נושא:

Simulation for noise cancellation using LMS adaptive filter

מנחים:   
גולן עין צבי ופרופסור יוסף בן עזרא

מגישים:   
רוסלן אוסמנוב - 327480026

טאל חיים - 312464639

The process included: 1

Load noise free and noise signals 2

Plot corrupted ,noise free and noise signal 2

LMS Adapt Filter 3

Frequency Response of Adaptive filter 3

Spectogram 3

Time Domain plots 4

Plot of the coeffs and weights 4

Frequency domain plots 5

%In this model we will show the process of canceling noise from corrupted  
%signal using an adaptive filter applied by LMS algorithem as an  
%implementation to an article called:  
%'Simulation for noise cancellation using LMS adaptive filter.'

The process included:

% 1. Import and export noise free and noise signal as a wav file.  
% 2. Implement noise components on the signal file randomly and create correlated noise   
% 3. Plot corrupted ,noise free and noise signal  
% 4. Run LMS algorithm according to:  
% Diffirent Length of FIR filter (M)  
% Diffirent Step Size (mu)  
% 5. Plot frequency responce of the last itteration coefficents  
% 6. Plot Spectogram of filtered ,noise free and corrupted signals  
% 7. Plots:  
% Absolute error beetwen noise free and filtered signal  
% Noise free signal  
% Filtered signal  
% 8. Relative error beetwen original and filtered signal  
% 9. Plots of signal and filter result combined  
%10. Plots of the coeffs and weights:  
% Initial Weights,Adapted Final Weights  
% Coefficient Trajectories  
%11. FFT:  
% Noise free signal  
% Filtered signal  
% Corrupted signal  
% Noise signal  
  
close all  
clear all

Load noise free and noise signals

[signal,Fs] = audioread('Signal.wav');  
[noise,~] = audioread('Noise1.wav');  
  
% Set the noise as a random configuration  
index = randi(numel(noise) - numel(signal) + 1,1,1);  
noiseSegment = noise(index:index + numel(signal) - 1);  
% To ensure that the noise is correlated, pass the noise through  
% a lowpass FIR filter and then add the filtered noise to the signal.  
filt = dsp.FIRFilter;  
filt.Numerator = fir1(11,0.8);  
fnoise = filt(noiseSegment);  
freqz(filt)  
%fnoise = circshift(noiseSegment,11); % shift noise for correlated noise  
% Calculate the power components of the siganls  
speechPower = sum(signal.^2);  
noisePower = sum(fnoise.^2);  
noise\_factor =sqrt(speechPower/noisePower); % snr  
  
% Define corrupted signal with noise factor  
d = signal + noise\_factor\*fnoise;  
corrcoef(fnoise,noiseSegment)

Plot corrupted ,noise free and noise signal

figure(1)  
dt = 1/Fs;  
t = 0:dt:(length(signal)-1)\*dt; % create time vector  
  
subplot(3,1,1)  
plot(t,signal);  
title('Noise free signal');  
xlabel('Time[s]');  
ylabel('Amplitude');  
  
subplot(3,1,2)  
plot(t,noiseSegment);  
title('Noise signal');  
xlabel('Time[s]');  
ylabel('Amplitude');  
  
subplot(3,1,3)  
plot(t,d);  
title('Corrupted signal');  
xlabel('Time[s]');  
ylabel('Amplitude');  
linkaxes([subplot(3,1,1) subplot(3,1,2) subplot(3,1,3)], 'xy');

LMS Adapt Filter

mu = input('Step size mu = '); % Set the step size  
M = input('Length of sequence M = '); % Filter length (num of taps)  
model\_info = strcat('\mu : ',string(mu) ,' M : ',string(M));  
  
% Initialization of weights  
coeffs = zeros(M,1); % column vector of init weights  
S.coeffs = coeffs; % insert weights to struct  
S.step = mu; % insert step size to the struct  
  
% Perform LMS-algo  
[~,e,S] = LMSadapt(noiseSegment,d,S);  
w = S.coeffs;  
% lms\_nonnormalized = dsp.LMSFilter(M,'StepSize',mu,...  
% 'Method','LMS','InitialConditions',coeffs);  
% [~,e,w] = lms\_nonnormalized(noiseSegment,d);

Frequency Response of Adaptive filter

figure(2)  
[h,f] = freqz(w,1,[],Fs);  
subplot(2,1,1);  
hold on  
plot(f,20\*log10(abs(h)),'DisplayName',model\_info); % we will use 20log10() for ploting the mag. response in dB  
title('Magnitude response')  
grid on % turning the grid on  
xlabel('Frequency(Hz)')  
ylabel('Magnitude(dB)')  
legend  
subplot(2,1,2);  
hold on  
legend  
plot(f,rad2deg(angle(h)),'DisplayName',model\_info);  
title('Phase response')  
grid on  
xlabel('Frequency(Hz)')  
ylabel('Phase(degree)')  
hold off

Spectogram

figure(3)  
subplot(3,1,1)  
spectrogram(signal,128,120,[],Fs,'yaxis' );  
title('Noise free signal');  
subplot(3,1,2)  
spectrogram(d,128,120,[],Fs,'yaxis' );  
title('Corrupted signal');  
subplot(3,1,3)  
spectrogram(e,128,120,[],Fs,'yaxis' );  
title(strcat('Filtered signal ',model\_info));  
linkaxes([subplot(3,1,1) subplot(3,1,2) subplot(3,1,3)], 'xy');  
%view(0,0)

Time Domain plots

figure(4)  
subplot(3,1,1)  
hold on  
plot(t,e-signal,'DisplayName',model\_info);% Filt.effectiveness  
title('Error beetwen noise free and filtered signals');  
xlabel('Time[s]');  
ylabel('Amplitude');  
legend  
disp(['Relative error beetwen noise free and filtered signal :',num2str(norm(e-signal)/norm(signal)\*100) ,' %'])  
hold off  
  
subplot(3,1,2)  
plot(t,signal);  
title('Noise free signal');  
xlabel('Time[s]');  
ylabel('Amplitude');  
  
subplot(3,1,3)  
plot(t,e);  
title(strcat('Filtered signal ',model\_info));  
xlabel('Time[s]');  
ylabel('Amplitude');  
linkaxes([subplot(3,1,1) subplot(3,1,2) subplot(3,1,3)], 'xy');  
% Combined plot of noise free and filter signal  
figure(5)  
plot(t,e,t,signal);  
legend('Filtered signal','Noise free signal');  
title('Result of noise cancellation');  
xlabel('Time[s]');  
ylabel('Amplitude');

Plot of the coeffs and weights

figure(6)  
subplot(2,1,1)  
stem(coeffs)  
hold on  
stem(w)  
legend('Initial Weights','Adapted Final Weights');  
title(strcat('Coefficents of the FIR filter ',model\_info));  
xlabel('Taps');  
ylabel('Coeff');  
hold off  
  
subplot(2,1,2)  
nn = length(e);  
plot(1:nn,S.W(:,1:nn))  
title('Coefficient Trajectories');  
xlabel('Iteration');  
ylabel('Coeff');  
legend(string(1:M),'Location','best')  
legend('boxoff')

Frequency domain plots

figure(7)  
limit = [-4e3,4e3];% Relevant spectrum of regular speech frequency  
subplot(4,1,1)  
[FFT\_amp,FFT\_freq] = FFT(Fs,signal,0);  
plot(FFT\_freq,FFT\_amp)  
xlim(limit)  
title('Noise free signal');  
xlabel('Frequency[Hz]');  
ylabel('Amplitude');  
  
subplot(4,1,2)  
[FFT\_amp\_n,FFT\_freq] = FFT(Fs,noise,0);  
plot(FFT\_freq,FFT\_amp\_n)  
xlim(limit)  
title('Noise Signal');  
xlabel('Frequency[Hz]');  
ylabel('Amplitude');  
  
subplot(4,1,3)  
[FFT\_amp\_d,FFT\_freq] = FFT(Fs,d,0);  
plot(FFT\_freq,FFT\_amp\_d)  
xlim(limit)  
title('Corrupted signal');  
xlabel('Frequency[Hz]');  
ylabel('Amplitude');  
  
subplot(4,1,4)  
[FFT\_amp\_filt,FFT\_freq] = FFT(Fs,e,0);  
plot(FFT\_freq,FFT\_amp\_filt)  
xlim(limit)  
title(strcat('Filter signal ',model\_info));  
xlabel('Frequency[Hz]');  
ylabel('Amplitude');  
  
linkaxes([subplot(4,1,1) subplot(4,1,2) subplot(4,1,3) subplot(4,1,4)], 'x');

sound(e,44100)

sound(d,44100)

sound(signal,44100)

sound(noise,44100)

Functions

function [yn,en,S] = LMSadapt(un,dn,S)  
 mu = S.step;  
 N = length(un); % number of samples un = dn  
 yn = zeros(N,1); % initialize filter output vector (estimation y2')  
 w = S.coeffs; % initialize filter coefficient vector  
 en = zeros(N,1); % initialize error vector  
 M = length(S.coeffs);  
 S.W = zeros(M,N); % filter coefficient matrix for coeff. history  
 for i = 1:N  
 if i <= M % assume zero-samples for delayed data that isn't available  
 k = i:-1:1;  
 u = [un(k); zeros(M-numel(k),1)];  
 else  
 u = un(i:-1:i-M+1); % M samples of x in reverse order  
 end  
 yn(i) = w'\*u; % filter output  
 en(i) = dn(i) - yn(i); % error  
 w = w + mu\*en(i)'\*u; % update filter coefficients  
 S.W(:,i) = w; % store current filter coefficients in matrix  
 end  
 S.coeffs = w ;  
end

function [FFT\_amp,FFT\_freq] = FFT(Fs,signal,display\_plot)  
 N = 2^nextpow2(10\*length(signal));  
 yf\_singal = abs(fftshift(fft(signal,N)));  
 FFT\_freq = linspace(-Fs/2,Fs/2,N);  
  
 Norm\_factor =1/length(signal);  
 FFT\_amp =(Norm\_factor\*yf\_singal);  
% FFT\_amp =20\*log10((Norm\_factor\*yf\_singal));  
 if display\_plot==1  
 figure('Name','Fast Fourier Transform')  
 plot(FFT\_freq,FFT\_amp)  
 xlim([-8e3,8e3])  
 end  
end

[*Published with MATLAB® R2020b*](https://www.mathworks.com/products/matlab)