EECS 844 – Fall 2017

Exam 3 Cover page*

Each student is expected to complete the exam individually using only course notes, the book, and technical literature, and without aid from outside sources.

Aside from the most general conversation of the exam material, I assert that I have neither provided help nor accepted help from another student in completing this exam. As such, the work herein is mine and mine alone.

_____ _____

Signature Date

MY SOLUTIONS

_____ _____

Name (printed) Student ID #

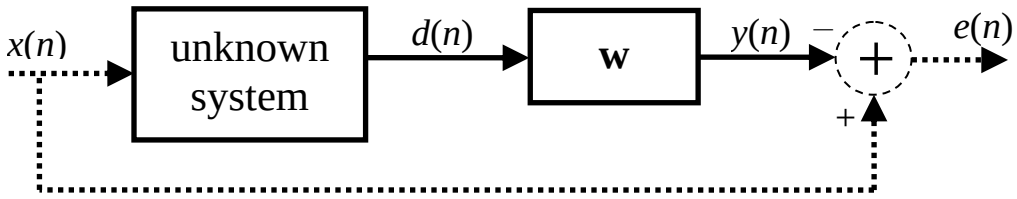
*** Attach as cover page to completed exam.**

1. In P1.mat the output of a hypothesized unknown linear system is the observed signal $d(n)$, to which we shall apply a whitening filter via linear prediction (the solid lines in the figure). In contrast, the dashed lines represent the inverse filtering formulation that we could perform if the hypothesized "driving signal" $x(n)$ were actually known (in practice it is not available).

a) Determine **w** as the linear prediction error (LPE) filter using only $d(n)$ when the linear prediction component is $M = 19$ (so the LPE filter is length 20).

b) Alternatively, determine **w** as the inverse system identification Wiener filter using both $d(n)$ and $x(n)$ for $M = 20$.

c) For each filter, plot the time-domain magnitude/phase and frequency response (use 'freqz' with 'whole'). Comment on how they are related/different.



Solution:

Figure 1.1 contains the magnitude (in dB) and phase of the LPE filter $\mathbf{w}_{\text{LPE}} = [1 \;\; -\mathbf{w}_{\text{LP}}^T]^T$, where $\mathbf{w}_{\text{LP}}$ is the linear prediction filter, and the inverse system identification version of the Wiener filter. We observe that for the large coefficient values (the first few) the magnitude and phase of these two filters are nearly identical. As a result, the frequency responses shown in Fig. 1.2 are also nearly identical.

While it may not be obvious from these solutions, the Wiener filter implementation is more accurate because it uses both the input $x(n)$ and the output $d(n)$ of the unknown system. However, for many applications only $d(n)$ is available (e.g. speech).

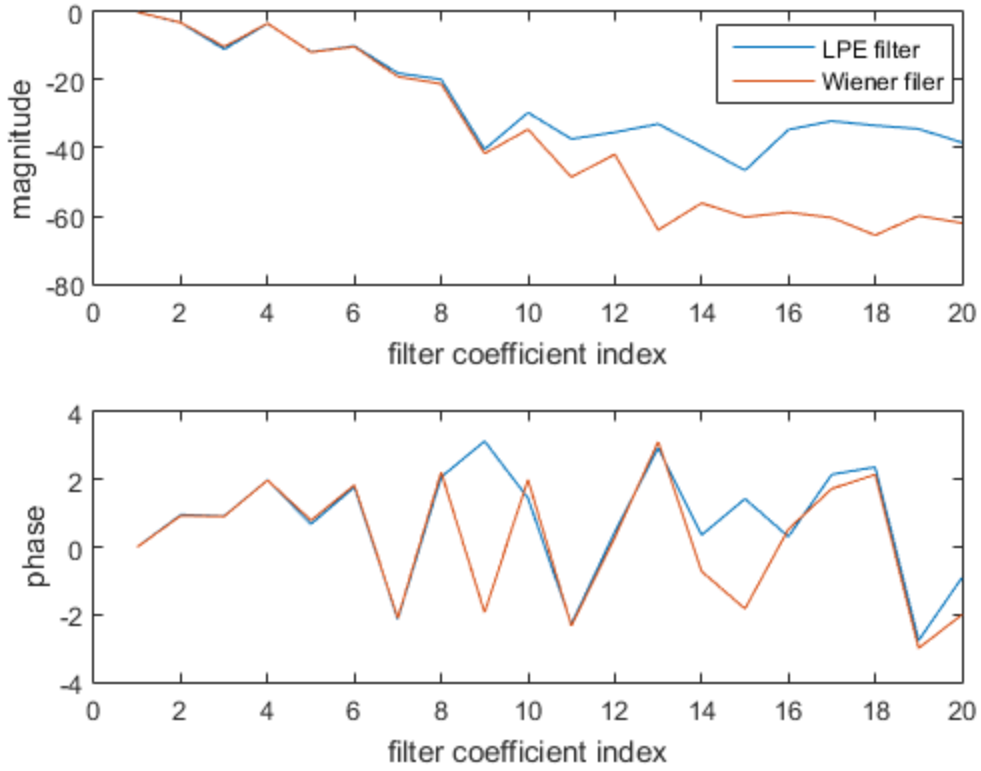


Figure 1.1. Magnitude and phase of the LPE filter and inverse system ID Wiener filter

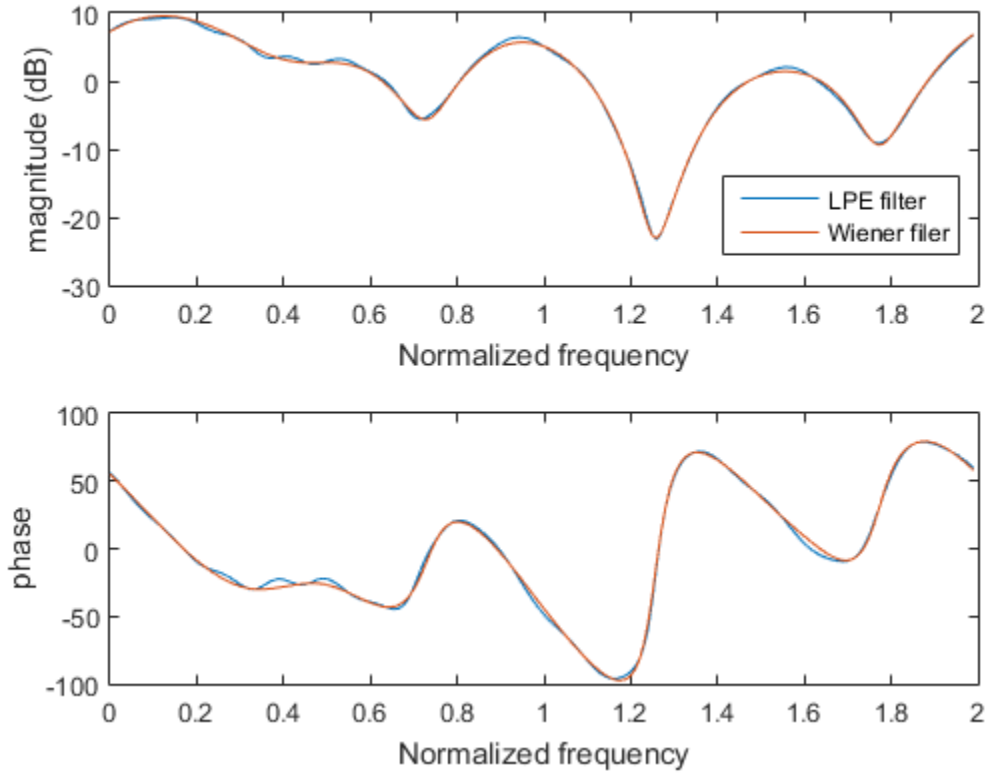


Figure 1.2. Frequency response of the LPE filter and the inverse system ID Wiener filter

Matlab Code for Problem 1

```
clear all
load P1
N = length(x);

M = 20;
MLP = M-1;

%% linear prediction
DLP = zeros(MLP,N-MLP+1);
for jm = 1:MLP
    DLP(MLP-jm+1,:) = d(jm:jm+N-MLP).';
end;
DLP2 = DLP(:,1:N-MLP);
RLP_d = (1./(N-(MLP+1)+1)).*DLP*DLP';
DLP_d = repmat(d(MLP+1:N)',MLP,1);
pLP_dd = sum(DLP2.*DLP_d,2)./(N-MLP);
LP_opt = [1; -inv(RLP_d)*pLP_dd];

% Wiener filter
D = zeros(M,N-M+1);
for jm = 1:M
    D(M-jm+1,:) = d(jm:jm+N-M).';
end;
R_d = (1./(N-M+1)).*D*D';
X_x = repmat(x(M:N)',M,1);
p_dx = sum(D.*X_x ,2)./(N-M+1);
WF_opt = [inv(R_d)*p_dx];

figure(1)
subplot(2,1,1)
plot(1:M,20*log10(abs(LP_opt)),1:M,20*log10(abs(WF_opt)));
ylabel('magnitude')
xlabel('filter coefficient index')
legend('LPE filter','Wiener filer')
subplot(2,1,2)
plot(1:M,angle(LP_opt),1:M,angle(WF_opt));
ylabel('phase')
xlabel('filter coefficient index')

[Hwf,Wwf] = freqz(WF_opt,1,10*M,'whole');
[Hlp,Wlp] = freqz(LP_opt,1,10*M,'whole');

figure(2)
subplot(2,1,1)
plot(Wlp./pi,20*log10(abs(Hlp)),Wwf./pi,20*log10(abs(Hwf)))
xlabel('Normalized frequency')
ylabel('magnitude (dB)')
legend('LPE filter','Wiener filer')
subplot(2,1,2)
plot(Wlp./pi,(180/pi).*angle(Hlp),Wwf./pi,(180/pi).*angle(Hwf))
xlabel('Normalized frequency')
ylabel('phase')
```

2. Repeat Problem 1 using the data set P2.mat. How do the linear prediction and Wiener filter solutions compare for this data set? Explain what you observe and demonstrate (using signal processing of the data) why your explanation is correct.

Solution:

In this case the linear predication error (LPE) filter does not realize the same solution as the Wiener filter (see Figures 2.1 and 2.2). The implication is that the unknown system does not impart all of the structure observed in $d(n)$ that is estimated via linear prediction. In other words, the LPE filter $\begin{bmatrix} 1 & -\mathbf{w}_{LP,d} \end{bmatrix}^T$ based on $d(n)$ is not the inverse model of only the unknown system. Instead, the LPE filter is an inverse model of the convolution between the unknown system and a linear estimate of the inherent structure of $x(n)$ since it is not white.

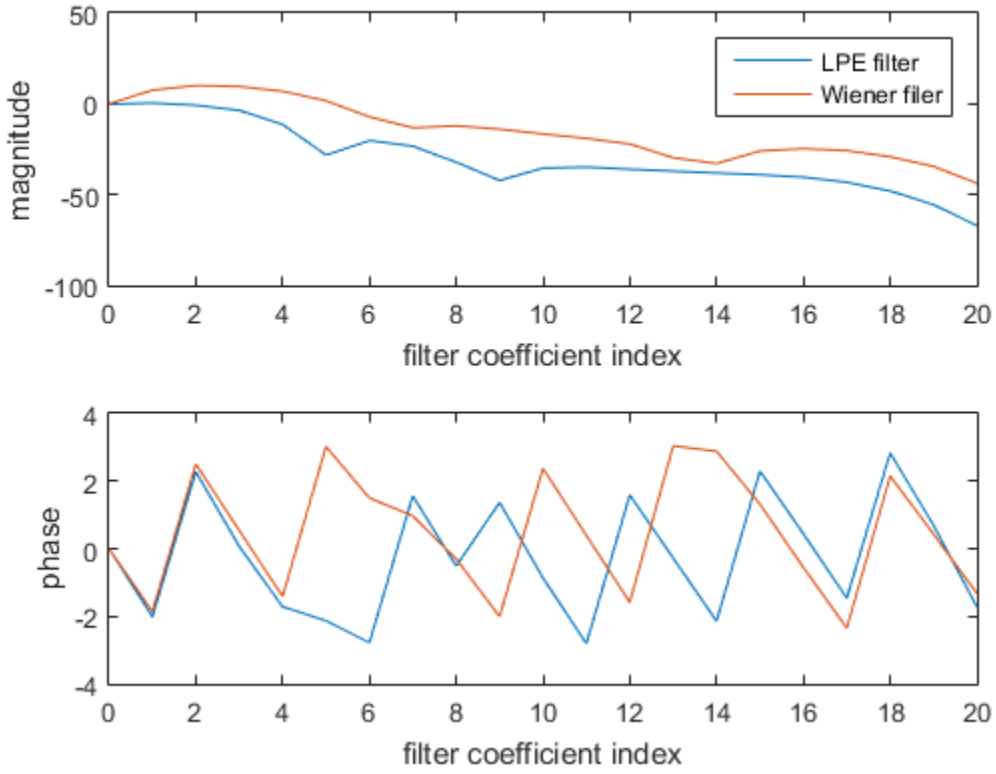


Figure 2.1. Comparison of LPE filter and Wiener filter

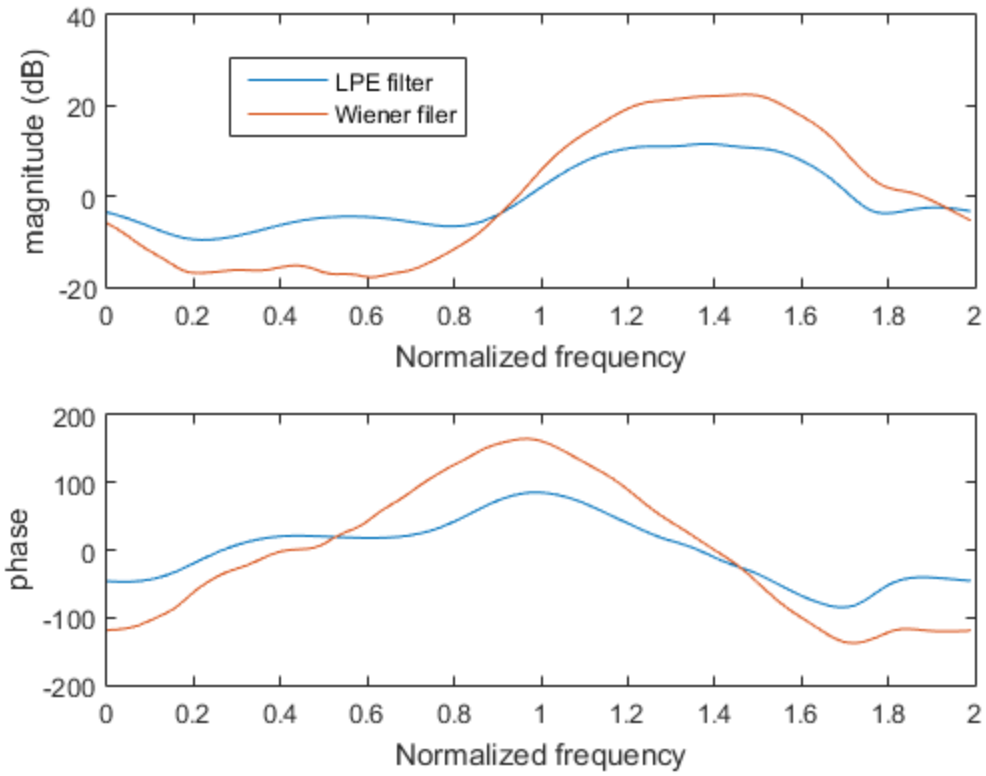


Figure 2.2. Frequency response comparison of LPE filter and Wiener filter

Now denote the Wiener filter estimate of the inverse model of the unknown system as $\mathbf{w}_{WF}$. If we use linear prediction to estimate the structure of the input signal $x(n)$ as the LPE filter $\begin{bmatrix} 1 & -\mathbf{w}_{LP,x} \end{bmatrix}^T$, then the convolution of the LPE filter for $x(n)$ and the above Wiener filter should approximate the LPE filter $\begin{bmatrix} 1 & -\mathbf{w}_{LP,d} \end{bmatrix}^T$ based on $d(n)$. The time domain and frequency domain comparisons in Figures 2.3 and 2.4, respectively, show this to be true.

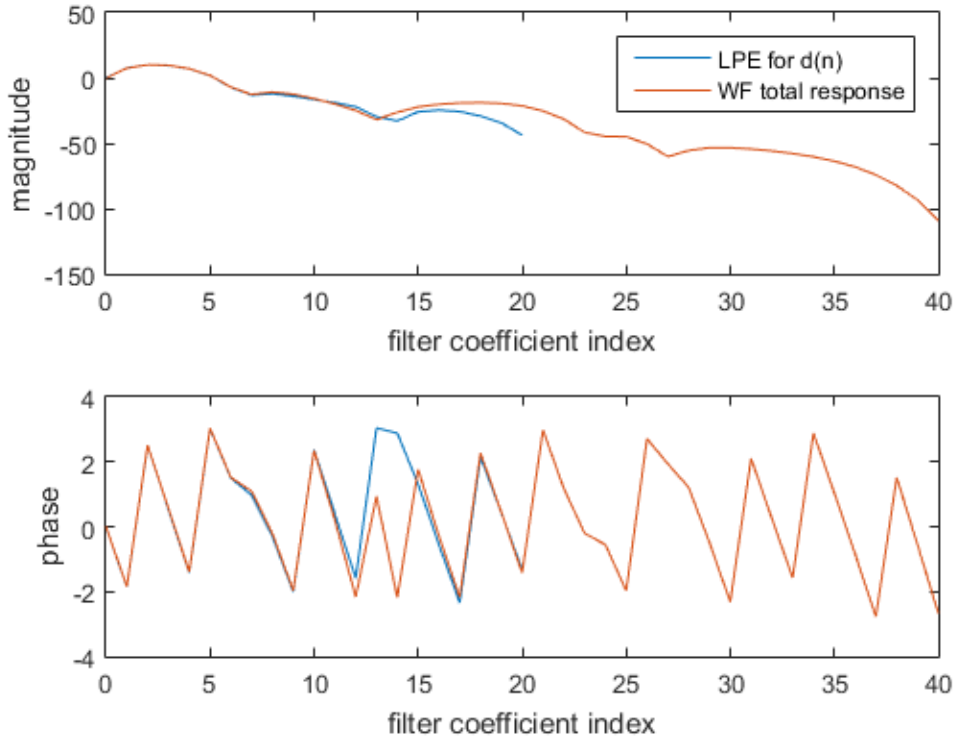


Figure 2.3. Comparison of LPE filter for $d(n)$ and the convolution between the Wiener filter with the LPE filter for $x(n)$ (the 'WF total response')

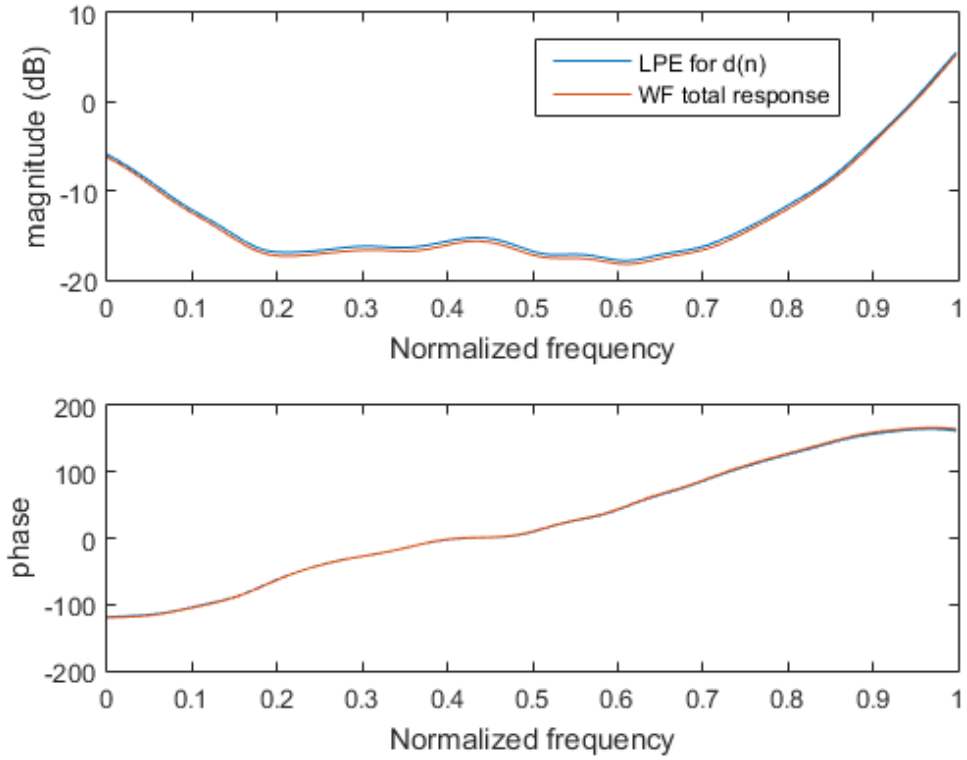


Figure 2.4. Frequency response comparison of LPE filter for $d(n)$ and the convolution of the Wiener filter with the LPE filter for $x(n)$ (the 'WF total response')

Matlab Code for Problem 2

```
clear all
load P2
N = length(x);

M = 20;

%% Wiener filter
D = zeros(M,N-(M+1)+1);
for jm = 1:M+1
    D(M+1-jm+1,:) = d(jm:jm+N-(M+1)).';
end;

R_d = (1./(N-(M+1)+1)).*D*D';
X_x = repmat(x(M+1:N)',M+1,1);
p_dx = sum(D.*X_x ,2)./(N-(M+1)+1);
wf_opt = [inv(R_d)*p_dx];

%% linear prediction
D = zeros(M,N-M+1);
for jm = 1:M
    D(M-jm+1,:) = d(jm:jm+N-M).';
end;
D2 = D(:,1:N-M);

R_d = (1./(N-(M+1)+1)).*D*D';
D_d = repmat(d(M+1:N)',M,1);
p_dd = sum(D2.*D_d,2)./(N-M);
lp_opt = [1; -inv(R_d)*p_dd];

figure(1)
subplot(2,1,1)
plot(0:M,20*log10(abs(wf_opt)),0:M,20*log10(abs(lp_opt)));
ylabel('magnitude')
xlabel('filter coefficient index')
legend('LPE filter','Wiener filter')
subplot(2,1,2)
plot(0:M,angle(wf_opt),0:M,angle(lp_opt));
ylabel('phase')
xlabel('filter coefficient index')

[Hwf,Wwf] = freqz(wf_opt,1,10*M,'whole');
[Hlp,Wlp] = freqz(lp_opt,1,10*M,'whole');

figure(2)
subplot(2,1,1)
plot(Wwf./pi,20*log10(abs(Hwf)),Wlp./pi,20*log10(abs(Hlp)))
xlabel('Normalized frequency')
ylabel('magnitude (dB)')
legend('LPE filter','Wiener filter')
subplot(2,1,2)
plot(Wwf./pi,(180/pi).*angle(Hwf),Wlp./pi,(180/pi).*angle(Hlp))
xlabel('Normalized frequency')
ylabel('phase')
```

```
%% linear prediction - x
X = zeros(M,N-M+1);
for jm = 1:M
    X(M-jm+1,:) = x(jm:jm+N-M).';
end;

R_x = (1./(N-M+1)).*X*X';
X2 = X(:,1:N-M);
X_x = repmat(x(M+1:N)',M,1);
p_xx = sum(X2.*X_x,2)./(N-M);
lp_opt_x = [1; -inv(R_x)*p_xx];
optfilt_tot = conv(lp_opt_x,wf_opt);

figure(3)
subplot(2,1,1)
plot(0:M,20*log10(abs(lp_opt)),0:2*M,20*log10(abs(optfilt_tot)));
ylabel('magnitude')
xlabel('filter coefficient index')
legend('LPE for d(n)','WF total response')
subplot(2,1,2)
plot(0:M,angle(lp_opt),0:2*M,angle(optfilt_tot));
ylabel('phase')
xlabel('filter coefficient index')

[Htot,Wtot] = freqz(optfilt_tot);
[Hlp,Wlp] = freqz(lp_opt);

figure(4)
subplot(2,1,1)
plot(Wlp./pi,20*log10(abs(Hlp)),Wtot./pi,20*log10(abs(Htot)))
xlabel('Normalized frequency')
ylabel('magnitude (dB)')
legend('LPE for d(n)','WF total response')
subplot(2,1,2)
plot(Wlp./pi,(180/pi).*angle(Hlp),Wtot./pi,(180/pi).*angle(Htot))
xlabel('Normalized frequency')
ylabel('phase')
```

3. We wish to derive and implement the steepest-descent algorithm for the cost function $J\left(\mathbf{w}(n)\right) = \mathbf{w}^H(n)\,\mathbf{R}\,\mathbf{w}(n) + \left|\mathbf{w}^H(n)\mathbf{s}_1(\theta) - 1\right|^2 + \left|\mathbf{w}^H(n)\mathbf{s}_2(\theta)\right|^2$. *{Note: this is <u>not</u> a LCMV or GSC formulation as these are not actually constraints}*

a) Analytically derive the optimum closed-form solution $\mathbf{w}_o$.

b) Using $\mathbf{R}$ from P3.mat, with $\mathbf{s}_1(\theta)$ and $\mathbf{s}_2(\theta)$ length N = 10 steering vectors pointed at electrical angles −20° and +20°, respectively, calculate $\mathbf{w}_o$ and plot the beampattern in terms of electrical angle.

c) For 3000 iterations and $\mathbf{w} = \mathbf{0}_{N\times 1}$ as initialization, now implement the steepest-descent algorithm. For five decimal precision, what is the largest constant step-size μ (denoted as $\mu_{\max}$) that still provides convergence. *{Hint: you can use trial and error, or determine the bound like in class}*

d) Using $\mu_{\max}$, plot the convergence of steepest descent in terms of the *squared deviation*, which is defined as $\left\|\mathbf{w}_o - \mathbf{w}(n)\right\|^2$ (plot versus iteration index *n*).

e) In general, now show analytically that the optimum adaptive step-size is $\mu_o(n) = \dfrac{\mathbf{g}^H(n)\,\mathbf{g}(n)}{\mathbf{g}^H(n)\,(\mathbf{R} + \mathbf{s}_1(\theta)\mathbf{s}_1^H(\theta) + \mathbf{s}_2(\theta)\mathbf{s}_2^H(\theta))\,\mathbf{g}(n)}$ by minimizing the modified cost function $J\left(\mathbf{w}(n) - 0.5\mu\,\mathbf{g}(n)\right)$ with respect to μ. You can assume $\mathbf{R} = \mathbf{R}^H$.

f) Implement the steepest-descent algorithm using $\mu_o(n)$ and plot the squared deviation for 3000 iterations using $\mathbf{w} = \mathbf{0}_{N\times 1}$ as initialization.

g) How many iterations are required for the squared deviation to surpass –30 dB when using $\mu_{\max}$ and $\mu_o(n)$, respectively?

<u>Solution:</u>

Derive the optimum solution $\mathbf{w}_o$ via direct matrix inversion.

$$\begin{aligned} J\left(\mathbf{w}\right) &= \mathbf{w}^H\mathbf{R}\mathbf{w} + \left|\mathbf{w}^H\mathbf{s}_1 - 1\right|^2 + \left|\mathbf{w}^H\mathbf{s}_2 - 0\right|^2 \\ &= \mathbf{w}^H\mathbf{R}\mathbf{w} + (\mathbf{w}^H\mathbf{s}_1 - 1)(\mathbf{w}^H\mathbf{s}_1 - 1)^* + (\mathbf{w}^H\mathbf{s}_2)(\mathbf{w}^H\mathbf{s}_2)^* \\ &= \mathbf{w}^H\mathbf{R}\mathbf{w} + \mathbf{w}^H\mathbf{s}_1\mathbf{s}_1^H\mathbf{w} - \mathbf{w}^H\mathbf{s}_1 - \mathbf{s}_1^H\mathbf{w} + 1 + \mathbf{w}^H\mathbf{s}_2\mathbf{s}_2^H\mathbf{w} \end{aligned}$$

$$\nabla_{\mathbf{w}^*} J = \mathbf{g} = 2(\mathbf{R} + \mathbf{s}_1\,\mathbf{s}_1^H + \mathbf{s}_2\,\mathbf{s}_2^H)\,\mathbf{w} - 2\,\mathbf{s}_1 \equiv 0$$

$$\mathbf{w}_o = [\mathbf{R} + \mathbf{s}_1\,\mathbf{s}_1^H + \mathbf{s}_2\,\mathbf{s}_2^H]^{-1}\mathbf{s}_1$$

The beampattern for the optimum solution is shown in Fig. 3.1, which does achieve a gain of 1 (0 dB) at −20° and a null at +20°. Note, though, that because we did not actually incorporate these responses as constraints, those results are not guaranteed in all cases. The maximum non-adaptive step-size is set by the largest eigenvalue of

$(\mathbf{R}+\mathbf{s}_1(\theta)\mathbf{s}_1^H(\theta)+\mathbf{s}_2(\theta)\mathbf{s}_2^H(\theta))$, which is $\lambda_{\max}=3179.1$, so $\mu_{\max}=2/\lambda_{\max}=0.000629$. To five digit precision (and noting that this maximum value cannot be exceeded, we obtain $\mu_{\max}=0.00062$.

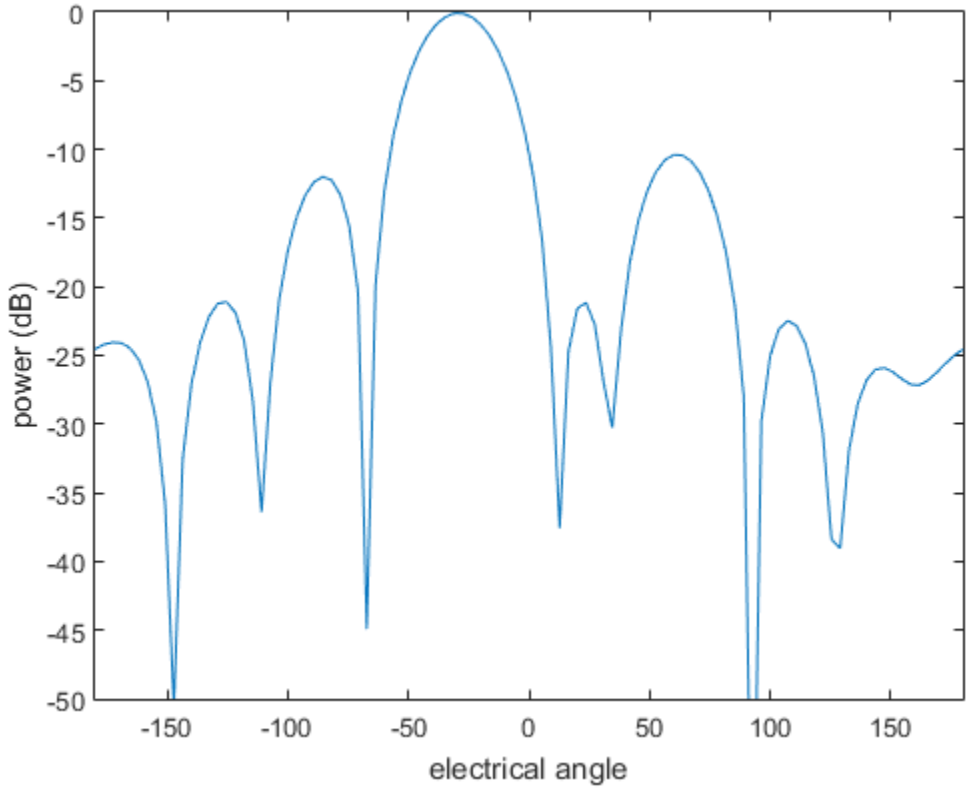


Fig. 3.1. Beampattern for the optimum solution

Now determine the optimum step-size:

$$J(\mathbf{w}-0.5\mu\,\mathbf{g})=(\mathbf{w}-0.5\mu\,\mathbf{g})^H\,\mathbf{R}\,(\mathbf{w}-0.5\mu\,\mathbf{g})+\left((\mathbf{w}-0.5\mu\,\mathbf{g})^H\,\mathbf{s}_1-1\right)\left(\mathbf{s}_1^H\,(\mathbf{w}-0.5\mu\,\mathbf{g})-1\right)$$
$$+\left((\mathbf{w}-0.5\mu\,\mathbf{g})^H\,\mathbf{s}_2\right)\left(\mathbf{s}_2^H\,(\mathbf{w}-0.5\mu\,\mathbf{g})\right)$$
$$=\mathbf{w}^H\mathbf{R}\mathbf{w}-0.5\mu\,\mathbf{g}^H\mathbf{R}\mathbf{w}-0.5\mu\,\mathbf{w}^H\mathbf{R}\mathbf{g}+0.25\mu^2\,\mathbf{g}^H\mathbf{R}\mathbf{g}$$
$$+\mathbf{w}^H\mathbf{s}_1\mathbf{s}_1^H\mathbf{w}-0.5\mu\mathbf{w}^H\mathbf{s}_1\mathbf{s}_1^H\mathbf{g}-\mathbf{w}^H\mathbf{s}_1-0.5\mu\mathbf{g}^H\mathbf{s}_1\mathbf{s}_1^H\mathbf{w}+0.25\mu^2\mathbf{g}^H\mathbf{s}_1\mathbf{s}_1^H\mathbf{g}$$
$$+0.5\mu\mathbf{g}^H\mathbf{s}_1-\mathbf{s}_1^H\mathbf{w}+0.5\mu\mathbf{s}_1^H\mathbf{g}+1$$
$$+\mathbf{w}^H\mathbf{s}_2\mathbf{s}_2^H\mathbf{w}-0.5\mu\mathbf{w}^H\mathbf{s}_2\mathbf{s}_2^H\mathbf{g}-0.5\mu\mathbf{g}^H\mathbf{s}_2\mathbf{s}_2^H\mathbf{w}+0.25\mu^2\mathbf{g}^H\mathbf{s}_2\mathbf{s}_2^H\mathbf{g}$$

$$\frac{\partial J}{\partial\mu}=-0.5\mathbf{g}^H\mathbf{R}\mathbf{w}-0.5\mathbf{w}^H\mathbf{R}\mathbf{g}+0.5\mu\,\mathbf{g}^H\mathbf{R}\mathbf{g}$$
$$-0.5\mathbf{w}^H\mathbf{s}_1\mathbf{s}_1^H\mathbf{g}-0.5\mathbf{g}^H\mathbf{s}_1\mathbf{s}_1^H\mathbf{w}+0.5\mu\mathbf{g}^H\mathbf{s}_1\mathbf{s}_1^H\mathbf{g}\quad .$$
$$+0.5\mathbf{g}^H\mathbf{s}_1+0.5\mathbf{s}_1^H\mathbf{g}$$
$$-0.5\mathbf{w}^H\mathbf{s}_2\mathbf{s}_2^H\mathbf{g}-0.5\mathbf{g}^H\mathbf{s}_2\mathbf{s}_2^H\mathbf{w}+0.5\mu\mathbf{g}^H\mathbf{s}_2\mathbf{s}_2^H\mathbf{g}\equiv 0$$

Solving for the step-size:

$$\mu_o = \frac{\left(\mathbf{g}^H\mathbf{R}\mathbf{w} + \mathbf{g}^H\mathbf{s}_1\mathbf{s}_1^H\mathbf{w} - \mathbf{g}^H\mathbf{s}_1 + \mathbf{g}^H\mathbf{s}_2\mathbf{s}_2^H\mathbf{w}\right) + \left(\mathbf{w}^H\mathbf{R}\mathbf{g} + \mathbf{w}^H\mathbf{s}_1\mathbf{s}_1^H\mathbf{g} - \mathbf{s}_1^H\mathbf{g} + \mathbf{w}^H\mathbf{s}_2\mathbf{s}_2^H\mathbf{g}\right)}{\mathbf{g}^H\mathbf{R}\mathbf{g} + \mathbf{g}^H\mathbf{s}_1\mathbf{s}_1^H\mathbf{g} + \mathbf{g}^H\mathbf{s}_2\mathbf{s}_2^H\mathbf{g}}$$

$$= \frac{\mathbf{g}^H\left(\mathbf{R}\mathbf{w} + \mathbf{s}_1\mathbf{s}_1^H\mathbf{w} - \mathbf{s}_1 + \mathbf{s}_2\mathbf{s}_2^H\mathbf{w}\right) + \left(\mathbf{R}\mathbf{w} + \mathbf{s}_1\mathbf{s}_1^H\mathbf{w} - \mathbf{s}_1 + \mathbf{s}_2\mathbf{s}_2^H\mathbf{w}\right)^H\mathbf{g}}{\mathbf{g}^H\mathbf{R}\mathbf{g} + \mathbf{g}^H\mathbf{s}_1\mathbf{s}_1^H\mathbf{g} + \mathbf{g}^H\mathbf{s}_2\mathbf{s}_2^H\mathbf{g}}$$

where we have used $\mathbf{R} = \mathbf{R}^H$ by definition in the problem. Noting that the two sets of terms in parentheses in the numerator are the gradient $\mathbf{g}$ (scaled by 0.5) yields

$$\mu_o = \frac{0.5\mathbf{g}^H\mathbf{g} + 0.5\mathbf{g}^H\mathbf{g}}{\mathbf{g}^H\left(\mathbf{R} + \mathbf{s}_1\mathbf{s}_1^H + \mathbf{s}_2\mathbf{s}_2^H\right)\mathbf{g}} = \frac{\mathbf{g}^H\mathbf{g}}{\mathbf{g}^H\left(\mathbf{R} + \mathbf{s}_1\mathbf{s}_1^H + \mathbf{s}_2\mathbf{s}_2^H\right)\mathbf{g}}.$$

Figure 3.2 presents the results when using the maximum constant and optimum step-sizes. The constant step-size requires 2003 iterations to reach –30 dB while the optimum only needs 713 iterations.

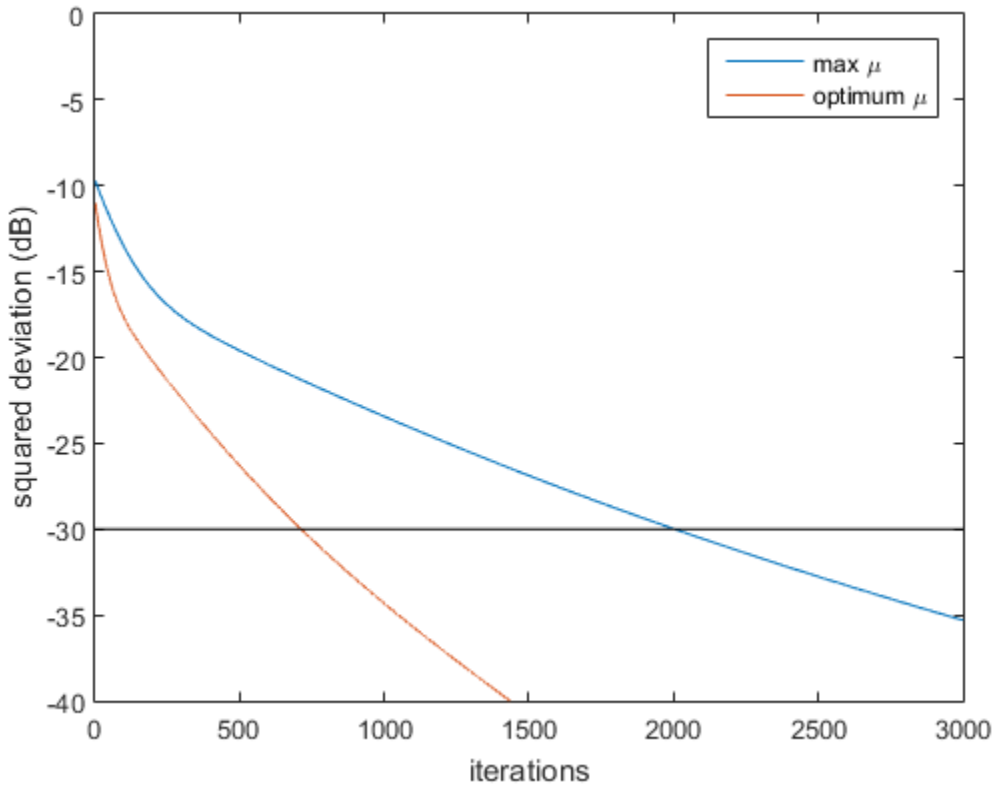


Fig. 3.2. Steepest-descent convergence in terms of squared deviation

Matlab Code for Problem 3

```
clear all;
load P3
N = 10;

s1 = (exp(-j.*(-20/180)*pi.*[0:N-1])).';
s2 = (exp(-j.*(+20/180)*pi.*[0:N-1])).';

B = R + s1*s1' + s2*s2';
w0 = inv(B)*s1;

LSP = linspace(-180,180,10*N);
for theta_i = 1:length(LSP)
    S(1:N,theta_i) = (exp(-j.*(LSP(theta_i)*pi/180).*[0:N-1]).');
end;
beam_w0 = S'*w0;

figure(1)
plot(LSP,20*log10(abs(beam_w0)));
axis([-180 180 -50 0])
xlabel('electrical angle')
ylabel('power (dB)')

[V,D] = eig(B);
mu_max = floor((2/max(diag(real(D))))*10^5)/10^5;

%%%%%%% via steepest-descent
mu = mu_max;
K = 3000; % number of iterations

w1 = zeros(N,1);
for k = 1:K
    del = 2*B*w1 - 2*s1;
    w1 = w1 - 0.5*mu*del;
    SqDe_Mumax(k) = (w1-w0)'*(w1-w0);
end;

w2 = zeros(N,1);
for k = 1:K
    del = 2*B*w2 - 2*s1;
    mu0(k) = real(del'*del/(del'*B*del));
    w2 = w2 - 0.5*mu0(k)*del;
    SqDe_Mu0(k) = (w2-w0)'*(w2-w0);
end;

figure(2)
plot(1:K,10*log10(SqDe_Mumax),1:K,10*log10(SqDe_Mu0))
axis([0 K -40 0])
line([0 K],[-30 -30],'Color',[0 0 0])
xlabel('iterations')
ylabel('squared deviation (dB)')
legend('max \mu','optimum \mu')
```

4. Data set P4.mat contains a known signal **x** of length M = 100 that we wish to deconvolve from other unknown signals.

a) Generate the *normalized matched filter* as $\mathbf{h}_{\text{NMF}} = \mathbf{x}^{B*}/(\mathbf{x}^H\mathbf{x})$ and plot the convolution of this filter with the signal **x**. Note that a filter output is an amplitude when plotting in dB.

b) Implement the Least-Squares *mismatched filter* $\mathbf{h}_{\text{MMF}}$ with a length of 2M and place the '1' in the elementary vector in element m = 1.5M. Once determined, normalize the MMF as

$$\mathbf{h}_{\text{NMMF}} = \frac{\mathbf{h}_{\text{MMF}}}{(\mathbf{h}_{\text{MMF}}^H\mathbf{h}_{\text{MMF}})^{1/2}(\mathbf{x}^H\mathbf{x})^{1/2}}$$

to allow for determination of the loss in SNR. Plot (in dB) the convolution of the normalized MMF with the signal **x**. Comment on what you observe. *{Hint: the 'toeplitz' command is useful for constructing the matrix* ***A*** *comprised of delay shifted versions of* ***x****}*

c) Repeat part *b)* except modify the matrix **A** by replacing the all values in the $(m-2)$, $(m-1)$, $(m+1)$, and $(m+2)$ rows with zeros. Comment on what you observe.

d) Repeat parts *b*) and *c*) except incorporate a diagonal load term that is 20 dB less than largest eigenvalue of the matrix $\mathbf{A}^H\mathbf{A}$.

e) For each of the four normalized mismatched filters obtained in the above steps, compute

$$\text{mismatch loss} = -20\log_{10}\left(\frac{\max|\mathbf{h}_{\text{NMMF}} * \mathbf{x}|}{\max|\mathbf{h}_{\text{NMF}} * \mathbf{x}|}\right)\text{dB},$$

for * representing convolution. Comment on what you observe.

Solution:

The matched filter (MF) and Least-Squares (LS) mismatched filter (MMF) responses to convolution with **x**, after aligning the peaks, are shown in Fig. 4.1. It is observed that the MMF cases produce lower sidelobes, thus providing a better approximation to an impulse for deconvolution. However, relative to the MF the different MMFs also exhibit mismatched losses as listed in the table below, with the case using zeroed rows and diagonal loading yielding the least mismatch loss (though it also provides the least sidelobe suppression among the MMFs).

Note that the zeroing of rows is a form of "beamspoiling" that actually degrades resolution slightly while provide some additional robustness to mismatch loss. These 3 factors: mismatch loss, sidelobe level, and resolution, can be traded off within the LS MMF framework by using these tools (and others).

LS MMF	Mismatch loss
normal	13.15 dB
zeroed rows	11.17 dB
diagonally loaded	4.84 dB
zeroes rows & diagonally loaded	3.04 dB

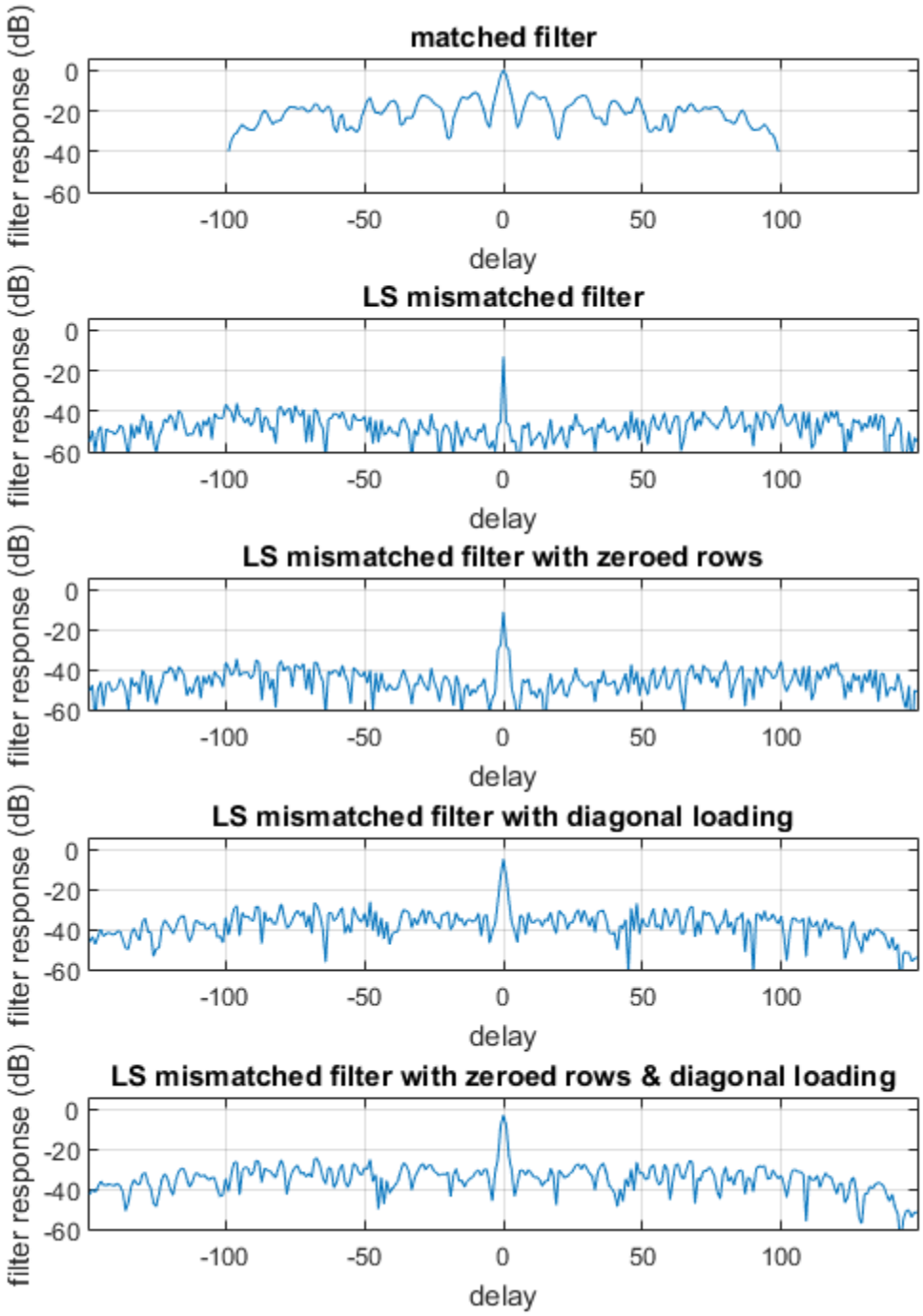


Figure 4.1. Matched filter and mismatched filter responses to convolution with **x**

Matlab Code for Problem 4

```
clear all
load P4
M = length(x);
mf = conj(flipud(x))./(x'*x);

K = 2;
mmfLen = M*K;
mmfOff = 1.5*M;

A = toeplitz([x; zeros(mmfLen-1,1)],[x(1) zeros(1,mmfLen-1)]);
Az = A;
Az(mmfOff-1,:) = 0;
Az(mmfOff+1,:) = 0;
Az(mmfOff-2,:) = 0;
Az(mmfOff+2,:) = 0;

[NA MA] = size(A);
[V D] = eig(A'*A);
lam_max = max(diag(D));

Phi = inv(A'*A)*A';
Phiz = inv(Az'*Az)*Az';
Phid = inv(A'*A + (0.01.*lam_max).*eye(MA))*A';
Phidz = inv(Az'*Az + (0.01.*lam_max).*eye(MA))*Az';

e = zeros(NA,1);
e(mmfOff) = 1;

mmf = Phi*e;
mmf = mmf./sqrt(mmf'*mmf)./sqrt(x'*x);
mmfz = Phiz*e;
mmfz = mmfz./sqrt(mmfz'*mmfz)./sqrt(x'*x);
mmfd = Phid*e;
mmfd = mmfd./sqrt(mmfd'*mmfd)./sqrt(x'*x);
mmfdz = Phidz*e;
mmfdz = mmfdz./sqrt(mmfdz'*mmfdz)./sqrt(x'*x);

mf_out = conv(mf,x);
mmf_out = conv(mmf,x);
mmfz_out = conv(mmfz,x);
mmfd_out = conv(mmfd,x);
mmfdz_out = conv(mmfdz,x);

MMloss = -20*log10(max(abs(mmf_out))/max(abs(mf_out)))
MMlossz = -20*log10(max(abs(mmfz_out))/max(abs(mf_out)))
MMlossd = -20*log10(max(abs(mmfd_out))/max(abs(mf_out)))
MMlossdz = -20*log10(max(abs(mmfdz_out))/max(abs(mf_out)))

figure(1)
subplot(5,1,1)
plot(-(M-1):(M-1),20*log10(abs(mf_out)))
axis([-((K+1)*M/2-1) ((K+1)*M/2-1) -60 5])
grid on
title('matched filter')
```

```
xlabel('delay')
ylabel('filter response (dB)')

subplot(5,1,2)
plot(-((K+1)*M/2-1):((K+1)*M/2-1),20*log10(abs(mmf_out)))
axis([-((K+1)*M/2-1) ((K+1)*M/2-1) -60 5])
title('LS mismatched filter')
grid on
xlabel('delay')
ylabel('filter response (dB)')

subplot(5,1,3)
plot(-((K+1)*M/2-1):((K+1)*M/2-1),20*log10(abs(mmfz_out)))
axis([-((K+1)*M/2-1) ((K+1)*M/2-1) -60 5])
title('LS mismatched filter with zeroed rows')
grid on
xlabel('delay')
ylabel('filter response (dB)')

subplot(5,1,4)
plot(-((K+1)*M/2-1):((K+1)*M/2-1),20*log10(abs(mmfd_out)))
axis([-((K+1)*M/2-1) ((K+1)*M/2-1) -60 5])
title('LS mismatched filter with diagonal loading')
grid on
xlabel('delay')
ylabel('filter response (dB)')

subplot(5,1,5)
plot(-((K+1)*M/2-1):((K+1)*M/2-1),20*log10(abs(mmfdz_out)))
axis([-((K+1)*M/2-1) ((K+1)*M/2-1) -60 5])
title('LS mismatched filter with zeroed rows & diagonal loading')
grid on
xlabel('delay')
ylabel('filter response (dB)')
```

5. Data set P4.mat also contains the received signal $y(n)$ that is the result of convolving the known signal $\mathbf{x}$ from Prob. 4 with some unknown system. Apply each of the 5 filters (MF and 4 MMF) from Prob. 4 to perform deconvolution to estimate the unknown system. Plot the results (in dB) and comment on what you observe.

Solution:

The matched filter and Least-Squares (LS) mismatched filter responses to the received signal $y(n)$ are depicted in Fig. 5.1. Based on the matched filter response, there appears to be only 1 component to the unknown system. However, the set of MMF responses reveal that there are actually two additional smaller components. The degree to which these smaller components are visible depends on the degree to which the sidelobes are reduced (arguably the MMF with the worst mismatch loss provides the best sidelobe suppression as a trade-off).

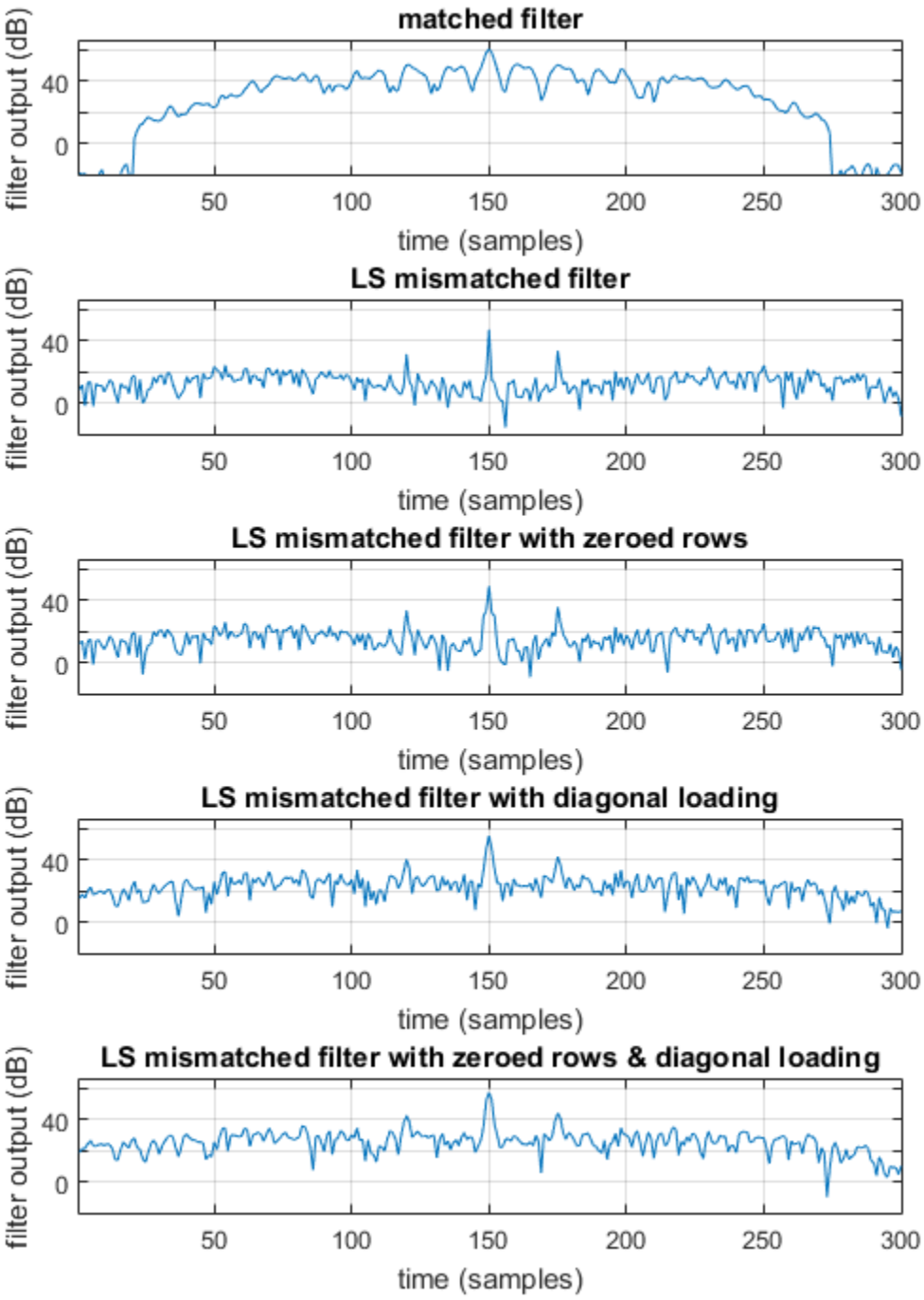


Figure 5.1. Matched filter and mismatched filter responses to $y(n)$

Matlab Code for Problem 5

```
clear all
load P4
M = length(x);
mf = conj(flipud(x))./(x'*x);

K = 2;
mmfLen = M*K;
mmfOff = 1.5*M;

A = toeplitz([x; zeros(mmfLen-1,1)],[x(1) zeros(1,mmfLen-1)]);
Az = A;
Az(mmfOff-1,:) = 0;
Az(mmfOff+1,:) = 0;
Az(mmfOff-2,:) = 0;
Az(mmfOff+2,:) = 0;

[NA MA] = size(A);
[V D] = eig(A'*A);
lam_max = max(diag(D));

Phi = inv(A'*A)*A';
Phiz = inv(Az'*Az)*Az';
Phid = inv(A'*A + (0.02.*lam_max).*eye(MA))*A';
Phidz = inv(Az'*Az + (0.02.*lam_max).*eye(MA))*Az';

e = zeros(NA,1);
e(mmfOff) = 1;

mmf = Phi*e;
mmf = mmf./sqrt(mmf'*mmf)./sqrt(x'*x);
mmfz = Phiz*e;
mmfz = mmfz./sqrt(mmfz'*mmfz)./sqrt(x'*x);
mmfd = Phid*e;
mmfd = mmfd./sqrt(mmfd'*mmfd)./sqrt(x'*x);
mmfdz = Phidz*e;
mmfdz = mmfdz./sqrt(mmfdz'*mmfdz)./sqrt(x'*x);

mf_out = conv(mf,x);
mmf_out = conv(mmf,x);
mmfz_out = conv(mmfz,x);
mmfd_out = conv(mmfd,x);
mmfdz_out = conv(mmfdz,x);

yf_mf = conv(mf,y);
ymf_shft = [zeros(1*M-1,1); yf_mf];
yf_mmf = conv(mmf,y);
yf_mmfz = conv(mmfz,y);
yf_mmfd = conv(mmfd,y);
yf_mmfdz = conv(mmfdz,y);

len = length(y);

figure(1)
```

```
subplot(5,1,1)
plot(1:len,20*log10(abs(ymf_shft(M:len+M-1))));
axis([1 len -20 65])
title('matched filter')
grid on
xlabel('time (samples)')
ylabel('filter output (dB)')

subplot(5,1,2)
plot(1:len,20*log10(abs(yf_mmf(M+1:len+M))));
axis([1 len -20 65])
title('LS mismatched filter')
grid on
xlabel('time (samples)')
ylabel('filter output (dB)')

subplot(5,1,3)
plot(1:len,20*log10(abs(yf_mmfz(M+1:len+M))));
axis([1 len -20 65])
title('LS mismatched filter with zeroed rows')
grid on
xlabel('time (samples)')
ylabel('filter output (dB)')

subplot(5,1,4)
plot(1:len,20*log10(abs(yf_mmfd(M+1:len+M))));
axis([1 len -20 65])
title('LS mismatched filter with diagonal loading')
grid on
xlabel('time (samples)')
ylabel('filter output (dB)')

subplot(5,1,5)
plot(1:len,20*log10(abs(yf_mmfdz(M+1:len+M))));
axis([1 len -20 65])
title('LS mismatched filter with zeroed rows & diagonal loading')
grid on
xlabel('time (samples)')
ylabel('filter output (dB)')
```

6. Data set P6.m contains time samples from five different $M = 20$ element uniform linear arrays (ULAs) with half-wavelength element spacing. Each array has a different degree of calibration error with respect to the idealized array manifold. None of the incident source signals are temporally correlated (i.e. no multipath effects).

a) Plot the MVDR power spectrum for each array using both the normal estimate of the correlation matrix and the forward/backward averaging estimate of the correlation matrix. Comment on what you observe.
b) For each array, plot the eigenvalues of the two correlation matrix estimates and comment on what you observe.
c) Based on the assumptions involved with forward/backward averaging and the relation to mismatch effects, use your previous observations to rank the five arrays in terms of the severity of calibration error.

Solution:

Array calibration errors are fundamentally a mismatch effect (the Vandermonde steering vectors we presume for a ULA will differ from reality). The forward/backward (F/B) averaging approach assumes that the ULA conforms to an idealized structure where the elements are uniformly spaced with the same gain/phase response for each antenna element (i.e. all the elements are identical as well as their associated RF receive chains prior to digitization). To the degree this assumption is incorrect F/B averaging will result in an increased rank of the corresponding sample correlation matrix (SCM) where these seemingly new signal components represent mismatch effects that are signal dependent (multiplicative "noise" instead of the usual additive noise).

Furthermore, we expect the MVDR response, which provides a super-resolution capability that is rather sensitive to calibration mismatch error, to be a good indicator of the degree of calibration error by examining the "sharpness" of the peaks (see Figs. 6.1 – 6.5). Clearly array #5 and to a slightly lesser degree array #1 have degraded sharpness in comparison to the other arrays (with array #4 somewhere in the middle). An examination of the eigenvalues for the SCM and F/B SCM for each case supports this explanation by considering the degree of difference between the principal eigenvalues with and without F/B averaging.

For the average gain error defined as the standard deviation of Δz over the array elements converted to a percentage and the average phase error defined as the standard deviation of the phase of z over the array elements (relative to a nominal phase error of 0°) the five arrays actually possess the calibration errors in the table below (this could not be discerned from the data you were provided). The results in Figs. 6.1-6.5, particularly the eigenvalue plots, agree with this ranking. Note that, it is generally found that signal processing algorithms are far more sensitive to phase errors than gain errors.

Array	gain error	phase error	Best to worst
1	17.5%	7.4°	4th
2	5.7%	2.1°	2nd
3	1.2%	0.4°	1st (best)
4	10.4%	4.9°	3rd
5	44.9%	23.1°	5th (worst)

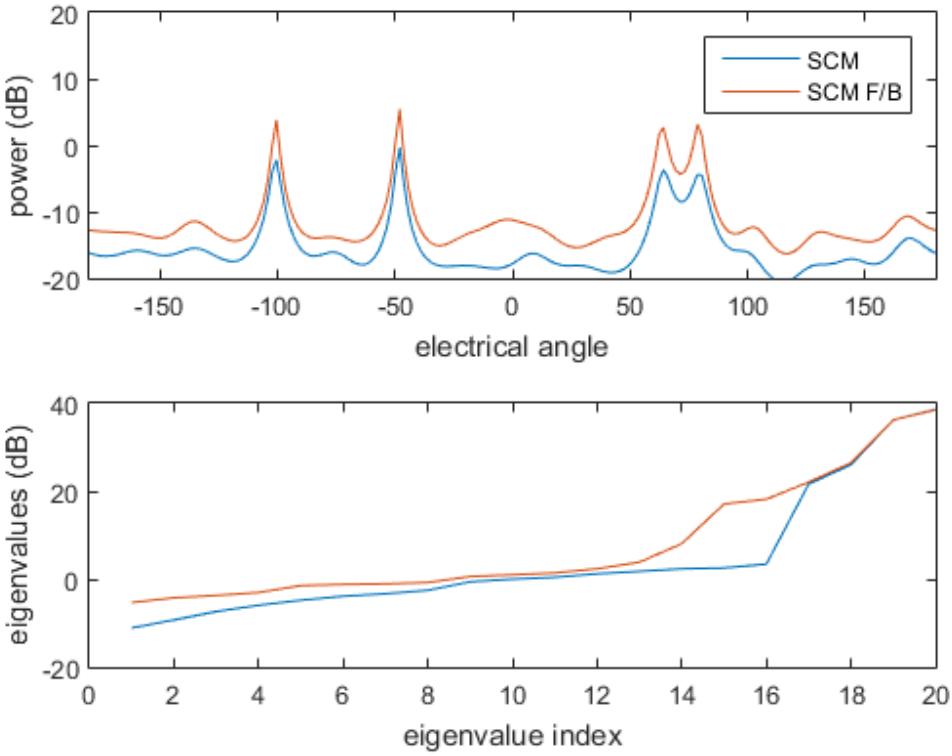


Figure 6.1. Top panel: MVDR power spectrum for set 1,
Bottom panel: eigenvalues for set 1

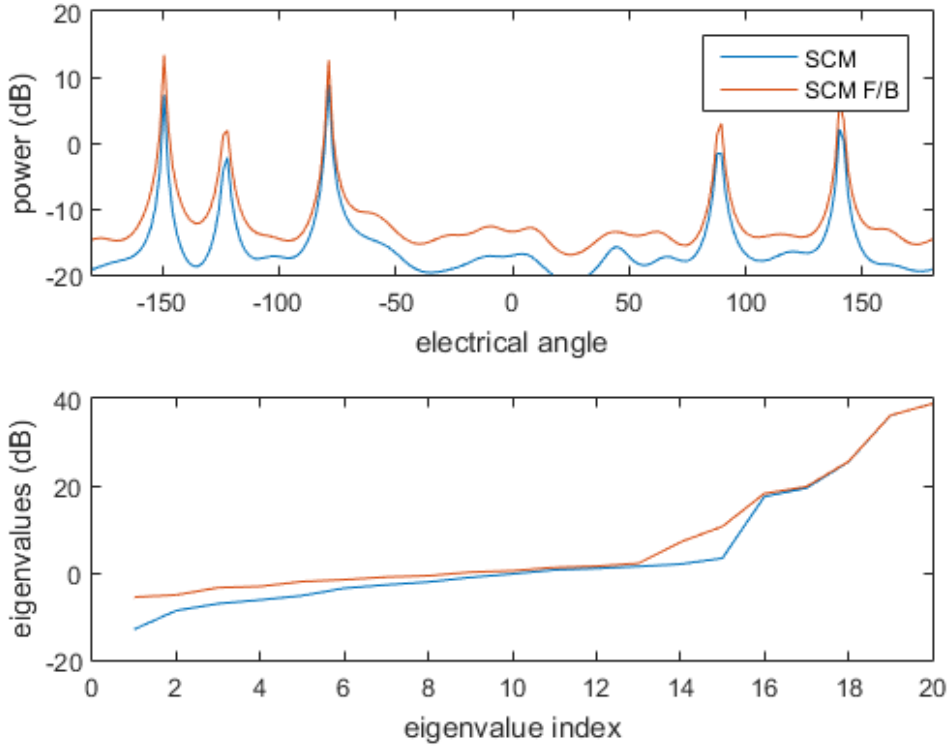


Figure 6.2. Top panel: MVDR power spectrum for set 2,
Bottom panel: eigenvalues for set 2

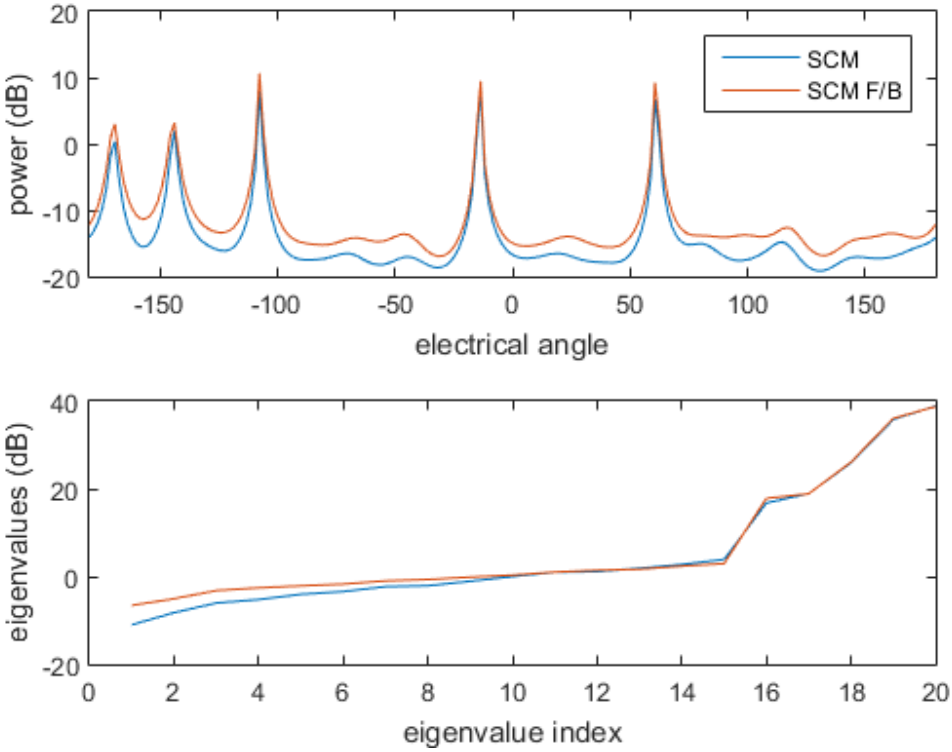


Figure 6.3. Top panel: MVDR power spectrum for set 3,
Bottom panel: eigenvalues for set 3

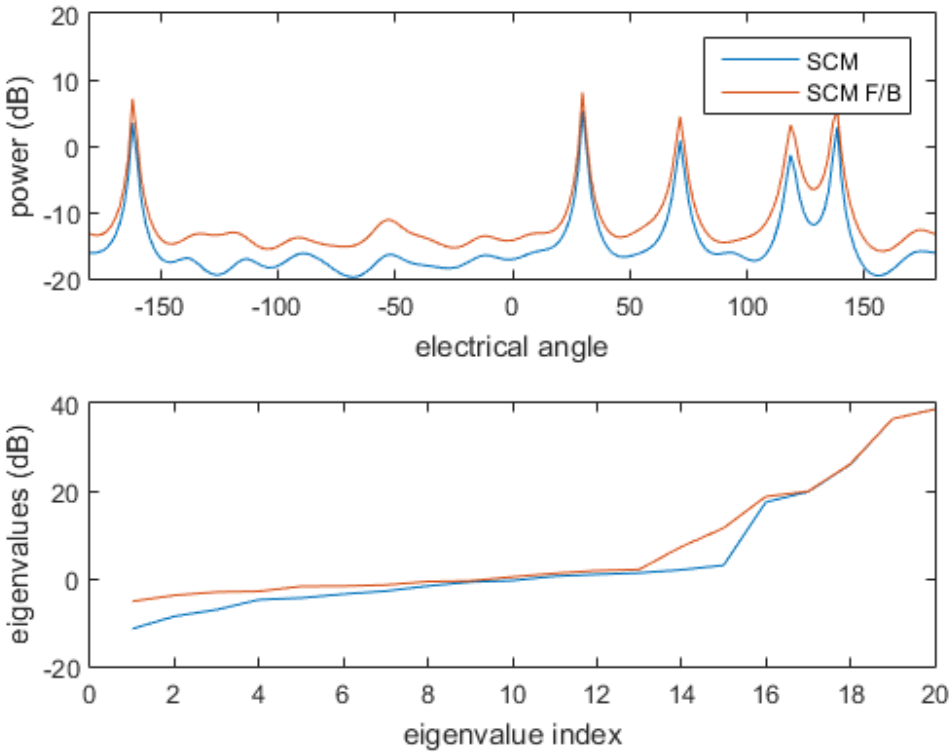


Figure 6.4. Top panel: MVDR power spectrum for set 4,
Bottom panel: eigenvalues for set 4

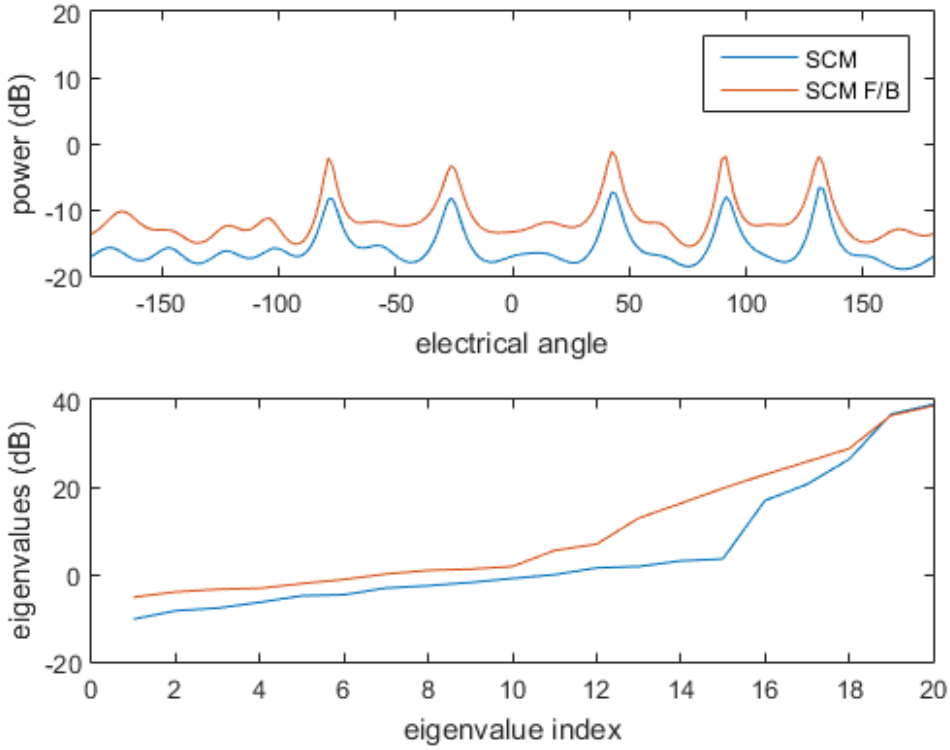


Figure 6.5. Top panel: MVDR power spectrum for set 5,
Bottom panel: eigenvalues for set 5

Matlab Code for Problem 6

```
clear all
load P6
[M K] = size(X1);

R1 = X1*X1'./K;
R1i = inv(R1);
R2 = X2*X2'./K;
R2i = inv(R2);
R3 = X3*X3'./K;
R3i = inv(R3);
R4 = X4*X4'./K;
R4i = inv(R4);
R5 = X5*X5'./K;
R5i = inv(R5);

X1f = conj(flipud(X1)); % flips the array elements
R1f = (X1*X1' + X1f*X1f')./(2*K);
R1fi = inv(R1f);
X2f = conj(flipud(X2)); % flips the array elements
R2f = (X2*X2' + X2f*X2f')./(2*K);
R2fi = inv(R2f);
X3f = conj(flipud(X3)); % flips the array elements
R3f = (X3*X3' + X3f*X3f')./(2*K);
R3fi = inv(R3f);
X4f = conj(flipud(X4)); % flips the array elements
R4f = (X4*X4' + X4f*X4f')./(2*K);
R4fi = inv(R4f);
X5f = conj(flipud(X5)); % flips the array elements
R5f = (X5*X5' + X5f*X5f')./(2*K);
R5fi = inv(R5f);

LSP = linspace(-180,180,M*10);

for theta_i = 1:length(LSP)
    Ct(1:M,theta_i) = (exp(j.*(LSP(theta_i)*pi/180).*[0:M-1]).');
    MVDR_pow1(theta_i) = 1./(Ct(:,theta_i)'*R1i*Ct(:,theta_i));
    MVDR_pow2(theta_i) = 1./(Ct(:,theta_i)'*R2i*Ct(:,theta_i));
    MVDR_pow3(theta_i) = 1./(Ct(:,theta_i)'*R3i*Ct(:,theta_i));
    MVDR_pow4(theta_i) = 1./(Ct(:,theta_i)'*R4i*Ct(:,theta_i));
    MVDR_pow5(theta_i) = 1./(Ct(:,theta_i)'*R5i*Ct(:,theta_i));

    MVDR_pow1f(theta_i) = 1./(Ct(:,theta_i)'*R1fi*Ct(:,theta_i));
    MVDR_pow2f(theta_i) = 1./(Ct(:,theta_i)'*R2fi*Ct(:,theta_i));
    MVDR_pow3f(theta_i) = 1./(Ct(:,theta_i)'*R3fi*Ct(:,theta_i));
    MVDR_pow4f(theta_i) = 1./(Ct(:,theta_i)'*R4fi*Ct(:,theta_i));
    MVDR_pow5f(theta_i) = 1./(Ct(:,theta_i)'*R5fi*Ct(:,theta_i));
end;

[V1,D1] = eig(R1);
[V1f,D1f] = eig(R1f);
[V2,D2] = eig(R2);
[V2f,D2f] = eig(R2f);
[V3,D3] = eig(R3);
[V3f,D3f] = eig(R3f);
```

```
[V4,D4] = eig(R4);
[V4f,D4f] = eig(R4f);
[V5,D5] = eig(R5);
[V5f,D5f] = eig(R5f);

figure(1)
subplot(2,1,1)
plot(LSP,10*log10(MVDR_pow1),LSP,10*log10(MVDR_pow1f));
axis([-180 180 -20 20])
legend('SCM','SCM F/B')
xlabel('electrical angle')
ylabel('power (dB)')
subplot(2,1,2)
plot(1:M,10*log10(diag(D1)),1:M,10*log10(diag(D1f)))
xlabel('eigenvalue index')
ylabel('eigenvalues (dB)')

figure(2)
subplot(2,1,1)
plot(LSP,10*log10(MVDR_pow2),LSP,10*log10(MVDR_pow2f));
axis([-180 180 -20 20])
legend('SCM','SCM F/B')
xlabel('electrical angle')
ylabel('power (dB)')
subplot(2,1,2)
plot(1:M,10*log10(diag(D2)),1:M,10*log10(diag(D2f)))
xlabel('eigenvalue index')
ylabel('eigenvalues (dB)')

figure(3)
subplot(2,1,1)
plot(LSP,10*log10(MVDR_pow3),LSP,10*log10(MVDR_pow3f));
axis([-180 180 -20 20])
legend('SCM','SCM F/B')
xlabel('electrical angle')
ylabel('power (dB)')
subplot(2,1,2)
plot(1:M,10*log10(diag(D3)),1:M,10*log10(diag(D3f)))
xlabel('eigenvalue index')
ylabel('eigenvalues (dB)')

figure(4)
subplot(2,1,1)
plot(LSP,10*log10(MVDR_pow4),LSP,10*log10(MVDR_pow4f));
axis([-180 180 -20 20])
legend('SCM','SCM F/B')
xlabel('electrical angle')
ylabel('power (dB)')
subplot(2,1,2)
plot(1:M,10*log10(diag(D4)),1:M,10*log10(diag(D4f)))
xlabel('eigenvalue index')
ylabel('eigenvalues (dB)')

figure(5)
subplot(2,1,1)
plot(LSP,10*log10(MVDR_pow5),LSP,10*log10(MVDR_pow5f));
```

```
axis([-180 180 -20 20])
legend('SCM','SCM F/B')
xlabel('electrical angle')
ylabel('power (dB)')
subplot(2,1,2)
plot(1:M,10*log10(diag(D5)),1:M,10*log10(diag(D5f)))
xlabel('eigenvalue index')
ylabel('eigenvalues (dB)')
```