
```

6. Interpolation and Decimation

6A. Interpolation

```
close all;
clear;
clc;


[x, fs] = audioread('C:/Users/Aditya/Documents/MATLAB/flute.wav');
x = x(:,1);
disp(['fs = ', num2str(fs)]);
xlen = length(x);
disp(['xlen = ', num2str(xlen)]);
n = (0:length(x)-1);
timeduration = xlen/fs;


data = input('Enter "N", "M", "L" as [N, M, L] ');
N = data(1); % upsampling factor
M = data(2); % filter order (even integer)
L = data(3); % length of upsampled sequence to plot


% Ensure filter order is even
if mod(M, 2) ~= 0
   M = M + 1;
end


% Upsample the signal by factor N (insert zeros)
xup = upsample(x, N);


% Design low-pass FIR filter with cutoff frequency 1/N (normalized)
h = fir1(M, 1/N);


% Filter the upsampled signal to remove spectral images
xinterp = filter(h, 1, xup);
```

```
figure(1);
subplot(3,1,1);
stem(x);
title('Original Signal (Complete)');
xlabel('samples n');


subplot(3,1,2);
stem(xup);
title('Upsampled Signal (with zeros, Complete)');
ylabel('Amplitude');
xlabel('samples n');


subplot(3,1,3);
stem(xinterp);
title('Interpolated Signal (Filtered, Complete)');
ylabel('Amplitude');
xlabel('samples n');


figure(2);
% To avoid indexing errors, clip indices with min()
maxIndexOriginal = min(L, xlen);
maxIndexUp = min(L*N, length(xinterp));


subplot(3,1,1);
stem(x(1:maxIndexOriginal));
title('Original Signal (Selected Portion)');
ylabel('Amplitude');
xlabel('samples n');
axis([1 maxIndexOriginal -0.2 0.2]);


subplot(3,1,2);
stem(xup(1:maxIndexUp));
title('Upsampled Signal (Selected Portion)');
ylabel('Amplitude');
xlabel('samples n');
axis([1 maxIndexUp -0.2 0.2]);


subplot(3,1,3);
stem(xinterp(1:maxIndexUp));
title('Interpolated Signal (Filtered, Selected Portion)');
ylabel('Amplitude');
xlabel('samples n');
axis([1 maxIndexUp -0.2 0.2]);
```

6B. Decimation

```
close all;
clear;
clc;


[x, fs] = audioread('C:/Users/Aditya/Documents/MATLAB/flute.wav');
x = x(:,1);
disp(['fs = ', num2str(fs)]);
xlen = length(x);
disp(['xlen = ', num2str(xlen)]);
n = (0:length(x)-1);
timeduration = xlen/fs;


data = input('Enter "N", "M", "L" as [N, M, L] ');
N = data(1); % decimation factor
M = data(2); % filter order (even integer)
L = data(3); % length of decimated sequence to plot


% Ensure filter order is even
if mod(M, 2) ~= 0
   M = M + 1;
end


% Design low-pass FIR filter with cutoff frequency 1/N (normalized)
h = fir1(M, 1/N);


% Filter the signal to prevent aliasing
xfilt = filter(h, 1, x);


% Downsample the filtered signal by factor N
xdec = downsample(xfilt, N);


figure(1);
subplot(3,1,1);
stem(x);
title('Original Signal (Complete)');
xlabel('samples n');


subplot(3,1,2);
stem(xfilt);
title('Low-pass Filtered Signal (Complete)');
ylabel('Amplitude');
```

```
xlabel('samples n');


subplot(3,1,3);
stem(xdec);
title('Decimated Signal (Complete)');
ylabel('Amplitude');
xlabel('samples n');


figure(2);
% To avoid indexing errors, clip indices with min()
maxIndexOriginal = min(L * N, xlen);
maxIndexDec = min(L, length(xdec));


subplot(3,1,1);
stem(x(1:maxIndexOriginal));
title('Original Signal (Selected Portion)');
ylabel('Amplitude');
xlabel('samples n');
axis([1 maxIndexOriginal -0.2 0.2]);


subplot(3,1,2);
stem(xfilt(1:maxIndexOriginal));
title('Low-pass Filtered Signal (Selected Portion)');
ylabel('Amplitude');
xlabel('samples n');
axis([1 maxIndexOriginal -0.2 0.2]);


subplot(3,1,3);
stem(xdec(1:maxIndexDec));
title('Decimated Signal (Selected Portion)');
ylabel('Amplitude');
xlabel('samples n');
axis([1 maxIndexDec -0.2 0.2]);
```

7. Circular Convolution on DSP Processor

```
#include <stdio.h>

int main(void)

{

   int x[15] = {0};

   int h[15] = {0};

   int a[15] = {0};

   int p[15] = {0};

   int y[15] = {0};

   int N,L,k,i,j = 0;


   printf("Enter the length of h(n): ");

   scanf("%d",&N);

   printf("Enter the length of x(n): ");

   scanf("%d",&L);

   printf("Enter the values of h(n)");

   for(i=0;i<N;i++)

   {

          scanf("%d",&h[i]);

   }

   printf("Enter the values of x(n)");

   for(j=0;j<L;j++)

   {

          scanf("%d",&x[j]);

   }


   // padding of zeros to make lengths of h(n) and x(n) equal to that of the
larger sequence

    if(N-L!=0)                        // if length of both sequences are not
equal

    {

                if(L>N)               // pad the smaller sequence with zeros

                {
```

```
            for(i=N;i<L;i++)
                        h[i]=0;
                        N=L;
            }
            for(i=L;i<N;i++)
                        x[i]=0;
                        L=N;
}

// Circular convolution of h(n) and x(n)
y[0]=0;
a[0]=h[0];
for(j=1;j<N;j++)        // folding h(n) to h(-n)
            a[j]=h[N-j];
for(i=0;i<N;i++)        // obtain first output value y(0)
            y[0]+=x[i]*a[i];
for(k=1;k<N;k++)        // obtain remaining output values
{
            y[k]=0;     // circular shift
            for(j=1;j<N;j++)
                        p[j]=a[j-1];
                        p[0]=a[N-1];
            for(i=0;i<N;i++)
            {
                        a[i]=p[i];
                        y[k]+=x[i]*p[i];
            }
}

// displaying the result
printf("\nCircular convolution output is:");
for(i=0;i<=N;i++)
```

```
    {
          printf("\n Output y[%d] = %d",i,y[i]);
    }
    while(1);
}
```

8. Linear Convolution on DSP Processor

```
#include <stdio.h>

int main()
{
    int x[15] = {0};
    int h[15] = {0};
    int y[15] = {0};
    int N,L,K,i,j = 0;
    printf("Enter the length of h(n): ");
    scanf("%d",&N);
    printf("Enter the length of x(n): ");
    scanf("%d",&L);
    K=N+L-1;
    printf("Enter the values of h(n)");
    for(i=0;i<N;i++)
    {
        scanf("%d",&h[i]);
    }
    printf("Enter the values of x(n)");
    for(j=0;j<L;j++)
    {
        scanf("%d",&x[j]);
    }

    // padding of zeros to make lengths of h(n) and x(n) equal to K = N+M-1

    for(i=N;i<=K;i++)
    {
        h[i]=0;
    }
    for(i=L;i<=K;i++)
    {
        x[i]=0;
    }

    // linear convolution of h(n) and x(n)

    for(i=0;i<K;i++)
    {
```

```
        y[i]=0;

        for(j=0;j<=i;j++)
        {
            y[i]=y[i]+((h[j])*(x[i-j]));
        }
    }

    printf("\nLinear convolution output is:");

    for(i=0;i<=K;i++)

    {
        printf("\n y(%d) = %d",i,y[i]);
    }

    while(1);
}
```