DSP

PART - A

Circular Convolution

```
clc;
clear all;
close all;

x = [1, 2, 3, 4];
h = [5, 6, 7, 8, 9, 1];

N = max(length(x),length(h));
y = zeros(1,N);

xz = [x, zeros(1,N-length(x))];
hz = [h, zeros(1,N-length(h))];

for n=1:N
   for m=1:N
      y(n) = y(n) + xz(m)*hz(mod(n-m,N)+1);
   end
end

X = fft(xz);
H = fft(hz);
Y = X.*H;
Y_ = ifft(Y);
disp(Y_);
disp(y);
```

Compute DFT/IDFT

```
clc;
clear all;
close all;

% sequence
k = 0:3;
n = 0:3;
N = length(k);
y = ones(1, N);
------------------------------
% signal
f = 100;
fs = 8000;
t = 0:1:100;
y = 5*sin(2*pi*(f/fs)*t) + rand(1, length(t)); % singal with noise
N = length(y);
k = 0:N-1;
n = 0:N-1;
------------------------------

matrix = k.*n';
Wn = exp(-1i*2*pi/N);
Wn_matrix = Wn.^matrix;

Y = Wn_matrix*y';
subplot(2,2,1);
stem(abs(Y));

subplot(2,2,2);
stem(abs(fft(y)));

Wn_con = conj(Wn_matrix);
y1 = (1/N)*(Wn_con*Y);
subplot(2,2,3);
stem(real(y1));
```

```
subplot(2,2,4);
stem(real(ifft(Y)))
```

PSD

```
clc;
clear all;
close all;

N = 200;
n = 0:N-1;
f1 = 1000;
f2 = 2000;
fs = 8000;

x = sin(2*pi*(f1/fs)*n) + 2*cos(2*pi*f2/fs*n)  + rand(1,N);

X = fft(x);
X_norm = X/N;
p = X_norm.^2;

f = (0:N-1)*(fs/N); % to get the actual frequencies on the plot

subplot(2,1,1);
plot(f, X_norm);

subplot(2,1,2);
plot(f, abs(p))
```

Sinusoidal signal using filtering

```
clc;
clear all;
close all;

N = 200;
f_req = 500;
```

```
fs = 8000;

w0 = 2*pi*f_req/fs;
a1 = 2*cos(w0);
a2 = -1;
b1 = sin(w0);

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

y = zeros(1,N);
y(1) = b1;
y(2) = b1*x(2) + a1*y(1);

for n = 3:N
   y(n) = b1*x(n) + a1*y(n-1) + a2*y(n-2);
end

plot(y)
```

DTMF

Frequency	1209 Hz	1336 Hz	1477 Hz
697 Hz	1	2	3
770 Hz	4	5	6
852 Hz	7	8	9
941 Hz	*	0	#

```
clc;
clear all;
close all;

fs = 8000;
T = 0.05;
```

```
t = 0:1/fs:T-1/fs;

key = input('Enter a key: ','s');

switch key
   case '1'
      fl = 697; fh = 1209;
   case '2'
      fl = 697; fh = 1336;
   case '3'
      fl = 697; fh = 1477;
   case '4'
      fl = 770; fh = 1209;
   case '5'
      fl = 770; fh = 1336;
   case '6'
      fl = 770; fh = 1477;
   case '7'
      fl = 852; fh = 1209;
   case '8'
      fl = 852; fh = 1336;
   case '9'
      fl = 852; fh = 1477;
   case '*'
      fl = 941; fh = 1209;
   case '0'
      fl = 941; fh = 1336;
   case '#'
      fl = 941; fh = 1477;
   otherwise
      error('Invalid key! Please enter 0–9, *, or #.');
end

y = sin(2*pi*fl*t) + sin(2*pi*fh*t);
sound(y);
plot(t, y);
```

FIR Filters

```
clc; clear; close all;

order = 50;              % Filter order (number of taps - 1)
cf = [500 1000 1500]/4000; % Normalized cutoff frequencies (fs = 8000 H
z)

% Design FIR lowpass filters with different windows
b_rect1 = fir1(order, cf(1),'low', boxcar(order+1));    % Rectangular
b_tri1  = fir1(order, cf(1),'low',bartlett(order+1));  % Triangular
b_kai1  = fir1(order, cf(1),'low', kaiser(order+1, 8)); % Kaiser (beta=8)

b_rect2 = fir1(order, cf(2), boxcar(order+1));
b_tri2  = fir1(order, cf(2), bartlett(order+1));
b_kai2  = fir1(order, cf(2), kaiser(order+1, 8));

b_rect3 = fir1(order, cf(3), boxcar(order+1));
b_tri3  = fir1(order, cf(3), bartlett(order+1));
b_kai3  = fir1(order, cf(3), kaiser(order+1, 8));

fvtool( ...
    b_rect1, 1, b_tri1, 1, b_kai1, 1, ...
    'Analysis', 'freq');
```

Decimation

```
clc;
clear all;
close all;

N = 1024;
f = 500;
fs = 8000;
n = 0:N-1;

x = sin(2*pi*f*n/fs);
```

```
M = 2;
xd = decimate(x,M);

subplot(2,1,1);
stem(n(1:50),x(1:50));
subplot(2,1,2);
stem(n(1:50), xd(1:50));
```

Interpolation

```
clc;
clear all;
close all;

N = 1024;
f = 500;
fs = 8000;
n = 0:N-1;

x = sin(2*pi*f*n/fs);

L = 2;
xd = interp(x,L);

subplot(2,1,1);
stem(n(1:50),x(1:50));
subplot(2,1,2);
stem(n(1:50), xd(1:50));
```

PART - B

Linear Convolution

```
#include<stdio.h>
#include<math.h>
```

```
int y[20];

void main(){
int m=6;
int n=6;

int i=0,j;
int x[15]={1,2,3,4,5,6,0,0,0,0,0,0}; /*Input Signal Samples*/
int h[15]={1,2,3,4,5,6,0,0,0,0,0,0}; /*Impulse Response Co-efficients*/

for(i=0;i<m+n-1;i++){
    y[i]=0;
    for(j=0;j<=i;j++)
        y[i]+=x[j]*h[i-j];
}

for(i=0;i<m+n-1;i++)
    printf("%d\n",y[i]);

}
```

Circular Convolution

```
#include <stdio.h>

int main() {
    int x[30], h[30], y[30];
    int m, n, N;
    int i, j, k;

    printf("Enter length of first sequence: ");
    scanf("%d", &m);
    printf("Enter length of second sequence: ");
    scanf("%d", &n);

    printf("Enter first sequence (x[n]): ");
    for (i = 0; i < m; i++)
```

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

    printf("Enter second sequence (h[n]): ");
    for (i = 0; i < n; i++)
        scanf("%d", &h[i]);

    // Make both sequences of equal length
    N = (m > n) ? m : n;
    for (i = m; i < N; i++) x[i] = 0;
    for (i = n; i < N; i++) h[i] = 0;

    // Circular convolution: y[k] = sum_{i=0}^{N-1} x[i]*h[(k-i) mod N]
    for (k = 0; k < N; k++) {
        y[k] = 0;
        for (i = 0; i < N; i++) {
            j = (k - i + N) % N;  // wrap-around index
            y[k] += x[i] * h[j];
        }
    }

    printf("\nCircular Convolution Result:\n");
    for (k = 0; k < N; k++)
        printf("%d ", y[k]);
    printf("\n");

    return 0;
}
```

Sine wave

```
#include <stdio.h>
#include <math.h>

#define PI    3.1415926535897793
#define FREQ  1000     // sine wave frequency in Hz
#define FS    24000    // sampling frequency in Hz
#define N     128      // number of samples
```

```
int main() {
    float a[N];
    int i;

    for (i = 0; i < N; i++) {
        a[i] = sin(2 * PI * FREQ * i / FS);
        printf("%f\n", a[i]);   // print each sample on a new line
    }

    return 0;
}
```

FFT

```
```

DFT

```
#include <stdio.h>
#include <math.h>

#define PI 3.1416

int main() {
    int N, n, k;
    float x[32], Xreal[32], Ximag[32];

    printf("Enter length of sequence N: ");
    scanf("%d", &N);

    printf("Enter the sequence:\n");
    for (n = 0; n < N; n++)
        scanf("%f", &x[n]);

    // DFT calculation
    for (k = 0; k < N; k++) {
        Xreal[k] = 0;
```

```
        Ximag[k] = 0;
        for (n = 0; n < N; n++) {
            Xreal[k] += x[n] * cos(2 * PI * k * n / N);
            Ximag[k] -= x[n] * sin(2 * PI * k * n / N);
        }
    }

    printf("\nDFT Output:\n");
    for (k = 0; k < N; k++)
        printf("X[%d] = %.3f + j%.3f\n", k, Xreal[k], Ximag[k]);

    return 0;
}
```