FINAL REPORT
GRANT NUMBER NGR 41-001-036
SUBMITTED TO THE
NATIONAL AERONAUTICS AND SPACE ADMINISTRATION

BY
R. W. SNELSIRE
DEPARTMENT OF ELECTRICAL AND COMPUTER ENGINEERING
CLEMSON UNIVERSITY
CLEMSON, SOUTH CAROLINA 29631

(NASA-CR-138466) [EVALUATION OF THREE CODING SCHEMES DESIGNED FOR IMPROVED DATA COMMUNICATION] Final Report (Clemson Univ.) ~~102~~ 101 p HC CSCL 09B
N74-25729
Unclas
G3/08 39739

MAY 1974

TABLE OF CONTENTS

ACKNOWLEDGEMENT

The author would like to thank Dr. H. J. Helgert and Mr. D. J. Healy, without whose work this report would not have been possible.

TABLE OF CONTENTS

INTRODUCTION

In this report, three coding schemes designed for improved data communication are evaluated. In Part A, four block codes are evaluated relative to a quality function, which is a function of both the amount of data rejected and the error rate.

Part B is an evaluation of the Viterbi Maximum Likelihood Decoding Algorithm as a decoding procedure. This evaluation is obtained by simulating the system on a digital computer.

In Part C, Short Constraint Length Rate 1/2 'Quick-Look' Codes are studied, and their performance is compared to general nonsystematic codes.

PART A

PERFORMANCE OF BLOCK CODES

I. Introduction:

Although the use of error control coding techniques in digital space communication systems has become fairly routine in recent years, there still exists a great deal of uncertainty as to the actual effectiveness of coding in achieving more reliable communication. The reason for this is to be found in the fact that the commonly used performance parameters do not take into account all the pertinent aspects of the coded transmission system. Thus, for example, the widely used Probability of Word Error criterion totally ignores the possibility that the decoder may incorporate some degree of data rejection. Likewise, the minimum distance criterion, another popular measure of code performance, is completely independent of the decoding algorithm and several other important system factors.

As a consequence of this state of affairs, it is virtually impossible to compare, say, a coding system with error correction and data rejection to one with error correction alone, using any of the existing criteria of performance, and it is therefore of value to define and evaluate measures which incorporate most, if not all, of the quantities affecting the overall system reliability. This is the objective of the present work.

II. Definition of Performance Measure:

For the simple types of block codes normally employed in space communication systems, the complexity of the encoder and decoder is of little consequence, since the use of integrated circuit technology allows the construction of the basic components in an inexpensive fashion. Furthermore, the complexity is essentially independent of the particular code-decoder used.

The processing speed is generally a function of the type of logic used and the technology in the construction of the integrated circuits. Although one could probably obtain cost figures as a function of processing speed, the importance of these costs in the overall system considerations is difficult to assess. Also, as with complexity, processing speed is not a strong function of the code-decoder combination.

Thus, the important factors determining the overall coding system performance are:

1. The accuracy of the data after decoding,
2. The amount of data rejected by the decoder,
3. The amount of redundancy in the code, and
4. The relative importance of data accuracy, data rejection, and data transmission rate.

Let us consider a situation in which N blocks of received digits from a binary (n, k) block code are to be decoded. The decoder generally rejects N-X blocks, leaving X blocks after decoding, of which Y are correct. (See Figure 1)

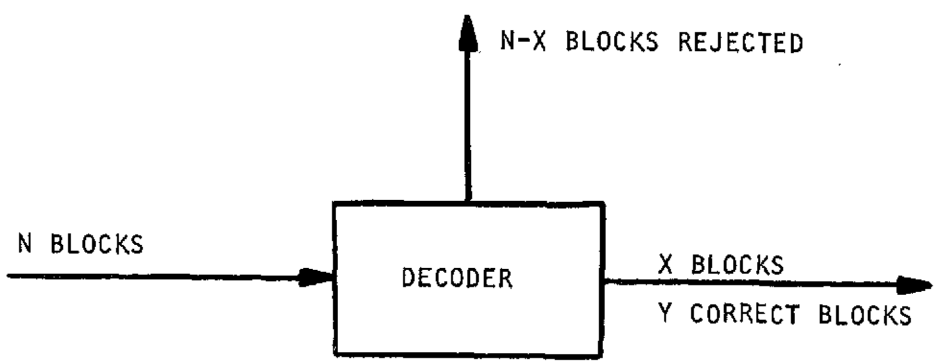


FIGURE 1. GENERAL DECODER CONFIGURATION

The amount of data passed by the decoder is measured by the quantity

$$F_1 = \frac{1}{N} E\{X\} ,$$

the accuracy of the data after decoding is measured by the quantity

$$F_2 = \frac{E\{Y\}}{E\{X\}},$$

and the amount of redundancy in the code is measured by the quantity

$$F_3 = \frac{k}{n} = \frac{\text{number of data digits per block}}{\text{total number of digits per block}},$$

Here E is the usual expectation operator.

We also define a quantity $0 \leq \alpha \leq 1$ which measures the relative importance of data accuracy and data rejection.

As an overall measure of performance of the code-decoder combination, we then take quantity

$$F = 1 - F_1{}^{\alpha} F_2{}^{(1-\alpha)}$$

as a function of the energy per information bit-to-noise ratio, E_b/N_o.

When the N blocks are transmitted independently of each other and are treated as such by the decoder, 1 - F reduces to the probability of word rejection. For a decoder with no data rejection, F_2 becomes the probability of correct decoding. Thus, in the two limiting cases $\alpha = 0$ and $\alpha = 1$, F reduces to the probability of word error and word rejection, respectively.

III. Evaluation of F for Hamming Type Block Codes Over the Binary Symmetric Channel:

We assume that N blocks are transmitted independently and with equal probability over a binary symmetric channel whose digit error probability is $p = 1-q$. The codes of interest are of two types: The standard (n, k) Hamming code described by the parity check matrix

$$H = \begin{bmatrix} 0 & 0 & . & . & . & 1 \\ 0 & 0 & . & . & . & 1 \\ . & & & & & . \\ . & & & & & . \\ 0 & 1 & . & . & . & 1 \\ 0 & 0 & . & . & . & 1 \end{bmatrix}$$

whose columns are all $2^m - 1$ nonzero binary m-tuples (m any integer greater than 2), and a modified Hamming code whose parity check matrix differs from the above only in having an additional row of ones on top. The first code has block length $n = 2^m - 1$, $k = n-m$ information digits and minimum distance 3 and is thus able to correct all single errors. The second code has the same block length, $k = n - m - 1$ and minimum distance 4 and can be decoded in either a single-error-correcting, double-error-detecting mode or a triple-error-detecting mode.

For both codes, the first step in decoding a received block $v = (v_1, v_2, . . . , v_n)$ consists of determining its syndrome. This is a binary (n-k)-tuple given by

$$s = vH^T$$

where T denotes matrix transposition and the multiplication and addition operations are modulo 2.

We now consider four cases, including, for purposes of comparison, the uncoded transmission of data blocks of length n.

Case 1. No Coding - (n, n) Code

Decoding Rule: Pass every block unchanged

Evidently, $X = E\{X\} = N$ and since a block is correct at the decoder output if and only if it is correct at the input, we have $E\{Y\} = Nq^n$

Hence $F_1 = 1$; $F_2 = q^n$

and

$$F = 1 - q^{n(1-\alpha)}$$

Since $F_3 = 1$, the relation between the channel error probability p and E_b/N_o is $p = \frac{1}{2} - \text{erf}\sqrt{2\ E_b/N_o}$

Case 2. Single-Error-Correcting Hamming Code

Decoding Rule: If the syndrome is zero, pass the block. If the syndrome is not equal to zero, assume a single error has occurred, correct it, and then pass the block.

Again, $E\{X\} = N$. For $E\{Y\}$ we have

$$E\{Y\} = N\{\text{Probability that a block has no error or a single error before decoding}\}$$

$$= N\{q^n + nq^{n-1}\ p\}$$

Therefore, $F_3 = \frac{k}{n}$, $F_1 = 1$, $F_2 = q^n + nq^{n-1}\ p$

and

$$F = 1 - (q^n + npq^{n-1})$$

where $p = \frac{1}{2} - \text{erf}\sqrt{2k/n\ \ E_b/N_o}$

Case 3. Single-Error-Correcting, Double-Error-Detecting Hamming Code

Decoding Rule: If the syndrome is zero, pass the block. If the first digit and at least one of the remaining digits in the syndrome are one, assume a single error has occurred, correct it, and then pass the block. For all other syndromes, reject the block.

We have $F_3 = \frac{k-1}{n}$,

$$F_1 = \frac{E\{X\}}{N} = \{\text{Probability that a block has zero syndrome or the first and at least one of the remaining digits equal one}\}$$

$$= \sum_{i=0}^{\frac{n-1}{2}} \{A_{2i}\ q^{n-2i}\ p^{2i} + [(2i^n + 1) - A_{2i+1}]\ q^{n-2i+1}\ p^{2i+1}\}$$

where A_j is the number of codewords of weight j of the Single-Error-Correcting Hamming Code,

and for F_2 we obtain

$$F_2 = \frac{E\{Y\}}{E\{X\}} = \frac{1}{F_1}\{\text{Probability that a received block is correct or has a single error}\}$$

Case 4. Triple-Error-Detecting Hamming Code

Decoding Rule: If the syndrome is zero, pass the block. Otherwise, reject the block.

Here, $F_3 = \frac{k-1}{n}$,

$$F_1 = \frac{E\{X\}}{N} = \{\text{Probability that a block has zero syndrome}\}$$

$$= \sum_{i=0}^{\frac{n-1}{2}} A_{2i}\ q^{n-2i}\ p^{2i} ,$$

and

$$F_2 = \frac{E\{Y\}}{E\{X\}} = \frac{q^n}{F_2}$$

The Hamming code weight spectra required for Cases 3 and 4 may be obtained as the coefficients of the polynomial.

$$f(x) = \frac{1}{n+1} \{(1+x)^n + n(1+x)^{\frac{n-1}{2}} (1-x)^{\frac{n+1}{2}}\}$$

where A_i is the coefficient of x^i.

IV. Evaluating the Performance Criterion:

A Fortran language program, reproduced in Appendix A, was written to evaluate the function F for the four cases described above. The program calculates F for 101 equally spaced values of E_b/N_o ranging from 2 db to 10 db and all values of redundancy from m = 3 to m = 10.

A major part of the program is devoted to calculating the coefficients of the function

$$f(X) = \frac{1}{n+1} ((1+X)^n + n(1+X)^{\frac{n-1}{2}} (1-X)^{\frac{n+1}{2}})$$

which are used in Cases 3 and 4. The main difficulty in this computation is that some of them have magnitudes on the order of 10^{300} for large values of n. Overflow on the IBM 370 occurs with numbers as small as 10^{77}. To overcome this problem, most calculations are done using logarithms. Thus, for example, the logarithms of the coefficients of $(1+X)^n$ are stored in an array called LGCOEF. Similarly, the components of $(1+X)^{n-1/2}$ and $(1+X)^{n+1/2}$ are stored in LCNM1 and LCNP1, respectively. Note that LCNP1 contains the logarithms of the coefficients of $(1+X)^{n+1/2}$ and not $(1-X)^{n+1/2}$, since the latter has negative coefficients whose logarithms do not exist. The variable SIGN, which always equals ± 1, is used to convert the coefficients of $(1+X)^{n-1/2}$ to the coefficients of $(1-X)^{n+1/2}$ when making calculations of f(X).

A special procedure is used throughout the program to achieve addition of these very large numbers. Obviously this addition cannot be achieved directly using logarithms. To illustrate this procedure consider adding the numbers $A = 7.3147 \times 10^{298}$ and $B = 2.1532 \times 10^{295}$ given the logarithms of these numbers.

$$ALOG = \log(A) = 298.864196 = 3.864196 + 295.$$

$$BLOG = \log(B) = 295.333084 = 0.333084 + 295.$$

Let Z = X+Y and ZLOG = log(Z).

$$\begin{aligned} ZLOG &= \log(10^{3.864196} + 10^{0.333084}) + 295 \\ &= \log(7314.7 + 2.1532) + 295 \\ &= \log(7316.8532) + 295 \\ &= 3.864324 + 295 \\ &= 298.864324 \end{aligned}$$

Thus, the log of the sum has been calculated using numbers no bigger than 7316.8532. Since the calculations on the IBM 370 have only 16 significant figures, numbers whose magnitudes differ by more than 10^{16} are not added by the above method. In this case the sum is set equal to the larger of the two numbers.

The coefficients of f(X) are calculated using the aforementioned techniques and stored in an array called RIGHT. The variable RINOM is set equal to the logs of certain binomial coefficients, and it is used in calculating terms of F1CAS3 (F_1 for Case 3; i.e., F_1 for SEC-DED) of the form

$$\left(\binom{n}{i} - A_i\right) q^{n-i} p^i.$$

These terms are stored in COEFC3.

In order to calculate the parameters for each case for any particular value of S/N, values for q and p, which are dependent on the code rate, must be evaluated. The dependence on code rate requires calculations of Q1 and P1, Q2 and P2, and Q3 and P3 for use with Cases 1, 2, and 3 and 4 respectively. Since Case 4 has the same code rate as Case 3, Q3 and P3 are applicable to both.

IV. Numerical Results and Conclusions:

In Figures 2-49, we have plotted the performance measure F as a function of the signal-to-noise ratio E_b/N_o of the binary symmetric channel in db, for

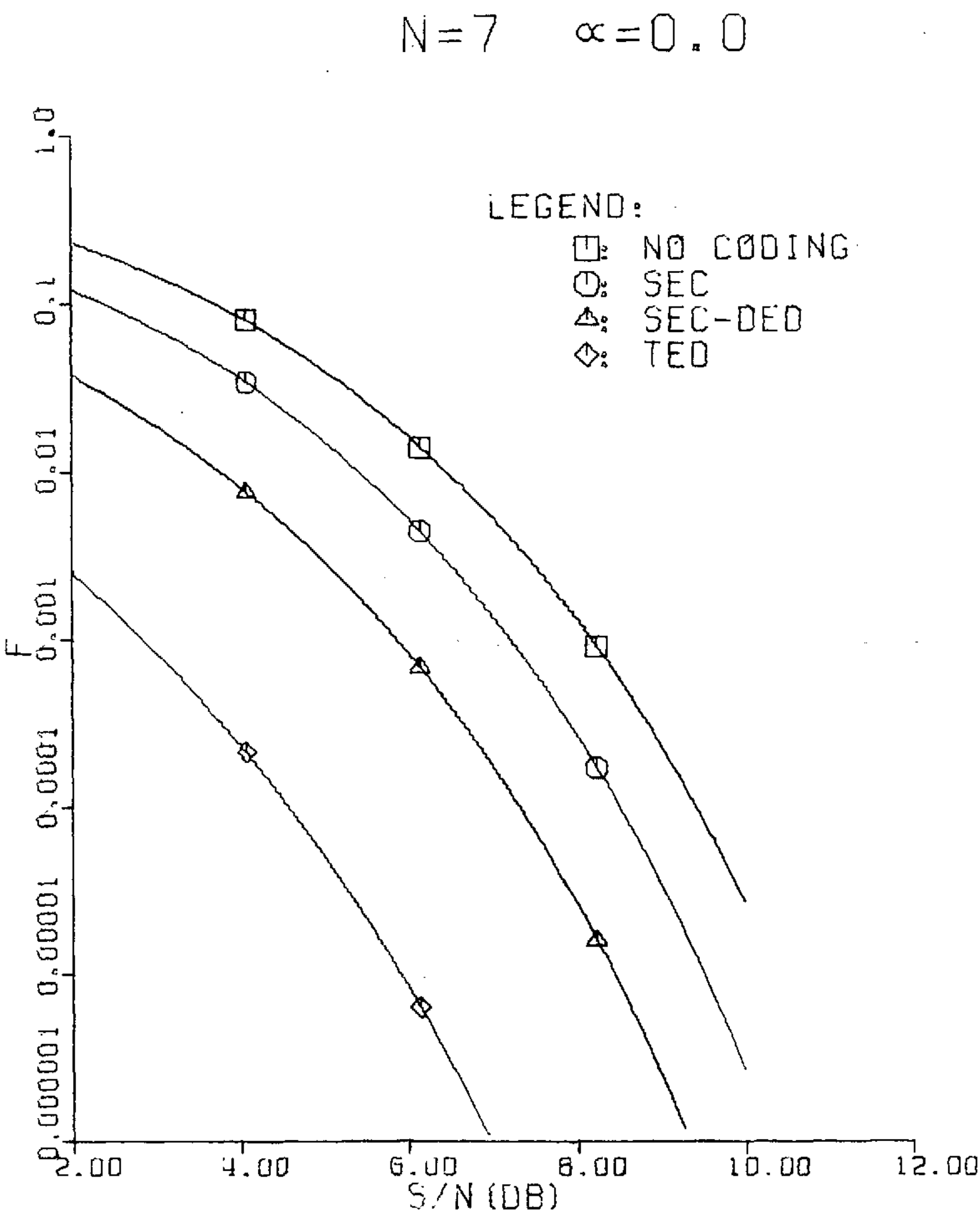
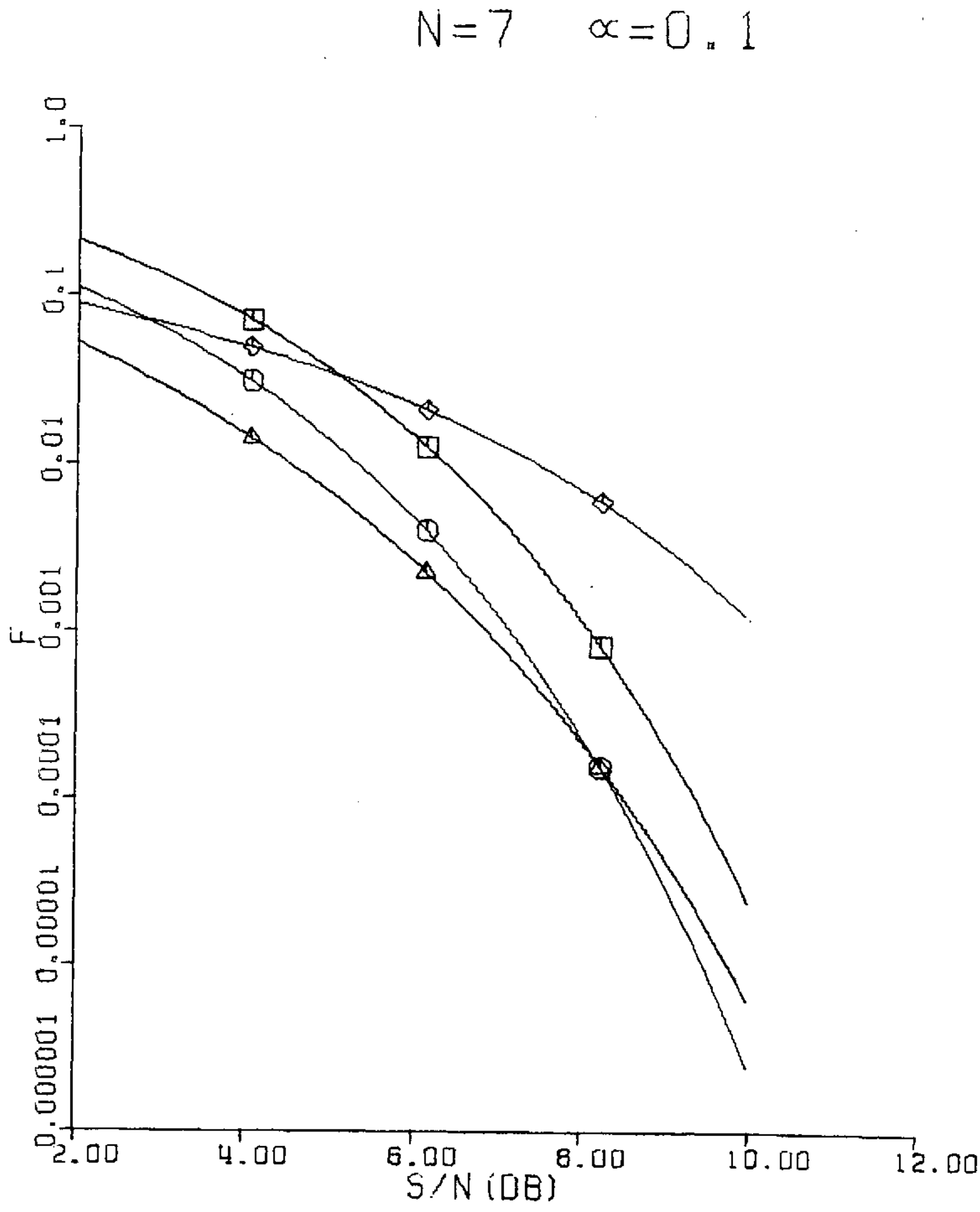
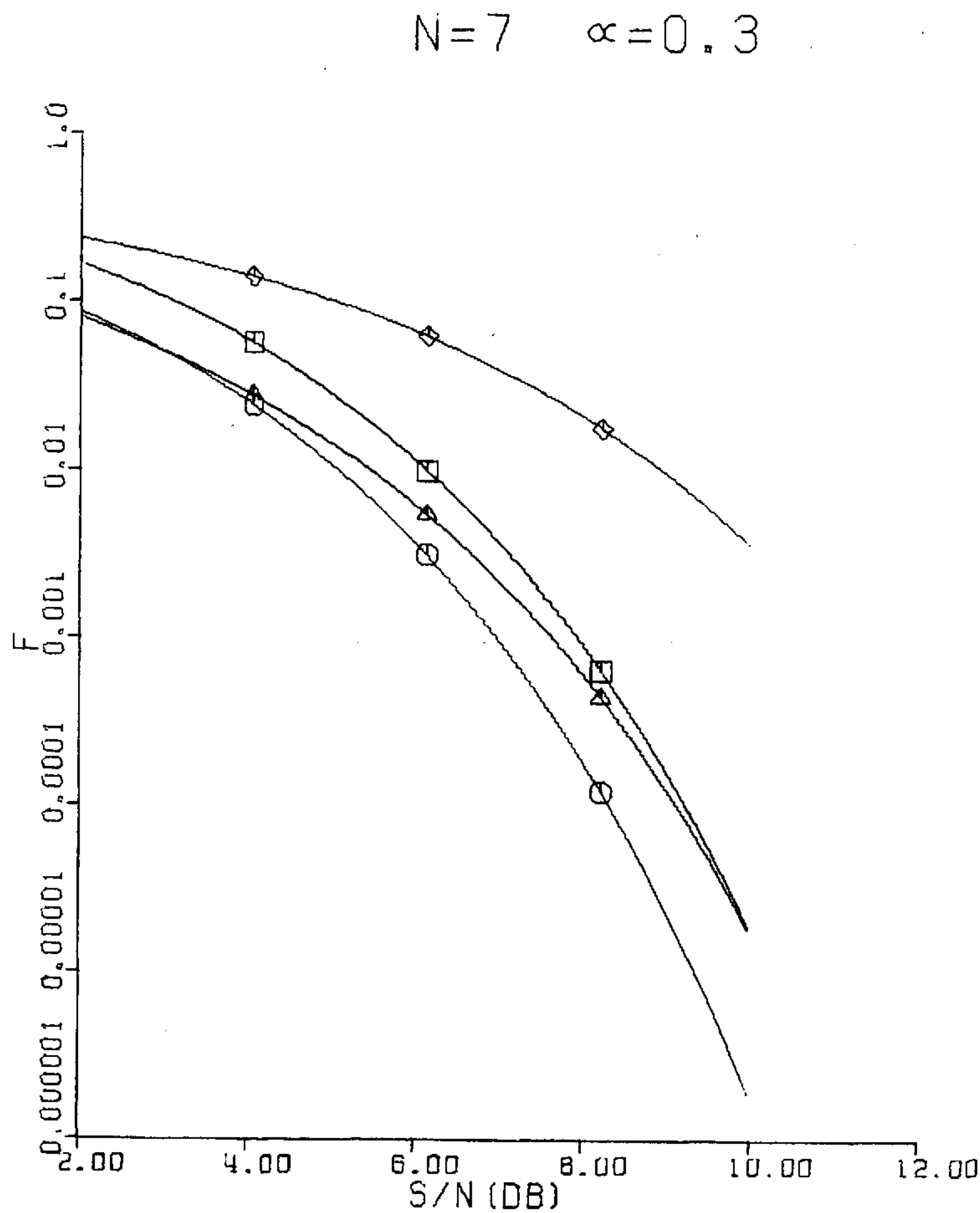

N=7 α=0.0
LEGEND:
NO CODING
SEC
SEC-DED
TED
1.0
0.1
0.01
0.001
0.0001
0.00001
0.000001
0.0000001
F
2.00
4.00
6.00
8.00
10.00
12.00
S/N (DB)

FIGURE 2



N=7 α=0.1
1.0
0.1
0.01
0.001
0.0001
0.00001
0.000001
0.
F
2.00
4.00
6.00
8.00
10.00
12.00
S/N (DB)

FIGURE 3



N=7 α=0.3
1.0
0.1
0.01
0.001
0.0001
0.00001
0.000001
0.0000001
F
2.00
4.00
6.00
8.00
10.00
12.00
S/N (DB)

FIGURE 4

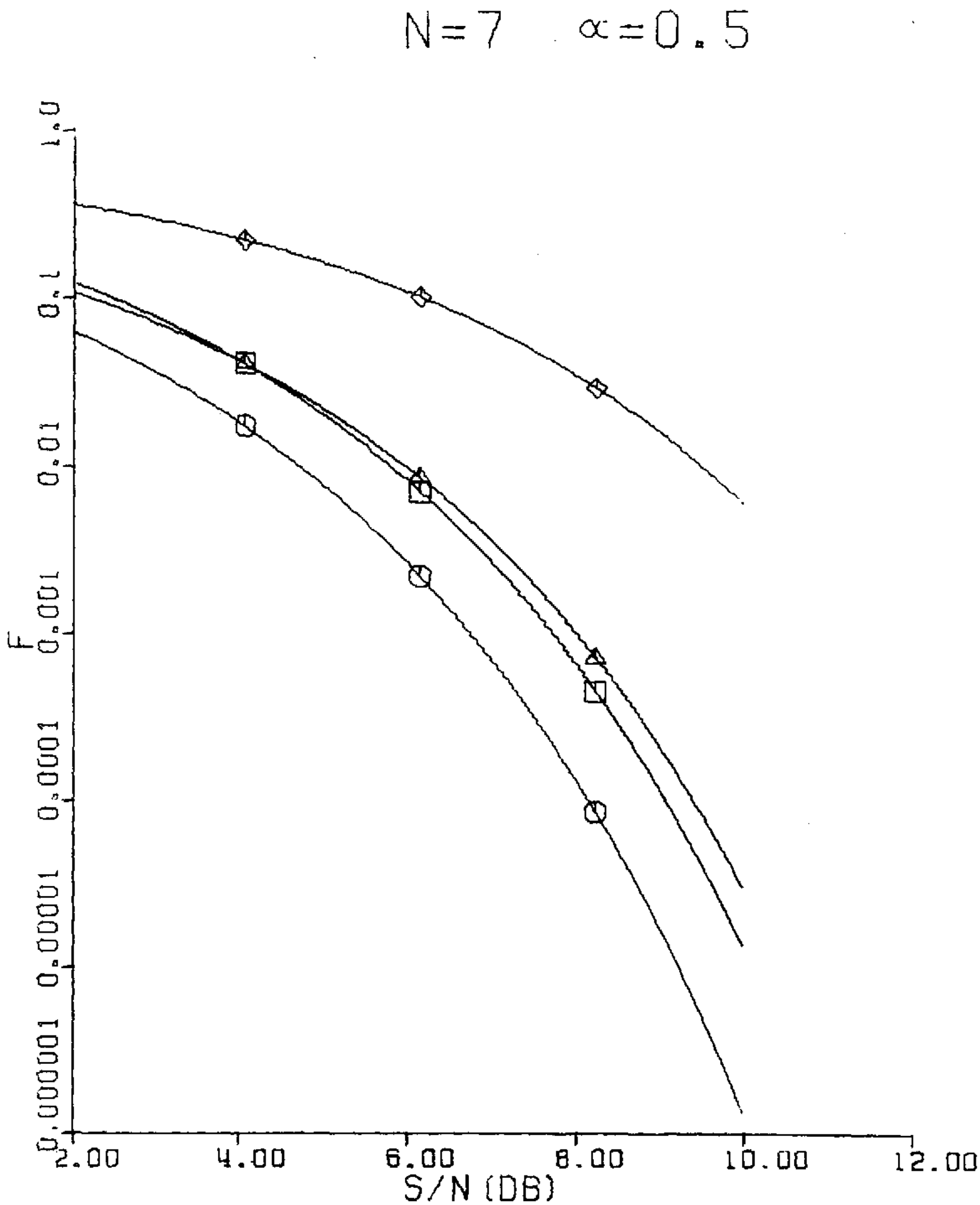

N=7 α=0.5
1.0
0.1
0.01
0.001
0.0001
0.00001
0.000001
0.0000001
F
2.00
4.00
6.00
8.00
10.00
12.00
S/N (DB)

FIGURE 5

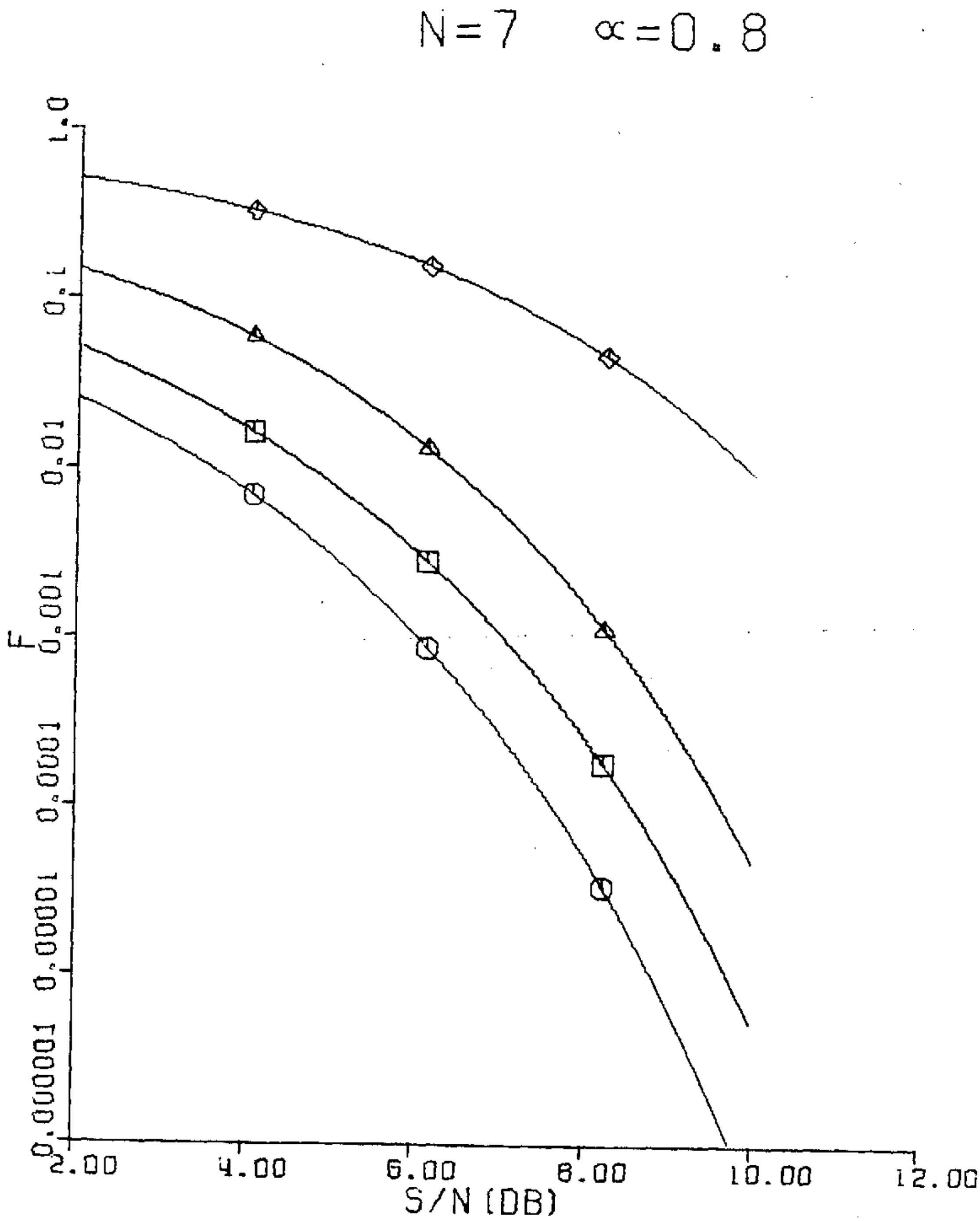
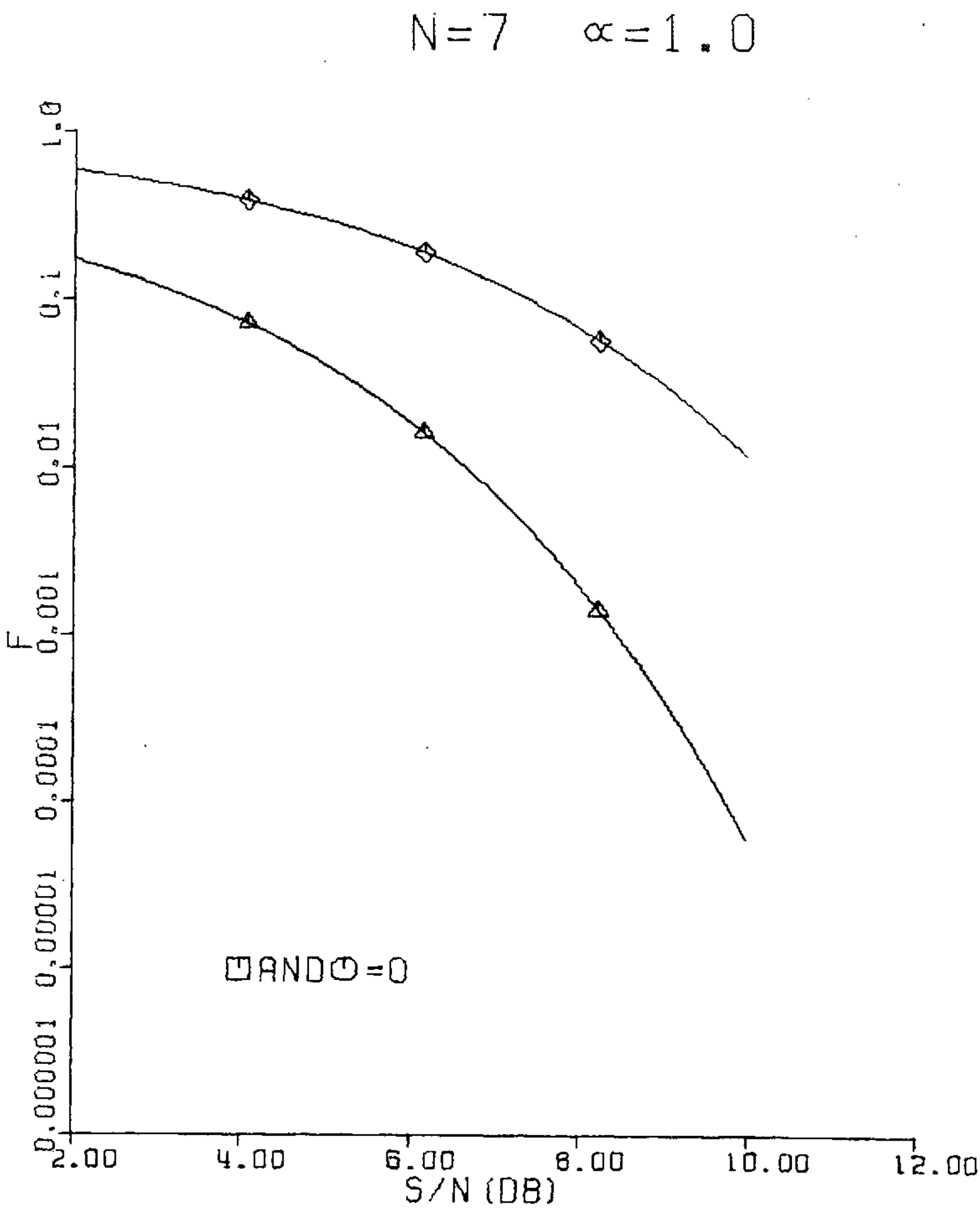

N=7 α=0.8
1.0
0.1
0.01
0.001
0.0001
0.00001
0.000001
0.0
F
2.00
4.00
6.00
8.00
10.00
12.00
S/N (DB)

FIGURE 6



N=7 α=1.0
1.0
0.1
0.01
0.001
0.0001
0.00001
0.000001
0.0000001
F
□ANDΦ=0
2.00
4.00
6.00
8.00
10.00
12.00
S/N (DB)

FIGURE 7

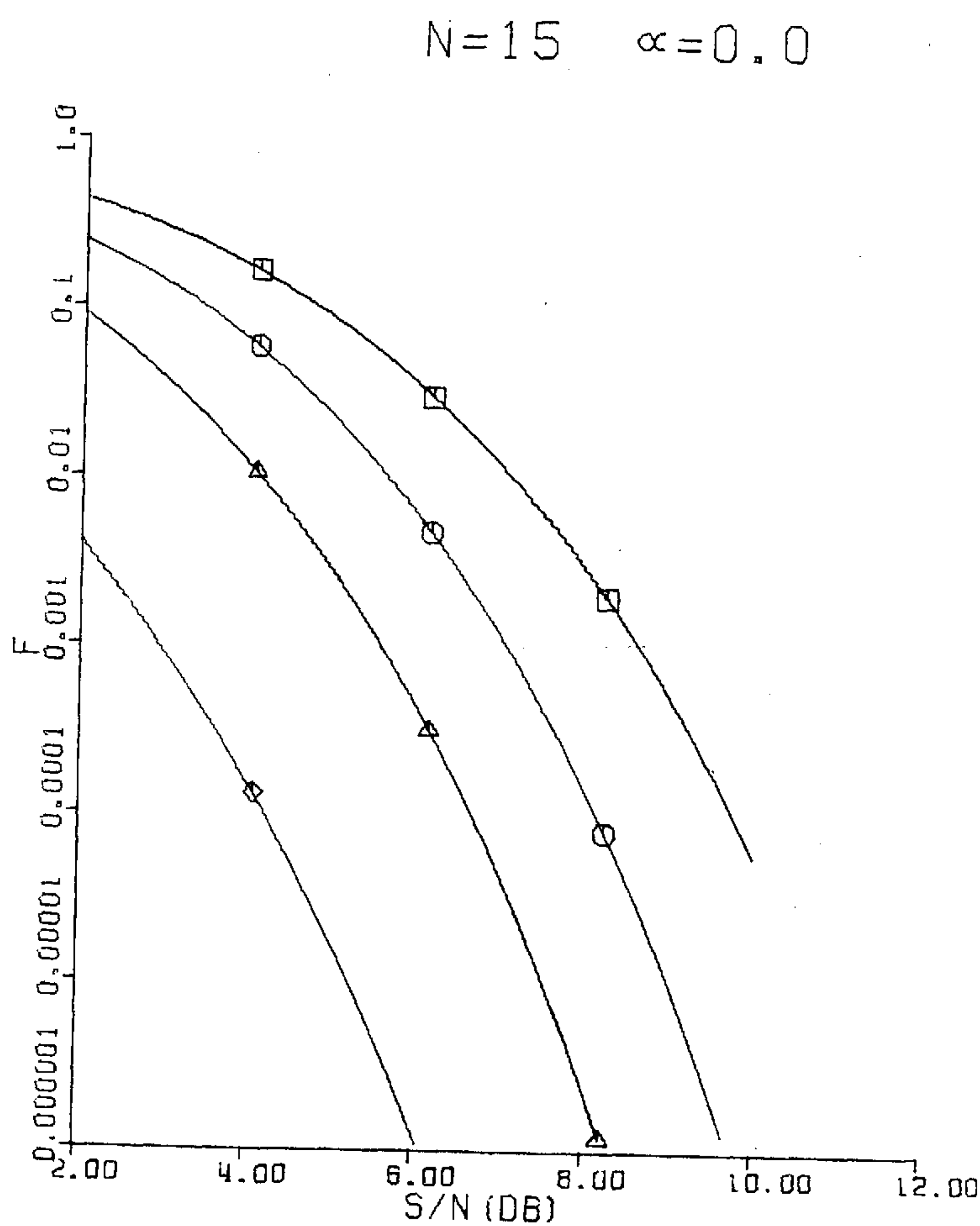
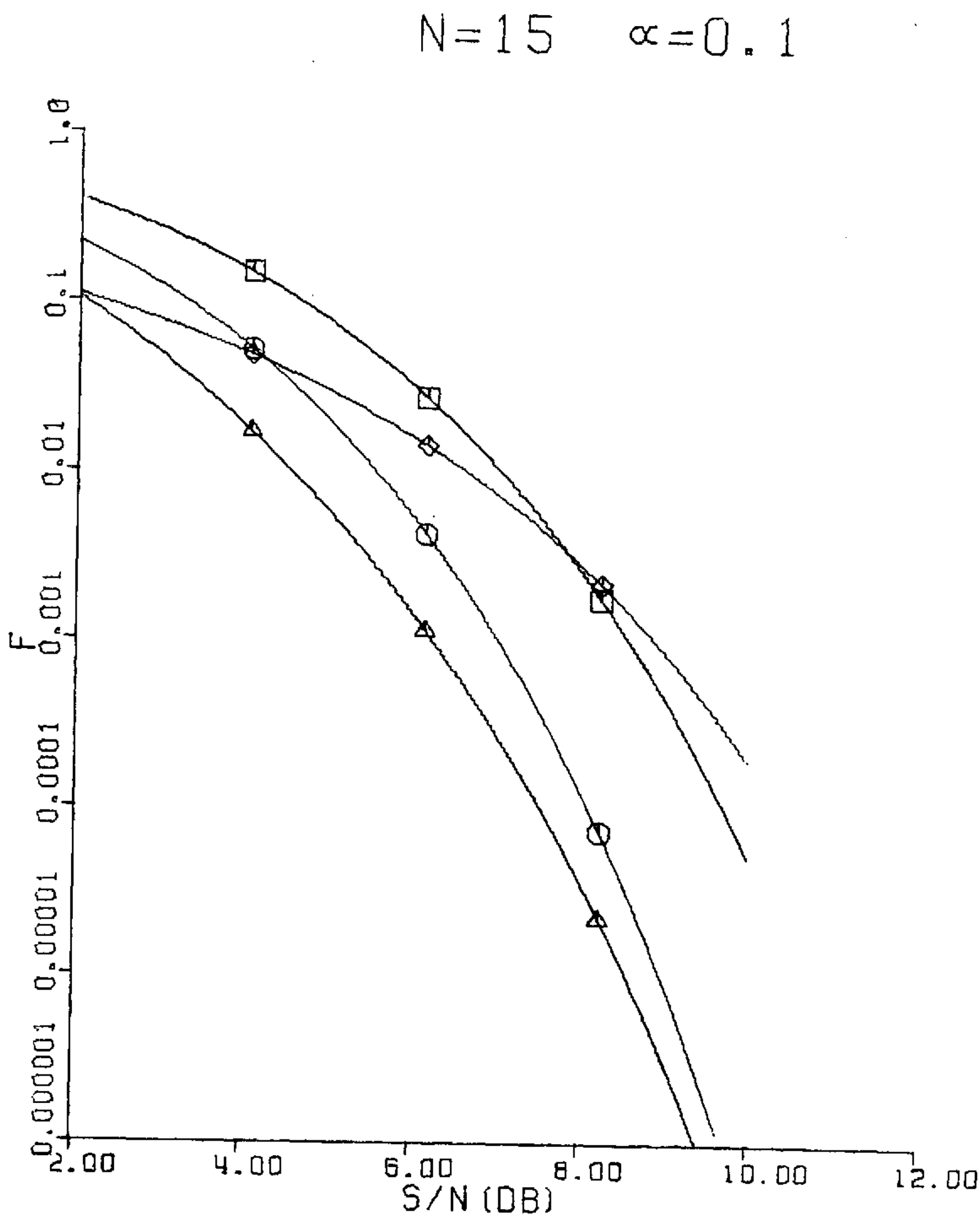

N=15 α=0.0
1.0
0.1
0.01
0.001
0.0001
0.00001
0.000001
0.0
F
2.00
4.00
6.00
8.00
10.00
12.00
S/N (DB)

FIGURE 8



N=15 α=0.1
1.0
0.1
0.01
0.001
0.0001
0.00001
0.000001
0.0000001
F
2.00
4.00
6.00
8.00
10.00
12.00
S/N (DB)

FIGURE 9

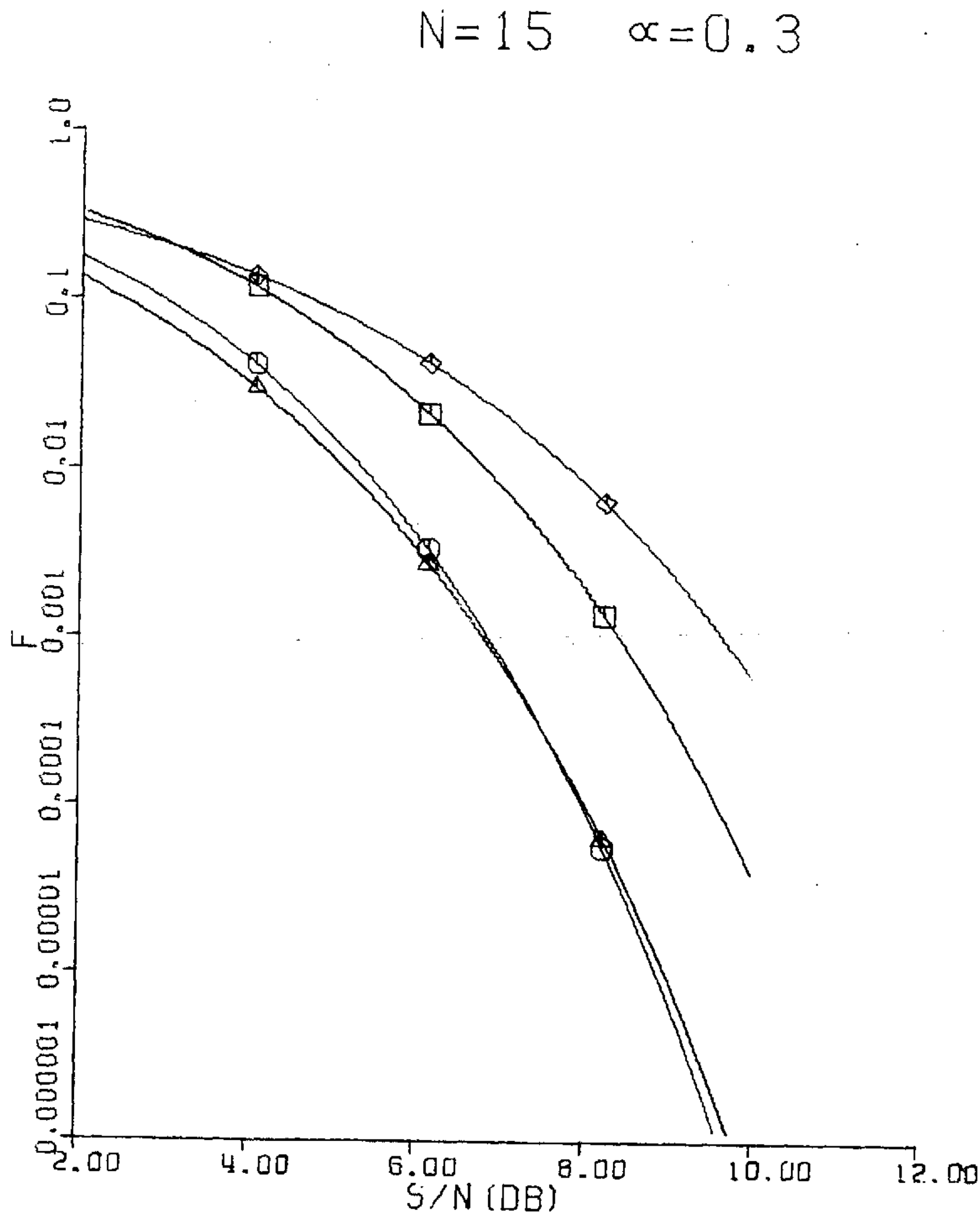

N=15 α=0.3
1.0
0.1
0.01
0.001
0.0001
0.00001
0.000001
0.0
F
2.00
4.00
6.00
8.00
10.00
12.00
S/N (DB)

FIGURE 10

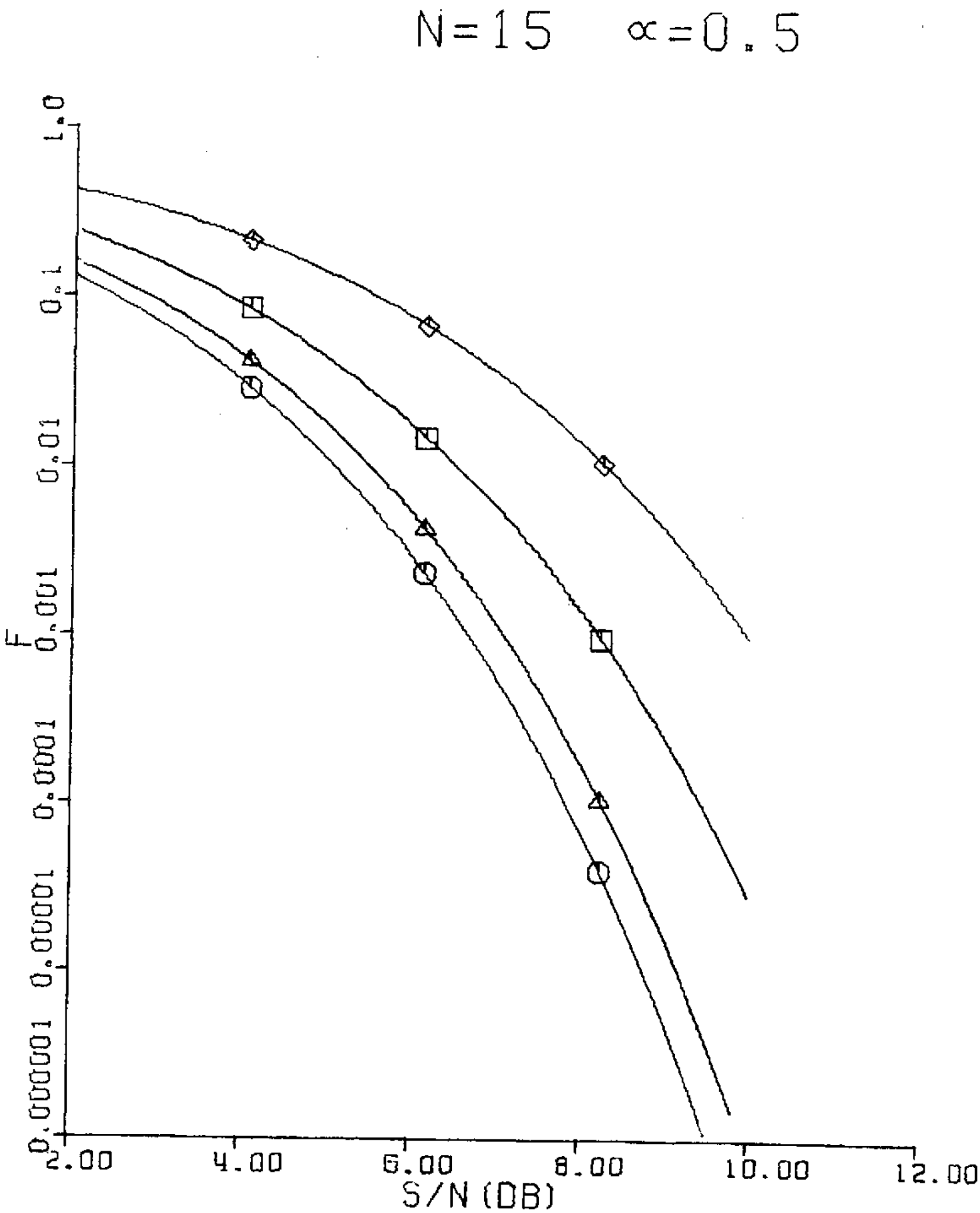

N=15 ∝=0.5
1.0
0.1
0.01
0.001
0.0001
0.00001
0.000001
0.0000001
F
2.00
4.00
6.00
8.00
10.00
12.00
S/N (DB)

FIGURE 11

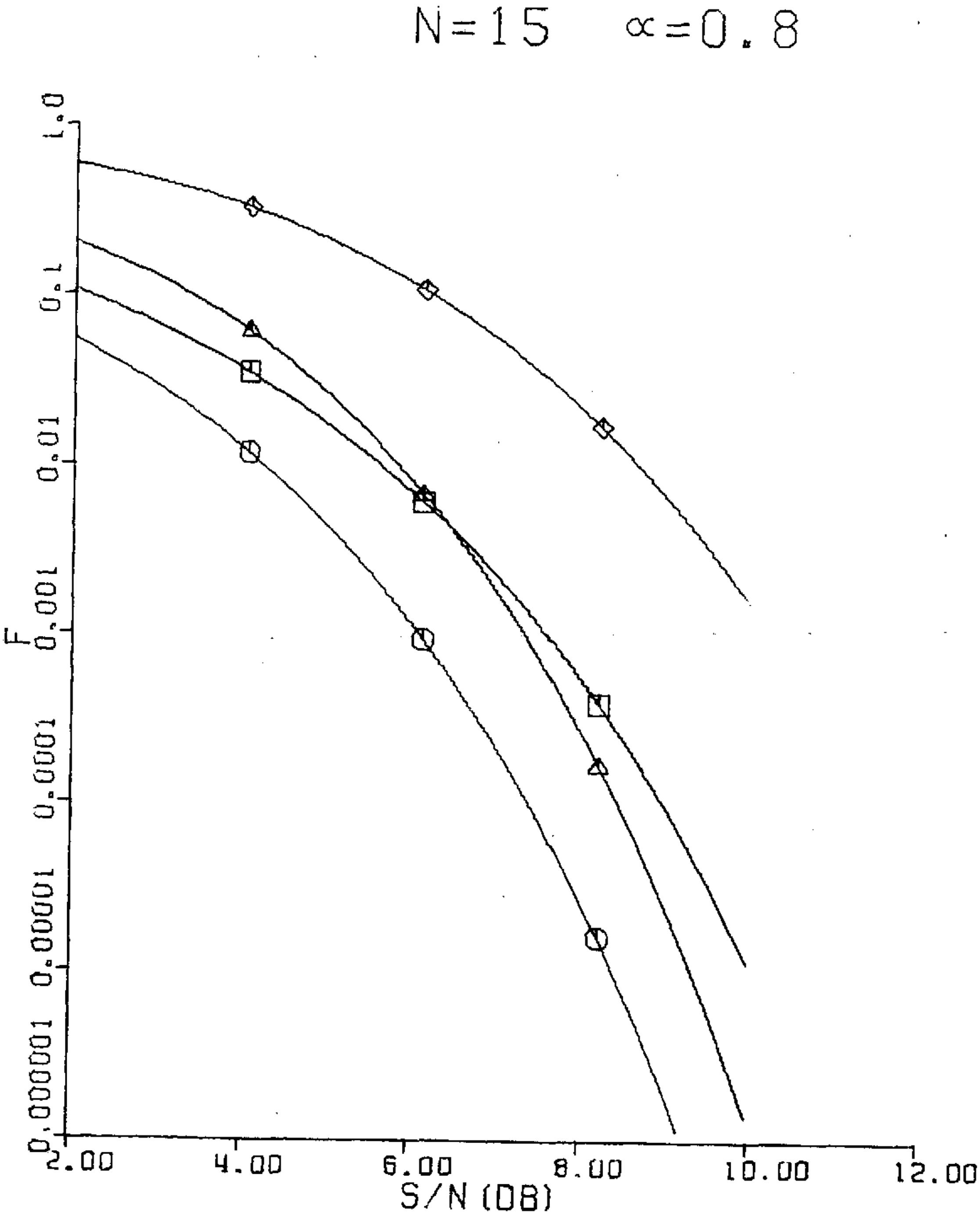

N=15 ∝=0.8
1.0
0.1
0.01
0.001
0.0001
0.00001
0.000001
0.0000001
F
2.00
4.00
6.00
8.00
10.00
12.00
S/N (DB)

FIGURE 12

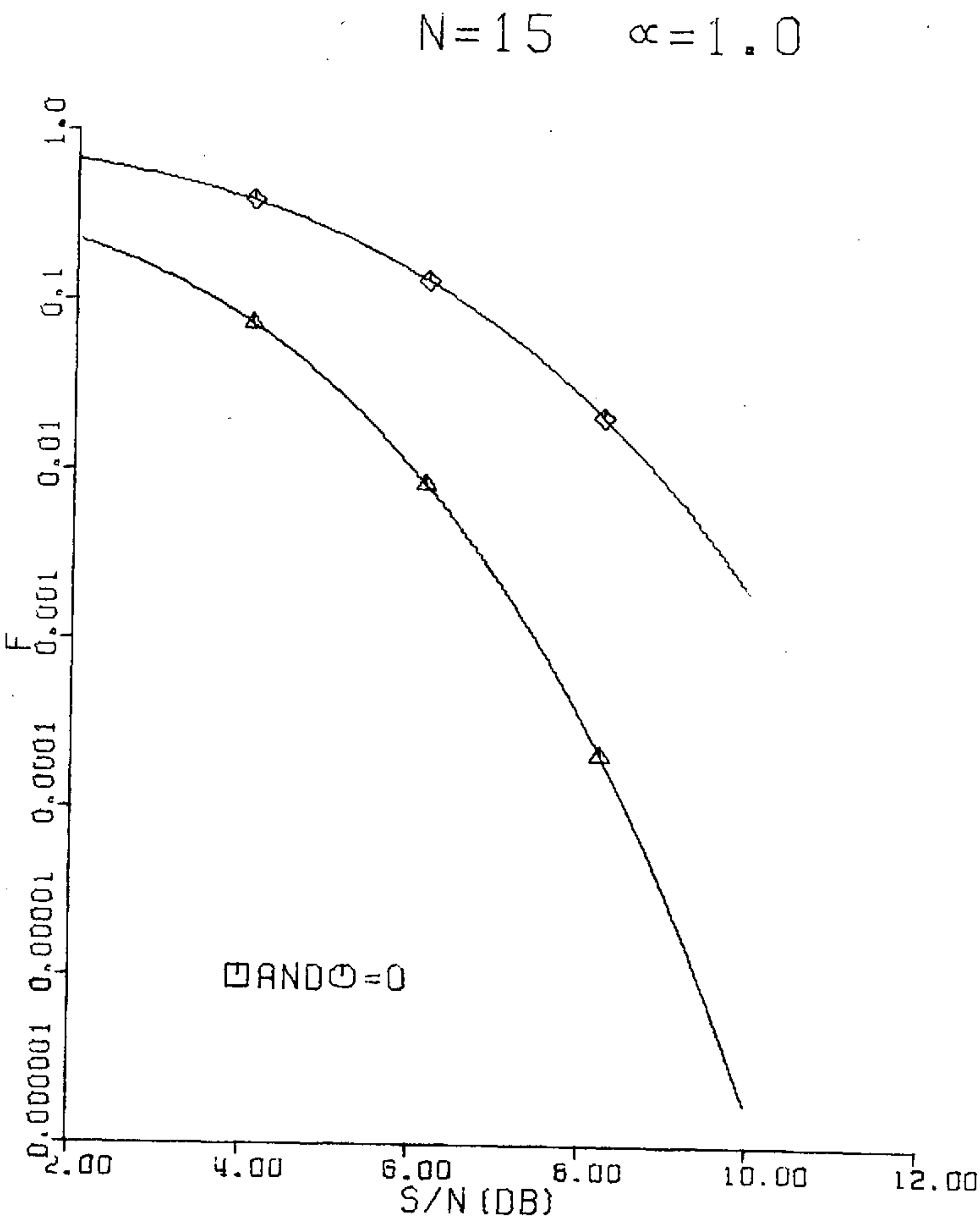

N=15 α=1.0
1.0
0.1
0.01
0.001
0.0001
0.00001
0.000001
0.0000001
F
ΘANDΦ=0
2.00
4.00
6.00
8.00
10.00
12.00
S/N (DB)

FIGURE 13

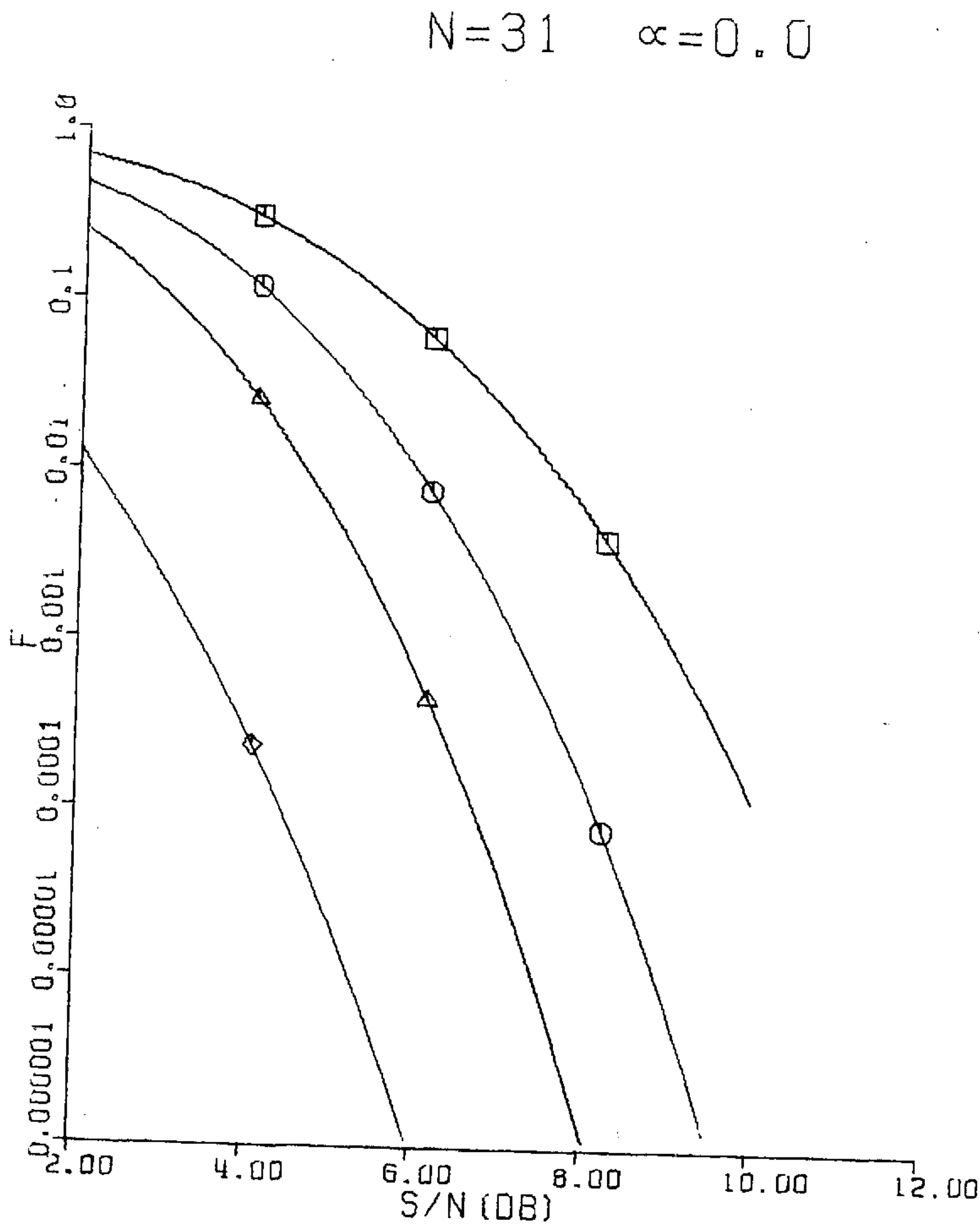
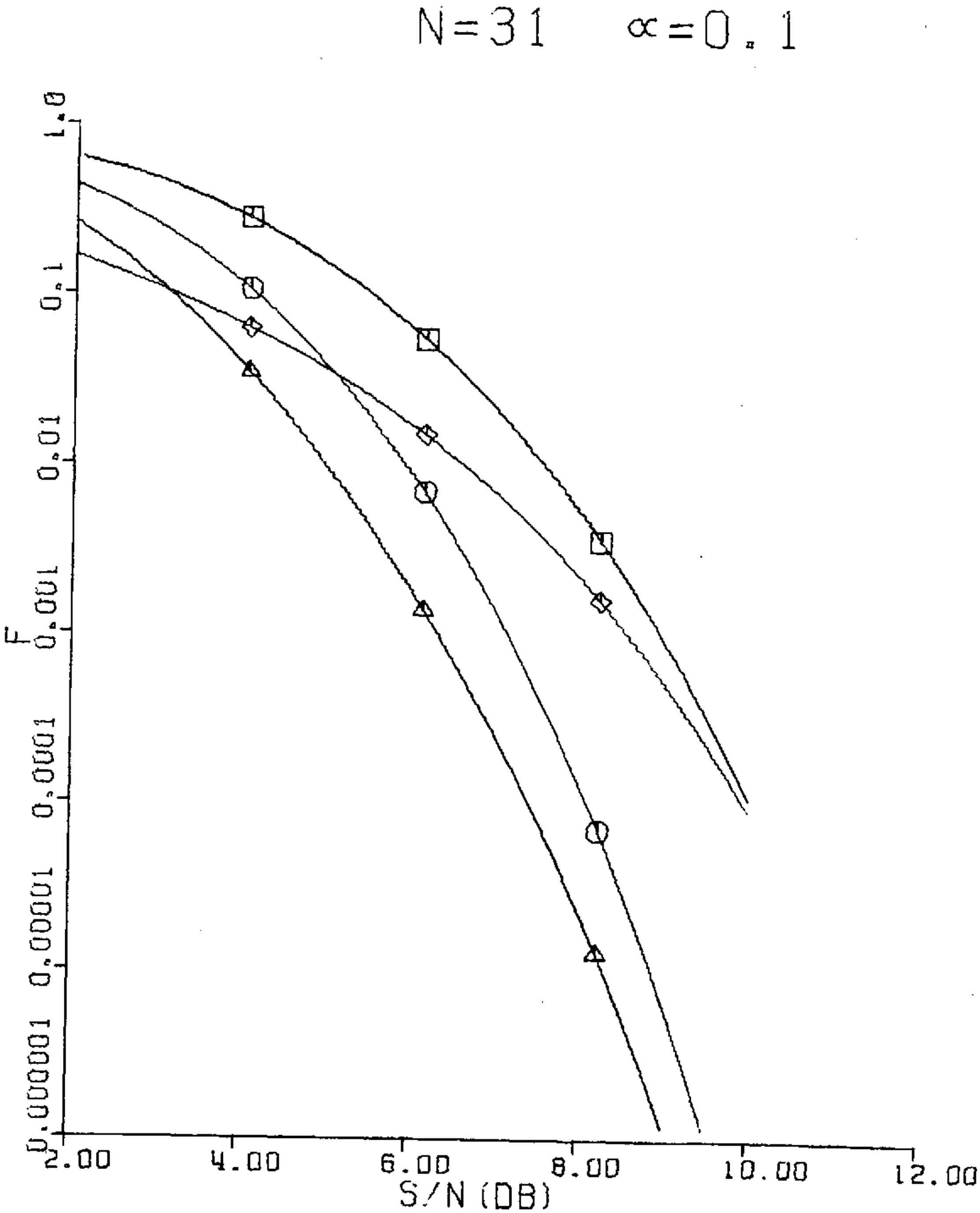

N=31 α=0.0
1.0
0.1
0.01
0.001
0.0001
0.00001
0.000001
0.0000001
F
2.00
4.00
6.00
8.00
10.00
12.00
S/N (DB)

FIGURE 14



N=31 α=0.1
1.0
0.1
0.01
0.001
0.0001
0.00001
0.000001
0.0000001
F
2.00
4.00
6.00
8.00
10.00
12.00
S/N (DB)

FIGURE 15

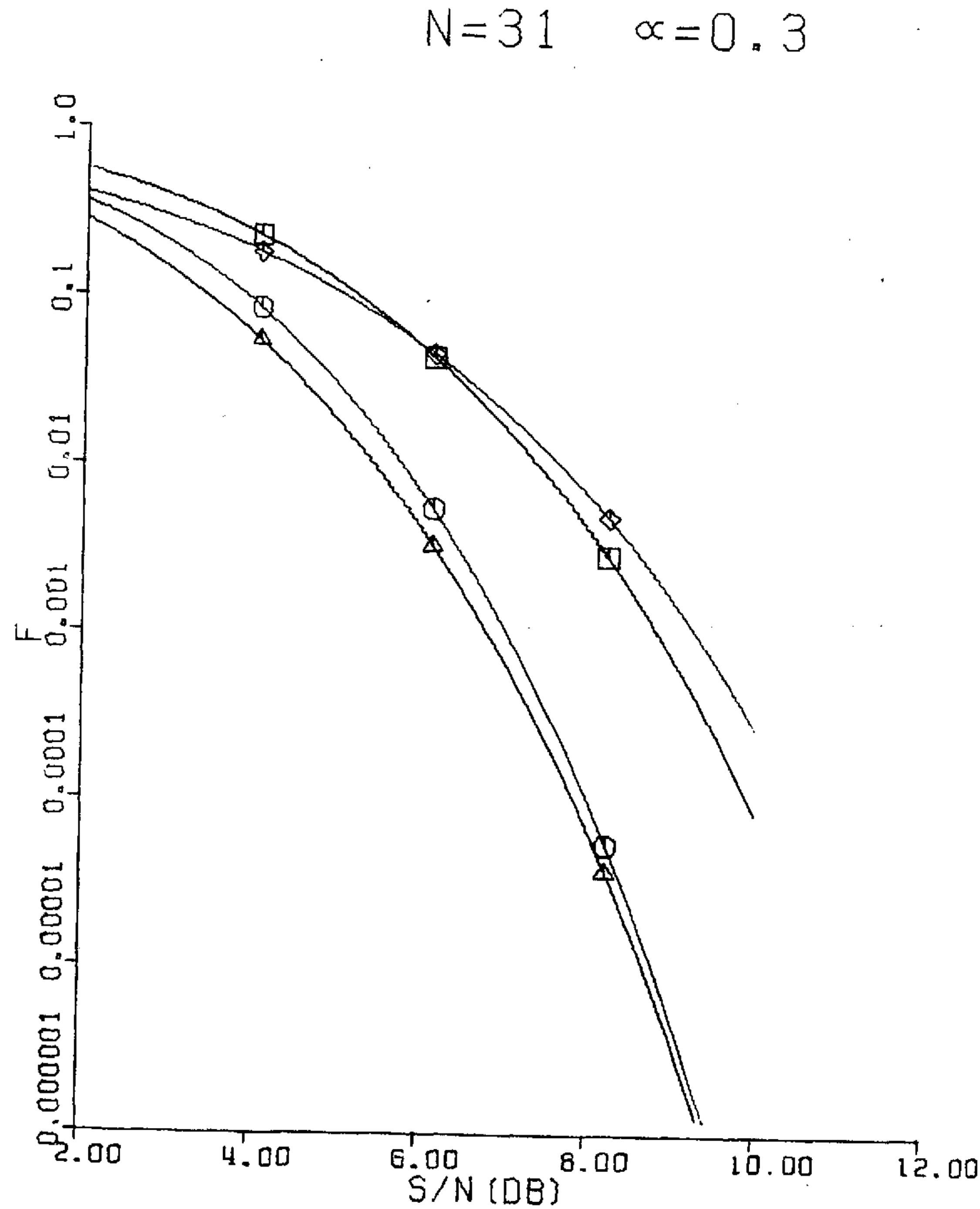

N=31 ∝=0.3
1.0
0.1
0.01
0.001
0.0001
0.00001
0.000001
0.0000001
F
2.00
4.00
6.00
8.00
10.00
12.00
S/N (DB)

FIGURE 16

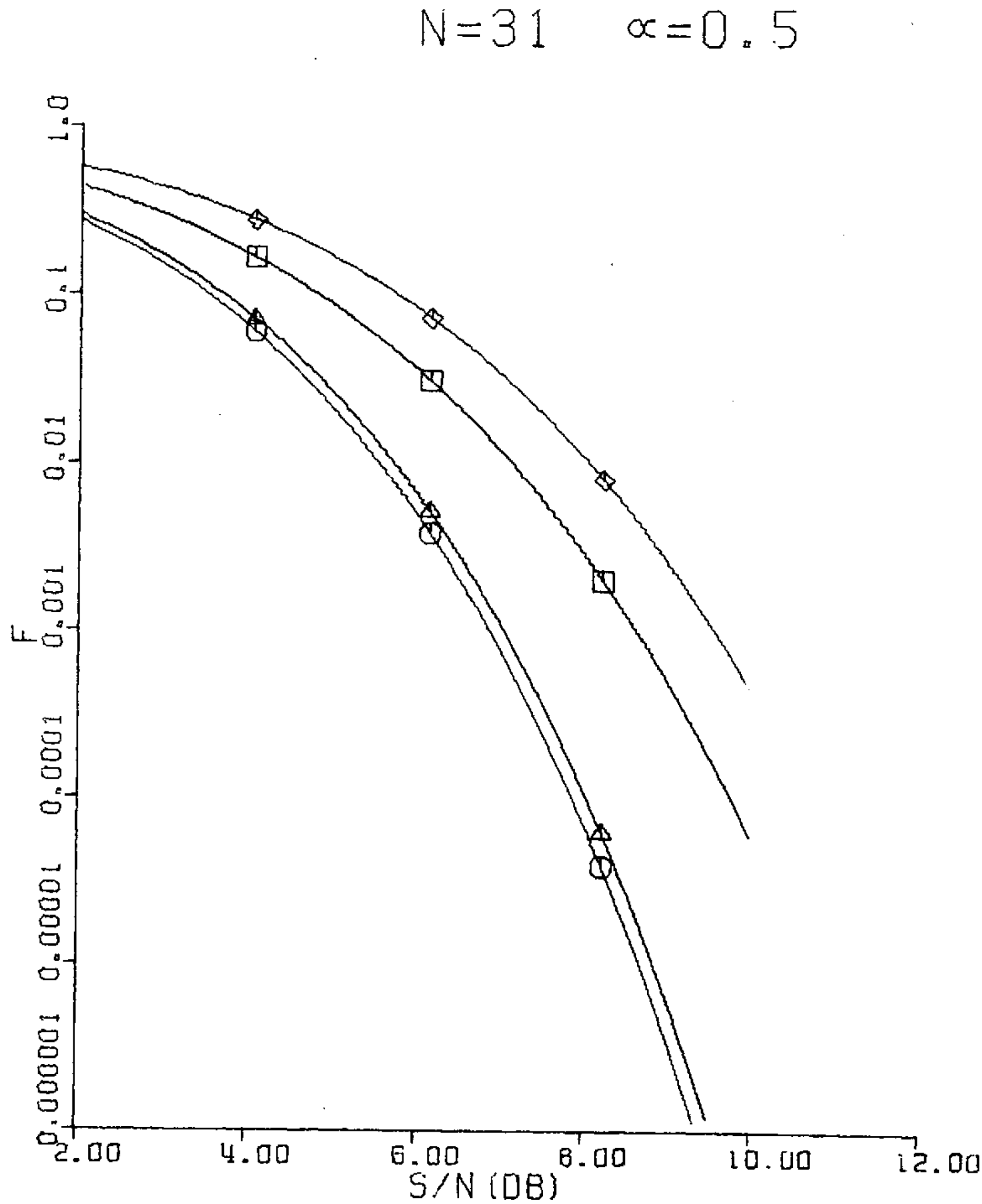

N=31 α=0.5
1.0
0.1
0.01
0.001
0.0001
0.00001
0.000001
0.0
F
2.00
4.00
6.00
8.00
10.00
12.00
S/N (DB)

FIGURE 17

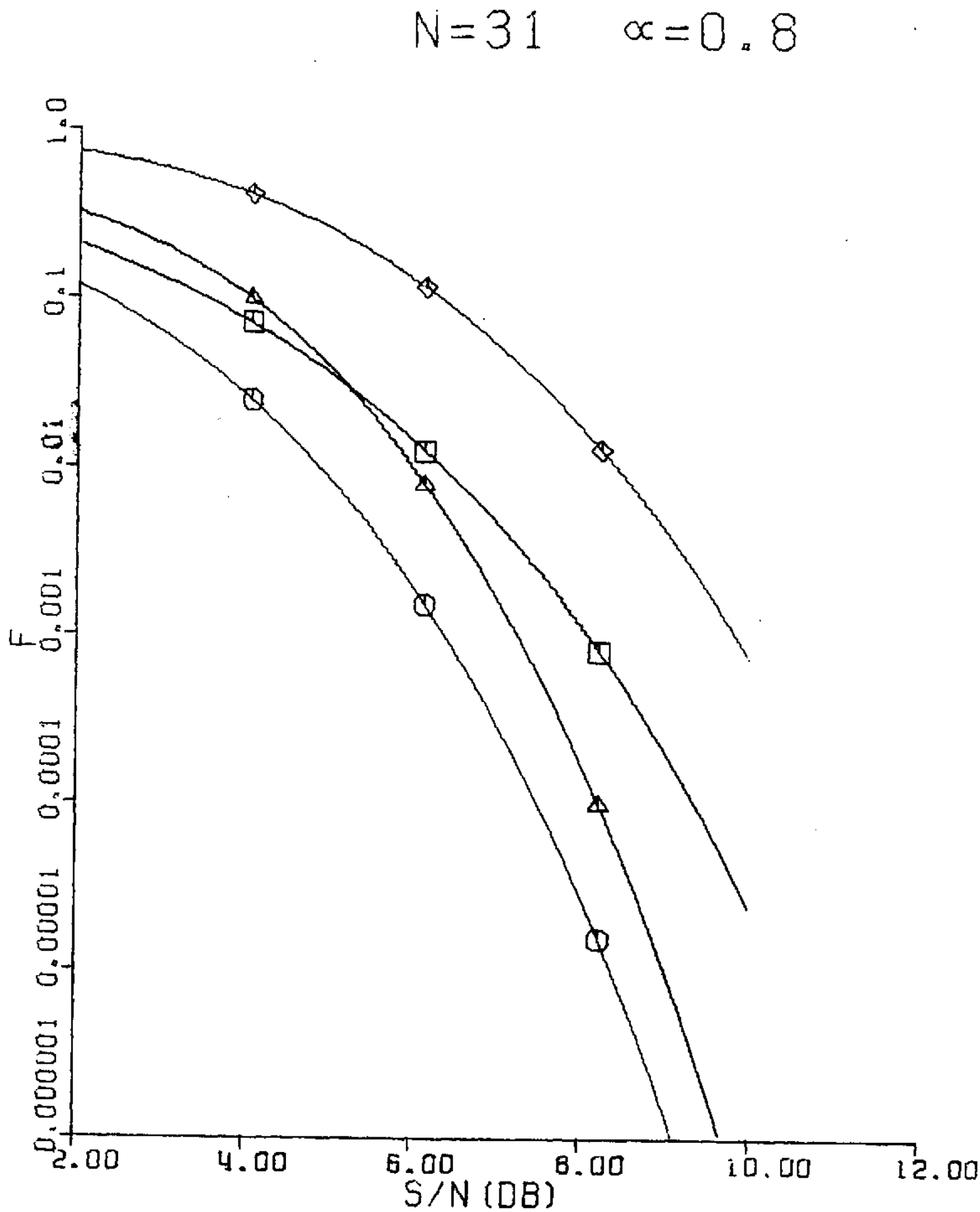

N=31 α=0.8
1.0
0.1
0.01
0.001
0.0001
0.00001
0.000001
0.0000001
F
2.00
4.00
6.00
8.00
10.00
12.00
S/N (DB)

FIGURE 18

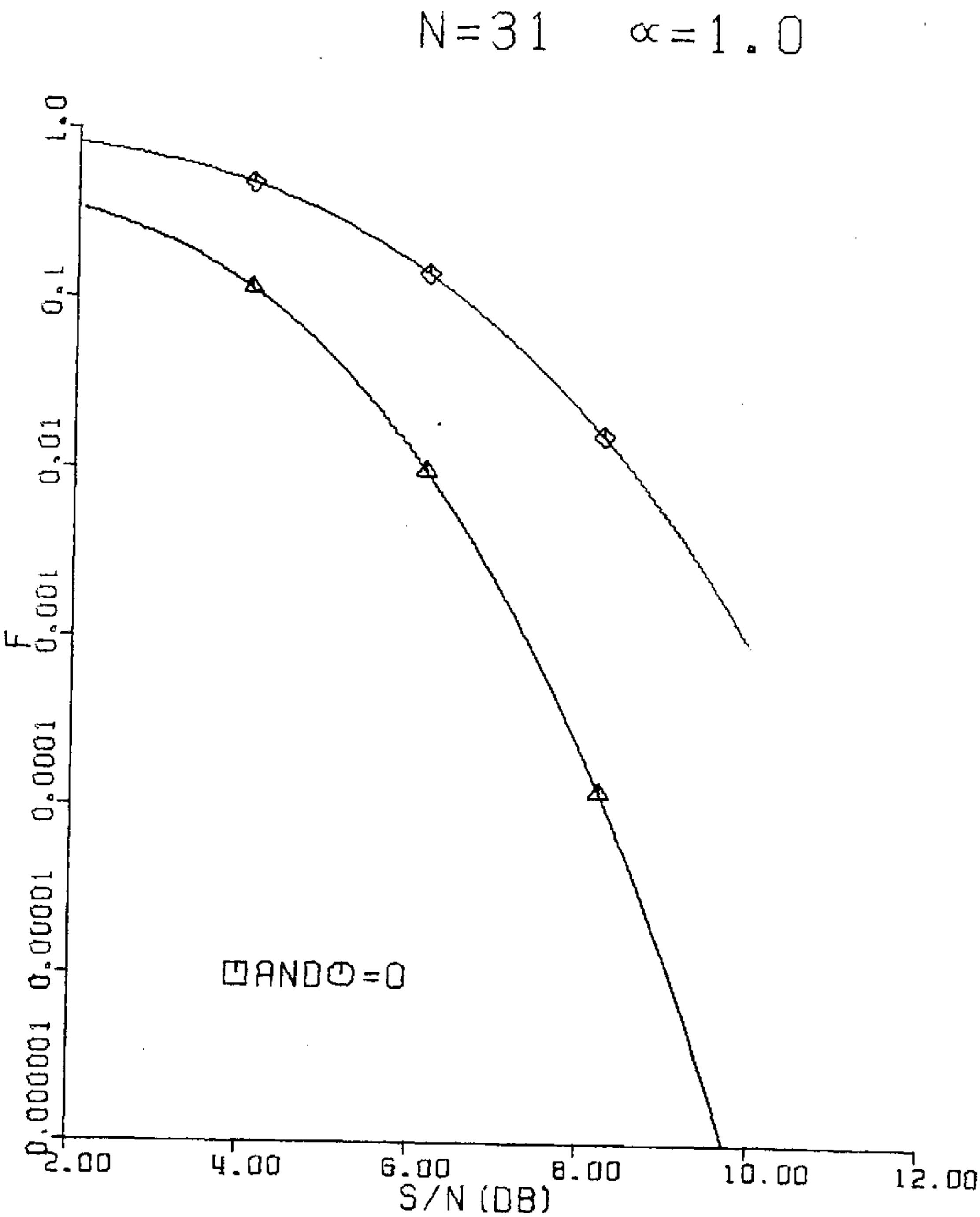
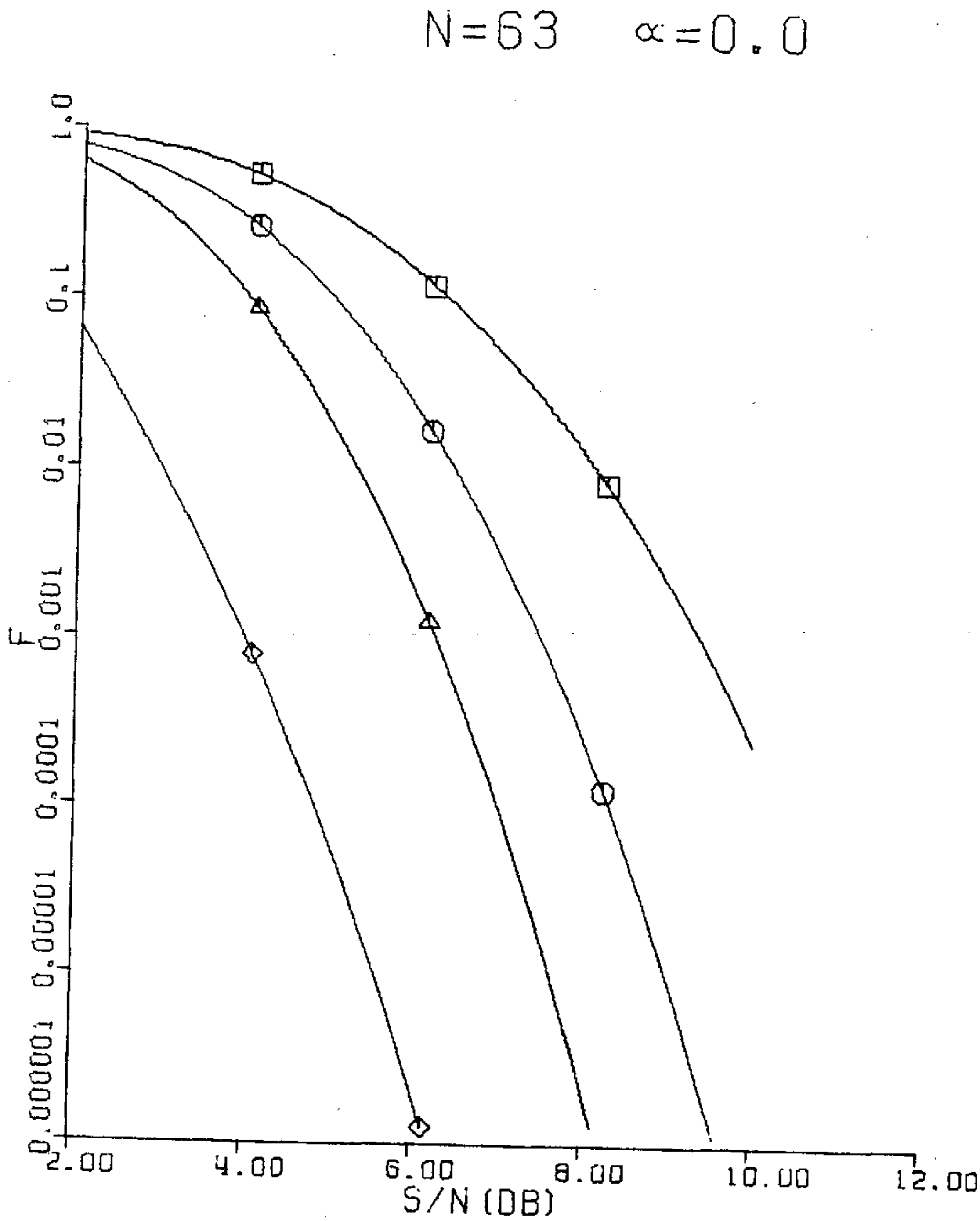

N=31 α=1.0
F
1.0
0.1
0.01
0.001
0.0001
0.00001
0.000001
2.00
4.00
6.00
8.00
10.00
12.00
S/N (DB)
□ AND ⊕ = 0

FIGURE 19



N=63 α=0.0
1.0
0.1
0.01
0.001
0.0001
0.00001
0.000001
F
2.00
4.00
6.00
8.00
10.00
12.00
S/N (DB)

FIGURE 20

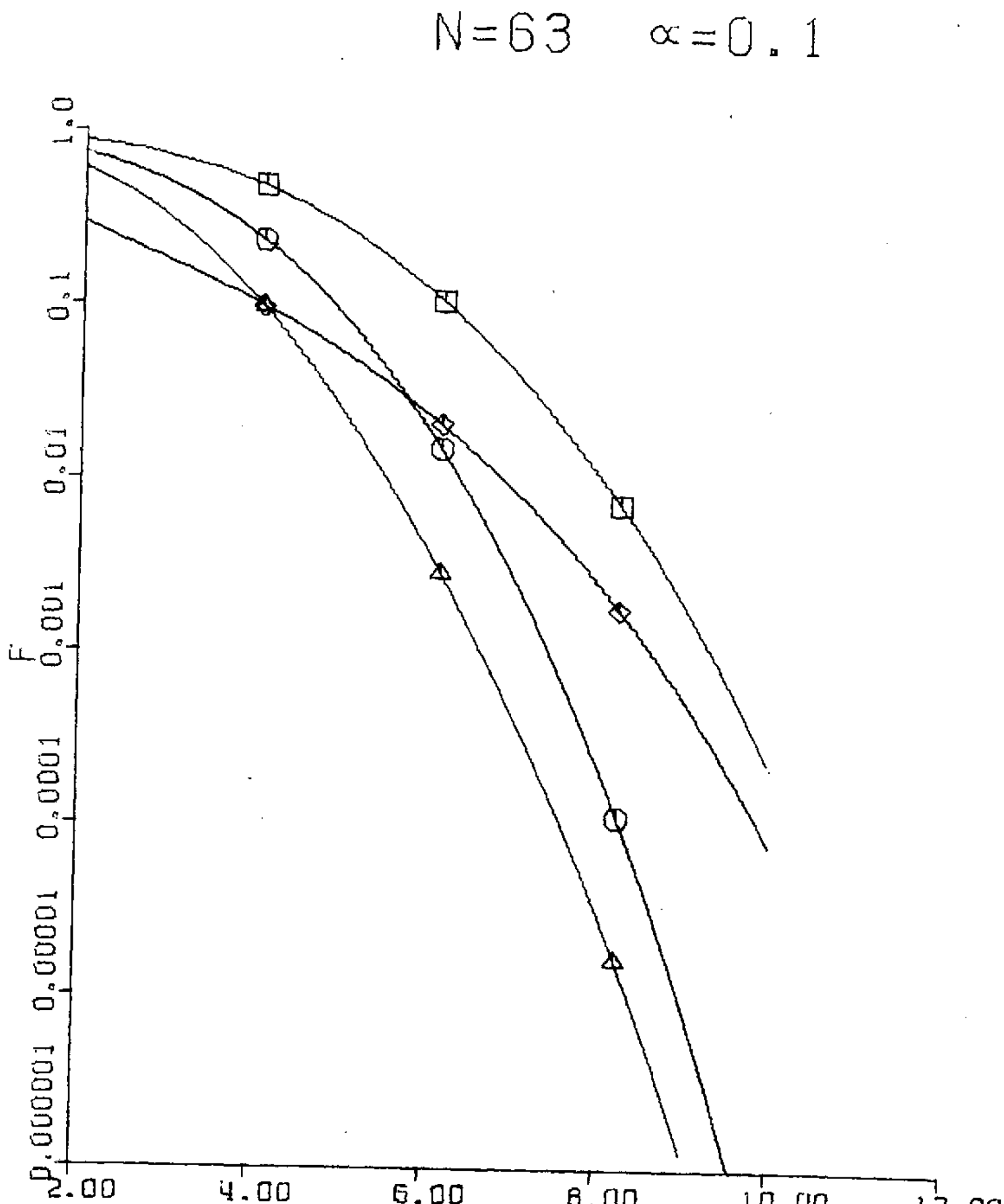
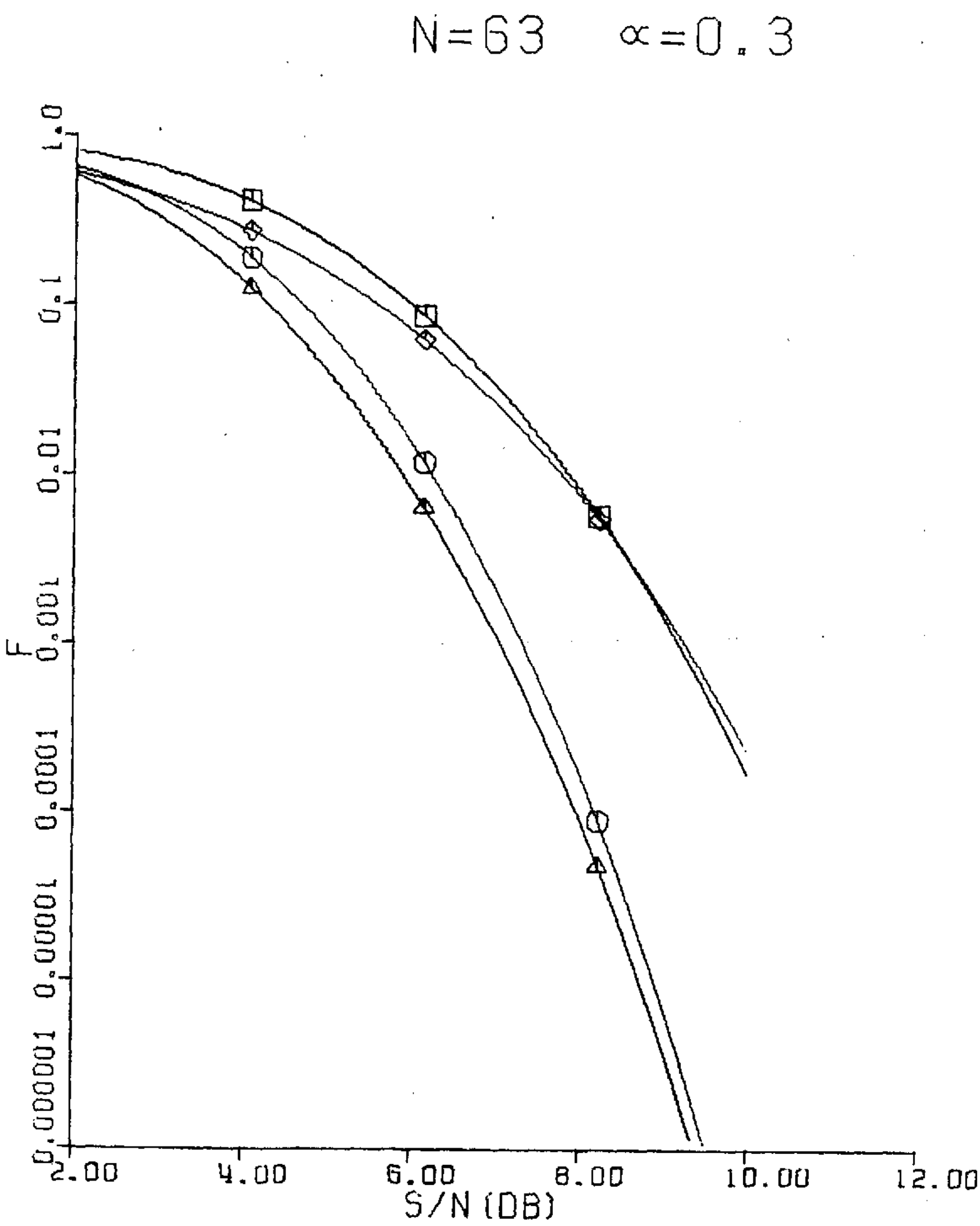

N=63 α=0.1
1.0
0.1
0.01
0.001
0.0001
0.00001
0.000001
0.0000001
F
2.00
4.00
6.00
8.00
10.00
12.00
S/N (DB)

FIGURE 21



N=63 α=0.3
1.0
0.1
0.01
0.001
0.0001
0.00001
0.000001
0.0000001
F
2.00
4.00
6.00
8.00
10.00
12.00
S/N (DB)

FIGURE 22

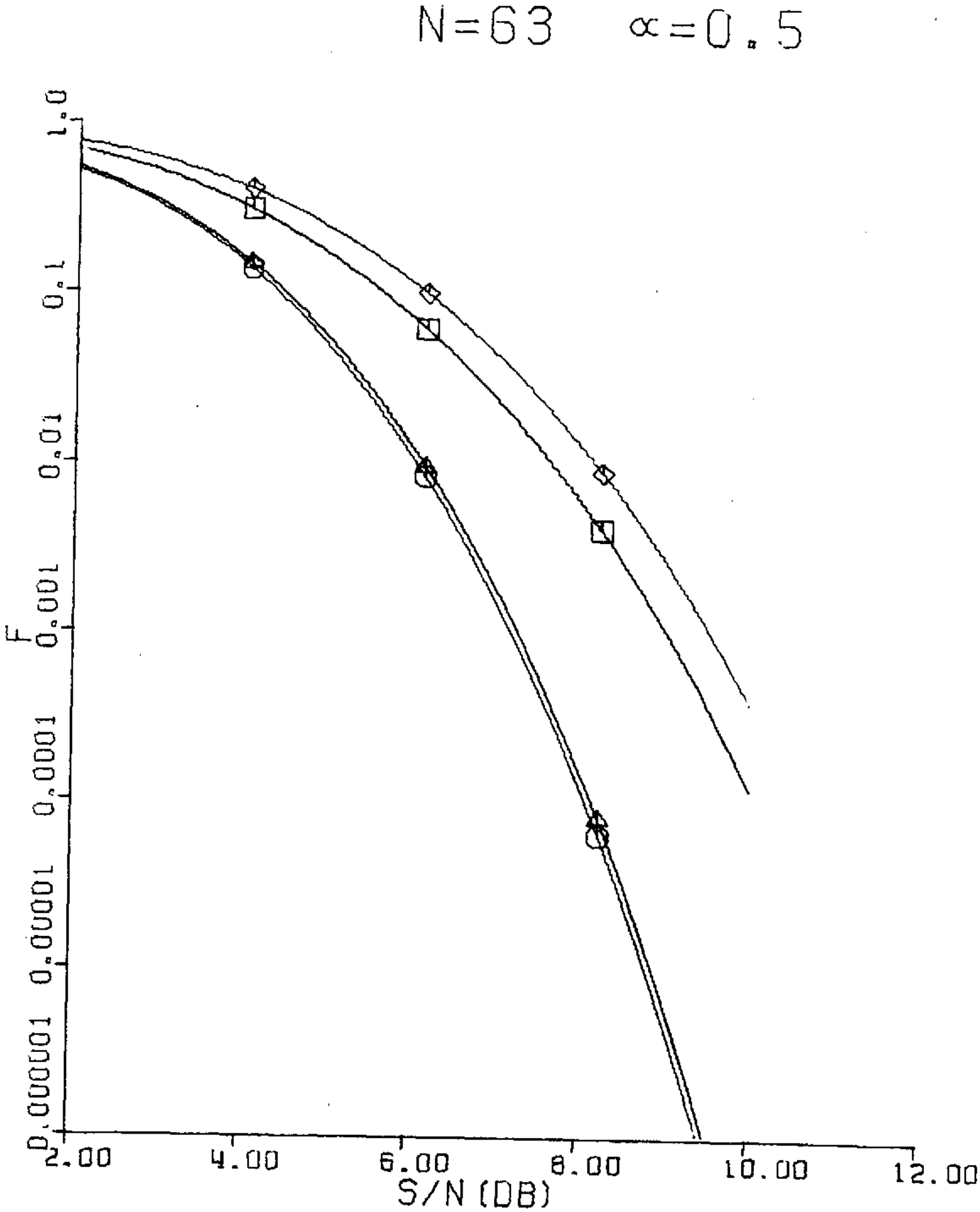

N=63 α=0.5
1.0
0.1
0.01
0.001
0.0001
0.00001
0.000001
0.0000001
F
2.00
4.00
6.00
8.00
10.00
12.00
S/N (DB)

FIGURE 23

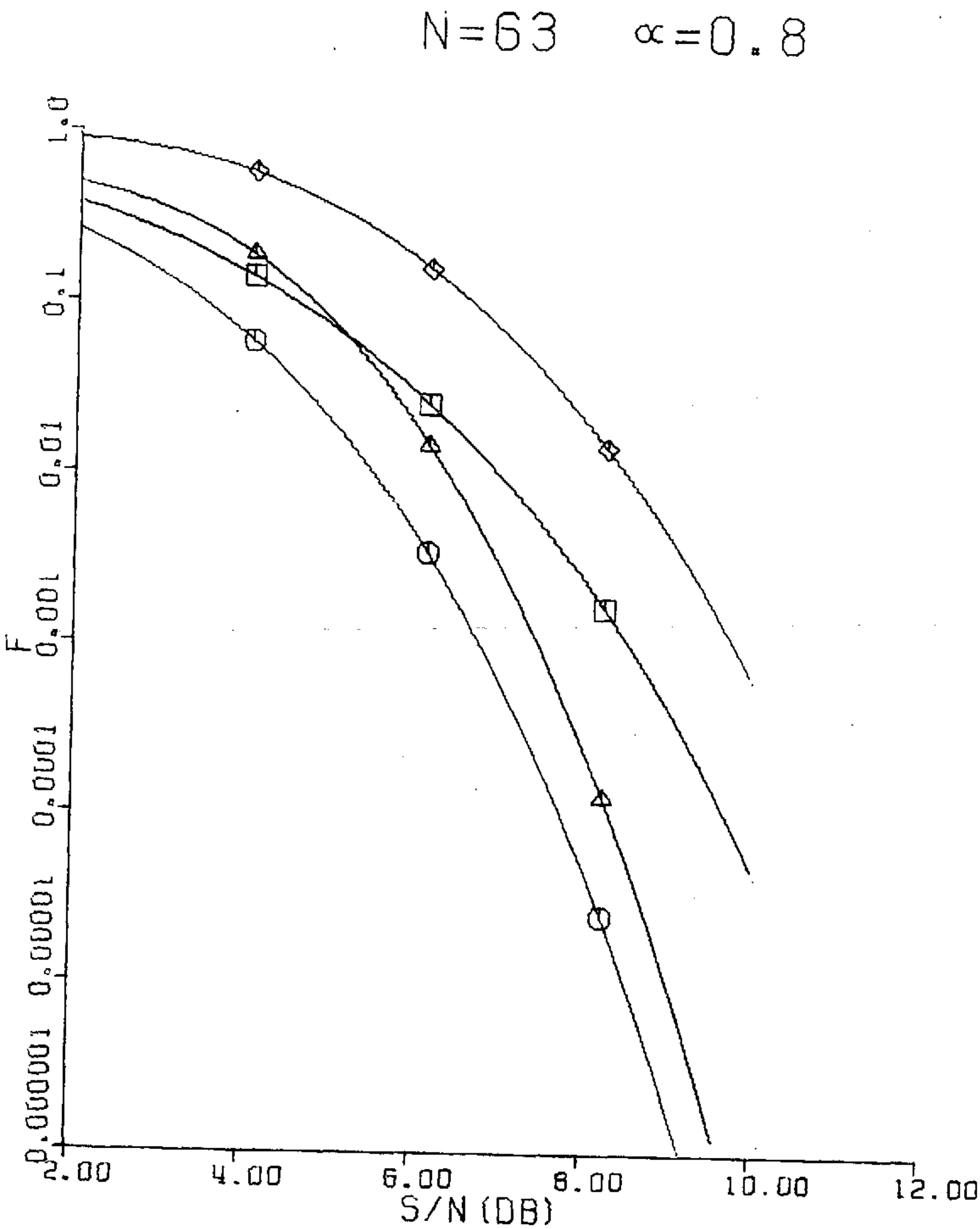
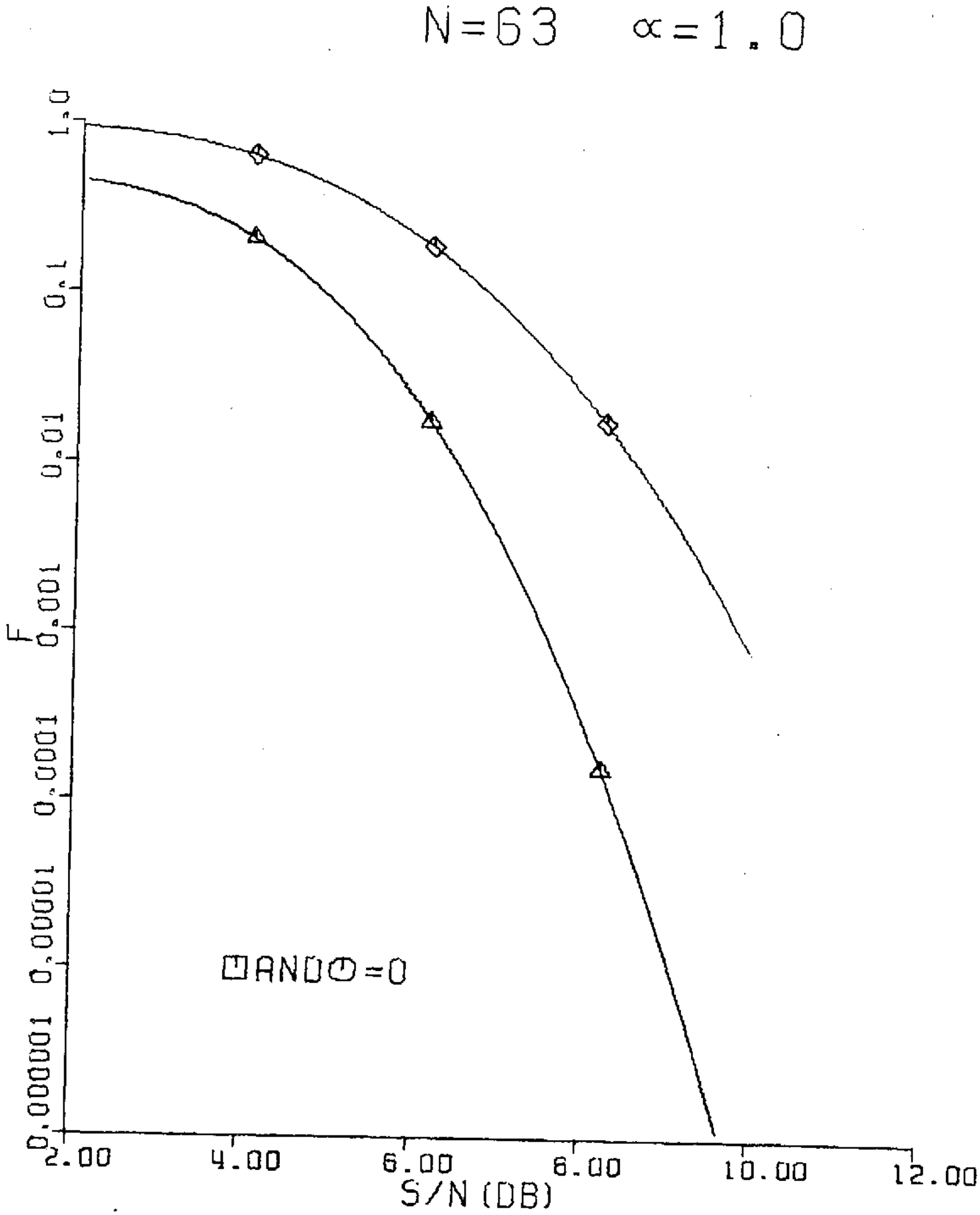
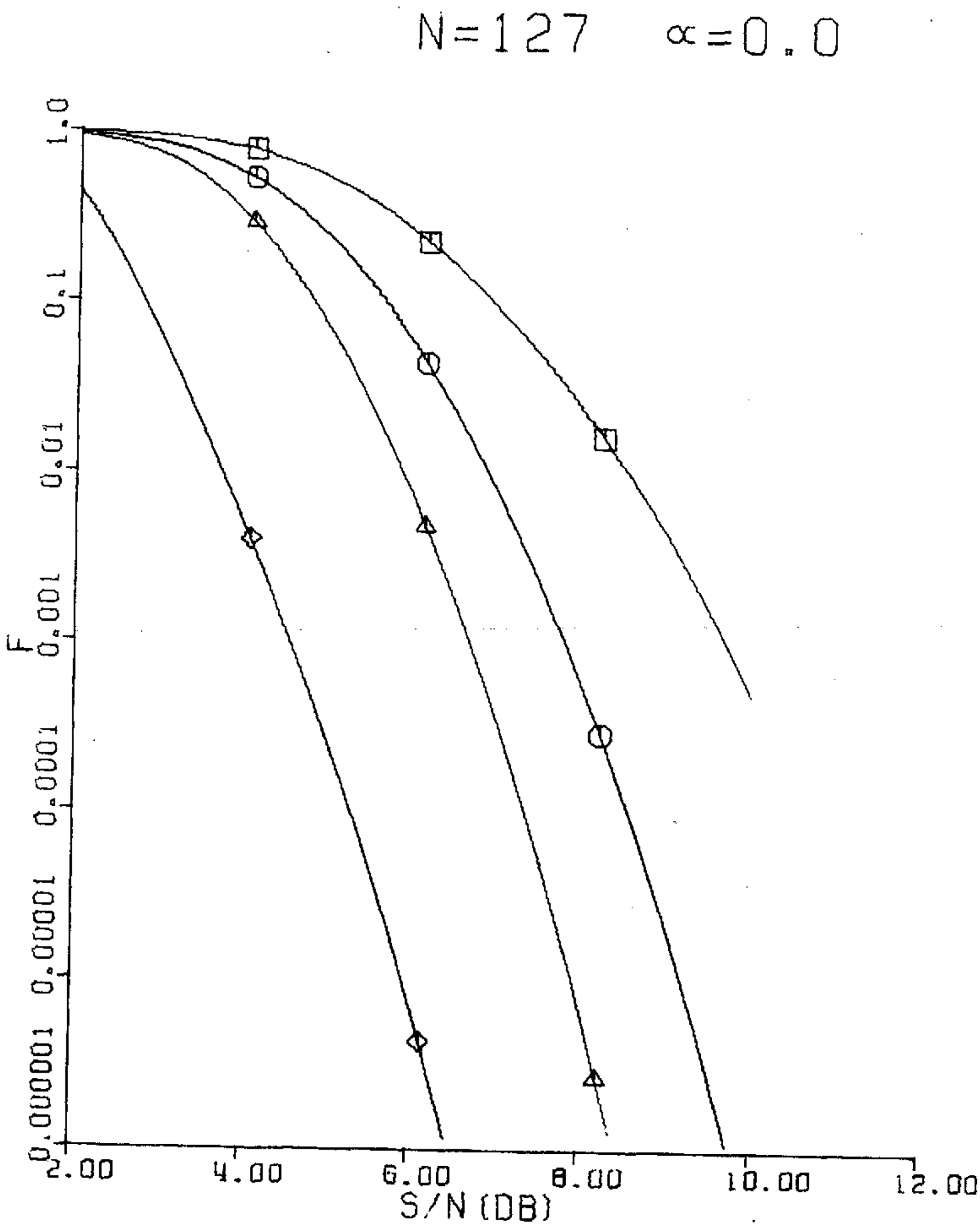

N=63 α=0.8
1.0
0.1
0.01
0.001
0.0001
0.00001
0.000001
0.0000001
F
2.00
4.00
6.00
8.00
10.00
12.00
S/N (DB)

FIGURE 24



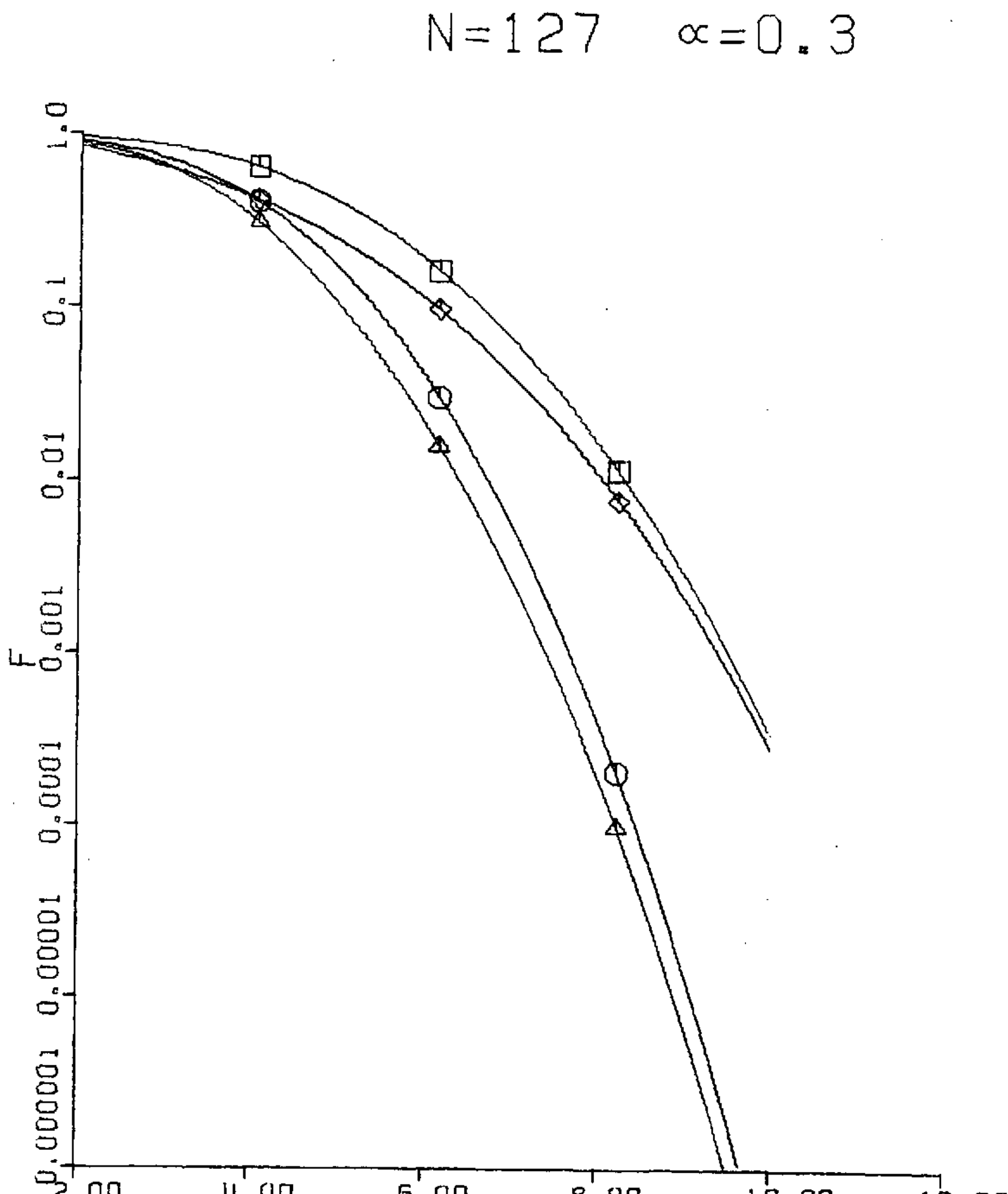
N=63 α=1.0
1.0
0.1
0.01
0.001
0.0001
0.00001
0.000001
0.0
F
□AND⊘=0
2.00
4.00
6.00
8.00
10.00
12.00
S/N (DB)

FIGURE 25



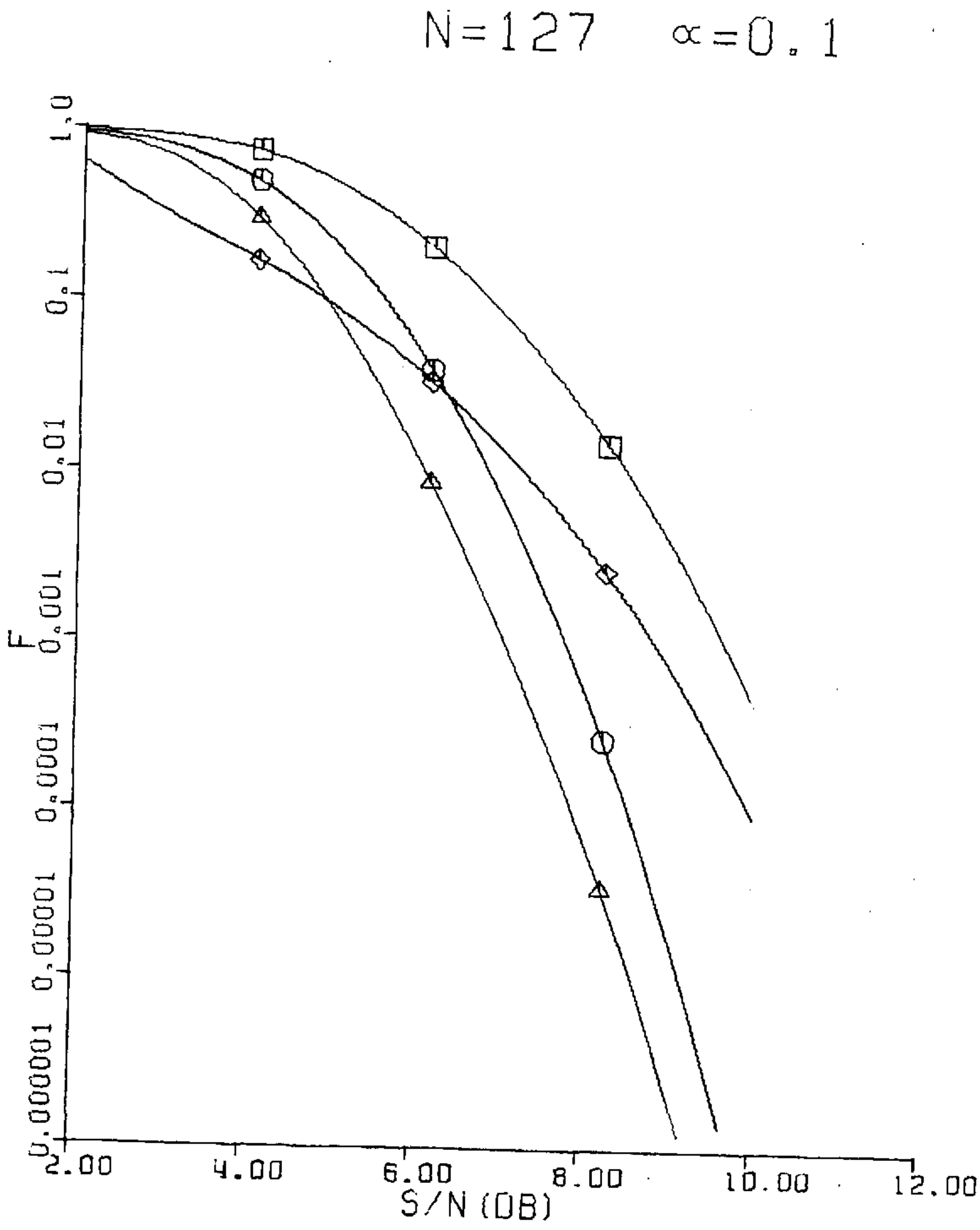
N=127 α=0.0
1.0
0.1
0.01
0.001
0.0001
0.00001
0.000001
0.0000001
F
2.00
4.00
6.00
8.00
10.00
12.00
S/N (DB)

FIGURE 26



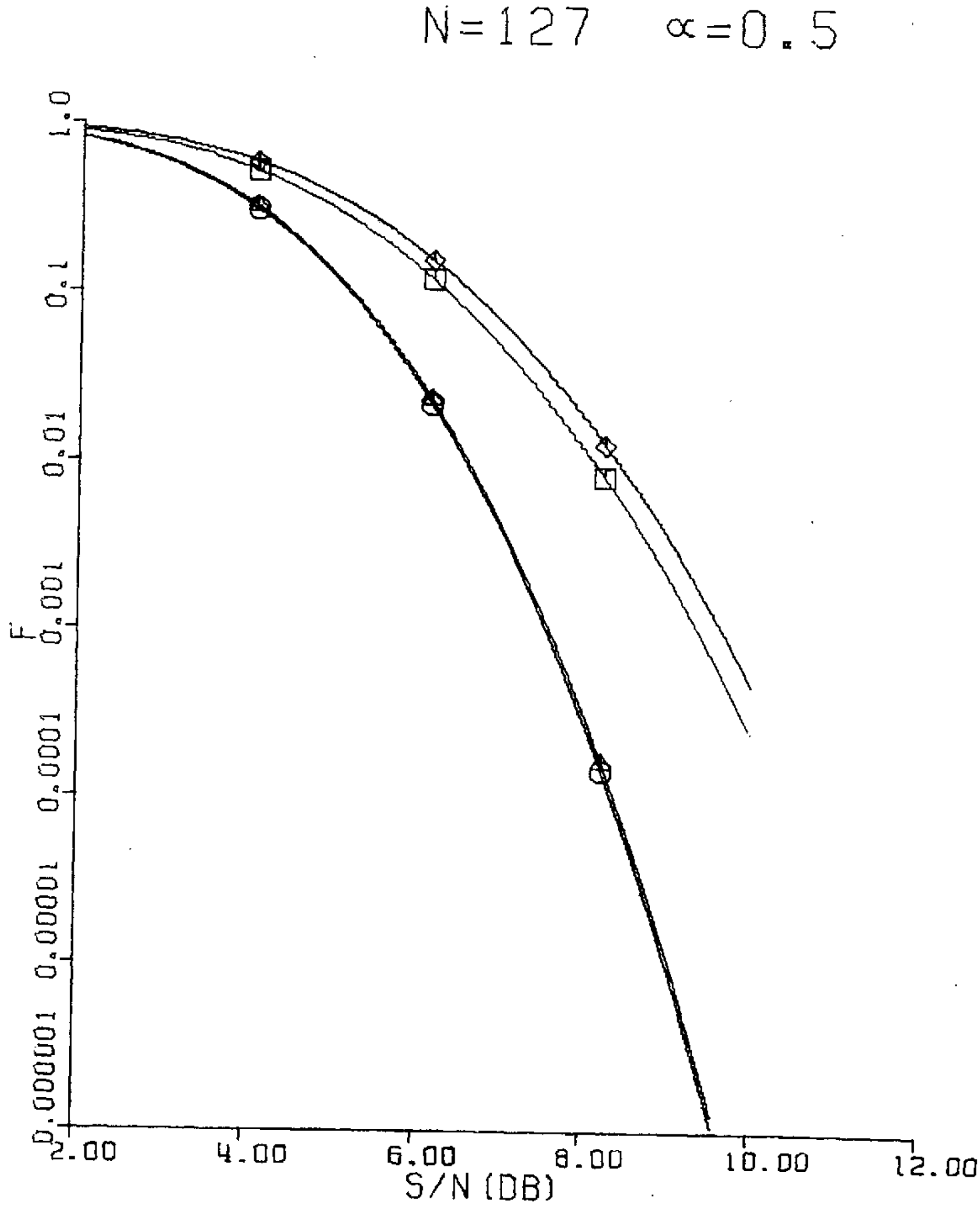
N=127 ∝=0.3
1.0
0.1
0.01
0.001
0.0001
0.00001
0.000001
0.0000001
F
2.00
4.00
6.00
8.00
10.00
12.00
S/N (DB)

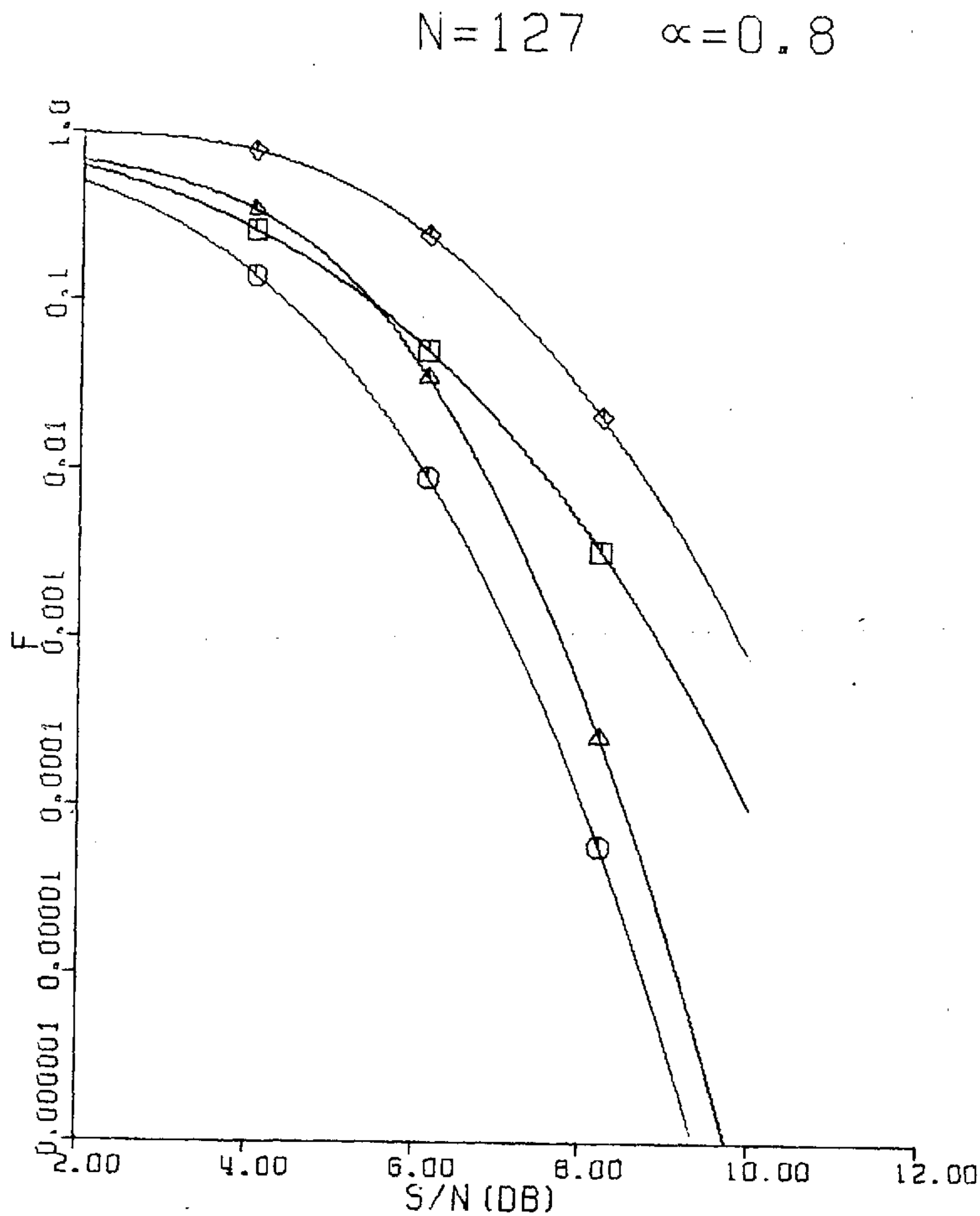
FIGURE 28



N=127 α=0.1
1.0
0.1
0.01
0.001
0.0001
0.00001
0.000001
0.0000001
F
2.00
4.00
6.00
8.00
10.00
12.00
S/N (DB)

FIGURE 27


N=127 α=0.5
1.0
0.1
0.01
0.001
0.0001
0.00001
0.000001
0.0000001
F
2.00
4.00
6.00
8.00
10.00
12.00
S/N (DB)
FIGURE 29



N=127 α=0.8
1.0
0.1
0.01
0.001
0.0001
0.00001
0.000001
F
2.00
4.00
6.00
8.00
10.00
12.00
S/N (DB)

FIGURE 30

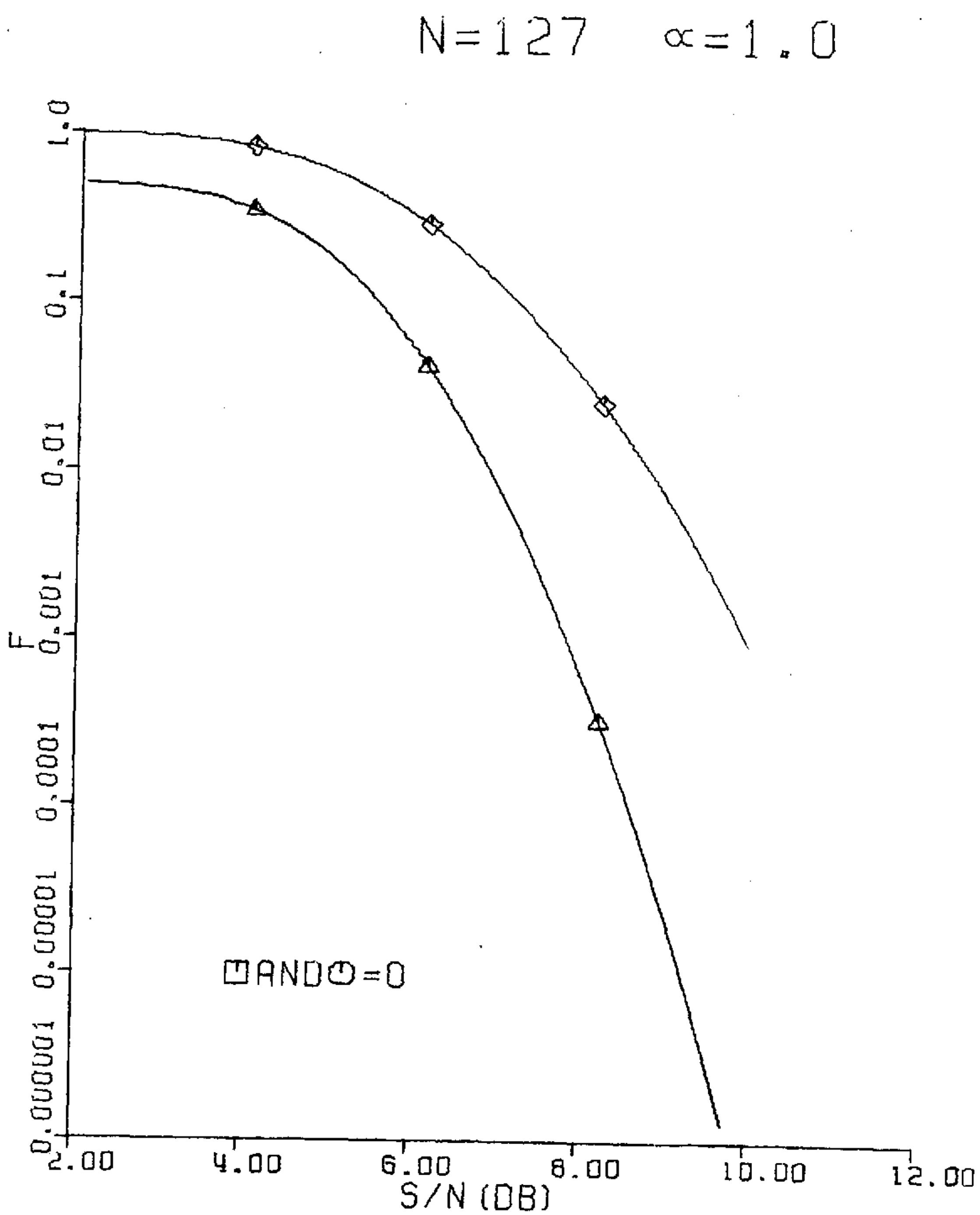

N=127 α=1.0
F
1.0
0.1
0.01
0.001
0.0001
0.00001
0.000001
0.0000001
□AND○=0
2.00
4.00
6.00
8.00
10.00
12.00
S/N (DB)

FIGURE 31

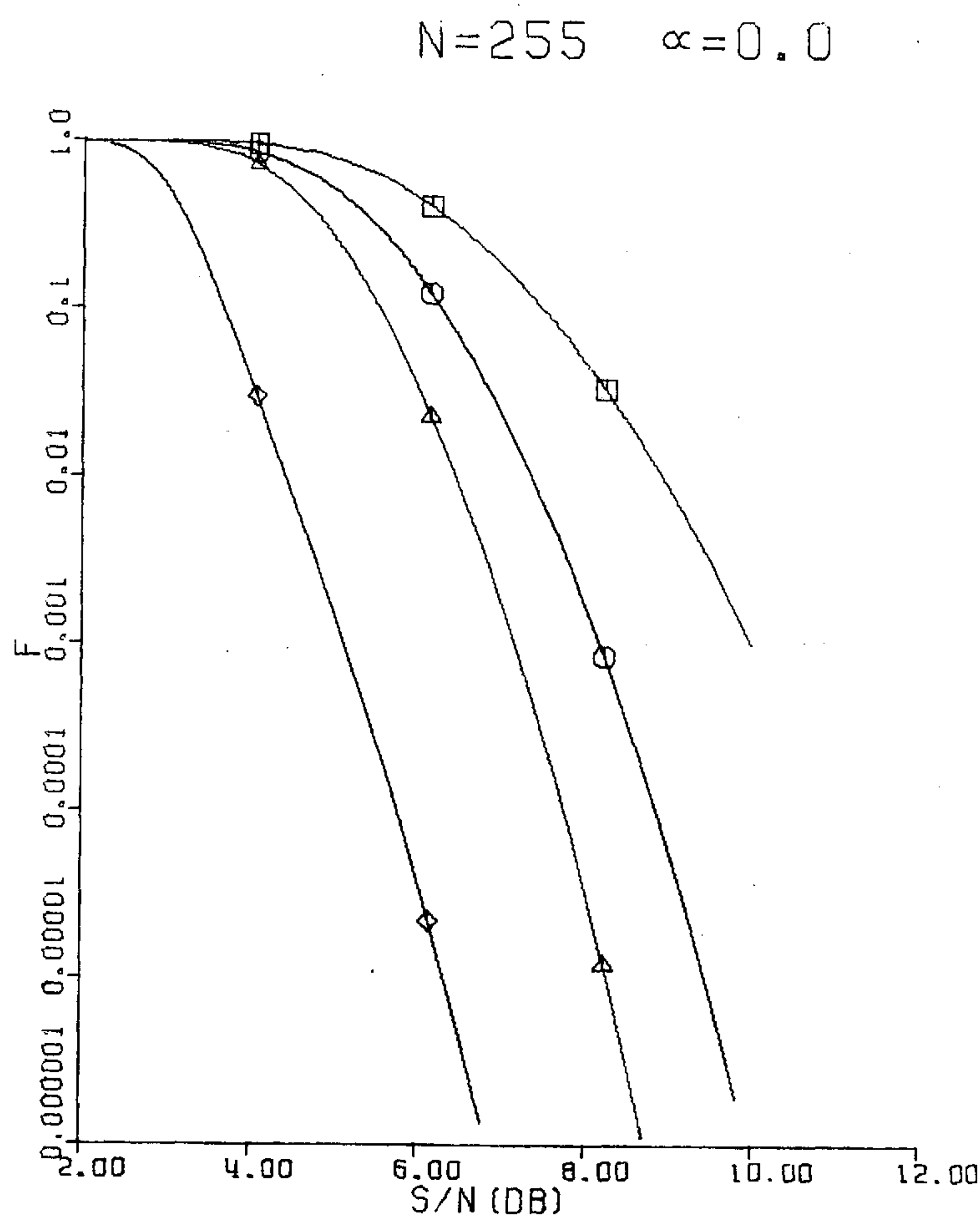

N=255 α=0.0
1.0
0.1
0.01
0.001
0.0001
0.00001
0.000001
F
2.00
4.00
6.00
8.00
10.00
12.00
S/N (DB)

FIGURE 32

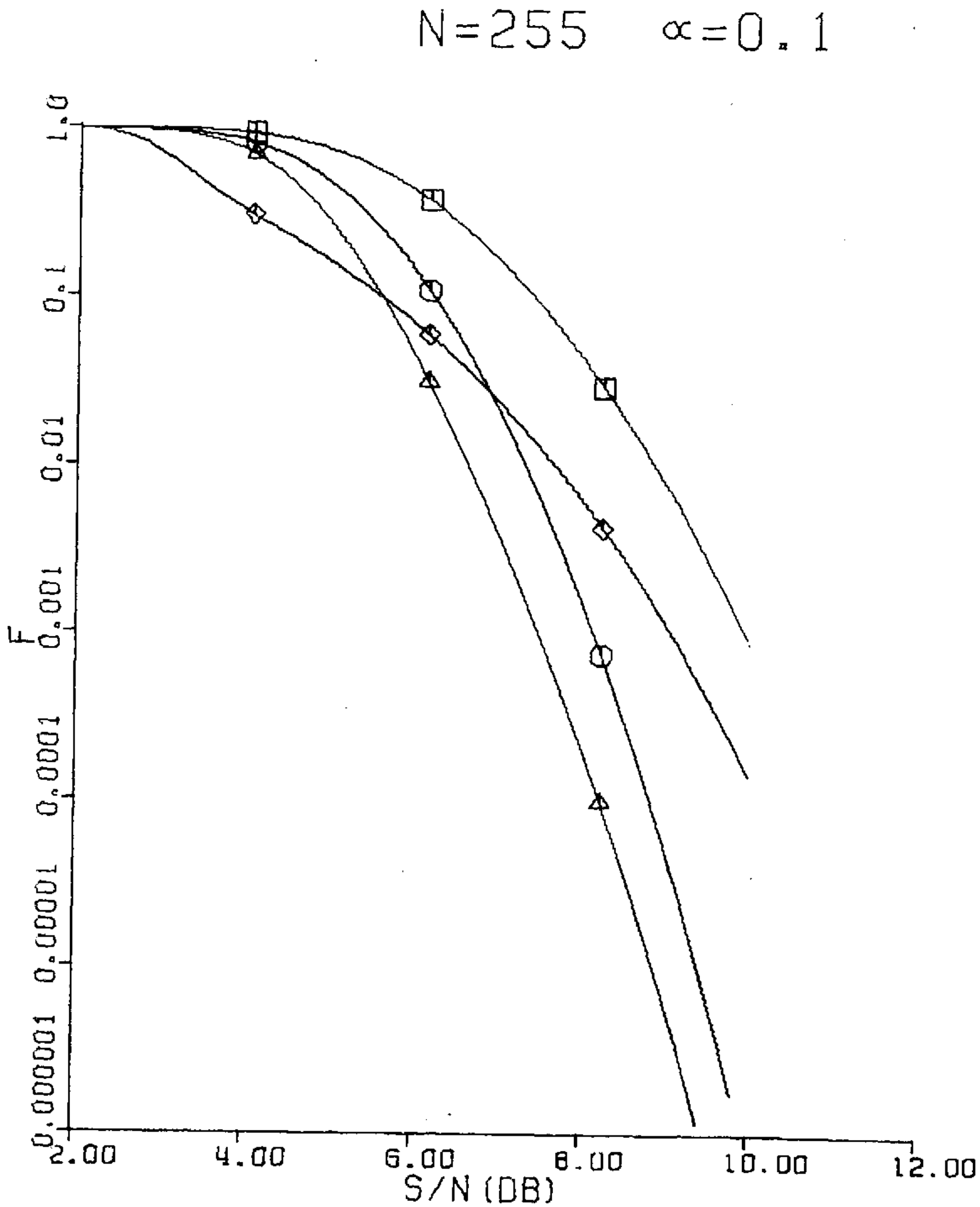

N=255 α=0.1
1.0
0.1
0.01
0.001
0.0001
0.00001
0.000001
F
2.00
4.00
6.00
8.00
10.00
12.00
S/N (DB)

FIGURE 33

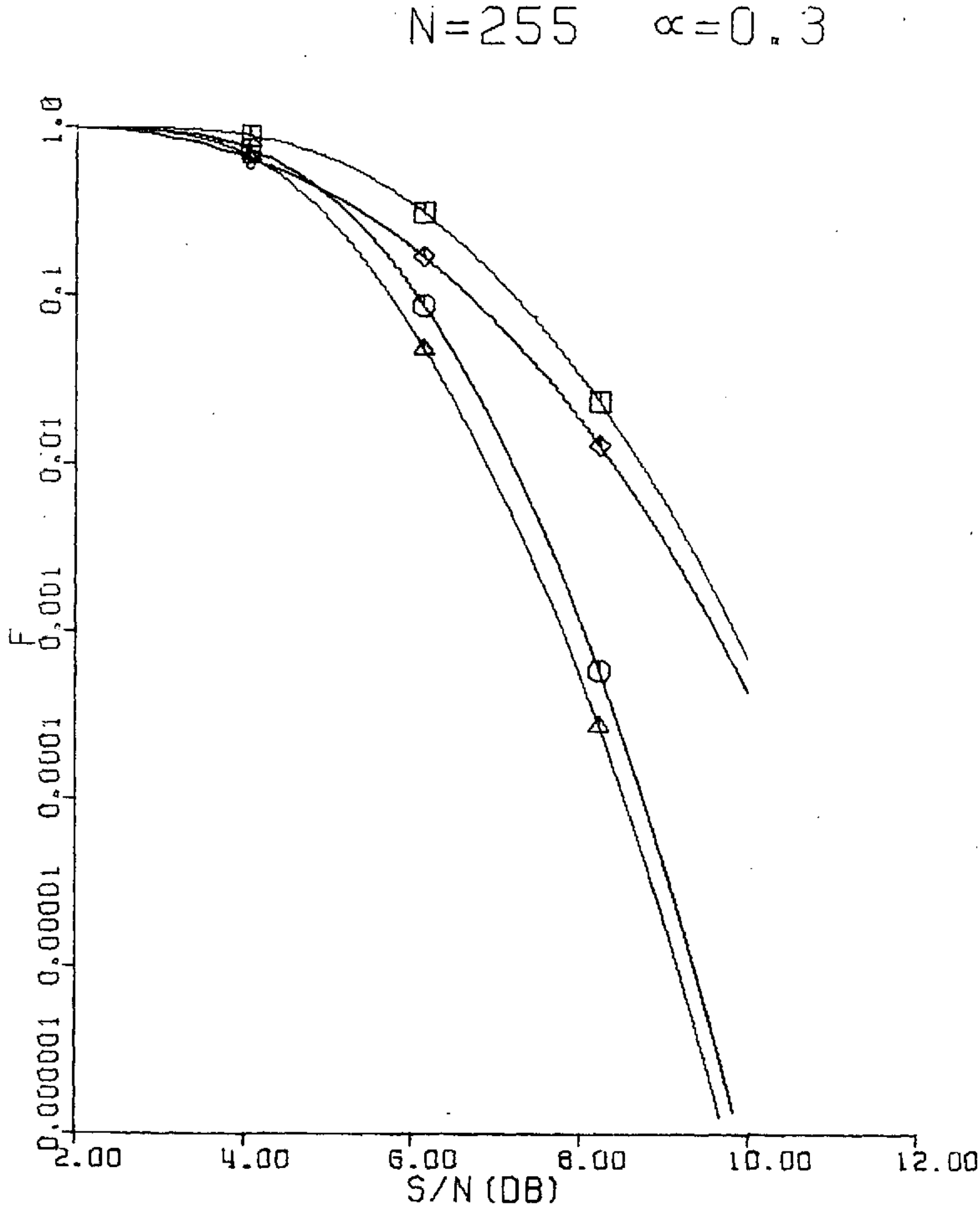

N=255 α=0.3
1.0
0.1
0.01
0.001
0.0001
0.00001
0.000001
0.0000001
F
2.00
4.00
6.00
8.00
10.00
12.00
S/N (DB)

FIGURE 34

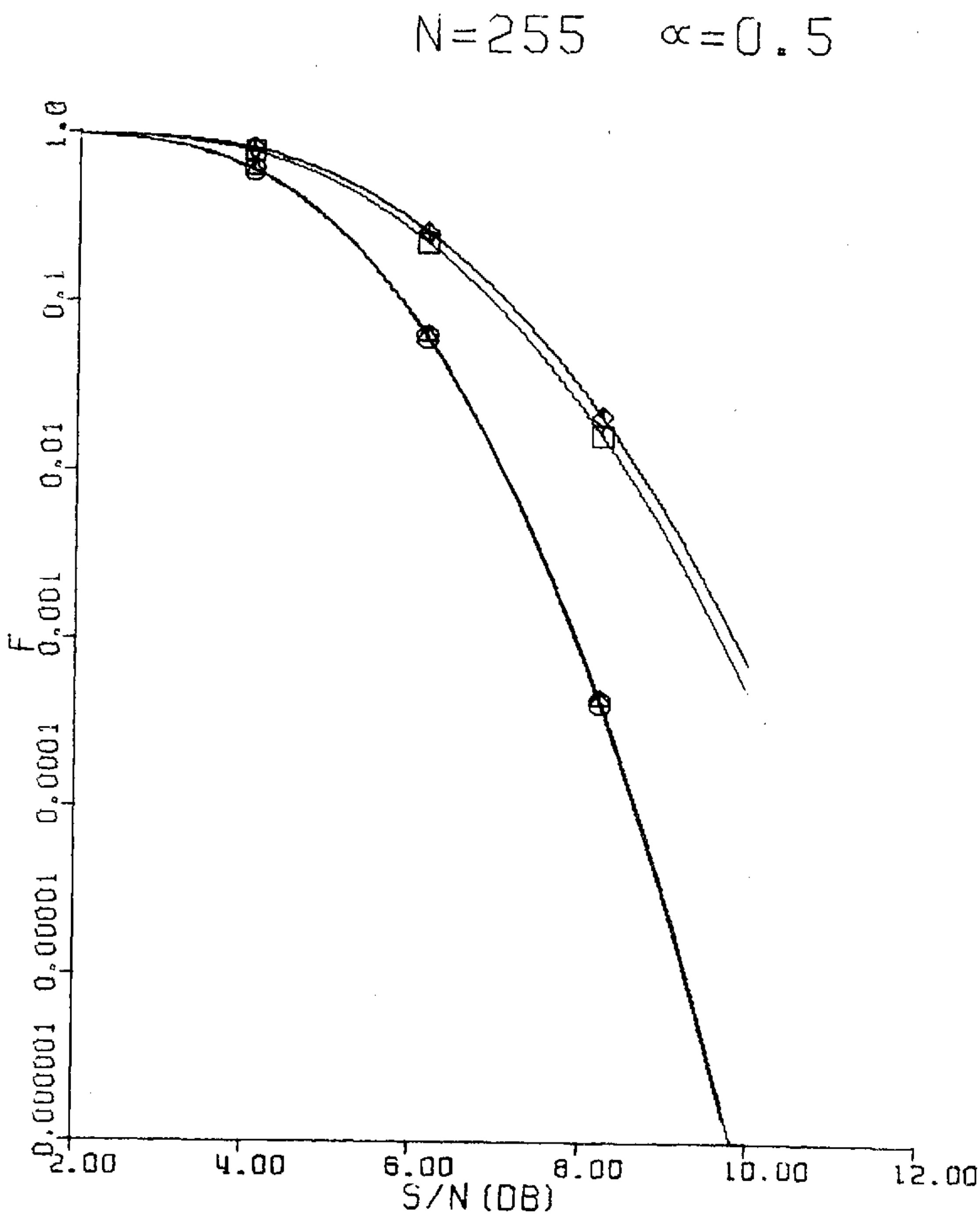
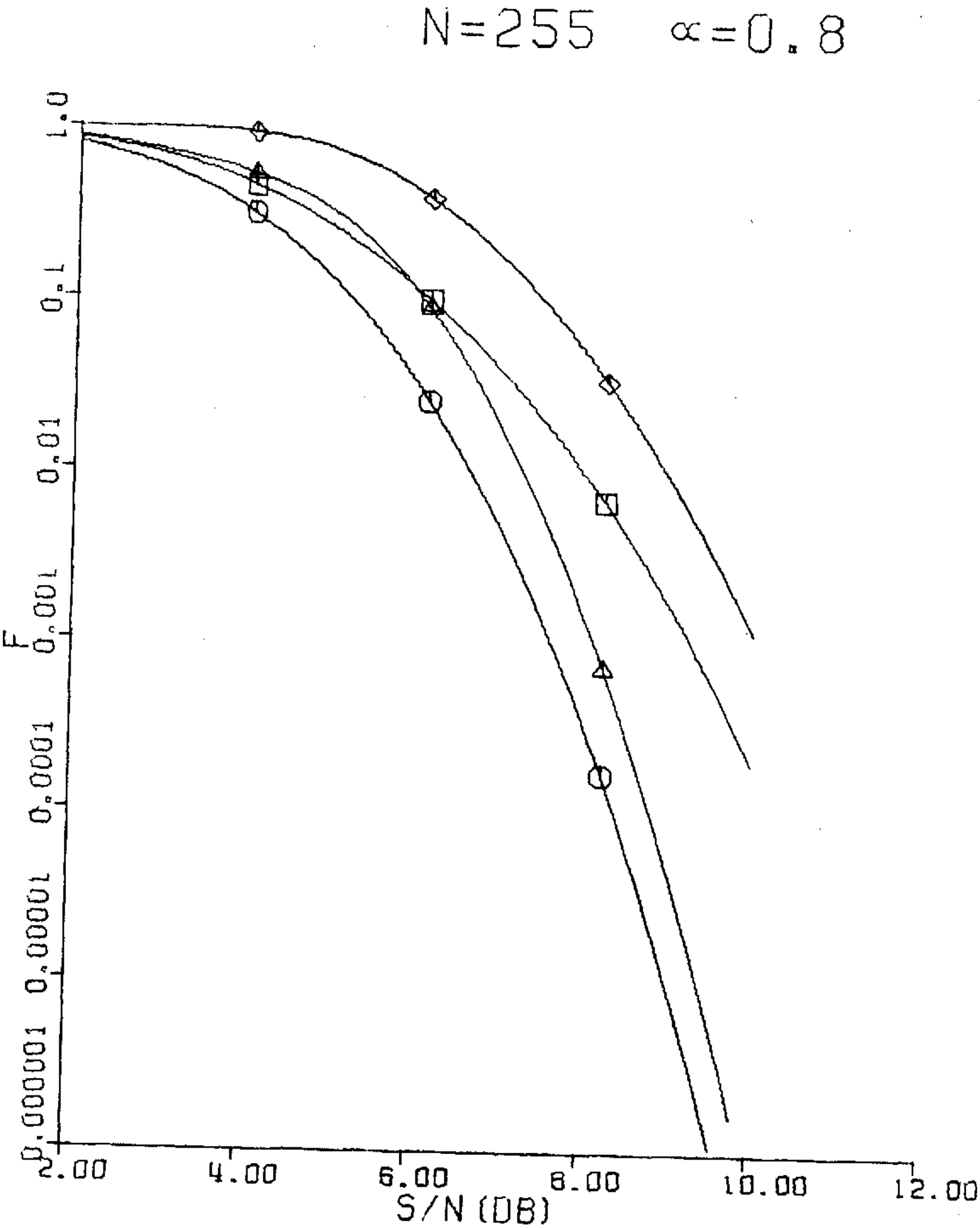

N=255 α=0.5
1.0
0.1
0.01
0.001
0.0001
0.00001
0.000001
F
2.00
4.00
6.00
8.00
10.00
12.00
S/N (DB)

FIGURE 35



N=255 α=0.8
1.0
0.1
0.01
0.001
0.0001
0.00001
0.000001
0.000001
F
2.00
4.00
6.00
8.00
10.00
12.00
S/N (DB)

FIGURE 36

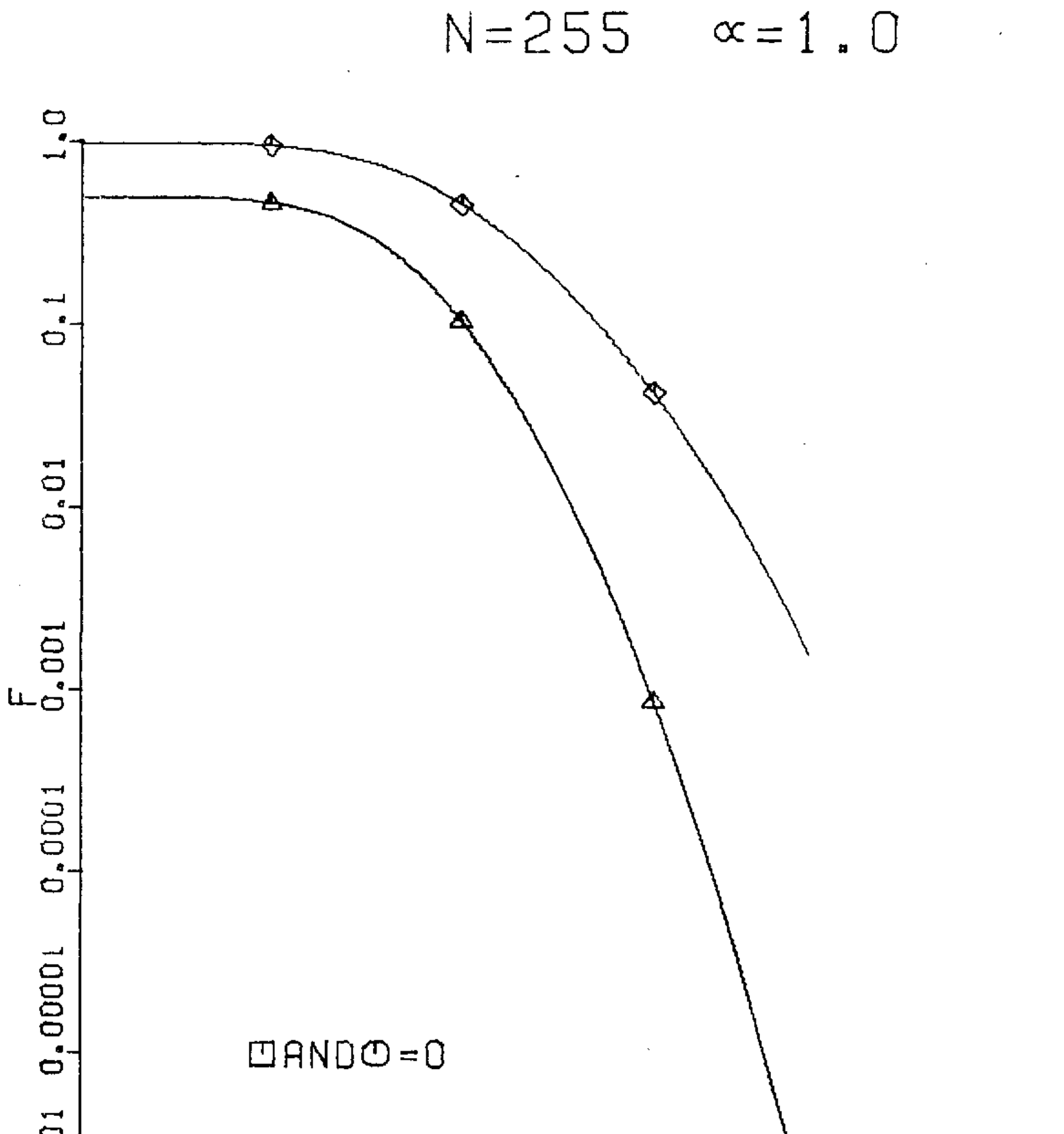
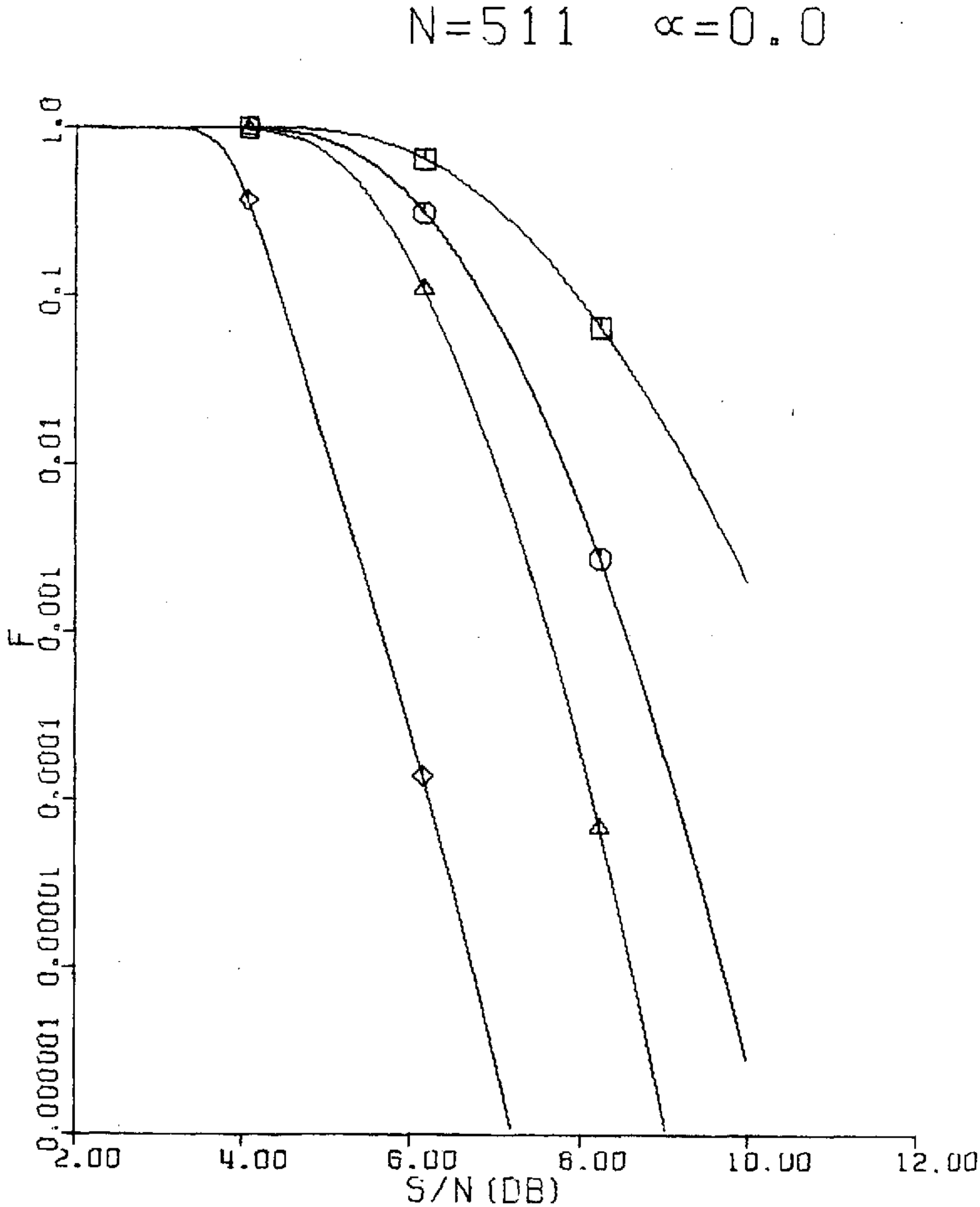

N=255 ∝=1.0
1.0
0.1
0.01
0.001
0.0001
0.00001
0.000001
F
□AND⊕=0
2.00
4.00
6.00
8.00
10.00
12.00
S/N (DB)

FIGURE 37



N=511 α=0.0
1.0
0.1
0.01
0.001
0.0001
0.00001
0.000001
0.0
F
2.00
4.00
6.00
8.00
10.00
12.00
S/N (DB)

FIGURE 38

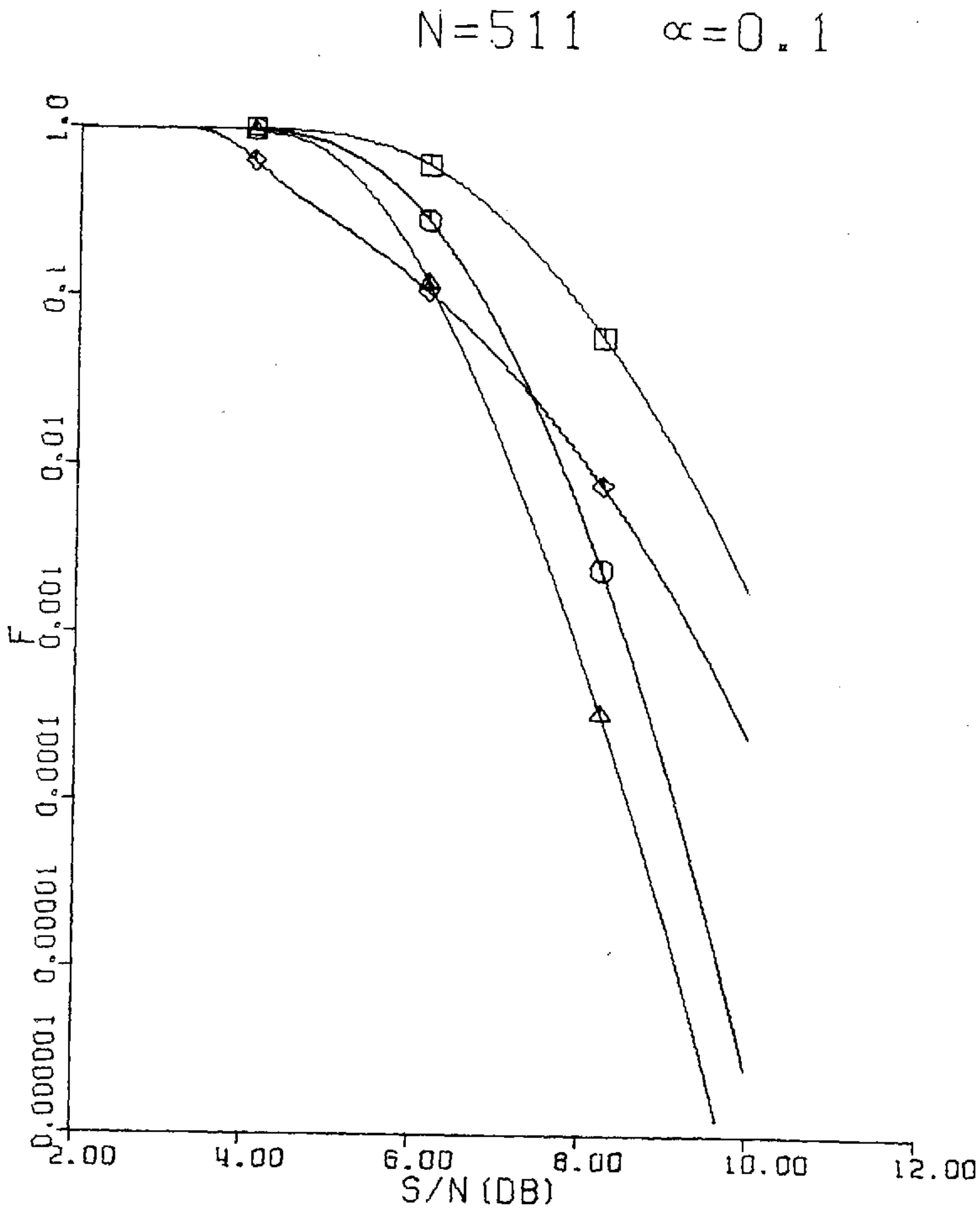

N=511 α=0.1
1.0
0.1
0.01
0.001
0.0001
0.00001
0.000001
0.0000001
F
2.00
4.00
6.00
8.00
10.00
12.00
S/N (DB)

FIGURE 39

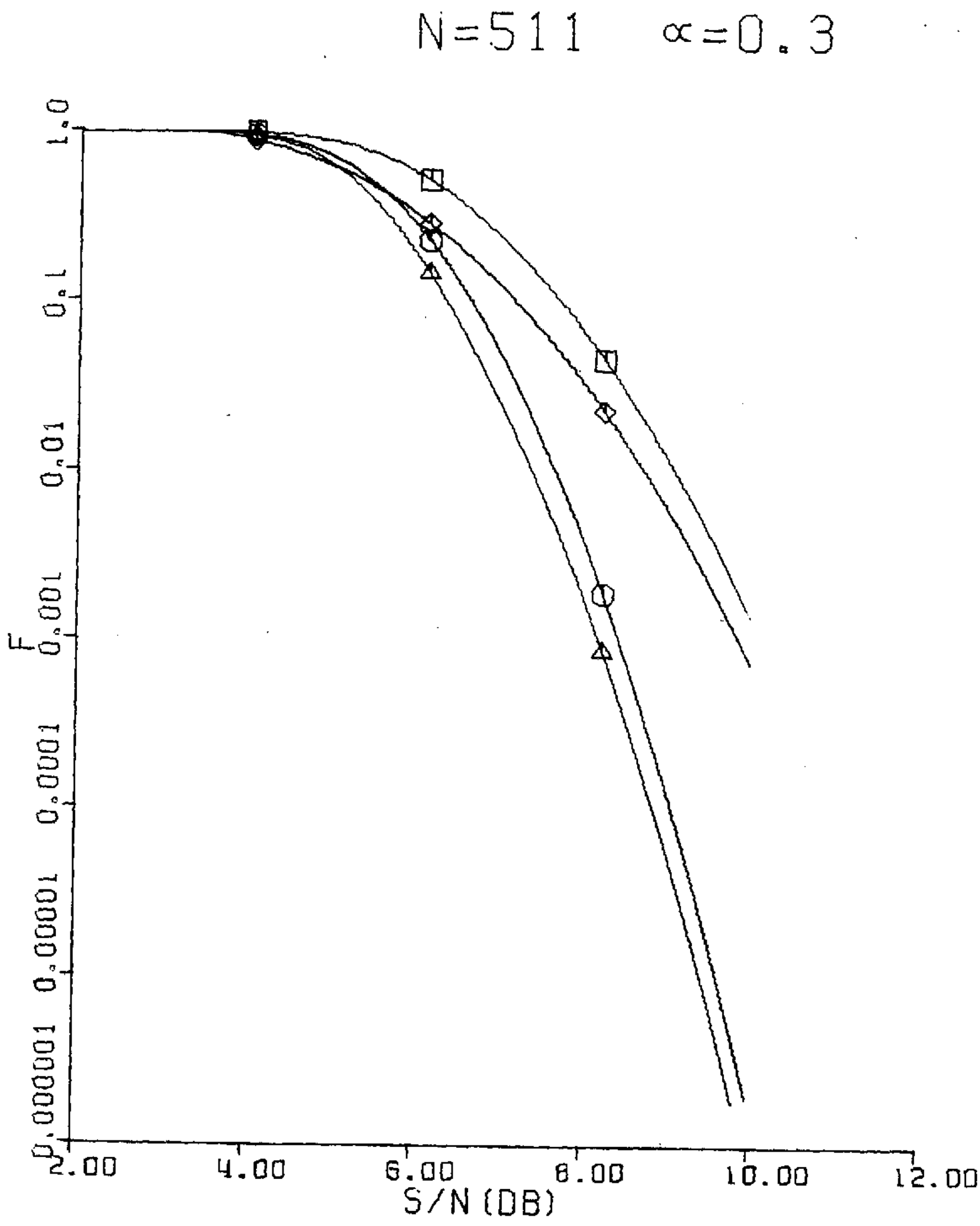
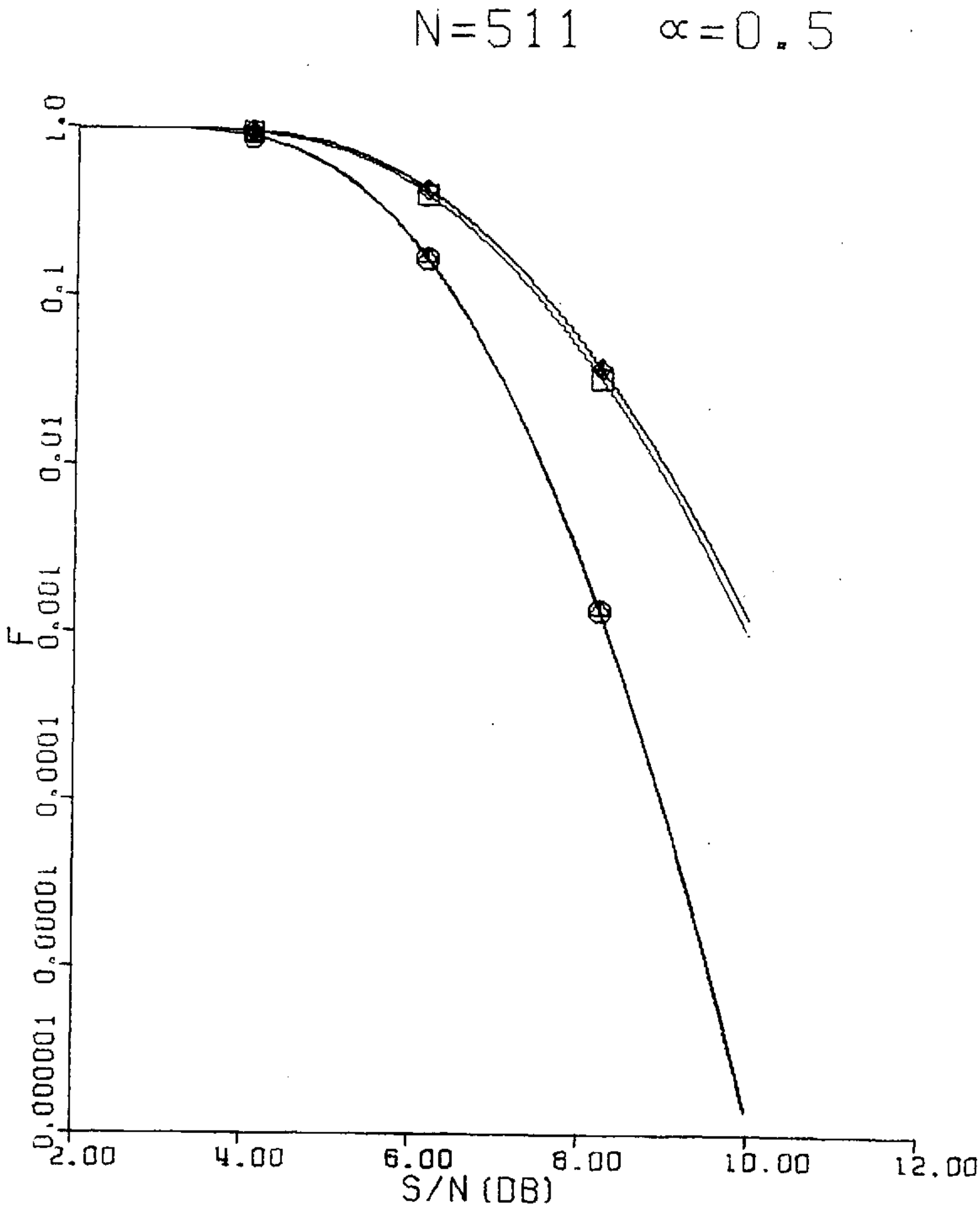

N=511 α=0.3
1.0
0.1
0.01
0.001
0.0001
0.00001
0.000001
F
2.00
4.00
6.00
8.00
10.00
12.00
S/N (DB)

FIGURE 40



N=511 α=0.5
1.0
0.1
0.01
0.001
0.0001
0.00001
0.000001
0.0000001
F
2.00
4.00
6.00
8.00
10.00
12.00
S/N (DB)

FIGURE 41

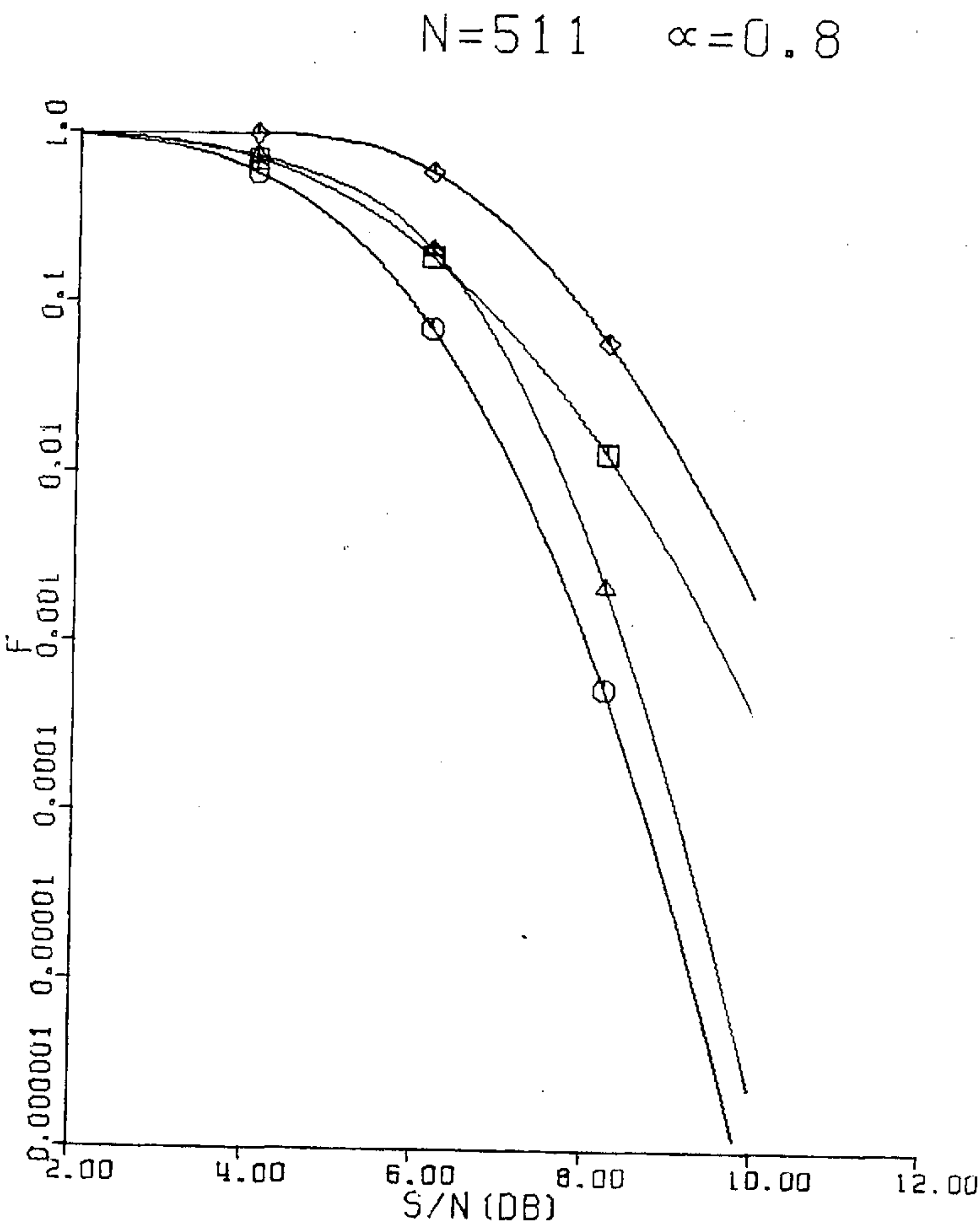
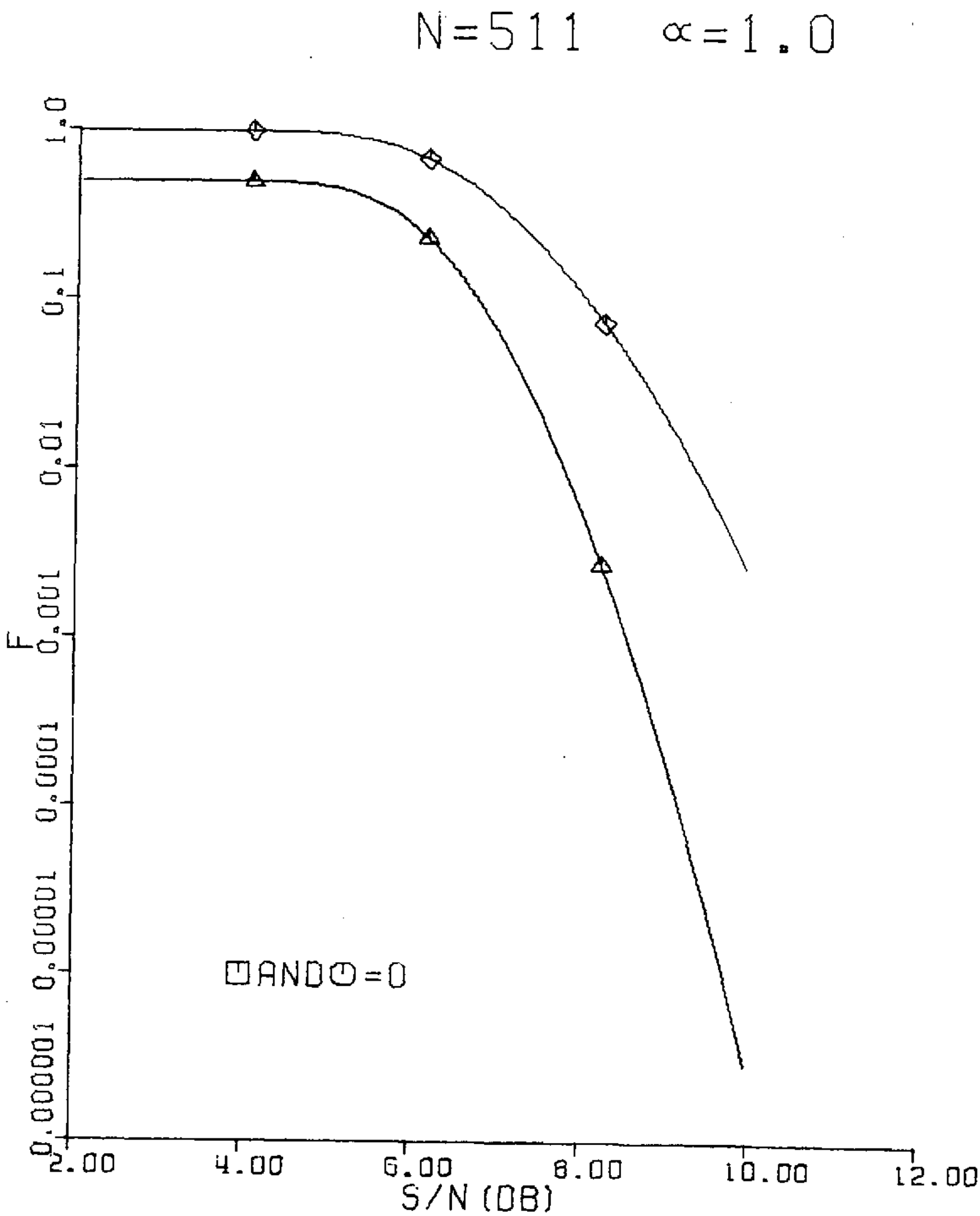
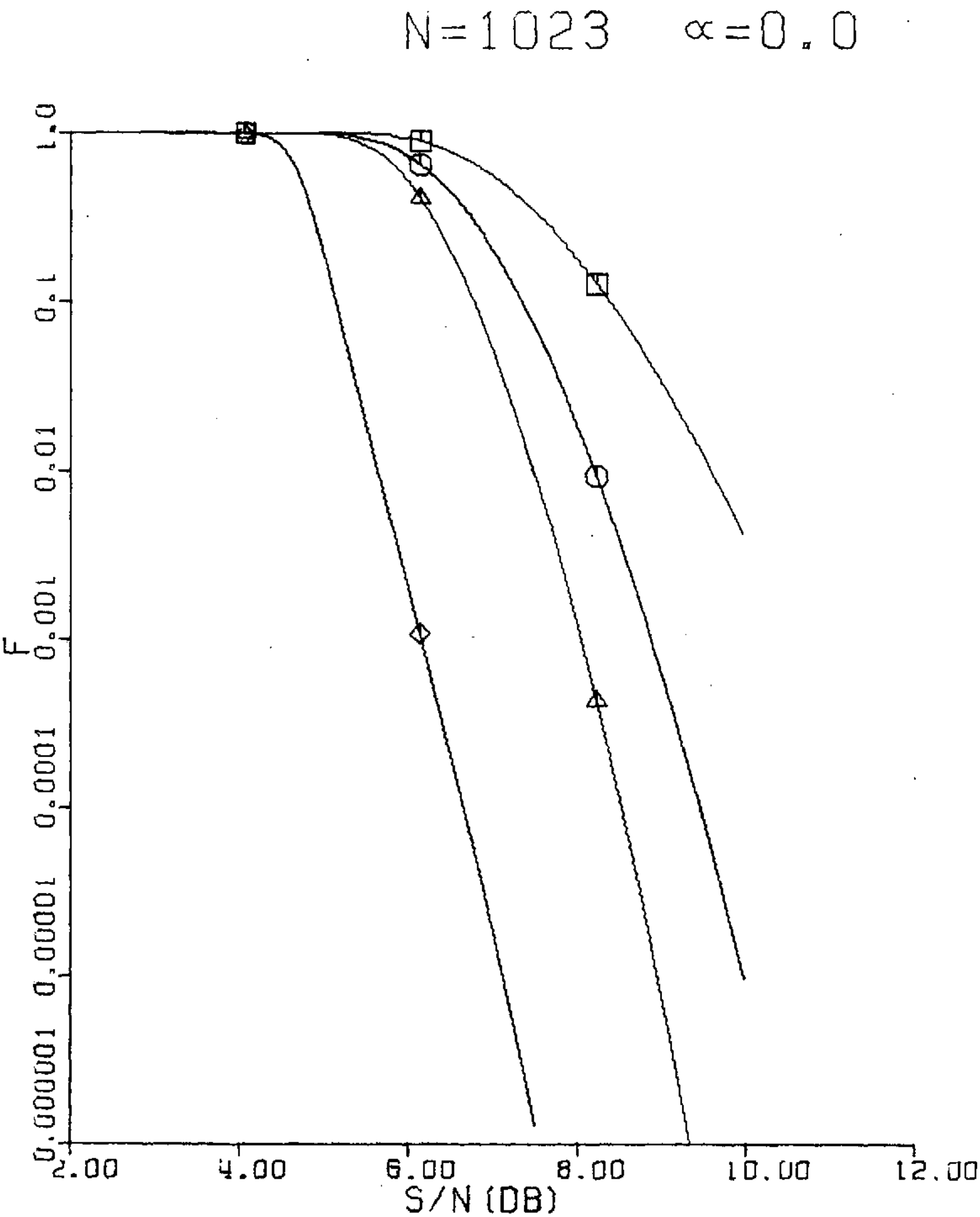

N=511 α=0.8
1.0
0.1
0.01
0.001
0.0001
0.00001
0.000001
0.0000001
F
2.00
4.00
6.00
8.00
10.00
12.00
S/N (DB)

FIGURE 42



N=511 α=1.0
1.0
0.1
0.01
0.001
0.0001
0.00001
0.000001
0.
F
□AND○=0
2.00
4.00
6.00
8.00
10.00
12.00
S/N (DB)

FIGURE 43



N=1023 α=0.0
1.0
0.1
0.01
0.001
0.0001
0.00001
0.000001
0.0000001
F
2.00
4.00
6.00
8.00
10.00
12.00
S/N (DB)

FIGURE 44

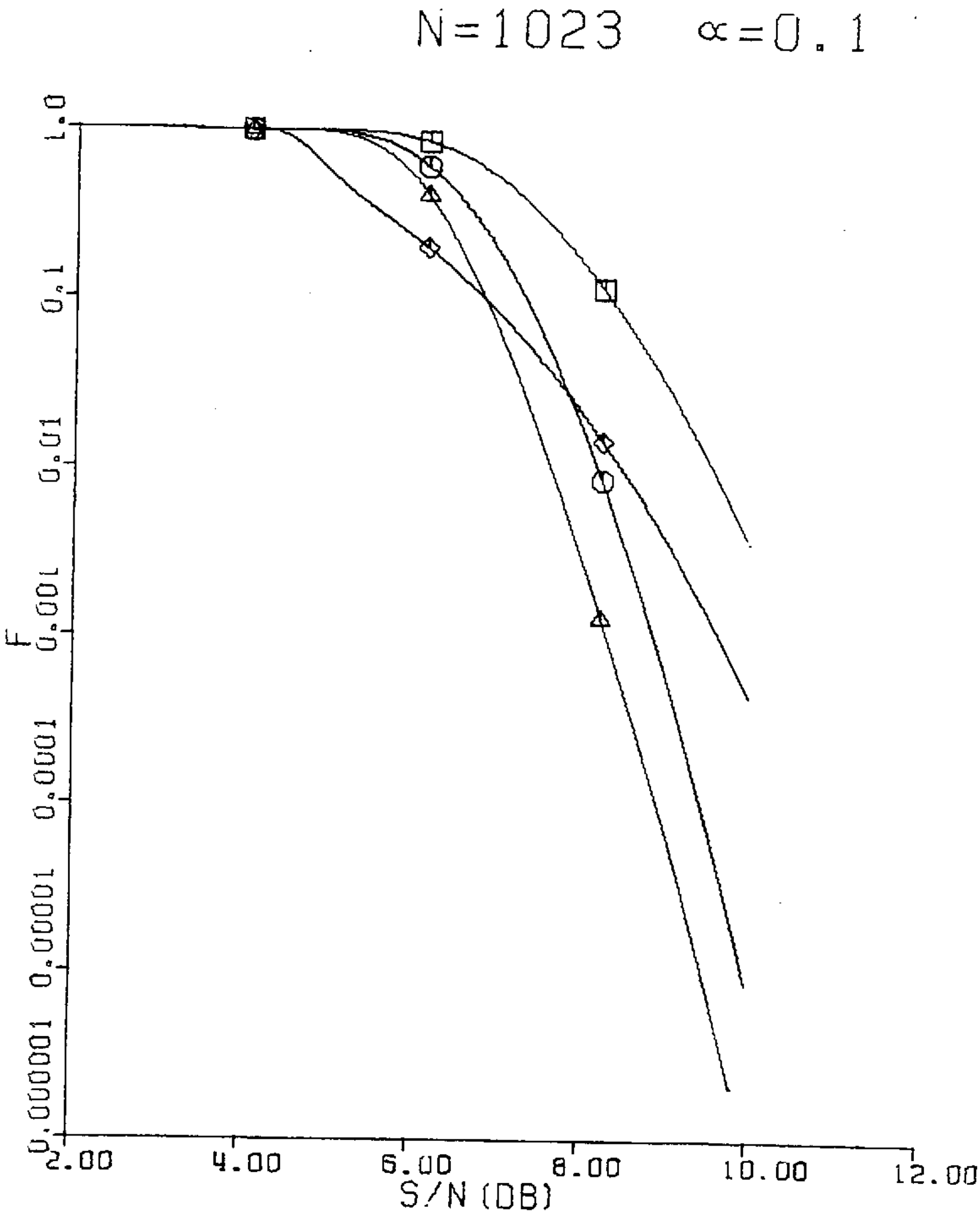
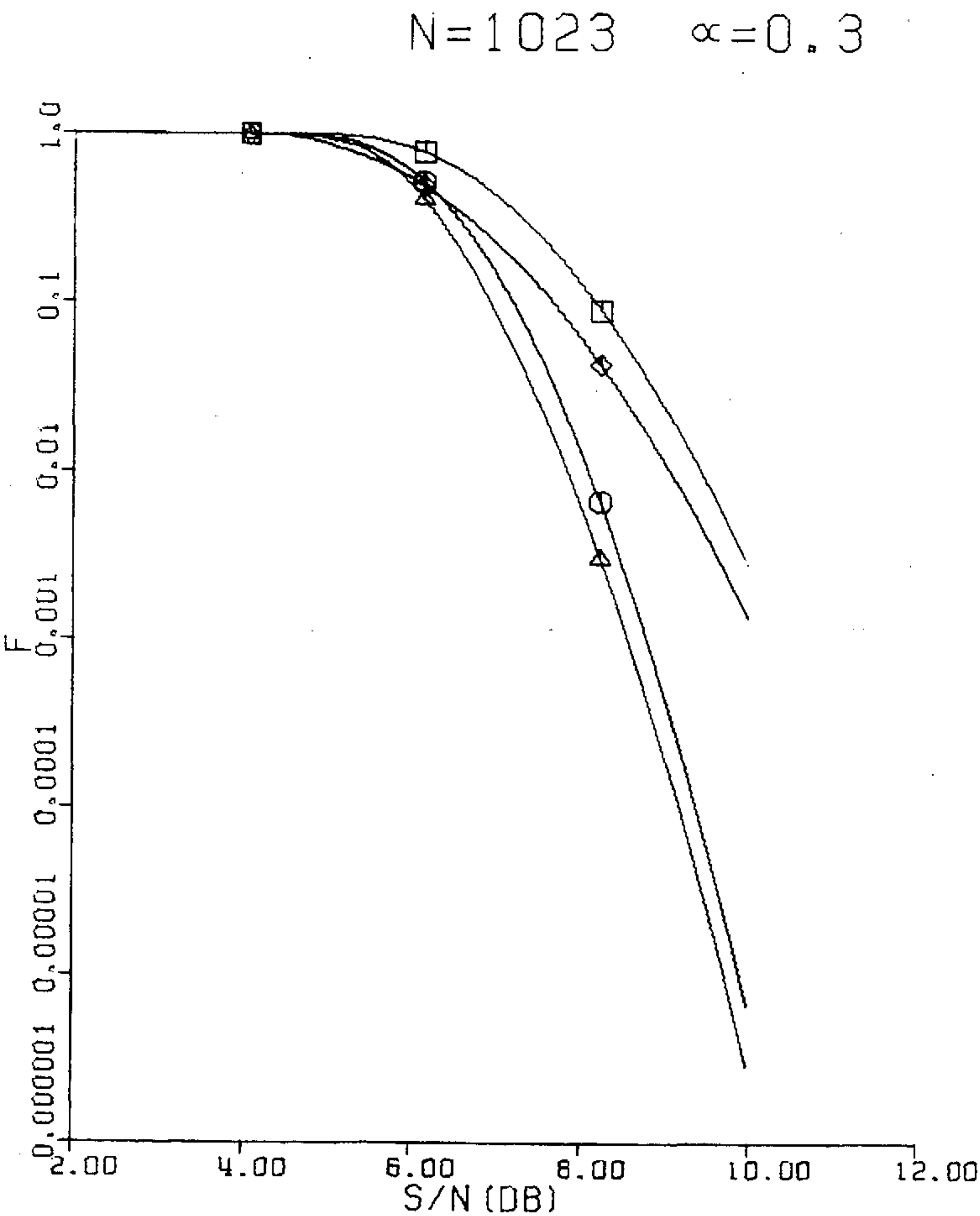
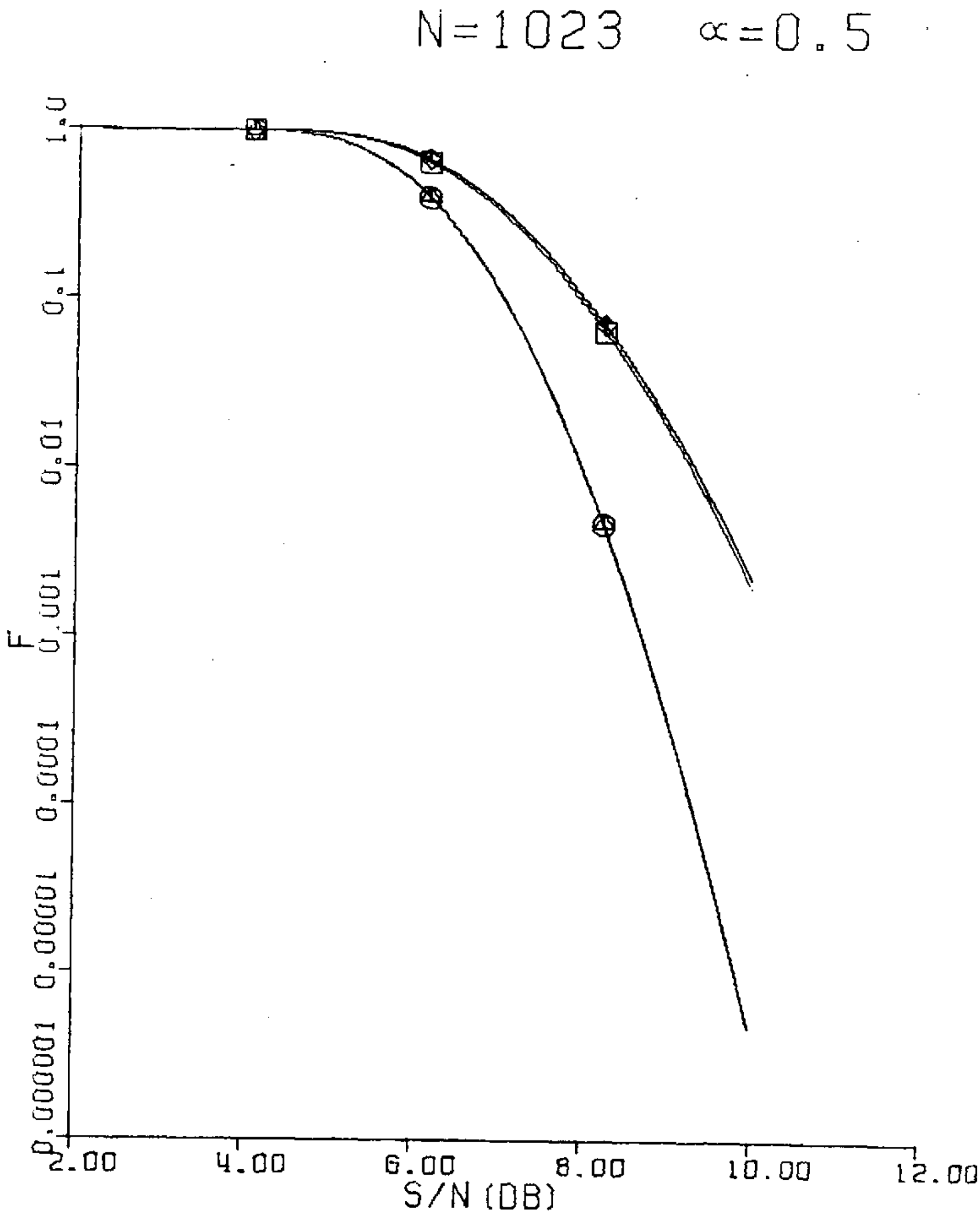

N=1023 α=0.1
1.0
0.1
0.01
0.001
0.0001
0.00001
0.000001
0.0
F
2.00
4.00
6.00
8.00
10.00
12.00
S/N (DB)

FIGURE 45



N=1023 α=0.3
1.0
0.1
0.01
0.001
0.0001
0.00001
0.000001
0.0000001
F
2.00
4.00
6.00
8.00
10.00
12.00
S/N (DB)

FIGURE 46



N=1023 α=0.5
1.0
0.1
0.01
0.001
0.0001
0.00001
0.000001
F
2.00
4.00
6.00
8.00
10.00
12.00
S/N (DB)

FIGURE 47

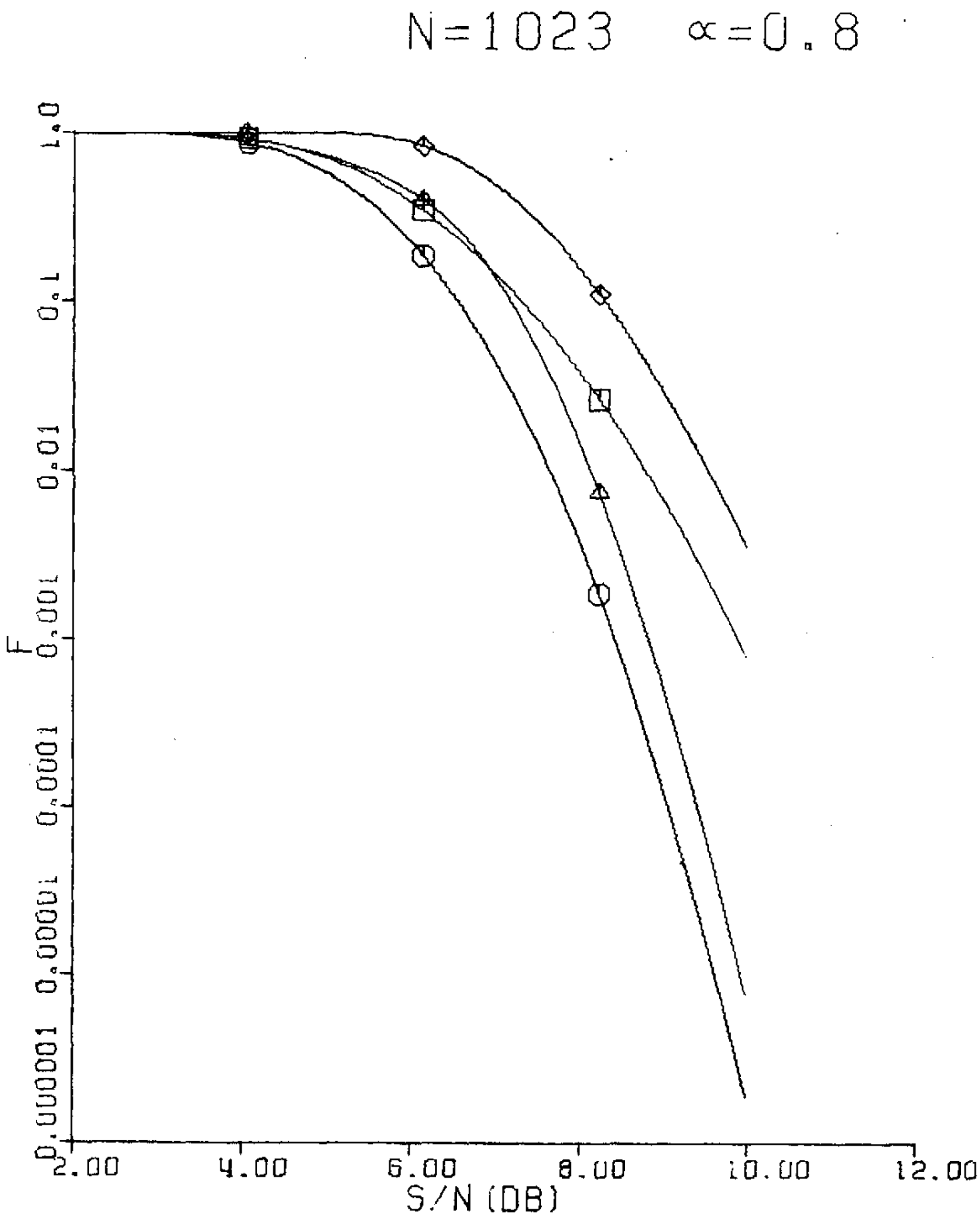

N=1023 α=0.8
1.0
0.1
0.01
0.001
0.0001
0.00001
0.000001
0.0000001
F
2.00
4.00
6.00
8.00
10.00
12.00
S/N (DB)

FIGURE 48

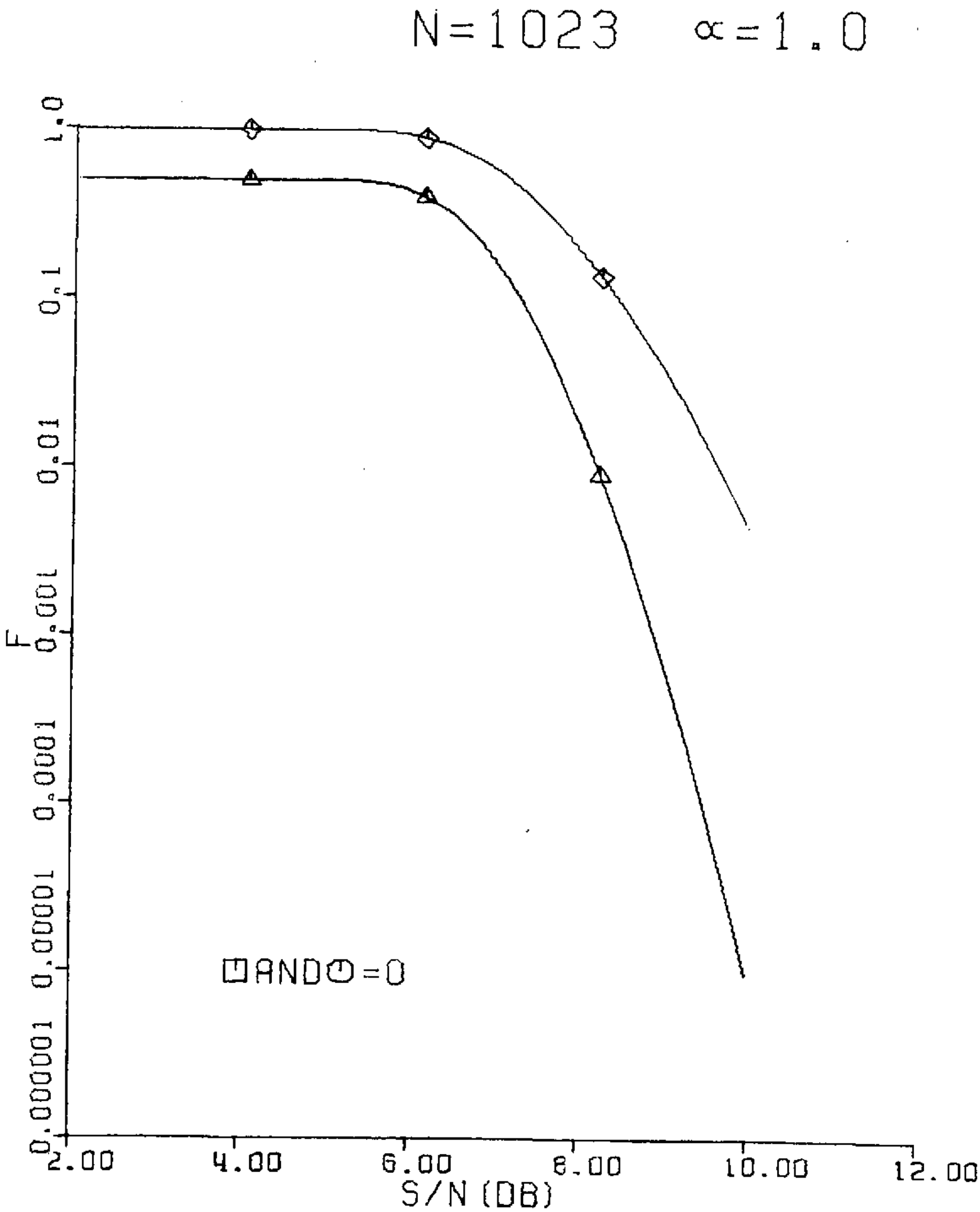

N=1023 α=1.0
1.0
0.1
0.01
0.001
0.0001
0.00001
0.000001
0.0000001
F
2.00
4.00
6.00
8.00
10.00
12.00
S/N (DB)

FIGURE 49

all four cases described above, and for n = 7, 15, 31, 63, 127, 255, 511, 1023, and α = 0, 0.1, 0.3, 0.5, 0.8, 1.0. We draw the following conclusions. For the extreme values of alpha the relative importance of each case remains fixed for all values of E_b/N_o. When $\alpha = 0$, indicating an interest only in the quality of the output, the relative ratings from best to worst are: TED, SEC-DED, SEC, No coding. As expected, when $\alpha = 1$, indicating an interest only in the quantity of output data, the relative ratings are just opposite to the $\alpha = 0$ cases. For $\alpha = .5$ and $n \geq 15$, the relative order also remains fixed: SEC, SEC-DED, No coding, TED. Notice that the extreme cases of large quantity of output achievable with no coding and high quality of output achievable with a TEC system are both given poor relative ratings for this value of alpha showing no preference of quantity over quality or vice-versa. Also, for $\alpha = .5$ $n = 7$, no coding becomes preferable to SEC-DED at signal-to-noise ratios below 4.1 db. This would be due to the increased data rejection by a SEC-DED decoder as the noise becomes greater.

A preference for quality over quantity without total disinterest in the latter is explored by the $\alpha = .1$ and $\alpha = .3$ cases. No coding for these values of alpha is never preferable to any other of the choices considered except TED. As S/N increases TED becomes relatively less desirable as triple errors become less likely and its low transmission rate becomes dominant. Similarly, as E_b/N_o increases SEC-DED becomes less important than just SEC. However, as the block size increases, TED and SEC-DED become more important since the probabilities of the errors these decoders are designed to correct increase.

Finally, when quantity is somewhat preferred over quality as with $\alpha = .8$, as might be expected the relatively extreme quality achieved by TED is shown to be undesirable for all values of S/N tested since this quality is achieved at the expense of quantity. The SEC decoder which employs no data rejection yet achieves some degree of error correction is found to be the best of all

PART B

four cases for all values of E_b/N_o considered. At very low noise levels the error correcting properties of SEC-DED make it more desirable than no coding, while at high noise levels the data-rejection of SEC-DED make it less desirable than no coding. For example, for n = 1023, SEC-DED is preferable to no coding for E_b/N_o from 4.4 to 6.9 db.

More insight into the nature of this function can be gained by looking at what is necessary to achieve a desired level of effectiveness. A typical example is shown in Table 1. Here the desired value of F is set at .01. With no coding or SEC, a higher value of E_b/N_o is required as quality becomes more preferable. However, with SEC-DED or TED a lower value of E_b/N_o is required to achieve the same value of F as emphasis is switched to quantity.

E_b/N_o IN DB REQUIRED TO

ACHIEVE F = .01 FOR n = 255

Alpha	No Coding	SEC	SEC-DED	TED
.1	8.88	7.36	6.64	7.68
.3	8.72	7.20	6.88	8.40
.5	8.56	7.04	7.12	8.72
.8	8.00	6.64	7.28	8.96

TABLE 1

DIGITAL SIMULATION OF THE VITERBI MAXIMUM LIKELIHOOD DECODING ALGORITHM

I. Introduction:

The Viterbi algorithm is a method for determining the most likely sequence of states of a time-discreet Markov process; and, as such it is an optimal method for decoding convolutional codes. An evaluation of the effectiveness of this algorithm as a decoding method is accomplished herein through simulation on an IBM 370 computer using a main program written in the Fortran language and three subroutines written in Assembler language.

II. The Simulation Procedure:

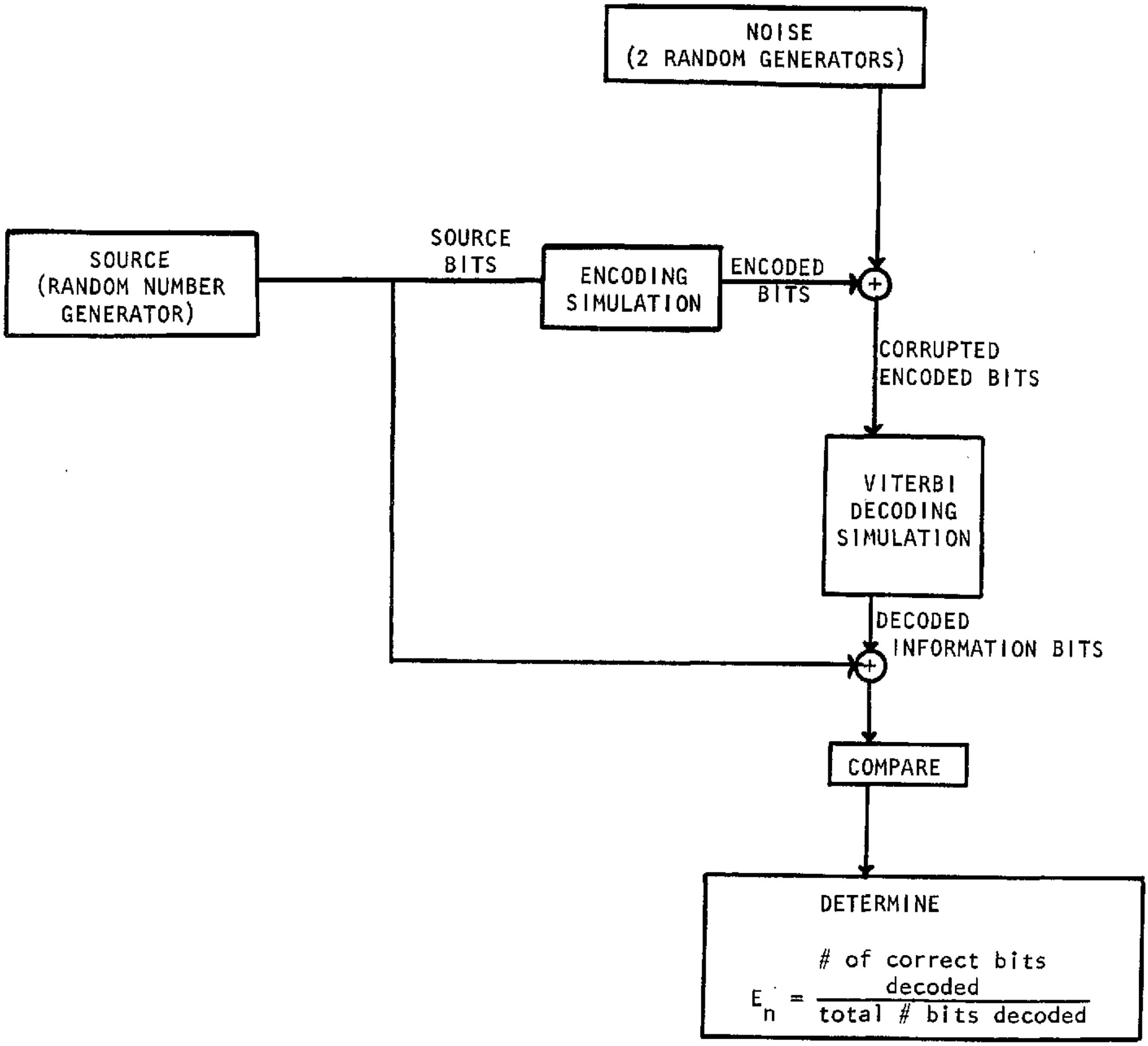
A block diagram of the simulation is shown in Figure 50. A pseudo-random number generator is used to independently generate binary source bits of equal probability and binary noise bits whose probabilities depend on the assumed channel characteristics. The source bits are encoded with the appropriate parity check bits in blocks of two. Each bit is added to a noise bit using modulo-2 arithmetic (simulating channel noise). The information noise bits and parity check noise bits are generated independently. The corrupted bits are then decoded using the Viterbi algorithm. Accuracy of the decoding algorithm is measured as

$$E = \lim_{n \to \infty} E_n = \text{(Number of correct information bits)/(Total bits)}$$

where E_n is the ratio after information bits have been transmitted.

III. The Viterbi Algorithm:

Given an observed output sequence $Z = (z_1, z_2, \ldots, z_k)$, the purpose of the Viterbi algorithm is to determine the most likely input sequence $X = (x_0,$



BLOCK DIAGRAM OF THE VITERBI ALGORITHM SIMULATION

FIGURE 50

$x_1, \ldots, x_k$). The subscripts refer to the discreet time states. Since the process is assumed to be Markov, the probability of state x_{i+1} depends only on the state x_i:

$$\text{i.e.,} \quad Pr(x_{i+1}|X_0, x_1, \ldots, x_i) = Pr(x_{i+1}|x_i).$$

The channel is assumed to be memoryless so that the observed output z_i at time i depends only on the transition from state x_i to state x_{i+1}. This transition is symbolized as t_i. We want to determine the maximum <u>a posteriori</u> Pr(X, Z). Because of the aforementioned Markov and memoryless assumptions:

$$Pr(X, Z) = Pr(X)Pr(Z|X)$$

$$= \prod_{i=0}^{k-1} Pr(x_{i+1}, x_i) \prod_{i=0}^{k-1} Pr(z_i|t_i)$$

The Viterbi algorithm is a method of determining the shortest path between two points. We, therefore, assign a 'path length' between each pair of possible states from time = i to time = i + 1. This length lambda(t_i) is defined as

$$\text{lambda}(t_i) = -\ln Pr(x_{i+1}|x_i) - \ln Pr(z_i|t_i).$$

The total length for some input sequence X would be

$$-\ln Pr(X, Z) = \sum_{i=0}^{k-1} \text{lambda}(t_i)$$

Since path length is a negative logarithm of the probability, the shortest (critical) path length between two points (i.e., the initial and final states) would be the one with the highest probability. This is the maximum <u>a posteriori</u> probability we are seeking. The Viterbi method of finding this critical path is based on the observation that at any given time i, each state x_i has associated with it a shortest path to the initial state. This shortest sequence is called a survivor, designated $\hat{X}(x_i)$. The path length of survivor $\hat{X}(x_i)$ is designated gamma (x_i). Extending these survivors to time $i + 1$ requires merely adding the appropriate digit ("bit" for our purposes) to the existing survivor and adding the corresponding path length gamma (x_{i+1}, x_i) for comparison purposes in determining the survivors for each state at time $i + 1$. At the end of the sequence (time = k) the survivor corresponding to the state with the shortest survivor path length is optimal.

For the purpose of convolutional code decoding, the states correspond to the possible binary state permutations of a block of m shift registers. Assuming that for the source $Pr(0) = Pr(1) = \frac{1}{2}$, it follows that for all possible transitions between states the term $Pr(x_{i+1}|x_i) = \frac{1}{2}$; and, since it is a constant for all possible transitions, it may be ignored when calculating the optimal path. Thus, only the term $Pr(z_i|t_i)$ is significant. For systematic codes, the observation z_i corresponds to both the information bit and the parity bit received as a block at time = i. For non-systematic codes, z_i corresponds to a block containing a parity bit for each subgenerator polynomial. These probabilities are pre-calculated for each state and each possible received block before decoding begins.

Since we are concerned with a communications system with a semi-infinite number of bits transmitted, corresponding to a semi-infinite sequence, and since storing the resulting semi-infinite survivors is impractical, a limit must be placed on the number of bits stored as a survivor (i.e., the survivor length). Call this limit delta. Thus, at time = i, a decision must be made concerning the bit at time = i - delta (i minus delta). This survivor transaction becomes insignificant for delta large enough because survivors tend to converge to the same state nodes.

IV. The Simulation Program for Rate ½ Codes:

A Fortran language main program was used in conjunction with three custom written Assembler language subroutines. The main thing to be aware of when using the program is that the delta defined in the program is one greater than the corresponding delta as defined in the Viterbi algorithm (e.g., if you wish to get results for delta = 75, set delta = 76 in this program!). For each state at time i there are two possible states to which it can branch at time i + 1 (one of which has an incoming 0 bit, the other has an incoming 1 bit). These possible transitions are stored in an array called NEXT. NEXT(I,1) corresponds to the branch of state I with an incoming 0; whereas, NEXT(I,2) corresponds to an incoming 1.

Probabilities which determine the path lengths are calculated prior to the main iterative loop. These calculations are done for all four possible two bit permutations corresponding to a received block containing an information bit and a parity check bit in the systematic case, or two parity check bits in the non-systematic case. Array POFZLN stores these predetermined path lengths. Thus in the main iterative loop the increase in the total path length GAMMA of each state can be determined by a simple table reference (i.e., POFZLN). Survivors are stored and saved by arrays SURVIV and SAVUR, while the corresponding path lengths are saved using arrays GAMMA and SAVE.

The appropriate subgenerator vectors are stored in the array GEN. In the case of a systematic code the second subgenerator is a 1 followed by m - 1 zeros. For example, the subgenerators for an m = 5 systematic code would be: 10000 and 11011. Note that the subgenerator 10000 merely generates the information bit. SHIFT2 saves the contents of the simulated encoding shift registers.

All random bit generation is done independently for each application. Source bits of equal probability are generated using a Scientific Subroutine Package member called RANDU. These source bits are stored in PRSOUR. The generated noise bits ($Pr(0) = q$) are stored in PNOISE. Decoded bits are stored in PROUT for comparison with the original source bits in PRSOUR.

Many different counters are used to keep track of time states corresponding to source bits, noise bits, and output bits. IOUT determines the printed increments for E_n. During the course of this research, IOUT was set equal to 1000 so that the accuracy E_n was printed out for n = 1000, 2000, 3000, etc. The n refers to the number of decoded bits and is called NDECOD within the program.

Read and punch statements are included to save the information necessary to restart the program where it left off. This feature is desirable to enable the programmer to check the convergence of the accuracy figures and compare them with other data in order to determine the desirability of decoding a greater number of bits. However, due to the fact that hexadecimal double precision accuracy used by the IBM 370 computer is equivalent to about 16.7 decimal digits, and the data cards are punched with decimal numbers, there is a slight loss in accuracy that is sometimes noticeable but generally insignificant.

Three Assembler languages subroutines were written to expedite the execution of the program. These are COPYAR, SHIFT, and TESTBT. COPYAR simply copies SAVSUR into SURVIV. SHIFT is used to shift the survivor of a row in SURVIV, add a 0 or a 1, and transfer the resulting survivor to the row specified by NEXT in the array SAVSUR for the next time increment. TESTBT determines whether the bit at time = k - delta (Viterbi definition of delta) is a 0 or a 1 in the survivor of the current state whose total path length is the least.

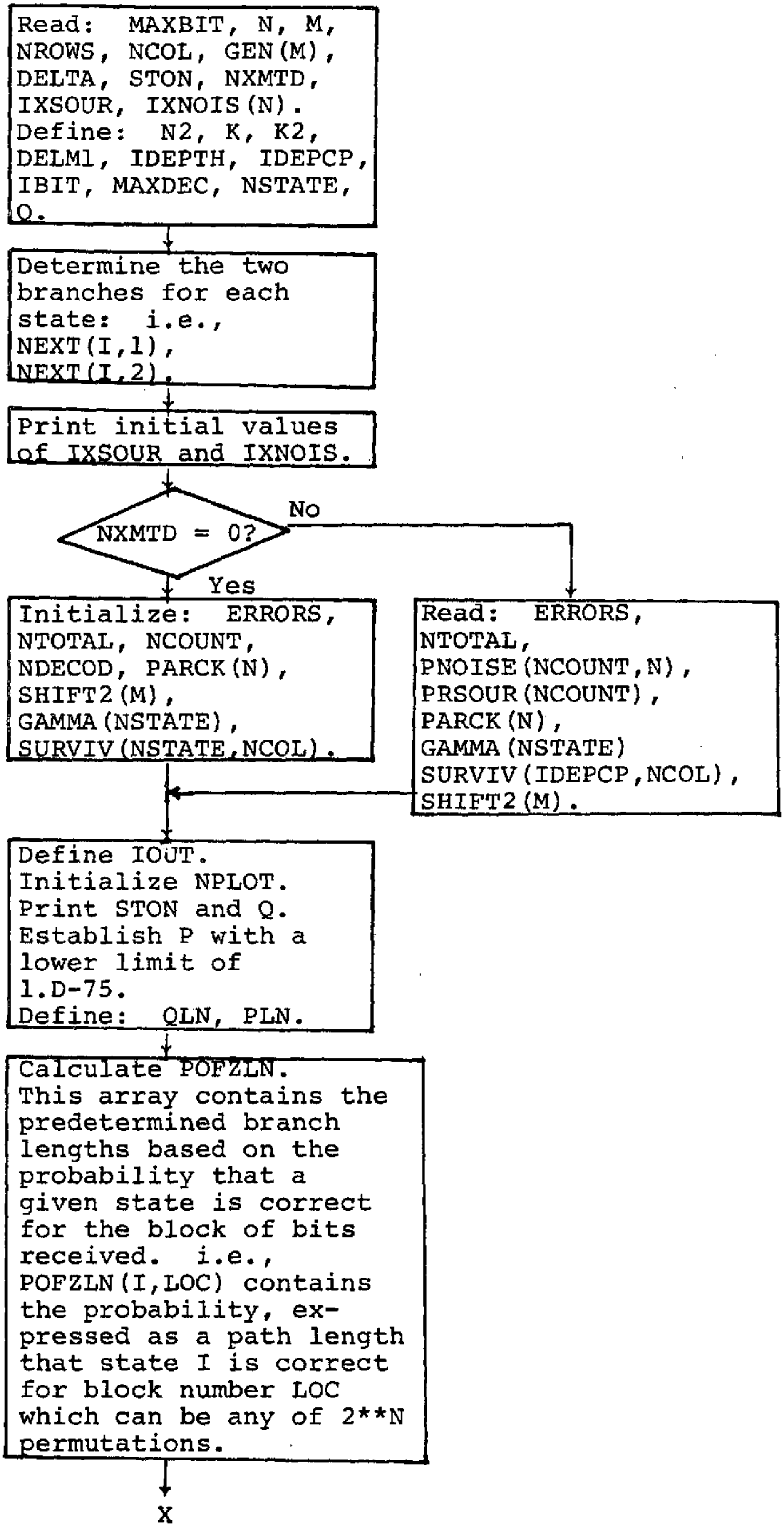
Typical decoding rates are shown in Table 2. The number of bits refers to the total number of both parity and information bits. To obtain the number of information bits decoded per second, multiply the rates shown in Table 2 by the code rate, which in all cases explored here is $\frac{1}{2}$. Note that decoding an $m = 7$ code is approximately twice as slow as decoding an $m = 6$ code since the latter has half the possible states of the former.

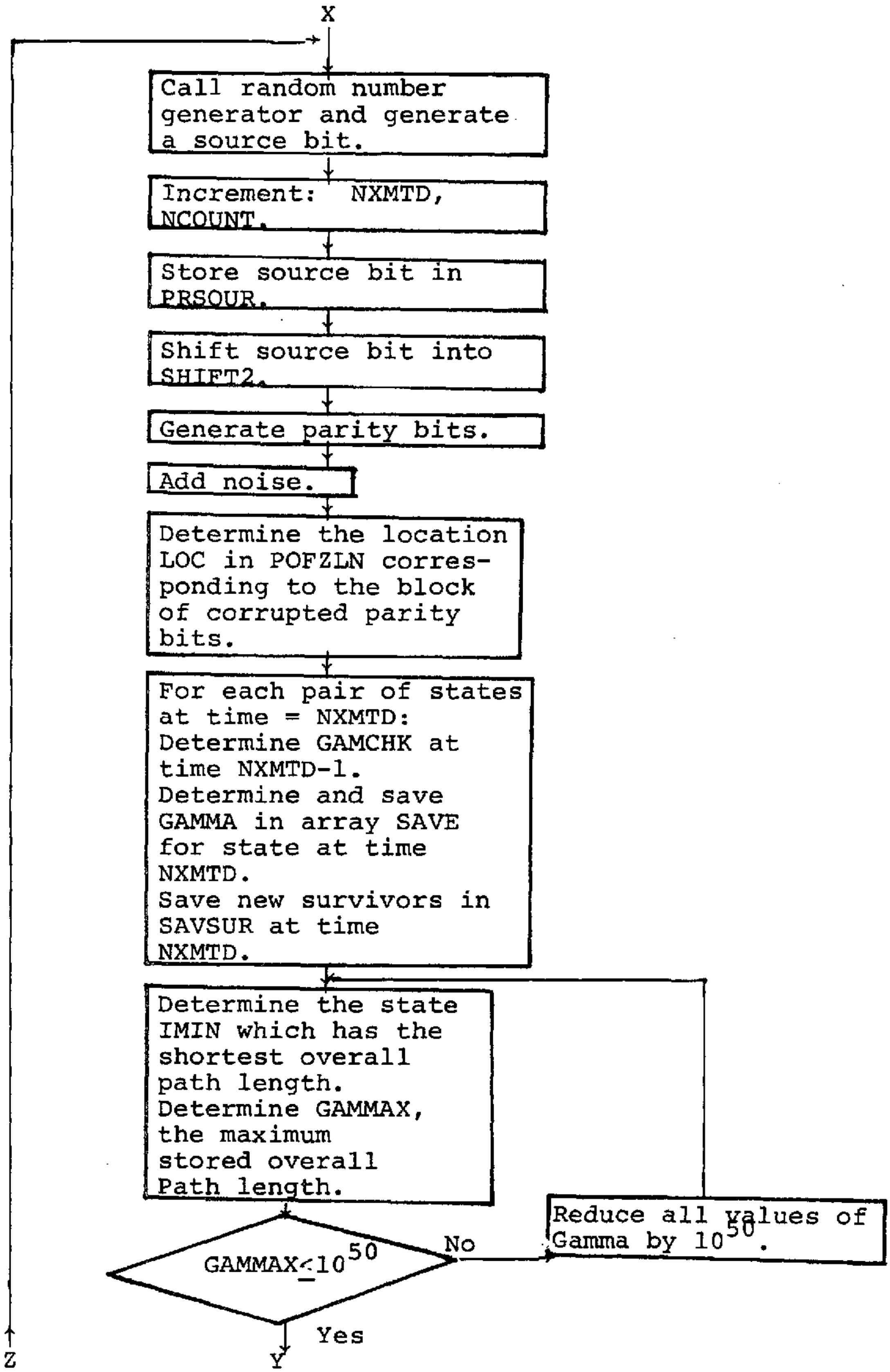
TYPICAL DECODING RATES

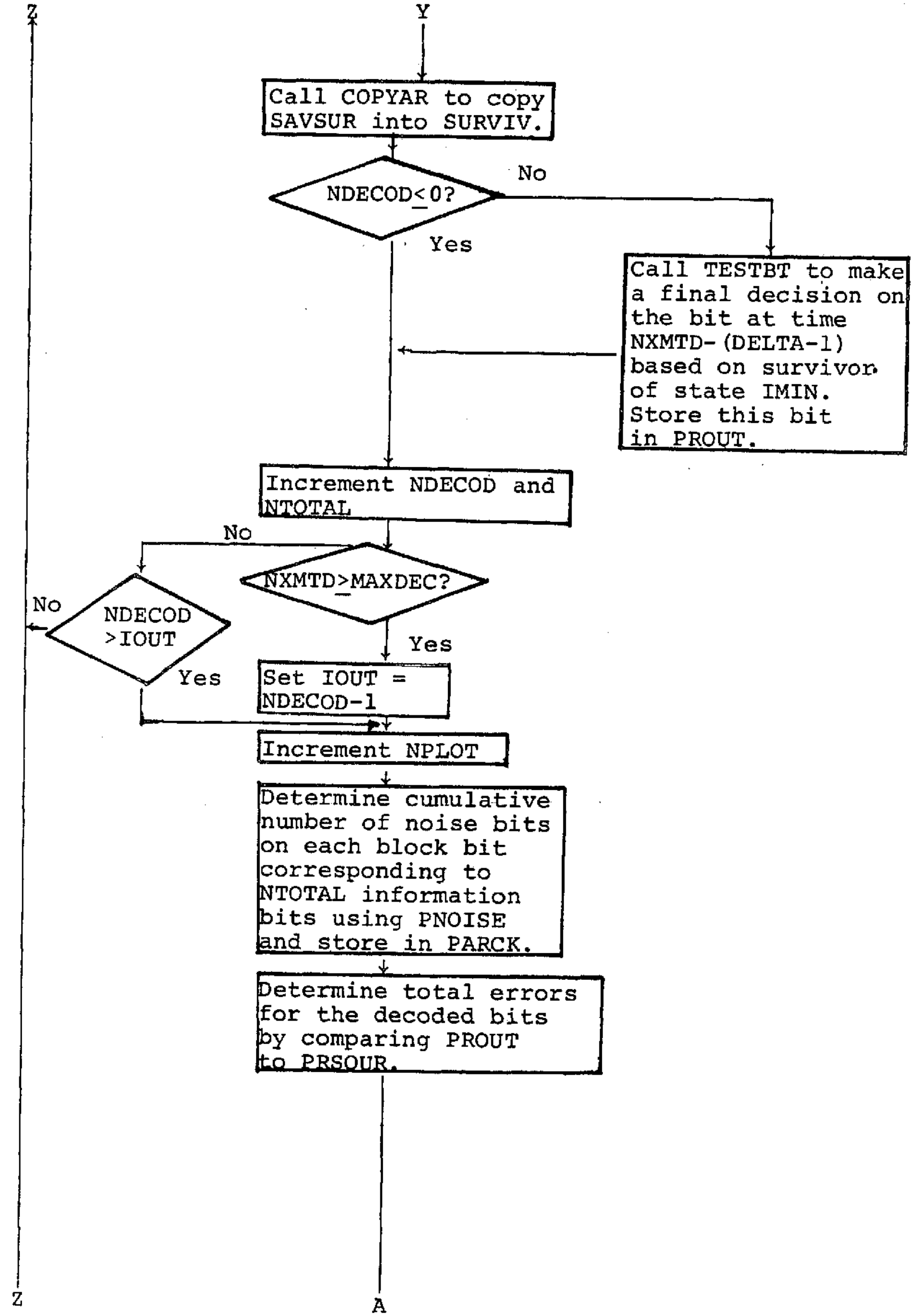
m	delta	bits/sec
5	31	578
5	52	506
6	31	324
6	59	275
7	27	165
7	59	145

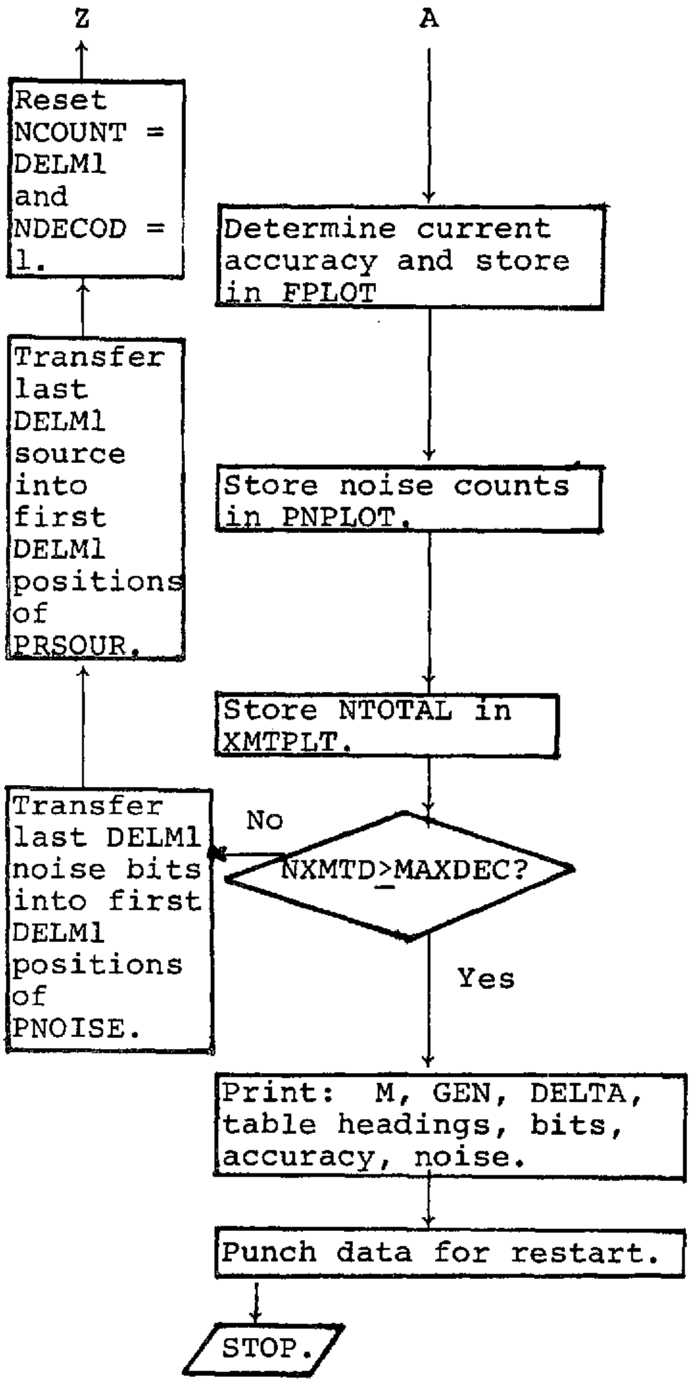
TABLE 2

V. Program for Simulating A Viterbi Decoder


Read: MAXBIT, N, M, NROWS, NCOL, GEN(M), DELTA, STON, NXMTD, IXSOUR, IXNOIS(N). Define: N2, K, K2, DELM1, IDEPTH, IDEPCP, IBIT, MAXDEC, NSTATE, Q.
Determine the two branches for each state: i.e., NEXT(I,1), NEXT(I,2).
Print initial values of IXSOUR and IXNOIS.
NXMTD = 0?
No
Yes
Initialize: ERRORS, NTOTAL, NCOUNT, NDECOD, PARCK(N), SHIFT2(M), GAMMA(NSTATE), SURVIV(NSTATE,NCOL).
Read: ERRORS, NTOTAL, PNOISE(NCOUNT,N), PRSOUR(NCOUNT), PARCK(N), GAMMA(NSTATE) SURVIV(IDEPCP,NCOL), SHIFT2(M).
Define IOUT. Initialize NPLOT. Print STON and Q. Establish P with a lower limit of 1.D-75. Define: QLN, PLN.
Calculate POFZLN. This array contains the predetermined branch lengths based on the probability that a given state is correct for the block of bits received. i.e., POFZLN(I,LOC) contains the probability, expressed as a path length that state I is correct for block number LOC which can be any of 2**N permutations.
X


X
Call random number generator and generate a source bit.
Increment: NXMTD, NCOUNT.
Store source bit in PRSOUR.
Shift source bit into SHIFT2.
Generate parity bits.
Add noise.
Determine the location LOC in POFZLN corresponding to the block of corrupted parity bits.
For each pair of states at time = NXMTD: Determine GAMCHK at time NXMTD-1. Determine and save GAMMA in array SAVE for state at time NXMTD. Save new survivors in SAVSUR at time NXMTD.
Determine the state IMIN which has the shortest overall path length. Determine GAMMAX, the maximum stored overall Path length.
GAMMAX≤10^50
No
Reduce all values of Gamma by 10^50.
Yes
Z
Y


Z
Y
Call COPYAR to copy SAVSUR into SURVIV.
NDECOD≤0?
No
Yes
Call TESTBT to make a final decision on the bit at time NXMTD-(DELTA-1) based on survivor of state IMIN. Store this bit in PROUT.
Increment NDECOD and NTOTAL
No
NXMTD≥MAXDEC?
No
NDECOD >IOUT
Yes
Yes
Set IOUT = NDECOD-1
Increment NPLOT
Determine cumulative number of noise bits on each block bit corresponding to NTOTAL information bits using PNOISE and store in PARCK.
Determine total errors for the decoded bits by comparing PROUT to PRSOUR.
Z
A


Z
A
Reset NCOUNT = DELM1 and NDECOD = 1.
Determine current accuracy and store in FPLOT
Transfer last DELM1 source into first DELM1 positions of PRSOUR.
Store noise counts in PNPLOT.
Store NTOTAL in XMTPLT.
Transfer last DELM1 noise bits into first DELM1 positions of PNOISE.
No
NXMTD≥MAXDEC?
Yes
Print: M, GEN, DELTA, table headings, bits, accuracy, noise.
Punch data for restart.
STOP.

```
C    PROGRAM FOR SIMULATING A VITERBI DECODER
C
C
C    FOR STORAGE REASONS DELTA IN THIS PROGRAM SHOULD BE SET TO ONE MORE
C         THAN THE CASE BEING SIMULATED.
C         I.E.  TO SIMULATE DELTA=13,  SET DELTA=14.
C    THE FOLLOWING MINIMUM ARRAY DIMENSIONS MUST BE ALLOCATED:
C      PNOISE  (IOUT+DELTA,N)
C      PRSOUR  (IOUT+DELTA)
C      PROUT  (IOUT)
C      PNPLOT  (IOUT,N)
C      GEN  (M,N)
C      SHIFT2  (M)
C      L  (M)
C      NEXT  (NSTATE,2)
C      PAR  (N)
C      OUTPUT  (N)
C      XMTPLT  (NUMBER OF LINES PRINTED=(NUMBER OF INFORMATION
C                              BITS DECODED THIS RUN/IOUT))
C      FPLOT  (SAME AS XMTPLT)
C      PARCK  (N)
C      POFZLN  (NSTATE,N2=2**N)
C      GAMMA  (NSTATE)
C      SAVE  (NSTATE)
C      IXNOIS  (N)
C
C
C
C
C    SURVIV AND SAVSUR MUST BE DIMENSIONED EXACTLY  (NROWS,NCOL) WHERE
C       NROWS MUST BE AT LEAST 2**M AND NCOL MUST BE AT LEAST IDEPTH
C
C
C
      INTEGER PARSUM
       INTEGER*2  PNOISE(1064,3),PRSOUR(1064)
      INTEGER GAMCHK,TEST
       INTEGER*2  PROUT(1000)
       INTEGER PNPLOT(1000,3)
       INTEGER*2  GEN(8,3),SHIFT2(8),JKL,     IOUT,M,K2,DELTA,DELTA1
       INTEGER*2  DELM1,NSTATE,L(8),LR,NEXT(256,2),PAR(3),PARITY
       INTEGER*2  OUTPUT(3),LOC,SOURCE
       INTEGER   SURVIV(256,2),SAVSUR(256,2)
       INTEGER   XMTPLT(1000)
       INTEGER     PARCK(3)
       REAL*8            POFZLN(256,4),GAMMA(256),SAVE(256)
       REAL*8   FPLOT(100)
       INTEGER   NTOTAL,ERRORS,NPLOT,NCOUNT,NDECOD,NXMTD,MAXBIT,IXNOIS(3)
       REAL*8   Q,P,DFLOAT,STON,DERF,DSQRT,QLN,PLN
       REAL*8   DLOG,GAMMIN,GAMMAX
C      MNPLOT AND PNPLOT STORE INFO AND PARITY NOISE COUNTS RESPECTIVELY
C     MAXBIT IS THE MAXIMUM NUMBER OF BITS TO BE XMTD
      READ1104,MAXBIT
1104  FORMAT(I7)
C      N IS BLOCK LENGTH
       READ1100,N
      N2=2**N
C   K IS THE NUMBER OF INFORMATION BITS REPRESENTED BY A BLOCK
C      OF LENGTH N.
C   THIS PROGRAM WILL ONLY SIMULATE CODES OF RATE 1/N.
C   THEREFORE K MUST ALWAYS EQUAL 1.
```

```
      K=1
      K2=2**K
C   M IS CONSTRAINT LENGTH
      READ1100,M
1100  FORMAT(I1)
      READ1104,NROWS
      READ1104,NCOL
C   GEN IS SUBGENERATOR VECTOR
      DO 1506 J=1,N
1506  READ1102,(GEN(I,J),I=1,M)
1102  FORMAT(9I1)
      PRINT698
698   FORMAT('1',' ')
C   DELTA=SURVIVOR LENGTH PLUS ONE
      READ1106,DELTA
1106  FORMAT(I3)
C    STON IS THE SIGNAL-TO-NOISE RATIO IN DECIBELS.
      READ1108,STON
1108  FORMAT(D15.7)
      Q=.5D0+.5D0*DERF(DSQRT(DFLOAT(K)/DFLOAT(N)*10.D0**(STON/10.D0)))
C   . NXMTD=TOTAL SOURCE BITS XMTD
      READ2000,NXMTD
2000  FORMAT(I20)
C    IXSOUR AND IXNOIS ARE RANDOM INTEGERS USED AS STARTING
C       POINTS FOR THE RANDOM NUMBER GENERATOR.
      READ2000,IXSOUR
      DO 1510  I=1,N
1510  READ2000,IXNOIS(I)
      DELM1=DELTA-1
C    IDEPTH IS THE NUMBER OF WORDS REQUIRED TO STORE ONE SURVIVOR.
      IDEPTH=DELM1/32+1
      IDEPCP=NCOL+1-IDEPTH
      IBIT=DELTA-32*(IDEPTH-1)
      MAXDEC=MAXBIT+DELM1
      NSTATE=2**M
      M1=M+1
      DO 9  I=1,M
9     L(I)=1
      DO 16  I=1,NSTATE
      DO 11  LR=1,M
      IF(L(LR))11,14,11
11    L(LR)=0
      GO TO 15
14    L(LR)=1
C    NEXT IS AN ARRAY CONTAINING THE NUMBERS CORRESPONDING
C       TO THE 2 STATES THAT ANY GIVEN STATE CAN BRANCH TO.
15    NEXT(I,1)=1
      DO 30  LM=2,M
30    NEXT(I,1)=NEXT(I,1)+2**(LM-2)*L(LM)
16    NEXT(I,2)=NEXT(I,1)+2**(M-1)
      PRINT1138,IXSOUR
1138  FORMAT('0','IXSOUR=',I20)
      DO 1139  I=1,N
1139  PRINT1140,I,IXNOIS(I)
1140  FORMAT('0','IXNOIS(',I1,')=',I20)
      IF(NXMTD.EQ.0)GO TO 1300
      READ2000,ERRORS
      READ2000,NTOTAL
      NCOUNT=DELM1
      NDECOD=1
2001  FORMAT(72I1)
```

```
      DO 1550  J=1,N
 1550  READ2001,(PNOISE(I,J),I=1,NCOUNT)
       READ2001,(PRSDUR(I),I=1,NCOUNT)
      DO 1551 I=1,N
1551  READ2000,PARCK(I)
       DO 1302  I=1,NSTATE
1302   READ2002,GAMMA(I)
2002   FORMAT(D23.16)
       DO 1303  I=1,NSTATE
      DO 1303  J=IDEPCP,NCOL
1303   READ2000,SURVIV(I,J)
       READ2001,(SHIFT2(I),I=1,M)
       GO TO 1301
1300    CONTINUE
C   INITIALIZE RANDU SOURCE AND NOISE
C   ERRORS=NUMBER OF DECODING ERRORS
       ERRORS=0
C   NTOTAL=TOTAL BITS PRINTED
       NTOTAL=1-DELTA
C   NCOUNT PLACES OUTPUT BITS IN CORRECT VECTOR POSITION
       NCOUNT=0
C   NDECOD=NUMBER OF BITS DECODED
       NDECOD=2-DELTA
C   PARCK=TOTAL NUMBER OF PRINTED PARITY CHECK BITS CORRUPTED BY NOISE
      DO 1552 I=1,N
1552  PARCK(I)=0
C   SHIFT2=CONTENTS OF ENCODING SHIFT REGISTER
       DO 1  I=1,M
1      SHIFT2(I)=0
       GAMMA(1)=0.D0
       DO 60  I=2,NSTATE
60     GAMMA(I)=1.D40
       DO 61  I=1,NROWS
      DO 61  J=1,NCOL
61     SURVIV(I,J)=0
1301   CONTINUE
C   IOUT=NUMBER OF BITS PRINTED PER LINE
       IOUT=1000
C   NPLOT COUNTS NUMBER OF TIMES PRINTING ALGORITHM IS USED
       NPLOT=0
       PRINT998,STON
998        FORMAT('-','S/N =',D15.7)
       PRINT704,Q
704        FORMAT('+',T30,'Q=',D15.7)
       P=1.D0-Q
       IF(P.LT.1.D-75)P=1.D-75
      QLN=-1.D0*DLOG(Q)
      PLN=-1.D0*DLOG(P)
       DO 49  I=1,M
49     L(I)=1
      DO 50  I=1,NSTATE
      DO 51  LR=1,M
      IF(L(LR))51,52,51
51    L(LR)=0
      GO TO 53
52    L(LR)=1
53    DO 1500  IBITNR=1,N
      PARSUM=0
      DO 20  I1=1,M
      IM1=M1-I1
20    PARSUM=PARSUM+GEN(I1,IBITNR)*L(IM1)
```

```
1500   PAR(IBITNR)=PARSUM-PARSUM/2*2
       DO 21   I3=1,N
21     OUTPUT(I3)=1
       DO 50   J=1,N2
       DO 55   I4=1,N
       I4MIN1=N+1-I4
       IF(OUTPUT(I4MIN1))55,56,55
55     OUTPUT(I4MIN1)=0
       GO TO 57
56     OUTPUT(I4MIN1)=1
57     POFZLN(I,J)=0.D0
       DO 50   IL77=1,N
       IF(OUTPUT(IL77).EQ.PAR(IL77))POFZLN(I,J)=POFZLN(I,J)+QLN
50     IF(OUTPUT(IL77).NE.PAR(IL77))POFZLN(I,J)=POFZLN(I,J)+PLN
699     CALL RANDU(IXSOUR,IYSOUR,YSOUR)
        IXSOUR=IYSOUR
       SOURCE=0
       IF(YSOUR-0.5)1511,1512,1512
1512   SOURCE=1
 1511   NXMTD=NXMTD+1
        NCOUNT=NCOUNT+1
        PRSOUR(NCOUNT)=SOURCE
        DO 2   I=2,M
        I1=M+2-I
2       SHIFT2(I1)=SHIFT2(I1-1)
        SHIFT2(1)=SOURCE
C  PARITY=PARITY CHECK DIGIT
       DO 1516   IBITNR=1,N
       PARSUM=0
       DO 1517   I1=1,M
1517   PARSUM=PARSUM+GEN(I1,IBITNR)*SHIFT2(I1)
1516   PAR(IBITNR)=PARSUM-PARSUM/2*2
       DO 1530   I=1,N
       PNOISE(NCOUNT,I)=0
       CALL RANDU(IXNOIS(I),IYNOIS,YNOIS)
       IXNOIS(I)=IYNOIS
       IF(Q-YNOIS)1531,1531,1530
1531   PNOISE(NCOUNT,I)=1
       KPAR=PAR(I)
       IF(KPAR.EQ.0)PAR(I)=1
       IF(KPAR.EQ.1)PAR(I)=0
1530   CONTINUE
C  LOC= LOCATION IN PROBABILITY MATRIX CORRESPONDING TO BLOCK RECEIVED
       LOC=1
       DO 1504   LOCSUM=1,N
1504   LOC=LOC+2**(N-LOCSUM)*PAR(LOCSUM)
C
C
C
C   STATES ARE NUMBERED SUCH THAT 1 AND 2, 3 AND 4, ETC. ARE
C       PAIRS THAT BRANCH TO THE SAME STATES.
C       I.E.  0X AND 1X BOTH BRANCH TO X0 AND X1, WHERE X REPRESENTS
C               A PERMUTATION OF M-1 BITS.
C
C
C   GAMCHK IS THE STATE OF GIVEN PAIR OF STATES WHICH HAS THE SHORTEST
C       TOTAL PATH LENGTH GAMMA.
C
C
C
        DO 100   I=1,NSTATE,2
```

```
      GAMCHK=I
      IF(GAMMA(I).GT.GAMMA(I+1))GAMCHK=I+1
      DO 101  J=1,K2
101   SAVE(NEXT(I,J))=GAMMA(GAMCHK)+POFZLN(NEXT(I,J),LOC)
     IROWO=NEXT(I,1)
     IROW1=NEXT(I,2)
     CALL SHIFT(SURVIV,SAVSUR,NROWS,NCOL,GAMCHK,IROWO,IROW1)
100   CONTINUE
      DO 139  I=1,NSTATE
139   GAMMA(I)=SAVE(I)
142   GAMMIN=1.D70
      GAMMAX=-1.D70
      IMIN=1
      DO 140  I=1,NSTATE
      IF(GAMMA(I).GT.GAMMAX)GAMMAX=GAMMA(I)
      IF(GAMMA(I).GE.GAMMIN)GO TO 140
      GAMMIN=GAMMA(I)
      IMIN=I
140   CONTINUE
155   IF(GAMMAX.LE.1.D50)GO TO 150
      DO 141  I=1,NSTATE
141   GAMMA(I)=GAMMA(I)-1.D50
      GO TO 142
150   CALL COPYAR(SAVSUR,SURVIV,NROWS,NCOL)
      IF(NDECOD.LE.0)GO TO 715
     CALL TESTBT(SURVIV(IMIN,IDEPCP),IBIT,TEST)
     PROUT(NDECOD)=TEST
715   NDECOD=NDECOD+1
      NTOTAL=NTOTAL+1
      IF(NXMTD.GE.MAXDEC)GO TO 750
      IF(NDECOD.GT.IOUT)GO TO 700
      GO TO 699
700   CONTINUE
      NPLOT=NPLOT+1
     DO 776  J=1,N
     DO 776  I=1,IOUT
776  IF(PNOISE(I,J).EQ.1)PARCK(J)=PARCK(J)+1
      DO 770  I=1,IOUT
770   IF(PRSOUR(I).NE.PROUT(I))ERRORS=ERRORS+1
          FPLOT(NPLOT)=DFLOAT(NTOTAL-ERRORS)/DFLOAT(NTOTAL)
     DO 1561  I=1,N
1561 PNPLOT(NPLOT,I)=PARCK(I)
      XMTPLT(NPLOT)=NTOTAL
      IF(NXMTD.GE.MAXDEC)GO TO 900
     DO 778  J=1,N
     DO 778  I=1,DELM1
778  PNOISE(I,J)=PNOISE(IOUT+I,J)
      DO 703  I=1,DELM1
703   PRSOUR(I)=PRSOUR(IOUT+I)
      NCOUNT=DELM1
      NDECOD=1
      GO TO 699
750   IOUT=NDECOD-1
      GO TO 700
900   CONTINUE
      PRINT500,M
500       FORMAT('-','CONSTRAINT LENGTH=',I3)
     DO 1565  J=1,N
 1565 PRINT501,(GEN(I,J),I=1,M)
501  FORMAT('0','SUBGENERATOR=',10I6)
      PRINT1001,DELTA
```

```
1001        FORMAT('-','DELTA=',I3)
      PRINT602
602   FORMAT('0',T2,'BITS',T19,'ACCURACY',T37,'NOISE COUNT:  BIT1,BIT2,
     1BIT3, ETC.')
      DO 600  J=1,NPLOT
600   PRINT604,XMTPLT(J),FPLOT(J),(PNPLOT(J,I),I=1,N)
604   FORMAT(' ',T2,I7,T15,D16.8,T39,8I9)
      PUNCH1105,MAXBIT
1105  FORMAT(I7,T73,'MAXBIT')
      PUNCH1135,N
1135  FORMAT(I1,T73,'N')
      PUNCH1101,M
1101  FORMAT(I1,T73,'M')
      PUNCH1136,NROWS
1136  FORMAT(I7,T73,'NROWS')
      PUNCH1137,NCOL
1137  FORMAT(I7,T73,'NCOL')
      DO 1566  J=1,N
1566  PUNCH1103,J,(GEN(I,J),I=1,M)
1103  FORMAT(T73,'GENER(',I1,')',T1,9I1)
      PUNCH1107,DELTA
1107  FORMAT(I3,T73,'DELTA')
      PUNCH1109,STON
1109  FORMAT(D15.7,T73,'S/N')
      PUNCH3000,NXMTD
3000   FORMAT(I20,T73,'NXMTD')
      PUNCH3100,IXSOUR
3100  FORMAT(I20,T73,'IXSOUR')
      DO 1567  J=1,N
1567  PUNCH3200,IXNOIS(J),J
3200  FORMAT(I20,T71,'IXNOIS(',I1,')')
      PUNCH3300,ERRORS
3300  FORMAT(I20,T73,'ERRORS')
      PUNCH3400,NTOTAL
3400  FORMAT(I20,T73,'NTOTAL')
      DO 1569  J=1,N
1569  PUNCH3101,J,(PNOISE(IOUT+I,J),I=1,DELM1)
3101  FORMAT(T73,'PNOISE',I1,T1,72I1)
      PUNCH3201,(PRSOUR(IOUT+I),I=1,DELM1)
3201  FORMAT(T73,'PRSOUR',T1,72I1)
      DO 1570  J=1,N
1570  PUNCH3500,PARCK(J),J
3500  FORMAT(I20,T73,'PARCK(',I1,')')
      DO 4000  I=1,NSTATE
4000   PUNCH3002,GAMMA(I),I
3002  FORMAT(D23.16,T73,'I=',I4)
      DO 4001  I=1,NSTATE
      DO 4001  J=IDEPCP,NCOL
4001  PUNCH3301,I,J,SURVIV(I,J)
3301  FORMAT(T70,'I=',I4,' J=',I2,T1,I20)
      PUNCH3401,(SHIFT2(I),I=1,M)
3401  FORMAT(T73,'SHIFT',T1,72I1)
      STOP
      END
```

```
TESTBT   PRCGRAM
***TESTBIT***CHECKS VALUE CF ANY BIT IN A FLLL WCRD IN MAIN STCRAGE.
*        R1--ADDR OF ARGUMENT LIST
*            0(1) ACDR OF WORD
*            4(1) ACCR CF TESTBIT
*            8(1) ACDR FOR RETURN CCDE
*
          L      2,C(1)             LCAD ACCR CF WCRC TC BE SHIFTED
          L      4,4(1)             LCAC ACCR OF TESTBIT
          L      C,C(4)             LCAC TESTBIT
         LA     3,0                 SET FCR CCMPARE
         L      4,C(2)              LCAC TEST WCRC
         LA     5,32                LCAC 32 FCR SUBTRACTICN
         SR     5,0                 FINC CCMPLEMENT
          SLL    4,C(5)             SHIFT SC TEST BIT IS IN THE SIGN PCSITICN
         CR     4,3                 CCMPARE RESULTS AGAINST ZERC
         BNL    NCTNEG
         LA     3,1                 LCAC ONE IF NEGATIVE
NCTNEG   ECU    *
          L      5,8(1)             LCAC ACCR CF RETURN WCRC
          ST     3,0(5)             STCRE BIT IN RETURN WCRC
          STCP
          ENC
```

```
COPYAR   PROGRAM
***COPYARAY***COPIES AN ARRAY THROUGH THE USE OF THE MVCL INSTRUCTION.
*   INPUT:
*          R1--ADDR OF ARGUMENT LIST
*              0(1)  ADDR OF INPUT ARRAY
*              4(1)  ADDR OF OUTPUT ARRAY
*              8(1)  ADDR OF NUMBER OF ROWS IN THE ARRAYS
*              12(1)  ADDR OF NUMBER OF COLUMNS IN THE ARRAYS
*
COPYARAY EQU   *
         L     2,4(1)           ADDR OF ARRAYO
         L     4,0(1)           ADDR OF ARRAYI
         L     6,8(1)           LOAD ADDR OF NUMBER OF ROWS
         L     5,12(1)          LOAD ADDR OF NUMBER OF COLUMNS
         L     5,0(5)           LOAD NUMBER OF COLUMNS
         MH    5,2(6)           MULTIPLY NUMBER OF ROWS BY COLUMNS
         SLA   5,2              (ROWS * COLUMNS) * 4
         LR    3,5              COPY LENGTH IN R3 FOR MVCL
         MVCL  2,4              COPY ARRAY
         STOP
         END
```

```
SHIFT    PROGRAM
***COPYROWS***CONTAINS THE LOGIC NECESSARY TO COPY AND SHIFT
*  THE SPECIFIC ROWS AS SPECIFIED.  COPYROWS WILL COPY ROWI
*  INTO ROWO AND ROW1, SHIFT THESE ROWS TO THE LEFT ONE BIT,
*  AND SET THE RIGHTMOST BIT OF ROWO AND ROW1 TO ZERO AND ONE
*  RESPECTIVELY.
*  INPUT:
*           R1--ADDR OF ARGUMENT LIST
*               0(1)  ADDR OF INPUT ARRAY (ARRAYI)
*               4(1)  ADDR OF OUTPUT ARRAY (ARRAYO)
*               8(1)  ADDR OF NUMBER OF ROWS
*              12(1)  ADDR OF NUMBER OF COLUMNS
*              16(1)  ADDR OF ROWI
*              20(1)  ADDR OF ROWO
*              24(1)  ADDR OF ROW1
*
COPYROWS EQU   *
         LA    7,0                ZERO OUT R7
         L     12,8(1)            LOAD ADDR OF NUMBER OF ROWS
         L     12,0(12)           LOAD NUMBER OF ROWS
         SLA   12,2               MULTIPLY BY FOUR FOR DISPLACEMENT
         L     6,0(1)             LOAD ADDR OF ARRAYI
         L     8,12(1)            LOAD ADDR OF NUMBER OF COLUMNS
         L     8,0(8)             LOAD NUMBER OF COLUMNS
         L     2,4(1)             LOAD ADDR OF ARRAYO
         LR    3,8                COPY NUMBER OF COLUMNS FROM R8
         BCTR  3,0                 SUBTRACT ONE FROM NUMBER OF COLUMNS
         L     4,8(1)              LOAD ADDR OF NUMBER OF ROWS
         MH    3,2(4)              COMPUTE ROWS*(COLUMNS-1)
         LR    9,3                 ***
         LR    10,3                *  *  COPY ROWS*(COLUMNS-1)
         LR    11,3                ***
         L     3,16(1)             ***
         L     4,20(1)             *  *  LOAD ADDRS OF ROWI,ROWO,AND ROW1
         L     5,24(1)             ***
         L     3,0(3)              ***
         L     4,0(4)              *  *  LOAD ROWI,ROWO,AND ROW1
         L     5,0(5)              ***
         BCTR  3,0                 ***
         BCTR  4,0                 *  *  DECREMENT EACH BY ONE
         BCTR  5,0                 ***
         AR    9,3                 ***
         AR    10,4                *  *  ADD PRODUCT AND ROWS MINUS ONE
         AR    11,5                ***
         SLA   9,2                 ***
         SLA   10,2                *  *  MULTIPLY BY 4 TO OBTAIN DISPLACMENT
         SLA   11,2                ***
         AR    9,6                 ***
         AR    10,2                *  *  COMPUTE ACTUAL ADDRESSES
         AR    11,2                ***
         MVC   0(4,10),0(9)        COPY LAST WORD OF ROWI TO ROWO
         LA    0,0                 INPUT FOR ROWO
         LR    2,10                INPUT FOR SLIDE
         BAL   14,SLIDE            BRANCH TO SLIDE
         MVC   0(4,11),0(9)        COPY LAST WORD OF ROWI TO ROW1
         LA    0,1                 INPUT FOR ROW1
         LR    2,11                INPUT FOR SLIDE
         BAL   14,SLIDE            BRANCH TO SLIDE
         BCTR  8,0                 DECREMENT COLUMN COUNT BY ONE
         CR    8,7                 COMPARE COLUMN COUNT AGAINST ZERO
         BE    THRU                IF EQUAL END
```

PART C

```
DECADDR  EQU    *
         SR     9,12             ***
         SR     10,12            *  *  DECREMENT ADDRS BY ROW MEMBER DIST
         SR     11,12            ***
         MVC    0(4,10),0(9)     COPY NEXT ROW1 MEMBER TO ROW0
         LR     2,10             SET UP FOR SLIDE--CARRY BIT ALREADY SET
         BAL    14,SLIDE         BRANCH TO SLIDE
         MVC    0(4,11),0(10)    COPY ROW0 MEMBER INTO ROW1 MEMBER
         BCT    8,DECADDR        DECREMENT AND TEST COLUMN COUNT
THRU     EQU    *
         STOP
ARGLSTAD DS     F
***SLIDE***SHIFTS ONE FULL WORD IN MAIN STORAGE TO THE LEFT ONE BIT
*  SETTING THE RIGHTMOST BIT IN THE WORD AS INDICATED BY R0.  THE
*  ADDR OF THE WORD TO BE SHIFTED IS CONTAINED IN R2 AND THE CARRY
*  OVER BIT IS PLACED IN R0.
*
SLIDE    EQU    *
         STM    3,4,SAVEREGS     SAVE WORKING REGISTERS
         LA     4,0              LOAD ZERO FOR COMPARE
         L      3,0(2)           COPY WORD TO BE SHIFTED
         CR     3,4              CHECK FOR CARRY
         SLL    3,1              SHIFT OVER ONE BIT
         BNL    CARRY0           BRANCH ACCORDINGLY
CARRY1   EQU    *
         LA     4,1             WORD NEGATIVE, THEREFORE SET CARRY BIT
CARRY0   AR     3,0              SHIFT IN BIT BY ADDITION
         LR     0,4              STORE CARRY VALUE
         ST     3,0(2)           REPLACE SHIFTED WORD
         LM     3,4,SAVEREGS     RESTORE WORKING REGISTERS
         BR     14               RETURN TO CALL--BAL  14,SLIDE
SAVEREGS DS     2F
         END
```

SHORT CONSTRAINT LENGTH RATE 1/2 'QUICK-LOOK' CODES

I. Introduction:

A binary convolutional code of constraint length K and rate $R = \frac{1}{2}$ is completely specified by a set of two generators which in transform notation have the form

$$G^{(j)}(D) = g_0^{(j)} + g_1^{(j)} D + g_2^{(j)} D^2 + \ldots + g_{K-1}^{(j)} D^{K-1} (j = 1, 2)$$

with coefficients from GF(2). (Throughout we assume the codes are nondegenerate, i.e., at least one of $g_0^{(1)}$ and $g_0^{(2)}$ are at least one of $g_{K-1}^{(1)}$ and $g_{K-1}^{(2)}$ are one). If

$$I(D) = i_0 + i_1 D + i_2 D^2 + \ldots$$

is a sequence of binary information digits, then the result of applying I(D) to the encoder is

$$T^{(j)}(D) = I(D)\, G^{(j)}(D) = t_0^{(j)} + t_1^{(j)} D + t_2^{(j)} D^2 + \ldots (j = 1, 2)$$

so that for each information digit i_k the encoder produces a block of two digits $[t_k^{(1)}, t_k^{(2)}]$ that are functions of i_k and the previous K-1 information digits. The linear sequential circuit that performs this operation consists of a shift register whose K stages are connected to two modulo-2 adders in accordance with the coefficients of $G^{(1)}(D)$ and $G^{(2)}(D)$, respectively. The outputs of the adders at time k then constitute the block $[t_k^{(1)}, t_k^{(2)}]$. For convenience we denote the sequence of these blocks by T(D).

In certain situations such as system check-out it is desirable to be able to recover the information sequence from the encoded sequence. Massey and Sain (1968) have shown that this is possible if and only if the code is noncatastrophic, i.e., if and only if

$$\gcd [G^{(1)}(D), G^{(2)}(D)] = D^{\ell}$$

for some $\ell \geq 0$. In this case, there always exists a linear sequential circuit that produces $I(D)$ with a delay of exactly L digits for any integer $L \geq \ell$ and it is completely described by two generator polynomials $P^{(1)}(D)$ and $P^{(2)}(D)$ that satisfy

$$P^{(1)}(D)\, G^{(1)}(D) + P^{(2)}(D)\, G^{(2)}(D) = D^{\ell}$$

To illustrate these ideas we consider the code

$$G^{(1)}(D) = 1 + D + D^2 + D^3 + D^6$$

$$G^{(2)}(D) = 1 + D^2 + D^3 + D^5 + D^6$$

This code has a constraint length K = 7 and its circuit realization is shown in Figure 51. If the input sequence is

1 0 0 1 1 0 1 0 1 . . .

then $T^{(1)}(D)$ and $T^{(2)}(D)$ are given by

1 1 1 0 0 0 0 0 0 . . .

and

1 0 1 0 1 0 0 1 1 . . .

respectively, and the encoder output sequence will be

1 1 1 0 1 1 0 0 0 1 0 0 0 0 0 1 0 1 . . .

Since $G^{(1)}(D)$ and $G^{(2)}(D)$ are relatively prime, an inverse circuit with delay zero exists and we may easily prove that $P^{(1)}(D)$ and $P^{(2)}(D)$ are given by

$$P^{(1)}(D) = 1 + D + D^2 + D^3 + D^4$$

$$P^{(2)}(D) = D^2 + D^4$$

Two versions of the circuit realization are shown in Figure 52.

Suppose now that the encoder output sequence T(D) is transmitted over a noisy channel prior to its inversion. Then, of course, the resulting sequence $\hat{I}(D)$ will generally not be a perfect match of the original information sequence

$\hat{I}(D)$. In fact, Massey and Costello (1971) have shown that over the binary symmetric channel and at high signal-to-noise ratios the probability of an error in $\hat{I}(D)$ is related to the probability of error in the channel by

$$p_{\hat{I}} = Ap_{BSC}$$

where A is the error amplification factor given by

$$A = W[P^{(1)}(D)] + W[P^{(2)}(D)]$$

and $W[P^{(i)}(D)]$ denotes the Hamming weight of $P^{(i)}(D)$.

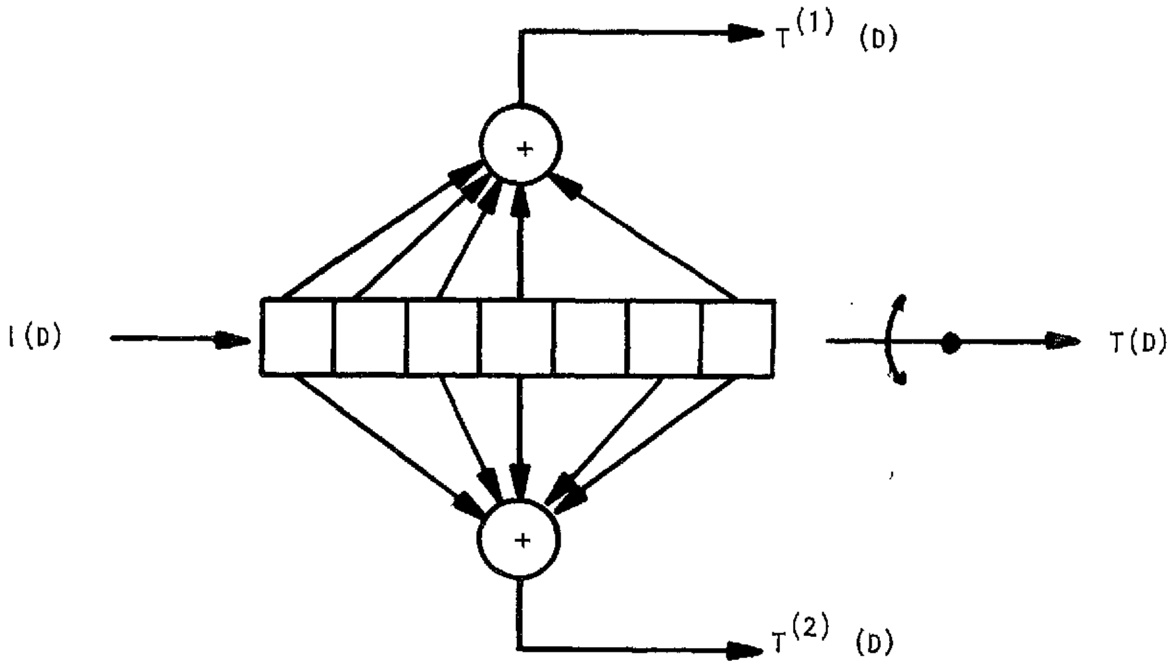


FIGURE 51. ENCODING CIRCUIT FOR THE CODE

1 1 1 1 0 0 1

1 0 1 1 0 1 1

In our example above A has the value 7, so that an error in $\hat{I}(D)$ is seven times more likely than an error in the channel. This is quite obvious from Figure 52b. For a single error in the channel will, as it propogates through

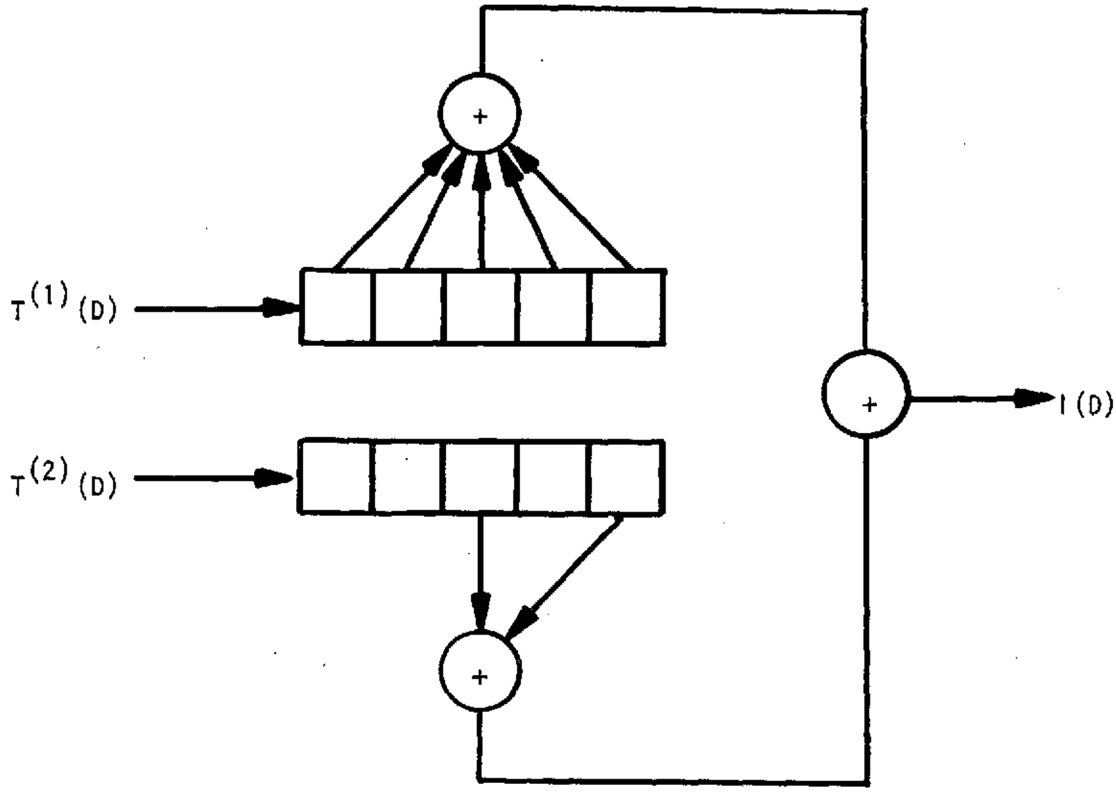


FIGURE 52a. INVERSE CIRCUIT FOR THE CODE

1 1 1 1 0 0 1

1 0 1 1 0 1 1

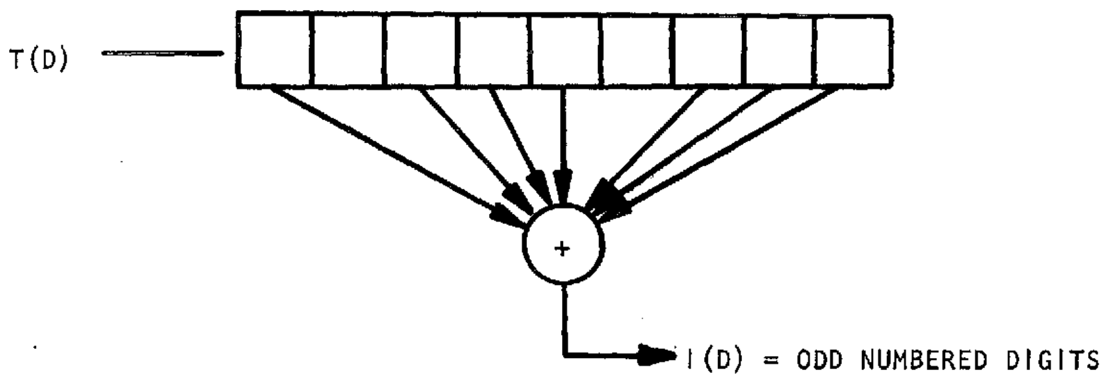


FIGURE 52b. ALTERNATE INVERSE CIRCUIT FOR THE CODE

1 1 1 1 0 0 1

1 0 1 1 0 1 1

the circuit, produce 7 errors in the output of the adder, assuming that the channel errors are spaced far enough apart.

For low signal-to-noise ratios the simple reasoning leading to (1) no longer applies and the value of the error amplification must be determined empirically. Figure 53 shows the result for the code in the above example.

Consider next the system configuration of Figure 54.

At high signal-to-noise ratios a well designed decoder will be able to correct the overwhelming majority of the errors introduced in the channel and deliver an essentially perfect copy of $I(D)$. If we then compare this output with that of the encoder inverse we obtain an indication of the signal-to-noise ratio in the channel.

With the binary symmetric channel, for example, we can get a good estimate of $P_{\hat{I}}$ by computing the ratio of the number of ones in which the outputs of the decoder and the encoder inverse differ to the total number of digits processed. Using (1) we are then able to determine the value of P_{BSC}.

The surprising fact is that this scheme also works for low signal-to-noise ratios, where the decoder output also includes errors, and produces a one-to-one relationship between p_{BSC} and the measured quantity, which we denote by $\hat{p}_{BSC}$.

Figure 55 shows the simulation results for the code in our previous example, the binary symmetric channel and a 32 bit path length Viterbi decoder.

From Figure 53 it is clear that if one attempts to reconstruct the original information sequence at the channel output without benefit of decoding, it is desirable to have a code with as low a value of error amplification as possible. The best in this regard are the socalled systematic codes for which one of the $P^{(i)}(D)$ is one and the other equals zero, resulting in $A = 1$. Unfortunately, the error correcting capability of these codes is markedly inferior to that of certain nonsystematic codes when used in conjunction with sequential or maximum likelihood decoding algorithms.

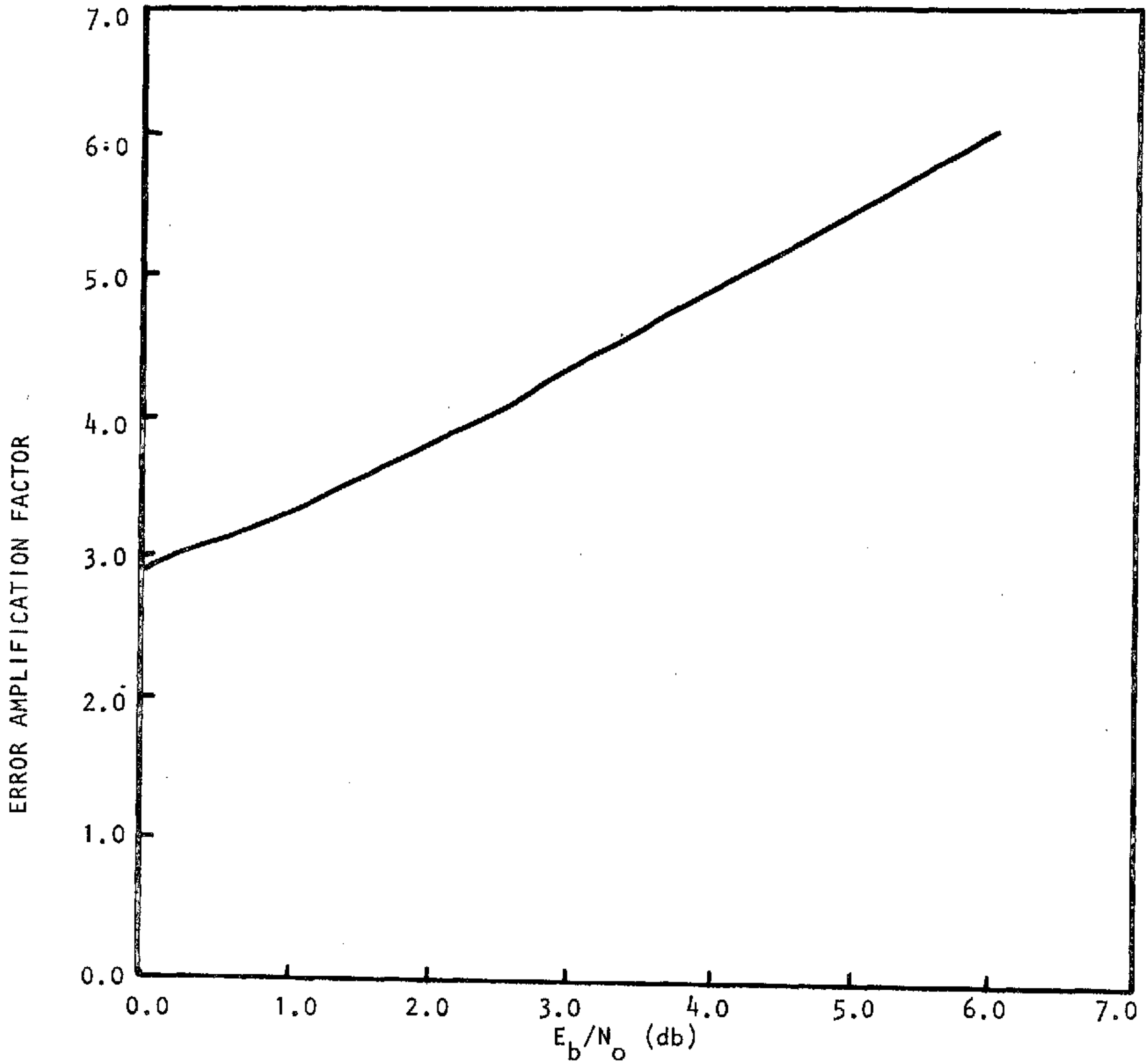


FIGURE 53. ERROR AMPLIFICATION FACTOR FOR THE CODE

1 1 1 1 0 0 1

1 0 1 1 0 1 1

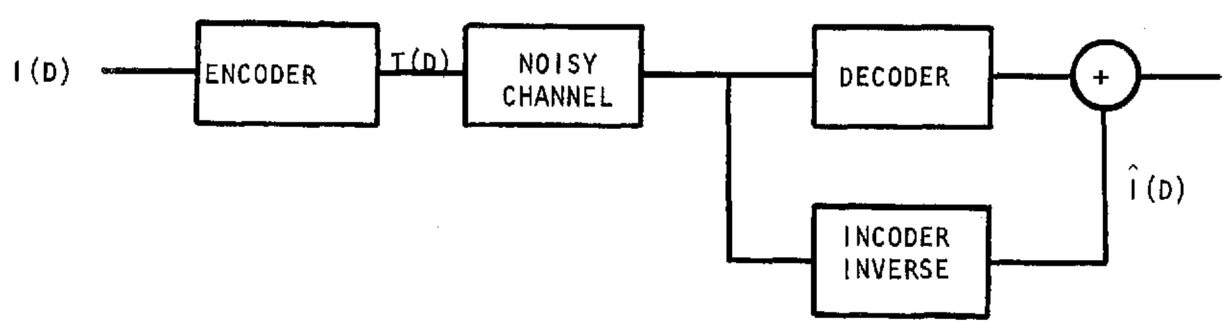


FIGURE 54. CHANNEL NOISE MEASURING SYSTEM

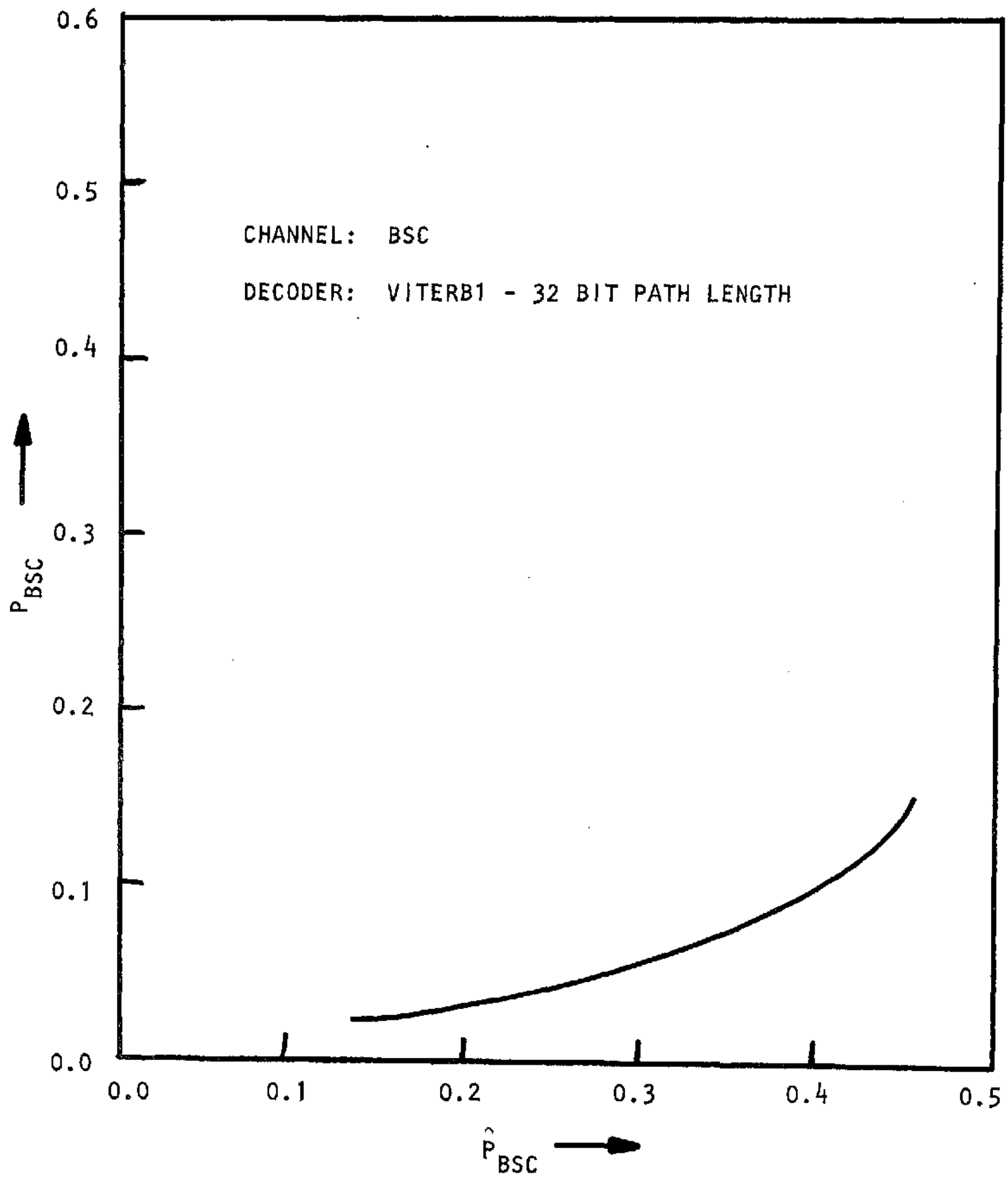


FIGURE 55. MEASURED VERSUS ACTUAL CHANNEL BIT ERROR PROBABILITY FOR CODE

1 1 1 1 0 0 1

1 0 1 1 0 1 1

For nonsystematic codes the lowest possible value of A is 2 and is attained by the socalled quick-look codes [2]. Our purpose in this note is to investigate their relevant characteristics and in the process we obtain a number of interesting and practically useful results. Since our primary motivation is the application of quick-look codes to Viterbi decoding, we restrict consideration to constraint lengths less than eight.

II. Quick-Look Codes:

We define a rate ½ quick-look code as any code in which the two generators differ in exactly one coefficient. Then

$$G^{(1)}(D) + G^{(2)}(D) = D^L$$

for some $0 < L < K - 1$ and an inverse circuit with delay L and error amplification factor A = 2 is given by

$$P^{(1)}(D) = P^{(2)}(D) = 1$$

This, of course, amounts to nothing more than the modulo-2 addition of $T^{(1)}(D)$ and $T^{(2)}(D)$. Hence the word 'Quick-Look' [2].

Since we are dealing with nondegenerate codes only, it follows easily that all quick-look codes have

$$\gcd\left[G^{(1)}(D), G^{(2)}(D)\right] = 1$$

Thus, there always exists an inverse with delay zero, which is generally different from the quick-look inverse if L > 0.

For example, when L = 1, the zero delay inverse takes the form

$$P^{(i)}(D) = \frac{1 + G^{(j)}(D)}{D} \qquad (i \neq j)$$

and its error amplification factor at high signal-to-noise ratios is

$$A = W[G^{(1)}(D)] + W[G^{(2)}(D)] - 2$$

For L = 2 the zero delay inverse becomes

$$P^{(i)}(D) = \frac{1 + (1 + a_1 D)\, G^{(j)}(D)}{D^2} \qquad i \neq j$$

Here A has the same value as above if $a_1 = 0$ and is a function of the coefficients of $G^{(1)}(D)$ and $G^{(2)}(D)$ if $a_1 = 1$.

As a concrete example, consider the constraint length 5 code

$$G^{(1)}(D) = 1 + D + D^2 + D^4$$
$$G^{(2)}(D) = 1 + D + D^4$$

Clearly, L = 2 and the quick-look inverse circuit takes either of the forms in Figure 56.

The inverse circuit with zero delay is given by

$$P^{(1)}(D) = 1 + D^2 + D^3$$
$$P^{(2)}(D) = D + D^2 + D^3$$

and Figure 57 shows the two alternate configurations for this case. Note that the error amplification factor increases from 2 to 6 over the quick-look inverse.

III. Maximum Free Distance Quick-Look Codes:

One commonly accepted measure of the performance of a convolutional code in conjunction with sequential or maximum likelihood decoding algorithms is free distance. For the codes under consideration here this is simply the smallest nonzero number of ones in the set of semi-infinite output sequences of the encoder.

Our objective is to find quick-look codes of constraint lengths $3 \leq K \leq 7$, with as large a free distance as possible.

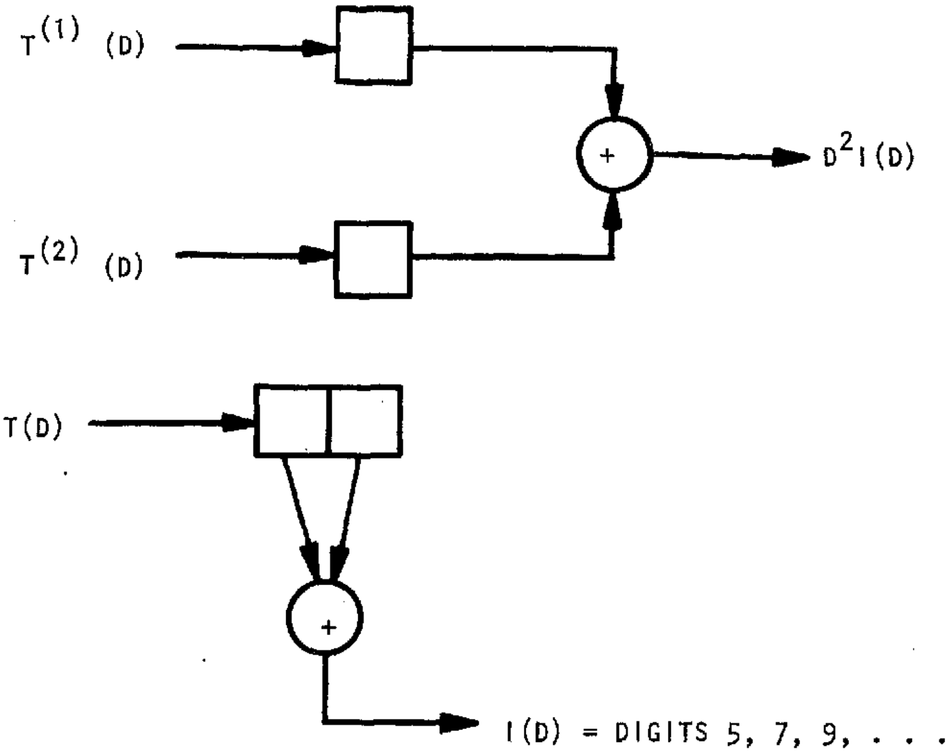


FIGURE 56. QUICK-LOOK INVERSE CIRCUITS FOR THE CODE

1 1 1 0 1

1 1 0 0 1

To narrow the search for such codes we first note that the maximum free distance of any rate ½ noncatastrophic convolutional code is bound by

$$d_f \le \begin{cases} K + 2; & 3 \le K \le 6 \\ K + 3; & K = 7 \end{cases}$$

and that there always exists a code for which equality holds (Larsen, 1973).

Second, since the input sequence 100 . . . produces as output sequence from each modulo-2 adder of the encoder the coefficients of the respective generator polynomial, the free distance of any code is evidently bounded by

$$d_f \le W[G^{(1)}(D)] + W[G^{(2)}(D)]$$

Finally, if $G*(D)$ denotes the reciprocal polynomial of $G(D)$, then the codes

$$G^{(1)}(D),\ G^{(2)}(D)$$

and

$$G^{(1)}*(D),\ G^{(2)}*(D)$$

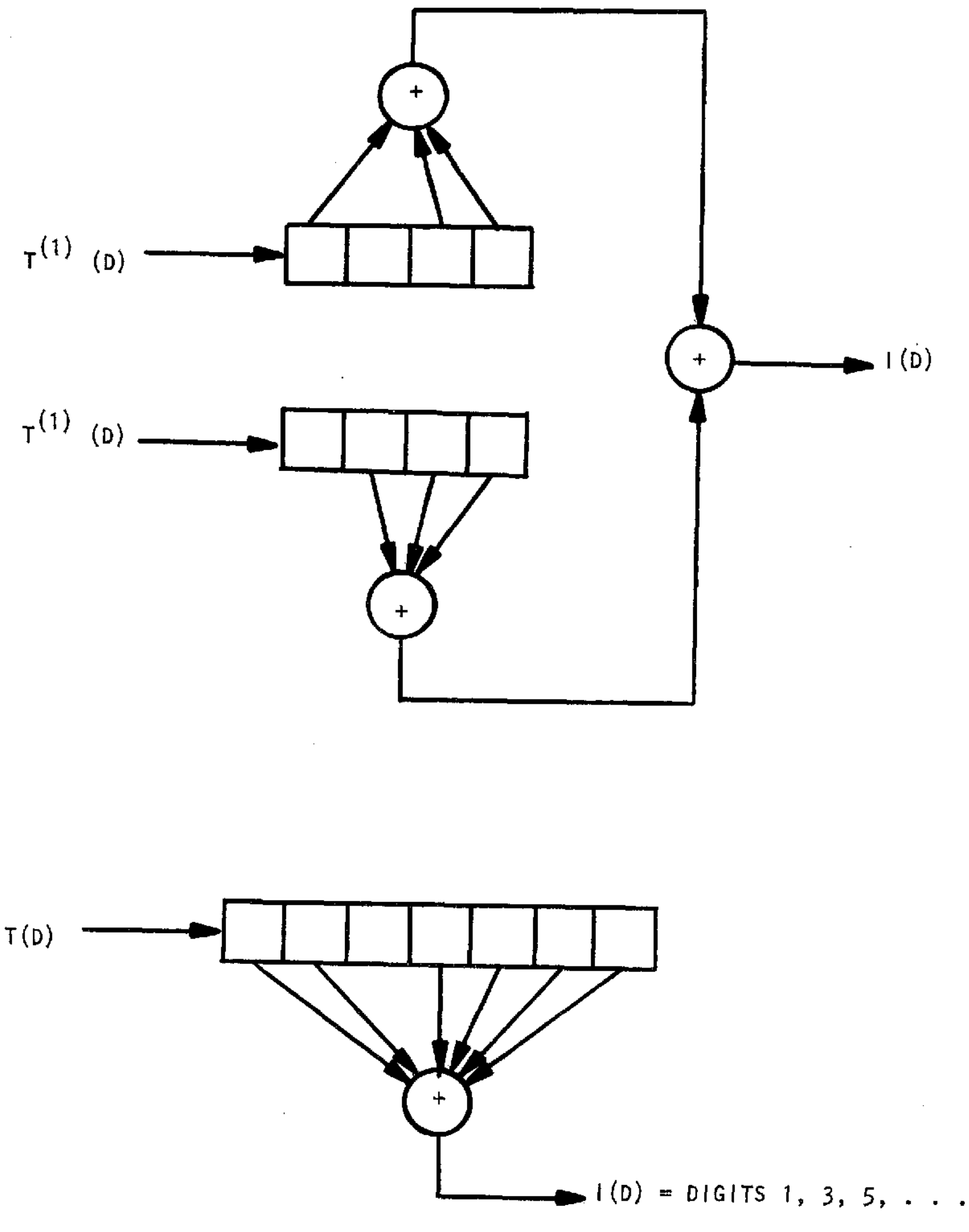


FIGURE 57. ZERO DELAY INVERSE CIRCUITS FOR THE CODE

1 1 1 0 1

1 1 0 0 1

are equivalent. This follows readily from the relation

$$[I(D)\, G(D)]^* = I^*(D)\, G^*(D)$$

and the fact that the weights of a polynomial and its reciprocal are identical.

We can, therefore, restrict our search to quick-look codes with delay $L \le [K/2]$ and an appropriate number of ones in the generator polynomials (the square brackets denote the integer part).

Now let $L = 0$. Then each 1 in the input sequence $I(D)$ will produce a 1 in the output sequence $T(D)$ as it enters the encoder shift register and in addition the last 1 in $I(D)$ will produce two 1's in $T(D)$ as it enters the last stage of the encoder. Therefore,

$$W[T(D)] \ge 2 + W[I(D)]$$

and it follows that in testing whether a code has free distance less than d_f only input sequences with fewer than $d_f - 2$ ones need to be considered.

Since Bahl and Jelinek (1971) have shown that without loss of generality input sequences with zero-runs of length $K - 2$ or more may likewise be ignored, it follows that the length of the input sequences that must be tested does not exceed

$$(d_f - 4)\,(K - 2) + 1$$

For $L > 0$, the first 1 in $I(D)$ produces two 1's in $T(D)$ as it enters the encoder and another 1 as it enters the $(L + 1)$st stage of the encoder. Every subsequent 1 in $I(D)$ likewise produces a 1 in $T(D)$ as it enters the $(L + 1)$st stage. In addition, the last 1 in $T(D)$ results in two 1's in $T(D)$ as it enters the last stage of the encoder. Thus, the total number of ones in the output sequence satisfies

$$W[T(D)] \ge 4 + W[I(D)]$$

and we can restrict consideration to input sequences with fewer than $d_f - 4$ ones and length no larger than

$$(d_f - 6)\ (K - 2) + 1$$

Using these principles we tested all quick-look codes of constraint length $3 \leq K \leq 7$. Table 3 summarizes our results. Note that for $3 \leq K \leq 6$ the best quick-look codes are comparable to the best general nonsystematic codes, whereas for $K = 7$ the free distance of the best quick-look codes is one less than the maximum achievable.

We also remark that the quick-look codes with $L = 0$ are uniformly inferior to those with $L > 0$, a result that reinforces the notion that among the best codes of a class there is always one whose generators possess complementarity (Bahl and Jelinek, 1972).

Although under normal circumstances free distance is a good indicator of a code's error correcting capability, this measure nevertheless depends only on the code and thus completely ignores the nature of the channel and the decoding algorithm. Even with the channel and decoder fixed, differences in the weight spectra of two codes with the same free distance can give rise to different decoder bit error rates.

For these reasons we have computed the decoder bit error rates of selected codes from Table 3 used over the binary symmetric channel and in conjunction with a Viterbi maximum likelihood decoding algorithm of 32 bit decoder path lengths. The results are presented in Figure 58. Note that these quick-look codes compare favorably to the best nonsystematic codes obtained in [7] and the complementary codes given by Jelinek and Bahl (1969).

In Figure 59 we show the error amplification factor for the same set of codes as above, as a function of the signal-to-noise ratio of a binary symmetric channel.

K	Code #	$G^{(1)}$ (octal)	L	d_f	d_{fmax}
3	1	7	1	5	5
4	2	17	1	6	6
5	3	33	1	7	7
	4	35	2	7	7
6	5	67	1	8	8
	6	75	1	8	8
7	7	153	1	9	10
	8	163	1	9	10
	9	127	2	9	10
	10	135	2	9	10
	11	165	2	9	10
	12	171	2	9	10
	13	175	2	9	10
	14	133	3	9	10

Best Rate 1/2 Quick-Look Codes

TABLE 3

Finally, Figure 60 presents the relationship between actual and measured channel bit error rates for the same codes, the binary symmetric channel and a 32 bit path length Viterbi decoder.

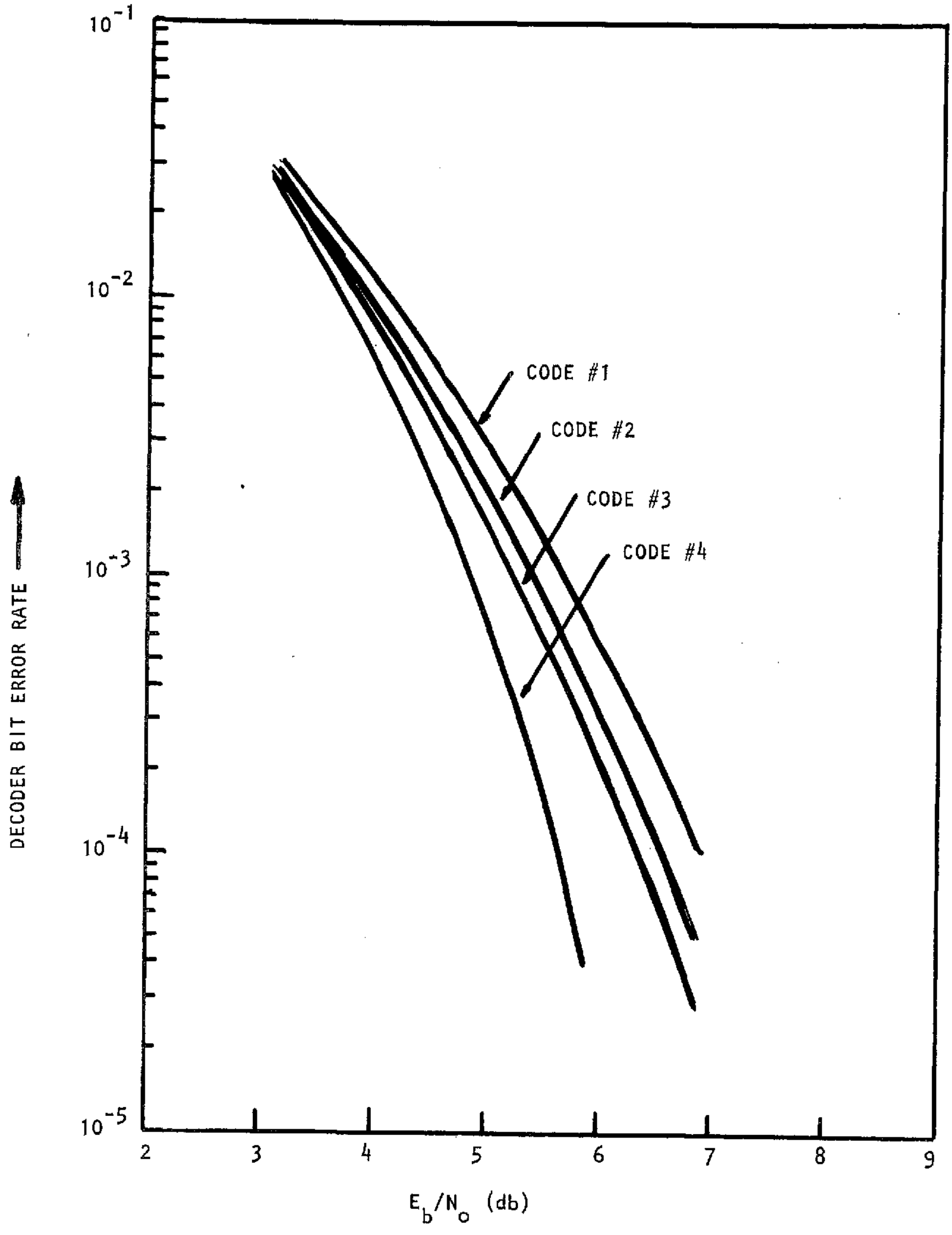


FIGURE 58. VITERBI MAXIMUM LIKELIHOOD DECODING PERFORMANCE $\Delta = 32$

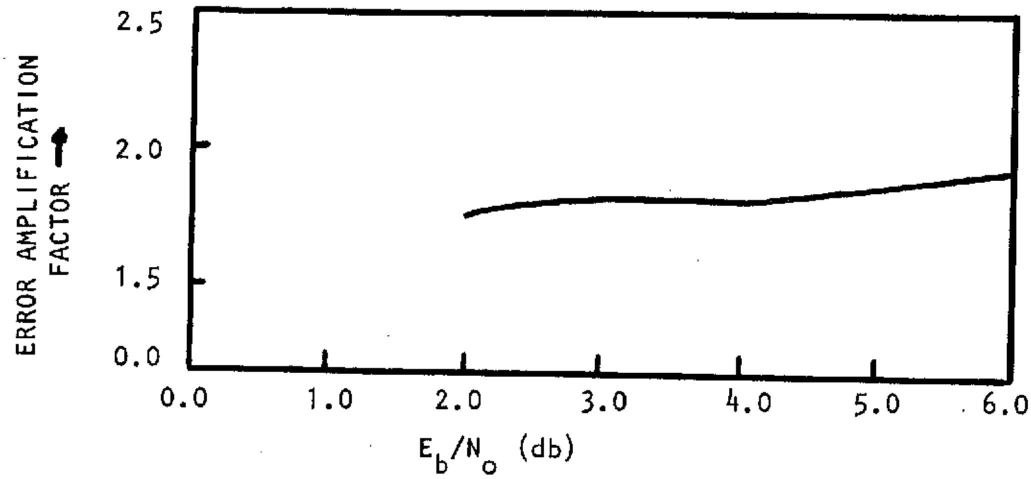


FIGURE 59. ERROR AMPLIFICATION FACTOR FOR CODES #1, 2, 3, 7

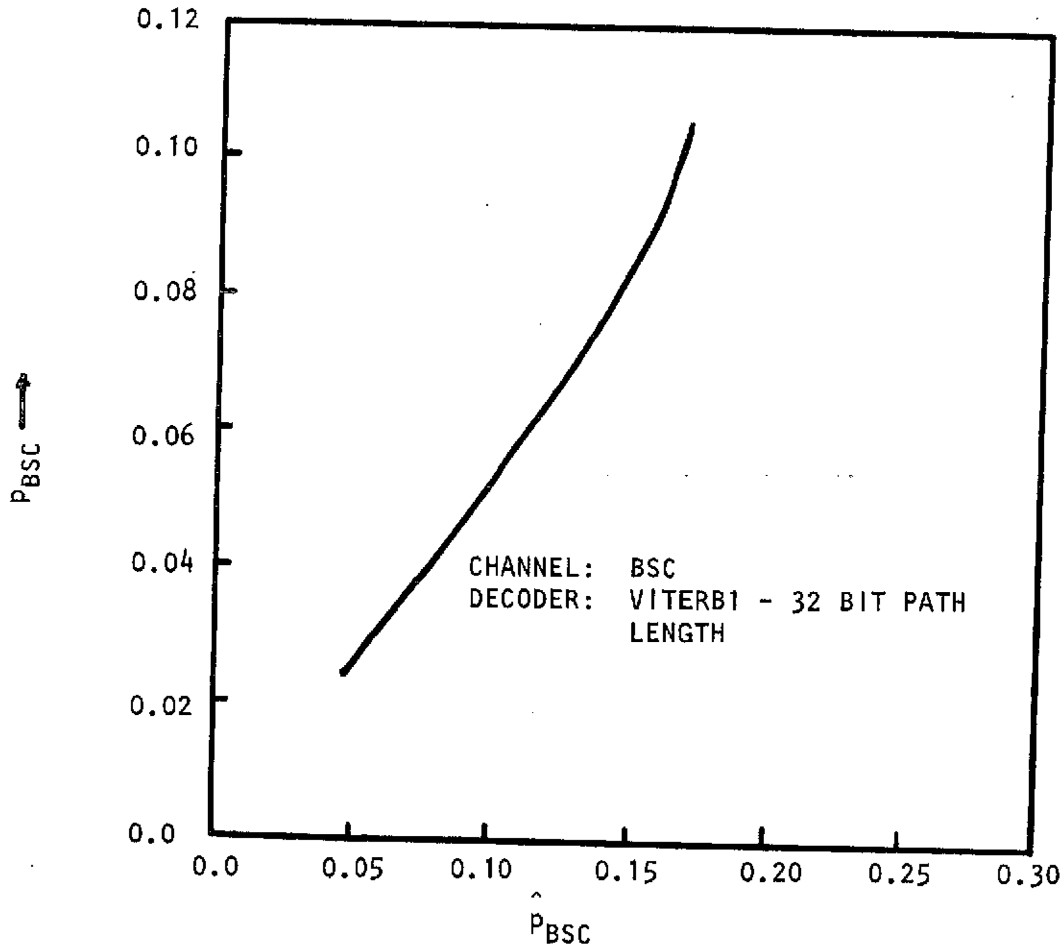


FIGURE 60. MEASURED VERSUS ACTUAL CHANNEL BIT ERROR PROBABILITY FOR CODES #1, 2, 3, 7

IV. References:

1. Massey, J. L. and M. K. Sain (1968), "Inverses of Linear Sequential Circuits," IEEE Trans. Computers, Vol. C-17, No. 4, pp. 330-337.

2. Massey, J. L. and D. J. Costello (1971), "Nonsystematic Convolutional Codes for Sequential Decoding in Space Applications," IEEE Trans. Comm. Technology, Vol. COM-19, No. 5, pp. 806-813.

3. Larsen, K. J. (1973), "Short Convolutional Codes with Maximal Free Distance for Rates 1/2, 1/3, and 1/4," IEEE Trans. Information Theory, Vol. IT-19, No. 3, pp. 371-372.

4. Jelinek, F. and L. R. Bahl (1966), "Maximum-Likelihood and Sequential Decoding of Short Constraint Length Convolutional Codes," Proc. 7th Allerton Conf. Circuit and System Theory, Monticello, Ill., pp. 130-139.

5. Bahl, L. R. and F. Jelinek (1971), "Rate 1/2 Convolutional Codes with Complementary Generators," IEEE Trans. Information Theory, Vol. IT-17, No. 6, pp. 718-727.

6. Bahl, L. R. and F. Jelinek (1972), "On the Structure of Rate 1/2 Convolutional Codes," IEEE Trans. Information Theory, Vol. IT-18, No. 1, pp. 192-196.

7. "Coding Systems Study for High Data Rate Telemetry Links," NASA CR 114278, Prepared by Linkabit Corporation, January 1971.