MATLAB Simulation Exercise

Statistical Signal Processing II

ESTIMATOR CORRELATOR

Group 13

Athmajan Vivekananthan

University of Oulu
Finland

March 1, 2024

1 Background

Neyman-Pearson detector can be used to find if a signal is present or not. The theretical background of this laboratory experiment covers an extensive study on how to derive such detectors. These detectors can sometimes be designed as energy detectors when the signal is random which follows a certain distribution.

This detector compares uses the results of the likelihood ratios observing the received signal under the hypothesis that the signal is present (signal-plus-noise hypothesis) to the likelihood under the hypothesis that only noise is present (noise-only hypothesis). In this task the detection is done based on detecting the energy of the signal. Hence the name, energy detector.

In practice, the performance of such detectors is often evaluated using probability of detection curves across various signal-to-noise ratio (SNR) levels. These curves depict the probability of correctly detecting the signal as a function of SNR, providing insights into the detector's performance under different noise conditions. By analyzing these curves, the report has been presented with various curves that are discussed in the next section. [1] [2]

2 Results and Discussion

Results for the given tasks are presented below as Probability of Detection curves.

2.1 Task 1 (a)

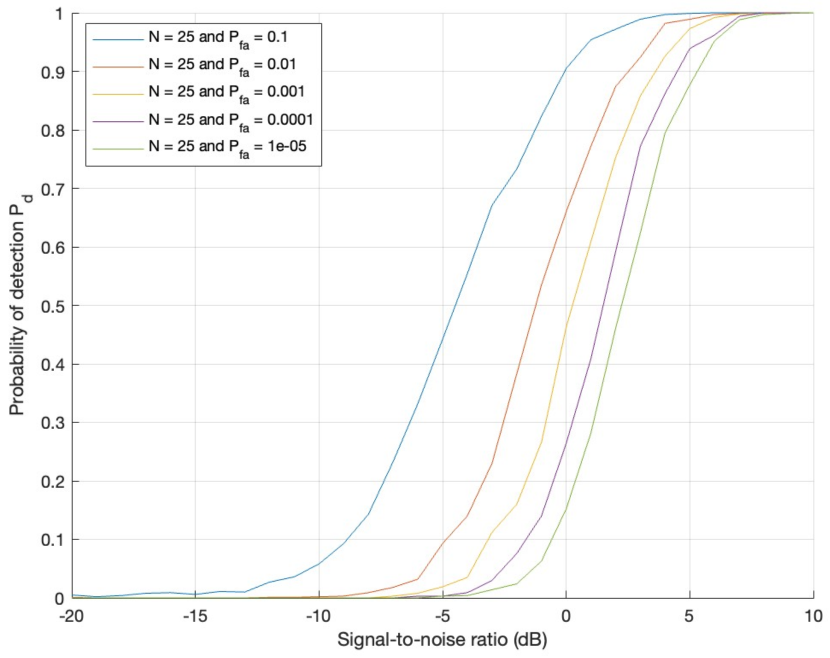

Figure 1: Energy Detector Performance for $N = 25$

2.2 Task 1 (b)

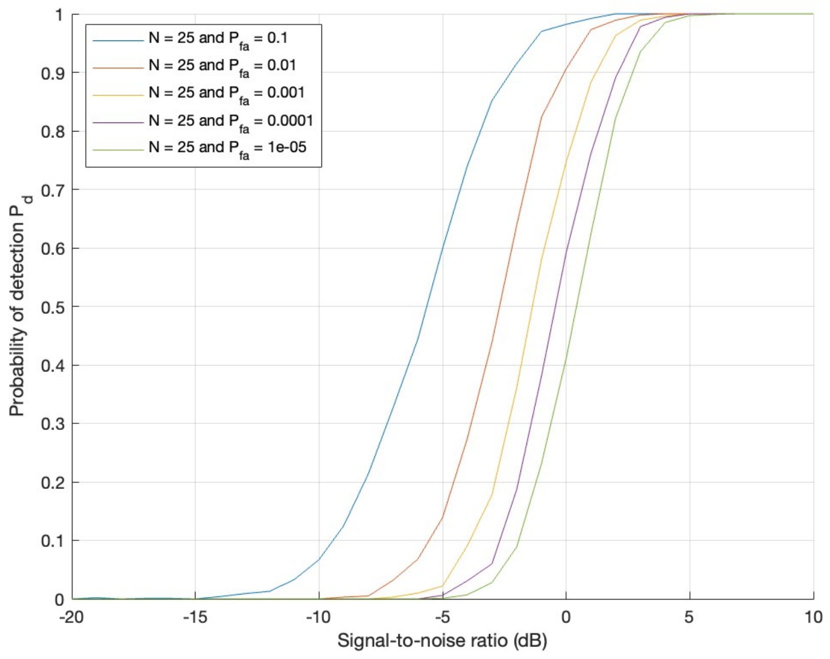

Figure 2: Energy Detector Performance for $N = 50$

2.3 Comparison between $N = 25$ and $N = 50$

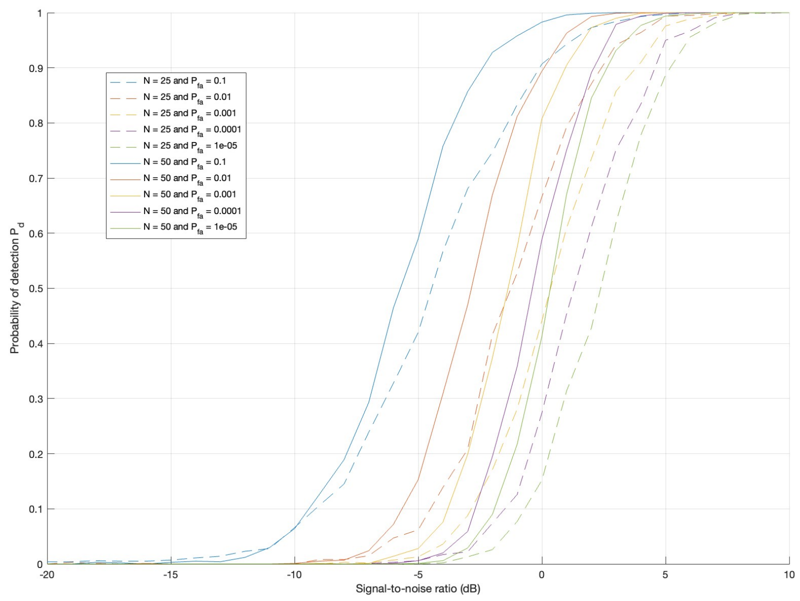

Figure 3: Energy Detector Performance for $N = 20$ and $N = 50$

Figure 3 shows the performance of the energy detector for both cases. It can be clearly seen that the performance of detection has been improved when the number of samples has been increased from 25 to 50 for the given probability of false alarm.

2.4 Task 2

Now the detection problem has been generalized to a linear model. A known channel matrix M has been used to drive the linearization of the system where the input data d is convoluted with the channel M.

The covariance of the signal $s = Hd$ can be shown in Equation 1.

$$\begin{aligned}
s &= Hd \\
E[s] &= E[Hd] \\
&= E[H] * E[d] \\
&= 0 \\
\mathrm{cov}(S) &= E[SS^T] \\
&= E[(Hd)(Hd)^T] \\
&= E[Hdd^T H^T] \\
&= E[\sigma_s^2 HH^T] \\
&= \sigma_s^2 E[HH^T] \\
&= \sigma_s^2 HH^T
\end{aligned} \tag{1}$$

To derive the performance metrics, the test statistics of the estimator correlator can be shown to be as derived in Equation 2 [1]. Where C_d is the covariance matrix of the input data and H is the channel matrix σ^2 is the variance of the WGN. Since the test metric is depended as a function of HH^T even though it has been given that H is a matrix of size $N \times 2$ the square matrix HH^T of size $N \times N$ decides $T(x)$. Therefore in the Matlab code an invertible square matrix which is assumed to be known has been generated to drive the detector.

$$T(x) = x^T HC_d H^T (HC_d H^T + \sigma^2 I)^{-1} x \tag{2}$$

Figures 4, 5 and 6 show the performance of the detector for changes in number of samples. It can be observed that similar to the observations done in Task 1, increasing the number of samples has a positive effect on the performance where as the linearization channel matrix has a huge impact of bringing the whole performance down by several percentages on the detection probability.

This effect can be further even observed when the channel matrix tap coefficients are uncorrelated from each other as seen in Figure 7. This effect can be done by setting the HH^T to be an identity matrix. When the channel tap coefficients become uncorrelated with each other the detection improves greatly.

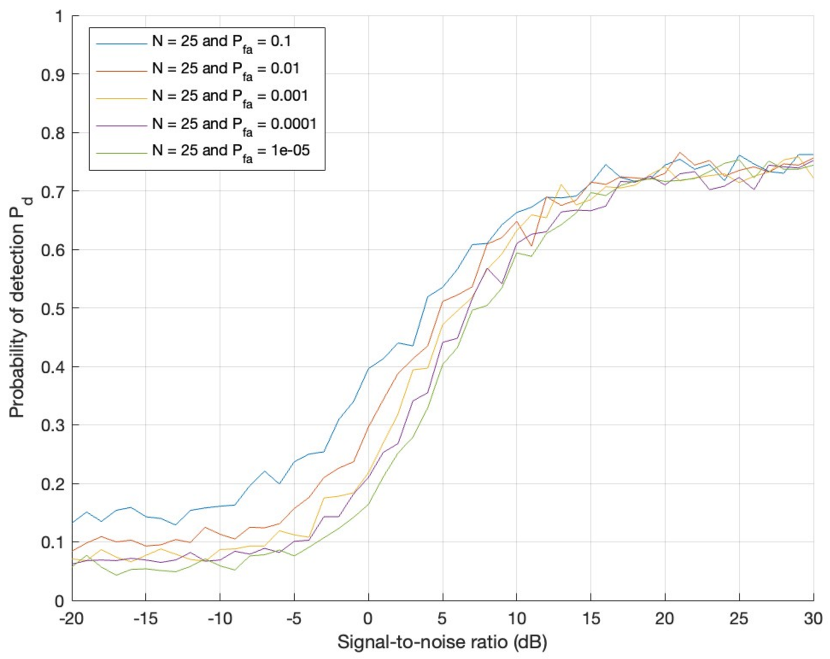

Figure 4: Linear Model Detector Performance for $N = 25$

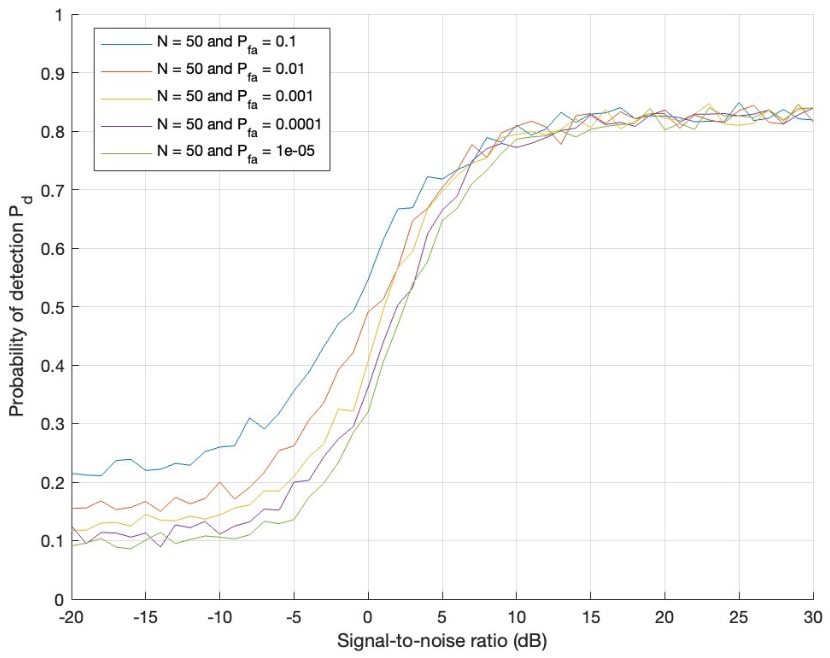

Figure 5: Linear Model Detector Performance for $N = 55$

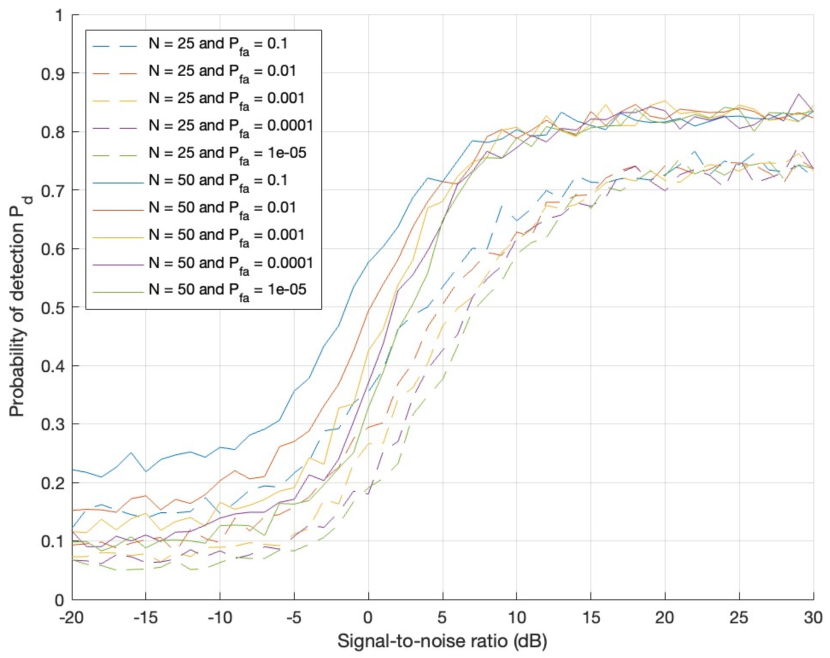

Figure 6: Linear Model Detector Performance for $N = 20$ and $N = 50$

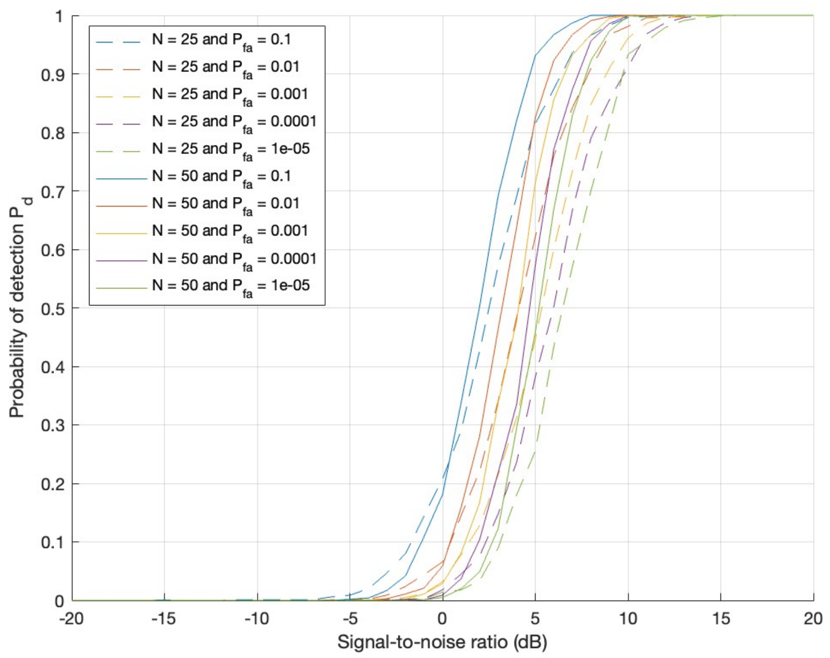

Figure 7: Linear Model Detector Performance for Uncorrelated Taps for $N = 20$ and $N = 50$

3 Matlab Code for Task 1

```
%% Lab 6
clear;clc;close all

figure
Ntrial = 1000;
Nsample = 25;
SNR_dB = -20:1:10;
Pfa = [1e-1,1e-2,1e-3,1e-4,1e-5];
num_colors = length(Pfa);

% Generate a color palette
color_palette = lines(num_colors);

grid on
hold on
xlabel('Signal-to-noise ratio (dB)');
ylabel('Probability of detection P_d');
ylim([0 1]);
for j=1:length(Pfa)
    for i=1:length(SNR_dB)
        SNR = 10.^(SNR_dB(i)/10);
        a = chi2inv(1-Pfa(j),Nsample);
        Pd(i,j) = 0;
        for itrial = 1:Ntrial
            %% signal generation            SNR = sigma_s^2/sigma^2
            s = sqrt(SNR) * randn(Nsample,1);
            w = sqrt(1) * randn(Nsample,1);
            x = s + w;

            % Since M is known matrix.
            % Generatng an invertible MM' square matrix
            %MMt = generate_invertible_matrix(Nsample);

            %% detection
            % calculate Tx
            Tx = sum(x.^2);

            % determine threshold gamma''
            threshold_pp = a*var(w);


            if Tx > threshold_pp
                Pd(i,j) = Pd(i,j) + 1/Ntrial;
            end
        end
    end
    plot(SNR_dB, Pd(:,j) ,'Color', color_palette(j, :),
    'DisplayName', ['N = 25 and P_{fa} = ', num2str(Pfa(j))]);
end



legend('show');


function A = generate_invertible_matrix(n)
    % Initialize a flag to check if the matrix is invertible
    invertible = false;

    while ~invertible
        % Generate a random square matrix of size n x n
        A = rand(n);

        % Check if the matrix is invertible (i.e., determinant is non-zero)
        if det(A) ~= 0
            invertible = true;
        end
    end
end
```

4 Matlab Code for Task 2

```
%% Lab 6
clear;clc;close all

figure
Ntrial = 1000;
Nsample = 25;
SNR_dB = -20:1:20;
Pfa = [1e-1,1e-2,1e-3,1e-4,1e-5];
num_colors = length(Pfa);

% Generate a color palette
color_palette = lines(num_colors);

grid on
hold on
xlabel('Signal-to-noise ratio (dB)');
ylabel('Probability of detection P_d');
ylim([0 1]);
for j=1:length(Pfa)
    for i=1:length(SNR_dB)
        SNR = 10.^(SNR_dB(i)/10);
        a = chi2inv(1-Pfa(j),Nsample);
        Pd(i,j) = 0;
        for itrial = 1:Ntrial
            %% signal generation            SNR = sigma_s^2/sigma^2
            s = sqrt(SNR) * randn(Nsample,1);
            w = sqrt(1) * randn(Nsample,1);
            d = sqrt(1) * randn(Nsample,1);
            Sigma_s_sqr = var(d);
            sigma_sqr = var(w);
            x = s + w;

            % Since H is known matrix.
            % Generatng an invertible HH' square matrix
            %HHt = generate_invertible_matrix(Nsample);
            HHt = eye(Nsample);

            %% detection
            % calculate Tx
            invPart = inv(Sigma_s_sqr*HHt + sigma_sqr*eye(Nsample));

            Tx = x'*Sigma_s_sqr*HHt*invPart*x;
            % determine threshold gamma''
            threshold_pp = a*var(Tx);


            if Tx > threshold_pp
                Pd(i,j) = Pd(i,j) + 1/Ntrial;
            end
        end
    end
    plot(SNR_dB, Pd(:,j) ,'--','Color', color_palette(j, :),
    'DisplayName', ['N = 25 and P_{fa} = ', num2str(Pfa(j))]);
end

hold on
Nsample = 50;

xlabel('Signal-to-noise ratio (dB)');
ylabel('Probability of detection P_d');
ylim([0 1]);
for j=1:length(Pfa)
    for i=1:length(SNR_dB)
        SNR = 10.^(SNR_dB(i)/10);
        a = chi2inv(1-Pfa(j),Nsample);
        Pd(i,j) = 0;
        for itrial = 1:Ntrial
            %% signal generation            SNR = sigma_s^2/sigma^2
            s = sqrt(SNR) * randn(Nsample,1);
            w = sqrt(1) * randn(Nsample,1);
            d = sqrt(1) * randn(Nsample,1);
            Sigma_s_sqr = var(d);
            sigma_sqr = var(w);
            x = s + w;

            % Since H is known matrix.
            % Generatng an invertible HH' square matrix
            %HHt = generate_invertible_matrix(Nsample);
            HHt = eye(Nsample);

            %% detection
            % calculate Tx
            invPart = inv(Sigma_s_sqr*HHt + sigma_sqr*eye(Nsample));

            Tx = x'*Sigma_s_sqr*HHt*invPart*x;
            % determine threshold gamma''
            threshold_pp = a*var(Tx);


            if Tx > threshold_pp
                Pd(i,j) = Pd(i,j) + 1/Ntrial;
            end
        end
    end
    plot(SNR_dB, Pd(:,j) ,'Color', color_palette(j, :),
    'DisplayName', ['N = 50 and P_{fa} = ', num2str(Pfa(j))]);
end



legend('show');


function A = generate_invertible_matrix(n)
    % Initialize a flag to check if the matrix is invertible
    invertible = false;

    while ~invertible
        % Generate a random square matrix of size n x n
        A = rand(n);

        % Check if the matrix is invertible (i.e., determinant is non-zero)
        if det(A) ~= 0
            invertible = true;
        end
    end
end
```

References

[1] S. M. Kay, *Fundamentals of Statistical Signal Processing: Estimation Theory.* Prentice Hall International, 1993.

[2] S. Haykin, *Adaptive Filter Theory.* Pearson, 2014. [Online]. Available: https://books.google.fi/books?id=J4GRKQEACAAJ