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
```