

AD-A159 309

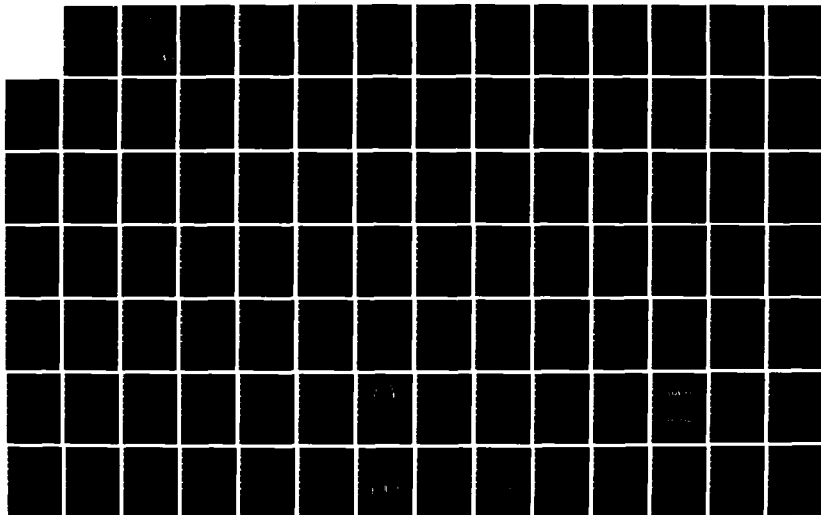
CODE SEQUENCE PERFORMANCE ANALYSIS USING  
CROSS-CORRELATION PARAMETERS IN (U) AIR FORCE INST OF  
TECH WRIGHT-PATTERSON AFB OH SCHOOL OF ENGI  
R C GONDER JUN 85 AFIT/GE/ENG/85J-1

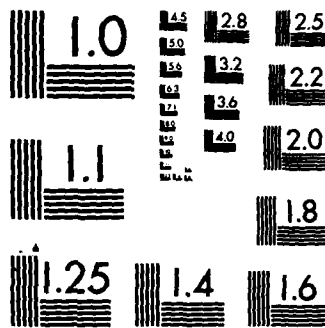
1/2

UNCLASSIFIED

F/G 17/2

NL





MICROCOPY RESOLUTION TEST CHART  
NATIONAL BUREAU OF STANDARDS-1963-A

AD-A159 309



CODE SEQUENCE PERFORMANCE ANALYSIS  
USING CROSS-CORRELATION PARAMETERS IN  
PHASE-CODED MULTIPLE ACCESS  
COMMUNICATION SYSTEMS

THESIS

Richard C. Gonder  
CPT, USA

AFIT/GE/ENG/85J-1

This document has been approved  
for public release and sale; its  
distribution is unlimited.

DEPARTMENT OF THE AIR FORCE  
AIR UNIVERSITY

**AIR FORCE INSTITUTE OF TECHNOLOGY**

Wright-Patterson Air Force Base, Ohio

DTIC  
ELECTE  
SEP 19 1985  
A

DTIC FILE COPY

85 09 17 025

AFIT/GE/ENG/85J-1

CODE SEQUENCE PERFORMANCE ANALYSIS  
USING CROSS-CORRELATION PARAMETERS IN  
PHASE-CODED MULTIPLE ACCESS  
COMMUNICATION SYSTEMS

THESIS

Richard C. Gonder  
CPT, USA

AFIT/GE/ENG/85J-1

DTIC  
ELECTE  
SEP 19 1985  
A

Approved for public release; distribution unlimited

CODE SEQUENCE PERFORMANCE ANALYSIS USING  
CROSS-CORRELATION PARAMETERS IN PHASE-CODED  
MULTIPLE ACCESS COMMUNICATION SYSTEMS

THESIS

Presented to the Faculty of the School of Engineering  
of the Air Force Institute of Technology

Air University

In Partial Fulfillment of the  
Requirements for the Degree of  
Master of Science in Electrical Engineering

Richard C. Gonder

CPT, USA

June 1985

Accession For	
NTIS GRA&I	<input checked="checked" type="checkbox"/>
DTIC TAB	<input type="checkbox"/>
Unannounced	<input type="checkbox"/>
Justification	<input type="checkbox"/>
By _____	
Distribution/	
Availability Codes	
Dist	_____
A1	_____

Approved for public release; distribution unlimited



### Acknowledgements

I would like to thank my advisor, Major Kenneth G. Castor of the Air Force Institute of Technology, for proposing this thesis topic and for his guidance. I would also like to express my appreciation to my wife, Lora, for her encouragement and support throughout the thesis effort.

## Contents

	<u>Page</u>
Acknowledgements.....	ii
List of Figures.....	v
List of Tables.....	vi
Abstract.....	vii
I. Introduction.....	I-1
II. Background.....	II-1
The Code Division Multiple Access	
System Model.....	-3
Discrete Odd and Even Correlation Functions....	-7
Worst Case Analysis.....	-10
Concept of Code Analysis Algorithm.....	-11
Spreading Sequence Code Generation -	
Pseudonoise.....	-13
Maximal Sequences.....	-14
Gold Code Generation.....	-15
Kasami Code Generation.....	-16
III. Software Overview.....	III-1
Program GenerateCodes.....	-1
GenLinear.....	-3
GenGold.....	-3
GenKasami.....	-4
Program PartialCorrelate.....	-4
Program PhaseCorrelate.....	-7
Program MassCorrelate.....	-9
IV. Code Performance Analysis.....	IV-1
Maximal Length Codes .....	-1
Gold Codes.....	-5
Kasami Codes.....	-8
Comparison of Maximal Gold and Kasami Codes....	-11
V. Software Performance.....	V-1
GenerateCodes.....	-1
Correlation Times.....	-1
Phase-Correlate Program Performace.....	-2
Thresholding Process.....	-3
Mass-correlate Performace.....	-3
VI. Conclusions and Recommendations.....	VI-1
Bibliography.....	BIB-1

Appendix A: Performance Plots.....	A-1
Appendix B: GenerateCodes Software.....	B-1
Appendix C: PartialCorrelate Software.....	C-1
Appendix D: PhaseCorrelate Software.....	D-1
Appendix E: MassCorrelate Software.....	E-1

### List of Figures

Figure		Page
II-1.	Phase Coded CDMA System Model.....	II-4
II-2.	Time Domain Analysis of Correlation Process.....	II-6
II-3.	Discrete Correlation Example.....	II-8
II-4.	Example Continuous Odd and Even Correlation.....	II-9
II-5.	Even Correlation With Threshold.....	II-12
II-6.	Amount of Code Exceeding Threshold as a Function of the Threshold.....	II-12
II-7.	Example Linear Feedback Shift Register.....	II-13
II-8.	General Linear Feedback Shift Register.....	II-14
III-1.	Structure Chart for GenerateCodes Algorithm.....	III-2
III-2.	Structure Chart for Partial Correlate Algorithm.....	III-5
III-3.	Structure Chart for PhaseCorrelate Algorithm....	III-8
III-4.	Structure Chart for MassCorrelate Algorithm.....	III-10

List of Tables

Table		Page
1.	Code Generation Polynomials for Maximal Length Code Sets.....	IV-2
2.	Code Generation Polynomials fo Gold Code Sets..	IV-6
3.	Code Generation Polynomials for Kasami Code Sets.....	IV-8
4.	Code Generator Software Performance Times.....	V-1
5.	Correlation Software Performance Times.....	V-2

### Abstract

This paper documents a technique to compare cross-correlation parameters of binary sequences used for spread-spectrum multiple-access communication systems, by performing a thresholding process on the correlation functions. The performance of Maximal length, Gold and Kasami code sequences is measured and analyzed for code lengths ranging from 63 to 1023. Comparisons of optimized codes versus unoptimized codes for each type of code sequence are analyzed in terms of the thresholding process. Comparisons are made of the performance of code sequences and code sets at lengths of 63, 127, 255 and 1023.

The software used to analyze the codes is discussed in terms of structure and performance, and is included as appendices in the thesis.

The results of this investigation indicate that the thresholding process can be used to evaluate binary sequence performance.

*Additional keywords: pseudo N-ary systems,  
2-ary. ←*

## I. Introduction

### Problem

One of the most important problems in spread-spectrum multiple-access communications is the selection of periodic sequences which have good cross-correlation properties. Previous efforts in (8,25) limit their evaluation of the code sequences to the maximum peak, or a metric of the peak, of the cross-correlation functions. The goal of this thesis effort is to investigate the parameters used to evaluate cross-correlation properties and to develop a robust technique which can be used to evaluate cross-correlation properties. To establish a baseline, calculations of parameters of known periodic sequences are compared. Maximal length, Gold and Kasami Code Sequences are used to do this.

### Scope

The thesis effort is restricted to the phase-coded spread-spectrum multiple-access systems as modeled in Chapter II. The known code sets, used for comparison, are limited to lengths less than or equal to 1023.

### Sequence of Presentation

The order of presentation in this thesis parallels the approach to the problem. Chapter II is a review of the literature and presents the background material necessary to understand the code-division multiple-access model, the correlation functions, the code analysis algorithm and the

construction of Maximal length, Gold and Kasami code sequences. Chapter III gives an overview of the software developed in this thesis effort. Chapter IV presents the analysis of the performance of the code analysis algorithm for varying lengths of Maximal length, Gold and Kasami codes and comparisons of these code sets are made. Chapter V presents the actual performance of the software developed in the thesis. Chapter VI presents the conclusions and recommendations for further study. Appendix A includes all the performance plots of the code performance algorithms. Appendices B - E include all the documented software used to generate the data needed to obtain the performance plots.

## II. Background

Spread Spectrum Communications is the term used to describe a class of transmission techniques, in which the actual signal bandwidth is much greater than that which is actually required to pass the information across the channel (10:855), and the transmitted bandwidth is not a function of the information bandwidth (1:1). The modulated signal bandwidth should be 10 - 1000 times greater than the information bandwidth (2:1). Spread Spectrum systems are divided into four categories: 1) Direct Sequence Modulation; 2) Frequency hopping; 3) Chirp Modulation; 4) Time Hopping and combinations of these four (2:1).

In Direct Sequence modulation or pseudonoise modulation a digital code sequence is used to modulate a carrier. (1:2) This code sequence must have a chip rate much faster than the bit rate of the information sequence. This operation, in effect, spreads the information over a larger bandwidth. The receiver uses the same spreading code sequence to despread the signal to obtain the original information signal which can then be demodulated as if the the direct sequence modulation had never existed on the carrier (2:2).

In Frequency Hopping, a carrier frequency is shifted through a large bandwidth as dictated by a digital code sequence (1:2). At any one instant of time, only one narrow band signal is transmitted within the broad spectrum allocated for the system. This contrasts sharply with the Direct spread system for it uses the entire spectrum at any instant of time.

### Worst Case Analysis

The discrete periodic correlations functions,  $\theta(\lambda)$  and  $\hat{\theta}(\lambda)$ , as defined by Massey and Uffner (8:539), are undefined for non integer values of  $\lambda$ . If we redefine the correlation functions as follows:

$$\theta(\tau) = \theta(\lambda) + [\theta(\lambda) + \theta(\lambda+1)](\tau-\lambda)$$

$$\hat{\theta}(\tau) = \hat{\theta}(\lambda) + [\hat{\theta}(\lambda) + \hat{\theta}(\lambda+1)](\tau-\lambda)$$

where  $\lambda$  = the integer part of  $\tau$  chips  
and  $0 \leq \tau < L$  chip times

then the discrete correlation functions are defined for the entire region  $\tau \in [0, L]$ .

By the same rationale:

$$\text{RIGHT}(\tau) = \text{RIGHT}(\lambda) + [\text{RIGHT}(\lambda) + \text{RIGHT}(\lambda+1)](\tau-\lambda)$$

$$\text{LEFT}(\tau) = \text{LEFT}(\lambda) + [\text{LEFT}(\lambda) + \text{LEFT}(\lambda+1)](\tau-\lambda)$$

where  $\lambda$  = the integer part of  $\tau$  chips  
and  $0 \leq \tau < L$  chip times

Now the continuous aperiodic correlation function can be redefined as:

$$\hat{R}_{k,i}(\tau) = \text{RIGHT}(\tau) \quad (2)$$

$$R_{k,i}(\tau) = \text{LEFT}(\tau) \quad (3)$$

And the periodic correlation functions can be redefined as:

$$\theta(\tau) = \text{RIGHT}(\tau) + \text{LEFT}(\tau) \quad (4)$$

$$\hat{\theta}(\tau) = \text{RIGHT}(\tau) - \text{LEFT}(\tau) \quad (5)$$

Substituting equations 2 and 3 into equation 1, yields:

$$v_{j,i}(\tau_j) = A_i[d_{j,-1}\text{RIGHT}(\tau_j) + d_{j,0}\text{LEFT}(\tau_j)]\cos(\theta_j - \omega_c\tau_j) \quad (6)$$

It is obvious that  $v_{j,i}(\tau_j)$  is maximized when  $\cos(\theta_j - \omega_c\tau_j) = 1$ .

correlation functions. The actual odd and even periodic functions are continuous linear functions between integer values of chip times in the region  $(0,L)$ , where  $L$  is the length of the code sequences in chips. The only discontinuities occur at the integer values of chips. It should be noted that the values obtained using this procedure are unnormalized values that are functions of the length of the codes. The actual odd and even continuous correlation functions are depicted in Figure II-4.

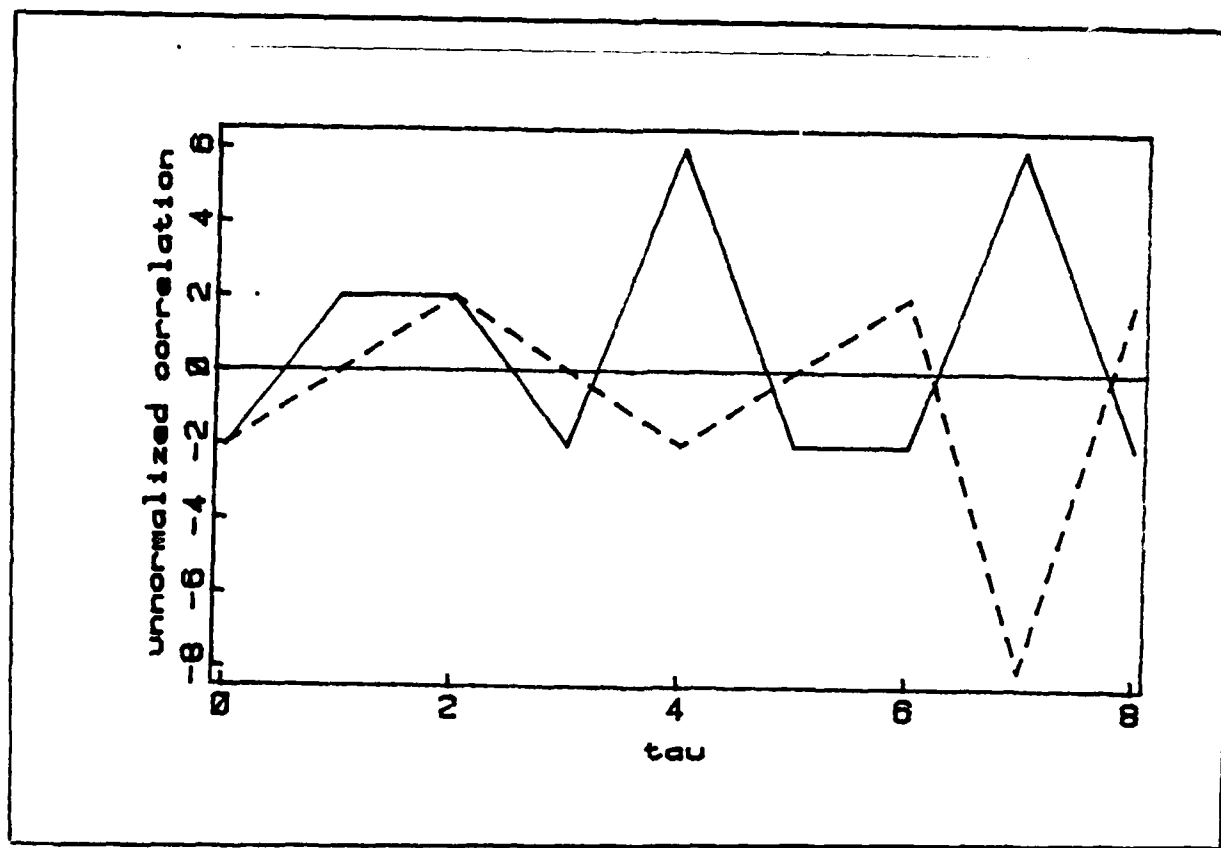


Figure II-4. Example Continuous Odd and Even Correlation

then the discrete even correlation value at that  $\ell$ , can be determined by:

- 1) Determining the number of agreements and disagreements in Region 1.
- 2) Determine the LEFT Correlation = number of agreements - number of disagreements in Region 1.
- 3) Repeat steps 1 and 2 for RIGHT correlation in Region 2.

Then the discrete even periodic correlation is:

$$\Theta(\ell) = \text{RIGHT}(\ell) + \text{LEFT}(\ell)$$

and the discrete odd periodic correlation is just:

$$\Theta(\ell) = \text{RIGHT}(\ell) - \text{LEFT}(\ell)$$

To illustrate this procedure using  $p_1$  and  $p_2$ , let the delay of  $d_2$  be  $\ell = 3$  chip times. Then the agreements and disagreements for the regions are shown in Figure II-3.

1	0	1	1	0	1	1	1	1	0	1	1	0	1	1	1
								0	1	1					
								D	D	D	A	A	D	D	A
Region 2								Region 1							

Figure II-3. Discrete Correlation Example

It can be clearly seen from Figure II-3, that the values for the procedure given a time delay of  $\ell$  are as follows:

$$\begin{aligned} \text{LEFT} &= -1 \\ \text{RIGHT} &= -1 \\ \Theta(3) &= -2 \\ \Theta(3) &= 0 \end{aligned}$$

It can be shown that both of the discrete correlation functions define endpoints to line segments of the continuous periodic

If the double frequency terms are removed by filtering (a good assumption because  $\omega_c$  is always much greater than  $1/T$  in spread spectrum multiple access systems) then the output of the  $i$ th correlator becomes:

$$Z_i = A_i \left\{ d_{i,0} T + \sum_{\substack{j=1 \\ j \neq i}}^K [d_{j,-1} R_{j,i}(\tau_j) + d_{j,0} \hat{R}_{j,i}(\tau_j)] \cos(\theta_j - \omega_c \tau_j) \right\} + \int_0^T n(t) p_j(t) \cos(\omega_c t) dt$$

When evaluating code sets for their use in CDMA systems, the term inside the summation is most important because that is where the contribution of the  $j$ th signal to the  $i$ th correlator exists. This term provides the amount of interference from the  $j$ th user in the  $i$ th user's communication channel. It will be defined as this:

$$v_{j,i}(\tau_j) = A_i [d_{j,-1} R_{j,i}(\tau_j) + d_{j,0} \hat{R}_{j,i}(\tau_j)] \cos(\theta_j - \omega_c \tau_j) \quad (1)$$

#### Discrete Odd and Even Correlation Functions

For mathematical purposes, if the spreading sequences which are bipolar NRZ signals, are mapped from  $[-1,1]$  into  $[1,0]$ , then the correlation of two codes can be done using modulo-2 arithmetic with equivalent results. The use of this isomorphism makes the problem easier to present as well as easier to apply to any algorithm. As an example, given the two code sequences  $p_1$  and  $p_2$  represented in vector form as:

$$p_1 = [1, 0, 1, 1, 0, 1, 1, 1]$$

$$p_2 = [0, 1, 0, 1, 1, 0, 1, 1]$$

and given a delay,  $\lambda$ , equal to an integer amount of chip times,

If  $d_{j,0} = d_{j,1}$  then

$$v_{j,i}(\tau_j)_{\max} = d_{j,0}[\text{RIGHT}(\tau_j) + \text{LEFT}(\tau_j)] = d_{j,0}\theta(\tau_j) \quad (7)$$

and if  $d_{j,0} \neq d_{j,1}$  then

$$v_{j,i}(\tau_j)_{\max} = d_{j,0}[\text{RIGHT}(\tau_j) - \text{LEFT}(\tau_j)] = d_{j,0}\hat{\theta}(\tau_j) \quad (8)$$

Normally, encoded data sequences have the property that  $\text{Pr}(d_j=1) = \text{Pr}(d_j = -1) = .5$ , then

$$v_{j,i}(\tau_j)_{\max} = \begin{cases} \pm\theta(\tau_j) \\ \text{or} \\ \pm\hat{\theta}(\tau_j) \end{cases}$$

#### Concept of the Code Analysis Algorithm

Figure II-5 displays the  $v_{j,i}(\tau_j)_{\max}$  for  $d_{j,0} = 1$ , using the example presented earlier, with a threshold set at 1. If  $\tau_j$  is considered to be uniformly random between the interval 0 - L chips, then a threshold as displayed in the figure can be adjusted to characterize this function. Using the previous example at a threshold = 1, the amount of the correlation function which exceeds the threshold is depicted as the shaded region on the figure. It is computed using similar triangle relationships to be a value of 0.5 when normalized by the length of the code. If the threshold moved from 0 to L in the unnormalized function, a set of data, Threshold vs Amount Exceeding Threshold, can be easily obtained. To include the effect of the sign of function in Equations 7 and 8, a second threshold of equal magnitude and opposite sign can be adjusted accordingly, to obtain a similar data set. If the two data sets are summed, then the Threshold vs

### Worst Case Analysis

The discrete periodic correlation functions,  $\theta(l)$  and  $\hat{\theta}(l)$ , as defined by Massey and Uhran (8:539), are undefined for non integer values of  $l$ . If we redefine the correlation functions as follows:

$$\theta(\tau) = \theta(l) + [\theta(l) + \theta(l+1)](\tau-l)$$

$$\hat{\theta}(\tau) = \hat{\theta}(l) + [\hat{\theta}(l) + \hat{\theta}(l+1)](\tau-l)$$

where  $l$  = the integer part of  $\tau$  chips  
and  $0 < \tau < L$  chip times

then the discrete correlation functions are defined for the entire region  $\tau \in [0, L]$ .

By the same rationale:

$$\text{RIGHT}(\tau) = \text{RIGHT}(l) + [\text{RIGHT}(l) + \text{RIGHT}(l+1)](\tau-l)$$

$$\text{LEFT}(\tau) = \text{LEFT}(l) + [\text{LEFT}(l) + \text{LEFT}(l+1)](\tau-l)$$

where  $l$  = the integer part of  $\tau$  chips  
and  $0 < \tau < L$  chip times

Now the continuous aperiodic correlation function can be redefined as:

$$\hat{R}_{k,i}(\tau) = \text{RIGHT}(\tau) \quad (2)$$

$$R_{k,i}(\tau) = \text{LEFT}(\tau) \quad (3)$$

And the periodic correlation functions can be redefined as:

$$\theta(\tau) = \text{RIGHT}(\tau) + \text{LEFT}(\tau) \quad (4)$$

$$\hat{\theta}(\tau) = \text{RIGHT}(\tau) - \text{LEFT}(\tau) \quad (5)$$

Substituting equations 2 and 3 into equation 1, yields:

$$v_{j,i}(\tau_j) = [d_{j,-1}\text{RIGHT}(\tau_j) + d_{j,0}\text{LEFT}(\tau_j)]\cos(\theta_j - \omega_c \tau_j) \quad (6)$$

It is obvious that  $v_{j,i}(\tau_j)$  is maximized when  $\cos(\theta_j - \omega_c \tau_j) = 1$ .

correlation functions. The actual odd and even periodic functions are continuous linear functions between integer values of chip times in the region  $(0,L)$ , where  $L$  is the length of the code sequences in chips. The only discontinuities occur at the integer values of chips. It should be noted that the values obtained using this procedure are unnormalized values that are functions of the length of the codes. The actual odd and even continuous correlation functions are depicted in Figure II-4.

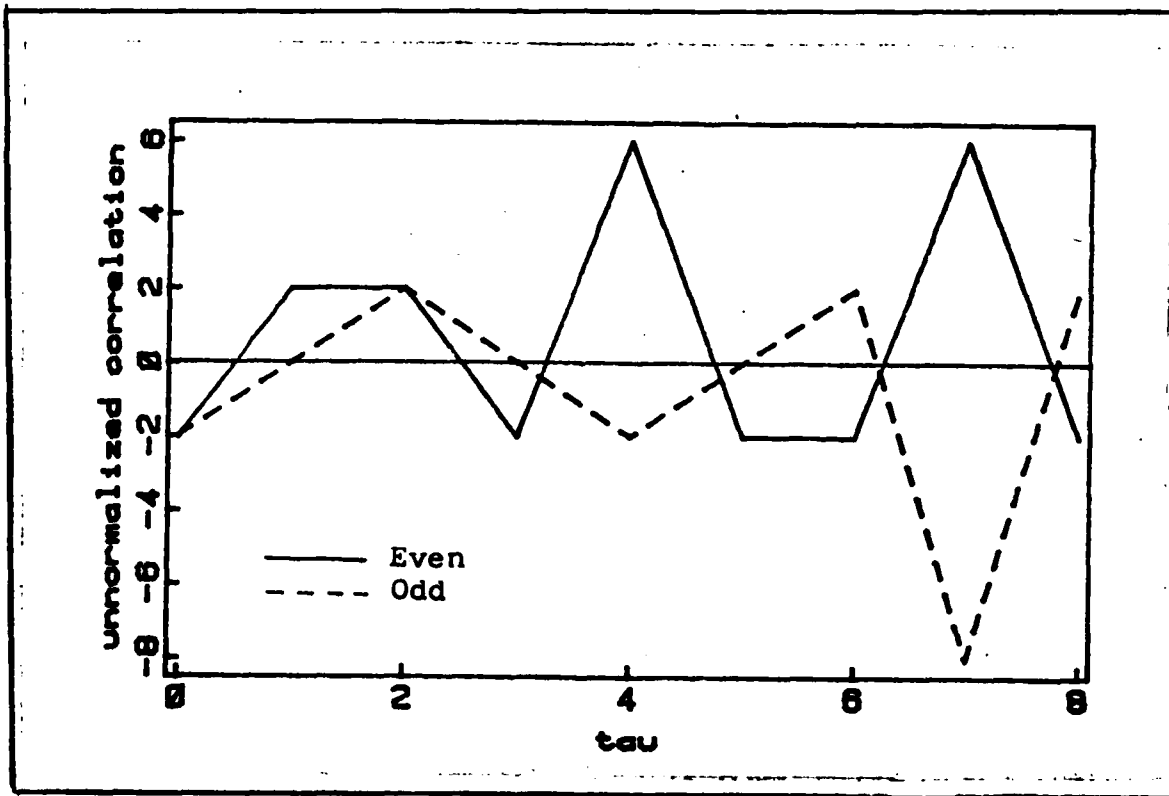


Figure II-4. Example Continuous Odd and Even Correlation

- 1) Determining the number of agreements and disagreements in Region 1.
- 2) Determine the LEFT Correlation = number of agreements - number of disagreements in Region 1.
- 3) Repeat steps 1 and 2 for RIGHT correlation in Region 2.

$$\theta(l) = \text{RIGHT}(l) + \text{LEFT}(l)$$
$$\hat{\theta}(\ell) = \text{RIGHT}(\ell) - \text{LEFT}(\ell)$$

1	0	1	1	0	1	1	1	1	0	1	1	0	1	1	1
$\frac{0 \quad 1 \quad 0 \quad 1 \quad 1}{D \quad D \quad D \quad A \quad A}$								$\frac{0 \quad 1 \quad 1}{D \quad D \quad A}$							
Region 2								Region 1							

It can be clearly seen from Figure II-3, that the values for the procedure given a time delay of 1 are as follows:

```

LEFT  = -1
RIGHT = -1
θ(3)  = -2
θ(3)  = 0

```

II-8

If the double frequency terms are removed by filtering (a good assumption because  $\omega_c$  is always much greater than  $1/T$  in spread spectrum multiple access systems), then the output of the  $i$ th correlator becomes (letting  $A_i = 1$ ,  $i = 1, 2, \dots, K$ ):

$$Z_i = \{d_{i,0}T + \sum_{\substack{j=1 \\ j \neq i}}^K [d_{j,-1}R_{j,i}(\tau_j) + d_{j,0}\hat{R}_{j,i}(\tau_j)]\cos(\theta_j - \omega_c\tau_j)\} + \int_0^T n(t)p_j(t)\cos(\omega_c t) dt$$

When evaluating code sets for their use in CDMA systems, the term inside the summation is most important because that is where the contribution of the  $j$ th signal to the  $i$ th correlator exists. This term provides the amount of interference from the  $j$ th user in the  $i$ th user's communication channel. It will be defined as this:

$$v_{j,i}(\tau_j) = [d_{j,-1}R_{j,i}(\tau_j) + d_{j,0}\hat{R}_{j,i}(\tau_j)]\cos(\theta_j - \omega_c\tau_j) \quad (1)$$

#### Discrete Odd and Even Correlation Functions

For mathematical purposes, if the spreading sequences which are bipolar NRZ signals, are mapped from  $[-1,1]$  into  $[1,0]$ , then the correlation of two codes can be done using modulo-2 arithmetic with equivalent results. The use of this isomorphism makes the problem easier to present as well as easier to apply to any algorithm. As an example, given the two code sequences  $p_1$  and  $p_2$  represented in vector form as:

$$p_1 = [1,0,1,1,0,1,1,1]$$

$$p_2 = [0,1,0,1,1,0,1,1]$$

and given a delay,  $\lambda$ , equal to an integer amount of chip times,

$$R_{k,i}(\tau) = \int_0^{\tau} p_k(t - \tau) p_i(t) dt \quad (\text{region 1})$$

$$\hat{R}_{k,i}(\tau) = \int_{\tau}^T p_k(t - \tau) p_i(t) dt \quad (\text{region 2})$$

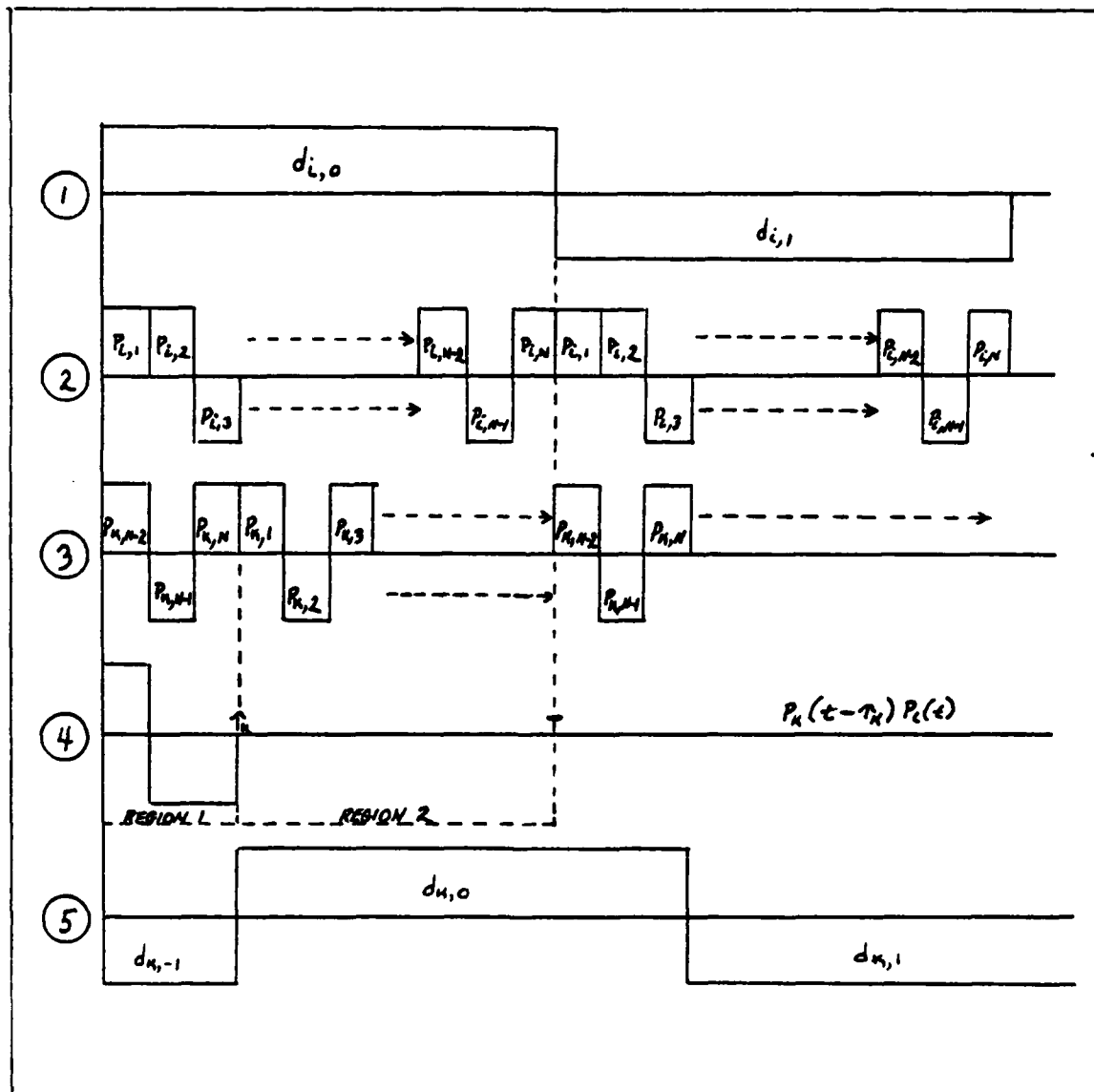


Figure II-2. Time-Domain Analysis of Correlation Process

$$r(t) = n(t) + A_i d_i(t) p_i(t) \cos(\omega_c t) +$$

$$\sum_{\substack{j=1 \\ j \neq i}}^K A_j p_j(t - \tau_j) d_j(t - \tau_j) \cos(\omega_c t + \theta_j - \omega_c \tau_j)$$

where  $A_i$  = amplitude of the  $i$ th carrier  
 $d_i(t)$  = polar NRZ message signal of the  $i$ th user  
 $p_i(t)$  = polar NRZ spreading sequence of the  $i$ th user  
 $\theta_i$  = phase of the  $i$ th carrier varying from  $0 - 2\pi$  uniformly  
 $\tau_i$  = time delay of the  $i$ th user varying from  $0 - T$  uniformly  
 $T$  = data symbol length in time duration  
 $n(t)$  = additive white Gaussian noise with two-sided spectral density  $N_0/2$

If the received signal is input to the integrate and dump correlator matched to the  $i$ th signal  $s_i(t)$ , then the output of this correlator at time  $T$  is given by:

$$Z_i = \int_0^T r(t) p_i(t) \cos(\omega_c t) dt$$

To illustrate what is going on in the system at this point refer to Figure II-2 where the top diagram depicts the two data stream bits,  $d_{i,0}$  and  $d_{i,1}$ . All of the other diagrams in this figure are in time reference to the top diagram which, as mentioned earlier, is in synchronization with the  $i$ th correlator. Diagram 5 depicts the  $k$ th data stream which is delayed by  $\tau_k$  which for this example is equal to 3 chip times of the spreading sequences. Diagram 4 depicts the product of the  $i$ th and  $k$ th spreading sequences,  $p_k(t - \tau_k) p_i(t)$ , and is segmented in time into two regions: Region 1, from  $t = 0$  to  $t = \tau_k$ , and Region 2, from  $t = \tau_k$  to  $t = T$ . The integrations of these two regions in time produce the continuous-time partial cross-correlation functions:

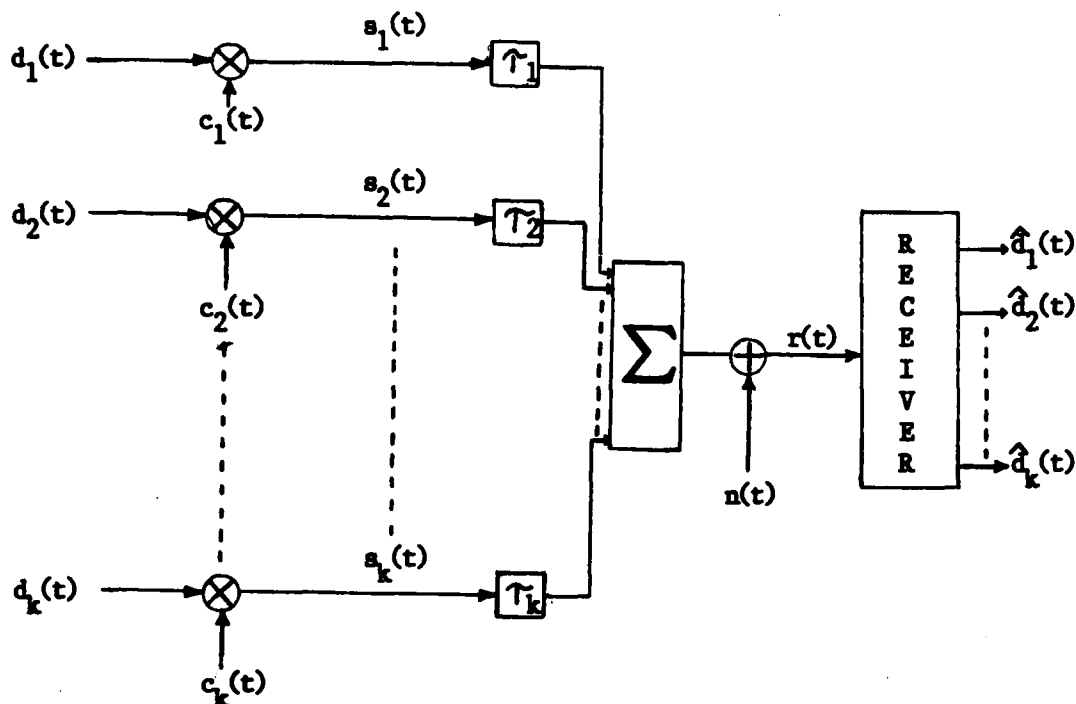


Figure II-1. Phase-coded CDMA System Model (12:797)

The data signal  $d_k(t)$  is modulated on the phase coded carrier  $c_k(t)$  to produce the transmitted signal  $s_k(t)$ , each given by:

$$c_k(t) = A_k p_k(t) \cos(\omega_c t + \theta_k)$$

$$s_k(t) = A_k p_k(t) d_k(t) \cos(\omega_c t + \theta_k)$$

The received signal  $r(t)$  is given by

$$r(t) = n(t) + \sum_{j=1}^K A_j p_j(t - \tau_j) d_j(t - \tau_j) \cos(\omega_c t + \theta_j - \omega_c \tau_j)$$

If it is assumed that the  $i$ th correlator is synchronized with the  $i$ th signal, i.e. that  $\tau_i = 0$  and the phases are referenced to signal 1, i.e.  $\theta_i = 0$ , then the received signal at the  $i$ th correlator becomes:

situations where interference and multipath problems exist, there are significant degradations in both the FDMA and TDMA systems. For reasons such as these, code division multiple accessing schemes have evolved. The CDMA systems allow system users to transmit simultaneously in time and occupy the same RF bandwidth (1:93). Generally, the CDMA system operates asynchronously to eliminate system timing problems. However, the imperfect orthogonality of the different spreading sequences reduces the capacity of the CDMA system. An important issue confronting system designers is design of sets of spreading sequences that operate efficiently in this system. The correlation properties of these sets of sequences have a major impact on the capacity of this type of system and its overall performance.

#### The Code Division Multiple Access System Model

The basic model of the Code Division Multiple Access system is depicted in Figure II-1. In this system, each of the  $K$  users is given its own spreading code sequence which has "orthogonal like" properties with the other assigned users. Because of the fact that any user can initiate or terminate its traffic at any time in an asynchronous system, it is more difficult to design and evaluate different code sets for use in the system. It is desirable to obtain code sets with good cross-correlation and partial-correlation properties (12:795). This fact will become evident as the model is explained in the following pages.

Chirp Modulation is most commonly used in radar, which transmits swept frequency pulses. The receiver uses a dispersive filter to compress the signal into a narrower time slot, so it behaves in the same way a high power pulse does (2:3).

In Time hopping, the transmission time and period is controlled with the digital code sequence. When used by itself, Time hopping can be viewed as pulse modulation under code sequence control (2:3).

Spread Spectrum systems may offer any or combinations of the following benefits: 1) Simultaneous use of the spectrum by multiple users; 2) Interference rejection caused by intentional and unintentional sources such as jamming and multipath; 3) Low density output signals providing a lower probability of interception; 4) High resolution ranging; 5) Accurate universal timing; and 6) Inherent message privacy (1,2).

The disadvantages of using spread spectrum are the increased complexity of the system and increased difficulty in allocation of frequencies (2:4).

The effort of this thesis is concentrated in the Direct Sequence Spread Spectrum area and the advantage of multiple access communications. Multiple access communications systems allow simultaneous access to the channel to many users. The three most common multiple access techniques are time division multiple access (TDMA), frequency division multiple access (FDMA), and code-division multiple access (CDMA). TDMA and FDMA separate the signals in time and frequency domains respectively. CDMA employs spread spectrum techniques (SSMA), and provides another way to separate the signals at the receiver (10:868). In

If  $d_{j,0} = d_{j,1}$ , and  $A_i = 1$  then

$$v_{j,i}(\tau_j)_{\max} = d_{j,0}[\text{RIGHT}(\tau_j) + \text{LEFT}(\tau_j)] = d_{j,0}\theta(\tau_j) \quad (7)$$

and if  $d_{j,0} \neq d_{j,1}$ , and  $A_i = 1$  then

$$v_{j,i}(\tau_j)_{\max} = d_{j,0}[\text{RIGHT}(\tau_j) - \text{LEFT}(\tau_j)] = d_{j,0}\hat{\theta}(\tau_j) \quad (8)$$

Normally, encoded data sequences have the property that  $\Pr(d_j=1) = \Pr(d_j = -1) = .5$ , then

$$v_{j,i}(\tau_j)_{\max} = \begin{cases} \pm\theta(\tau_j) \\ \text{or} \\ \pm\hat{\theta}(\tau_j) \end{cases}$$

#### Concept of the Code Analysis Algorithm

Figure II-5 displays the  $v_{j,i}(\tau_j)_{\max}$  for  $d_{j,0} = 1$ , using the example presented earlier, with a threshold set at 1. If  $\tau_j$  is considered to be uniformly random between the interval 0 - L chips, then a threshold as displayed in the figure can be adjusted to characterize this function. Using the previous example at a threshold = 1, the amount of the correlation function exceeding the threshold is depicted as the shaded region on the figure. It is computed using similar triangle relationships to be a value of 0.5 when the length of the code is normalized. If the threshold moved from 0 to L in the unnormalized function, a set of data, Threshold vs Amount Exceeding Threshold, can be easily obtained. To include the effect of the sign of function in Equations 7 and 8, a second threshold of equal magnitude and opposite sign can be adjusted accordingly, to obtain a similar data set. If the two data sets are summed, then the Threshold vs

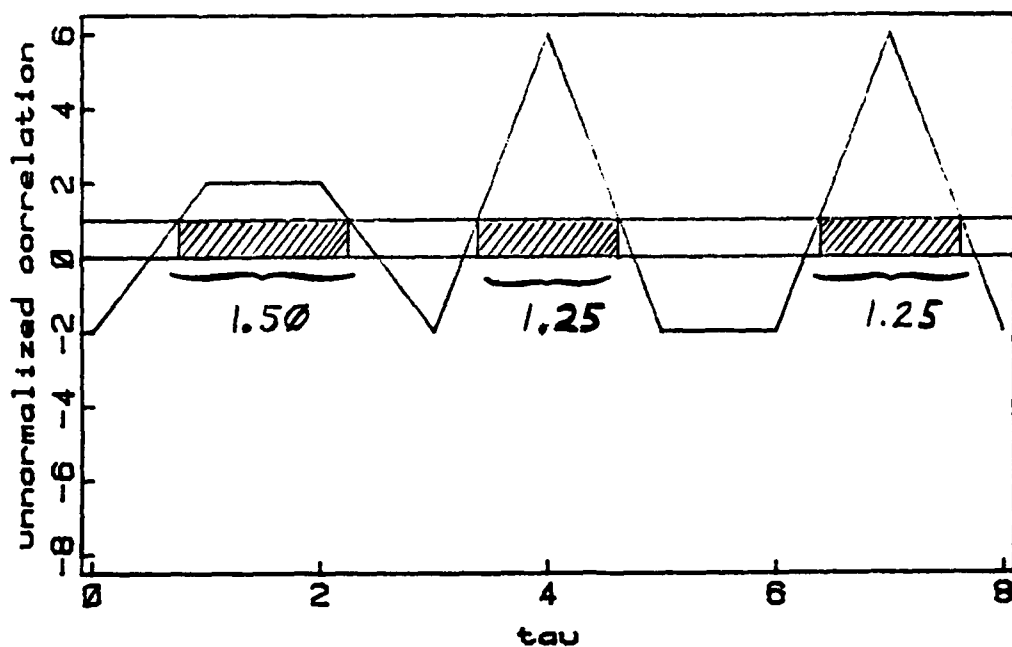


Figure II-5. Even Correlation  $\theta_{j,i}(\tau_j)$  with Threshold

Amount Exceeding Threshold for the previous example produces plot shown in Figure II-6.

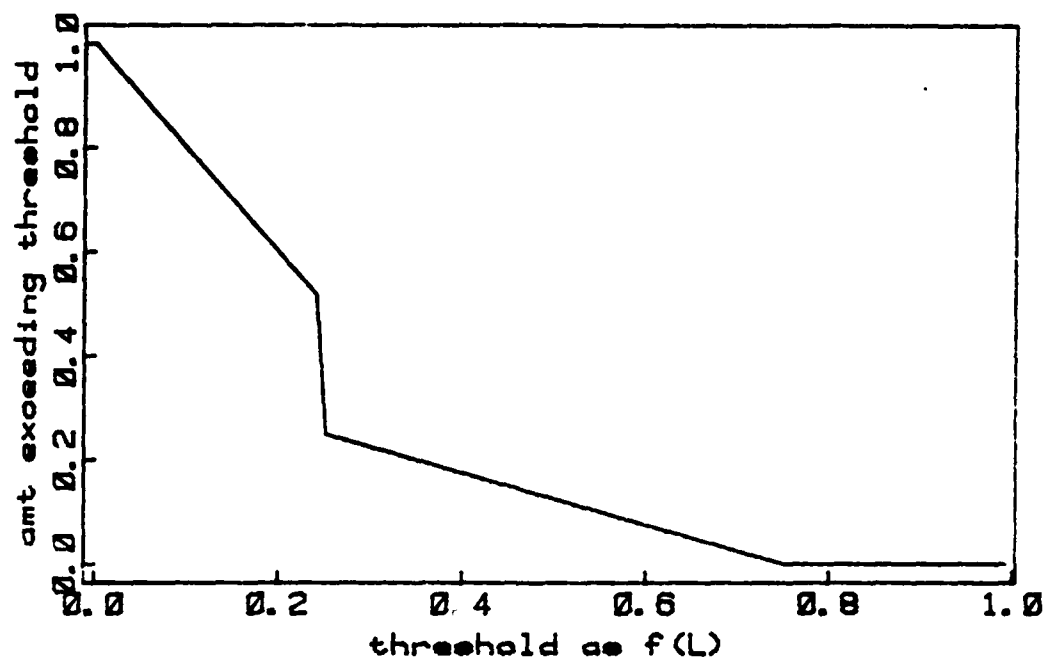


Figure II-6 Amount of Code Exceeding Threshold As a Function of the Threshold

### Spreading Code Sequence Generation

Pseudonoise (PN) sequences or Linear Feedback Shift Register (LFSR) sequences are easily generated at the transmitter and the receiver. The LFSR, as shown in Figure II-7 is composed of binary storage elements, taps for reading storage contents, and a device that performs the modulo-2 addition of the tapped storage elements. The binary storage elements must be initially loaded with 0's and 1's (not all 0). As the LFSR is clocked, the modulo-2 addition device generates the output of the code generator as well as the input to the first binary storage location.

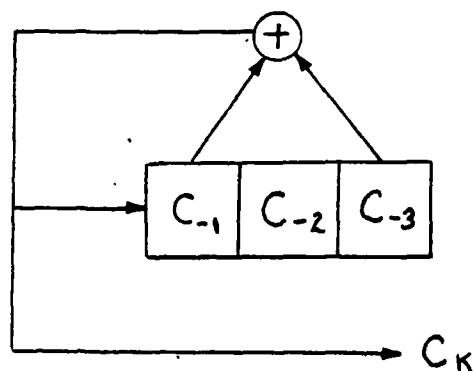


Figure II-7. Example Linear Feedback Shift Register

This LFSR is also known as a Simple Shift Register Generator (SSRG) because all feedback signals are returned to a single input. If the initial loading of the generator is all 0's with the exception of the last position, then the output sequence will not have any transient bits. The general form of the LFSR is depicted in Figure II-8 which has  $N$  storage locations and taps feeding into one summation device.

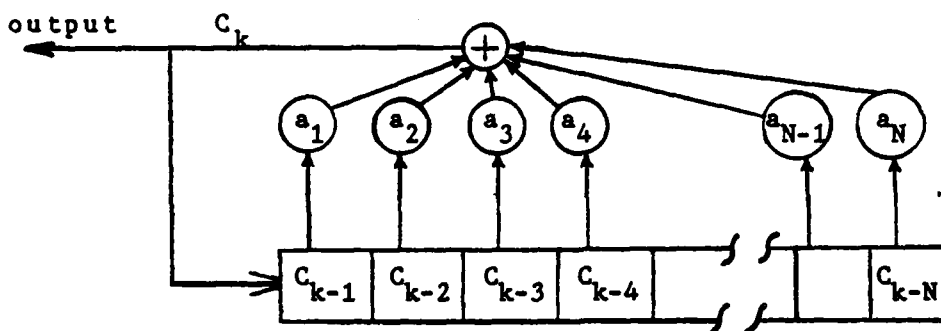


Figure II-8. General Linear Feedback Shift Register (1:30).

The fundamental linear recursion relation which provides the code sequence element  $C_k$  is written as:

$$C_k = \sum_{i=1}^N a_i C_{k-i} \quad (\text{refs 1,3})$$

### Maximal Sequences

Maximal length sequences are produced using the LFSR in Figure II-8 using a primitive polynomial of degree  $N$  to designate which tap positions will be used. They have a maximal length period of  $2^N - 1$  chips. The maximal sequences have the following properties:

- 1) Balance property - There are  $2^{N-1}$  ones and  $2^{N-1} - 1$  zeroes
- 2) Run property - In any period, half of the runs of 1's or 0's is of length 1,  $1/4$  are of length 2,  $1/8$  are of length 3 etc.
- 3) The autocorrelation function  $\Theta(\tau)$  is

$$\Theta(\tau) = \begin{cases} L & \tau = 0, L, 2L \dots \text{chips} \\ -1 & \text{for } \tau < L \text{ and } \tau < L-1 \text{ chips} \\ (\tau-L)(L-1) & \text{for } -1 < \tau < L, \quad l = \text{integer part of } \tau \text{ for each period of the function (refs 1,4,10).} \end{cases}$$

### Gold Code Generation

Gold codes can be generated using one period of the output sequences of two linear feedback shift registers, which have tap assignments designated by preferred pair polynomials. The preferred pair must be of degree  $N$ , such that  $N$  is not divisible by 4. If the output sequences are modulo-2 added to one another for each possible phase difference, then  $L$  different Gold sequences are produced. If the original two sequences are included then this process yields a set of  $L+2$  Gold sequences (ref 1,4,5,11). To illustrate by the following example:

$f_1(x) = x^5 + x^2 + 1$  is a primitive polynomial of degree 5

To obtain the preferred pair, Gold's theorem states that:

Given  $f_1(x)$  is a primitive polynomial of degree  $N$  such that  $N$  is not divisible by 4 then let

$$t = \begin{cases} 1 + 2^{(N-1)/2} & N \text{ odd} \\ 1 + 2^{(N-2)/2} & N \text{ even} \end{cases}$$

then  $f_2(x)$  is the minimal polynomial of  $\alpha^t$

In this example  $t = 1 + 2^{(5-1)/2} = 5$

The minimal polynomial of  $\alpha^5$  of degree 5 is:

$$f_2(x) = x^5 + x^4 + x^2 + x + 1$$

Gold's theorem also states that using a preferred pair of primitive polynomials, of degree  $N$ , whose LFSR generator sequences are maximal length with period  $L = 2^N - 1$ , to produce a set,  $C$ , of  $L + 2$  sequences of length  $L = 2^N - 1$ . The discrete even cross correlation function:

$$|\theta_{1,j}(k)| < \begin{cases} 1 + 2^{(N+1)/2} & N \text{ odd} \\ 1 + 2^{(N+2)/2} & N \text{ even} \end{cases}$$

and  $\theta_{i,j}(k) =$  one of the three  $[-1, -t, t-2]$   
for  $k$  an integer in  $[0, L]$   
(refs 1,4,5,11).

For this example,  $|\theta_{i,j}(k)| < 1 + 2^{(5+1)/2} = 9$ , and the three values that it obtains are  $[-1, -6, 8]$ .

### Kasami Code Generation

A set of Kasami codes can be generated using the maximal length sequence generated in the LFSR of even length. Using the decimation of this sequence by  $2^{N/2} + 1$  to produce a second sequence with length  $2^{N/2} - 1$ . A third sequence of length  $L$ , period  $2^{N/2} - 1$ , is produced by the second sequence. The modulo-2 summation of the first and third sequence through the  $2^{N/2} - 2$  different phase shifts of the third sequence will produce a set of  $2^{N/2} - 1$  sequences. If the first sequence is included in the set, then the entire Kasami set generated will have  $2^{N/2}$  sequences (11:569). The following example will illustrate this procedure: Given the maximal sequence generated by the primitive polynomial  $x^6 + x + 1$  is

111111010101100110111011010010011100010111100101000110000100000

the decimation by 9 yields the second sequence

1110010

the third sequence is made by the second to yield

111001011100101110010111001011100101110010111001011100101110010

The modulo addition through the 6 phase shifts produces 7 different sequences. The entire Kasami set includes the first

sequence for a total of 8 sequences. Like the Gold Sequences, the discrete even autocorrelation and cross-correlation are 3 valued. However this set has  $[-1, -(2^{N/2} + 1), 2^{N/2} - 1]$  and for this example have values  $[-1, -9, 7]$ . Hence the discrete even cross-correlation function is

$$|\theta_{i,j}(k)| \leq 2^{N/2} + 1 \quad N \text{ even only}$$

### III. Software Overview

This chapter provides a general discussion of the software designed in the thesis, in the order as the fully documented programs appear in Appendices B, C, D, and E. This discussion is presented at the structure-function level for understandability. If the reader desires more detailed discussion of each module presented here, he should refer to these Appendices. It should be noted that the software was designed and tested using the Turbo-Pascal Compiler, version 2.0, from Borland International. This compiler accepts all standard Pascal types and identifiers and has many non-standard types and identifiers available to the user as well. For transportability purposes, the software was written such that few nonstandard types are used. When they were used, a comment identifying its use, directly follows the code as it appears in the appendices. It is worth noting that the EOLN and EOF functions are performed differently in many compilers, and for this reason, any modules performing character I/O may have to be modified accordingly. The programs were run using the Berkely Pascal Compiler, and required additional procedures for string I/O because a string is a non-standard type.

#### Program GenerateCodes

Figure III-1 depicts the structure of the code generating program, GenerateCodes. It uses linear feedback shift registers (LFSR's) constructed from an array of characters, 1's and 0's, to generate Maximal Length Codes, Gold Codes, and Kasami Codes. The outputted sequences are stored in user designated character

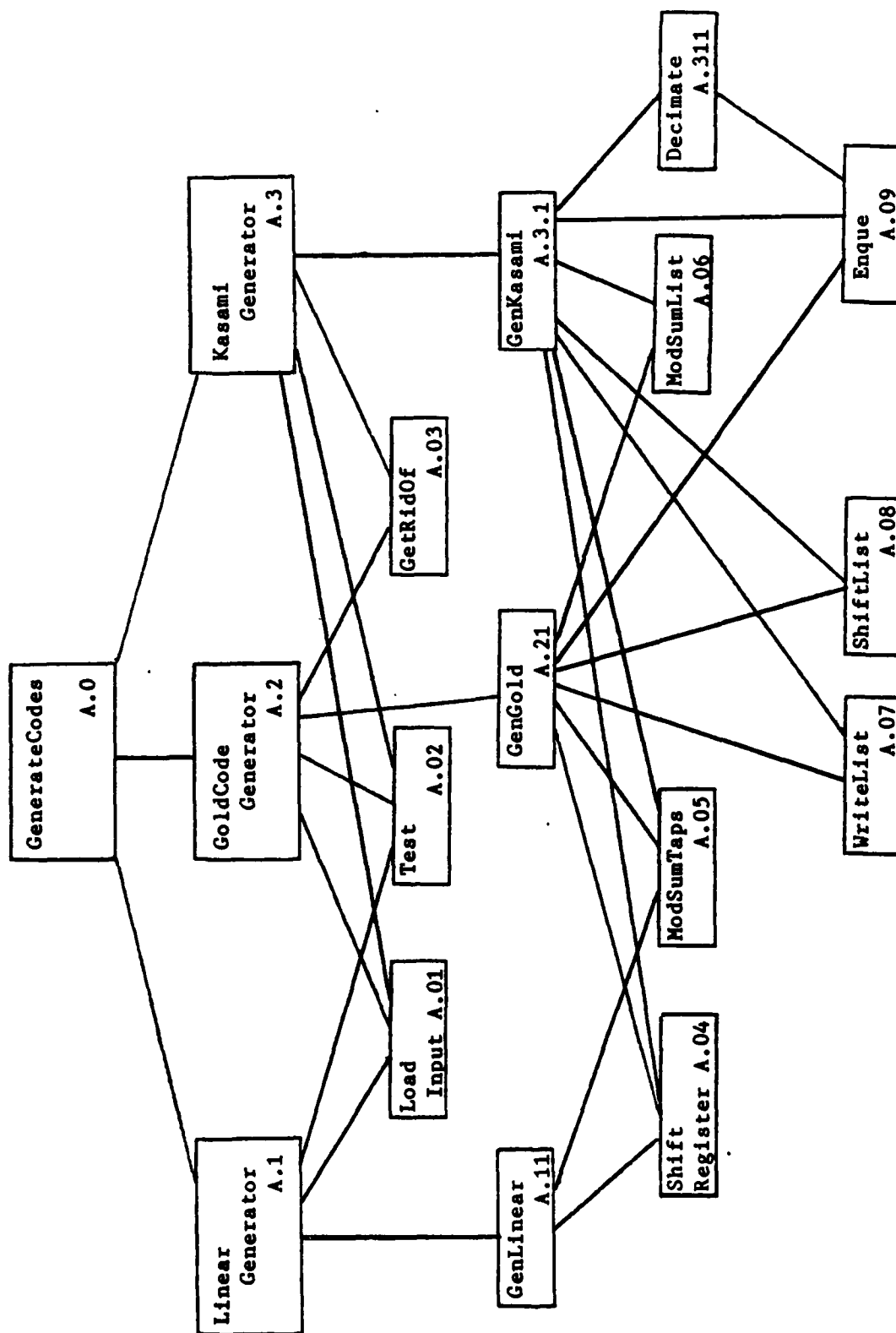


Figure III-1. Structure Chart for GenerateCodes Algorithm

files, with modified extensions for each code generated within a code set (i.e. .001, .002, for the first and second code generated in a code set). The main module queries the user to determine which type of code to generate, and calls one of three modules; LinearGenerator, GoldCodeGenerator, and KasamiGenerator. Each of these performs most of the user I/O required to generate the codes. They each make calls to LoadInput and Test, which make tap assignments and validate the inputted filenames respectively. The GoldCode Generator and the Kasami Generator use the function GetRidOf to free the memory of linked lists, representing code-lists, that each generator no longer requires for generation.

GenLinear. The GenLinear module is called by the LinearGenerator after the input has been validated. It loads an array, representing the LFSR, with all 0's and a 1 in position  $C_N$ , as described in Chapter II. It calls module ModSumTaps to do the modulo-2 addition of the array elements, identified by the tap assignments, to produce the output bit, CK. CK is written to the code file and is passed to the module ShiftRegister, which shifts the contents of the LFSR by one position, and loads CK into the  $C_{K-1}$  position. This procedure is continued until  $2^N - 1$  bits have been generated.

GenGold. The GenGold module is called by the GoldCodeGenerator after the tap assignments have been made for two LFSR's and the filename has been validated. The user identifies the number of sequences he wants generated from 1 to  $2^N + 1$ . It produces two LFSR sequences in the same manner as the GenLinear module, using modules ModSumTaps and ShiftRegister, but places

the two output sequences into linked lists in memory, using module Enque. The first two sequences generated are written to code files with extensions .001 and .002 using the Writelist module. The third sequence is generated by calling module ModSumList which performs a modulo-2 addition of the two lists and outputs the result into the third file. Module ShiftList is called to perform a cyclic shift of the first code list. The forth through the  $2^N + 1$  sequence is generated by calling modules ShiftList and ModSumList.

GenKasami. The GenKasami module is called when the module Kasami- Generator has made the tap assignments for one LFSR and validated the inputted filename. It generates an LFSR sequence in the same manner as the GenLinear module but stores the output sequence in a linked list in memory. It calls module Decimate to perform a decimation of the LFSR sequence by a factor  $N + 2$ , and produce a second linked list in memory, using module Enque. The second list is then used to produce a third list, with period of the second list and length of the first list, using module Enque. The first code sequence is generated by writing the first list to file, with the extension .001, using module Writelist. Successive sequences are generated using ModSumList and ShiftList, as described previously. Depending on the user's input, the GenKasami module will produce from 1 to  $2^{N/2}$  sequences.

#### Program PartialCorrelate

The PartialCorrelate program, as depicted in Figure III-2, will perform correlations on user designated code files composed of ASCII 1's and 0's. The code files are read into linked-lists

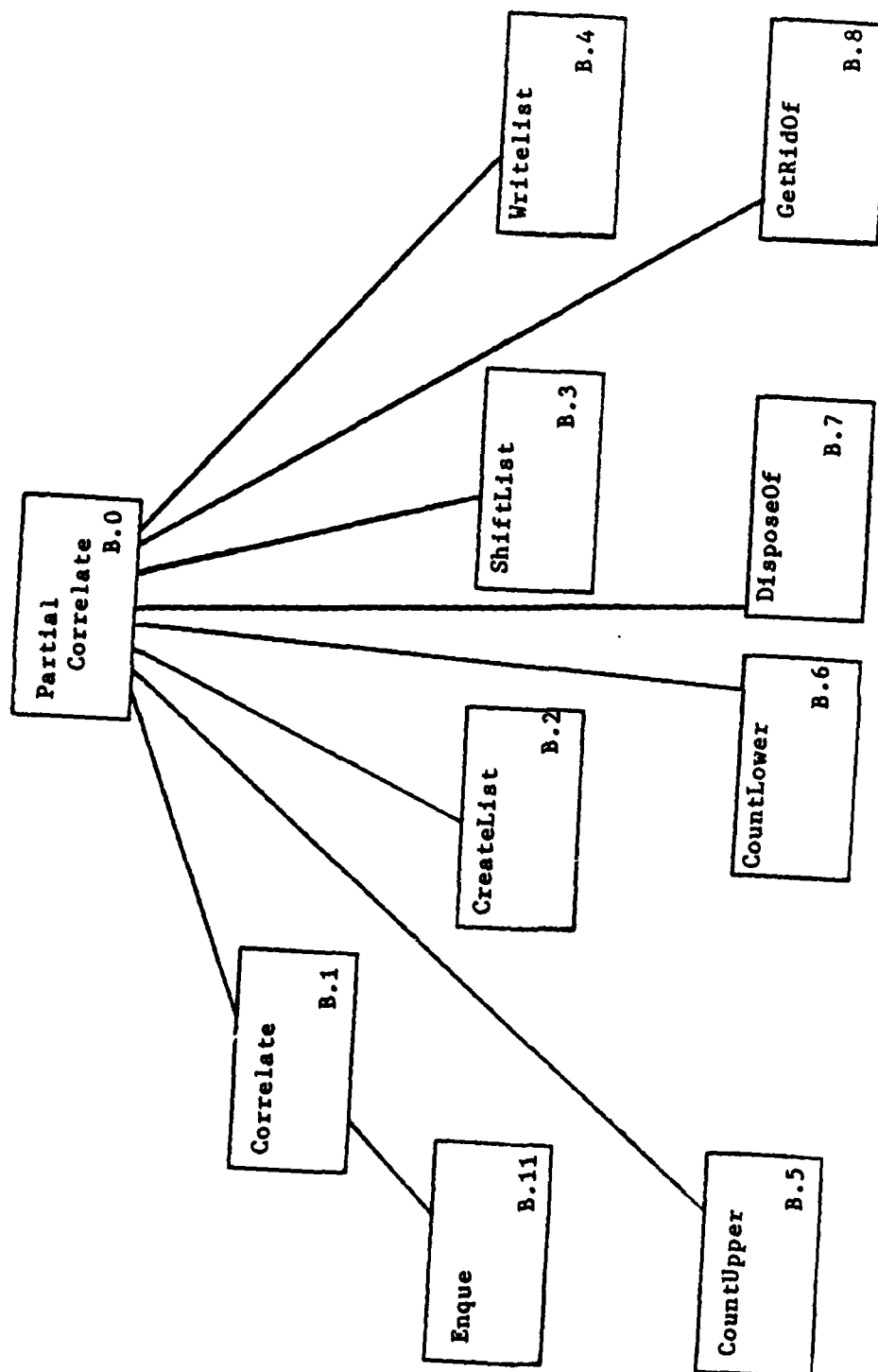


Figure III-2. Structure Chart for Partial Correlate Algorithm

using the module CreateList. The program has many options available to the user that are presented in a menu driven fashion. In all options the actual discrete Odd, and Even correlation functions are computed using the Correlate module, for each integer value of tau in the interval  $[0, L]$ . The correlation function values are placed into an ordered linked-list of records using the module Enque. If the user desires, this list of data can be written to file using the module Writelist. The module ShiftList is only used when the user wants to evaluate different phase shifts of the original codes. When called it will shift both code files a designated number of bits before correlating. If desired, the thresholding process, as described in chapter II, can be performed on the correlation functions. The user has the option of thresholding any or both of the correlation functions using a positive threshold, negative threshold or both. The positive threshold process is performed in the CountUpper module, while the negative threshold process is performed in the CountLower module. The main module calls the modules, CountUpper and CountLower, and passes a specific threshold value to obtain a value for the amount of the correlation function exceeding the threshold. The threshold values passed in, vary from 0 to the length of the codes, and are incremented a fixed amount until the threshold value has reached the length of the code or the amount exceeding the threshold goes to 0. During each cycle of this process, the normalized threshold value and the amount of the correlation function exceeding the threshold are written to a user designated file. The modules DisposeOf and GetRidOf are used to free the memory of the code lists and the correlation

lists, when their use is no longer needed.

#### Program PhaseCorrelate

The PhaseCorrelate program determines the best and worst phases of the two inputted codes in terms of the Odd correlation function. The structure of the program is shown in Figure III-3. The program reads the code sequences into linked-lists using the module CreateList. The program will perform correlations of the user designated code files using module Correlate, and determine the minimum and maximum values of the Odd function for a certain phase in module Minimax. The program repeats this process for every phase shift of the code files, and uses module Enque to place these values into another linked-list. The ShiftList module cyclically shifts the code lists appropriately for each phase prior to commencing the correlation process. After correlations have been performed on all phase shifts of the code files, then the module Maximimize is called. This module will traverse the linked-list of minimum and maximum Odd values to determine the maximum and minimum spread for all phases. The main program will then traverse the linked list and write the phase, maximum and minimum Odd values for each phase shift that has a Maximum or Minimum spread. Just as before, the DisposeOf and GetRidOf modules free the memory used for the Data linked-lists, and code lists, when no longer required in the program. The user has the menu-driven option of performing off-peak autocorrelations or total cross correlations. This program can easily be modified to find the min and max of the even or odd correlation functions for each phase, if desired.

detected.

#### Comparison of Maximal, Gold and Kasami Codes

Figures A-73 and A-74 show the maximum even and odd thresholds obtained from the Mass-correlation algorithm for all three un-optimized classes of codes of length 63. In the threshold range 0.15 to 0.4, the Kasami set clearly has fewer peaks and sub-peaks than either of the other sets of even correlations. In terms of the even correlations, the Maximal set shows better performance than the Gold set within this range. The odd correlations are very similar in this range, with the Kasami set showing a slightly better performance.

Figures A-75 and A-76 show the maximum even and odd thresholds obtained from the Mass-correlation algorithm for the Linear and Gold un-optimized classes of codes of length 127. In terms of the even thresholds in the range 0.13 to 0.20, the Gold codes show fewer peaks than the Maximal code set. The odd thresholds show the odd correlation functions to be quite similar for both Maximal and Gold code set over the entire range of threshold values.

Figures A-77 and A-78 show the maximum even and odd thresholds obtained from the Mass-correlation algorithm for the Linear and Kasami un-optimized classes of codes of length 255. In the threshold range of 0.07 to 0.2 of the even threshold functions, the Kasami code set has fewer peaks than the Maximal set. In the odd threshold functions, both the sets are similar for thresholds above 0.1. Because of the equal importance of the even and odd

number of sub-peaks in the threshold range of 0.1 and 0.13. The odd correlations however, show variations below 0.32.

Length 255 Kasami. The odd and even cross-correlation functions for Kasami code 2 and 3, of length 255, are shown in Figures A-64 and A-65. The even correlation function is three valued [15, -1, -17] and somewhat irregular while the odd correlation is vary irregular with maximum peaks at  $\pm 46$ . The threshold function for both is depicted in Figure A-66. Neither threshold plot has significant values above 0.13, but the even function shows more peaks below this threshold. The Mass-correlation output of entire subset of 10 un-optimized Kasami code sequences is plotted in Figures A-67 and A-68. There is some variation in the number of sub-peaks region between 0.06 and 0.07 for the even correlations and the odd correlations show increasing variations below the threshold of 0.15.

Length 1023 Kasami. Figures A-69 and A-70 show the output of the Mass-correlate algorithm for the 8 un-optimized Kasami codes of length 1023. There is no variation in the even correlations above the threshold of 0.03 and minor fluctuations below this value. The odd correlations show slight variations below the threshold value of 0.06.

Comparison of Kasami Code Sets. Figures A-71 and A-72 show the maximum even and odd thresholds obtained from the Mass-correlate algorithm for the Kasami codes generated in Table 3. As the length of the code increases, the performance dramatically increases for both functions. The even functions clearly outperforming the odd in terms of the number of peaks and sub-peaks

Length 63 Kasami. Figures A-54 and A-55 show the even and odd cross-correlation functions codes 2 and 3. The even function is three valued [7, -1, -9], for integer values of tau. The odd cross-correlation function is irregular with maximum peaks of  $\pm 17$ . Figure A-56 shows the threshold plots of this cross-correlation for both functions, and obviously the odd function has more peaks in a range above 0.11. It is worth noting that the apparent discontinuities of the even threshold occur as the threshold passes the three values mentioned earlier.

The off-peak odd auto-correlation of code 2 and the odd cross-correlation of codes 2 and 3 were optimized in terms of phase and maximum peaks and the resulting correlation functions are depicted in Figures A-57 and A-58. The worst and best phase of the odd auto and cross-correlation threshold functions are plotted in Figures A-59 and A-60. In both cases, the odd correlation functions show a significant difference in the number of peaks and sub-peaks in the threshold range of 0.05 to 0.25. When the even and odd cross-correlation are analyzed in Figure A-60, it is apparent that the odd function has more peaks and sub-peaks in the range between 0.15 and 0.25. When the minimum odd and even cross-correlation functions are compared in Figure A-61 the even function appears to provide more interference in the threshold range of 0.15 to 0.21.

Figures A-62 and A-63 show the output of the Mass-correlate algorithm for all 8 of the un-optimized codes of length 63. The even correlations show little or no variation above a threshold of 0.13 for the entire set, and substantial variation in the

threshold below a value of 0.15 which indicates that the even correlation functions have greater variation in the number of sub-peaks in this range.

Length 1023 Gold. Figures A-50 and A-51 show the output of the Mass-correlate algorithm for the 8 un-optimized sequences in the Gold subset of length 1023. There is little deviation across the subset for both the even and odd threshold functions.

Comparison of Gold Code Sets. Figures A-52 and A-53 show the maximum even and odd threshold functions obtained from the Mass-correlate algorithm for all of the code sets defined in Table 2. As the length of the code increases, the performance increases significantly for both the even and odd threshold functions.

#### Kasami Codes

The tap assignments used for the generation of Kasami code sets are given by the polynomials in Table 3. Each polynomial yields a set of  $2^{N/2}$  code sequences, of which the number used for analysis is indicated in the table. Three different lengths of Kasami code sets are analyzed; 63, 255, and 1023.

Table 3

Code Generation Polynomials for Kasami Code Sets

<u>Length</u>	<u>Polynomial</u>	<u>Number Generated</u>
63	[1,6]	8
255	[2,3,4,8]	10
1023	[3,10]	8

and odd functions within the set significantly vary in terms of sub-peaks below a threshold of 0.28.

Length 127 Gold. Figures A-40 and A-41 show the even and odd cross-correlation functions for codes 3 and 4, length 127 Gold code set. The even correlation function is three valued [15, -1, -17] at integer values of tau. The odd correlation function is irregular with max peaks at  $\pm 28$ . The threshold algorithm plot in Figure A-42, of these two functions shows the even function having fewer peaks above a threshold of 0.14 and more sub-peaks below this threshold.

The off-peak odd auto-correlation of code 3 and the odd cross-correlation of codes 3 and 4 were optimized in terms of phase and maximum peaks and yielded the correlation functions in Figures A-43 and A-44. The worst and best phase of the odd auto-correlation function was thresholded to provide the plot in Figure A-45. It is apparent that there is a significant difference in the number of peaks and sub-peaks in threshold values above 0.08. Figure A-46 shows similar results for the worst and best phase of the odd cross-correlation function above a threshold of 0.75. When the minimum odd and even cross-correlation functions are compared in Figure A-47 the even function appears to provide more interference below a threshold of 0.14.

Figures A-48 and A-49 show the output of the Mass-correlation algorithm for the 10 un-optimized Gold codes in the subset of length 127. The even functions show no variation above a 0.14 threshold while the odd functions begin varying at a threshold value of 0.22. The even threshold varies much more than the odd

Table 2  
Code Generation Polynomials for Gold Code Sets

<u>Length</u>	<u>Polynomial</u>	<u>Number Generated</u>
63	[1,6]	10
127	[3,7]	10
1023	[3,10]	8

plot of this cross correlation for both even and odd functions. Both functions perform similarly between threshold values of 0.30 and 0.25. The odd function performs slightly better than the even between threshold values of 0.15 and 0.25. This can be attributed to fewer sub-peaks in the odd function.

The off-peak odd auto-correlation of code 3 and the odd cross-correlation of codes 3 and 4 were optimized by varying the phase and obtaining the minimum peak functions depicted in Figures A-33 and A-34. The worst and best phase of the odd auto and cross-correlation functions is shown in Figures A-35 and A-36. The difference between the worst and best phase is significant in both cases, and indicates that optimization of phase relationships will realize significant gains. Figure A-37 depicts the optimized odd, and even cross-correlation functions in terms of the threshold algorithm. This figure indicates that the even function has more peaks in the region above 0.18.

Figures A-38 and A-39 depict the output of the Mass-correlation algorithm for ten of the un-optimized Gold codes in the set of codes of length 63. These figures indicate that both the even

codes is plotted in Figures A-24 and A-25. There is little variation above a threshold value of 0.15 for both even and odd threshold plots. Below 0.15, the even threshold shows much more variation than the odd threshold.

Length 1023 Maximal. Figures A-26 and A-27 show the output of the Mass-correlate algorithm for all 8 un-optimized Maximal codes of length 1023. There is little variation above a threshold of 0.075 for both the even and odd threshold plots. Below 0.075, the even threshold shows much more variation again.

Comparison of Maximal Code Sets. Figures A-28 and A-29 show the maximum even and odd thresholds obtained from the Mass-correlate algorithm for all the Maximal code sets defined herein. As the length of the code increases, the performance dramatically increases for both the even and odd threshold functions. The gains made from shorter to longer code sets appear to gradually decrease in magnitude as the order increases.

#### Gold Codes

The tap assignments used for the generation of Gold code sets are given by the polynomials in Table 2. Each pair of polynomials yields a set of  $2^N + 2$  code sequences, of which the number used for analysis is indicated in the table. Three different lengths of Gold code sets are analyzed; 63, 127, and 1023.

Length 63 Gold. Figures A-30 and A-31 depict the even and odd cross-correlation functions for codes 3 and 4. The even correlation function is three valued [15, -1, -17] at integer values of tau. The odd cross-correlation function is irregular, and has peaks of 19 and -18. Figure A-32 shows the threshold

threshold functions of each is plotted in Figure A-13. For this case the odd cross-correlation has more impact.

The optimization of the odd auto and cross-correlation of code 1 and 2 produce the correlation functions shown in Figure A-14 and 15. Figure A-16 shows the worst and best phase of the auto-correlation function of code 1. Again, it shows a significant difference between the worst and best odd auto-correlation. Figure A-17 show similar results for the odd cross-correlation of codes 1 and 2. The even threshold function in this figure as well as Figure A-18 indicates that it has slightly less impact than either the worst or best odd cross-correlation function for thresholds above 0.14. As the threshold is lowered below 0.14, the even cross-correlation has greater impact.

Figures A-19 and A-20 show the output of the Mass-correlate algorithm for all 10 un-optimized codes of length 127. There is little variation with threshold above 0.15. The variation increases slightly below that threshold.

Length 255 Maximal. The odd and even cross correlation functions for Maximal code 1 and 2 of Length 255, are shown in Figures A-21 and A-22. The even function is 5 valued and appears to be more irregular than previous even cross-correlation functions. The odd function is irregular and has more peaks around 40 than the even function. Figure A-23 depicts the threshold functions of both the even and odd cross-correlation. Neither threshold plot has significant values above 0.13, but the even function shows more peaks below this threshold. The Mass-correlation algorithm for the entire code set of 8 un-optimized

important. This portion of the function characterizes the peak as well as all of the significant sub-peaks. For this cross-correlation, the even and odd functions are similar with the even function providing slightly more interference on the  $i$ th correlators received signal.

The off-peak odd auto-correlation of code 1 and the odd cross-correlation of codes 1 and 2 were optimized in terms of phase and maximum peaks and are depicted in Figures A-4 and A-5. The worst and best phase of the odd auto-correlation threshold functions is depicted in Figure A-6. All of the other phase shifts of code 1 have threshold plots between these two. The difference between the worst and best phase is significant in terms of the odd auto-correlation. Figure A-7 shows the minimum and maximum odd cross-correlation. The minimum threshold function reaches zero quicker than the maximum function thereby providing slight improvement. When the minimum odd cross-correlation is compared to the even cross-correlation function in Figure A-8, the even function appears to provide more interference.

Figures A-9 and A-10 show the output of the Mass-correlate algorithm for all six of the un-optimized Maximal codes of length 63. These figures show little variation in the code set in terms of peaks and sub-peaks over all the cross-correlations, but show significant variations for lower thresholds (below 0.3).

Length 127 Maximal. Figures A-11 and A-12 show the odd and even cross-correlation functions for code 1 and 2, length 127 Maximal code set. Again the even function is three valued at integer value of  $\tau$  and the odd function is irregular. The

Table 1

## Code Generation Polynomials for Maximal Length Code Sets

<u>Length</u>	<u>Polynomial</u>	<u>Name</u>
63	[1,6]	1
	[1,2,5,6]	2
	[2,3,5,6]	3
	[5,6]	4
	[1,4,5,6]	5
	[1,2,3,6]	6
127	[3,7]	1
	[1,2,3,7]	2
	[1,2,4,5,6,7]	3
	[2,3,4,7]	4
	[1,2,3,4,5,7]	5
	[2,4,6,7]	6
	[1,7]	7
	[1,3,6,7]	8
	[2,5,6,7]	9
	[4,7]	10
255	[2,3,4,8]	1
	[3,5,6,8]	2
	[1,2,5,6,7,8]	3
	[1,3,5,8]	4
	[2,5,6,8]	5
	[1,5,6,8]	6
	[1,2,3,4,6,8]	7
	[1,6,7,8]	8
1023	[3,10]	1
	[2,3,8,10]	2
	[3,4,5,6,7,8,9,10]	3
	[1,2,3,5,6,10]	4
	[2,3,6,8,9,10]	5
	[1,3,4,5,6,7,8,10]	6
	[7,10]	7
	[2,7,8,10]	8

threshold function is due to the threshold passing through the value  $\pm 1$ , and is common for most even cross-correlation functions of linear codes. The odd cross-correlation threshold function has a more general slope and this can be attributed to the irregularity of the odd function. In terms of the system model in Chapter II, the lower part of the threshold functions are most

#### IV. Code Performance Analysis

In this chapter, the performance of three classes of codes, as measured by the algorithms presented in Chapter III, are analyzed. The three classes analyzed are Maximal Length Codes, Gold Codes and Kasami Codes. The performance of individual codes within a code set are analyzed first, followed by a measure of the code set itself. This analysis is repeated for varying length code sets for each code class. Then the different length code sets are compared within each class. Finally, the three classes are compared for varying lengths of code sets.

Appendix A contains all the figures referenced in this chapter. This appendix contains actual even and odd correlation functions for code sets, as well as plots from the thresholding algorithm (threshold plots).

##### Maximal Length Codes

The tap assignments used for the generation of Maximal length code sets are given by the polynomials in Table 1. Four different lengths of Maximal length codes are analyzed; 63, 127, 255 and 1023.

Length 63 Maximal. Figures A-1 and A-2 depict the even and odd cross-correlation functions for codes 1 and 2. It should be noted that the even correlation function is three valued [15, -1, -17] at integer values of tau. The odd cross-correlation function is more erratic however, and has maximum peaks at 17 and -19. Figure A-3 shows the threshold plot of this cross-correlation for both functions. The apparent discontinuity of the even

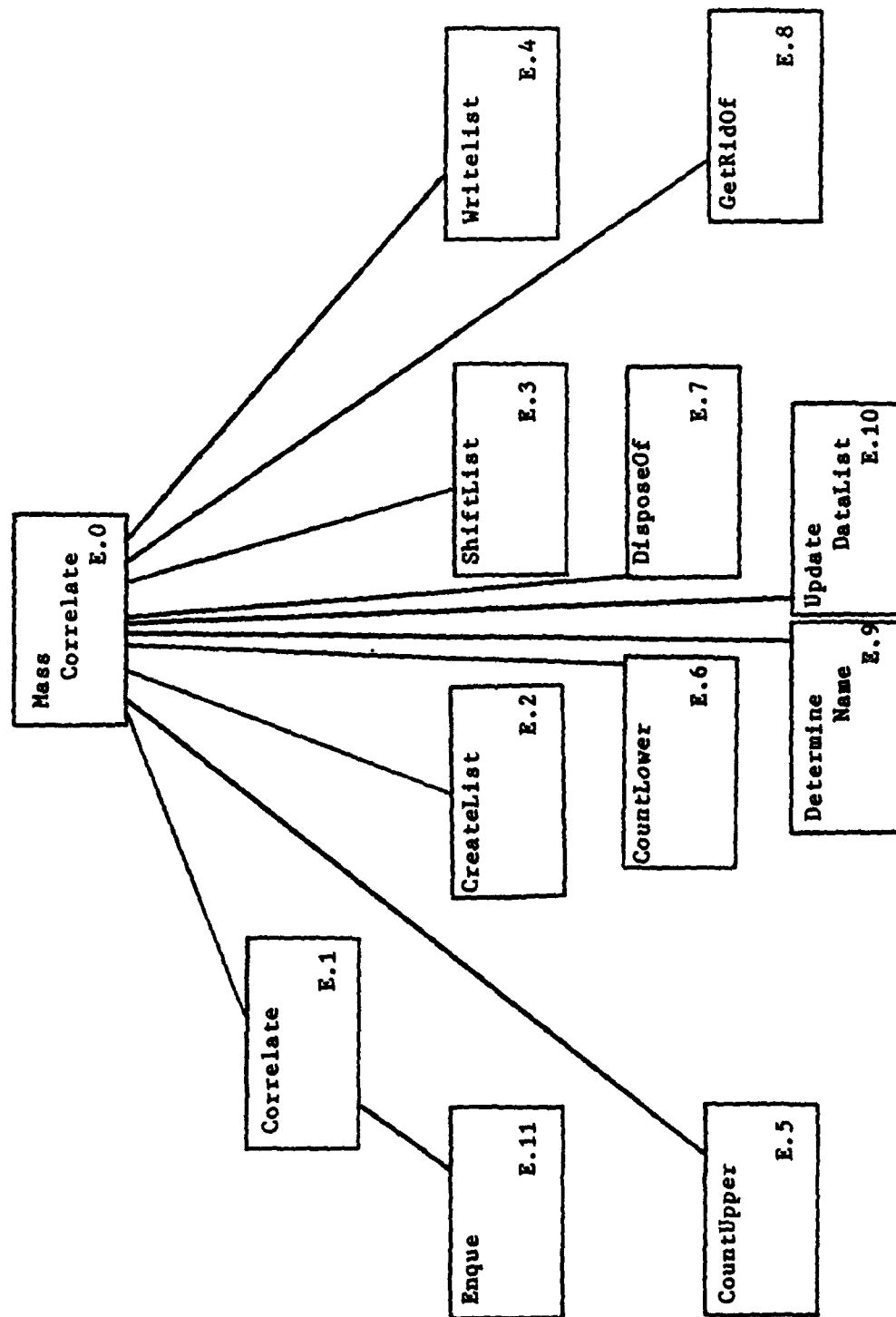


Figure III-4. Structure Chart for Mass Correlate Algorithm

### Program MassCorrelate

The structure of the MassCorrelate program is depicted in Figure III-4. It is similar to the PartialCorrelate structure, with the exception of the modules DetermineName and Update DataList. The program is much more limited in user options than the PartialCorrelate program. This program is designed to calculate both the Even and Odd correlation functions, and performs both the positive and negative thresholding process on each to determine the amount of the correlation function exceeding these thresholds. However, this program will perform this process on an entire code set rather than just two code sequences. All of modules in this program that have the same names as those in the PartialCorrelate program perform the same function. The Module DetermineName is called to convert the entered filename extensions to integer values, in order to set up looping controls within the program, which perform correlations of pairs of code sequences. Instead of writing the threshold and amount exceeding threshold directly to an output file, this program will enter this information into a datalist in memory using module Update-DataList. Each record in the list contains fields for the threshold value, the minimum, maximum and average value for both even and odd functions. As each pair of code sequences is processed, the fields of the data list are updated accordingly. When all code sequences in the user defined code set have been cross-correlated and thresholded, the program will write the data list to a user designated file.

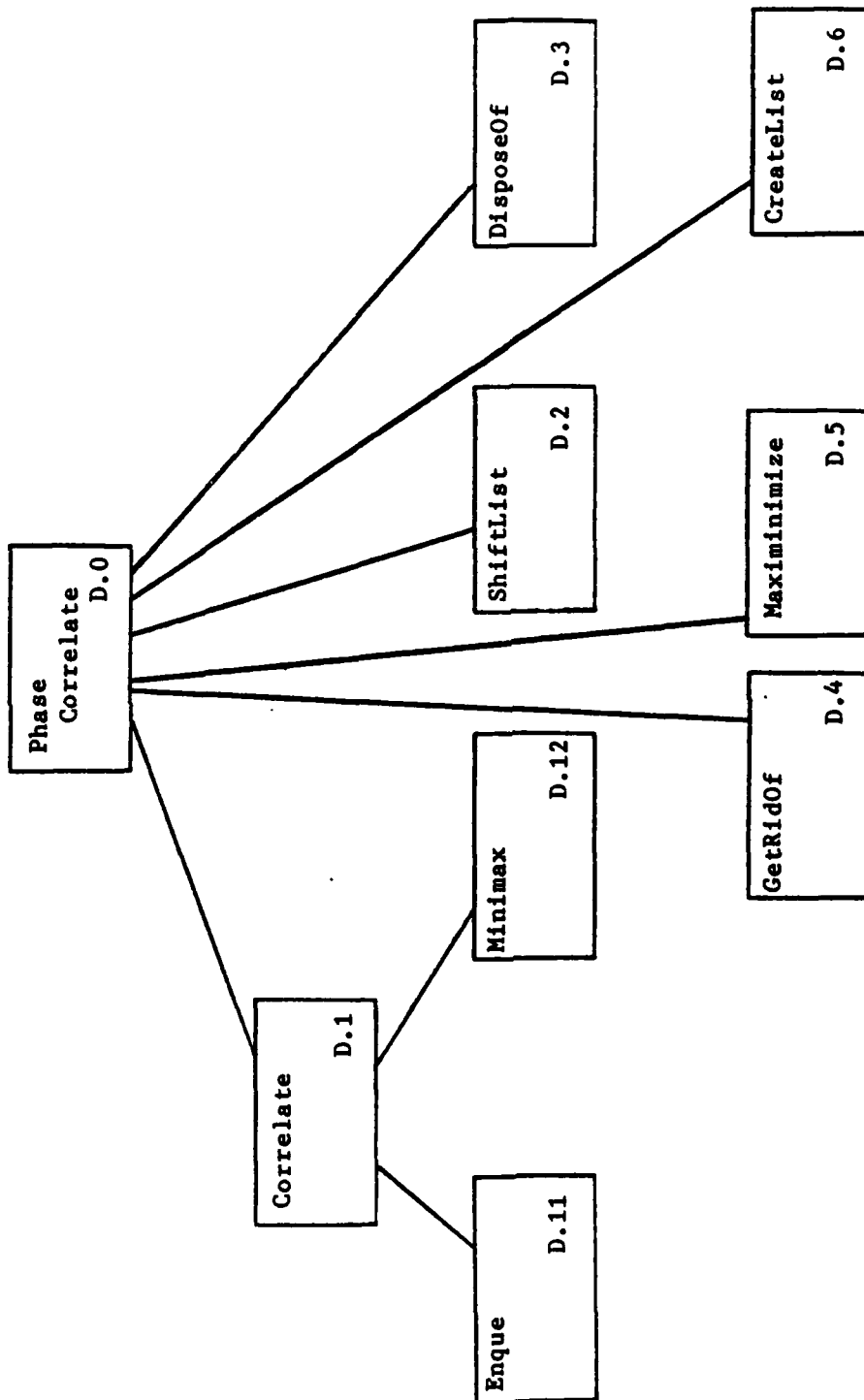


Figure III-3. Structure Chart for Phase Correlate Algorithm

cross-correlation functions, the Kasami set appears to show better performance in the range above 0.1.

Figures A-79 and A-80 show the maximum even and odd thresholds obtained from the Mass-correlation algorithm for all three un-optimized classes of codes of length 1023. In the threshold range 0.03 to 0.10 the Kasami set is clearly the best performer in terms of the number of peaks in the even threshold functions. In the same range the Maximal set slightly out-performs the Gold Set. In the odd threshold functions of the same range, the three sets are very similar. The Kasami set slightly out-performs the other two followed by the Maximal and Gold sets.

## V. Software Performance

This chapter discusses the performance of each of the programs in terms of amount of computer runtime that is required for typical inputs. All data presented in this chapter was compiled using a Kaypro-2 micro-computer using a 2 MHz clock, and having 64K memory capacity.

### Program GenerateCodes

The code generation software has performance specifications as listed in Table 4. Because the Gold and Kasami code generators use a linear generator to set up for their respective code generation, the times listed in their columns reflect a set up time followed by the amount of time to generate each sequence in the code set.

Table 4

Code Generator Software Performance Times (Seconds)

<u>Register Length</u>	<u>Maximal</u>	<u>Gold</u>	<u>Kasami</u>
15	151	**	---
14	75	**	**
13	38	**	---
12	19	19/3	19/3
11	9	9/2.5	---
10	4	4/2	4/2
9	2	2/1.5	---
8	1	1/1	1/1

--- Code generation not possible for this register length  
\*\* Insufficient amount of memory for generation set up

### Correlation Times

The correlation process is used in the Partial-Correlate, Phase-Correlate and the Mass-correlate programs, and has the run-

times listed in Table 5. Based on this data, the following equation can be used to approximate the required runtime as a function of the length of the code sequences to be correlated:

$$\text{Runtime} = (1.326 \times 10^{-4})L^2 + (1.2 \times 10^{-2})L \quad (\text{secs})$$

Typical runtimes for the correlation process are listed in Table 5.

Table 5  
Correlation Software Performance Times (Secs)

<u>Code Length</u>	<u>Runtimes</u>
63	2
127	3.5
255	14
511	48
1023	148
2047	570

#### Phase-Correlate Program Performance

The Phase-Correlate program must cycle through L correlation processes to find the best phase performance of each pair of codes. For this reason, the runtime required for each pair of sequences balloons rapidly. The runtime function can be approximated as follows:

$$\text{Runtime} = (1.326 \times 10^{-4})L^3 + (1.2 \times 10^{-2})L^2 + \text{OH} \quad (\text{secs})$$

OH is the overhead time associated with storing the maximum peaks for each phase, and is small relative to the other factors in the equation. It is obvious that attempting to optimize an entire code set for the best phase cross-correlation relationships would be futile for any reasonable length. For this reason, heuristic approaches must be used to "optimize" the performance of each

code set.

### Thresholding Process

The thresholding algorithm runtime varies with the the length of the code, correlation functions themselves, and the number of threshold values required. All runs used in this work, varied the threshold values from 0 to the highest peak in the correlation function being thresholded. Generally the threshold process (using approximately 150 thresholds) required less time than the correlation process for all work in this thesis. In order to shorten required runtimes, the threshold region can be reduced to a smaller region of interest. At worst, the runtime of this process is a linear function of length of the code.

### Mass-correlate performance

The Mass-correlate program performs  $(N^2 - N)/2$  correlations, where  $N$  is the number of code sequences within the user defined code set. The runtime for this program is a function of the cross-correlations required, the length of the code sequences, and the number of thresholds of interest, and is in the order of  $L^2$ . This software can be improved by limiting the thresholds used to a desired region of interest.

## VI. Conclusions and Recommendations

### Conclusions

From the analysis of Chapter IV it can be concluded that the thresholding algorithm can be used to effectively evaluate the performance of binary code sequences in the phase-coded code-division multiple-access (CDMA) model. The MassCorrelate program will perform the thresholding algorithm on an entire code set of binary sequences and can be used to evaluate the performance of the set. The computing time required to run the MassCorrelate program can be reduced significantly by reducing the region of threshold interest as dictated by particular receiver specifications.

### Recommendations

The following recommendations are proposed for further study in this area:

1. Using algorithms for finding suboptimal sequence phases, as outlined by Gahutu (25:22-26), and evaluating the performance of these and any other optimization techniques, using the thresholding algorithm.
2. Optimize the software in terms of memory requirements and/or runtime to improve performance of the algorithms.
3. In order to reduce the runtime of the correlation algorithm for long code sequences, it seems feasible to implement this algorithm using discrete Fourier techniques on an array processor.
4. Incorporating additional code generation capabilities

in the GenerateCodes program, to include other linear and non-linear generators, so that these code sets can be evaluated by the thresholding algorithm and compared with the Maximal length, Gold and Kasami code sets.

## Bibliography

1. Castor, Kenneth G., Lecture materials distributed in EE 6.73, Applications of Communications Technology - Spread Spectrum Communications. School of Engineering, Air Force Institute of Technology (AU), Wright-Patterson AFB OH, September 1984.
2. Dixon, R.C., Spread Spectrum Techniques. New York: IEEE Press, 1976.
3. Dixon, R.C., Spread Spectrum Systems. New York: John Wiley, 1976.
4. Gold, R., "Optimum Binary Sequences for Spread Spectrum Multiplexing," IEEE Transactions on Information Theory, IT-13: 619 - 621 (October 1967).
5. Gold, R., "Maximal Recursive Sequences with 3-Valued Recursive Cross-correlation Functions," IEEE Transactions on Information Theory, IT-14: 154 - 156 (January 1968).
6. Holmes, J. K., Coherent Spread Spectrum Systems. New York: John Wiley, 1982.
7. Lin, Shu and Daniel J. Costello, Jr., Error Control Coding: Fundamentals and Applications. Englewood Cliffs, New Jersey: Prentice-Hall, Inc., 1983.
8. Massey, J. L. and J. J. Urhan, Jr., "Sub-baud Coding," Proceedings of the 13th Annual Allerton Conference on Circuit and Systems Theory: 539 - 547 (October 1975).
9. Olsen, J. D., R. A. Scholtz, and L. R. Welch, "Bent Function Sequences," IEEE Transactions on Information Theory, IT-28: 858 - 864 (November 1982).
10. Pickholtz, Raymond L., Donald L. Schilling and Laurence B. Milstein, "Theory of Spread-Spectrum Communications--A Tutorial," IEEE Transactions on Communications, COM-30: 855-884 (May 1982).
11. Proakis, John G., Digital Communications. New York: McGraw-Hill, Inc., 1983.
12. Pursley, Michael B., "Performance Evaluation for Phase-Coded Spread-Spectrum Multiple-Access Communication - Part I: System Analysis," IEEE Transactions on Communications, COM-25: 795 - 799 (August 1977).
13. Pursley, Michael B. and Dilip V. Sarwate, "Performance Evaluation Phase-Coded Spread-Spectrum Multiple-Access Communication - Part II: Code Sequence Analysis," IEEE Transactions on Communications, COM-25: 800 - 803 (August 1977).

14. Ristenblatt, Marlin P. and James L. Daws, Jr., "Performance Criteria for Spread Spectrum Communications," IEEE Transactions on Communications, COM-25: 756-763 (August 1977).
15. Rowe, Harrison E., "Bounds on the Number of Signals with Restricted Cross Correlation," IEEE Transactions on Communications, COM-30: 174 - 182 (May 1982).
16. Sarwate, Dilip V., "Bounds on Crosscorrelation and Autocorrelation of Sequences," IEEE Transactions on Information Theory, IT-25: 720 - 724 (November 1979).
17. Sarwate, Dilip V. and Michael D. Pursley, "Crosscorrelation Properties of Pseudorandom and Related Sequences," Proceedings of IEEE, Volume 68: 593 - 619 (May 1980).
18. Sarwate, Dilip V., "Cross-Correlation Properties of Sequences with Applications to Spread-Spectrum Multiple Access Communications," AFSOR Workshop in Communication Theory and Applications, AFOSR 78-3715: 88 - 91 (August 1979).
19. Simon, Marvin K., Jim K. Omura, Robert A. Scholtz, and Barry K. Levitt. Spread Spectrum Communications, Volume I. Rockville, Maryland: Computer Science Press, Inc., 1985.
20. Spellman, Marc, "A Comparison Between Frequency Hopping and Direct Spread PN as Antijam Techniques," IEEE Communications Magazine, Volume 21: 26 - 33 (July 1983).
21. Utlaut, W. F., "Spread Spectrum - Principles and Possible Applications to Spectrum Utilization and Allocation," ITU Telecommunications Journal, Volume 45: 20 - 32 (January 1978).
22. Viterbi, Andrew J., "Spread Spectrum Communications - Myths and Realities," IEEE Communications Magazine, Volume 17: 11 - 18 (May 1979).
23. Welch, L. R., "Lower Bounds on the Maximum Cross Correlation of Signals," IEEE Transactions on Information Theory, IT-20: 397 - 399 (May 1974).
24. Yao, Kung, "Error Probability of Asynchronous Spread Spectrum Multiple Access Communication Systems," IEEE Transactions on Communications, COM-25: 803 - 809 (August 1977).
25. Gahuto, David William Haguma, On Correlation Parameters for Some Binary Sequences of Lengths 31 and 63 for Spread-Spectrum Multiple Access Communications. MS Thesis, Coordinated Science Laboratory, University of Illinois at Urbana-Champaign, Urbana, Illinois, May 1979.

## Appendix A

### Performance Plots

This appendix contains all the performance plots of the data obtained from the PartialCorrelate program and the MassCorrelate program. Its order is parallel to that of the order of presentation in Chapter IV. All plots were obtained using the S plotting package available on the VAX 11-780 SSC, and the HP 7220 plotter. Each plot that is obtained from the threshold algorithm is normalized in terms of length of the code sequence. All plots of actual correlation functions are unnormalized.

## APPENDIX A

### List of Figures

Figure		Page
A-1	Typical Even Cross Correlation Function of Maximal Length Code of Length 63.....	A-8
A-2	Typical Odd Cross Correlation Function of Maximal Length Code of Length 63.....	A-8
A-3	Typical Amount Exceeding Threshold vs Threshold of Even and Odd Correlation Functions Maximal Length Code of Length 63.....	A-9
A-4	Optimal Odd Auto-Correlation Function of Maximal Codes 1 & 2, Length 63.....	A-9
A-5	Optimal Odd Cross-Correlation Function of Maximal Codes 1 & 2, Length 63.....	A-10
A-6	Threshold Plot of Minimum and Maximum Odd Auto- Correlation and Even Auto-Correlation of Maximal Codes 1 & 2, Length 63.....	A-10
A-7	Threshold Plot of Minimum and Maximum Odd Cross- Correlation and Even Cross-Correlation of Maximal Codes 1 & 2, Length 63.....	A-11
A-8	Threshold Plot of Minimum Odd Auto-Correlation, Minimum Odd Cross-Correlation, and Even Cross-Cor- relation of Maximal Codes 1 & 2, Length 63.....	A-11
A-9	Threshold Plot of Minimum, Maximum and Average Even Correlation Functions from Mass-Correlation Algorithm for Maximal Codes 1-6, Length 63.....	A-12
A-10	Threshold Plot of Minimum, Maximum and Average Odd Correlation Functions from Mass-Correlation Algorithm for Maximal Codes 1-6, Length 63.....	A-12
A-11	Typical Even Cross Correlation Function of Maximal Length Code of Length 127.....	A-13
A-12	Typical Odd Cross Correlation Function of Maximal Length Code of Length 127.....	A-13
A-13	Typical Amount Exceeding Threshold vs Threshold of Even and Odd Correlation Functions Maximal Length Code of Length 127.....	A-14

Figure		Page
A-14	Optimal Odd Auto-Correlation Function of Maximal Codes 1 & 2, Length 127.....	A-14
A-15	Optimal Odd Cross-Correlation Function of Maximal Codes 1 & 2, Length 127.....	A-15
A-16	Threshold Plot of Minimum and Maximum Odd Auto-Correlation and Even Auto-Correlation of Maximal Codes 1 & 2, Length 127.....	A-15
A-17	Threshold Plot of Minimum and Maximum Odd Cross-Correlation and Even Cross-Correlation of Maximal Codes 1 & 2, Length 127.....	A-16
A-18	Threshold Plot of Minimum Odd Auto-Correlation, Minimum Odd Cross-Correlation, and Even Cross-Correlation of Maximal Codes 1 & 2, Length 127.....	A-16
A-19	Threshold Plot of Minimum, Maximum and Average Even Correlation Functions from Mass-Correlation Algorithm for Maximal Codes 1-10, Length 127.....	A-17
A-20	Threshold Plot of Minimum, Maximum and Average Odd Correlation Functions from Mass-Correlation Algorithm for Maximal Codes 1-10, Length 127.....	A-17
A-21	Typical Even Cross Correlation Function of Maximal Length Code of Length 255.....	A-18
A-22	Typical Odd Cross Correlation Function of Maximal Length Code of Length 255.....	A-18
A-23	Typical Amount Exceeding Threshold vs Threshold of Even and Odd Correlation Functions Maximal Length Code of Length 255.....	A-19
A-24	Threshold Plot of Minimum, Maximum and Average Even Correlation Functions from Mass-Correlation Algorithm for Maximal Codes 1-8, Length 255.....	A-19
A-25	Threshold Plot of Minimum, Maximum and Average Odd Correlation Functions from Mass-Correlation Algorithm for Maximal Codes 1-8, Length 255.....	A-20
A-26	Threshold Plot of Minimum, Maximum and Average Even Correlation Functions from Mass-Correlation Algorithm for Maximal Codes 1-8, Length 1023.....	A-20
A-27	Threshold Plot of Minimum, Maximum and Average Odd Correlation Functions from Mass-Correlation Algorithm for Maximal Codes 1-8, Length 1023.....	A-21

Figure		Page
A-28	Threshold Plot of Maximum Even Correlation Functions from Mass-Correlation Algorithm for Maximal Codes of Length 63, 127, 255, 1023.....	A-21
A-29	Threshold Plot of Maximum Odd Correlation Functions from Mass-Correlation Algorithm for Maximal Codes of Length 63, 127, 255, 1023.....	A-22
A-30	Typical Even Cross Correlation Function of Gold Code of Length 63.....	A-22
A-31	Typical Odd Cross Correlation Function of Gold Code of Length 63.....	A-23
A-32	Typical Amount Exceeding Threshold vs Threshold of Even and Odd Correlation Functions Gold Code of Length 63.....	A-23
A-33	Optimal Odd Auto-Correlation Function of Gold Codes 3 & 4, Length 63.....	A-24
A-34	Optimal Odd Cross-Correlation Function of Gold Codes 3 & 4, Length 63.....	A-24
A-35	Threshold Plot of Minimum and Maximum Odd Auto-Correlation and Even Auto-Correlation of Gold Codes 3 & 4, Length 63.....	A-25
A-36	Threshold Plot of Minimum and Maximum Odd Cross-Correlation and Even Cross-Correlation of Gold Codes 3 & 4, Length 63.....	A-25
A-37	Threshold Plot of Minimum Odd Auto-Correlation, Minimum Odd Cross-Correlation, and Even Cross-Correlation of Gold Codes 3 & 4, Length 63.....	A-26
A-38	Threshold Plot of Minimum, Maximum and Average Even Correlation Functions from Mass-Correlation Algorithm for Gold Codes 1-10, Length 63.....	A-26
A-39	Threshold Plot of Minimum, Maximum and Average Odd Correlation Functions from Mass-Correlation Algorithm for Gold Codes 1-10, Length 63.....	A-27
A-40	Typical Even Cross Correlation Function of Gold Code of Length 127.....	A-27
A-41	Typical Odd Cross Correlation Function of Gold Code of Length 127.....	A-28
A-42	Typical Amount Exceeding Threshold vs Threshold of Even and Odd Correlation Functions Gold Code of Length 127.....	A-28

Figure		Page
A-43	Optimal Odd Auto-Correlation Function of Gold Codes 3 & 4, Length 127.....	A-29
A-44	Optimal Odd Cross-Correlation Function of Gold Codes 3 & 4, Length 127.....	A-29
A-45	Threshold Plot of Minimum and Maximum Odd Auto-Correlation and Even Auto-Correlation of Gold Codes 3 & 4, Length 127.....	A-30
A-46	Threshold Plot of Minimum and Maximum Odd Cross-Correlation and Even Cross-Correlation of Gold Codes 3 & 4, Length 127.....	A-30
A-47	Threshold Plot of Minimum Odd Auto-Correlation, Minimum Odd Cross-Correlation, and Even Cross-Correlation of Gold Codes 3 & 4, Length 127.....	A-31
A-48	Threshold Plot of Minimum, Maximum and Average Even Correlation Functions from Mass-Correlation Algorithm for Gold Codes 1-10, Length 127.....	A-31
A-49	Threshold Plot of Minimum, Maximum and Average Odd Correlation Functions from Mass-Correlation Algorithm for Gold Codes 1-10, Length 127.....	A-32
A-50	Threshold Plot of Minimum, Maximum and Average Even Correlation Functions from Mass-Correlation Algorithm for Gold Codes 1-8, Length 1023.....	A-32
A-51	Threshold Plot of Minimum, Maximum and Average Odd Correlation Functions from Mass-Correlation Algorithm for Gold Codes 1-8, Length 1023.....	A-33
A-52	Threshold Plot of Maximum Even Correlation Functions from Mass-Correlation Algorithm for Gold Codes of Length 63, 127, 1023.....	A-33
A-53	Threshold Plot of Maximum Odd Correlation Functions from Mass-Correlation Algorithm for Gold Codes of Length 63, 127, 1023.....	A-34
A-54	Typical Even Cross Correlation Function of Kasami Code of Length 63.....	A-34
A-55	Typical Odd Cross Correlation Function of Kasami Code of Length 63.....	A-35
A-56	Typical Amount Exceeding Threshold vs Threshold of Even and Odd Correlation Functions Kasami Code of Length 63.....	A-35

Figure		Page
A-57	Optimal Odd Auto-Correlation Function of Kasami Codes 2 & 3, Length 63.....	A-36
A-58	Optimal Odd Cross-Correlation Function of Kasami Codes 2 & 3, Length 63.....	A-36
A-59	Threshold Plot of Minimum and Maximum Odd Auto-Correlation and Even Auto-Correlation of Kasami Codes 2 & 3, Length 63.....	A-37
A-60	Threshold Plot of Minimum and Maximum Odd Cross-Correlation and Even Cross-Correlation of Kasami Codes 2 & 3, Length 63.....	A-37
A-61	Threshold Plot of Minimum Odd Auto-Correlation, Minimum Odd Cross-Correlation, and Even Cross-Correlation of Kasami Codes 2 & 3, Length 63.....	A-38
A-62	Threshold Plot of Minimum, Maximum and Average Even Correlation Functions from Mass-Correlation Algorithm for Kasami Codes 1-8, Length 63.....	A-38
A-63	Threshold Plot of Minimum, Maximum and Average Odd Correlation Functions from Mass-Correlation Algorithm for Kasami Codes 1-8, Length 63.....	A-39
A-64	Typical Even Cross Correlation Function of Kasami Code of Length 255.....	A-39
A-65	Typical Odd Cross Correlation Function of Kasami Code of Length 255.....	A-40
A-66	Typical Amount Exceeding Threshold vs Threshold of Even and Odd Correlation Functions Kasami Code of Length 255.....	A-40
A-67	Threshold Plot of Minimum, Maximum and Average Even Correlation Functions from Mass-Correlation Algorithm for Kasami Codes 1-10, Length 255.....	A-41
A-68	Threshold Plot of Minimum, Maximum and Average Odd Correlation Functions from Mass-Correlation Algorithm for Kasami Codes 1-10, Length 255.....	A-41
A-69	Threshold Plot of Minimum, Maximum and Average Even Correlation Functions from Mass-Correlation Algorithm for Kasami Codes 1-8, Length 1023.....	A-42
A-70	Threshold Plot of Minimum, Maximum and Average Odd Correlation Functions from Mass-Correlation Algorithm for Kasami Codes 1-8, Length 1023.....	A-42

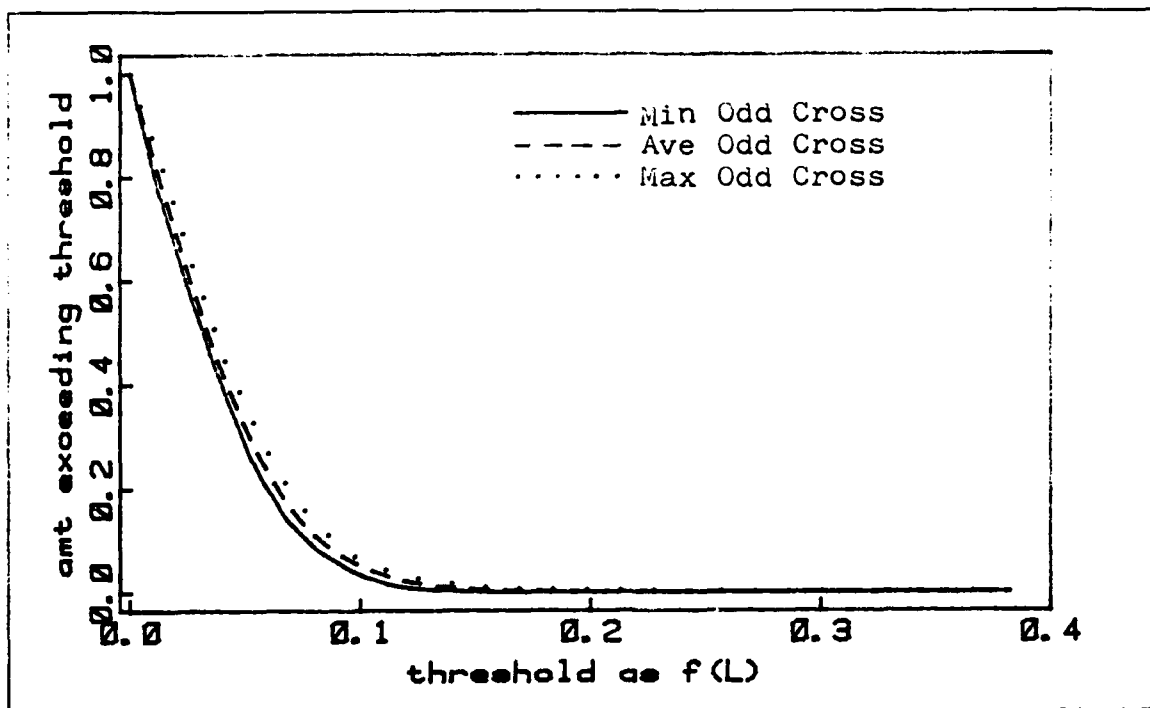


Fig. A-25. Threshold Plot of Minimum, Maximum and Average Odd Correlation Functions from Mass-Correlation Algorithm for Maximal Codes 1-8, Length 255

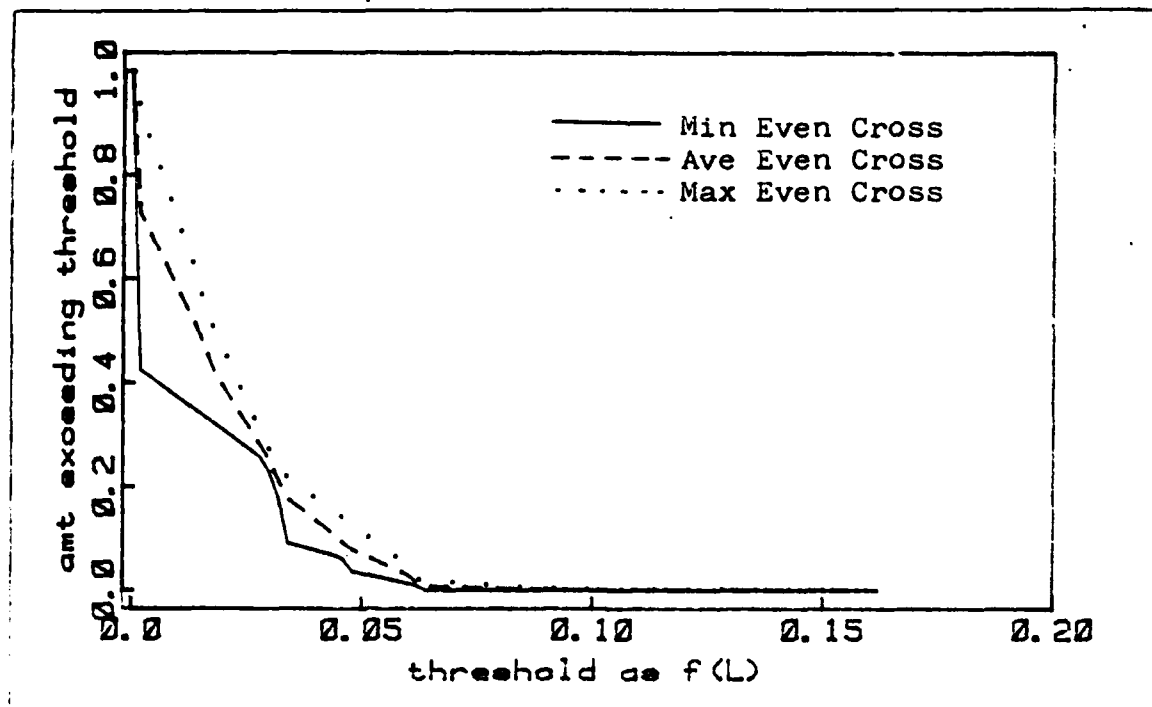


Fig. A-26. Threshold Plot of Minimum, Maximum and Average Even Correlation Functions from Mass-Correlation Algorithm for Maximal Codes 1-8, Length 1023

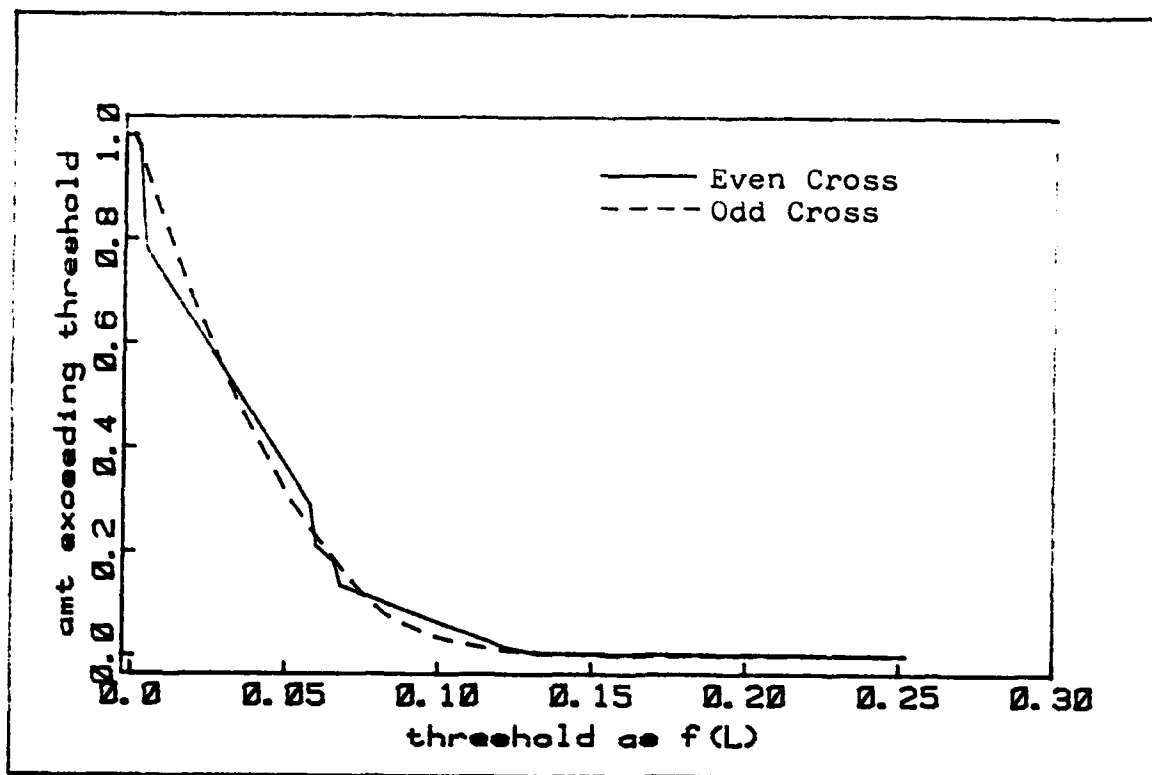


Fig. A-23. Typical Amount Exceeding Threshold vs Threshold of Even and Odd Correlation Functions Maximal Length Code of Length 255

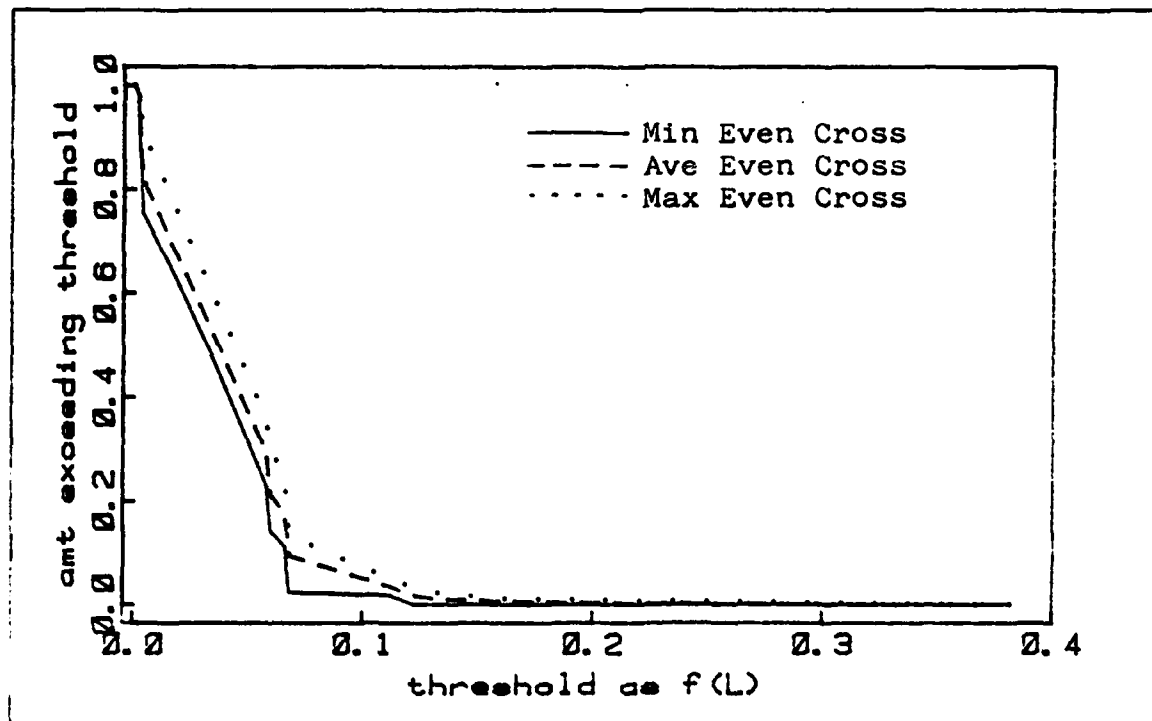


Fig. A-24. Threshold Plot of Minimum, Maximum and Average Even Correlation Functions from Mass-Correlation Algorithm for Maximal Codes 1-8, Length 255

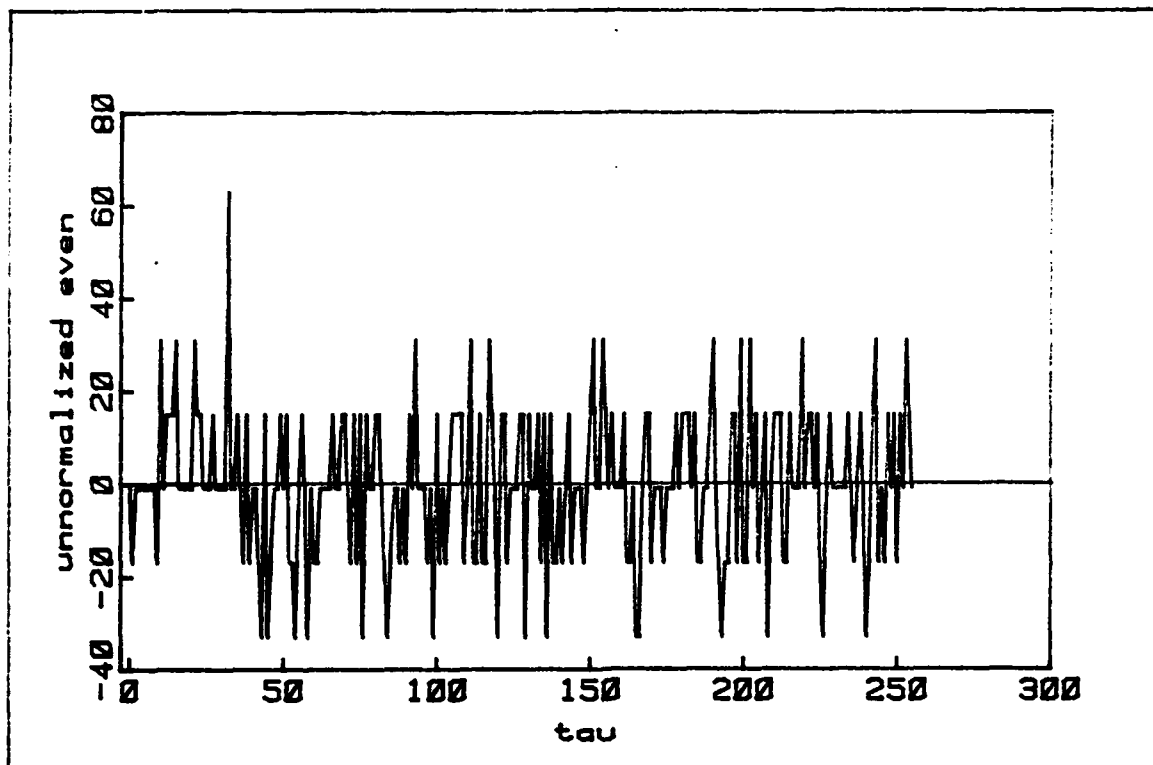


Fig. A-21. Typical Even Cross Correlation Function of Maximal Length Code of Length 255

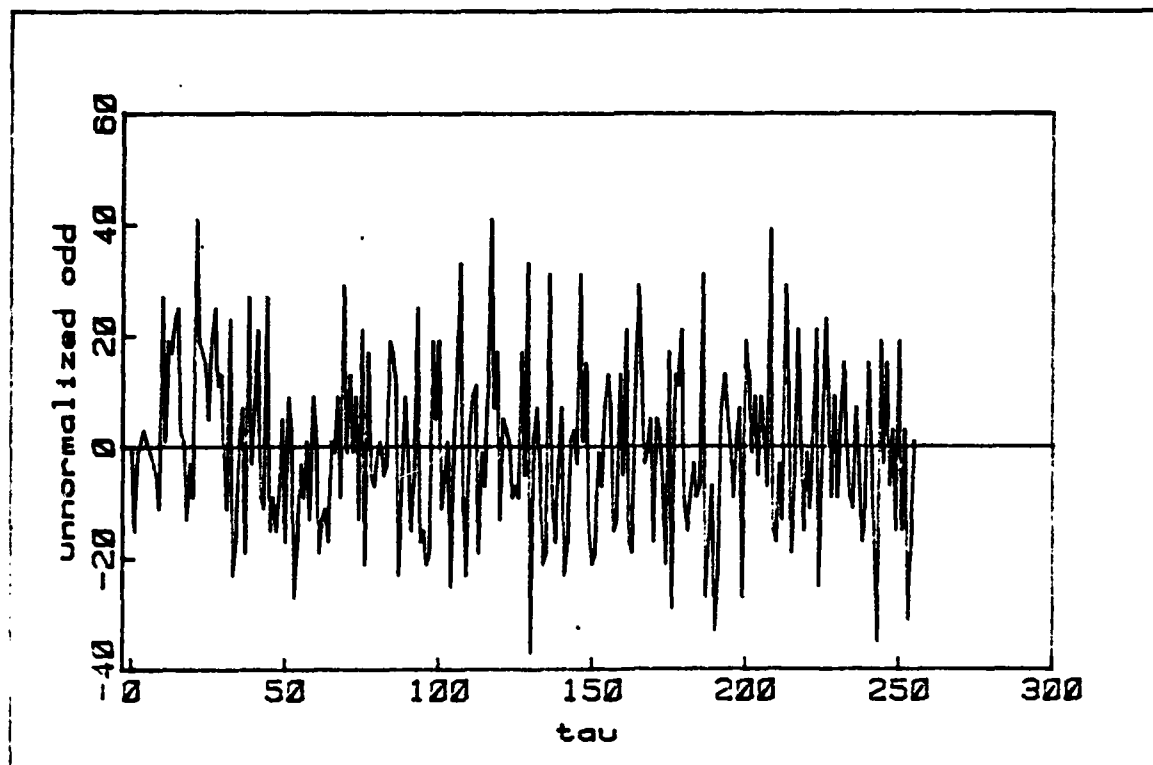


Fig. A-22. Typical Odd Cross Correlation Function of Maximal Length Code of Length 255

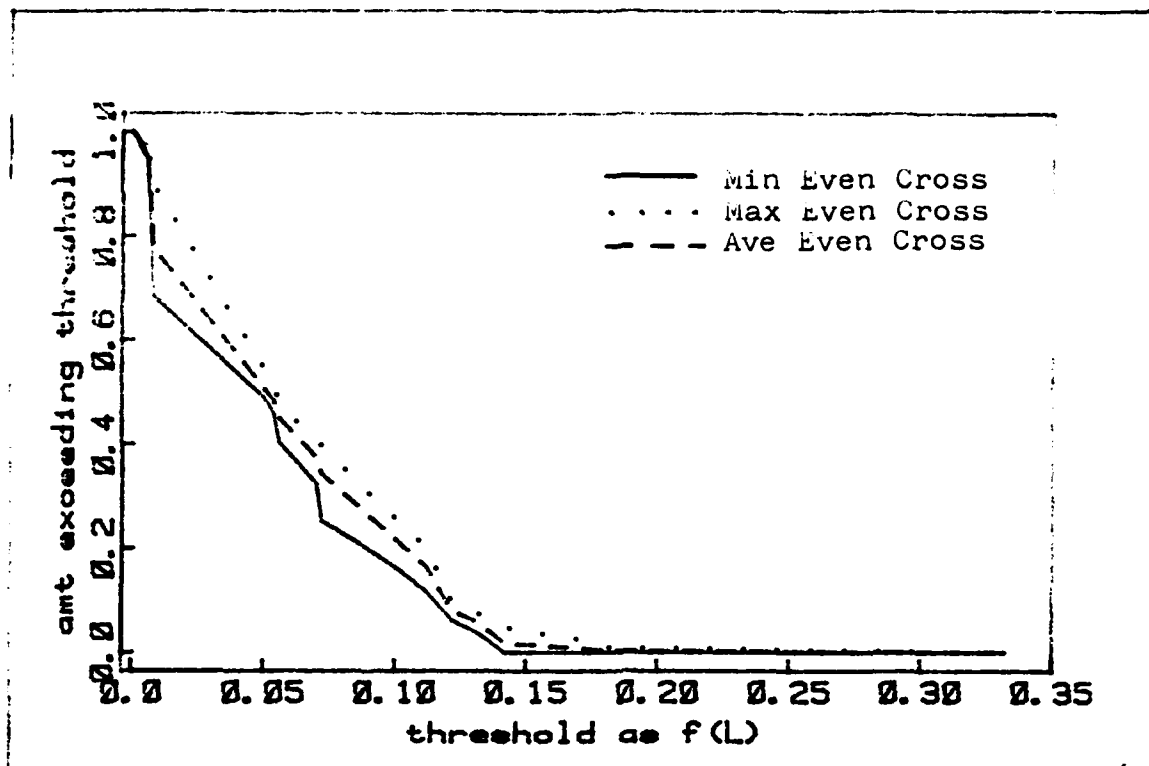


Fig. A-19. Threshold Plot of Minimum, Maximum and Average Even Correlation Functions from Mass-Correlation Algorithm for Maximal Codes 1-10, Length 127

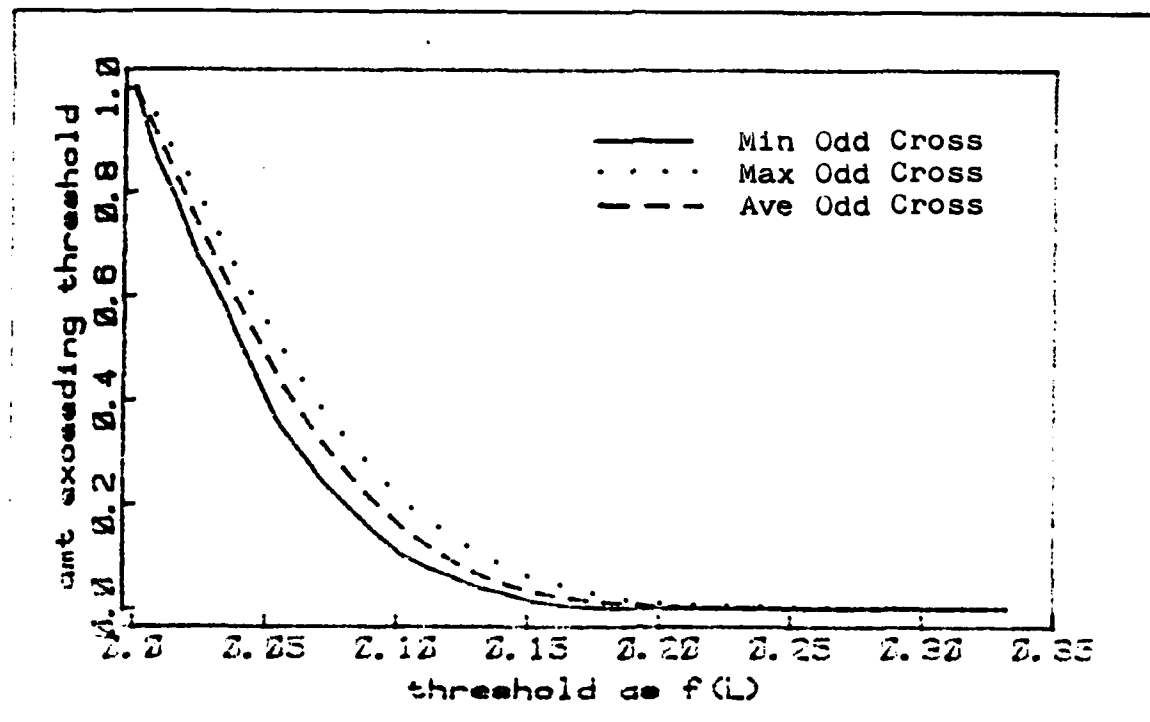


Fig. A-20. Threshold Plot of Minimum, Maximum and Average Odd Correlation Functions from Mass-Correlation Algorithm for Maximal Codes 1-10, Length 127

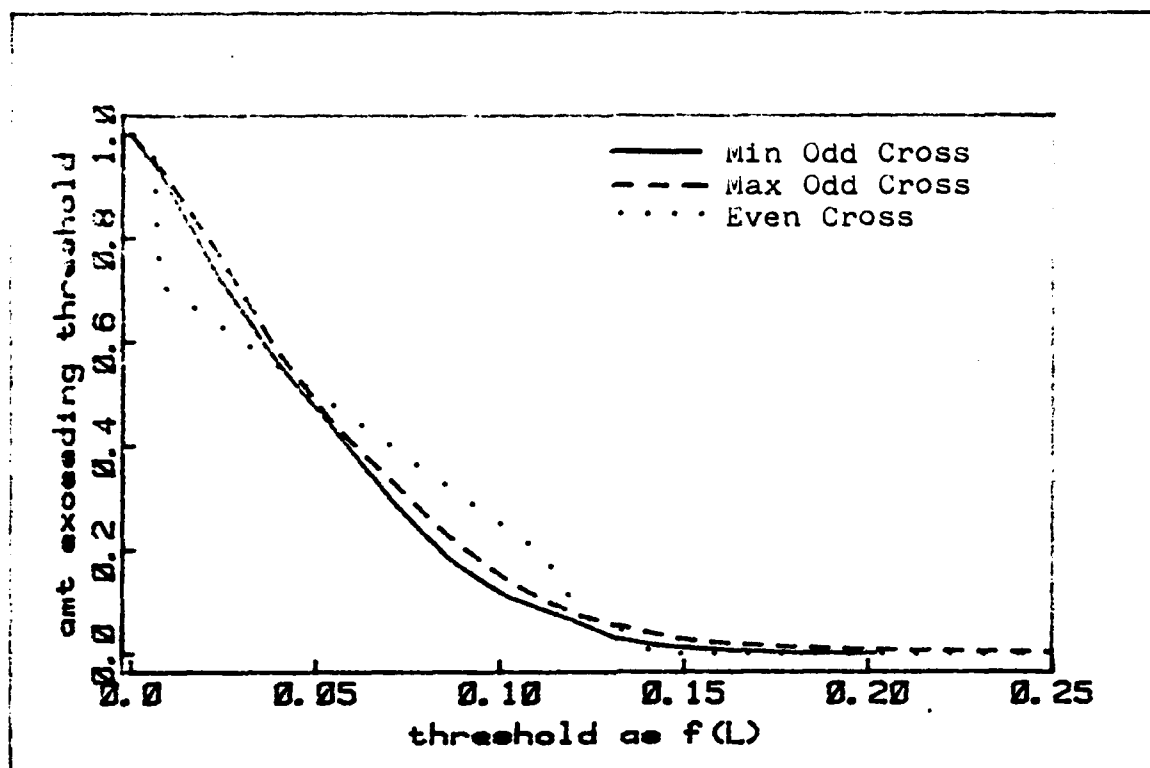


Fig. A-17. Threshold Plot of Minimum and Maximum Odd Cross-Correlation and Even Cross-Correlation of Maximal Codes 1 & 2, Length 127

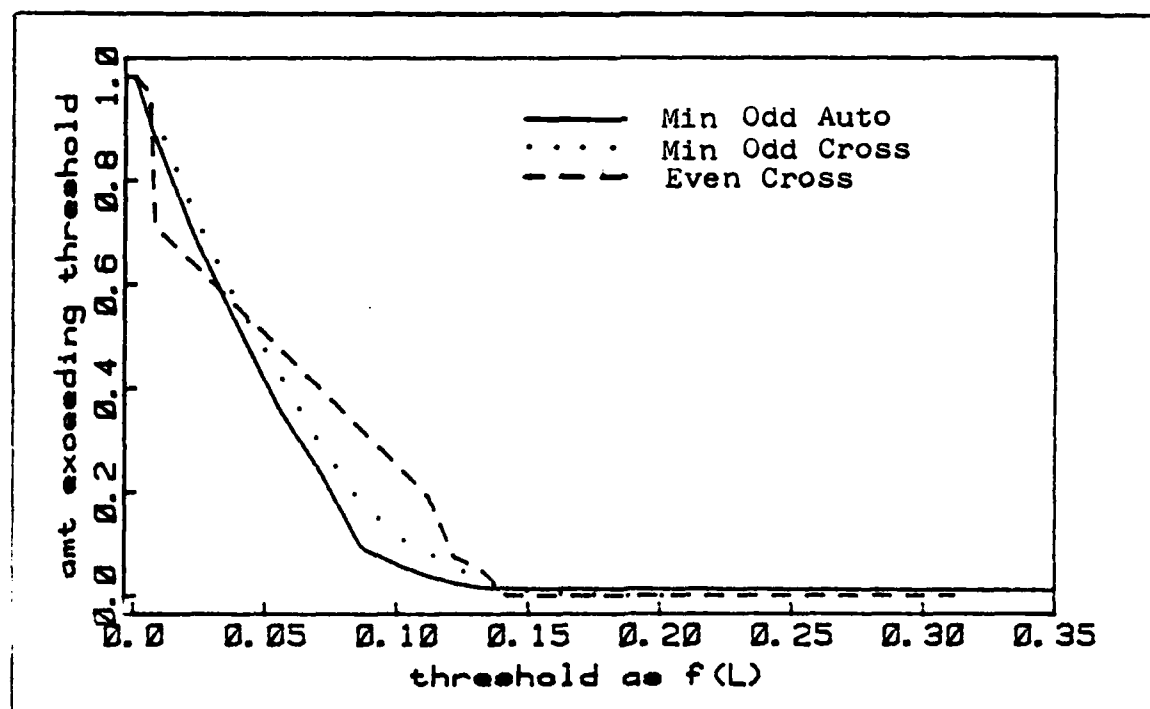


Fig. A-18. Threshold Plot of Minimum Odd Auto-Correlation, Minimum Odd Cross-Correlation, and Even Cross-Correlation of Maximal Codes 1 & 2, Length 127

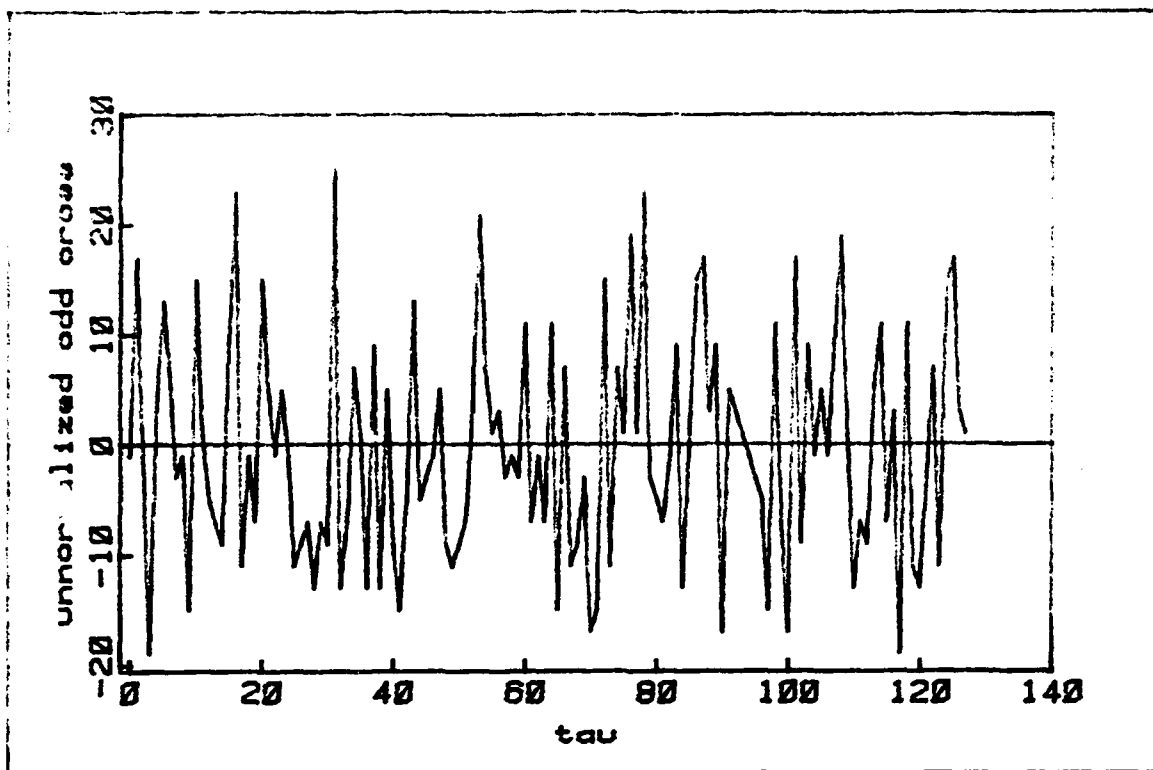


Fig. A-15. Optimal Odd Cross-Correlation Function of Maximal Codes 1 & 2, Length 127

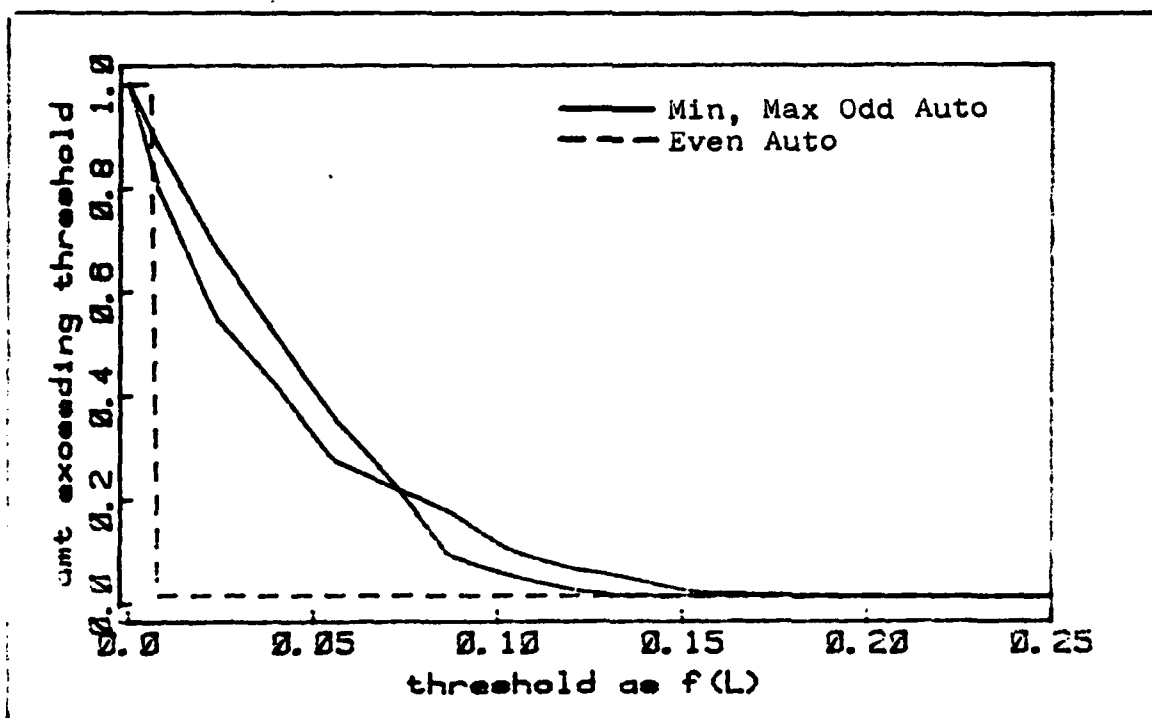


Fig. A-16. Threshold Plot of Minimum and Maximum Odd Auto-Correlation and Even Auto-Correlation of Maximal Codes 1 & 2, Length 127

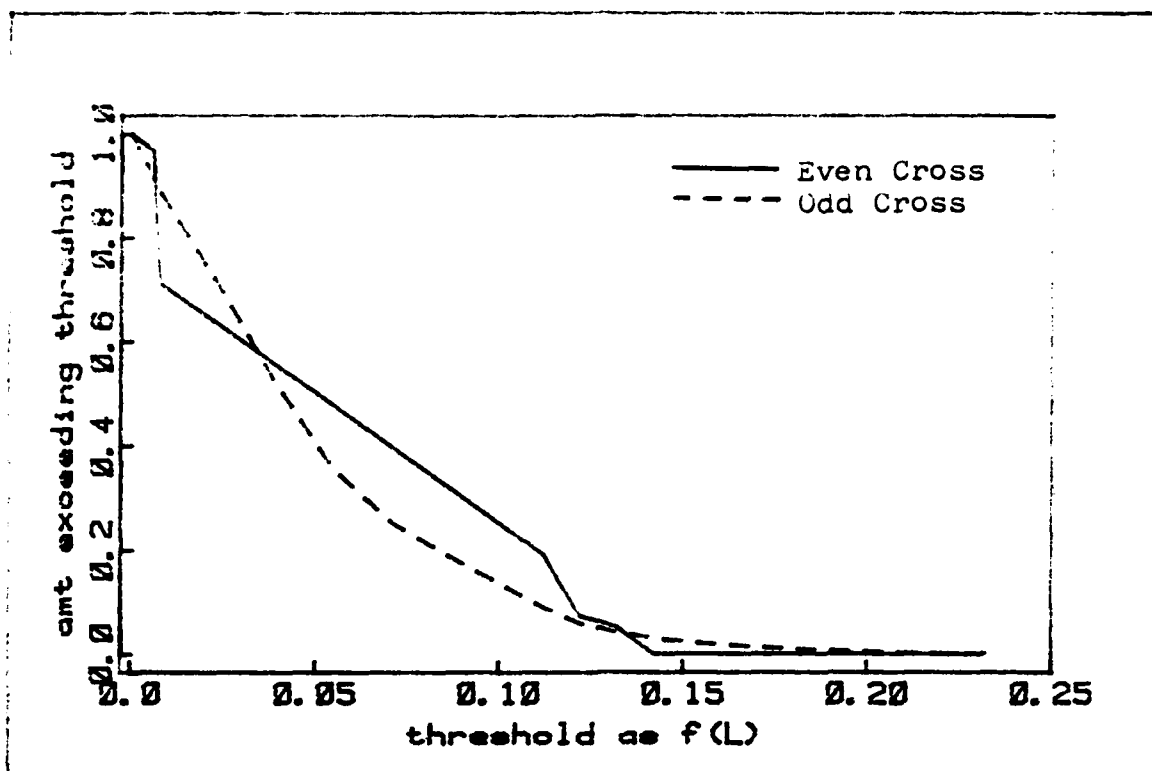


Fig. A-13. Typical Amount Exceeding Threshold vs Threshold of Even and Odd Correlation Functions Maximal Length Code of Length 127

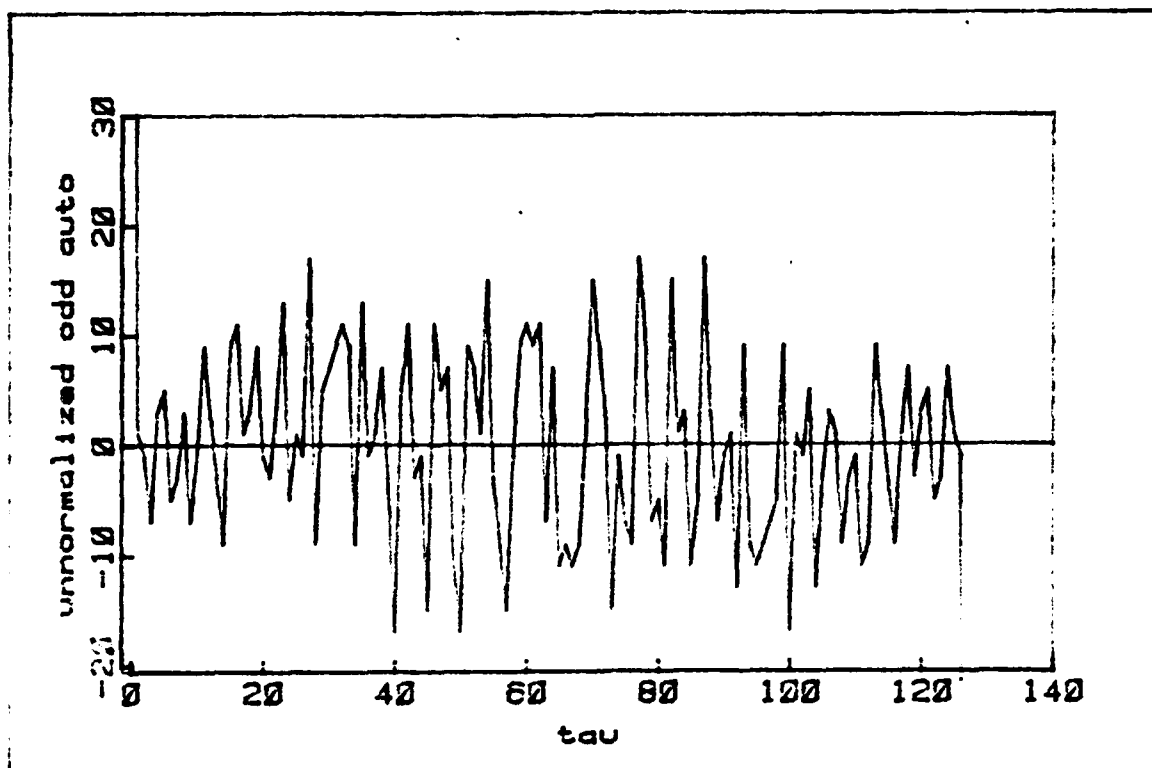


Fig. A-14. Optimal Odd Auto-Correlation Function of Maximal Codes 1 & 2, Length 127

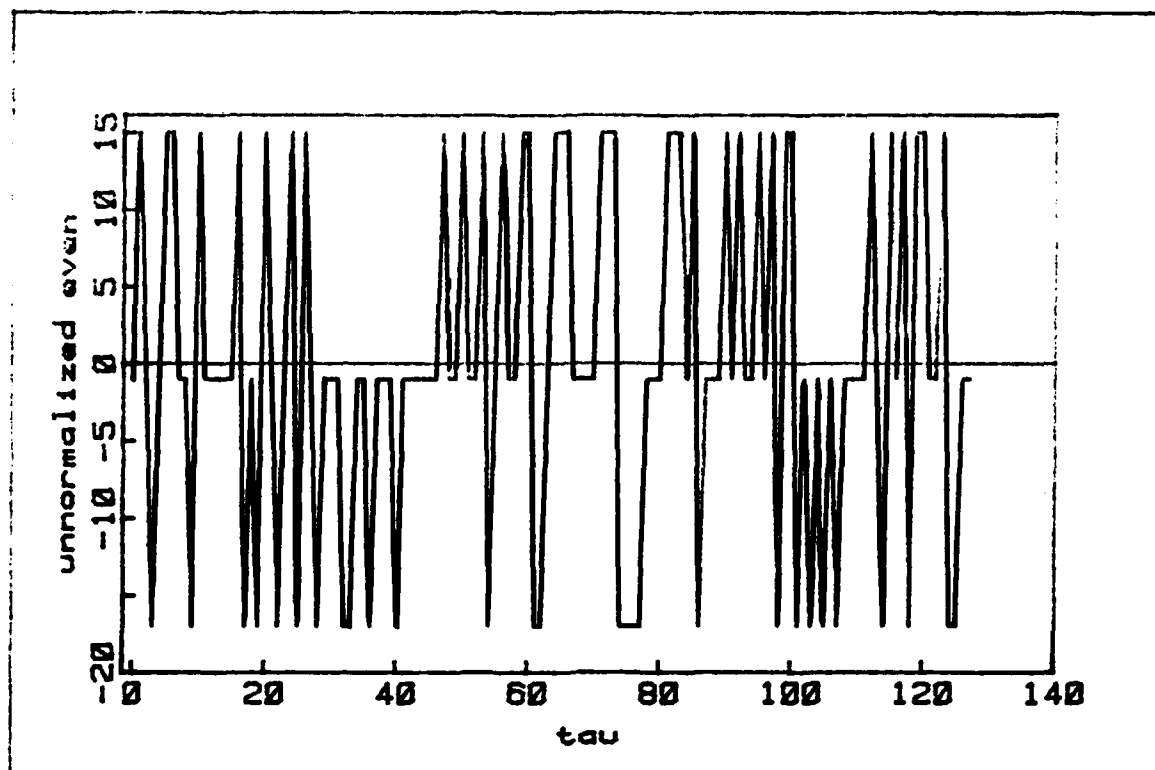


Fig. A-11. Typical Even Cross Correlation Function of Maximal Length Code of Length 127

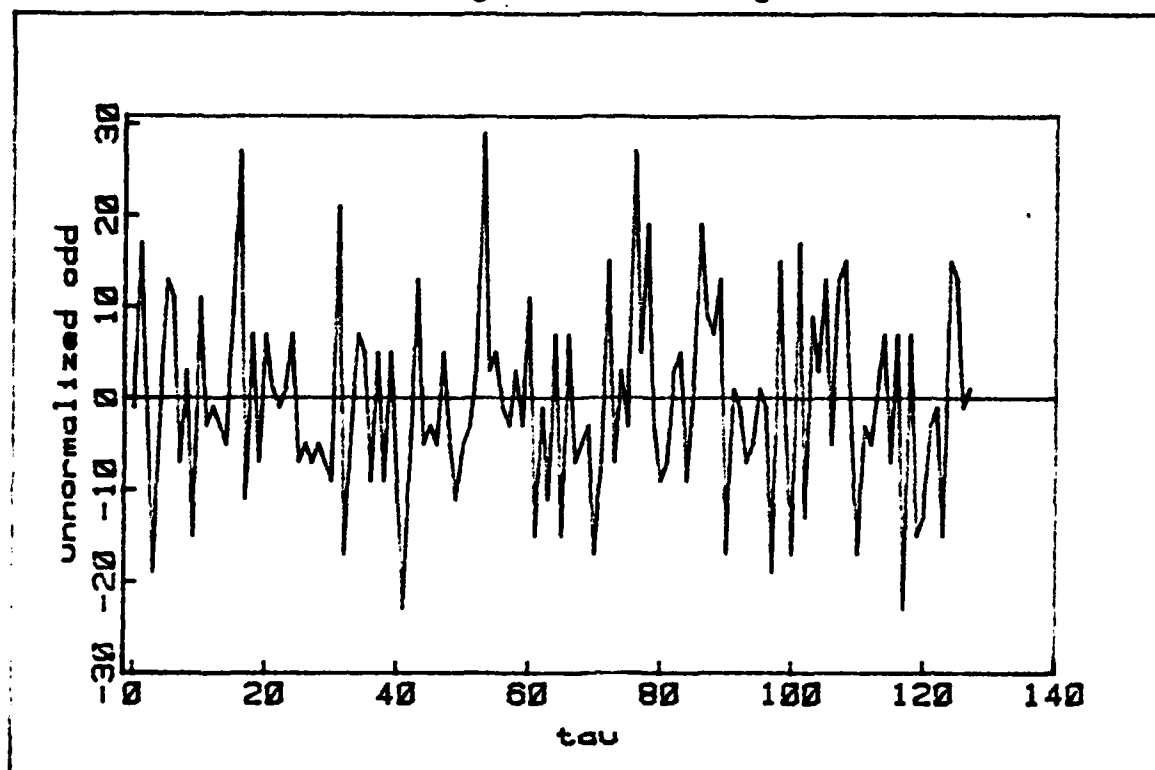


Fig. A-12. Typical Odd Cross Correlation Function of Maximal Length Code of Length 127

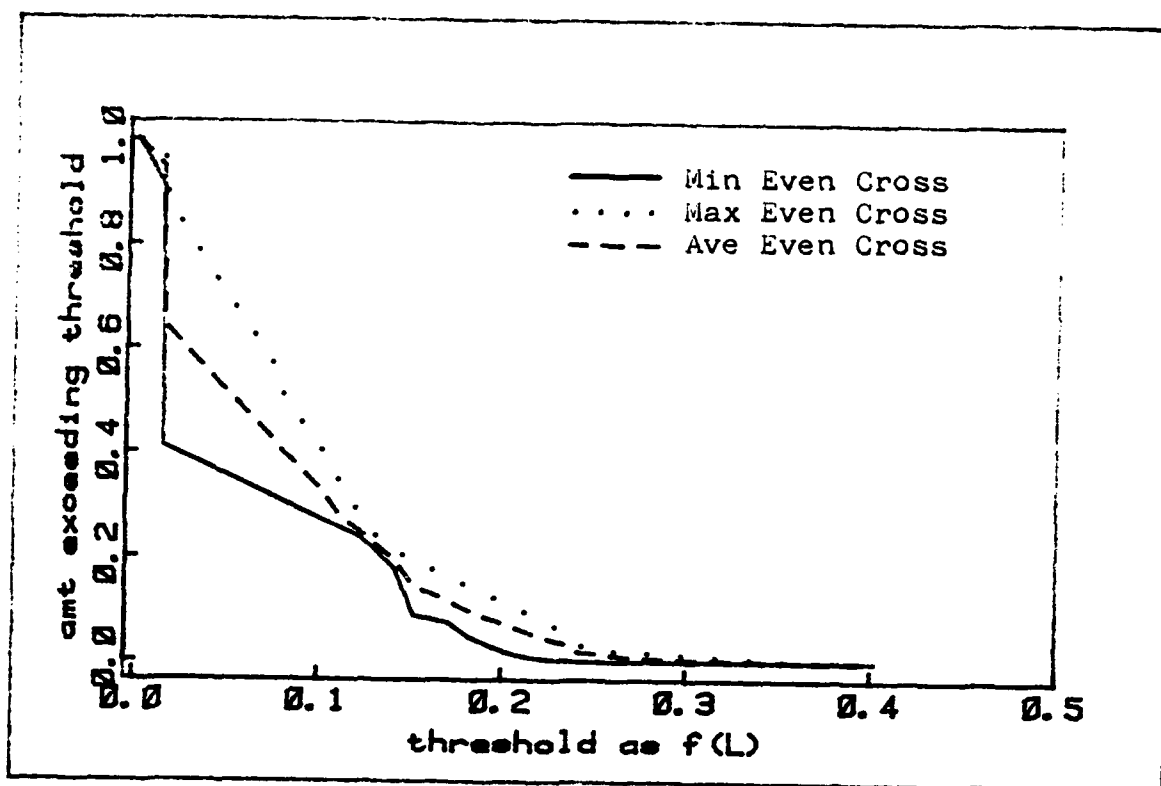


Fig. A-9. Threshold Plot of Minimum, Maximum and Average Even Correlation Functions from Mass-Correlation Algorithm for Maximal Codes 1-6, Length 63

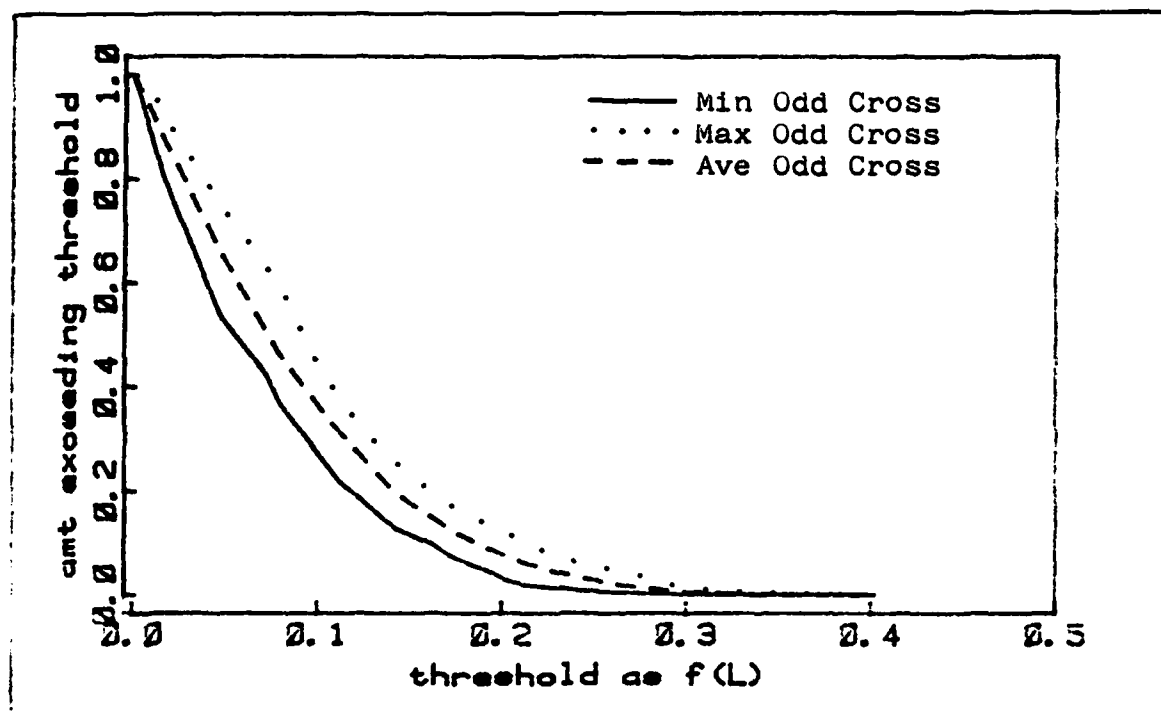


Fig. A-10. Threshold Plot of Minimum, Maximum and Average Odd Correlation Functions from Mass-Correlation Algorithm for Maximal Codes 1-6, Length 63

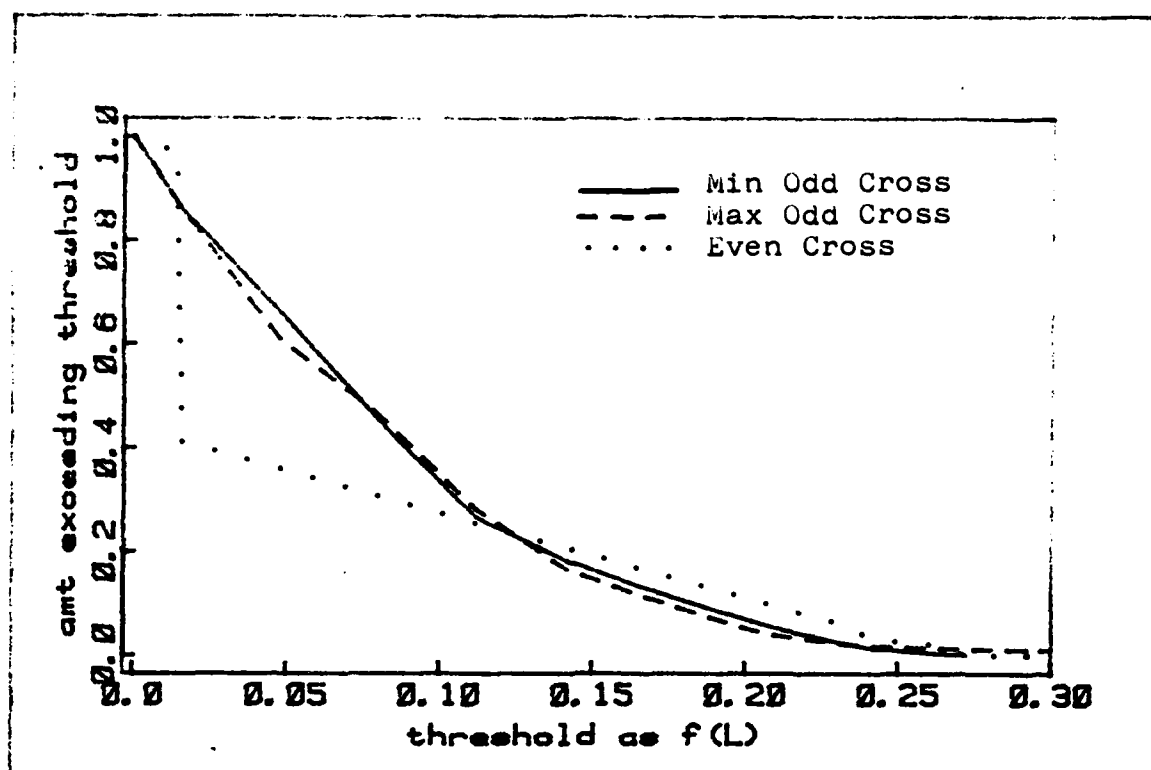


Fig. A-7. Threshold Plot of Minimum and Maximum Odd Cross-Correlation and Even Cross-Correlation of Maximal Codes 1 & 2, Length 63

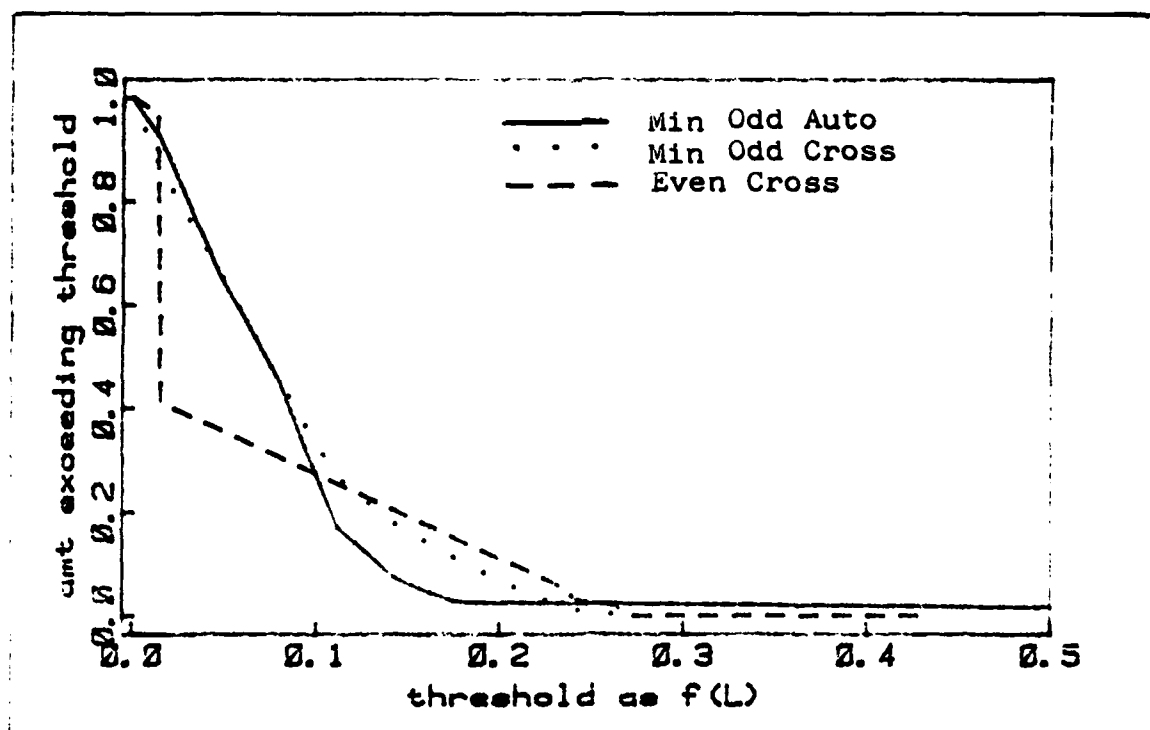


Fig. A-8. Threshold Plot of Minimum Odd Auto-Correlation, Minimum Odd Cross-Correlation, and Even Cross-Correlation of Maximal Codes 1 & 2, Length 63

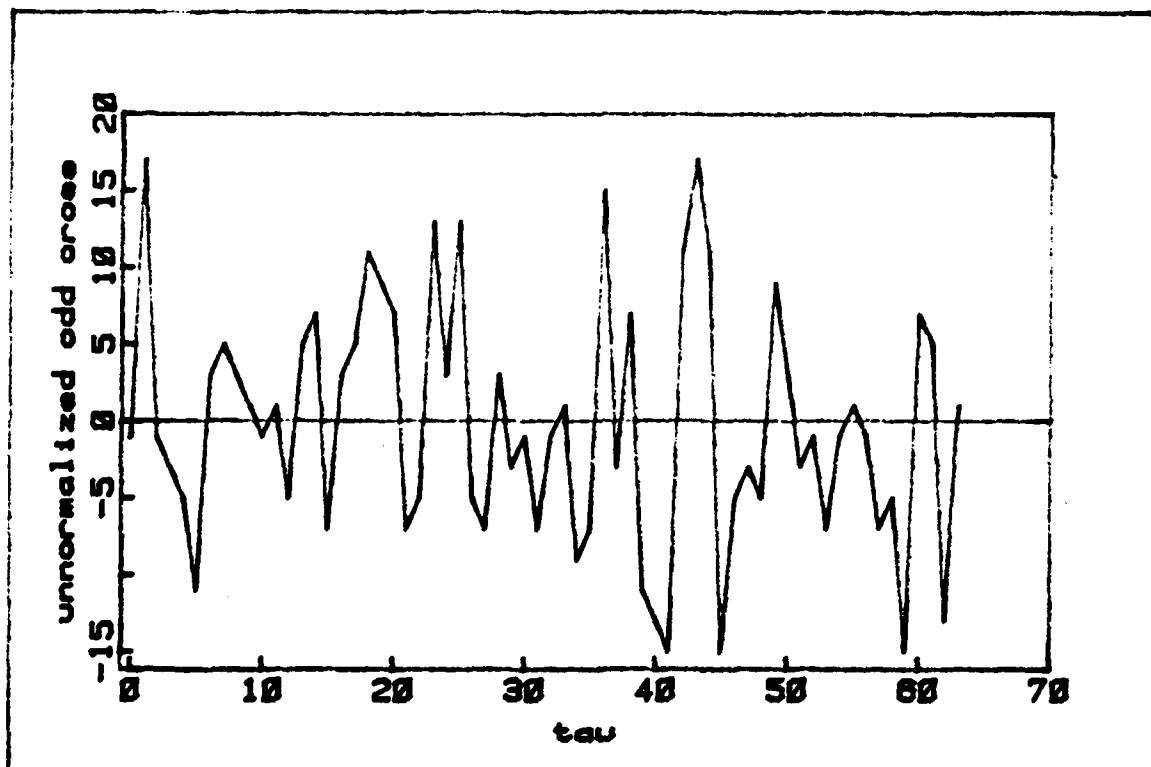


Fig. A-5. Optimal Odd Cross-Correlation Function of Maximal Codes 1 & 2, Length 63

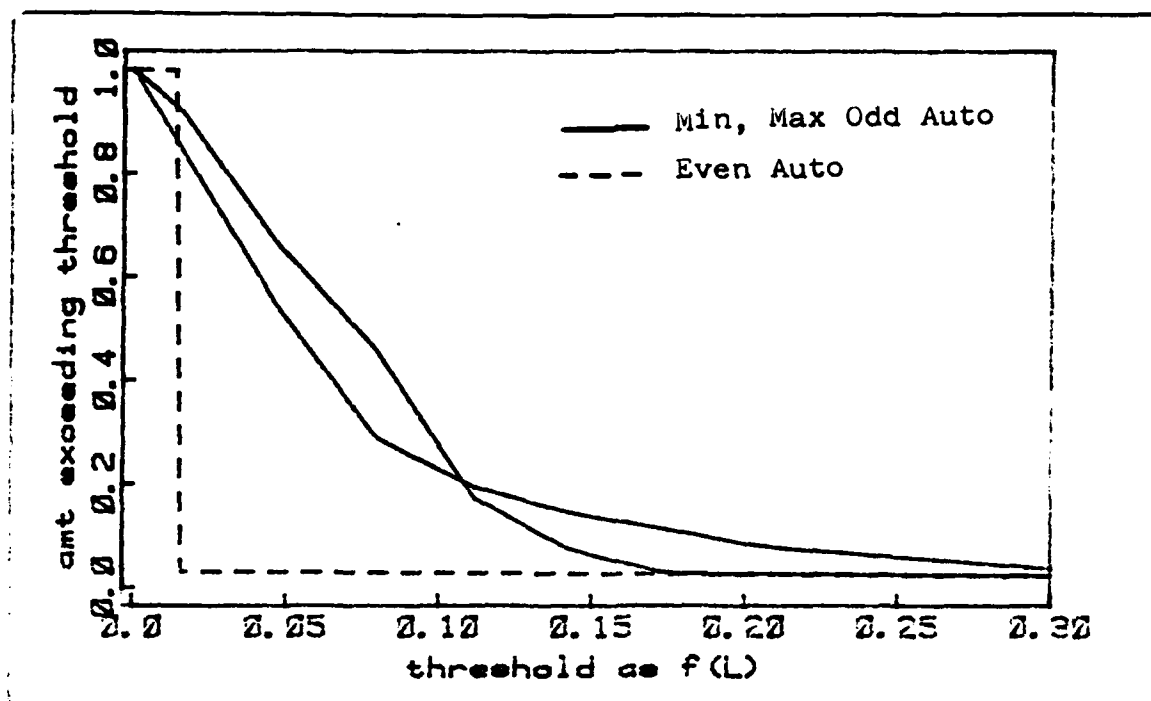


Fig. A-6. Threshold Plot of Minimum and Maximum Odd Auto-Correlation and Even Auto-Correlation of Maximal Codes 1 & 2, Length 63

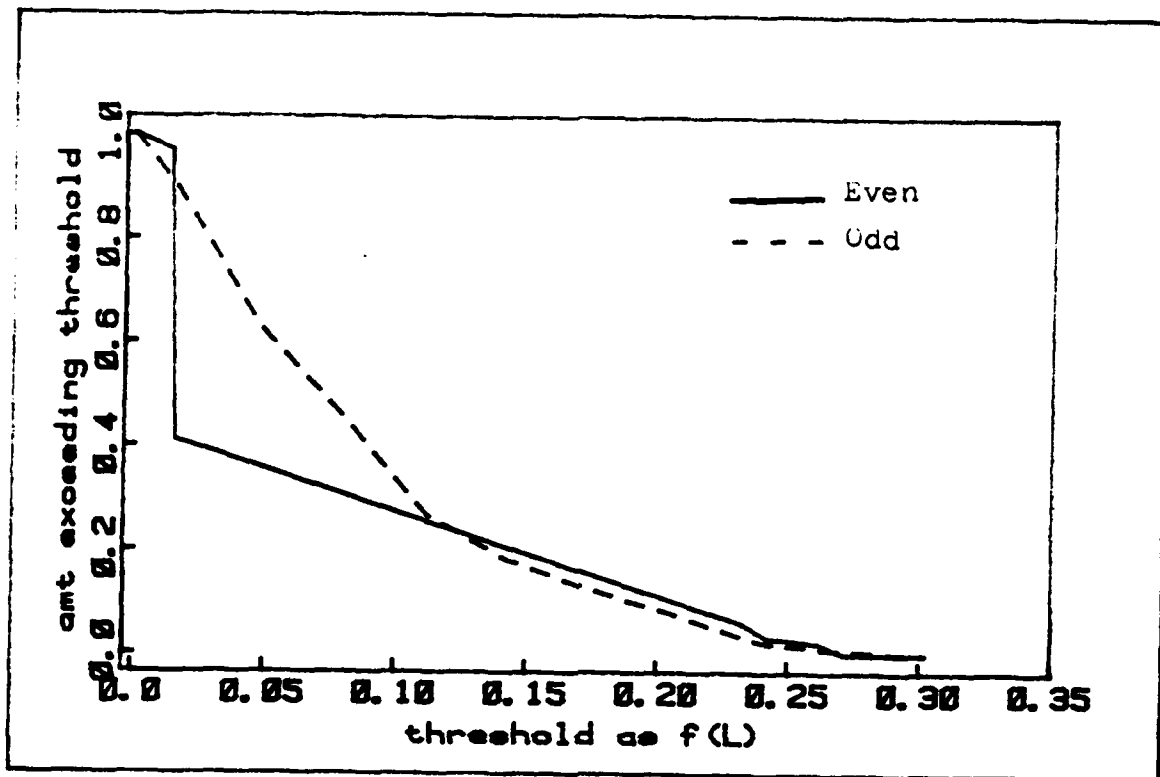


Fig. A-3. Typical Amount Exceeding Threshold vs Threshold of Even and Odd Correlation Functions Maximal Length Code of Length 63

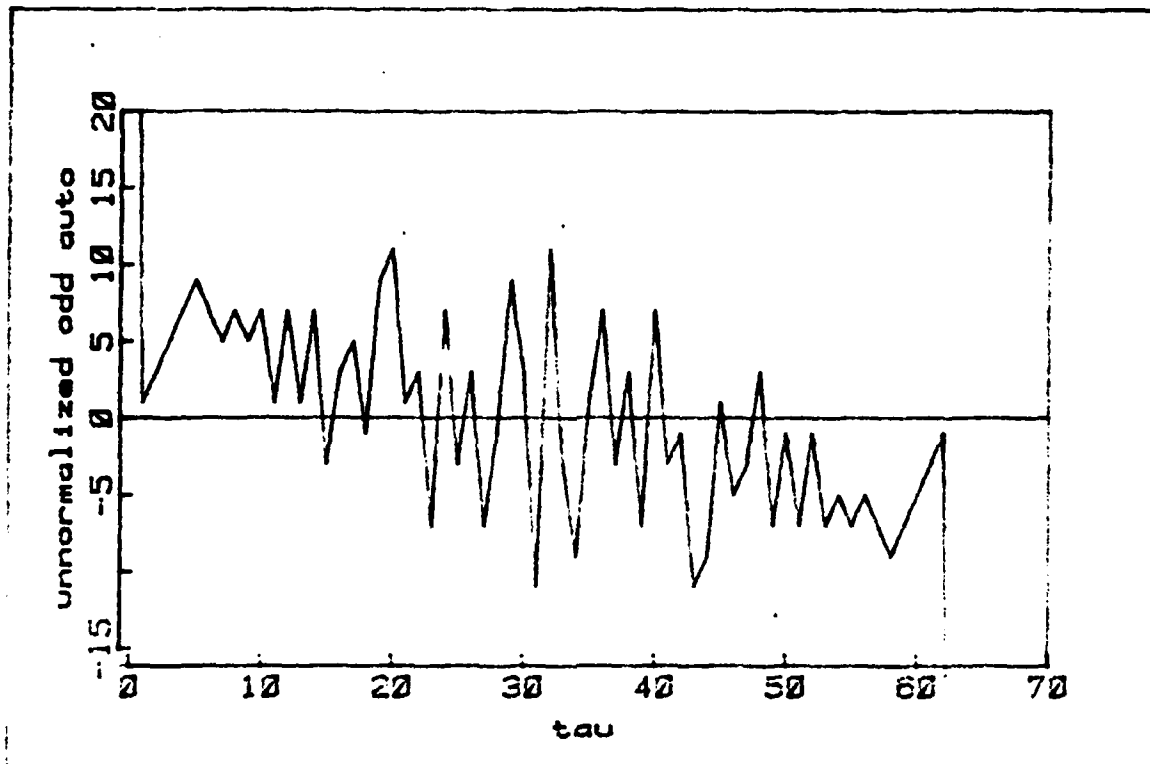


Fig. A-4. Optimal Odd Auto-Correlation Function of Maximal Codes 1 & 2, Length 63

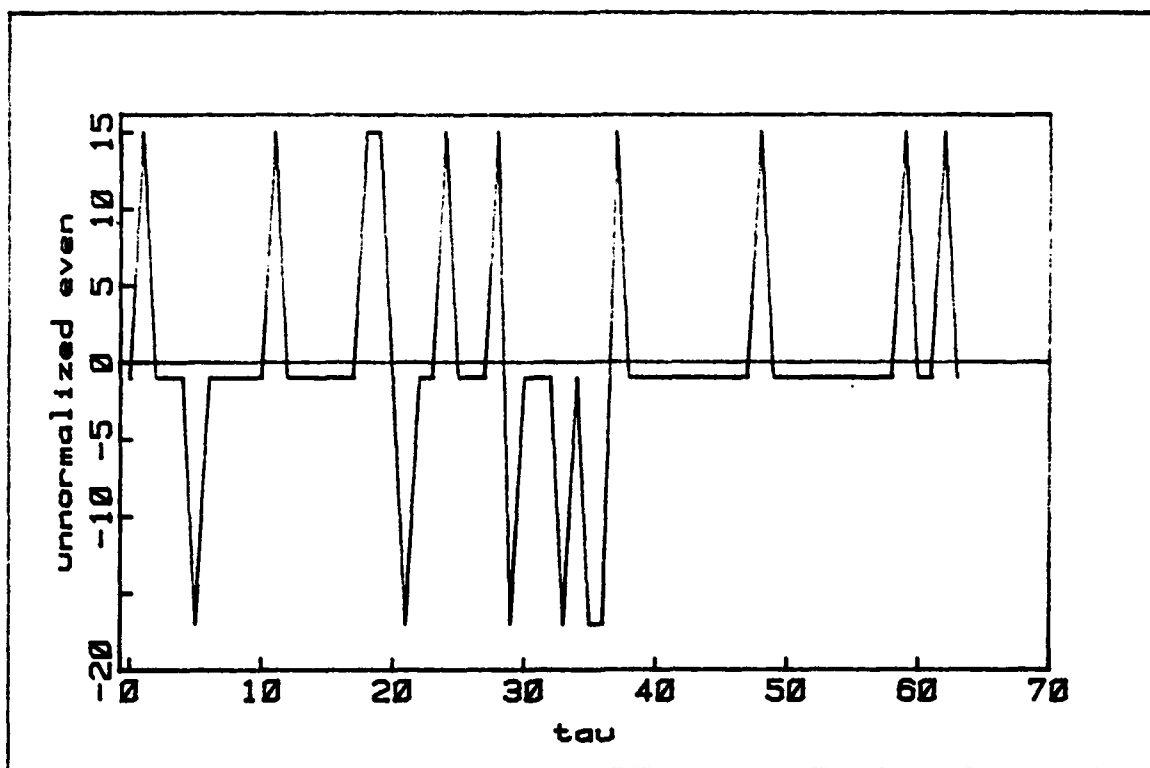


Fig. A-1. Typical Even Cross Correlation Function of Maximal Length Code of Length 63

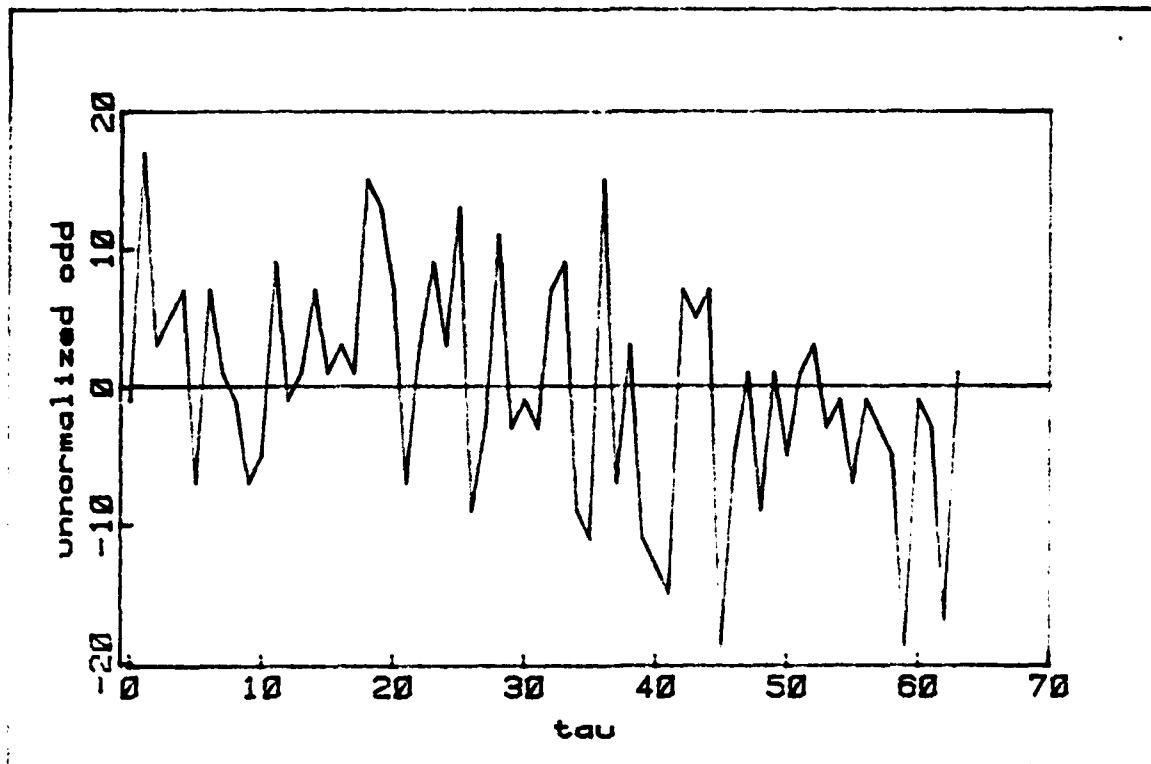


Fig. A-2. Typical Odd Cross Correlation Function of Maximal Length Code of Length 63

Figure		Page
A-71	Threshold Plot of Maximum Even Correlation Functions from Mass-Correlation Algorithm for Kasami Codes of Length 63, 255, 1023.....	A-43
A-72	Threshold Plot of Maximum Odd Correlation Functions from Mass-Correlation Algorithm for Kasami Codes of Length 63, 255, 1023.....	A-43
A-73	Threshold Plot of Maximum Even Correlation Functions from Mass-Correlation Algorithm for Maximal, Gold and Kasami Codes of Length 63.....	A-44
A-74	Threshold Plot of Maximum Odd Correlation Functions from Mass-Correlation Algorithm for Maximal, Gold and Kasami Codes of Length 63.....	A-44
A-75	Threshold Plot of Maximum Even Correlation Functions from Mass-Correlation Algorithm for Maximal and Gold Codes of Length 127.....	A-45
A-76	Threshold Plot of Maximum Odd Correlation Functions from Mass-Correlation Algorithm for Maximal and Gold Codes of Length 127.....	A-45
A-77	Threshold Plot of Maximum Even Correlation Functions from Mass-Correlation Algorithm for Maximal and Kasami Codes of Length 255.....	A-46
A-78	Threshold Plot of Maximum Odd Correlation Functions from Mass-Correlation Algorithm for Maximal and Kasami Codes of Length 255.....	A-46
A-79	Threshold Plot of Maximum Even Correlation Functions from Mass-Correlation Algorithm for Maximal, Gold and Kasami Codes of Length 1023.....	A-47
A-80	Threshold Plot of Maximum Odd Correlation Functions from Mass-Correlation Algorithm for Maximal, Gold and Kasami Codes of Length 1023.....	A-47

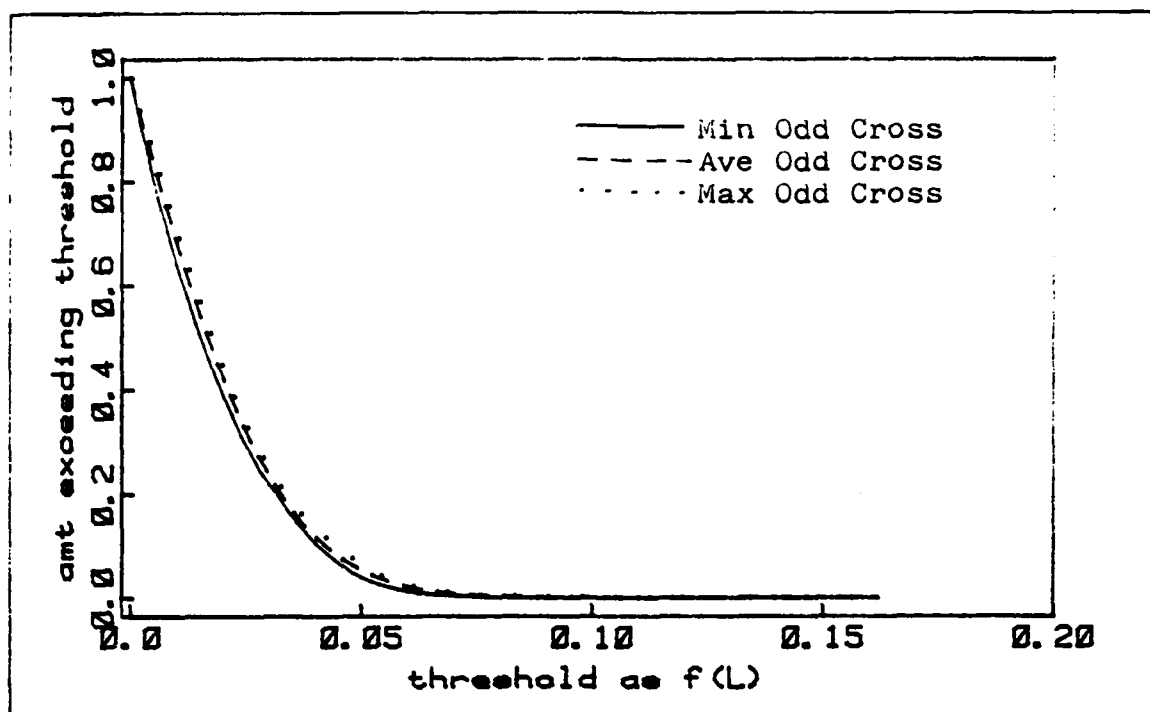


Fig. A-27. Threshold Plot of Minimum, Maximum and Average Odd Correlation Functions from Mass-Correlation Algorithm for Maximal Codes 1-8, Length 1023

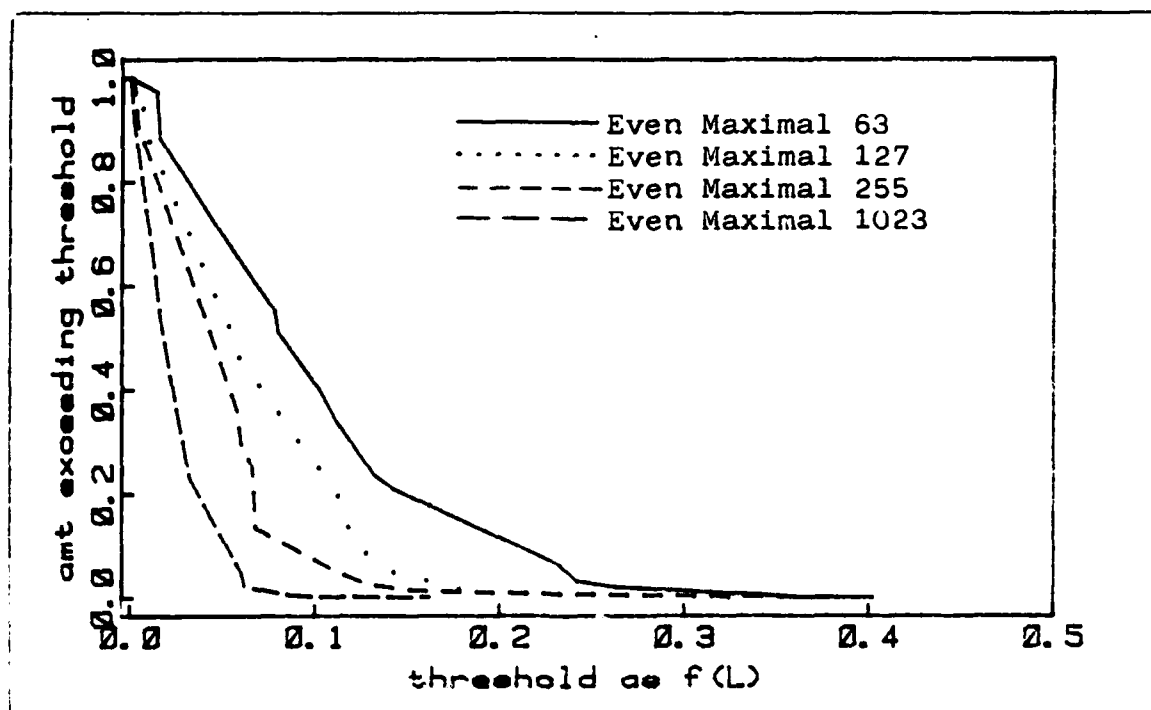


Fig. A-28. Threshold Plot of Maximum Even Correlation Functions from Mass-Correlation Algorithm for Maximal Codes of Length 63, 127, 255, 1023

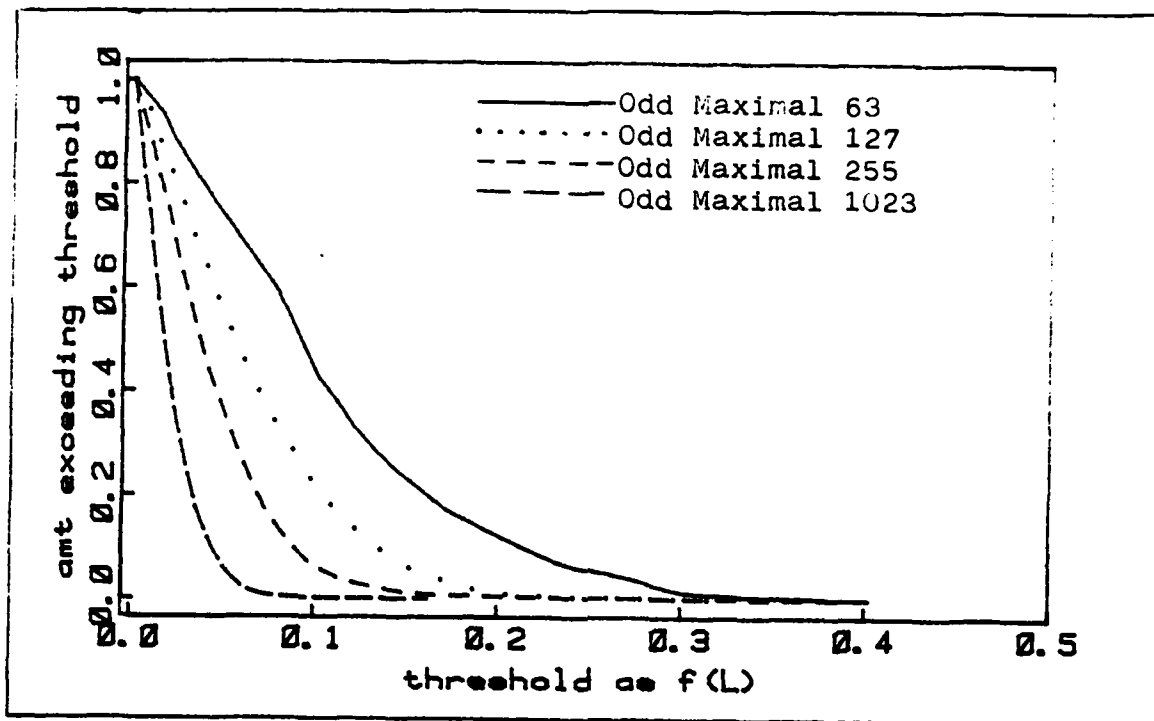


Fig. A-29. Threshold Plot of Maximum Odd Correlation Functions from Mass-Correlation Algorithm for Maximal Codes of Length 63, 127, 255, 1023

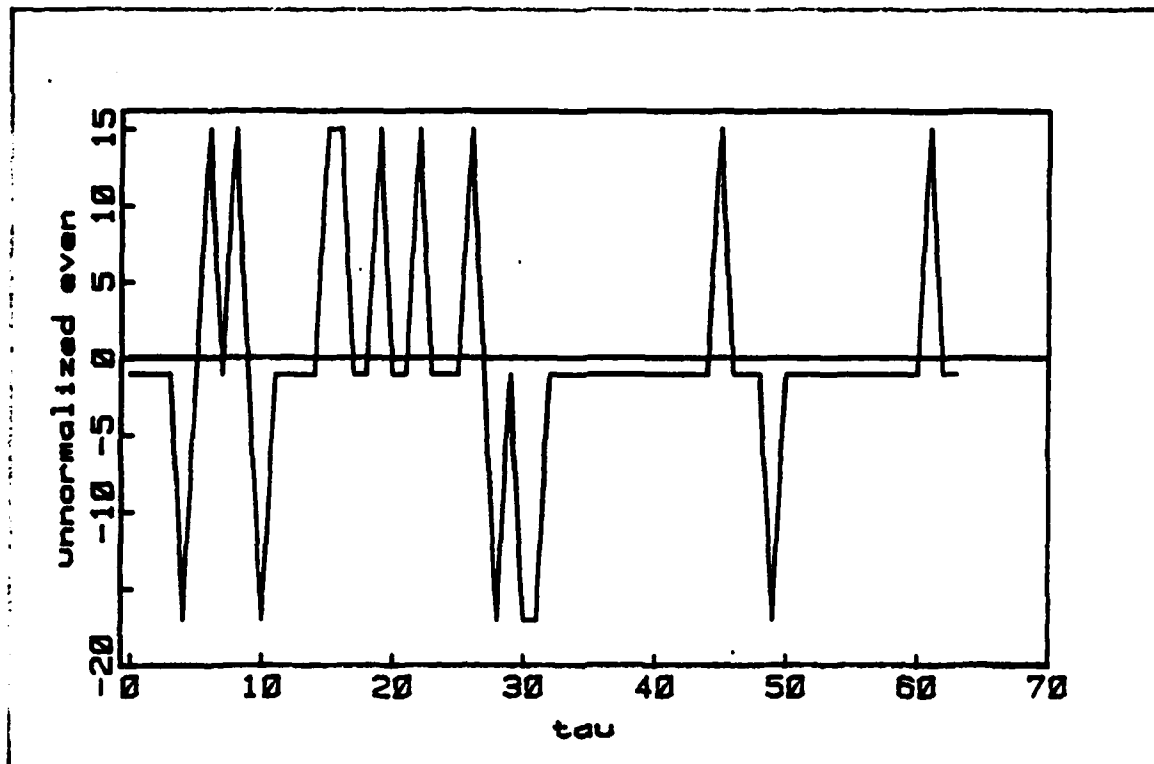


Fig. A-30. Typical Even Cross Correlation Function of Gold Code of Length 63

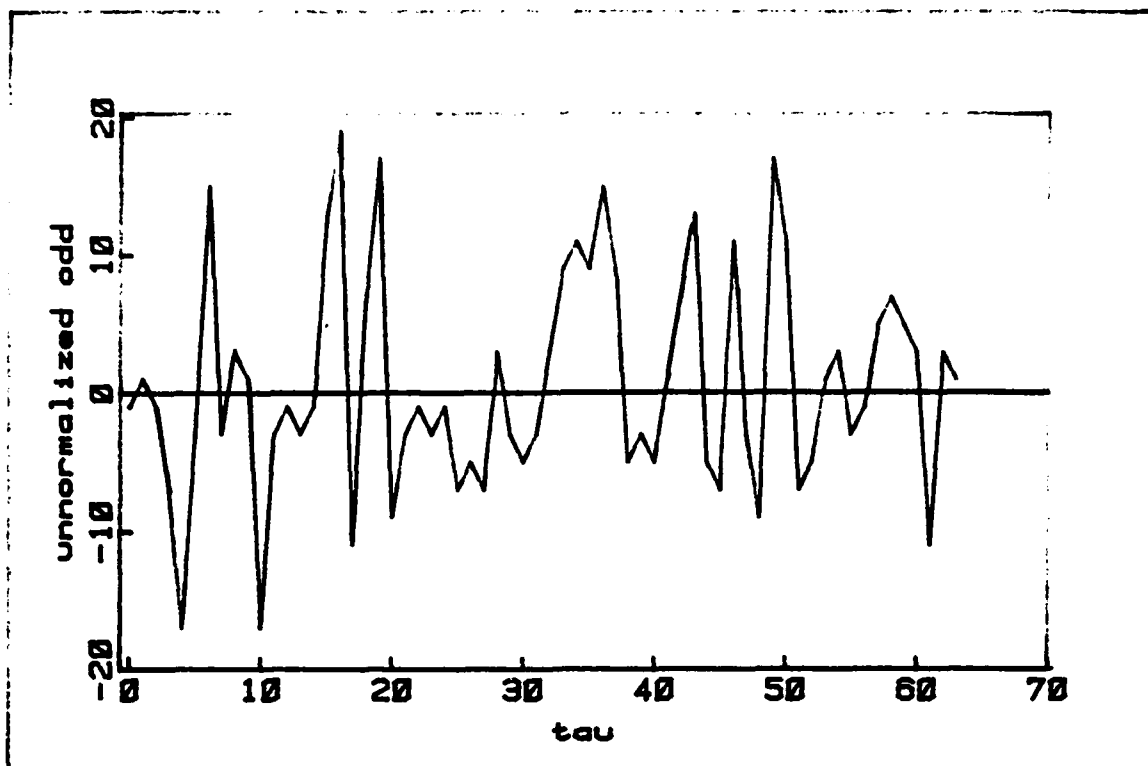


Fig. A-31. Typical Odd Cross Correlation Function of Gold Code of Length 63

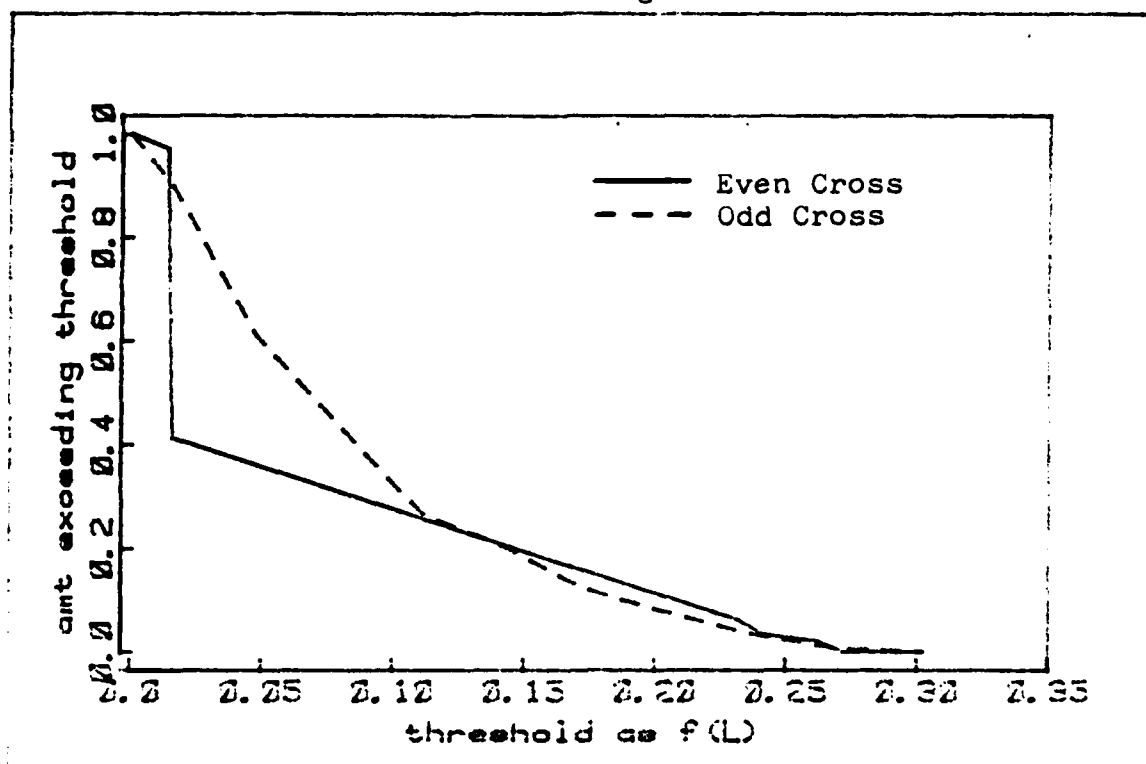


Fig. A-32. Typical Amount Exceeding Threshold vs Threshold of Even and Odd Correlation Functions Gold Code of Length 63

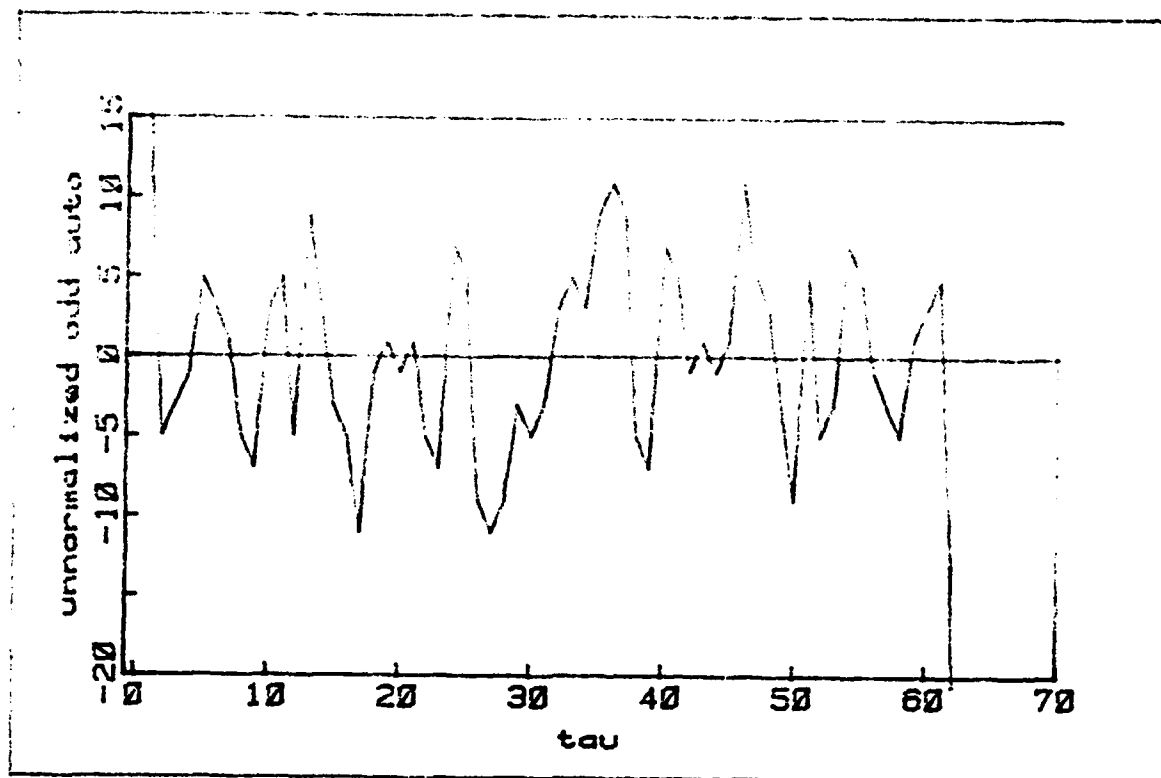


Fig. A-33. Optimal Odd Auto-Correlation Function of Gold Codes 3 & 4, Length 63

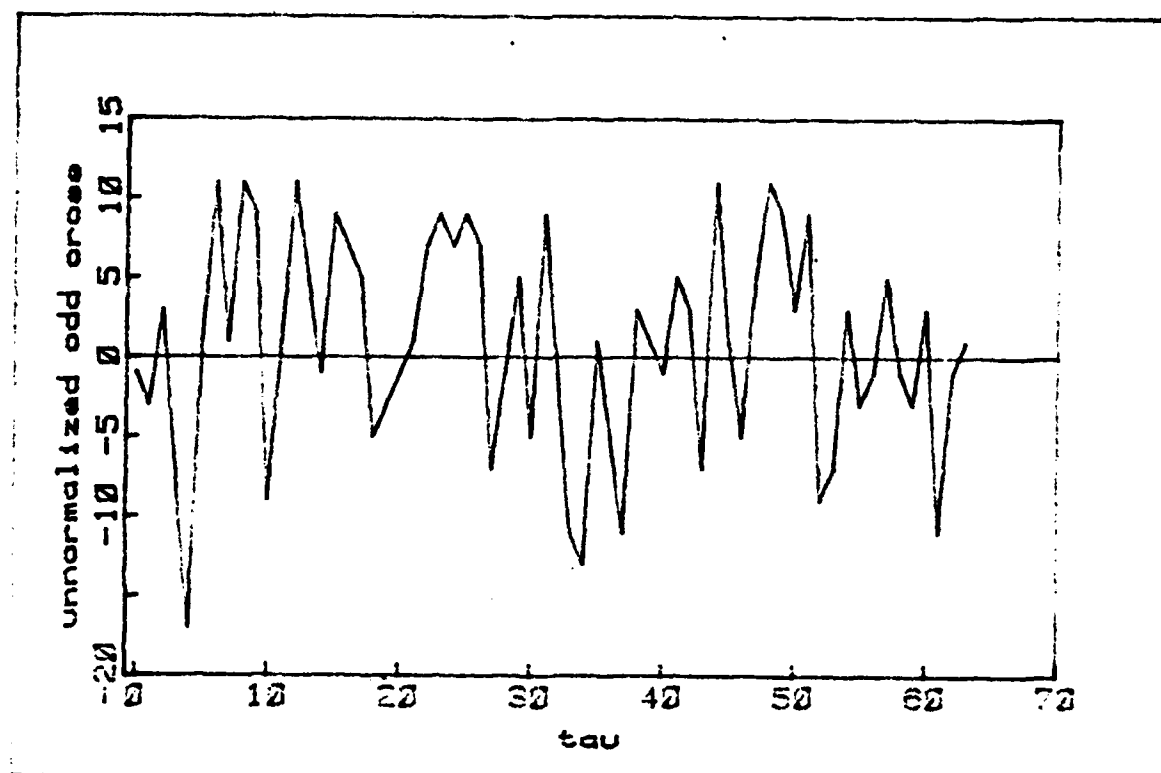


Fig. A-34. Optimal Odd Cross-Correlation Function of Gold Codes 3 & 4, Length 63

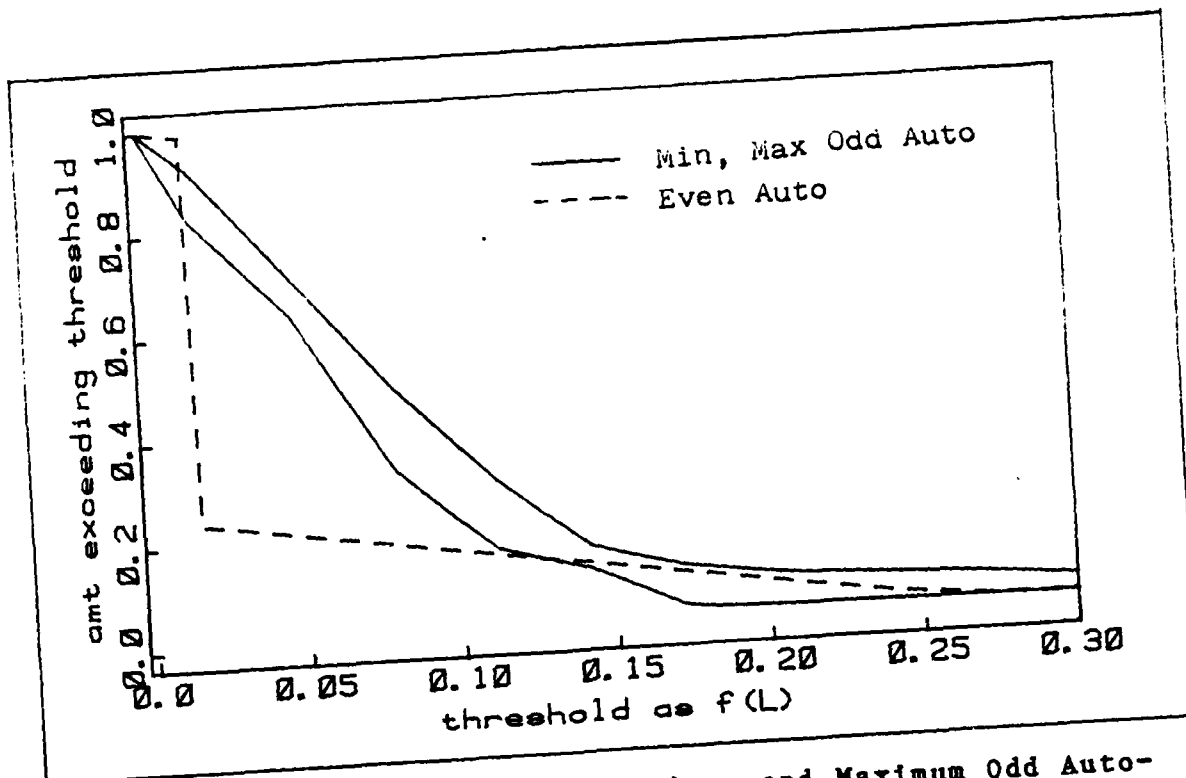


Fig. A-35. Threshold Plot of Minimum and Maximum Odd Auto-Correlation and Even Auto-Correlation of Gold Codes 3 & 4, Length 63

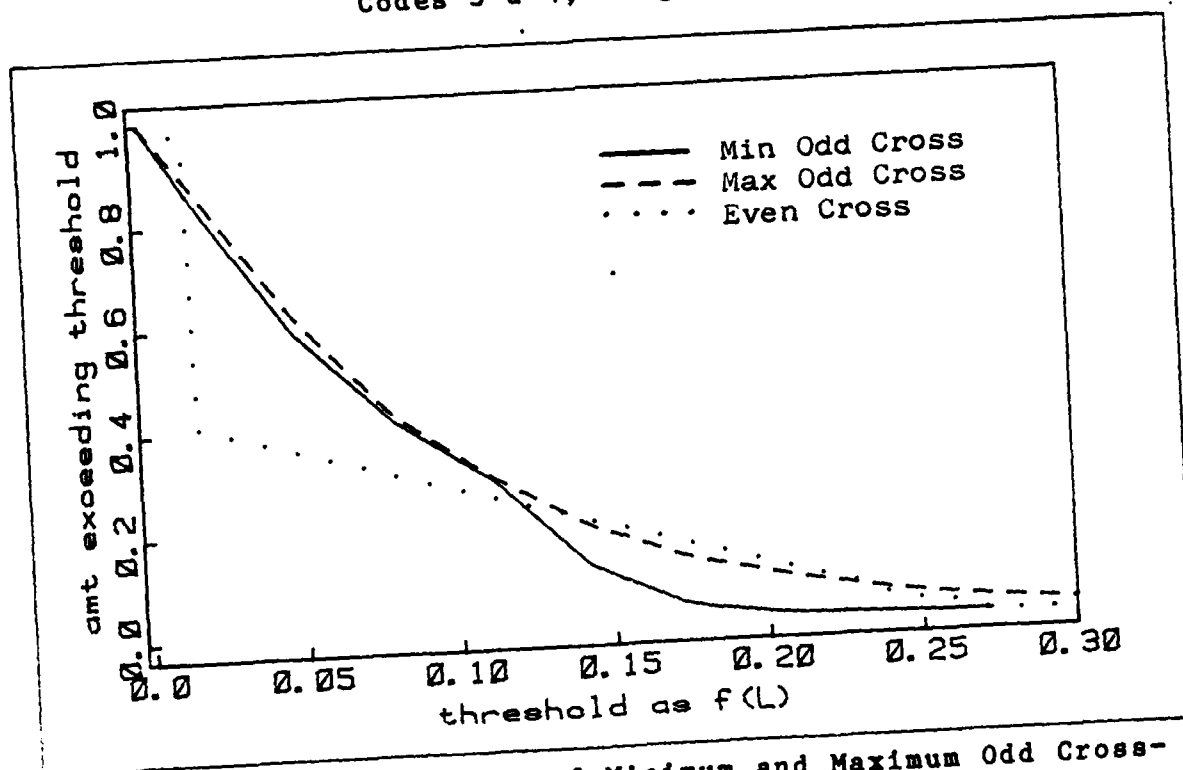


Fig. A-36. Threshold Plot of Minimum and Maximum Odd Cross-Correlation and Even Cross-Correlation of Gold Codes 3 & 4, Length 63

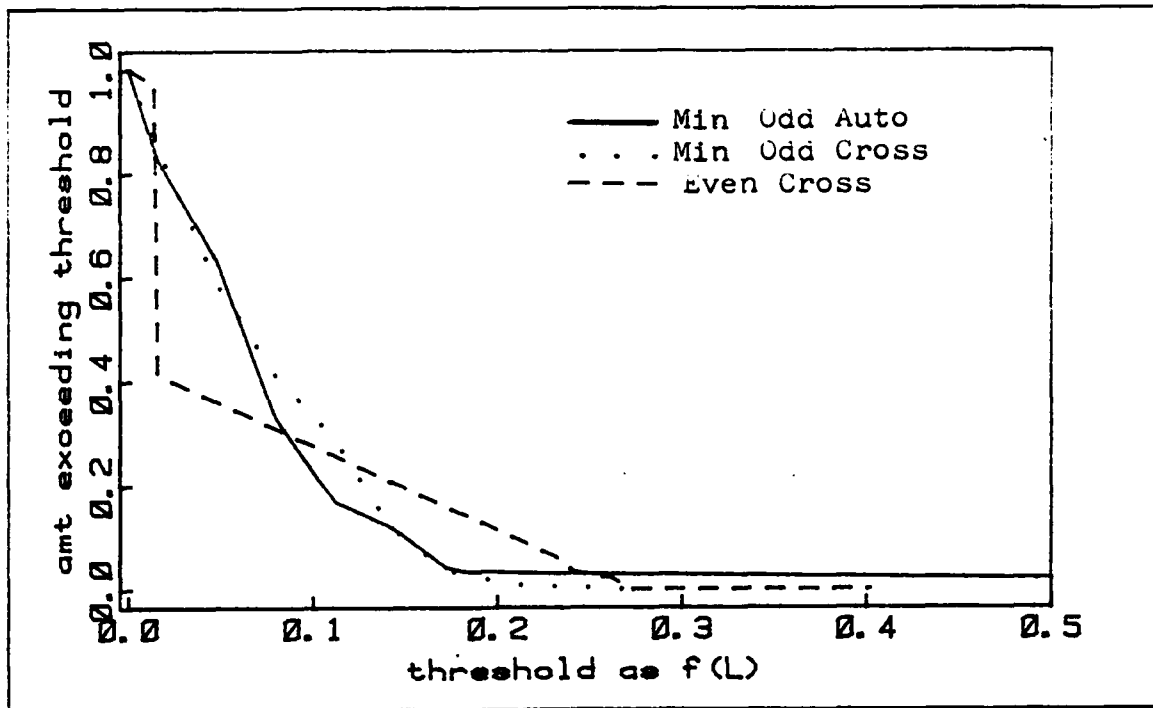


Fig. A-37. Threshold Plot of Minimum Odd Auto-Correlation, Minimum Odd Cross-Correlation, and Even Cross-Correlation of Gold Codes 3 & 4, Length 63

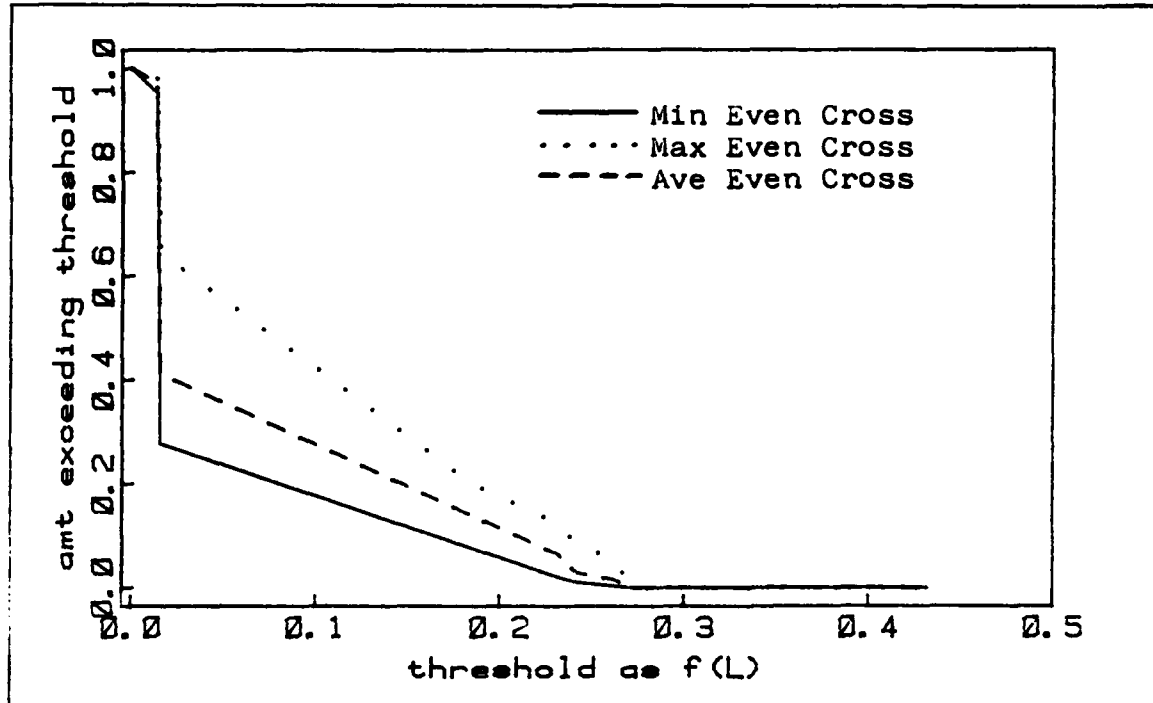


Fig. A-38. Threshold Plot of Minimum, Maximum and Average Even Correlation Functions from Mass-Correlation Algorithm for Gold Codes 1-10, Length 63

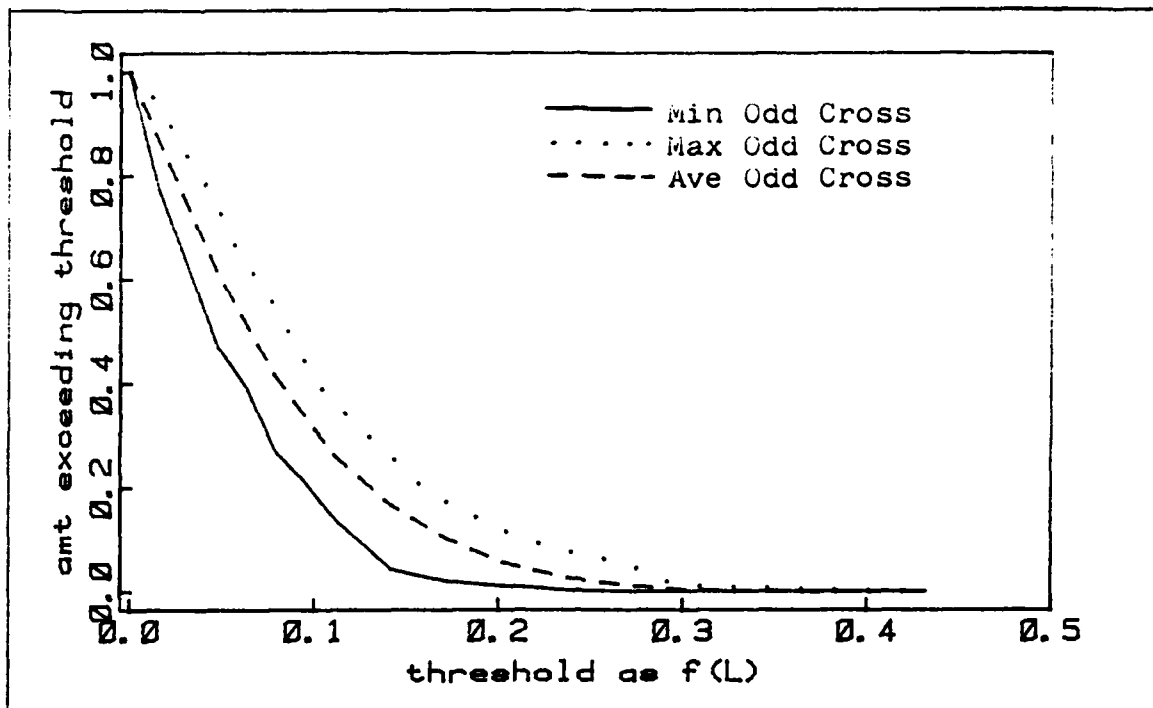


Fig. A-39. Threshold Plot of Minimum, Maximum and Average Odd Correlation Functions from Mass-Correlation Algorithm for Gold Codes 1-10, Length 63

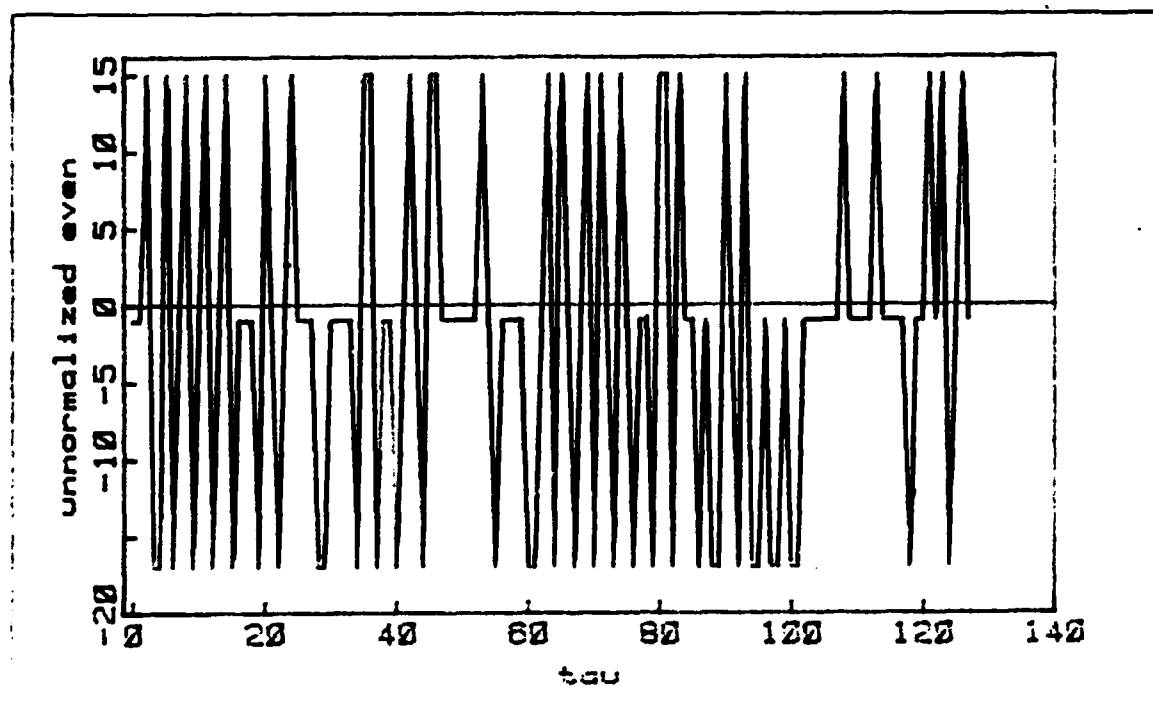


Fig. A-40. Typical Even Cross Correlation Function of Gold Code of Length 127

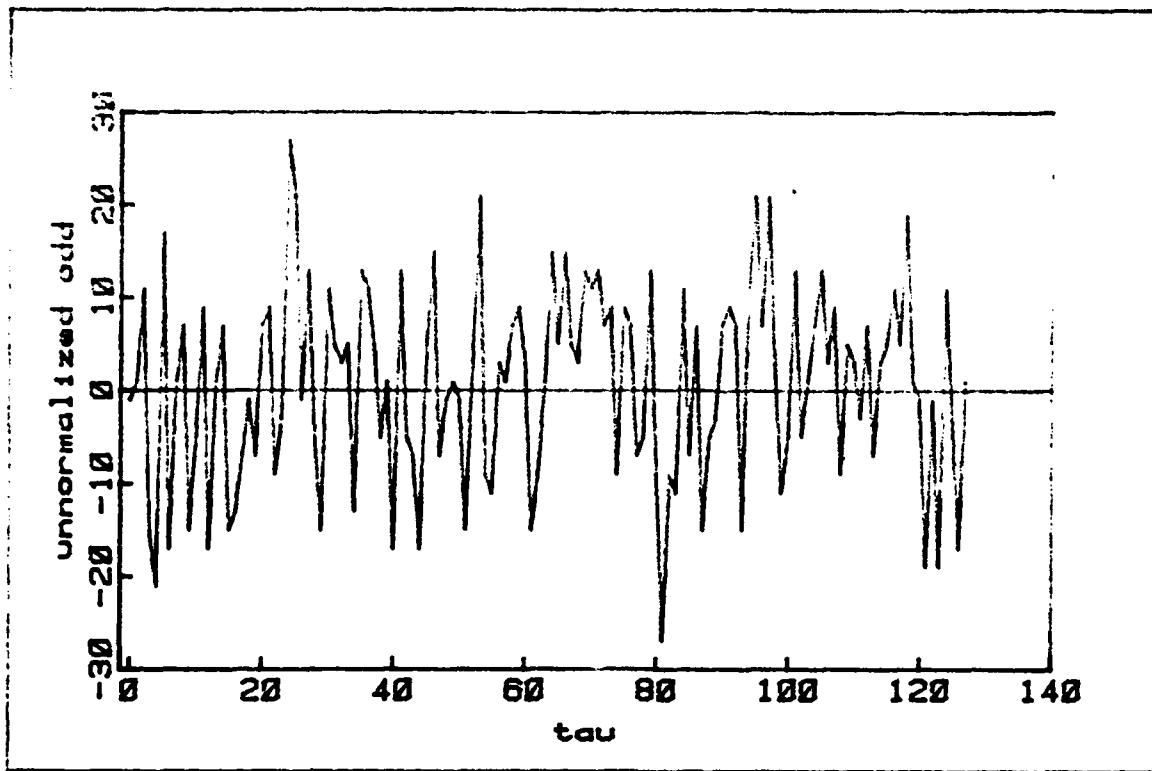


Fig. A-41. Typical Odd Cross Correlation Function of Gold Code of Length 127

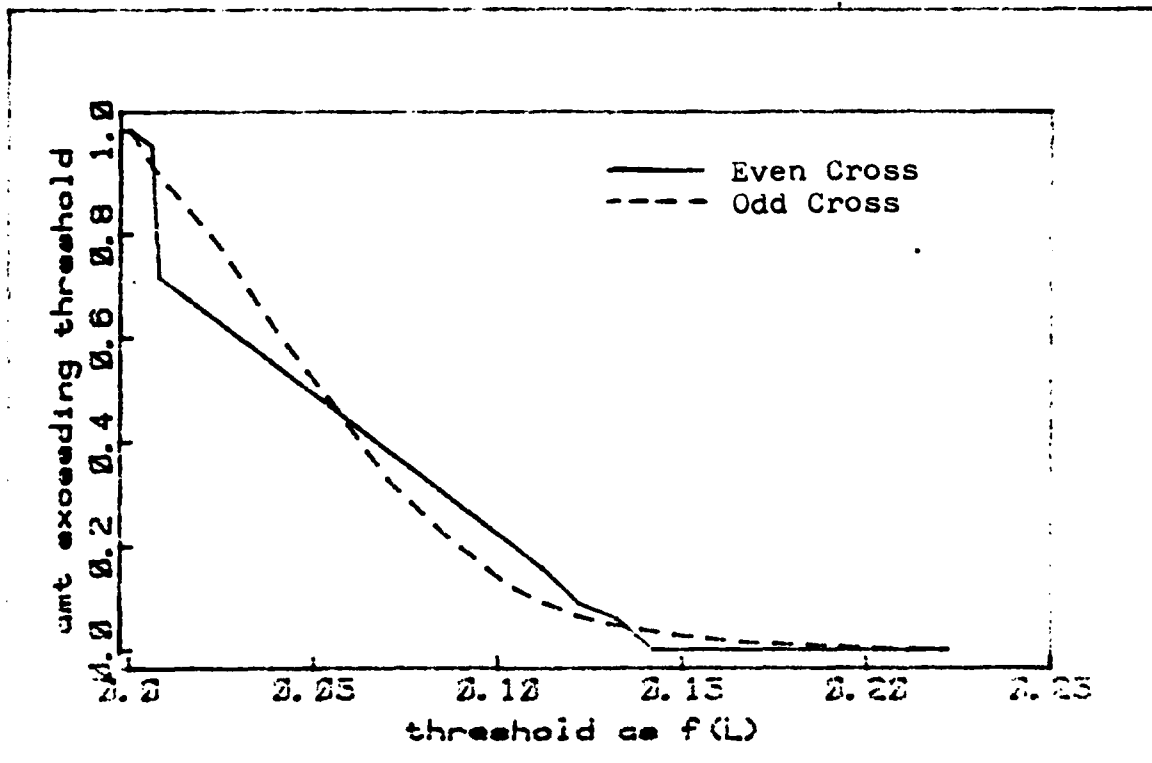


Fig. A-42. Typical Amount Exceeding Threshold vs Threshold of Even and Odd Correlation Functions Gold Code of Length 127

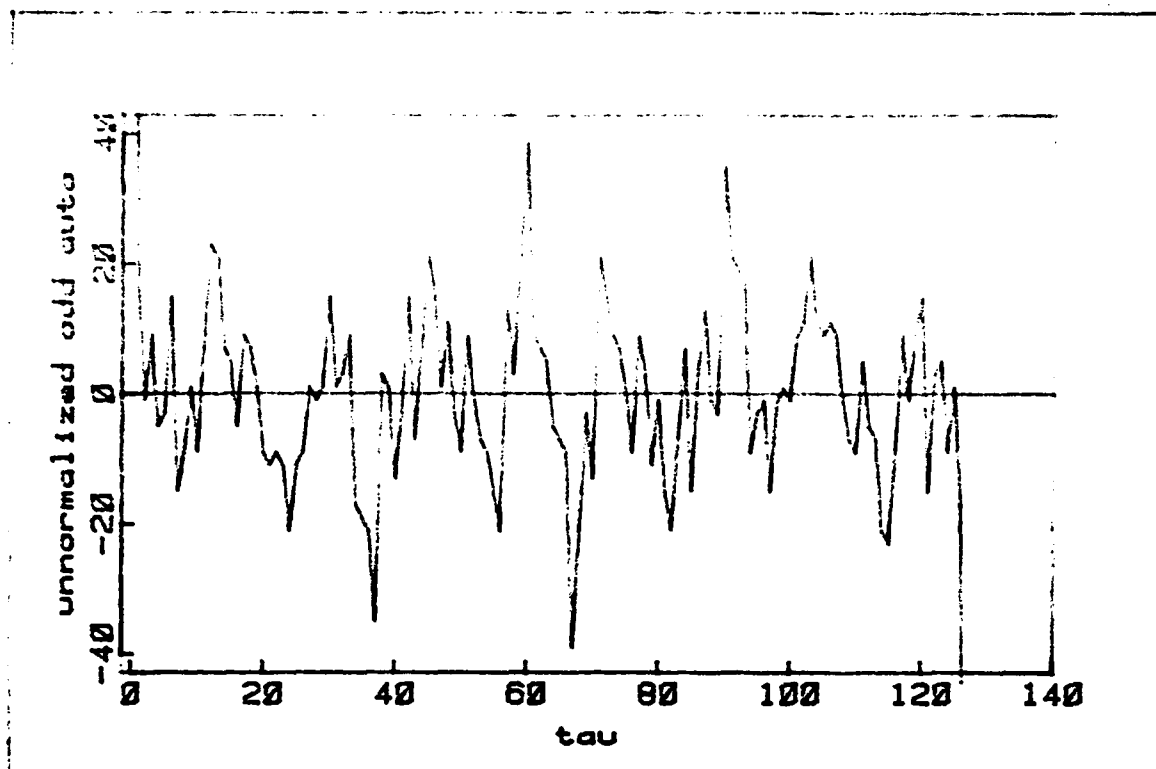


Fig. A-43. Optimal Odd Auto-Correlation Function of Gold Codes 3 & 4, Length 127

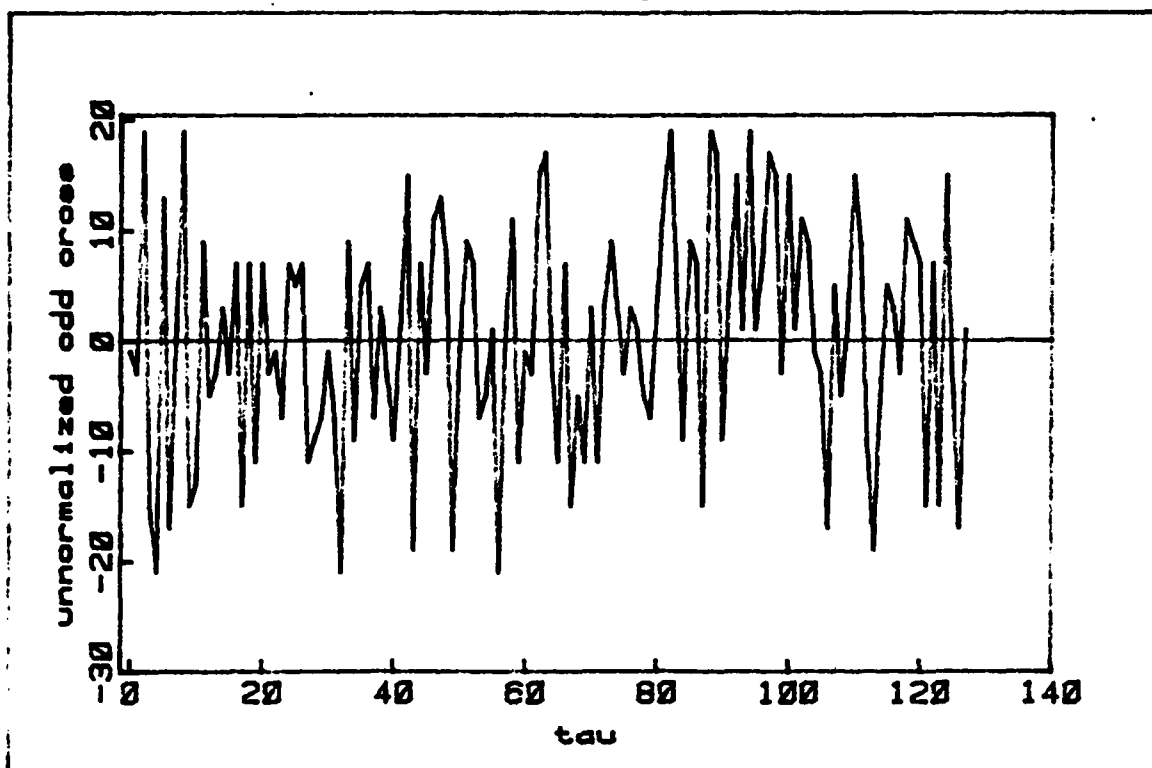


Fig. A-44. Optimal Odd Cross-Correlation Function of Gold Codes 3 & 4, Length 127

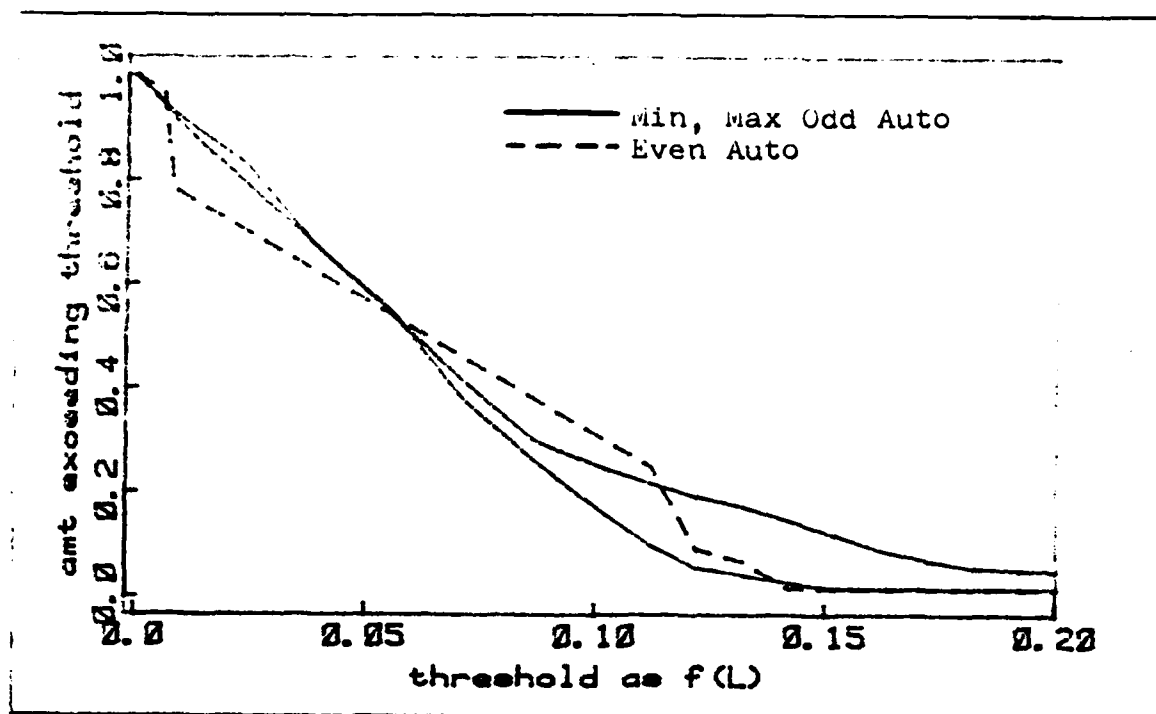


Fig. A-45. Threshold Plot of Minimum and Maximum Odd Auto-Correlation and Even Auto-Correlation of Gold Codes 3 & 4, Length 127

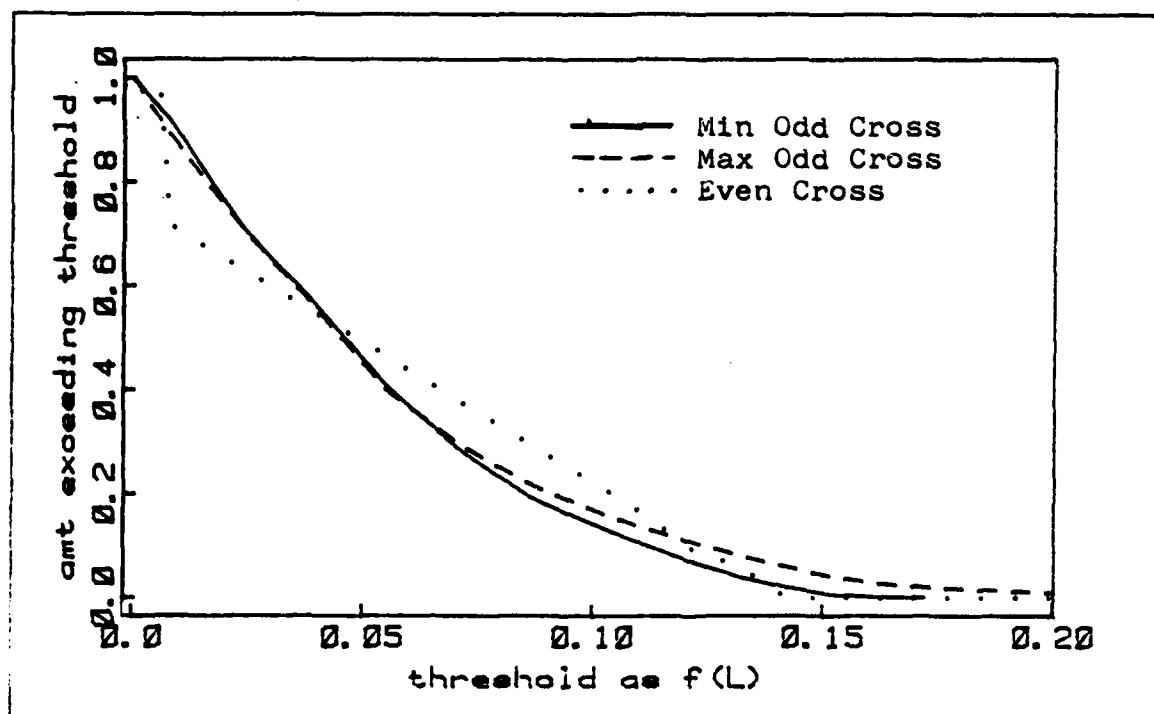


Fig. A-46. Threshold Plot of Minimum and Maximum Odd Cross-Correlation and Even Cross-Correlation of Gold Codes 3 & 4, Length 127

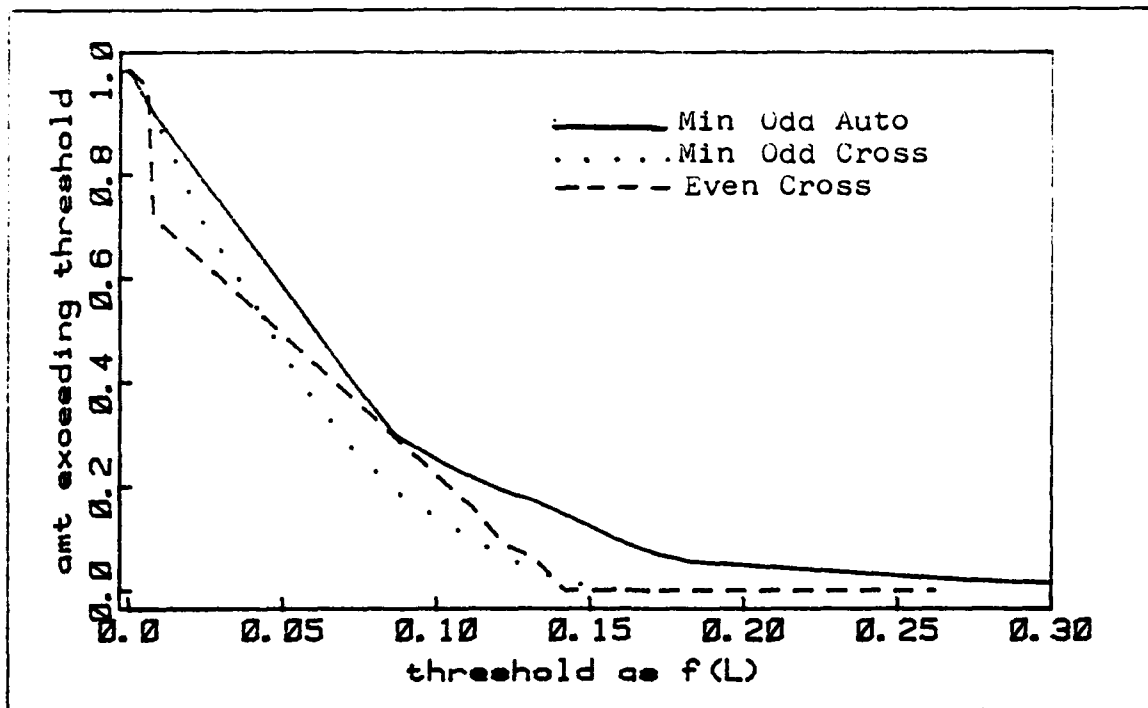


Fig. A-47. Threshold Plot of Minimum Odd Auto-Correlation, Minimum Odd Cross-Correlation, and Even Cross-Correlation of Gold Codes 3 & 4, Length 127

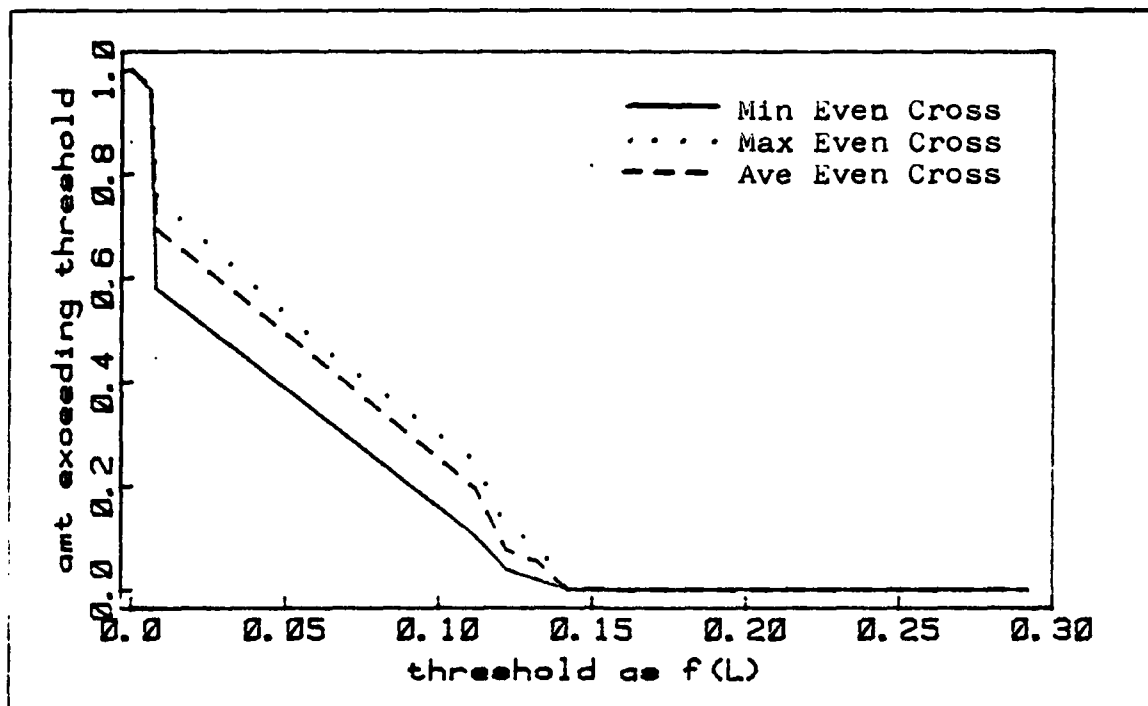


Fig. A-48. Threshold Plot of Minimum, Maximum and Average Even Correlation Functions from Mass-Correlation Algorithm for Gold Codes 1-10, Length 127

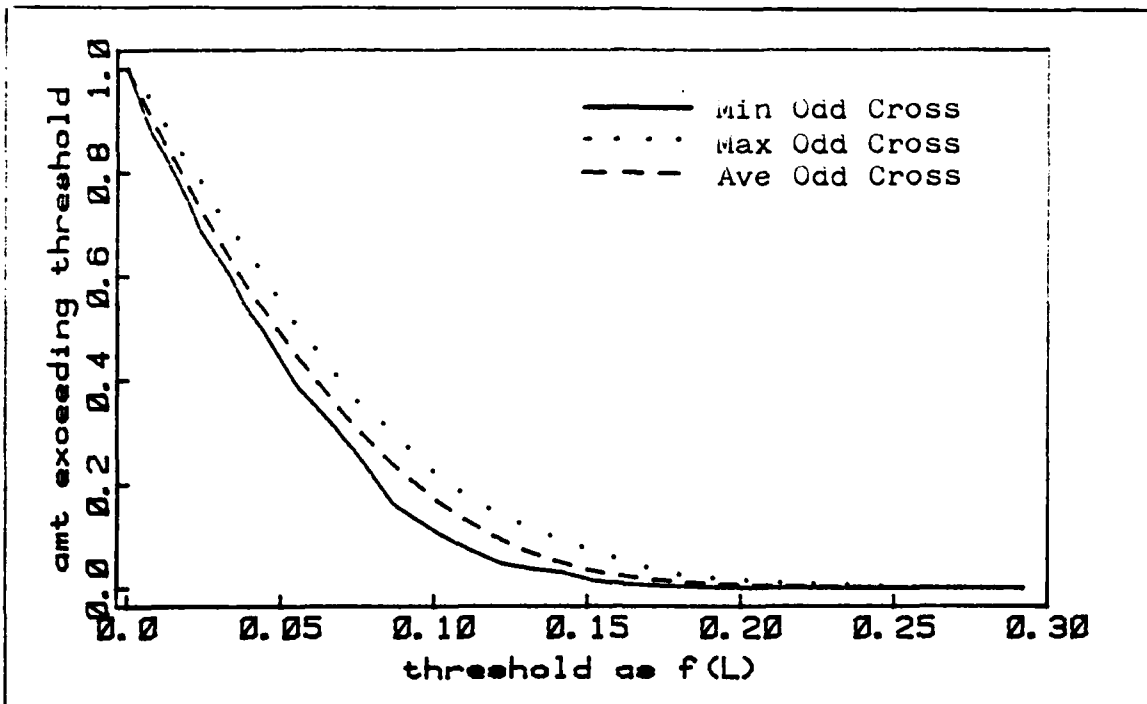


Fig. A-49. Threshold Plot of Minimum, Maximum and Average Odd Correlation Functions from Mass-Correlation Algorithm for Gold Codes 1-10, Length 127

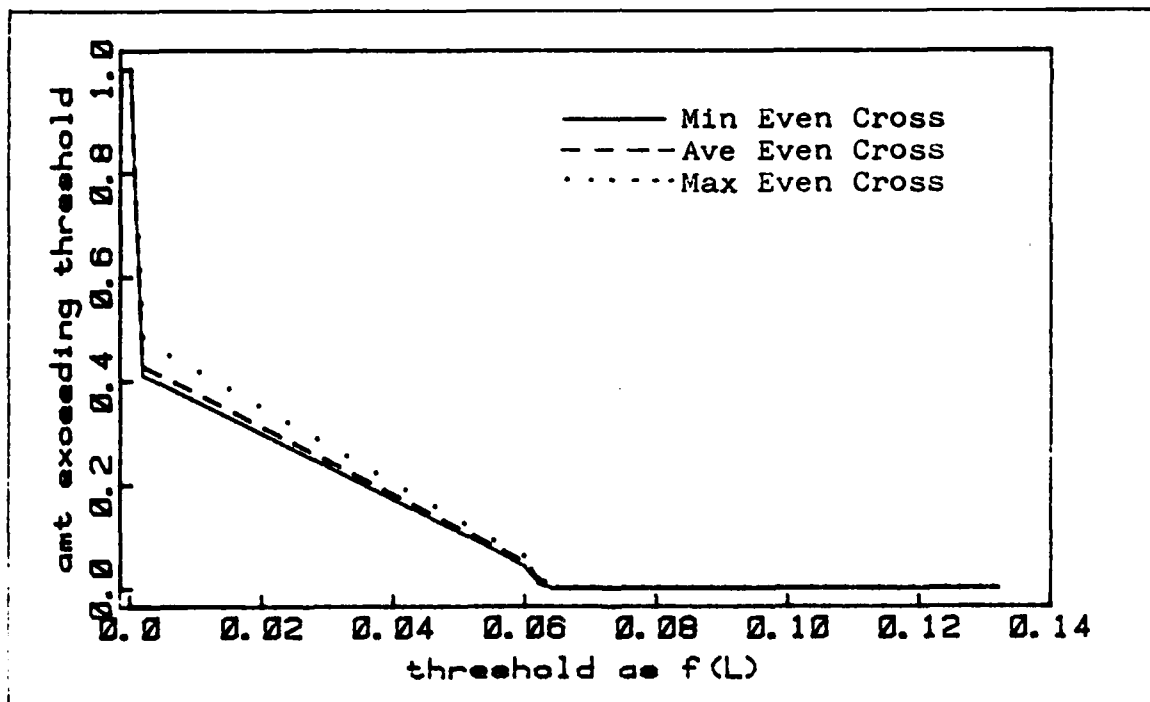


Fig. A-50. Threshold Plot of Minimum, Maximum and Average Even Correlation Functions from Mass-Correlation Algorithm for Gold Codes 1-8, Length 1023

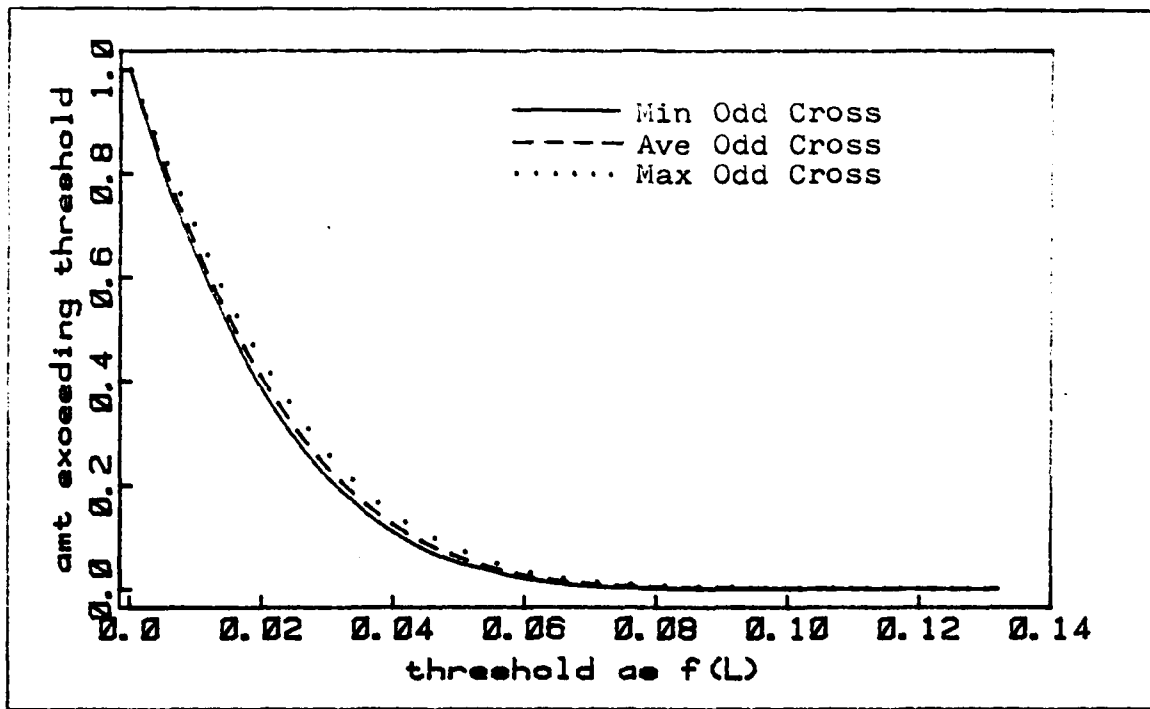


Fig. A-51. Threshold Plot of Minimum, Maximum and Average Odd Correlation Functions from Mass-Correlation Algorithm for Gold Codes 1-8, Length 1023

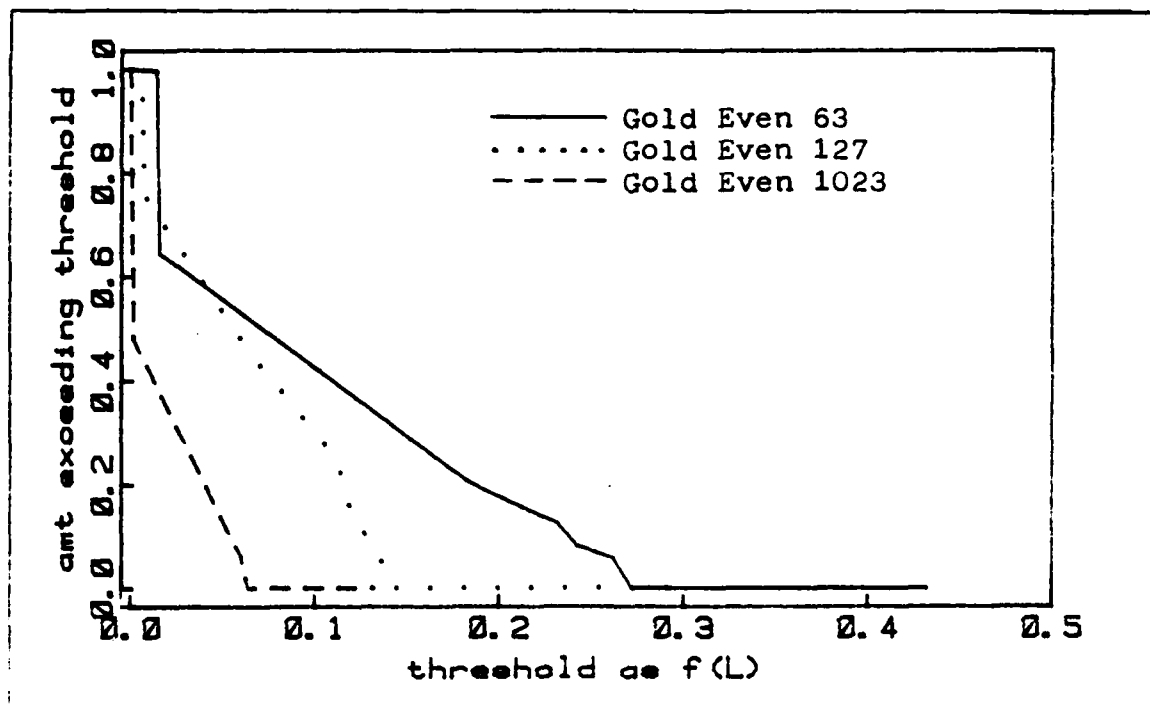


Fig. A-52. Threshold Plot of Maximum Even Correlation Functions from Mass-Correlation Algorithm for Gold Codes of Length 63, 127, 1023

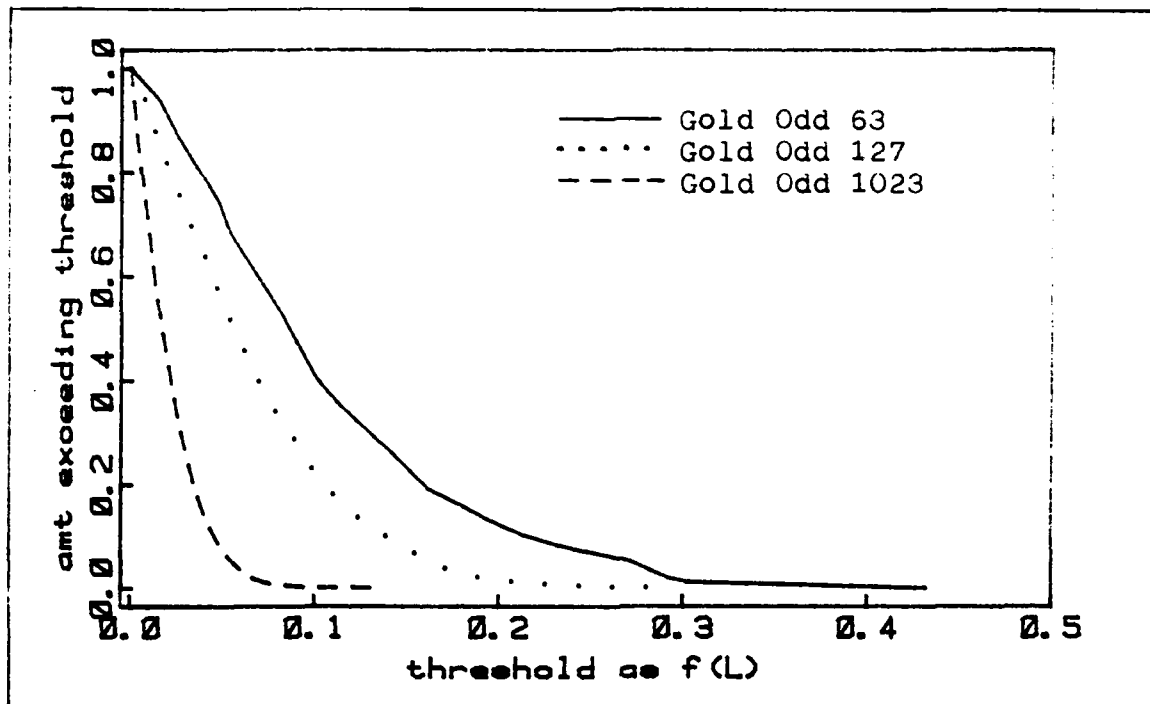


Fig. A-53. Threshold Plot of Maximum Odd Correlation Functions from Mass-Correlation Algorithm for Gold Codes of Length 63, 127, 1023

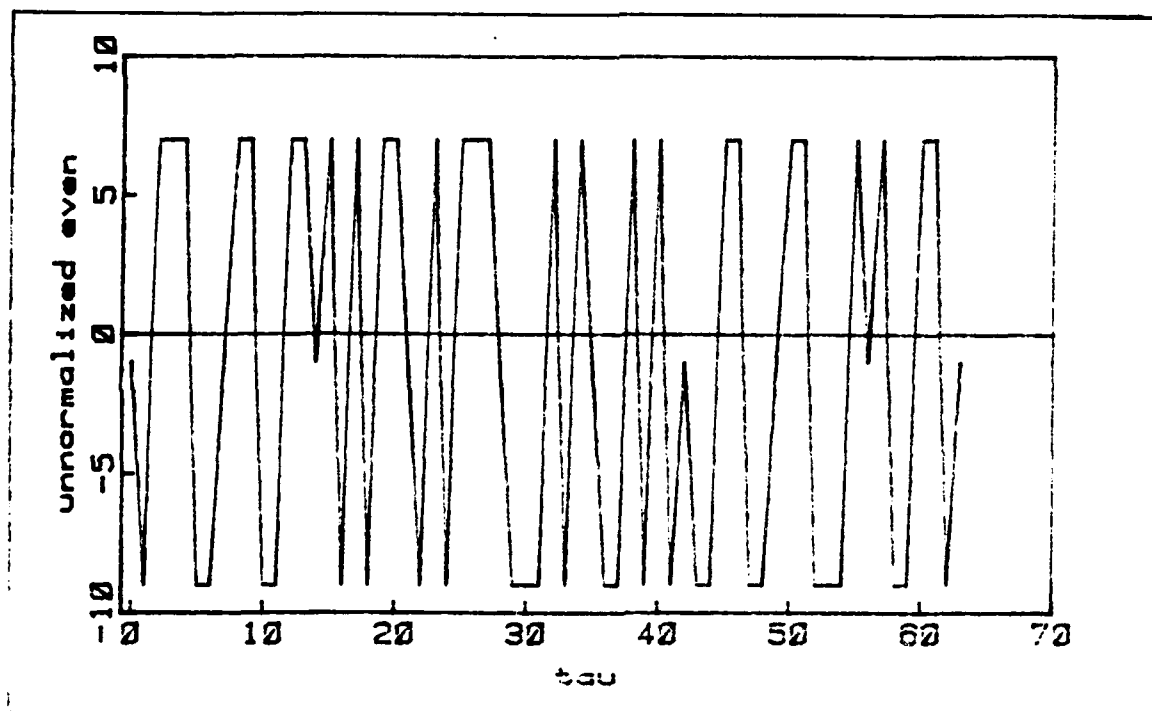


Fig. A-54. Typical Even Cross Correlation Function of Kasami Code of Length 63

AD-A159 309

CODE SEQUENCE PERFORMANCE ANALYSIS USING  
TECH-CORRELATION PARAMETERS IN (U) AIR FORCE INST OF  
TECH WRIGHT-PATTERSON AFB OH SCHOOL OF ENGI  
R C GONDER JUN 85 AFIT/GE/ENG/85J-1 F/G 17/2

2/2

UNCLASSIFIED

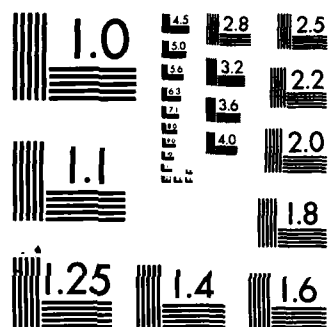
F/G 17/2

NL

END

File Name:

2116



MICROCOPY RESOLUTION TEST CHART  
NATIONAL BUREAU OF STANDARDS-1963-A

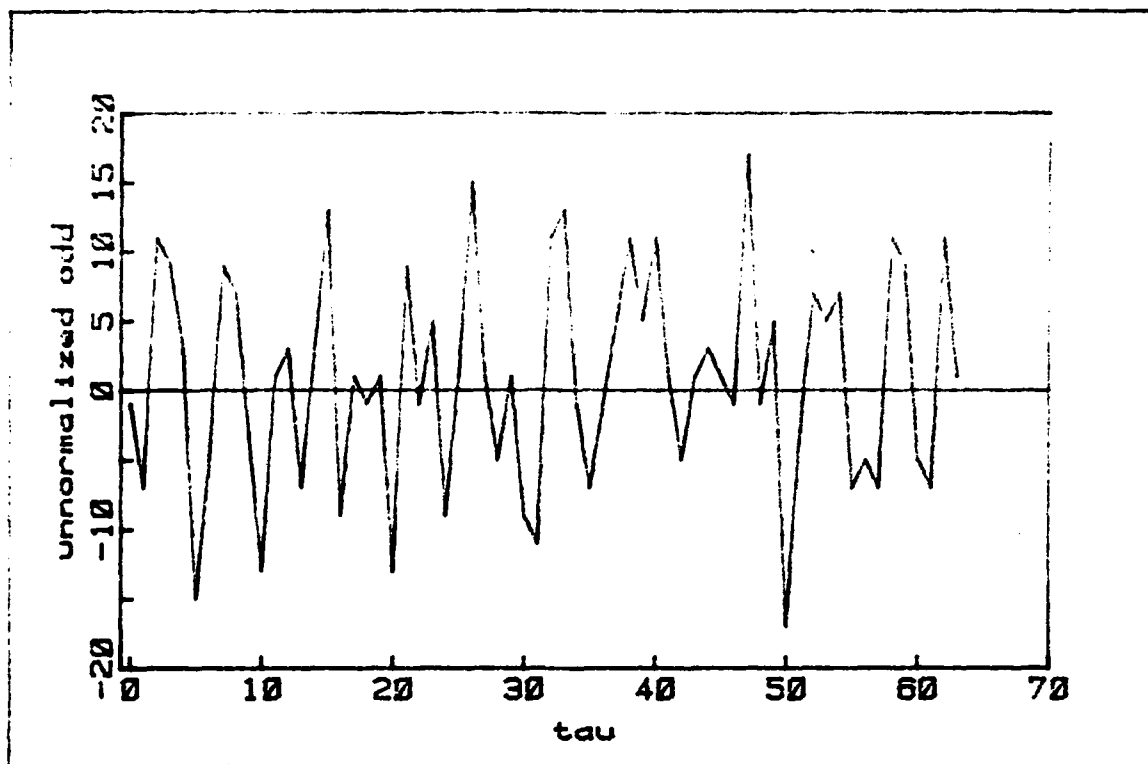


Fig. A-55. Typical Odd Cross Correlation Function of Kasami Code of Length 63

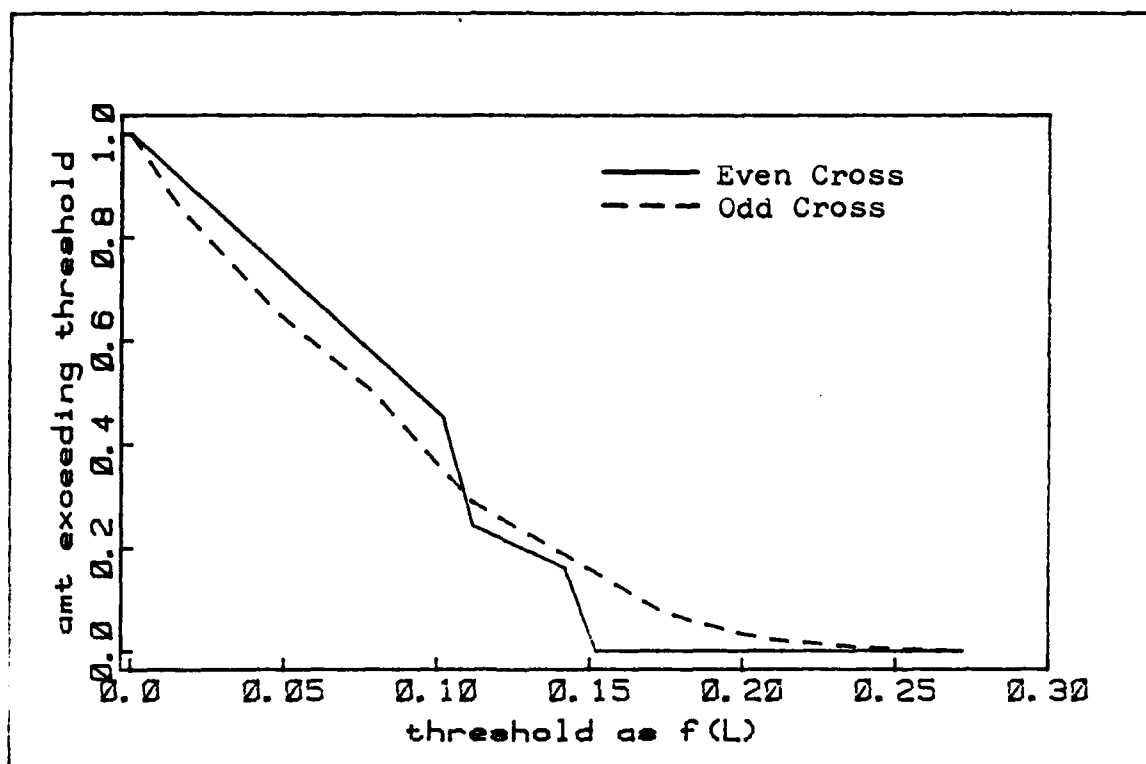


Fig. A-56. Typical Amount Exceeding Threshold vs Threshold of Even and Odd Correlation Functions Kasami Code of Length 63

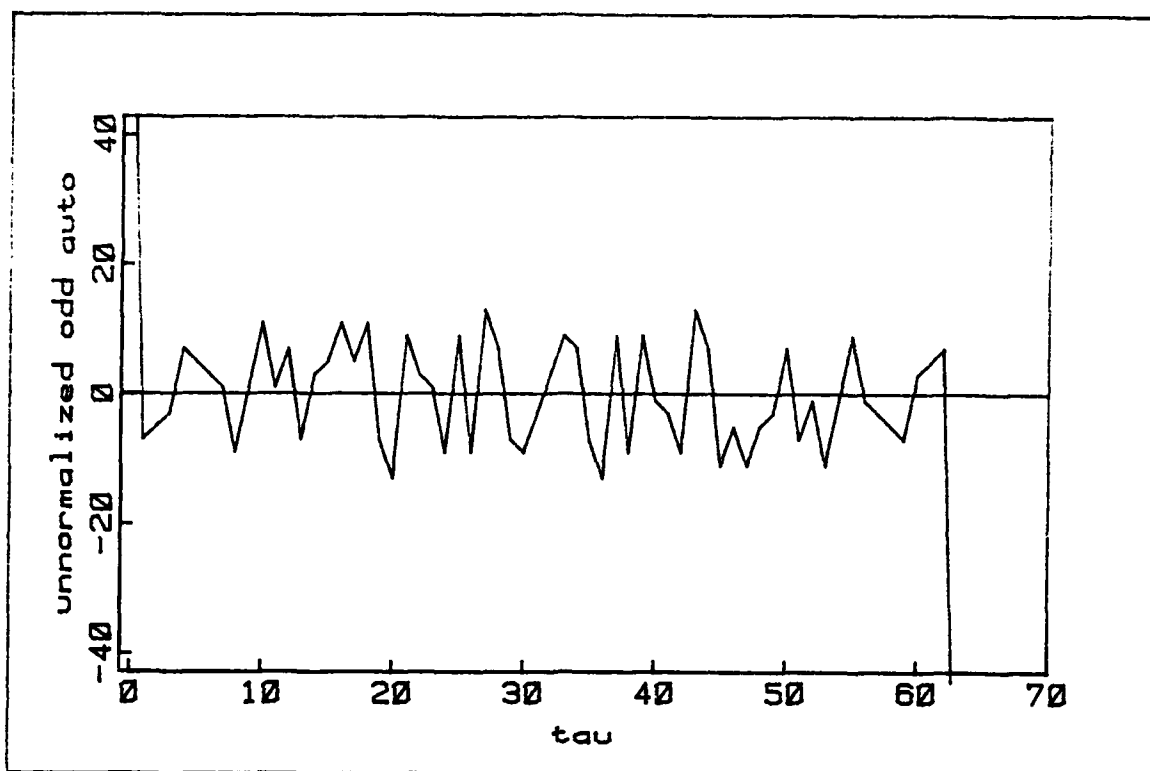


Fig. A-57. Optimal Odd Auto-Correlation Function of Kasami Codes 2 & 3, Length 63

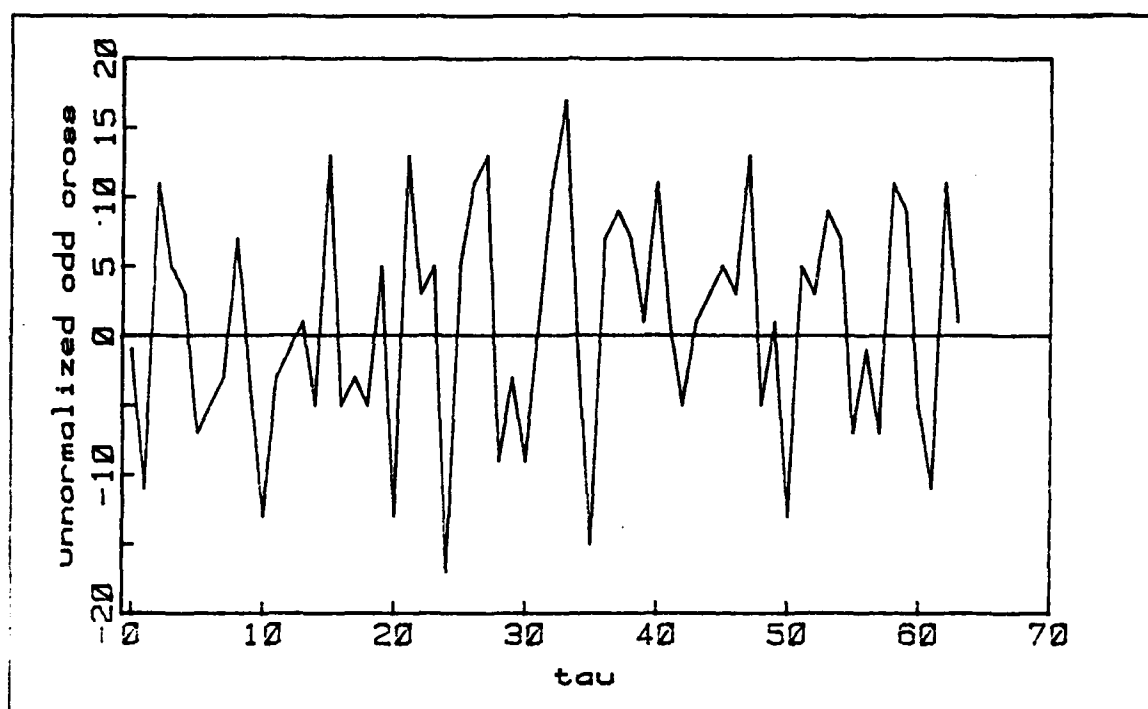


Fig. A-58. Optimal Odd Cross-Correlation Function of Kasami Codes 2 & 3, Length 63

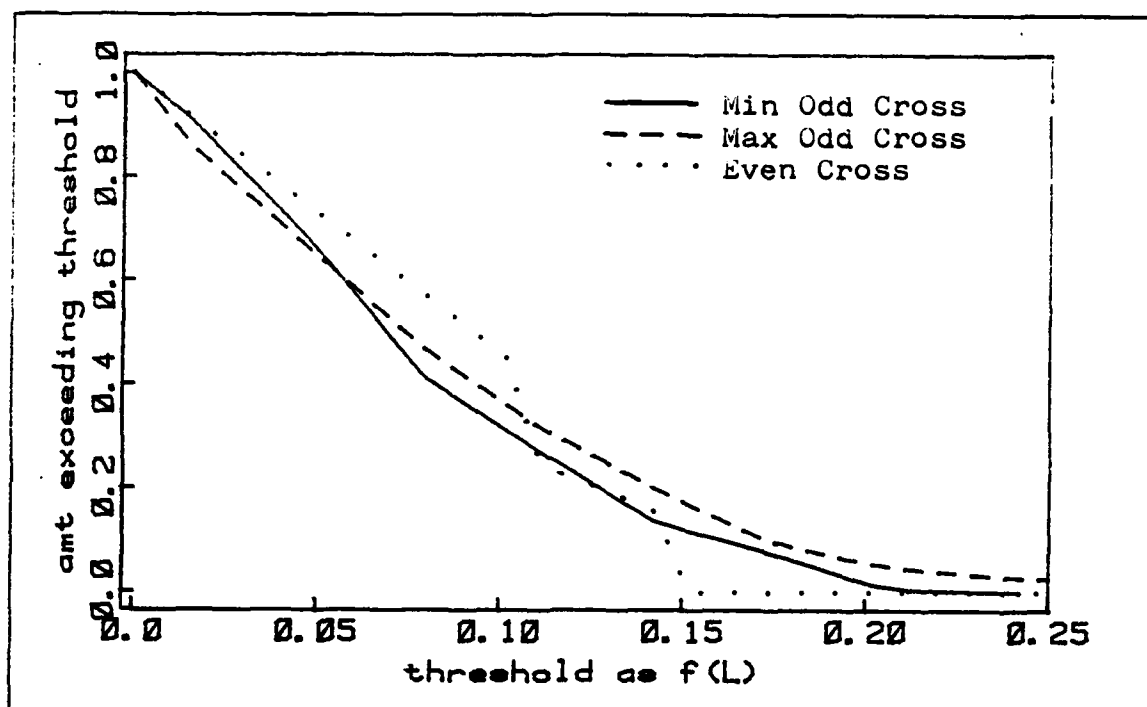


Fig. A-59. Threshold Plot of Minimum and Maximum Odd Auto-Correlation and Even Auto-Correlation of Kasami Codes 2 & 3, Length 63

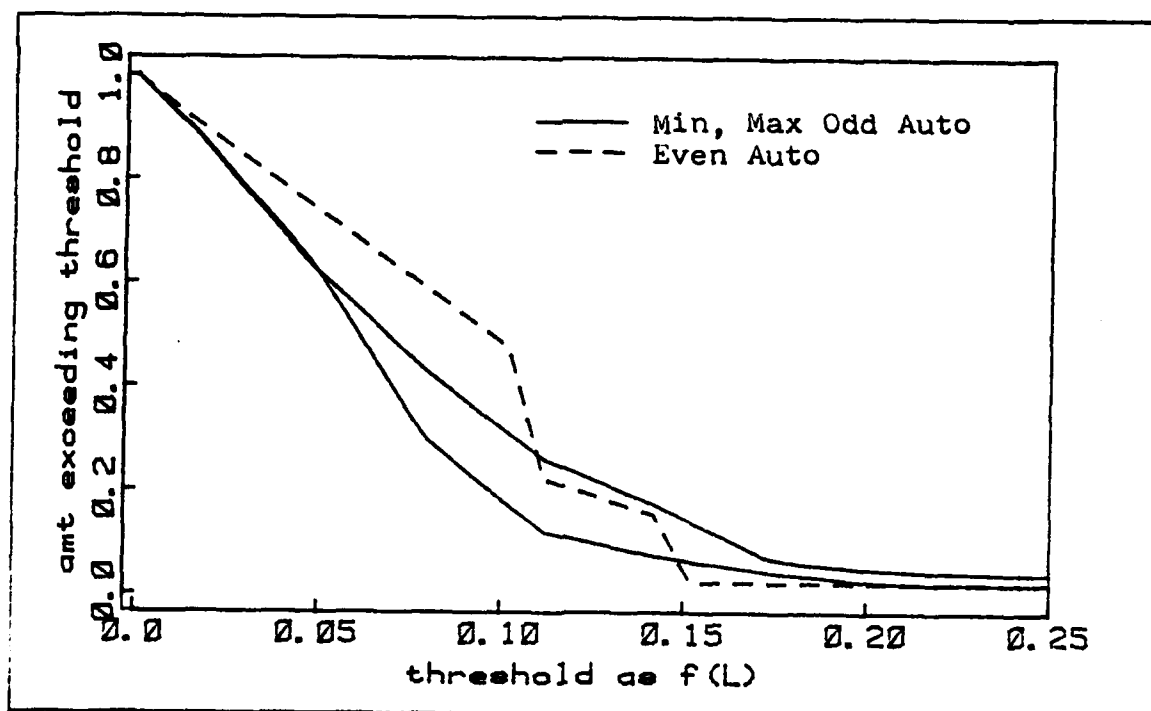


Fig. A-60. Threshold Plot of Minimum and Maximum Odd Cross-Correlation and Even Cross-Correlation of Kasami Codes 2 & 3, Length 63

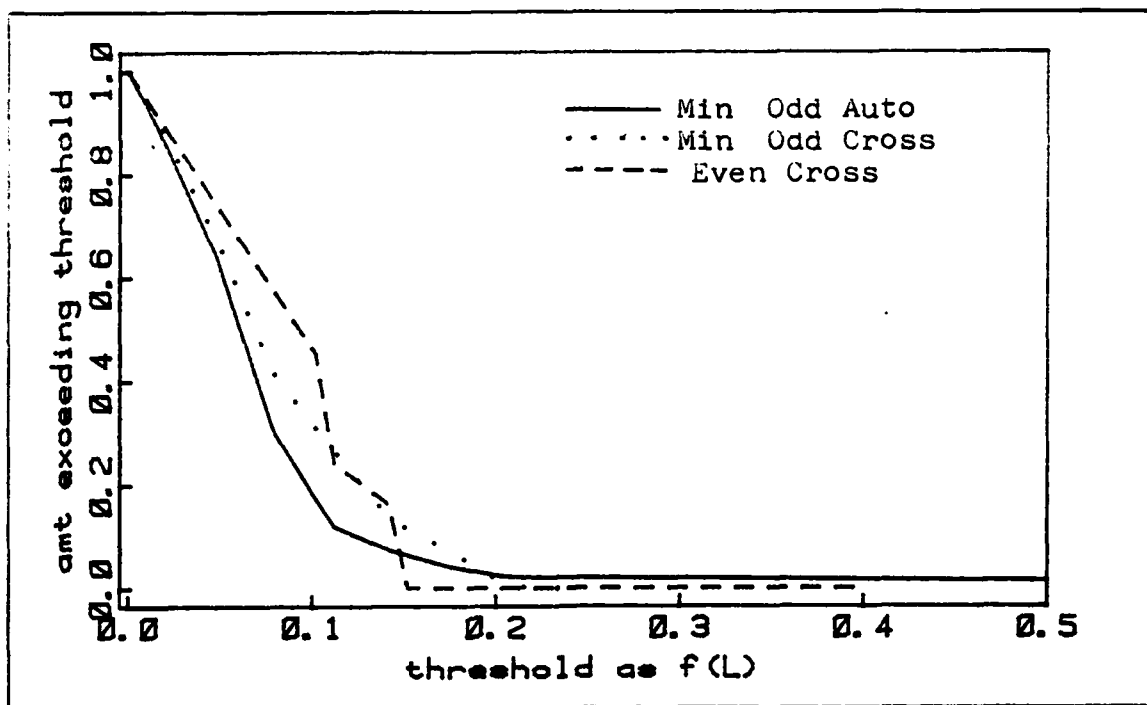


Fig. A-61. Threshold Plot of Minimum Odd Auto-Correlation, Minimum Odd Cross-Correlation, and Even Cross-Correlation of Kasami Codes 2 & 3, Length 63

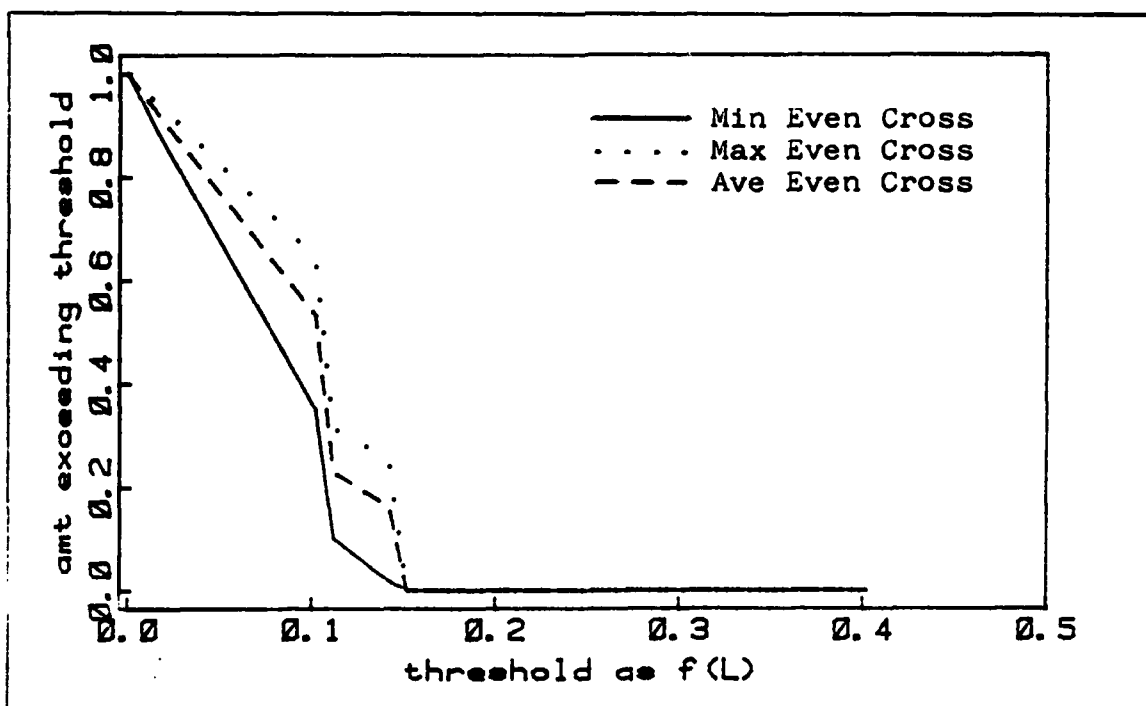


Fig. A-62. Threshold Plot of Minimum, Maximum and Average Even Correlation Functions from Mass-Correlation Algorithm for Kasami Codes 1-8, Length 63

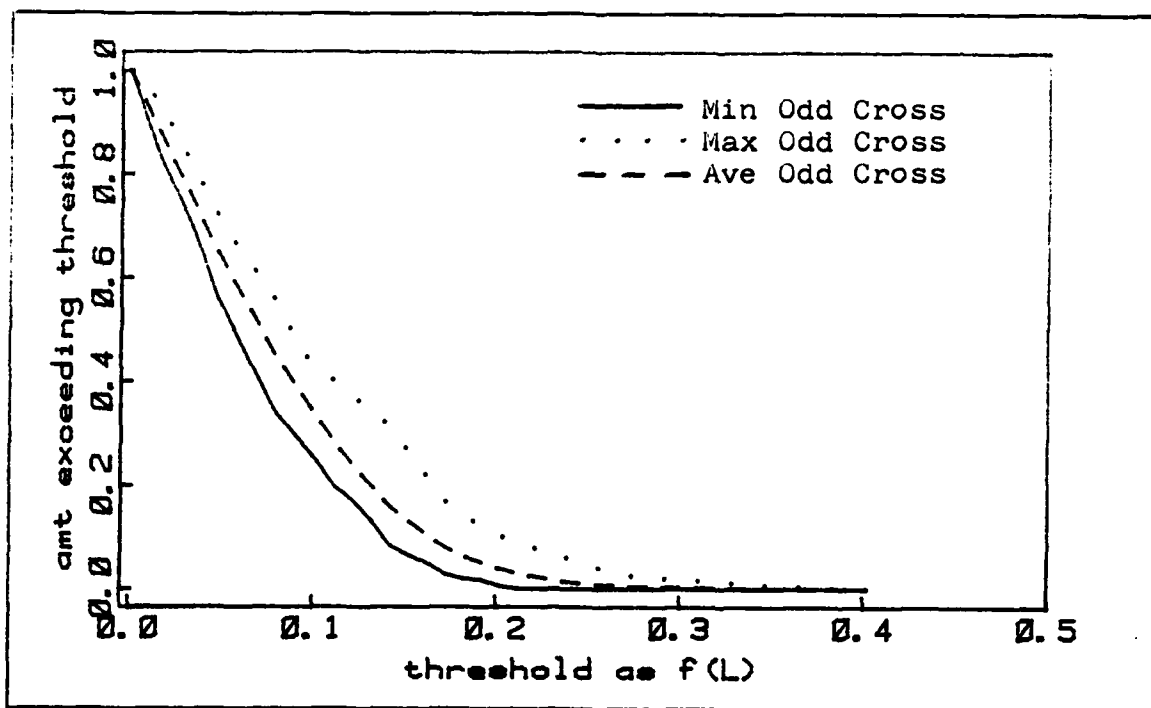


Fig. A-63. Threshold Plot of Minimum, Maximum and Average Odd Correlation Functions from Mass-Correlation Algorithm for Kasami Codes 1-8, Length 63

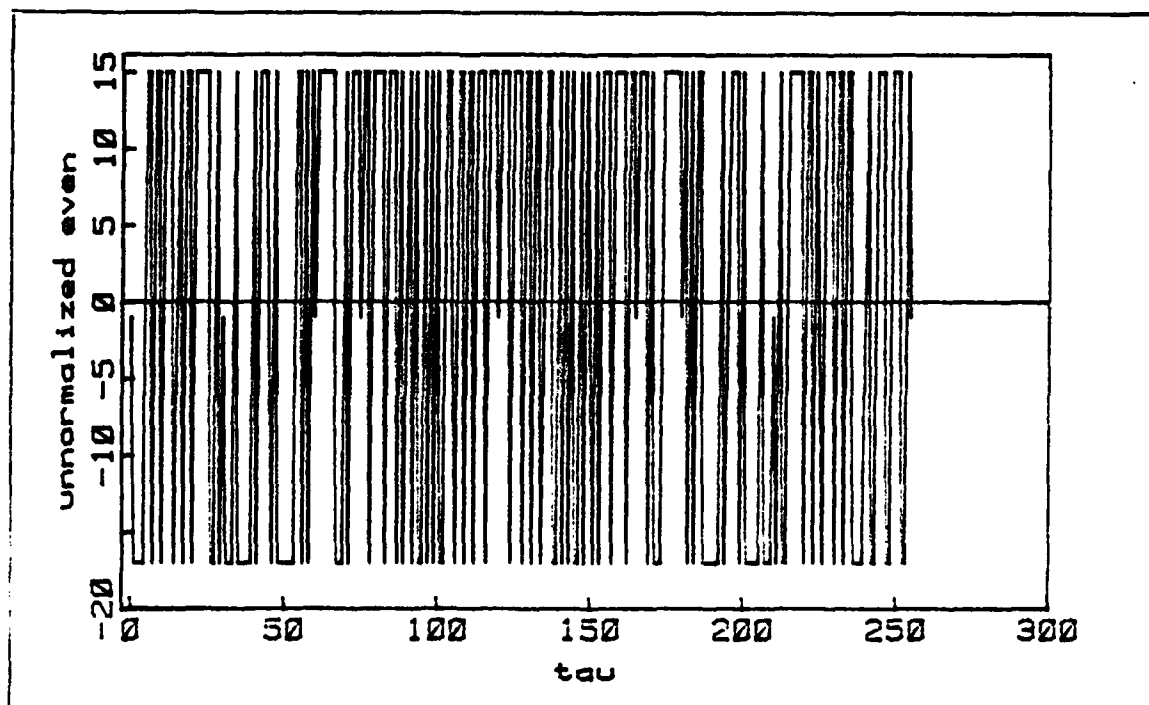


Fig. A-64. Typical Even Cross Correlation Function of Kasami Code of Length 255

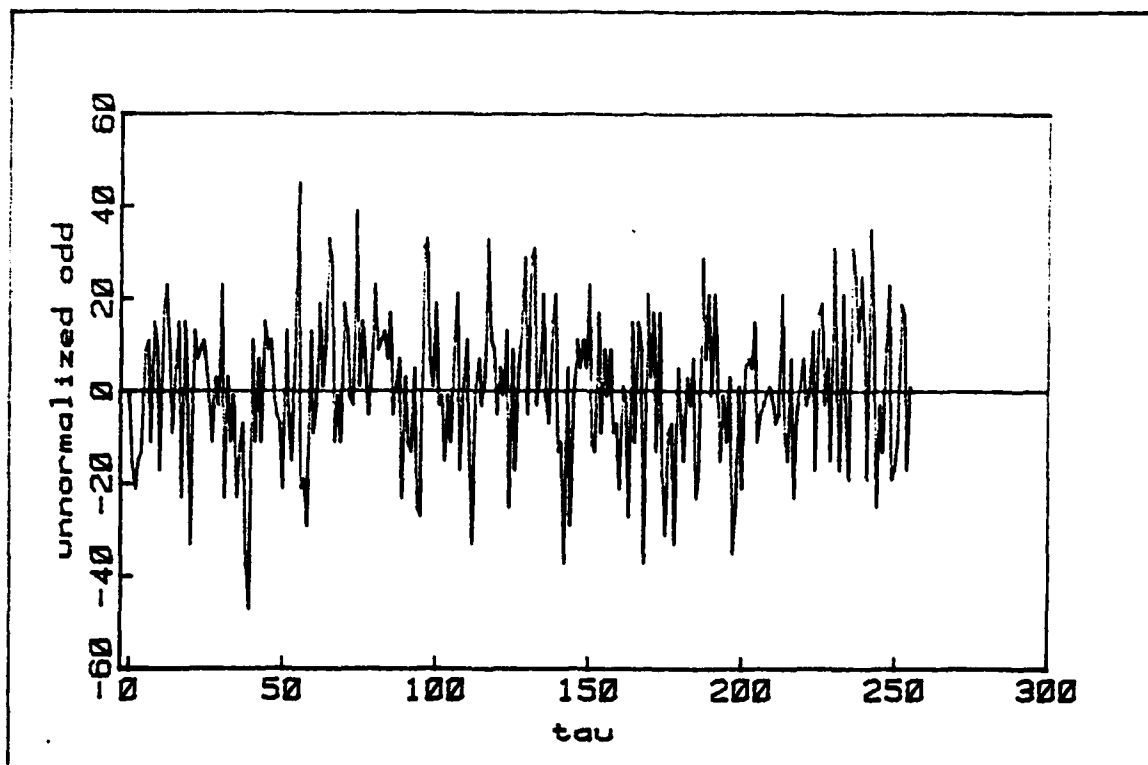


Fig. A-65. Typical Odd Cross Correlation Function of Kasami Code of Length 255

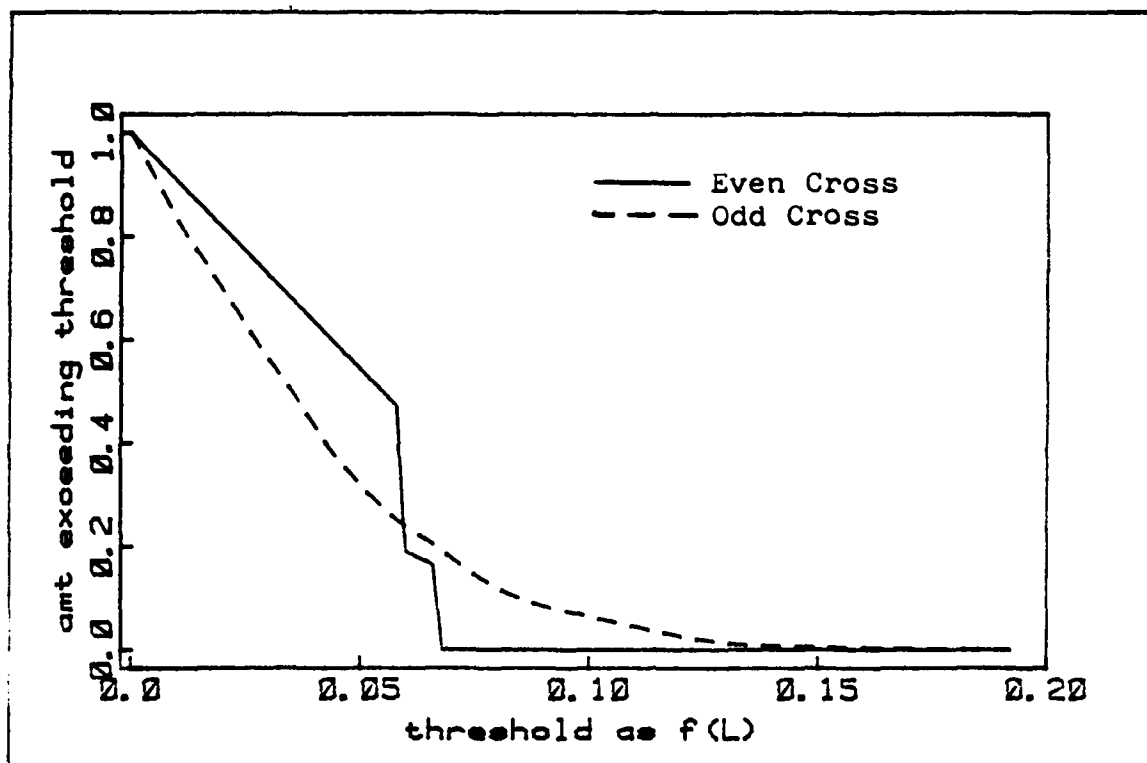


Fig. A-66. Typical Amount Exceeding Threshold vs Threshold of Even and Odd Correlation Functions Kasami Code of Length 255

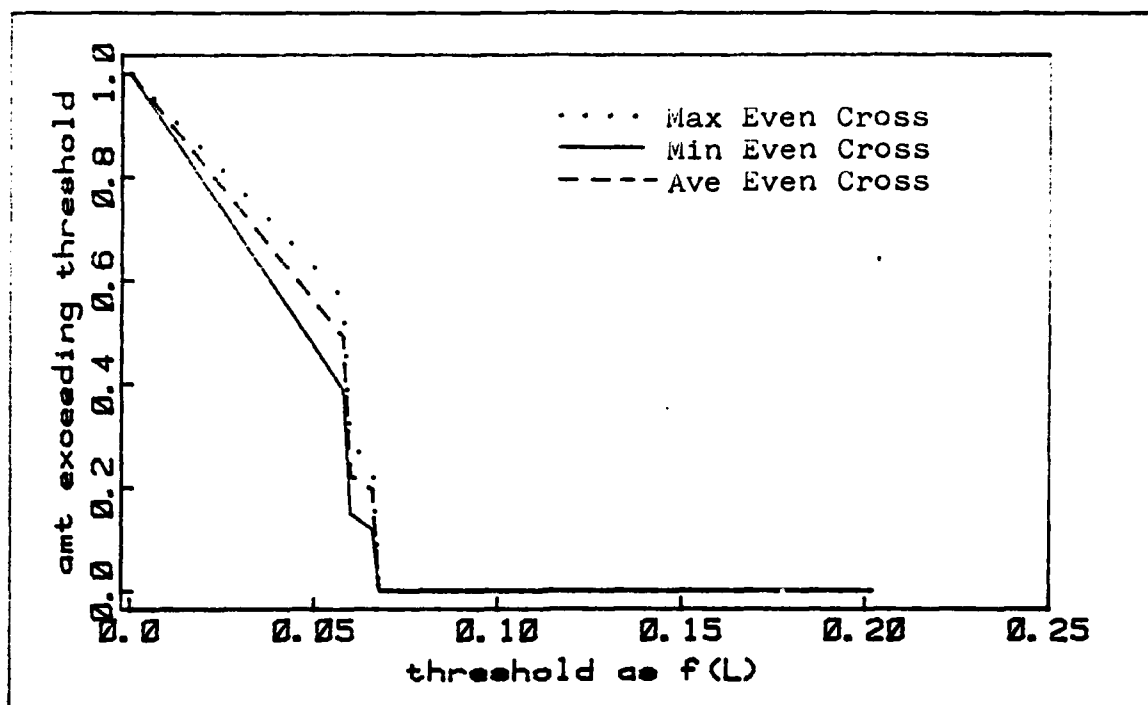


Fig. A-67. Threshold Plot of Minimum, Maximum and Average Even Correlation Functions from Mass-Correlation Algorithm for Kasami Codes 1-10, Length 255

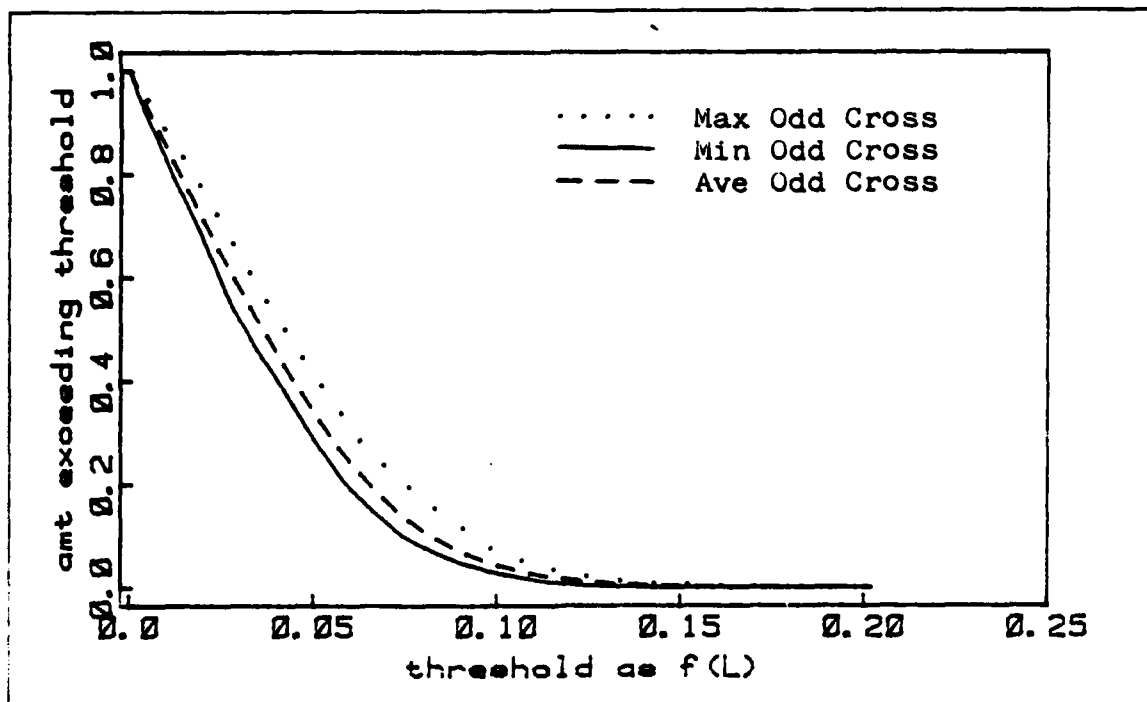


Fig. A-68. Threshold Plot of Minimum, Maximum and Average Odd Correlation Functions from Mass-Correlation Algorithm for Kasami Codes 1-10, Length 255

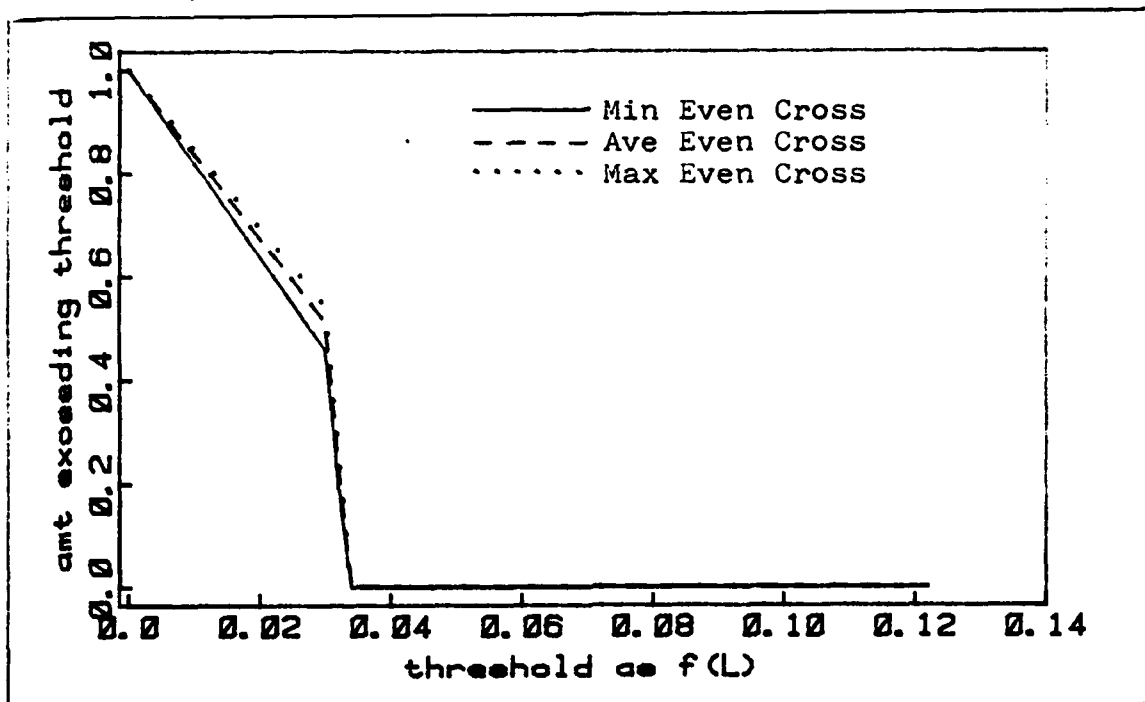


Fig. A-69. Threshold Plot of Minimum, Maximum and Average Even Correlation Functions from Mass-Correlation Algorithm for Kasami Codes 1-8, Length 1023

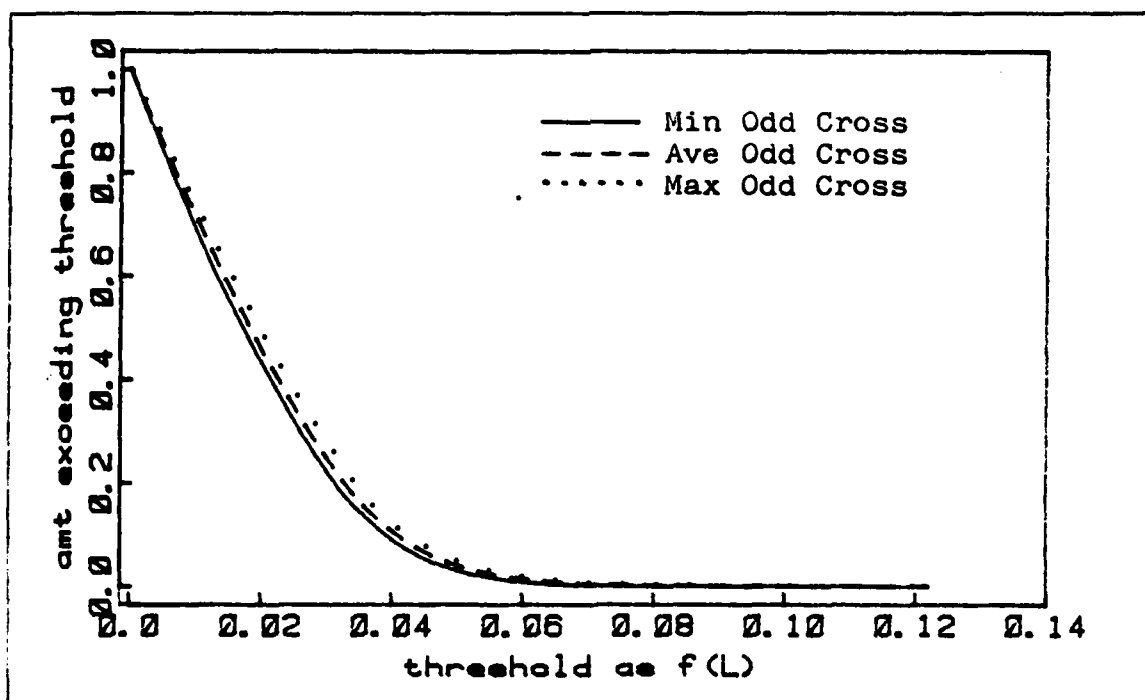


Fig. A-70. Threshold Plot of Minimum, Maximum and Average Odd Correlation Functions from Mass-Correlation Algorithm for Kasami Codes 1-8, Length 1023

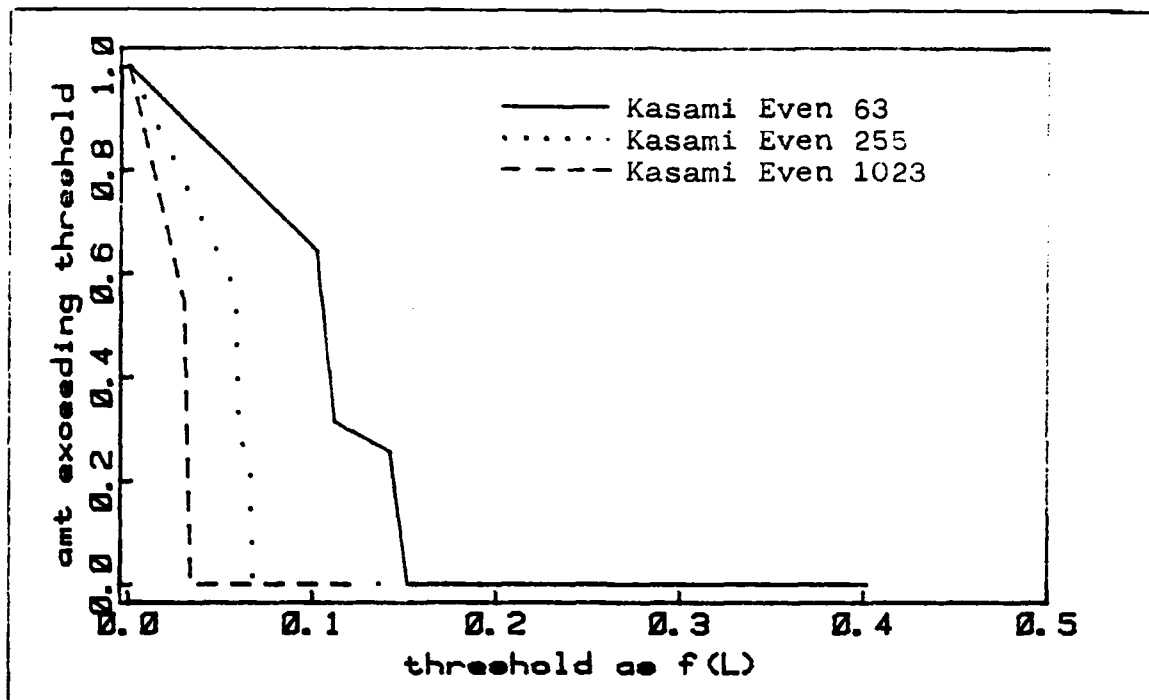


Fig. A-71. Threshold Plot of Maximum Even Correlation Functions from Mass-Correlation Algorithm for Kasami Codes of Length 63, 255, 1023

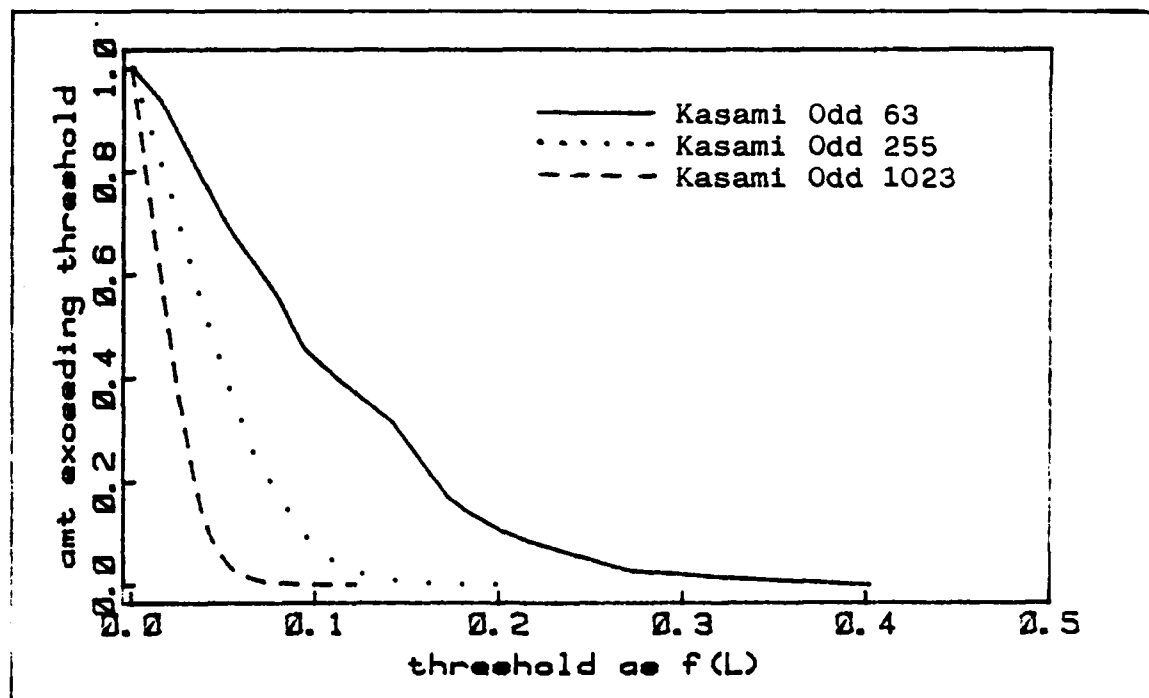


Fig. A-72. Threshold Plot of Maximum Odd Correlation Functions from Mass-Correlation Algorithm for Kasami Codes of Length 63, 255, 1023

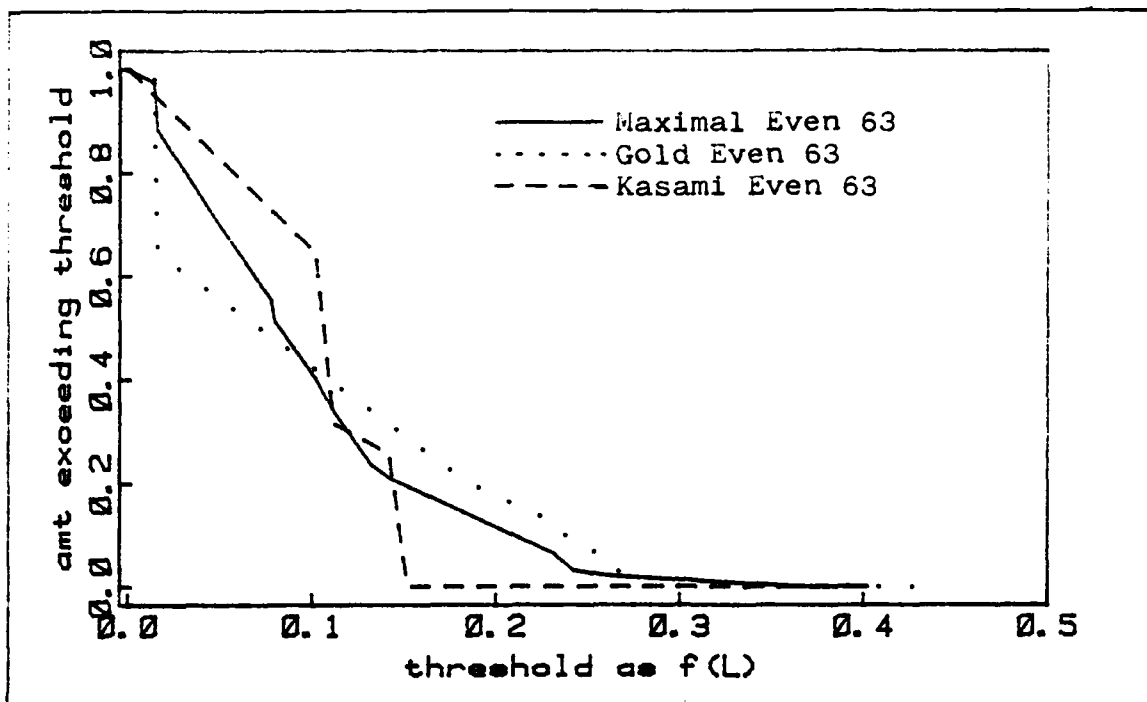


Fig. A-73. Threshold Plot of Maximum Even Correlation Functions from Mass-Correlation Algorithm for Maximal, Gold and Kasami Codes of Length 63

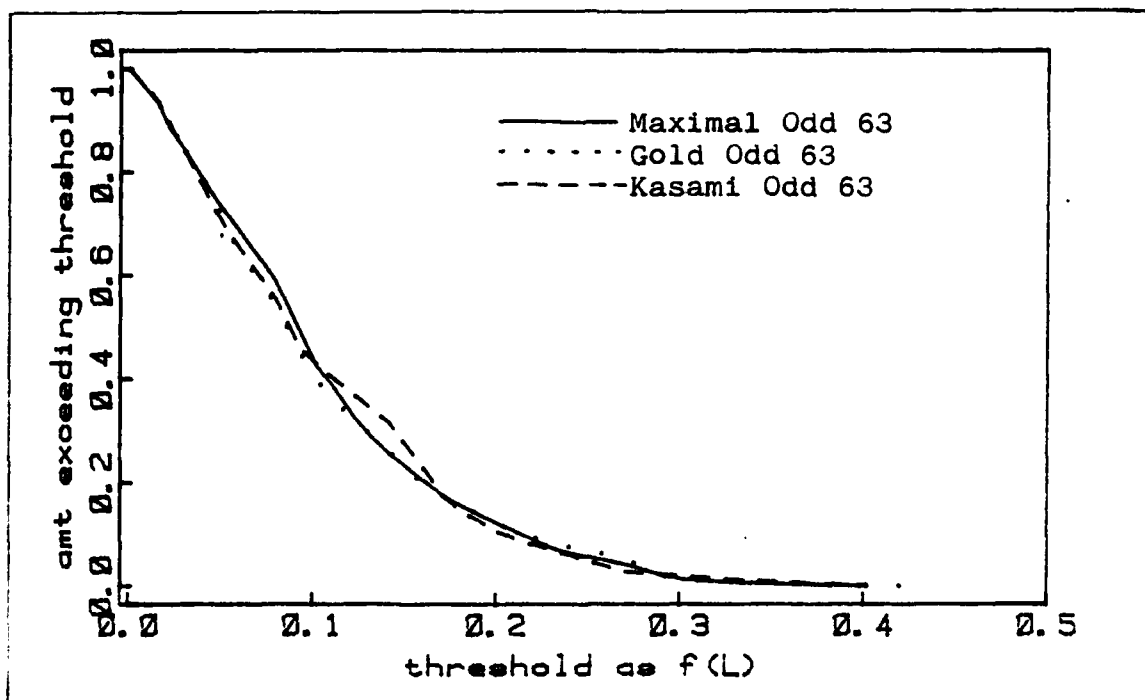


Fig. A-74. Threshold Plot of Maximum Odd Correlation Functions from Mass-Correlation Algorithm for Maximal, Gold and Kasami Codes of Length 63

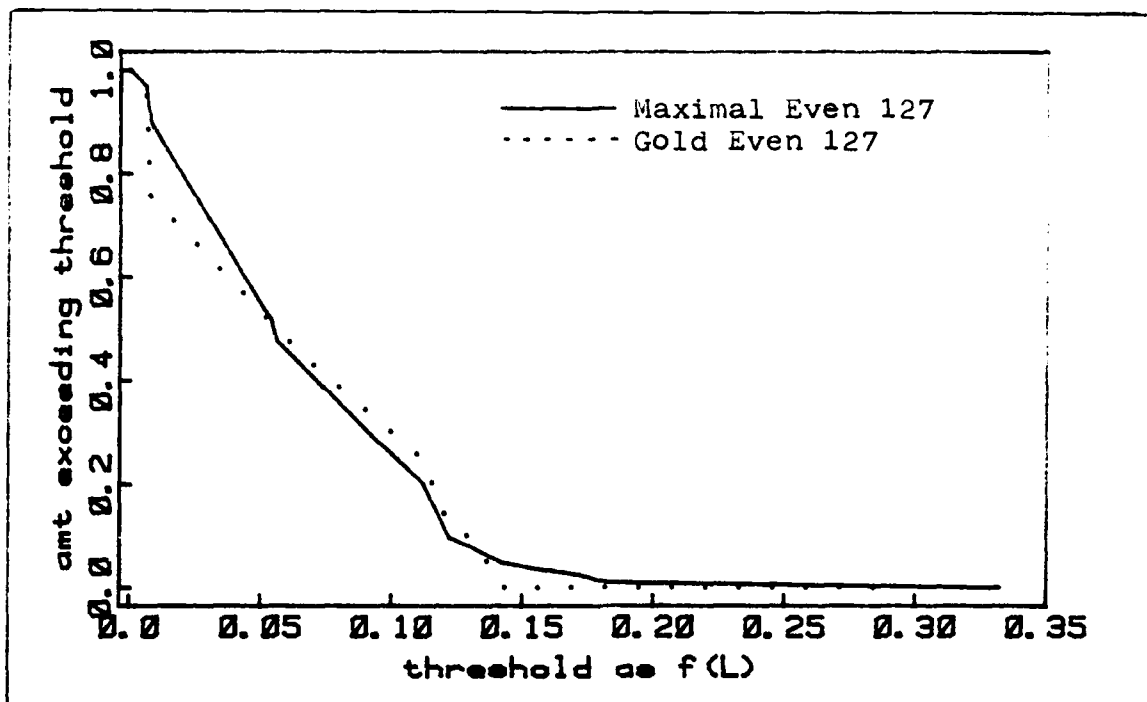


Fig. A-75. Threshold Plot of Maximum Even Correlation Functions from Mass-Correlation Algorithm for Maximal and Gold Codes of Length 127

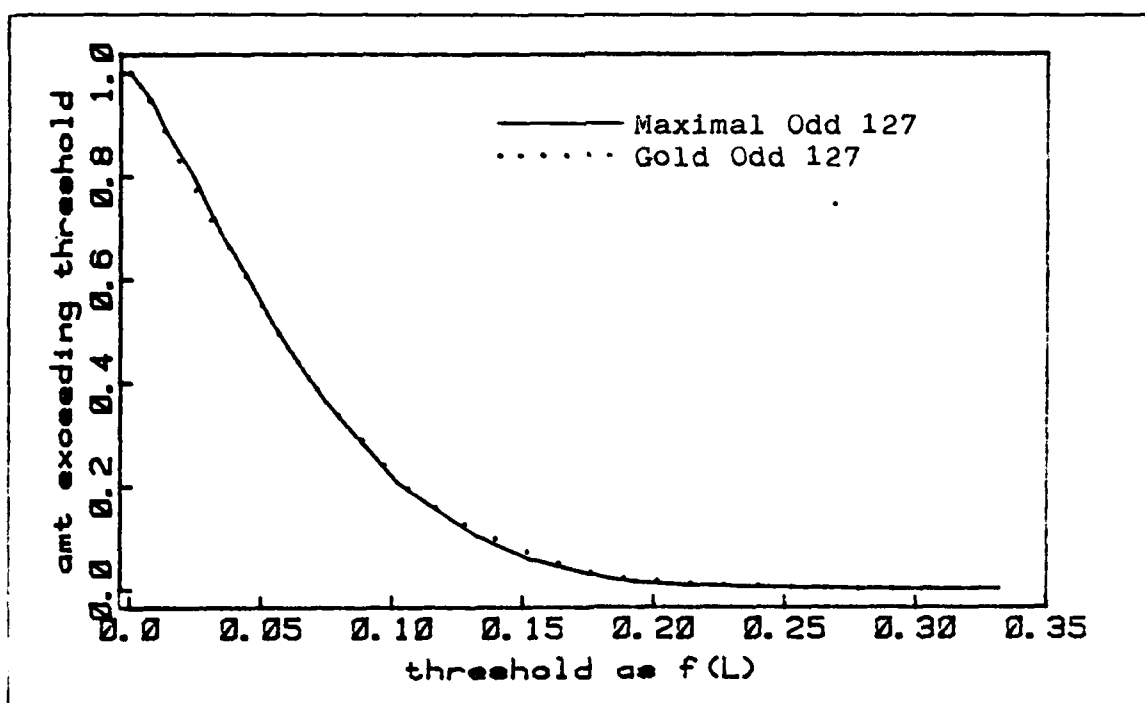


Fig. A-76. Threshold Plot of Maximum Odd Correlation Functions from Mass-Correlation Algorithm for Maximal and Gold Codes of Length 127

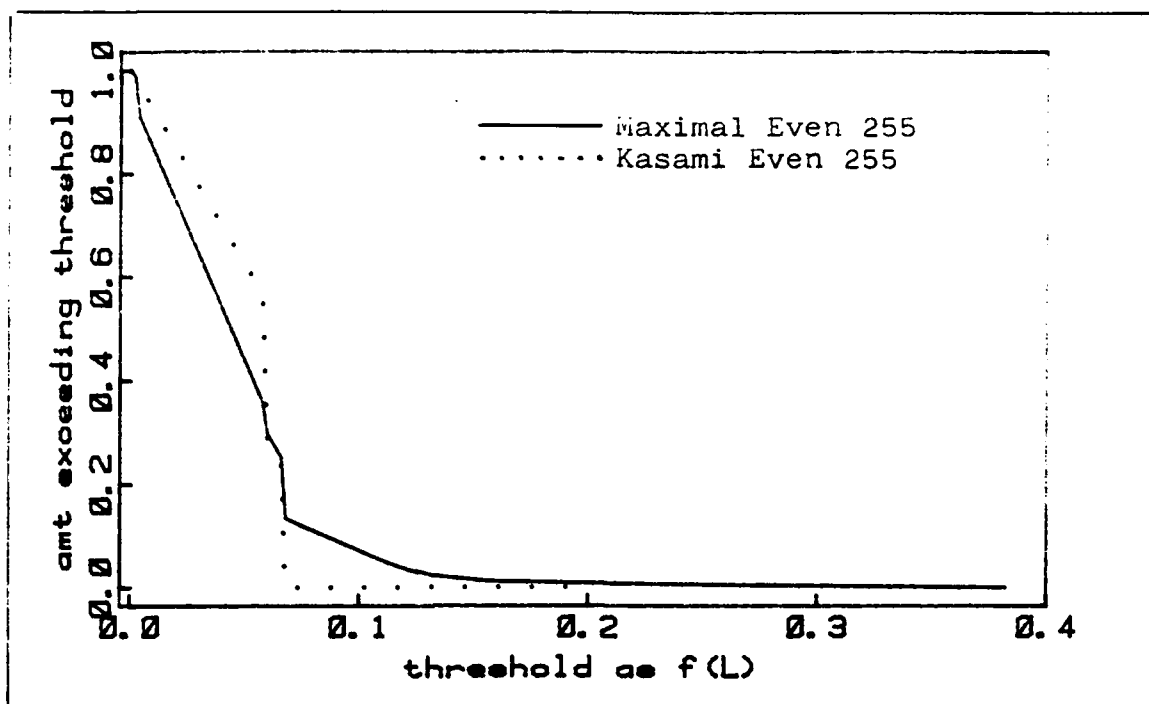


Fig. A-77. Threshold Plot of Maximum Even Correlation Functions from Mass-Correlation Algorithm for Maximal and Kasami Codes of Length 255

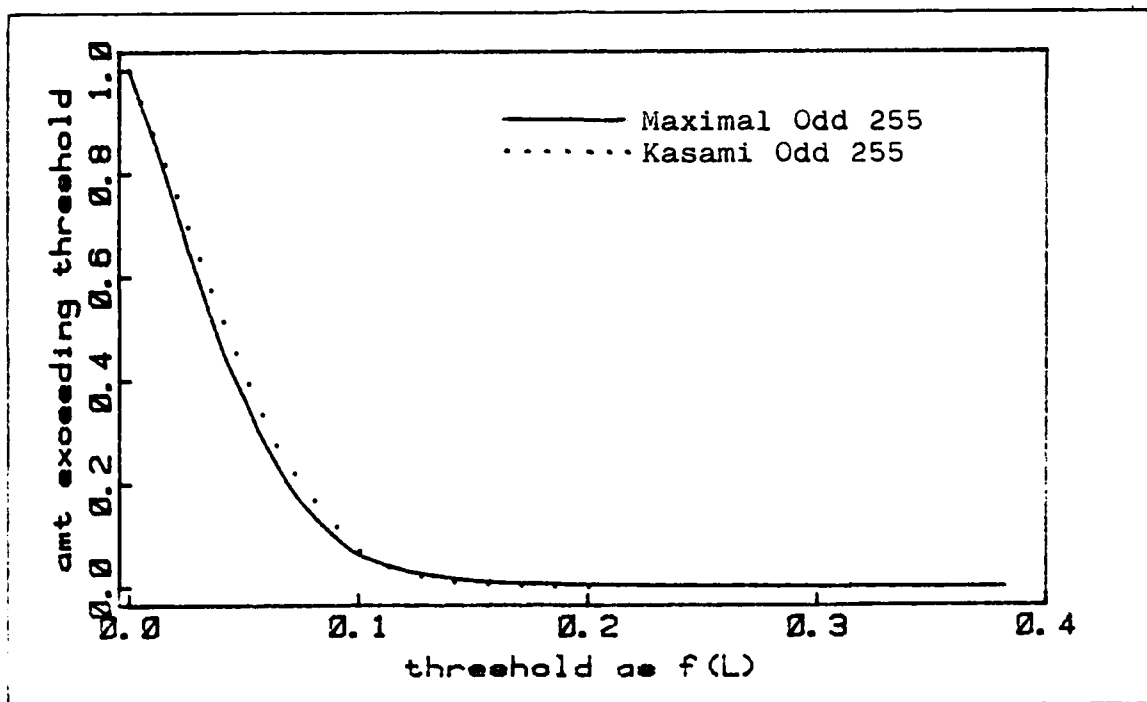


Fig. A-78. Threshold Plot of Maximum Odd Correlation Functions from Mass-Correlation Algorithm for Maximal and Kasami Codes of Length 255

```

begin
  M := 0;
  L := 1;
  while (M < N) and (L < 16383) do begin
    L := 2 * L;
    M := M + 1;
  end; (*while loop*)
  for I:= 1 to N do
    Reg1[I]:= '0';
  Reg1[N]:= '1';
  M:= 1;
  while M < L do begin
    ModSumTaps(Tap1,N,Reg1,CK);
    Enque(List1,Back1,CK);
    ShiftRegister(Reg1,N,CK);
    M:= M + 1;
  end; (*while loop*)
  for I:= 1 to N do
    Reg1[I]:= '0';
  Reg1[N]:= '1';
  M:= 1;
  while M < L do begin
    ModSumTaps(Tap2,N,Reg1,CK);
    Enque(List2,Back2,CK);
    ShiftRegister(Reg1,N,CK);
    M:= M + 1;
  end; (*while loop*)
  writeln('How many codes do you want generated? You may choose');
  writeln('from 1 to ',L+1,' Please input number');
  readln(Limit);
  if Limit >= 1 then begin
    Filename:= Name + '.001';
    writeln(Filename);
    WriteList(List1, Filename);
  end; (*if*)
  if Limit >= 2 then begin
    Filename:= Name + '.002';
    writeln(Filename);
    WriteList(List2,Filename);
  end;(*if*)
  Limit:= Limit - 2;
  X:= 1;
  while (X < L) and (X <= Limit) do begin
    str(X+2:3,st);
    if st[1] = ' ' then st[1]:= '0';
    if st[2] = ' ' then st[2]:= '0';
    Filename:= Name + '.' + st;
    writeln(Filename);
    ModSumList(List1,List2,Filename);
    ShiftList(List1);
    X:= X+1;
  end (*while loop*)
end;(*procedure GenGold*)

```

```

(*****
(*)
(*)           G e n G o l d
(*)
(*****
(*) DESIGNER:  Richard C. Gonder          MASTER THESIS WORK
(*) ADVISOR:  MAJ Ken Castor             MODULE NUMBER A.21
(*) DESIGN DATE: Dec 84                  LAST UPDATE: 4 Jan 85
(*)-----
(*) DESCRIPTION: This procedure constructs and manipulates two linked
(*) lists representing the outputs of two LFSR's. The shifting and
(*) modulo-2 addition of the lists are required to produce Gold codes.
(*) The user has the option of producing 1 to the maximum number of
(*) unique gold codes the registers will produce and stores the codes
(*) in text files with the same name and different extensions.
(*)-----
(*) CALLING ARGUMENTS: Name--string of characters representing the file
(*)                      the user desires to store output
(*)                      Tap1--Tap assignments for LFSR
(*)                      N--Length of LFSR used to generate code
(*)-----
(*) INTERNAL VARIABLES: I,M,X--counters used for looping
(*)                      L--Length of LFSR sequences
(*)                      Limit--Max number of Gold Codes to be produced
(*)                      CK--output of LFSR after addition of taps assigned
(*)                      Codedta--Text file variable for storage of output
(*)                      st--string of character representing filename ex-
(*)                          tensions for storage of output
(*)                      Reg1--LFSR array contents
(*)                      Back1,Back2--pointers to end of lists
(*)                      Filename--string of 12 characters w/extension
(*)-----
(*) CONSTANTS: None
(*)-----
(*) FILES USED: Codedta--storage of outputted code
(*)-----
(*) MODULES CALLED: ModSumTaps
(*)                  Enque
(*)                  ShiftRegister
(*)                  ModSumList
(*)                  ShiftList
(*)                  WriteList
(*)-----

```

```

    procedure GenGold(Name:Word; N:integer; Tap1,Tap2:RegStructure;
                      var List1, List2:dataptr);

```

```

    var Limit,I,M,L,X:integer;
        CK:Binary;
        Codedata: text;
        st: string[3];
        Reg1: RegStructure;
        Back1,Back2:dataptr;
        Filename:NameArray;

```

```

(*****)
(*)
(*)          D e c i m a t e          (*)
(*)
(*****)
(*) DESIGNER:  Richard C. Gonder          MASTER THESIS WORK          (*)
(*) ADVISOR:  MAJ Ken Castor             MODULE NUMBER A.311          (*)
(*) DESIGN DATE: Feb 85                  LAST UPDATE: 14 Mar 85          (*)
(*)-----(*)
(*) DESCRIPTION: This procedure decimates a linked list representing a (*)
(*) maximal sequence of length Length, at decimates by a factor of (*)
(*) N+2 to produce a sequence of length L. It then uses the second (*)
(*) sequence to produce a third sequence of period L, length Length. (*)
(*)-----(*)
(*) CALLING ARGUMENTS: List1,List2,List3--pointers to beginning of lists (*)
(*) Back1,Back2,Back3--pointers to ends of lists (*)
(*) L--Length of sequence two (*)
(*) Length--length of sequence three (*)
(*)-----(*)
(*) INTERNAL VARIABLES: P,ptr--pointer used to traverse the list (*)
(*) Count,K--counters used to do process (*)
(*)-----(*)
(*) CONSTANTS: None (*)
(*)-----(*)
(*) FILES USED: None (*)
(*)-----(*)
(*) MODULES CALLED: Enqueue (*)
(*)-----(*)

```

```

procedure Decimate(var List1,Back1,List2,Back2,List3,Back3: dataptr;
                  L,Length:integer);

```

```

var P,Ptr:dataptr;
    Count,K:integer;

```

```

begin
  P:= List1;
  Enqueue(List2,Back2,P^.info);
  Count:=1;
  repeat
    for K:= 1 to (L+2) do
      P:= P^.next;
      Enqueue(List2,Back2,P^.info);
      Count:= Count + 1;
  until Count = L+1;
  Count:= 0;
  while(Count < Length) do begin
    P:= List2;
    repeat
      Enqueue(List3,Back3,P^.info);
      Count:= Count + 1;
      P:= P^.next;
    until (P^.next = nil);
  end(*while Count*)
end; (*procedure Decimate*)

```

```

(*****)
(*)
(*)           G e t R i d O f
(*)
(*****)
(*) DESIGNER:  Richard C. Gonder           MASTER THESIS WORK
(*) ADVISOR:  MAJ Ken Castor             MODULE NUMBER A.03
(*) DESIGN DATE: Jan 85                 LAST UPDATE: 14 Jan 85
(*)-----(*)
(*) DESCRIPTION: This procedure releases a linked list from memory allo-
(*) cation
(*)-----(*)
(*) CALLING ARGUMENTS: List--pointer to beginning of linked list
(*)-----(*)
(*) INTERNAL VARIABLES: P,ptr--pointer used to traverse the list
(*)-----(*)
(*) CONSTANTS: None
(*)-----(*)
(*) FILES USED: None
(*)-----(*)
(*) MODULES CALLED: None
(*)-----*)

```

```

procedure GetRidOf(var List:dataptr);

```

```

    var P,ptr:dataptr;

```

```

begin
    P:= List;
    ptr:=P;
    while P^.next <> nil do begin
        ptr:= P^.next;
        dispose(P);
        P:=ptr;
    end (*while loop*);
    dispose(P);
    List:= nil;
end;(*procedure GetRidOf*)

```

```

(*****
(*)
(*)           W r i t e L i s t           (*)
(*)
(*****)
(*) DESIGNER:  Richard C. Gonder           MASTER THESIS WORK           (*)
(*) ADVISOR:  MAJ Ken Castor             MODULE NUMBER A.07           (*)
(*) DESIGN DATE: Dec 84                 LAST UPDATE: 10 Dec 84           (*)
(*)
(*) DESCRIPTION: This procedure traverses a linked list of records and (*)
(*)           outputs the information in the records in file Codedta. (*)
(*)
(*) CALLING ARGUMENTS: List--pointer to beginning of linked list (*)
(*)           Filename--string of char identifying file to be (*)
(*)           written to (*)
(*)
(*) INTERNAL VARIABLES: P--pointer used to traverse the list (*)
(*)
(*) CONSTANTS: None (*)
(*)
(*) FILES USED: Writes to Codedta (*)
(*)
(*) MODULES CALLED: None (*)
(*)

```

```

  procedure WriteList(List:dataptr;Filename:NameArray);

```

```

    var P:dataptr;
        Codedta:text;

    begin
        assign(Codedta,Filename);
        rewrite(Codedta);
        P:=List;
        while P <> nil do begin
            write(Codedta,P^.info);
            P:= P^.next;
        end; (*while loop*)
        close(Codedta);
    end;

```

```

(*****
(*)
(*)           M o d S u m L i s t
(*)
(*****
(*) DESIGNER:  Richard C. Gonder           MASTER THESIS WORK
(*) ADVISOR:  MAJ Ken Castor             MODULE NUMBER A.06
(*) DESIGN DATE: Dec 84                 LAST UPDATE: 4 Jan 1985
(*)
(*) DESCRIPTION: This procedure traverses two Linked Lists of binary data
(*) and performs a bit by bit modulo-2 sumation of the lists and writes
(*) the result into the text file, Codedta.
(*)
(*) CALLING ARGUMENTS: List1,List2—pointer values indicating the first
(*)                      record of each linked list or records
(*)                      Filename—string of characters indicating name of
(*)                      file output should be stored in
(*)
(*) INTERNAL VARIABLES: P1,P2—pointers used for traversing the lists
(*)                      bit—used for result of modulo-2 addition of two
(*)                      records
(*)
(*) CONSTANTS: None
(*)
(*) FILES USED: writes to Codedta
(*)
(*) MODULES CALLED: None
(*)

```

```

procedure ModSumList(List1:dataptr;List2:dataptr;Filename:NameArray);

```

```

    var P1,P2 : dataptr;
        Bit:binary;
        Codedta: text;

    begin
        assign(Codedta,Filename);
        rewrite(Codedta);
        P1:= List1;
        P2:= List2;
        while P1^.next <> nil do begin
            if P1^.info = P2^.info then
                Bit:= '0'
            else
                Bit:= '1';
            write(Codedta,Bit);
            P1:= P1^.next;
            P2:= P2^.next;
        end(*while loop*);
        if P1^.info = P2^.info then
            Bit:= '0'
        else
            Bit:= '1';
        write(Codedta,Bit);
        close(Codedta);
    end (*procedure ModSumList*);

```

```

(*****
(*)
(*)           S h i f t L i s t           (*)
(*)
(*****
(*) DESIGNER:  Richard C. Gonder           MASTER THESIS WORK (*)
(*) ADVISOR:  MAJ Ken Castor             MODULE NUMBER A.08 (*)
(*) DESIGN DATE: Dec 84                 LAST UPDATE: 12 Dec 84 (*)
(*)-----(*)
(*) DESCRIPTION: This procedure takes a linked list of records and places (*)
(*) the first record at the end of the list. (*)
(*)-----(*)
(*) CALLING ARGUMENTS: List--pointer to the first record in the linked (*)
(*) list (*)
(*)-----(*)
(*) INTERNAL VARIABLES: First,P--pointers used for traversing the list (*)
(*) and assigning a new beginning of the list (*)
(*)-----(*)
(*) CONSTANTS: None (*)
(*)-----(*)
(*) FILES USED: None (*)
(*)-----(*)
(*) MODULES CALLED: None (*)
(*)-----*)

```

```

procedure ShiftList(var List:dataptr);

```

```

    var First,P:dataptr;

```

```

begin

```

```

    First:= List;

```

```

    P:= List;

```

```

    while P^.next <> nil do

```

```

        P:= P^.next;

```

```

    First:= First^.next;

```

```

    P^.next:= List;

```

```

    List^.next:= nil;

```

```

    List:= First;

```

```

end; (*procedure ShiftList*)

```

```

(*****)
(*)
(*)           E n q u e
(*)
(*****)
(*) DESIGNER:  Richard C. Gonder           MASTER THESIS WORK
(*) ADVISOR:  MAJ Ken Castor             MODULE NUMBER A.09
(*) DESIGN DATE: Dec 84                   LAST UPDATE: 12 Dec 1984
(*)
(*) DESCRIPTION: This procedure takes an inputted value (Bit) and places
(*) it in a record, and then places the record at the bottom of the list
(*) identified by the calling argument List. It updates the calling mod-
(*) ule of the pointer values for the front and back of the list.
(*)
(*) CALLING ARGUMENTS: List--pointer indicating the beginning of the list
(*)                      Back--pointer indicating the end of the list
(*)                      Bit--the data value to be placed in the linked list
(*)
(*) INTERNAL VARIABLES: P--temporary pointer used for traversing the list
(*)
(*) CONSTANTS: None
(*)
(*) FILES USED: None
(*)
(*) MODULES CALLED: None
(*)

```

```

procedure Enqueue(var List:dataptr;var Back:dataptr; Bit:binary);

```

```

    var P: dataptr;

```

```

    begin

```

```

        if List = nil then begin

```

```

            New(P);

```

```

            List:= P;

```

```

            P^.info:= Bit;

```

```

            Back:= P;

```

```

            Back^.next:= nil;

```

```

        end

```

```

        else begin

```

```

            New(P);

```

```

            P^.info:= Bit;

```

```

            Back^.next:= P;

```

```

            Back:= P;

```

```

            Back^.Next:= nil;

```

```

        end (*if then else*)

```

```

    end; (*procedure Enqueue*)

```

```

(******)
(*)
(*)           S h i f t R e g i s t e r
(*)
(******)
(*) DESIGNER:  Richard C. Gonder           MASTER THESIS WORK
(*) ADVISOR:  MAJ Ken Castor             MODULE NUMBER A.04
(*) DESIGN DATE: Nov 84                  LAST UPDATE: 15 Nov 1984
(*)
(*) DESCRIPTION: This procedure takes the contents of an LFSR and shifts
(*) it to the right one position. It Loads the first position with the
(*) value CK (input parameter).
(*)
(*) CALLING ARGUMENTS: C--Array representing the LFSR values
(*) N--Length or the LFSR
(*) CK--value to be input to first position of LFSR
(*) after shifting the register one position
(*)
(*) INTERNAL VARIABLES: I--used as a counter
(*)
(*) CONSTANTS: None
(*)
(*) FILES USED: None
(*)
(*) MODULES CALLED: None
(*)

```

```

procedure ShiftRegister(var C:RegStructure; N:integer; CK:binary);

```

```

    var I:integer;

```

```

    begin

```

```

        I:= N;

```

```

        repeat

```

```

            C[I]:= C[I-1];

```

```

            I:= I - 1;

```

```

        until I < 2;

```

```

        C[1]:= CK;

```

```

    end (*procedure ShiftRegister*);

```

```

(*****
(*)
(*)           M o d S u m T a p s
(*)
(*****
(*) DESIGNER:  Richard C. Gonder           MASTER THESIS WORK
(*) ADVISOR:  MAJ Ken Castor             MODULE NUMBER A.05
(*) DESIGN DATE: Nov 84                   LAST UPDATE: 15 Nov 84
(*)
(*) DESCRIPTION: This procedure takes an LFSR and its designated taps and
(*) performs a modulo 2 addition of the contents of the register as des-
(*) ignated by the taps and produces the result--CK
(*)
(*) CALLING ARGUMENTS: T--Tap assignments
(*)                      N--Length of LFSR
(*)                      R--Register array
(*)                      CK--output character of the operation
(*)
(*) INTERNAL VARIABLES: I--used as a counter
(*)
(*) CONSTANTS: None
(*)
(*) FILES USED: None
(*)
(*) MODULES CALLED: None
(*)

```

```

procedure ModSumTaps(T:RegStructure; N:integer;
                    R:RegStructure;var CK:binary);

```

```

var I : integer;

```

```

begin
  CK := '0';
  I:=0;
  repeat
    I:= I + 1;
    if T[I] = '1' then begin
      if CK = R[I] then
        CK := '0'
      else CK := '1'
    end; (*if*)
  until I = N;
end;(*procedure ModSumTaps*)

```

```

(*****
(*)
(*)           L o a d I n p u t           (*)
(*)
(*****
(*) DESIGNER:  Richard C. Gonder           MASTER THESIS WORK           (*)
(*) ADVISOR:  MAJ Ken Castor              MODULE NUMBER A.01           (*)
(*) DESIGN DATE: Nov 84                  LAST UPDATE: 15 Nov 84           (*)
(*)
(*) DESCRIPTION: This procedure loads the tap assignments for an LFSR and (*)
(*) returns the loading to the calling module.                          (*)
(*)
(*)
(*) CALLING ARGUMENTS: N--Length of the register                       (*)
(*)                   T--array of taps that are assigned in module      ~)
(*)                   Identifier--String of characters used to communi- (*)
(*)                   cate with the user                                (*)
(*)
(*) INTERNAL VARIABLES: I--Counting variable                           (*)
(*)
(*) CONSTANTS:  None                                                  (*)
(*)
(*) FILES USED:  None                                                  (*)
(*)
(*) MODULES CALLED: None                                              (*)
(*)
    procedure LoadInput(N:integer; var T:RegStructure; Identifier:String30);

    var I: integer;

    begin
        I:=1;
        while I <=N do begin
            T[I]:= '0';
            I:=I+1;
        end;(*while loop*)
        writeln('Now Select the positions for the ',Identifier,
            ' ...when complete');
        writeln('input value greater than 15');
        readln(I);
        while I < 16 do begin
            T[I] := '1';
            readln(I)
        end (*while loop*);
    end; (*procedure LoadInput*)

```



```

(*****)
(*)
(*)      G e n e r a t e C o d e s      (*)
(*)
(*****)
(*) DESIGNER:  Richard C. Gonder          MASTER THESIS WORK      (*)
(*) ADVISOR:  MAJ Ken Castor             MODULE NUMBER A.0        (*)
(*) DESIGN DATE: Nov 1984                 LAST UPDATE: 4 Mar 1985   (*)
(*)-----(*)
(*) DESCRIPTION: This program will generate codes using linear feedback (*)
(*) shift registers (LFSR) and store the codes in ASCII text files as  (*)
(*) designated by the user for further manipulation. It can be used for (*)
(*) generating Maximal Length Codes, Gold Codes and Kasami codes provided*)
(*) the user identifies the appropriate tap assignments for each. This  (*)
(*) version is limited to register lengths of 15 because it was written  (*)
(*) for an 8-bit machine. (*)
(*)-----(*)
(*) CALLING ARGUMENTS: Std Input, Output (*)
(*)-----(*)
(*) INTERNAL VARIABLES: (*)
(*)      ch—character input to the program from user (*)
(*)-----(*)
(*) CONSTANTS: None (*)
(*)-----(*)
(*) FILES USED:  User designated Filename for code storage (*)
(*)-----(*)
(*) MODULES CALLED: LinearGenerator (*)
(*)                  GoldCodeGenerator (*)
(*)                  KasamiGenerator (*)
(*)-----*)

```

```

program GenerateCodes(input,output);

```

```

    type  Binary = '0'..'1';
          RegStructure = array [1..15] of Binary;
          NameArray = string[12];
          Word = string[8];
          String30 = string[30];
          dataptr = ^data;
          data = record
              info: Binary;
              next: dataptr;
              end;(*record*)

```

```

    var  ch: char;

```

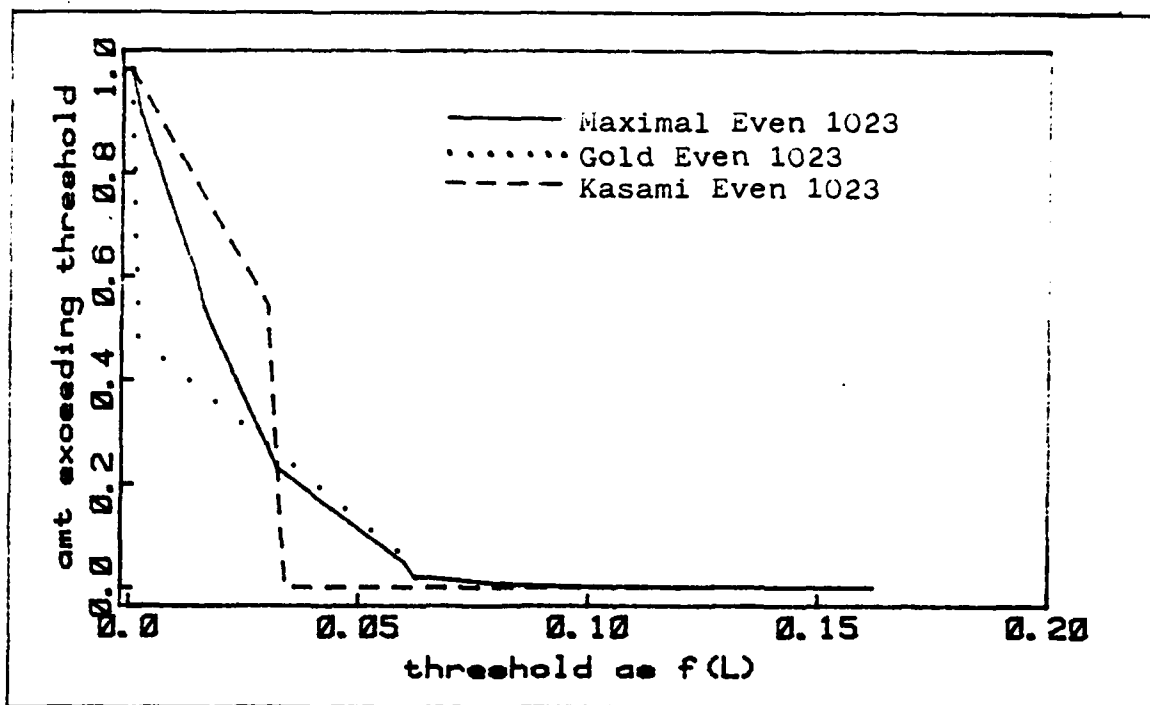


Fig. A-79. Threshold Plot of Maximum Even Correlation Functions from Mass-Correlation Algorithm for Maximal, Gold and Kasami Codes of Length 1023

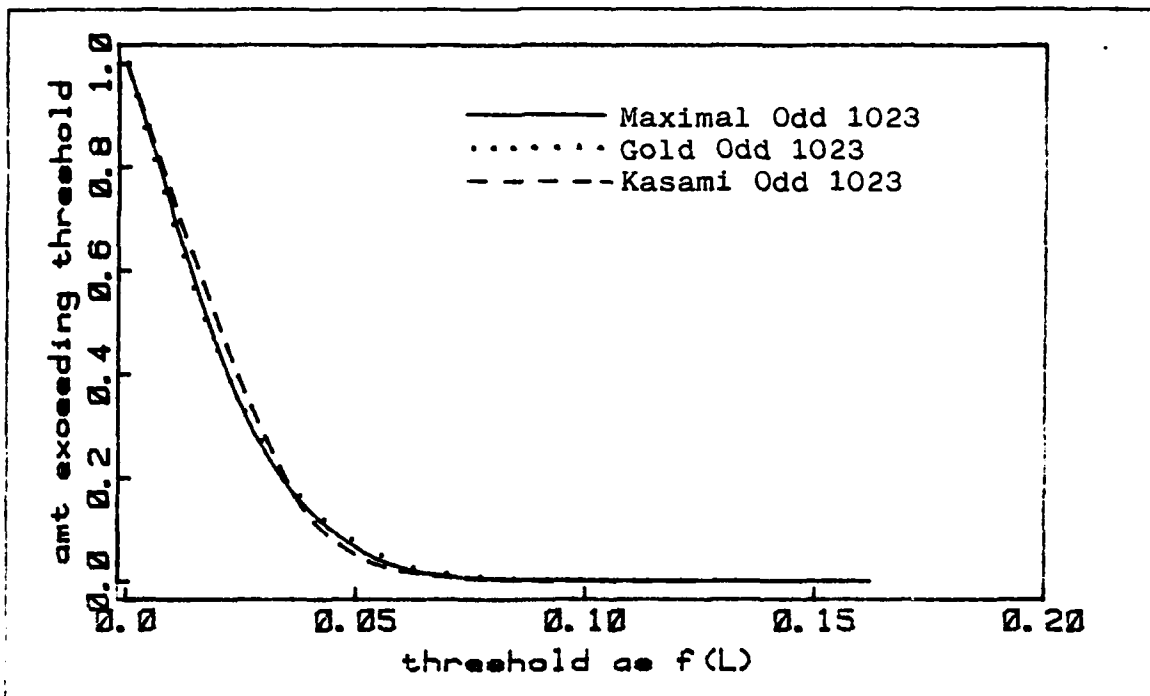


Fig. A-80. Threshold Plot of Maximum Odd Correlation Functions from Mass-Correlation Algorithm for Maximal, Gold and Kasami Codes of Length 1023

```

(*****
(*)
(*)           G e n L i n e a r
(*)
(*****
(*) DESIGNER:  Richard C. Gonder           MASTER THESIS WORK
(*) ADVISOR:  MAJ Ken Castor             MODULE NUMBER A.11
(*) DESIGN DATE: Nov 84                 LAST UPDATE: 4 Jan 85
(*)
(*) DESCRIPTION: This procedure produces a sequence of bits from an LFSR
(*) and writes the sequence into file Codedta. Maximum length of LFSR
(*) is 15 positions.
(*)
(*) CALLING ARGUMENTS: Filename—string representing storage file
(*)                      N—Length of LFSR
(*)                      Tap1—array of tap assignments
(*)
(*) INTERNAL VARIABLES: M—counter for looping
(*)                      L,Leng—Length of output sequence
(*)                      Maxi—Number of ones in output sequence
(*)                      CK—Output of generator at each clock cycle
(*)                      Maxi—number of ones generated in code
(*)
(*) CONSTANTS: None
(*)
(*) FILES USED: Writes to Codedta
(*)
(*) MODULES CALLED: ModSumTaps
(*)                  ShiftRegister
(*)
procedure GenLinear(Filename:Namearray;N:integer;var Tap1:RegStructure);

```

```

    var Maxi,M,L,Leng:integer;
        Codedta: text;
        CK: Binary;
        Reg1:RegStructure;

    begin
        Leng:= 0;
        M := 0;
        L := 1;
        while (M < N) do begin
            L := 2 * L;
            M := M + 1;
        end; (* while loop*)
        for M:= 1 to N do
            Reg1[M]:= '0';
        Reg1[N]:= '1';
        L := L - 1;
        Maxi:= 0;
        M := 0;
        assign(Codedta,Filename);
        rewrite(Codedta);
    end;

```

```

while (M < L) do begin
  ModSumTaps(Tap1,N,Reg1,CK);
  Leng := Leng + 1;
  ShiftRegister(Reg1,N,CK);
  write(Codedta,CK);
  if CK = '1' then
    Maxi:= Maxi + 1;
  M := M + 1
end; (*while loop*)
close(Codedta);
writeln('Length of code is ',Leng,' # of ones is ', Maxi);
end(*procedure GenLinear*);

```

```

(*****
(*)
(*)           G e n K a s a m i
(*)
(*****
(*) DESIGNER:  Richard C. Gonder           MASTER THESIS WORK
(*) ADVISOR:  MAJ Ken Castor             MODULE NUMBER A.3.1
(*) DESIGN DATE: Nov 84                 LAST UPDATE: 4 Jan 85
(*)
(*) DESCRIPTION: This procedure produces a sequence of bits from an LFSR
(*) and places the sequence into a linked list in memory. It then dec-
(*) imates this sequence to produce a second sequence. The second se-
(*) quence is used to produce a third sequence of the period of the se-
(*) cond that is the same length of the first. Then the procedure modu-
(*) lo-2 sums the first sequence with phase shifts of the third to pro-
(*) duce the set of Kasami Sequences. The first sequence must be a max-
(*) sequence for the generator to produce Kasami sequences.
(*)
(*) CALLING ARGUMENTS: Name-- representing storage file
(*)                      N--Length of LFSR
(*)                      Tap1--Tap assignments for LFSR
(*)                      List1,List2,list3--pointers to fronts of lists
(*)                      Back1,Back2,Back3--pointer to ends of lists of se-
(*)                      quences used to generate Kasami Sequences
(*)
(*) INTERNAL VARIABLES: Length--Length of sequence one
(*)                      L--Decimating factor
(*)                      Limit--maximum number of Kasami Sequences that
(*)                      can be produced in this configuration
(*)                      Reg1--array representing LFSR producing seq 1
(*)                      Filename--Filename w/extension added for output
(*)                      sequences
(*)                      CK--output of LFSR produces seq 1
(*)                      I,K,M,X--counters used in procedure
(*)                      st--string of length 3 used as filename extension
(*)                      L,Leng--Length of output sequence
(*)
(*) CONSTANTS: None
(*)
(*) FILES USED: Writes to Codedta
(*)
(*) MODULES CALLED: ModSumTaps
(*)                  ShiftRegister
(*)                  Enque
(*)                  Decimate
(*)                  Writelst
(*)                  ShiftList
(*)                  ModSumList
(*)
procedure GenKasami(Name:Word; N:integer; Tap1:RegStructure;
var List1, List2,List3,Back1,Back2,Back3:dataptr);

```

```

var Length,K,Limit,I,M,L,X:integer;
    CK:Binary;
    Codedata:file of char;
    st: string[3];
    Regl: RegStructure;
    Filename:NameArray;

begin
    M := 0;
    Length := 1;
    while (M < N) and (Length<16383) do begin
        Length := 2 * Length;
        M := M + 1;
    end;(*while loop*)
    Length:=Length-1;
    M:= 0;
    L:= 1;
    while (M< N/2) do begin
        L:= 2 * L;
        M:=M+1;
    end(*while loop*);
    L:= L -1;
    for I:= 1 to N do
        Regl[I]:= '0';
    Regl[N]:= '1';
    M:= 1;
    while M <= Length do begin
        ModSumTaps(Tap1,N,Regl,CK);
        Enqueue(List1,Back1,CK);
        ShiftRegister(Regl,N,CK);
        M:= M + 1;
    end; (*while loop*)
    Decimate(List1,Back1,List2,Back2,List3,Back3,L,Length);
    M:= 1;
    writeln('How many codes do you want generated? You may choose');
    writeln('from 1 to ',L,' Please input number');
    readln(Limit);
    if Limit >= 1 then begin
        Filename:= Name + '.001';
        writeln(Filename);
        WriteList(List1, Filename);
    end; (*if*)
    Limit:= Limit - 1;
    X:= 1;
    while (X < L) and (X <= Limit) do begin
        str(X+1:3,st);
        if st[1] = ' ' then st[1]:= '0';
        if st[2] = ' ' then st[2]:= '0';
        Filename:= Name + '.' + st;
        writeln(Filename);
        ModSumList(List1,List3,Filename);
        ShiftList(List1);
        X:= X+1;
    end; (*while loop*)
end;(*procedure GenKasami*)

```

```

(*****
(*)
(*)      G o l d C o d e G e n e r a t o r      (*)
(*)
(*****
(*) DESIGNER:  Richard C. Gonder          MASTER THESIS WORK      (*)
(*) ADVISOR:  MAJ Ken Castor             MODULE NUMBER A.2        (*)
(*) DESIGN DATE: Dec 84                  LAST UPDATE: 4 Jan 85    (*)
(*)-----(*)
(*) DESCRIPTION: This procedure does most of the program/user interaction (*)
(*) to generate Gold codes. Reads in Length of LFSR, Tap assignments and (*)
(*) Filename for storage. (*)
(*)-----(*)
(*) CALLING ARGUMENTS: None (*)
(*)-----(*)
(*) INTERNAL VARIABLES: Reg1--array representing LFSR (*)
(*) Tap1,Tap2--Tap assignments for two LFSR's used to (*)
(*) produce two linear Sequences (*)
(*) N--length of LFSR's (*)
(*) Name--user inputted filename for storage of Gold (*)
(*) sequences...w/o extension (*)
(*) List1,List2--pointers to front of LFSR sequences (*)
(*) Back1,Back2--pointers to end of LFSR sequences (*)
(*) Flag--boolean used for control (*)
(*) ch--character used for user I/O (*)
(*)-----(*)
(*) CONSTANTS: None (*)
(*)-----(*)
(*) FILES USED: None (*)
(*)-----(*)
(*) MODULES CALLED: Test (*)
(*) LoadInput (*)
(*) GenGold (*)
(*) GetRidOf (*)
(*)-----*)

```

```

procedure GoldCodeGenerator;

```

```

var   Reg1,Tap1,Tap2 : RegStructure;
      N : integer;
      Name: Word;
      ch: char;
      Flag: boolean;
      List1,List2,Back1,Back2: dataptr;

```

```

begin
  repeat
    List1:= nil;
    List2:= nil;
    writeln('Input the length of the two registers.
            ..2 - 15 please ');
    readln(N);
    while (N < 2) or (N > 15) do begin
      writeln('range must be 2 to 15 please');

```

```

        readln(N)
    end; (*while loop*)
    LoadInput(N, Tap1, 'Taps for register number 1');
    LoadInput(N, Tap2, 'Taps for register number 2');
    writeln('What file do you want to store the sequence
            generated');
    writeln('No more than 8 characters in filename');
    repeat
        Flag:= true;
        readln(Name);
        Test(Name, Flag);
        if not Flag then begin
            writeln('Invalid Input Please try again');
            writeln('No Special Characters on Inputted Name');
        end; (*if statement*)
    until Flag;
    writeln('Code will be stored in file ', Name);
    GenGold(Name, N, Tap1, Tap2, List1, List2);
    writeln;
    writeln('Do you wish to generate any other Gold codefiles?');
    writeln('If so, input Y if not, input N');
    readln(ch);
    GetRidOf(List1);
    GetRidOf(List2);
    until (ch = 'N') or (ch = 'n');
end; (*of procedure GoldCodeGenerator*)

```

```

(*****
(*)
(*)           L i n e a r G e n e r a t o r
(*)
(*****
(*) DESIGNER:  Richard C. Gonder           MASTER THESIS WORK
(*) ADVISOR:  MAJ Ken Castor             MODULE NUMBER A.1
(*) DESIGN DATE: Nov 84                 LAST UPDATE: 4 Dec 84
(*)
(*) DESCRIPTION: This procedure does most of the user/program interaction
(*) required to produce a sequence from a single LFSR.  The Length of
(*) The LFSR, tap assignments and Filename for storage are obtained
(*) here.
(*)
(*) CALLING ARGUMENTS: None
(*)
(*) INTERNAL VARIABLES: Tap1--tap assignment for LFSR
(*)                      N--length of LFSR
(*)                      Filename--string for output filename w/ext
(*)                      Name--string for output filename w/o ext
(*)                      Flag--boolean used for control
(*)                      ch--character used for user I/O
(*)
(*) CONSTANTS: None
(*)
(*) FILES USED: None
(*)
(*) MODULES CALLED: LoadInput
(*)                  Test
(*)                  GenLinear
(*)

```

```

  procedure LinearGenerator;

```

```

  var  Tap1 : RegStructure;
        N : integer;
        Filename: NameArray;
        Name: Word;
        ch: char;
        Flag: boolean;

```

```

  begin
    repeat
      writeln('Input the length of the linear shift register');
      writeln('Length should be between 2-15 please');
      readln(N);
      while (N < 2) or (N > 15) do begin
        writeln('range must be between 2 to 15..please enter again');
        readln(N);
      end(*while loop*);
      LoadInput(N,Tap1,'Taps for the shiftregister');
      writeln('What file do you want to store the sequence
              generated?');
      writeln('No more than 8 characters in the filename');

```

```

repeat
  Flag:= true;
  readln(Name);
  Test(Name,Flag);
  if not Flag then begin
    writeln('Invalid input...Please try again');
    writeln('No special characters in filename');
  end;(*if*)
until Flag;
Filename := Name + '.DTA';
writeln('Code will be stored in file ',Filename);
GenLinear(Filename,N,Tap1);
writeln('Do you wish to generate any Linear shift codefiles?');
writeln('If so, input Y...if not, input N');
readln(ch);
until (ch = 'N') or (ch = 'n');
end; (*procedure Linear Generator*)

```

```

(*****
(*)
(*)           K a s a m i G e n e r a t o r           (*)
(*)
(*****
(*) DESIGNER:  Richard C. Gonder           MASTER THESIS WORK (*)
(*) ADVISOR:  MAJ Ken Castor             MODULE NUMBER A.3 (*)
(*) DESIGN DATE: Jan 85                 LAST UPDATE: 9 Mar 85 (*)
(*)-----(*)
(*) DESCRIPTION: This procedure does most of the user/program interaction (*)
(*) required to produce a set of Kasami codes. The Length of the LFSR, (*)
(*) tap assignments and Filename for storage are obtained here. (*)
(*)-----(*)
(*) CALLING ARGUMENTS: None (*)
(*)-----(*)
(*) INTERNAL VARIABLES: Tap1--tap assignment for LFSR (*)
(*) N--length of LFSR (*)
(*) Name--string for output filename w/o ext (*)
(*) Flag--boolean used for control (*)
(*) ch--character used for user I/O (*)
(*) Reg1--Register array for LFSR (*)
(*) List1,List2,List3--pointers to sequences used to (*)
(*) generate Kasami Sequences (*)
(*) Back1,Back2,Back3--pointers to ends of sequences (*)
(*)-----(*)
(*) CONSTANTS: None (*)
(*)-----(*)
(*) FILES USED: None (*)
(*)-----(*)
(*) MODULES CALLED: LoadInput (*)
(*) Test (*)
(*) GenKasami, GetRidOf (*)
(*)-----*)

```

```

procedure KasamiGenerator;

```

```

var   Reg1,Tap1 : RegStructure;
      N : integer;
      Name: Word;
      ch: char;
      Flag: boolean;
      List1,List2,List3,Back1,Back2,Back3: dataptr;

```

```

begin
  repeat
    List1:= nil;
    List2:= nil;
    Back1:= nil;
    Back2:= nil;
    List3:= nil;
    Back3:= nil;
    writeln('This generation uses one linear sequence and its
            decimation to produce');
    writeln('the desired code set');

```

```

writeln('Input the length of the register...2 - 15 please ');
readln(N);
while (N < 2) or (N > 15) or odd(N) do begin
    writeln('range must be 2 to 14 and an even integer please');
    readln(N)
end; (*while loop*)
LoadInput(N,Tapl,'Taps for register number 1');
writeln('What file do you want to store the sequence generated');
writeln('No more than 8 characters in filename');
repeat
    Flag:= true;
    readln(Name);
    Test(Name,Flag);
    if not Flag then begin
        writeln('Invalid Input Please try again');
        writeln('No Special Characters on Inputted Name');
    end; (*if statement*)
until Flag;
writeln('Code will be stored in file ',Name);
GenKasami(Name,N,Tapl,List1,List2,List3,Back1,Back2,Back3);
writeln;
writeln('Do you wish to generate any other Kasami codefiles?');
writeln('If so, input Y if not, input N');
readln(ch);
GetRidOf(List1);
GetRidOf(List2);
GetRidOf(List3);
until (ch = 'N') or (ch = 'n');
end; (*of procedure KasamiGenerator*)

```

```

(*****
(*)
(*)          M A I N   P R O G R A M          (*)
(*)
(*****)
begin (*Main program*)
  writeln('This program generates code sequences of Linear shift
    registers');
  writeln('and stores them in designated files for further
    manipulation');
  writeln('You have the option to generate Maximal codes using the
    Linear');
  writeln('Shift register option with appropriate input for tap
    assignments');
  writeln('or you can produce gold codes using two linear shift
    registers');
  writeln('and inputting the appropriate taps as dictated by the
    preferred');
  writeln('pair polynomials');
  writeln;
  writeln('UNFORTUNATELY the software does not catch improper tap
    assignments');
  writeln('for maximal or gold code sequences, Therefore you must
    insure that');
  writeln('your assignments are correct...Check a book if necessary!');
  writeln;
  writeln('Would you like to generate any codefiles at this time?');
  writeln('Input Y for yes or N for no');
  readln(ch);
  while (ch = 'Y') or (ch = 'y') do begin
    writeln('Which type of code would you like to generate at this
      time?');
    writeln('Input L for Linear shift register code generation');
    writeln('Input G for Gold code generation');
    writeln('Input K for Kasami Generation');
    readln(ch);
    case ch of
      'L' , 'l' : begin
        ClrScr;
        LinearGenerator;
      end;
      'G' , 'g' : begin
        ClrScr;
        GoldCodeGenerator;
      end;
      'K' , 'k' : begin
        ClrScr;
        KasamiGenerator;
      end;
    end;(*case*)
  end;
end;

```

```
        writeln('Would you like to generate any more codefiles Y/N ?');  
        readln(ch);  
    end(*while loop*);  
end.(*of program GenerateCodes*)
```

```

(*****
(*)
(*)           P a r t i a l C o r r e l a t e
(*)
(*****
(*) DESIGNER:  Richard C. Gonder           MASTER THESIS WORK
(*) ADVISOR:  MAJ Ken Castor             MODULE NUMBER B.0
(*) DESIGN DATE: Jan 1985                LAST UPDATE: 1 Apr 1985
(*)
(*) DESCRIPTION:  This program will perform correlations on user designa-
(*) code files composed of ASCII 1's and 0's. The options offered by
(*) the program in a menu driven fashion, are to produce the periodic
(*) odd and even correlation functions. Other options performed are the
(*) thresholding of the Odd, Even functions. This process will produce
(*) data that can be plotted, threshold vs amount exceeding threshold.
(*) The user is given the option of which thresholding process positive,
(*) negative or both. All manipulation of data is done dynamically with
(*) pointers, so that the program needs no modification when larger mem-
(*) ory devices are used. The size of the memory of the machine used is
(*) the only limitation on the length of the input codes..
(*)
(*) CALLING ARGUMENTS: Std Input, Output, Designated Text File
(*)
(*) INTERNAL VARIABLES:
(*)   Phase—the phase shift of desired process
(*)   PctRight,PctLeft—used for output of thresholding values
(*)   Sumleven,Sum2even,Sumlodd,Sum2odd—used temp storage values
(*)       for thresholding process
(*)   Threshold—actual threshold passed into threshold process
(*)   DataList,Back—pointers used to create/manipulate output data
(*)   Resultfile,Codedta—File variable for file I/O
(*)   Result,Filenal,Filena2: string for filename with extension
(*)   Name—User-inputted filename for storage of the code
(*)   ch—character input to the program from user
(*)   FlagA,FlagB,FlwgE,stop—booleans used to control processes
(*)       performed as set by user's response to menus
(*)   List1,List2 used to identify the beginning of linked lists
(*)       representing the codes to be correlated
(*)
(*) CONSTANTS: None
(*)
(*) FILES USED:  User designated Filename for code readin and data output
(*)
(*) MODULES CALLED:  CreateList
(*)                   ShiftList
(*)                   Correlate
(*)                   Writelst
(*)                   CountUpper
(*)                   CountLower
(*)                   DisposeOf
(*)                   GetRidOf
(*)

```

```

program PartialCorrelate(input, output);

type
  Name = string[12];
  pointer = ^ datarecord;
  datarecord =
    record
      data: char;
      next: pointer
    end; (*of record*)
  dataptr = ^ info;
  info =
    record
      Even: real;
      Odd: real;
      next1: dataptr
    end; (*record*)

var
  Phase, Delay, J, N: integer;
  List1, List2: pointer;
  DataList, Back: dataptr;
  Result, Filena1, Filena2: Name;
  Resultfile, Codedta: text;
  PctRight, PctLeft, PctEven, PctOdd,
  Sumleven, Sum2even, Sumlodd, Sum2odd, Threshold: real;
  ch: char;
  FlagA, FlagB, FlagE, stop: boolean;

```

```

writeln('A--Even function only');
writeln('B--Odd function only');
writeln('E--both of the above');
writeln;
writeln('Select one please');
FlagA := false;
FlagB := false;
readln(ch);
case ch of
    'A', 'a':
        begin
            FlagA := true;
            writeln('Even function selected')
        end;
    'B', 'b':
        begin
            FlagB := true;
            writeln('Odd function selected')
        end;
    'E', 'e':
        begin
            FlagA := true;
            FlagB := true;
            FlagE := true;
            writeln('All selected')
        end
end; (*case*)
writeln('You have a choice of --');
writeln(' U--using upper Threshold only');
writeln(' L--using lower Threshold only');
writeln(' B--using both Threshold');
writeln('please select one');
readln(ch);
Threshold := 0;
stop := false;
writeln('WORKING ON Threshold ALGORITHM');
Sumleven := 0;
Sum2even := 0;
Sumlodd := 0;
Sum2odd := 0;
while (Threshold <= N) and not stop do begin
    case ch of
        'U', 'u':
            begin
                CountUpper(DataList, Threshold,
                    Sumleven, Sumlodd)
            end;
    end;
end;

```

```

(*****)
(*)
(*)      M A I N   P R O G R A M      (*)
(*)
(*)
(*****)
begin
  writeln('This program allows you to do correlations of ASCII code files');
  writeln('representing the pseudorandom sequences. You may get output of');
  writeln('the actual Odd and Even correlation function, Right and Left ');
  writeln('Partial correlations or the threshold functions of any or all of');
  writeln('of these four. ');
  writeln('The Phase of the codefiles may be varied from 0 to L');
  writeln;
  writeln('Do you want to do any correlations at this time?..Y/N');
  readln(ch);
  if (ch = 'y') or (ch = 'Y') then begin
    repeat
      writeln('enter the first file you wish to correlate');
      readln(Filenal);
      writeln('now enter the second Filename');
      readln(Filena2);
      writeln('Enter phaseshift desired for run...0 to length');
      readln(Phase);
      writeln('What of the following options do you desire?');
      writeln('A — a listing of actual correlation functions');
      writeln('B — a listing of Threshold functions');
      readln(ch);
      CreateList(Filenal, List1, N);
      CreateList(Filena2, List2, N);
      for J := 1 to Phase do begin
        ShiftList(List1);
        ShiftList(List2)
      end; (*for loop*)
      DataList := nil;
      Back := nil;
      Correlate(DataList, Back, Delay, List1, List2, N);
      repeat
        writeln('Where do you want to store the output?');
        readln(Result);
        assign(Resultfile, Result);
        rewrite(Resultfile);
        case ch of
          'A', 'a':
            begin
              Writelst(DataList, Resultfile)
            end;
          'B', 'b':
            begin
              ClrScr;
              writeln('You have the choice of the following listings
              for Threshold functions:');

```

```

(*****)
(*)
(*)           W r i t e l i s t
(*)
(*****)
(*) DESIGNER:  Richard C. Gonder           MASTER THESIS WORK
(*) ADVISOR:  MAJ Ken Castor             MODULE NUMBER B.4
(*) DESIGN DATE: Nov 1984                 LAST UPDATE: 14 Mar 1985
(*)
(*) DESCRIPTION:  This procedure takes a linked list of data and writes
(*) the data fields, even, odd, right and left to the designated file, as
(*) well as the chip location from 0 to L, the length of the list.
(*)
(*) CALLING ARGUMENTS: List--pointer to front of datalist
(*) OutFile--string for output filename
(*)
(*) INTERNAL VARIABLES:
(*) P--temporary pointer used to traverse list
(*) K--Counter used as chip position in list
(*)
(*) CONSTANTS: None
(*)
(*) FILES USED:  Writes to OutFile
(*)
(*) MODULES CALLED:  None
(*)

```

```

procedure Writelst(List: dataptr; var OutFile: text);

```

```

var

```

```

    K: integer;

```

```

    P: dataptr;

```

```

begin

```

```

    K := 0;

```

```

    P := List;

```

```

    while P <> nil do begin

```

```

        writeln(OutFile, K, ',', P^.Even: 5: 2, ',', P^.Odd: 5: 2);

```

```

        K := K + 1;

```

```

        P := P^.next1

```

```

    end (*while loop*)

```

```

end; (*procedure Writelst*)

```

```

        R := (-Ptr1^.Even - Thresh) / (-Ptr1^.Even + Ptr2^.Even);
        Count1 := Count1 + R
    end
end (*if*); (*if FlagA*)
if FlagB then begin
    if (-Ptr1^.Odd < Thresh) and (-Ptr2^.Odd >= Thresh) then begin
        R := (-Ptr2^.Odd - Thresh) / (-Ptr2^.Odd + Ptr1^.Odd);
        Count2 := Count2 + R
    end; (*if*)
    if (-Ptr1^.Odd >= Thresh) and (-Ptr2^.Odd >= Thresh) then begin
        Count2 := Count2 + 1
    end; (*if*)
    if (-Ptr1^.Odd >= Thresh) and (-Ptr2^.Odd < Thresh) then begin
        R := (-Ptr1^.Odd - Thresh) / (-Ptr1^.Odd + Ptr2^.Odd);
        Count2 := Count2 + R
    end
end (*if*); (*if FlagB*)
Ptr1 := Ptr2;
until Ptr1^.next1 = nil
end; (*procedure CountLower*)

```

```

(*****
(*)
(*)           C o u n t L o w e r
(*)
(*****
(*) DESIGNER:  Richard C. Gonder          MASTER THESIS WORK
(*) ADVISOR:  MAJ Ken Castor             MODULE NUMBER B.6
(*) DESIGN DATE: Jan 1985                LAST UPDATE: 26 Jan 1985
(*)
(*) DESCRIPTION:  This procedure takes a data linked list representing the
(*) the even an odd correlation functions, and determines what amount of
(*) of each curve lies below the negative threshold. It utilizes the
(*) fact that from one data record to the next, 1 chip time elapses.
(*) Essentially a similar triangle operation using the correlation values
(*) from one point to the next, and the threshold.
(*)
(*) CALLING ARGUMENTS: List--pointer to front of datalist
(*)                      Thresh--threshold value
(*)                      Count1,Count2--Counters used to add amount of curve
(*)                      exceeds the threshold for each function
(*)
(*) INTERNAL VARIABLES:
(*)                      Ptr1,Ptr2--temp pointer used to do the process
(*)                      R--temp storage location used with counter
(*)
(*) CONSTANTS: None
(*)
(*) FILES USED:  None
(*)
(*) MODULES CALLED:  None
(*)

```

```

procedure CountLower(List: dataptr; Thresh: real;
                      var Count1, Count2: real);

```

```

var
  Ptr1, Ptr2: dataptr;
  R: real;

begin
  Count1 := 0;
  Count2 := 0;
  Ptr1 := List;
  repeat
    Ptr2 := Ptr1^.next1;
    if FlagA then begin
      if (-Ptr1^.Even < Thresh) and (-Ptr2^.Even >= Thresh) then begin
        R := (-Ptr2^.Even - Thresh) / (-Ptr2^.Even + Ptr1^.Even);
        Count1 := Count1 + R
      end; (*if*)
      if (-Ptr1^.Even >= Thresh) and (-Ptr2^.Even >= Thresh) then begin
        Count1 := Count1 + 1
      end; (*if*)
      if (-Ptr1^.Even >= Thresh) and (-Ptr2^.Even < Thresh) then begin

```

```

        R := (Ptr1^.Even - Thresh) / (Ptr1^.Even - Ptr2^.Even);
        Count1 := Count1 + R
    end; (*if*)
end; (*if Flag A*)
if FlagB then begin
    if (Ptr1^.Odd < Thresh) and (Ptr2^.Odd >= Thresh) then begin
        R := (Ptr2^.Odd - Thresh) / (Ptr2^.Odd - Ptr1^.Odd);
        Count2 := Count2 + R
    end; (*if*)
    if (Ptr1^.Odd >= Thresh) and (Ptr2^.Odd >= Thresh) then begin
        Count2 := Count2 + 1
    end; (*if*)
    if (Ptr1^.Odd >= Thresh) and (Ptr2^.Odd < Thresh) then begin
        R := (Ptr1^.Odd - Thresh) / (Ptr1^.Odd - Ptr2^.Odd);
        Count2 := Count2 + R
    end
end
end (*if*); (*if FlagB*)
Ptr1 := Ptr2;
until Ptr1^.next1 = nil
end; (*procedure CountUpper*)

```

```

(*****
(*)
(*)           C o u n t U p p e r           (*)
(*)
(*****
(*) DESIGNER:  Richard C. Gonder           MASTER THESIS WORK (*)
(*) ADVISOR:  MAJ Ken Castor             MODULE NUMBER B.5 (*)
(*) DESIGN DATE: Jan 1985                 LAST UPDATE: 26 Jan 1985 (*)
(*)-----(*)
(*) DESCRIPTION: This procedure takes a data linked list representing the*)
(*) the even and odd correlation functions, and determines what amount of*)
(*) each function lies above the inputted threshold. It utilizes the *)
(*) fact that from one data record to the next, 1 chip time elapses. *)
(*) Essentially a similar triangle operation using the correlation values*)
(*) from one point to the next, and the threshold. *)
(*)-----(*)
(*) CALLING ARGUMENTS: List--pointer to front of datalist (*)
(*)                      Thresh--threshold value (*)
(*)                      Count1,Count2--Counters used to add amount of curve*)
(*)                      exceeds the threshold for each function (*)
(*)-----(*)
(*) INTERNAL VARIABLES: (*)
(*)                      Ptr1,Ptr2--temp pointer used to do the process (*)
(*)                      R--temp storage location used with counter (*)
(*)-----(*)
(*) CONSTANTS: None (*)
(*)-----(*)
(*) FILES USED: None (*)
(*)-----(*)
(*) MODULES CALLED: None (*)
(*)-----*)

```

```

procedure CountUpper(List: dataptr; Thresh: real;
                      var Count1, Count2: real);

```

```

var

```

```

    Ptr1, Ptr2: dataptr;
    R: real;

```

```

begin

```

```

    Count1 := 0;
    Count2 := 0;
    Ptr1 := List;
    repeat
        Ptr2 := Ptr1^.next1;
        if FlagA then begin
            if (Ptr1^.Even < Thresh) and (Ptr2^.Even >= Thresh) then begin
                R := (Ptr2^.Even - Thresh) / (Ptr2^.Even - Ptr1^.Even);
                Count1 := Count1 + R
            end; (*if*)
            if (Ptr1^.Even >= Thresh) and (Ptr2^.Even >= Thresh) then begin
                Count1 := Count1 + 1
            end; (*if*)
            if (Ptr1^.Even >= Thresh) and (Ptr2^.Even < Thresh) then begin

```

```

        if Ptr1^.data = Ptr2^.data then
            Agree := Agree + 1
        else
            DisAgree := DisAgree + 1;
            Ptr1 := Ptr1^.next;
            Ptr2 := Ptr2^.next
        end; (*while Loop*)
    RightCor := Agree - DisAgree;
    Agree := 0;
    DisAgree := 0;
    Ptr1 := List1;
    while Ptr2 <> nil do begin
        if Ptr1^.data = Ptr2^.data then
            Agree := Agree + 1
        else
            DisAgree := DisAgree + 1;
            Ptr1 := Ptr1^.next;
            Ptr2 := Ptr2^.next
        end; (*while Loop*)
    LeftCor := Agree - DisAgree;
    Even := LeftCor + RightCor;
    Odd := RightCor - LeftCor;
    Enqueue(DataList, Back, Odd, Even);
    K := K + 1;
    Delay := Delay + 1
end; (*while loop*)
Agree := 0;
DisAgree := 0;
Ptr1 := List1;
Ptr2 := List2;
while Ptr2 <> nil do begin
    if Ptr1^.data = Ptr2^.data then
        Agree := Agree + 1
    else
        DisAgree := DisAgree + 1;
        Ptr1 := Ptr1^.next;
        Ptr2 := Ptr2^.next
    end; (*while Loop*)
LeftCor := Agree - DisAgree;
Even := LeftCor;
Odd := -LeftCor;
RightCor := 0;
Enqueue(DataList, Back, Odd, Even)
end; (*procedure correlate*)

```

```

(*****
(*)
(*)           C o r r e l a t e
(*)
(*****
(*) DESIGNER:  Richard C. Gonder           MASTER THESIS WORK
(*) ADVISOR:   MAJ Ken Castor             MODULE NUMBER B.1
(*) DESIGN DATE: Jan 1985                 LAST UPDATE: 14 Jan 1985
(*)
(*) DESCRIPTION: This procedure takes the 2 Linked Lists of characters
(*) representing codesets and determines the even, odd, right and left
(*) correlation functions at discrete values. It places the determined
(*) values at corresponding positions in a third linked list
(*)
(*) CALLING ARGUMENTS: DataList,Back—pointers to front and rear of output
(*)                      Datalist
(*)                      Delay—offset of lists during correlation cycle(TAU)
(*)                      List1,List2—pointers for front of codelists
(*)                      N—Length of List
(*)
(*) INTERNAL VARIABLES:
(*)      Ptr1,Ptr2—temp pointer used to do the process
(*)      LeftCor, RightCor—partial correlations values used to de-
(*)                      termine the even and odd correlation functions
(*)      Even,Odd—discrete correlation values
(*)      Agree,Disagree—used as counters during correlation process
(*)      Count,K—counters used in process
(*)
(*) CONSTANTS: None
(*)
(*) FILES USED:  None
(*)
(*) MODULES CALLED:  Enque
(*)

```

```

procedure Correlate(var DataList, Back: dataptr; Delay: integer;
                    List1: pointer; List2: pointer; N: integer);

```

```

var
  LeftCor, RightCor, Even, Odd: real;
  Count, Agree, DisAgree, K: integer;
  Ptr1, Ptr2: pointer;

```

```

begin
  K := 1;
  while K <= N do begin
    Ptr1 := List1;
    Ptr2 := List2;
    Agree := 0;
    DisAgree := 0;
    for Count := 2 to K do
      Ptr1 := Ptr1^.next;
    while Ptr1 <> nil do begin

```

```

(*****)
(*)
(*)           S h i f t L i s t           (*)
(*)
(*****)
(*) DESIGNER:  Richard C. Gonder           MASTER THESIS WORK (*)
(*) ADVISOR:  MAJ Ken Castor             MODULE NUMBER B.3 (*)
(*) DESIGN DATE: Jan 1985                 LAST UPDATE: 14 Jan 1985 (*)
(*)
(*) DESCRIPTION: This procedure merely takes the first record of the (*)
(*) linked list and places it at the end of the list. (*)
(*)
(*) CALLING ARGUMENTS: List—pointers to front of list (*)
(*)
(*) INTERNAL VARIABLES: (*)
(*) P,Ptr—temp pointer used to do the process (*)
(*)
(*) CONSTANTS: None (*)
(*)
(*) FILES USED: None (*)
(*)
(*) MODULES CALLED: None (*)
(*)

```

```

procedure ShiftList(var List: pointer);

```

```

var

```

```

    First, P: pointer;

```

```

begin

```

```

    First := List;

```

```

    P := List;

```

```

    while P^.next <> nil do

```

```

        P := P^.next;

```

```

    First := First^.next;

```

```

    P^.next := List;

```

```

    List^.next := nil;

```

```

    List := First

```

```

end; (*procedure ShiftList*)

```

```

(*****)
(*)
(*)           G e t R i d O f           (*)
(*)
(*****)
(*) DESIGNER:  Richard C. Gonder           MASTER THESIS WORK           (*)
(*) ADVISOR:  MAJ Ken Castor             MODULE NUMBER B.8           (*)
(*) DESIGN DATE: Jan 1985                 LAST UPDATE: 14 Jan 1985       (*)
(*)-----(*)
(*) DESCRIPTION:  This procedure releases a linked list from memory (*)
(*) allocation.  Specifically used on output data lists destruction. (*)
(*)-----(*)
(*) CALLING ARGUMENTS: List--pointers to front of list (*)
(*)-----(*)
(*) INTERNAL VARIABLES: (*)
(*)      P,Ptr--temp pointer used to do the process (*)
(*)-----(*)
(*) CONSTANTS: None (*)
(*)-----(*)
(*) FILES USED:  None (*)
(*)-----(*)
(*) MODULES CALLED:  None (*)
(*)-----(*)

procedure GetRidOf(var List: dataptr);

var
    P, ptr: dataptr;

begin
    P := List;
    ptr := P;
    while P^.nextl <> nil do begin
        ptr := P^.nextl;
        dispose(P);
        P := ptr
    end (*while loop*);
    dispose(P);
    List := nil
end; (*procedure GetRidOf*)

```

```

(*****)
(*)
(*)           D i s p o s e O f           (*)
(*)
(*****)
(*) DESIGNER:  Richard C. Gonder           MASTER THESIS WORK (*)
(*) ADVISOR:  MAJ Ken Castor             MODULE NUMBER B.7 (*)
(*) DESIGN DATE: Jan 1985                 LAST UPDATE: 14 Jan 1985 (*)
(*)-----(*)
(*) DESCRIPTION: This procedure releases a linked list from memory (*)
(*) allocation. Specifically used on input data lists destruction. (*)
(*)-----(*)
(*) CALLING ARGUMENTS: List—pointers to front of list (*)
(*)-----(*)
(*) INTERNAL VARIABLES: (*)
(*) P,Ptr—temp pointer used to do the process (*)
(*)-----(*)
(*) CONSTANTS: None (*)
(*)-----(*)
(*) FILES USED: None (*)
(*)-----(*)
(*) MODULES CALLED: None (*)
(*)-----(*)

procedure DisposeOf(var List: pointer);

var
    P, ptr: pointer;

begin
    P := List;
    ptr := P;
    while P^.next <> nil do begin
        ptr := P^.next;
        dispose(P);
        P := ptr
    end (*while loop*);
    dispose(P);
    List := nil
end; (*procedure DisposeOf*)

```

```

(*****
(*)
(*)           E n q u e
(*)
(*****
(*) DESIGNER:  Richard C. Gonder           MASTER THESIS WORK
(*) ADVISOR:  MAJ Ken Castor             MODULE NUMBER B.1.1
(*) DESIGN DATE: Nov 1984                 LAST UPDATE: 14 Dec 1984
(*)
(*) DESCRIPTION:  This procedure enqueue a data record representing the
(*) values of the even, odd, right and left correlation functions at a
(*) discrete value in an ordered linked list.
(*)
(*) CALLING ARGUMENTS: List,Back—pointers to front and rear of list
(*)                      Odd,Even—values of correlation functions
(*)
(*) INTERNAL VARIABLES:
(*)      P—temp pointer used to do the process
(*)
(*) CONSTANTS: None
(*)
(*) FILES USED:  None
(*)
(*) MODULES CALLED:  None
(*)

```

```

procedure Enqueue(var List, Back: dataptr; Odd, Even: real);

```

```

var

```

```

    P: dataptr;

```

```

begin

```

```

    if List = nil then begin

```

```

        new(P);

```

```

        List := P;

```

```

        P^.Even := Even;

```

```

        P^.Odd := Odd;

```

```

        Back := P;

```

```

        Back^.nextl := nil

```

```

    end else begin

```

```

        new(P);

```

```

        P^.Even := Even;

```

```

        P^.Odd := Odd;

```

```

        Back^.nextl := P;

```

```

        Back := P;

```

```

        Back^.nextl := nil

```

```

    end (*if then else*)

```

```

end; (*procedure Enqueue*)

```

```

(*****)
(*)
(*)          C r e a t e L i s t          (*)
(*)
(*****)
(*) DESIGNER:  Richard C. Gonder          MASTER THESIS WORK          (*)
(*) ADVISOR:  MAJ Ken Castor              MODULE NUMBER B.2          (*)
(*) DESIGN DATE: Jan 1985                LAST UPDATE: 4 Jan 1985      (*)
(*)-----(*)
(*) DESCRIPTION: This procedure will read in a file of char, placing each*)
(*) character into a linked list. It terminates at the EOF of the file *)
(*) read.                                     *)
(*)-----(*)
(*) CALLING ARGUMENTS: Filename--string of char                          *)
(*)                      List--pointer to the beginning of linked list    *)
(*)                      K--number of characters read into list            *)
(*)-----(*)
(*) INTERNAL VARIABLES:                                                  *)
(*)      P,ptr--used to construct list                                    *)
(*)      bit--temp location for character read                            *)
(*)-----(*)
(*) CONSTANTS: None                                                    *)
(*)-----(*)
(*) FILES USED:  User designated Filename for code storage              *)
(*)-----(*)
(*) MODULES CALLED:  None                                              *)
(*)-----(*)
  procedure CreateList(Filename: Name; var List: pointer; var K: integer);

  var
    P, ptr: pointer;
    bit: char;

  begin
    assign(Codedta,Filename);
    reset(Codedta);
    new(P);
    List := P;
    ptr := P;
    K := 0;
    while not eof(Codedta) do begin
      read(Codedta, bit);
      K := K + 1;
      P^.data := bit;
      ptr^.next := P;
      ptr := P;
      new(P)
    end; (*while loop*)
    close(Codedta);
    ptr^.next := nil;
  end; (*procedure CreateList*)

```

```

        'L', 'l':
            begin
                CountLower(DataList, Threshold,
                           Sum2even, Sum2odd)
            end;
        else
            CountUpper(DataList, Threshold,
                       Sumleven, Sumlodd);
            CountLower(DataList, Threshold,
                       Sum2even, Sum2odd)
        end; (*case*)
        PctEven := (Sumleven + Sum2even) / N;
        PctOdd := (Sumlodd + Sum2odd) / N;
        write(Resultfile, Threshold / N: 5: 4);
        if FlagA then
            write(Resultfile, ',', PctEven: 8: 7);
        if FlagB then
            write(Resultfile, ',', PctOdd: 8: 7);
        write(Threshold / N: 5: 4);
        if FlagA then
            write(',', PctEven: 8: 7);
        if FlagB then
            write(',', PctOdd: 8: 7);
        writeln;
        writeln(Resultfile);
        if Threshold / N > 0.10 then
            Threshold := Threshold + N / 100
        else
            Threshold := Threshold + N / 500;
        if not FlagE and FlagA and (PctEven = 0.0) then
            stop := true;
        if not FlagE and FlagB and (PctOdd = 0.0) then
            stop := true;
        if FlagE and (PctEven = 0.0) and (PctOdd = 0.0) then
            stop := true
        end
    end
end (*while loop*) (*B -option*); (*case*)
close(Resultfile);
writeln('Do you want any other files listed with this
dataset of codes?');
writeln('What of the following options do you desire?');
writeln('A — a listing of all actual correlation functions');
writeln('B — a listing of Threshold functions');
writeln('N — None, You want to leave this mode');
readln(ch)
until (ch = 'n') or (ch = 'N');
DisposeOf(List2);
DisposeOf(List1);
GetRidOf(DataList);
writeln('do you wish to do any more correlations?..Y/N');
readln(ch)
until (ch = 'N') or (ch = 'n')
end (*if y/n*)
end. [ PartialCorrelate ]

```

```

(*****
(*)
(*)           P h a s e C o r r e l a t e
(*)
(*****
(*) DESIGNER:  Richard C. Gonder           MASTER THESIS WORK
(*) ADVISOR:  MAJ Ken Castor             MODULE NUMBER D.1
(*) DESIGN DATE: Jan 1985                LAST UPDATE: 1 Apr 1985
(*)
(*) DESCRIPTION:  This program will perform correlations on user designa-
(*) code files composed of ASCII 1's and 0's, and determine the minimum
(*) and maximum values of the Odd correlation function for L different
(*) phases of the code file. For each phase, this program will shift
(*) both code files and compute the Odd function. It then determines the
(*) minimum and maximum values of Odd function for that phase, and places
(*) this information into a list. Once all the phases have been cycled
(*) through, a listing of all of the phases where the maximum and minimum
(*) spreads of the Odd function, as well as their max and min values, is
(*) written to a user designated file. It will perform off-peak autocor-
(*) relation and total cross-correlation of codes as determined by user
(*) input.
(*)
(*) CALLING ARGUMENTS: Std Input, Output, Designated Text File
(*)
(*) INTERNAL VARIABLES:
(*)     J—used as counter for control
(*)     N—Length of codefiles in characters
(*)     Phasemax,Phasemin—Value of Maximum and Minimum spread for
(*)                       for Odd function for all phase shifts
(*)     Look,DataList,Back—pointers used to create/manipulate data
(*)                       lists to obtain outputted data file
(*)     Resultfile,Codedta—File variable for file I/O
(*)     Result,Filenal,Filena2: string for filename with extension
(*)     ch—character input to the program from user
(*)     FlagA—boolean used to control process
(*)     List1,List2 used to identify the beginning of linked lists
(*)                       representing the codes to be correlated
(*)     K,S—used to set correlation process for off-peak-autocorrel-
(*)          ation or complete-cross-correlation
(*)
(*) CONSTANTS: None
(*)
(*) FILES USED:  User designated Filename for code readin and data output
(*)
(*) MODULES CALLED: CreateList
(*)                  ShiftList
(*)                  Correlate
(*)                  DisposeOf
(*)                  GetRidOf
(*)                  Maximimize
(*)

```

```

program PhaseCorrelate(input,output);

type Name = string[12];
   pointer    = ^datarecord;
   datarecord = record
       data : char;
       next : pointer;
       end; (*of record*)
   dataptr    = ^info;
   info       = record
       MinOdd : real;
       MaxOdd : real;
       next1 : dataptr;
       end (*record*);

var K,J,N,S: integer;
    List1,List2: pointer;
    Look,DataList,Back: dataptr;
    Result,Filena1,Filena2: Name;
    Resultfile,Codedta: Text;
    Phasemax,Phasemin: real;
    ch: char;
    FlagA: boolean;

```

```

(*****
*)
*)           C r e a t e L i s t
*)
(*****
*) DESIGNER:  Richard C. Gonder           MASTER THESIS WORK
*) ADVISOR:   MAJ Ken Castor             MODULE NUMBER D.2
*) DESIGN DATE: Jan 1985                 LAST UPDATE: 4 Jan 1985
*)
*) DESCRIPTION: This procedure will read in a file of char, placing each*)
*) character into a linked list. It terminates at the EOF of the file *)
*) read.
*)
*) CALLING ARGUMENTS: Filename—string of char
*)                      List—pointer to the beginning of linked list
*)                      K—number of characters read into list
*)
*) INTERNAL VARIABLES:
*)      P,Ptr—used to construct list
*)      bit—temp location for character read
*)
*) CONSTANTS: None
*)
*) FILES USED:  User designated Filename for code storage
*)
*) MODULES CALLED:  None
*)
*)
*) procedure CreateList(Filename: Name; var List: pointer; var K: integer);
*)
*)   var
*)     P, Ptr: pointer;
*)     bit: char;
*)
*)   begin
*)     assign(Codedta,Filename);
*)     reset(Codedta);
*)     new(P);
*)     List := P;
*)     Ptr := P;
*)     K := 0;
*)     while not eof(Codedta) do begin
*)       read(Codedta, bit);
*)       K := K + 1;
*)       P^.data := bit;
*)       Ptr^.next := P;
*)       Ptr := P;
*)       new(P)
*)     end; (*while loop*)
*)     close(Codedta);
*)     Ptr^.next:= nil;
*)   end; (*procedure CreateList*)

```

```

(*****
(*)
(*)           E n q u e
(*)
(*****
(*) DESIGNER:  Richard C. Gonder           MASTER THESIS WORK
(*) ADVISOR:  MAJ Ken Castor             MODULE NUMBER D.4.1
(*) DESIGN DATE: Nov 1984                LAST UPDATE: 14 Dec 1984
(*)
(*) DESCRIPTION:  This procedure enques a data record representing the
(*) values of the Minimum and Maximum Odd values for a phase of the Odd
(*) correlation function. The list is ordered by phase, from 0 to L.
(*)
(*) CALLING ARGUMENTS: List,Back—pointers to front and rear of list
(*)                      MinOdd,MaxOdd—min and max values of Odd function
(*)                      for a particular phase
(*)
(*) INTERNAL VARIABLES:
(*)      P—temp pointer used to do the process
(*)
(*) CONSTANTS: None
(*)
(*) FILES USED:  None
(*)
(*) MODULES CALLED:  None
(*)
procedure Enque(var List, Back:dataptr; MinOdd, MaxOdd:real);

var P: dataptr;

begin
  if List = nil then begin
    New(P);
    List:= P;
    P^.MaxOdd:= MaxOdd;
    P^.MinOdd:= MinOdd;
    Back:= P;
    Back^.nextl:= nil;
  end
  else begin
    New(P);
    P^.MaxOdd:= MaxOdd;
    P^.MinOdd:= MinOdd;
    Back^.nextl:= P;
    Back:= P;
    Back^.nextl:= nil;
  end (*if then else*)
end; (*procedure Enque*)

```

```

(*****
(*)
(*)           D i s p o s e O f           (*)
(*)
(*****
(*) DESIGNER:  Richard C. Gonder           MASTER THESIS WORK           (*)
(*) ADVISOR:  MAJ Ken Castor             MODULE NUMBER D.5           (*)
(*) DESIGN DATE: Jan 1985                 LAST UPDATE: 14 Jan 1985       (*)
(*)-----(*)
(*) DESCRIPTION: This procedure releases a linked list from memory (*)
(*) allocation.                               (*)
(*)-----(*)
(*) CALLING ARGUMENTS: List--pointers to front of list (*)
(*)-----(*)
(*) INTERNAL VARIABLES: (*)
(*)      P,Ptr--temp pointer used to do the process (*)
(*)-----(*)
(*) CONSTANTS: None (*)
(*)-----(*)
(*) FILES USED:  None (*)
(*)-----(*)
(*) MODULES CALLED:  None (*)
(*)-----*)

```

```

procedure DisposeOf(var List:pointer);

```

```

    var P,Ptr:pointer;

```

```

    begin

```

```

        P:= List;

```

```

        Ptr:=P;

```

```

        while P^.next <> nil do begin

```

```

            Ptr:= P^.next;

```

```

            dispose(P);

```

```

            P:=Ptr;

```

```

        end (*while loop*);

```

```

        dispose(P);

```

```

        List:= nil;

```

```

    end;(*procedure DisposeOf*)

```

```

(*****)
(*)
(*)          G e t R i d O f          (*)
(*)
(*****)
(*) DESIGNER:  Richard C. Gonder          MASTER THESIS WORK          (*)
(*) ADVISOR:  MAJ Ken Castor             MODULE NUMBER D.6          (*)
(*) DESIGN DATE: Jan 1985                LAST UPDATE: 14 Jan 1985    (*)
(*)-----(*)
(*) DESCRIPTION:  This procedure releases a linked list from memory (*)
(*) allocation.                                     (*)
(*)-----(*)
(*) CALLING ARGUMENTS: List--pointers to front of list          (*)
(*)-----(*)
(*) INTERNAL VARIABLES:                                          (*)
(*)      P,Ptr--temp pointer used to do the process            (*)
(*)-----(*)
(*) CONSTANTS: None                                             (*)
(*)-----(*)
(*) FILES USED:  None                                           (*)
(*)-----(*)
(*) MODULES CALLED:  None                                       (*)
(*)-----(*)

```

```

procedure GetRidOf(List:dataptr);

```

```

    var P,Ptr:dataptr;

```

```

    begin

```

```

        P:= List;

```

```

        Ptr:=P;

```

```

        while P^.next1 <> nil do begin

```

```

            Ptr:= P^.next1;

```

```

            dispose(P);

```

```

            P:=Ptr;

```

```

        end (*while loop*);

```

```

        dispose(P);

```

```

        List:= nil;

```

```

    end;(*procedure GetRidOf*)

```

```

(*****)
(*)
(*)           S h i f t L i s t
(*)
(*****)
(*) DESIGNER:  Richard C. Gonder           MASTER THESIS WORK
(*) ADVISOR:  MAJ Ken Castor             MODULE NUMBER D.3
(*) DESIGN DATE: Jan 1985                LAST UPDATE: 14 Jan 1985
(*)
(*) DESCRIPTION:  This procedure merely takes the first record of the
(*) Linked list and places it at the end of the list.
(*)
(*) CALLING ARGUMENTS: List--pointers to front of list
(*)
(*) INTERNAL VARIABLES:
(*)      P,First--temp pointer used to do the process
(*)
(*) CONSTANTS: None
(*)
(*) FILES USED:  None
(*)
(*) MODULES CALLED:  None
(*)

```

```

procedure ShiftList(var List:pointer);

```

```

    var First,P:pointer;

```

```

    begin

```

```

        First:= List;

```

```

        P:= List;

```

```

        while P^.next <> nil do

```

```

            P:= P^.next;

```

```

        First:= First^.next;

```

```

        P^.next:= List;

```

```

        List^.next:= nil;

```

```

        List:= First;

```

```

    end; (*procedure ShiftList*)

```

```

(*****
(*)
(*)           M i n i M a x           (*)
(*)
(*****
(*) DESIGNER:  Richard C. Gonder          MASTER THESIS WORK      (*)
(*) ADVISOR:  MAJ Ken Castor             MODULE NUMBER D.4.2      (*)
(*) DESIGN DATE: Jan 1985                LAST UPDATE: 14 Jan 1985 (*)
(*)
(*) DESCRIPTION:  This procedure merely takes in the values min, max and (*)
(*) Temp, and sets Min or Max to Temp when it is less than Min or greater*)
(*) than Max.                                     (*)
(*)
(*) CALLING ARGUMENTS: Temp--input value (*)
(*)           Min--input set to temp when temp is less than Min (*)
(*)           Max--input set to temp when temp is greater than (*)
(*)
(*) INTERNAL VARIABLES:  None (*)
(*)
(*) CONSTANTS: None (*)
(*)
(*) FILES USED:  None (*)
(*)
(*) MODULES CALLED:  None (*)
(*)

```

```

procedure Minimax(Temp:real;var Min,Max:real);

```

```

begin
  if Temp < Min then
    Min:= Temp;
  if Temp > Max then
    Max:= Temp;
end; (*procedure Minimax*)

```

```

(*****)
(*)
(*)           C o r r e l a t e
(*)
(*****)
(*) DESIGNER:  Richard C. Gonder           MASTER THESIS WORK
(*) ADVISOR:  MAJ Ken Castor             MODULE NUMBER D.4
(*) DESIGN DATE: Jan 1985                LAST UPDATE: 14 Jan 1985
(*)
(*) DESCRIPTION: This procedure takes the 2 Linked Lists of characters
(*) representing codesets and determines the odd correlation function at
(*) discrete values. It determines the maximum and minimum Odd function
(*) values and places these values in a linked list of records.
(*)
(*) CALLING ARGUMENTS: Start,Stop—values setting region over which cor-
(*) relation is performed.
(*) List1,List2—pointers for front of codelists
(*) N—Length of List
(*)
(*) INTERNAL VARIABLES:
(*) Ptr1,Ptr2—temp pointer used to do the process
(*) LeftCor, RightCor—partial correlations values used to de-
(*) termine odd correlation value
(*) Odd—discrete correlation values
(*) Agree,Disagree—used as counters during correlation process
(*) Count,K—counters used in process
(*) MinOdd,MaxOdd—represent min/max Odd value for each correl-
(*) ation phase
(*)
(*) CONSTANTS: None
(*)
(*) FILES USED:  None
(*)
(*) MODULES CALLED:  Enque
(*)                  Minimax
(*)

```

```

procedure Correlate(List1:pointer; List2:pointer; N:integer;
                    var Start,Stop:integer);

```

```

var MinOdd,MaxOdd,Leftcor,Rightcor,Odd,:real;
    Count,Agree,DisAgree,K:integer;
    Ptr1,Ptr2:pointer;

```

```

begin
  MinOdd:= N;
  MaxOdd:= -N;
  while Start <= Stop do begin
    Ptr1:= List1;
    Ptr2:= List2;
    Agree:= 0;
    DisAgree:= 0;

```

```

for Count:= 2 to Start do
  Ptr1:= Ptr1^.next;
while Ptr1 <> nil do begin
  if Ptr1^.data = Ptr2^.data then
    Agree:= Agree + 1
  else DisAgree:= DisAgree + 1;
  Ptr1:=Ptr1^.next;
  Ptr2:=Ptr2^.next;
end;(*while Loop*)
RightCor:= (Agree-DisAgree);
Agree:= 0;
DisAgree:= 0;
Ptr1:= List1;
while Ptr2 <> nil do begin
  if Ptr1^.data = Ptr2^.data then
    Agree:= Agree + 1
  else DisAgree:= DisAgree + 1;
  Ptr1:=Ptr1^.next;
  Ptr2:=Ptr2^.next;
end;(*while Loop*)
LeftCor:= (Agree-DisAgree);
Odd:= RightCor - LeftCor ;
Minimax(Odd,MinOdd,MaxOdd);
Start:=Start + 1;
end; (*while loop*)
Enque(DataList,Back,MinOdd,MaxOdd);
end;(*procedure correlate)

```

```

(*****
(*)
(*)           M a x i m i n i m i z e
(*)
(*****
(*) DESIGNER:  Richard C. Gonder           MASTER THESIS WORK
(*) ADVISOR:  MAJ Ken Castor             MODULE NUMBER D.7
(*) DESIGN DATE: Jan 1985                LAST UPDATE: 12 Feb 1985
(*)-----
(*) DESCRIPTION: This procedure takes a linked list of min and max values*)
(*) representing each phase shifted correlation function, and determines *)
(*) the maximum and minimum difference for all phases. It will write *)
(*) these values to the screen.
(*)-----
(*) CALLING ARGUMENTS: DataList—pointer to front of list of records
(*)                      Phasemax,Phasemin—output value of Minumum and
(*)                      maximum spread.
(*)                      N—Length of List
(*)-----
(*) INTERNAL VARIABLES:
(*)                      Ptr—temp pointer used to do the process
(*)-----
(*) CONSTANTS: None
(*)-----
(*) FILES USED:  None
(*)-----
(*) MODULES CALLED:  None
(*)-----

```

```

procedure Maximimize(DataList:dataptr; var Phasemax, Phasemin:real;
N:integer);

```

```

var  Ptr: dataptr;

```

```

begin

```

```

    Phasemax:= -N;

```

```

    Phasemin:= N;

```

```

    Ptr:=datalist;

```

```

    while Ptr^.nextl <> nil do begin

```

```

        if (Ptr^.MaxOdd -Ptr^.MinOdd) > Phasemax then

```

```

            Phasemax:= Ptr^.MaxOdd-Ptr^.MinOdd;

```

```

        if (Ptr^.MaxOdd -Ptr^.MinOdd) < Phasemin then

```

```

            Phasemin:= Ptr^.MaxOdd-Ptr^.MinOdd;

```

```

        Ptr:= Ptr^.nextl;

```

```

    end;(*while loop*)

```

```

    writeln('min spread is ',Phasemin:5:3,' max spread is ',
            Phasemax:5:3);

```

```

end; (*procedure MaxiMinimize*)

```

```

(*****
(*)
(*)          M A I N   P R O G R A M          (*)
(*)
(*)
(*****)

```

```
begin
```

```

  FlagA:= false;
  writeln('Select  A--for off peak minimax of autocorrelation');
  writeln('Select  B--for crosscorrelation minimax procedure');
  readln(ch);
  DataList:= nil;
  Back:= nil;
  case ch of
    'A' , 'a' :begin
      FlagA:= true;
      writeln('Autocorrelation selected--enter filename of
              code');
      readln(Filenal);
      CreateList(filenal,List1,N);
      CreateList(filenal,List2,N);
      writeln('enter filename where you would like to store
              results');
      readln(Result);
      ClrScr;
      GoToXY(22,14);
      write('WORKING ON PHASE SHIFT OF ');
      for J:= 0 to N-1 do begin
        GoToXY(48,14);
        write(J);
        K:= 2;
        S:= N-1;
        Correlate(List1,List2,N,K,N);
        ShiftList(List1);
        ShiftList(List2);
      end;(*for loop*)
    end;

```

```
  'B' , 'b' :begin
```

```

    FlagA:= true;
    writeln('Cross correlation selected..Enter the first
            code filename');
    readln(filenal);
    writeln('Now enter the second filename');
    readln(filena2);
    writeln('enter filename where you would like to store
            results');
    readln(Result);
    CreateList(filenal,List1,N);
    CreateList(filena2,List2,N);
    ClrScr;
    GoToXY(22,14);
    WRITE('WORKING ON PHASE SHIFT OF ');
    for J:= 0 to N-1 do begin

```

```

        GoToXY(48,14);
        write(J);
        K:= 1;
        S:= N;
        Correlate(List1,List2,N,K,S);
        ShiftList(List1);
        ShiftList(List2);
        end;(*for loop*)
    end; (*B select*)
end;(* case*)
writeln;writeln;
if flagA then begin
    DisposeOf(List2);
    DisposeOf(List1);
    assign(Resultfile,Result);
    rewrite(Resultfile);
    Maximimize(Datalist,Phasemax,Phasemin,N);
    Look:=Datalist;
    J:=0;
    while Look <> nil do begin
        if Look^.MaxOdd - Look^.MinOdd = Phasemax then
            writeln(Resultfile, 'phasemax ',j,' ',Look^.MinOdd:5:2,' to ',
                Look^.MaxOdd:5:2);
        if Look^.MaxOdd - Look^.MinOdd = Phasemin then
            writeln(Resultfile, 'phasemin ',j,' ',Look^.MinOdd:5:2,' to ',
                Look^.MaxOdd:5:2);
        Look:=Look^.next1;
        J:= J+1;
    end(*while loop*);
    close(Resultfile);
end (*if flagA*)
end.

```

```

(*****)
(*)
(*)          C o u n t L o w e r          (*)
(*)
(*****)
(*) DESIGNER:  Richard C. Gonder          MASTER THESIS WORK          (*)
(*) ADVISOR:   MAJ Ken Castor             MODULE NUMBER E.6          (*)
(*) DESIGN DATE: Jan 1985                 LAST UPDATE: 26 Jan 1985    (*)
(*)-----(*)
(*) DESCRIPTION: This procedure takes a data linked list representing the (*)
(*) the Odd and Even correlation functions, and determines what percent- (*)
(*) age of these functions lie below the inputted threshold.          (*)
(*)-----(*)
(*) CALLING ARGUMENTS: List—pointer to front of datalist          (*)
(*)                      Thresh—threshold value                  (*)
(*)                      Count1,Count2—used to add amount of each function (*)
(*)                      that exceeds the threshold              (*)
(*)-----(*)
(*) INTERNAL VARIABLES:                                          (*)
(*)      Ptr1,Ptr2—temp pointer used to do the process          (*)
(*)      R—temp storage location used with counter              (*)
(*)-----(*)
(*) CONSTANTS: None                                          (*)
(*)-----(*)
(*) FILES USED:  None                                          (*)
(*)-----(*)
(*) MODULES CALLED:  None                                          (*)
(*)-----(*)

```

```

procedure CountLower(List: dataptr; Thresh: real; var Count1,
                                     Count2: real);

```

```

var

```

```

    Ptr1, Ptr2: dataptr;

```

```

    R: real;

```

```

begin

```

```

    Count1 := 0;

```

```

    Count2 := 0;

```

```

    Ptr1 := List;

```

```

    repeat

```

```

        Ptr2 := Ptr1^.Next1;

```

```

        if (-Ptr1^.Even < Thresh) and (-Ptr2^.Even >= Thresh) then begin

```

```

            R := (-Ptr2^.Even - Thresh) / (-Ptr2^.Even + Ptr1^.Even);

```

```

            Count1 := Count1 + R

```

```

        end; (*if*)

```

```

        if (-Ptr1^.Even >= Thresh) and (-Ptr2^.Even >= Thresh) then begin

```

```

            Count1 := Count1 + 1

```

```

        end; (*if*)

```

```

        if (-Ptr1^.Even >= Thresh) and (-Ptr2^.Even < Thresh) then begin

```

```

            R := (-Ptr1^.Even - Thresh) / (-Ptr1^.Even + Ptr2^.Even);

```

```

            Count1 := Count1 + R

```

```

        end;

```

```

if (Ptr1^.Odd < Thresh) and (Ptr2^.Odd >= Thresh) then begin
  R := (Ptr2^.Odd - Thresh) / (Ptr2^.Odd - Ptr1^.Odd);
  Count2 := Count2 + R
end; (*if*)
if (Ptr1^.Odd >= Thresh) and (Ptr2^.Odd >= Thresh) then begin
  Count2 := Count2 + 1
end; (*if*)
if (Ptr1^.Odd >= Thresh) and (Ptr2^.Odd < Thresh) then begin
  R := (Ptr1^.Odd - Thresh) / (Ptr1^.Odd - Ptr2^.Odd);
  Count2 := Count2 + R
end;
Ptr1 := Ptr2;
until Ptr1^.Next1 = nil
end; (*procedure CountUpper*)

```

```

(******)
(*)
(*)          C o u n t U p p e r          (*)
(*)
(******)
(*) DESIGNER:  Richard C. Gonder          MASTER THESIS WORK          (*)
(*) ADVISOR:  MAJ Ken Castor             MODULE NUMBER E.5          (*)
(*) DESIGN DATE: Jan 1985                LAST UPDATE: 26 Jan 1985    (*)
(*)-----(*)
(*) DESCRIPTION: This procedure takes a data linked list representing the*)
(*) the Even and Odd correlation functions, and determines what amount  *)
(*) of these functions lie above the inputted threshold.                  *)
(*)-----(*)
(*) CALLING ARGUMENTS: List--pointer to front of datalist                *)
(*)                      Thresh--threshold value                        *)
(*)                      Count1,Count2--used to add amount of curve that  *)
(*)                      exceeds the threshold                           *)
(*)-----(*)
(*) INTERNAL VARIABLES:                                                  (*)
(*)      Ptr1,Ptr2--temp pointer used to do the process                  *)
(*)      R--temp storage location used with counter                      *)
(*)-----(*)
(*) CONSTANTS: None                                                    (*)
(*)-----(*)
(*) FILES USED:  None                                                  (*)
(*)-----(*)
(*) MODULES CALLED:  None                                              (*)
(*)-----*)

```

```

procedure CountUpper(List: dataptr; Thresh: real; var Count1,
                                Count2: real);

```

```

var

```

```

    Ptr1, Ptr2: dataptr;
    R: real;

```

```

begin

```

```

    Count1 := 0;

```

```

    Count2 := 0;

```

```

    Ptr1 := List;

```

```

    repeat

```

```

        Ptr2 := Ptr1^.Next1;

```

```

        if (Ptr1^.Even < Thresh) and (Ptr2^.Even >= Thresh) then begin

```

```

            R := (Ptr2^.Even - Thresh) / (Ptr2^.Even - Ptr1^.Even);

```

```

            Count1 := Count1 + R

```

```

        end; (*if*)

```

```

        if (Ptr1^.Even >= Thresh) and (Ptr2^.Even >= Thresh) then begin

```

```

            Count1 := Count1 + 1

```

```

        end; (*if*)

```

```

        if (Ptr1^.Even >= Thresh) and (Ptr2^.Even < Thresh) then begin

```

```

            R := (Ptr1^.Even - Thresh) / (Ptr1^.Even - Ptr2^.Even);

```

```

            Count1 := Count1 + R

```

```

        end;

```

```

        DisAgree := DisAgree + 1;
        Ptr1 := Ptr1^.Next;
        Ptr2 := Ptr2^.Next
    end; (*while Loop*)
    RightCor := Agree - DisAgree;
    Agree := 0;
    DisAgree := 0;
    Ptr1 := List1;
    while Ptr2 <> nil do begin
        if Ptr1^.data = Ptr2^.data then
            Agree := Agree + 1
        else
            DisAgree := DisAgree + 1;
            Ptr1 := Ptr1^.Next;
            Ptr2 := Ptr2^.Next
        end; (*while Loop*)
        LeftCor := Agree - DisAgree;
        Even := LeftCor + RightCor;
        Odd := RightCor - LeftCor;
        Enque(DataList, Back, Odd, Even);
        K := K + 1;
    end; (*while loop*)
    Agree := 0;
    DisAgree := 0;
    Ptr1 := List1;
    Ptr2 := List2;
    while Ptr2 <> nil do begin
        if Ptr1^.data = Ptr2^.data then
            Agree := Agree + 1
        else
            DisAgree := DisAgree + 1;
            Ptr1 := Ptr1^.next;
            Ptr2 := Ptr2^.next;
        end; (*while loop*)
        LeftCor := Agree - DisAgree;
        Even := LeftCor;
        Odd := -LeftCor;
        Enque(DataList, Back, Odd, Even)
    end; (*procedure correlate*)

```

```

(*****
*)
*)           C o r r e l a t e
*)
*)
(*****
*) DESIGNER:  Richard C. Gonder          MASTER THESIS WORK
*) ADVISOR:  MAJ Ken Castor             MODULE NUMBER E.1
*) DESIGN DATE: Jan 1985                LAST UPDATE: 14 Jan 1985
*)
*) DESCRIPTION:  This procedure takes the 2 Linked Lists of characters
*) representing codesets and determines the Even and Odd correlation
*) functions at discrete values.  It places the determined values at
*) corresponding positions in a third linked list.
*)
*) CALLING ARGUMENTS:  DataList,Back--pointers to front and rear of output
*)                      Datalist
*)                      List1,List2--pointers for front of codelists
*)                      N--Length of List
*)
*) INTERNAL VARIABLES:
*)      Ptr1,Ptr2--temp pointer used to do the process
*)      LeftCor, RightCor--partial corelations values used to de-
*)                      termine the Even and Odd values
*)      Even,Odd,--discrete correlation values
*)      Agree,Disagree--used as counters during correlation process
*)      Count,K--counters used in process
*)
*) CONSTANTS:  None
*)
*) FILES USED:  None
*)
*) MODULES CALLED:  Enque
*)
*)
*) procedure Correlate(var DataList,Back:dataptr;List1,List2:pointer;
*)                      N:integer);
*)
*) var
*)     LeftCor, RightCor, Even, Odd: real;
*)     Count, Agree, DisAgree, K: integer;
*)     Ptr1, Ptr2: pointer;
*)
*) begin
*)     K := 1;
*)     while K <= N do begin
*)         Ptr1 := List1;
*)         Ptr2 := List2;
*)         Agree := 0;
*)         DisAgree := 0;
*)         for Count := 2 to K do
*)             Ptr1 := Ptr1^.Next;
*)         while Ptr1 <> nil do begin
*)             if Ptr1^.data = Ptr2^.data then
*)                 Agree := Agree + 1
*)             else

```

```

(*****)
(*)
(*)           S h i f t L i s t           (*)
(*)
(*****)
(*) DESIGNER:  Richard C. Gonder           MASTER THESIS WORK (*)
(*) ADVISOR:  MAJ Ken Castor             MODULE NUMBER E.3 (*)
(*) DESIGN DATE: Jan 1985                 LAST UPDATE: 14 Jan 1985 (*)
(*)-----(*)
(*) DESCRIPTION: This procedure merely takes the first record of the (*)
(*) linked list and places it at the end of the list. (*)
(*)-----(*)
(*) CALLING ARGUMENTS: List--pointers to front of list (*)
(*)-----(*)
(*) INTERNAL VARIABLES: (*)
(*) P,Ptr--temp pointer sed to do the process (*)
(*)-----(*)
(*) CONSTANTS: None (*)
(*)-----(*)
(*) FILES USED: None (*)
(*)-----(*)
(*) MODULES CALLED: None (*)
(*)-----(*)

```

```

procedure ShiftList(var List: pointer);

```

```

var
    First, P: pointer;

begin
    First := List;
    P := List;
    while P^.Next <> nil do
        P := P^.Next;
    First := First^.Next;
    P^.Next := List;
    List^.Next := nil;
    List := First
end; (*procedure ShiftList*)

```

```

(*****)
(*)
(*)           G e t R i d O f           (*)
(*)
(*****)
(*) DESIGNER:  Richard C. Gonder           MASTER THESIS WORK (*)
(*) ADVISOR:  MAJ Ken Castor             MODULE NUMBER E.8 (*)
(*) DESIGN DATE: Jan 1985                 LAST UPDATE: 14 Jan 1985 (*)
(*)-----(*)
(*) DESCRIPTION: This procedure releases a linked list from memory (*)
(*) allocation. Specifically used on input data lists destruction. (*)
(*)-----(*)
(*) CALLING ARGUMENTS: List--pointers to front of list (*)
(*)-----(*)
(*) INTERNAL VARIABLES: P,Ptr--pointers used to do process (*)
(*)-----(*)
(*) CONSTANTS: None (*)
(*)-----(*)
(*) FILES USED: None (*)
(*)-----(*)
(*) MODULES CALLED: None (*)
(*)-----(*)

```

```

procedure GetRidOf(var List: dataptr);

```

```

var P,Ptr: dataptr;

```

```

begin
  P:= List;
  Ptr:= P;
  while P^.nextl <> nil do begin
    Ptr:= P^.nextl;
    dispose(P);
    P:= Ptr;
  end; (*while loop*)
  dispose(P);
  List:= nil;
end; (*procedure GetRidOf*)

```

```

(*****)
(*)
(*)           D i s p o s e O f           (*)
(*)
(*****)
(*) DESIGNER:  Richard C. Gonder           MASTER THESIS WORK (*)
(*) ADVISOR:  MAJ Ken Castor              MODULE NUMBER E.7   (*)
(*) DESIGN DATE: Jan 1985                 LAST UPDATE: 14 Jan 1985 (*)
(*)-----(*)
(*) DESCRIPTION: This procedure releases a linked list from memory (*)
(*) allocation. Specifically used on input data lists destruction. (*)
(*)-----(*)
(*) CALLING ARGUMENTS: List—pointers to front of list (*)
(*)-----(*)
(*) INTERNAL VARIABLES: P,Ptr—pointers used to do process (*)
(*)-----(*)
(*) CONSTANTS: None (*)
(*)-----(*)
(*) FILES USED: None (*)
(*)-----(*)
(*) MODULES CALLED: None (*)
(*)-----(*)

```

```

procedure DisposeOf(var List: pointer);

```

```

var P,Ptr: pointer;

```

```

begin

```

```

    P:= List;

```

```

    Ptr:= P;

```

```

    while P^.next <> nil do begin

```

```

        Ptr:= P^.next;

```

```

        dispose(P);

```

```

        P:= Ptr;

```

```

    end; (*while loop*)

```

```

    dispose(P);

```

```

    List:= nil;

```

```

end; (*procedure DisposeOf*)

```

```

while( P <> nil) and (ABS(P^.thresh - thresh) > 0.0005 ) do begin
  Ptr:= P;
  P:= P^.Next;
end; (*while loop*)
if P = nil then begin (*record not found*)
  new(P);
  Ptr^.Next:= P;
  if Number > 1 then begin
    P^.mineven:= 0.0;
    P^.aveeven:= PctEven/Number;
    P^.minodd:= 0.0;
    P^.aveodd:= PctOdd/Number;
  end
  else begin
    P^.mineven:= PctEven;
    P^.aveeven:=PctEven;
    P^.minodd:= PctOdd;
    P^.aveodd:=PctOdd;
  end;(*if Number then else*)
  P^.maxeven:= PctEven;
  P^.maxodd:= PctOdd;
  P^.thresh:= thresh;
  P^.Next := nil
end (*front clause of if*)
else begin
  Ptr:= P;
  if PctEven < P^.mineven then P^.mineven:= PctEven;
  if PctEven > P^.maxeven then P^.maxeven:= PctEven;
  P^.aveeven := ((P^.aveeven * (Number-1) + PctEven)/Number);
  if PctOdd < P^.minodd then P^.minodd:= PctOdd;
  if PctOdd > P^.maxodd then P^.maxodd:= PctOdd;
  P^.aveodd := ((P^.aveodd * (Number-1) + PctOdd)/Number);
  end (*if then else*)
end (*if then else*)
end; (*UpdateData*)

```

```

(*****
(*)
(*)           U p d a t e D a t a L i s t
(*)
(*****
(*) DESIGNER:  Richard C. Gonder           MASTER THESIS WORK
(*) ADVISOR:  MAJ Ken Castor             MODULE NUMBER E.10
(*) DESIGN DATE: Mar 1985                LAST UPDATE: 4 Apr 1985
(*)
(*) DESCRIPTION:  This procedure takes the values of Threshold, PctEven
(*) and PctOdd from the thresholding process. It finds the record that
(*) has stored Minimum, Maximum and Average values for a specific thresh-
(*) old value, for both the Odd and Even functions, and updates these
(*) fields with the appropriate info. If a record is not found, it cre-
(*) ates the record and places appropriate initial values in the fields.
(*)
(*) CALLING ARGUMENTS: List—pointer to front of list
(*)                      Even,Odd—real values of correlation functions
(*)                      Thresh—value of threshold
(*)                      Number—Number of thresholding processes already
(*)                      accomplished
(*)
(*) INTERNAL VARIABLES:
(*)                      Ptr,P—temp pointers used to do the process
(*)
(*) CONSTANTS: None
(*)
(*) FILES USED:  None
(*)
(*) MODULES CALLED:  None
(*)
(*****

```

```

procedure UpdateDataList(var List : Pctptr; Number:integer; PctEven,
                        PctOdd, Thresh: real);

```

```

var
  Ptr,P: Pctptr;

begin
  P:= List;
  if List = nil then begin
    new(P);
    List := P;
    P^.mineven:= PctEven;
    P^.maxeven:= PctEven;
    P^.aveeven:= PctEven;
    P^.minodd:= PctOdd;
    P^.maxodd:= PctOdd;
    P^.aveodd:= PctOdd;
    P^.thresh:= thresh;
    P^.Next := nil
  end (*first clause*)
  else begin

```

```

(*****)
(*)
(*)           E n q u e
(*)
(*****)
(*) DESIGNER:  Richard C. Gonder           MASTER THESIS WORK
(*) ADVISOR:  MAJ Ken Castor             MODULE NUMBER E.11
(*) DESIGN DATE: Nov 1984                 LAST UPDATE: 14 Dec 1984
(*)
(*) DESCRIPTION: This procedure enques a data record representing the
(*) values of the Odd and Even correlation functions at a discrete
(*) value in an ordered linked list.
(*)
(*) CALLING ARGUMENTS: List,Back—pointers to front and rear of list
(*) Even,Odd—values of correlation functions
(*)
(*) INTERNAL VARIABLES:
(*) P—temp pointer used to do the process
(*)
(*) CONSTANTS: None
(*)
(*) FILES USED: None
(*)
(*) MODULES CALLED: None
(*)

```

```

procedure Enque(var List, Back: dataptr; Odd, Even: real);

```

```

var

```

```

    P: dataptr;

```

```

begin

```

```

    if List = nil then begin

```

```

        new(P);

```

```

        List := P;

```

```

        P^.Even := Even;

```

```

        P^.Odd := Odd;

```

```

        Back := P;

```

```

        Back^.Nextl := nil

```

```

    end else begin

```

```

        new(P);

```

```

        P^.Even := Even;

```

```

        P^.Odd := Odd;

```

```

        Back^.Nextl := P;

```

```

        Back := P;

```

```

        Back^.Nextl := nil

```

```

    end (*if then else*)

```

```

end; (*procedure Enque*)

```

```

(*****)
(*)
(*)          C r e a t e L i s t          (*)
(*)
(*****)
(*) DESIGNER:  Richard C. Gonder          MASTER THESIS WORK          (*)
(*) ADVISOR:  MAJ Ken Castor             MODULE NUMBER E.2          (*)
(*) DESIGN DATE: Jan 1985                LAST UPDATE: 4 Jan 1985      (*)
(*)-----(*)
(*) DESCRIPTION: This procedure will read in a file of char, placing each*)
(*) character into a linked list. It terminates at the EOF of the file *)
(*) read.                                     (*)
(*)-----(*)
(*) CALLING ARGUMENTS: Filename--string of char          (*)
(*)                  List--pointer to the beginning of linked list (*)
(*)                  K--number of characters read into list (*)
(*)-----(*)
(*) INTERNAL VARIABLES:                                  (*)
(*)      P,ptr--used to construct list                    (*)
(*)      bit--temp location for character read            (*)
(*)-----(*)
(*) CONSTANTS: None                                       (*)
(*)-----(*)
(*) FILES USED:  User designated Filename for code storage (*)
(*)-----(*)
(*) MODULES CALLED:  None                                  (*)
(*)-----(*)

```

```

procedure CreateList(Filename: Name; var List: pointer; var K: integer);

```

```

var

```

```

    P, ptr: pointer;
    bit: char;

```

```

begin

```

```

    assign(Codedta,Filename);
    reset(Codedta);
    new(P);
    List := P;
    ptr := P;
    K := 0;
    while not eof(Codedta) do begin
        read(Codedta, bit);
        K := K + 1;
        P^.data := bit;
        ptr^.Next := P;
        ptr := P;
        new(P)
    end; (*while loop*)
    close(Codedta);
    dispose(P);
    ptr^.Next:= nil;
end; (*procedure CreateList*)

```

```

program MassCorrelate(input, output);

type
  Name = string[14];
  St3 = string[3];
  St10 = string[10];
  pointer = ^ datarecord;
  datarecord =
    record
      data: char;
      Next: pointer
    end; (*of record*)
  dataptr = ^ info;
  info =
    record
      Even: real;
      Odd: real;
      Next1: dataptr
    end; (*record*)

  Pctptr = ^ infol;
  infol =
    record
      thresh: real;
      mineven: real;
      maxeven: real;
      aveeven: real;
      minodd: real;
      maxodd: real;
      aveodd: real;
      Next: Pctptr
    end; (*record*)

var
  Number, Outside, Inside, Startfile, StopFile, Phase, J, N: integer;
  List1, List2: pointer;
  DataList, Back: dataptr;
  PctList: Pctptr;
  Result, Filen1, Filen2: Name;
  St1, St2: St3;
  CodeName: St10;
  Resultfile, Codedta: text;
  PctEven, PctOdd, S,
  Sum1even, Sum2even, Sum1odd, Sum2odd, Threshold: real;
  ch: char;
  Stop: boolean;

```

```

(*****
(*)
(*)           M a s s C o r r e l a t e
(*)
(*****
(*) DESIGNER:  Richard C. Gonder           MASTER THESIS WORK
(*) ADVISOR:  MAJ Ken Castor             MODULE NUMBER E.0
(*) DESIGN DATE: Jan 1985                LAST UPDATE: 1 Apr 1985
(*)
(*) DESCRIPTION:  This program will perform correlations on user designa-
(*) ted codesets consisting of files composed of ASCII 1's and 0's. The
(*) Codeset is identified by the user, and bounded by the extensions i.e.
(*) NAME.001 NAME.005. If these are the user inputted filenames, the
(*) program will do the cross correlations of all pairs of codes in this
(*) codeset. It determines the pct out of threshold function for each
(*) pair, in terms of the even and odd correlation function and stores
(*) the minimum, maximum and average value for each threshold value.
(*) When the process is completed, these values are written to the user
(*) identified output file.
(*)
(*) CALLING ARGUMENTS: Std Input, Output, Designated Text File
(*)
(*) INTERNAL VARIABLES:
(*)   Phase—the phase shift of desired process
(*)   PctEven,PctOdd—used for output of thresholding values
(*)   Sum1even,Sum2even,Sum1Odd,Sum2Odd—temp storages values for
(*)   thresholding process
(*)   Threshold—actual threshold passed into threshold process
(*)   DataList,Back—pointers used to create/manipulate the thresh
(*)   old functions
(*)   Resultfile,Codedta—File variable for file I/O
(*)   Result,Filenal,Filena2: string for filename with extension
(*)   CodeName—User-inputted filename for storage of the code
(*)   ch—character input to the program from user
(*)   List1,List2 used to identify the beginning of linked lists
(*)   representing the codes to be correlated
(*)   PctList—used to identify beginning of compiled output data
(*)   Stop—boolean used to stop correlation process
(*)   St1,St2—string expressions for filename extensions
(*)   Startfile,Stopfile—used to store integer value of filename
(*)   extensions which bound the entire process
(*)   Number—represents the number of cycles the program has ac-
(*)   plished and is used to determine the average values
(*)   Inside,Outside—used to control looping structures
(*)
(*) CONSTANTS: None
(*)
(*) FILES USED:  User designated Filename for code readin and data output
(*)
(*) MODULES CALLED:  CreateList, UpdateDataList, GetRidOf
(*)                  ShiftList, Correlate, DetermineName
(*)                  Writelst, CountUpper, FixList, CountLower, DisposeOf*)

```

```

if (-Ptr1^.Odd < Thresh) and (-Ptr2^.Odd >= Thresh) then begin
    R := (-Ptr2^.Odd - Thresh) / (-Ptr2^.Odd + Ptr1^.Odd);
    Count2 := Count2 + R
end; (*if*)
if (-Ptr1^.Odd >= Thresh) and (-Ptr2^.Odd >= Thresh) then begin
    Count2 := Count2 + 1
end; (*if*)
if (-Ptr1^.Odd >= Thresh) and (-Ptr2^.Odd < Thresh) then begin
    R := (-Ptr1^.Odd - Thresh) / (-Ptr1^.Odd + Ptr2^.Odd);
    Count2 := Count2 + R
end;
Ptr1 := Ptr2;
until Ptr1^.Next1 = nil
end; (*procedure CountLower*)

```

```

(*****
(*)
(*)           W r i t e l i s t           (*)
(*)
(*****
(*) DESIGNER:  Richard C. Gonder           MASTER THESIS WORK (*)
(*) ADVISOR:  MAJ Ken Castor             MODULE NUMBER E.4 (*)
(*) DESIGN DATE: Nov 1984                 LAST UPDATE: 14 Mar 1985 (*)
(*)-----(*)
(*) DESCRIPTION: This procedure takes a linked list of data and writes (*)
(*) the data fields, mineven, minodd, maxeven, maxodd, aveeven, aveodd (*)
(*) and Threshold, to the designated file. (*)
(*)-----(*)
(*) CALLING ARGUMENTS: List--pointer to front of datalist (*)
(*) OutFile--string for output filename (*)
(*)-----(*)
(*) INTERNAL VARIABLES: (*)
(*) P--temporary pointer used to traverse list (*)
(*) K--Counter used as chip position in list (*)
(*)-----(*)
(*) CONSTANTS: None (*)
(*)-----(*)
(*) FILES USED: Writes to OutFile (*)
(*)-----(*)
(*) MODULES CALLED: None (*)
(*)-----*)

```

```

procedure Writelist(List: Pctptr; var OutFile: text);

```

```

var

```

```

    K: integer;

```

```

    P: Pctptr;

```

```

begin

```

```

    P := List;

```

```

    GoToXY(20,14);

```

```

    writeln('WRITING THE OUTPUT FILE NOW');

```

```

    while P <> nil do begin

```

```

        write(OutFile, P^.thresh:5:4, ',', P^.mineven: 8: 7, ',',

```

```

            P^.maxeven: 8: 7, ',', P^.aveeven: 8: 7);

```

```

        writeln(OutFile, ',', P^.minodd: 8: 7, ',', P^.maxodd: 8: 7, ',',

```

```

            P^.aveodd: 8: 7);

```

```

        P := P^.Next

```

```

    end (*while loop*)

```

```

end; (*procedure Writelist*)

```

```

(*****
(*)
(*)           D e t e r m i n e N a m e           (*)
(*)
(*****
(*) DESIGNER:  Richard C. Gonder           MASTER THESIS WORK (*)
(*) ADVISOR:  MAJ Ken Castor             MODULE NUMBER E.9 (*)
(*) DESIGN DATE: Mar 1985                 LAST UPDATE: 14 Mar 1985 (*)
(*)
(*) DESCRIPTION: This procedure takes two strings representing filenames (*)
(*) with extensions, and separates the extensions from the filenames. It (*)
(*) then convertst the extension strings to integer values, Startfile and (*)
(*) Stopfile. It should be noted that the function calls in this module (*)
(*) are not standard and the Module may need to be modified for different (*)
(*) systems. (*)
(*)
(*) CALLING ARGUMENTS: Filenal,Filena2--strings representing filenames (*)
(*) Startfile,Stopfile--integer value of filename ex- (*)
(*) tensions (*)
(*)
(*) INTERNAL VARIABLES: (*)
(*) St1,St2--strings representing filename extensions (*)
(*) K,I--temp storage for character positions in strings (*)
(*)
(*) CONSTANTS: None (*)
(*)
(*) FILES USED: None (*)
(*)
(*) MODULES CALLED: None (*)
(*)

```

```

procedure DetermineName(var Filenal,Filena2:Name; var Startfile,
                        StopFile:integer; var CodeName:st10);

```

```

var K,I:integer;
    St1,St2: St3;

```

```

begin
  I:= pos('.',filenal);
  St1:= copy(Filenal,I+1,I+4);
  St2:= copy(Filena2,I+1,I+4);
  CodeName:= copy(Filenal,1,I);
  val(St1,Startfile,K);
  val(St2,Stopfile,K);
end; (*procedure DetermineName*)

```



```

(*****
(*)
(*)          M A I N   P R O G R A M          (*)
(*)
(*****)

```

begin

```

  PctList:=nil;
  writeln('This program allows you to do correlations of ASCII code files');
  writeln('representing a set of pseudorandom sequences. You input only');
  writeln('the first and last codefile in the set identified by its
    extension');
  writeln('and the program does the crosscorelation and threshholding');
  writeln('procedure on each pair. It will determine the minimum,');
  writeln('maximum and average values of the Threshold process and writes
    the');
  writeln('results to the desired file');
  writeln;
  writeln('Do you want to do any correlations at this time?..Y/N');
  readln(ch);
  if (ch = 'y') or (ch = 'Y') then begin
    repeat
      writeln('enter the first file you wish to correlate');
      writeln('with extension .. i.e. .001, .002 etc');
      readln(Filenal);
      writeln('now enter the second Filename w/extentsion');
      readln(Filena2);
      writeln('Enter phaseshift desired for run...0 to length');
      readln(Phase);
      writeln('enter the file to write results');
      readln(Result);
      DetermineName(Filenal,Filena2,Startfile,StopFile,CodeName);
      Number:= 1;
      ClrScr;
      for Outside:= (Startfile) to (Stopfile -1) do begin
        str(Outside:3,st1);
        if st1[1] = ' ' then st1[1]:= '0';
        if st1[2] = ' ' then st1[2]:= '0';
        Filenal:= CodeName + st1;
        CreateList(Filenal, List1, N);
        for J := 1 to Phase do
          ShiftList(List1);
        for Inside:= (Outside + 1) to Stopfile do begin
          str(Inside:3,st2);
          if st2[1] = ' ' then st2[1]:= '0';
          if st2[2] = ' ' then st2[2]:= '0';
          Filena2:= CodeName + st2;
          CreateList(Filena2, List2, N);
          for J := 1 to Phase do
            ShiftList(List2);
          DataList := nil;
          Back := nil;
          Correlate(DataList, Back, List1, List2, N);

```

```

Threshold := 0;
stop := false;
GoToXY(20,14);
writeln('WORKING ON ',filena1,' and ',filena2);
Sumleven := 0;
Sum2even := 0;
Sumlodd := 0;
Sum2odd := 0;
while (Threshold <= N) and not stop do begin
  CountUpper(DataList, Threshold,Sumleven,Sumlodd);
  CountLower(DataList, Threshold,Sum2even,Sum2odd);
  PctEven := (Sumleven + Sum2even) / N;
  PctOdd := (Sumlodd + Sum2odd) / N;
  S:= Threshold/N;
  UpdateDataList(PctList, Number, PctEven, PctOdd, S);
  if Threshold / N > 0.10 then
    Threshold := Threshold + N / 100
  else
    Threshold := Threshold + N / 500;
    if (PctEven = 0.0) and (PctOdd = 0.0) then
      stop := true;
    end(*while loop*);
    FixList(Threshold/N,Number,PctList);
    GetRidOf(DataList);
    DisposeOf(List2);
    Number:= Number + 1;
  end (*for Inside loop*);
  DisposeOf(List1);
end;(*for Outside Loop*)
assign(Resultfile,Result);
rewrite(Resultfile);
Writeln(Pctlist,Resultfile);
close(Resultfile);
Mark(Pctlist);
Release(PctList);
ClrScr;
writeln('Do you want to do any other codesets? Y/N');
readln(ch);
until (ch = 'N') or (ch = 'n');
end(*if*)
end. [ PartialCorrelate ]

```

## VITA


Captain Richard C. Gonder was born on 4 February 1954 in San Antonio, Texas. He graduated from high school in Williamsville, New York in 1972, and accepted an appointment to the United States Military Accademy at West Point, New York. He graduated from the Military Accademy in 1976 with a Bachelor of Science Degree and a Commission in the United States Army Signal Corps. After completion of the Signal Officer Basic Course at Fort Gordon, Georgia, he was assigned as a Signal Communications Officer in the 7th Infantry Division at Fort Ord, California. After completion of the Signal Officer Advanced Course in 1980, he served as Assistant Operations Officer at the 2d Signal Brigade in Mannheim, West Germany, and as Commander of the 324th Signal Company in Karlsruhe, West Germany, until entering the School of Engineering, Air Force Institute of Technology, in May 1983.

Permanent address: 1432 Point Drive

Sioux Falls, SD 57103

AD-A159309

REPORT DOCUMENTATION PAGE

1a. REPORT SECURITY CLASSIFICATION <b>UNCLASSIFIED</b>			1b. RESTRICTIVE MARKINGS		
2a. SECURITY CLASSIFICATION AUTHORITY			3. DISTRIBUTION/AVAILABILITY OF REPORT  Approved for public release; distribution unlimited.		
2b. DECLASSIFICATION/DOWNGRADING SCHEDULE					
4. PERFORMING ORGANIZATION REPORT NUMBER(S)  AFIT/GE/ENG/85J-1			5. MONITORING ORGANIZATION REPORT NUMBER(S)		
6a. NAME OF PERFORMING ORGANIZATION  School of Engineering		6b. OFFICE SYMBOL (If applicable) AFIT/ENG		7a. NAME OF MONITORING ORGANIZATION	
6c. ADDRESS (City, State and ZIP Code)  Air Force Institute of Technology Wright-Patterson AFB, Ohio 45433			7b. ADDRESS (City, State and ZIP Code)		
8a. NAME OF FUNDING/SPONSORING ORGANIZATION		8b. OFFICE SYMBOL (If applicable)		9. PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER	
8c. ADDRESS (City, State and ZIP Code)			10. SOURCE OF FUNDING NOS.		
			PROGRAM ELEMENT NO.		PROJECT NO.
			TASK NO.		WORK UNIT NO.
11. TITLE (Include Security Classification) See Box 19					
12. PERSONAL AUTHOR(S) Richard C. Gonder, CPT, USA					
13a. TYPE OF REPORT MS Thesis		13b. TIME COVERED FROM _____ TO _____		14. DATE OF REPORT (Yr., Mo., Day) 1985 June	
				15. PAGE COUNT 181	
16. SUPPLEMENTARY NOTATION					
17. COSATI CODES			18. SUBJECT TERMS (Continue on reverse if necessary and identify by block number)		
FIELD	GROUP	SUB. GR.	Radio Communications, Code-Division Multiple- Access, Spreading Sequences, Pseudonoise Sequences		
17	02	1			
19. ABSTRACT (Continue on reverse if necessary and identify by block number)					
<p>Title: CODE SEQUENCE PERFORMANCE ANALYSIS USING CROSS-CORRELATION PARAMETERS IN PHASE-CODED MULTIPLE ACCESS COMMUNICATION SYSTEMS</p> <p>Thesis Advisor: Kenneth G. Castor, Major, USAF</p> <div style="text-align: right;"> <p>Approved for public release; LAW AFB 180-14.                LYNN E. WOLAVER 19 AUG 85              Dean for Research and Professional Development              Air Force Institute of Technology (AFIT)              Wright-Patterson AFB OH 45433</p> </div>					
20. DISTRIBUTION/AVAILABILITY OF ABSTRACT  UNCLASSIFIED/UNLIMITED <input checked="" type="checkbox"/> SAME AS RPT. <input type="checkbox"/> DTIC USERS <input type="checkbox"/>			21. ABSTRACT SECURITY CLASSIFICATION  UNCLASSIFIED		
22. NAME OF RESPONSIBLE INDIVIDUAL  Kenneth G. Castor, Major, USAF			22b. TELEPHONE NUMBER (Include Area Code) 513-255-2024		22c. OFFICE SYMBOL AFIT/ENG

This paper documents a technique to compare cross-correlation parameters of binary sequences used for spread-spectrum multiple-access communication systems, by performing a thresholding process on the correlation functions. The performance of Maximal length, Gold and Kasami code sequences is measured and analyzed for code lengths ranging from 63 to 1023. Comparisons of optimized codes versus unoptimized codes for each type of code sequence are analyzed in terms of the thresholding process. Comparisons are made of the performance of code sequences and code sets at lengths of 63, 127, 255 and 1023.

The software used to analyze the codes is discussed in terms of structure and performance, and is included as appendices in the thesis.

The results of this investigation indicate that the thresholding process can be used to evaluate binary sequence performance.

**END**

**FILMED**

**11-85**

**DTIC**