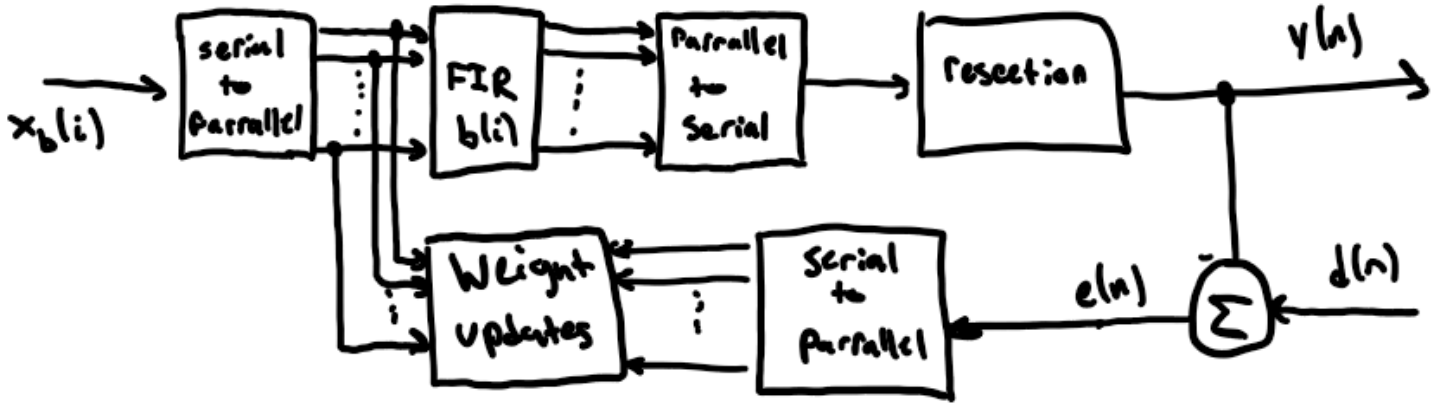
1) Give a block diagram and state the main equations of the BLMS algorithm. Derive in detail convergence in the mean (try and describe in your own words the algorithm and its convergence – and any assumptions made).



i = iteration/block index

input matrix = $x_B(i) =$

$$\begin{bmatrix} x(is) & x(is-1) \cdots & x(is-L) \\ x(is+1) & x(is) \cdots & x(is-L+1) \\ \vdots & & \\ x(is+N-1) & x(is+N-2) \cdots & x(is+N-L-1) \end{bmatrix}$$

weight vector = $b(i) = \begin{bmatrix} b_0(i) \\ b_1(i) \\ \vdots \\ b_L(i) \end{bmatrix}$ error $e(i) = \begin{bmatrix} e(is) \\ e(is+1) \\ \vdots \\ e(is+N-1) \end{bmatrix}$

desired response = $d(i) = \begin{bmatrix} d(is) \\ d(is+1) \\ \vdots \\ d(is+N-1) \end{bmatrix}$ output of filter = $y(i) = \begin{bmatrix} y(is) \\ y(is+1) \\ \vdots \\ y(is+N-1) \end{bmatrix}$

important equations

$$y(i) = X_B(i)b(i)$$

$$e(i) = d(i) - y(i)$$

$$\nabla_B \approx -\frac{2}{N} X_B^T e(i)$$

$$b(i+1) = b(i) - \mu \nabla_B$$

This algorithm works by taking a block of data, and organizing it into the matrix described by X_b above. After multiplying by filter weights, the error can be calculated from the desired signal. From the error a gradient can be calculated and used to update the filter coefficients. The next block can reuse data by not shifting by the entire block, which can help converge more quickly at the expense of more computations. Rinse and repeat until converging.

Proof assuming input signal is stationary

$$\varepsilon_B = \frac{1}{L} E(e_i^T e_i) = E\left(\frac{1}{L}\sum_{k=iL}^{jL} e^2\right) \rightarrow \text{We are trying to minimize MSE}$$

$$= \left(d(i) - y(i)\right)^2 \rightarrow \text{equals } e^2(i) \text{ from eq above}$$

$$= E\left[d(i)d(i)\right] - E\left[d(i)y(i)\right] - E\left[d(i)y(i)\right] + E\left[y^T(i)y(i)\right] \rightarrow \text{expanding}$$

$$= E\left[d(i)^2\right] - 2E\left[\underbrace{d(i)x(i)}_{P}\right]b(i) + b(i)E\left[\underbrace{x(i)^T x(i)}_{R}\right]b(i) \rightarrow \text{back substituting}$$

$$\frac{\partial \varepsilon_B}{\partial b} = E\left[\cancel{d(i)^2}\right] - 2P \cdot b(i) + b(i) \cdot R\, b(i) \rightarrow \text{take partial derivative with respect to } b \text{ and set it equal to zero}$$

$$= -2P + 2R \cdot b(i) = 0 \rightarrow$$

$$\cancel{2}RB = \cancel{2}P \rightarrow \text{solving for } b$$

$$b_o = R^{-1}P \rightarrow \text{the Wiener Solution}$$

as stated in Clark and Mitra paper the LMS algorithm is a special case of the block LMS algorithm where the length of the block is one. It therefore makes sense that the Wiener solution would hold here.

$$E\left[\hat{b}(n+1)\right] = E\left[\hat{b}(n)\right] + 2\mu R\left(b_o - E\left(\hat{b}(n)\right)\right)$$

$$= \left(I - 2\mu R\right)E\left[\hat{b}(n)\right]$$

$$= \left(I - 2\mu Q \Lambda Q^T\right)E\left[\hat{b}(n)\right] \Rightarrow \tilde{b}(n) = Q^T E\left[\hat{b}(n)\right]$$

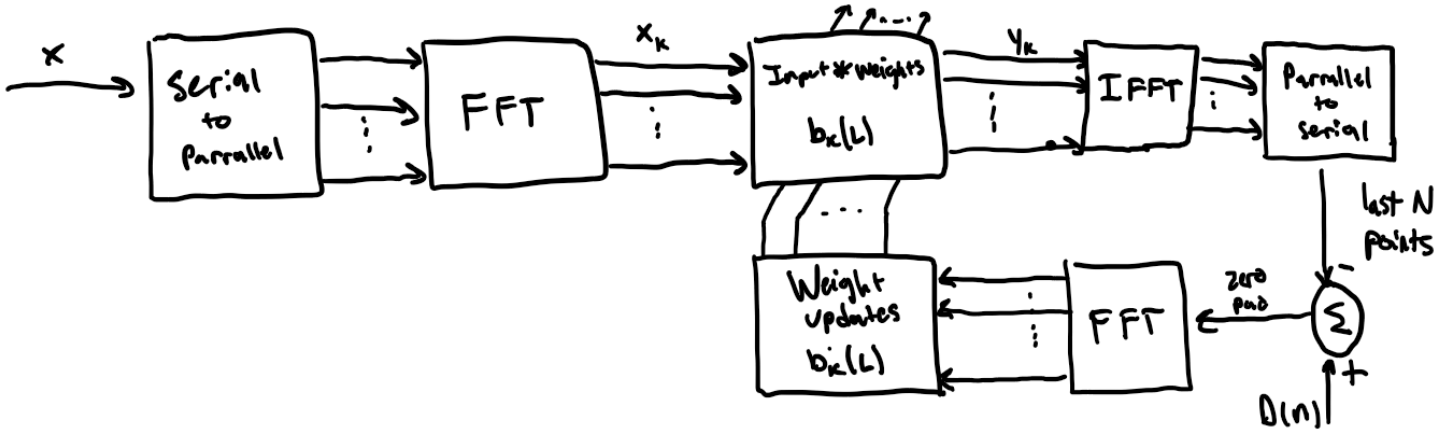
$$\tilde{b}(n+1) = \left(I - 2\mu\Lambda\right)\tilde{b}(n)$$

$$\tilde{b}(n) = \left(I - 2\mu\Lambda\right)\tilde{b}(0)$$

stable if $|I - 2\mu\Lambda| < I$

$$\boxed{0 < \mu < \frac{1}{\lambda_{max}}}$$

2) State the frequency-domain implementation of BLMS and give a block diagram.



The frequency implementation is quite similar to the BLMS but rather than doing the convolution on a block, a Fast convolution is performed. Where an FFT of the block is taken and is multiplied by a filter in the frequency domain. The output is then taken back to the time domain to calculate the error. The error is then taken back to frequency domain to update the weights in the frequency domain

3) Express the BLMS algorithm in MATLAB. Profile the algorithm in terms of CPU time.
4) Express the frequency domain implementation of BLMS in MATLAB and profile the algorithm in terms of CPU time. Show that the results of the frequency domain implementation match the time domain (Hint: you need to match block sizes and μ to get consistency between time and frequency domain).

Below is the output from my code in matlab comparing the time for the 2 algorithms. Nb is my block size, The first plots are in MSE since for parts 3 and 4 its not specified and the rest are in NEE. Its easy to see that with the right mu value that the algorithms behave almost identically. When looking at the MSE plots they converge at about the same time with the same slope and about the same mis adjustment.

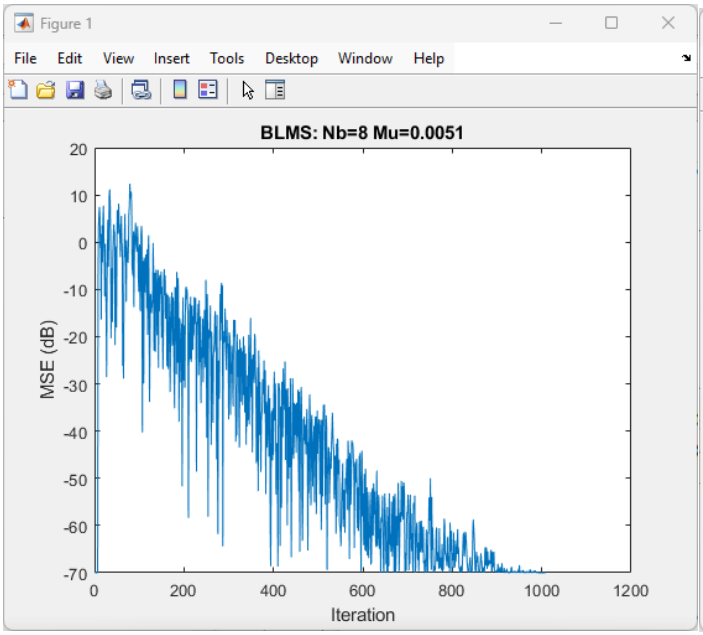
```
BLMS time: 0.0015586 seconds for Nb=8

ans =

    1.0000
    1.0000
    1.0000
    1.0000
    0.0000
    0.0000
   -0.0000
   -0.0000
```

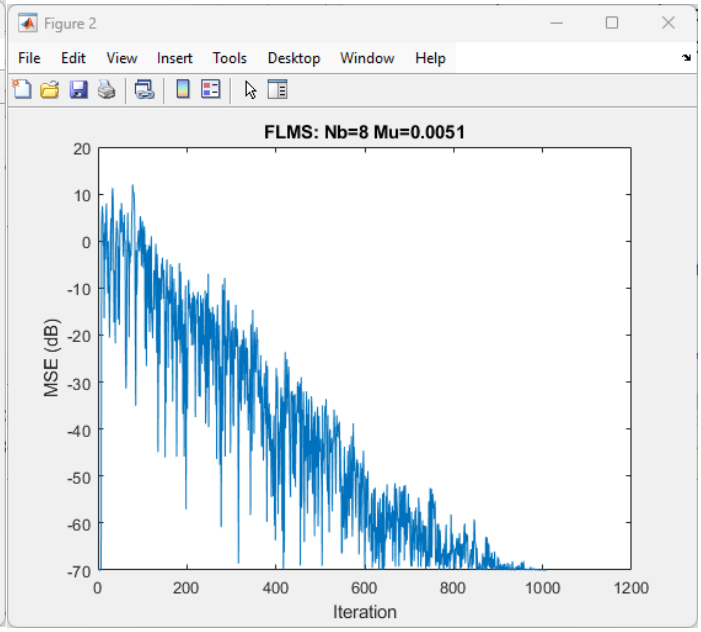
Bhat from BLMS

```
FLMS time: 0.0013711 seconds for Nb=8

ans =

    1.0000
    1.0000
    1.0000
    1.0000
    0.0000
    0.0000
   -0.0000
    0.0000
```

Bhat from FLMS

Note this is for a unknown filter of b = [1 1 1 1] used for charactering the time

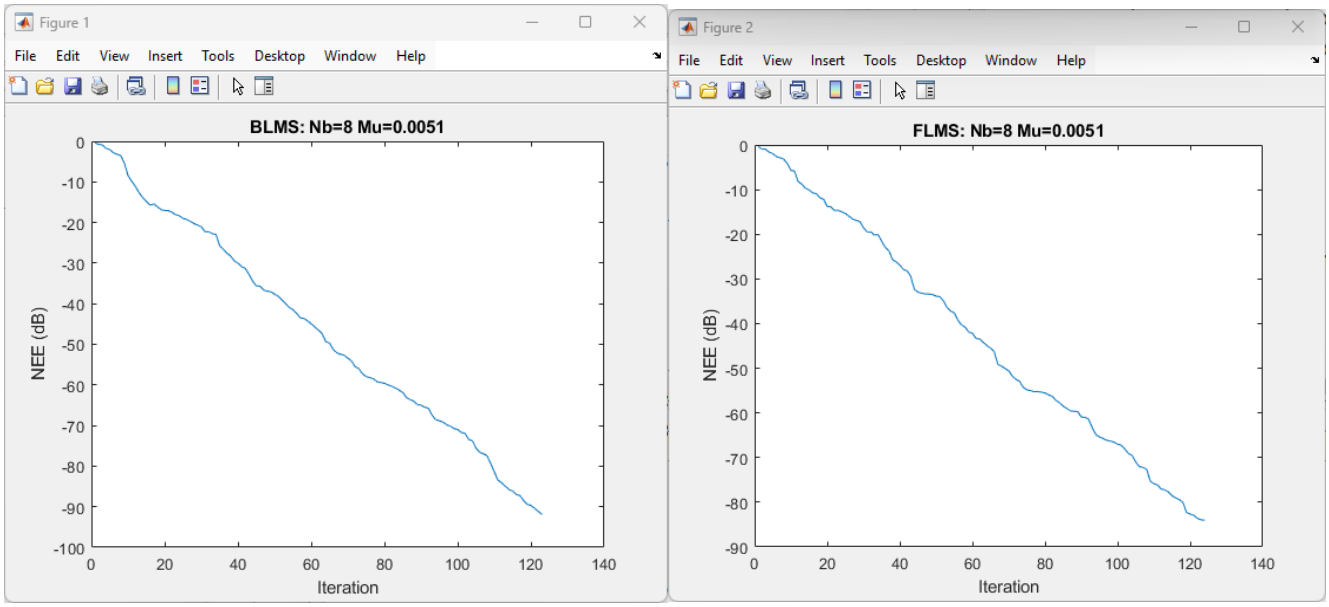




5) Develop MATLAB code for the Normalized Error Energy (NEE) (see below) in dB which we will use to evaluate in a system ID. (Hint: Below are continuous frequency domain functions – you need to sample them in frequency and employ radix-2 length vectors to enable the use of FFT for fast computation of the NEE).

$$NEE = \frac{\int_{-\pi}^{\pi} |\, H(e^{j\Omega}) - \hat{H}(e^{j\Omega}) \,|^2 \, d\Omega}{\int_{-\pi}^{\pi} |\, H(e^{j\Omega}) \,|^2 \, d\Omega}, \quad NEE_{dB} = 10\log(NEE)$$

Here is the output of my plots after I got the NEE working. This is assuming the same unknown filter mentioned in the previous problems. The slopes here are about the same and it much easier to see what the algorithm is doing when comparing this to MSE plot up above.



6) Perform the following simulations:

a. FIR system ID with zero mean white noise of variance 1. Evaluate BLMS with 8 and 16 coefficients. No overlap (S=N).

Use N=L+1 (i.e., the size of the block equal to the number of filter coefficients). Run two simulations and give NEE (dB) vs iteration curves (for N=8 and N=16).

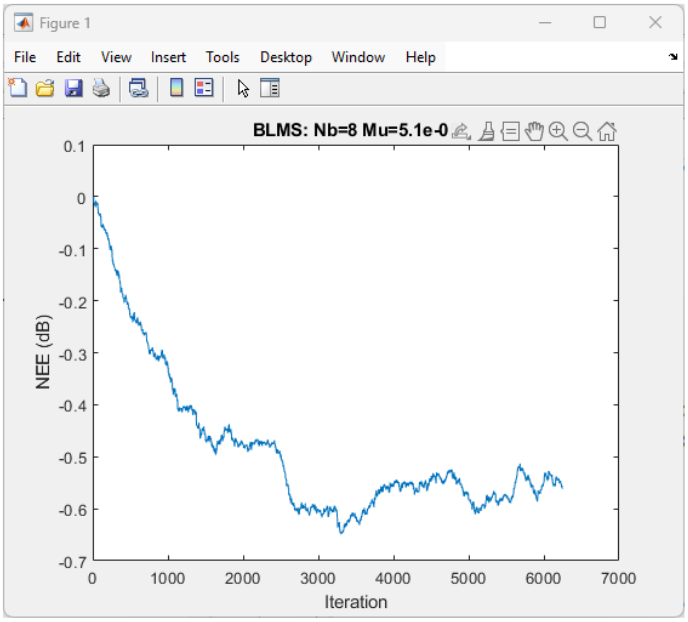
Use as "unknown system" an *H(z)* with impulse response

h=[0.776 2.397 1.966 1.859 1.171 0.123 0.525 -0.994 0.588 -1.177 -0.102 1.471 3.161 4.329 2.023 2.666]

For 50,000 samples, since the unkown system is a filter of 16 and we are trying to model it with a filter of 8 coefficients it doesnt converge very well. μ is also very small to get best results

```
BLMS time: 1.8884 seconds for Nb=8

ans =

    0.7229
    2.3383
    1.9568
    1.8299
    1.1474
    0.1155
    0.4632
   -0.9556
```

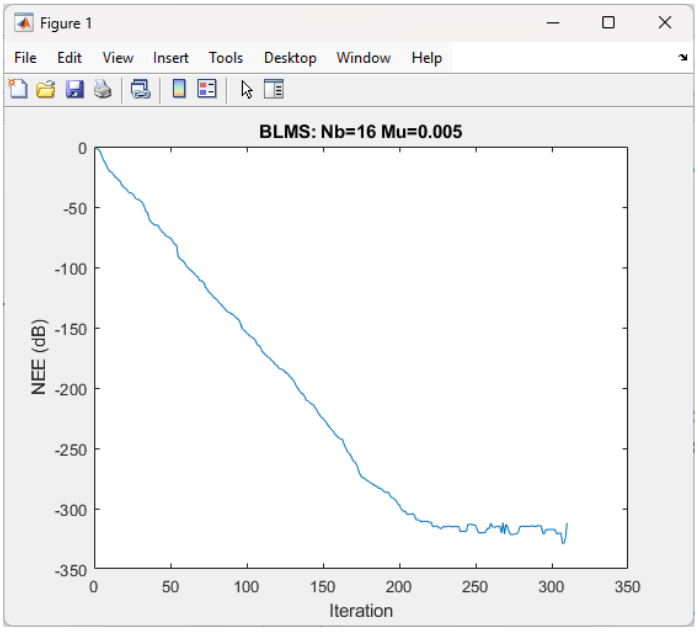
bhat from BLMS



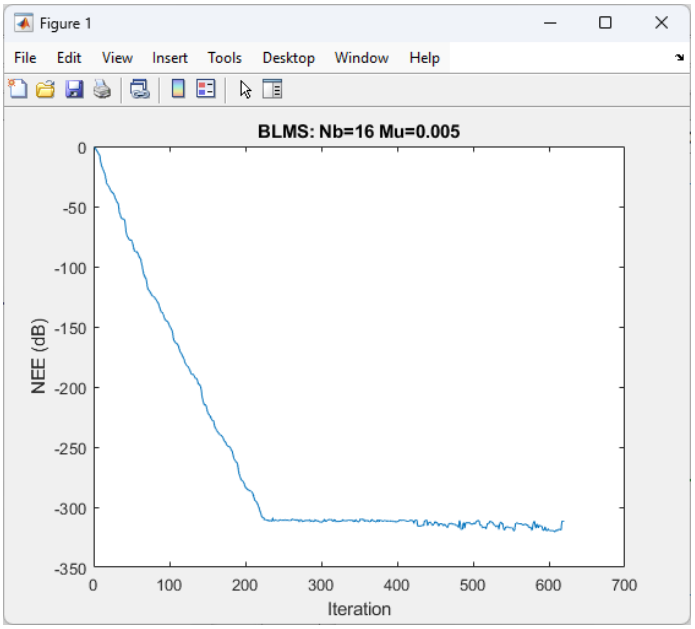
For 5,000 samples, and 16 coefficients we are able to represent the "unkown" system perfectly

```
BLMS time: 0.11131 seconds for Nb=16

ans =

    0.7760
    2.3970
    1.9660
    1.8590
    1.1710
    0.1230
    0.5250
   -0.9940
    0.5880
   -1.1770
   -0.1020
    1.4710
    3.1610
    4.3290
    2.0230
    2.6660
```

What from BLMS



b. Also evaluate in a graph NEE vs iteration for the case of half overlap S=N/2 (comment on this, specifically convergence and overall complexity in CPU time). Comment/compare on complexity per sample processed during adaptation. Do this only for N=16. Any difference in convergence speed?



At first glance it seems like its converging faster and sharper but it seems to be converging at the same rate, but its "earlier" in the data set. You can see it bottoms out at about the same iteration number (about 200) but since we are shifting by half the block size we are doubling our iterations. Which means we are converging earlier in the data set, at the expense of more computations/complexity.

c. Repeat system ID (same "unknown" system) with zero-mean colored noise input with power spectral density.

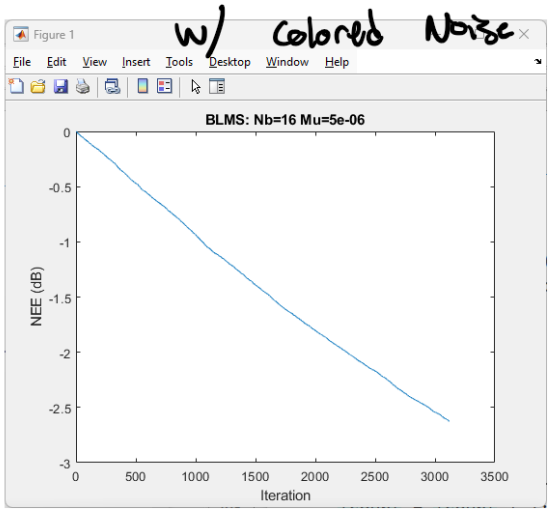
$$R_{xx}(\Omega) = \frac{1}{|1 + 0.64e^{-j2\Omega}|^2}$$

as you can see, in the results we are not converging as well as before because we now have eigenvalue disparity. This is with 25,000 samples and much smaller μ

```
BLMS time: 0.66517 seconds for Nb=16

ans =

    0.3152
    0.7349
    0.3104
    0.5403
    0.1249
   -0.0206
    0.2185
   -0.5261
    0.5536
   -0.8084
    0.4772
    0.2805
    1.0826
    1.0167
    0.1617
    0.9411
```

what from BLMS

w/ colored Noise

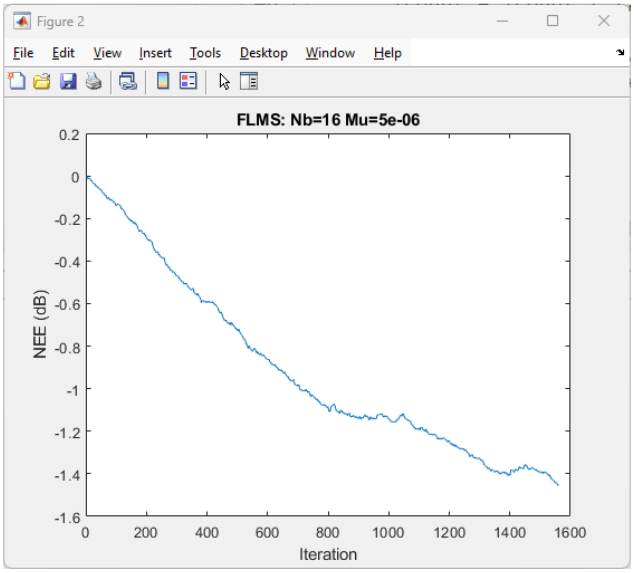


7) Repeat the same simulation only for color noise for the frequency domain BLMS with N=16 and show that results are the same (NEE vs iteration) for every iteration (hint: need careful adjustment of μ to show equivalence).

```
FLMS time: 0.32407 seconds for Nb=16

ans =

    0.1782
    0.4123
    0.1754
    0.3039
    0.0713
   -0.0123
    0.1223
   -0.2942
    0.3104
   -0.4538
    0.2681
    0.1588
    0.6096
    0.5722
    0.0911
    0.5297
```

what from FLMS

w/ colored noise



As you can see the FLMS doesn't converge very well either and does about as good as the BLMS. Just a little worse.

What I learned:

I struggled very much with getting the BLMS algorithm to work. I spent a lot of time here trying to get this algorithm to work. I learned after reading the Clark and Mitra paper, they talked about how the LMS is a special case for the BLMS and since we had working code for the LMS algorithm I compared the 2 algorithms. To do this I had the BLMS with a block size of 1 and slowly stepped through the code in both algorithms to see where the differences were. I quickly noticed that my error and desired signal were different and realized that my issue was with indexing the output. I was starting at the Nbth input data but was grabbing the block from the desired signal at time equals zero to start. But that was after a whole weekend of studying the algorithm. I had a lot of fun getting this to work but it was tougher than I thought it would be. The FLMS, I followed the lecture notes and got it working almost right away. I think some of that was because I struggled so much with the BLMS that I knew how to be much more careful when writing out my code and I also didn't make the same mistakes that I did with the BLMS. I also started this test out using python because I am more comfortable in python syntax. But ended up switching over to MATLAB after realizing the filter function I was using in python was not working as intended. After switching over to MATLAB, my eyes were opened at how easy the debugging capability is and how well MATLAB is structured for matrix and vector math. I could see all my vector/matrix sizes and easily see the data updating as I stepped through my code. All in all, I had a blast with this test and I learned a ton about how to read the notation and how to apply it in a simulation. I learned how to write my code in a way that would make it easy to debug and rapidly iterate. After applying the BLMS I could easily see how it connects to the FLMS and FDAF. I understood better how the convolution worked in the BLMS and the advantages of the Frequency domain application. I also want to give a shout out to the Adaptive Filter Theory textbook by Simon Haykin. Reading through his description of the BLMS algorithm helped me understand quite a bit better how the algorithm worked and how to expect to code it.

Appendix for Code

The file labeled Midterm top is the main function. This then calls the BLMS algorithm and the FLMS algorithm at which point call the nee_num and nee_denom functions which is short for numerator and denominator respectively. The parameters for adjustment are near the top of the midterm_top file. A quick rundown of those:

- Nb: Block Size
- mu: the step size
- N_points: sets the number of samples for the input and desired signals.
- eflag: sets how to calculate and plot the error. 1 for NEE and 0 for MSE
- s: Shifting parameter for the how much to shift the block after each iteration
- d_flag: sets the desired signal to by created by white noise or colored noise.

This file structure makes it easy to ensure same block size, mu, shifting parameter, input signal, desired signals, etc are arriving to both the BLMS and the FLMS algorithms to make comparison easier.

Function Tree

Midterm_top

|

|→BLMS

|→ nee_num

|→ nee_denom

|-->FLMS

|→ nee_num

|→ nee_denom

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% Midterm top level

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

clear;

close all;


%% Parameters


Nb      = 16;    %block size

mu      = .0051; %step size

N_points = 1008;  %number of points

eflag    = 0;     %1 for NEE and 0 for MSE

s       = Nb;    %shifting parameter

d_flag   = 1;     %1 for white noise input, 0 for colored noise input




%% Unknown systems

%b = [1 1 1 1];

b=[0.776  2.397  1.966  1.859  1.171  0.123  0.525  -0.994  0.588  -1.177  -0.102  1.471 3.161  4.329  2.023
2.666];

cb = [1];

ca = [1 0.64];

try

   bpad = zeros(1, Nb -length(b));

catch

   disp("b is already correct size")

end

b = [b, bpad];

a = [1];
```

```
%% Create the desired signal
rng(3) % seed the random number generator to produce the same numbers
x = randn(N_points, 1); %white noise


d1 = filter(b, a, x); %nominal output

cd = filter(cb, ca, x); %colored noise
d2 = filter(b, a, cd); %colored output

if d_flag == 1
   d = d1; %desired signal for white noise input
else %d_flag == 0
   d = d2;
end

%% BLMS
blms(Nb, mu, x, d, N_points, b, eflag, s)


%% Fast LMS
flms(Nb, mu, x, d, N_points, b, eflag, s)
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% BLMS Function

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function bhat = blms(Nb, mu, x, d, N_points, b, eflag, s)


%% Variable creation


%N_points = 6000;

%eflag = 1;

%algorithm controls

%mu = .001;

%Nb = 16;

L  = Nb-1;%Nb-1;

coef = L+1;

%s = Nb/2; % shifting parameter size




%% Algorithm

xb   = zeros(Nb, coef);

xbb   = zeros(Nb, coef);

yb   = zeros(coef, 1);

eb   = zeros(coef, 1);

e    = zeros(N_points,1);

E    = zeros(floor((N_points-Nb)/s),1);

bhat = zeros(coef,1);



tic

icount = 0;
```

```
for i = 1: s : N_points- (3*coef) % i is the block index
   icount = icount + 1;
   %disp(icount)
   %create the input block
   for j = 1:Nb
      xv = (x(i+j-1 : i+j-1+L)); %the ith block and the jth sample to the jth+L sample
      xbb(j, :) = xv';
      xb(j, :) = flip(xv');
   end
   xbt = xb';
   %calculate output
   yb = xb * bhat; % Nbx(L+1) * (L+1)x1

   %calculate the error
   db = d(coef + i-1 : Nb + coef + i-2); %get the current block %this is where I had all my issues, since I am starting the convolution at the Nbth sample I need to get the Nbth desired sample
   eb = db - yb;
   e(coef + i-1 : Nb + coef + i-2) = eb; %save the error

   %calc gradient
   grad =  xb' * eb;

   %update weights
   bhat = bhat + (2) * mu * grad;

   if eflag == 1
      B = fft(b);
      Bhat = fft(bhat);
      num = nee_num(B, Bhat, Nb); %freq of B - freq of Bhat
      denom = nee_denom(B, Nb);
```

```
        E(icount) = trapz(num)/trapz(denom);
    end


end
elapsed_time = toc;
disp(['BLMS time: ', num2str(elapsed_time), ' seconds for Nb=', num2str(Nb)]);


figure;
title_str = ['BLMS: Nb=', num2str(Nb), ' Mu=', num2str(mu)];
if eflag == 1
    plote = 10*log10(E);
    plot(plote(:));
    title(title_str);
    ylabel('NEE (dB)')
    xlabel('Iteration')
else
    plote = 10*log10(e.*e +0.0000001);
    plot(plote(:));
    title(title_str);
    ylabel('MSE (dB)')
    xlabel('Iteration')
end
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% BLMS Function
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function bhat = flms(Nb, mu, x, d, N_points, b, eflag, s)

%Nb = block size
%mu = step size
%x  = input signal
%d  = desired signal
%N_points = number of points in the signal



%% Variable creation



%eflag = 0;
%algorithm controls
%mu = .001;
%Nb = 8;
%s = Nb; % shifting parameter size



%% Algorithm

Ek   = zeros(2*Nb, 1);
e    = zeros(N_points,1);
```

```
E    = zeros(floor((N_points-Nb)/s),1);


Bhat = zeros(2*Nb,1);
zpad = zeros(1, Nb);
b = [b, zpad];


tic
icount = 0;
for i = 1: s : N_points- 2*Nb % i is the block index
   icount = icount + 1;


   %get the input block
   xv = x(i : (i-1)+Nb*2); %the ith block and the jth sample to the jth+L sample
   db = d(Nb + (i-1) : Nb + (i-1) + (2*Nb-1)); %get the current block, which to start is the Nbth d sample


   %do the ffts
   Xk = fft(xv);
   Dk = fft(db);


   %calculate output
   Xkd = diag(Xk);
   Yk = Xkd * Bhat; % Nbx(L+1) * (L+1)x1


   %calculate the error
   yb = ifft(Yk);
   yn = yb(Nb+1: Nb*2); %grabbing the last N points
   dn = d(Nb+1 + (i-1) : Nb+1 + (i-1) + (Nb-1)); %get the Nbth output from desired signal
   en = dn - yn; %generating the error in time domain
   e(Nb + (i-1) : Nb + (i-1) + (Nb-1)) = en;
   eb = [zpad'; en]; %pad the error signal before ffting
```

```
    Ek = fft(eb);



    %calc gradient + update weights
    Bhat = Bhat + 2*mu*Xkd'*Ek ;



    if eflag == 1
        B = fft(b);
        num = nee_num(B, Bhat, Nb); %freq of B - freq of Bhat
        denom = nee_denom(B, Nb);
        E(icount) = trapz(num)/trapz(denom);
    end

end
b = ifft(Bhat);
bhat = b(1:Nb);
elapsed_time = toc;
disp(['FLMS time: ', num2str(elapsed_time), ' seconds for Nb=', num2str(Nb)]);

figure;
title_str = ['FLMS: Nb=', num2str(Nb), ' Mu=', num2str(mu)];
if eflag == 1
    plote = 10*log10(E);
    plot(plote(:));
    title(title_str);
    ylabel('NEE (dB)');
    xlabel('Iteration');
else
    plote = 10*log10(e.*e +0.0000001);
```

```
    plot(plote(:));
    title(title_str);
    ylabel('MSE (dB)');
    xlabel('Iteration');
end
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% NEE Functions for integration

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function num = nee_num(B, Bhat, N)

    freqB    = freqz(B, 1, N);

    freqBhat = freqz(Bhat, 1, N);

    num = abs(freqB - freqBhat).^2;

end

function denom = nee_denom(B, N)

    freqB    = freqz(B, 1, N);

    denom = abs(freqB).^2;

end
```