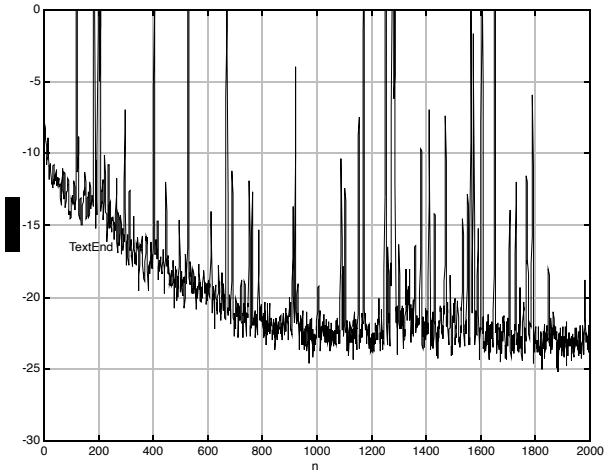
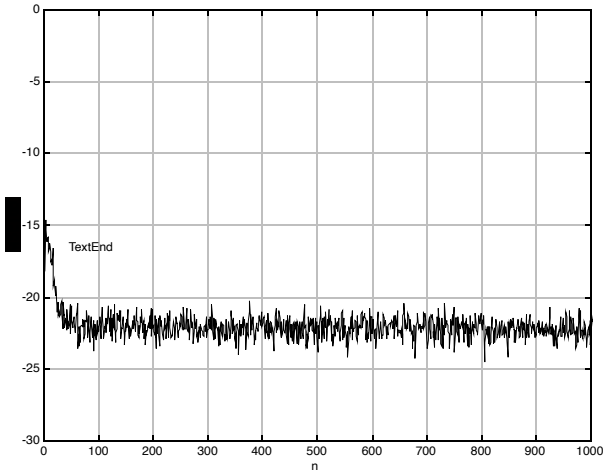
Solution #5: NLMS, APA, and Transform Domain LMS Adaptive Filters

1)

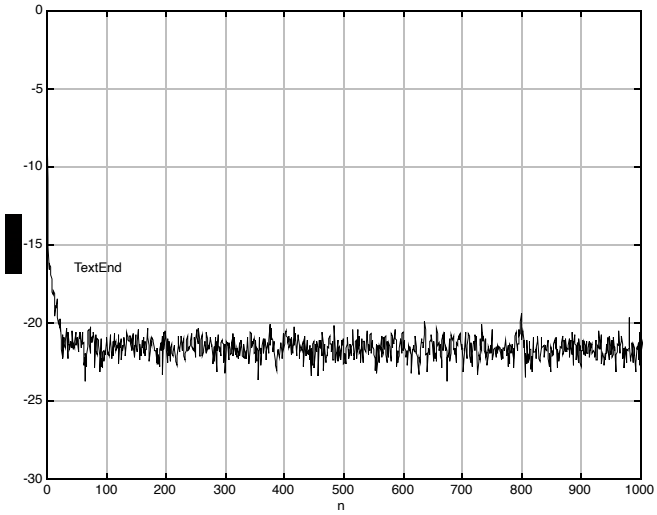
(a)



(b)



(c)



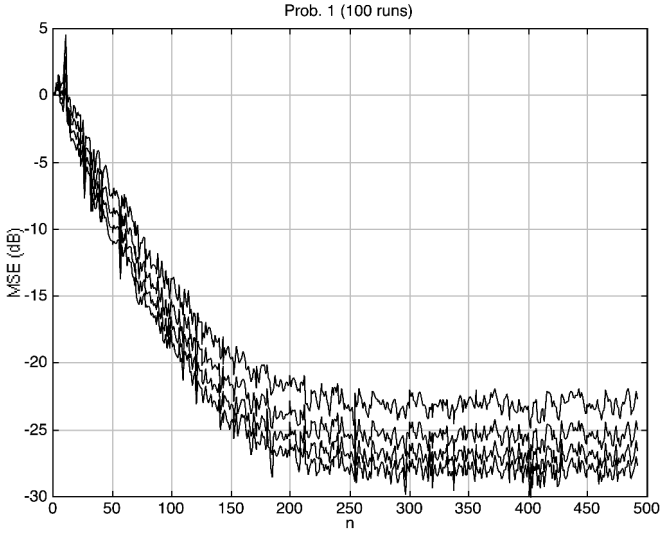
LMS (100 runs)	NLMS (100 runs)	GNLMS (100 runs)
μ = 0.13	μ~ = 0.6	μ = 0.4
J_{ex} = 0.0051	J_{ex} = 0.0062	J_{ex} = 0.0071
M = 1.3364	M = 1.6333	M = 1.8538

(d) Comment.

For a fixed level of misadjustment, we see that NLMS and GNLMS converge much faster than LMS. For this experiment, the NLMS converges in about 40 samples while the GNLMS converges in about 10 samples. The increase in convergence speed comes with an increase in computational complexity. We also note the superior convergence properties of NLMS compared to LMS as per Rupp.

---

2) (Adaptive Equalization) Recreate the results from Experiment 2 p. 473 to verify correct operation of your DCT-LMS. Plot MSEs in dB.

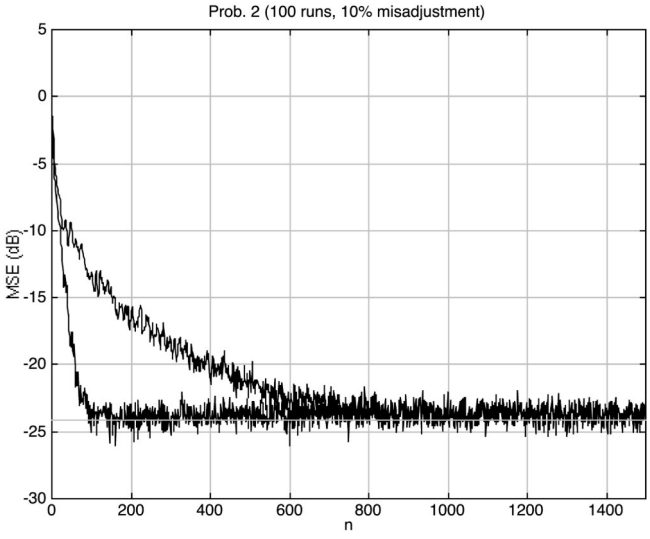


The above simulations were conducted with the following parameters in dct_lms: $\alpha = 1/(5M)$, $\beta = 1.0$, $\gamma = 1.0$, $\delta = 0$. The textbook plots required 400 simulations per *W* as well as the removing of the 66 worst-case error signals from the ensemble (not really fair).

---

3) (NLMS/SOAF comparisons) Redo Experiment 1 Case 4 in Section 8.4 of your text using the NLMS and SOAF algorithms to update the coefficient vector. Note that the final values for the NLMS adaptive filter and SOAF adaptive filter will differ by **Q**.

(a) On a single figure plot the MSEs for both algorithms in dB (for equal misadjustments). Comment on the convergence rates for each case.



We see that for equal misadjustments, the NLMS ($\mu = 0.046$) converges in about 800 samples versus SOAF (alpha = 1/(5.85*M)) which converges in about 80 samples. The SOAF is much faster.

(b) List the filter coefficients for the Wiener solution, NLMS, and SOAF with and without the inverse transform by **Q**.

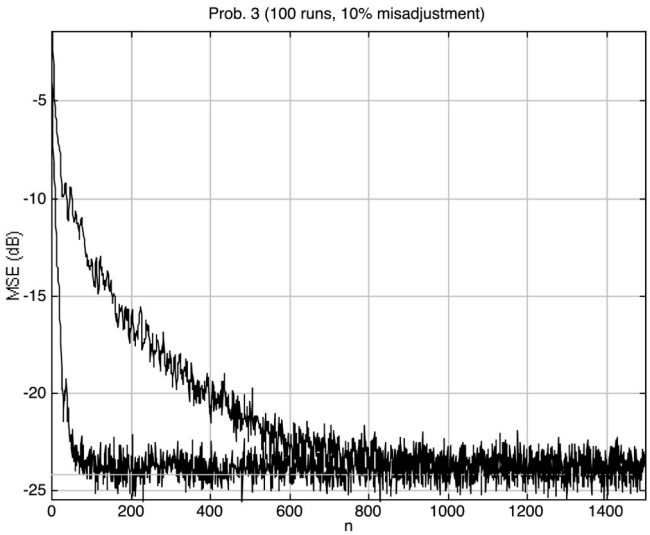
Wiener solution	NLMS	SOAF w/o **Q**	SOAF w/**Q**

1.9114 -0.9500	1.9312 -0.9429	1.8842 0.6864	1.8177 -0.8470

Q =
0.7071 0.7071
-0.7071 0.7071

---

4) (NLMS/DCT-LMS comparisons) Same as 2) but with the more practical DCT-LMS instead of SOAF. [$\alpha = 1/(6.8M)$, $\beta = \gamma = 1.0$, $\delta = 0.0$]



(b) List the filter coefficients for the Wiener solution, NLMS, and DCT-LMS with and without the inverse transform by **Q**.

Wiener solution	NLMS	DCT-LMS w/o **Q'**	DCT-LMS w/ **Q'**
1.9114 -0.9500	1.9312 -0.9429	0.6693 -1.8263	1.7647 -0.8181

Since the DCT-LMS approximates the true KLT (as in SOAF) we expect a difference in the transformed filter coefficients.

```
% EE594 - Fall 2002 - Homework #5

%---
% 1
%---
a = [1;-1.9114;0.95];  % Section 8.4, Experiment 1, Case 4
sigma_v2 = 0.00382179;
Jmin = sigma_v2;

M = 2;
w_init = zeros(M,1);
L = 2020;
ts = 20;
acc_sqrd_e = zeros(L-ts,1);
num_runs = 100;
randn('seed',0);
```

```
for runs = 1:num_runs
   runs
   u = AR_synthesizer(a,L,sigma_v2);
   [e,w] = lms(w_init,u(ts:L-1),u(ts+1:L),0.13,0);  % 0.1 -> Jex = 0.0083
   %[e,w] = nlms(w_init,u(ts:L-1),u(ts+1:L),0.6);  % Jex = 0.0084
   %[e,w] = gnlms(w_init,u(ts:L-1),u(ts+1:L),0.4,2);  % mu = 0.2 Jex = 0.0066
   acc_sqrd_e = acc_sqrd_e + e.^2;
end;

MSE = acc_sqrd_e / num_runs;
MSE_plot(MSE,0,L-ts-1,1,1);
Jex = mean(MSE(L-ts-99:L-ts))
M = Jex / Jmin
%title('Prob. 3, LMS, mu = 0.1');
%title('Prob. 3, NLMS, mu = 1.0');
%title('Prob. 3, GNLMS, mu = 1.0');

hold on
plot([0 L-ts-1],[Jmin Jmin],':')
axis([0 L-ts-1 0 0.1]); % for normal unit plots
hold off

%---
% 2
%---
% p. 415 Section 9.7 Adaptive Equalizer

% Channel Model
W = 3.5;  % 2.9 3.1 3.3 3.5
N = 3;  % length of channel impulse response
nn = [1:N]';
h = [0;(1+cos(2*pi/W*(nn-2)))/2];

% Signal information
L = 500;  % signal lengths
rand('seed',0);
randn('seed',0);
sigma_v2 = 0.001;  % variance of additive channel noise

% Adaptive filter information
M = 11;
w_init = zeros(M,1);
delay = 7;
num_runs = 100;

% DCT-LMS
alpha = 1/(5*M);
sum_e2 = zeros(L-delay,1);
for run = 1:num_runs
   run
   x = sign(rand(L,1)-0.5*ones(L,1));  % +/- 1 data
   y = filter(h,1,x);  % output of channel
   v = sqrt(sigma_v2)*randn(L,1);  % channel noise
   u = y + v;  % input to adaptive filter
   u = u(delay+1:L);
   d = x(1:L-delay);  % d is x delayed by delta
   [e,w] = dct_lms(w_init,u,d,alpha);
```

```
   sum_e2 = sum_e2 + e.^2;
end;
MSE = sum_e2 / num_runs;
%hold on
MSE_plot(MSE);
%plot([0 L-ts-1],10*log10([Jmin Jmin]),':')
%hold off
%axis([0 L-ts-1 -30 5])
title(['Prob. 1 (',sprintf('%d',num_runs),' runs)'])


%---
% 3
%---
a = [1 -1.9114 0.95]';
sigma_v = (1-a(3))/(1+a(3))*((1+a(3))^2-a(2)^2);  % p. 351
sigma_u = (1+a(3))/(1-a(3))*(sigma_v/((1+a(3))^2-a(2)^2));
r = [sigma_u;-a(2)/(1+a(3))*sigma_u;(-a(3) + (a(2)^2)/(1+a(3)))*sigma_u];
R = toeplitz(r(1:2));
p = [r(2) r(3)]';
w_opt = pinv(R)*p;
Jmin = sigma_v;

M = 2;
w_init = zeros(M,1);
num_runs = 100;
L = 1520;
ts = 20;

% NLMS
mu = 0.046;  % 10% misadjustment
sum_e2 = zeros(L-ts,1);
randn('seed',0);
for run = 1:num_runs
   u = AR_synthesizer(a,L,sigma_v);
   [e,w] = nlms(w_init,u(ts:L-1),u(ts+1:L),mu);
   sum_e2 = sum_e2 + e.^2;
end;
MSE = sum_e2 / num_runs;
Jex = mean(MSE(L-ts-99:L-ts)) - Jmin
Misadjustment = Jex / Jmin
MSE_plot(MSE);
disp('Hit any key to go on...');
pause;

% SOAF
alpha = 1/(5.85*M);  % 10% misadjustment
sum_e2 = zeros(L-ts,1);
randn('seed',0);
[Q,LAMBDA] = eig(R);
for run = 1:num_runs
   u = AR_synthesizer(a,L,sigma_v);
   [e,w] = soaf(w_init,u(ts:L-1),u(ts+1:L),alpha,Q,diag(LAMBDA));
   sum_e2 = sum_e2 + e.^2;
end;
MSE = sum_e2 / num_runs;
Jex = mean(MSE(L-ts-99:L-ts)) - Jmin
```

```
Misadjustment = Jex / Jmin
hold on
MSE_plot(MSE);
plot([0 L-ts-1],10*log10([Jmin Jmin]),':')
hold off
axis([0 L-ts-1 -30 5])
title('Prob. 2 (100 runs, 10% misadjustment)')

%---
% 4
%---
a = [1 -1.9114 0.95]';
sigma_v = (1-a(3))/(1+a(3))*((1+a(3))^2-a(2)^2);  % p. 351
sigma_u = (1+a(3))/(1-a(3))*(sigma_v/((1+a(3))^2-a(2)^2));
r = [sigma_u;-a(2)/(1+a(3))*sigma_u;(-a(3) + (a(2)^2)/(1+a(3)))*sigma_u];
R = toeplitz(r(1:2));
p = [r(2) r(3)]';
w_opt = pinv(R)*p;
Jmin = sigma_v;

M = 2;
w_init = zeros(M,1);
num_runs = 100;
L = 1500;
ts = 20;

% DCT-LMS
alpha = 1/(6.8*M);  % 10% misadjustment
sum_e2 = zeros(L-ts,1);
randn('seed',0);
for run = 1:num_runs
   run
   u = AR_synthesizer(a,L,sigma_v);
   [e,w] = dct_lms(w_init,u(ts:L-1),u(ts+1:L),alpha);
   sum_e2 = sum_e2 + e.^2;
end;
MSE = sum_e2 / num_runs;
Jex = mean(MSE(L-ts-99:L-ts)) - Jmin
Misadjustment = Jex / Jmin
MSE_plot(MSE);
hold on
plot([0 L-ts-1],10*log10([Jmin Jmin]),':')
hold off
axis([0 L-ts-1 -30 5])
title('Prob. 3 (100 runs, 10% misadjustment)')
```