Script:

Wiener filter:

%% EEL 5840/EEL4930 Elements of Machine Intelligence

clear

close all

clc

%% Initialize system

music = importdata('music.txt');

mix = importdata('corrupted\_speech.txt');

fs = importdata('fs.txt');

music = (music-mean(music))';

mix = (mix-mean(mix))';

% w = 50000;

% w\_start = 5;%140000;%

% music = music(w\_start:w\_start+w);

% mix = mix(w\_start:w\_start+w);

%%

tic

M\_list = 5:5:100;

lambda\_list = 0:0.01:0.5;

N = length(music);

MSE = zeros(length(M\_list),length(lambda\_list));

NMSE = zeros(length(M\_list),length(lambda\_list));

W = cell(length(M\_list),length(lambda\_list));

for idx\_M = 1:length(M\_list)

    d\_mix = mix(M\_list(idx\_M):(end-1));

    for idx\_lambda = 1:length(lambda\_list)

        [W{idx\_M,idx\_lambda}, speech{idx\_M,idx\_lambda}, MSE(idx\_M,idx\_lambda), NMSE(idx\_M,idx\_lambda), erle\_wiener(idx\_M,idx\_lambda)] =...

            Wiener\_Estimization(music,d\_mix,M\_list(idx\_M),lambda\_list(idx\_lambda));

    end

    display(['Filter order: ',num2str(M\_list(idx\_M)),'/',num2str(M\_list(end))]);

end

toc

%% Plots

% [a\_min,b\_min]=find(MSE==min(min(MSE)));

% display(['MSE is minimum for filter order=',num2str(M\_list(a\_min)),...

%     ' and regularization=',num2str(lambda\_list(b\_min))]);

% figure,

% surf(M\_list,lambda\_list,MSE');xlabel('Filter order M');zlabel('Mean Square Error (MSE)');

% ylabel('Regularization parameter \lambda');colorbar;

% title(['Minimum MSE for M=',num2str(M\_list(a\_min)),' and \lambda=',num2str(lambda\_list(b\_min))]);

%%

% [c\_min,d\_min]=find(NMSE==min(min(NMSE)));

% display(['NMSE is minimum for filter order=',num2str(M\_list(c\_min)),...

%     ' and regularization=',num2str(lambda\_list(d\_min))]);

% figure,

% surf(M\_list,lambda\_list,NMSE');xlabel('Filter order M');zlabel('Normalized Mean Square Error (NMSE)');

% ylabel('Regularization parameter \lambda');colorbar;

% title(['Minimum NMSE for M=',num2str(M\_list(c\_min)),' and \lambda=',num2str(lambda\_list(d\_min))]);

%

% figure,stem(W{a\_min,b\_min})

% xlabel('Time lags/taps');

% ylabel('Weight Coefficients');

% ylim([-1 1])

% title(['Weights for Filter Order = ' num2str(M\_list(c\_min))])

%

% figure,stem(2:96, W{a\_min,b\_min}(2:96))

% xlabel('Time lags/taps');

% ylabel('Weight Coefficients');

% ylim([-0.3 0.15])

% title('Zoom from 2 to 96')

[M\_max, lambda\_max] = find(erle\_wiener == max(max(erle\_wiener)));

display(['ERLE is maximum for filter order=',num2str(M\_list(M\_max)),...

    ' and regularization=',num2str(lambda\_list(lambda\_max))]);

% figure

% plot(M\_list, erle\_wiener(:,lambda\_max)', 'Linewidth',4)

% hold on

% title('ERLE Curve as a Function of Filter Order - Wiener Filter')

% legend(['\lambda = ' num2str(lambda\_list(lambda\_max))])

figure

plot(mix)

hold on

plot(speech{M\_max, lambda\_max})

legend('Corrupted Speech', 'Recovered Speech')

title('Comparison of Corrupted Speech and Recovered Speech - Wiener Filter')

Function:

Wiener\_estimization:

function [W\_optimal,E\_wiener,MSE,NMSE,erle\_wiener] = Wiener\_Estimization(input,desire,order,reg)

% This function implements the Wiener solution for Echo Cancellation

%

% INPUT

% input: input signal

% desire: desired signal

% order: filter order

% reg: (optional) regularizer parameter. Default value is reg=0.

%

% OUTPUT

% W\_optimal: analytical weights

% E\_wiener: error signal

% MSE: mean squared error

% NMSE: normalized mean squared error

% erle\_wiener: ERLE

%

%

if nargin < 4

    reg = 0;

end

Xmean=mean(input.^2);       %input power

% D = desire(order+1:end);      %desired response

D = desire;

% Construction of the input matrix

DataMatrix = zeros(order,length(input)-order);

for i = 1:(length(input)-order)

    DataMatrix(:,i)=input(i+order-1:-1:i);

end

% Computation of R and P

R = DataMatrix\*DataMatrix'/length(DataMatrix);     %auto-correlation

P = DataMatrix\*D/length(DataMatrix);      %cross-correlation

% Adding regularizer term to R

Rreg = R + reg.\*eye(order);

% Optimal Weights

W\_optimal = Rreg\P;       %weights

% Error

E\_wiener = D - DataMatrix'\*W\_optimal;

% MSE

MSE = mean(E\_wiener.^2);

% Normalized MSE

NMSE = mean(E\_wiener.^2)/Xmean;

% ERLE

erle\_wiener = ERLE(D,E\_wiener);

Script:

LMS filter:

close all

clc

dbstop if error

load('music.txt')

load('corrupted\_speech.txt')

load('fs.txt')

music = (music - mean(music))';

corrupted\_speech = (corrupted\_speech - mean(corrupted\_speech))';

%%

M\_list = 5:5:100;

ita\_list = [10^(-6) 10^(-5) 10^(-4) 10^(-3)];

% ita\_list = []

N = length(music);

MSE = zeros(length(M\_list), length(ita\_list));

% Wk = cell(length(M\_list) , length(ita\_list));

tic

for iter = 1:100;%:max\_iter

    for idx\_M = 1:length(M\_list)

        d\_mix = corrupted\_speech(M\_list(idx\_M):(end-1));

        for idx\_ita = 1:length(ita\_list)

        ww{idx\_M, idx\_ita}(:,1) = zeros(M\_list(idx\_M),1);

%         [Wk{idx\_M, idx\_ita}, Ek{idx\_M, idx\_ita}, ~, ~,Xk{idx\_M}] = LMS\_prediction(music, d\_mix, M\_list(idx\_M), ita\_list(idx\_ita), 1, 0);

            [Wk{idx\_M, idx\_ita},Ek{idx\_M, idx\_ita},~,Xk{idx\_M}] = LMS\_estimation(music,d\_mix,M\_list(idx\_M),ita\_list(idx\_ita),1,ww{idx\_M, idx\_ita}(:,iter));

            speech{idx\_M, idx\_ita} = d\_mix - Xk{idx\_M}'\* Wk{idx\_M, idx\_ita}(:,end);

            [erle{idx\_M, idx\_ita}] = ERLE(d\_mix,speech{idx\_M, idx\_ita});

            ww{idx\_M, idx\_ita}(:,iter+1) = Wk{idx\_M, idx\_ita}(:,end);

        end

    end

    display(['Order ',num2str( M\_list(idx\_M)),'/',num2str(ita\_list(idx\_ita)),' done!'])

end

Function:

LMS\_Estimization:

function [Wk,Ek,Y,R,Xk] = LMS\_estimation(X,Dk,order,mu,method,random)

% function [Wk,Ek,Y,Xk] = LMS\_estimation(X,Dk,order,eta,method,W)

%This funtion performs adaptive LMS/NLMS filter given the input data, desired

%output data, order and gain constant. It returns the weight values, the

%error, the output, the NMSE and auto-correlation function.

%

%   X     : input signal

%   Dk    : desired signal

%   order : order of LMS filter

%   mu    : learning rate (gain constant)

%   method: if 1, uses regular LMS. If 2, uses NLMS. default NLMS

%   random: if 1, uses random initialization of W. If 0, uses zeros as

%   weight initialization.

%

%   Wk    : weights

%   Ek    : error

%   Y     : output

% NMSE    : MSE normalized by the input power

%   R     : Auto-Correlation Function

%  MSE    : Mean Squared Error

%

%

% if nargin < 5

%     method = 2; %NLMS

%     random = 1; %random initialization of the weights

% end

if nargin < 6

    W = zeros(order,1); %random initialization of the weights

end

Samples = length(X); %number of samples

% Initialization

Xk = zeros(order,length(X)-order);

Ek=zeros(1,Samples);

Y=zeros(1,Samples);

% Input-delayed Matrix

for i=1:length(X)-order

    Xk(:,i)=X(i+order-1:-1:i);

end

% Auto-correlation Matrix

% R = Xk\*Xk'./length(Xk);

% % Choose between random or zero initialization of the weights

% if random==1

%     W = randn(order,1); %random

% else

%     W=zeros(order,1); %zeros

% end

if method ==1 % LMS algorithm

for k=1:Samples-order

    Y(k)= Xk(:,k)'\*W; %output

    E = Dk(k) - Y(k); %instantaneous error

    Ek(:,k)=E;

    W = W + 2 \* eta \* E \* Xk(:,k); %weight update equation

    Wk(:,k)=W;

    NMSE(:,k) = mean((Dk-Xk'\*W).^2)/mean(X.^2); %local NMSE

    MSE(:,k) = mean((Dk-Xk'\*W).^2); %local MSE

end

elseif method ==2 % NLMS algorithm

    reg = 10^-10; %regularization term

    for k=1:Samples-order

    Y(k)= Xk(:,k)'\*W; %output

    E = Dk(k) - Y(k); %instantaneous error

    Ek(:,k)=E;

    W = W + 2 \* eta \* E \* Xk(:,k) ./ (reg+(Xk(:,k)'\*Xk(:,k))); %weight update equation

    Wk(:,k)=W;

    NMSE(:,k) = mean((Dk-Xk'\*W).^2)/mean(X.^2); %local NMSE

    MSE(:,k) = mean((Dk-Xk'\*W).^2); %local MSE

    end

end

Script:

Gamma filter:

clear

close all;

clc;

dbstop if error

%% Initialize system

fs = importdata('fs.txt');

music = importdata('music.txt');

mix = importdata('corrupted\_speech.txt');

music = (music-mean(music))';

mix = (mix-mean(mix))';

tic

M\_list = 5:5:100;

eta\_list = [10^(-5) 10^(-4) 5\*10^(-4) 10^(-3) 5\*10^(-3)];

N = length(mix);

mu\_list = 0.2;

idx\_mu = 1;

tic

display('Order | Step-size');

% for iter = 1:20

iter = 1;

for idx\_M = 1:length(M\_list)

    d\_mix = mix(M\_list(idx\_M):end);

    for idx\_eta = 1:length(eta\_list)

        ww{idx\_M, idx\_eta}(:,1) = zeros(M\_list(idx\_M),1);

        [Wk{idx\_M, idx\_eta}, Ek{idx\_M, idx\_eta}, ~, Xk{idx\_M}] = GAMMA\_estimization(music,d\_mix,M\_list(idx\_M),eta\_list(idx\_eta),1,mu\_list(idx\_mu), ww{idx\_M, idx\_eta}(:,iter));

%         [Wk{idx\_M, idx\_ita}, Ek{idx\_M, idx\_ita}, ~, Xk{idx\_M}, MSE{idx\_M, idx\_ita}] = GAMMA\_estimization(music,d\_mix,M\_list(idx\_M),ita\_list(idx\_ita),1,mu\_list(idx\_mu), ww{idx\_M, idx\_ita}(:,iter));

        speech{idx\_M, idx\_eta} = d\_mix - Xk{idx\_M}'\* Wk{idx\_M, idx\_eta}(:,end);

        [erle{idx\_M, idx\_eta}] = ERLE(d\_mix,speech{idx\_M, idx\_eta});

        ww{idx\_M, idx\_eta}(:,iter+1) = Wk{idx\_M, idx\_eta}(:,end);

    end

    display(['Order ',num2str(M\_list(idx\_M)),'/',num2str(length(M\_list)),' done!'])

end

% end

% save('Gamma\_weight.mat', 'ww')

[erle{idx\_M, idx\_eta}] = ERLE(d\_mix,speech{idx\_M, idx\_eta});

toc

erle = cell2mat(erle);

[idx\_Mmax, idx\_itamax] = find(erle == max(max(erle)));

display(['ERLE is maximum for filter order=',num2str(M\_list(idx\_Mmax)),' and step size=',num2str(eta\_list(idx\_itamax))]);

figure

plot(M\_list,erle,'Linewidth',2)

legend(['\eta = ' num2str(eta\_list(1))])

xlabel('Filter Order')

ylabel('Echo Return Loss Enhancement')

title(['ERLE curve as a function of the filter order - Gamma Filter with \mu = ' num2str(mu\_list(idx\_mu))])

figure

plot(mix)

hold on

plot(speech{idx\_Mmax})

title(['Comparison of Corrupted Speech and Recovered Speech - Gamma Filter with \mu = ' num2str(mu\_list(idx\_mu))])

legend('Corrupted Speech', 'Recovered Speech')

Function:

Gamma \_estimization:

% function [Wk,Ek,Y,Xk,MSE] = GAMMA\_estimization(X,Dk,order,eta,method,mu,W)

function [Wk,Ek,Y,Xk] = GAMMA\_estimization(X,Dk,order,eta,method,mu,W)

%This funtion performs adaptive LMS/NLMS filter given the input data, desired

%output data, order and gain constant. It returns the weight values, the

%error, the output, the NMSE and auto-correlation function.

%

%   X     : input signal

%   Dk    : desired signal

%   order : order of LMS filter

%   ita    : learning rate (gain constant)

%   mu  :

%   method: if 1, uses regular LMS. If 2, uses NLMS. default NLMS

%   random: if 1, uses random initialization of W. If 0, uses zeros as

%   weight initialization.

%

%   Wk    : weights

%   Ek    : error

%   Y     : output

% NMSE    : MSE normalized by the input power

%   R     : Auto-Correlation Function

%  MSE    : Mean Squared Error

%

%

% if nargin < 5

%     method = 2; %NLMS

%     random = 1; %random initialization of the weights

% end

if nargin < 7

    W = zeros(order,1); %random initialization of the weights

end

Samples = length(X); %number of samples

% Initialization

% Xk = zeros(order+1,length(X)-order);

Xk = zeros(order,length(X)-order+1);

Ek=zeros(1,Samples);

Y=zeros(1,Samples);

% Input-delayed Matrix

% Xk(1,:) = X(order+1:length(X));

% Xk(:,1) = X(order+1:-1:1);

Xk(1,:) = X(order:length(X));

Xk(:,1) = X(order:-1:1);

% for i=2:order+1

for i=2:order

    for j = 2:(length(X)-order+1)

    Xk(i,j) = (1-mu)\*Xk(i,j-1)+mu\*Xk(i-1, j-1);

    end

end

% Auto-correlation Matrix

% R = Xk\*Xk'./length(Xk);

% Choose between random or zero initialization of the weights

% if random==1

%     W = randn(order,1); %random

% else

%     W=zeros(order,1); %zeros

% end

if method ==1 % LMS algorithm

for k=1:Samples-order

    Y(k)= Xk(:,k)'\*W; %output

    E = Dk(k) - Y(k); %instantaneous error

    Ek(:,k)=E;

    W = W + 2 \* eta \* E \* Xk(:,k); %weight update equation

    Wk(:,k)=W;

%     NMSE(:,k) = mean((Dk-Xk'\*W).^2)/mean(X.^2); %local NMSE

%     MSE(:,k) = mean((Dk-Xk'\*W).^2); %local MSE

end

elseif method ==2 % NLMS algorithm

    reg = 10^-10; %regularization term

    for k=1:Samples-order

    Y(k)= Xk(:,k)'\*W; %output

    E = Dk(k) - Y(k); %instantaneous error

    Ek(:,k)=E;

    W = W + 2 \* eta \* E \* Xk(:,k) ./ (reg+(Xk(:,k)'\*Xk(:,k))); %weight update equation

    Wk(:,k)=W;

%     NMSE(:,k) = mean((Dk-Xk'\*W).^2)/mean(X.^2); %local NMSE

%     MSE(:,k) = mean((Dk-Xk'\*W).^2); %local MSE

    end

end

Function:

ERLE:

function [erle] = ERLE(d,e)

% This function implements SNR improvement in dB by the

%            ERLE = 10\*log(E{d^2}/E{e^2})

%

% INPUT

% d: desired signal

% e: error signal

%

% OUTPUT

% erle: SNR in dB

%

D2 = mean(d.^2); %power of the desired signal

E2 = mean(e.^2); %power of the error signal

f = D2/E2; % ratio

erle = 10\*log10(f); % dB