DSP project codes:

* **ANC using simple LMS algo-**
* clear;
* M=16; %order of filter
* mu=0.04; %step-size
* N=200; %Iterations
* f=750;
* Ts=1/(4\*f); %fs=4 times the freq of the signal
* noise=(rand(N,1)-0.5);
* n=zeros(M,1);
* w=zeros(M,1);
* Wn=[0.1 0.5]; %see w/o filter
* [B,A]=butter(2,Wn);
* x=filter(B,A,n);
* for i=1:N
* t=(i-1)\*Ts;
* for k=M:-1:2
* n(k)=n(k-1);
* end
* s(i)=cos(2\*pi\*f\*t);
* n(1)=0.2\*(cos(2\*pi\*50\*t)+sin(2\*pi\*100\*t)+cos(2\*pi\*60\*t)+sin(2\*pi\*80\*t)+cos(2\*pi\*30\*t)+ sin(2\*pi\*20\*t)+sin(2\*pi\*10\*t)+ sin(2\*pi\*90\*t)); %noise(i);
* d(i)=s(i)+n(1);
* x=filter(B,A,n);
* d\_out(i)=w'\*x;
* e(i)=d(i)-d\_out(i);
* w=w+mu\*e(i)\*x;
* end
* i=1:N;
* subplot(3,1,1);
* plot(i,d,'g');
* title('Adaptive Noise Canceller');
* xlabel('Iterations');
* ylabel('ANC i/p');
* axis([1 N -2 2]);
* subplot(3,1,2);
* plot(i,s,'b');
* xlabel('Iterations');
* ylabel('desired');
* axis([1 N -2 2]);
* subplot(3,1,3);
* plot(i,e,'r');
* xlabel('Iterations');
* ylabel('ANC o/p');
* axis([1 N -2 2]);
* **ANC with uncorrelated noise in primary and reference input**
* clc
* clear all;
* close all;
* sampling\_time=1/(8\*200);
* mu=0.04;
* M=16;
* Iterations=500;
* u=zeros(M,1);
* x=zeros(M,1);
* w=zeros(M,1);
* e=zeros(Iterations,1);
* Wn=[0.1 0.5];
* [B,A]=butter(4,Wn);
* pri\_n=0.5\*(rand(Iterations,1)-0.5);
* ref\_n=0.5\*(rand(Iterations,1)-0.5);
* for n=0:Iterations-1
* t=n\*sampling\_time;
* for i=M:-1:2
* u(i)=u(i-1);
* end
* u(1)=0.5\*(cos(2\*pi\*50\*t)+sin(2\*pi\*100\*t)+cos(2\*pi\*60\*t)+sin(2\*pi\*80\*t)+cos(2\*pi\*30\*t)+sin(2\*pi\*20\*t)); %rand(1);
* d(n+1)=cos(2\*pi\*200\*n\*sampling\_time)+u(1)+pri\_n(n+1);
* x=filter(B,A,u)+ref\_n(n+1);
* d\_out=conj(w')\*x;
* e(n+1)=d(n+1)-d\_out;
* w=w+mu\*x\*conj(e(n+1));
* end
* n=1:Iterations;
* subplot(3,1,1);
* plot(n,d,'g');
* title('ANC performance-uncorrelated noise in pri and ref i/p');
* axis([0,500,-3,3]);
* xlabel('Iterations');
* ylabel('pri. i/p');
* subplot(3,1,3);
* plot(n,e','b');
* axis([0,500,-3,3]);
* xlabel('Iterations');
* ylabel('ANC output');
* subplot(3,1,2);
* plot(n,cos(2\*pi\*200\*n\*sampling\_time));
* ylabel('desired response');
* xlabel('Iterations');
* axis([0,500,-3,3]);
* **ANC with signal components in reference input**
* clc;
* clear all;
* close all;
* sampling\_time=1/(8\*200);
* mu=0.05;
* M=16;
* Iterations=1000;
* u=zeros(M,1);
* x=zeros(M,1);
* w=zeros(M,1);
* s=zeros(M,1);
* e=zeros(Iterations,1);
* Wn=[0.1 0.5];
* [Bn,An]=butter(2,Wn);
* Ws=0.5;
* [Bs,As]=butter(2,Wn);
* for n=0:Iterations-1
* t=n\*sampling\_time;
* for i=M:-1:2
* u(i)=u(i-1);
* s(i)=s(i-1);
* end
* u(1)= 0.2\*(cos(2\*pi\*50\*t)+sin(2\*pi\*100\*t)+cos(2\*pi\*60\*t)+sin(2\*pi\*80\*t)+cos(2\*pi\*30\*t)+sin(2\*pi\*20\*t)); %rand(1)-0.5;
* s(1)=cos(2\*pi\*200\*n\*sampling\_time);
* sig(n+1)=s(1);
* noi(n+1)=u(1);
* d(n+1)=s(1)+u(1);
* x=filter(Bn,An,u)+0.04\*filter(Bs,As,s);
* d\_out=conj(w')\*x;
* e(n+1)=d(n+1)-d\_out;
* w=w+mu\*x\*conj(e(n+1));
* end
* n=0:Iterations-1;
* subplot(3,1,1);
* plot(n,d,'r');
* title('Noise Canceller-Signal Comps. in ref. input');
* axis([0 Iterations-1 -2 2]);
* xlabel('Iterations');
* ylabel('ANC i/p');
* subplot(3,1,2);
* plot(n,cos(2\*pi\*200\*n\*sampling\_time),'g');
* axis([0 Iterations-1 -2 2]);
* xlabel('Iterations');
* ylabel('desired');
* subplot(3,1,3);
* plot(n,e','b');
* axis([0 Iterations-1 -2 2]);
* xlabel('Iterations');
* ylabel('ANC o/p');
* ZOOM XON;
* **ANC as notch**
* clear;
* mu=0.3;
* M=2;
* Iterations=512;
* C=0.2;
* w0=2\*pi\*30;
* phi=pi/4;
* phi1=phi;
* phi2=phi+pi/2;
* w=zeros(M,1);
* e=zeros(Iterations,1);
* fs=(20:1:40)';
* ws=2\*pi\*fs;
* A=rand(size(ws));
* theta=2\*pi\*rand(size(ws));
* Ts=1/(8\*max(fs));
* for n=1:Iterations
* t=(n-1)\*Ts;
* s(1:size(ws),n)=(cos(ws\*t));
* signal(n)=sum(s(1:size(ws),n));
* pri\_noise(n)=n\*cos(w0\*t+2\*pi\*n);
* d(n)=signal(n)+pri\_noise(n);
* x(1,1)=C\*cos(w0\*t+phi1);
* x(2,1)=C\*cos(w0\*t+phi2);
* y=conj(w')\*x; e(n)=d(n)-y;
* w=w+2\*mu\*conj(e(n))\*x;
* end
* f=0:100;
* Se=zeros;
* Sd=zeros;
* Ssignal=zeros;
* for n=1:Iterations
* Se=Se+e(n)\*exp(-sqrt(-1)\*2\*pi\*f\*(n-1)\*Ts);
* Sd=Sd+d(n)\*exp(-sqrt(-1)\*2\*pi\*f\*(n-1)\*Ts);
* Ssignal=Ssignal+signal(n)\*exp(-sqrt(-1)\*2\*pi\*f\*(n-1)\*Ts);
* end;
* subplot(2,1,1);
* xlabel('Iterations');
* plot(f,abs(Sd),'b');
* title('Noise Canceller as Notch Filter');
* ylabel('I/p spectrum');
* xlabel('Frequency');
* subplot(2,1,2);
* plot(f,abs(Se),'r');
* ylabel('ANC o/p spectrum');
* xlabel('Frequency');
* text(70,250,'mu = 0.3');
* text(70,210,'C = 0.2');
* **ANC for drift/bias removal**
* clear;
* N=1024;
* mu=0.01; %change mu=0.001 and see the result-mmse vs. speed of adaptation
* ws=200;
* Ts=1/(4\*ws);
* %drift=1.2; %constant bias;
* w(1)=0;
* for n=1:N
* t=(n-1)\*Ts;
* s(n)=cos(ws\*t);
* drift(n)=(-1+exp(0.0008\*n));
* d(n)=s(n)+drift(n);
* drift\_=w(n);
* s\_(n)=d(n)-drift\_;
* e(n)=s\_(n);
* w(n+1)=w(n)+2\*mu\*e(n);
* end
* subplot(2,1,1);
* plot(d);
* title('Bias/Drift removal using ANC');
* ylabel('i/p signal');
* xlabel('Iterations');
* axis([0 N -2 2]);
* subplot(2,1,2);
* plot(s\_);
* ylabel('ANC o/p');
* xlabel('Iterations');
* axis([0 N -2 2]);
* **ANC as adaptive line enhancer**
* clear;
* f=200;
* Ts=1/(16\*f);
* L=1024;
* for l=1:L
* t=(l-1)\*Ts;
* s(l,1)=0.1\*cos(2\*pi\*f\*t);
* end
* N=8;
* for l=1:L
* n(l,1:N)=normrnd(0,0.8,1,N);
* end
* mu=0.01;
* M=128;
* for i=1:N
* w=zeros(M,N);
* d=s+n(1:L,i);
* ref=zeros(M,1);
* for l=1:L
* for k=M:-1:2
* ref(k)=ref(k-1);
* end
* ref(1)=n(l,i);
* d\_out(l)=w(1:M,i)'\*ref;
* e(l)=d(l)-d\_out(l);
* w(1:M,i)=w(1:M,i)+2\*mu\*ref\*conj(e(l));
* end
* end
* w\_avg=zeros(M,1);
* for i=1:N
* w\_avg=w\_avg+w(1:M,i);
* end
* w\_ens=w\_avg/N;
* DFT=dftmtx(length(w\_ens))\*w\_ens;
* Power\_spec=DFT.\*conj(DFT);
* plot(Power\_spec);
* title('ALE - Adaptive Line Enhancer');
* xlabel('frequency');
* ylabel('Transfer function freq. response');
* **ANC using LMS with real time example**
* clear all;
* close all;
* clc;
* tic
* % Storing data of the test signal in double-precision array
* [desired,Fs] = audioread('near\_end.wav');% Loading of test signal
* desired = desired / rms(desired, 1);%Normalization of the signal
* m = length(desired);
* t=(1:m)';
* %Reference signal u(k)
* %refer = wgn(m,1,-10); % Addition of noise directly in MATLAB
* %refer = wgn(m,1,-5);
* %refer = wgn(m,1,0);
* %refer = wgn(m,1,5);
* refer = wgn(m,1,10);
* fil = fir1(11, 0.4);% Designing a FIR filter
* u = filter(fil, 1, refer);%Filtering the reference signal
* %Primary signal s(k)+n(k)
* primary = desired+ u;
* %Calculation of LMS filter weights
* order = 11;
* mu = 0.003642;
* n = length(primary);
* w = zeros(order,1);
* E = zeros(1,m);
* for k = 11:n
* U = u(k-10:k);
* y = U'\*w;
* E(k) = primary(k)-y;
* w = w + mu\*E(k)\*U;
* end
* %Plotting of the signals
* T=(1:m)';
* figure(1);
* subplot(4,1,1);
* plot(t,desired);
* title('Desired Signal in time domain');
* xlabel('Time');
* ylabel('desired signal');
* subplot(4,1,2);
* plot(t,u);
* title('Reference Signal in time domain');
* xlabel('Time');
* ylabel('Reference signal');
* subplot(4,1,3);
* plot(t,primary);
* xlabel('Time');
* ylabel('primary signal');
* title('Primary Signal in time domain');
* subplot(4,1,4);
* plot(T,E);
* xlabel('Time');
* ylabel('Denoised Signal');
* title('Denoised signal in time domain');
* figure(2);
* freqz(desired);
* title('Desired Signal in frequency domain');
* figure(3);
* freqz(u);
* title('Reference Signal in frequency domain');
* figure(4);
* freqz(primary);
* title('Primary Signal in frequency domain');
* figure(5);
* freqz(E);
* title('Denoised signal in frequency domain');
* %Signal to noise ratio(SNR) and varience of desired signal and denoised signal(E)
* disp(snr(desired));%Signal to noise ratio of the desired signal
* disp(var(desired));%Varience of the desired signal
* disp(snr(E));%Signal to noise ratio of the output signal E
* disp(var(E));%Varience of the output signal E
* toc;
* **ANC using NLMS with real time example**
* clc
* clear all;
* close all;
* input = importdata('sample\_data.mat');
* fs = input.fs;
* primary = input.reference;
* reference = input.primary;
* primary\_size = size(primary,2);
* Epsilon = 0.0001;
* AllData = zeros(1,3);
* for order = 30
* W\_1 = zeros(order,1);
* W\_2 = zeros(order,1);
* performance\_curve1 = zeros(46500,1);
* performance\_curve2 = zeros(18461,1);
* primary\_wrt\_filter = primary(1 , order:end); %truncate primary
* reference\_wrt\_filter = zeros((primary\_size - order),order);
* for update = (order) : primary\_size %make reference\_wrt\_filter according to filter
* for update1=1:order
* reference\_wrt\_filter((update-order+1),update1) = reference(update-update1+1);
* end
* end
* disp(size(reference\_wrt\_filter,1));
* % for Nu = 0.001:(2 \* 0.01):1
* for Nu = 0.05
* % for iterateReference = 1: size(reference\_wrt\_filter,1)
* for iterateReference = 1: size(reference\_wrt\_filter,1)
* MSE =0;
* Error = primary\_wrt\_filter(1, iterateReference) - (reference\_wrt\_filter(iterateReference,:) \* W\_2(:,1));
* X = reference\_wrt\_filter(iterateReference,:);
* Nu\_by\_Epsilon = Nu / (Epsilon + (X \* X'));
* if iterateReference < 46501
* Error = primary\_wrt\_filter(1, iterateReference) - (reference\_wrt\_filter(iterateReference,:) \* W\_1(:,1));
* W\_1 = W\_1 + (Nu\_by\_Epsilon \* (Error \* X)');
* errorSquare = (primary\_wrt\_filter(1, 1:iterateReference)' - (reference\_wrt\_filter(1:iterateReference, :) \* W\_1(:,1))).^2;
* MSE = sum(errorSquare)/(iterateReference);
* performance\_curve1(iterateReference,1) = MSE;
* end
* W\_2 = W\_2 + (Nu\_by\_Epsilon \* (Error \* X)');
* if iterateReference >= 46501
* errorSquare1 = (primary\_wrt\_filter(1, 46501:iterateReference)' - (reference\_wrt\_filter(46501:iterateReference, :) \* W\_2(:,1))).^2;
* MSE = sum(errorSquare1)/(iterateReference-46500);
* performance\_curve2(iterateReference-46500,1) = MSE;
* end
* end
* % MSE = sum(primary\_wrt\_filter - (reference\_wrt\_filter \* W\_1)')^2;
* % AllData(size(AllData,1)+1,1) = order;
* % AllData(size(AllData,1),2) = Nu;
* % AllData(size(AllData,1),3) = MSE;
* end
* Out = (primary\_wrt\_filter(1, 1:46500) - (reference\_wrt\_filter(1:46500,:) \* W\_1)');
* Out1 = (primary\_wrt\_filter(1, 46501:end) - (reference\_wrt\_filter(46501:end,:) \* W\_2)');
* Out3 = vertcat(Out', Out1');
* SNR\_parameter = mean(primary\_wrt\_filter.^2)/mean(Out3.^2);
* SNR\_After = 10 \* log10(SNR\_parameter);
* figure;
* plot(performance\_curve1);
* title('Learning Curve For Filter Order = 50 and Iteration < 46.5K');
* xlabel('Iteration -->');
* ylabel('MSE -->');
* legend('Nu = 0.05');
* figure;
* plot(performance\_curve2);
* title('Learning Curve For Filter Order = 50 and Iteration > 46.5K');
* xlabel('Iteration -->');
* ylabel('MSE -->');
* legend('Nu = 0.05');
* %
* figure;
* plot(Out3);
* title('Error Signal After Applying NLMS For Filter Order = 50');
* xlabel('Iteration -->');
* ylabel('Error (Desired Output Signal) -->');
* legend('Nu = 0.05');
* %soundsc(Out3,fs);
* end
* **Project with comparative study of three major algorithms**
* clc;
* clear all;
* close all;
* load('desnoi');
* x2=x2(:);
* s2=s2(:);
* a = input('enter a value'); %initializing the values of a and c for step size
* c = input('enter c value');
* L = 512;
* N=length(x2);
* xin=zeros(L,1);
* w=zeros(L,1);
* %initialization of variables
* y = 0\*x2;
* e = 0\*x2;
* powerS = 0\*x2;
* powerE = 0\*x2;
* %NLMS algorithm
* for i=1:N
* xin = [x2(i); xin(1:end-1)];
* y(i)=w'\*xin;
* error=s2(i)-y(i); %ERROR
* e(i)=error; %Store estimation error
* mu=a/(c+xin'\*xin); %Calculate Step-size
* wtemp = w + 2\*mu\*error\*xin; %Update filter
* w = wtemp;
* powerS(i) = abs(s2(i))^2; %Power of Microphone signal
* powerE(i)=abs(e(i))^2; %power of Error signal
* end
* %LMS algorithm
* for i=1:N
* xin = [x2(i); xin(1:end-1)];
* y1(i)=w'\*xin;
* error1=s2(i)-y1(i); %ERROR
* e1(i)=error1; %Store estimation error
* mu=0.01; %Calculate Step-size
* wtemp = w + 2\*mu\*error1\*xin; %Update filter
* w = wtemp;
* powerS1(i) = abs(s2(i))^2; %Power of Microphone signal
* powerE1(i)=abs(e1(i))^2; %power of Error signal
* end
* %VL-LMS algorithm
* for i=1:N
* xin = [x2(i); xin(1:end-1)];
* y2(i)=w'\*xin;
* error2=s2(i)-y2(i); %ERROR
* e2(i)=error2; %Store estimation error
* mu=0.01; %Calculate Step-size
* gamma=0.05;
* wtemp = (1-2\*mu\*gamma)\*w + 2\*mu\*error2\*xin; %Update filter
* w = wtemp;
* powerS2(i) = abs(s2(i))^2; %Power of Microphone signal
* powerE2(i)=abs(e2(i))^2; %power of Error signal
* end
* for i=1:N-L
* %Echo Return Loss Enhancement
* ERLE(i)=10\*log10(mean(powerS(i:i+L))/mean(powerE(i:i+L))); %Calculating the ERLE
* ERLE1(i)=10\*log10(mean(powerS1(i:i+L))/mean(powerE1(i:i+L))); %Calculating the ERLE
* ERLE2(i)=10\*log10(mean(powerS2(i:i+L))/mean(powerE2(i:i+L))); %Calculating the ERLE
* end
* %-------Echo signal--------
* figure
* subplot(4,1,1)
* plot(s2) %plotting the echo signal
* xlabel('time (samples)','FontSize', 18);
* ylabel('echo(n)','FontSize', 18);
* title('ECHO SIGNAL: echo(n)','FontSize', 18)
* grid on
* axis([0 N -1 1]);
* %-------Input signal-------------------
* subplot(4,1,2)
* % figure\*
* plot(x2) %plotting the input signal
* xlabel('time (samples)','FontSize', 18);
* ylabel('x(n)','FontSize', 18);
* title('INPUT SIGNAL: x(n)','FontSize', 18)
* grid on
* axis([0 N -1 1]);
* %-------Output signal x(n)----------------
* subplot(4,1,3)
* % figure
* plot(y) %plotting output signal
* xlabel('time (samples)','FontSize', 18);
* ylabel('y(n)','FontSize', 18);
* title('OUTPUT SIGNAL y(n) for NLMS','FontSize', 18)
* grid on
* axis([0 N -1 1]);
* %-------Error signal x(n)-----------------
* subplot(4,1,4)
* % figure
* plot(e,'red') %plotting the error signal
* xlabel('time (samples)','FontSize', 18);
* ylabel('E(n)','FontSize', 18);
* title('ERROR SIGNAL: e(n) for NLMS','FontSize', 18)
* axis([0 N -1 1]);
* grid on
* %ERLE calculation
* % ylabel('Desired signal/Error signal (dB)','FontSize', 20);
* figure
* subplot(311)
* plot(ERLE)
* hold on; %plotting the ERLE w.r.t desired signal
* plot(s2,'r');
* xlabel('Sample number (n)','FontSize', 20);
* % ylabel('Desired signal/Error signal (dB)','FontSize', 20);
* legend('ERLE', 'DESIRED SIGNAL')
* title('ECHO RETURN LOSS ENHANCEMENT for NLMS','FontSize', 20)
* axis([0 N -20 40]);
* grid on
* subplot(312)
* plot(ERLE1)
* hold on; %plotting the ERLE w.r.t desired signal
* plot(s2,'r');
* xlabel('Sample number (n)','FontSize', 20);
* % ylabel('Desired signal/Error signal (dB)','FontSize', 20);
* legend('ERLE', 'DESIRED SIGNAL')
* title('ECHO RETURN LOSS ENHANCEMENT for LMS','FontSize', 20)
* axis([0 N -20 40]);
* grid on
* subplot(313)
* plot(ERLE2)
* hold on; %plotting the ERLE w.r.t desired signal
* plot(s2,'r');
* xlabel('Sample number (n)','FontSize', 20);
* % ylabel('Desired signal/Error signal (dB)','FontSize', 20);
* legend('ERLE', 'DESIRED SIGNAL')
* title('ECHO RETURN LOSS ENHANCEMENT for LLMS','FontSize', 20)
* axis([0 N -20 40]);
* grid on
* %LMS algorithm
* figure
* subplot(4,1,1)
* plot(s2) %plotting the echo signal
* xlabel('time (samples)','FontSize', 18);
* ylabel('echo(n)','FontSize', 18);
* title('ECHO SIGNAL: echo(n)','FontSize', 18)
* grid on
* axis([0 N -1 1]);
* %-------Input signal-------------------
* subplot(4,1,2)
* % figure\*
* plot(x2) %plotting the input signal
* xlabel('time (samples)','FontSize', 18);
* ylabel('x(n)','FontSize', 18);
* title('INPUT SIGNAL: x(n)','FontSize', 18)
* grid on
* axis([0 N -1 1]);
* % figure
* subplot(4,1,3)
* plot(y1) %plotting output signal
* xlabel('time (samples)','FontSize', 18);
* ylabel('y(n)','FontSize', 18);
* title('OUTPUT SIGNAL y(n) for LMS','FontSize', 18)
* grid on
* axis([0 N -1 1]);
* %-------Error signal x(n)-----------------
* subplot(4,1,4)
* % figure
* plot(e1,'red') %plotting the error signal
* xlabel('time (samples)','FontSize', 18);
* ylabel('E(n)','FontSize', 18);
* title('ERROR SIGNAL: e(n) for LMS','FontSize', 18)
* axis([0 N -1 1]);
* grid on
* % LLMS agorithm
* figure
* subplot(4,1,1)
* plot(s2) %plotting the echo signal
* xlabel('time (samples)','FontSize', 18);
* ylabel('echo(n)','FontSize', 18);
* title('ECHO SIGNAL: echo(n)','FontSize', 18)
* grid on
* axis([0 N -1 1]);
* %-------Input signal-------------------
* subplot(4,1,2)
* % figure\*
* plot(x2) %plotting the input signal
* xlabel('time (samples)','FontSize', 18);
* ylabel('x(n)','FontSize', 18);
* title('INPUT SIGNAL: x(n)','FontSize', 18)
* grid on
* axis([0 N -1 1]);
* % figure
* subplot(413)
* plot(y2) %plotting output signal
* xlabel('time (samples)','FontSize', 18);
* ylabel('y(n)','FontSize', 18);
* title('OUTPUT SIGNAL y(n) for LLMS','FontSize', 18)
* grid on
* axis([0 N -1 1]);
* %-------Error signal x(n)-----------------
* subplot(4,1,4)
* % figure
* plot(e2,'red') %plotting the error signal
* xlabel('time (samples)','FontSize', 18);
* ylabel('E(n)','FontSize', 18);
* title('ERROR SIGNAL: e(n) for LLMS','FontSize', 18)
* axis([0 N -1 1]);
* grid on