Boston University

Department of Electrical and Computer Engineering

EC505 STOCHASTIC PROCESSES

Problem Set No. 8

Fall 2016

Issued: Monday, Nov. 14, 2016 **Due:** Monday, Nov. 21, 2016

Problem 8.1

Let X and Y be jointly Gaussian random variables, with $E(X) = 4$, $E(Y) = 2$, $\text{Var}(X) = 4$, $\text{Var}(Y) = 8$, $\text{Cov}(X,Y) = 4$.

(a) Determine $\hat{x}_{BLS}(y)$, the Bayes least-squares estimate of X based on observation of Y, $\lambda_{X|Y}(y)$ the conditional covariance, and λ_{BLS_X}, the resulting mean square estimation error.

(b) Now consider two other random variables: $Z = e^X$ and $W = e^Y$. Compute $\hat{z}_{BLS}(w)$, the Bayes least-squares estimate of Z based on observation of W along with $\lambda_{Z|W}$, and λ_{BLS_Z}, the resulting mean square estimation error. Hint: For random variable R, $E(e^R) = \Psi_R(\nu) \mid_{\nu=-j}$, where $\Psi_R(\nu)$ is the characteristic function of R.

Problem 8.2

Suppose the intensity V of a light beam is a random variable. We shine the light beam on a photomultiplier and observe the set of photocount measurements $\underline{Y} = [Y_1, Y_2, \cdots, Y_L]^T$ where Y_i is the number of counts in the ith of a set of L non-overlapping intervals each of duration T. When the intensity is known, the number of photocounts observed in an interval of duration T is a Poisson random variable with mean αv, where $\alpha > 0$ is a known constant and the numbers of photocounts observed in non-overlapping intervals are statistically independent random variables. Thus, for each $1 \le i \le L$ we have,

$$\Pr[Y_i = k|V = v] = \frac{(\alpha v)^k e^{-\alpha v}}{k!} \qquad \text{for } k = 0, 1, 2, \cdots .$$

In this problem we investigate estimating the intensity V of the light beam from such Poisson distributed photomultiplier observations.

(a) Suppose we have the following prior statistical model for V: $p_V(v) = \mu_v^{-1} e^{-v/\mu_v} u(v)$, where μ_v is the mean of the density. Determine $\hat{v}_{BLS}(\underline{y})$, the Bayes least-squares estimate of V based on $\underline{y}$, and the resulting mean-square estimation error λ_{BLS}. You may find the following information useful:

$$\int_0^\infty x^k e^{-ax}\, dx = \frac{k!}{a^{k+1}} \qquad \text{Erlang pdf: } \begin{cases} p_X(x) = \frac{a^n x^{n-1} e^{-ax}}{(n-1)!} u(x), \quad a > 0, n = 1, 2, \cdots \\ E(x) = \frac{n}{a} \\ \text{Var}(x) = \frac{n}{a^2} \end{cases}$$

(b) The prior distribution on V in (a) is exponential. We might expect that as the prior distribution approaches a uniform one, our estimate of αV would approach a simple average of the data. Verify that when $\mu_v \to \infty$, the estimate of part (a) reduces to

$$\hat{v}_{BLS} = \frac{1}{\alpha L}\left(\sum_{i=1}^{L} y_i + 1\right).$$

Problem 8.3

Let X and Y be random variables such that the random variable X is exponential, and, conditioned on knowledge of X, Y is exponentially distributed with parameter X, i.e.,

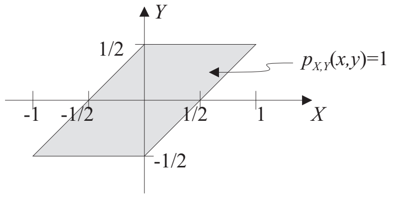
$$\begin{aligned} p_X(x) &= \frac{1}{a} e^{-x/a} u(x) \\ p_{Y|X}(y|x) &= x e^{-xy} u(y) \end{aligned}$$

(a) Determine $\hat{x}_{BLS}(y)$, $\Lambda_{X|Y}(y) = E\left\{[x - \hat{x}_{BLS}(y)]^2 \,|y\right\}$, and $\Lambda_{BLS} = E\left\{[x - \hat{x}_{BLS}(y)]^2\right\}$.

(b) Determine $\hat{x}_{MAP}(y)$, the MAP estimate of X based on observations of Y.

Problem 8.4 (Old Exam Question)

The random variables X and Y are uniformly distributed over the region shown in the figure.



(a) Find $\widehat{x}_{BLS}$ the Bayes least square estimate of X given Y, $\lambda_{X|Y}$ the corresponding conditional covariance, and λ_{BLS} the corresponding mean square error.

(b) Find $\widehat{x}_{LLS}$ the <u>linear</u> least square estimate of X based on both Y and Y^2. This is the minimum mean square estimator constrained to <u>linear</u> functions of Y and Y^2. Hint: Think before you calculate.

(c) Find $\widehat{x}_{MAP}$ the MAP estimate of X based on Y.

Problem 8.5

Let Z be a 2-dimensional Gaussian random vector with mean $\underline{m}_Z$ and covariance matrix Λ_Z as specified below:

$$\underline{Z} = \begin{bmatrix} W \\ V \end{bmatrix}, \qquad \underline{m}_Z = \begin{bmatrix} 0 \\ 0 \end{bmatrix}, \qquad \Lambda_Z = \begin{bmatrix} 2 & 1 \\ 1 & 4 \end{bmatrix}.$$

Let X be a Gaussian random variable with mean $m_X = 2$ and variance $\lambda_X = 8$. Assume X and $\underline{Z}$ are statistically independent. The random variable Y is related to X and $\underline{Z}$ as follows:

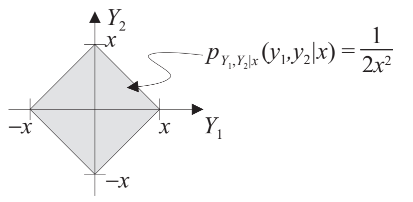
$$Y = (2 + W)X + V\ .$$

(a) Find $\hat{x}_{LLS}(y)$, the linear least-squares estimate of X based on observation of Y.

(b) Determine $\lambda_{LLS} = E\left[(X - \hat{x}_{LLS}(y))^2\right]$, the resulting mean-square estimation error.

Problem 8.6 (Old Exam Problem)

We wish to estimate an unknown, deterministic parameter x from two observations y_1 and y_2 that are uniformly distributed over the diamond shaped region parameterized by $x > 0$ shown in the figure. Hint: All parts can be answered with minimal calculation with some thought.



(a) Find $p_{Y_1|x}(y_1|x)$.

(b) Find a maximum-likelihood estimate of x based on y_1 alone.

(c) Find a maximum-likelihood estimate of x given both y_1 and y_2.

Problem 8.7

The purpose of this problem is to determine the (unknown) probability of heads when flipping a particular coin.

(a) Suppose that the coin is flipped N times in succession, each toss statistically independent of all others, each with (unknown) probability p of heads. Let n be the number of heads that is observed. Find the maximum likelihood (ML) estimate $\widehat{p}_{ML}(n)$ of p based on knowledge of n.

(b) Evaluate the bias $E[p - \widehat{p}_{ML}(n) \mid p]$ and the mean-square error $E[(p - \widehat{p}_{ML}(n))^2 \mid p]$ of the ML estimate.

(c) Is the ML estimate efficient (i.e. does the mean square error attain the Cramer-Rao bound)? An estimate is termed "consistent" if its mean square error goes to zero as $N \to \infty$. Is the ML estimate of p consistent. Briefly explain.

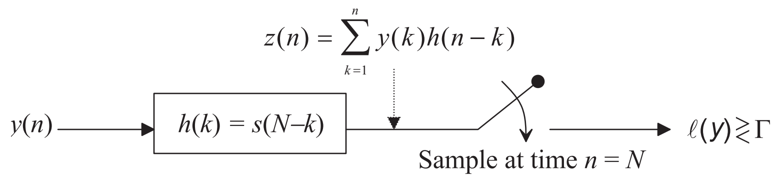
Computer Problems

Problem 8.8 The Matched Filter

One of the most common structures for making decisions from waveform observations is the "matched filter". In this project we will implement a matched filter detector and investigate its behavior. Consider the following discrete-time signal detection problem:

$$\begin{aligned} H_0 : y(n) &= w(n) \\ H_1 : y(n) &= s(n) + w(n) \end{aligned} \tag{1}$$

where the signal $s(n)$ is deterministic and known, $w(n)$ is discrete-time, zero-mean, white noise with $R_{WW}(k) = \sigma^2\delta(k)$ and the interval of observation is from $n = 1$ to $n = N_s$. Recall that the optimal detector for the signal $s(n)$ in noise is given by the matched filter structure, in which the test statistic $\ell(y) = \sum_{k=1}^{N_s} s(k)y(k)$ is compared to a threshold Γ. This statistic $\ell(y)$ can be interpreted as the output of a linear system "matched" to the signal of interest, as shown in the figure below:



(a) Using this definition you will write a MATLAB function `matched1.m` to implement a 1-dimensional matched filter waveform detector. The function will generate the matched filter output and detection decisions given data, thresholds, and the known signal s. The function call of your program will be the following: `[D,z] = matched1(y,Gamma,s)`, where `y` is a matrix of data samples with each row corresponding to a separate observation, `Gamma` is a vector of threshold values, and `s` is a row vector containing the deterministic signal $s(n)$. The function returns as output the entire sequence of values of the filter output $z(n)$ in the matrix `z`, with `z(i,:)` corresponding to the filter output associated with data `y(i,:)`. In addition, the matrix of decisions `D` is returned, where `D(i,j)` is a `0` or `1` corresponding to the decision associated to data vector `y(i,:)` at threshold `Gamma(j)`. This decision is obtained by comparing `z(Ns)` to the threshold, where `Ns` is the length of `s`. While this decision is ultimately important, the behavior of the entire filter output `z` is often of interest. Each step below will form a line of the program:

(i) The first step is to get the problem dimensions and make sure the target signal `s` is a row vector. Note: `Ns` is the length of `s` and `Ny`, the number or rows of `y`, is the number of independent experiments. The length of `s` must be the same as the number of columns in `y`.

```
[Ny,Ns] = size(y);
Ng = length(Gamma);
s = s(:)';
```

(ii) Next we find the matched filter impulse response. Recall that $h(n) = s(N - n)$, so we need to reflect the signal vector `s`:

```
h = fliplr(s);
```

(iii) Now for each independent data vector in the matrix `y` we need to convolve the data with the matched impulse response. The matrix `z` will store the matched filter output – row i of `z` is the output corresponding to row i of `y`:

```
z = zeros(Ny,2*Ns-1);
for i = 1:Ny
  z(i,:) = conv(y(i,:),h);
end;
```

(iv) Next, we take as our test statistic $\ell(y)$ the filter output `z` sampled at time `n=Ns`:

```
ell = z(:,Ns);
```

(v) Finally, we compare this value to the thresholds in `Gamma`. We can do all these comparisons for all experiments at once by cleverly using MATLAB's array functions:

```
D = ell*ones(1,Ng) > ones(Ny,1)*Gamma;
```

Add these steps together to create your program. You can now perform matched filtering on any signal.

Test your function by applying it to a noise free case. In particular, suppose that the signal is the hat function given by: `s = [zeros(1,9),[0:1:5],[4:-1:0],zeros(1,10)];` and that `y=s` so the observation is the signal itself. Find and plot the matched filter output `z`. In the absence of noise the peak of the filter output should also be the value of the test statistic `ell` and must occur at `n=Ns`. Make sure this is the case for your outputs.

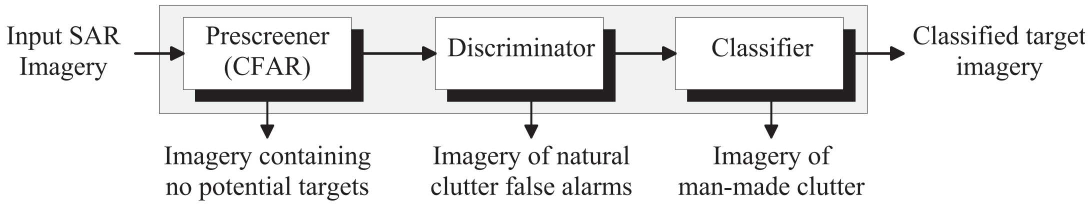
(b) We now apply the matched filter to the problem in (2). The signal $s(n)$ will be the hat function given above: `s = [zeros(1,9),[0:1:5],[4:-1:0],zeros(1,10)];`. The white Gaussian noise will have variance $\sigma^2 = 16$: `w = 4*randn(1,length(s));`. The data under H_0 is given by `y0=w` while the data under H_1 is given by `y1=s+w`. Plot and compare the observation under each hypothesis. Can you visually tell which case is H_0 and which H_1? It should be difficult at this noise level.

Now apply the matched filter. Plot the filter output `z` for both the H_0 data case `y0` as well as the H_1 data case `y1`. To generate the test statistic $\ell(y)$ for this example, the value of the matched filter output should be sampled at `n=Ns`, which is 30 for this example. Examine your graphs of `z` at the time `n=Ns`. Are the values for the different hypotheses well separated at this point?

Recall that the optimal ML decision rule threshold for this problem is given by `Gamma = s*s'/2` $=$ 42.5 for the given signal. Thus for good performance we would want `z(30)` $<$ 42.5 under H_0 and `z(30)` $>$ 42.5 under H_1. Repeat the experiment a few times and see if this seems indeed true.

Problem 8.9 Automatic Target Recognition Systems and CFAR Detectors

A preliminary processing step, used on virtually all current practical automatic target recognition (ATR) systems, is the "Constant False-Alarm Rate" or CFAR detector. In this project we develop the theory and then apply this detector. First consider the block diagram of an ATR system given below. Such systems are typically composed of a number of stages. In the early stages – typically the CFAR – the aim is to throw away parts of the data unlikely to contain objects of interest. The task of actually finding targets is left to later stages. The idea is to reduce the amount of data that has to undergo intensive processing to a more reasonable level. Thus, the job of the first stage is to simply find regions in the data that appear "interesting" and pass them along for further consideration. Note that data that is thrown away at one stage is gone from consideration forever, so early stages of the processor are typically adjusted to have very high P_D at the expense of P_F – it is better to pass along many non-targets than to throw away a real target. This means they typically operate at the top of their ROC.


Input SAR Imagery
Prescreener (CFAR)
Discriminator
Classifier
Classified target imagery
Imagery containing no potential targets
Imagery of natural clutter false alarms
Imagery of man-made clutter

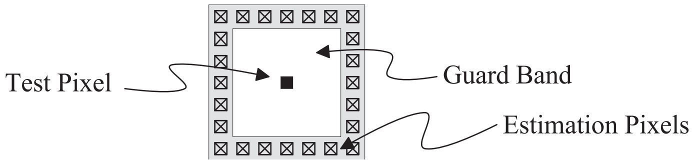
The idea behind the CFAR is very simple and we develop it next. First, suppose we have as our observation model that every pixel in an image is independent of every other and under each hypothesis an observed pixel y satisfies:

$$\begin{aligned} H_0 : y &= w \\ H_1 : y &= x + w \end{aligned}$$

where $w \sim N(m, \sigma^2)$. In other words, in the absence of a target in a pixel we just observe a nonzero-mean Gaussian random variable, while in the presence of a target x, the mean is shifted. The optimal decision rule for this problem is clearly to just compare each separate observation pixel in the image to a threshold.

Rather than directly comparing the pixel observation y to a threshold, we may equivalently take as our test statistic the quantity $\ell(y) = \frac{y-m}{\sigma}$ and then compare $\ell(y)$ to a threshold Γ (i.e. our test becomes $\ell(y) \gtrless \Gamma$). Notice that this does not really change the detector, only the particular value of the threshold used. But by subtracting the noise mean m and dividing by the noise standard deviation σ, the test statistic under H_0 is distributed $N(0,1)$, i.e. according to the standard Gaussian. Thus, the false alarm probability at each pixel for this problem is simply given by $Q(\Gamma)$, where $Q(\cdot)$ is the "Q"-function. When written in this way, the false alarm probability does not depend on the noise mean m or standard deviation σ. Of course, it never depends on the target x. Thus if we were to apply this detector to each pixel in an image with a stationary noise model (constant m and σ over the image) we would have a detector with a constant false alarm probability.

The problem, of course, is that images are very nonstationary with the mean and variance of the background noise varying considerably over the image. If we use a constant mean and standard deviation in our detector (not matched to the true local statistics of a given pixel), then the false alarm probability will fluctuate across the image. We might consider changing the noise mean and standard deviation to reflect the variation across the image, but in practice we do not know this variation a priori. The CFAR approach to this difficulty is to estimate the local mean $\widehat{m}$ and standard deviation $\widehat{\sigma}$ of the image around each pixel and then use these estimated values in the test statistic for that pixel $\ell(y) = \frac{y-\widehat{m}}{\widehat{\sigma}}$. If our estimates actually matched the true local mean and standard deviation, then our detector would again have a constant false alarm probability. The way this is typically done is that a mask is moved across the image and centered on each pixel in turn. The data in the mask is used to estimate the background mean and standard deviation, which are then used for the hypothesis test of the center pixel. A typical mask is shown in the figure. A guard band is typically left around the test pixel so that any target in the center will not contaminate the background estimates. In this way, under the simple Gaussian assumptions above, we have a test whose false alarm probability depends only on Γ, and is therefore constant across the image.



(a) Using this definition you will write two MATLAB functions to implement a 2-dimensional CFAR. The first function will be a core program called `cfar.m`. The function call of this program will be the following: `z = cfar(y,Nw)`, where `y` is an image matrix of data samples and `Nw` is the size of the $Nw \times Nw$ window used for background estimation. The function returns as output an image matrix `z` containing the value of the test statistic ℓ at each pixel. Now the CFAR simply slides a window across the image and uses the pixels at the edges of this window to normalize the center pixel of the window. This sliding-window-based processing is easily done in MATLAB using the function `nlfilter.m` (part of the image processing toolbox). In particular, the only line we need in the core program `cfar.m` is:

```
z = nlfilter(y,[Nw Nw],'findell');
```

which moves a sliding window over the image and passes that window of data to the (yet to be written) function `findell.m`.

The second function you must generate is `findell.m`, which implements the processing within each window of data. The function call of this program is `ell = findell(w)`, where `w` is an image matrix containing the current window of data (as illustrated in the figure) and `ell` is the corresponding normalized center pixel or test statistic. Recall, that for each window of data we use the edge pixels to estimate the mean and standard deviation of the window, then use these values to normalize the center pixel. Each step below will form a line of the `findell.m` program:

(i) The first step is to get the problem dimensions and extract the boundary pixels we will use to estimate the mean and variance. For simplicity we will only use the pixels on the boundary of the window, though other approaches are possible. These window boundary pixels are stored in the vector `x`.

```
[Nr,Nc] = size(w);
x = [w(:,1);w(:,Nc);w(1,2:Nc-1)';w(Nr,2:Nc-1)'];
```

(ii) Next we estimate the mean and standard deviation of the background in the window using the boundary data in `x`:

```
m = mean(x);
sigma = std(x);
```

(iii) Finally we generate the test statistic for this pixel by normalizing the center pixel using these values:

```
ell = (w((Nr+1)/2,(Nc+1)/2) - m)/sigma;
```

Add these steps together to create the program `findell.m`. Together these pair of programs implement a CFAR! Note that we have not included the threshold as part of the program. Instead, the output `z` is an image of the test statistic, which we may display or subsequently threshold as desired. One of the penalities we pay for our simple MATLAB implementation is speed – since our function loops over all the pixels in the image it will be slow.

(b) On the web site there is a data set in `cfar1.mat` with a variable `X` containing a clean "target" image, a variable `N` containing a corresponding noise or clutter image, and a data image `Y` equal to `X+N`. You may view the image using e.g. `imagesc(Y)`. Can you see the targets? Plot a few rows of `Y`. Note that the background is spatially varying in both mean and variance. A single threshold will fail to correctly locate all the target regions.

Now apply the CFAR to this data for a variety of window sizes. Plot the input `Y` and output `z` so you can compare them. You can use `imagesc.m` and `colormap(gray)` to display them. In general, we wish to make the window large enough so the target regions do not corrupt our moving estimates (i.e. we want the window at least as large as the targets) and also large enough that we have enough data points in them for reliable mean and variance estimates. Try window sizes of 7, 9, 13, 15. What is the effect of larger window sizes on your ability to detect the targets? Why not use very large windows? Try thresholding the output `z` of the CFAR at various levels by using `z>Gamma`. Can you find a combination of window size and threshold that isolates the targets from the background?

(c) (Optional)

On the web site there is additional data from an actual SAR scene with "targets" in `sarscene.mat` and data from simulated IR imagery with targets in `irscene.mat`. This data is actually used in the development and testing of ATR algorithms. You may wish to try your CFAR skills on it. The data also contains two vectors `xlocation` and `ylocation` containing the actual locations of the targets, which can be displayed via `plot(xlocation,ylocation,'rx')`. The IR scene is large and you may wish to excise a piece of it.

While it is not a focus of this course, you may also wish to experiment and create functions for the other two blocks of the typical ATR. For example, after thresholding the output of the CFAR a binary image is obtained. Typically there are isolated "false alarm" pixels in addition to the clusters of target detection pixels. Often the next stage of processing cleans up this binary image by looking for clusters. One way of doing this easily in MATLAB is to use the morphological function `bwmorph(z>Gamma,'open')`, where `z>Gamma` thresholds the image at `Gamma` and `bwmorph.m` then does a morphological opening on the resulting binary image. You can try this on the data in `cfar1.mat`.