GEORGIA INSTITUTE OF TECHNOLOGY

SCHOOL OF ELECTRICAL ENGINEERING

ECE 6272 FALL 2010

COMPUTER PROJECT #4

Assigned: Tuesday, Nov. 2, 2010

**Due Date for On-Campus Students: Thursday, Nov. 18, 2010 @ 9:35 AM EDT**

**Due Date for Distance Learning Students: Tuesday, Nov. 30, 2010 @ 4:00 PM EDT**

*Note: DL students are getting a few extra days due to the Thanksgiving holiday; enjoy!*

* This project is to be done *individually.* Each student must develop his or her own computer code in its entirety. Students are not to discuss the theory or approaches to coding the theory with one another, nor are they to assist in debugging each other’s work. You may ask Dr. Richards questions regarding theory and implementation of the project, including asking them at the beginning or end of class or during office hours, when others can benefit as well. You may post questions to the T-Square discussion board; that way, everyone can see the question and the answer. MATLAB is the preferred language, but others are acceptable; the point is to try the experiments, not to improve your MATLAB skills.
* Data required for this project are available for download from the class T-Square site in the Computer Projects area. ***Whether you begin working right away or not, be sure you download your data set and make sure you can load it into MATLAB (or other computing environment) as soon as possible to avoid last minute difficulties. Also, if you are NOT using MATLAB, be sure to note special instructions in Section 4.2*.**
* Commented code alone is *NOT* acceptable as a report; a written report clearly answering the specific questions asked in this assignment, with supporting code and figures, is required. Also, some specific format directions are given below; these must be followed to avoid a penalty. Reports will be graded on completeness in addressing the assignment and quality of results. They will not be graded on programming style or efficiency or on writing quality, except that the programming and the writing should be clear enough to be reasonably understandable. Questions or clarifications about the assignment should be directed to Dr. Richards.[[1]](#footnote-1) Errata, revisions and hints (if any) will be made available via the class T-Square site or during class.

# PROBLEM

This assignment is comprised of two largely independent sub-parts. In the first, we consider the problems involved in simulating the simplest cases of threshold detection. In the second, we implement a simple cell-averaging CFAR detector and analyze its threshold and false alarm performance.

# REPORT FORMAT

To aid in the grading, please observe the following format constraints:

* The first page of your report should be a cover page with your name, the class number, and the title of this project.
* The second page (or two, if you can’t get it on one page) of your report should contain the following specific items pertaining to part 1 of the project:
  + the two tables, filled in with your data, as described below;
  + the value you found for ** (the derivation goes in the middle section of the report); and
  + the value you found for *S* (the derivation goes in the middle section of the report).
* The third page (or two, if you can’t get it on one page) of your report should contain the following specific items pertaining to part 2 of the project:
  + the value (in linear units, not dB) you found for the ideal threshold corresponding to *PFA* = 10–2 (the derivation goes in the middle section of the report);
  + the value (in linear units, not dB) you found for the non-ideal CFAR threshold multiplier corresponding to *PFA* = 10–2 (the derivation goes in the middle section of the report);
  + the two plots (data with thresholds, and threshold histogram) for the CFAR\_even.mat data. The data with thresholds should be on a linear units amplitude scale; the histogram should be linear units on both axes (which it will be if you input linear-scale data to the hist() function); and
  + the observed *PFA* for the CFAR\_even.mat data;
* The fourth page (or two, if you can’t get it on one page) of your report should contain the following specific items pertaining to part 2 of the project:
  + the value (in linear units, not dB) you found for the *two* ideal thresholds corresponding to *PFA* = 10–2 for the two sections of the data (the derivation goes in the middle section of the report);
  + the value (in linear units, not dB) you found for the non-ideal CFAR threshold multiplier corresponding to *PFA* = 10–2 (the derivation goes in the middle section of the report);
  + the two plots (data with thresholds, and threshold histogram) for the CFAR\_uneven.mat data. The data with thresholds should be on a linear units amplitude scale; the histogram should be linear units on both axes (which it will be if you input linear-scale data to the hist() function);
  + the observed *PFA* for the CFAR\_uneven.mat data; and
  + your answers to the questions regarding behavior in the vicinity of the interference level transition.
* The fifth page (or two, if you can’t get it on one page) of your report should contain the following specific items pertaining to part 2 of the project:
  + the plot of the specified region of the data with CA, SOCA, and GOCA CFAR thresholds for the CFAR\_target.mat data. The data with thresholds should be on a *decibel* units amplitude scale; and
  + your answers to the questions regarding target masking behavior.
* All code should be kept together at the end.
* All other plots, diagrams, derivations, and explanations should be between these two things.

# Part 1: Monte Carlo Simulation of Detection and False Alarm Probabilities

The class lectures on radar detection give analytic results for the probability of false alarm, *PFA*, and probability of detection, *PD*, for the case of Gaussian I/Q noise and various target models. In this exercise, we will do a basic Monte Carlo simulation of the detection process for the simplest realistic case, that of the nonfluctuating target with random phase and linear detector, and compare our simulated *PFA* and *PD* against the analytical predictions.

## Part 1 Requirements

You must submit a brief summary of your findings for Part 1 that must include:

1. A listing of the MATLAB program used to compute the analytical probabilities, and to estimate them using a Monte Carlo simulation. This is sufficient to convey most details. Code must be sufficiently modular and well-commented to be understood readily. Listings must also be included for any functions used which are not supplied by MATLAB.

2. A table containing the following information. (Please use scientific notation for the estimated *PFA* values.)

|  |  |  |  |
| --- | --- | --- | --- |
| *PFA* | Analytically-Determined Threshold Value, *T* | Monte Carlo Estimate of *PFA* | Number of trials used to estimate *PFA* |
| 1x10–2 |  |  |  |
| 1x10–3 |  |  |  |
| 1x10–4 |  |  |  |
| 1x10–5 |  |  |  |
| 1x10–6 |  |  |  |

3. Another table containing the following information:

|  |  |  |  |
| --- | --- | --- | --- |
| Threshold Value, *T*, from above | Value of *PD* Predicted by Marcum’s *Q* Function | Value of *PD* Predicted by Albersheim’s Equation | Monte Carlo Estimate of *PD* |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

**The information in this list must be arranged in the format specified in Section of this assignment!**

## Part 1 Simulation

A “Monte Carlo” simulation is the technique of estimating the probability of an event (*e.g.*, the probability of target detection, *PD*) by repeatedly simulating the process leading to the event (the detection test) using random input data and observing the results.

The basic approach to the Monte Carlo simulation of *PFA* needed to fill in the first table consists of three steps:

1. Use analytic results from class to determine the threshold required to achieve a specified *PFA*; do this for the five values of *PFA* indicated in the first table above.
2. Generate a large number of random variables having a Rayleigh distribution *with mean = 1*. See Section 3.3.3 of this assignment for comments on what constitutes “large.”
3. Using the analytically derived thresholds, determine how many of the simulated random variables cross that threshold and convert that into a probability.

We want to simulate the data generated by a radar suing a coherent receiver with a linear detector when the interference is complex white Gaussian noise. One way to do this involves mimicking the process by which the radar receiver operates. Begin by generating two independent zero-mean Gaussian sequences, each with the same variance **2/2. Form a complex signal *y*[*n*] by treating one random vector as the real part of *y* and the other as its imaginary part, and then get the final linearly-detected signal *z*[*n*] by taking the magnitude of *y*[*n*], *z*[*n*] = |*y*[*n*]|. You want to choose the parameter ** so that the mean of *z* ends up equal to 1 (I had to specify something for the Rayleigh mean; 1 is as good as anything); see the textbook, or the formulas in the discussion in Section 3.3.1 of this handout, to get the relationship between ** and the mean of *z*[*n*]

It is useful to retain your complex data *y*[*n*]; you will need it when it is time to add a target to the data, which is what you will do shortly.

Because we are not performing any noncoherent integration before the threshold test, the theoretical results of Section 6.2.2 in the textbook apply. This is the case of an unknown absolute phase and a linear detector. The receiver diagram of Figure 6-5 is repeated here:



Figure 6-5. Structure of optimal detector when the absolute signal phase is unknown. A linear magnitude characteristic has been selected.

Eqn. (6.48) provides the expression for computing the threshold given a desired *PFA*:

 (6.48)

Recall that in deriving this equation, the signal energy *E* arose as the peak matched filter output , where  is the desired signal vector. Because we are operating on only a single sample without any coherent integration, our expected signal “vector”  is just the scalar *S* and its energy *E* is just *S*2; thus the equation for the threshold reduces to



Note that this is the appropriate threshold for a signal that has been passed through the matched filter as well as the linear detector, as shown in the above figure. Again, in this case the matched filter reduces to a simple multiplication by *S*, so the threshold above will be applied to the data sequence *z*[*n*] = |*Sy*[*n*]|. However, note also that if we scale both the data and the threshold by any real constant, it will not change which data samples do, or do not, cross the threshold. Thus, we can just as well let *S* = 1 in the equation for the threshold so that

, (6.53)

and apply it to the unscaled data sequence *z*[*n*] = |*y*[*n*]|. This procedure, discussed in the paragraph leading to Eqn. (6.53), is what is usually applied in practice, since it does not require that we know the signal strength *S* in advance; we usually won’t.

Now that the threshold has been determined, we want to determine the probability of detection for a given SNR. We will work this out specifically for an SNR of ** = 13 dB.

*PD* is computed using the Marcum *Q* function, *QM*; a MATLAB routine for its computation is included in the file Project\_4\_files.zip, and a listing is included at the end of this assignment. Equation (6.51) from the class notes is the appropriate result:

 (6.51)

The expected detection performance for this problem is given by the 13 dB curve from the receiver operating characteristic (ROC) plot of Figure 6-6.

envelope_gauss_example

Figure 6-6. Performance of the linear envelope detector for the Gaussian example with unknown phase.

A simple approximation to the expected *PD* for a given *PFA* in the nonfluctuating target case can be computed via Albersheim’s equation with the number *N* of noncoherently integrated samples equal to one and a fixed SNR **1 of 13 dB. Albersheim’s equation, recast it into a form that gives *PD* in terms of *PFA*, was given in Eqn. (6.89) of the textbook. Use Albersheim’s equation to fill in the appropriate column of the *PD* table.

For estimating probabilities of detection via Monte Carlo simulation to complete the second table, a constant *S* should be added to each complex interference sample *y*[*n*] to represent a target. (WARNING: Do not confuse adding *S* with adding *S*+*jS*.) The value of the constant should be chosen to give a signal-to-noise ratio of 13 dB; see Section 3.3.1 for more on this. The complex noise+target data should again be subjected to the linear detector and compared against the various thresholds to obtain a Monte Carlo estimate of *PD*.

## Discussion of Additional Part 1 Data and Processing Issues

### Signal to Noise Ratio

Remember that if the I and Q noise sequences are each zero mean Gaussian processes with variance **2/2, then the magnitude of *x* = I + *j*Q is Rayleigh-distributed. By examining the forms for the Rayleigh pdf given in Chapter 2 of the textbook, you can see that the mean of |*x*| is  and the variance is ; thus the interference power is  = **2. This makes sense since the Gaussian I and Q signals each contribute a power of **2/2. Our “signal” component is of the form *S*exp(*j*), where I have arbitrarily let **= 0; the power in this signal is just *S*2. Therefore the signal-to-noise ratio is *S*2/**2. Thus, given ** = 1 (because I specified use of a Rayleigh distribution with mean = 1), you can compute ** using the results above and then figure out the value of the constant *S* to add to the data such that the SNR *S*2/**2 is 13 dB, as specified. (Remember to be careful about choosing 20log10 *vs.* 10log10 in deriving *S*.) Give the numerical results for the values of ** and *S*.

### Threshold Testing

Given a threshold value *T* and a vector of random numbers *z*[*n*], suppose the corresponding MATLAB variables are T and z. One simple and quick way (there may be a number of others as well) to count the number of samples in z that exceed T is with this statement:

ncrossings = sum( z > T );

This count of threshold crossings can then be turned into an estimated probability in the obvious fashion.

### Number of Random Samples and Estimation of PFA

A common rule of thumb says that, to estimate a probability *p* using Monte Carlo techniques, you should use at least ten times the greater of 1/*p* or 1/(1−*p*) samples; 100 times is much better. Thus, if we are trying to estimate a detection probability of 0.9, we need at least 10(1/0.1) = 100 samples, and 1000 would be preferable. This becomes a problem when estimating small probabilities, such as probabilities of false alarm. Estimating a *PFA* of 10–5 require one million samples, with ten million preferable! This does not actually take an unreasonable amount of time, but it can take too much memory to work on a personal computer with limited memory. One way to overcome this is with a loop that builds up, say, 100,000 samples at a time. However, for this problem, it is acceptable and perhaps even instructive to just use the longest sequence you can without running out of memory.[[2]](#footnote-2)

If you are unable to use as many samples as you would like for simulating some of the smaller false alarm probabilities, you will probably find that the simulated *PFA* either falls to zero, indicating that none of the samples tested crossed the threshold, or is too high by an order of magnitude or so. Consider the case where *PFA* = 10–6, but we use only 105 samples to simulate *PFA*. The probability of at least one of the 105 samples crossing the threshold is only 0.1, so in about 9 trials out of 10, you will estimate *PFA* = 0. On the other hand, suppose precisely one of the samples *does* cross the threshold; your estimated *PFA* will then be 1/105 = 10–5, which is too *big* by an order of magnitude. Both of these problems merely indicate that you have too few samples to reliably estimate a *PFA* of 10–6. You will have less trouble with estimation of *PD* because you are not trying to simulate one-in-a-million probabilities.

# Part 2: CFAR Threshold Estimation

Part 1 of this project dealt in part with analytical prediction of the threshold for a desired *PFA*. Because it is rare for us to know exactly what the interference parameter(s) are, and because in the real world they vary, we now consider CFAR.

## Part 2 Requirements

You must submit a brief summary of your findings for Part 2 that must include:

1. A listing of the MATLAB program used to develop the required plots. I recommend that this be a completely separate code from that of Part 1 for ease of reading. Code must be sufficiently modular and well-commented to be understood readily. Listings must also be included for any functions used which are not supplied by MATLAB.

2. Answers to the specific questions, and the specific plots requested in the instructions and discussion below.

**The information in this list must be arranged in the format specified in Section of this assignment!**  A *limited number* of hard copy plots should be provided to substantiate your conclusions.

## Part 2 Data

All of the data you need is obtained by downloading the Winzip archive Project\_4\_files.zip available from the class T-Square site in the section on Computer Projects. When unzipped, this will produce the following files:

* Three radar data files. Each is a MATLAB .mat file; the file names are CFAR\_even.mat, CFAR\_uneven.mat, and CFAR\_target.mat. CFAR\_even.mat and CFAR\_uneven.mat each contain one 5000x1 array of positive real-valued data samples; CFAR\_target.mat contains one 1000x1 array of positive real-valued data samples; The name of the data variable is z in all three cases.
* The distribution also includes the MATLAB m-file marcum.m used in Part 1 of this project. It is not needed for Part 2.

Some of the distance learning students may not be using MATLAB. In this case, I will provide on request the equivalent text files CFAR\_even.txt, CFAR\_uneven.txt, and CFAR\_target.txt to replace CFAR\_even.mat, CFAR\_uneven.mat, and CFAR\_target.mat. Each line of the file CFAR\_even.txt will contain one element of the of the data vector z. A similar pattern is followed for the other files You will need to import these into your programming environment. I will not provide these files unless specifically requested by e-mail.

***Even if you don’t start work right away, be sure to download the data files and make sure you can load and work with them as soon as possible!***

## Part 2 Simulation

### Processing of CFAR\_even.mat

The data file CFAR\_even.mat contains a single column vector, z, of 5000 samples of complex Gaussian noise that has been passed through a *square law* detector. Consequently, the resulting real-valued, positive sequence has an exponential probability density function. The data has been scaled to have a mean value (thus mean power of the original complex data) equal to 1.0. This data is therefore the same as that in Part 1 with ** = 1. The only difference is that now we have squared the detected data to represent a square law instead of linear detector; and because we have not designed for a specific signal, the matched filter reduces to an identity operation, *i.e.*  = 1.

Begin by computing the analytical value of threshold required to obtain *PFA* = 10–2 for this data. Since we are not doing any noncoherent integration, the threshold formula for the *linear* detector in Part 1 can be used, and the resulting threshold simply squared to obtain the correct threshold value for the square law data. Clearly show your calculation of the ideal threshold.

Next, implement a cell-averaging CFAR processor to estimate the threshold from the data. You should use 50 test samples in *each* of the lead and lag windows (100 total). Also use 3 guard cells on each side of the cell under test. Thus, your total CFAR window will be 107 samples (50+50 reference cells, 3+3 guard cells, and the cell under test). Compute the estimated threshold by sliding this window over the data sequence, computing the local mean, and multiplying the estimated mean by the appropriate multiplier for a cell-averaging CFAR of size *N* = 100; the required multiplier ** was given in Eqn. (7.17) of the textbook:

 (7.17)

Note that this is applied to the data mean, as described in Eqn. (7.12).

 (7.12)

See Section 4.4.1 of this handout for a note on handling end effects.

You must illustrate your CFAR results with two plots. The first should show the data sequence z; the ideal threshold (this should just be a straight line across the data); and the estimated threshold, all on a linear units scale, not dB. The estimated threshold should be properly aligned with the data; that is, it should only exist for cells 54 through 4947 (see Section 4.4.1). If your results work out well the estimated threshold will approximately track the ideal threshold. The second required plot is a histogram of the threshold values. I recommend about 50 bins in the histogram. Also provide a marker or line of some sort on the histogram indicating the ideal threshold value. Histograms should be on a linear scale on both axes.

Finally, count (automatically or manually is OK, but with this much data, automatically is probably much easier) the number of threshold crossings observed and compare the observed *PFA* to the design value of 10–2.

### Processing of CFAR\_uneven.mat

The data file CFAR\_uneven.mat is similar to CFAR\_even.mat, except that the mean level of the noise increases by a factor of 10 halfway through the data, *i.e.* at sample 2501. This represents a case of nonhomogeneous interference and is a good example of why it is important to estimate the interference level by observing the received data, rather than pre-setting a fixed value.

Repeat the calculations and plots of Section 4.3.1 for this case. You will now report two ideal threshold values, one for the first half of the data and one for the second. You should plot the ideal and CFAR-estimated thresholds and all of the data in a single plot; do not break it into separate plots for the first and second halves of the data. Your histograms of the estimated thresholds should now cluster around two values. Indicate both ideal values on the histogram plot. Again, compare the observed and ideal *PFA* over the entire set of data.

Finally, consider the regions just prior to, and just after the increase in the interference level. Generally speaking (not just for this particular realization of the random process), for the cells just prior to the increase, is *PFA* increased or decreased compared to the design value of 10–2? Why? What about *PD*, and why? Answer the same two questions for the cells just after the transition.

### Processing of CFAR\_target.mat

The data file CFAR\_target.mat is actually a subset of the data in CFAR\_even.mat, except that three targets of varying amplitudes have been added at samples 500, 540, and 600. The purpose of this exercise is to illustrate the common CFAR problem of target masking, and to compare cell averaging, “greatest-of” cell averaging, and “smallest of” cell averaging CFAR detection laws (CA, GOCA, and SOCA CFAR) in responding to this problem.

Process the data in z using a cell-averaging CFAR in exactly the same manner as done in Section 4.3.1. Next, reprocess the data using a SOCA CFAR and a GOCA CFAR rule. This will give you three different threshold functions, one for each type of CFAR. Plot the data and all three CFAR thresholds on a single plot. **NOTE: for this plot only, display the data on a decibel amplitude scale**. We want to zoom in on the behavior of the thresholds around the three targets, so use the axis() command to limit the *x* axis to samples 450 through 650, and to limit the *y* axis to the range of 0 to 25 dB (if this isn’t the right amplitude range to home in on the target peaks, then you probably have used the wrong scale factor in converting to dB. Remember that this data is square-law detected.)

To do this correctly, we need to use different threshold multipliers for the SOCA and GOCA CFARs than we use for the standard CA CFAR; see Eqns. (7.35) and (7.37). It is straightforward to program these equations and find the appropriate multipliers **SO and **GO by trial and error, using your multiplier for CA CFAR as a starting point and noting that **SO an **GO will always be larger than the CA CFAR multiplier **. For this problem, the results are **SO = 5.175 and **GO = 4.4. You do not need to prove this yourself; you can simply use these two numbers.

Comment on the results. Of the three CFAR rules, state which of the targets will be detected by each of the three CFAR rules, and which will not. You should find that masking effects are more deleterious in detecting the first target than in detecting the other two; explain why.

## Discussion of Additional Part 2 Data and Processing Issues

### End Effects

Because your CFAR window is 107 samples long (a center “test cell” and 53 reference or guard cells to either side), it will not fully overlap the data for the first 53 data samples to be tested, or for the last 53. Rather than deal with these end effects, you need only estimate the threshold and perform the detection test for data samples 54 through 4947. This still leaves you with 4894 trials to estimate a probability of 0.01. The rule in Section 3.3.3 suggests that 1000 samples would be adequate to obtain good results in this case, so our 4894 samples gives a considerable margin above the minimum.

**Listing of marcum.m**

function q=marcum(a,t,prec)

% For (x,y) ~ N(0,a,1,1,0) marcum(a,t,prec) computes the

% probability that x^2+y^2 > t^2 using a method from

% Brennan & Reed, IEEE Trans. Info Theory, 1965,

% pp. 312-313

%

% Original MATLAB code obtained from P.F. Swaszek

% at URL http://www.ele.uri.edu/Courses/ele510/Marcum.m.

%

% Modified by M. Richards July 2002

% Modified again by M. Richards in March 2004 to avoid infinite loops

% for very small probabiliities ( q < 1e-6), which occur for large values

% of 'a', which occur when the SNR in the calling function

% is large

%

% prec=precision of computation. prec=0.001 => compute Q to 0.1% accuracy.

%

ct = .5\*t^2;

ca = .5\*a^2;

et = exp(-ct);

g = 1 - et;

k = exp(-ca);

gnew = et;

q = 1 - g\*k;

n=0;

while ( n < a\*t/2) | ( g\*k/(1-(.5\*a\*t/n)^2) > prec\*q )

if (q < 1e-6)

return

end

n = n + 1;

gnew = gnew \* ct/n;

g = g - gnew;

k = k \* ca/n;

q = q - g\*k;

end

1. Office: Klaus 3354, 404-894-2714, <mark.richards@ece.gatech.edu>. Office hours TBD, but drop-ins and appointments welcome. [↑](#footnote-ref-1)
2. This course is normally taught every two years. Over the years, the amount of memory commonly available in PCs tended to rise about 50%, and the processor speed doubled, between offerings of the course. Processor clock rates no longer double every two years, but core counts do. In any event, speed hasn’t been an issue the last couple of times the course was taught. Now it is probably the case that the amount of data that can be reasonably processed is not really an issue either due to expanded computer memory. [↑](#footnote-ref-2)