**1. Butterworth filter design** [b,a]=butter(2,1,'s'); systf=tf(b,a);

pzmap(systf);

This is first step in designing Butterworth filter. b=0.0843\*[1 2 1];

a=[1 -1.028 0.3651];

w=0:0.01\*pi:pi; h=freqz(b,a,w); gain=20\*log10(abs(h)); figure;

plot(w/pi,gain); grid;

Helps in designing Butterworth filter using billinear transformation method and gives the Bode-plot which helps in reading stability analysis of the filter.

#### Chebyshev filter designing

b=0.052\*[1 2 1];

a=[1 -1.349 0.6084];

w=0:0.01\*pi:pi; h=freqz(b,a,w); gain=20\*log10(abs(h));

figure; plot(w/pi,gain); grid;

We are designing Chebyshev Type – 1 filter which is useful in eliminating ripples in passband frequency range and giving a sharp cutoff in stop band frequency range.

Another important algorithm we have used is **Least Mean Squares (LMS)** algorithm. This algorithm helps changing the coefficients of filters more dynamically to reduce the background noise.

#### 3. Least Mean Squares (LMS) Algorithm:

clear all % For sinusoidal input %n=1:10000; %x=sin(0.4*pi*n); %x=x'; x = randn(10000,1);%random input signal

%[B,A] = ellip(4,0.25,10,0.25);%IIR filter

%x=[1 zeros(1,9999)]'; % Input is a unit sample sequence [B,A]=butter(10,0.3);

d = filter(B,A,x);%reference signal figure;freqz(B,A)%view frequency response N=32;% length of the filter M=length(x);%length of the input signal

y=zeros(1,M);%intializing the response of the adaptive filter h=zeros(1,N);%intializing the filter coefficents to zero e(1:N-1)=x(1:N-1);%intializing the error array

% calculating the value of step size x2=0;

for t=1:M x2=x2+x(t).^2;

end

de=(M+1)/(10\*N\*x2)%de is the value of step size

% the lms algorithm calculates the value of output array and error function for n =N:M

x1= x(n:-1:n-N+1);

y(n)=h\*x1; e(n)=d(n)-y(n); h=h+de\*e(n)\*x1';

end

figure; freqz(h,1,512);

figure; plot(d);

title('Desired response');

figure; plot(y);

title('Adaptive filter response');

% Testing for error in o/p for Unit sample sequence

%err\_impuls=d-y';

%the mean square error is calculated in the following module K=10\*N

L=fix(M/K); for m=1:L-1 se(m)=0;

ase(m)=0;

for t=(m-1)\*K+1:m\*K se(m)=se(m)+e(t).^2;

end ase(m)=se(m)/K;

end

figure; m1=1:L-1;

stem(m1,ase);

% Plotting the outputs before convergence has occurred. It is seen from % plot of 'ase' that error is negligible after m=5. Hence the 2nd frame % is only plotted figure; plot(d(0*K+1:0*K+K))figure;

plot(y(0\*K+1:0\*K+K));

% Plotting the outputs after convergence has occurred. It is seen from

% plot of 'ase' that error is negligible after m=5. Hence the 5th frame

% is only plotted

figure;

plot(d(5\*K+1:5\*K+K));

figure; plot(y(5\*K+1:5\*K+K));

It would dynamically adjust the filter coefficients according to the need.

So now we have the overall code for implementation of **Adaptive Noise Cancellation.**

### 4.MATLAB Code for Adaptive Noise Cancellation

#### 4.1. Testing Butterworth filtering on given manual sinusoids

fs=10000;

t=0:1/fs:1; x1=2\*sin(2\*pi\*100\*t); sound(x1,fs); x2=sin(2\*pi\*1000\*t); sound(x2,fs)

x=x1+x2; sound(x,fs);

FT= 10000;

FP= 300;

FS= 700;

RP= 3;

RS= 40;

wp= FP\*2/FT; ws= FS\*2/FT;

[N,wn]= buttord(wp,ws,RP,RS);

disp('order of the butterworth filter '); disp(N); disp('Cutoff frequency '); disp(wn);

[b,a]=butter( N,wn);

y=filter(b,a,x);

%pause sound(y,fs);

w=0:0.01/pi:pi; h1=freqz(x,1,w); FT\_x=20\*log10(abs(h1)); figure;

%plot(w/pi,FT\_x); grid; plot((w/pi)\*fs/2,FT\_x); grid; xlabel('frequency'); ylabel('FT\_x,dB');

h2=freqz(y,1,w); FT\_y=20\*log10(abs(h2)); figure;

%plot(w/pi,FT\_y); grid;

%xlabel('Normalized frequency');

plot((w/pi)\*fs/2,FT\_y); grid; xlabel('frequency') ylabel('FT\_y,dB');

#### 4.2 Testing Butterworth filtering on sound files

[x,fs]=wavread('horn'); sound(x,fs);

w=0:0.01/pi:pi; h1=freqz(x,1,w); FT\_x=20\*log10(abs(h1)); figure;

%plot(w/pi,FT\_x); grid;

%xlabel('Normalized frequency'); plot((w/pi)\*fs/2,FT\_x); grid; xlabel('frequency'); ylabel('FT\_x,dB');

FT= fs; FP= 500;

FS= 800;

RP= 3;

RS= 40;

wp= FP\*2/FT;

ws= FS\*2/FT;

[N,wn]= buttord(wp,ws,RP,RS);

disp('order of the butterworth filter '); disp(N); disp('Cutoff frequency '); disp(wn); [b,a]=butter( N,wn);

y=filter(b,a,x); sound(5\*y,fs); h2=freqz(y,1,w); FT\_y=20\*log10(abs(h2)); figure;

% plot(w/pi,FT\_y); grid;

% xlabel('Normalized frequency'); plot((w/pi)\*fs/2,FT\_y); grid; xlabel('frequency'); ylabel('FT\_y,dB');

randn('state',0); x=randn(5000,1); sound(x,5000);

[b,a]=butter(2,1,'s'); systf=tf(b,a); pzmap(systf);

b=0.0843\*[1 2 1];

a=[1 -1.028 0.3651];

w=0:0.01\*pi:pi; h=freqz(b,a,w); gain=20\*log10(abs(h)); figure;

plot(w/pi,gain); grid;

b=0.052\*[1 2 1];

a=[1 -1.349 0.6084];

w=0:0.01\*pi:pi; h=freqz(b,a,w); gain=20\*log10(abs(h)); figure;

plot(w/pi,gain); grid; b=0.3016;

a=[1 -1.044 0.3588];

w=0:0.01\*pi:pi; h=freqz(b,a,w); gain=20\*log10(abs(h));

figure; plot(w/pi,gain); grid;

clear all

% For sinusoidal input

%n=1:10000;

%x=sin(0.4\*pi\*n);

%x=x';

x = randn(10000,1);%random input signal

%[B,A] = ellip(4,0.25,10,0.25);%IIR filter

%x=[1 zeros(1,9999)]'; % Input is a unit sample sequence [B,A]=butter(10,0.3);

d = filter(B,A,x);%reference signal figure;freqz(B,A)%view frequency response N=32;% length of the filter M=length(x);%length of the input signal

y=zeros(1,M);%intializing the response of the adaptive filter h=zeros(1,N);%intializing the filter coefficents to zero e(1:N-1)=x(1:N-1);%intializing the error array

% calculating the value of step size x2=0;

for t=1:M x2=x2+x(t).^2;

end

de=(M+1)/(10\*N\*x2)%de is the value of step size

% the lms algorithm calculates the value of output array and error function for n =N:M

x1= x(n:-1:n-N+1);

y(n)=h\*x1; e(n)=d(n)-y(n); h=h+de\*e(n)\*x1';

end figure;

freqz(h,1,512); figure;

plot(d);

title('Desired response'); figure;

plot(y);

title('Adaptive filter response');

% Testing for error in o/p for Unit sample sequence

%err\_impuls=d-y';

%the mean square error is calculated in the following module K=10\*N

L=fix(M/K); for m=1:L-1 se(m)=0;

ase(m)=0;

for t=(m-1)\*K+1:m\*K se(m)=se(m)+e(t).^2;

end ase(m)=se(m)/K;

end

figure; m1=1:L-1;

stem(m1,ase);

% Plotting the outputs before convergence has occurred. It is seen from

% plot of 'ase' that error is negligible after m=5. Hence the 2nd frame

% is only plotted figure;

plot(d(0\*K+1:0\*K+K)); figure; plot(y(0\*K+1:0\*K+K));

% Plotting the outputs after convergence has occurred. It is seen from

% plot of 'ase' that error is negligible after m=5. Hence the 5th frame

% is only plotted figure;

plot(d(5\*K+1:5\*K+K)); figure; plot(y(5\*K+1:5\*K+K));

This MATLAB code has been verified.