Binary Integration Gain

Mark A. Richards
September 2016

1 Acknowledgement

Thanks to Gabriel Beltrão for bringing this issue to my attention, and for providing an independent check of the calculations.

2 Introduction

Binary integration (BI), also called M-of-N processing or integration, is a technique for combining multiple threshold detection test outcomes to form a single, final detection test outcome that achieves specified probabilities of detection and false alarm, P_D and P_{FA}, with a lower single-sample signal-to-noise ratio (SNR) than would be required for a single threshold test based on a single measurement. Alternatively, for a given SNR, binary integration can achieve lower P_{FA} and/or higher P_D than a detection test using a single measurement. A description of the basic technique is given in section 6.4 of [1] and will not be repeated here.

A shortcoming of the discussion in [1] is that it only computes the effect of the processing on the post-BI probabilities given a single-trial probability, and then suggests that the best choice of M for a given N is the one that maximizes the range of pre-BI detection probabilities for which the post-BI detection probability is increased. (Any reasonable pre-BI false alarm probability will be decreased for any choice of M except M = 1.) A more useful basis for evaluation would be to compute a binary integration gain G_{BI}, i.e. the factor by which the required single-measurement SNR needed to achieve a given P_D and P_{FA} is decreased when using BI, vs. that required when detection is based on only a single measurement. This is in exact analogy to the idea of both coherent and noncoherent integration. Accordingly, the goal of this memo is to develop a simple numerical procedure for evaluating G_{BI} for any M and N and compare the result to the corresponding noncoherent and coherent integration gains.

In the discussion that follows, the target is always assumed to be nonfluctuating. The Swerling target fluctuations case is commented on briefly in Section 5.3.

3 Procedure for Computing Binary Integration Gain

Let P_D and P_{FA} be the desired final probabilities of detection and false alarm, whether one is using a single threshold test, or using binary integration. If using BI, let the number of individual threshold tests to be combined be N, and let the decision rule be that a detection is declared if and only if at least M or more of the N individual threshold tests indicate a detection, where M can be between 1 and N. The detections need not be continguous. For example, a 3-of-8 BI rule (M = 3, N = 8) would conduct eight

separate threshold tests, and would only declare a detection if the threshold were crossed on any 3 or more of those tests. When using the BI procedure, let the probabilities of detection and false alarm for a single trial be P_{D1} and P_{FA1}, and note that the desired probabilities after BI are just P_D and P_{FA} . Let the reference SNR required to achieve a specified P_D and P_{FA} using a single measurement be χ, and the SNR required to achieve a specified P_{D1} and P_{FA1} on a single measurement be χ_1. We restrict ourselves to the following additional conditions, mainly because they comprise the easiest form of the problem:

- Each of the N BI threshold tests is statistically independent of the others.
- The SNR of the data in each of those N tests is identical. This effectively assumes a nonfluctuating target and no significant changes in the data collection scenario over the N measurements, e.g. no changes in antenna gain due to scanning, no range changes, etc.
- The interference is circular white Gaussian noise.
- A square law detector is used. This is necessary because we use Shnidman's SNR equation (Section 6.3.5 in [1]) to estimate the required SNRs, and it assumes a square law detector. Albersheim's SNR equation could also be used [1]; doing so effectively assumes a linear detector. There is little difference in the outcomes.

The first two conditions in this list mean that when using BI, the "cumulative" probability of detection P_D is related to the single-trial detection probability P_{D1} according to Eq. (6.116) of [1], repeated here in the notation of this memo:

$$P_D = \sum_{r=M}^{N} \binom{N}{r} P_{D1}^r \left(1 - P_{D1}\right)^{N-r} \tag{1}$$

An identical equation relates P_{FA} and P_{FA1}. A numerical procedure for computing the binary integration gain is straightforward:

1. Given desired values of M, N, P_D, and P_{FA}:
    a. Find the reference SNR χ required to achieve P_D and P_{FA} using a single measurement and threshold test. This is easily done to a good approximation (0.5 dB or better in most cases) in closed form using Shnidman's SNR equation. Alternatively, a numerical search can be performed using the more exact Marcum's Q function, but that seems overkill for this analysis.
    b. Solve Eq. (1) for the value of P_{D1} required to achieve the specified P_D. This must be done numerically.[1]
    c. Repeat (b) to find the value of P_{FA1} required to achieve the specified P_{FA}.
    d. Find the SNR χ_1 required to achieve the single-trial probabilities P_{FA1} and P_{D1} on a single measurement and threshold test.
    e. The binary integration gain G_{BI} is the ratio χ/χ_1. Convert to decibels if desired.
2. Repeat for various values of M, N, P_D, and P_{FA} as needed.

[1] In MATLAB®, the solution can be accomplished very efficiently by appropriate use of the `fzero` function.

4 Example

Figure 1 illustrates the binary integration gain for a square-law detector, $P_{FA} = 10^{-6}$, $N = 8$, and all choices of M (1 through 8). Qualitatively similar results are obtained for other P_{FA} values from 10^{-2} to 10^{-8}, and for N from 3 through 7. Following is an example of computing one point on this chart, namely the case of 3-of-8 processing with a P_{FA} of 10^{-6} and P_D of 0.7:

1. Shnidman's SNR equation gives the SNR required to achieve this performance with a single measurement as $\chi = 15.97$ on a linear scale, which is 12.03 dB.
2. The single-trial probability P_{FA1} required to achieve $P_{FA} = 10^{-6}$ with 3-of-8 binary integration is found by a numerical search to be 0.0026.
3. Similarly, the single-trial probability P_{D1} required to achieve $P_D = 0.7$ with 3-of-8 binary integration is found by a numerical search to be 0.4075.
4. Shnidman's SNR equation gives the SNR required to achieve $P_{FA1} = 0.0026$ and $P_{D1} = 0.4075$ with a single measurement to be $\chi_1 = 4.78 = 6.8$ dB.
5. The binary integration gain for this case is $G_{BI} = \chi/\chi_1 = 3.34 = 5.24$ dB. Within roundoff errors, this of course is also equal to the difference of the two SNRs in steps 1 and 4 in dB, i.e. $12.03 - 6.8 = 5.23$ dB.

The data point in Figure 1 corresponding to this example is shown by the red circle.

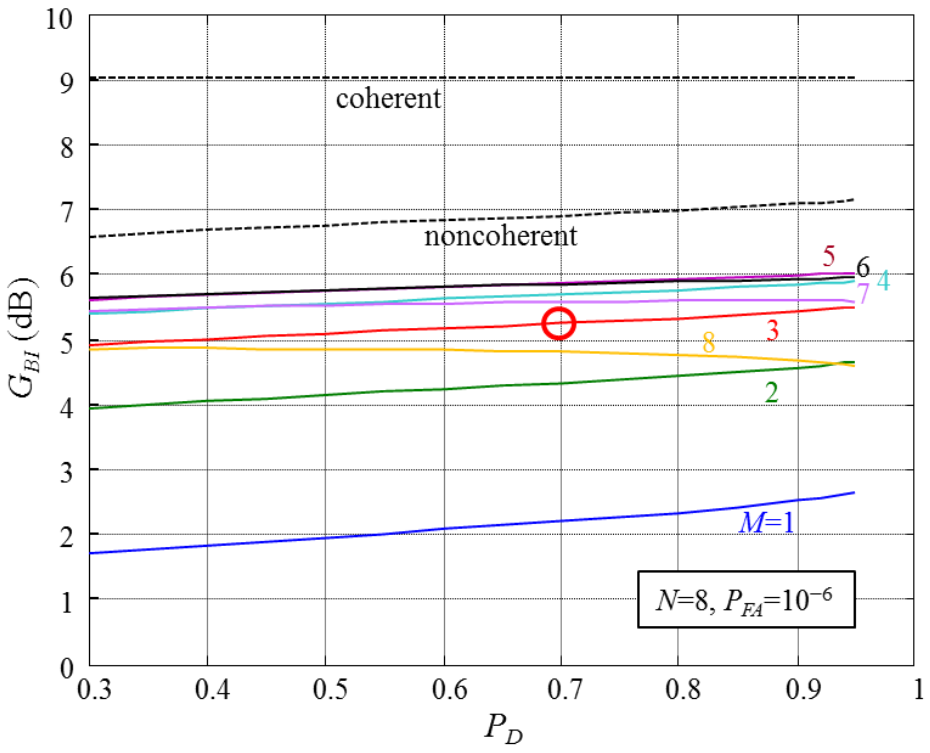

Figure 1. Binary integration gain G_{BI} due to binary integration for $N = 8$ and all possible values of M. In all cases, $P_{FA} = 10^{-6}$. The circled data point corresponds to the numerical example above. Also shown are the noncoherent gain G_{nc} and the coherent integration gain G_c for $N = 8$.

5 Discussion

5.1 Best Choice of M

An obvious question is which value of M is best, in the sense of providing the largest integration gain. From the data shown in Figure 1, it would appear that the optimum choice of M for N = 8 and $P_{FA} = 10^{-6}$ is M_{opt} = 5 or 6. The choice of M is not very critical; in this example, the values 3, 4, and 7 also provide gains within 1 dB or less of the maximum. The values also do not vary with P_{FA} ranging from 10^{-2} to 10^{-8}. Similar broad maxima with respect to M and insensitivity to P_{FA} is observed for other values of N between 3 and 8.

In [3], Shnidman considers binary integration gain. He defines the optimum value of M to be that which maximizes P_D for a given N, P_{FA}, SNR χ, and target fluctuation model. This is equivalent to our definition of the optimum M as the one that maximizes integration gain, since that corresponds to minimizing the required χ for a given P_D, P_{FA}, and N. He observes a similar broad maximum in the value of M_{opt} over all of the target models considered as well as a much broader range of N (up to 1000) than considered here. He also proposes an empirical estimate of M_{opt} for the nonfluctuating target:

$$M_{opt} = \texttt{round}\left(10^{-0.02}\, N^{0.8}\right) = \texttt{round}\left(0.955\, N^{0.8}\right) \quad \text{(nonfluctuating target)} \tag{2}$$

The `round(·)` function was added here to give an integer result, as required in practice. This estimate is stated to be valid for N from 5 to 700. Table 1 shows that the value of M_{opt} predicted by Eqn. (2) usually matches those observed from calculations like those leading to Figure 1.

Table 1. Observed and predicted values of M_{opt} for a nonfluctuating target with $P_{FA} = 10^{-6}$.

	N = 3	4	5	6	7	8
Observed M_{opt}	2	3	4	4	5	5
Eqn. (2)	2	3	3	4	5	5

5.2 Comparison to Coherent and Noncoherent Integration

Also shown in Figure 1 are the curves for coherent integration gain G_c and noncoherent integration gain G_{nc} using N = 8 samples. The coherent integration gain is just a factor of N. The noncoherent integration gain is computed using Shnidman's SNR equation. Figure 2 illustrates the amount by which the binary integration gain (using the appropriate M_{opt} for each N) is reduced compared to the noncoherent integration gain that could be achieved for the same value of N and a nonfluctuating target, i.e. $G_{nc} - G_{BI}$. For the range of N shown, the loss in integration gain due to the use of binary integration is on the order of 1 ± 0.15 dB. In [3], it is shown that in fact the loss is less than 1.5 dB for a much wider range of N than considered here.

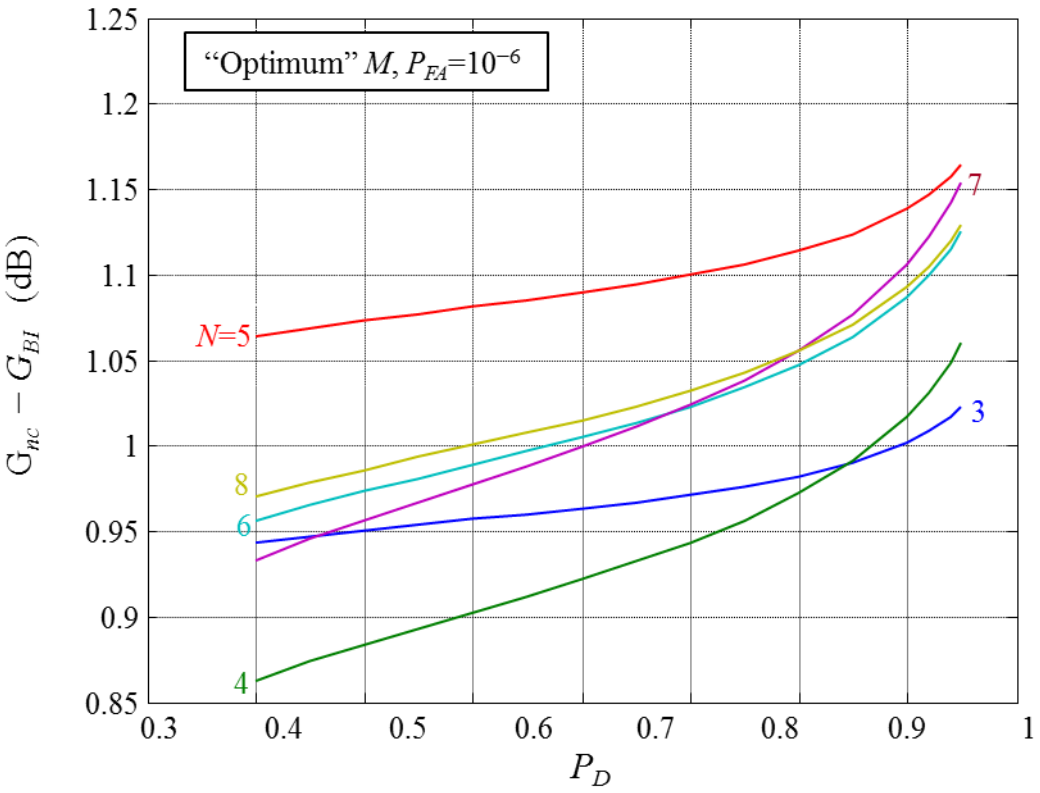

Figure 2. Reduction in integration gain ("loss") when using binary integration with the optimum value of M, as compared to using noncoherent integration for a nonfluctuating target and P_{FA} = 10^{-6}.

5.3 Fluctuating Targets

In [3], Shnidman also addresses the value of M_{opt} and the reduction in integration gain relative to noncoherent integration for the four Swerling target fluctuation models. The estimated value of M_{opt} for other target models follows the same functional form as Eqn. (2) but with different exponent values. It is also shown that the loss $G_{nc} - G_{BI}$ is still less than 1.5 dB for a wide range of N and all of the Swerling model targets. See [3] for details regarding these issues.

5.4 Data Usage Priorities

The main advantages of binary integration are simplicity and robustness. It achieves an integration gain within 1 to 1.5 dB of noncoherent integration with a somewhat simpler implementation. It also, like noncoherent integration, is more robust than coherent integration in that it does not depend on maintaining coherence of the target echo components, which can be difficult. However, the noncoherent integration gain G_{nc} exceeds the best-choice binary integration gain G_{BI} in every case examined, and the coherent integration gain G_c in turn exceeds G_{nc}.[2] Thus, given multiple

[2] There are some cases with Swerling 2 and 4 targets where the noncoherent integration gain actually exceeds the coherent integration gain. However, even in these cases the SNR required to achieve a given detection

measurements of a nonfluctuating target in noise, and assuming the goal is to detect the target's presence with the minimum required SNR, coherent integration should be performed first to the maximum extent possible. If coherent integration is not feasible or if it is limited to fewer than N samples at a time (perhaps due to radar-target motion-induced phase errors, transceiver phase instabilities, etc.), then there will still be $N' \leq N$ samples available to be combined. The results above show that these samples should be noncoherently integrated if possible. Finally, if noncoherent integration of the remaining samples is not feasible, binary integration can be applied with only a minor additional loss.

It is worth noting that binary integration is often combined with multiple pulse repetition frequency (PRF) data acquisition to achieve more than just detection. In particular, collecting N coherent processing intervals (CPIs) of data in a pulse Doppler radar, each at a different PRF; applying appropriate threshold detection processing to each CPI; and then applying binary integration across the CPIs in each range-Doppler cell not only achieves an integration gain for detection but can also be used to reduce or avoid range-Doppler blind zones and resolve range and Doppler ambiguities. See [1] for an introduction to this style of processing.

6 References

[1] M. A. Richards, *Fundamentals of Radar Signal Processing*, second edition. McGraw-Hill, 2014.

[2] M. A. Richards, "Notes on Noncoherent Integration Gain", technical memorandum, July 17, 2014. Available at www.radarsp.com.

[3] D. A. Shnidman, "Binary Integration for Swerling Target Fluctuations", IEEE *Trans. Aerospace and Electronic Systems*, vol. 34, no. 3, pp. 1043-1053, July 1998.

7 MATLAB® Code

Versions of the following code were used to generate the figures in this memo.

```
%
% binary_int_gain_nonfluc
%
% Computes the integration gain from using binary integration. A
% nonfluctuating target is assumed. Shnidman's equation is used to
% estimate various SNRs needed, so the result is approximate but probably
% pretty good.
%
% Reference: Section 6.4 of Richards, "Fundamentals of Radar Signal
% Processing".
%
% Mark A. Richards, September 2016.

clear all
close all

% Specify N for the N-of-N (binary integration) scheme. We compute
```

performance is less with coherent integration than with noncoherent integration. A discussion of this phenomenon can be found in [2].

```
% results for all choices of M from M=1 (1-of-N) to M=N (N-of-N). We'll
% also loop over multiple values of N.
NN = 3:8';
NNlen = length(NN);

% Specify Pfa and Pd ranges
PFA = 10.^(-[2:8]');
PFAlen = length(PFA);

PD = [0.3:0.05:0.9,0.92,0.94,0.95];
% PD = [0.3 0.4 0.5 0.6 0.7 0.8 0.9 0.95]';
PDlen = length(PD);

% Let's loop over PFA, then PD, then N, then M

% matrix to hold the results
gain_BI_dB = zeros(PFAlen,PDlen,NNlen,max(NN));
gain_noncoh_dB = zeros(PFAlen,PDlen,NNlen);
gain_coh_dB = zeros(PFAlen,PDlen,NNlen);

for PFAn = 1:PFAlen
    Pfa = PFA(PFAn);

    for PDn = 1:PDlen
        Pd = PD(PDn);

        % Compute the SNR required to achieve this Pd and Pfa with a single
        % test, i.e. not using M-of-N processing.
        SNR_1 = shnidman(1,Pd,Pfa,0);
        SNR_1dB = 10*log10(SNR_1);

        for Nn = 1:NNlen
            N = NN(Nn);

            % Compute the SNR for achieving this Pd and Pfa using N samples
            % and noncoherent integration; use that to compute noncoherent
            % integration gain for this case
            SNR_N = shnidman(N,Pd,Pfa,0);
            SNR_NdB = 10*log10(SNR_N);
            gain_noncoh_dB(PFAn,PDn,Nn) = SNR_1dB - SNR_NdB;
            gain_coh_dB(PFAn,PDn,Nn) = 10*log10(N);

            for M = 1:N
                M;

                % find required single-trial Pfa1 such that the cumulative
                % false alarm probability is the desired value Pfa.  This
                % requires a numerical search, but we can make an excellent
                % initial guess. See reference.
                k = factorial(N)/factorial(N-M)/factorial(M);
                p0 = (Pfa/k)^(1/M);
                fun = @(p) (MofN_probability(p,M,N) - Pfa);
                Pfa1 = fzero(fun,p0);

                % Repeat for detection probability.  The initial guess is
                % nowhere near as accurate in this case, but still seems to
                % work OK.
                fun = @(p) (MofN_probability(p,M,N) - Pd);
                Pd1 = fzero(fun,[0 1]);

                % Now use Shnidman's equation to get the SNR required to
                % achieve these single-trial probabilities. This is the SNR
                % needed to meet the overall PD and PFA goal with the
```

```
                % M-of-N method.
                SNR_MN = shnidman(1,Pd1,Pfa1,0);
                SNR_MNdB = 10*log10(SNR_MN);

                % Compute the integration gain due to the M-of-N scheme
                gain_BI_dB(PFAn,PDn,Nn,M) = SNR_1dB - SNR_MNdB;

                %                 if (M==3) & (N==8) & (Pd == 0.7) & (Pfa == 1e-6)
                %                     M
                %                     N
                %                     Pd
                %                     Pfa
                %                     Pfa1
                %                     Pd1
                %                     SNR_MN
                %                     SNR_MNdB
                %                     SNR_1dB - SNR_MNdB
                %                 end

            end % of loop over M

        end % of loop over Nn

    end % of loop over PDn

end  % of loop over PFAn

%  All of the results are in the gain(:,:,:,:) matrix.  Now some plots.

% Gain vs. Pd for fixed Pfa and different M-N combinations

% This parameter chooses which Pfa to use. PFAn = 1 for 1e-2, PFAn = 2 for
% 1e-3, ,,, , PFAn = 7 for 1e-8

% PFAn = 7;  % Pfa = 1e-8
% PFAn = 6;  % Pfa = 1e-7
PFAn = 5;  % Pfa = 1e-6
% PFAn = 4;  % Pfa = 1e-5
% PFAn = 3;  % Pfa = 1e-4
% PFAn = 2;  % Pfa = 1e-3
% PFAn = 1;  % Pfa = 1e-2

% plot the M-of-3 cases
gain_coh_dB(:,:,1) = 10*log10(3);
figure
x = PD;
y = [gain_BI_dB(PFAn,:,1,1); % Pfa = 1e-6, all Pd values, N = 3, M = 1
    gain_BI_dB(PFAn,:,1,2); % Pfa = 1e-6, all Pd values, N = 3, M = 2
    gain_BI_dB(PFAn,:,1,3); % Pfa = 1e-6, all Pd values, N = 3, M = 3
    gain_noncoh_dB(PFAn,:,1); % noncoherent gain, Pfa = 1e-6, all Pd values, N = 3
    gain_coh_dB(PFAn,:,1)]; % coherent gain, N = 3
plot(x,y')
xlim([0.3 1]);
ylim([0,10]);
grid
xlabel('Pd')
ylabel('Gain (dB)')
title(['M-of-3, Pfa = ',num2str(PFA(PFAn))])
legend('1 of 3','2 of 3','3 of 3','Noncoherent','Coherent')

% plot the M-of-4 cases
gain_coh_dB(:,:,2) = 10*log10(4);
figure
```

```
x = PD;
y = [gain_BI_dB(PFAn,:,2,1); % Pfa = 1e-6, all Pd values, N = 4, M = 1
    gain_BI_dB(PFAn,:,2,2); % Pfa = 1e-6, all Pd values, N = 4, M = 2
    gain_BI_dB(PFAn,:,2,3); % Pfa = 1e-6, all Pd values, N = 4, M = 3
    gain_BI_dB(PFAn,:,2,4); % Pfa = 1e-6, all Pd values, N = 4, M = 4
    gain_noncoh_dB(PFAn,:,2); % noncoherent gain, Pfa = 1e-6, all Pd values, N = 4
    gain_coh_dB(PFAn,:,2)]; % coherent gain, N = 4
plot(x,y')
xlim([0.3 1]);
ylim([0,10]);
grid
xlabel('Pd')
ylabel('Gain (dB)')
title(['M-of-4, Pfa = ',num2str(PFA(PFAn))])
legend('1 of 4','2 of 4','3 of 4','4 of 4','Noncoherent','Coherent')

% plot the M-of-5 cases
gain_coh_dB(:,:,3) = 10*log10(5);
figure
x = PD;
y = [gain_BI_dB(PFAn,:,3,1); % Pfa = 1e-6, all Pd values, N = 5, M = 1
    gain_BI_dB(PFAn,:,3,2); % Pfa = 1e-6, all Pd values, N = 5, M = 2
    gain_BI_dB(PFAn,:,3,3); % Pfa = 1e-6, all Pd values, N = 5, M = 3
    gain_BI_dB(PFAn,:,3,4); % Pfa = 1e-6, all Pd values, N = 5, M = 4
    gain_BI_dB(PFAn,:,3,5); % Pfa = 1e-6, all Pd values, N = 5, M = 5
    gain_noncoh_dB(PFAn,:,3); % noncoherent gain, Pfa = 1e-6, all Pd values, N = 5
    gain_coh_dB(PFAn,:,3)]; % coherent gain, N = 4
plot(x,y')
xlim([0.3 1]);
ylim([0,10]);
grid
xlabel('Pd')
ylabel('Gain (dB)')
title(['M-of-5, Pfa = ',num2str(PFA(PFAn))])
legend('1 of 5','2 of 5','3 of 5','4 of 5','5 of 5','Noncoherent','Coherent')

% plot the M-of-6 cases
gain_coh_dB(:,:,4) = 10*log10(6);
figure
x = PD;
y = [gain_BI_dB(PFAn,:,4,1); % Pfa = 1e-6, all Pd values, N = 6, M = 1
    gain_BI_dB(PFAn,:,4,2); % Pfa = 1e-6, all Pd values, N = 6, M = 2
    gain_BI_dB(PFAn,:,4,3); % Pfa = 1e-6, all Pd values, N = 6, M = 3
    gain_BI_dB(PFAn,:,4,4); % Pfa = 1e-6, all Pd values, N = 6, M = 4
    gain_BI_dB(PFAn,:,4,5); % Pfa = 1e-6, all Pd values, N = 6, M = 5
    gain_BI_dB(PFAn,:,4,6); % Pfa = 1e-6, all Pd values, N = 6, M = 6
    gain_noncoh_dB(PFAn,:,4); % noncoherent gain, Pfa = 1e-6, all Pd values, N = 6
    gain_coh_dB(PFAn,:,4)]; % coherent gain, N = 6
plot(x,y')
xlim([0.3 1]);
ylim([0,10]);
grid
xlabel('Pd')
ylabel('Gain (dB)')
title(['M-of-6, Pfa = ',num2str(PFA(PFAn))])
legend('1 of 6','2 of 6','3 of 6','4 of 6','5 of 6','6 of 6', ...
    'Noncoherent','Coherent')

% plot the M-of-7 cases
gain_coh_dB(:,:,5) = 10*log10(7);
figure
x = PD;
y = [gain_BI_dB(PFAn,:,5,1); % Pfa = 1e-6, all Pd values, N = 7, M = 1
```

```
    gain_BI_dB(PFAn,:,5,2); % Pfa = 1e-6, all Pd values, N = 7, M = 2
    gain_BI_dB(PFAn,:,5,3); % Pfa = 1e-6, all Pd values, N = 7, M = 3
    gain_BI_dB(PFAn,:,5,4); % Pfa = 1e-6, all Pd values, N = 7, M = 4
    gain_BI_dB(PFAn,:,5,5); % Pfa = 1e-6, all Pd values, N = 7, M = 5
    gain_BI_dB(PFAn,:,5,6); % Pfa = 1e-6, all Pd values, N = 7, M = 6
    gain_BI_dB(PFAn,:,5,7); % Pfa = 1e-6, all Pd values, N = 7, M = 7
    gain_noncoh_dB(PFAn,:,5); % noncoherent gain, Pfa = 1e-6, all Pd values, N = 7
    gain_coh_dB(PFAn,:,5)]; % coherent gain, N = 7
plot(x,y')
xlim([0.3 1]);
ylim([0,10]);
grid
xlabel('Pd')
ylabel('Gain (dB)')
title(['M-of-7, Pfa = ',num2str(PFA(PFAn))])
legend('1 of 7','2 of 7','3 of 7','4 of 7','5 of 7','6 of 7','7 of 7', ...
    'Noncoherent','Coherent')

% plot the M-of-8 cases
gain_coh_dB(:,:,6) = 10*log10(8);
figure
x = PD;
y = [gain_BI_dB(PFAn,:,6,1); % Pfa = 1e-6, all Pd values, N = 8, M = 1
    gain_BI_dB(PFAn,:,6,2); % Pfa = 1e-6, all Pd values, N = 8, M = 2
    gain_BI_dB(PFAn,:,6,3); % Pfa = 1e-6, all Pd values, N = 8, M = 3
    gain_BI_dB(PFAn,:,6,4); % Pfa = 1e-6, all Pd values, N = 8, M = 4
    gain_BI_dB(PFAn,:,6,5); % Pfa = 1e-6, all Pd values, N = 8, M = 5
    gain_BI_dB(PFAn,:,6,6); % Pfa = 1e-6, all Pd values, N = 8, M = 6
    gain_BI_dB(PFAn,:,6,7); % Pfa = 1e-6, all Pd values, N = 8, M = 7
    gain_BI_dB(PFAn,:,6,8); % Pfa = 1e-6, all Pd values, N = 8, M = 8
    gain_noncoh_dB(PFAn,:,6); % noncoherent gain, Pfa = 1e-6, all Pd values, N = 8
    gain_coh_dB(PFAn,:,6)]; % coherent gain, N = 8
plot(x,y')
xlim([0.3 1]);
ylim([0,10]);
grid
xlabel('Pd')
ylabel('Gain (dB)')
title(['M-of-8, Pfa = ',num2str(PFA(PFAn))])
legend('1 of 8','2 of 8','3 of 8','4 of 8','5 of 8','6 of 8','7 of 8', ...
    '8 of 8','Noncoherent','Coherent')


% Now let's use our data to see what the integratino loss is compared to
% noncoherent. Generate a plot of the difference in integration gain
% between the noncoherent case and the best binary case, as a function of
% Pd, for a fixed Pfa.

loss_NCmBI_dB = zeros(PDlen,NNlen);
for Nn = 1:NNlen
    Mopt = round((10^-0.02)*NN(Nn)^0.8)
    for PDn = 1:PDlen
        %         loss_NCmBI_dB(PDn,Nn) = gain_noncoh_dB(PFAn,PDn,Nn) -
max(gain_BI_dB(PFAn,PDn,Nn,:));
        loss_NCmBI_dB(PDn,Nn) = gain_noncoh_dB(PFAn,PDn,Nn) -
gain_BI_dB(PFAn,PDn,Nn,Mopt);
    end
end

figure
plot(PD,loss_NCmBI_dB)
grid
xlabel('Pd')
```

```
ylabel('Loss Relative to Noncoherent Integration (dB)')
title(['Optimum M, Pfa = ',num2str(PFA(PFAn))])
legend('3=N','4','5','6','7','8','Location','Best')
```