GEORGIA INSTITUTE OF TECHNOLOGY

SCHOOL OF ELECTRICAL ENGINEERING

ECE 6272 FALL 2010

COMPUTER PROJECT #4

**SOLUTION NOTES**

# My Solution for Part 1: Simulation

## Monte Carlo Simulation Results

Here are the numbers I came up with for the two required tables, using a single run of 500,000 samples. Additional runs produce very similar results. I could have used a longer sequence to get better results for the very low values of *PFA*, but I deliberately did not do so as to emphasize the problem of simulating very low probabilities.

|  |  |  |  |
| --- | --- | --- | --- |
| Analytical *PFA* | Required Threshold Value, *T* | Alternate Threshold Value if *S* included | Monte Carlo Estimate of *PFA* |
| 1x10–-2 | 2.42146 | 12.2049 | 0.997x10–-2 |
| 1x10–-3 | 2.96567 | 14.9478 | 0.948x10–-3 |
| 1x10–-4 | 3.42447 | 17.2603 | 1.02x10–-4 |
| 1x10–-5 | 3.82867 | 19.2976 | 0.8x10–-5 |
| 1x10–-6 | 4.1941 | 21.1394 | 0.0 |

|  |  |  |  |
| --- | --- | --- | --- |
| Threshold Value, *T*, from above | Analytical Value of *PD* using Marcum *Q* Function | Analytical Value of *PD* using Albersheim’s Equation | Monte Carlo Estimate of *PD* |
| 2.42146 | 0.9997 | 0.9993 | 0.9997 |
| 2.96567 | 0.9968 | 0.9960 | 0.9966 |
| 3.42447 | 0.9833 | 0.9840 | 0.9830 |
| 3.82867 | 0.9465 | 0.9517 | 0.9463 |
| 4.1941 | 0.8746 | 0.8844 | 0.8746 |

The program I used to obtain these numbers is called PdPfa.m; a listing is given at the end of this document.

In fact, I generated theoretical and simulated *PD* and *PFA* values for several additional choices of *PFA* in my program and then plotted the *PD* *vs.* *PFA* curves, which are one form of what is known as a “receiver operating characteristic” (ROC) curve. Figure 1 shows the results. I observe very good agreement between the simulated results and the predictions using both the Marcum and Albersheim analyses. Furthermore, by comparing specific data points, you can see that these results also match the ROC given in Figure 6-6 of the textbook (also repeated in the problem assignment) fairly well.

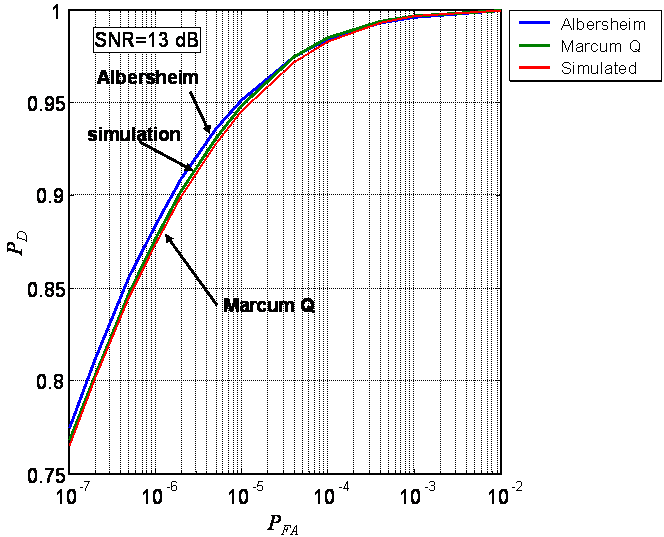


Figure . Comparison of simulated ROC curve *vs.* analytical predictions using the Marcum *Q* function and Albersheim’s equation.

## Discussion of Part 1 Results

### Threshold

From the problem handout, or from Ch. 2 of the textbook, I know that a Rayleigh distribution is obtained when I take the magnitude of complex Gaussian noise having zero mean and a variance of **2/2 in both the real and imaginary parts, and that the mean of the resulting Rayleigh distribution is . Since I have ** = 1, as specified in the problem statement, the value of ** must be

 = 1.1284

From the problem handout or from the textbook, Section 6.2.2, I also know that the threshold required to achieve a given *PFA* is then



where *S* is the signal amplitude value. 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 Fig. 6-5 of the textbook (also repeated in the problem assignment). Since in this case the matched filter reduces to a simple multiplication by *S*, the threshold above will be applied to the *scaled* data sequence *z*[*n*] = |*Sx*[*n*]|. However, note also that if I rescale *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, I can just as well let *S* = 1 in the equation for the threshold so that , and apply it to the *un*scaled data sequence *z*[*n*] = |*x*[*n*]|. This is the procedure usually applied in practice, since it does not require that I know the signal strength *S* in advance; I usually won’t. Thus, my code uses the above formula with ** = 1.1284 and *S* = 1 to get the thresholds I show in the table above, and then applies them to the unscaled data to get the values of *PD* and *PFA*.

### Monte Carlo Estimate of Probability of False Alarm

A close look at my simulated *PFA* results shows that they are reasonably good for the higher values of *PFA*, but fall apart for very low *PFA*. Recall that it was pointed out in the problem assignment 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. This means *at least* 1,000 samples for estimating *PFA* = 1x10–-2 and 10,000,000 for estimating *PFA* = 1x10–-6; preferable would be 10,000 and 100,000,000. I generated my results with a sequence of only 500,000 samples. This is adequate for estimating *PFA* = 1x10–-4 or higher, but totally inadequate for the cases of *PFA* = 1x10–-5 and 1x10–-6. This was deliberate so I could illustrate what happens with too few samples. The inadequate sequence length does in fact show in my data: as *PFA* gets smaller, the error in the simulated *PFA* increases from about 0.3% (0.997 *vs.* 1) at *PFA* = 1x10–-2, to about 2% for *PFA* = 1x10–-4, then to 20% at *PFA* =  
1x10–5. For *PFA* = 1x10–-6, my estimated *PFA* falls to zero, indicating that no samples crossed the threshold, an example of the phenomenon discussed in the problem handout. The number of samples tested is simply not enough to reliably estimate such small probabilities, so I see large discrepancies. With only 500,000 samples, the probability of one sample crossing the threshold when it is set for *PFA* = 1x10–-6 is only (5x105)(10–-6) = 0.5, *i.e.* there is only a 50-50 chance of having a threshold crossing. I re-ran my code until I got something other than zero for the estimated *PFA* for this threshold setting. On the second run, I got precisely one threshold crossing; but then this gives me an estimated *PFA* of 1/500,000 = 2x10–-6, still off by a very large amount! Again, I simply do not have enough samples to estimate a one-in-a-million probability.

### Target Amplitude

To simulate target + noise, I need to pick the target amplitude correctly to meet the specification of a signal-to-noise ratio of 13 dB. This means I want



The problem handout showed that the term *S*2/**2 is the signal to noise ratio; and since it is in units of power (*S*2 and **2 instead of *S* and **), I use 10log10 instead of 20log10 to convert to decibels. This gives you an amplitude of *S* = 5.04.

### Probability of Detection

To get the analytic value for *PD*, I used Eqn. (6.51) out of the textbook (also repeated in the problem handout):



The second form is obtained by just letting *E* = *S*2. However, remember that this formula was derived based on a threshold that is applied to scaled data, as discussed above in Section 1.2.1. Since I am applying it to *un*scaled data (in effect, rescaling the data and threshold both down by a factor of *S*), the threshold values I computed are already lower by a factor of *S*, so this equation is modified to



where in the last step I have substituted our specific values of *S* and **. I then evaluate for each threshold value of *T´* from Section 1.1 or 1.2.1.

An alternative way to analytically predict the expected value of *PFA* is to recast Albersheim’s equation to give *PFA* in terms of *SNR* and *PFA*. Equation (7.106) of the textbook does this for the case of *N* noncoherently integrated samples. I have *N* = 1; for this case, the equations simplify to



The simulated values of *PD* are obtained in exactly the same manner as the simulated values of *PFA*; the only difference is that a constant *S* = 5.04 has been added to each complex data sample before it is passed through the linear detector. Note that adding a constant of *S* to the data does *not* mean adding it to both the real and imaginary parts of the complex data; that would be adding a constant of (1+*j*)*S*. This is the most common error in working this assignment. One could actually add a complex constant of the form *Sej* to the data and obtain the same results; I have simply arbitrarily let ** = 0.

The second table shows the results obtained for estimating *PD*, as compared to the analytical predictions using both the Marcum analysis and Albersheim’s equation. These are much better behaved, with the simulated results varying by at most about 0.25% from the Marcum predictions, and about 0.9% from the Albersheim predictions. This more consistent behavior occurs because the highest analytic probability I am estimating is 0.9997. The rule of thumb suggests at least 33,000 samples to estimate this, with 333,000 preferable. Our 500,000 samples are therefore reasonably adequate for estimating this probability. Estimating the lowest *PD* of 0.876 requires only about 800 samples.

Figure 2 shows the probability density functions of the noise-only and signal+noise data, as estimated from the 500,000 samples using MATLAB’s hist function. I see that the two pdf’s are fairly well separated for the 13 dB SNR. Note also the Rayleigh shape of the noise-only histogram, and the approximately Gaussian shape (it’s actually Rician, of course) of the signal+noise histogram.



Figure . Histograms of noise only and signal+noise.

# My Solution for Part 2: Cell-Averaging CFAR

## Part 2 Simulation Results

The code I used to compute the CFAR threshold is cfar.mat; a copy of the code is included at the end of this handout. Figure 3 shows the homogeneous (constant statistics) data sequence cfar\_even.mat. Superimposed are the ideal threshold value of 4.6052 required for a *PFA* of 10–-2 (I’ll derive this in Section 2.2.1), which appears as a horizontal line, and the CFAR-computed threshold. The CFAR threshold multiplier is computed using the average power of the 100 cells in the neighborhood of the test cell, multiplied by the CFAR multiplier derived in Eqn. (8.17) of the text (see Section 2.2.2 for the details); the result is ** = 4.7129, higher than the ideal multiplier of 4.6052, as should be expected because I now have to estimate the interference power, rather than knowing it exactly.

Note that the CFAR threshold is a random process itself; it varies around the ideal value in a “noisy-looking” fashion. The mean value of the estimated CFAR values is 4.7161, higher than the ideal of 4.6052 because of the higher CFAR multiplier and random variations. There are 53 threshold crossings in 4894 tests (because of end effects, I defined the CFAR threshold only for indices 54 through 4947, inclusive). My observed *PFA* is therefore 1.08x10–-2, reasonably close to, but still higher than the design value of 10–-2. Note that I am using 4894 sample tests to estimate a probability of 10–-2; this meets my minimum standard of 1,000 samples discussed in Part 1, but still falls short of the more desirable 10,000 tests to estimate this reliably.

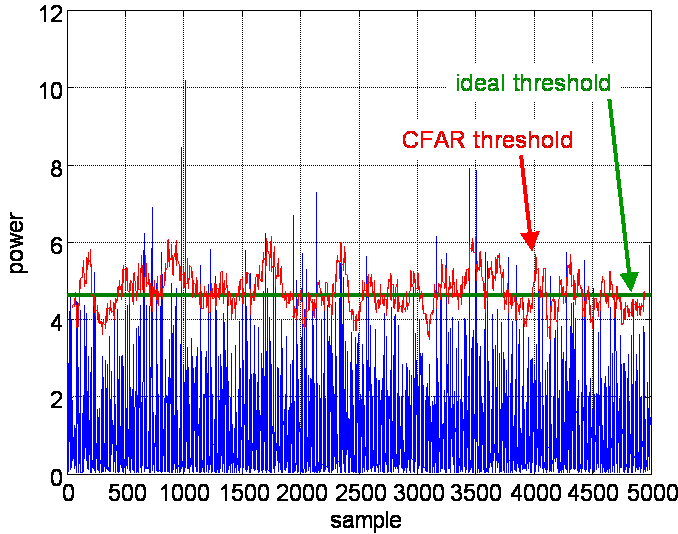


Figure . Data, ideal threshold (at 4.6052), and CFAR estimated threshold for the data file CFAR\_even.mat.

Figure 4 is a histogram of the CFAR threshold values. A vertical marker has been superimposed at the ideal threshold value of 4.6052. It is again apparent that the mean of the estimates is approximately equal to the ideal. Clearly also the distribution of values is a little asymmetric. (In fact, I gave the pdf of the estimated mean in lecture 19.)

Figure 5 and Figure 7 show the same information for the nonhomogeneous data in cfar\_uneven.mat. In this case, the ideal threshold remains 4.6052 in the first half of the data. In the second half, the noise power is increased by a factor of 10, and the threshold required to maintain *PFA* at about 1x10–-2 rises by a factor of 10 also, to 46.052. Both ideal thresholds are shown on the plot. I again had 53 threshold crossings in 4894 trials, giving again an estimated *PFA* of 1.08x10–-2. This is a bit of a coincidence. In this particular data set, the change in the threshold around the discontinuity did not result in any new false alarms on the high side, or eliminate any on the low side. Away from the transition region, the false alarm occur at exactly the same sample locations as in the constant-interference case.

The CFAR estimates are based only on the low-power noise for test cells 54 through 2447. The estimates are based only on the high-power noise for test cells 2554 through 4947. Test cells 2448 through 2553 are the transition region, where the lead window is at least partially in the high-power noise, and the lag window is at least partially in the low-power noise, so that the estimate is combining data from two different noise processes. The transition in CFAR threshold is expanded in Figure 6.



Figure . Histogram of CFAR threshold values for CFAR\_even.mat. The vertical marker is the ideal threshold value of 4.6052.

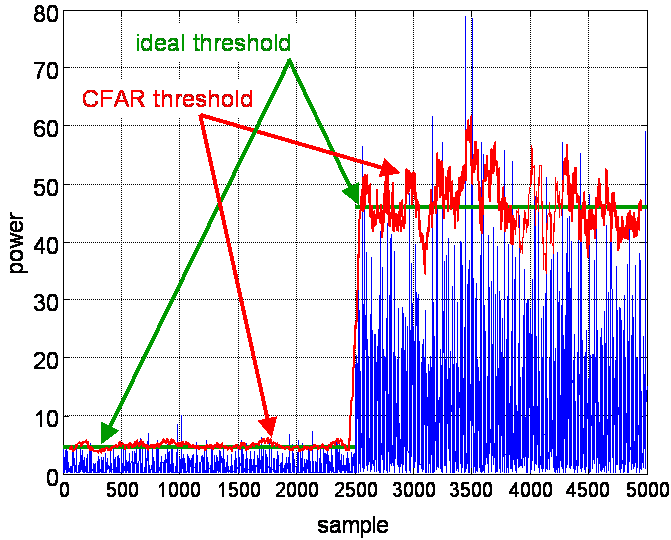


Figure . Data, ideal thresholds (at 4.6052 and 46.052), and CFAR estimated threshold for the data file CFAR\_uneven.mat.

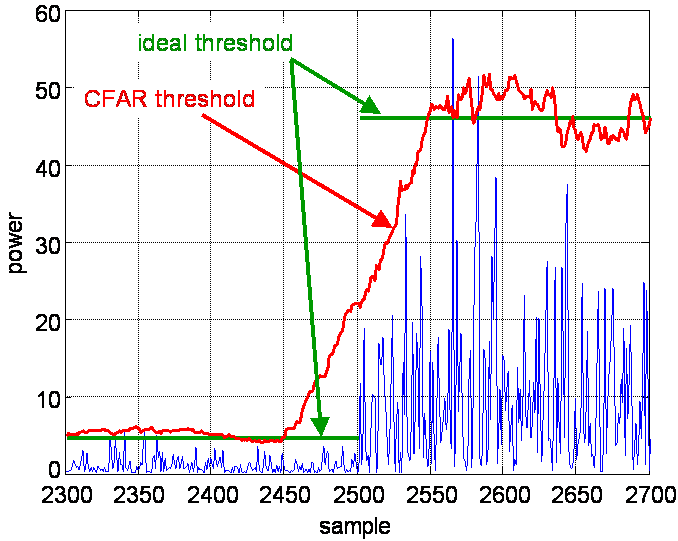


Figure . Expanded view of central portion of to show the CFAR threshold transition region.



Figure . Histogram of CFAR threshold values for CFAR\_uneven.mat. The vertical markers are the ideal threshold values of 4.6052 and 46.052.

The project assignment also asked you to consider the effect of the transition region on *PFA*. Consider test cells 2448 through 2500. The actual noise power in this region is 1.0, but the threshold is rising steadily above the ideal value of 4.6052 due to the effect of the lead window being in the high power region. Because of the increasing threshold, *PFA* decreases rapidly as the cell under test progresses from #2448 to #2500. For the same reason, *PD* will decrease for any target that may be located in this region. The particular data set in cfar\_uneven.mat supports this observation; none of the false alarms occur in the first half of the transition region.

For the second half of the transition, test cells #2501 to 2553, the converse is true. The actual interference power level is 10.0 in this region, and the ideal threshold level is 46.052. Because the lag window is still partially in the low-power region during this interval, the mean estimates are too low, and so are the resulting thresholds. Thus in this region, *PFA* and *PD* should both be increased above the intended values. The particular data set in cfar\_uneven.mat does *not* support this part of the prediction, as only one of the 13 false alarms occurs in the second half of the transition region. There is a close call at cell #2734, but the noise spike does not quite make it across the threshold. Figure 7-9 in the textbook shows an example where a false alarm does occur in the CFAR threshold transition region due to the failure of the cell-averaging CFAR threshold to rise immediately with a change in interference level.

Finally, we look at the target masking effects obtained by processing the data in cfar\_target.mat. Figure 8 shows the central portion of the processed data, with the a mplitude on a decibel scale. the targets are at samples 500, 540, and 600. The adaptive thresholds computed using the CA, SOCA, and GOCA CFARs are superimposed. The CA CFAR threshold was computed with the same multiplier ** = 4.7129 as before. The SOCA and GOCA CFARs use the provided multipliers **SO = 5.175 and **GO = 4.4.



Figure . Expanded view of central portion of cfar\_target.mat data to show target masking behavior.

First look at the CA CFAR threshold. It will detect the center and right targets, but will miss the left target because the threshold is above the target amplitude. The SOCA CFAR threshold maintains a more constant threshold level, lacking the plateaus to either side of a target that are typical of CA CFAR. As a result, it detects all three targets; this was the intent of the SOCA CFAR algorithm. The GOCA CFAR exhibits the highest threshold, missing detection of the left target by a wide margin, but still managing to detect the center and right targets. This is no improvement over the CA CFAR, but this is not surprising since the GOCA CFAR algorithm is primarily intended to combat clutter edge false alarms, not target masking.

These results indicate that masking of a target by a stronger adjacent target is more of a problem for the left target than for the right target. The reason is that the size of the lead and lag windows in our CFAR is 53 bins, including the guard cells. The right target is separated from the center target by 60 bins, so that when the CFAR window is centered on the right target, the center target is not in the lag window and therefore it does not influence the threshold at sample 600. In contrast, the spacing between the left and center targets is only 40 samples; when the CFAR window is centered on bin #500, the large target in bin #540 is in the lead window and therefore elevates the threshold.

## Discussion of Part 2 Results

### Idealized Threshold Computation

For the homogeneous case, I again have the same situation as in Section , where I computed the threshold using the formula , except that now the data has been squared, and it has been scaled to have a mean power = 1.0. Since I am not noncoherently integrating data samples (*i.e.*, I am testing each sample independently, one at a time), the new threshold, call it *T*´, will just be the square of the threshold used with the linear detector: . Furthermore, since **2 is the total noise power (sum of I and Q components, which were **2/2 each), and I stated that the mean power is now scaled to be equal to 1.0 (you can verify this by computing the mean of the data when you read it in), then I must now have ** = 1.0. Thus, the threshold to be used with cfar\_even.mat is simply

.

When I consider interference with an exponential distribution but a mean power *not* equal to 1.0, I will use the more general



I see that for a square law detector, Gaussian noise (exponential after the detector), and a design value of *PFA* = 10–-2, the value 4.6052 is the threshold multiplier; the threshold is simply 4.6052 times whatever the interference power is. Thus, in cfar\_uneven.mat, when the noise power rises from 1.0 to 10.0, the ideal threshold should rise from 4.6052 to 46.052.

### CFAR Threshold Computation

The value 4.6052 is the correct multiplier for the case where the interference mean power is known exactly. This is not true in a CFAR processor, in fact the whole point of CFAR is that I *don’t* know this and have to estimate it instead. In section 7.3.1 of the textbook, I derived the pdf of the resulting mean estimates, and in turn the expected value of *PFA*, which is also now a random variable. From this, I was able to solve for the correct threshold multiplier when I am using cell-averaging CFAR (Eqn. (7.17)). The multiplier that results is ** = , where *N* is the number of cells being averaged to estimate the interference power (100 for our CFAR), and *PFA* is the desired probability of false alarm (1x10–-2). For these numbers, the threshold multiplier becomes 4.7129 instead of 4.6052. This is the value used in the CA CFAR threshold calculation.

### Variance of Threshold Estimate

Notice in the histograms of Figure 7 that the variance of the threshold estimates is larger for the high-power interference than for the low-power interference. This is a direct consequence of the fact that the exponential distribution of the interference at the square law detector output is a one-parameter pdf. Specifically, the variance of the square-law detected interference is the square of its mean (see Table 2.3 in the textbook). If the power of the interference is increased by a factor of 10, the variance rises by a factor of 100. This in turn causes the variance of the mean estimates to rise as well.

**Listing of PdPfa.m**

%

% PdPfa

%

% Pd/Pfa via analysis and simulation for

% nonfluctuating target and Rayleigh interference

%

% Written by M. A. Richards

% February 1997

% Updated April 2000

% Update again March 2004

%

clear, hold off

close all

format compact

j = sqrt(-1);

% Form complex interference signal with Rayleigh amplitude

% and unit mean

N = input('Enter desired noise sequence length: ');

snrdB = input('Enter desired SNR (dB): ');

ray = rayleigh(N,1);

mean\_ray = mean(ray) % just checking to make sure mean is correct

std\_ray = std(ray) % just checking to make sure std is correct

beta = 2/sqrt(pi)\*1;

nray = ray.\*exp(j\*2\*pi\*rand(N,1)); % complex data with rayleigh amplitude, uniform phase

% Now create another version of each sequence with a target

% added to each sample at the specified SNR.

npow = beta^2;

S = sqrt((10^(snrdB/10))\*npow);

sray = nray + S;

mnray = abs(nray);

msray = abs(sray);

%mean\_nray = mean(mnray)

%std\_nray = std(mnray)

%figure(1)

%hist(mnray,50);hold;hist(msray,50);hold off

%pause

% The matched filter output is simply the input multiplied

% by conj(S) (but S is real here); the test statistic is the magnitude of that

% quantity (see Fig. 6.5 in the textbook). However, since we usually don't

% know S in advance, I will assume S = 1 for scaling the data and

% establishing the threshold.

znray = abs(nray);

zsray = abs(sray);

% Now let's set up a series of threshold settings to achieve

% desired Pfa's using analytic formula from text

Pfa = [1e-7 2e-7 5e-7 1e-6 2e-6 5e-6 1e-5 4e-5 1e-4 4e-4 1e-3 1e-2]';

T = beta\*sqrt(-log(Pfa));

% Now let's measure the Pfa with these thresholds and see how

% the results stack up against the design goal

for k=1:length(T)

Pfa\_sim(k) = sum( znray>T(k) )/N;

end

fprintf('\n\n Threshold Theoretical Pfa Simulated Pfa\n')

for k=1:length(T)

fprintf('%9.6g %5.3g %5.3g\n',T(k),Pfa(k),Pfa\_sim(k))

end

figure(2)

loglog(Pfa\_sim,Pfa);xlabel('Theoretical Pfa');ylabel('Simulated Pfa');

grid

pause

% Now let's predict Pd for the same thresholds analytically,

% and again compare against simulation

% Do two versions, one using Marcum Q to predict Pd, the other

% using Albersheim with N=1 (and rearranged to compute Pd instead

% of SNR)

% Some caution needed here in using the marcum function. For the

% simulation, where I was mimicking the actual processing of data, I

% couldn't assume I knew S, so I just used S = 1 in the threshold

% calculation. To use the Marcum function, however, I do have to plug in

% the actual SNR, which means using the actual value of S, and I need to

% use the threshold I would have computed if I had known S. That threshold

% would be larger than my actual threshold by a factor of S, so where the

% Marcum function uses sqrt(2)\*T(k)/S/beta for the second argument, I just

% want to use sqrt(2)\*T(k)/beta, because my threshold is already smaller by

% the factor S.

for k=1:length(T)

A=log(0.62/Pfa(k));

Z=snrdB/(6.2+(4.54/sqrt(1.44)));

B=(10^Z-A)/(1.7+0.12\*A);

Pd\_Alb(k)=1/(1+exp(-B));

Pd\_Q(k)=marcum(sqrt(2)\*S/beta,sqrt(2)\*T(k)/beta,0.01);

Pd\_sim(k) = sum( zsray>T(k) )/N;

end

fprintf('\n\n Threshold Pd (Marcum Q) Pd (Albersheim) Simulated Pd\n')

for k=1:length(T)

fprintf('%9.6g %5.3g %5.3g %5.3g\n',T(k),Pd\_Q(k),Pd\_Alb(k),Pd\_sim(k))

end

% Finally, let's plot Pd vs. Pfa in "receiver operating characteristic"

% (ROC) curve. Plot both predicted and simulated Pd.

figure(3)

semilogx(Pfa,Pd\_Alb,Pfa,Pd\_Q,Pfa,Pd\_sim); grid; xlabel('Pfa'); ylabel('Pd');

legend('Albersheim','Marcum Q','Simulated',-1)

title(['SNR=',int2str(snrdB),' dB'])

figure(4)

hist([znray,zsray],200)

**Listing of rayleigh.m**

function x=rayleigh(N,m)

%

% rayleigh

%

% M-file to generate a random sequence with a Rayleigh pdf

% and specified mean.

%

% N = desired sequence length

% m = desired mean of sequence

%

% M. A. Richards

% February 1997

%

% Generate uniforms, then transform.

x=rand(N,1);

s=4\*(m^2)/pi;

x=sqrt(-s\*log(x));

% Various tests:

% Compute sample mean and standard deviation

%sample\_mean=mean(x)

%sample\_standard\_deviation=std(x)

%fprintf('Theoretical standard deviation = %g\n',sqrt((4-pi)/pi)\*m)

% Show 100-bin histogram and also a theoretical Rayleigh

% curve with the desired mean 'm'.

%figure(2)

%[counts bins]=hist(x,100);

%pdf=((pi/(2\*m^2))\*bins).\*exp(-(pi/(4\*m^2))\*(bins.^2));

%bar(bins,counts/sum(counts))

%hold

%plot(bins,pdf/sum(pdf),'r')

%xlabel('x')

%ylabel('density')

%hold off

**Listing of cfar.m**

% cfar.m

%

% simple cell-averaging lead/lag CFAR

%

% Mark A. Richards

% April 2000

clear all, close all, hold off

load cfar\_even.mat;

% Compute threshold for Pfa = 10^(-2) and mean power = 1.

% Because the mean power = 1, this is also the threshold multiplier

% for any mean power level.

Pfa = 10^(-2);

Tmult = -log(Pfa)

% Set up the ideal threshold

T\_ideal = Tmult\*ones(size(z));

% The CFAR has 50 each lead and lag cells, and 3 guard

% cells on each side of the cell under test.

Nref = 50;

Nguard = 3;

Nz=length(z);

% Compute cell-averaging CFAR multiplier

Nc = 2\*Nref;

alpha = Nc\*((Pfa^(-1/Nc)) - 1)

% Do a brute force sliding window CFAR over the range of

% indices where the reference windows fully overlap the data.

% This is inefficient but easy to understand.

% Use NaNs ("not-a-numbers") for the undefined values in order

% to create a full length output sequence for plotting convenience.

first = Nref + Nguard + 1;

last = Nz - Nref - Nguard;

avg = NaN\*ones(size(z));

for k=first:last

avg(k) = (sum(z(k-Nguard-Nref:k-Nguard-1)) + ...

sum(z(k+Nguard+1:k+Nguard+Nref)))/2/Nref;

end

T\_cfar = alpha\*avg;

% plot the results. First, data and thresholds

figure(1)

plot([z T\_ideal T\_cfar]);

xlabel('sample'); ylabel('power');grid

% now the threshold histogram

figure(2)

hist(T\_cfar,50); axis([0 20 0 1200]);

xlabel('threshold value'); ylabel('number of occurrences');

title('Histogram of CFAR Threshold Values')

% add vertical marker at the value of the ideal threshold

hold on; v = axis;

plot([Tmult,Tmult],[v(3),v(4)],'r')

hold off;

% Observed Pfa

Ncross = sum(z>T\_cfar)

Pfa\_obs = Ncross/(Nz-2\*Nref-2\*Nguard)

% Now repeat the whole business for the CFAR\_uneven.mat data file

load cfar\_uneven.mat;

% Set up the ideal thresholds

T\_ideal1 = [Tmult\*ones(Nz/2,1); NaN\*ones(Nz-Nz/2,1)];

T\_ideal10 = [NaN\*ones(Nz/2,1); 10\*Tmult\*ones(Nz-Nz/2,1)];

% Do the CFAR.

avg = NaN\*ones(size(z));

for k=first:last

avg(k) = (sum(z(k-Nguard-Nref:k-Nguard-1)) + ...

sum(z(k+Nguard+1:k+Nguard+Nref)))/2/Nref;

end

T\_cfar = alpha\*avg;

% plot the results. First, data and thresholds

figure(3)

plot([z T\_ideal1 T\_ideal10 T\_cfar]);

xlabel('sample'); ylabel('power');grid

% re-plot, but just the transition region

figure(4)

plot([z(2300:2700) T\_ideal1(2300:2700) T\_ideal10(2300:2700) T\_cfar(2300:2700)]);

xlabel('sample'); ylabel('power');grid; axis([0 400 0 60])

% now the threshold histogram

figure(5)

hist(T\_cfar,100); axis([0 100 0 1200]);

xlabel('threshold value'); ylabel('number of occurrences');

title('Histogram of CFAR Threshold Values')

% add vertical marker at the values of the ideal thresholds

hold on; v=axis;

plot([Tmult,Tmult],[v(3),v(4)],'r')

plot(10\*[Tmult,Tmult],[v(3),v(4)],'r')

hold off;

% Observed Pfa

Ncross = sum(z>T\_cfar)

Pfa\_obs = Ncross/(Nz-2\*Nref-2\*Nguard)

% Now repeat for the target case

load cfar\_target.mat

Nz = length(z);

% Standard CA CFAR

first = Nref + Nguard + 1;

last = Nz - Nref - Nguard;

avg\_ca = NaN\*ones(size(z));

for k=first:last

avg\_ca(k) = (sum(z(k-Nguard-Nref:k-Nguard-1)) + ...

sum(z(k+Nguard+1:k+Nguard+Nref)))/2/Nref;

end

% Repeat for SOCA CFAR

avg\_soca = NaN\*ones(size(z));

for k=first:last

avg\_soca(k) = min((sum(z(k-Nguard-Nref:k-Nguard-1))), ...

sum(z(k+Nguard+1:k+Nguard+Nref)))/Nref;

end

% Repeat for GOCA CFAR

avg\_goca = NaN\*ones(size(z));

for k=first:last

avg\_goca(k) = max((sum(z(k-Nguard-Nref:k-Nguard-1))), ...

sum(z(k+Nguard+1:k+Nguard+Nref)))/Nref;

end

alpha\_soca = 5.175;

alpha\_goca = 4.4;

T\_ca = alpha\*avg\_ca;

T\_soca = alpha\_soca\*avg\_soca;

T\_goca = alpha\_goca\*avg\_goca;

T\_ideal = Tmult\*ones(size(z));

% plot the results. First, data and thresholds

figure(1)

plot(db([z T\_ideal T\_ca T\_soca T\_goca],'power'));

axis([450 650 0 25]);

xlabel('sample'); ylabel('power');grid