POLITECNICO DI MILANO

---

ADVANCED DIGITAL SIGNAL PROCESSING

# HOMEWORK #1

# MIMO System Identification and Deconvolution


Professor:

**Umberto Spagnolini**

Autor:

**Ludovico Mazzi**


---

Year 2021/2022

# Contents

# Chapter 1

# Abstract

The objective of the homework is to evaluate MSE in MIMO channel estimation and deconvolution.

MIMO channel is estimated through Q pilot samples and then unknown $P - Q$ signals are estimated through deconvolution using the MIMO channel just estimated.

With respect to the first paragraph, it is clear that MSE performs better with higher values of L and instead it increases with higher values of $\rho$.

Regarding MIMO Estimation and Deconvolution paragraphs, it can be stated that there is a general increasing of performance of the MSE in accordance with the increasing of the SNR.

MLE in the limit, for growing SNR, approaches MMSE.

The implementation is made by using MATLAB.

# Chapter 2

# Noise Generation

## 2.1 Statistic Characterization

Input signal is denoted by $x[k]$, it contains both pilot and information samples. It is generated by uncorrelated and white random processes:

$$x[k] \sim \mathcal{N}(0, \sigma^2 I) \tag{2.1}$$

Noise is Gaussian and temporally white, but it is mutually correlated among paired lines:

$$E\{w[k]w[l]\} = C_w \delta[l-k] \tag{2.2}$$

$$w[k] \sim \mathcal{N}(0, C_w) \tag{2.3}$$

With elements of $C_w$:

$$\begin{cases} \rho & if \ \ i \neq j \\ 1 & if \ \ i = j \end{cases} \tag{2.4}$$

Hence, if $\rho = 0$ the noise becomes white.

$randn(M, P)$ function generates P samples of white noise, with M channels.

$chol(A)$ function computes Cholesky Factorization, which is a special form of LU factorization for symmetric definite positive matrices.

This function is used in order to create correlated noise from white noise.

## 2.2 Estimation of the Covariance Matrix

Sample covariance matrix is estimated using the MATLAB function $cov()$.

Then, the estimate is evaluated through the MSE, with respect to all its entries.

Estimation improves in accordance with the growth of L.

Obviously, in this environment, when more data are given, the results are better.

On the other side, results show detriments when $\rho$ increases.

The results are shown in the following page:

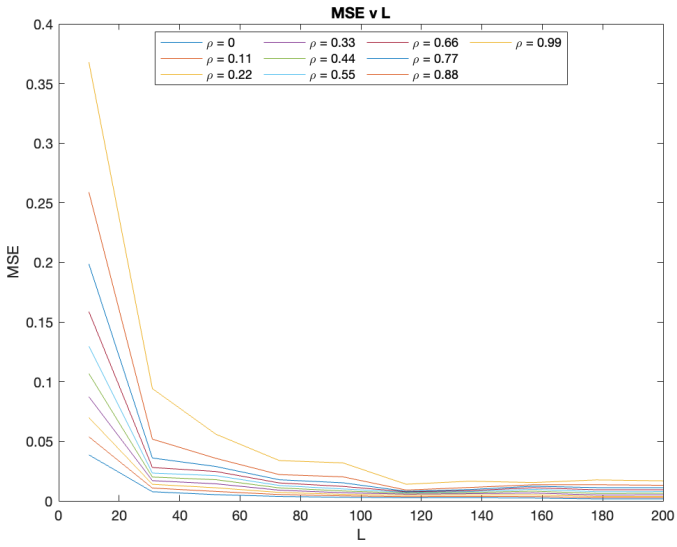


Figure 2.1: MSE v. L

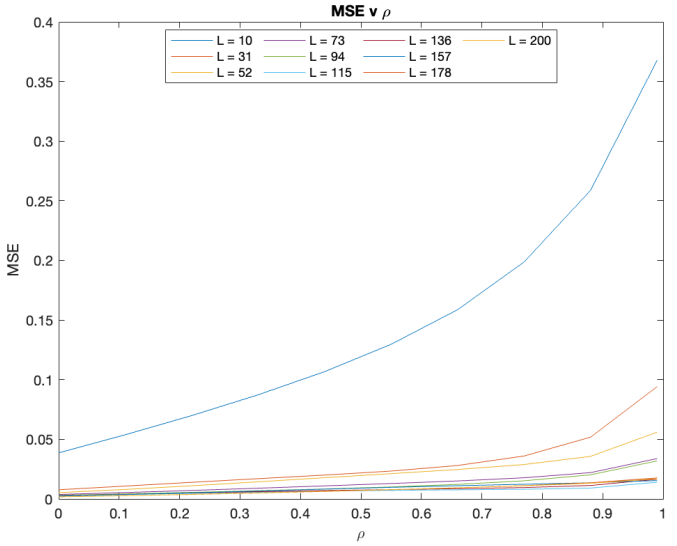


Figure 2.2: MSE v. $\rho$

# Chapter 3

# MIMO Estimation

## 3.1 Memoryless Filter

The term *memoryless*, associated to filter, means that $K = 1$.

The original system:

$$y = Hx + w \tag{3.1}$$

is transformed into:

$$y = Xh + w \tag{3.2}$$

where $X$ is the convolutional matrix of the input signal and $h$ is the filter response in vector form.

Since the system is modified, we can write a closed form of $\hat{h}_{ML}$ as follows:

$$\hat{h}_{ML} = (X^T C_w^{-1} X)^{-1} X^T C_w^{-1} y \tag{3.3}$$

CRB represents the lower bound of MSE reachable by MLE estimation.

Here it is:

$$CRB = (X^T C_w^{-1} X)^{-1} \tag{3.4}$$

It has to be noted that CRB is a matrix, but a scalar is needed to be compared with the MSE.

Therefore, the average of the diagonal entries of CRB matrix is considered as lower bound value.

In the first place, it is clear to note that the estimates improve by increasing the SNR.

From figure 3.1, as we use more Q samples for channel estimation, the estimate result gets better, therefore MSE decreases.

From figure 3.2, it can be said that different values of $\alpha$ do not affect the performance of the MSE.

It has to be mentioned that $\rho$ is chosen to be equal to 0.1.

The results are shown in the following page:

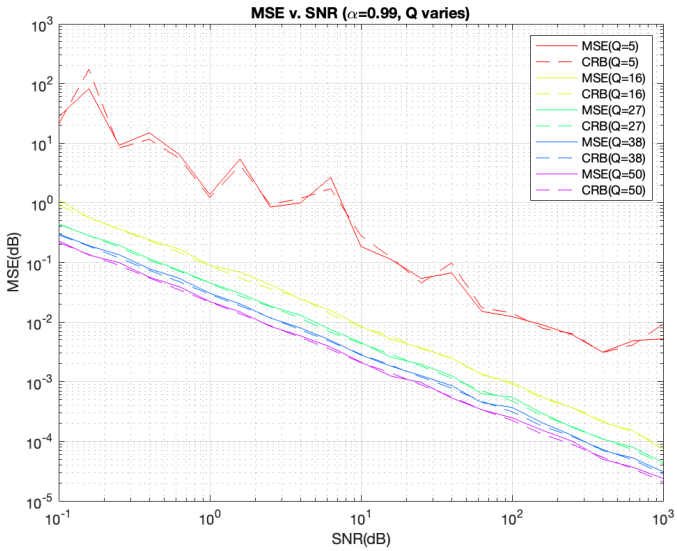


Figure 3.1: MSE v. SNR ($\alpha\, fixed, Q\, varies$)

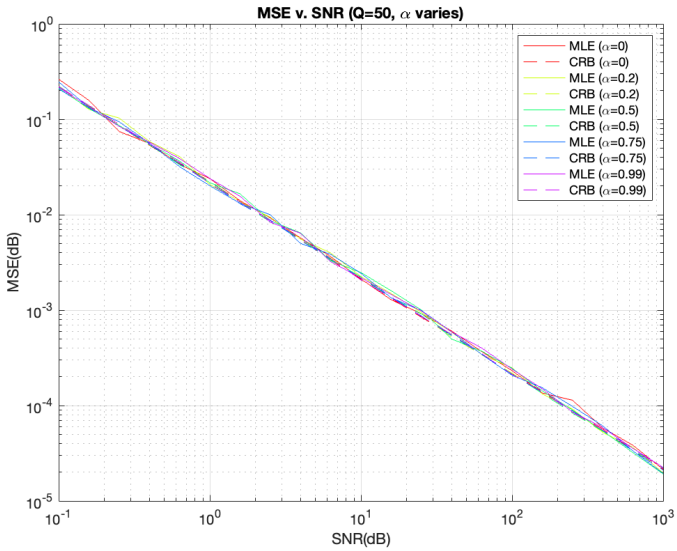


Figure 3.2: MSE v. SNR ($\alpha\, varies, Q\, fixed$)

## 3.2 Memory Filter

For the memory filter there is no more a $M \times N$ filter, rather $M \times N \times K$ filter. Kronecker product is used in order to manage the new dimensions:

$$y_{ML\times 1} = (x_{L\times NK} \otimes I_{M\times N})_{ML\times MNK} \;\; h_{MNK\times 1} + w_{ML\times 1} \tag{3.5}$$

with $L = Q + K - 1$.

So:

$$X_{conv} = \begin{bmatrix} \hat{x}_1[1] & 0 & 0 & 0 & \dots & \hat{x}_1[K] & 0 & 0 & 0 \\ 0 & \hat{x}_2[1] & 0 & 0 & \dots & 0 & \hat{x}_2[K] & 0 & 0 \\ 0 & 0 & \hat{x}_3[1] & 0 & \dots & 0 & 0 & \hat{x}_3[K] & 0 \\ 0 & 0 & 0 & \hat{x}_4[1] & \dots & 0 & 0 & 0 & \hat{x}_4[K] \end{bmatrix}_{ML\times MNK} \tag{3.6}$$

with $K = 1, 2, 3, 4$.

and with: $\hat{x}_i = \begin{bmatrix} x_i[1], x_i[2], \dots, x_i[Q] \end{bmatrix}^T \quad i = 1, 2, 3, 4$

Finally:

$$\begin{bmatrix} \hat{y}_1 \\ \hat{y}_2 \\ \hat{y}_3 \\ \hat{y}_4 \end{bmatrix}_{ML\times 1} = X_{conv_{ML\times MNK}} \begin{bmatrix} \hat{h}_{11} \\ \hat{h}_{12} \\ \hat{h}_{13} \\ \vdots \\ \hat{h}_{MN} \end{bmatrix}_{MNK\times 1} + \begin{bmatrix} \hat{w}_1 \\ \hat{w}_2 \\ \hat{w}_3 \\ \hat{w}_4 \end{bmatrix}_{ML\times 1} \tag{3.7}$$

with: $\hat{y}_i = \begin{bmatrix} y_i[1], y_i[2], \dots, y_i[Q + K - 1] \end{bmatrix}^T \quad i = 1, 2, 3, 4$

The results are shown below:

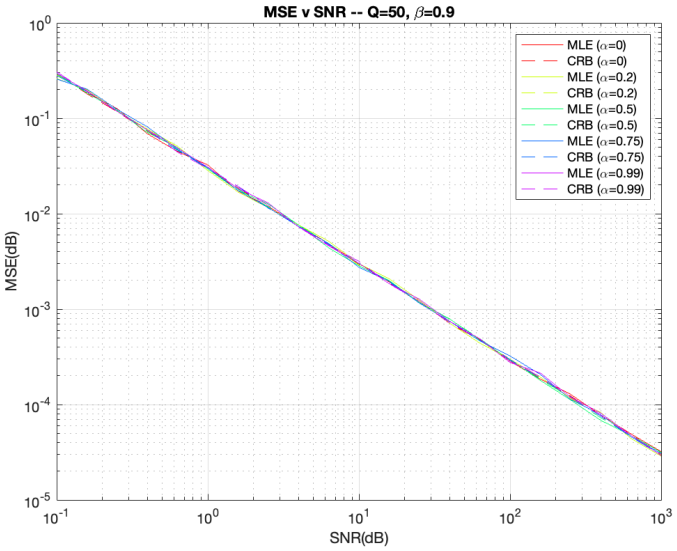


Figure 3.3: MSE v. SNR ($\alpha\, varies, Q = 50, \beta = 0.9$)

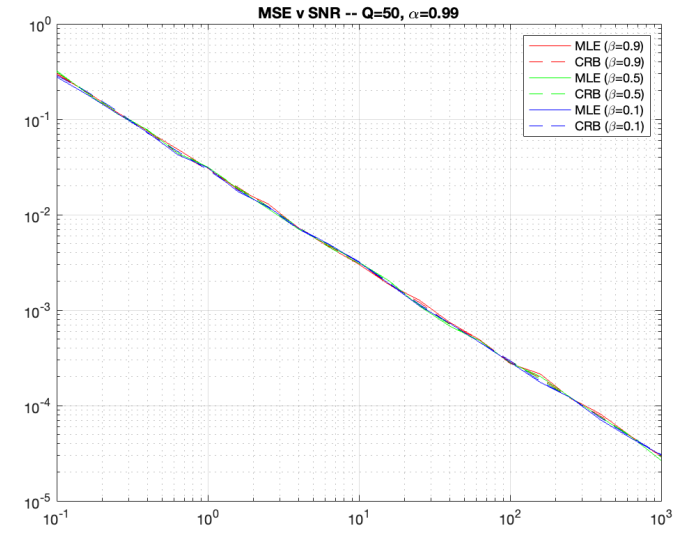


Figure 3.4: MSE v. SNR ($\beta\, varies, Q = 50, \alpha = 0.99$)

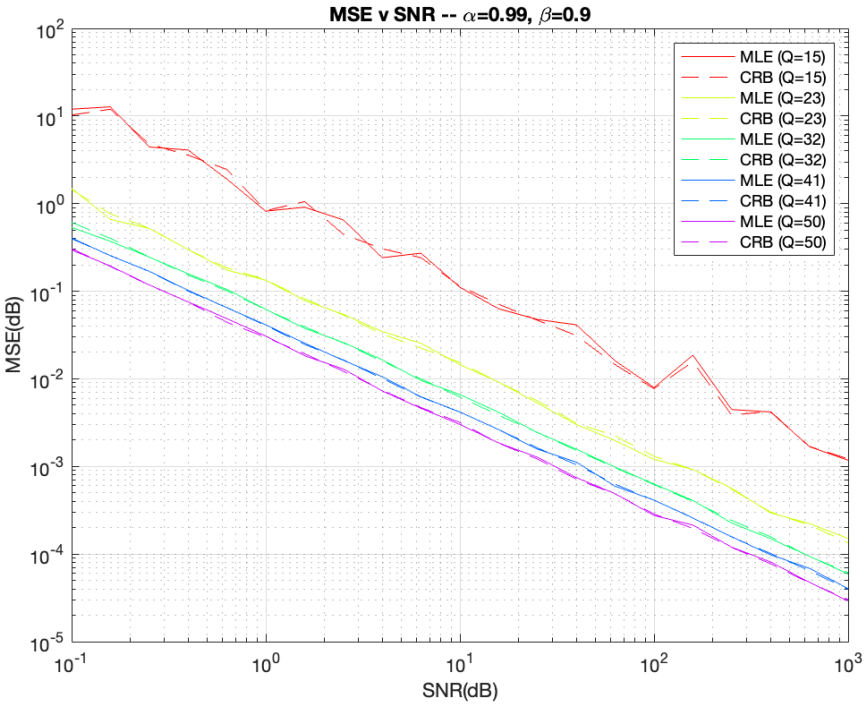


Figure 3.5: MSE v. SNR (Q varies, $\alpha = 0.99, \beta = 0.9$)

# Chapter 4

# MIMO Deconvolution

## 4.1 General Scheme

True data are in first place generated: so $x$ is generated, $w$ is generated and then, $y$ is computed. By using those, $x_{pilot}$ and $y_{pilot}$ are generated. So the first part is treated in a similar way as in MIMO Estimation chapter.

After the filter $\hat{H}$ is estimated, we can estimate the data through:

$$y_{est} = \hat{H}x_{est} + w \tag{4.1}$$

From the result above, MLE and MMSE estimators of $x$ are computed as follows:

$$x_{MLE} = (\hat{H}^T C_w^{-1} \hat{H})^{-1} \hat{H}^T C_w^{-1} y \tag{4.2}$$

$$x_{MMSE} = \mu_x + (\hat{H}^T C_w^{-1} \hat{H} + C_{xx}^{-1})^{-1} \hat{H}^T C_w^{-1} (y - \hat{H}\mu_x) \tag{4.3}$$

## 4.2 MLE v. MMSE

Since the signal x has zero mean value, hence MMSE estimation formula converts into:

$$x_{MMSE} = (\hat{H}^T C_w^{-1} \hat{H} + C_{xx}^{-1})^{-1} \hat{H}^T C_w^{-1} y \tag{4.4}$$

By a comparison between equations (4.2) and (4.4) it is obvious that they differ only for the term $C_{xx}^{-1}$ inside the parentheses.

$C_{xx} = \sigma_x^2 I$, therefore its inverse is really small when SNR is large. thus, as SNR increases MLE grows in its performance and MLE estimator approaches MMSE estimator values.

Instead, if SNR is very small, $C_{xx}^{-1}$ will grow, in a way such that:

$$x_{MMSE} \approx (C_{xx}^{-1})^{-1} \hat{H}^T C_w^{-1} y \tag{4.5}$$

Therefore, MMSE estimator reaches the proximity of zero.

From a theoretical perspective, the MLE uses a-posteriori information, while the MMSE involves both a-priori and a-posteriori information.

It is clear to understand that when SNR is high, all the information coming from a-priori data will be not needed, since the received signal is powerful enough.

But, on the other hand, if SNR is low, received signal will be too weak to be used directly, therefore a-priori information is needed.

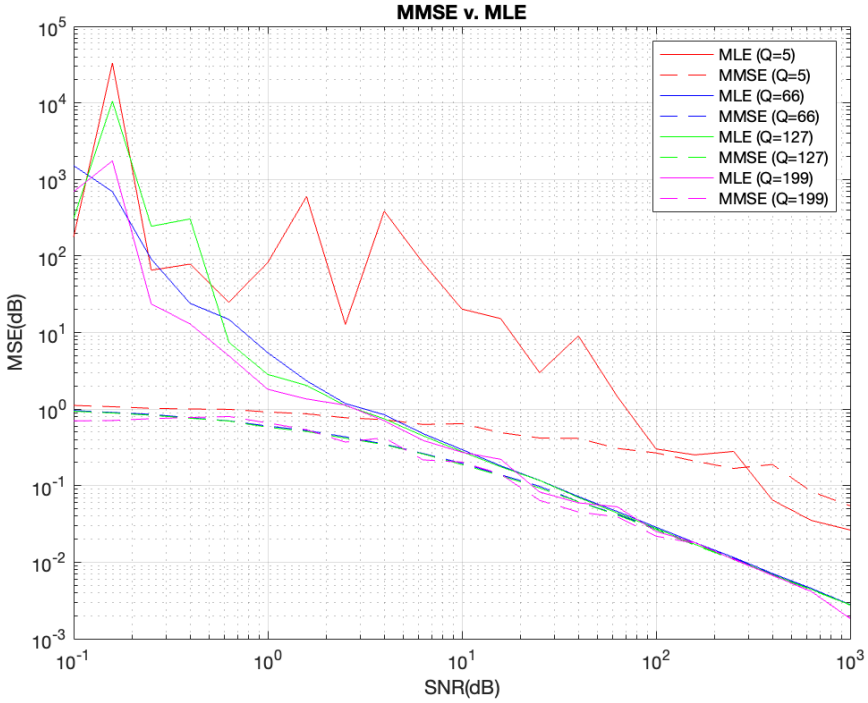


Figure 4.1: MMSE v. MLE

That is why, considering low SNR, it can be said that MMSE performs better than MLE.

It has to be noted that in all the simulations, the Monte Carlo Method is used to decrease the randomness caused by the noise, hence, to smooth the graphs.

## 4.3 Loss Function v. Pilot Samples

The Loss Function is defined as follows:

$$\mathbb{Q}(Q) = k(MSE_{MMSE}) + \frac{Q}{2} \tag{4.6}$$

with k as normalization factor.

With high SNR the choice of low Q is suitable. With low SNR, to estimate the channel in a proper way a big number Q of pilots is adopted. It is a trade off between the estimation of the channel response and the transmission of actual information.

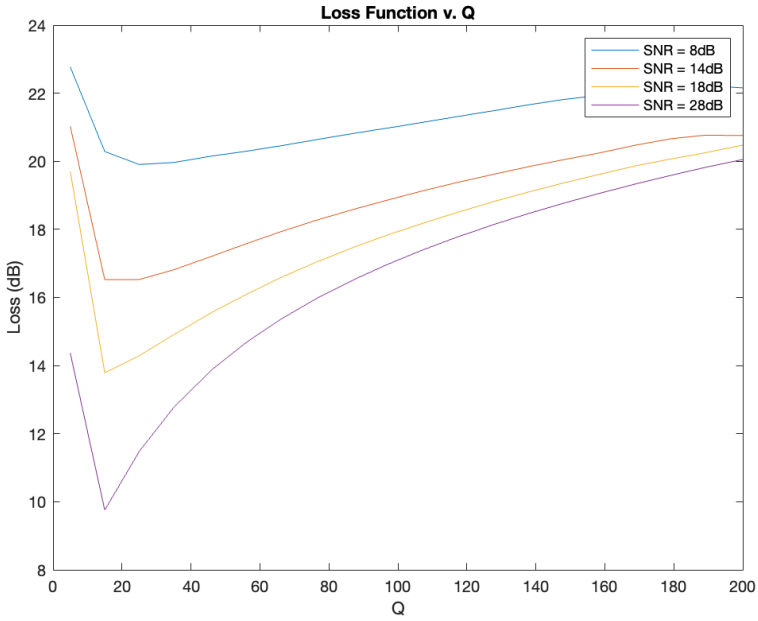


Figure 4.2: MMSE v. MLE

# Chapter 5

# Conclusion

Kronecker tensor product is only a possible choice to solve the problem.
Several loops can be used to iterate through K dimensions and Q pilot samples, but the approach here is more general, since it is suitable for higher matrix dimensions. Kronecker product lowers computational costs, which are huge also for a $4 \times 4$ MIMO system without memory.

# Chapter 6

# Appendix With Code

```
%% 1.1 - Noise Generation
M = 4;
MC = 10; % Montecarlo Iterations
MSE_H = zeros(10,10,MC);
for iter = 1:MC % iterate through Montecarlo
    i = 1;
    u = randn([M,200]); % returns a Mx200 matrix containing "random" values from normal distrib
    for L = floor(linspace(10,200,10)) % iterate through L
        k = 1;
        for rho = linspace(0, 0.99,10) %iterate through values of rho
            Cw_true = toeplitz([1,rho,rho,rho]);
            % true Covariance Matrix, it makes like sum of diags in a toeplitz form
            w = chol(Cw_true, 'lower')*u(:,1:L);
            % noise, chol gives a lower triangular matrix drawn from Cw_true
            Cw_est = cov(w.');
            % estimate of the Covariance Matrix (we put the transpose since for cov each column is a variable, each
            MSE_H(k, i, iter) = mean(mean(Cw_true - Cw_est).^2); % .^ since element wise,
            mean of mean compute the mean of the vector of the mean
            k = k + 1; % to increment index associated to rho
        end
        i = i + 1; % to increment index associated to L
    end
end
MSE_H = mean(MSE_H,3); % mean for each combination of x and y of MSE
figure()
bar3(MSE_H);xlabel('L');ylabel('\rho');zlabel(' MSE') % 3D PLOT
figure()
for l = 1:10 % iterate through L, to have different lines of MSE v. rho wrt L
    plot(linspace(0, 0.99, 10), MSE_H(:,l))
    hold on
end
```

```
title('MSE v \rho')
figure()
for rho = 1:10 % iterate through rho, to have different lines of MSE v. L wrt rho
plot(floor(linspace(10,200,10)), MSE_H(rho, :))
hold on;
end
title('MSE v L')
%% 1.2(i) - MIMO Estimation: Memoryless Filter
MC = 25; % montecarlo
K = 1; % memoryless
M = 4; % first dimensions
N = 4; % second dimensions
SNR = -10:2:30; % snr in db
SNR_lin = 10.^(SNR/10); % snr linear
sigma_x = 1;
rho = 0.1;
C_true = toeplitz([1, rho, rho, rho]);
MLE = zeros(21, 5, 5, MC);
CRB = zeros(21,5,5,MC);
for iter = 1:MC %Montecarlo iterations
    sigma_index = 1;
    for sigma_w = 1./SNR_lin
        C = sigma_w*C_true;
        u_all = randn(M,200);
        w_all = chol(C, 'lower')*u_all;
        alpha_index = 1;
        for alpha = linspace(0, 0.99, 5)
            x_all = sigma_x*randn(N,200);
            h = toeplitz([alpha^0, alpha^1, alpha^2, alpha^3]);
            h_reshape = reshape(h,1,[])';
            i_q = 1;
            for Q = floor(linspace(5,50,5))
                w = w_all(:,1:Q);
                x = x_all(:,1:Q);
                x_kron = kron(eye(N), x');
                C_kron = kron(eye(Q), C);
                w_kron = reshape(w.', 1, [])';
                y_kron = x_kron*h_reshape + w_kron;
                h_est_mle = inv(x_kron.'*inv(C_kron)*x_kron)*x_kron.'*inv(C_kron)*y_kron;
                CRB(sigma_index, i_q, alpha_index, iter) = trace(inv(x_kron.'*inv(C_kron)*x_kron))/16;
                MLE(sigma_index, i_q, alpha_index, iter) = mean((h_reshape - h_est_mle).^2);
                i_q = i_q + 1;
            end
            alpha_index = alpha_index + 1;
        end
        sigma_index = sigma_index + 1;
    end
end
MLE = mean(MLE, 4);
CRB = mean(CRB, 4);
%PLOTS
```

```
figure()
alpha_plot = 5;
cmap = hsv(5);
for i = 1:5
    loglog(SNR_lin, (MLE(:, i, alpha_plot)), 'color', cmap(i,:))
    hold on;
    loglog(SNR_lin, (CRB(:, i , alpha_plot)), '--', 'color', cmap(i,:))
end
hold off;
title('MSE v. SNR (\alpha=0.99, Q varies)')
figure()
Q_plot = 5;
alpha_str = linspace(0, 0.99, 5);
for i = 1:5
    loglog(SNR_lin, (MLE(:,Q_plot, i)), 'color', cmap(i,:))
    hold on;
    loglog(SNR_lin, CRB(:, Q_plot, i), '--', 'color', cmap(i,:))
end
hold off;
title('MSE v. SNR (Q=50, \alpha varies)')
%% 1.2(ii) - MIMO Estimation: Memory Filter
MC = 25; % Montecarlo Iterations
K = 4;
M = 4;
N = 4;
SNR = -10:2:30;
SNR_lin = 10.^(SNR/10);
rho = 0.1;
beta = [0.9, 0.5, 0.1];
sigma_x = 1;
MLE = zeros(21,5,5,3,MC);
CRB = zeros(21,5,5,3,MC);
for iter = 1:MC
    sigma_i = 1; % index for sigma
    for sigma_w = 1./SNR_lin
        alpha_i = 1; % index for alpha
        C_true = toeplitz([1, rho, rho, rho]);
        C = sigma_w*C_true;
        for alpha = linspace(0, .99, 5)
            for beta_i = 1:3
                h = zeros(M,N,K);
                for i = 1:K
                    h(:,:,i) = (beta(beta_i)^(i-1))*toeplitz([alpha^0, alpha^1, alpha^2, alpha^3]);
                end
                h_reshape = reshape(h, 1, [])';
                q = 1;
                for Q = floor(linspace(15,50,5))
                    u = randn(M*(Q+K-1),1);
                    C_kron = kron(C,eye(Q+K-1));
                    w_kron = chol(C_kron, 'lower')*u;
                    x_mem = zeros(Q+K-1, N*K);
```

```
                    for n = 1:N
                        x = sigma_x*randn(Q,N);
                        for i = 1:K
                            x_mem(i:Q+i-1, i+N*(n-1)) = x(:,n);
                        end
                    end
                    x_kron = kron(eye(4), x_mem);
                    y = x_kron*h_reshape + w_kron;
                    h_est_mle = inv(x_kron'*inv(C_kron)*x_kron)*x_kron'*inv(C_kron)*y; % MLE est
                    MLE(sigma_i, q, alpha_i, beta_i,iter) = mean((h_reshape - h_est_mle).^2);
                    CRB(sigma_i, q, alpha_i, beta_i,iter) = trace(inv(x_kron.'*inv(C_kron)*x_kron))/64;
                    q = q + 1;
                end
            end
            alpha_i = alpha_i + 1;
        end
        sigma_i = sigma_i + 1;
    end
end
MLE = mean(MLE, 5);
CRB = mean(CRB, 5);
%PLOTS
figure()
Q_plot = 5;
alpha_plot = 5;
cmap = hsv(3);
for i = 1:3
    loglog(SNR_lin, (MLE(:, Q_plot, alpha_plot, i)), 'color', cmap(i,:))
    hold on;
    loglog(SNR_lin, (CRB(:, Q_plot, alpha_plot, i)),'--', 'color', cmap(i,:))
end
hold off;
title('MSE v SNR -- Q=50, \alpha=0.99')
figure()
Q_plot = 5;
beta_plot = 1;
cmap = hsv(5);
for i = 1:5
    loglog(SNR_lin, (MLE(:,Q_plot, i, beta_plot)), 'color', cmap(i, :))
    hold on;
    loglog(SNR_lin, (CRB(:,Q_plot, i, beta_plot)), '--', 'color', cmap(i, :))
end
hold off;
title('MSE v SNR -- Q=50, \beta=0.9')
figure()
alpha_plot = 5;
beta_plot = 1;
cmap = hsv(5);
for i = 1:5
    loglog(SNR_lin, (MLE(:, i, alpha_plot, beta_plot)), 'color', cmap(i,:))
    hold on;
```

```
    loglog(SNR_lin, (CRB(:, i, alpha_plot, beta_plot)), '--', 'color', cmap(i, :))
end
hold off;
title('MSE v SNR --- \alpha=0.99, \beta=0.9')
%% 1.3 - MIMO Deconvolution
MC = 25; % Montecarlo Iterations
K = 1; % Memoryless (it's like the third dimensions)
M = 4; % first dimensions of filter response matrix
N = 4; % second dimension of filter response matrix
SNR = -10:2:30; % SNR in dB (21 values)
SNR_lin = 10.^(SNR/10); % SNR in linear
sigma_x = 1; % we put it to 1 for simplicity
rho = 0.1; % value wanted by professor
alpha = 0.5; % value wanted by professor (coupling effect)
h = toeplitz([alpha^0, alpha^1, alpha^2, alpha^3]);
h_reshape = reshape(h, 1, [])'; % it puts the matrix h in a vector, column vector since it is transpose
C_true = toeplitz([1, rho, rho, rho]);
MLE = zeros(21,20, MC);
MMSE = zeros(21,20,MC);
MLE_H = zeros(21,20, MC);
MSE_H = zeros(21,20,MC);
for iter = 1:MC
    sigma_index = 1;
    for sigma_w = 1./SNR_lin % by definition
        C = sigma_w*C_true;
        u_al = randn(M,200);
        w_al = chol(C,'lower')*u_al;
        x_al = sigma_x*randn(N,200);
        y_al = h*x_al + w_al;
        q_index = 1;
        for Q = floor(linspace(5, 199, 20))
        %number of pilots samples known both at the receiver and the transmitter
            w = w_al(:, 1:Q);
            x = x_al(:, 1:Q); % used as pilot
            x_rest = x_al(:, Q+1:end); % in the end there is always at least one information sample
            x_kron = kron(eye(N), x'); % to reshape it
            C_kron = kron(eye(Q),C); % reshape of sigma_w*C_true
            C_eq_h = kron(eye(N), C);
            w_kron = reshape(w.', 1, [])';
            y_kron = x_kron*h_reshape + w_kron;
            C_x = cov(x');
            h_est_mle = inv(x_kron'*inv(C_kron)*x_kron)*x_kron'*inv(C_kron)*y_kron;
            y_rest = y_al(:,Q+1:end);
            h_rest = reshape(h_est_mle, 4, 4);
            x_est_mle = inv(h_rest'*inv(C)*h_rest)*h_rest'*inv(C)*y_rest; % mle formula
            x_est_mmse = inv(h_rest'*inv(C)*h_rest + inv(C_x))*h_rest'*inv(C)*y_rest; % mmse formula

            MLE(sigma_index, q_index, iter) = mean(mean((x_rest - x_est_mle).^2));
            MMSE(sigma_index, q_index, iter) = mean(mean((x_rest - x_est_mmse).^2));
            MLE_H(sigma_index, q_index, iter) = mean((h_est_mle - h_reshape).^2);
            q_index = q_index + 1; % iterate through q pilots
```

```
        end
        sigma_index = sigma_index + 1; %iterate in 1/SNR
    end
end
%to mean through iter
MLE = mean(MLE, 3);
MMSE = mean(MMSE, 3);
MLE_H = mean(MLE_H, 3);
%METRIC FOR Q
cost = zeros(21,20); % cost function dimension
P = 200; %length of input signal, when P = Q we are transmitting only pilot samples
sigma_index = 1; % index for sigma
MSE_dB = 10*log10(MMSE);
normal = mean(floor(linspace(5, P, 20)))/(mean(mean(MMSE))); % normalization
for sigma_tx = SNR_lin
    q_index = 1;
    signal_p = sigma_tx * randn(4,P);
    power_p = 10*log10(mean(mean(signal_p.^2)));
    for Q = floor(linspace(5, P, 20))
        signal_q = signal_p(:,1:Q);
        signal_info = signal_p(:,Q+1:end);
        power_q = mean(mean(signal_q.^2));
        power_info = mean(mean(signal_info.^2));
        cost(sigma_index, q_index) = normal*MMSE(sigma_index, q_index) + 0.5*Q;
        q_index = q_index + 1;
    end
    sigma_index = sigma_index + 1; %increment of index of sigma
end
Q = floor(linspace(5, P, 20));
%PLOTS
figure()
for i = [10,13,15,20]
    plot(Q, 10*log10((cost(i,:))))
    hold on;
end
title('Loss Function v. Q')
figure()
color_i = 1;
color = 'rbgm';
for i = floor(linspace(1,20,4))
    loglog(SNR_lin, (MLE(:,i)), 'color', color(color_i))
    hold on;
    loglog(SNR_lin, (MMSE(:,i)), '--', 'color', color(color_i))
    color_i = color_i + 1;
end
hold off;
title('MMSE v. MLE')
```