Methods

8 – 1

For problem 8 – 1, check the former PDF to see the deduction.

We got that

$$\ln\lambda = z - \frac{1}{2\sigma^2}\sum_i\sum_j s_{ij}^2$$

Which means that $\ln\lambda(z)$ is a function of z.

8 – 2(a)

For the first part of the problem, it's required to use brute-force to calculate the $f(\ln\lambda \mid H_0)$ and $f(\ln\lambda \mid H_1)$ by generate 500 Gaussian matrices for each $\frac{E_s}{\sigma_n^2}$ so that the set of $\ln\lambda$ could be an approximation of the real conditional p.d.f. of both $f(\ln\lambda \mid H)$.

Then we have a set of $\ln\lambda$ under the condition of H_0 and another under H_1. Therefore, we can separately calculate the P_D and P_F with a selection of β in the likelihood ratio space. In my program, I selected 1000 β s uniformly in the range of the maximum and minimum of both the $\ln\lambda$ corresponding to $f(\ln\lambda \mid H_0)$ and $f(\ln\lambda \mid H_1)$. Then I calculated the sum of those frequency numbers to obtain the brute-force probability of P_D and P_F pairs. After obtaining those points, I've plotted the figure 1 below.

8 – 2(b)

For the second part of the problem, I deducted the theoretical expressions of $f(\ln\lambda \mid H_0)$ and $f(\ln\lambda \mid H_1)$ first.

$$f(\ln\lambda \mid H_0) = \frac{1}{\sqrt{2\pi D(z)}}\exp\left(-\frac{(z - D(z))^2}{2D(z)}\right)$$

$$f(\ln\lambda \mid H_1) = \frac{1}{\sqrt{2\pi D(z)}}\exp\left(-\frac{(z - 2D(z))^2}{2D(z)}\right)$$

Where $\mathrm{D}(z)=\frac{1}{\sigma^2}\sum_i\sum_j s_{ij}^2$

Then I generated a set of z which ranges between $[-50,50]$ and has an interval of 0.01, so that I can plot a rather smooth curve without losing too many data at both of the furthest ends of the curve. After that, I used numeric integral method to calculate the probability with another uniformly distributed beta, to draw the theoretical curves, shown in figure 2 below.

Finally I plotted the curves in a superimposing method to compare the simulation and the theoretical deduction. Figure 3 shown below is the result.

Analysis and discussion

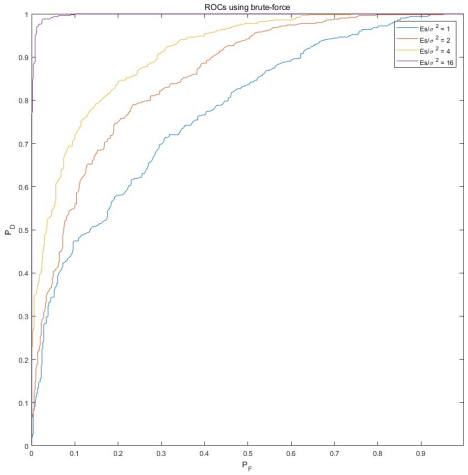

Figure 1 Simulation result of ROCs using brute-force

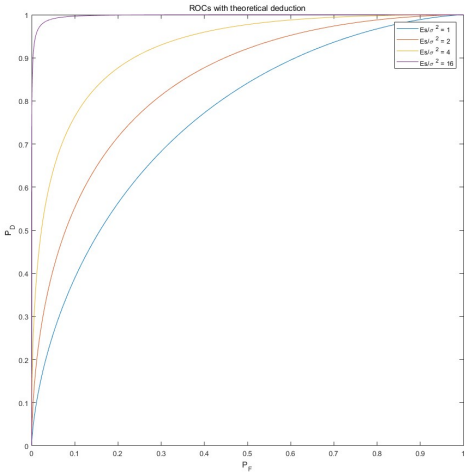

Figure 2 Theoretical result of ROCs

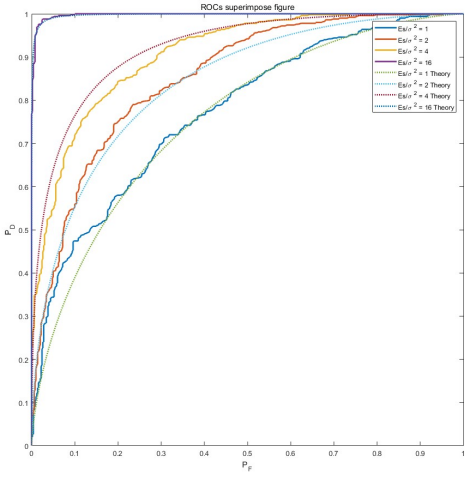

Figure 3 Superimposing curves of ROCs

While it might be too small to see the figures, you can enlarge it with 'zoom' in the PDF reader. I did not adjust the output linewidth and font size of the figures.

First of all, we can see that the theoretical and simulated results match with each other. That means the theoretical deduction is correct and we can use it to decide our detection threshold.

Second, considering the former homework, we can arbitrarily say that when $\frac{E_s}{\sigma_n^2}$ is larger (the curve more to the up left corner), we can always have an easier decision, or a lower error rate with the correct judgement threshold. The ROC curves can also show that: the greater the area under the curve, the lower error rate. Also, the slope of the tangent line at a cut point gives the likelihood ratio of at that point.

Finally, we cannot say that we will always detect better with our developed criteria comparing with the 'eye detection' in homework 1, since we need to set a certain error rate standard to judge whether it's a good detection. However, it's always easier to have a practical standard other than just saying 'I can see the line in the noise!'

If you can patiently read through the text toward here, could you please give me a full score…? (*/ω＼*)

Source Code

```
clear;clc;close all;

sigma = 1; % so sigma^2 also = 1;
S = diag(ones(1,1024));
%% 1
```

```
% H0 = normrnd(0,sigma^2,1024,1024); % noise only
% Es1 = 4;
% scale1 = sqrt(Es1/1024);
% S1 = S.*scale1;

% H1 = S0 + H0;

%% 2
% plot the ROC

Es = zeros(1,4);

Es(1) = 1 * sigma^2;
Es(2) = 2 * sigma^2;
Es(3) = 4 * sigma^2;
Es(4) = 16 * sigma^2;
iterate_times = 500;
ln_l0 = zeros(iterate_times, 1);
ln_l1 = zeros(iterate_times, 1);
PfPdpoints = zeros(4,1000,2);


% h = 1; % for testing
for h = 1 : 4
    for i = 1:iterate_times
        % generate a different noise matrix each time
        H2_0 = normrnd(0,sigma^2,1024,1024);
        S2 = sqrt(Es(h)/1024) .* S;
        % H2_1 = S2 + H2_0;

        % calculate f(ln(lambda)|H0)
        ln_lambda0 = 1/sigma^2 * sum(sum(H2_0 .* S2 - 1/2 .* S2.^2));
        ln_l0(i) = ln_lambda0;
        % calculate f(ln(lambda)|H1)
        % regenerate the Gaussian matrix for H1
        H2_0 = normrnd(0,sigma^2,1024,1024);
        ln_lambda1 = 1/sigma^2 * sum(sum(H2_0 .* S2 + 1/2 .* S2.^2));
        ln_l1(i) = ln_lambda1;
    end

    max_ln_l0 = max(ln_l0);
    max_ln_l1 = max(ln_l1);
    min_ln_l0 = min(ln_l0);
    min_ln_l1 = min(ln_l1);
```

```
    % to sample beta from the range of min and max of ln_lambda
    start = min(min_ln_l0, min_ln_l1);
    stop = max(max_ln_l0, max_ln_l1);
    interval = (stop - start) / 1000;
    for i = linspace(1, 1000, 1000)
        % the probability is the count for the matrix between and beside the threshold
        % beta = i / (1000* min(max_ln_l0, max_ln_l1));
        beta = start + (i * interval);
        Pf_sum_l0 = length(ln_l0(ln_l0 > beta));
        Pd_sum_l1 = length(ln_l1(ln_l1 > beta));
        PfPdpoints(h,i,1) = Pf_sum_l0/iterate_times;
        PfPdpoints(h,i,2) = Pd_sum_l1/iterate_times;
    end

end

%% 3
% theoretically calculate the corresponding values and curves
% generate a different noise matrix each time
Es(1) = 1 * sigma^2;
Es(2) = 2 * sigma^2;
Es(3) = 4 * sigma^2;
Es(4) = 16 * sigma^2;

interval3 = 0.01;
z = -50: interval3: 50;
len_z = length(z);
PfPdpoints_theoretical = zeros(4,len_z,2);
for h = 1 : 4
    S3 = sqrt(Es(h)/1024) .* S;
    D_z = (1/sigma^2) * sum(sum(S3.^2));
    % for convenience, I just used the same range of Q#2 to do the
    % theoretical calculation.
    % But after trying, I found it too small. Change it?
    % z = start : interval : stop;

    % calculate f(ln(lambda)|H0), f_H0 has the same length as z
    f_H0 = 1/sqrt(2*pi*D_z) .* exp(-(((z-D_z).^2)./(2*D_z)));
    % calculate f(ln(lambda)|H1)
    f_H1 = 1/sqrt(2*pi*D_z) .* exp(-(((z-2*D_z).^2)./(2*D_z)));

    for i = 1:len_z - 1
        % the probability is the count for the matrix between and beside the threshold
```

```
        % beta = i / (1000* min(max_ln_l0, max_ln_l1));
        beta = -50 + (i * interval3);
        % integrate the f to obtain the possibility
        x_int_range = -50:interval3:beta;

        Pf_integral_l0 = trapz(x_int_range, f_H0(1:i+1));
        Pd_integral_l1 = trapz(x_int_range, f_H1(1:i+1));
        PfPdpoints_theoretical(h,i,1) = 1 - Pf_integral_l0;
        PfPdpoints_theoretical(h,i,2) = 1 - Pd_integral_l1;
    end

end

%% 3.2 superimpose the figures
% I put all the figures into one section so that I do not need to re run
% the program to update each figure, just run this section.
figure(1);
hold on;
for h = 1:4
    plot(PfPdpoints(h,:,1),PfPdpoints(h,:,2));
end
axis([0 1 0 1]);axis equal square tight; box on;
title('ROCs using brute-force');
xlabel('P_F');ylabel('P_D');
legend(['Es/\sigma ^2 = ' num2str(Es(1))], ['Es/\sigma ^2 = ' num2str(Es(2))], ['Es/\sigma ^2
= ' num2str(Es(3))], ['Es/\sigma ^2 = ' num2str(Es(4))],'Location','northeast');


figure(2);
hold on;
for h = 1:4
    plot(PfPdpoints_theoretical(h,1:len_z - 1,1),PfPdpoints_theoretical(h,1:len_z - 1,2));
end
axis([0 1 0 1]);axis equal square tight; box on;
title('ROCs with theoretical deduction');
xlabel('P_F');ylabel('P_D');
legend(['Es/\sigma ^2 = ' num2str(Es(1))], ['Es/\sigma ^2 = ' num2str(Es(2))], ['Es/\sigma ^2
= ' num2str(Es(3))], ['Es/\sigma ^2 = ' num2str(Es(4))],'Location','northeast');

figure(3);
hold on;
for h = 1:4
    plot(PfPdpoints(h,:,1),PfPdpoints(h,:,2), 'linewidth', 2);
end
```

```
for h = 1:4
   plot(PfPdpoints_theoretical(h,1:len_z - 1,1),PfPdpoints_theoretical(h,1:len_z - 1,2),':',
'linewidth', 2);
end
axis([0 1 0 1]);axis equal square tight; box on;
title('ROCs superimpose figure');
xlabel('P_F');ylabel('P_D');

legend(['Es/\sigma ^2 = ' num2str(Es(1))], ['Es/\sigma ^2 = ' num2str(Es(2))], ['Es/\sigma ^2
= ' num2str(Es(3))], ['Es/\sigma ^2 = ' num2str(Es(4))], ['Es/\sigma ^2 = ' num2str(Es(1)) '
Theory'], ['Es/\sigma ^2 = ' num2str(Es(2)) ' Theory'], ['Es/\sigma ^2 = ' num2str(Es(3)) '
Theory'], ['Es/\sigma ^2 = ' num2str(Es(4)) ' Theory'],'Location','northeast');
```