GEORGIA INSTITUTE OF TECHNOLOGY

SCHOOL OF ELECTRICAL AND COMPUTER ENGINEERING

ECE 6272 FALL 2010

COMPUTER PROJECT #3

**SAMPLE SOLUTION NOTES**

There were actually only five distinct data sets distributed. To determine which one was yours (and therefore which are the correct answers for your file), use your data file number modulo 5. For instance, data6\_mys.mat, data11\_mys.mat, and data\_16\_mys.mat are all identical to data1\_mys.mat.

Table 1 (next page) gives the values of the range, velocity, and relative RCS parameters used to create the five data files data1\_mys.mat through data5\_mys.mat. In other words, Table 1 has the correct answers. Your answers should approximate the values for the particular file that you used.[[1]](#footnote-1).

The program I used to process the data is called procdata; a listing is included at the end of this document, along with listings of a couple of simple subroutines used by procdata.

Figures 1 and 2 shows two views of the starting data in the matrix y form the file data1\_mys.mat. Figure 1 is just an overlay of the 20 pulses of data; Figure 2 is a mesh plot of the same data. In either case, no particular structure is evident. There are many local peaks, so it is not at all obvious where the targets are in range; and there is no Doppler information yet to localize them in Doppler frequency.

|  |  |
| --- | --- |
|  | *untitled* |
| *Figure 1. Overlay of the range traces from each of the 20 pulses of data in* data1\_mys.mat*.* | *Figure 2. Noncoherent integration of the range traces in Fig. 1.* |

Table 1. The Correct Answers.

|  |  |  |  |
| --- | --- | --- | --- |
| **File: data1\_mys.mat** | | | |
| Target # | Range (km) | Velocity (m/s) | Relative RCS (dB) |
| 1 | 2 | -60 | -26.7 |
| 2 | 3.8 | 37.5 | -7.55 |
| 3 | 4.4 | -45 | -3 |
| 4 | 4.4 | 22.5 | 0 |
| **File: data2\_mys.mat** | | | |
| Target # | Range (km) | Velocity (m/s) | Relative RCS (dB) |
| 1 | 1.9 | -30 | -10.9 |
| 2 | 3 | 22.5 | 0 |
| 3 | 3 | 52.5 | -13 |
| 4 | 4.2 | -60 | -4.15 |
| **File: data3\_mys.mat** | | | |
| Target # | Range (km) | Velocity (m/s) | Relative RCS (dB) |
| 1 | 2.1 | -30 | -14.3 |
| 2 | 2.1 | 67.5 | -20.3 |
| 3 | 3.2 | 22.5 | 0 |
| 4 | 4.4 | -30 | -4.47 |
| **File: data4\_mys.mat** | | | |
| Target # | Range (km) | Velocity (m/s) | Relative RCS (dB) |
| 1 | 2.5 | 52.5 | -11.2 |
| 2 | 3.4 | -60 | -15.8 |
| 3 | 3.4 | 37.5 | -9.82 |
| 4 | 4 | 22.5 | 0 |
| **File: data5\_mys.mat** | | | |
| Target # | Range (km) | Velocity (m/s) | Relative RCS (dB) |
| 1 | 2.5 | -52.5 | -8.34 |
| 2 | 3.4 | -37.5 | -7 |
| 3 | 3.4 | 22.5 | 0 |
| 4 | 4.3 | 67.5 | -5.92 |

A more useful view, and one I use frequently, is a range-Doppler plot. This is obtained by transforming the fast time/slow time matrix *y*[*l*,*m*] in the slow time (*m*) dimension to obtain a range-Doppler matrix *Y*[*l*,*F*). Since sampling did not begin until 12 **s after pulse transmission and successive samples are taken at a rate of 12 Msamples/sec, a peak in range bin #0 (#1 in MATLAB indexing) corresponds to a range of *c*(12 **s)/2 = 1.8 km, and each range bin increment corresponds to an increase in range of *c*(1/12 MHz))/2 = 12.5 m. Since I used the DFT to perform the Doppler analysis, bin #0 (#1 in MATLAB indexing) corresponds to zero Doppler shift, while each successive bin represents an increment of *PRF/K* Hz. I used a modest DFT size of 256 to get adequate Doppler definition, the corresponding increment in velocity is (**/2)(*PRF/K*) = 150/*K* m/s. With *K* = 256, this is 0.586 m/s per FFT bin. I also use a Hamming window on all FFTs for Doppler sidelobe reduction.

Note that the velocity sample spacing of 0.586 m/s does not meet the 0.5 m/s standard for “very good” velocity accuracy. I could easily fix this by using a larger DFT, and with current computer processing power, I expect that most students will do exactly that. However, I am going to stick with the relatively small DFT, and then later I will use the quadratic peak interpolation method to improve my accuracy to meet the specs, just to demonstrate the use of that technique.

The magnitude of *Y*[*l*,*F*) is plotted on a dB scale in Figures 3 and 4, in surface and contour plot formats, respectively.[[2]](#footnote-2) The DFT causes the target energy to concentrate along the frequency axis; a target signal that was a sinusoid across all 20 slow time samples and was lost among the noise and clutter is now an asinc function peaking at the appropriate Doppler shift. The asinc is still fairly “fat” simply because I have only 20 slow-time samples, which limits the Doppler resolution. The noise remains spread out across all of the range and velocity bins. The clutter is now a clearly evident concentration of energy around zero velocity, extending through all of the range bins. Note that the clutter energy falls off as range increases; the program that created the simulated data assumes that clutter power decreases as *R*-2, *i.e.* that the radar is operating in a beam-limited clutter scenario.

Note also that the four targets are now clearly visible. Each is in the form of a “hump” of energy extending across 1.5 km of range. This is because I have not yet done pulse compression. The signal from a target on a given pulse is just an echo of my transmitted pulse, which was 10 **s long. This is equivalent to (*c*/2)(10 **s) = 1.5 km. It is also evident that the power of each of the target signals is different, and that we have two apparently at the same range. You could actually determine the ranges of the targets at this stage from the leading edge of the pulse echoes. Those ranges appear to be 2, 3.8, and 4.4 km. One could also estimate the velocities from the center of each hump. Amplitude measurements are probably too noisy to be reliable at this point. We will obtain better measurements if we use further processing to improve the signal-to-interference ratio.

|  |  |
| --- | --- |
| *untitled2* | *untitled3* |
| *Figure 3. Range-Doppler mesh plot of data in* data1\_mys.mat*.* | *Figure 4. Range-Doppler contour plot of data in* data1\_mys.mat*.* |

My first processing step is pulse compression. Later I will do MTI filtering. I could do these in either order, since both are linear shift-invariant operations, and end up with the same results. However, many systems do pulse compression first because it can be performed pulse-by-pulse as the data is received. In my case, I chose to use a Hamming window on the matched filter to reduce range sidelobes. This is not really necessary in this data set; I do not have sidelobes from one target masking another target. However, I am using this solution to demonstrate the combination of many of the algorithms we have talked about, so I’ll include it. I took advantage of the special properties of the LFM pulse and applied the window in the time domain, which I find easier to do, and then applied the resulting impulse response to each fast time column of data using MATLAB’s conv function. Figures 5 and 6 are the mesh and contour range-Doppler plots of the result.

|  |  |
| --- | --- |
| *untitled* | *untitled* |
| *Figure 5. Range-Doppler mesh plot of pulse-compressed data.* | *Figure 6. Range-Doppler contour plot of pulse-compressed data.* |

Note that the 1.5 km extended humps of Figures 3 and 4 are now compressed into a peak in a single range bin, again at ranges 2, 3.8, and 4.4 km. Less obvious from these figures is that the amplitude of each target peak is now stronger relative to both the noise and the clutter. The absolute level of the noise, clutter, and target signals are all increased by the pulse compression filtering. The impulse response has length *L* = 120, the same as the pulse waveform. Linear filtering is a form of integration: the output is a weighted sum of input samples, with the weights being the impulse response coefficients. For white random processes such as noise and clutter[[3]](#footnote-3), integrating (summing) *N* samples increases the power by a factor of *L*. The coherent addition of the target samples by the matched filter increases the target power at its peak by *L*2. Thus, the target signal increases in power relative to the noise and clutter by a factor of *L*, which is 20.8 dB here. Since I windowed my impulse response and then used a DFT for Doppler processing, I have to take into account the losses caused by windowing (the *loss in processing gain LPG*) and the DFT (*straddle loss*) if I want to examine the details of before-and-after amplitudes and see how they match up with my simulation. I’m not going to go into that much detail here. The bottom line is, pulse compression will increase the signal-to-clutter and signal-to-noise ratios for each target by something on the order of 20 dB, and we can see this happening in the figures above.

Linear filters introduce delay in the filter output. My *L* = 120-point filter will introduce *L*-1 = 119 samples of delay into the output. This means that the range bin corresponding to the beginning of the range window, 1.8 km = (*c*/2)(12 **s), is no longer bin #0 (#1 in MATLAB) but is now #119 (120 in MATLAB). The first sample of the filter output (sample #0, #1 in MATLAB) corresponds instead to (*c*/2)(12 **s – 119(1/*Fs*) **s) = 0.3125 km. (*Fs*is the fast time sampling rate, 12 MHz.) Thus, it is necessary to re-label the range axis after pulse compression. The total number of range bins is now 337+120−1 = 456.

One of the most common errors in this project is being off by one in compensating the filter delay. It is very common for students to adjust ranges by 120 range bins instead of the correct 119. Since range bins correspond to 12.5 m, this creates an error of 12.5 m = 0.0125 km in each peak location in range. If you use xcorr to do the matched filtering, it takes some additional work to figure out the right adjustment. Another common error is to not adjust the ranges at all for the filter delay, in which case all ranges will be off by 1.4875 km. I hope that most students worked out these details in project 2 on LFM pulse compression.

At this point, the clutter is not really interfering with estimating the range and Doppler peaks, but I applied a 3-pulse canceller anyway to reduce the clutter power and clean up the images. Again, I am trying to demonstrate the combination of all of these techniques; it is not actually necessary to do MTI filtering to get good results. The impulse response of the three-pulse canceller is [1, -2, 1]; I used the conv function again, this time operating in slow time (*not* in Doppler; the range Doppler plots have, so far, been for display only). I did not remove any of the transients at the beginning or end of the convolution, so the result now has 20+3-1 = 22 samples in slow time. Figures 7 and 8 are the mesh and contour plots after MTI filtering. The major change is that there is now a significant trough where the clutter hump used to be, indicating successful suppression of the clutter energy.

|  |  |
| --- | --- |
| *untitled* | *untitled* |
| *Figure 7. Range-Doppler mesh plot of pulse-compressed data.* | *Figure 8. Range-Doppler contour plot of pulse-compressed data.* |

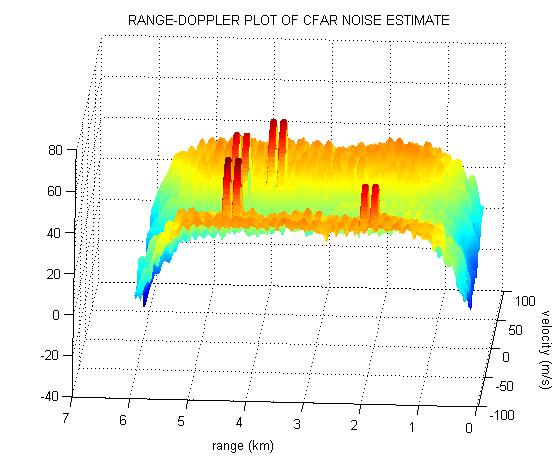
I am now ready to measure the velocities and relative powers of each of the target echoes. Velocities are measured from the peak of each target in the velocity axis of the range-Doppler plot. Because there are two targets at a range of 4.4 km, it is going to be necessary to measure the power of those two target echoes in the Doppler domain, so they can be separated based on their differing velocities. I need to make all my amplitude measurements in the same domain for consistency, so I will make my amplitude measurements directly from the magnitude of the range-Doppler map. The velocity and amplitude of each peak can be determined by manual inspection of the data in MATLAB. This approach is perfectly acceptable for this project. However, I decided to have some fun by implementing a semi-automatic detection algorithm, which occupies nearly the last half of the procdata.m M file.

My first step was to implement a cell averaging CFAR to estimate the interference level. This was applied to the magnitude-squared of the range-Doppler map, so results for a square-law detector are appropriate for designing the CFAR. I used a range-only 1D CFAR with a total window length of 23 bins. Of these, the center bin is the test cell; I used two guard cells on either side; this then leave 9 cells in each of the lead and lag windows. I used the conv function to efficiently apply this to each column, and discarded the transients at each end to keep the same matrix size at the output. Figure 9 shows the CFAR threshold in range and Doppler that resulted. The plot has been rotated to make more visible the characteristic CA-CFAR behavior around a target, namely an elevated threshold to either side of the target with a notch at the actual target location.

Here’s where I got lazy: I set my threshold multiplier heuristically, at a multiplier value of ** = 16. Since the data is magnitude-squared, I would expect noise to be exponentially distributed. We know that the threshold multiplier and average false alarm probability for cell averaging CFAR in this case are related by



With ** = 16 and *N* = 18, we expect  ≈ 10-5. There are a total of 456 range bins and 256 Doppler bins to which I apply the threshold test, giving a total of 116,736 = 1.17x105 tests, so “on average” I might expect about one false alarm with this threshold setting. In fact, I had no false alarms on this particular data set.



*Figure 9. CFAR threshold map.*

To detect the range bins for targets (which of course I already know by inspection anyway), I noncoherently integrated all of the Doppler bins to get a single column of total power *vs.* range. I then implemented some logic to search for local peaks above my CFAR threshold in range and to record their range bins. The result of these operations is shown in Figure 10.

My code then searches each range bin identified in the previous step for peaks in Doppler and marks those, recording the peak power of each. Figures 11-13 show the peak detections in Doppler for the three range bins of interest. I also used the function peakinterp to apply the quadratic interpolation formula of Section 5.3.4 in the textbook and refine the velocity and power estimates. (Of course, you could also just use a bigger DFT .)



*Figure 10. Detection of range peaks.*

|  |  |  |
| --- | --- | --- |
|  |  |  |
| *Figure 11. Doppler peaks at R = 2.0 km.* | *Figure 12. Doppler peaks at R = 3.8 km.* | *Figure 13. Doppler peaks at R = 4.4 km.* |

At this point, my algorithm has generated the following table of targets. Note that because of the peak interpolation routine, by DFT indices are not integers. If you just use a big DFT, yours will be integers.

INTERIM PARAMETERS OF DETECTED TARGETS:

Number Power Range (km) “DFT Index”

1 1.84e+003 2 26.91

2 2.59e+003 3.8 193.4

3 4.25e+003 4.4 51.8

4 1.9e+003 4.4 168.2

To finish the estimates, I need to convert DFT indices to velocity, and measured powers to relative RCS. DFT indices *k* are converted to velocity using the relationship *v* = *k*(**/2)(*PRF*/*K*). Remember that *k* counts from 0 in this formula, whereas MATLAB counts from 1. In the table above, the DFT index is the correct value of *k* for using this formula. Velocities above (**/2)*PRF*/2 (75 m/s here) are aliased to the negative velocities by subtracting off (**/2)*PRF* m/s (150 m/s in this case). A note about another common error here: I have seen many cases where a student does a good job of estimating frequency, except that they forget to do an fftshift on their data, *or* forget that they did do an fftshift on their data. Either way, they end up with all of their velocities shifted by half the unambiguous velocity range of 150 m/s, an error of ±75 m/s.

Now that we know the velocities of our moving targets, we might wonder if range-Doppler coupling is significant enough to have altered our range estimates. Since we are using an LFM waveform, range-Doppler coupling will occur. For a maximum Doppler shift of *FD* = *PRF*/2 = 5 kHz, a swept bandwidth of ** = 10 MHz, and a pulse length of ** = 10 **s, the maximum magnitude of the range shift of the LFM peak is  = 0.75 m. Since my range bins are 12.5 m apart, this shift is only 6% of a range bin, not enough to cause the peak to shift to the wrong range bin. Besides, this is well within the 12.5 “very good accuracy” requirement on range measurements. If I did interpolation of the range peaks to combat straddle losses, I might be able to detect this shift; I would then need to compensate for it based on the estimated Doppler shift of the target. Note also that I have assumed the target velocities are unambiguous; if target Doppler shifts can be higher than 5 kHz, the range shift will also be higher and might become more significant.

Because I used an MTI filter, its frequency response will have altered the relative amplitudes of each target signal, depending on their particular Doppler shift. Since I now know the Doppler shift of each target, this effect can be compensated. The magnitude of the frequency response of a 3-pulse canceller is |*H*(**)| = 4sin2(**/2)) = 4sin2(*FD*/*PRF*)) = 4sin2(2*v*/*PRF*)). The power of each target should be divided by the value of this function at the Doppler shift of that target to remove the effect of the MTI filter.

I still need relative RCS, not power. The radar range equation is



If we ignore atmospheric attenuation, then all of the terms on the right hand side of this equation are the same for every target except for the RCS and the range; thus received power is proportional to RCS divided by *R*4, which means that RCS is proportional to *R*4*Pr*:



Since I know the range of each target, I can convert power to a quantity proportional to RCS by simply multiplying the measured powers by *R*4. Relative RCS is then obtained by dividing each RCS value by the largest RCS value and converting the result to a dB scale. Since RCS is a power measurement, this means using 10log10, not 20log10.

Note that the *R*4 is the correct scaling for target data, but not for clutter, which would require either *R*2 or *R*3, depending on whether it was beam- or pulse-limited clutter. (Earlier I noted that the code that generated the data assumed beam-limited, but you don’t know that based on just the information provided to you.) However, I have filtered out most of the clutter and I am concerned with target amplitudes, not clutter amplitude. Also, I applied the range correction to power (magnitude-squared) data. If I wanted to do the correction on voltage data, I would use *R*2 instead of *R*4.

How well did I do? Table 2 repeats the correct answer from Table 1, and the final answers from my algorithm:

Table 2. Correct Answers vs. My Answers for data1\_mys.mat.

|  |  |  |  |
| --- | --- | --- | --- |
| **Correct Answers** | | | |
| Target # | Range (km) | Velocity (m/s) | Relative RCS (dB) |
| 1 | 2 | -60 | -26.7 |
| 2 | 3.8 | 37.5 | -7.55 |
| 3 | 4.4 | -45 | -3 |
| 4 | 4.4 | 22.5 | 0 |
| **My Answers** | | | |
| Target # | Range (km) | Velocity (m/s) | Relative RCS (dB) |
| 1 | 2 | -59.82 | -26.5 |
| 2 | 3.8 | 37.74 | -7.35 |
| 3 | 4.4 | -45.23 | -2.82 |
| 4 | 4.4 | 22.94 | 0 |

Not bad! My ranges are correct. My maximum Doppler error is 0.23 m/s. Recall that my Doppler bin spacing is 0.586 m/s, so this is an error of about 0.4 bins. This is actually a little disappointing, I would have hoped that my quadratic interpolation would reduce the error more than that; on the other hand, there is still noise in the data. My relative RCS values are correct to within 0.2 dB, which I consider a very good result. Again, I do still have some noise, which will perturb the peak power estimates. I declare victory!

**Listing of procdata.m**

%

% procdata

%

% process pulse Doppler radar project data

%

% Written by J. H. McClellan

% Modified by M. A. Richards

%

% Updated by M. A. Richards, Oct. 2006

%

clear, hold off

format compact

J = sqrt(-1);

close all

% Get root file name for reading results

file=input('Enter root file name for data file: ','s');

eval(['load ',file,'.mat'])

fprintf('\nPulse length = %g microseconds\n',T/1e-6)

fprintf('Chirp bandwidth = %g Mhz\n',W/1e6)

fprintf('Sampling rate = %g Msamples/sec\n',fs/1e6)

figure

plot((1e6/fs)\*(0:length(s)-1),[real(s) imag(s)])

title('Real and Imaginary Parts of Chirp Pulse')

xlabel('time (usec)')

ylabel('amplitude')

grid

PRI = 1/PRF;

fprintf('\nWe are simulating %g pulses at an RF of %g GHz',Np,fc/1e9)

fprintf('\nand a PRF of %g kHz, giving a PRI of %g usec.',PRF/1e3,PRI/1e-6)

fprintf('\nThe range window limits are %g to %g usec.\n', ...

T\_out(1)/1e-6,T\_out(2)/1e-6)

% Compute unambiguous Doppler interval in m/sec

% Compute unambiguous range interval in meters

vua = 3e8\*PRF/(2\*fc);

rmin = 3e8\*T\_out(1)/2;

rmax = 3e8\*T\_out(2)/2;

rua = rmax-rmin;

fprintf('\nThe unambiguous velocity interval is %g m/s.',vua)

fprintf('\nThe range window starts at %g km.',rmin/1e3)

fprintf('\nThe range window ends at %g km.',rmax/1e3)

fprintf('\nThe unambiguous range interval is %g km.\n\n',rua/1e3)

% Convert range samples to absolute range units.

[My,Ny]=size(y);

range=(3e8/2)\*((0:My-1)\*(1/fs) + T\_out(1))/1e3;

pulse = (1:Ny);

% Force oversize FFT, and compute doppler scale factor

Lfft = 2^(nextpow2(Ny)+3);

doppler = (((0:Lfft-1)/Lfft)-0.5)\*vua;

fprintf('\nThe Doppler increment is %g Hz.',PRF/Lfft)

fprintf('\nThe velocity increment is %g m/s.',3e8\*PRF/Lfft/2/fc)

% Start with a few plots to examine the data

% plot power of raw data in dB

ydB=db(abs(y)/max(max(abs(y))),'voltage');

figure

mesh(pulse,range,ydB)

title('FAST-TIME/SLOW-TIME PLOT OF RAW DATA')

ylabel('range (km)')

xlabel('pulse number')

% Plot overlay of individual range traces

disp(' ')

disp(' ')

disp('...plotting overlay of range traces')

figure

plot(range,db(y,'voltage'))

title('OVERLAY OF RANGE TRACES')

xlabel('distance (km)')

ylabel('amplitude (dB)')

grid

% Noncoherently integrate the range traces and display

disp('...plotting integrated range trace')

figure

plot(range,db(sum((abs(y).^2)')','power'))

title('NONCOHERENTLY INTEGRATED RANGE TRACE')

xlabel('range bin')

ylabel('power')

grid

% Doppler process and square-law detect the whole

% unprocessed array and display mesh.

% Use Hamming window throughout.

disp('...computing raw range-Doppler map')

Y=fft(conj(y').\*(hamming(Ny)\*ones(1,My)),Lfft);

Y=git\_rotate(Y.\*conj(Y),Lfft/2); % note we take mag-squared of Y here also

YdB=db(abs(Y),'power');

figure

mesh(doppler,range,YdB')

title('RANGE-DOPPLER PLOT OF UNPROCESSED DATA')

ylabel('range (km)')

xlabel('velocity (m/s)')

levels=(max(YdB(:))+[-1 -5 -10 -15 -20 -25 -30]);

figure

contour(doppler,range,YdB',levels)

title('RANGE-DOPPLER CONTOUR PLOT OF UNPROCESSED DATA')

ylabel('range (km)')

xlabel('velocity (m/s)')

grid

% Now start processing the data ...

% Pulse compression first. Use time-domain Hamming weighting of the

% impulse response for range sidelobe control

Ls = length(s);

disp('...performing matched filtering')

h = conj( s(Ls:-1:1) );

h = h.\*hamming(length(h)); % time-domain Hamming window for range sidelobe control

yp = zeros(My+length(h)-1,Ny);

for i=1:Ny

yp(:,i) = conv(h,y(:,i));

end

[Myp,Nyp]=size(yp);

% yp = fftfilt( h, y ); % using fftfilt instead of conv because it filters

% % multiple columns with one call

% compute new range and time scales here to take account of increased range

% length due to convolution and offset due to filter delay of Ls-1 samples.

rangep = (3e8/2)\*(((0:Myp-1)-(Ls-1))\*(1/fs) + T\_out(1))/1e3;

timep = (1e3\*rangep)\*2/3e8;

ypdB=db(abs(yp),'voltage');

figure

mesh(pulse,rangep,ypdB)

title('FAST-TIME/SLOW-TIME PLOT OF PULSE-COMPRESSED DATA')

ylabel('range (km)')

xlabel('pulse number')

levels=(max(ypdB(:))+[-1 -5 -10 -15 -20]);

figure

contour(pulse,rangep,ypdB,levels)

title('FAST-TIME/SLOW-TIME CONTOUR PLOT OF PULSE-COMPRESSED DATA')

ylabel('range (km)')

xlabel('pulse number')

grid

% Range-Doppler plots of pulse-compressed data

YP=fft(conj(yp').\*(hamming(Nyp)\*ones(1,Myp)),Lfft);

YP=git\_rotate(YP.\*conj(YP),Lfft/2);

YPdB=db(abs(YP),'power');

figure

mesh(doppler,rangep,YPdB')

title('RANGE-DOPPLER PLOT OF PULSE-COMPRESSED DATA')

ylabel('range (km)')

xlabel('velocity (m/s)')

levels=(max(YPdB(:))+[-1 -5 -10 -15 -20 -25 -30]);

figure

contour(doppler,rangep,YPdB',levels)

title('RANGE-DOPPLER CONTOUR PLOT OF PULSE-COMPRESSED DATA')

ylabel('range (km)')

xlabel('velocity (m/s)')

grid

% Apply three-pulse canceller in each range bin to raw data

disp('...performing 3-pulse clutter cancellation')

h = [1 -2 1]';

ypm = zeros(Myp,Nyp+length(h)-1);

for i=1:Myp

ypm(i,:) = conv(h,yp(i,:));

end

[Mypm,Nypm]=size(ypm);

% Doppler process and square-law detect the whole

% clutter-cancelled array and display mesh

disp('...computing clutter-cancelled range-Doppler map')

YPM=fft(conj(ypm').\*(hamming(Nypm)\*ones(1,Mypm)),Lfft);

YPM=git\_rotate(YPM.\*conj(YPM),Lfft/2);

YPMdB=db(abs(YPM),'power');

figure

mesh(doppler,rangep,YPMdB')

title('RANGE-DOPPLER PLOT OF CLUTTER-CANCELLED DATA')

ylabel('range (km)')

xlabel('velocity (m/s)')

levels=(max(YPMdB(:))+[-1 -5 -10 -15 -20 -25]);

figure

contour(doppler,rangep,YPMdB',levels)

title('RANGE-DOPPLER CONTOUR PLOT OF CLUTTER-CANCELLED DATA')

ylabel('range (km)')

xlabel('velocity (m/s)')

grid

% OK, now let's do a 1D range-only CFAR to set a threshold map.

% Define CFAR window 23 in range, including test cell and 2 guard cells, so

% 18 actual averaging cells

cfar = ones(23,1)/18;

cfar(10:14)=0;

% Now convolve it in range dimension with range-Doppler map YPM

% to get average noise map; discard half-transients to maintain data array size

N = zeros(size(YPM));

for i=1:Lfft

temp = conv(YPM(i,:),cfar);

N(i,:) = temp(12:end-11);

end

% plot the resulting noise map

NdB=db(abs(N),'power');

figure

mesh(doppler,rangep,NdB')

title('RANGE-DOPPLER PLOT OF CFAR NOISE ESTIMATE')

ylabel('range (km)')

xlabel('velocity (m/s)')

% multiply the noise by a factor of 16 (20 dB) and then threshold the data

threshold = 16\*N;

detected = NaN\*ones(size(YPM));

detected(YPM>threshold) = YPM(YPM>threshold);

% plot the resulting detection map on a dB scale

figure

mesh(doppler,rangep,db(detected','power'))

title('RANGE-DOPPLER DETECTION MAP')

ylabel('range (km)')

xlabel('velocity (m/s)')

% To search for range bins with targets, first noncoherently integrate

% across the frequency bins

YPMrange = sum(YPM);

% noise floor estimate based on median to avoid elevation of estimate by

% target responses

Nrange = median(YPMrange);

Trange = 8\*Nrange; % threshold 8x (9 DB) above noise estimate

% This loop identifies which range bins have local peaks above the

% threshold. It also sets up a vector for plotting convenience to circle

% the peaks that are found.

spikesr = [];

marker\_r = NaN\*ones(1,length(YPMrange));

for i=2:Mypm-1

if ((YPMrange(i) > YPMrange(i+1)) & (YPMrange(i) > YPMrange(i-1)) & (YPMrange(i) > Trange))

spikesr=[spikesr;i,YPMrange(i)];

marker\_r(i) = YPMrange(i);

end

end

[Mspikesr,Nspikesr]=size(spikesr);

figure

plot(rangep,db([YPMrange;Nrange\*ones(1,length(YPMrange)); ...

Trange\*ones(1,length(YPMrange))],'power'));

hold on

plot(rangep,db(marker\_r,'power'),'-ro')

hold off

xlabel('range (km)')

ylabel('power (dB)')

title('RANGE PEAKS')

grid

targets = [];

% Now find the Doppler peak(s) for each range bin having a target(s). Keep

% adjoining Doppler values as well to support subsequent interpolation.

for i = 1:Mspikesr

rb = spikesr(i,1) % current range bin

% search in Doppler in this range bin. Have to do this modulo the

% DFT size to allow for Doppler peaks near the spectrum edges.

spikesd = [];

marker\_d = NaN\*ones(1,Lfft);

for k=1:Lfft

km1 = k-1;

if km1 < 1

km1 = km1+Lfft;

end

kp1 = k+1;

if kp1 > Lfft

kp1 = kp1-128;

end

if ((YPM(k,rb) > YPM(kp1,rb)) & (YPM(k,rb) > YPM(km1,rb)) & (YPM(k,rb) > Nrange/2))

spikesd=[spikesd;k,YPM(km1,rb),YPM(k,rb),YPM(kp1,rb)];

marker\_d(k) = YPM(k,rb);

end

end % end of loop over Doppler bins

[Mspikesd,Nspikesd]=size(spikesd);

figure

plot(doppler,db([YPM(:,rb),Nrange\*ones(Lfft,1)],'power'));

hold on

plot(doppler,db(marker\_d,'power'),'-ro')

hold off

xlabel('velocity (m/s)')

ylabel('power (dB)')

title(['DOPPLER PEAKS FOR RANGE BIN ',int2str(rb)])

grid

% For each local peak in Doppler, use 'peakinterp'

% to refine the amplitude and location estimate using a quadratic

% interpolation

for i=1:Mspikesd

% note that peakinterp works on magnitude, not magnitude-squared

[amp del\_k] = peakinterp(sqrt(spikesd(i,2:4)));

targets = [targets;[amp, rangep(rb), spikesd(i,1)+del\_k]];

end

end % end of loop over range bins containing peaks

[Mtargets, Ntargets] = size(targets);

fprintf('\n\nINTERIM PARAMETERS OF DETECTED TARGETS:\n')

fprintf('\nNumber Power Range (km) DFT Index')

for i = 1:Mtargets

fprintf('\n %2.0g %7.3g %6.3g %9.4g', ...

i,targets(i,:))

end

disp(' ')

disp(' ')

% OK, now 'targets' contains a list, we hope, of all 4 targets with the

% range bin, interpolated peak magnitude, and interpolated Doppler bin of

% each. Now we need to get in the desired units and adjust the amplitudes

% for a couple of factors.

% Put Doppler peaks into m/s units

targets(:,3) = (((targets(:,3)-1)/Lfft)-0.5)\*vua;

% Adjust target amplitudes for MTI filter,

% and then convert to relative RCS

targets(:,1) = targets(:,1)./(4\*(sin(targets(:,3)\*(pi/vua))).^2);

targets(:,1) = targets(:,1).\*targets(:,2).^2;

targets(:,1) = targets(:,1)/max(targets(:,1));

targets(:,1) = db(targets(:,1),'voltage');

% List out detected target amplitudes, ranges, Dopplers

fprintf('\n\nESTIMATED PARAMETERS OF DETECTED TARGETS:\n')

fprintf('\nNumber Rel RCS (dB) Range (km) Vel (m/s)')

for i = 1:Mtargets

fprintf('\n %2.0g %7.3g %6.3g %9.4g', ...

i,targets(i,:))

end

disp(' ')

disp(' ')

**Listing of git\_rotate.m**

function rotated = git\_rotate(x,num\_places)

%GIT\_ROTATE [jMc 2/89]

% git\_rotate(V,r) circularly shifts the elements in the columns of V

% by r places right (r>0); or r places left (r<0).

% (Right is down; left is up.)

% If the input is a row or column vector, the shift is

% performed on the vector.

% If the input is a signal matrix, each column is shifted

% see also SHIFTM, SHIFT, ZEROPAD

[M,N] = size(x);

if M > 1 % ------- rotate columns ----------------

num\_places = mod(num\_places,M); % make num\_places in range [0,M-1]

rotated = [ x(M-num\_places+1:M,:); x(1:M-num\_places,:) ];

elseif N > 1 % ------- rotate row vector -------------

num\_places = mod(num\_places,N); % make num\_places in range [0,N-1]

rotated = [ x(N-num\_places+1:N) x(1:N-num\_places) ];

end

**Listing of peakinterp.m**

function [amp,del\_index] = peakinterp(z)

%PEAKINTERP

% peakinterp performs a quadratic interpolation of the

% peak defined by three consecutive data values in the

% three-element vector z. The middle value, z(2), must be

% the largest element. No checking is done of this

% requirement. Real data is assumed. There is no

% guarding against edge effects, either.

% peakinterp returns the interpolated peak amplitude and

% the interpolated peak location relative to the center

% element in range bins, e.g. index = -0.4 means the

% interpolated peak lies 0.4 samples to the "left" of the

% center element, i.e. at sample #1.6, so to speak.

%

% Mark Richards, February 1997

del\_index = -0.5\*(z(3) - z(1))/(z(1) - 2\*z(2) + z(3));

k=del\_index;

amp = 0.5\*((k-1)\*k\*z(1) - 2\*(k-1)\*(k+1)\*z(2) + (k+1)\*k\*z(3));

1. The relative RCS values are not nice round numbers in most cases because the program that creates the data actually works on relative SNR values as inputs; relative RCS values are then derived from the SNR and range of each target. [↑](#footnote-ref-1)
2. Most of my contour plots are done using contour levels of -1, -5, -10, -15, -20, -25, and -30 dB relative to the peak of the data being plotted. [↑](#footnote-ref-2)
3. Clutter is white (uncorrelated) in the fast time dimension, but not in the slow time dimension. Here we are processing in fast time. [↑](#footnote-ref-3)