A VOLTERRA SERIES APPROACH TO CALCULATING THE PROBABILITY OF ERROR IN A NONLINEAR DIGITAL COMMUNICATION CHANNEL

by

FREDERICK W. RATCLIFFE

B. A., LaVerne University, 1975
B. S., Kansas State University, 1977

A MASTER'S REPORT

submitted in partial fulfillment of the

requirements for the degree

MASTER OF SCIENCE

Department of Electrical Engineering

KANSAS STATE UNIVERSITY
Manhattan, Kansas

1979

Approved by:

Ronald R. Hummels
Major Professor

Spec Coll.
LD
2668
.R4
1979
R39
c.2

TABLE OF CONTENTS

LIST OF FIGURES

CHAPTER I

INTRODUCTION

Many, perhaps most, of the communication systems in use today contain nonlinear elements. Unfortunately most of the techniques for solving linear systems are not applicable to nonlinear systems. If a linear approximation of a nonlinear system does not yield satisfactory results, a designer is faced with three possible alternatives. The engineer can actually construct the system and then measure its performance. Aside from the financial cost, the designer can easily find himself spending more time and energy on the "nuts and bolts" aspects without ever addressing the central design problem. An attractive alternative to prototyping an unproven design is simulation. Simulation is relatively easy and straightforward to implement. Due to the physical limitations of simulation hardware, high speed communication channels are often modeled at a fraction of the desired rate. A disadvantage of simulation then becomes the very long time intervals that may be required to obtain a confident estimate of the performance of a system. The analytic approach offers the advantage that the designer can describe the system more precisely. Unfortunately, many systems are either impossible to describe analytically, or the resulting description may be so complex as to be unmanageable.

One approach which can be used for a large class of nonlinear systems involves the use of Volterra series [1,2,3]. An overdriven

transistor amplifier is one example of a physical system having sufficiently well behaved nonlinearities with memory which can be analytically described in terms of a Volterra series. This paper will consider the model of the nonlinear digital communication channel shown in Figure 1.

The basic mathematical model of the Volterra series approach to calculating probability of error is developed in Chapter II. The computer algorithms used to implement the mathematical model are described in Chapter III. Results for two channels, one having Gaussian filters and the other having casual filters, are presented in Chapter IV. The results from the channel having Gaussian filters are essentially that of Benedetto, Biglieri, and Daffara [4].

THIS BOOK CONTAINS NUMEROUS PAGES WITH DIAGRAMS THAT ARE CROOKED COMPARED TO THE REST OF THE INFORMATION ON THE PAGE.

THIS IS AS RECEIVED FROM CUSTOMER.

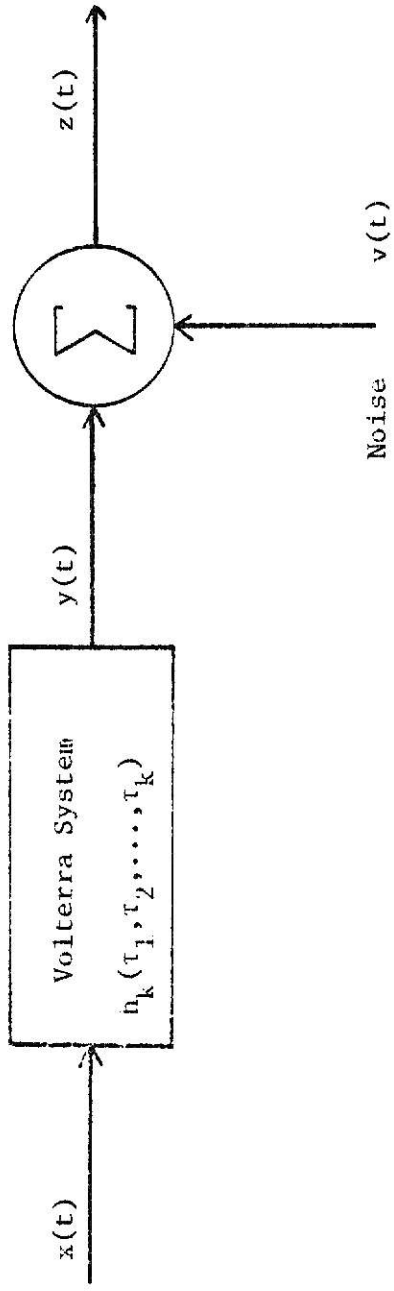

Figure 1. Communication Channel Model

CHAPTER II

BASIC MATHEMATICAL MODEL

Validity of the Volterra Series Approach

A sufficiently well behaved nonlinear system with memory can be represented as a Volterra series [5] of the form

$$y(t) = \sum_{k=1}^{\infty} \int\int\cdots\int_{-\infty}^{\infty} h_k(\tau_1,\tau_2,\ldots,\tau_k)\ x(t-\tau_1)x(t-\tau_2)\cdots x(t-\tau_k)\ d\tau_1 d\tau_2\cdots d\tau_k \quad (1)$$

where the Volterra kernels, $h_k(\tau_1,\tau_2,\ldots,\tau_k)$, describe the behavior of the system. To show how one would go about solving for the Volterra kernels consider the expanded Volterra system shown in Figure 2. This model describes a nonlinear channel which has linear filters on either side of a memoryless nonlinearity. If sufficiently well behaved, the nonlinearity can be expanded in form of the power series

$$f(r) = \sum_{k=1}^{\infty} \gamma_k \frac{r^k}{k!} \quad . \quad (2)$$

Analysis of the model begins by observing that the output of the first filter in the channel is

$$r(t) = \int_{-\infty}^{\infty} h_{in}(p)\ x(t-p)\ dp \quad . \quad (3)$$

The output of the memoryless nonlinear device is then

$$s(t) = f\left[\int_{-\infty}^{\infty} h_{in}(p)\ x(t-p)\ dp\right] \quad . \quad (4)$$

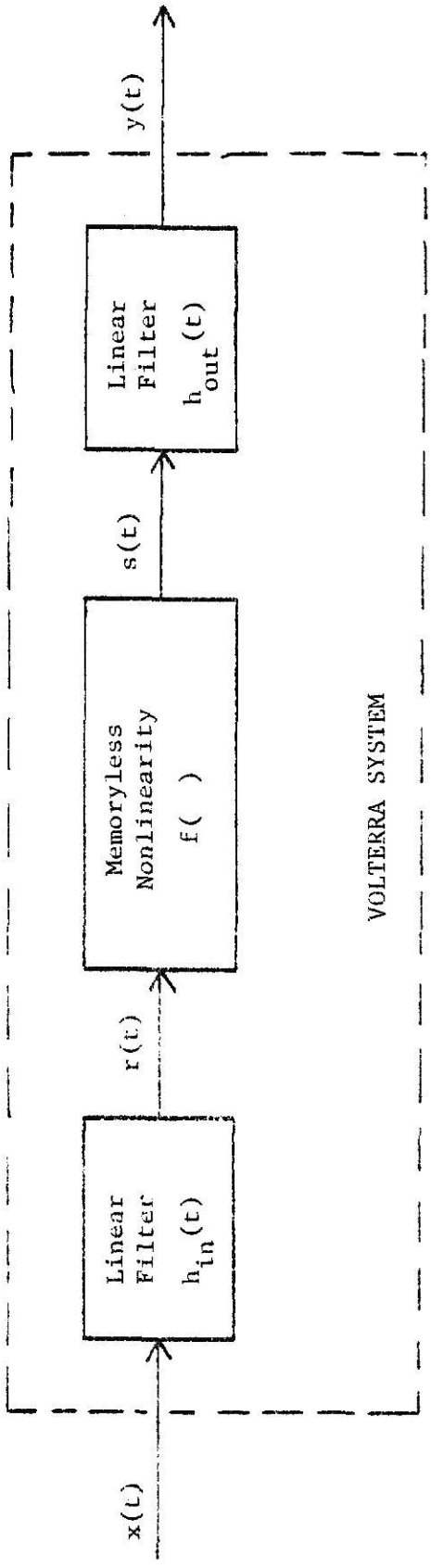

Figure 2. Expanded Volterra System

A second convolution will yield the channel output

$$y(t) = \int_{-\infty}^{\infty} h_{out}(\tau)\ s(t-\tau)\ d\tau$$

$$= \int_{-\infty}^{\infty} h_{out}(\tau)\ f[\int_{-\infty}^{\infty} h_{in}(p)\ x(t-\tau-p)\ dp]\ d\tau \quad .$$

Making the change of variables $\tau + p = u$ will yield

$$y(t) = \int_{-\infty}^{\infty} h_{out}(\tau)\ f[\int_{-\infty}^{\infty} h_{in}(u-\tau)\ x(t-u)\ du]\ d\tau \quad , \tag{5}$$

which after the substitution for f( ) using (2) will give

$$y(t) = \int_{-\infty}^{\infty} h_{out}(\tau)\left\{\sum_{k=1}^{\infty} \frac{\gamma_k}{k!} [\int_{-\infty}^{\infty} h_{in}(u-\tau)\ x(t-u)\ dh]^k\right\} d\tau \quad . \tag{6}$$

Interchanging the order of the first integral and summation yields

$$y(t) = \sum_{k=1}^{\infty} \int_{-\infty}^{\infty} \frac{\gamma_k}{k!} h_{out}(\tau)[\int_{-\infty}^{\infty} h(u-\tau)\ x(t-u)\ du]^k\ d\tau \quad . \tag{7}$$

Expanding (7) on a term by term basis obtains

$$y(t) = \int_{-\infty}^{\infty} \frac{\gamma_1}{1!} h_{out}(\tau) \int_{-\infty}^{\infty} h_{in}(u_1-\tau)\ x(t-u_1)\ du_1 d\tau$$

$$+ \int_{-\infty}^{\infty} \frac{\gamma_2}{2!} h_{out}(\tau) \int_{-\infty}^{\infty} h_{in}(u_1-\tau)\ x(t-u_1)\ du_1 \int_{-\infty}^{\infty} h_{in}(u_2-\tau)\ x(t-u_2)\ du_2 d\tau$$

$$+ \ldots \quad , \tag{8}$$

which can be rearranged as

$$y(t) = \int_{-\infty}^{\infty} \frac{\gamma_1}{1!} \int_{-\infty}^{\infty} h_{out}(\tau)\ h_{in}(u_1-\tau)\ d\tau\ x(t-u_1)\ du_1$$

$$+ \iint \frac{\gamma_2}{2!} \int_{-\infty}^{\infty} h_{out}(\tau)\ h_{in}(u_1-\tau)\ h_{in}(u_2-\tau)\ d\tau\ x(t-u_1)\ x(t-u_2)\ du_1 du_2$$

$$+ \ldots + \int\cdots\int_{\infty}^{\infty} \frac{\gamma_k}{k!} \int_{-\infty}^{\infty} h_{out}(\tau) \prod_{i=1}^{k} h_{in}(u_i-\tau)\ d\tau$$

$$\cdot\ x(t-u_1)\ x(t-u_2) \ldots x(t-u_k)\ du_1 du_2 \ldots du_k + \ldots \quad . \tag{9}$$

Comparison of (1) with (9) shows that if

$$h_k(u_1,u_2,\ldots,u_k) = \frac{\gamma_k}{k!} \int_{-\infty}^{\infty} h_{out}(\tau) \prod_{i=1}^{k} h_{in}(u_1-\tau)\, d\tau \quad , \tag{10}$$

where $h_k(\tau_1,\tau_2,\ldots,\tau_k) \equiv h_k(u_1,u_2,\ldots,u_k)$,

then (1) and (9) are equivalent. Figure 2 thus describes a nonlinear communication channel which can be expressed in terms of a Volterra series.

Error Calculations

Returning to Figure 1, an expression will now be formulated for calculating the error probability for a channel where zero mean Gaussian noise is added after the nonlinearity. The development presented here parallels the one used by Benedetto et al. [4].

The input is specified as

$$x(t) = \sum_{n=-\infty}^{\infty} a_n\, \delta(t-nT) \quad , \tag{11}$$

where $\{a_n\}$ is a sequence of random data symbols which are statistically independent for different values of n. Substitution of (11) into (1) yields

$$y(t) = \sum_{k=1}^{\infty} \int\cdots\int_{-\infty}^{\infty} h_k(\tau_1,\tau_2,\ldots,\tau_k) \left[\sum_{n_1=-\infty}^{\infty} a_{n_1}\, \delta(t-n_1T)\right]$$

$$\cdots \left[\sum_{n_k=-\infty}^{\infty} a_{n_k}\, \delta(t-n_kT)\right] d\tau_1 d\tau_2 \ldots d\tau_k \quad ,$$

which due to the presence of the impulse functions can be easily integrated to get

$$y(t) = \sum_{k=1}^{\infty} \sum_{n_1=-\infty}^{\infty} \cdots \sum_{n_k=-\infty}^{\infty} a_{n_1} a_{n_2} \ldots a_{n_k}\, h_k(t-n_1T, t-n_2T, \ldots, t-n_kT) \quad . \tag{12}$$

Although choosing a series of impulses as our input simplified (1) to (12), one might argue that an impulse generator does not exist in the real world. It turns out that using a series of impluses for an input does not constrain the model. Referring to Figure 2, the first device the input encounters in the channel model is a linear filter. When specifying the impulse response, $h_{in}(t)$, any pulse shaping characteristics desired can be included. An example of adding a pulse shaping filter to $h_{in}(t)$ will be presented in Chapter IV.

The output of the nonlinear channel is

$$z(t) = y(t) + v(t) \quad , \tag{13}$$

where the noise, $v(t)$, is a zero mean Gaussian random process. The process is assumed to be stationary with the variance of its samples equal to σ_v^2. At the t_o instant during the current signal interval, the receiver decides which symbol a_o represents.

What (13) implies is more meaningful if written in an expanded form at the instant t_o. That is

$$z(t_o) = a_o\, h_1(t_o) + \sum_{\substack{n_1=-\infty \\ n_1 \neq 0}}^{\infty} a_{n_1}\, h_1(t_o - n_1 T)$$

$$+ \sum_{n_1=-\infty}^{\infty} \sum_{n_2=-\infty}^{\infty} a_{n_1} a_{n_2}\, h_2(t_o - n_1 T, t_o - n_2 T)$$

$$+ \text{higher order terms} + v(t_o) \quad . \tag{14}$$

The output $z(t_o)$ has parts which can be interpreted as

$$z(t_o) = \begin{matrix}\text{desired} \\ \text{signal}\end{matrix} + \begin{matrix}\text{linear intersymbol} \\ \text{interference}\end{matrix} + \begin{matrix}\text{square law} \\ \text{distortion terms}\end{matrix}$$

$$+ \begin{matrix}\text{higher order} \\ \text{distortion terms}\end{matrix} + \text{noise} \quad .$$

A more compact representation for (14) can be obtained if the random variable $\Xi(a_o)$ is defined which includes all the system distortion terms. That is

$$\Xi(a_o) = \sum_{\substack{n_1=-\infty \\ n_1 \neq 0}}^{\infty} a_{n_1} h_1(t_o - n_1 T) + \sum_{k=2}^{\infty} \sum_{n_1} \cdots \sum_{n_k} a_{n_1} a_{n_2} \ldots a_{n_k} h_k(t_o - n_1 T, t_o - n_2 T, \ldots, t_o - n_k T) \quad . \tag{15}$$

Then (14) can be restated

$$z(t_o) = a_o h_1(t_o) + \Xi(a_o) + v(t_o) \quad . \tag{16}$$

Given (16) as the output the probability of error in correctly interpreting the symbol a_o can now be determined. The overall probability of error is

$$P_{error} = \sum_{i=1}^{L} P_{s_i} P[e|s_i] \quad , \tag{17}$$

where L is the number of symbols; P_{s_i} is the probability that a given symbol will be transmitted; $P[e|s_i]$ is the probability that s_i will be incorrectly interpreted if transmitted. Note that

$$P_{s_1} + P_{s_2} + \ldots + P_{s_L} = 1 \tag{18}$$

is always true.

For the remainder of this report the signal set will be restricted to an even number of equally likely symbols defined by

$$s_i = -L - 1 + 2i \quad \text{for} \quad i = 1,2,\ldots,L \quad . \tag{19}$$

It can be shown for the signal set chosen that the lowest probability of error will occur when the decision boundaries are equally spaced between the symbols. A signal set containing four symbols and the associated

decision boundaries are shown in Figure 3. For example, a received signal falling into the R_2 region would be interpreted as the S_2 symbol.

The extreme lower symbol will be incorrectly interpreted when the noise and channel distortion are sufficiently large to place the symbol in any other region. More specifically, when

$$v(t_o) + \Xi(a_o = -L + 1) > 1 \tag{20}$$

then the probability of error for the lower symbol will be

$$P[e|s_{lower}] = P\left\{v(t_o) + \Xi(a_o = -L + 1) > 1\right\}$$

$$= P\left\{v(t_o) > 1 - \Xi(a_o = -L + 1)\right\} \quad . \tag{21}$$

Thus, if $v(t_o)$ is greater than $1 - \Xi(a_o = -L + 1)$ an error will be made in interpreting the symbol. Previously it was stated that $v(t)$ is a zero mean Gaussian process. This means that $P[e|s_{lower}]$ can be determined by integrating over the shaded region shown in Figure 4. Integrating one gets

$$P[e|s_{lower}] = E\left\{ \int_{1-\Xi(a_o = -L+1)}^{\infty} f_v(v)\, dv \right\}$$

$$= E\left\{ \int_{1-\Xi(a_o = -L+1)}^{\infty} \frac{e^{-v^2/2\sigma_v^2}}{\sqrt{2\pi}\,\sigma_v} dv \right\} \quad . \tag{22}$$

The change of variables $\beta = \frac{v}{v_o}$ enables (22) to be written as

$$P[e|s_{lower}] = E\left\{ \int_{\frac{1-\Xi(a_o = -L+1)}{\sigma_v}}^{\infty} \frac{e^{-\beta^2/2}}{\sqrt{2\pi}} d\beta \right\}$$

$$= E\left\{ Q\left[\frac{1-\Xi(a_o = -L+1)}{\sigma_v} \right] \right\} \quad . \tag{23}$$

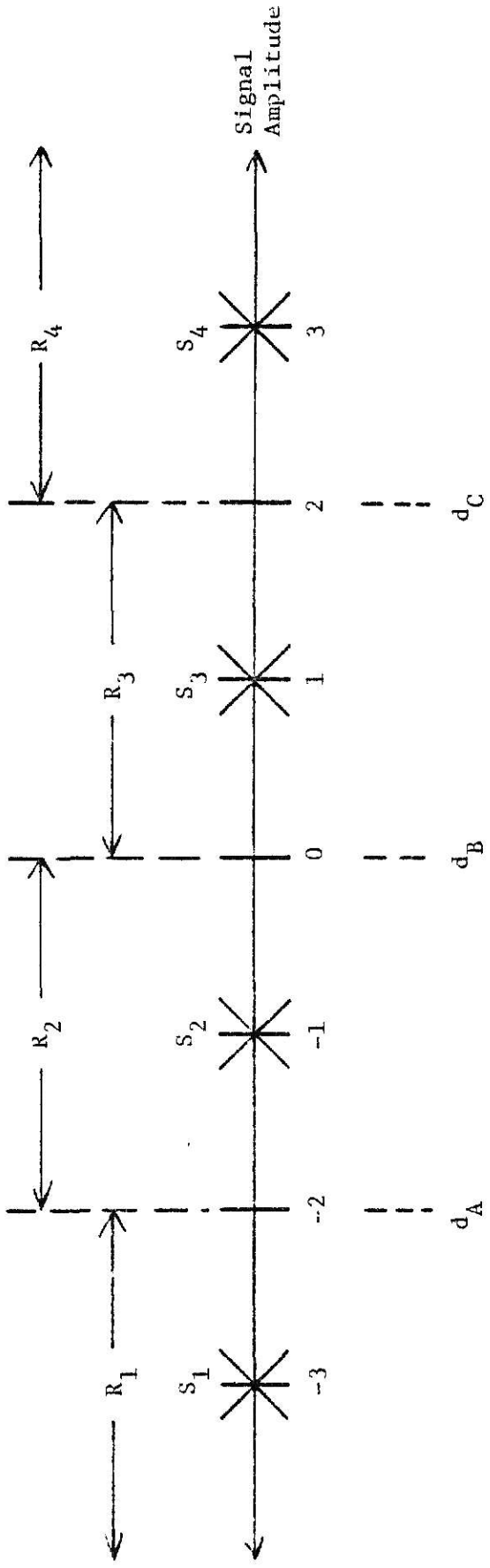

Figure 3. Sample Signal Map

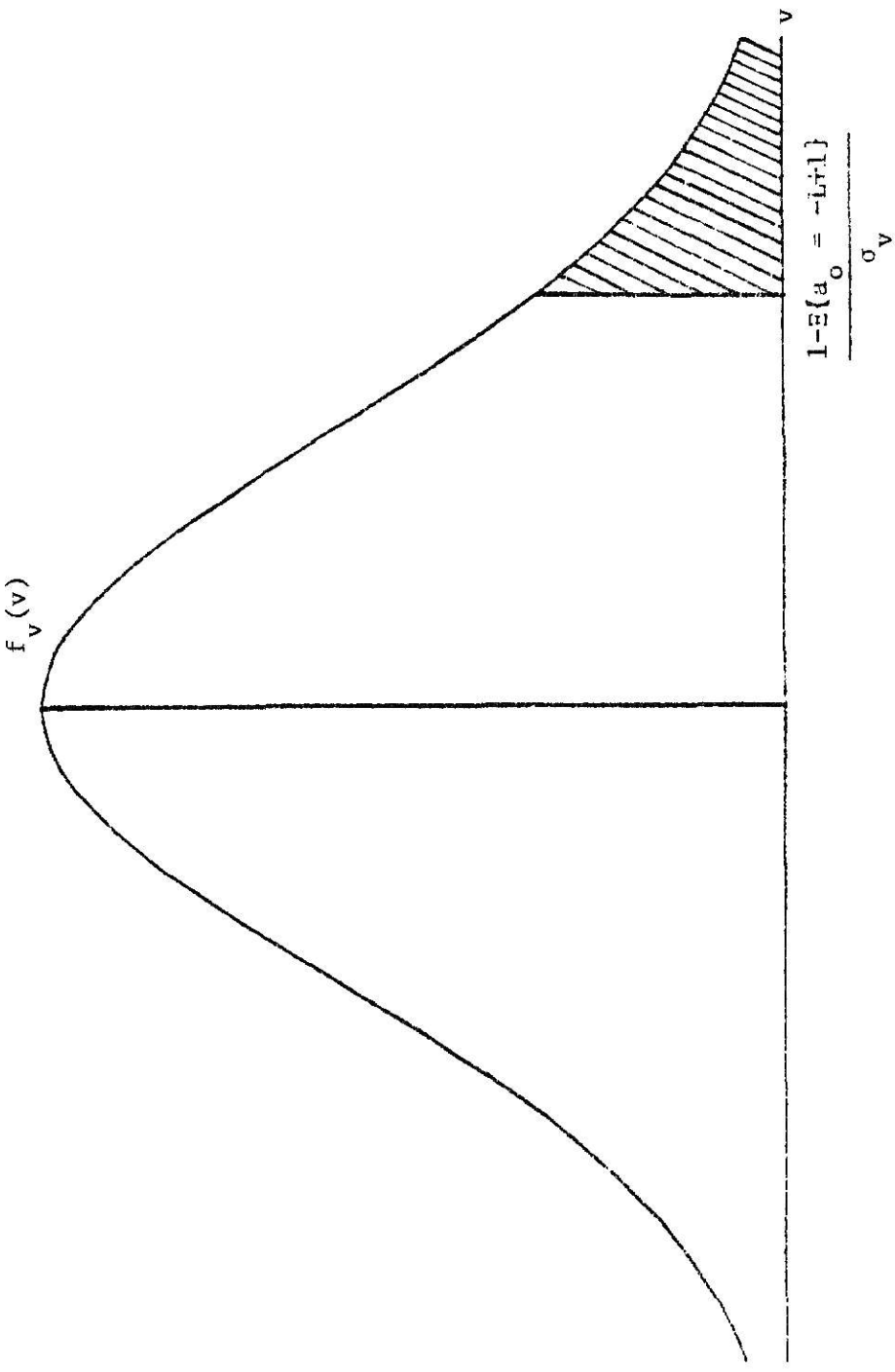

Figure 4. Noise Probability Density Function

The same procedure applied to the extreme upper symbol yields

$$P[e|s_{upper}] = E\left\{Q\left[\frac{1 + \Xi(a_o = L-1)}{\sigma_v}\right]\right\} \quad . \tag{24}$$

If the signal set contains any intermediate symbols, incorrect interpretation can occur if they fall into decision regions either above or below their own. For example, referring again to Figure 3, s_2 would not be received correctly if

$$\left|v(t_o) + \Xi(a_o = -1)\right| > 1 \quad .$$

Consequently the probability an intermediate symbol will be incorrectly interpreted is

$$P[e|s_{intermediate}] = E\left\{Q\left[\frac{1+\Xi(a_o=s_i)}{\sigma_v}\right]\right\} + E\left\{Q\left[\frac{1-\Xi(a_o=s_i)}{\sigma_v}\right]\right\} \quad . \tag{25}$$

For L symbols we can state the general result as

$$P_{error} = \frac{1}{L}\sum_{i=1}^{L-1} E\left\{Q\left[\frac{1-\Xi(a_o=-L-1+2i)}{\sigma_v}\right]\right\} + \frac{1}{L}\sum_{i=2}^{L} E\left\{Q\left[\frac{1+\Xi(a_o=-L-1+2i)}{\sigma_v}\right]\right\} \quad . \tag{26}$$

If $\alpha \geq 2.0$, a good approximation for $Q(\alpha)$ is

$$Q(\alpha) \approx \frac{e^{-\alpha^2/2}}{\sqrt{2\pi}\,\alpha} \quad \text{for} \quad \alpha \geq 2 \quad . \tag{27}$$

Evaluation of (26) can be performed by using the fundamental theorem of expectation [6] which states

$$E\{g(x)\} \equiv \int_{-\infty}^{\infty} g(x)\, f_x(x)\, dx \quad , \tag{28}$$

where $f_x(x)$ is the probability density function (pdf) for the random variable x.

Fortunately an analytic description of $f_x(x)$ is not needed. If the first 2N moments of $f_x(x)$ are known, then a Gauss Quadrature Rule (GQR)

consisting of N abscissa/weight values can be calculated using a procedure outlined by Golub and Welsch [7]. Using the GQR obtained, (28) can be evaluated as

$$\int_{-\infty}^{\infty} g(x)\, f_x(x) \simeq \sum_{j=1}^{N} w_j\, g(A_j) \quad . \tag{29}$$

Using (27), (28) and (29) the overall expression for error probability becomes

$$P_{error} = \frac{1}{L}\left\{ \sum_{i=1}^{L-1} \sum_{j=1}^{N} \frac{W_{ij}\, e^{-\left(\frac{1-A_{ij}}{\sigma_v}\right)^2/2}}{\sqrt{2\pi}\left(\frac{1-A_{ij}}{\sigma_v}\right)} + \sum_{i=2}^{L} \sum_{j=1}^{N} \frac{W_{ij}\, e^{-\left(\frac{1+A_{ij}}{\sigma_v}\right)^2/2}}{\sqrt{2\pi}\left(\frac{1+A_{ij}}{\sigma_v}\right)} \right\}$$

$$\text{for} \quad \left(\frac{1\pm A_{ij}}{\sigma_v}\right) \geq 2 \quad . \tag{30}$$

All that remains is to develop a method for obtaining the 2N moments of $\Xi\{a_o\}$ to be used in calculating the GQR.

Calculation of the Moments of $\Xi\{a_o\}$

The first moment of $\Xi\{a_o\}$ is obtained by taking the expected value of (15). That results in

$$E\{\Xi(a_o)\} = E\left\{ \left[\sum_{\substack{n_1=-\infty \\ n_1 \neq 0}}^{\infty} a_{n_1} H_1(n_1) + \sum_{k=2}^{\infty} \sum_{n_1} \sum_{n_2} \cdots \sum_{n_k} a_{n_1} a_{n_2} \cdots a_{n_k} H_k(n_1, n_2, \ldots, n_k) \right] \middle| a_o \right\} \quad , \tag{31}$$

where for brevity $H_k(n_1, n_2, \ldots, n_k) \equiv h_k(t_o - n_1 T, t_o - n_2 T, \ldots, t_o - n_k T)$.

The kernels $H_k(n_1,n_2,\ldots,n_k)$ in (31) are known quantities. Only the $E\{a_{n_1},a_{n_2},\ldots,a_{n_k}|a_o\}$ needs to be evaluated. As n_i can range in value from minus infinity to plus infinity, each a_{n_i} can take on any value from the set $\{\ldots a_{-2},a_{-1},a_o,a_1,a_2,\ldots\}$. For example if k = 8, one term of $E\{a_o\}$ might have a subscript distribution which results in

$$E\{a_e a_3 a_o a_1 a_o a_3 a_1 a_3|a_o\} = a_o^2\ E\{a_1^2\}\ E\{a_3^4\} \quad . \tag{32}$$

Recall that the a_{n_i} s not having the same subscript n_i are statistically independent. Since each of the possible symbols are equally likely,

$$E\{a_i^\ell\} = \frac{1}{L}[(-L+1)^\ell + \ldots + (-3)^\ell + (-1)^\ell + (1)^\ell + (3)^\ell + \ldots + (L-1)^\ell] \quad . \tag{33}$$

With the aid of (32) and (33), it is relatively easy to evaluate (31) for a specific subscript distribution. Note that if ℓ is odd then $E\{a_i^\ell\} = 0$. Now is needed a method for obtaining the higher moments of $E(a_o)$.

When two Volterra series

$$V_F = \sum_{n_1} a_{n_1} F_1(n_1) + \sum_{n_1}\sum_{n_2} a_{n_1}a_{n_2} F_2(n_1,n_2) + \ldots$$

and

$$V_G = \sum_{m_1} a_{m_1} G_1(m_1) + \sum_{m_1}\sum_{m_2} a_{m_1}a_{m_2} G_2(m_1,m_2) + \ldots \tag{34}$$

are multipled together the result is

$$\begin{aligned} V_Q = V_F\,V_G = &\sum_{n_1}\sum_{m_1} a_{n_1}a_{m_1} F_1(n_1)\,G_1(m_1) \\ &+ \sum_{n_1}\sum_{m_1}\sum_{m_2} a_{n_1}a_{m_1}a_{m_2} F_1(n_1)\,G_2(m_1,m_2) \\ &+ \sum_{n_1}\sum_{n_2}\sum_{m_1} a_{n_1}a_{n_2}a_{m_1} F_2(n_1,n_2)\,G_1(m_1) \\ &+ \ldots \quad . \end{aligned} \tag{35}$$

Regrouping and renumbering the subscripts will yield

$$V_Q = \sum_{n_1} \sum_{n_2} a_{n_1} a_{n_2} Q_2(n_1,n_2) + \sum_{n_1} \sum_{n_2} \sum_{n_3} a_{n_1} a_{n_2} a_{n_3} Q_3(n_1,n_2,n_3) + \ldots \quad , \quad (36)$$

where

$$Q_k(n_1,n_2,\ldots,n_k) = \sum_{i=1}^{k-1} F_i(n_1,n_2,\ldots,n_i)\, G_{k-i}(n_{i+1},\ldots,n_k) \quad \text{for } k \geq 2$$

$$= 0 \quad \text{for } k < 2 \quad . \quad (37)$$

Thus the product of two Volterra series can also be expressed as a Volterra series. Taking this one step further, specify

$$F_i(n_1,n_2,\ldots,n_i) = H_i^{NCM-1}(n_1,n_2,\ldots,n_i) \quad ,$$

$$G_j(n_1,n_2,\ldots,n_j) = H_j^{1} \quad (n_1,n_2,\ldots,n_j) \quad ,$$

and $$Q_k(n_1,n_2,\ldots,n_k) = H_k^{NCM} \quad (n_1,n_2,\ldots,n_k) \quad , \quad (38)$$

where the superscripts indicate the moment. Substitution of (38) into (37) yields

$$H_k^{NCM}(n_1,n_2,\ldots,n_k) = \sum_{i=\ell-1}^{k-1} H_i^{NCM-1}(n_1,n_2,\ldots,n_i)\, H_{k-i}^{1}(n_{i+1},\ldots,n_k)$$

$$\text{for } k \geq \ell$$

and $$= 0 \quad \text{for } k < \ell \quad , \quad (39)$$

which is the product of the kernel terms used to calculate the previous and first moments. The next higher moment of $\Xi\{a_o\}$ is just the expected value of (39).

CHAPTER III

COMPUTER ALGORITHM IMPLEMENTATION

In Chapter II procedures were developed for calculating the error probability using the Volterra series to represent a nonlinear system. The sequence of computations is summed up in Figure 5. This chapter will discuss the implementation of these procedures on the computer. Listings of the Fortran programs described in this chapter have been included in the Appendix.

Subroutine Kernel

The first step in arriving at an analytical result is to specify the impulse responses of the channel filters, $h_{in}(t)$ and $h_{out}(t)$, along with the order and severity of the channel nonlinearities. Subroutine Kernel outlined in Figure 6 can then be programmed to calculate the kernel terms of the first moment.

The highest nonlinearity used to describe the channel defines the value of k in (1). The infinite summations for the subscripts n_is are also reduced to some finite value $-N_A$ to $+N_B$, which has the effect of limiting the channel impulse response to some multiple of the bit rate. A negative n_i references a signal interval which has already occurred. A zero value for n_i specifies the current signal interval while a positive value for n_i specifies a future signal interval. Note that for causal filters n_i would not take on positive values.

A combination of subscripts is selected and the corresponding kernel magnitude calculated. To reduce the number of terms required without significantly affecting the accuracy of the final result, only those

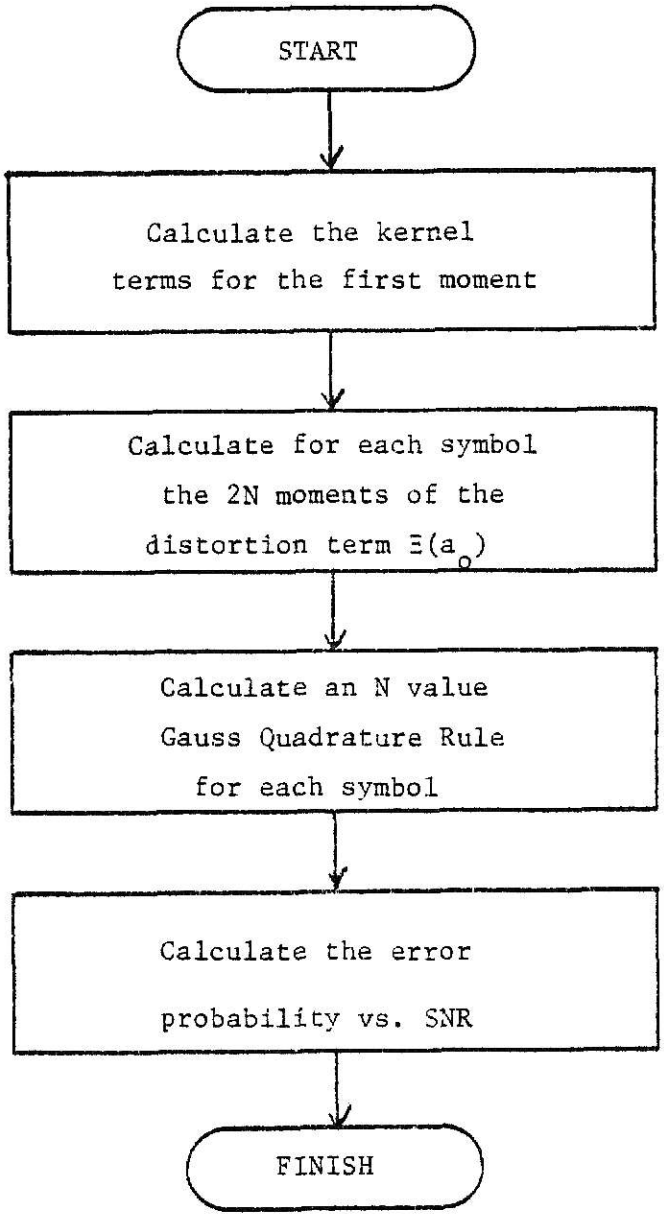

Figure 5. Overall Information Flow

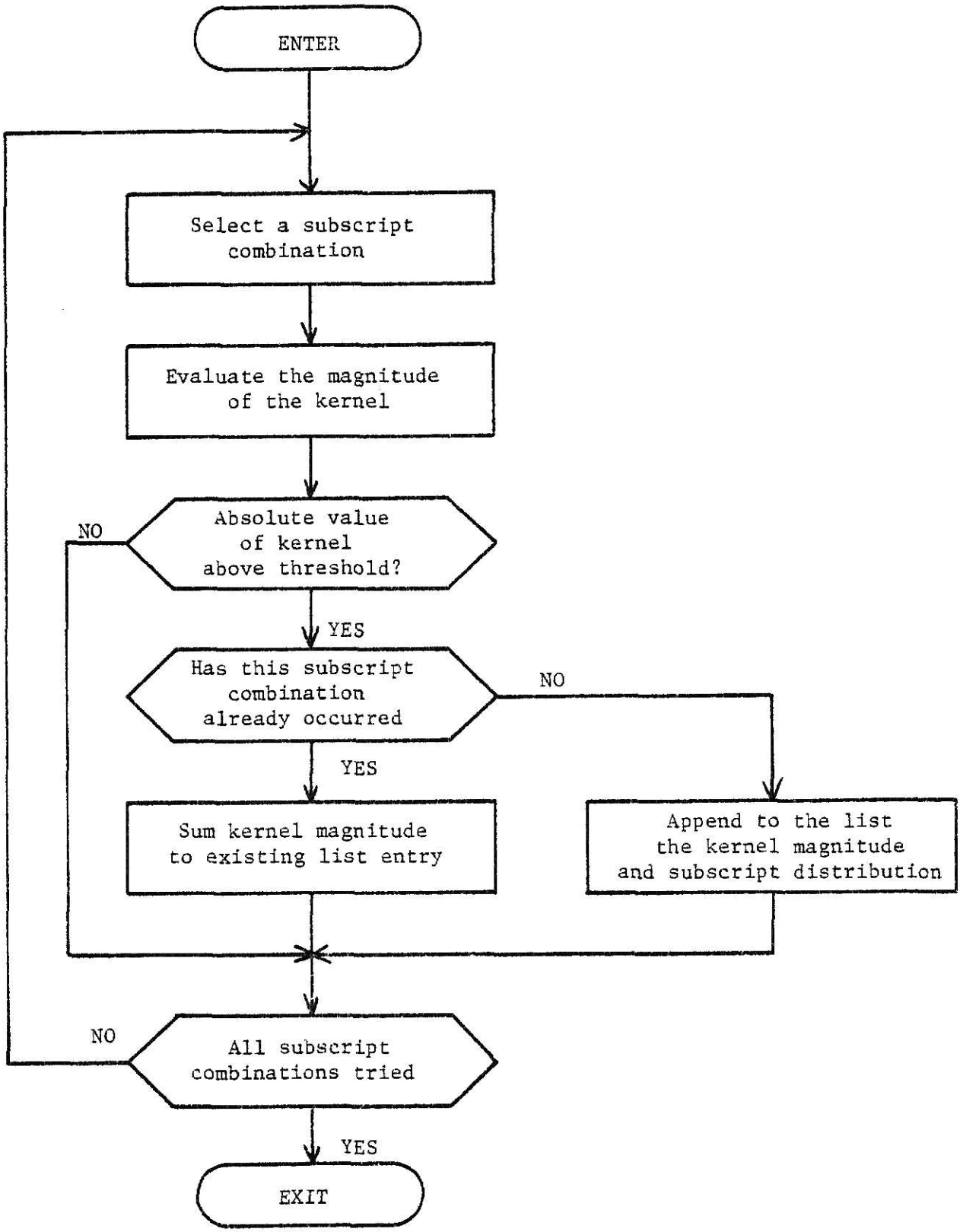

Figure 6. Subroutine Kernel

kernel terms whose absolute magnitude is above a specified threshold are retained. Even with the use of a threshold it is easy to visualize that the calculation of the higher order kernels will yield a very large number of terms. The number of kernel terms that must be saved can be reduced by exploiting their inherent symmetry. Examination of (1) implies that $a_o a_1 a_3 a_o H_4(n_o, n_1, n_3 n_o)$ has the same contribution as $a_3 a_o a_o a_1 H_4(n_3, n_o, n_o, n_1)$. Therefore the previously calculated kernel terms are searched for a match in subscript distribution. If a match is found then the newly calculated kernel magnitude is added to the existing kernel magnitude. If no match is found, new entries are made to store the kernel magnitude and subscript distribution.

Both the magnitude and subscript distribution are required to describe a kernel term. Thus, for each kernel magnitude there is an associated vector containing the subscript description. When the magnitude terms are collected into a vector, the corresponding subscript distribution vectors are merged into a two dimensional array. To reduce the amount of storage required, the two dimensional array can be reduced to a vector. This can be accomplished by using a suitable base to enable the count for each n_i to be interpreted as a digit of an integer number. The integer number containing the subscript information is constructed by calculating

$$I = \sum_{i=1}^{N_T} (\text{number of } a_{n_i} \text{ occurring}) \text{ Base}^{(i-1)} \quad ,$$

where N_T is the number of different values n_i can have. More specifically, consider the example where the suitable base is 10 and n_i can have a value of -1, 0, or 1. Then the ones digit would be the number of n_is which have

a value of -1, the tens digit would be the number of n_is having a value of 0, and so on. The number 131_{10} would say that one $n_i = -1$, three $n_i = 0$, and one $n_i = 1$. It is straightforward to extend this example to an arbitrary base and number of n_is. However to represent numbers exactly on the computer, a base which is a multiple of 2 is required. One word of caution, the base chose must be of sufficient size to enable the largest number of times that any n_i can occur to be represented as a single digit. Returning to the earlier example of using 10 as a base, if $n_i = 0$ occurred ten times it would be incorrectly stored, as one $n_i = 1$.

The discussion of Subroutine Kernel so far is independent of any channel characteristics. Two specific examples of calculating the kernel terms are presented in Chapter IV.

Expected Value Program

The next step is to take the terms supplied by Subroutine Kernel and calculate the desired number of moments of the distortion terms. The Expected Value Program performs this function in a manner shown in Figure 7. Referring to Figure 8, the discussion begins with how the current moment of the distortion term for each symbol is calculated.

In order to calculate the distortion terms, the expected value of the random data samples, $a_{n_1} a_{n_2} \ldots a_{n_k}$, needs to be evaluated. Recall that (33) showed that for odd values of ℓ, the $E\{a_{n_i}^{\ell}\} = 0$. This fact is used to save a considerable amount of computer time. If any of the subscripts occurs an odd number of times, then the corresponding kernel magnitude does not contribute to the $E\{\Xi^{\ell}(a_o)\}$. Note that $E\{a_o^{\ell}\} = a_o^{\ell}$ since the current symbol being received is known. Therefore the $n_i = 0$ subscript is not checked.

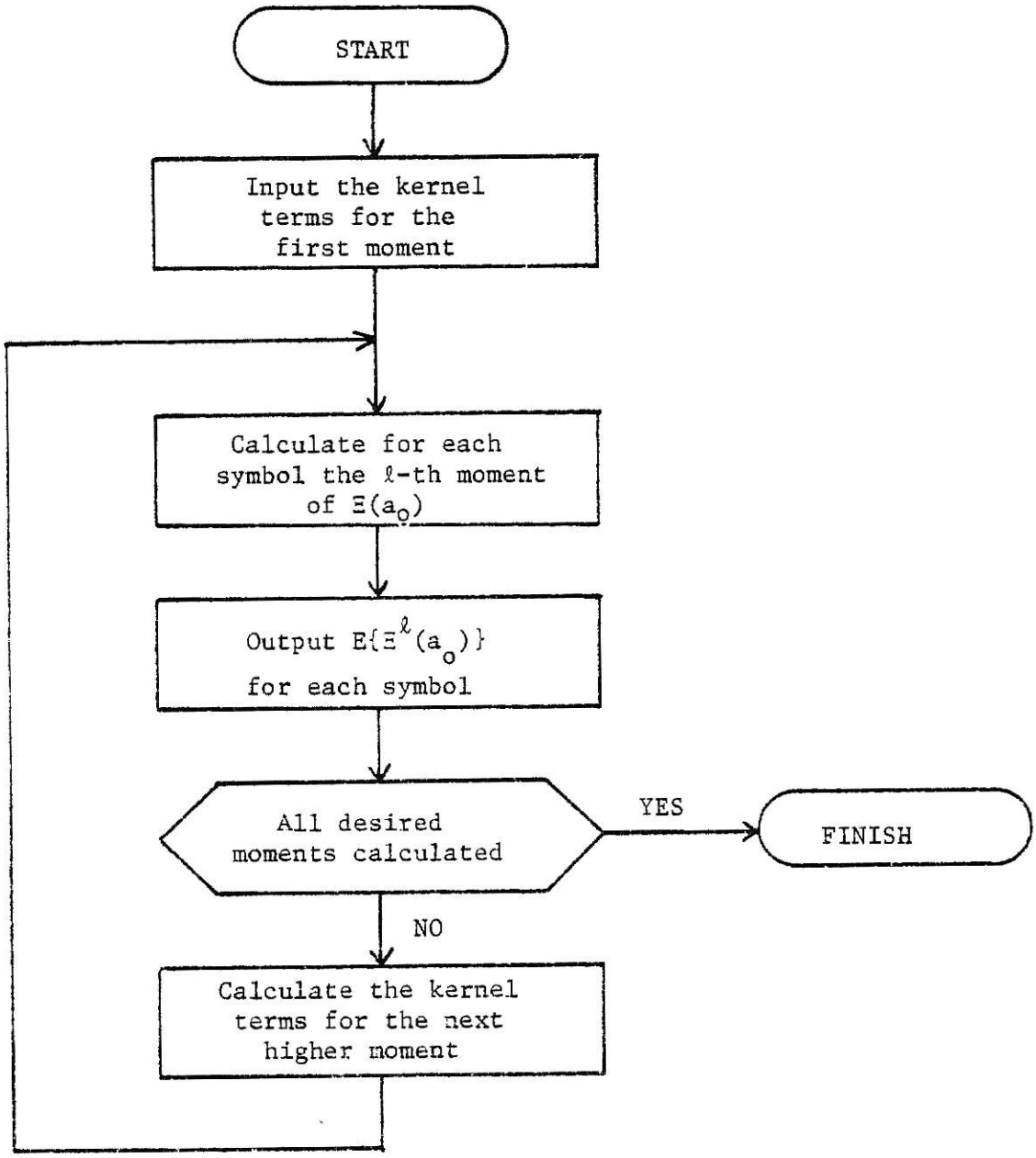

Figure 7. Expected Value Program

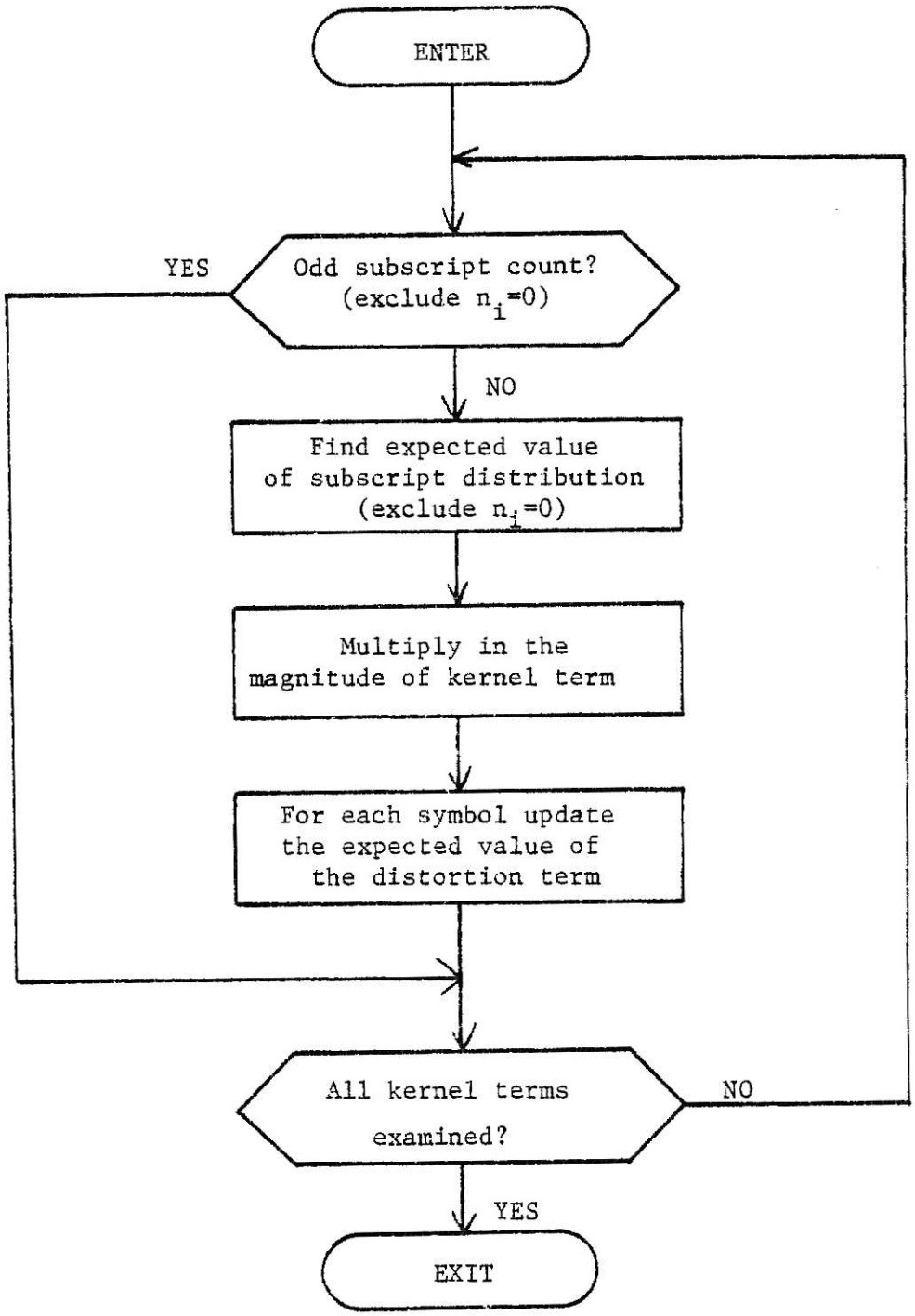

Figure 8. Calculating the Expected Value of the Distortion Terms

After checking for subscript having odd occurences, the expected value of the random data samples, $a_{n_1} a_{n_2} \ldots a_{n_k}$, is evaluated. Except for $n_i = 0$, each subscript count is used to evaluate (33). A close look at (33) shows that as the signal set grows so does the computation. Note however that for the same value of ℓ, (33) will yield the same result regardless of the data sample being considered. To eliminate this redundant computation, an upsilon table is calculated which contains all the needed values of (33). The count of how often a subscript, n_i, occurred is then used as an index to locate in the upsilon table the expected value of that data sample. The magnitude of the kernel term along with the results of calculating (33) for each subscript count are multiplied together.

As each symbol in the signal set is equally likely to occur at $n_i = 0$, the current signal interval, an expected value of the distortion is maintained for each symbol. Each kernel term which passes the subscript test is used to update all these expected values. That is

$$E\left\{\Xi^{NCM}_{new}(s_i)\right\} = E\left\{\Xi^{NCM}_{old}(s_i)\right\} + (s_i)^{\ell} o \left\{\begin{matrix}\text{kernel term's} \\ \text{contribution}\end{matrix}\right\} \quad . \tag{40}$$

Note that each symbol s_i is exponentiated using the $n_i = 0$ count for a given kernel term.

After all the kernel terms have been examined, the distortion term for each symbol for the current moment is printed out. If all the desired moments have not been considered, the kernel terms for the next higher moment are calculated.

As indicated in Figure 9, calculating the kernel terms for the next higher moment is very similar to the method used by Subroutine Kernel. First, the threshold delta is reduced by a factor of ten to increase

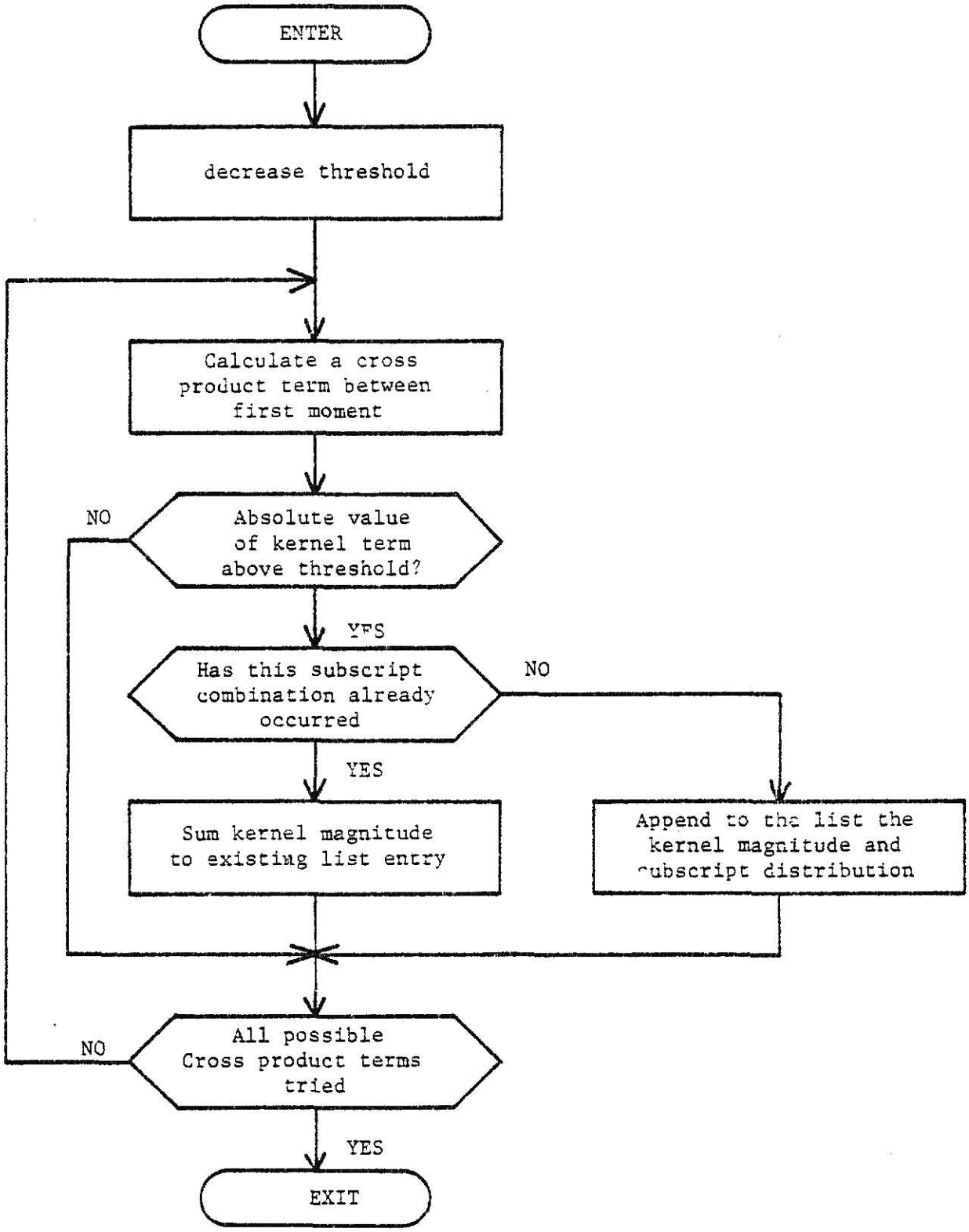

Figure 9. Calculating the Kernel Terms for the Next Higher Moment

accuracy. Next, the absolute value of each of all the possible products between the kernel terms of the first and previous moment are compared against the threshold. Terms falling below the threshold are discarded. Terms above the threshold are added to the new list of kernel terms in the same manner as described in Subroutine Kernel.

Gauss Quadrature Program

The Gauss Quadratrue Program uses the results from the Expected Value Program to obtain a GQR of N weights and abscissas for each symbol. The Fortran version appearing in the Appendix is a translation of an Algol program published by Golub and Welsh [7]. The Algol program utilizes the procedures GAUSSQUADRULE and GENORTHOPOLY. The Algol procedure GAUSSQUAD-RULE was found to contain an error which if not corrected always resulted in the weight vector equaling zero.† Also the Algol procedure GENORTHOPOLY was modified to indicate if the input moment vector failed the required positive definite condition.

Probability of Error Program

The results from the GQR program are used by the Probability of Error Program to calculate the probability of error vs. signal to noise ratio (SNR). The program implements (30) directly. Note that (27) is used as an approximation for the Q function. If the argument of the Q function falls below two in value then the algorithm outputs a warning indicating the approximation is no longer valid. The warning did not occur for any of the results presented in this report.

---

† w(1) was not set equal to one in the block labeled SETUP.

CHAPTER IV

EXPERIMENTAL RESULTS

The results documented in this report fall into two areas. First, a communication channel having filters with Gaussian impulse responses was used to validate the results published by Benedetto et al. [4]. The second area applied the Volterra approach to a channel characterized by causal impulse response filters.

Channel Filters Having Gaussian Impulse Responses

Previously it was shown that the Volterra kernels for the system depicted in Figure 2 can be analytically described by (10). In order to compare results with Benedetto, Biglieri, and Daffara [4],

$$h_{in}(t) = h_{out}(t) = \left(\frac{4\beta}{\pi T^2}\right)^{1/4} e^{-\frac{2\beta t^2}{T^2}} \tag{41}$$

were used as the impulse responses of the Gaussian filters. For a linear channel with unity gain, $\gamma_1 = 1$, substitution of (41) into (10) yields

$$h_1(\tau_1) = \left(\frac{4\beta}{\pi T^2}\right)^{1/2} \int_{-\infty}^{\infty} e^{-\frac{2\beta\tau^2}{T^2}} e^{-\frac{2\beta}{T^2}(\tau_1 - 2\tau_1\tau + \tau^2)} d\tau \ , \tag{42}$$

which can be written as

$$h_1(\tau_1) = \left(\frac{4\beta}{\pi T^2}\right)^{1/2} e^{-\frac{\beta\tau_1^2}{T^2}} \int_{-\infty}^{\infty} e^{-\frac{4\beta}{T^2}\left(\tau - \frac{\tau_1}{2}\right)^2} d\tau \ . \tag{43}$$

Examination of (43) reveals that the quantity inside the integral can be put in the form of a Gaussian distribution function which when integrated between plus and minus infinity will equal one. That is

$$h_1(\tau_1) = e^{-\frac{\beta\tau_1^2}{T^2}} \int_{-\infty}^{\infty} \frac{e^{-\frac{4\beta}{T^2}\left(\tau - \frac{\tau_1}{2}\right)^2}}{\sqrt{\frac{\pi T^2}{4\beta}}}\, d\tau$$

or

$$h_1(\tau_1) = e^{-\frac{\beta\tau_1^2}{T^2}} \quad . \tag{44}$$

Similarly, considering a third order nonlinearity, substitution of (41) into (10) yields

$$h_3(\tau_1,\tau_2,\tau_3) = \frac{\gamma_3}{3!} \int_{-\infty}^{\infty} \left(\frac{4\beta}{\pi T^2}\right)^{1/4} e^{-\frac{2\beta\tau^2}{T^2}} \cdot \prod_{r=1}^{3} \left(\frac{4\beta}{\pi T^2}\right)^{1/4} e^{-\frac{2\beta}{T^2}(\tau_r - \tau)^2} d\tau \quad , \tag{45}$$

which can be rewritten as

$$h_3(\tau_1,\tau_2,\tau_3) = \frac{\gamma_3}{3!}\left(\frac{4\beta}{\pi T^2}\right) \cdot \int_{-\infty}^{\infty} e^{-\frac{2\beta\tau^2}{T^2}} e^{-\frac{2\beta}{T^2}(\tau_1-\tau)^2} e^{-\frac{2\beta}{T^2}(\tau_2-\tau)^2} e^{-\frac{2\beta}{T^2}(\tau_3-\tau)^2} d\tau \, . \tag{46}$$

This can be simplified to

$$h_3(\tau_1,\tau_2,\tau_3) = \frac{\gamma_3}{3!}\left(\frac{4\beta}{\pi T^2}\right) e^{-\frac{2\beta}{T^2}(\tau_1^2+\tau_2^2+\tau_3^2)} e^{\frac{\beta}{2T^2}(\tau_1+\tau_2+\tau_3)^2} \cdot \int_{-\infty}^{\infty} e^{-\frac{8\beta}{T^2}\left[\tau - \frac{(\tau_1+\tau_2+\tau_3)}{4}\right]^2} d\tau \quad .$$

As with the linear case, the quantity under the integral sign can be expressed as a Gaussian distribution function. After a suitable

rearrangement of terms the preceding expression can be integrated to yield the result

$$h_3(\tau_1,\tau_2,\tau_3) = \frac{\gamma_3}{3!} \quad \frac{2\beta}{\pi T^2} \; e^{-\beta[(\tau_1^2+\tau_2^2+\tau_3^2) - \frac{1}{4}(\tau_1+\tau_2+\tau_3)^2]} \quad . \tag{47}$$

Recall that $\tau_i = t_o - n_i T$ where T is the signal interval and t_o is the sampling instant. For the results presented, T and t_o were set equal to one. The subscripts n_is which in (12) are run from minus to plus infinity would be impossible to implement. A truncated sequence from minus two to plus two was therefore used. The resulting impulse responses for two values of beta are shown in Figure 10. Visual observation confirms that the impulse responses have decayed sufficiently that the loss of terms caused by truncation does not affect the result.

The multiple summations appearing in (12) suggest that a large number of terms might be needed for an adequate channel description. Experimentation with the threshold showed that after a critical number of terms were retained, subsequent lowering of the threshold yielded no visual improvement of the error curve. Although varying from run to run, the number of terms retained for the last moment calculated was usually less than 300. On an Itel AS5 Model 3 computer, the runtime of the Expected Value Program was 5 minutes or less. Each application of the GQR and Probability of Error programs required less than a minute on a Data General Nova 1200 computer.

The results for three runs are presented in Figure 11. The error probability for each run is plotted versus the signal-to-noise ratio (SNR) defined as

$$\text{SNR} = 10 \log_{10} \frac{1}{\sigma_v^2} \quad . \tag{48}$$

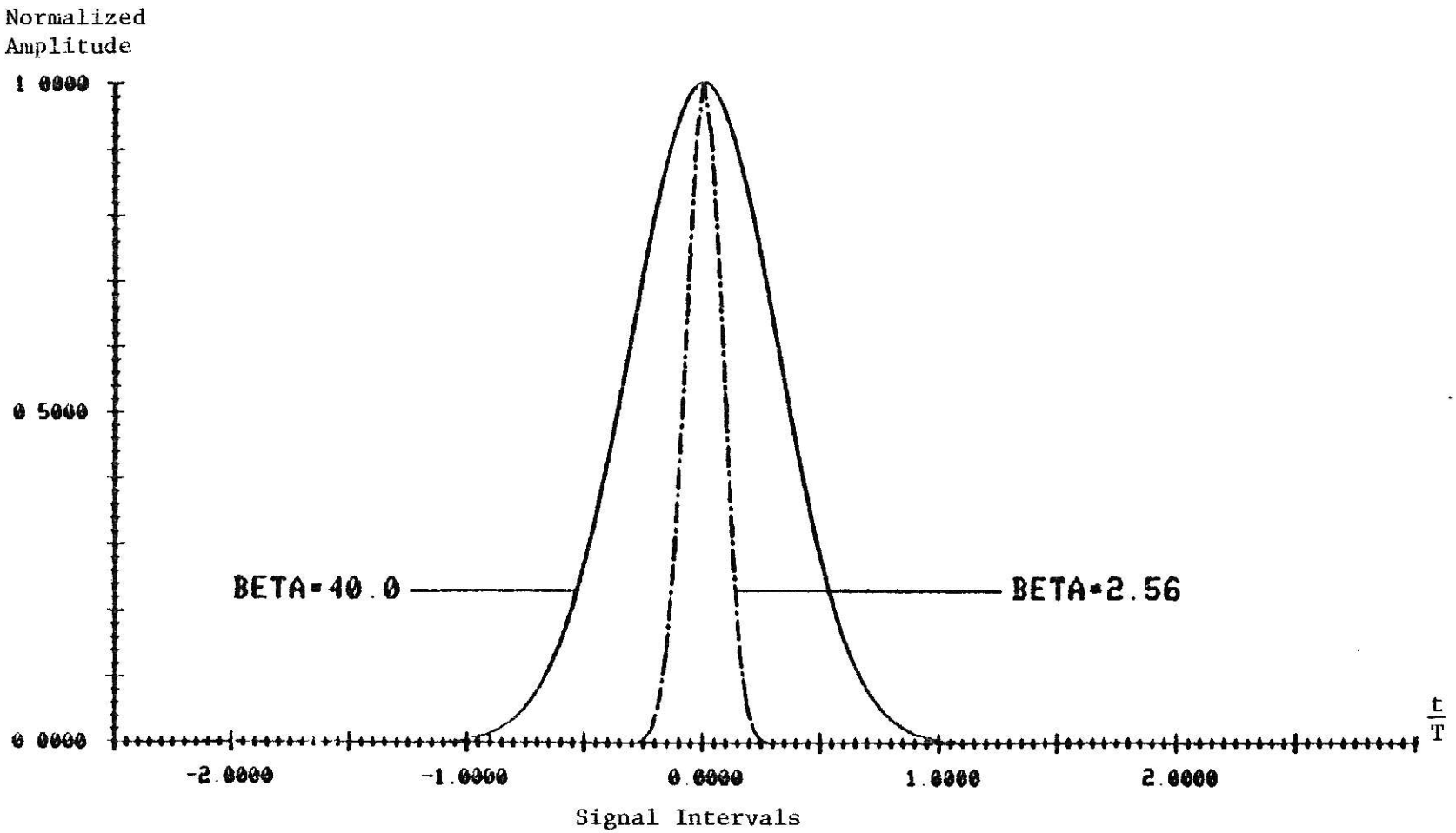

Figure 10. Truncated Impulse Responses of the Gaussian Filters

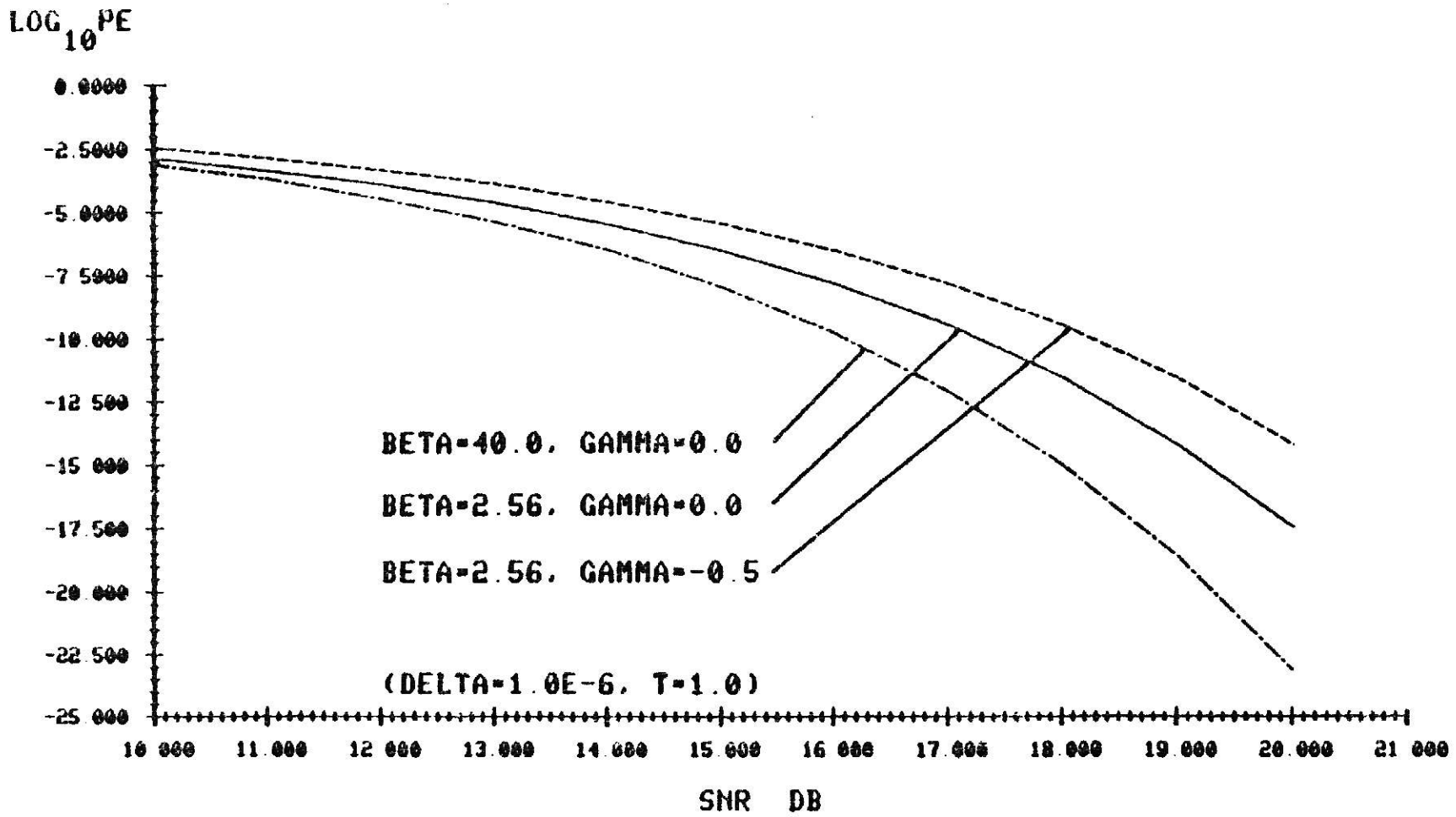

Figure 11. Results for a Communication Channel Having Filters with Gaussian Impulse Responses

The lowest probability of error occurred for the linear case $\beta = 40$. This result would be expected as a linear channel with wide band filters would add little distortion. Error probability for the linear channel cases were also calculated by direct enumeration. The results between the direct and the Volterra series approaches coincided to two or three decimal places.

The remaining two curves in Figure 11 closely match curves presented by Benedetto, Biglieri, and Daffara [4]. The linear case with $\beta = 2.56$ has a higher probability of error than the linear case with $\beta = 40.0$. This is as expected since decreasing the bandwidth of the channel filters causes intersymbol interference to increase. The negative nonlinear case with $\beta = 2.56$ had the surprising result of improving the performance of the channel. A positive nonlinearity, however, does degrade the channel performance.

Channel Filters with Causal Impulse Responses

While Gaussian filters and impulsive inputs are convenient to work with mathematically, they suffer from the disadvantage that they cannot be physically constructed. Therefore the Volterra approach was applied next to a communication channel having causal filters. By including a rectangular pulse shaping filter as part of $h_{in}(t)$ in Figure 2, a multi-level rectangular pulse input signal can be described. Although the mathematics becomes a bit more messy, the basic approach of calculating the probability of error remains unchanged.

The first step in calculating the first moment kernel terms is to specify the impulse responses $h_{in}(t)$ and $h_{out}(t)$ which characterize the

Volterra system. For example, consider a communication channel having single pole low-pass filters on either side of a non memory nonlinearity. As mentioned previously $h_{in}(t)$ will include the response of a pulse shaping filter to convert the input sequence of impulses to a sequence of rectangular pulses. Thus $h_{in}(t)$ can be considered to be two distinct filters connected in cascade as shown in Figure 12. The impulse response of $h_{in}(t)$ is the convolution of the impulse responses of pulse shaping filter with that of the low-pass filter. That is

$$h_{in}(t) = \int_{-\infty}^{\infty} h(\tau)\, p(t-\tau)\, d\tau \quad , \tag{49}$$

where

$$p(t) = \begin{cases} 1 & \text{for} \quad 0 \leq t \leq T \\ 0 & \text{elsewhere} \end{cases}$$

$$h(t) = \begin{cases} \alpha e^{-\alpha t} & \text{for} \quad 0 \leq t \\ 0 & \text{elsewhere} \end{cases} \quad . \tag{50}$$

Due to the discontinuity in $p(t)$, $h_{in}(t)$ must be integrated in two parts to yield the result

$$h_{in}(t) = \begin{cases} 1 - e^{-\alpha t} & \text{for} \quad 0 \leq t < T \\ (e^{\alpha t} - 1)\, e^{-\alpha t} & \text{for} \quad T \leq t \\ 0 & \text{elsewhere} \end{cases} \quad . \tag{51}$$

The second single pole low-pass filter of arbitrary bandwidth is

$$h_{out}(t) = \begin{cases} \beta\, e^{-\beta T} & \text{for} \quad 0 \leq t \\ 0 & \text{elsewhere} \end{cases} \quad . \tag{52}$$

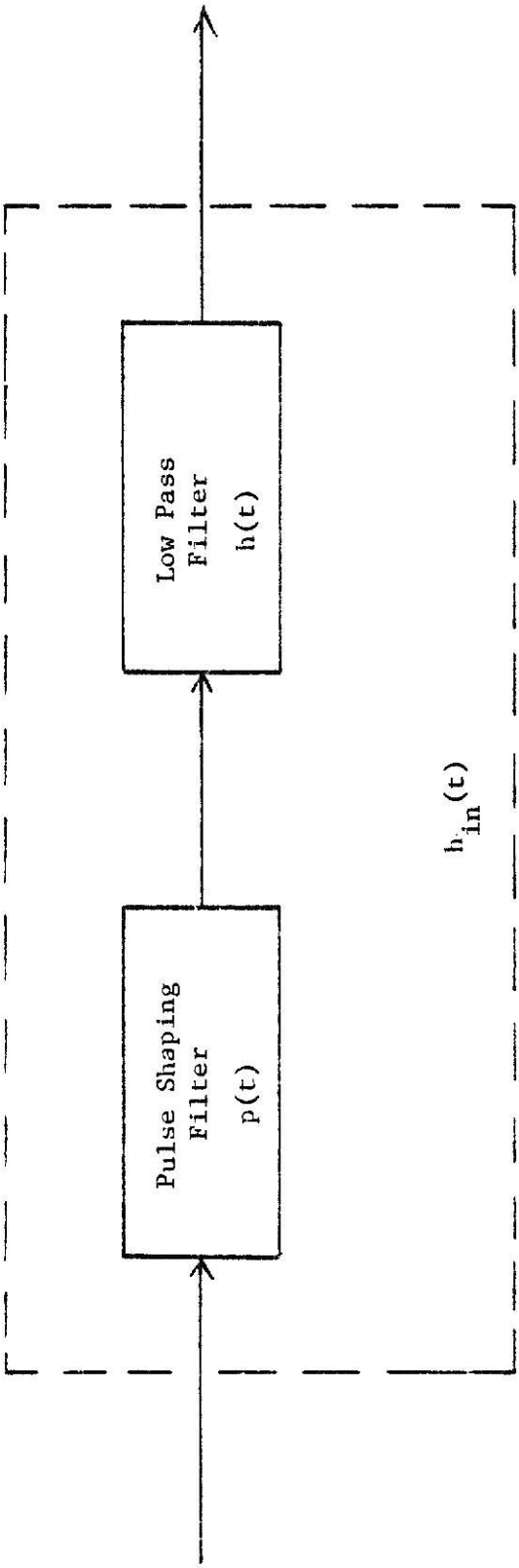

Figure 12. First Channel Filter

Two facts emerge when examining $h_{in}(t)$ and $h_{out}(t)$ shown in Figure 13. First the causal impulse responses limit the integration of (1) from zero to some value $\tau_{r_{min}}$. Second, due to the discontinuity in $h_{in}(t)$ there are two cases, $\tau_{r_{min}} < T$ and $\tau_{r_{min}} \geq T$, to consider. Figure 14 shows a graphical presentation of these two cases. Since the sampling instant t_o is set equal to T in the results which follow, the second case will be presented. Substitution of the impulse responses (51) and (50) into (10) yields

$$h_k(\tau_1,\tau_2,\ldots,\tau_3) = \frac{\gamma_k \beta}{k!} \left\{ \int_0^{\tau_{r_{min}}-T} e^{-\beta\tau} \prod_{j=0}^{k} (e^{\alpha T} - 1)\, e^{-\alpha(\tau_j-\tau)}\, d\tau \right.$$

$$+ \int_{\tau_{r_{min}}-T}^{\tau_{r_{min}}} e^{-\beta\tau} \left[1-e^{-\alpha(\tau_{r_{min}}-\tau)}\right]^n$$

$$\left. \cdot \prod_{\substack{p=1 \\ p\neq r_{min}}}^{k-m} (e^{\alpha T} - 1)\, e^{-\alpha(\tau_p-\tau)}\, d\tau \right\} , \qquad (53)$$

where the index m indicates the number of τ_{n_i} subscripts coinciding with $\tau_{r_{min}}$. Let each of the two integrals in (53) be considered separately. That is, let

$$h_k(\tau_1,\tau_2,\ldots,\tau_k) = \frac{\gamma_k \beta}{k!} \{I_1 + I_2\} \quad . \qquad (54)$$

The integral I_1 is

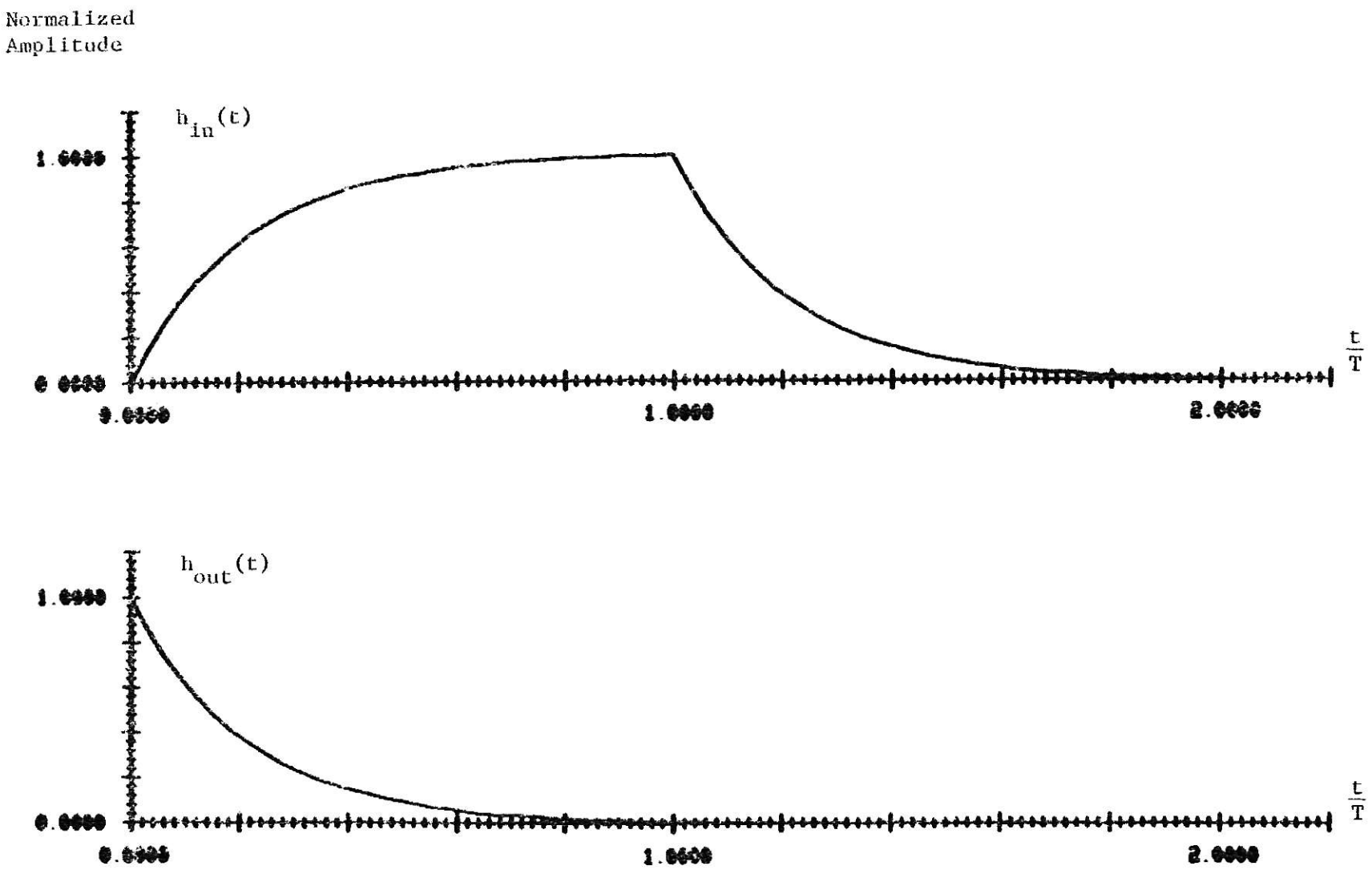

Figure 13. Impulse Responses of the Causal Filters

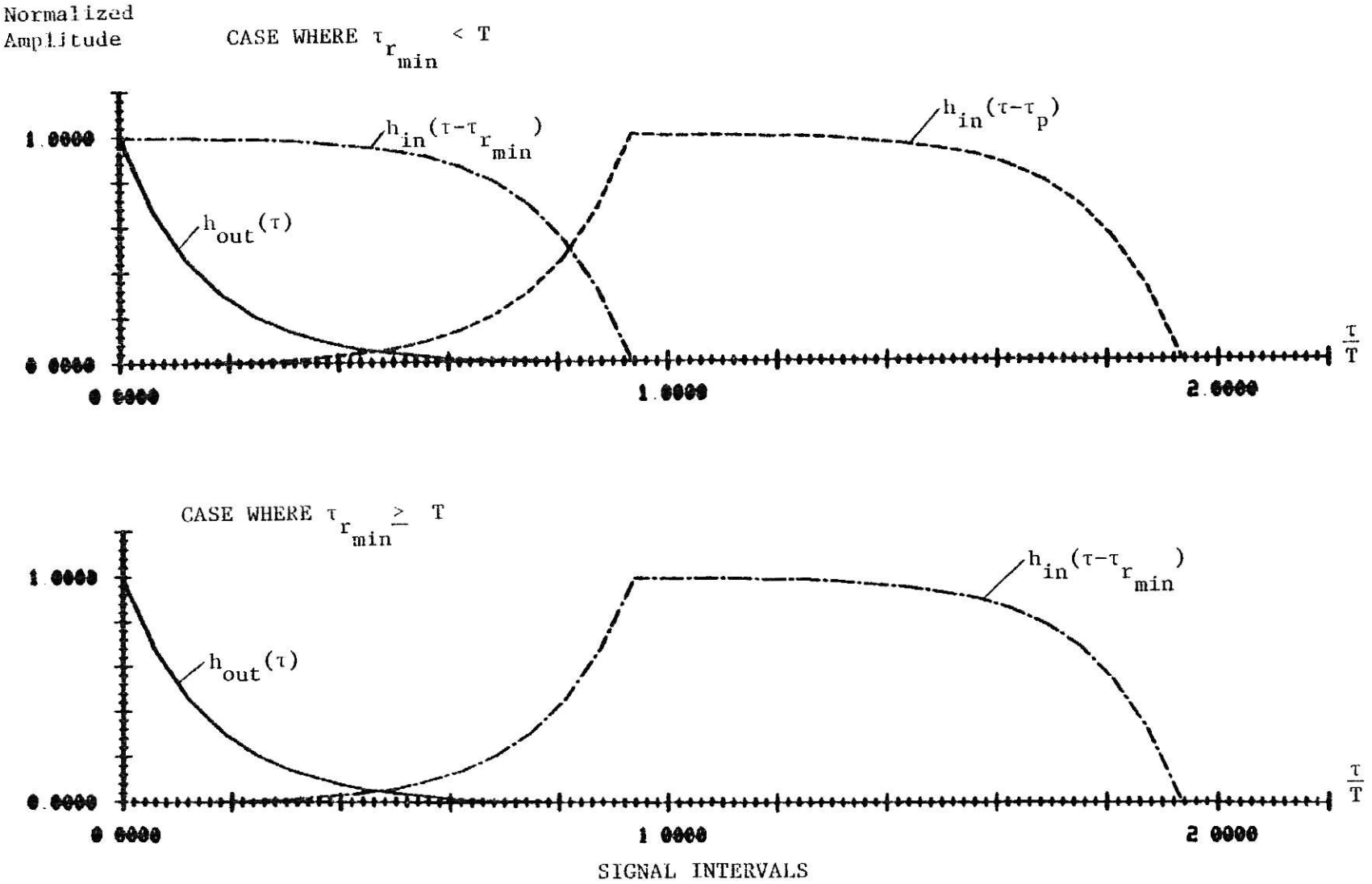

Figure 14. Convolution of Impulse Responses of Causal Filters

$$I_1 = \int_0^{\tau_{r_{min}}-T} e^{-\beta\tau} \prod_{j=1}^{k} (e^{\alpha T} - 1)\, e^{-\alpha(\tau_j - \tau)}\, d\tau$$

$$= (e^{\alpha T} - 1)^k\, e^{-\alpha \sum_{j-1}^{k} \tau_j} \int_0^{\tau_{r_{min}}-T} e^{(k\alpha-\beta)\tau}\, d\tau \quad , \tag{55}$$

which after integration yields

$$I_1 = \frac{(e^{\alpha T}-1)^k\, e^{-\alpha \sum_{j=1}^{k} \tau_j}}{(k\alpha - \beta)} \left[e^{(k\alpha-\beta)\tau_{r_{min}}}\, e^{-(k\alpha-\beta)T} - 1 \right] . \tag{56}$$

Note that I_1 is part of the final result only if $\tau_{r_{min}} > T$.

The second integral I_2 is

$$I_2 = \int_{\tau_{r_{min}}-T}^{\tau_{r_{min}}} e^{-\beta\tau} \left[1 - e^{-\alpha(\tau_{r_{min}} - \tau)} \right]^m \prod_{\substack{p=1 \\ p \neq r_{min}}}^{k-m} (e^{\alpha T} - 1)\, e^{-\alpha(\tau_p - \tau)}\, d\tau \quad . \tag{57}$$

In order to proceed further, it is convenient to make use of the Binomial Theorem which states

$$(a-b)^m = a^m - \frac{m a^{m-1} b}{1!} + \frac{m(m-1) a^{m-2} b^2}{2!} + \ldots + \frac{(-1)^{m-1} m a b^{m-1}}{1!} + (-1)^m b^m$$

$$= \sum_{v=o}^{m} \frac{(-1)^v\, m!\, a^{m-v}\, b^v}{v!\, (m-v)!} \quad . \tag{58}$$

By letting $a = 1$ and $b = -e^{-\alpha(\tau_{r_{min}} - T)}$ observe that

$$\left[1 - e^{-\alpha(\tau_{r_{min}} - \tau)}\right]^m = \sum_{v=o}^{m} \frac{(-1)^v\, m!}{v!\ (m-v)!}\, e^{-v\alpha(\tau_{r_{min}} - \tau)} \quad . \tag{59}$$

By making use of (59), (57) can be restated to give

$$I_2 = (e^{\alpha T} - 1)^{k-m}\, e^{-\alpha \sum_{\substack{p=1 \\ p\neq r_{min}}}^{k-m} \tau_p} \int_{\tau_{r_{min}} - T}^{\tau_{r_{min}}} e^{-\beta\tau}\, e^{(k-m)d\tau}$$

$$\cdot \sum_{v=o}^{m} \frac{(-1)^v\, m!\, e^{-v\alpha(\tau_{r_{min}} - \tau)}}{v!\ (m-v)!}\, d\tau \quad . \tag{60}$$

Interchanging the order of the summation and integration yields

$$I_2 = (e^{\alpha T} - 1)^{k-m}\, e^{-\alpha \sum_{\substack{p=1 \\ p\neq r_{min}}}^{k-m} \tau_p} \sum_{v=o}^{m} \frac{(-1)^v\, m!\, e^{-v\alpha\tau_{r_{min}}}}{v!\ (m-v)!}$$

$$\cdot \int_{\tau_{r_{min}} - T}^{\tau_{r_{min}}} e^{[\alpha(k-m+v) - \beta]\tau}\, d\tau \quad . \tag{61}$$

Carrying out the integration yields

$$I_2 = (e^{\alpha T} - 1)^{k-m}\, e^{-\alpha \sum_{\substack{p=1 \\ p\neq r_{min}}}^{k-m} \tau_p}$$

$$\cdot \left\{ \sum_{v=o}^{m} \frac{(-1)^v\, m!\, e^{[\alpha(k-m)-\beta]\tau_{r_{min}}} \left[1 - e^{-[\alpha(k-m+v)-\beta]T}\right]}{v!\ (m-v)!\ [\alpha(k-m+v)-\beta]} \right\} \quad . \tag{62}$$

Substitution for I_1 and I_2 into (54) yields the final result

$$h_k(\tau_1,\tau_2,\ldots,\tau_k) = \frac{\gamma_k^{\beta}}{k!}\left\{\underbrace{\frac{(e^{\alpha T}-1)^k\, e^{-\alpha\sum_{j=1}^{k}\tau_j}}{(k\alpha-\beta)}\left[e^{(k\alpha-\beta)(\tau_{r_{min}}-T)}-1\right]}_{\text{term present only if } \tau_{r_{min}} > T}\right.$$

$$\left. + (e^{\alpha T}-1)^{k-m}\, e^{-\alpha\sum_{\substack{p=1\\ p\neq r_{min}}}^{k-m}} e^{[\alpha(k-m)-\beta]\tau_{r_{min}}} \cdot \left\{\sum_{v=o}^{m}\frac{(-1)^v\, m!}{v!\,(m-v)!\,[\alpha(k-m+v)-\beta]}\left[1-e^{-[\alpha(k-m+v)-\beta]T}\right]\right\}\right\} \tag{63}$$

for the case $\tau_{r_{min}} \geq T$.

Presented for completeness is

$$h_k(\tau_1,\tau_2,\tau_3) = \frac{\gamma_k^{\beta}}{k!}(e^{\alpha T}-1)^{k-m}\, e^{-\alpha\sum_{\substack{p=1\\ p\neq r_{min}}}^{k-m}\tau_p} e^{[\alpha(k-m)-\beta]\tau_{r_{min}}} \cdot \left\{\sum_{v=o}^{m}\frac{(-1)^v\, m!}{v!\,(m-v)!\,[\alpha(k-m+v)-\beta]}\left[1-e^{-[\alpha(k-m+v)-\beta]\tau_{r_{min}}}\right]\right\} \tag{64}$$

for the case $\tau_{r_{min}} < T$. Note there are combinations of values which sometimes yield the indeterminate result of zero over zero. The correct result is obtained by invoking l'Hopital's Rule.

The results which follow are for linear and third order nonlinear channels. SNR is defined by (48). The bit time T and the sampling instant t_o were set equal to one to give the receiver the maximum amount

of time to make the decision which symbol is present. The linear channel with no intersymbol interference was adjusted for unity gain. To make the computation of (12) possible, the n_i subscripts were truncated to a finite set. With causal filters it is not necessary to consider sampling intervals which have yet to occur. Therefore depending upon the threshold chosen, the present and up to four previous signal intervals were taken into account.

The threshold was adjusted to keep a sufficient number of terms for accuracy without requiring excessive computer time. Experimentation with the threshold yielded results behaving like those obtained for a channel having Gaussian filters. For example, varying the threshold from 10^{-4} to 10^{-10} for the run $\alpha = \beta = 2\pi$ and $\gamma_3 = -.5$ showed no - significant change in the final error probability. For all the realizable filter runs, the highest moment calculated required that less than 300 terms be retained. The runtime for all programs was under a minute.

Figure 15 shows the results of varying the bandwidth of a linear channel. The best performance was obtained using the wideband filter case of $\alpha = \beta = 20$. Decreasing the bandwidth of the filters had the effect of progressively degrading performance at high SNR. This would be expected because at high SNR the distortion caused by additive noise is small in relation to the distortion caused by intersymbol interference. If a constant nonlinearity is added to the channel, the narrower bandwidths show a loss in performance as shown in Figure 16. Varying the amount of third order nonlinearity with a constant bandwidth yields the curves in Figure 17. It is interesting to note that, like the Gaussian results for the case of Gaussian filters obtained previously, a positive third order nonlinearity improves performance.

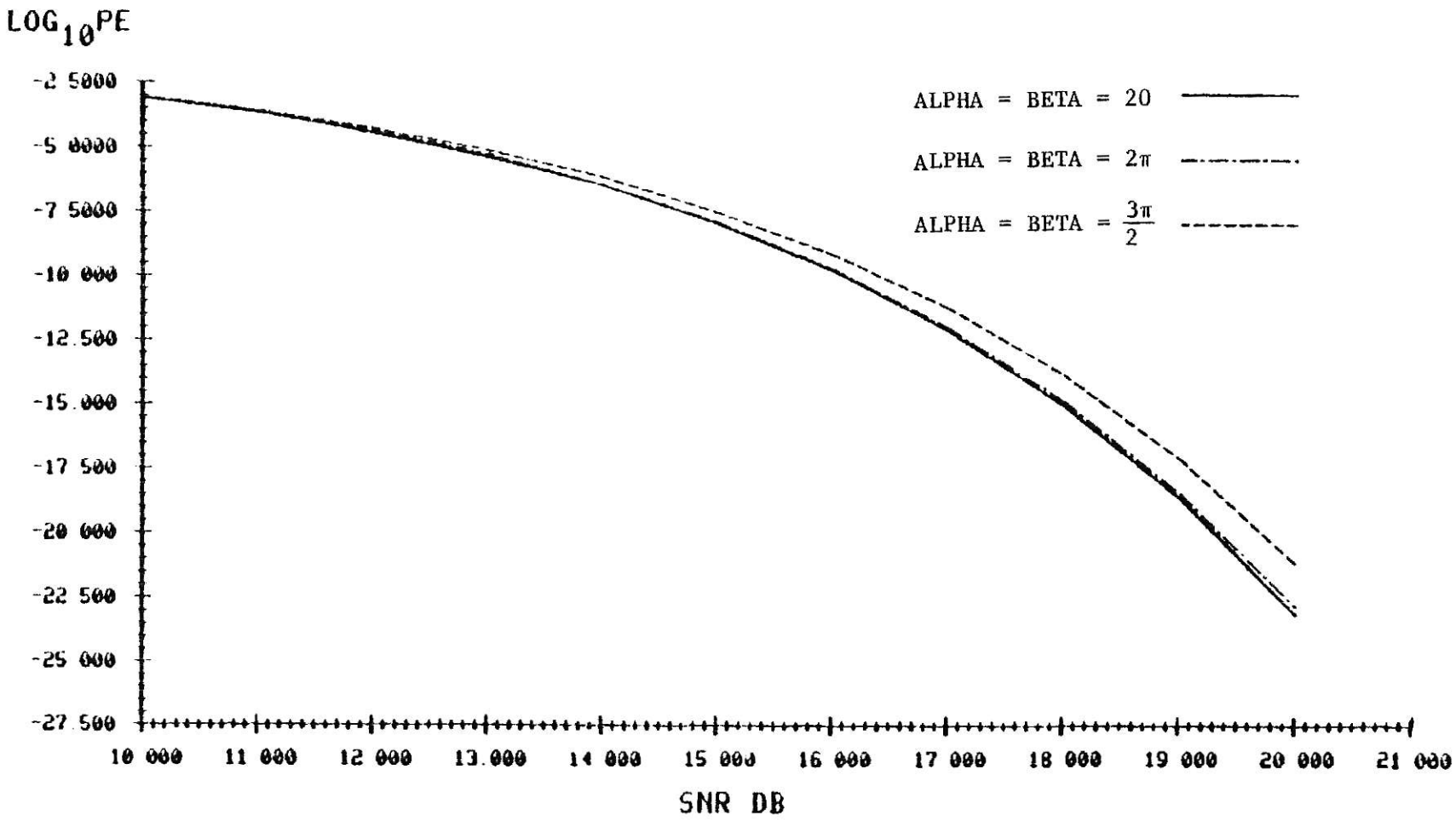

Figure 15. Varying the Bandwidth in a Linear Communication Channel

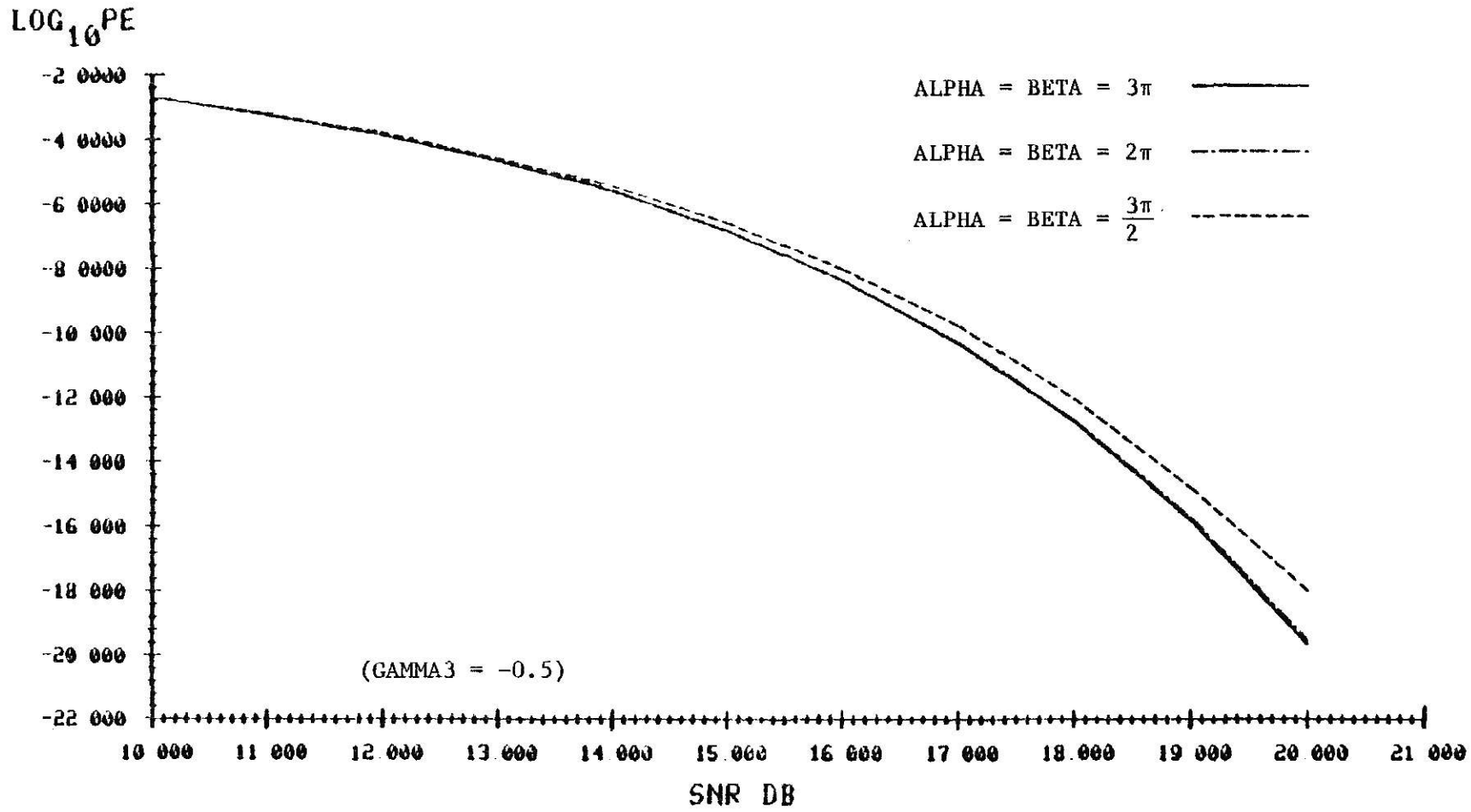

Figure 16. Varying the Bandwidth in a Nonlinear Communication Channel

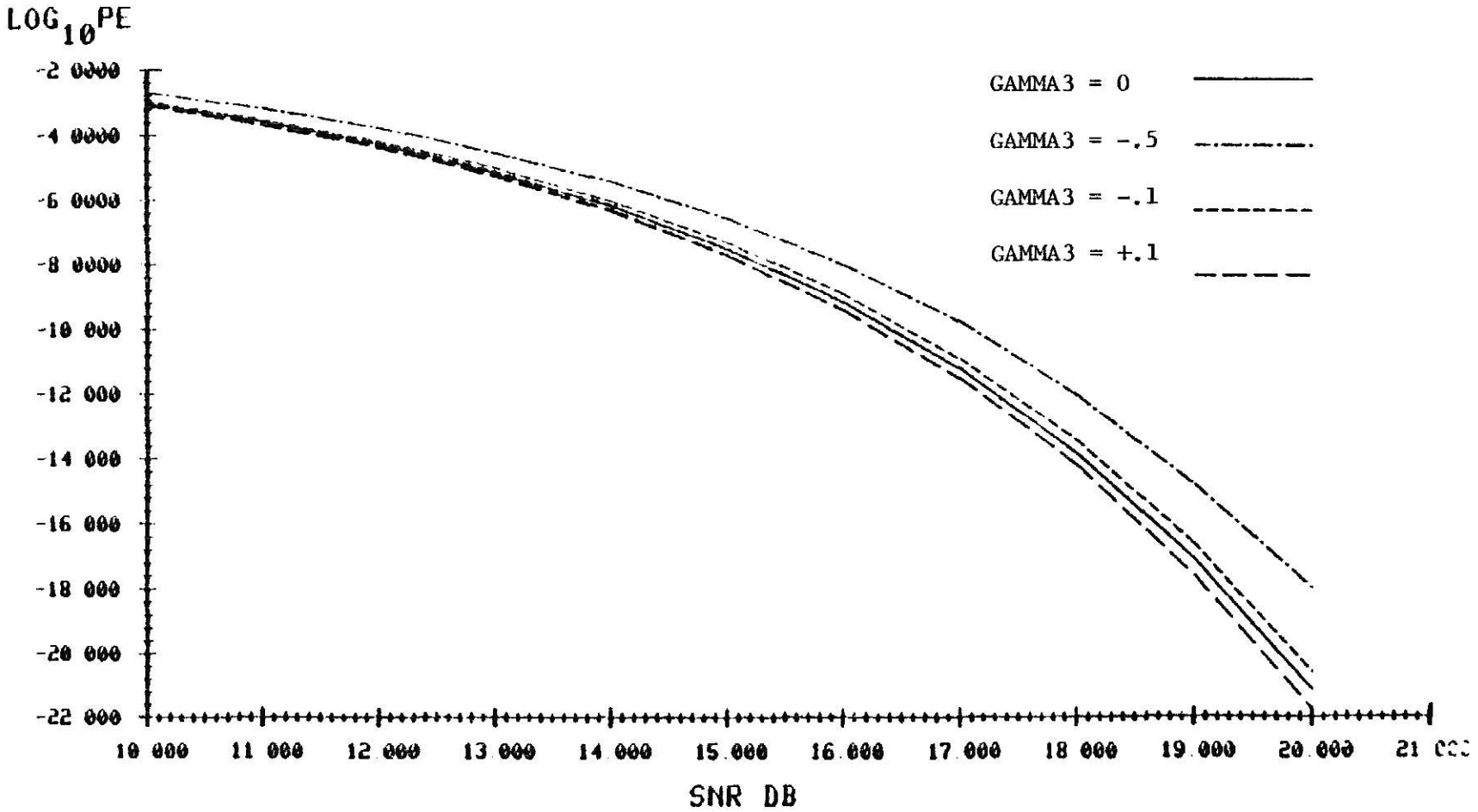

Figure 17. Varying the Nonlinearity in a Constant Bandwidth Communication Channel

CHAPTER V

CONCLUSIONS

The principal effort of this report was the development of the Volterra series approach for calculating the error probability in nonlinear communication channels. The use of the Volterra model was shown to have a firm mathematical foundation. A series of computer programs implementing the analytical analysis were then presented. The algorithms implemented were validated against work by Benedetto, Biglieri, and Daffara [4] in modeling a channel having filters with Gaussian impulse responses. Subsequent work applied the Volterra approach to practical channels by modeling filters having causal impulse responses. To enable the system model to accept real world inputs, pulse shaping characteristics were added to the first channel filter.

The Volterra series method of modeling nonlinear channels can be extended in many directions. Moving the noise source before the nonlinearity is one example. Other areas include introducing correlation between signaling intervals and the ability to express the impulse responses of the channel filters in terms of their power density spectrums.

CHAPTER VI

REFERENCES

[1] S. Benedetto, E. Biglieri and R. Daffar, "Modeling and Performance Evaluation of Nonlinear Satellite Links--A Volterra Series Approach," Istituto di Elettronica e Telecommunicazion Politecnico--Corso Duca d'Abruzzi 24, Torino, Italy, Paper in review for IEEE Transactions on Aerospace and Electronics Systems.

[2] R. E. Maurer and S. Narayanan, "Noise Loading Analysis of a Third-Order Nonlinear System with Memory," IEEE Transactions on Communication Technology, Vol. COM-16, No. 5, October 1968.

[3] S. Narayanan, "Transistor Distortion Analysis using Volterra Series Representation," The Bell System Technical Journal, pp. 991-1024, May-June 1967.

[4] S. Benedetto, E. Biglieri, and R. Daffara, "Performance of Multilevel Baseband Digital Systems in a Nonlinear Environment," IEEE Transactions on Communications, Vol. COM-24, No. 10, October 1976.

[5] D. A. George, "Continuous Nonlinear Systems," M.I.T. Research Lab of Electronics, Cambridge, Mass., Technical Report 355, July 24, 1959.

[6] J. M. Wozencraft and I. M. Jacobs, "Principles of Communication Engineering," John Wiley & Sons, Inc., New York, N.Y., 1965.

[7] G. H. Golub and J. H. Welsch, "Calculation of Gauss Quadrature Rules," Journal of Mathematical Computation, Vol. 23, pp. 221-230, April 1969.

[8] A. Mircea and H. Sinnreich, "Distortion noise in frequency-dependent nonlinear networks," Proceedings of the Institution of Electrical Engineers, Vol. 116, No. 10, October 1969.

[9] E. Bedrosian and S. O. Rice, "The Output Properties of Volterra Systems (Nonlinear Systems with Memory) Driven by Harmonic and Gaussian Inputs," Proceedings of the IEEE, Vol. 59, No. 12, December 1971.

ACKNOWLEDGEMENTS

I wish to express my appreciation for the support afforded me by my parents and particularly that from the members of my graduate committee, Dr. D. R. Hummels, Dr. N. Ahmed, and Dr. L. Fuller, for without their assistance and support this report would not have been possible.

APPENDIX

COMPUTER PROGRAMS

```
C       * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
C
C       FRED64.FR           EXPECTED VALUE PROGRAM
C
C       KANSAS STATE UNVERSITY
C
C       F W RATCLIFFE       FEB 27, 79     REV:  APR 22, 79
C
C       * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
C
C       THIS PROGAM CALCULATES THE EXPECTED VALUES FOR THE MOMENT(S)
C       OF THE VOLTERRA KERNEL(S) USED TO DESCRIBE A NONLINEAR
C       DIGITAL COMMUNICATION CHANNEL
C
C       PROGRAM REQUIRES SUBROUTINE(S)
C               KERNEL
C
C       NOTE:  FOR  GAUSSIAN  FILTERS ACTIVATE LINES HAVING A GAUSS ID
C              FOR REALIZABLE FILTERS ACTIVATE LINES HAVING A REAL  ID
C
C       * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
C
        DIMENSION  E(8), H1(100), HA(1000), HB(1000), S(8), U(64)
        INTEGER  IS(8), L1(100), LA(1000), LB(1000)
C
1       FORMAT ('1EXPECTED VALUE PROGRAM')
2       FORMAT (F30.20)
3       FORMAT ('0      DELTA = ', G15.6)
4       FORMAT ('       MAIN ROUTINE ERROR => DELTA NOT GREATER THAN',
     1           ' OR EQUAL TO ZERO')
5       FORMAT (I2)
6       FORMAT ('0      DESIRED NUMBER OF MOMENTS = ', I3)
7       FORMAT ('       MAIN ROUTINE ERROR => NM OUT OF THE RANGE (1-9)')
CB      FORMAT ('0      THE CHANNEL RANGES FROM ', I2, ' TO ', I1)        GAUSS
8       FORMAT ('0      THE CHANNEL RANGES FROM ', I2, ' TO ZERO')        REAL
C9      FORMAT ('       MAIN ROUTINE ERROR => N  OUT OF THE RANGE (1-2)')GAUSS
9       FORMAT ('       MAIN ROUTINE ERROR => N  OUT OF THE RANGE (1-5)')REAL
10      FORMAT ('       MAIN ROUTINE ERROR => NS OUT OF THE RANGE (2-8)')
11      FORMAT ('       MAIN ROUTINE ERROR => NS NOT EVEN')
12      FORMAT ('0EXPECTED VALUES')
13      FORMAT ('0MOMENT # ', I3)
14      FORMAT ('0      HA HAS ', I5, ' TERMS', /)
15      FORMAT ('       E(', I3, ') = ', G15.6)
C
C       SET UP KNOWN PARAMETER(S) AND OUTPUT PROGRAM TITLE
C
        DIVIDE = 10.0
        IBASE = 2 ** 5
        ITMAX = 64
        ITTI = 5
        ITTO = 6
C       MAXN = 2                                                          GAUSS
        MAXN = 5                                                          REAL
        NCM = 1
        NMMAX = 9
        NSMAX = 8
        WRITE (ITTO, 1)
C
C       INPUT THE VALUE FOR DELTA
C
        READ  (ITTI, 2)  DELTA
```

```
      WRITE (ITTO, 3)  DELTA
      IF (DELTA.LT.0.0)  WRITE (ITTO, 4)
      IF (DELTA.LT.0.0)  STOP
C
C     INPUT THE DESIRED NUMBER OF MOMENT(S)
C
      READ  (ITTI, 5)  NM
      WRITE (ITTO, 6)  NM
      IF (NM.LT.1.OR.NM.GT.NMMAX)  WRITE (ITTO, 7)
      IF (NM.LT.1.OR.NM.GT.NMMAX)  STOP
C
C     INPUT THE NUMBER OF CHANNEL SAMPLE(S)
C
      READ  (ITTI, 5)  N
C     MN = - N                                                    GAUSS
C     WRITE (ITTO, 8)  MN, N                                      GAUSS
      MNP1 = - N + 1                                              REAL
      WRITE (ITTO, 8)  MNP1                                       REAL
      IF (N.LT.1.OR.N.GT.MAXN)  WRITE (ITTO, 9)
      IF (N.LT.1.OR.N.GT.MAXN)  STOP
C     NP1 = N + 1                                                 GAUSS
C     NP2 = N + 2                                                 GAUSS
C     NT2P1 = N * 2 + 1                                           GAUSS
C
C     INPUT THE NUMBER OF SYMBOLS
C
      READ  (ITTI, 5)  NS
      IF (NS.LT.2.OR.NS.GT.NSMAX)  WRITE (ITTO, 10)
      IF (NS.LT.2.OR.NS.GT.NSMAX)  STOP
      NSTEST = NS / 2 * 2
      IF (NS.NE.NSTEST)  WRITE (ITTO, 11)
      IF (NS.NE.NSTEST)  STOP
      FNS = FLOAT(NS)
C
C     CALCULATE THE SYMBOL TABLES
C
          DO 125  I = 1, NS
          IS(I) = - NS - 1 + 2 * I
          S(I) = FLOAT(IS(I))
125       CONTINUE
C
C     CALCULATE THE UPSILON TABLE
C
          DO 175  I = 1, ITMAX, 2
          SUM = 0.0
              DO 150  J = 1, NS
              SUM = SUM + S(J) ** (I - 1)
150           CONTINUE
C
          U(I) = SUM / FNS
          U(I + 1) = 0.0
175       CONTINUE
C
C     CALCULATE THE FIRST MOMENT KERNEL TERM(S)
C
      CALL KERNEL (H1, L1, DELTA, IBASE, ITTI, ITTO, N, M1)
      WRITE (ITTO, 12)
C
C     COPY THE FIRST MOMENT KERNEL TERM(S) INTO THE A ARRAYS
C
      MA = M1
```

```
            DO 225  I = 1, MA
            HA(I) = H1(I)
            LA(I) = L1(I)
225         CONTINUE
235     CONTINUE
C
C       ZERO OUT THE EXPECTED VALUE ARRAY
C
            DO 250  I = 1, NS
            E(I) = 0.0
250         CONTINUE
C
C       MAJOR LOOP ENTERED ONCE FOR EACH TERM IN THE HA ARRAY
C
            DO 350  I = 1, MA
C
C           EXCLUDING A0, CHECK FOR AN ODD NUMBER OF SUBSCRIPTS
C
C           LTEMP = LA(I)                                              GAUSS
            LTEMP = LA(I) / IBASE                                      REAL
C               DO 275  J = 1, N                                       GAUSS
                DO 275  J = 2, N                                       REAL
                ICK = LTEMP - LTEMP / IBASE * IBASE
                JCK = ICK / 2 * 2
                IF (ICK.NE.JCK)  GO TO 350
                LTEMP = LTEMP / IBASE
275             CONTINUE
C
C           LTEMP = LTEMP / IBASE                                      GAUSS
C               DO 280  J = NP2, NT2P1                                 GAUSS
C               ICK = LTEMP - LTEMP / IBASE * IBASE                    GAUSS
C               JCK = ICK / 2 * 2                                      GAUSS
C               IF (ICK.NE.JCK)  GO TO 350                             GAUSS
C               LTEMP = LTEMP / IBASE                                  GAUSS
C280            CONTINUE                                               GAUSS
C
C           EXCLUDING A0, CALCULATE EACH SUBSCRIPT'S CONTRIBUTION
C
            LTEMP = LA(I)
            LA0 = LTEMP - LTEMP / IBASE * IBASE                        REAL
            LTEMP = LTEMP / IBASE                                      REAL
            TIMES = 1.0
C               DO 300  J = 1, N                                       GAUSS
                DO 300  J = 2, N                                       REAL
                ICK = LTEMP - LTEMP / IBASE * IBASE + 1
                TIMES = TIMES * U(ICK)
                LTEMP = LTEMP / IBASE
300             CONTINUE
C
C           LA0 = LTEMP - LTEMP / IBASE * IBASE                        GAUSS
C           LTEMP = LTEMP / IBASE                                      GAUSS
C               DO 310  J = NP2, NT2P1                                 GAUSS
C               ICK = LTEMP - LTEMP / IBASE * IBASE + 1                GAUSS
C               TIMES = TIMES * U(ICK)                                 GAUSS
C               LTEMP = LTEMP / IBASE                                  GAUSS
C310            CONTINUE                                               GAUSS
C
C           ADD THE EFFECT OF THE KERNEL MAGNITUDE
C
            TIMES = TIMES * HA(I)
C
```

```
C           CALCULATE THE CONTRIBURION TO EACH SYMBOL'S
C           EXPECTED VALUE RY CURRENT HA TERM
C
                DO 325  J = 1, NS
                E(J) = E(J) + TIMES * S(J) ** LAO
325             CONTINUE
350         CONTINUE
C
C       OUTPUT EXPECTED VALUE ARRAY TO THE LINE PRINTER
C
        WRITE (ITTO, 13)  NCM
        WRITE (ITTO, 14)  MA
            DO 375  I = 1, NS
            WRITE (ITTO, 15)  IS(I), E(I)
375         CONTINUE
C
C       CHECK TO SEE IF WE ARE DONE
C
        NCM = NCM + 1
        IF (NCM.GT.NM)  STOP
C
C       DECREASE DELTA TO INCREASE ACCURACY
C
        DELTA = DELTA / DIVIDE
C
C       CALCULATE CROSS PRODUCT TERM(S) BETWEEN H1 AND HA ARRAYS
C
        MB = 0
            DO 600  I = 1, M1
                DO 575  J = 1, MA
                HTEMP = H1(I) * HA(J)
                IF (ABS(HTEMP).LT.DELTA)  GO TO 575
                LTEMP = L1(I) + LA(J)
                IF (MB.LT.1)  GO TO 500
                    DO 475  K = 1, MB
                    IF (LTEMP.EQ.LB(K))  GO TO 550
475                 CONTINUE
500             CONTINUE
C
                MB = MB + 1
                HB(MB) = HTEMP
                LB(MB) = LTEMP
                GO TO 575
550             CONTINUE
C
                HB(K) = HB(K) + HTEMP
575             CONTINUE
600         CONTINUE
C
C       COPY THE B ARRAYS INTO THE A ARRAYS
C
        MA = MB
            DO 650  I = 1, MB
            HA(I) = HB(I)
            LA(I) = LB(I)
650         CONTINUE
C
C       RETURN FOR NEXT THE MOMENT
C
        GO TO 235
        END
```

```
C       *  * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
C
C       FSUBM.FR            SUBROUTINE KERNEL
C
C       KANSAS STATE UNIVERSITY
C
C       F W RATCLIFFE       OCT 27, 78     REV:  APR 22, 79
C
C       * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
C
C       THIS VERSION OF KERNEL IMPLEMENTS A GAUSSIAN COMMUNICATION
C       CHANNEL HAVING A THIRD ORDER NONLINEARITY
C
C       * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
C
        SUBROUTINE KERNEL (H1, L1, DELTA, IBASE, ITTI, ITTO, N, M1)
C
        DIMENSION  H1(1), L1(1)
C
1       FORMAT ('0FIRST KERNEL PARAMETERS')
2       FORMAT (F30.20)
3       FORMAT ('0      BETA  = ', G15.6)
4       FORMAT ('       SUBROUTINE KERNEL ERROR => BETA  NOT GREATER THAN',
     1           ' OR EQUAL TO ZERO')
5       FORMAT ('0      GAMMA = ', G15.6)
6       FORMAT ('0      T     = ', G15.6)
7       FORMAT ('       SUBROUTINE KERNEL ERROR => T     NOT GREATER THAN',
     1           ' ZERO')
C
C       SET UP KNOWN PARAMETER(S) AND OUTPUT SUBROUTINE TITLE
C
        M1 = 0
        MNM1 = - N - 1
        NT2P1 = N * 2 + 1
        PI = 3.1415926
        WRITE (ITTO, 1)
C
C       INPUT THE VALUE FOR BETA
C
        READ  (ITTI, 2)  BETA
        WRITE (ITTO, 3)  BETA
        IF (BETA.LT.0.0)  WRITE (ITTO, 4)
        IF (BETA.LT.0.0)  STOP
C
C       INPUT THE VALUE FOR GAMMA
C
        READ  (ITTI, 2)  GAMMA
        WRITE (ITTO, 5)  GAMMA
C
C       INPUT THE VALUE FOR T
C
        READ  (ITTI, 2)  T
        WRITE (ITTO, 6)  T
        IF (T.LE.0.0)  WRITE (ITTO, 7)
```

```
      IF (T.LE.0.0)  STOP
C
C     CALCULATE THE FIRST KERNEL TERM(S)
C
          DO 300  I = 1, NT2P1
          N1 = MNM1 + I
          IF (N1.EQ.0)  GO TO 300
          HTEMP = EXP(- BETA * FLOAT(N1 ** 2))
          IF (ABS(HTEMP).LT.DELTA)  GO TO 300
          M1 = M1 + 1
          H1(M1) = HTEMP
          L1(M1) = IBASE ** (I - 1)
300       CONTINUE
C
C     CALCULATE THE THIRD KERNEL TERM(S)
C
      BETAT2 = BETA * 2.0
      C3 = GAMMA * SQRT(BETAT2 / PI) / (6.0 * T)
          DO 800  I = 1, NT2P1
          LI = IBASE ** (I - 1)
          N1 = MNM1 + I
          N1S = N1 ** 2
              DO 750  J = 1, NT2P1
              LIJ = LI + IBASE ** (J - 1)
              N2 = MNM1 + J
              N12 = N1 + N2
              N12S = N1S + N2 ** 2
                  DO 700  K = 1, NT2P1
                  N3 = MNM1 + K
                  NT = N12 + N3
                  NST = N12S + N3 ** 2
                  VALUE = FLOAT(NST) - 0.25 * FLOAT(NT ** 2)
                  HTEMP = C3 * EXP(- BETAT2 * VALUE)
                  IF (ABS(HTEMP).LT.DELTA)  GO TO 700
                  LTEMP = LIJ + IBASE ** (K - 1)
                  IF (M1.LT.1)  GO TO 550
                      DO 500  L = 1, M1
                      IF (LTEMP.EQ.L1(L))  GO TO 650
500                   CONTINUE
550               CONTINUE
C
                  M1 = M1 + 1
                  H1(M1) = HTEMP
                  L1(M1) = LTEMP
                  GO TO 700
650               CONTINUE
C
                  H1(L) = H1(L) + HTEMP
700               CONTINUE
750           CONTINUE
800       CONTINUE
C
      RETURN
      END
```

```
C       * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
C
C       ASUB.FR                 SUBROUTINE KERNEL
C
C       KANSAS STATE UNIVERSITY
C
C       F W RATCLIFFE           FEB 27, 79      REV:  APR 22, 79
C
C       * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
C
C       THIS VERSION OF KERNEL IMPLEMENTS A REALIZABLE COMMUNICATION
C       CHANNEL HAVING SECOND AND THIRD ORDER NONLINEARITIES
C
C       * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
C
        SUBROUTINE KERNEL (H1, L1, DELTA, IBASE, ITTI, ITTO, N, M1)
C
        DIMENSION  CK3(3), H1(1), L1(1)
C
1       FORMAT ('0SUBROUTINE KERNEL PARAMETERS')
2       FORMAT (F30.20)
3       FORMAT ('0      ALPHA  = ', G15.6)
4       FORMAT ('       SUBROUTINE KERNEL ERROR => ALPHA NOT GREATER THAN',
     1           ' OR EQUAL TO ZERO')
5       FORMAT ('0      BETA   = ', G15.6)
6       FORMAT ('       SUBROUTINE KERNEL ERROR => BETA  NOT GREATER THAN',
     1           ' OR EQUAL TO ZERO')
7       FORMAT ('0      GAMMA1 = ', G15.6)
8       FORMAT ('0      GAMMA2 = ', G15.6)
9       FORMAT ('0      GAMMA3 = ', G15.6)
10      FORMAT ('0      T      = ', G15.6)
11      FORMAT ('       SUBROUTINE KERNEL ERROR => T     NOT GREATER THAN',
     1           ' ZERO')
12      FORMAT ('0      TZERO  = ', G15.6)
13      FORMAT ('       SUBROUTINE KERNEL ERROR => TZERO NOT GREATER THAN',
     1           ' OR EQUAL TO T')
C
C       OUTPUT SUBROUTINE TITLE
C
        WRITE (ITTO, 1)
C
C       INPUT THE VALUE FOR ALPHA
C
        READ  (ITTI, 2)  ALPHA
        WRITE (ITTO, 3)  ALPHA
        IF (ALPHA.LT.0.0)  WRITE (ITTO, 4)
        IF (ALPHA.LT.0.0)  STOP
C
C       INPUT THE VALUE FOR BETA
C
        READ  (ITTI, 2)  BETA
        WRITE (ITTO, 5)  BETA
        IF (BETA.LT.0.0)  WRITE (ITTO, 6)
        IF (BETA.LT.0.0)  STOP
C
C       INPUT THE VALUES FOR THE GAMMAS
C
        READ  (ITTI, 2)  G1
        WRITE (ITTO, 7)  G1
        READ  (ITTI, 2)  G2
        WRITE (ITTO, 8)  G2
        READ  (ITTI, 2)  G3
        WRITE (ITTO, 9)  G3
```

```
C
C       INPUT THE VALUE FOR T
C
        READ  (ITTI, 2)  T
        WRITE (ITTO, 10)  T
        IF (T.LE.0.0)  WRITE (ITTO, 11)
        IF (T.LE.0.0)  STOP
C
C       INPUT THE VALUE FOR TZERO
C
        READ  (ITTI, 2)  TZERO
        WRITE (ITTO, 12)  TZERO
        IF (TZERO.LT.T)  WRITE (ITTO, 13)
        IF (TZERO.LT.T)  STOP
C
C       CALCULATE THE KERNEL CONSTANTS
C
        AMB = ALPHA - BETA
        AT2 = ALPHA * 2.0
        AT2MB = AT2 - BETA
        AT3 = ALPHA * 3.0
        AT3MB = AT3 - BETA
        ATT = ALPHA * T
        BTT = BETA * T
        C1 = G1 * BETA
        C2 = G2 * BETA / 2.0
        C3 = G3 * BETA / 6.0
        ECA = EXP(ATT) - 1.0
        ECA1 = ECA
        IF (ALPHA.NE.BETA)  ECA1 = ECA / AMB
        ECA2 = ECA ** 2 / AT2MB
        ECA3 = ECA ** 3 / AT3MB
        ECB = (EXP(BTT) - 1.0) / BETA
        TERM1 = T
        IF (ALPHA.NE.BETA)  TERM1 = (1.0 - EXP(- AMB * T)) / AMB
        TERM2 = (1.0 - EXP(- AT2MB * T)) / AT2MB
        TERM3 = (1.0 - EXP(- AT3MB * T)) / AT3MB
        CK1 = ECB - TERM1
        CK2M1 = ECA * (TERM1 - TERM2)
        CK2M2 = ECB - 2.0 * TERM1 + TERM2
        CK3(1) = ECA ** 2 * (TERM2 - TERM3)
        CK3(2) = ECA * (TERM1 - 2.0 * TERM2 + TERM3)
        CK3(3) = ECB - 3.0 * (TERM1 - TERM2) - TERM3
C
C       CALCULATE H1(A0) AND NORMALIZE CONSTANTS
C
        HTEMP = C1 * CK1 * EXP(- BETA * TZERO)
        C1 = C1 / HTEMP
        C2 = C2 / HTEMP
        C3 = C3 / HTEMP
        M1 = 0
C
C       CALCULATE THE REMAINDER OF THE FIRST KERNEL TERM(S) IF NECESSARY
C
        IF (G1.EQ.0.0)  GO TO 420
            DO 300  I = 2, N
            HTEMP = 0.0
            N1 = 1 - I
            TR1 = TZERO - FLOAT(N1) * T
            IF (TR1.LE.T)  GO TO 240
            ETA = EXP(- ALPHA * TR1)
            ETB = TR1 - T
            IF (ALPHA.NE.BETA)  ETB = EXP(AMB * (TR1 - T)) - 1.0
```

```
            HTEMP = ECA1 * ETA * ETB
240         CONTINUE
C
            ETA = EXP(- BETA * TR1)
            HTEMP = C1 * (HTEMP + ETA * CK1)
            IF (ABS(HTEMP).LT.DELTA)  GO TO 300
            M1 = M1 + 1
            H1(M1) = HTEMP
            L1(M1) = IBASE ** (I - 1)
300         CONTINUE
420     CONTINUE
C
C       CALCULATE THE SECOND KERNEL TERM(S) IF NECESSARY
C
        IF (G2.EQ.0.0)  GO TO 720
            DO 700  I = 1, N
            LI = IBASE ** (I - 1)
            N1 = 1 - I
            TR1 = TZERO - FLOAT(N1) * T
                DO 680  J = 1, N
                N2 = 1 - J
                TR2 = TZERO - FLOAT(N2) * T
                HTEMP = 0.0
                TRM = TR1
                TP = TR2
                IF (N1.GE.N2)  GO TO 440
                TRM = TR2
                TP = TR1
440             CONTINUE
C
                IF (TRM.LE.T)  GO TO 460
                ETA = EXP(- ALPHA * (TR1 + TR2))
                ETB = EXP(AT2MB * (TRM - T))
                HTEMP = ECA2 * ETA * (ETB - 1.0)
460             CONTINUE
C
                IF (N1.NE.N2)  GO TO 480
                ETA = EXP(- BETA * TRM)
                HTEMP = C2 * (HTEMP + ETA * CK2M2)
                GO TO 500
480             CONTINUE
C
                ETA = EXP(- ALPHA * TP)
                ETB = EXP(AMB * TRM)
                HTEMP = C2 * (HTEMP + ETA * ETB * CK2M1)
500             CONTINUE
C
                IF (ABS(HTEMP).LT.DELTA)  GO TO 680
                LTEMP = LI + IBASE ** (J - 1)
                IF (M1.EQ.0)  GO TO 580
                    DO 560  K = 1, M1
                    IF (LTEMP.EQ.L1(K))  GO TO 620
560                 CONTINUE
580             CONTINUE
C
                M1 = M1 + 1
                H1(M1) = HTEMP
                L1(M1) = LTEMP
                GO TO 680
620             CONTINUE
C
                H1(K) = H1(K) + HTEMP
680             CONTINUE
```

```
700         CONTINUE
720     CONTINUE
C
C       CALCULATE THE THIRD KERNEL TERM(S) IF NECESSARY
C
        IF (G3.EQ.0.0)  GO TO 940
            DO 920  I = 1, N
            LI = IBASE ** (I - 1)
            N1 = 1 - I
            TR1 = TZERO - FLOAT(N1) * T
                DO 900  J = 1, N
                LIJ = LI + IBASE ** (J - 1)
                N2 = 1 - J
                TR2 = TZERO - FLOAT(N2) * T
                    DO 880  K = 1, N
                    N3 = 1 - K
                    TR3 = TZERO - FLOAT(N3) * T
                    HTEMP = 0.0
                    TRM = TR1
                    IF (TR2.LT.TRM)  TRM = TR2
                    IF (TR3.LT.TRM)  TRM = TR3
                    IF (TRM.LE.T)  GO TO 740
                    ETA = EXP(- ALPHA * (TR1 + TR2 + TR3))
                    ETB = EXP(AT3MB * (TRM - T))
                    HTEMP = EGA3 * ETA * (ETB - 1.0)
740                 CONTINUE
C
                    ETA = EXP(- BETA * TRM)
                    M = 0
                    IF (TR1.EQ.TRM)  M = 1
                    IF (TR2.EQ.TRM)  M = M + 1
                    IF (TR3.EQ.TRM)  M = M + 1
                    TP = 0.0
                    IF (TR1.NE.TRM)  TP = TR1
                    IF (TR2.NE.TRM)  TP = TP + TR2
                    IF (TR3.NE.TRM)  TP = TP + TR3
                    ETB = EXP(- ALPHA * TP)
                    ETC = EXP(ALPHA * FLOAT(3 - M) * TRM)
                    HTEMP = C3 * (HTEMP + ETA * ETB * ETC * CN3(M))
                    IF (ABS(HTEMP).LT.DELTA)  GO TO 880
                    LTEMP = LIJ + IBASE ** (K - 1)
                    IF (M1.EQ.0)  GO TO 820
                        DO 800  L = 1, M1
                        IF (LTEMP.EQ.L1(L))  GO TO 860
800                     CONTINUE
820                 CONTINUE
C
                    M1 = M1 + 1
                    H1(M1) = HTEMP
                    L1(M1) = LTEMP
                    GO TO 880
860                 CONTINUE
C
                    H1(L) = H1(L) + HTEMP
880                 CONTINUE
900             CONTINUE
920         CONTINUE
940     CONTINUE
C
        RETURN
        END
```

```
C       * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
C
C       FRED68.FR               GAUSS QUADRATURE PROGRAM
C
C       KANSAS STATE UNIVERSITY
C
C       F W RATCLIFFE           MAR 31, 79      REV: APR 08, 79
C
C       * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
C
C       THIS PROGRAM CALCULATES A SET OF N ABSCISSA/WEIGHT VALUE(S)
C       FROM THE FIRST 2N MOMENTS OF A WEIGHT FUNCTION.
C
C       PROGRAM REQUIRES SUBROUITNE(S)
C               BLKIO
C               CKVEC
C               FLCK
C               GPOLY
C               GQRULE
C
C       NOTE:   TWO SUBROUTINES (GPOLY AND GQRULE) USED BY THIS
C               PROGRAM ARE FORTRAN VERSIONS OF ALGOL PROCEDURES
C               SUPPLIED BY GOLUB AND WELSCH IN THIER PAPER
C               'GENERATION OF GAUSS QUADRATURE RULES'.
C
C       * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
C
        DIMENSION  A(20), B(21), INFO(3), NAME(12), T(20), W(20)
        LOGICAL  DECOUT, DISKOUT
        REAL  MU(41), MUZERO
C
1       FORMAT ('0GAUSS QUADRATURE PROGRAM', 10X, I2, 2('/', I2))
2       FORMAT ('0NUMBER OF ABSCISSA/WEIGHT VALUE(S) DESIRED (1-20) ? ', Z)
3       FORMAT ('0OUTPUT DATA TO A DISK FILE (YES/NO) ? ', Z)
4       FORMAT (S1)
5       FORMAT ('0OUTPUT DATA TO THE DECWRITER (YES/NO) ? ', Z)
6       FORMAT (//, '0ABSCISSA/WEIGHT DATA STORED IN DISK FILE:  ', S23)
7       FORMAT ('0INPUT MOMENT VECTOR')
8       FORMAT ('    MU(', I2, ') ? ', Z)
9       FORMAT (//, '0MOMENT VALUES', 14X, 'ABSCISSA VALUE(S)',
     1                                    9X, 'WEIGHT VALUE(S)')
10      FORMAT ('    MU(', I2, ') = ', G13.6, 5X, 'T(', I2, ') = ', G13.6,
     1                                         5X, 'W(', I2, ') = ', G13.6)
11      FORMAT ('    MU(', I2, ') = ', G13.6)
12      FORMAT ('0* * WARNING => ONLY ', I2, ' ABSCISSA/WEIGHT',
     1          ' VALUE(S) DEFINED * *')
13      FORMAT ('0RUN PROGRAM AGAIN (YES/NO) ? ', Z)
14      FORMAT ('0CREATE NEW RECORD OR MODIFY OLD RECORD (C OR M) ? ', Z)
C
C       SET UP KNOWN PARAMETER(S) AND OUTPUT PROGRAM TITLE
C
        ITTI = 11
        ITTO = 10
        NR1 = 0
        NWMAX = 20
        CALL DATE (INFO, IERROR)
        IF (IERROR.NE.1)  STOP => ERROR DETERMINING DATE
        WRITE (ITTO, 1)  INFO(1), INFO(2), INFO(3)
200     CONTINUE
```

```
                DO 560  I = NP1, NW
                T(I) = 0.0
                W(I) = 0.0
560             CONTINUE
580     CONTINUE
C
C       OUTPUT DATA TO A DISK FILE IF NECESSARY
C
        IF (.NOT.DISKOUT)  GO TO 620
        CALL BLKIO (T, 1, 3, NAME, NR1, NW)
        NR1 = NR1 + 1
        CALL BLKIO (W, 1, 3, NAME, NR1, NW)
        NR1 = NR1 + 1
620     CONTINUE
C
C       OUTPUT DATA EITHER TO THE DECWRITER OR TO THE TELETYPE
C
        WRITE (ICHAN, 9)
                DO 640  I = 1, NW
                IM1 = I - 1
                WRITE (ICHAN, 10)  IM1, MU(I), I, T(I), I, W(I)
640             CONTINUE
C
        NWP1 = NW + 1
                DO 660  I = NWP1, NM
                IM1 = I - 1
                WRITE (ICHAN, 11)  IM1, MU(I)
660             CONTINUE
700     CONTINUE
C
C       OUTPUT WARNING IF UNDEFINED ABSCISSA/WEIGHT
C       VALUE(S) OCCURRED
C
        IF (N.NE.NW)  WRITE (ICHAN, 12)  N
C
C       ASK IF PROGRAM IS TO BE RUN AGAIN
C
        WRITE (ITTO, 13)
        READ  (ITTI, 4)  IANS
        IF (IANS.NE.'Y<0>')  GO TO 720
C
C       ASK IF OLD MOMENT VECTOR IS TO BE MODIFIED
C       OR IS AN ENTIRELY NEW MOMENT VECTOR TO BE WRITTEN
C
        WRITE (ITTO, 14)
        READ  (ITTI, 4)  IANS
        IF (IANS.NE.'M<0>')  GO TO 500
        GO TO 540
720     CONTINUE
C
C       CLOSE THE DECWRITER IF NECESSARY
C
        IF (.NOT.DECOUT)  GO TO 760
        CALL CLOSE (ICHAN, IERR2)
        IF (IERR2.NE.1)  STOP => ERROR CLOSE DECWRITER
760     CONTINUE
C
        STOP => FRED68
        END
```

```
C
C       INPUT THE NUMBER OF ABSCISSA/WEIGHT VALUES DESIRED
C
        WRITE (ITTO, 2)
        READ  (ITTI)  NW
        IF (NW.LT.1.OR.NW.GT.NWMAX)  GO TO 200
        NM = NW * 2
C
C       ASK IF A DISK FILE IS TO BE USED FOR OUTPUT
C
        WRITE (ITTO, 3)
        READ  (ITTI, 4)  IANS
        DISKOUT = .FALSE.
        IF (IANS.NE.'Y<0>')  GO TO 420
        DISKOUT = .TRUE.
        CALL FLCK ('OUTPUT FILENAME ?', 3, NAME)
420     CONTINUE
C
C       ASK IF DATA IS TO BE SAVED ON THE DECWRITER
C
        WRITE (ITTO, 5)
        READ  (ITTI, 4)  IANS
        DECOUT = .FALSE.
        ICHAN = ITTO
        IF (IANS.NE.'Y<0>')  GO TO 500
        DECOUT = .TRUE.
        ICHAN = 2
        CALL OPEN (ICHAN, '$TTO1', 3, IERR2)
        IF (IERR2.NE.1)  STOP => ERROR OPEN DECWRITER
        WRITE (ICHAN, 1)  INFO(1), INFO(2), INFO(3)
        IF (DISKOUT)  WRITE (ICHAN, 6)  NAME(1)
500     CONTINUE
C
C       INPUT THE MOMENT VECTOR
C
        WRITE (ITTO, 7)
                DO 520  I = 1, NM
                IM1 = I - 1
                WRITE (ITTO, 8)  IM1
                READ  (ITTI)  MU(I)
520             CONTINUE
540     CONTINUE
C
C       CORRECT ANY DESIRED VALUE(S)
C
        CALL CKVEC (0, NM, MU)
C
C       HERE TO RUN THE GAUSS QUADRATURE PROCEDURE
C
        N = NW
        CALL GPOLY (N, MU, A, B)
        IF (N.LT.1)  GO TO 700
        MUZERO = MU(1)
        CALL GQRULE (N, A, B, MUZERO, T, W)
C
C       ZERO OUT UNDEFINED ABSCISSA/WEIGHT VALUE(S) IF NECESSARY
C
        NP1 = N + 1
        IF (NP1.GT.NW)  GO TO 580
```

```
                W(I) = 0.0
200             CONTINUE
220     CONTINUE
C
        NORM = AMAX1(NORM, ABS(B(N)) + ABS(A(N)))
C
C       RELATIVE ZERO TOLERANCE
C       NOTE:   THE EXPONENT IS THE NEGATIVE OF THE NUMBER OF
C               HEXADECIMAL DIGITS IN THE FRACTION OF A
C               FLOATING POINT NUMBER
C
        ZERO = 16.0 ** (- 6)
        EPS = NORM * ZERO
        W(1) = 1.0
        W(N) = 0.0
        M = N
        LAMBDA = NORM
        LAMBD1 = NORM
        LAMBD2 = NORM
        RHO = NORM
300     CONTINUE
C
C       LOOK FOR CONVERGENCE OF LOWER DIAGONAL ELEMENT
C
        IF (M.EQ.0)  GO TO 800
        I = M - 1
        K = I
        M1 = I
        IF (ABS(B(M1 + 1)).GT.EPS)  GO TO 400
        T(M) = A(M)
        W(M) = MUZERO * W(M) ** 2
        RHO = AMIN1(LAMBD1, LAMBD2)
        M = M1
        GO TO 300
400     CONTINUE
C
C       SMALL OFF DIAGONAL ELEMENT MEANS MATRIX CAN BE SPLIT
C
        I = I - 1
        IF (ABS(B(I + 1)).LE.EPS)  GO TO 440
        K = I
        GO TO 400
440     CONTINUE
C
C       FIND EIGENVALUES OF LOWER 2X2 AND SELECT ACCELERATING SHIFT
C
        B2 = B(M1 + 1) ** 2
        DET = SQRT((A(M1) - A(M)) ** 2 + 4.0 * B2)
        AA = A(M1) + A(M)
        LAMBD2 = AA - DET
        IF (AA.GE.0.0)  LAMBD2 = AA + DET
        LAMBD2 = 0.5 * LAMBD2
        LAMBD1 = (A(M1) * A(M) - B2) / LAMBD2
        EIGMAX = AMAX1(LAMBD1, LAMBD2)
        RHO = EIGMAX
```

```
C       * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
C
C       CSUB.FR                 SUBROUTINE GPOLY
C
C       KANSAS STATE UNIVERSITY
C
C       F W RATCLIFFE           MAR 31, 79
C
C       * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
C
C       THIS SUBROUTINE IMPLEMENTS IN FORTRAN THE ALGOL
C       PROCEDURE GENORTHOPOLY SUPPLIED BY GOLUB AND WELSH
C       IN THEIR PAPER 'GENERATION OF GAUSS QUADRATURE RULES'
C
C       SUBROUTINE REQUIRES
C               MU    => VECTOR CONTAINING THE 2N MOMENTS OF THE
C                        WEIGHT FUNCTION
C               N     => NUMBER OF ABSCISSA/WEIGHT VALUES DESIRED
C                        (N MUST HAVE A VALUE OF ONE OR GREATER)
C
C       SUBROUTINE SUPPLIES
C               A, B => ALPHA AND BETA VECTORS TO BE USED BY GQRULE
C               N     => THE ACTUAL NUMBER OF ABSCISSA/WEIGHT VALUES THAT
C                        CAN BE OBTAINED FROM THE MOMENT VECTOR SUPPLIED
C
C       NOTE THE FOLLOWING DIFFERENCES BETWEEN THIS FORTRAN
C               VERSION AND THE ORIGINAL ALGOL PROCEDURE
C               1) THE B VECTOR SUBSCRIPT IS SHIFTED PLUS ONE
C                  WITH RESPECT TO THE ALGOL PROCEDURE
C               2) ELIMINATION OF THE IF STATEMENT
C               3) THE FIRST COLUMN, FIRST ROW, AND LAST ROW
C                  OF THE R ARRAY ARE NOT CALCULATED
C               4) ONLY 2N MOMENTS OF THE WEIGHT FUNCTION ARE
C                  USED IN THE CALCULATIONS
C               5) THE MATRIX R MUST DIMENSIONED R(N, N + 1)
C                  BY THE SUBROUTINE
C               6) THE CALCULATION OF THE R ARRAY TERMINATES
C                  IF A NEGATIVE DIAGONAL TERM IS ENCOUNTERED
C
C       * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
C
        SUBROUTINE GPOLY (N, MU, A, B)
C
C       GIVEN THE 2N MOMENTS (MU) OF THE WEIGHT FUNCTION,
C       GENERATE THE RECURSION COEFFICIENTS (A, B) OF THE
C       NORMALIZED ORTHOGONAL POLYNOMIALS.
C
        DIMENSION  A(1), B(1), R(20, 21)
        REAL  MU(1)
C
C       INITIALIZE THE ALPHA VECTOR, BETA VECTOR, AND R ARRAY
C
        NP1 = N + 1
        B(NP1) = 0.0
                DO 160  I = 1, N
                A(I) = 0.0
                B(I) = 0.0
                        DO 140  J = 1, NP1
                        R(I, J) = 0.0
140                     CONTINUE
```

```
      IF (ABS(EIGMAX - RHO).LE.0.125 * ABS(EIGMAX))  LAMBDA = EIGMAX
C
C     TRAMSFORM BLOCK FROM K TO M
C
      CJ = B(K + 1)
      B(K) = A(K) - LAMBDA
      IF (K.GT.M1)  GO TO 640
            DO 600  J = K, M1
            JP1 = J + 1
            JP2 = J + 2
            R = SQRT(CJ ** 2 + B(J) ** 2)
            ST = CJ / R
            CT = B(J) / R
            AJ = A(J)
            B(J) = R
            CJ = B(JP2) * ST
            B(JP2) = - B(JP2) * CT
            F = AJ * CT + B(JP1) * ST
            Q = B(JP1) * CT + A(JP1) * ST
            A(J) = F * CT + Q * ST
            B(JP1) = F * ST - Q * CT
            WJ = W(J)
            A(JP1) = AJ + A(JP1) - A(J)
            W(J) = WJ * CT + W(JP1) * ST
            W(JP1) = WJ * ST - W(JP1) * CT
600         CONTINUE
640   CONTINUE
C
      B(K) = 0.0
      GO TO 300
800   CONTINUE
C
C     ARRANGE ABSCISSAS IN ASCENDING ORDER
C
      IF (N.LT.2)  RETURN
      IEND = N
            DO 880  M = 2, N
            EX = .FALSE.
                  DO 840  I = 2, IEND
                  IF (T(I - 1).LE.T(I))  GO TO 840
                  WJ = T(I - 1)
                  T(I - 1) = T(I)
                  T(I) = WJ
                  WJ = W(I - 1)
                  W(I - 1) = W(I)
                  W(I) = WJ
                  EX = .TRUE.
840               CONTINUE
C
            IF (.NOT.EX)  RETURN
            IEND = IEND - 1
880         CONTINUE
C
      RETURN
      END
```

```
C       * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
C
C       DSUB.FR                 SUBROUTINE GQRULE
C
C       KANSAS STATE UNIVERSITY
C
C       F W RATCLIFFE           MAR 31, 79
C
C       * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
C
C       THIS SUBROUTINE IMPLEMENTS IN FORTRAN THE ALGOL
C       PROCEDURE GAUSSQUADRULE SUPPLIED BY GOLUB AND WELSCH
C       IN THIER PAPER 'GENERATION OF GAUSS QUADRATURE RULES'.
C
C       SUBROUTINE REQUIRES
C               A, B    => ALPHA AND BETA VECTORS OBTAINED
C                          FROM GPOLY
C               N       => NUMBER OF ABSCISSA/WEIGHT VALUES
C               MUZERO  => ZERO ORDER MOMENT OF WEIGHT FUNCTION
C
C       SUBROUTINE SUPPLIES
C               T       => ABSCISSA VECTOR
C               W       => WEIGHT VECTOR
C
C       NOTE THE FOLLOWING DIFFERENCES BETWEEN THIS FORTRAN
C               VERSION AND THE ORIGINAL ALGOL PROCEDURE
C               1) THE B VECTOR SUBSCRIPTS ARE SHIFTED PLUS ONE
C                  WITH RESPECT TO THE ALGOL PROCEDURE
C               2) THE SECTION FOR SYMMETERIZING THE MATRIX
C                  HAS BEEN OMITTED
C               3) THE C VECTOR AND THE SYMM PARAMETER
C                  HAVE BEEN OMITTED
C
C       * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
C
        SUBROUTINE GQRULE (N, A, B, MUZERO, T, W)
C
C       GIVEN THE COEFFICIENTS  (A, B)  OF THE TWO TERM
C       RECURRENCE RELATION:  P(K) = (A(K)X + B(K))P(K - 1),
C       THIS PROCEDURE COMPUTES THE ABSCISSAS T AND THE WEIGHTS W
C       OF THE GAUSSIAN TYPE QUADRATURE RULE ASSOCIATED WITH THE ORTHO-
C       GONAL POLYNOMIAL BY  QR  TYPE ITERATION WITH ORIGIN SHIFTING.
C
        DIMENSION  A(1), B(1), T(1), W(1)
        LOGICAL  EX
        REAL  LAMBDA, LAMBD1, LAMBD2, MUZERO, NORM
C
C       FIND THE MAXIMUM ROW SUM NORM AND INITIALIZE W( ).
C
        B(1) = 0.0
        NORM = 0.0
        IF (N.LT.2)  GO TO 220
        NM1 = N - 1
                DO 200  I = 1, NM1
                NORM = AMAX1(NORM, ABS(B(I)) + ABS(A(I)) + ABS(B(I+1)))
```

```
160             CONTINUE
C
C       PLACE THE PARTIAL CHOLESKY DECOMPOSITION OF THE
C       MOMENT MATRIX IN R().
C
        IF (MU(1).GE.0.0)  GO TO 220
        N = 0
        RETURN
220     CONTINUE
C
        DIAG = SQRT(MU(1))
        R(1, 1) = DIAG
                DO 240  I = 2, NP1
                R(1, I) = MU(I) / DIAG
240             CONTINUE
        IF (N.LT.2)  GO TO 620
        KEND = 1
                DO 600  I = 2, N
                SUM = MU(I * 2 - 1)
                                DO 300  K = 1, KEND
                                SUM = SUM - R(K, I) ** 2
300                             CONTINUE
C
                IF (SUM.GE.0.0)  GO TO 340
                N = I
                NP1 = N + 1
                GO TO 620
340             CONTINUE
C
                DIAG = SQRT(SUM)
                R(I, I) = DIAG
                IP1 = I + 1
                        DO 500  J = IP1, NP1
                        SUM = MU(I + J - 1)
                                DO 400  K = 1, KEND
                                SUM = SUM - R(K, I) * R(K, J)
400                             CONTINUE
                        R(I, J) = SUM / DIAG
500                     CONTINUE
                KEND = KEND + 1
600             CONTINUE
620     CONTINUE
C
C       COMPUTE THE RECURSION COEFFICIENTS FROM THE PARTIAL
C       DECOMPOSITION R().
C
        A(1) = R(1, 2) / R(1, 1)
        IF (N.LT.2)  GO TO 820
                DO 800  I = 2, N
                IM1 = I - 1
                IP1 = I + 1
                CDIAG = R(I, I)
                PDIAG = R(IM1, IM1)
                A(I) = R(I, IP1) / CDIAG - R(IM1, I) / PDIAG
                B(I) = CDIAG / PDIAG
800             CONTINUE
820     CONTINUE
C
        RETURN
        END
```

```
C       * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
C
C       FRED72.FR               PROBABABILITY OF ERROR
C
C       KANSAS STATE UNIVERSITY
C
C       F W RATCLIFFE           APR 07, 79      REV:  APR 20, 79
C
C       * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
C
C       THIS PROGRAM CALCULATES THE PROBABILITY OF ERROR VS SNR
C       USING DATA OBTAINED FROM THE VOLTERRA SERIES APPROACH
C
C       PROGRAM REQUIRES SUBROUTINE(S)
C               BLKIO
C               FLCK
C               PERROR
C               STATUS
C
C       * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
C
        DIMENSION  AW(60), E(128), SD(128), TEMP(10)
        INTEGER  INFO(3), NAME(12)
        LOGICAL  DB, DECOUT, FLAG, PLOT
C
1       FORMAT ('0PROBABILITY OF ERROR', 10X, I2, '/', I2, '/', I2)
2       FORMAT ('0      NUMBER OF SYMBOLS (2, 4, -- 6) ? ', Z)
3       FORMAT ('0      NUMBER OF ABSCISSAS/WEIGHTS (1-10) ? ', Z)
4       FORMAT (' ERROR => DISK FILE NOT THE CORRECT SIZE')
5       FORMAT ('0      SNR START VALUE (DB) ? ', Z)
6       FORMAT ('       SNR  END  VALUE (DB) ? ', Z)
7       FORMAT ('0      NUMBER OF STEPS (1-127) ? ', Z)
8       FORMAT ('0CONVERT PROBABILITY OF ERROR INTO DB (YES/NO) ? ', Z)
9       FORMAT (S1)
10      FORMAT ('0SAVE DATA ON THE DECWRITER (YES/NO) ? ', Z)
11      FORMAT (//, '0ABSCISSA/WEIGHT DATA OBTAINED FROM DISK FILE:  ', S23)
12      FORMAT ('0INPUT DATA FOR SYMBOL : ', I4)
13      FORMAT ('       ABSCISSA VALUES', 18X, 'WEIGHT VALUES')
14      FORMAT (' ', 2(10X, 'V(', I2, ') = ', G15.6))
15      FORMAT (//, '0PROBABILITY OF ERROR')
16      FORMAT (//, '0PROBABILITY OF ERROR (DB)')
17      FORMAT ('       E(SNR = ', G10.4, 'DB) = ', G15.6)
18      FORMAT ('0* * WARNING => ALPHA LESS THAN 2.0 * *')
19      FORMAT ('0PLOT THE PROBABILITY OF ERROR (YES/NO) ? ', Z)
20      FORMAT ('0NOTE:  PROBABILITY OF ERROR EXPRESSED IN DB')
21      FORMAT ('0OUTPUT PROBABILITY OF ERROR TO DISK (YES/NO) ? ', Z)
22      FORMAT ('0RUN PROGRAM AGAIN (YES/NO) ? ', Z)
C
C       SET UP KNOWN PARAMETER(S) AND OUTPUT PROGRAM TITLE
C
        ITTI = 11
        ITTO = 10
        MSTEP = 127
        NWMAX = 10
        NR = 0
```

```
      NSMAX = 6
      CALL DATE (INFO, IERROR)
      IF (IERROR.NE.1)  STOP  ERROR => DETERMINING DATE
      WRITE (ITTO, 1)  INFO(1), INFO(2), INFO(3)
200   CONTINUE
C
C     INPUT THE NUMBER OF SYMBOLS
C
      WRITE (ITTO, 2)
      READ  (ITTI)  NS
      IF (NS.LT.2.OR.NS.GT.NSMAX)  GO TO 200
      NSTEST = NS / 2 * 2
      IF (NS.NE.NSTEST)  GO TO 200
220   CONTINUE
C
C     INPUT THE NUMBER OF ABSCISSA/WEIGHT VALUES
C
      WRITE (ITTO, 3)
      READ  (ITTI)  NW
      IF (NW.LT.1.OR.NW.GT.NWMAX)  GO TO 220
      NWT2 = NW * 2
      NMOVE = NS * NWT2
300   CONTINUE
C
C     INPUT THE ABSCISSA/WEIGHT VALUES FROM DISK
C
      CALL FLCK ('ABSCISSA/WEIGHT FILENAME ? ', 1, NAME)
      CALL STATUS (IERROR, 1, NAME, NBYTE)
      IF (NBYTE.EQ.NMOVE * 4)  GO TO 320
      WRITE (ITTO, 4)
      GO TO 300
320   CONTINUE
C
      CALL BLKIO (AW, 0, 1, NAME, 0, NMOVE)
C
C     INPUT SNR PARAMETERS
C
      WRITE (ITTO, 5)
      READ  (ITTI)  START
      WRITE (ITTO, 6)
      READ  (ITTI)  FINAL
380   CONTINUE
C
C     DETERMINE CURVE RESOLUTION
C
      WRITE (ITTO, 7)
      READ  (ITTI)  NSTEP
      IF (NSTEP.LT.1.OR.NSTEP.GT.MSTEP)  GO TO 380
      NSP1 = NSTEP + 1
C
C     CALCULATE THE PROBABILITY OF ERROR VECTOR
C
      CALL PERROR (AW, E, FINAL, FLAG, NSP1, NS, NW, SD, START)
C
C     ASK IF PROBABILITY OF ERROR VECTOR IS TO BE CONVERTED INTO DB
```

```
C
      WRITE (ITTO, 8)
      READ  (ITTI, 9)  IANS
      DB = .FALSE.
      IF (IANS.NE.'Y<0>')  GO TO 420
      DB = .TRUE.
            DO 400  I = 1, NSP1
            E(I) = ALOG10(E(I))
400         CONTINUE
420   CONTINUE
C
C     ASK IF DATA IS TO BE SAVED ON THE DECWRITER
C
      WRITE (ITTO, 10)
      READ  (ITTI, 9)  IANS
      DECOUT = .FALSE.
      ICHAN = ITTO
      IF (IANS.NE.'Y<0>')  GO TO 600
C
C     HERE TO OPEN THE DECWRITER
C
      DECOUT = .TRUE.
      ICHAN = 1
      CALL OPEN (ICHAN, '$TTO1', 3, IERR1)
      IF (IERR1.NE.1)  STOP => ERROR OPEN DECWRITER
C
C     SAVE THE INPUT DATA IF THE DECWRITER IS OPEN
C
      WRITE (ICHAN, 1)  INFO(1), INFO(2), INFO(3)
      WRITE (ICHAN, 11)  NAME(1)
            DO 580  I = 1, NS
            IOFF = NWT2 * (I - 1)
            IS = - NS - 1 + I * 2
            WRITE (ICHAN, 12)  IS
            WRITE (ICHAN, 13)
                  DO 560  J = 1, NW
                  JOFF = J + IOFF
                  JOPNW = JOFF + NW
                  WRITE (ICHAN, 14)  J, AW(JOFF), J, AW(JOPNW)
560               CONTINUE
580         CONTINUE
600   CONTINUE
C
C     HERE TO OUTPUT THE PROBABILITY OF ERROR VECTOR
C     EITHER TO THE DECWRITER OR TO THE TELETYPE
C
      IF (.NOT.DB)  WRITE (ICHAN, 15)
      IF (DB)  WRITE (ICHAN, 16)
      SSIZE = (FINAL - START) / FLOAT(NSTEP)
      SNR = START
            DO 700  I = 1, NSP1
            WRITE (ICHAN, 17)  SNR, E(I)
            SNR = SNR + SSIZE
700         CONTINUE
C
```

```
C       OUTPUT ALPHA WARNING IF NEEDED
C
        IF (.NOT.FLAG)  WRITE (ICHAN, 18)
C
C       CLOSE THE DECWRITER IF OPEN
C
        IF (.NOT.DECOUT)  GO TO 720
        CALL CLOSE (ICHAN, IERR1)
        IF (IERR1.NE.1)  STOP => ERROR CLOSE THE DECWRITER
720     CONTINUE
C
C       PLOT THE PROBABILITY OF ERROR VECTOR IF DESIRED
C
        WRITE (ITTO, 19)
        READ  (ITTI, 9)  IANS
        PLOT = .FALSE.
        IF (IANS.NE.'Y<0>')  GO TO 800
        PLOT = .TRUE.
        CALL PAGE (3)
        CALL STATXY (IDUMMY, KWNX, KWNY)
        CALL GRAPH (E, 1, FINAL, 3, 'SNR',
     1           NSP1, 1, START, 10, 'ERROR PROB')
        CALL INITXY (1, KWNX, KWNY)
C
C       OUTPUT ALPHA WARNING IF NEEDED
C
        IF (.NOT.FLAG)  WRITE (ITTO, 18)
C
C       OUTPUT NOTE IF PROBABILITY OF ERROR VECTOR IS EXPRESSED IN DB
C
        IF (DB)  WRITE (ITTO, 20)
C
C       WAIT FOR OPERATOR RESPONSE
C
        READ  (ITTI, 9)  IANS
800     CONTINUE
C
C       OUTPUT PROBABILITY OF ERROR VECTOR TO DISK IF DESIRED
C
        WRITE (ITTO, 21)
        READ  (ITTI, 9)  IANS
        IF (IANS.NE.'Y<0>')  GO TO 840
        CALL FLCK ('OUTPUT FILENAME ? ', 3, NAME)
        CALL BLKIO (E, 2, 3, NAME, 0, NSP1)
840     CONTINUE
C
C       ASK IF THE PROGRAM IS TO BE RUN AGAIN
C
        WRITE (ITTO, 22)
        READ  (ITTI, 9)  IANS
        IF (IANS.EQ.'Y<0>'.AND.PLOT)  CALL PAGE (3)
        IF (IANS.EQ.'Y<0>')  GO TO 200
        STOP => FRED72
        END
```

```
C       * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
C
C       FSUB.FR                 SUBROUTINE PERROR
C
C       KANSAS STATE UNIVERSITY
C
C       F W RATCLIFFE           APR 05, 79
C
C       * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
C
C       THIS SUBROUTINE CALCULATES THE PROBABILITY OF ERROR
C       VERSUS SNR FOR A NONLINEAR COMMUNICATIONS CHANNEL
C
C       SUBROUTINE REQUIRES
C               AW      => ABSCISSA/WEIGHT VECTOR
C               FINAL   => FINAL SNR VALUE (DB)
C               NPOINT  => NUMBER OF POINTS DESIRED IN
C                          PROBABILITY OF ERROR VECTOR
C               NS      => NUMBER OF SYMBOLS
C               NW      => NUMBER OF ABSCISSA/WEIGHT VALUES
C                          PER SYMBOL
C               SD      => WORKING VECTOR USED BY SUBROUTINE
C               START   => STARTING SNR VALUE (DB)
C
C       SUBROUTINE SUPPLIES
C               E       => PROBABILITY OF ERROR VECTOR
C               FLAG    => WHEN .FALSE. THE APPROXMATION FOR THE
C                          Q FUNCTION IS NOT VALID
C
C       NOTES:  1) THE AW VECTOR MUST BE DIMENSIONED AW(NS * NW * 2) BY THE
C                  CALLING ROUTINE.  THAT IS THE ABSCISSA/WEIGHT
C                  INFORMATION FOR A SYMBOL IS STORED TOGETHER FOLLOWED BY
C                  THE ABSCISSA/WEIGHT INFORMATION FOR THE NEXT SYMBOL.
C                  FOR A GIVEN SYMBOL THE ABSCISSA VALUES ARE STORED FIRST
C                  FOLLOWED BY THE WEIGHT VALUES.
C               2) THE E AND SD VECTORS MUST BE DIMENSIONED
C                  THE SAME BY THE CALLING ROUTINE.
C
C       * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
C
        SUBROUTINE PERROR (AW, E, FINAL, FLAG, NPOINT, NS, NW, SD, START)
C
        DIMENSION  AW(1), E(1), SD(1)
        LOGICAL  FLAG
C
C       SET UP KNOWN PARAMETER(S)
C
        C = SQRT(2.0 * 3.1415926)
        FLAG = .TRUE.
        SSIZE = (FINAL - START) / FLOAT(NPOINT - 1)
        TEST = 2.0
        VMAX = 150.0
        VMIN = -150.0
C
C       CALCULATE THE STANDARD DEVIATION VECTOR
C
        SNR = START
            DO 200  I = 1, NPOINT
```

```
            SD(I) = 1.0 / SQRT(10.0 ** (SNR / 10.0))
            SNR = SNR + SSIZE
200         CONTINUE
C
C       CALCULATE THE PROBABILITY OF ERROR VECTOR
C       LOOP ENTERED ONCE FOR EACH POINT IN THE
C       PROBABILITY OF ERROR VECTOR
C
            DO 600  I = 1, NPOINT
            E(I) = 0.0
C
C           LOOP ENTERED ONCE FOR EACH SYMBOL
C
                DO 500  J = 1, NS
                SUM = 0.0
                JOFF = NW * 2 * (J - 1)
                IF (J.EQ.NS)  GO TO 320
                    DO 300  K = 1, NW
                    KOFF = K + JOFF
                    IF (AW(KOFF + NW).EQ.0.0)  GO TO 300
                    ALPHA = (1.0 - AW(KOFF)) / SD(I)
                    IF (ALPHA.LT.TEST)  FLAG = .FALSE.
                    VALUE = - ALPHA ** 2 / 2.0
                    IF (VALUE.GT.VMAX)  VALUE = VMAX
                    IF (VALUE.LT.VMIN)  VALUE = VMIN
                    ETA = EXP(VALUE)
                    SUM = SUM + (AW(KOFF + NW) * ETA) / (C * ALPHA)
300                 CONTINUE
320             CONTINUE
C
                IF (J.EQ.1)  GO TO 420
                    DO 400  K = 1, NW
                    KOFF = K + JOFF
                    IF (AW(KOFF + NW).EQ.0.0)  GO TO 400
                    ALPHA = (1.0 + AW(KOFF)) / SD(I)
                    IF (ALPHA.LT.TEST)  FLAG = .FALSE.
                    VALUE = - ALPHA ** 2 / 2.0
                    IF (VALUE.GT.VMAX)  VALUE = VMAX
                    IF (VALUE.LT.VMIN)  VALUE = VMIN
                    ETA = EXP(VALUE)
                    SUM = SUM + (AW(KOFF + NW) * ETA) / (C * ALPHA)
400                 CONTINUE
420             CONTINUE
C
                E(I) = E(I) + SUM
500             CONTINUE
600         CONTINUE
C
C       ADJUST THE PROBABILITY OF ERROR VECTOR BASED
C       ON THE NUMBER OF SYMBOLS
C
        FNS = FLOAT(NS)
            DO 700  I = 1, NPOINT
            E(I) = E(I) / FNS
700         CONTINUE
C
        RETURN
        END
```

A VOLTERRA SERIES APPROACH TO CALCULATING THE PROBABILITY OF ERROR IN A NONLINEAR DIGITAL COMMUNICATION CHANNEL

by

FREDERICK W. RATCLIFFE

B. A., LaVerne University, 1975
B. S., Kansas State University, 1977

AN ABSTRACT OF A MASTER'S REPORT

submitted in partial fulfillment of the

requirements for the degree

MASTER OF SCIENCE

Department of Electrical Engineering

KANSAS STATE UNIVERSITY
Manhattan, Kansas

1979

ABSTRACT

The principal effort of this report was the development of the Volterra series approach for calculating the error probability in nonlinear communication channels. The use of the Volterra model was shown to have a firm mathematical foundation. A series of computer programs implementing the analytical analysis were then presented. The algorithms implemented were validated against work by Benedetto, Biglieri, and Daffara [4] in modeling a channel having filters with Gaussian impulse responses. Subsequent work applied the Volterra approach to practical channels by modeling filters having causal impulse responses. To enable the system model to accept real world inputs, pulse shaping characteristics were added to the first channel filter.