DEMODULATION/DECODING OF A DC FREE CONSTRUCTION OF THE GOSSET LATTICE

NORMAN PIERRE SECORD, M. Eng

A Thesis Submitted in Partial Fulfillment

of the Requirements for the Degree

Doctor of Philosophy

Department of Electrical and Computer Engineering

McMaster Univerity

November 1989

DEMODULATION/DECODING OF A DC FREE

CONSTRUCTION OF THE GOSSET LATTICE

DOCTOR OF PHILOSOPHY (1989) McMASTER UNIVERSITY
(Electrical and Computer Engineering) Hamilton, Ontario

TITLE: Demodulation/Decoding of a DC Free Construction of the Gosset Lattice

AUTHOR: Norman Pierre Secord, B. Eng. (Elect. Eng) McMaster University
M. Eng. (Elect. Eng) McMaster University

SUPERVISORS: Professors R. de Buda, D.P. Taylor, C.R. Carter, P. Yip and K.M. Wong

ABSTRACT

Over the past decade, lattices have been increasingly recognized as an important source of codes for the Gaussian channel. The 8–dimensional Gosset lattice has figured prominently in these new developments because it offers an asymptotic coding gain of 3 dB over conventional pulse amplitude modulation and can be soft decision demodulated/decoded with a reasonable amount of speed. In the present work, we revive a little used definition of the Gosset lattice and show that codes derived from this construction exhibit a null at dc in their baseband spectrum. Such codes are useful as line codes for baseband signalling on channels that do not support a dc spectral component or for bandpass transmission where spectral shaping is required to combat intersymbol interference.

Previous applications of lattice codes have been aimed primarily at voiceband data communications. This thesis was motivated by the need to develop a decoder that would make applications in the multi–megabit range of data transmission possible. To achieve this goal, a two–stage approach to demodulation was developed. The first stage makes a fast estimate of the transmitted vector and has the ability to declare an erasure when it knows its estimate is unreliable. This initial erasure declaring stage controls the throughput of the demodulator. Because it is far simpler than a maximum likelihood demodulator, greater speed is achieved. The second stage is provided to correct the occasional occurrence of an erasure and maintain the error performance of the lattice. To complete the decoder structure, we outline a method of lexicographic ordering the signal set that leads to a compact set of decoder look–up tables used to obtain a binary message sequence from each demodulated vector. Finally, we evaluate the effects of quantization on the probabilities of erasure and error and give results from a Monte–Carlo simulation undertaken to verify the demodulators performance.

ACKNOWLEDGEMENTS

Dr. Rudi de Buda had been my supervisor and mentor for over four years when he passed away in September of 1988. Despite being very ill for over a year, Rudi's enthusiasm and support of this work never seemed to wane. When I learned that Rudi had been reading and correcting some of my thesis material up until the day before he died, it struck me all the more how dedicated he was to seeing this work completed. I have missed Rudi a great deal and will always be grateful to have had the opportunity to know him and work with him.

I would like to thank my thesis committee, Drs. Taylor, Carter, Yip and Wong, for quickly taking charge of the supervision of my thesis work after Rudi's death. I am especially indebted to Des Taylor for taking the extra time to read the drafts of this thesis and provide the type of objective criticism that I needed to make this a more complete piece of work. Des has been a continual source of support and advice throughout my graduate career, and I am very grateful for the enthusiasm he has shown towards my work. I would also thank Dr. Max Wong for ensuring that I had adequate financial support until I finished my thesis.

I would like to thank all my colleagues at the Communications Research Lab for their friendship and for making the lab an interesting place to work. I would like to thank Greg Pottie in particular for taking the time to critique my papers and for the lively discussions of shared research interests.

Finally, to my brother Patrick, my sister Michelle and especially to my parents Lise and Lloyd, I owe you the world of thanks for always being there to back me in whatever I do. Your support and love has made life all the more enjoyable to live.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

CHAPTER 1

INTRODUCTION

1.1 Lattices And Their Importance In Communications Theory

Of the resources provided to the communications engineer, power and bandwidth are the most valuable. On *band–limited channels*, which in general encompass terrestrial communications such as the telephone system, the large number of users simultaneously requesting a portion of the frequency spectrum dictates that tight restrictions be place on the range of frequencies, or *bandwidth*, allotted to each user. Power is always readily available in terrestrial communications. However, methods of reducing the amount of power consumed, thereby reducing transmission costs, are continually being researched.

In communication via satellite, on the other hand, both power and bandwidth are in limited availability. Power is supplied to the electronics of the transmission system by large solar panels, with battery backup when the satellite is eclipsed by the earth. Hence power consumption greatly affects the weight of the satellite, and the weight directly controls the cost and lifetime of the space craft. Since the cost of putting a satellite into orbit is very high, one also wants to make very efficient use of bandwidth to accomodate as many users as possible. Obviously, to get the greatest return from one's investment in a satellite, it is crucial that the transmission system use the available power and bandwidth as efficiently as is possible.

The set of *channel signals* used to transmit the information from a given source, usually considered to be a binary source, through the channel to its

destination are generally referred to as the *modulation*. When choosing a modulation for a particular transmission system, the designer attempts to find a set of signals that are robust, in the sense that they can be reproduced with a high degree of fidelity at the receiver, while making the most efficient use of power and bandwidth. While this may seem like a simple objective to fulfill, experience has shown that the complexity of the transmission system needed to maintain a given level of fidelity increases dramatically as one attempts to decrease either the power or the bandwidth, or both.

The problem considered in this thesis can be briefly summarized as the development of a high speed decoder structure to implement a new form of efficient modulation for communication over the band–limited channel. For many years the predominant form of digital transmission used on band–limited channels has been *pulse amplitude modulation* or *PAM*. To overcome the limitations placed on the signaling rate by a restriction in available bandwidth, a pulse amplitude modulation system maps binary information into a set of multi–level pulses or *channel symbols*. Every T seconds, called the *symbol interval* or *symbol duration*, a pulse is transmitted through the channel. The reciprocal of the symbol duration, $1/T$, is referred to as the *symbol rate* or *signaling rate* and is quoted in *symbols per second*. When the pulses are modulated onto a carrier for *bandpass* transmission, the signaling rate represents the minimum bandwidth required by the system; for *baseband* transmission, where no carrier is used, the minimum bandwidth required is half the signaling rate. If k bits are mapped into each symbol then the *bit rate* or *data rate* is k/T *bits per second*.

While increasing the number of modulation levels (i.e., the *symbol alphabet*) in a pulse amplitude modulation system allows us to increase the data rate without increasing the bandwidth, a rather large increase in power is required to maintain

the same level of performance. In the late 1960's and early 1970's, researchers began to look for new forms of multi–level modulation which could provide the higher data rates desired without sacrificing power or performance. Shannon's work [51] on the capacity of a channel had shown that the power required by a pulse amplitude modulation system was 9 decibels (dB) greater (roughly 8 times larger) than that required by the optimum transmission system for a band–limited channel. It was obvious that there must exist better modulation schemes which could make up at least some of the 9 dB difference. The question was what form would they take.

At about the same time, mathematicians were making use of error correcting codes in constructing dense lattice packings of spheres [40],[41],[42]. It had been clear since the late 1950's [52] that the problem of finding the optimum code for the band–limited channel was closely related to the sphere packing problem in mathematics.[1] Since lattices are easy to construct and provide the densest known sphere packings in a given number of dimensions, it would seem inevitable that communications theorists would look to this branch of mathematics for solutions to some of their problems. Although the early use of lattices in communications has been attributed to Lang [18],[27], the first known publication proposing the use of lattices as codes for the band–limited Gaussian channel was that of Blake [3] in 1971.

Stated rather simply, an *n–dimensional lattice* Λ is an infinite, regular array of

---

[1] The mathematical problem is to determine the best arrangement of n–dimensional spheres of equal size so that the largest possible number fill a given volume of n–space. The communications problem is to find, for a given power constraint, the arrangement of M signal points in n–space that minimizes the probability of an incorrect detection when the signals are perturbed by additive white Gaussian noise. The relationship between the two problems becomes clear if one associates the M signal points with the centres of the M equal sized spheres. A constraint in power translates to having a fixed volume in which to pack the spheres, while minimization of the probability of error relates to attempting to use as large a sphere as possible so that there is the greatest possible distance between their centres.

points in real Euclidean m–space $\mathbb{R}^m$, where $m \geq n$. The vectors x which make up the lattice Λ can be generated through the linear combination of a set of n basis vectors $e_1, e_2, \ldots, e_n$ with the integer coefficients $a_1, a_2, \ldots, a_n$, i.e.,

$$x = a_1 e_1 + a_2 e_2 + \ldots + a_n e_n .$$

If the dimension m of the basis vectors equals the dimension n of the lattice, the basis vectors are said to span the space; otherwise Λ is said to be an n–dimensional lattice embedded in a space of m dimensions. All lattices have the property that the vector sum of any two points is another point in the lattice Λ. Also, the null or zero vector 0 is a vector in all lattices and for every vector $x \in \Lambda$, its additive inverse $-x$ is in Λ. These last properties endow a lattice with its regularity and make it an *additive group.*

It is instructive at this point to diverge from our narrative and give some of the common terminology associated with lattices. This terminology can be found in any of a number of sources, see for example [17],[27],[56].

The *kissing number* of a lattice is the number of nearest neighbours to any point in the lattice. This is a term commonly used in sphere packing: if we were to arrange a number of equal sized balls around a central ball, the kissing number would be the number of balls which just touch or kiss the central ball. When the centres of the balls are the points of a lattice, the sphere packing is of course referred to as a *lattice sphere packing.*

A key parameter used to measure the performance of a code is its *minimum squared distance* d^2_{min} which for lattices is the squared distance from any point to its nearest neighbours. This minimum squared distance may alternatively be defined as the squared magnitude of the vectors closest to the origin (usually referred to as the *vectors of minimal norm* or simply the *minimal vectors*) since these are the vectors which connect any point to all its nearest neighbours. It

follows that the kissing number and the number of minimal vectors are one and the same.

The *Dirichlet region* $D_\Lambda(x)$ of a point x in the lattice Λ (also called the *fundamental region* or *Voronoi cell*) consists of all points in Euclidean n–space $\mathbb{R}^n$ that are closer to x than to any other point in the lattice and a portion of the boundary between x and its nearest neighbours. The Dirichlet region represents a fundamental building block which allows us to tile all of n–space with a set of equal non–overlapping cells whose centres are the points of the lattice. Thus lattices are associated geometrically with *honeycombs* or *regular tesselations* of n–space [21],[27]. Examples of simple lattices are the integer lattice $\mathbb{Z}^2$ and the hexagonal lattice A_2 shown in Figures 1.1*a* and *b*, respectively. If we divide the space around each point evenly, then we see that the Dirichlet region associated with the integer lattice $\mathbb{Z}^2$ is a square and for A_2 it is a hexagon. In Table 1.1, we have listed a number of lattices that are frequently encountered in discussions of lattices and lattice codes. Although the symbolic notation is becoming more commonly used to refer to each of these lattices, they are still known by the name of their discoverer. The dimension of the lattice is in each case given by the subscript of the symbol.

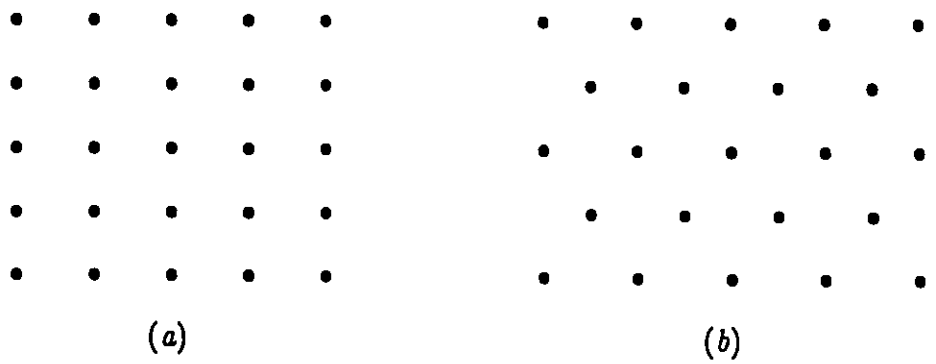

Figure 1.1: Portions of (*a*) the integer lattice $\mathbb{Z}^2$, and (*b*) the hexagonal lattice A_2

Common Name	Symbol
Schlafli	D_4
Gosset	E_8
Barnes–Wall	Λ_{16}
Leech	Λ_{24}
Barnes–Wall	Λ_{32}

Table 1.1: A list of important lattice and their symbolic notation

On the side of the algebraic properties of a lattice, we may define a *sublattice* Λ' as any subset of the points of Λ that is itself a lattice. The sublattice induces a natural *partitioning* of the lattice into constituent parts. That is to say, the lattice Λ may be considered the union of a sublattice Λ' with a finite number of translates $\Lambda'+e$ of the sublattice Λ' called *cosets*. Such a partition is denoted symbolically as Λ/Λ' and the number of cosets in the partition, $|\Lambda/\Lambda'|$, including the sublattice which may be considered the zero coset $\Lambda'+0$, is called the *index of* Λ' *in* Λ.

Returning to the communications problem, a *lattice code* may be defined as some finite set of vectors from a lattice for which algorithms have been developed to encode (map information from the source to a lattice vector) and decode (identify the transmitted vector and reproduce the original source information). In essence, a lattice code is a form of pulse amplitude modulation in which the components of the lattice vectors specify the amplitudes of the pulses. When the process of estimating the transmitted vector involves nothing more than a simple quantization, as is the case with standard PAM, the decoder is said to be a *hard decision decoder*. When an estimate of the transmitted vector is obtained at the receiver by finding the code vector that is closest to the received vector in terms of squared Euclidean distance, the decoder is said to be a *maximum likelihood decoder* or *soft decision decoder*.

The significance of lattice codes as a form of modulation became apparent in 1975, when R. de Buda [6] used a theorem from the geometry of numbers to prove that the error performance of codes derived from lattices could, for high signal–to–noise ratios, approach Shannon's upper bound [52] on the error performance of the optimal code for the additive white Gaussian noise channel. Unfortunately, de Buda's bound fell significantly short of Shannon's bound at low signal–to–noise ratios. Further refinements were made by Kassem in 1981 [37] who, with de Buda [8], developed bounds for different classes of lattice codes. More recently, de Buda [7] has shown that there must exist lattice codes whose upper error bound is only a factor of four larger than Shannon's bound for the optimal code.

1.2 A History Of The Decoding Problem For Lattice Codes

Although de Buda's result [6] was a very significant discovery, applications of lattice codes were slow in following and did not begin to appear in print until the early 1980's. This is in part due to the fact that encoding and decoding a lattice code is not trivial. At the receiver, not only must an estimate of the original transmitted vector be obtained but once the vector has been identified the encoding process must be reversed to obtain a message sequence. These can be very time consuming tasks to perform and devising the fastest, most effective approach to decoding a lattice code is something of a black art. Fortunately, advances in the technology of very large scale integrated (VLSI) circuits and in microprocessor technology have made the use of lattices in data transmission systems not only feasible but a reality.

It has become popular within the literature to refer to the identification of the closest point on the infinite lattice to an arbitrary point of n–space as *decoding.* We

refer to such a process here as *demodulation* and reserve the use of decoding to indicate the two step process of finding the nearest code vector to the received vector, and identifying the message sequence associated with the code vector. It could be argued that, since there is a one–to–one mapping between any code vector and its message sequence, decoding is complete once the code vector has been identified. However, this is not valid when considering the speed of decoding because it does not take into account that obtaining a message sequence from a code vector requires a finite number of operations. For this reason, demodulation will be used to refer to finding the closest point either on the infinite lattice or within a finite code to an arbitrary point $r \in \mathbb{R}^n$ of n–space.

The first complete encoding and decoding algorithms for a lattice code were given by P. de Buda in 1981 [4]. He chose a set of 2^{14} vectors of equal energy from the fourth energy shell of the 8–dimensional Gosset lattice E_8. With 14 bits of information mapped into 8 real components or alternatively 4 complex components, the lattice code had an efficiency of 1.75 bits per real symbol or 3.5 bits per complex symbol, halfway between the efficiency of an 8–phase modulation (8–PSK) system (3 bits per complex symbol) and a 16–point quadrature amplitude modulation (16–QAM) constellation (4 bits per complex symbol). For a probability of symbol error of 10^{-5}, the bit energy to noise spectral density ratio, E_b/N_0, required by the lattice code was 2 dB less than that required by the 16–QAM system, showing that there were indeed instrumentable lattice codes that could provide a gain over conventional forms of modulation.

In 1982, Conway and Sloane [14] gave, for a number of different lattices, algorithms to detect the closest point on the infinite lattice to an arbitrary point in n–space. The following year, the same pair of authors presented a technique [15] for finding the *index* or message associated with a vector in a lattice code using the *dual*

basis of the lattice. By combining this method of indexing the code vectors with the appropriate demodulation algorithm in [14], one could then construct a complete decoding structure. Also in 1983, ESE Ltd. [23] submitted a proposal to Study Group XVII of the CCITT for the use of the Gosset lattice in new 2400, 4800 and 9600 bit/s modem standards for the switched telephone network. The ESE proposal recommended the use of Conway and Sloane's E_8 demodulation algorithm [14] to detect the closest code point to the received signal.

Further applications of lattices and lattice codes appeared in the September 1984 issue of the IEEE Journal of Selected Areas in Communication. In this special issue on bandwidth efficient coding and modulation, Gersho and Lawrence [32] outlined encoding and decoding algorithms for lattice codes in four and eight dimensions. These algorithms were similar to the earlier work of P. de Buda [4] in that both viewed the code as an ensemble of permutation modulations (a code generated through all permutations and sign changes made to the components of a single prototype vector [53]). Gersho and Lawrence, however, chose a series of vectors lying in successive shells of the lattice to arrive at a code with an average energy constraint, in contrast to the equal energy code of P. de Buda. The four and eight dimensional lattice codes of Gersho and Lawrence both had efficiencies of 4 bits per complex symbol and they reported [32,p. 687] improvements in the signal–to–noise ratio, over 16–QAM, of 1.2 dB and 2.4 dB, respectively, for the same level of error performance.

In the same special issue, Forney *et al.* [30] introduced a technique for representing lattices by a type of graph known as a *trellis*. This representation was highly influenced by the *trellis coded modulations* of Ungerboeck [58]. Ungerboeck had developed a rule for mapping the outputs of a binary convolutional encoder onto one and two dimensional signal constellations that he referred to as *mapping by set*

partitioning. Several authors [22],[30] were quick to recognize that if you replaced the convolutional code with a binary block code, you could construct a lattice. The set partitioning that Ungerboeck was referring to, when considered in the context of lattices, is the natural partitioning of a lattice into cosets. Although the work of Forney et al. [30] hinted at much deeper relationship between lattice and trellis codes, they were still considered quite separate modulation schemes for several more years. The lines between these two schemes became permanently blurred with Calderbank and Sloane's [11] introduction of trellis codes based on lattice partitions. The concepts of Calderbank and Sloane have been further refined by Forney [27],[28] who has attempted to put lattice and trellis codes into a common framework. The current thinking on trellis and lattice codes is that they fall under the general heading of *coset codes.* In a coset code, the signal constellation is viewed as some finite set of points from an n–dimensional lattice (or a translate of the lattice) that can be partitioned into cosets with an equal number of points in each coset. To encode, k bits from a b bit block are sent into a rate $k/k+r$ binary encoder (block or convolutional) and the outputs of the encoder select a coset in the lattice partition Λ/Λ'. The remaining b–k bits select a particular point within the coset through some mapping process and this point is then the signal to be transmitted through the channel. This simple process is illustrated by the block diagram given in Figure 1.2 [27]. This structure can be seen to be very simple and to cover a large class of codes. Further generalizations to the coset code structure have been proposed recently by Calderbank [9] and by Pottie and Taylor [47]. These multilevel codes are designed to allow several levels of binary coding that may include the use of block and convolutional codes at different levels within the partition structure. As well, the partitions need not be lattice partitions making it possible to define, under a single structure, the combination of coding with many varied forms of modulation.

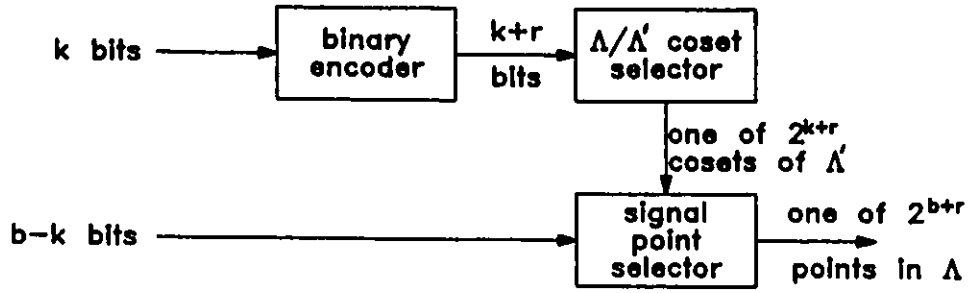

Figure 1.2: Block diagram of basic encoder for a coset code

Prior to the papers of Calderbank and Sloane [11] and Forney [27],[28], there were two other works on lattices and lattice codes in 1986. In [16], Conway and Sloane provided another set of demodulation algorithms for lattices, this time turning to the constructions of lattices involving error correcting codes. In this paper, they gave the first detection scheme for the very important 24–dimensional Leech lattice Λ_{24}. Several months later, Secord and R. de Buda [47] outlined encoding and decoding algorithms for a 2^{14} vector equal energy code taken from the fourth shell of the Gosset lattice. The construction used by Secord and R. de Buda was different from that used by P. de Buda [4] resulting in a quite different set of code vectors and consequently different encoding and decoding algorithms. With a simple change in the coordinate system, Secord and R. de Buda were able to develop a much faster decoding strategy, one that was in fact fast enough to provide a data rate of 28 kilobits/s on a simple microprocessor although a voiceband channel could not supply information at this rate. They were also able to show that an 8 bit microprocessor was sufficient to achieve the theoretical error performance of the code and achieve 2.4 dB of gain over 16 QAM at an error rate of 10^{-6}.

All the applications discussed within this section have been, either directly or

indirectly, aimed at the kilobit range of data rates associated with voiceband data modems. To date there has been no real attempt made to utilize lattice codes in multi-megabit data transmission systems such as those found on the digital microwave radio channel. The reason why pulse amplitude modulation has survived as the modulation format on these channels is quite simple. Pulse amplitude modulation requires a simple hard decision during each symbol interval to identify the transmitted pulse. When the symbol interval is only a few tens of nanoseconds long, as in the case of a microwave radio channel, there is very little time to perform more than a few operations, even with the best of todays available digital technology. Many of the soft decision demodulation and decoding strategies proposed for lattice codes require into the tens and hundreds of operations per symbol interval. While certain strategies may be developed to perform a number of these operations in parallel, in general this will still not be sufficient to make the application of lattice codes feasible at such data rates.

The motivation for this thesis was thus to find a way of reducing the number of operations per symbol interval without sacrificing the gain provided by the lattice code. In other words, we wanted to find a way of marrying the speed of a hard decision decoder to the improved error performance provided by a soft decision decoded lattice code. The particular lattice chosen was the Gosset lattice E_8. As this narrative has indicated, the Gosset lattice has figured prominently in most of the applications of lattice codes [4],[23],[32],[49]. This is primarily due to the fact that it provides a modest 3 dB of asymptotic coding gain over conventional pulse amplitude modulation and has a block length which is manageable for demodulation/decoding purposes.

The approach taken in the research leading up to this thesis was to first develop a demodulation strategy for finding the closest point on the infinite lattice

and to make this demodulator as fast as possible. As with all such demodulation algorithms [14],[16],[28], it can be used to demodulate a finite code provided the code is chosen in such a way that the underlying structure of the lattice is preserved over some finite region of space. In other words, the code must consist of all lattice points that lie within some bounded region of n-space. In general, the bounding region is chosen so that the signal constellation is as symmetric about the region's centroid (usually the origin) as is possible.

To complete the decoder, not only was a method of choosing a signal constellation needed but also a fast technique for mapping information to and from each code vector. In the present case, it was the design of the mapping algorithm which more or less dictated the shape of the signal constellation. A procedure was developed to generate a set of decoder look–up tables that required a signal constellation which was hypercubic in nature. The decoder look–up tables then made it possible to find the message sequence associated with any vector through a sequence of look–ups and adds. The technique is sufficiently general in its description that codes of many different sizes can be generated using the same algorithm. By combining these decoder look–up tables with the detection scheme outlined in Chapter 3, a complete decoder can be obtained.

It is felt that the decoder structure discussed in this thesis has sufficient speed and simplicity that it could be utilized in a digital microwave radio system. This cannot, however, be stated with absolute certainty since this is only a theoretical study of the decoder's development and error performance. The actual system implementation is in fact beyond the scope of the present work and is left for future development.

1.3 The Intersymbol Interference Problem And Partial–Response Signaling

Whenever the frequency content of a signal is restricted, either through the use of filtering so that it might fit into a band–limited channel or by the channel itself band–limiting the signal, we encounter a problem known as *intersymbol interference.* Intersymbol interference arises in a pulse amplitude modulation system when the effects of one pulse have not died out before the next pulse is transmitted [34]. The signal which the receiver sees is the sum or *superposition* of the individually transmitted pulses, thus if the effects of transmitting one pulse have not fallen to zero before the next pulse is transmitted, (i.e., the pulse has a non–zero response outside its symbol interval), it will influence or interfere with the reception of the next pulse. If we consider that not just one but many symbol intervals may be required for the amplitude response of a pulse to decay to zero, the effects of many previously transmitted pulses may be felt in the reception of a particular symbol. This interference can become extremely severe and substantially degrade the performance of the receiver, which must still contend with channel noise as well.

To combat intersymbol interference, what is often done is to introduce some form of correlation between symbols, (in other words a controlled intersymbol interference), with the intention that since it is known it can be removed. This correlation is a form of *memory* that is added to the pulse amplitude signal and as such signaling schemes containing a known intersymbol interference can be classed as *linear modulations with memory* but are most often referred to as *correlative coding schemes* or *partial response signaling schemes* [36],[39].

There are several methods of introducing a controlled intersymbol interference into a pulse amplitude modulation signal. One such method is to take the symbol sequence x_k that specifies the amplitudes of the pulses in each symbol interval $kT \leq t < (k+1)T$ and form a new sequence y_k by subtracting the previous symbol

x_{k-1} from the current symbol x_k, i.e.

$$y_k = x_k - x_{k-1} .$$

Such a sequence y_k can be obtained by passing x_k through what is known as a $1 - D$ *partial response filter*, where D is the *delay operator*. The delay element D is what introduces memory into the sequence y_k . This induced memory of one symbol interval creates periodic *nulls* in the frequency spectrum of the $1 - D$ partial response filtered sequence y_k, as is shown in Figure 1.3. Before the sequence y_k is transmitted, it is sent through a *pulse shaping* or *zonal filter* to limit the frequency content of the transmitted signal to a finite range of values. The pulse shaping filter response that would yield the minimum bandwidth signal is the rectangular or "brick wall" response shown as a dashed line in Figure 1.3.

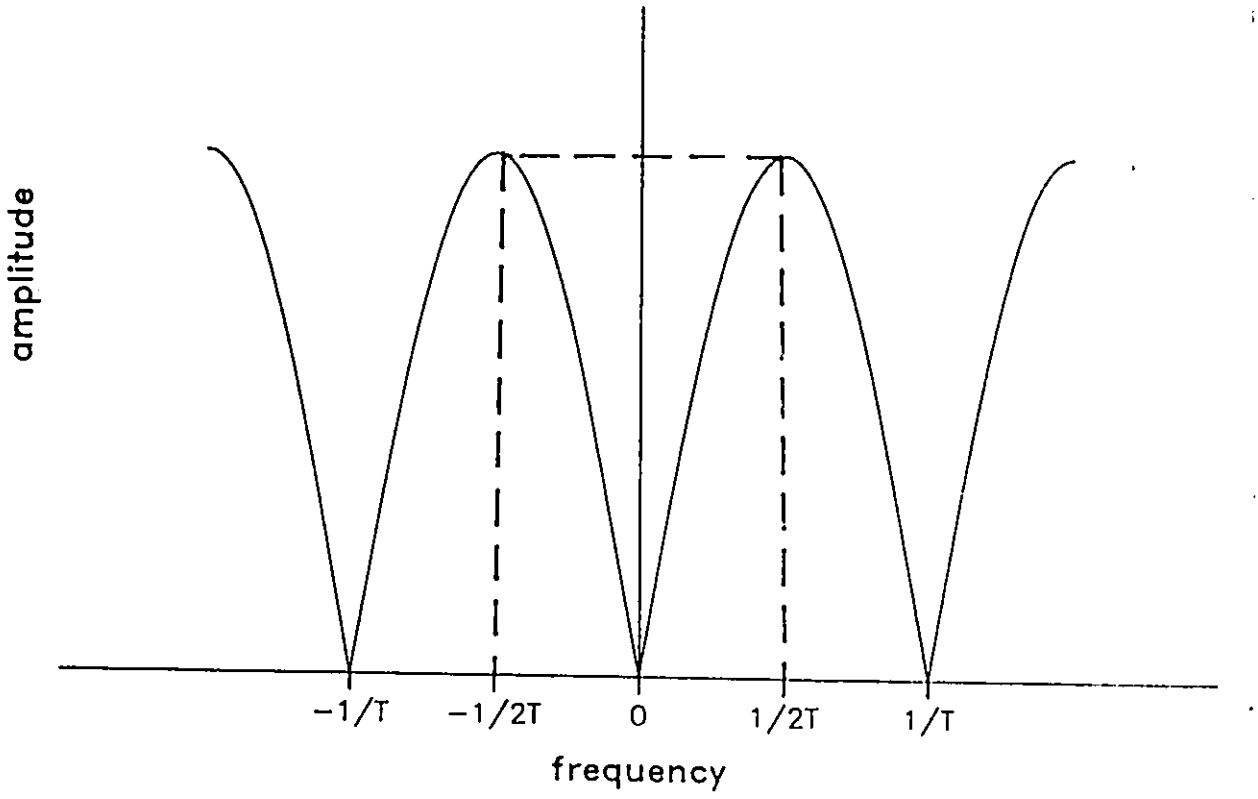

Figure 1.3: Spectrum of a $1 - D$ partial response filter

Another currently popular method of inducing a dc null in the baseband response involves the use of *running digital sum feedback* [10]. The running digital sum is simply the accumulated sum of the amplitudes of all transmitted symbols from start up to the present time interval. In designing a code for use with running digital sum feedback, for every vector $\boldsymbol{x}$ included in the code, its negative $-\boldsymbol{x}$ is also included. When it comes time to transmit a given code vector, the encoder chooses either $\boldsymbol{x}$ or $-\boldsymbol{x}$ whichever will reduce the running digital sum towards zero. The new running digital sum is then computed and fed back for use with the next transmitted vector. This feedback essentially bounds the running digital sum and pushes its expected value towards zero. If the expected value of the running digital sum is zero for all time then it can be shown that the sequence of transmitted code vectors has a null at dc in its baseband spectrum.

We mention $1 - D$ partial response filtering and running digital sum feedback to provide background for some of the discussion to follow in Chapter 2. This thesis is not primarily concerned with the intersymbol interference problem or partial response signaling. Rather, it is an interesting consequence of the choice of lattice construction that the codes discussed in this thesis have a null in their baseband spectrum and therefore fit into the $1 - D$ partial response channel. This lattice and other constructions of its type are an interesting phenomenon in lattice theory since neither $1 - D$ partial response filtering nor running digital sum feedback is needed to produce this spectral null, it is an inherent property. Prior to the present work, such a property had never been identified in any lattice construction. Calderbank, Lee and Mazo [10] have constructed trellis codes with spectral nulls using the running digital sum method and this technique has been extended by Forney and Calderbank [29] to cover the general class of coset codes. These coset codes with spectral nulls form the main basis of comparison with the lattice codes discussed in

the next chapter.

1.4 Additional Sources of Reference

In this chapter, we have attempted to give a brief background to both the history and the terminology of the problem to be considered in this thesis. For the interested reader, there are a number of excellent tutorial and introductory papers to the subjects of sphere packings, lattices and partial response signaling. Sloane's article in Scientific American [53] provides a very readable introduction to the sphere packing problem and the uses of sphere packings in communications. A mathematicians perspective on the development of error correcting codes and their use in constructing sphere packings can be found in the monograph of Thompson [57]. Some of the anecdotes recounted there make for very amusing reading and point to the fact that mathematics and engineering are not always the exact sciences we perceive them to be.

The two recent papers of Forney [27],[28] provide an excellent introduction to the subject of lattices and their application in coding and modulation. Although written with the engineer in mind, some background in binary coding and the algebra of groups, however brief, would considerably aid one's understanding and comprehension of the material presented, (reading the first three chapters of [43] would be sufficient). For the more mathematically inclined, the book of Conway and Sloane [17] is a definitive catalogue of all the major developments in sphere packings and lattice theory over the past 25 years. In particular, the book contains just about everything one would ever want to know about the Leech lattice.

In the area of partial response signaling and the intersymbol interference problem, the tutorial papers of Kabal and Pasupathy [36], and Kobayashi [39] are considered essential reading. The more recent paper of Forney and Calderbank [29]

gives a much updated view of coding for the partial response channel and looks into the tradeoffs between power and the width of the spectral null at dc.

1.5 Outline of the Thesis

In Chapter 2, we begin by defining the particular construction of the Gosset lattice to be used throughout this thesis. This is followed with a derivation of the baseband spectrum of codes taken from this particular lattice construction. These codes are shown to have a spectral null at dc and a comparison is made with other codes having a similar property.

Chapter 3 is devoted to the two–stage demodulation structure proposed for this construction of the Gosset lattice. This demodulator makes use of *erasures* to reduce the complexity of demodulation to just over two operations per component while maintaining the error performance at a level that very quickly approaches maximum likelihood performance at error rates below 10^{-3}.

Chapter 4 outlines a method of ordering and indexing the vectors of a code that is referred to as a *lexicographic ordering.* This technique allows us to generate a set of very compact look–up tables that can be used to find the index of a code vector very quickly. When these look–up tables are combined with the demodulation algorithm discussed in Chapter 3, a complete decoder structure can be obtained.

Chapter 5 provides Monte–Carlo simulation results which essentially demonstrate that the theoretical error performance predicted in Chapter 3 can be reproduced through computer simulation. The simulations were written with the assumption that finite quantization was applied to the received signal before it entered the decoder. As such, the first section of the chapter is devoted to determining the effects of quantization on system performance through a derivation

of the mean square distortion due to quantization. Results were obtained for both 6 and 8 bits of quantization. It has been shown that 8 bits of quantization are sufficient, for the particular code used, to guarantee the type of soft decision performance predicted.

Finally in Chapter 6, conclusions and suggestions for further research are given.

CHAPTER 2

THE GOSSET LATTICE A_8^3 AND ITS SPECTRUM

2.1 The A_8^3 Construction

There are a number of ways to construct the Gosset lattice, (see [5] for five different definitions), which apart from the choice of coordinates, and possibly a scale factor, are *equivalent*. By equivalent it is meant that the *kissing number of nearest neighbours*, the *average energy*, the *minimum distance* and the shape of the *Dirichlet region* are the same for all constructions of the Gosset lattice. Thus, although the method of assigning coordinates will differ for each construction, the underlying structure of the lattice remains the same. In mathematical terms, it is said that the lattice is *invariant* to any rotation of the coordinate axes.

In the context of coding and modulation, the invariance of the lattice implies that there is no advantage in terms of error performance in choosing one construction of the Gosset lattice over another. That is to say, the results of maximum likelihood decoding of two codes of equal size taken from the same region in $\mathbb{R}^8$, but from different constructions, must also be invariant. However, the difference in the coordinate systems between constructions necessarily implies that the algorithms for encoding and decoding will differ. The use of one construction of the Gosset lattice over another thus becomes strictly a design choice. The predominant criterion affecting this choice has been the simplicity of decoding. With the present work, the spectral characteristics of the code may also be considered a criterion. To this end. we introduce the following construction of the Gosset lattice due to Coxeter [18],[19],[20].

Definition: **The Gosset lattice in $\mathbb{R}^8$ is the lattice whose vertices, in an orthonormal coordinate system in $\mathbb{R}^9$, are the set of vectors whose 9 integer components**

i) are all congruent[2] modulo 3 to each other, and

ii) have a sum of zero.

Although the Gosset lattice is most commonly given the symbolic notation E_8 or Λ_8,[3] Coxeter [19] denoted this construction of the Gosset lattice as ${A_8}^3$ because i defines the lattice as the union of a sublattice $3A_8$ with two of its cosets,

$$ {A_8}^3 = 3A_8 \cup 3A_8 + (-2^3,1^6) \cup 3A_8 - (-2^3,1^6), \qquad (2.1) $$

where A_8 is the lattice of all 9–tuples of integers $z \in \mathbb{I}^9$ with a zero component sum [13] and $(-2^3,1^6)$ is a shorthand notation for the vector (–2,–2,–2,1,1,1,1,1,1). Relating equation (2.1) to the above definition, it can be seen that the sublattice $3A_8$ consists of all vectors whose 9 components are mutually congruent to 0 modulo 3, (or more simply, 9 integer multiples of 3). The cosets $3A_8 + (-2^3,1^6)$ and $3A_8 - (-2^3,1^6)$ account for the vectors whose components are mutually congruent, modulo 3, to +1 and –1 respectively. Of course, all the vectors also satisfy the zero sum condition of part (ii) of the definition.

The vectors of minimum length within a lattice are the vectors which join any point to its nearest neighbours. In the Gosset lattice there are 240 such minimal vectors. For the ${A_8}^3$ construction, they consist of the 72 vectors obtained by permuting the components of the vector (3,–3,0,0,0,0,0,0,0) and the 168 vectors obtained through all permutations of the components of (–2,–2,–2,1,1,1,1,1,1) and

---

[2] An integer x is said to be congruent modulo 3 to the integer j, ($j = 0$, 1, or –1), if and only if $x - j$ is a multiple of 3. Symbolically, this is denoted $x \equiv j \bmod 3$.

[3] In Appendix A, a set of linear transformations are given which link the three Gosset lattice constructions E_8, Λ_8, and ${A_8}^3$.

its negative (2,2,2,–1,–1,–1,–1,–1,–1). These minimal vectors all have a squared magnitude or energy of 18, which is also the minimum squared distance between any two ponts in this construction of the lattice.

The most intriguing feature of this construction of the Gosset lattice is the zero component sum of part (ii) of the definition. It is this restriction which reduces the configuration from 9 dimensions to 8 by forcing all the lattice vectors to lie within an 8–dimensional hyperplane in 9–space. As will be seen in later chapters, this zero sum restriction can be exploited in the designing of a demodulator and in the designing of a series of decoder look–up tables used to obtain the bit–sequence associated with a code vector. At present, the frequency spectrum of codes derived from this A_8^3 construction of the Gosset lattice will be investigated to determine how the zero sum restriction affects the spectrum. It has been shown [46] that any sequence whose *running digital sum* is uniformly bounded over all time has a null in its baseband spectrum at zero frequency or dc. If the components of any A_8^3 vector are interpreted as a sequence of real samples of a baseband signal, then the running digital sum of this sequence must go to zero with the transmission of the ninth component of the vector. Consequently, a code derived from this A_8^3 construction has a null at dc in its baseband spectrum. In the following section, this result will be derived mathematically.

2.2 Derivation of the Baseband Power Spectrum

As was stated in the introduction, a lattice code can be viewed as a decodable modulation in which sequences of independent data are mapped into a finite subset of the vectors of a lattice. Let $\{x_m, -\infty < m < \infty\}$ denote the sequence of transmitted code vectors. During the time interval $mT_n \le t < (m+1)T_n$, the components of the code vector $x_m = (x_{m,1}, x_{m,2},\ldots,x_{m,n})$ are transmitted at T

second intervals, where T is the *symbol duration*, $T_n = nT$ is the *block duration* or period required to transmit one vector and n is the *block length*. The baseband signal $s(t)$ associated with such a transmission is expressed as [10],[12]

$$s(t) = \sum_{m=-\infty}^{+\infty} \sum_{i=1}^{n} x_{m,i}\, g(t - mT_n - (i-1)T), \tag{2.2}$$

where $g(t)$ is the pulse shape used in transmission. To determine the power spectral density of the signal $s(t)$, it is easiest to first find the autocorrelation function $R_s(\tau) = E\{s(t)\, s(t+\tau)\}$ of the signal and then obtain the spectral density $S(f)$ by taking the Fourier transform of the autocorrelation function. Finding the autocorrelation function is not exactly straightforward, however, because the signal is not *stationary* but only *periodically stationary* (or *cyclostationary*).[4] Bennett [60] derived an expression for the autocorrelation function of a cyclostationary process, and subsequently its power spectral density, by assuming that the codeword sequence $\{x_m\}$ is *wide-sense stationary* and by introducing a random "phase" into the shaping pulse $g(t)$ that is uniformly distributed over the block duration T_n. The signal $s(t)$ then appears wide-sense stationary since the observer of $s(t)$ knows only that the codewords are emitted at equal time intervals T_n but is completely ignorant of the time instant at which they are emitted. The autocorrelation function $R_s(\tau)$ of the signal $s(t)$ can then be shown [12],[60] to equal

$$R_s(\tau) = \frac{1}{T_n} \sum_{k=-\infty}^{+\infty} \sum_{i,\,l=1}^{n} R_k(i,l)\, R_g(\tau - kT_n - (l-i)T) \tag{2.3}$$

---

[4] A random process is said to be *strictly stationary* if its statistical properties are invariant to a shift of the time origin. If the random process has a constant mean, its autocorrelation function has a finite value at a lag τ of zero and the autocorrelation function is invariant to a shift of the time origin, then the process is said to be *wide-sense stationary*. In the present case, although the statistics of the signal $s(t)$ vary from symbol to symbol, they are invariant to shifts of the time origin that are integral multiples of the block duration T_n. The signal is thus classified as *periodically stationary* or *cyclostationary*. For further details on the properties and classification of such processes see [45, pp. 219–231].

where $R_k(i,l)$ is the (i,l) entry in the $n \times n$ codeword correlation matrix

$$R_k = E[x_0^{\prime} x_k], \tag{2.4}$$

and $R_g(\tau)$ is the autocorrelation function of the shaping pulse,

$$R_g(\tau) = \int_{-\infty}^{+\infty} g(t)\, g(t+\tau)\, dt\,. \tag{2.5}$$

If $G(f)$ denotes the Fourier transform of the shaping pulse $g(t)$, then the Fourier transform of its autocorrelation function $R_g(\tau)$ is

$$F\{R_g(\tau)\} = |G(f)|^2\,. \tag{2.6}$$

By taking the Fourier transform of equation (2.3), and substituting equation (2.6), the power spectral density $S(f)$ of the signal $s(t)$ is given by [12]

$$\begin{aligned} S(f) &= |G(f)|^2 \sum_{k=-\infty}^{+\infty} \sum_{i,l=1}^{n} R_k(i,l)\, exp(-j2\pi f[kT_n+(l-i)T]) \\ &= |G(f)|^2 \sum_{k=-\infty}^{+\infty} \frac{1}{T_n} exp(-j2k\pi fT_n)\, V(f) R_k V^{\prime\prime}(f), \end{aligned} \tag{2.7}$$

where $V(f)$ is the n–dimensional row vector,

$$V(f) = [exp(j2\pi fT_n/n), exp(j4\pi fT_n/n), \ldots, exp(j2\pi fT_n)], \tag{2.8}$$

$^{\prime}$ denotes transpose and $^{\prime\prime}$ denotes hermitian or complex conjugate transpose. Of more interest to the present discussion is the code spectrum $C(f)$ which may be obtained through dividing $S(f)$ by the spectral density of the shaping pulse, i.e.,

$$C(f) = \frac{S(f)}{|G(f)|^2} = \sum_{k=-\infty}^{+\infty} \frac{1}{T_n} exp(-j2k\pi fT_n)\, V(f) R_k V^{\prime\prime}(f)\,. \tag{2.9}$$

Since data is mapped into each vector independently, the transmission of any vector x_m is completely independent of (or *uncorrelated* with) all other transmissions. As a consequence, the only correlation matrix which is non–zero is R_0 and equation (2.9) reduces to

$$C(f) = \frac{1}{T_n} V(f) R_0 V(f)\,. \tag{2.10}$$

If a lattice code is loosely defined to be the set of all vectors lying on the

surface of or in the interior of some hypersphere,[5] then it can be quite easily shown that the 9 × 9 correlation matrix R_0 of the A_8^3 construction of this code is

$$R_0 = \frac{1}{9}E_a \begin{bmatrix} 1 & -\frac{1}{8} & -\frac{1}{8} & -\frac{1}{8} & -\frac{1}{8} & -\frac{1}{8} & -\frac{1}{8} & -\frac{1}{8} & -\frac{1}{8} \\ -\frac{1}{8} & 1 & -\frac{1}{8} & -\frac{1}{8} & -\frac{1}{8} & -\frac{1}{8} & -\frac{1}{8} & -\frac{1}{8} & -\frac{1}{8} \\ -\frac{1}{8} & -\frac{1}{8} & 1 & -\frac{1}{8} & -\frac{1}{8} & -\frac{1}{8} & -\frac{1}{8} & -\frac{1}{8} & -\frac{1}{8} \\ -\frac{1}{8} & -\frac{1}{8} & -\frac{1}{8} & 1 & -\frac{1}{8} & -\frac{1}{8} & -\frac{1}{8} & -\frac{1}{8} & -\frac{1}{8} \\ -\frac{1}{8} & -\frac{1}{8} & -\frac{1}{8} & -\frac{1}{8} & 1 & -\frac{1}{8} & -\frac{1}{8} & -\frac{1}{8} & -\frac{1}{8} \\ -\frac{1}{8} & -\frac{1}{8} & -\frac{1}{8} & -\frac{1}{8} & -\frac{1}{8} & 1 & -\frac{1}{8} & -\frac{1}{8} & -\frac{1}{8} \\ -\frac{1}{8} & -\frac{1}{8} & -\frac{1}{8} & -\frac{1}{8} & -\frac{1}{8} & -\frac{1}{8} & 1 & -\frac{1}{8} & -\frac{1}{8} \\ -\frac{1}{8} & -\frac{1}{8} & -\frac{1}{8} & -\frac{1}{8} & -\frac{1}{8} & -\frac{1}{8} & -\frac{1}{8} & 1 & -\frac{1}{8} \\ -\frac{1}{8} & -\frac{1}{8} & -\frac{1}{8} & -\frac{1}{8} & -\frac{1}{8} & -\frac{1}{8} & -\frac{1}{8} & -\frac{1}{8} & 1 \end{bmatrix} \tag{2.11}$$

where E_a is the average energy of the code vectors. Substituting the correlation matrix (2.11) into equation (2.10), the power spectrum of the code, $C(f)$, can be expressed as

$$\begin{aligned} C(f) &= \frac{E_a}{T_n}\left[\, 1 - \frac{1}{9}\sum_{k=1}^{8}\frac{9-k}{8}\left\{exp(j2k\pi fT_n/9) + exp(-j2k\pi fT_n/9)\right\}\,\right] \\ &= \frac{E_a}{T_n}\left[\, 1 - \frac{1}{9}\sum_{k=1}^{8}\frac{9-k}{4}\cos(2k\pi fT_n/9)\,\right] \end{aligned} \tag{2.12}$$

where $T_n = 9T$ is the block duration for the A_8^3 code. The summation of cosine terms on the right of equation (2.12) can be reduced using Chebyshev polynomials of the first and second kind [1]. Complete details of this reduction are left to Appendix B and only the result is stated here: The power spectrum of an A_8^3 code with correlation matrix (2.11) is given by

---

[5] Note that for the A_8^3 code, this defines some code on or interior to an 8–dimensional hypersphere that lies within a hyperplane in 9–space. The analogous situation in 3–space would be to define a lattice whose points all lie in some planar cross–section of 3–space. The lattice code would then be all points on or interior to some circle in the plane.

$$C(f) = \frac{E_a}{T_n}\left[\frac{9}{8} - \frac{1}{72}\left[\frac{\sin(\pi f T_n)}{\sin(\pi f T_n/9)}\right]^2\right]. \qquad (2.13)$$

The baseband spectrum of equation (2.13) is shown in Figure 2.1, normalized in amplitude with respect to the average power E_a/T_n of one block and in time with respect to twice the block duration T_n. If the spectrum shown in Figure 2.1 is compared with that of Figure 1.3, we see that the codes derived from the A_8^3 construction of the Gosset lattice exhibit similar periodic nulls to those observed in a $1-D$ partial response filtered sequence. Codes derived from A_8^3 can thus be said to fit into the $1-D$ partial response channel or, equivalently, to belong to the class of *dc free codes* [39].

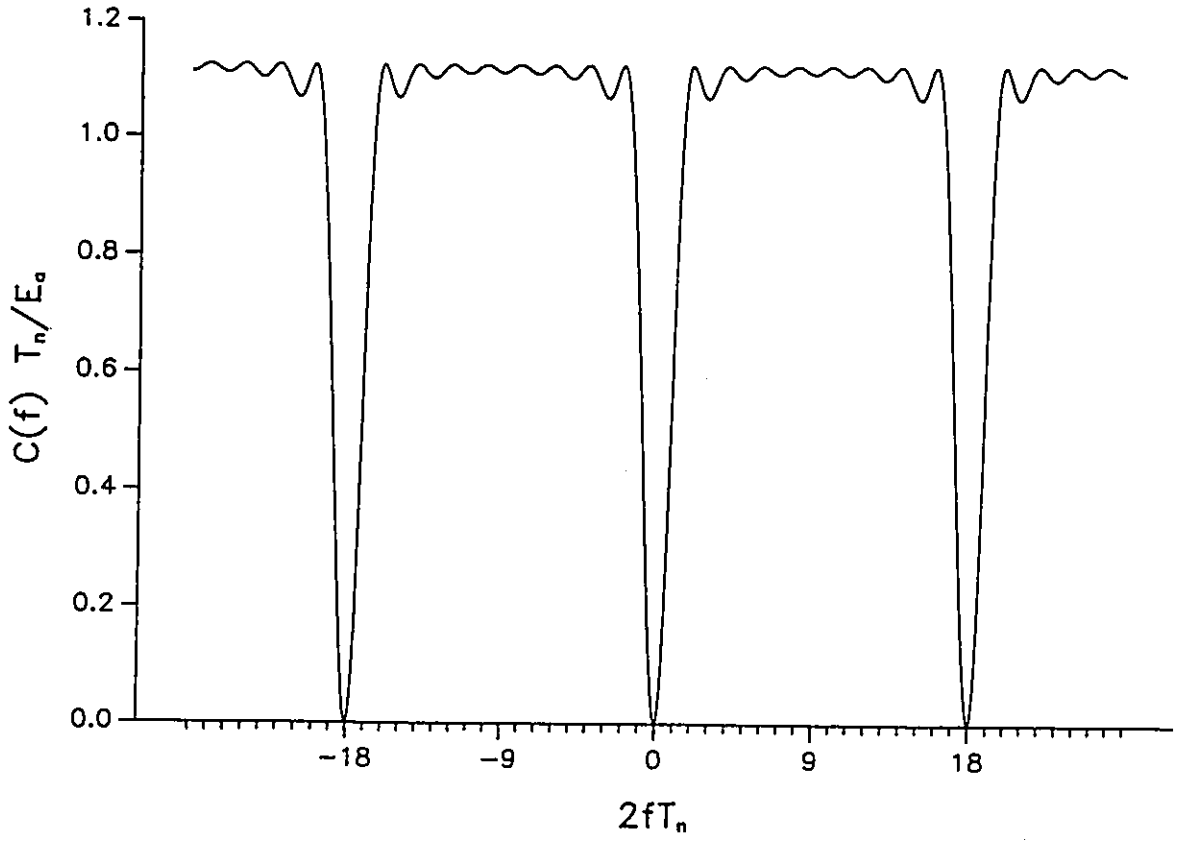

Figure 2.1: Spectrum of an A_8^3 Gosset lattice code.

A simple code for which the validity of equations (2.11) and (2.13) is easily checked is the code made up of the 240 minimal vectors defined in the previous section. Such a selection of points is symmetrically distributed throughout the space and consequently results in a symmetric correlation matrix. In general, a particular configuration of M points, selected to encode $log_2 M$ bits per vector, will not be as symmetric as choosing all vectors on the surface of or interior to some hypersphere. Consequently, the off-diagonal and diagonal elements of the correlation matrix R_0 will not all be identical as they are in (2.11). The correlation matrix will still be symmetric about its main diagonal since for any $i \neq l$, $i,l = 1,2,\ldots,9$

$$R_0(i,l) = E[x_{0,i}\, x_{0,l}] = E[x_{0,l}\, x_{0,i}] = R_0(l,i)\,.$$

Because the correlation matrix is symmetric, the code spectrum equation will reduce to a summation of cosines as in equation (2.12), but with a much different set of scalar coefficients than the values of $\frac{9-k}{4}$ seen in (2.12). The difference that one will see in the spectrum as a result of a change in the coefficients of the cosines will be a slightly greater overshoot and more pronounced ripples across the frequency band. The width of the spectral null will, however, remain the same because it is inversely proportional to the length of time required for the running digital sum to go to zero [10],[29]. Since the lattice construction forces the running digital sum to zero with the transmission of the ninth component of any vector, the width of the spectral null will remain fixed despite variations in the code.

2.3 Comparison with the Spectra of Other Gosset Lattice Codes

The periodic nulls observed in the code spectrum of this A_8^3 construction of a Gosset lattice is a property not observed in codes derived from other constructions of the lattice, which have flat spectra [10]. A worthwhile exercise would thus be to

compare the signal spectrum of an A_8^3 Gosset lattice code with that of an E_8 construction of the same code to determine their individual bandwidth requirements. As discussed in section 2.1, the invariance of the lattice to a rotation of the coordinate axes ensures that the results of maximum likelihood decoding these two codes will be identical. Thus for the comparison to be fair, the mean energy E_a and the block duration T_n of the two codes must be the same. Satisfying the former condition is simply a matter of appropriate scaling. Satisfying the latter condition does however pose a bit of a problem in that the two codes have unequal block lengths; the A_8^3 code vectors consist of 8 independent samples plus a further dependent sample, while the E_8 code vectors consist of only 8 independent samples. If the block durations T_n are to be equal, the 9 samples of the A_8^3 code vector must be transmitted within the same amount of time alotted to transmit the 8 samples of the E_8 construction of the code. This necessitates that the sample spacings for the A_8^3 code vectors be smaller than those for the E_8 code. If T is the symbol duration for the A_8^3 code, and T' is the symbol duration for the E_8 code, then for equal block durations T_n, $T = T_n/9$ and $T' = T_n/8$. If a sinc pulse of unit energy is to be used as the shaping pulse $g(t)$ then, because of the difference in symbol durations, $|G(f)|^2 = T = T_n/9$ for the A_8^3 code ($|f| \leq 9/2T_n$) and $|G'(f)|^2 = T' = T_n/8$ for the E_8 code ($|f| \leq 8/2T_n$).

An E_8 construction of the Gosset lattice that has been used in several applications [14],[23],[32],[49] is the following.

Definition: The Gosset lattice in $\mathbb{R}^8$ is the set of all vectors of 8 integers with an even sum or 8 halves of odd integers with an even sum.

The lattice of all n–tuples of integers with an even sum is known as the

checkerboard lattice D_n [17, p.117]. In this construction therefore, the Gosset lattice E_8 is the union of D_8 with one of its cosets,

$$E_8 = D_8 \cup D_8 + (\tfrac{1}{2},\tfrac{1}{2},\tfrac{1}{2},\tfrac{1}{2},\tfrac{1}{2},\tfrac{1}{2},\tfrac{1}{2},\tfrac{1}{2}) . \tag{2.14}$$

The 240 minimal vectors of this E_8 construction consist of the 112 vectors obtained through sign changes and permutations made to components of the vector (1,1,0,0,0,0,0,0) and the 128 vectors obtained through an even number of sign changes made to the vector $(\frac{1}{2},\frac{1}{2},\frac{1}{2},\frac{1}{2},\frac{1}{2},\frac{1}{2},\frac{1}{2},\frac{1}{2})$. These 240 minimal vectors have a squared magnitude of 2, which is also the minimum squared distance between any two points in this construction of the lattice. If a code is again loosely defined to consist of all vectors on the surface of or interior to some hypersphere, the 8×8 correlation matrix R'_0 of the E_8 version of this code will be

$$R'_0 = \tfrac{1}{8} E_a I , \tag{2.15}$$

where I is an 8×8 identity matrix and E_a is the mean energy of the code. The spectrum of the E_8 code is then

$$C'(f) = \frac{E_a}{T_n} , \tag{2.16}$$

which is flat over the entire spectrum as we expected. After pulse shaping, the signal spectrum is given by,

$$S'(f) = \frac{E_a}{T_n} \cdot T' = \tfrac{1}{8} E_a , \qquad |f| \le \frac{8}{2T_n} . \tag{2.17}$$

For the $A_8{}^3$ construction as defined in section 2.1, the 240 minimal vectors have a squared madnitude of 18, which is a factor of 9 larger than the equivalent set of E_8 vectors defined above. Consequently, if the two constructions are to be fairly compared, the $A_8{}^3$ vectors must be scaled by $\frac{1}{3}$ so that two equivalent codes will have the same average energy E_a. Returning to equation (2.13), by factoring $\frac{9}{8}$ out of the brackets and multiplying by $|G(f)|^2 = T = T_n/9$, the signal spectrum $S(f)$ for the $A_8{}^3$ code is arrived at,

$$S(f) = \frac{1}{8}E_a\left[1 - \left[\frac{\sin(\pi f T_n)}{9\sin(\pi f T_n/9)}\right]^2\right], \qquad |f| \le \frac{9}{2T_n}. \tag{2.18}$$

The baseband spectra $S(f)$ and $S'(f)$ are shown in Figure 2.2, again normalizing in time with respect to twice the block duration T_n and in amplitude with respect to one-eighth the average energy E_a. This comparison is not exceptionally revealing. Although the $A_8{}^3$ signal spectrum covers a wider range of frequencies, the two spectra have the same area due to the null at zero frequency in $S(f)$. Unfortunately, this fact does not come across very well in Figure 2.2. A better comparison of the spectral occupancy is obtained by manipulating the $A_8{}^3$ baseband spectrum to produce a bandpass spectrum with nulls at the band edges. If we refer to Figure 2.1, this would mean taking the portion of the code spectrum from 0 to 18 in normalized frequency and placing it symmetrically about some centre frequency f_c. There are a numbers of ways of accomplishing this feat. One approach is to take the baseband signal in Figure 2.2 and single sideband modulate [34] it twice; the first single sideband modulator uses a carrier frequency of $f_c - \frac{9}{2T_n}$ and keeps the upper sideband, while the second single sideband modulator uses a carrier frequency of $f_c + \frac{9}{2T_n}$ and keeps the lower sideband. This has the effect of reversing the positive and negative frequency portions of the baseband spectrum of Figure 2.2 and translating the result by f_c hertz. A second approach is to use a shaping pulse whose magnitude response $G(f)^2$ is still rectangular but covers the range of normalized frequencies from –18 to +18. The resultant baseband signal could then be single sideband modulated with a carrier of frequency $f_c - \frac{9}{2T_n}$ hertz and the upper sideband kept. These two approaches will yield the equivalent result.

Figure 2.3 compares this bandpass spectrum with the spectrum of the E_8 code that is obtained by product modulating the baseband signal with a sinusoidal carrier

of frequency f_c. Because of the smaller symbol duration of the A_8^3 code, the spacing between the nulls in the A_8^3 spectrum is necessarily 12.5% larger than the minimum spectral occupancy of the E_8 construction of the code. However, the two spectra have roughly the same 3 dB bandwidth. One might expect such a result since the spectral occupancy is governed by the number of independent samples in each block, which in both cases is 8. In other words, while the signaling rate has been increased to accomodate the 9 samples of each A_8^3 vector, the *effective* signaling rate has not changed because only 8 of these 9 samples are independent. With the invariance of the lattice ensuring identical error performance for equal average energy E_a and block duration T_n, the extra dependent sample associated with each A_8^3 vector simply provides the shaping that yields a more realizable spectrum than the "brick wall" spectrum of the equivalent E_8 code.

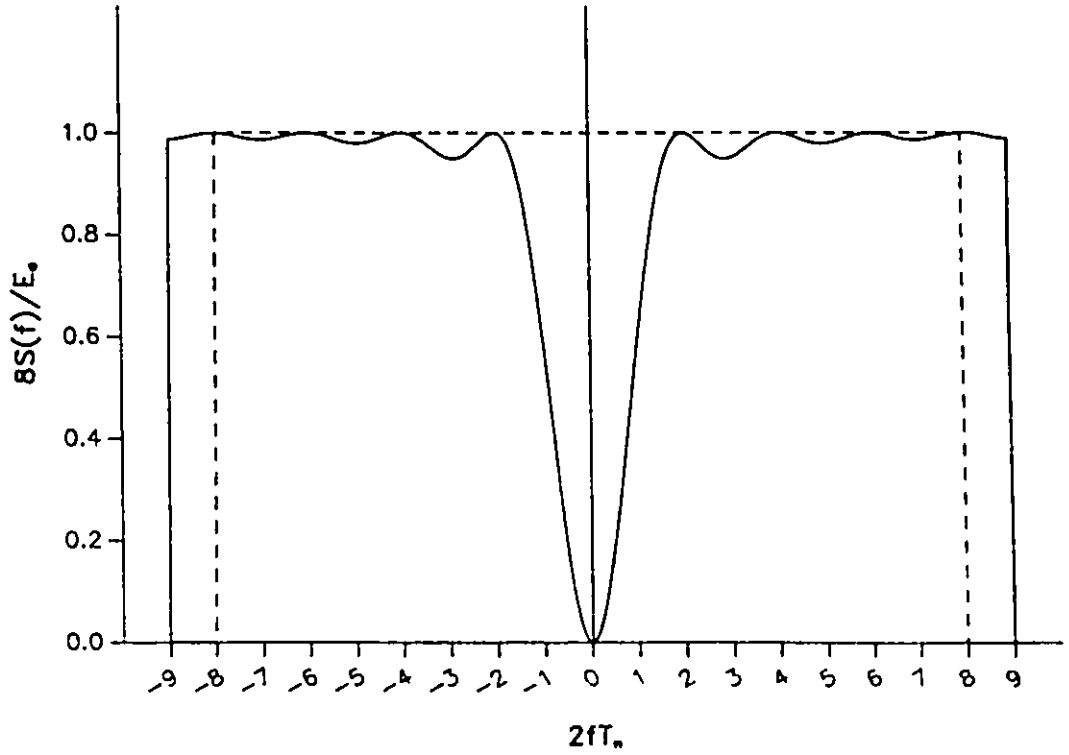

Figure 2.2: Baseband signal spectra of equivalent E_8 and A_8^3 versions of a Gosset lattice code.

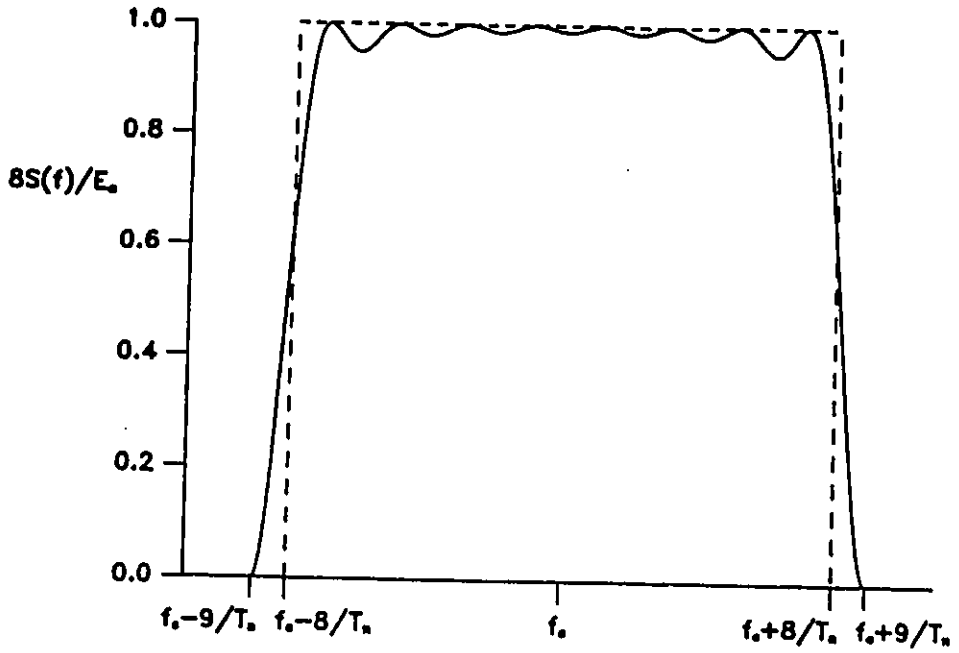

Figure 2.3: A comparison of spectral occupancy at bandpass.

Thus far, the comparisons that have been made against the A_8^3 spectrum have been on the basis of equal code size and block duration. If the criteria is changed slightly and a restriction placed on the signaling rate, then a whole different set of comparisons arise. When the A_8^3 code is constrained to have the same signaling rate (or symbol duration) as an E_8 code, the number of bits encoded per A_8^3 vector must be increased by a factor of $\frac{9}{8}$ over an E_8 code to compensate for the longer block length. An alternative statement is that if we wish to signal at a rate of k bits per transmitted sample, then a signal constellation of 2^{9k} A_8^3 vectors is required to maintain the data rate achieved by a 2^{8k} vector E_8 code. Obviously, there is a power penalty to be paid for this increase in the size of the signal constellation. Using equations 49 and 50 of [11] for the average power in a lattice code, the 2^k expansion factor translates to an increase in average power of

$$10\log_{10}\left[\frac{8}{9}\cdot 2^{k/4}\right] \approx 0.75k - 0.5 \text{ dB}, \tag{2.19}$$

where the factor $\frac{8}{9}$ takes into account that the power in an A_8^3 code is spread over 9 samples instead of 8. It is obvious that one begins to lose a large amount of power as the number of bits per sample increases. However, if one considers that there is a 3 dB loss in signal–to–noise ratio associated with $1 - D$ partial response filtering of pulse amplitude signals, and that the asymptotic coding gain of Gosset lattice codes is 3 dB over pulse amplitude modulation, there are still obvious advantages to using A_8^3 lattice codes for low numbers of bits per sample. Most of the 3 dB in signal–to–noise ratio lost through $1 - D$ partial response filtering can be regained if maximum likelihood sequence estimation [26] is employed. We would, however, contend that maximum likelihood sequence estimation is on the same order of complexity as soft decision decoding of an A_8^3 code and certainly much slower than the demodulation scheme to be discussed in the next chapter.

In Chapter 1, we mentioned that Calderbank, Lee and Mazo [10] have used running digital sum feedback to induce a dc null in the baseband spectrum of n–dimensional trellis codes. This method does not require the signaling rate to be increased, only a doubling of the signal constellation so that for every vector $\boldsymbol{x}$ in the code, $-\boldsymbol{x}$ is also in the code. For this doubling of the signal constellation, they pay a power penalty of $6/n$ dB, which for E_8 Gosset lattice codes translates to a constant power penalty of 0.75 dB. Obviously a comparison, based on equal signaling rate, of our A_8^3 lattice codes with these trellis codes favours the latter since Calderbank, Lee and Mazo pay a fixed penalty in power, whereas equation (2.19) indicates that our power penalty increases with the size of the signal set. However, if we return to our original argument of constraining the block durations to be equal, there is a fixed expansion of the bandwidth by a factor of $\frac{9}{8}$ for A_8^3 codes but no penalty in power, regardless of the code size. Thus the better tradeoff for A_8^3 lattice codes is

to allow for the bandwidth expansion and suffer no penalty in error performance. Clearly, one has to be careful in comparing the trellis codes of [10] with our A_8^3 lattice codes since, as the saying goes, it is a little like comparing apples with oranges. They are otherwise comparable from the point of view of spectral shape. The spectrum of the E_8 trellis code in Figure 10 of [10] exhibits a slightly larger overshoot and a sharper rise from its null at dc than is seen in the A_8^3 baseband spectrum of Figure 2.2.

CHAPTER 3

A TWO-STAGE APPROACH TO DEMODULATION

3.1 The L_9 Configuration and Its Demodulation

In the previous chapter, the A_8^3 construction of the Gosset lattice was defined as the set of all vectors whose 9 components are mutually congruent modulo 3 and have a sum of zero. Consider for a moment the first part of this definition. It is possible to construct a 9–dimensional lattice of vectors whose 9 components are mutually congruent modulo 3 by forming the union of a scaled copy of the integer lattice $\mathbb{Z}^9$ (namely $3\mathbb{Z}^9$) with two of its cosets. Let L_9 denote this lattice configuration, defined by

$$L_9 = 3\mathbb{Z}^9 \cup 3\mathbb{Z}^9 + (1^9) \cup 3\mathbb{Z}^9 - (1^9) \tag{3.1}$$

where $(1^9) = (1,1,1,1,1,1,1,1,1)$. The sublattice $3\mathbb{Z}^9$ of L_9 contains all those vectors whose components are congruent to 0 (mod 3). The cosets $3\mathbb{Z}^9 + (1^9)$ and $3\mathbb{Z}^9 - (1^9)$ then contain the vectors whose components are congruent to +1 (mod 3) and –1 (mod 3) respectively.

It is quite easy to see that the Gosset lattice A_8^3 consists of all vectors in L_9 whose components sum to zero. In other words, A_8^3 is a *sublattice* of L_9. Because of the simplicity of the L_9 configuration, demodulating L_9 provides a good starting point in the demodulation of A_8^3.

To demodulate the received signal vector $r \in \mathbb{R}^9$ to the closest point in L_9, we must determine the closest point in each of the sublattice $3\mathbb{Z}^9$ and the two cosets $3\mathbb{Z}^9 + (1^9)$, $3\mathbb{Z}^9 - (1^9)$. This can be easily accomplished throught the use of

quantization functions or look–up tables. Consider the quantization function $f(x)$, shown in Figure 3.1, which for any real x gives the closest multiple of 3. If this quantization function is applied to each component r_i of the received vector r then the resultant set of outputs $f(r_i)$, $i = 1,...,9$, will be the components of the closest vector in the sublattice $3\mathbb{Z}^9$. The closest point in the coset $3\mathbb{Z}^9 + (1^9)$ can be obtained by using the quantization function $f(x)$ to find the nearest multiple of 3 to $r_i - 1$ and then adding one to this result. Similarly, the closest point in the coset $3\mathbb{Z}^9 - (1^9)$ is specified by the 9 quantities $f(r_i + 1) - 1$. The three quantization functions, $f(x)$, $f(x-1)+1$, $f(x+1)-1$, are shown together in Figure 3.2. Should the received components r_i have been quantized before reaching this stage, the quantization functions shown in Figure 3.2 would be implemented as look–up tables.

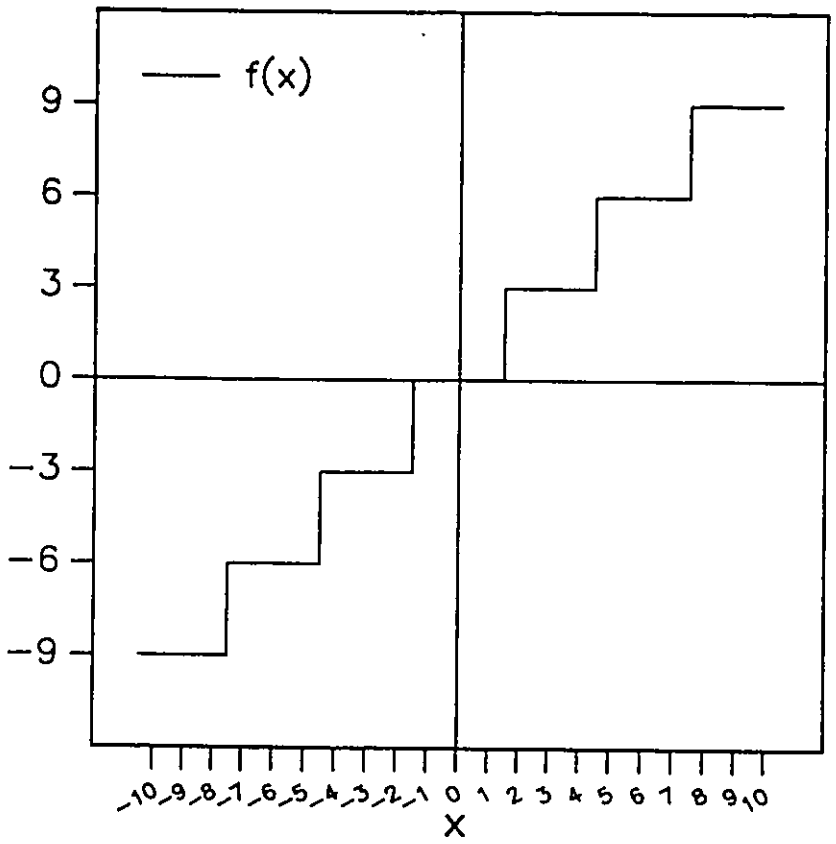

Figure 3.1: A quantization function $f(x)$ which for any real x provides the nearest integer multiple of 3.

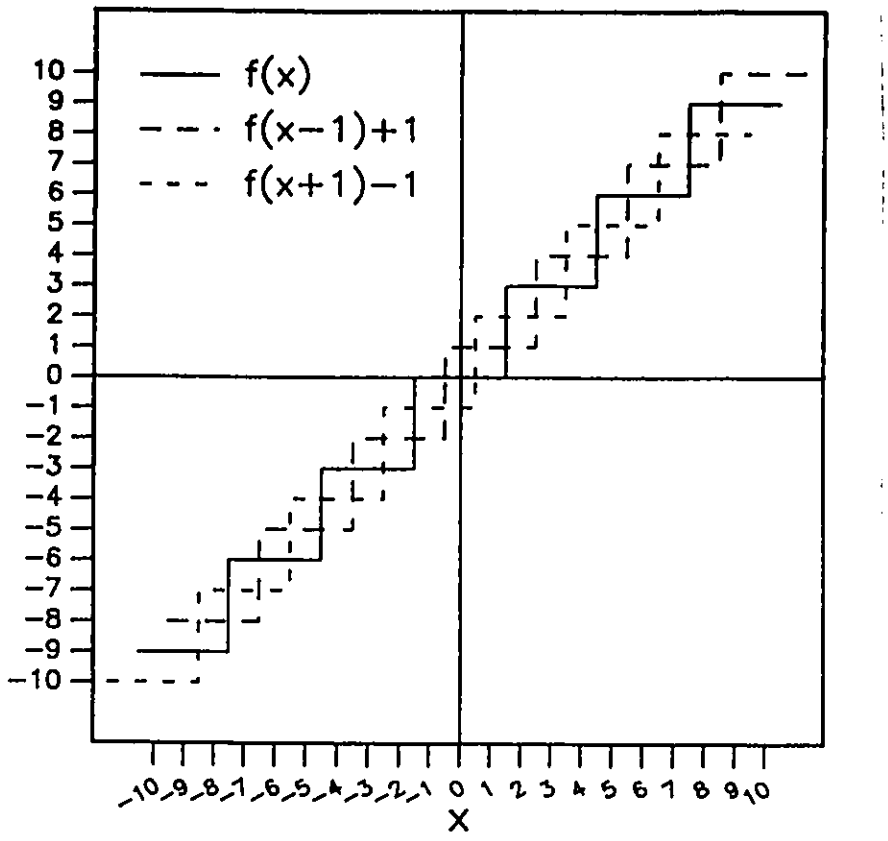

Figure 3.2: Quantization functions $f(x)$, $f(x-1)+1$ and $f(x+1)-1$.

Given the candidate vectors from the sublattice $3\mathbb{Z}^9$ and the cosets $3\mathbb{Z}^9 + (1^9)$, $3\mathbb{Z}^9 - (1^9)$, the closest point in L_9 is determined by that vector of the three which is closest to the received vector $\mathbf{r}$ in the sense of minimum squared Euclidean distance. This involves the computation of three *metrics*, (the squared magnitude of the difference vectors between the received vector and each of the candidate vectors), followed by a three way comparison to determine the smallest of these values. The number of metrics can be reduced from three to two through the following observations.

Let $3z_i$ denote the i^{th} component of the candidate point in the sublattice $3\mathbb{Z}^9$;

in other words, $3z_i = f(r_i)$. When $3z_i$ is the closest multiple of 3 to r_i, the i^{th} component of the candidate point in $3\mathbb{Z}^9 + (1^9)$ must be either $3z_i + 1$ or $3z_i - 2$, whichever is closer to r_i. The contribution to the squared distance made by this component is then the lesser of $|r_i - 3z_i - 1|^2$ and $|r_i - 3z_i + 2|^2$. Subtract this contribution from the corresponding contribution in $3\mathbb{Z}^9$, namely $|r_i - 3z_i|^2$, and call the difference a_i. The quantity a_i is then the larger of

$$a_i = \begin{cases} |r_i - 3z_i|^2 - |r_i - 3z_i - 1|^2 = +1 \cdot (2r_i - 6z_i - 1) \\ |r_i - 3z_i|^2 - |r_i - 3z_i + 2|^2 = -2 \cdot (2r_i - 6z_i + 2) \end{cases} \tag{3.2}$$

where $|2r_i - 6z_i| \leq 3$. If a_i is plotted as a function of $r_i - 3z_i$, the discriminator function of Figure 3.3a is obtained. Note that this discriminator function has zero crossings at 0.5 and –1. These are the points at which the received component r_i is equidistant from the sublattice component $3z_i$ and the coset components $3z_i + 1$ and $3z_i - 2$ respectively.

By summing the contributions a_i, it is possible to determine whether the received vector r is closer to the point of the sublattice $3\mathbb{Z}^9$ or the candidate point in the coset $3\mathbb{Z}^9 + (1^9)$. Let a denote the sum of the contributions a_i, $a = \sum_{i=1}^{9} a_i$. Then if $a < 0$, the received vector r is closer to the point in the sublattice $3\mathbb{Z}^9$ than to the point of the coset $3\mathbb{Z}^9 + (1^9)$; for $a > 0$, the converse is true.

The same strategy can be applied to coset $3\mathbb{Z}^9 - (1^9)$. When $3z_i$ is the i^{th} component of the closest point in the sublattice $3\mathbb{Z}^9$, the i^{th} component of the candidate point in $3\mathbb{Z}^9 - (1^9)$ must be either $3z_i - 1$ or $3z_i + 2$. By subtracting the contribution to the squared distance made by this component from the contribution $|r_i - 3z_i|^2$, a distance quantity b_i can be obtained as the larger of

$$b_i = \begin{cases} |r_i - 3z_i|^2 - |r_i - 3z_i + 1|^2 = -1 \cdot (2r_i - 6z_i + 1) \\ |r_i - 3z_i|^2 - |r_i - 3z_i - 2|^2 = +2 \cdot (2r_i - 6z_i - 2) \end{cases} \tag{3.3}$$

This discriminator function b_i is shown in Figure 3.3*b*. Let b denote the sum of the contributions b_i, $b = \sum_{i=1}^{9} b_i$. Then if $b < 0$, the point of the sublattice $3\mathbb{Z}^9$ is closer to the received vector r than is the point in the coset $3\mathbb{Z}^9 - (1^9)$, and vice versa for $b > 0$.

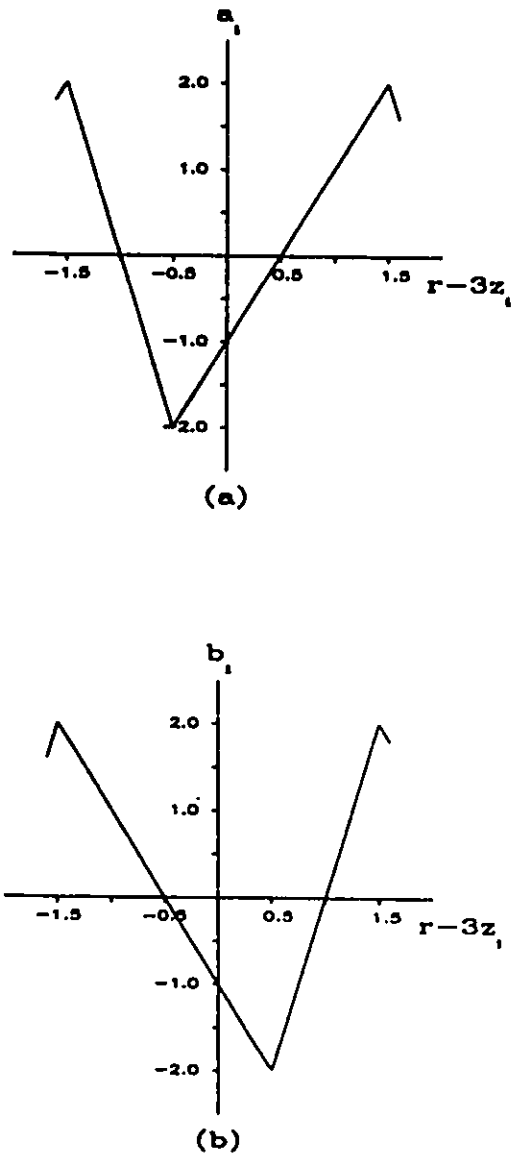

Figure 3.3: Discriminator functions a_i and b_i

The final decision as to which candidate vector is closest to the received vector r, and thus the closest point in L_9, reduces to determining the largest among 0, a and b. Being single valued functions of r_i alone, the discriminator functions a_i and b_i can be easily implemented in their analog form, Figure 3.3, or for digital input, as look–up tables. Figure 3.4 shows quantized forms of the discriminator function a_i for 6 and 8 bits of quantization and assuming that components of the vectors take on values between –6 and +5, (i.e. 13 amplitude levels).[6] Since the quantization functions of Figure 3.2 and the discriminator functions of Figure 3.3 are all functions of the real received components r_i alone, they can be implemented in parallel. In other words, it is possible to simultaneously look–up the components of the three candidate vectors and the distance contributions a_i and b_i, making for a very fast pipelined architecture.

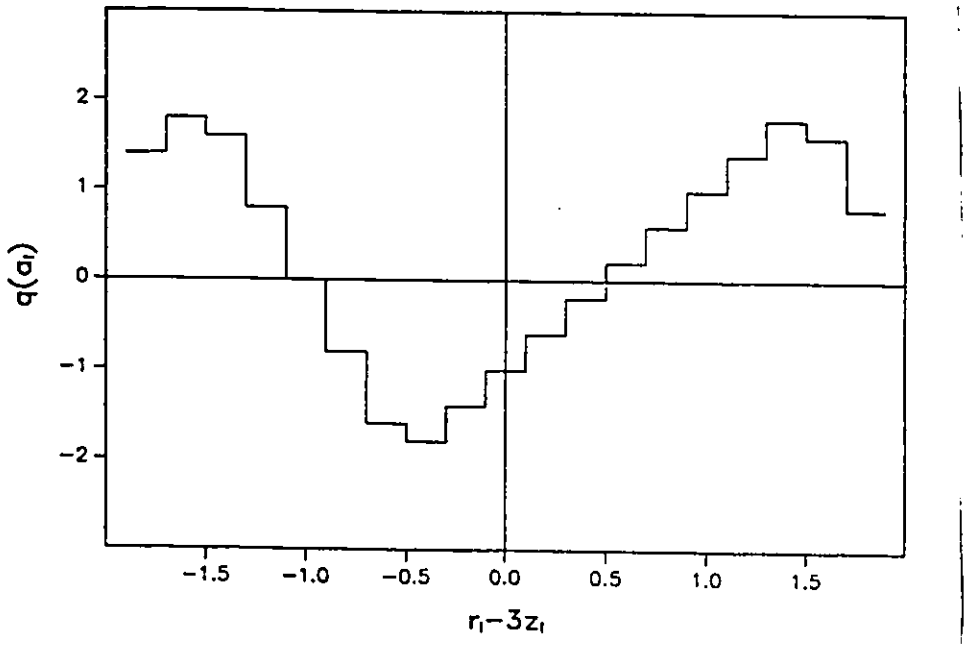

Figure 3.4*a*: A quantized form of the discriminator function a_i assuming 13 modulation levels and 6 bits of quantization

[6] This range of levels is sufficient for a 2^{16} vector code as will be outlined in Chapter 4. The quantized discriminator functions also appear again in Chapter 5 where the effects of quantization on system performance are discussed.

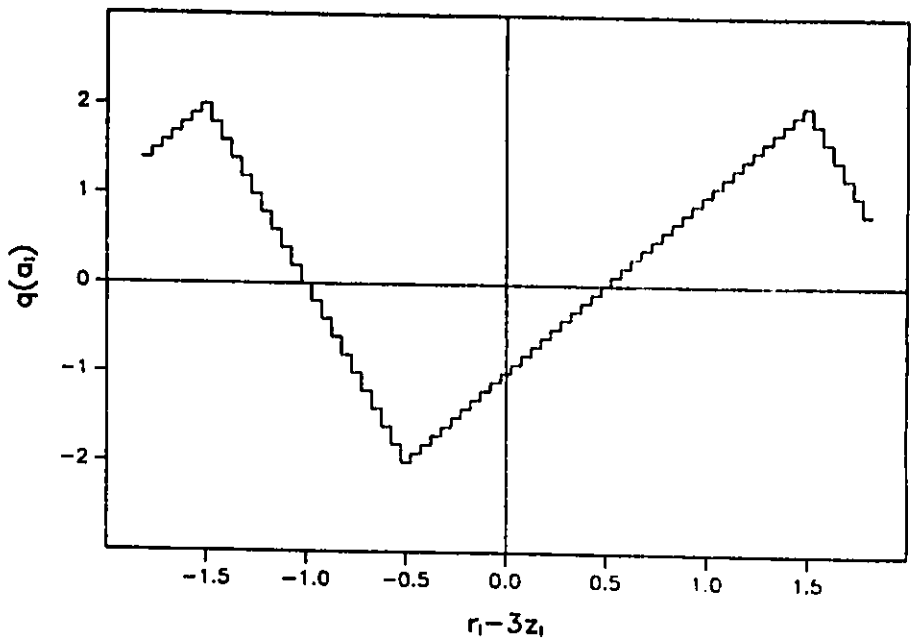

Figure 3.4*b*: A quantized form of the discriminator function a_i assuming 13 modulation levels and 8 bits of quantization

3.2 The First Stage in the Demodulation of the Gosset Lattice A_8^3

At the outset of this chapter, the L_9 configuration was defined and it was stated that the Gosset lattice A_8^3 is the sublattice of all L_9 vectors whose components sum to zero. It follows that if the components of the vector chosen by the above demodulation algorithm for L_9 sum to zero, this must also be the closest point in the Gosset lattice A_8^3. If another point in the Gosset lattice were closer to r, this would be a closer point in L_9 than that provided by the maximum likelihood solution, an impossiblity. Consequently, if an additional step is added to the above algorithm to check the component sum of the demodulated L_9 vector, this algorithm could be used as a stand alone initial stage in the demodulation of A_8^3. Should the components of the demodulated vector not sum to zero, the option is available to declare an *erasure* and allow a separate erasure correction circuit to find the closest point in the Gosset lattice. This two–stage approach to demodulation is illustrated by the block diagram of Figure 3.5.

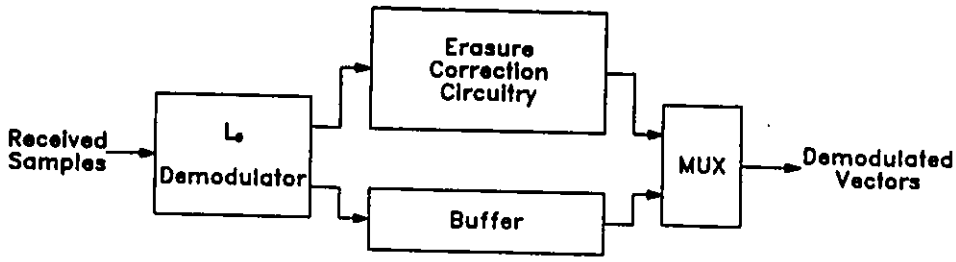

Figure 3.5: Block diagram of the proposed two–stage demodulator.

The zero component sum of the A_8^3 lattice vectors has in this instance been used as a form of parity check. By demodulating over the simplified L_9 configuration, we are estimating what the transmitted A_8^3 vector was but avoiding the time consuming task of forcing each candidate vector to satisfy the zero sum condition. If the demodulated L_9 vector satisfies the parity or zero sum condition, then we know that this is the best estimate of the transmitted vector. If the parity condition is not satisfied, then the best estimate of the transmitted vector is still unknown. The declaration of an erasure indicates that the estimate of the transmitted vector provided by the L_9 demodulator is unreliable and that additional demodulation steps are required.

By declaring an erasure, the circuit performing the L_9 algorithm is freed to demodulate the next received vector. This makes it possible to operate the L_9 demodulator and the erasure correction circuit simultaneously, provided the outputs of the L_9 circuit are buffered. The buffer allows the correction algorithm to operate "off–line" to find the closest point in the Gosset lattice A_8^3, thus eliminating the need to halt the L_9 algorithm when an erasure occurs. The erased vector can then be replaced as it leaves the buffer.

The advantage to this two–stage approach to demodulation is that its speed is determined by the L_9 demodulator. Extensive use of parallelism within the L_9 algorithm makes it possible to limit the number of operations to 19 per demodulated vector: 9 table look–ups to find a_i, b_i and the components of the 3 candidate vectors; 8 additions to sum, in parallel, the a_i, the b_i and the components of the 3 candidate vectors; 1 greatest of 3 operation to find the largest of 0, a and b; and 1 verification that the components of the demodulated vector sum to zero. With just over 2 steps per component, the speed of the L_9 demodulator approaches that of a hard decision demodulator.

Given that it is the L_9 algorithm which determines the overall speed into and out of the demodulator depicted in Figure 3.5, the 19 steps required by this algorithm is the number to use when comparing other approaches to demodulation with that presented here. Table 3.1 lists all the presently known decoding/demodulation algorithms for the Gosset lattice. These algorithms, with the exception of the present one, are based upon two well known constructions of the Gosset lattice. The algorithms of [14], [32] and [49] use the E_8 construction,

$$E_8 = D_8 \cup D_8 + (\tfrac{1}{2},\tfrac{1}{2},\tfrac{1}{2},\tfrac{1}{2},\tfrac{1}{2},\tfrac{1}{2},\tfrac{1}{2},\tfrac{1}{2}) , \tag{3.5}$$

previously defined in Chapter 2. The algorithms of [4], [15] and [28], on the other hand, use the construction of the Gosset lattice as a member of the Barnes–Wall series [17, pp. 139,234]

$$\Lambda_8 = c + 2z \tag{3.6}$$

where c is one of the 16 codewords of the (8,4) first order Reed–Muller code and $z \in \mathbb{Z}^8$. The number of steps to demodulate/decode are the figures published by the authors, except for [4] and [32] where none were available. The number of decoding operations quoted for the present work includes the 13 steps needed to find the index of a vector using the decoder tables described in Chapter 4.

Algorithms	Number of Steps to demodulate	decode	Version of the Gosset lattice	Region of the Code
P. de Buda, 1981 [4]	—	345 [1]	Λ_8	4^{th} shell
Conway–Sloane, 1982 [14]	104	—	E_8	infinite
Gersho–Lawrence, 1984 [32]	—	> 345 [2]	E_8	shells 1 to 6
Conway–Sloane, 1986 [16]	72	—	Λ_8	infinite
Secord–R. de Buda, 1986 [49]	—	282 [3]	E_8	4^{th} shell
Forney 1988 [27],[28]	47	—	Λ_8	infinite
Secord–R. de Buda, 1989 [50]	19	32	A_8^3	(*)

(*) – intersection of the hyperplane $\sum_{i=1}^{9} x_i = 0$ with the interior of a 9–cube.

Notes:

1. In [4], the number of steps is not explicitly stated, however, sufficient detail of the decoding operations is given to estimate the number of decoding steps. The figure given above assumes that 45 table look–ups are assigned to the third operation in the algorithm.

2. Both [4] and [32] decode by viewing the code as an ensemble of group codes. Since the code of [32] consists of 20 group codes, while the code of [8] contains only 4, the number of decoding steps for [32] must be greater than that for [8].

3. The number of steps quoted is the number of 6502 machine instructions in the longest path through the algorithm.

4. A shell refers to the set of all lattice vectors of equal energy.

Some caution should be taken in interpreting the relative speeds of the various algorithms based on the number of steps to decode or demodulate. No two authors seem to use the same definition for what constitutes a decoding operation or step. For example, a step in the algorithms of Conway and Sloane [14],[16] can be either an addition, a comparison or a multiplication. Forney [27],[28], on the other hand, quotes only the binary operations (addition or comparison of two numbers) needed to accumulate a series of metrics and choose the closest vector to the received signal. In doing so, he assumes that the components and their metrics are already available and no additional steps are required to obtain these values. For the demodulation algorithm discussed in this section, a step is a table look–up, add or compare. Despite these discrepancies, it was felt that some form of comparison was better than none at all.

3.3 The Probability of Occurence of an Erasure

The effectiveness of the two–stage demodulator depicted in Figure 3.5 depends on the infrequent occurence of erasures, which in turn makes their correction feasible. To determine the probability of an erasure, there are several types of observable error events which must be distinguished. We may say that a demodulation error will occur when some $x \in A_8^3$ is transmitted and the L_9 demodulator chooses some $u \neq x$ as the closest point to the received vector r. To describe the various types of error events, it is convenient to express u as the vector sum $u = x + w$ of the desired vector x and some point w in L_9. If w also happens to be a point in the Gosset lattice A_8^3, then it follows that $u = x + w$ is in A_8^3 and the error is *undetectable*. If w is not in A_8^3, the error is *detectable* and will be declared as an erasure by the L_9 demodulator. Table 3.2 lists the vectors in the first 4 shells of L_9, which include the 240 minimal vectors of the Gosset lattice A_8^3.

This list gives the most probable choices of the vector w for both detectable and undetectable errors.

Shell m	Vector Type	# of vectors	d_m^2
1	1 1 1 1 1 1 1 1 1	2	9
	3 0 0 0 0 0 0 0 0	18	9
2	2 −1 −1 −1 −1 −1 −1 −1 −1	18	12
3	2 2 −1 −1 −1 −1 −1 −1 −1	72	15
4	2 2 2 −1 −1 −1 −1 −1 −1	168 (*)	18
	3 −3 0 0 0 0 0 0 0	72 (*)	18
	3 3 0 0 0 0 0 0 0	72	18

(*) vectors in the first shell of A_8^3

Table 3.2: Vectors of the first 4 shells of L_9

The probability that the L_9 demodulator makes an error, whether it be detectable or undetectable, can be upper bounded by the probability of crossing a decision hyperplane half way between $x \in A_8^3$ and $u = x + w$, where w is any one of the vectors listed in Table 3.2,

$$P_e(L_9) \leq \sum_{m=1}^{4} N_m \cdot Q(d_m/2\sigma) \tag{3.7}$$

where N_m is the number of vectors in the m^{th} shell of L_9, d_m is the length of any vector in shell m, σ^2 is the noise variance in each dimension and $Q(\cdot)$ is the Gaussian Q-function [59]

$$Q(\alpha) = \frac{1}{\sqrt{2\pi}} \int_{\alpha}^{\infty} exp(-\lambda^2/2)\, d\lambda \,. \tag{3.8}$$

Given that the average energy or squared magnitude of the transmitted signal

vectors is E_a and that each of the 9 components is corrupted by additive white Gaussian noise of variance σ^2, the signal–to–noise ratio ρ at the input to the L_9 demodulator is $\rho = E_a/9\sigma^2$. Substituting $\sigma = \sqrt{E_a/9\rho}$ in (3.7), the probability of an error for L_9 demodulation of an ${A_8}^3$ code can be rewritten as

$$P_e(L_9) \leq \sum_{m=1}^{4} N_m \cdot Q(1.5\, d_m \sqrt{\rho/E_a}) . \tag{3.9}$$

If that portion of $P_e(L_9)$ associated with crossing a decision hyperplane to a vector of ${A_8}^3$ is removed, what remains is an upper bound on the probability of an erasure P_{eras},

$$P_{eras} \leq \sum_{m=1}^{4} N_m \cdot Q(1.5\, d_m \sqrt{\rho/E_a}) - 240 \cdot Q(6.364\sqrt{\rho/E_a}) . \tag{3.10}$$

When considering the probability of error for maximum likelihood demodulation of an ${A_8}^3$ code, it must be remembered that all the code vectors lie in an 8–dimensional subspace within $\mathbb{R}^9$. Consequently, the component of noise in the direction perpendicular to the plane containing the code vectors does not contribute to the error performance, and the signal–to–noise ratio is $\rho = E_a/8\sigma^2$. The upper bound on the probability of error for an ${A_8}^3$ Gosset lattice code is then

$$P_e({A_8}^3) \leq 240 \cdot Q(6\sqrt{\rho/E_a}) . \tag{3.11}$$

Note that the probability of error $P_e({A_8}^3)$ is greater than the probability of the L_9 demodulator making an undetectable error, $240 \cdot Q(6.364\sqrt{\rho/E_a})$. This should be expected since there will exist erasure events which lead to an uncorrectable error. The probability of erasure P_{eras} is plotted in Figure 3.6 against the probability of error for maximum likelihood demodulation, $P_e({A_8}^3)$, of a 2^{16} vector lattice code obtained using the lexicographic ordering discussed in Chapter 4. The average energy of this code is approximately $E_a = 93.7$. The probability of error for 16QAM is also included in Figure 3.6 for comparison.

If no erasure correction were provided in the second stage of this two–stage

demodulator, the probability of erasure P_{eras} would set the limit on performance. On the other hand, the upper limit set by maximum likelihood performance can be achieved only if the second stage corrects *all* erasures. Given sufficient buffer space and extensive erasure correction circuitry perfect maximum likelihood performance could be achieved with the two-stage demodulator of Figure 3.5. There are, however, practical limits to the maximum demodulation delay if the system is to be viable. In the following sections, it will be shown that it is possible to approach maximum likelihood performance asymptotically with a relatively small erasure correction effort.

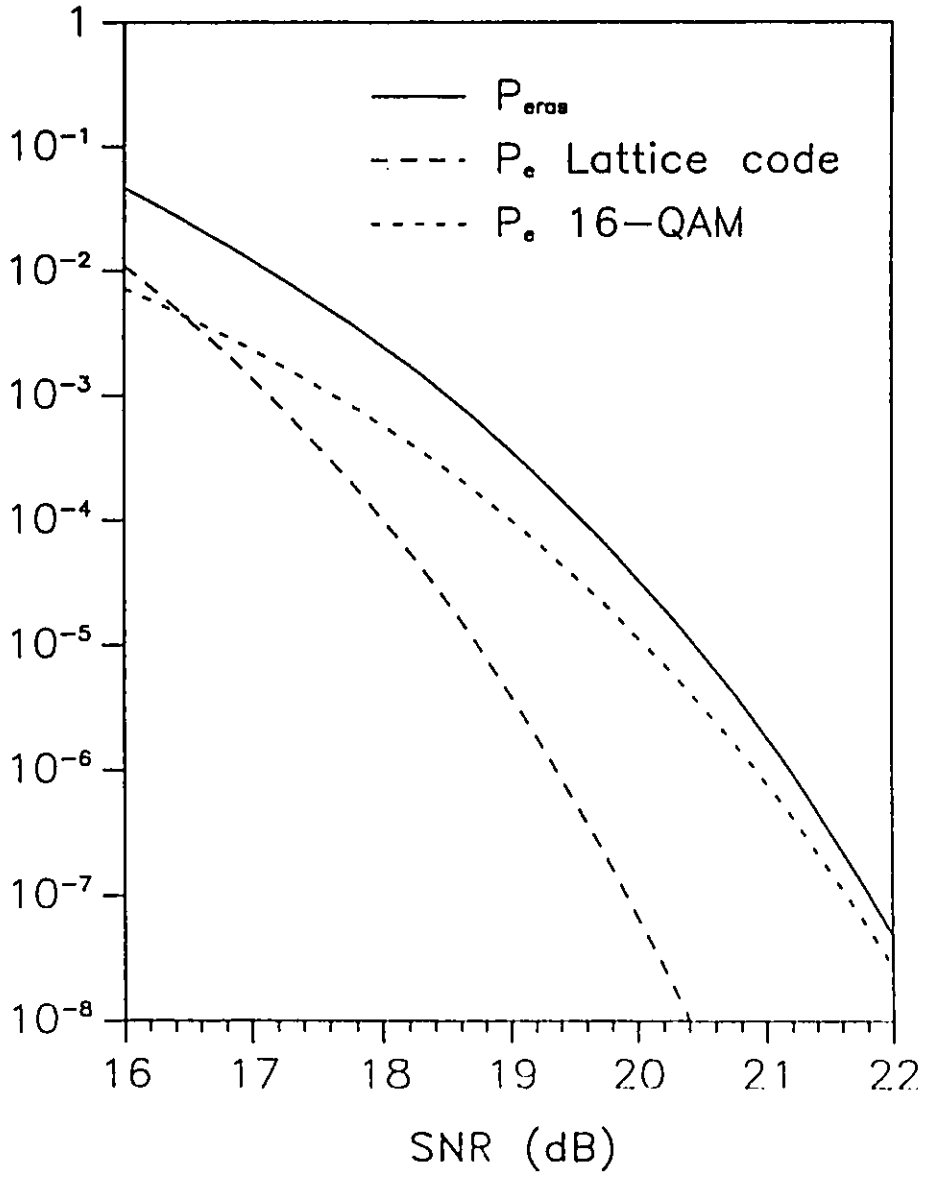

Figure 3.6: Probability of an erasure P_{eras} versus the probability of error $P_e(\Lambda_8^3)$ for a 2^{16} vector code and the probability of error for 16–QAM

3.4 Erasure Correction with a Maximum Likelihood Algorithm

One approach to the demodulation of the A_8^3 construction of the Gosset lattice is to use the Conway–Sloane algorithm for demodulating the lattice A_n [14] to find the closest point in each coset of A_8^3 and then choose the closest of the three candidates to the received vector r. The algorithm demodulates A_n by finding the closest point in $\mathbb{Z}^{n+1}$ and then adding or subtracting 1 from the components most likely to be in error until a vector with zero component sum is found. The L_9 demodulation algorithm of section 3.1 performs the first of these two steps for each coset of A_8^3. It follows, therefore, that with some modifications, the Conway–Sloane algorithm could be adapted to correct erasures. The L_9 algorithm besides providing the closest points in $3\mathbb{Z}^9$, $3\mathbb{Z}^9 + (1^9)$ and $3\mathbb{Z}^9 - (1^9)$ also has knowledge of the component sums of these three points. If these component sums are divided by 3, we have what Conway and Sloane refer to as the "deficiency" [14], or number of components in error in the initial estimates. To correct for this deficiency, we must add or subtract 3 from the deficient number of components in each coset vector until finally the components sum to zero. This may be done as follows:

1) determine the distance vectors from r to the coset candidates in $3\mathbb{Z}^9$, $3\mathbb{Z}^9 + (1^9)$ and $3\mathbb{Z}^9 - (1^9)$.

2) if the components of a coset vector already sum to zero, no correction is needed,

   if the components of a coset vector have a sum $s < 0$, find the $|s/3|$ largest components of the distance vector and add 3 to the corresponding coset vector components,

   if the components of a coset vector have a sum $s > 0$, find the $|s/3|$ smallest components of the distance vector and subtract 3 from the corresponding coset vector components.

Once these corrections have been made, we must determine which of the three resultant points in A_8^3 is closest to the received vector r. This can be done by updating the metrics 0, a and b of the L_9 algorithm for each correction made to a coset vector. Let x_k denote the k^{th} component of one of the three L_9 vectors being corrected and let $d_k = r_k - x_k$ denote the distance from the received component r_k to this component. When a correction is made, say for example by adding 3 to x_k, the squared distance between the received vector r and the corrected point increases, (over that between r and the uncorrected point), by

$$\begin{aligned} \delta &= |r_k - (x_k + 3)|^2 - |r_k - x_k|^2 \\ &= |d_k - 3|^2 - d_k^2 \\ &= -6d_k + 9 . \end{aligned} \tag{3.12}$$

It should be remembered that d_k is in the range $-1.5 \le d_k < 1.5$, so that $\delta > 0$. When the correction involves a subtraction of 3 from x_k, the increase in the squared distance is given by

$$\delta = 6d_k + 9 . \tag{3.13}$$

If the increase δ in the squared distance resulting from each correction is subtracted from the appropriate distance metric 0, a or b of the L_9 algorithm, then the largest of the adjusted metrics will determine which point is closest to the received vector r. The subtraction of δ follows from the fact that to reduce the number of metrics used in the L_9 algorithm, the squared distance from the received vector r to each of the three candidate points was subtracted from the squared distance to the candidate point in the sublattice $3\mathbb{Z}^9$, (see equations 3.2 and 3.3).

The most time consuming operation in the correcting of erasures is the finding of the $\lfloor s/3 \rfloor$ smallest or largest components of each distance vector. As indicated by the vectors in Table 3.2, the number of components which have to be corrected will very rarely exceed 3. According to Knuth [38, p. 212], the number of comparisons

needed to select the t largest or smallest of n items is at most

$$n - t + \sum_{j=n-t}^{n} \lceil \log_2 t \rceil \tag{3.14}$$

where $\lceil x \rceil$ indicates the greatest integer less than or equal to x. For $n = 9$ and $t = 3$, the maximum number of comparisons needed is 13. Each correction consists of 3 steps: 1 addition or subtraction of 3, 1 look-up to find δ and 1 subtraction of δ from the appropriate L_9 metric. Therefore, including the 9 subtractions needed to obtain the distance vector and 1 comparison to determine the sign of the component sum, the correction of an L_9 lattice vector to obtain a vector in the same coset in ${A_8}^3$ should require at most 9+1+13+3×3 = 32 operations. When only one or two components of a coset candidate need to be corrected the number of operations is of course less: 21 operations for one correction, 27 operations for two corrections. The complete erasure correction algorithm thus requires between 4 and 5 times as many operations as the L_9 demodulation algorithm. This implies that the buffer at the output of the L_9 demodulator, as shown in Figure 3.5, must be capable of holding 5 demodulated vectors to allow the erasure correction circuit to complete its work.

While an erasure correction time of 5 demodulation intervals does not create an excessive delay, it is sufficient for there to be a finite probability of additional erasures occuring during this period. These additional erasures must be considered *overflow errors* unless additional erasure correction circuits are provided in parallel. Given that the erasure correction algorithm will be unavailable for 5 demodulation intervals, the probability of 1, 2, 3 or 4 additional erasures occuring within this period is

$$\sum_{j=1}^{4} \binom{4}{j} (P_{eras})^{j+1} (1 - P_{eras})^{4-j} \tag{3.15}$$

The dominant term in the above summation is that associated with the occurence of a second erasure, namely $4P_{eras}^{2}$. Consequently, the probability of error for the

two-stage demodulator of Figure 3.5, employing the erasure correction algorithm described above, is

$$P_e' \simeq P_e(A_8^3) + 4P_{eras}^2 \tag{3.16}$$

which, as shown in Figure 3.7, asymptotically approaches the probability of error for maximum likelihood demodulation. If the additional circuitry is provided for a second erasure correction algorithm, in parallel with the first, all occurences of two erasures within 5 demodulation intervals can be corrected. Since the probability of three or more erasures occuring within 5 demodulation intervals is several orders of magnitude less than the probability of error for maximum likelihood demodulation, maximum likelihood performance can essentially be achieved with such a configuration. There is, however, the cost of the additional circuitry to be paid. Such additional costs and complexity would make the proposed scheme uneconomical. Consequently, it is best to look elsewhere for solutions to the overflow error problem.

3.5 Enumerating the Predominant Erasure Events

If the individual contributions to the probability of an erasure are listed, as in Table 3.3 for $P_{eras} = 10^{-2}$, 10^{-3} and 10^{-4}, it can be seen that the dominant term is that associated with erasure events of the form $u = x \pm (3,0,0,0,0,0,0,0,0)$. In other words, *the majority of erasure events will have been caused by the L_9 demodulator choosing the wrong point in the same coset as our desired point of A_8^3 and for which a single correction is required.* There are of course numerous other types of erasure events, however, if it is possible to recognize the most predominant of these events, a scheme may be devised to correct them as quickly as possible, thereby reducing the overall erasure correction delay. For example, erasures in which the closest point in L_9 has a component sum of ± 9 can almost certainly be attributed to the L_9

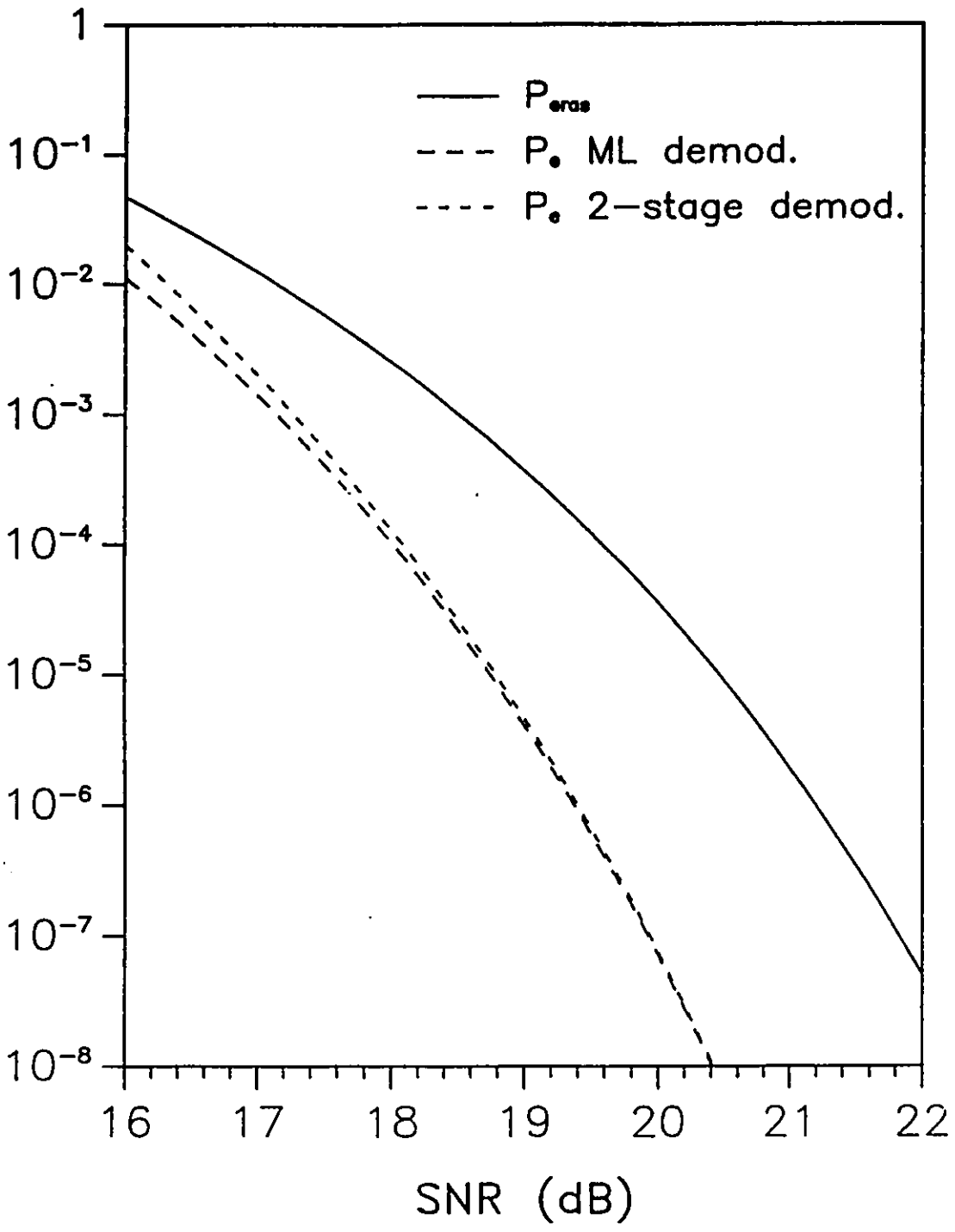

Figure 3.7: Probability of error performance of the two–stage demodulator employing a correction algorithm based on the Conway–Sloane algorithm for A_n versus maximum likelihood performance and the probability of an erasure.

algorithm choosing a vector of the form $u = x \pm (1,1,1,1,1,1,1,1)$, since the probability of crossing 3 decision boundaries to obtain a vector of the form $u = x \pm (3,3,3,0,0,0,0,0)$ is extremely small in comparison. The L_9 demodulator has in this instance chosen a point in the wrong coset and correcting three components of this vector to reduce its component sum to zero would not under any circumstances yield the correct point in A_8^3. It therefore seems logical to ignore this vector and look to the next closest point in L_9 for a possible solution to our demodulation problem. In this way, the 32 operations needed to correct a vector of component sum ±9 are avoided.

Possible choices of w when $u = x + w$	P_{eras} =	10^{-2}	10^{-3}	10^{-4}
	$P_e(A_8^3)$ =	1.0×10^{-3}	2.3×10^{-5}	4.4×10^{-7}
3 0 0 0 0 0 0 0 0		74.9×10^{-4}	83.8×10^{-5}	87.0×10^{-6}
1 1 1 1 1 1 1 1 1		8.3×10^{-4}	9.3×10^{-5}	9.7×10^{-6}
2 −1 −1 −1 −1 −1 −1 −1 −1		10.3×10^{-4}	5.8×10^{-5}	2.9×10^{-6}
2 2 −1 −1 −1 −1 −1 −1 −1		5.7×10^{-4}	1.6×10^{-5}	0.4×10^{-6}
3 3 0 0 0 0 0 0 0		0.8×10^{-4}	0.1×10^{-5}	0.01×10^{-6}

Table 3.3: Probability of crossing a decision boundary between $x \in A_8^3$ and $u = x + w \in L_9$ for various choices of w.

The events of most interest for a selective erasure correction scheme are those in which the received vector is closer to two incorrect points in L_9 than to a point in the Gosset lattice A_8^3. This situation is illustrated in Figure 3.8*a*. The point x is the closest point in A_8^3 to the received vector r, while u and v are two points in

different cosets of L_9, but not in A_8^3. In this scenario, the L_9 demodulator picks u as the closest point in L_9 and v is the next closest point; both are, however, closer to r than is x. The assumption is made that x will eventually be arrived at by making corrections to either u or v. If one or more corrections are made to u, v becomes the closest point to r. It must then be decided whether to make corrections to v also or to accept the corrected vector u'.

A decision hyperplane *H1* perpendicularly bisects the vector joining x and u delimiting the region of space where r is closer to u than to x and vice versa. Similarly, *H2* represents a decision hyperplane for distinguishing whether r is closer to v or to x. The probability that r is closer to both u and v than to x is then the probability that r lies in the region of intersection enclosed by these two hyperplanes, the shaded area in Figure 3.8*a*. There exists a point γ that is equidistant from all three vectors x, u and v. If a hyperplane is drawn at this equidistant point γ, perpendicular to the vector joining x to γ, the probability that r lies beyond this hyperplane forms an upper bound on the probability that r is closer to both u and v than to x. In pictorial form, this is the probability that r lies in the shaded region in Figure 3.8*b*.

The most probable sets of points u and v are those obtained as the vector sum of the desired point x with one of the pairs of vectors listed in Table 3.4, taken in either order. The distance d_γ is the distance from x to the equidistant point γ, and N_γ is the number of permutations of the two vectors having an equidistant point γ at distance d_γ, or equivalently, the number of permutations of γ. There is an equal number of points for the negative of each pair. The probability that the received vector r is closer to two incorrect points in L_9 having an equidistant point γ with the point $x \in A_8^3$ is then upper bounded by

$$P_\gamma \leq N_\gamma \cdot Q(d_\gamma/\sigma) = N_\gamma \cdot Q(3d_\gamma\sqrt{\rho/E_a}) . \qquad (3.17)$$

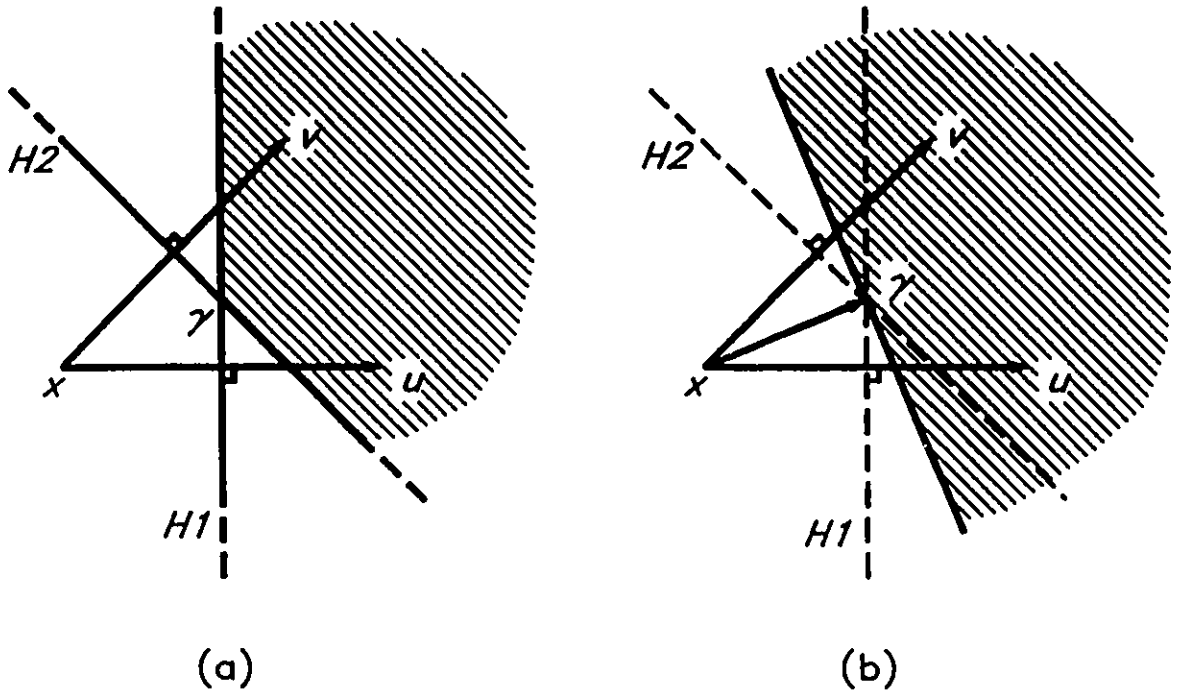

Figure 3.8: *a)* The shaded area indicates the region of space where a received vector r is closer to two incorrect points u and v of L_9 than to the desired point x in $A_8{}^3$.

b) The shaded area to the right of the hyperplane at the equidistant point γ can be used to bound the region of space where r is closer to both u and v than to x.

Pairs of L_9 vectors																		d_γ	N_γ
3	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1	1	1.837	9
3	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1	−2	2.023	72
3	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	−2	−2	2.196	252
3	0	0	0	0	0	0	0	0	2	−1	−1	−1	−1	−1	−1	−1	−1	1.837	9
3	0	0	0	0	0	0	0	0	2	2	−1	−1	−1	−1	−1	−1	−1	2.023	72
3	3	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1	1	2.196	36
3	3	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1	−2	2.324	252
3	3	0	0	0	0	0	0	0	1	1	1	1	1	1	1	−2	−2	2.461	756
3	3	0	0	0	0	0	0	0	2	2	−1	−1	−1	−1	−1	−1	−1	2.196	36

Table 3.4: Pairs of L_9 vectors and the distance to their equidistant point γ with the origin 0.

The values of d_γ and N_γ given in Table 3.4 make it possible to predict both how often certain erasure events are likely to occur and how often errors will be made by correcting only certain erasure patterns. For example, if u has a component sum of –3 and v has a component sum of +6 and the choice is made to correct one component of u and ignore v, then when $u = x + (2,2,-1,-1,-1,-1,-1,-1,-1)$ and $v = x + (3,3,0,0,0,0,0,0,0)$ an error will be made. If the shaded region in Figure 3.8*b* can be split in half to indicate that in one half r is closest to u and in the other half r is closest to v, then such error events will occur with roughly a probability of $\frac{1}{2}P_\gamma$.

3.6 A Near Maximum Likelihood Correction Algorithm

With the ability to predict the influence on error performance of selectively correcting erasure events according to the sum of the components of the two closest points in L_9, u and v, a selective correction algorithm can now be given. A flow chart for this algorithm is given in Figure 3.9. In this flow chart, $s(u)$ and $s(v)$ have been used to denote the sums of the components of u and v, respectively, i.e. $s(u) = \sum_{i=1}^{9} u_i$ and $s(v) = \sum_{i=1}^{9} v_i$. The various paths through the algorithm are labeled (a) through (i).

Given that the dominant erasure event is one in which the demodulated vector is of the form $u = x \pm (3,0,0,0,0,0,0,0,0)$, the first step in correcting erasures is obviously to determine if the components of the demodulated vector sum to ±3. If they do, a correction is instantly made to one component of u to reduce its component sum to zero. If u', the corrected version of u, is still the closest point in L_9 to the received vector, we have our desired point in A_8^3 and are finished. This is path a of the algorithm. If v is now closer to r than the corrected vector u', then we

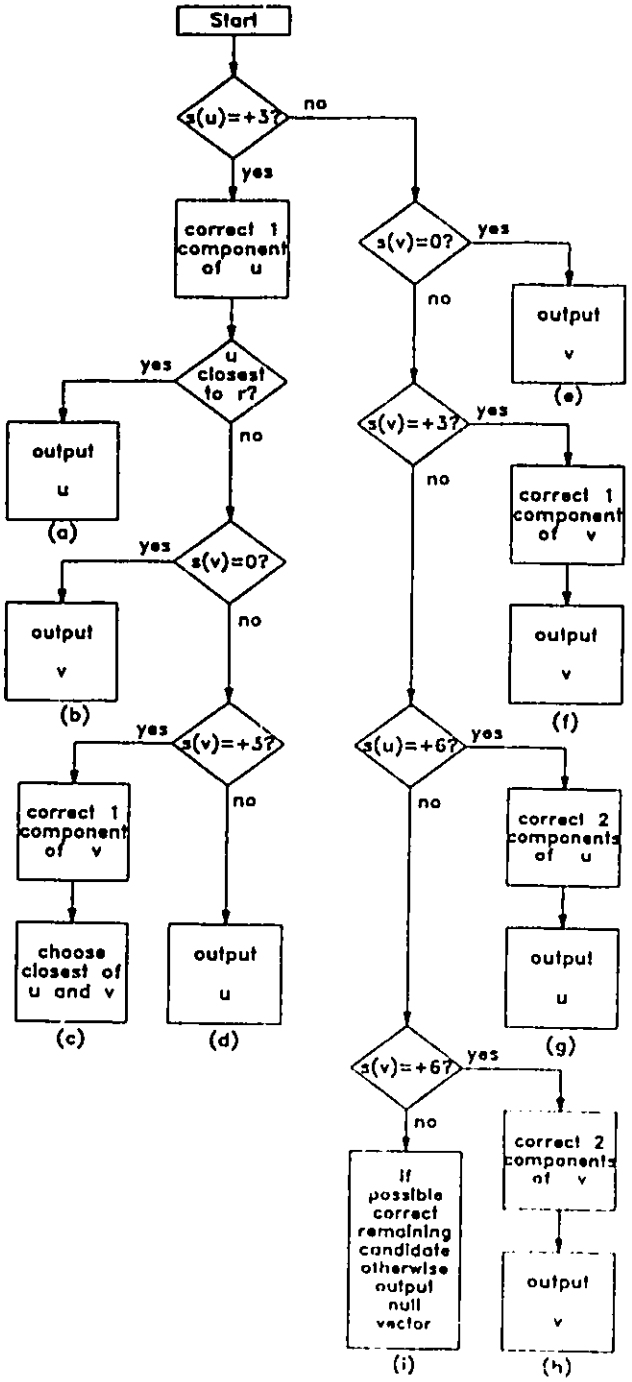

Figure 3.9: Flow chart for an algorithm which selectively corrects erasures according to the component sums of the closest and next closest points in L_9, u and v respectively.

must decide whether to: i) accept v if its components sum to zero (path b), or ii) correct one component of v and choose the closest of u' and v' to r (path c), or iii) if the components of v sum to ±6 or more, to ignore v and accept u' (path d).

When the components of the demodulated vector do not sum to ±3, then it is most likely to have a component sum of ±6 or ±9 and be of the form $u = x \pm (-2,1,1, 1,1,1,1,1,1)$ or $u = x \pm (1,1,1,1,1,1,1,1,1)$, respectively. The next course of action is then to determine if v, the second closest point in L_9, has a component sum of zero or ±3. If it has a component sum of zero, it is immediately accepted; if it has a component sum of ±3, one component is corrected and the result accepted. These are paths e and f, respectively, in the flow chart of Figure 3.9. Beyond this, it is a matter of determining whether to correct two components of u along path g, or two components of v along path h. The final resort is path i where if both u and v have components sums of ±9 or greater, we look to the third candidate point in L_9 and if it has a component sum of 0, ±3 or ±6, correct the appropriate number of components and output the result. If this third candidate vector in L_9 also has a component sum of ±9 or greater, it is unlikely that the erasure can be corrected successfully so we simply output the null vector 0 and quit.

If the algorithm terminates in one of the first three paths a, b or c, then this is the maximum likelihood solution for the demodulation of A_8^3. On the remaining paths, d through i, assumptions are made about u and v based on their component sums which means that certain erasure events will not be corrected properly. These errors are listed in Table 3.5 for paths (d) through (i). Although this list is incomplete, it covers the most probable events and will allow us to predict the error performance of the algorithm.

The process of making a correction is the same as that outlined in section 3.4, except that the number of components corrected is limited to one or two. As stated

Path	Erasure events not corrected on indicated path	Probability of occurence
d	$u = x \pm (2, 2, -1, -1, -1, -1, -1, -1, -1)$ $v = x \pm (3, 3, 0, 0, 0, 0, 0, 0, 0)$	$36 \cdot Q(6.588\sqrt{\rho/E_a})$
	$u = x \pm (1, 1, 1, 1, 1, 1, 1, -2, -2)$ $v = x \pm (3, 3, 0, 0, 0, 0, 0, 0, 0)$	$756 \cdot Q(7.383\sqrt{\rho/E_a})$
e	$u = x \pm (3, 3, 0, 0, 0, 0, 0, 0, 0)$ $v = x \pm (2, 2, 2, -1, -1, -1, -1, -1, -1)$	$504 \cdot Q(6.972\sqrt{\rho/E_a})$
	$u = x \pm (3, 3, 0, 0, 0, 0, 0, 0, 0)$ $v = x \pm (1, 1, 1, 1, 1, 1, -2, -2, -2)$	$2520 \cdot Q(7.794\sqrt{\rho/E_a})$
f	$u = x \pm (3, 3, 0, 0, 0, 0, 0, 0, 0)$ $v = x \pm (2, 2, -1, -1, -1, -1, -1, -1, -1)$	$36 \cdot Q(6.588\sqrt{\rho/E_a})$
	$u = x \pm (3, 3, 0, 0, 0, 0, 0, 0, 0)$ $v = x \pm (1, 1, 1, 1, 1, 1, 1, -2, -2)$	$756 \cdot Q(7.383\sqrt{\rho/E_a})$
g	$u = x \pm (1, 1, 1, 1, 1, 1, 1, 1, -2)$ $v = x \pm (3, 3, 0, 0, 0, 0, 0, 0, 0)$	$252 \cdot Q(6.972\sqrt{\rho/E_a})$
h	$u = x \pm (3, 3, 3, 0, 0, 0, 0, 0, 0)$ $v = x \pm (1, 1, 1, 1, 1, 1, 1, 1, -2)$	$504 \cdot Q(7.937\sqrt{\rho/E_a})$
i	$u = x \pm (3, 3, 3, 0, 0, 0, 0, 0, 0)$ $v = x \pm (1, 1, 1, 1, 1, 1, 1, 1, 1)$	$84 \cdot Q(7.974\sqrt{\rho/E_a})$
	$u = x \pm (1, 1, 1, 1, 1, 1, 1, 1, 1)$ $v = x \pm (3, 3, 0, 0, 0, 0, 0, 0, 0)$	$84 \cdot Q(7.974\sqrt{\rho/E_a})$

Table 3.5: Erasure events not corrected along paths (d) through (i).

Path	a	b	c	d	e	f	g	h	i
Number of Steps	23	24	46	25	2	24	31	32	2,26,32

Table 3.6: Number of steps along each path in the correction algorithm of Figure 3.9

there, the number of operations required for one correction is 21, and for two corrections it is 27. These figures were used to arrive at the number of operations along each path summarized in Table 3.6. The longest path through the algorithm is path c which requires 46 operations to correct one component in each of u and v, slightly greater than twice the length of the L_9 demodulation algorithm. Although this path is followed less than 0.5% of the time, it is the path which dictates that the buffer at the output of the L_9 algorithm be of length 3. For path i, three values have been given for the number of steps. These indicate the number of steps if 0, 1 or 2 corrections are made to the third candidate point in L_9, respectively.

The paths which cover the bulk of erasure events, paths a, b, d and f, require between 23 and 25 operations. With erasure events being corrected just slightly slower than the L_9 algorithm demodulates, the problem of erasures overflowing the correction circuit is essentially limited to the occurrence of back–to–back erasures, which occur with probability $P_{eras}^{\ 2}$. If in the case of back–to–back erasures, the information released by the L_9 demodulator is stored for a short period of time rather than simply rejecting it, the problem of overflow errors is eliminated. This involves placing a single element queue in front of the correction circuit which either sends information straight in if the circuit is free or holds the information until the last few steps of the correction algorithm are performed and the circuit becomes available.

With the problem of overflow errors eliminated, it is only the selective erasure coverage which prevents this two–stage demodulator from achieving maximum likelihood performance. By accumulating the probabilities listed in Table 3.5 and adding these values to the probability of error $P_e(A_8^{\ 3})$, the estimate of error performance shown in Figure 3.10 is obtained. Comparing Figure 3.7 and 3.10, the error performance of the above erasure correction algorithm can be seen to approach

the performance of maximum likelihood demodulation much sooner than the correction algorithm of section 3.4. Another important difference is that the buffer length has been reduced from 5 to 3 with the present selective erasure correction scheme, thereby reducing the overall delay.

The selective erasure correction algorithm given in Figure 3.9 will be the algorithm employed with the L_9 demodulator in a simulation study of the performance of the two-stage demodulator. Details of the structure of this simulation, and of course the results, will be discussed in Chapter 5.

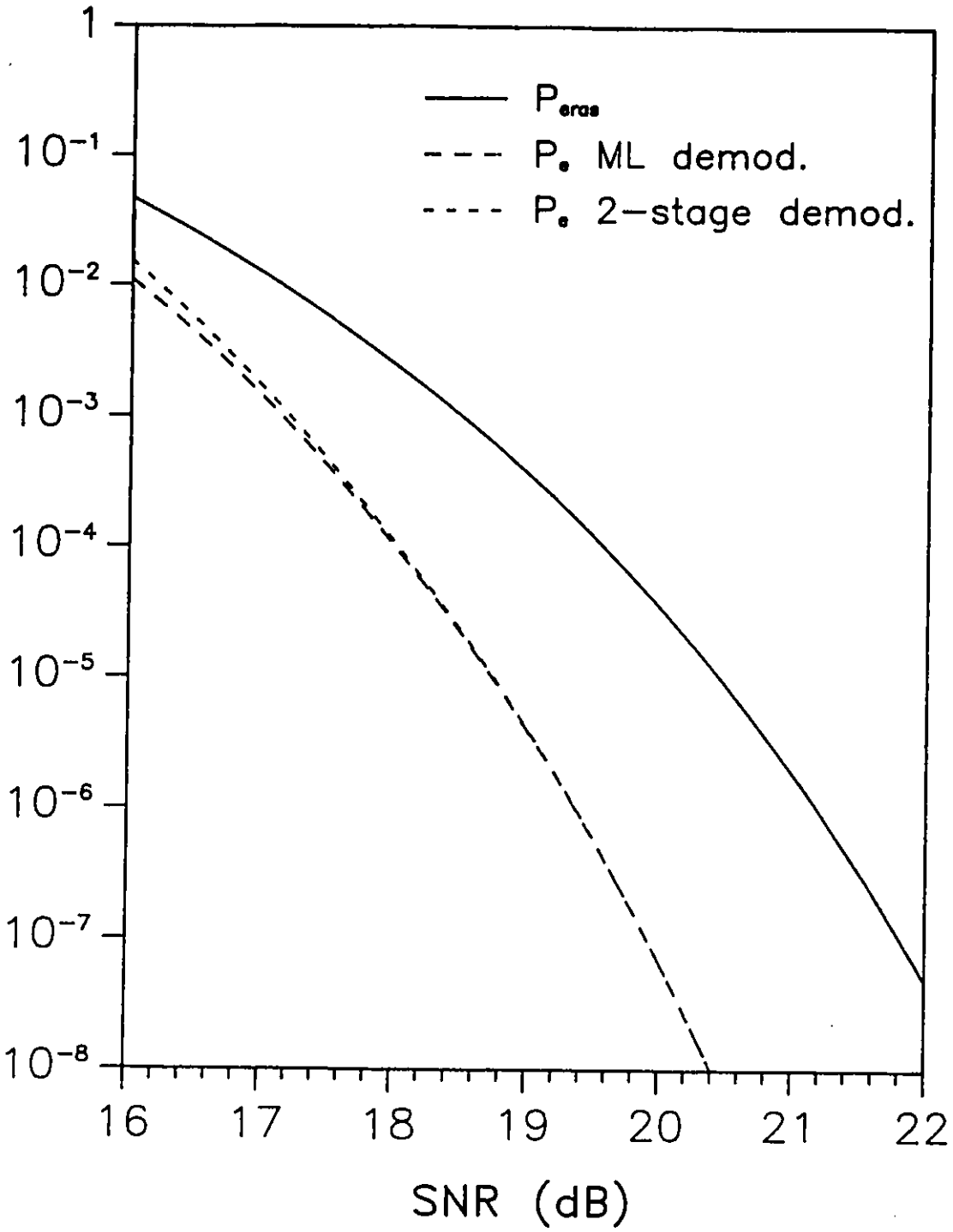

Figure 3.10: Plot of the error performance of the two–stage demodulator, employing the selective erasure correction algorithm of Figure 3.9, versus the probability of error for maximum likelihood demodulation and the probability of erasure.

CHAPTER 4

A LEXICOGRAPHIC ORDERING OF THE SIGNAL SET

For a lattice code, encoding is defined as the process of assigning one of M possible messages $m = 0, 1,...,M-1$ to a vector of the code. In the receiver, the inverse function is performed by mapping each demodulated vector $\boldsymbol{x}$ to a message $\hat{m}$. This has also been referred to as finding the index $\hat{m}$ of $\boldsymbol{x}$ within the code [15].

A number of methods have been proposed for performing this mapping in the encoder and decoder. In [15], Conway and Sloane gave a simple algorithm to find the index of a vector using the dual basis of the lattice. The decoders of P. de Buda [4], Gersho and Lawrence [32], and Secord and R. de Buda [49] all used a variant of permutation modulation [53] to map information into the sign changes and permutations that generate all the vectors within a shell from a set of prototype vectors. Adoul et al. [2] generated codebooks for vector quantizers by using an equivalence relation to lexicographically order the vectors of successive shells of the Gosset lattice. While each of these methods has its merits, they either cannot be easily adapted to the present scheme or are too slow to be of use.

The simplest and fastest solution to the mapping problem would be to employ a look–up table which when presented with a demodulated vector would output its index within the code. Such look–up tables, if not properly designed, can consume large amounts of memory. To reduce the memory consumption without drastically affecting speed, a variant on lexicographic orderings is proposed. The ordering is described in terms of the lattice A_8 consisting of all 9–tuples of integers with a sum of zero. The extension to the Gosset lattice A_8^3 is straightforward since it is the

union of three scaled copies of A_8 (see equation 2.1). This implies that the procedure outlined below can be used to develop decoder look–up tables for each coset of the lattice code. Since an integral part of the demodulation process is the identification of the coset from which the demodulated vector came, it is a simple matter to determine which set of decoder tables is to be used. More will be said on these issues later in the chapter.

4.1 Determining the Index of a Vector in a Lexicographic Ordering

In a *lexicographic ordering*, a vector y is said to be *lexicographically smaller* than a vector x if the leftmost non–zero component of the difference vector $x-y$ is positive [35, p.193]. The index $\hat{m}$ of the vector x is then equal to the order $o(S)$ or size of the set $S=\{y\}$ of all code vectors y lexicographically smaller than x. The set S can be subdivided into a series of smaller sets S_h consisting of all vectors y lexicographically smaller than x at a *depth* h. That is to say, S_h contains all vectors y in which the first non–zero component of the difference vector $x-y$ is the h^{th} component,

$$\begin{aligned} y_i &= x_i \quad i = 1,2,\ldots,h-1 \\ y_h &< x_h \,. \end{aligned} \tag{4.1}$$

Note that in a finite code, each component x_h will only be able to take on a finite number of values. Consequently, S_h will be the null or empty set ϕ when x_h is at the minimum of its range. Also, for $x,y \in A_8$, if x and y have the same first 8 components, they must also have the same ninth component to satisfy the zero sum restriction of the lattice, thus $h \leq 8$.

By breaking up the set S into a series of smaller sets S_h, the index $\hat{m}$ of the vector x can be expressed as the sum of the indices $m_h = o(S_h)$,

$$\hat{m} = m_1 + m_2 + \ldots + m_8 \,. \tag{4.2}$$

In this way, rather than use one large look–up table to store the index $\hat{m}$ associated with each code vector $\boldsymbol{x}$, a succession of much smaller tables can be used to store the indices m_h which will greatly reduce the required memory. (The savings will become apparent later when the memory requirements of the tables are computed.) Equation (4.2) indicates that the cost paid for reducing the amount of memory needed is that now 8 table look–ups and 7 additions are required to find the index $\hat{m}$ instead of a single table look–up. However, this is still a viable approach to finding the index of the demodulated vector since it requires fewer operations than the L_9 algorithm.

To determine the indices m_h it is not necessary to know the first h–1 components of $\boldsymbol{x}$ as (4.1) would imply. Rather, the zero sum condition of the lattice A_8 makes it possible to determine m_h using only the sum of the first h–1 components of the vector $\boldsymbol{x}$ and the h^{th} component x_h. Let s_i denote the sum of the first i components of $\boldsymbol{x}$ and let s_{9-i} denote the sum of the remaining 9–i components;

$$s_i = x_1 + x_2 + \ldots + x_i , \tag{4.3a}$$

$$s_{9-i} = x_{i+1} + x_{i+2} + \ldots + x_9 . \tag{4.3b}$$

Since by definition $s_i + s_{9-i} = 0$, $s_i = -s_{9-i}$ for $i = 1, 2,\ldots,8$. Using this fact, the number of vectors $\boldsymbol{y}$ in the set S_h may be found by letting y_h range over all possible values less than x_h and determining the number of possible combinations of the remaining 9–h components that sum to $-s_{h-1}-y_h$. If the components of each vector are equally likely to take on one of L integer values in the range $-a \le x_i \le +b$, $L = a + b + 1$, then it can be shown [44, pp. 69–70] that the number of possible combinations of 9–h integers having a sum of s, $-(9-h)a \le s \le +(9-h)b$, is equal to

$$N(9-h,s) = \sum_{k=0}^{q} (-1)^k \binom{9-h}{k} \frac{\lfloor s+(9-h)a-kL-h+8 \rfloor !}{\lfloor s+(9-h)a-kL \rfloor ! \ \lfloor 8-h \rfloor !} \tag{4.4}$$

where q is the greatest integer less than or equal to $[s+(9-h)a]/L$. By setting $h = 0$,

this formula can be used to enumerate the number of vectors in an A_8 signal set, which for the component limits of $-a \leq x_i \leq +b$ is

$$N(9,0) = \sum_{k=0}^{q} (-1)^k \binom{9}{k} \frac{[9a-kL+8]!}{[9a-kL]!\ 8!}, \tag{4.5}$$

where q is the greatest integer less than or equal to $9a/L$.

There are some restrictions on the use of equation (4.4) in calculating m_h which must be explained. Given $s_{h-1} = x_1 + \ldots + x_{h-1}$ and $x_h > -a$, we would like to use $N(9-h,s)$ to calculate the number of combinations of $9-h$ components having a sum of $s = -s_{h-1}-y_h$ for each value of y_h in the range $-a \leq y_h < x_h$. This would lead to an expression for m_h of the form

$$m_h = \sum_{y_h=-a}^{x_h-1} N(9-h,-s_{h-1}-y_h) \tag{4.6}$$

However, if we are not careful the sum $-s_{h-1}-y_h$ can exceed the limits of $-(9-h)a \leq s \leq +(9-h)b$ for which $N(9-h,s)$ is defined. For example, we may want to find the number of vectors y lexicographically smaller than x at a depth 5 when the first 4 components of x are all equal to the lower limit $-a$ and the fifth component x_5 is some value greater than $-a$. For this case, the first term in the summation of equation (4.6) would be $N(4,-5a)$ which indicates the number of vectors y with their first 5 components equal to $-a$. The remaining four components in each vector y must have a positive sum of $5a$ to satisfy the zero sum restriction of the lattice. However, no such vectors will exist if the maximum sum possible for the remaining four components, namely $4b$, is less than $5a$. In such instances, attempting to use $N(9-h,s)$ would lead to erroneous results. To overcome this difficulty, define $N'(9-h,-s_{h-1}-y_h)$ according to

$$N'(9-h,-s_{h-1}-y_h) = \begin{cases} N(9-h,-s_{h-1}-y_h) & -(9-h)a \leq -s_{h-1}-y_h \leq +(9-h)b \\ 0 & \text{otherwise} \end{cases} \tag{4.7}$$

When $x_h > -a$, the index m_h can now be expressed as the summation

$$m_h = \sum_{y_h=-a}^{x_h-1} N'(9-h,-s_{h-1}-y_h) \tag{4.8}$$

and for $x_h = -a$, $m_h = 0$.

When each component x_i is permitted to take on any integer value in the range $-a \leq x_i \leq +b$ with equal probability, the resultant signal set is the intersection of the interior of a 9–cube with the 8–dimensional hyperplane $\sum_{i=1}^{9} x_i = 0$. While such a choice of signals is less efficient, in terms of average energy E_a, than choosing a signal set enclosed by a hypersphere, the indices m_h could not be easily computed with the latter choice of signal set. A signal set enclosed within a hypersphere could not use equations (4.4), (4.7) and (4.8), but would in fact require an extremely laborous computer search first to enumerate the vectors in the signal set and then to arrange them in lexicographic order. Thus while the lexicographically ordered signal set is less efficient in terms of its average energy, it has definite practical advantages due to the ease with which it can be implemented and used.

4.2 Implementing the Lexicographic Ordering Through Look–up Tables

Using equation (4.8), it is possible to generate the indices m_h and store them in look–up tables K_h for each component. Note that to find m_h we require both the component x_h and the sum of the first h–1 components s_{h-1}. Rather than compute the sums $s_h = s_{h-1} + x_h$ with each new component x_h, it will require little extra memory to store s_h so that it can be passed forward for use with the next table. Thus the tables K_h receive the pair (s_{h-1},x_h) as inputs and output the pair (m_h,s_h); that is, $K_h(s_{h-1},x_h) = (m_h,s_h)$. Then for h+1, all we require is x_{h+1} since s_h is precomputed and available.

For $h = 1$, the sum sent into table K_1 is s_0, which is of course zero, and K_1 outputs for the sum s_1 the component x_1 that was sent in. Since table K_1 has really only one input and one output, it is best to combine this table with K_2 to produce a table K_2' that receives the input pair (x_1,x_2) and has as outputs the index $m_2' = m_1 + m_2$ and the sum $s_2 = x_1 + x_2$. Not only does this save memory but it also reduces the number of table look–ups and additions each by one. The tables K_2', K_3,...,K_8 are thus organized as in Figure 4.1. An extra adder is shown to sum s_8 and x_9 simply to verify that the sum of the components is in fact zero. This extra sum could be used as a way of detecting violations in the table, however, if the tables are properly designed it is not needed.

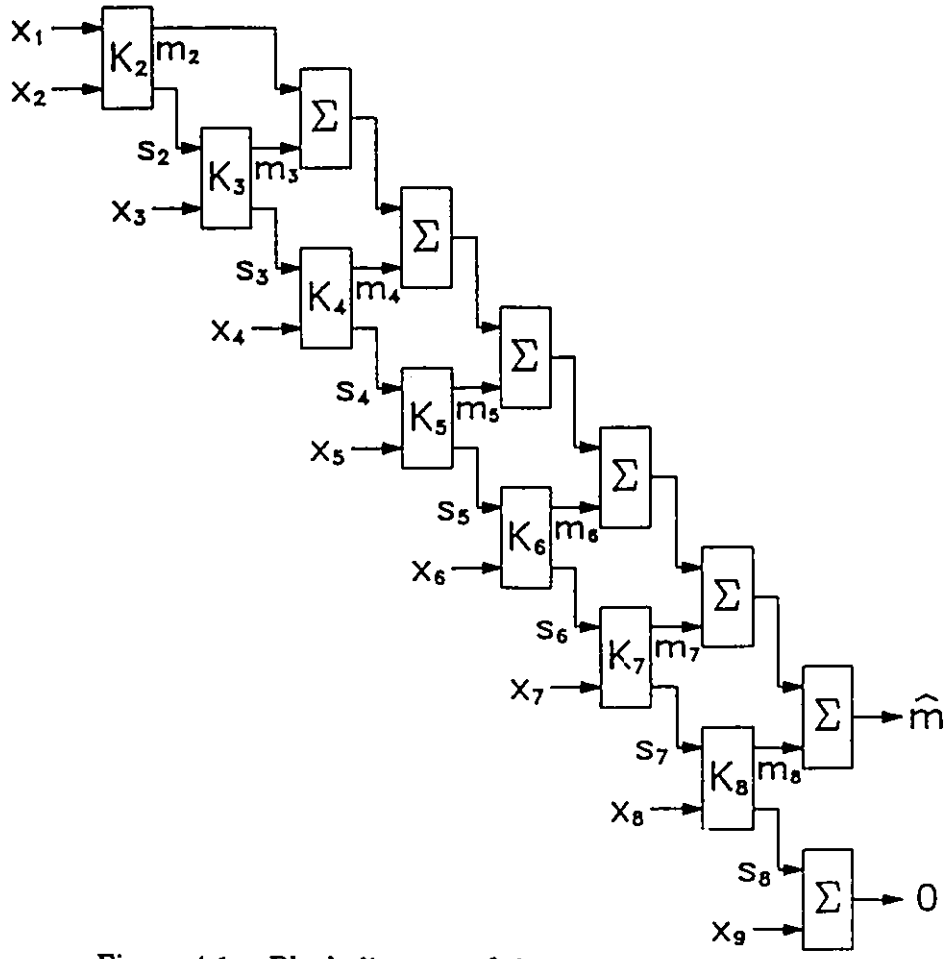

Figure 4.1: Block diagram of the decoder look–up tables for one coset

4.3 Adaptation Of The Tables For The Gosset Lattice A_8^3

As was stated at the outset, the decoder look–up tables generated using equation (4.8) and the structure depicted in Figure 4.1 are for the lattice A_8. The extension of this method to the generation of decoder look–up tables for the Gosset lattice A_8^3 is fairly easily explained. For convenience, we repeat the defining equation of A_8^3 found in Chapter 2, namely

$$A_8^3 = 3A_8 \cup 3A_8 + (-2^3,1^6) \cup 3A_8 + (2^3,-1^6) . \qquad (4.9)$$

It is easy to see that the generation of a set of decoder look–up tables for the sublattice $3A_8$ is simply a matter of scaling. The scaling can either be applied to the decoder look–up tables so that $K_2',K_3,...,K_8$ accept integer multiples of three as inputs or we can scale down the outputs of the L_9 algorithm to produce a vector of A_8 for the sublattice instead of a vector of $3A_8$. The latter is perhaps more preferable since not only will it lead to a savings in memory for the decoder look–up tables but may, depending on the number of modulation levels, also mean a savings of one bit per component in the width of the buffer required at the output of the L_9 algorithm.

To generate the decoder look–up tables for the cosets $3A_8+(-2^3,1^6)$ and $3A_8+(2^3,-1^6)$ is slightly more compiicated since they require both a linear shift and a scaling in the range of component values. The components of a vector x in the coset $3A_8+(-2^3,1^6)$ are all congruent modulo 3 to +1. According then to the definition of a congruence, $(x_i-1)/3$ must be an integer. The set of shifted and scaled components $x_i' = (x_i-1)/3$ can be used to generate decoder tables for coset $3A_8+(-2^3,1^6)$ if an adjustment is made in the component sum restriction. With the augmented components x_i', we are now looking for 9–tuples of integers that sum to –3 rather than to zero. This means that in equation (4.8) we would use $N'(9-h,-s_{h-1}-y_h-3)$ to calculate the indices m_h for the decoder tables of the coset

$3A_8+(-2^3,1^6)$. The number of vectors in the coset for the range of augmented components $-a \le x_i' \le +b$ can be computed using equation (4.4) as

$$N(9,-3) = \sum_{k=0}^{q} (-1)^k \binom{9}{k} \frac{[9a-kL+5]!}{[9a-kL-3]!\ 8!} \tag{4.10}$$

where again $L = a + b + 1$ and q is the greatest integer less than or equal to $(9a-3)/L$. Once the indices m_h have been computed, the inverse transformation can be applied to the augmented components x_i' to obtain a proper range of values $-9a+1 \le x_i \le 9b+1$ for the components x_i of a coset vector $x \in 3A_8+(-2^3,1^6)$. This will also require the recomputing of the sums s_h stored in the tables. Alternatively, we may choose to use the augmented components x_i' within the demodulator and the decoder tables to save memory and narrow the width of the buffer used at the output of the L_9 algorithm.

In a similar manner, a set of decoder tables for the coset $3A_8+(2^3,-1^6)$ can be generated using the normalization $x_i'' = (x_i+1)/3$ and by searching for the number of 9–tuples of integers with a sum of +3. The number of vectors in this coset, when its normalized components x_i'' take on integer values in the range $-a \le x_i'' \le +b$, is given by

$$N(9,+3) = \sum_{k=0}^{q} (-1)^k \binom{9}{k} \frac{[9a-kL+11]!}{[9a-kL+3]!\ 8!} \tag{4.11}$$

where q is the greatest integer less that or equal to $(9a+3)/L$.

The final point to be discussed concerns the integration of these three sets of decoder tables into one complete set for an $A_8{}^3$ Gosset lattice code. It must first be decided how many vectors are to be taken from each coset of the lattice. Once this is decided, one coset is designated as covering the range of indices $\hat{m}$ from zero to its maximum, say M_1, less one. For the next coset, the number M_1 of vectors in the first coset is added as a bias term into the first decoder table K_2' so that the decoder tables for the second coset covers the range of indices from M_1 to say

$M_1 + M_2 - 1$, where M_2 is the number of vectors in the second coset. For the final coset, the total number of vectors in the first two cosets, $M_1 + M_2$, is added as a bias into its first table K_2' so that the range of indices covered by these tables is from $M_1 + M_2$ to say $M_1 + M_2 + M_3 - 1$, where M_3 is the number of vectors in the third coset. The total number of vectors in the signal set is of course $M = M_1 + M_2 + M_3$.

Generally, the selection of a particular range of modulation levels for each component of a code vector will result in a much larger set of vectors than we require. The number of entries in the decoder look–up tables must then be trimmed to arrive at the desired range of indices. This can be accomplished through again using a bias term.

When trimming the number of vectors in a coset, we will in general want to choose those vectors in the middle of the lexicographic ordering and eliminate the vectors at the extremes of the range since this will lead to a lower average energy for the signal set. Choosing the vectors in the middle of the lexicographic ordering is done by using as a bias half the difference between the desired number of vectors and the total number of vectors generated by a particular range of component values. This number indicates how far into the lexicographic ordering our desired ordering is to begin. The bias term can either be worked into the tables or it can be subtracted at the end when all the m_h values have been accumulated.

There is some danger in this trimming of the cosets in that the demodulator may choose a vector which the decoder tables were originally designed to accommodate but that now falls outside the set of vectors we wish to use as a code. Protection against such an event may be achieved by placing flag values into the tables for the indices m_h that have been eliminated through trimming. These flag values would then cause one or more adders in Figure 4.1 to either underflow or

overflow depending on whether the vector was below or above the desired range in the ordering. The underflow or overflow then signals that either the minimum or maximum index of the coset should be output when the violation occurs. There are various methods of detecting violations in the tables; this is but one simple technique.

Problems such as the one just described are a feature common to decoding strategies in which an algorithm designed to demodulate on the infinite lattice is used to demodulate a finite code. The question always arises as to what should be done when the demodulator picks a vector which lies just beyond the boundaries of the code, or how to prevent it from doing so. There exists no fast or simple solution to this *edge effect* problem, only adhoc ones such as that given above. The only real way of forcing the demodulator to pick a vector within the boundaries of the code is to more closely integrate the detection scheme and the mapping process, essentially developing a code specific decoding algorithm like those of [4], [32] and [49]. The obvious disadvantage to this is that the portability of the scheme is lost and much more development is now needed in adapting the decoder structure to fit a variety of codes.

4.4 The Memory Required for the Decoder Tables of a 16 Bit Per Vector Code

A sufficient number of modulation levels for a 2^{16} vector code is to allow the components of the A_8^3 lattice vectors to take on integer values in the range –6 to +5. The components of a vector of the sublattice $3A_8$ will thus be in the set $\{-6,-3,0,+3\}$, while the components of the vectors in cosets $3A_8+(-2^3,1^6)$ and $3A_8+(2^3,-1^6)$ are in the sets $\{-5,-2,+1,+4\}$ and $\{-4,-1,+2,+5\}$ respectively. With four levels for the component values in each coset, we can use a set of normalized components as discussed in the previous section that require only 2 bits to represent

a component. Note that we also require an additional 2 bits to identify the coset from which the demodulated vector came so that we can choose the correct set of decoder tables to use. The buffer at the output of the L_9 demodulator in Figure 3.5 must then be 20 bits wide. The number of vectors in the sublattice $3A_8$, coset $3A_8+(-2^3,1^6)$ and coset $3A_8+(2^3,-1^6)$ for the above choice of levels are, respectively,

$$N(9,0) = \sum_{k=0}^{4} (-1)^k \binom{9}{k} \frac{[26-4k]!}{[18-4k]!\ 8!} = 13051,$$

$$N(9,-3) = \sum_{k=0}^{3} (-1)^k \binom{9}{k} \frac{[23-4k]!}{[15-4k]!\ 8!} = 27876,$$

$$N(9,+3) = \sum_{k=0}^{3} (-1)^k \binom{9}{k} \frac{[20-4k]!}{[12-4k]!\ 8!} = 27876,$$

for a total of 68803 vectors in the entire $A_8{}^3$ signal set. To obtain the 65536 vectors needed for the code, we can take 12430 vectors from the sublattice and 26553 vectors from each of the two cosets. The average energy of this set of vectors, using the original set of modulation levels from –6 to +5, is $E_a = 93.7$.

Table 4.1 lists the memory requirements for decoder look–up tables of one coset. Since the number of modulation levels used by the vectors in a coset is the same for all three cosets, the memory requirements are also the same. The maximum number of bits of input to any table is 6, however, the smallest denomination of erasable programmable read–only memory (EPROM) available is 1K×8 which accepts 10 bits of input and provides 8 bits of output. The memory used by the tables is thus quoted in terms of the number of 1K×8 EPROM's required and then the total is expressed in kilobytes. As the bottom line indicates, each coset requires 12 kilobytes of memory for its tables, giving a grand total of 36 kilobytes of memory for the entire code.

If we had chosen to generate a single look–up table that accepted the nine

components of a demodulated vector as input and produced 16 bits of output, the memory required would be 2^{28} bytes (27 bits of input, 3 bits/component, and 2 bytes of output) or 256 megabytes. The lexicographic ordering described in this chapter thus reduces our memory consumption by a factor of over 7000, quite a dramatic decrease. Of course, this does not come without some price. The lexicographic ordering requires 7 table look–ups and 6 additions to find the index of a vector, an additional 13 operations which must be counted when determining the decoders overall speed. This is, however, a small price to pay for the savings in memory.

Table K_h	Input Bits s_{h-1}	x_h	Output Bits m_h	s_h	No. of 1K×8 EPROMS	Total Memory
2	2	2	16	3	3	3 K
3	3	2	12	4	2	2 K
4	4	2	10	4	2	2 K
5	4	2	8	4	2	2 K
6	4	2	5	3	1	1 K
7	3	2	4	2	1	1 K
8	2	2	2	2	1	1 K
					Total	12 K

Table 4.1: Memory requirements of the decoder look–up tables for one coset of the lattice code

CHAPTER 5

QUANTIZATION EFFECTS AND SIMULATION RESULTS

In Chapter 3, the probabilities of erasure and error were estimated using a union bound approach. In the present chapter, we will outline a simulation study that was undertaken to back up those theoretical calculations and present results from that simulation. The assumptions made in the theoretical calculations, (that the channel noise was additive white Gaussian noise and that perfect synchronization was attained), will be carried over to the simulation. Also, since our primary concerns are to verify the probabilities of erasure and error for the demodulator, the decoder look–up tables discussed in Chapter 4 will not be implemented. The underlying structure of these tables will, however, be followed when the mapping process is inverted to obtain an encoding procedure.

One assumption that was not made in calculating the probabilities of erasure and error is that finite quantization would be employed in the demodulator. In any practical implementation, the received signal will be quantized to a finite number of levels to allow for the use of digital circuitry in performing the demodulation. Therefore, to make the simulation realistic, we followed suit and performed two sets of tests; one under the assumption that 6 bits of quantization were used and the other assuming 8 bits of quantization. It is instructive at this point to predict what effect quantization will have on the erasure and error performance of the demodulator before moving on to describe the simulation.

5.1 The Effects of Quantization on the Probabilities of Erasure and Error

In the description of the L_9 algorithm, it was stated that look–up tables could be used to determine the coset vector components and the values a_i and b_i of the discriminator functions, which implies the use of quantized received data. Quantization introduces distortion by limiting the received data to a predetermined set of levels. This distortion can, however, be made negligibly small by employing an analog–to– digital (A/D) converter with a sufficiently large number of levels. If the number of quantization levels is large and the input samples to the A/D converter are only "moderately correlated", then the noise due to quantization has a flat spectrum and can be treated as an additive noise that is uncorrelated with the input samples [13]. To obtain an estimate of the effect of quantization on the erasure and error probabilities, we will adopt this argument and determine the mean square distortion due to quantization. This mean square distortion will then be added to the variance of the Gaussian noise component to determine a signal–to–noise ratio for the case of additive channel and quantization noises.

At the end of Chapter 4, we gave the requirements for a 2^{16} vector code that will be used in the simulation. The components of the code vectors take on integer values in the range –6 to +5. When designing a quantization function for the demodulator, the quantization levels should be uniformly distributed over the range of possible signal levels so that there is an equal number of quantization levels between modulation levels. As well, each modulation level should be at the centre of a specific quantization level, and not the edge, to ensure that the quantization will not introduce distortion when no noise is present in the received signal. The quantization function that fits this description for the application of 6 bits of quantization is shown Figure 5.1. The 64 levels of the quantization function $q(r)$ are on uniform centres from –6.8 to +5.8 with a step size of $\Delta = 0.2$. By applying this

quantization function to the analog discriminator function a_i, we obtain its discrete form shown in Figure 5.2. The quantized form of b_i follows a similar pattern to that of a_i. It is these quantized discriminator functions which will be implemented in look–up tables for the simulation. For 8 bits of quantization, the set of 256 levels are uniformly spaced from –6.9 to +5.85 at equal intervals of $\Delta = 0.05$. With the 8 bit quantizer having four quantization levels for every one of the 6 bit A/D, it is easily seen through the one cycle of a_i shown in Figure 5.3 that the 8 bit A/D will much more closely approximate the original analog discriminator function.

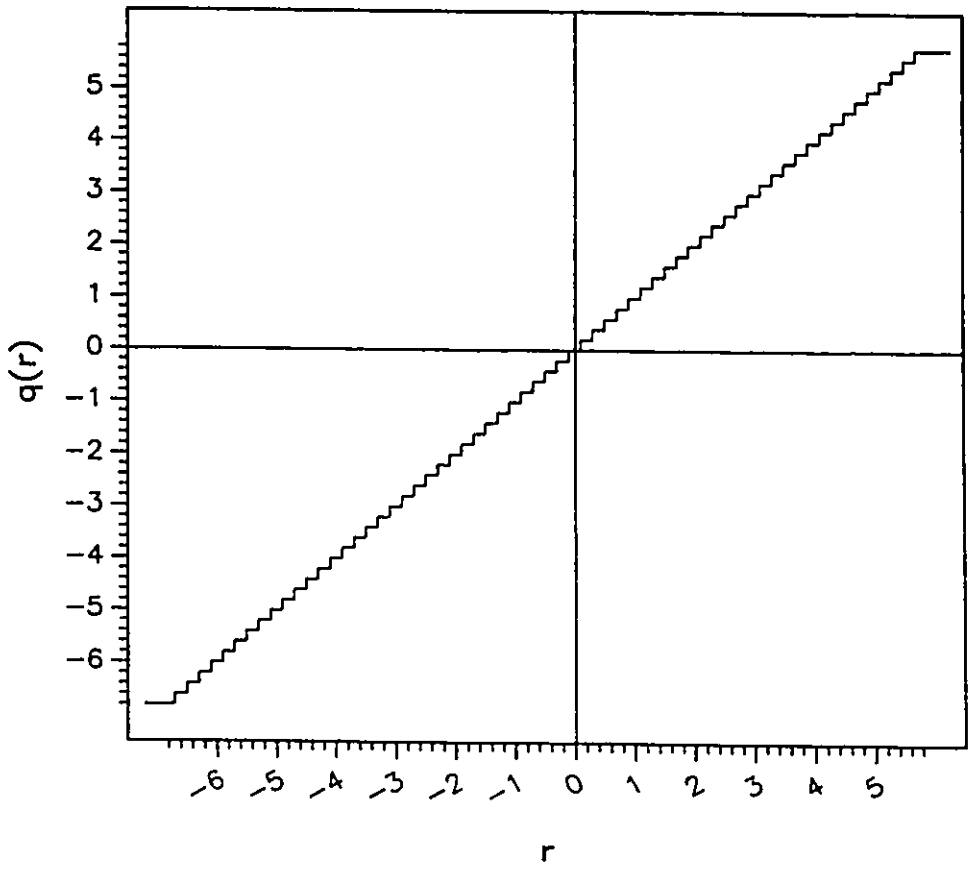

Figure 5.1: A 64 level quantization function $q(r)$ for a code whose components take on integer values between –6 and +5

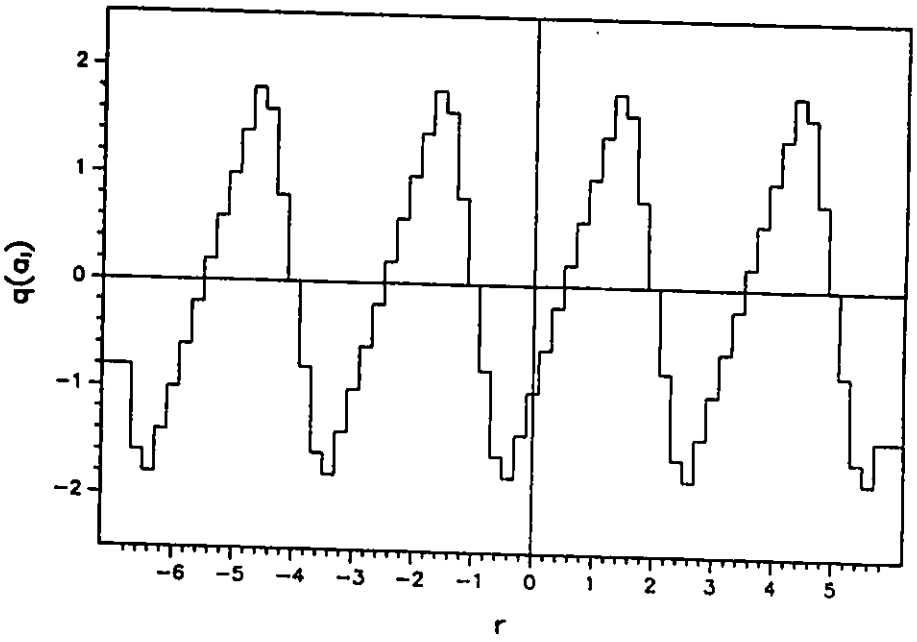

Figure 5.2: Discrete form of the discriminator function a_i obtained by applying the quantization function of Figure 5.1

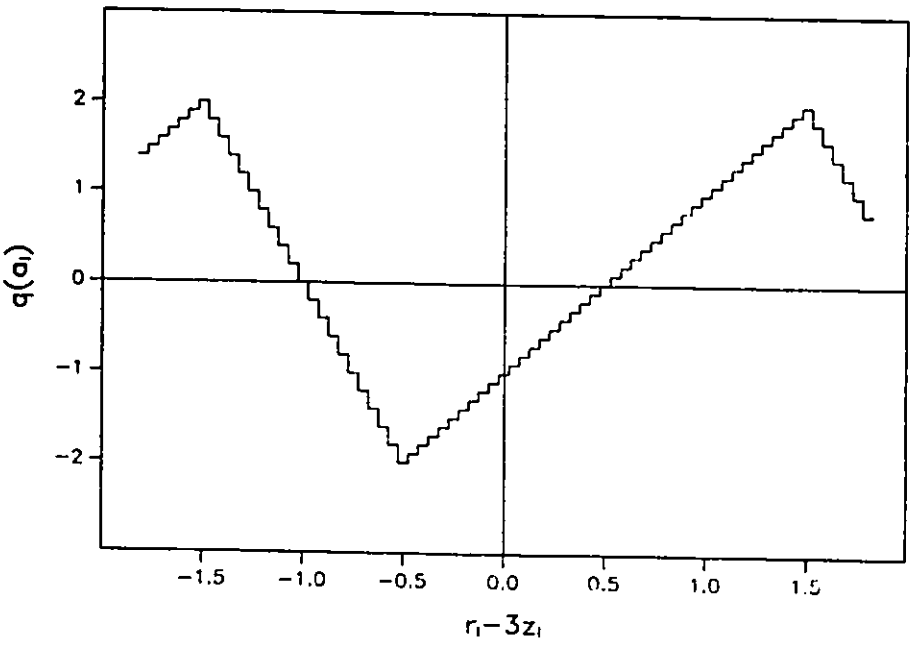

Figure 5.3: One cycle of the discriminator function a_i quantized using an 8 bit A/D.

The mean square distortion D introduced by quantizing any real $r \in \mathbb{R}$ with an N level quantizer $q(r)$ is given by [13]

$$D = \int_{-\infty}^{+\infty} [q(r) - r]^2 \, p(r) \, dr \tag{5.1}$$

where $p(r)$ is the probability density function of the random variable r. If the real line is broken up into a set of N intervals I_k such that when $r \in I_k$, the output of the quantizer is the k^{th} level $q(r) = y_k$, then the mean square distortion D can be evaluated as [13]

$$\begin{aligned} D &= \sum_{k=1}^{N} \int_{I_k} [y_k - r]^2 \, p(r) \, dr \\ &= \int_{-\infty}^{y_1 + \frac{\Delta}{2}} [y_1 - r]^2 \, p(r) \, dr + \sum_{k=2}^{N-1} \int_{y_k - \frac{\Delta}{2}}^{y_k + \frac{\Delta}{2}} [y_k - r]^2 \, p(r) \, dr \\ &\qquad + \int_{y_N - \frac{\Delta}{2}}^{+\infty} [y_N - r]^2 \, p(r) \, dr \end{aligned} \tag{5.2}$$

Equation (5.2) can be evaluated numerically to determine the mean square distortion D, but first we must define the probability density function $p(r)$ of a received component r. Each received component $r = x + w$ will consist of a signal component x, which can be any integer between –6 and +5, and an additive white Gaussian noise component w of zero mean and variance σ^2. The probability density function of the received component r is then a sum of Gaussian density functions of mean x and variance σ^2,

$$p(r) = \sum_{x=-6}^{+5} (\sqrt{2\pi}\sigma)^{-1} \cdot exp(-[r-x]^2/2\sigma^2) \cdot P(x) \tag{5.3}$$

where $P(x)$ is the probability that the signal component takes on the integer value x. Table 5.1 lists the probabilities $P(x)$ for each component value in the range

$-6 \leq x \leq +5$.

x	$P(x)$
−6	0.020755
−5	0.077438
−4	0.117611
−3	0.035929
−2	0.097786
−1	0.112321
0	0.055564
1	0.112321
2	0.097786
3	0.077438
4	0.117611
5	0.077436

Table 5.1: Probability $P(x)$ that a component of a code vector will take on the value x.

Using these probabilities and substituting equation (5.3) into (5.2), we can evaluate the mean square distortion D for various values of the signal–to–noise ratio. Table 5.2 lists the mean square distortions of the 6 and 8 bit quantizers for the range of signal–to–noise ratios from 16 dB to 21 dB. Gersho [13] has stated that for the case of uniform quantization, the mean square distortion can be closely approximated by

$$D \approx \frac{\Delta^2}{12}. \tag{5.4}$$

Although this is a fairly poor approximation at low signal–to–noise ratios, the mean square distortion does approach the values of 3.33×10^{-3} ($\Delta = 0.2$) and 2.08×10^{-4} ($\Delta = 0.05$) predicted by equation 5.4 as the signal–to–noise ratio improves.

The final step in determining the effects of quantization on the probabilities of erasure and error is to adjust the signal–to–noise ratio to indicate the degradation due to quantization. In equation (3.10), a signal–to–noise ratio of $\rho = E_a/9\sigma^2$ was

used to evaluate the probability of erasure, while in equation (3.11), the probability of error for maximum likelihood demodulation was evaluated using $\rho = E_a/8\sigma^2$. To the denominator of both these signal–to–noise ratios we must add a value of $9D$ indicating the accumulated mean square distortion due to the quantizing of the received vector. Using the adjusted value of $\rho' = E_a/(9\sigma^2+9D)$ in equation (3.10), the probability of erasure was calculated for both 6 and 8 bits of quantization and plotted against the original signal–to–noise ratio ρ. Since the simulation to follow uses the selective erasure correction algorithm of section 3.6, the effects of quantization on the error performance of this scheme was calculated, (again for 6 and 8 bits of quantization), using the adjusted signal–to–noise ratio $\rho' = E_a/(8\sigma^2+9D)$. These results are shown in Figure 5.4 for 6 bits of quantization and Figure 5.5 for 8 bits of quantization. Figure 5.4 indicates that a small amount of distortion, on the order of 0.1 dB, is to be expected with 6 bits of quantization, while almost no loss in performance is seen in Figure 5.5. One can then come to the conclusion that 8 bits of quantization should be sufficient for a digital implementation of the demodulator to obtain the performance predicted by theory. For those cases where the conversion speed of the A/D converter sets the limit on system speed, and a sufficiently fast 8 bit A/D cannot be found, 6 bits of quantization still provides acceptable error performance.

SNR (dB)	Mean Square Distortion D 6 bits	8 bits
16.0	4.037×10^{-3}	7.406×10^{-4}
17.0	3.687×10^{-3}	4.594×10^{-4}
18.0	3.502×10^{-3}	3.165×10^{-4}
19.0	3.412×10^{-3}	2.503×10^{-4}
20.0	3.374×10^{-3}	2.231×10^{-4}
21.0	3.133×10^{-3}	2.133×10^{-4}

Table 5.2: The mean square distortion due to 6 and 8 bits of quantization evaluated at various signal–to–noise ratios

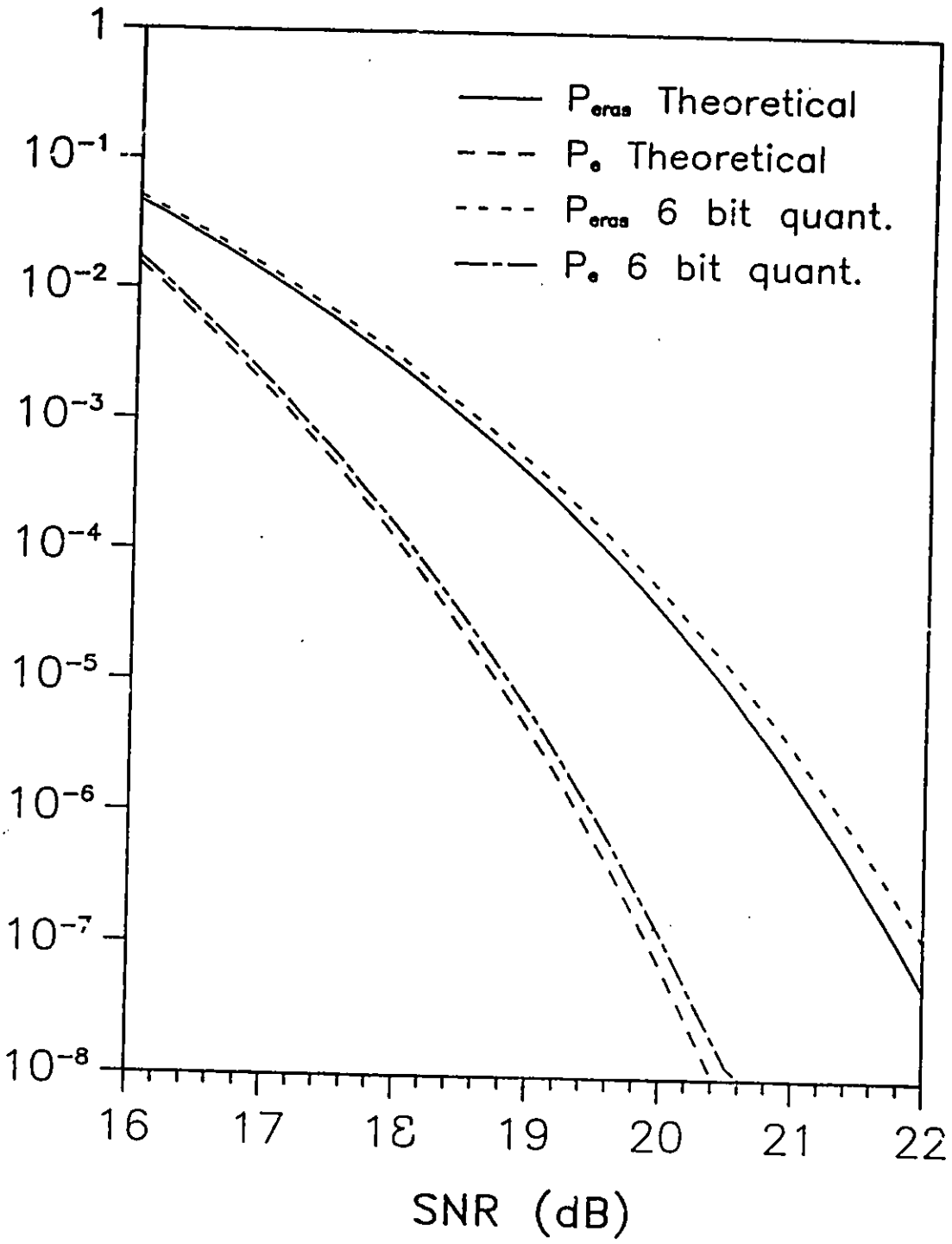

Figure 5.4: The probabilities of erasure and error for an unquantized system and for 6 bits of quantization

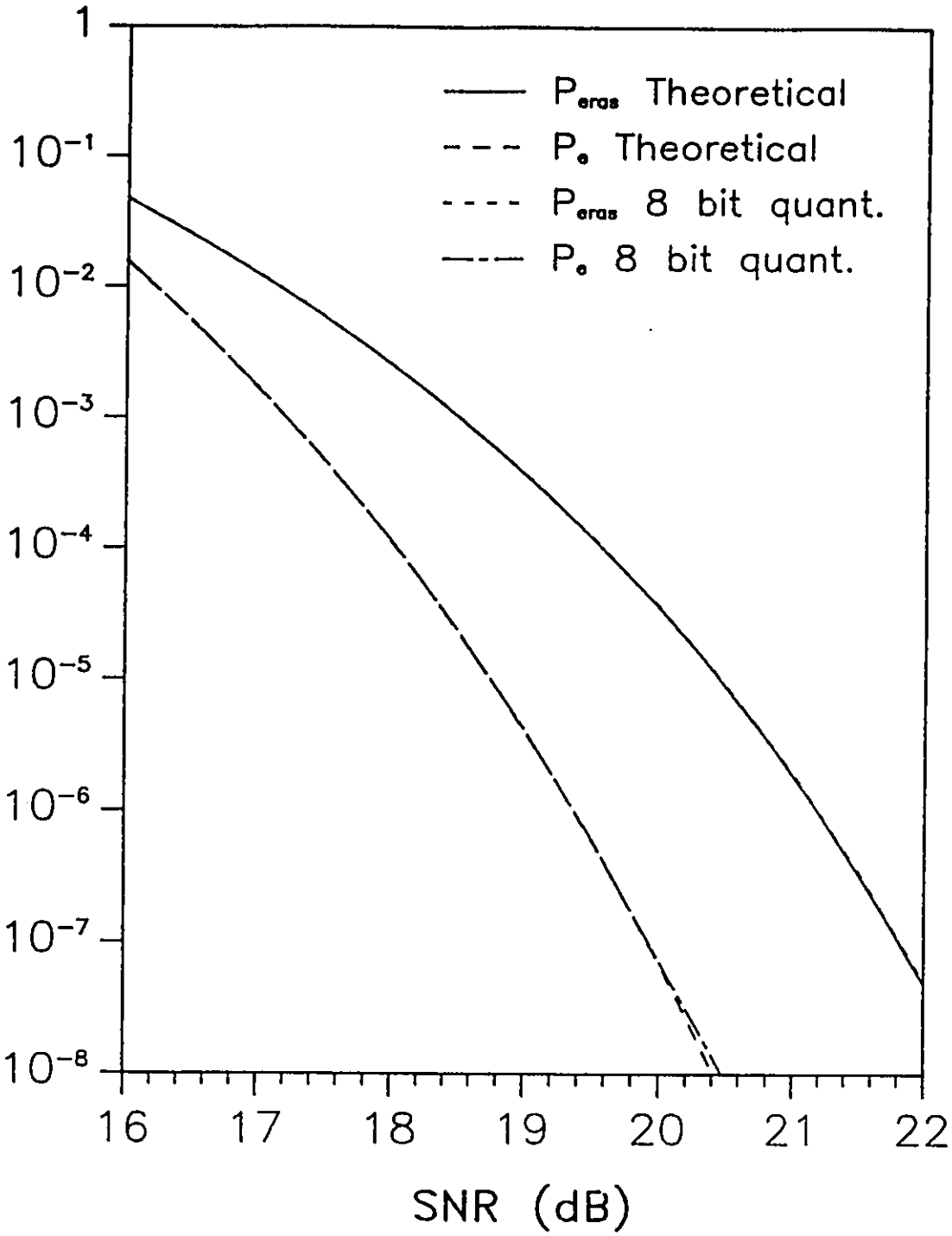

Figure 5.5: The probabilities of erasure and error for an unquantized system and for 8 bits of quantization

5.2 Time Trials

The basic principle behind the design of the two–stage demodulator outlined in Chapter 3 was that greater speed could be obtained by dividing the demodulation algorithm into two discrete processes which could be run independently and simultaneously. In a real implementation, a separate processor or hard–wired circuit would be assigned to each stage and the circuitry developed to allow the L_9 algorithm to feed information to the erasure correction circuitry when necessary. However, the simulation study that was undertaken to verify the error performance of the demodulator was conducted on a personal computer with a single microprocessor. With only one processor, which did not support multi–tasking, it was obviously physically impossible to simultaneously run the two stages of the demodulator.

Despite this fact, a realistic simulation of the demodulator's performance could still be developed by assigning a timer or clock to each of the two stages. These clocks could be used to record the execution time as each process performs its designated task, and then compared to determine the operating conditions of each stage. Specifically, when an erasure occurs, the clock associated with the L_9 algorithm indicates the time at which the erasure occured. By passing this time to the erasure correction algorithm, and adding to it the time needed to correct the erasure, we obtain the time at which the erasure correction algorithm will next be free. We can then give the appearance of the two stages working simultaneously by comparing the time at which the erasure occured to the time when the correction algorithm is to be available and thereby determine whether the second stage is ready to correct another erasure or is still busy correcting the first erasure.

There are, however, certain practical problems that must be contended with in implementing this timing procedure. The largest problem faced is that the system

clock of the microprocessor, (the sole source of timing information), does not have sufficient accuracy to measure each execution of the L_9 algorithm or the erasure correction algorithm. Also, there is a periodic interupt that is executed to update the system clock. If this interupt were to occur while either algorithm was working, it would lengthen the algorithm's execution time, leading to erroneous results. Consequently, timing information had to be obtained through a series of time trials and the results of these trials could then be used to assign fixed times to each stage.

The time trials consisted of isolating that portion of the demodulation process whose time we wished to measure and then that set of operations was repeated over and over again within a loop. By measuring the total execution time of the loop, an average run time for the process could be computed. The first set of time trials conducted was to determine the length of the L_9 algorithm. Before they could begin, it was of course necessary to perform the initial steps of encoding a vector, generating a noise vector and adding it to the encoded vector, and then quantizing the resultant received vector. Once this was done, a timer was started, the loop was entered and the L_9 algorithm repeatedly demodulated the received vector for 50000 loop iterations. When the loop terminated, the difference between the start and finish times was used to compute an average exectution time for the L_9 algorithm.

The time trials for the erasure correction algorithm were only slightly different. Table 3.6 gives a different number of operations for each path of the selective erasure correction algorithm. Consequently, it was necessary to go through the processes of encoding, noise generation, L_9 demodulation and erasure correction until it was found that the erasure correction algorithm terminated along a pre–designated path. Once such an event was found, the particular received vector which created the event was repeatedly demodulated by the L_9 algorithm and the erasure correction scheme in a loop of 50000 iterations. Since the execution time for

the L_9 algorithm was already known, its value could be subtracted out and an average time computed for correcting an erasure along the designated path.

The results of the time trials are summarized in Table 5.3. The first column of numbers is the accumulated time of 20 trials of 50000 loop iterations each. For each path of the erasure correction algorithm the 1874.07 seconds associated with the 20 trials of the L_9 algorithm has already been subtracted out to arrive at an average path time. By dividing the times in the first column by 1000000, we arrive at an average time per iteration. For path i, there are three times that had to be recorded. The first is for making no corrections; either the third candidate vector in L_9 has a zero component sum or its component sum is greater than 6 and the null vector is output. The second time is for making one correction to the third candidate vector in L_9 and the third time is for making two corrections to this vector. Unfortunately, we were not able to obtain any timing information for the third case. After 48 hours of continuous running, the time trial program had not been able to find one single incidence of the erasure correction algorithm correcting two components of the third candidate vector. Since it seemed doubtful that the time trials could be completed in any reasonable amount of time, (i.e., anything under a month), we decided not to pursue this case any further. We would expect the results of the time trials to have been very similar to that of path h since in both cases two components are corrected. It therefore would seem a reasonable compromise to use the information from path h for this third case in path i.

To compare the results of the time trials with what was predicted in theory through counting demodulation steps, the second column gives a ratio of average path time to average L_9 demodulation time and the third column is a ratio of the number of steps along the particular path of the correction algorithm to the 19 steps of the L_9 demodulation algorithm. A comparison of the two columns seems to

indicate that the theoretical count of the number of steps overbounds what is experienced in practice. The count of the number of steps is a fairly accurate representation of what will take place in a hardware implementation since the individual elements of the circuit can be synchronized so that each operation requires a fixed amount of time. In a software implementation, however, one has to be a little more careful in counting steps since different operations within the microprocessor's instruction set require different numbers of machine cycles.

Path	Accumulated Time Of 20 Trials	Path Time / L_9 Time	Path Steps / L_9 Steps
L_9 Algorithm	1874.07 s.	——	——
Path a	1881.77 s.	1.004	1.211
Path b	1884.20 s.	1.005	1.263
Path c	3606.57 s.	1.924	2.421
Path d	1933.48 s.	1.032	1.316
Path e	219.83 s.	0.117	0.105
Path f	1871.51 s.	0.999	1.263
Path g	2066.61 s.	1.103	1.632
Path h	2092.74 s.	1.117	1.684
Path i	336.36 s.	0.179	0.263
	1991.86 s.	1.063	1.368
	not available	——	1.684

Table 5.3: Results of time trials conducted to determine the execution time along each path.

Part of the discrepancy between the observed ratio of path time to L_9 algorithm time and that predicted by theory can also be attributed to the fact that in implementing the L_9 algorithm in software, the look–ups and adds were performed sequentially and not in parallel as discussed in Chapter 3. A parallel implementation could have been developed by carefully concatenating the three components and their associated a_i and b_i values into a single 32 bit word. For the

particular code implemented, the number of bits per component is small enough that the parallel additions outlined in the algorithm could have been performed in 32 bit arithmetic without having the carry bits from each addition run into one another. However, the microprocessor used is a 16 bit microprocessor with an 8 bit data bus. Although it is possible to define each operation as a long word or long integer operation, the number of cycles needed to perform 32 bit arithmetic takes away from any gains made in going to a parallel implementation. Another alternative approach would have been to determine the average time per look–up and per addition and then compute a relative time for what a parallel implementation should require, but this would not have been a fair representation of what was taking place within the simulation. Since our primary concern was a simple demonstration of the demodulator's error performance and not necessarily raw speed, the decision was made to use the sequential implementation of the L_9 algorithm and to abide by the outcome of the time trials seen in Table 5.3.

Within the simulation, the timing of the two stages can be done on a relative basis using the ratios of the average correction time to the L_9 demodulator time found in the second column of Table 5.3. By using a normalized timing, the count of the number of tests performed can serve as the clock for the L_9 demodulator. When an erasure occurs, the test count indicates the point at which the erasure occured and if the erasure correction algorithm is available to correct the erasure, adding the test count to the appropriate ratio from Table 5.3 gives a relative measure of when the erasure correction algorithm will next be able to correct an erasure. At the occurrence of the next erasure, the clock of the erasure correction algorithm is checked to determine if the algorithm is busy or idle.

5.2 Outline Of The Simulation

Figure 5.6 shows a flow chart of the simulation. The simulation software implementing this flow chart was written in the *C* programming language. The advantage to using a structured language such as *C* is that a separate self–contained function can be written to implement each major block such as encoding, noise generation, L_9 demodulation and erasure correction. The main body of the program then consists of a sequence of calls to the individual functions. By a judicious choice of function names, it becomes quite simple to read the program listing and relate it to what is seen in the flow chart.

Before the actual simulation runs can begin, the look–up tables for the encoder and the L_9 demodulator must be generated as the first two blocks of the flow chart indicate. The tables for the L_9 algorithm are set up to accept either 6 bits or 8 bits of input, whichever is specified at the start of the simulation. There are 5 separate look–up tables used in the L_9 demodulation algorithm, one to look–up the component values of the candidate vector in each of the three cosets and one for each of the discriminator functions a_i and b_i.

The tables for the encoder are the complementary tables to the decoder look–up tables discussed in Chapter 4. To be compatible with the decoder tables, each coset has its own set of look–up tables, each covering a specified range of message values. The first step in encoding is thus to find the range of values into which the 16 bit, randomly generated, message m falls and thereby decide to which coset the encoded vector should belong. Following this, the first look–up table is scanned to find the greatest value of m_2' less than or equal to the message value m. The first two components, x_1 and x_2, of the encoded vector x are then the components associated with this value of m_2'. We then compute the difference between the data value m and m_2' so that it can be passed to the next table along

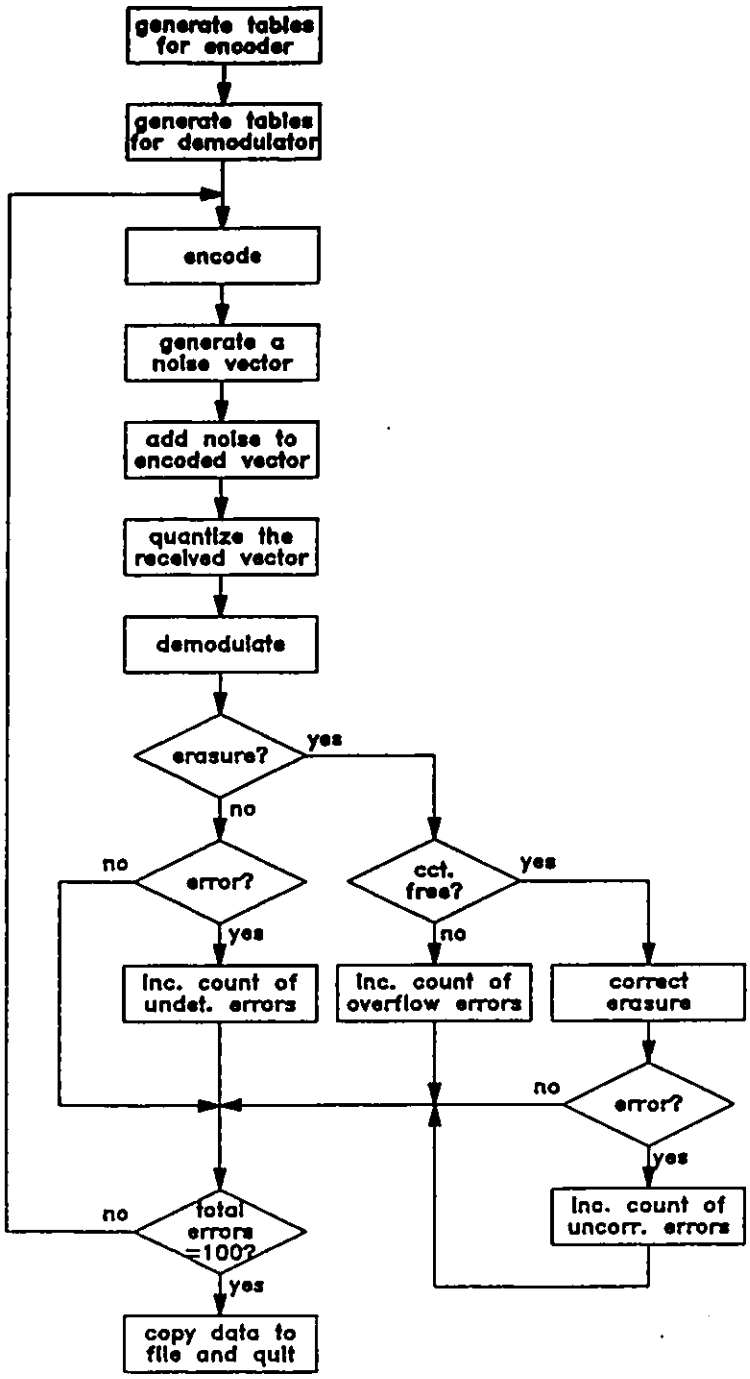

Figure 5.6: Flow chart for the Monte–Carlo simulation

with the sum $s_2 = x_1 + x_2$ of the first two components. The sum s_2 and the difference value $m - m_2'$ are then used in the next table to find the greatest integer m_3 less than or equal to $m - m_2'$ and the component x_3 associated with this value of m_3. Again the sum $s_3 = s_2 + x_3$ and the difference $m - m_2' - m_3$ are computed and sent to the next table to find the fourth component x_4. The same procedure is repeated in the subsequent encoder look–up tables until all the components of the encoded vector are obtained. Clearly, this is the reverse of the procedure used in the decoder look–up tables where the components of the demodulated vector were sent into a succession of tables to find the m_h values whose sum is the desired estimate $\hat{m}$ of the message sequence.

Following the encoding of a message sequence, a noise vector must be generated and added to the encoded vector to obtain the received signal vector. The noise generation algorithm encorporates the functions "*gasdev*( )" and "*ran0*( )" of [48,Ch.7] to generate a 9–dimensional vector of Gaussian deviates. The random number generator *rand*( ) supplied with this and most other systems uses an *m*–sequence approximation to the uniform distribution that is generally of poor quality because of short term sample to sample correlations [48, p.206]. The function *ran0*( ) serves as a "scrambler", removing any correlation between samples of the system deviate generator by shuffling them around, thereby providing a better approximation to the uniform distribution on the interval [0,1). To generate a vector of Gaussian deviates of a given variance, the function *gasdev*( ) [48,p.217] uses the the so–called "Box–Muller transformation". The advantage to using this transformation is that it removes the need for the sine and cosine function calls used in the more common transformation [1]

$$x_1 = \sqrt{-2\ ln\ u_1}\ cos(2\pi u_2) \qquad (5.5a)$$

$$x_2 = \sqrt{-2\ ln\ u_1}\ sin(2\pi u_2) \qquad (5.5a)$$

to generate zero mean, unit variance Gaussian deviates x_1, x_2 from a pair of uniform deviates $u_1, u_2 \in [0,1)$. With the Box–Muller transformation, we search for a pair of uniform deviates $v_1, v_2 \in [-1,1)$ that lie within the unit circle (i.e. $R = v_1^2 + v_2^2 \leq 1$), then the desired Gaussian deviates are given by

$$x_1 = v_1 \cdot \sqrt{-2 \ln R / R}\ , \tag{5.6a}$$

$$x_2 = v_2 \cdot \sqrt{-2 \ln R / R}\ . \tag{5.6b}$$

The ratio of the area of a unit circle to a square of side two is 0.785. Thus only 78.5% of all pairs (v_1, v_2) will lie within the unit circle, however, this will still prove to be faster than evaluating the sine and cosine functions.

After the noise vector has been added to the encoded vector, the received vector is quantized to either 6 or 8 bits and the resultant quantized received vector is demodulated by the L_9 algorithm. Once the closest point in L_9 has been decided, the component sum is checked to determine if an erasure has occured. If no erasure has occured but there is an error in the demodulated vector, an undetectable error is counted.

In section 3.6 it was stated that a single element queue would be placed at the front of the erasure correction algorithm to allow for the correcting of back–to–back erasures. Since the average length of the erasure correction algorithm is just slightly greater than that of the L_9 demodulator, as Table 5.3 shows, the single element queue simply provides the little extra time needed to correct the first erasure. There is a remote possibility that a third erasure event might occur before the second is corrected, and here there may be a problem with overflow errors. It may happen that either of the first two erasures, or both, is corrected along path *c*. If so, the erasure correction algorithm will not be available to correct the third erasure until between one and two demodulation intervals after its occurrence. In other words, the third erasure will stay in the queue longer than it should and there will

not be sufficient time to perform the corrections before the vector is scheduled to leave the buffer. It will therefore have to be counted as an overflow error. The probability that such an event should occur is extremely small, but nevertheless it must be taken into account in the simulation. This is why the decision block appears in the flow chart of Figure 5.6 asking if the erasure correction circuit is free. The erasure correction circuit is defined as being available if its clock value is less than the number of test plus one. Since the number of test serves as the clock for the L_9 algorithm, this says that the erasure correction circuit will become available sometime between the occurrence of the present erasure and the finish of the next demodulation interval, if is it not already available.

When an erasure has been corrected and the resultant vector does not agree with the encoded vector, an uncorrectable error is scored. During the simulation, the three types of error events, (undetectable, uncorrectable and overflow), are accumulated to obtain a total error count. The simulation will countinue to loop around and perform tests until the total error count reaches 100. The accumulation of 100 errors is sufficient to give a one standard deviation confidence interval of ±10% of the experimental probability of error.

(There is a certain amount of inaccuracy in all simulations in which a finite number of trials of an experiment are performed. A confidence interval simply provides an interval about an experimental data point where one expects, with a certain level of confidence, that the true value should lie. For a Gaussian density function, the area within one standard deviation of the mean is roughly 67% of the total area. Thus in the present simulation where the noise is Gaussian distributed, a one standard deviation confidence interval says that we are 67% sure that the actual probability of error lies within the error bars. The confidence interval is measured by taking the square root of the number of errors and dividing it by the

number of tests.)

The simulation was run at signal–to–noise ratios ranging from 16 dB to 18.5 dB in 0.5 dB increments and the results of these runs are presented in Tables 5.4 and 5.5. Table 5.4 shows results for which the components of the received vector were quantized to 6 bits, while Table 5.5 is the results of the simulation in which 8 bits of quantization were used. In both sets of results, no overflow errors have been recorded. As was stated earlier, the chances of an overflow error occuring are so small that we did not expect to record such an event.

The experimental probabilities of erasure and error, and their confidence intervals, are listed in Tables 5.6 and 5.7 and then plotted in Figure 5.7 and 5.8 respectively. In both figures, the confidence intervals have been omitted because they were not large enough to be clearly seen and would only have confused matters.

	SNR (dB)					
Events	16.0	16.5	17.0	17.5	18.0	18.5
tests	18800	40578	108653	362874	1270937	5402898
erasures	1223	1651	2869	5430	10625	23704
undetectable	26	26	22	27	41	33
uncorrectable	74	74	78	73	59	67
overflow	0	0	0	0	0	0

Table 5.4: Simulation results using 6 bits of quantization in the demodulator.

	SNR (dB)					
Events	16.0	16.5	17.0	17.5	18.0	18.5
tests	20679	52660	127017	446559	1904440	7143853
erasures	1010	1526	2175	3844	8077	13848
undetectable	34	31	31	39	37	45
uncorrectable	66	69	69	61	63	55
overflow	0	0	0	0	0	0

Table 5.5: Simulation results using 8 bits of quantization in the demodulator.

SNR (dB)	P_{eras}	P_e
16.0	$(6.51 \pm 0.19) \times 10^{-2}$	$(5.32 \pm 0.53) \times 10^{-3}$
16.5	$(4.07 \pm 0.10) \times 10^{-2}$	$(2.46 \pm 0.25) \times 10^{-3}$
17.0	$(2.64 \pm 0.05) \times 10^{-2}$	$(9.20 \pm 0.92) \times 10^{-4}$
17.5	$(1.50 \pm 0.02) \times 10^{-2}$	$(2.76 \pm 0.28) \times 10^{-4}$
18.0	$(8.36 \pm 0.08) \times 10^{-3}$	$(7.87 \pm 0.79) \times 10^{-5}$
18.5	$(4.39 \pm 0.03) \times 10^{-3}$	$(1.85 \pm 0.19) \times 10^{-5}$

Table 5.6: Experimental probabilities of erasure and error for 6 bits of quantization.

SNR (dB)	P_{eras}	P_e
16.0	$(4.88 \pm 0.15) \times 10^{-2}$	$(4.84 \pm 0.48) \times 10^{-3}$
16.5	$(2.90 \pm 0.07) \times 10^{-2}$	$(1.90 \pm 0.19) \times 10^{-3}$
17.0	$(1.71 \pm 0.04) \times 10^{-2}$	$(7.87 \pm 0.79) \times 10^{-4}$
17.5	$(8.61 \pm 0.14) \times 10^{-3}$	$(2.24 \pm 0.22) \times 10^{-4}$
18.0	$(4.24 \pm 0.05) \times 10^{-3}$	$(5.25 \pm 0.53) \times 10^{-5}$
18.5	$(1.94 \pm 0.02) \times 10^{-3}$	$(1.40 \pm 0.14) \times 10^{-5}$

Table 5.7: Experimental probabilities of erasure and error for 8 bits of quantization.

Both Figures 5.7 and 5.8 indicate that for low signal–to–noise ratios, the union bound on the probability of error significantly overbounds the experimental error rate but the bound does perform well for high signal to noise ratios. This is a common observation whenever a union bound is used to approximate the probability of error. In a union bound, one draws a hyperplane between any point and its nearest neighbours and integrate the noise density function, centred about the transmitted signal point, in the region of space belonging to the neighbour. By accumulating or forming the union of all such integrals, one arrives at an

approximation to the probability of error. There is a certain amount of overlap in the regions of space bounded by these hyperplanes and thus a certain portion of space will be covered by a number of integrations leading to an overbounding of the probability of error. When the noise variance is large, as it is for low signal–to–noise ratios, a significant portion of the noise density function will lie in these regions of overlap. Thus the accumulated effect is an over exaggerated estimate of the probability of error. On the hand, when the noise variance is small, very little of the noise density function lies in the overlapped region and thus we obtain an estimate that is much closer to the true probability of error.

The two sets of simulation results for the probability of erasure exhibit the opposite trend to that observed for the probability of error, starting out by following the theoretical curve for the probability of erasure and then deviating more and more as the signal–to–noise ratio increases. This deviation can most likely be traced to the effects of quantization. As the variance of the Gaussian noise component decreases, one expects the quantization noise to account for a larger percentage of the total noise and as such have a much more prominent impact on the systems performance. If this is the case with the results that we see in Figures 5.7 and 5.8, then quantization has a much greater effect on the probability of erasure than we had previously predicted. In particular, there is a rather dramatic increase in the probability of erasure when 6 bits of quantization are employed. Nevertheless, the results are encouraging in that the probability of an erasure P_{eras} still runs at something less than the square root of the probability of error P_e for both 6 and 8 bits of quantiztion. Thus at a reasonable error rate such as 10^{-6}, fewer than one in every thousand received vectors will create an erasure. With erasure occurring this infrequently, the two–stage demodulator should have no problems making the necessary corrections and maintaining error performance.

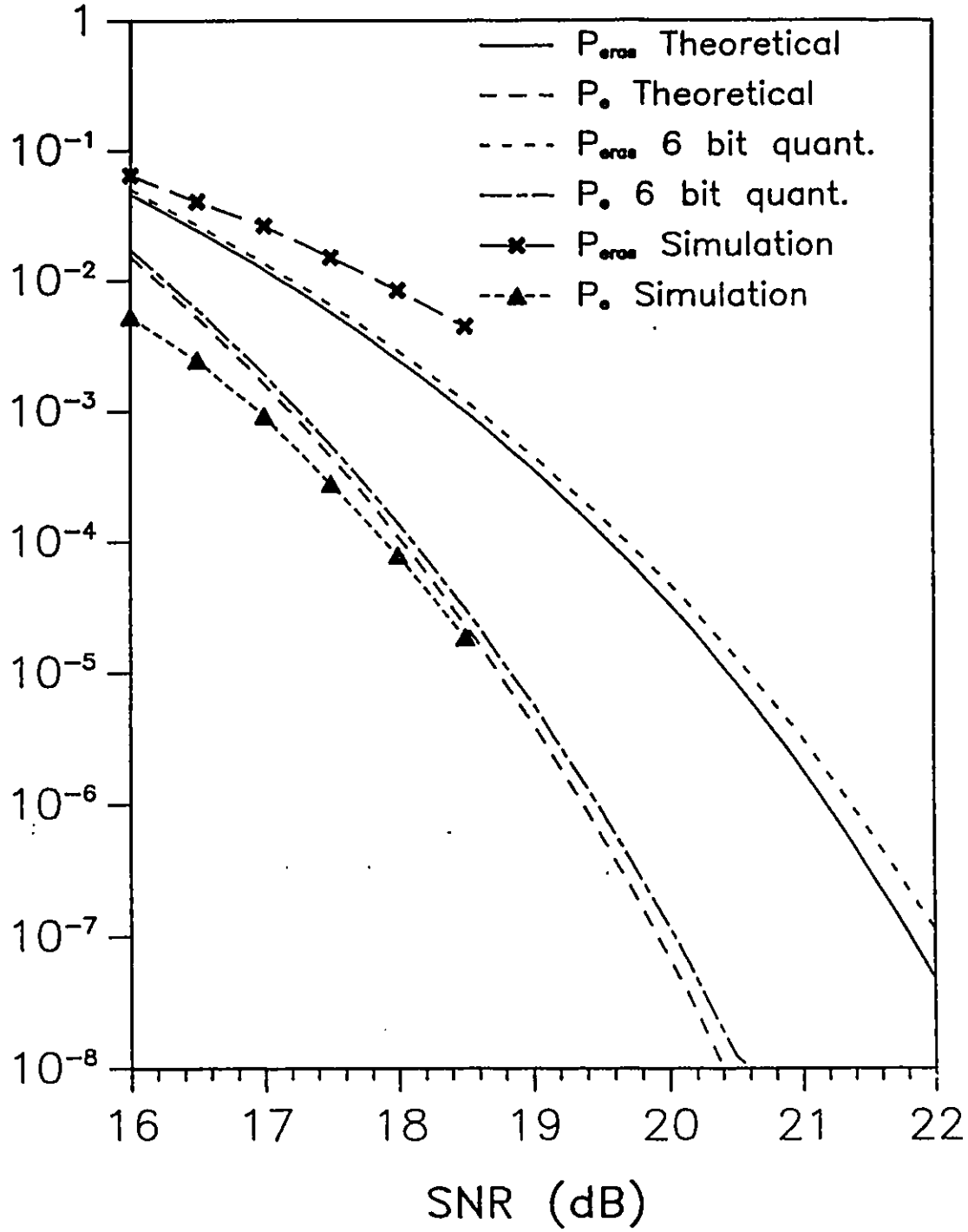

Figure 5.7: Plot of the simulation results versus theoretically calculations for 6 bits of quantizations

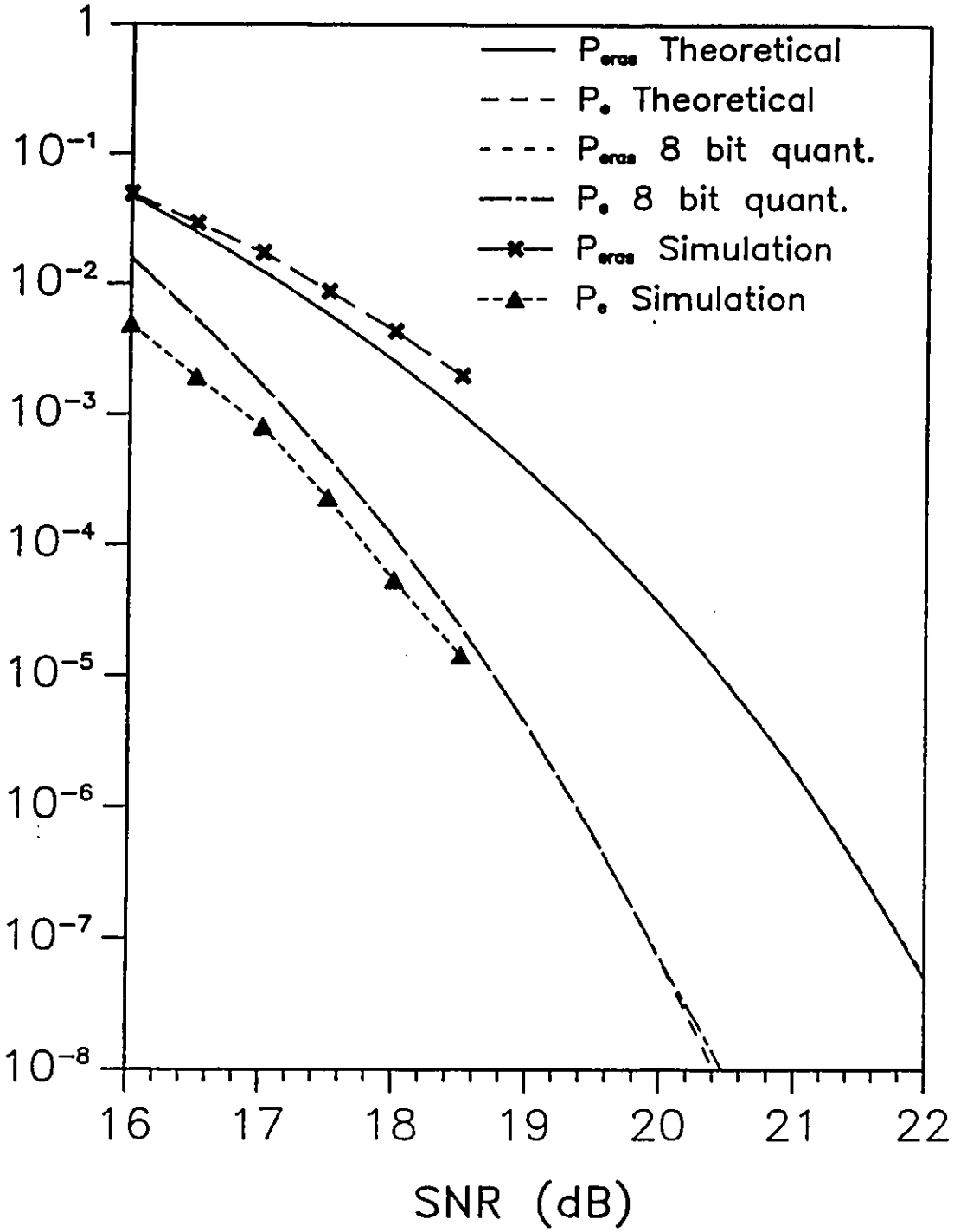

Figure 5.8: Plot of the simulation results versus theoretically calculations for 8 bits of quantizations

CHAPTER 6

CONCLUSIONS

6.1 Summary and Comments

The three main topics of discussion within this thesis have been (*i*) the A_8^3 Gosset lattice construction and its spectrum, (*ii*) the design of a fast two–stage demodulator for this construction of the Gosset lattice, and (*iii*) the designing of a set of decoder look–up tables to obtain a binary sequence from each demodulated vector. The present work is the first to point out that there exist lattice constructions with spectral shaping properties. Although we have focused on the Gosset lattice construction A_8^3, it is but one lattice in the series of lattices A_n^r [19] constructed through the union of r cosets of the lattice A_n, where $n+1/r$ is an integer. This series of lattices includes A_n $(r=1)$, A_n^* $(r=n+1)$, $A_7^2 = E_7$, and $A_7^4 = E_7^*$. The lattices A_n^r all have similar spectral characteristics, differing mainly in the width of the spectral null at dc which is inversely proportional to the dimension n of the lattice. The beauty of these lattice constructions is that the spectral null comes as a natural consequence of the choice of the coordinate system and does not require any additional effort, through either partial response filtering or running digital sum feedback, to induce a null in the spectrum. As with other such codes, however, there are small tradeoffs to be made. For the n–dimensional trellis codes with spectral nulls of Calderbank, Lee and Mazo [10], the tradeoff comes in the form of a power penalty of $6/n$ dB whereas for the codes derived from the lattices A_n^r, the bandwidth is a factor of $n+1/n$ larger than the minimum

bandwidth required by a comparable n–dimensional lattice code without a spectral null.

In Chapter 3, we outlined a two–stage approach to demodulating the Gosset lattice A_8^3. The motivation for the design of this demodulator was to find a way of approaching the speed of a hard decision demodulator without giving up the improved error performance afforded by soft decision techniques. The solution was to break up the demodulation process into two separate parts that could be run simultaneously. The initial stage uses the simpler L_9 lattice configuration to make a fast estimate of the closest vector in A_8^3. If the components of the closest point in L_9 sum to zero, then this is also the closest point in A_8^3 and nothing further is required. If the L_9 demodulator does not find the closest point in A_8^3, it declares an erasure and passes what information it has to a second stage which corrects the erasure. By connecting the two stages through a buffer, each stage is able to perform its designated task without slowing down the other. The speed or throughput of the demodulator is set by the speed of the initial stage, which is far less complex compared to a maximum likelihood demodulator.

The idea of using erasures to bridge the gap between hard and soft decision demodulation is not new, it was introduced in the mid–60's [24],[25] to improve the performance of decoding binary block codes, but this is the first instance in which erasures have been used in demodulating a lattice code. When the design of the two–stage demodulator was being formulated, it was thought that the erasure correction effort would be on the order of 10 to 20 times that required by the initial stage and that two erasure correction circuits would be required to keep the erasures from overflowing the buffer. We were therefore pleasantly surprised when it was discovered that we could approach maximum likelihood performance so very closely with such a simple erasure correction algorithm as that described in section 3.6.

To obtain a binary message sequence from each demodulated vector, we outlined in Chapter 4 a method of ordering and numbering the vectors of an A_8^3 lattice code. Although somewhat difficult and mathematically tedious to describe, this lexicographic ordering allows us to make efficient use of memory in the resultant look–up tables. The net result is that for a 2^{16} vector code, 36 kilobytes of memory are required for our complete set of decoder look–up tables compared to 256 megabytes of memory if a single look–up table had been used. We do, of course, have to take into consideration that 7 table look–ups and 6 additions must now be performed to find the binary sequence associated with any demodulated code vector. Also, the signal constellation is hypercubic in nature, making it less efficient in its use of power than a signal set enclosed within a hypersphere. However, our aim was to find a solution to the mapping problem that could be used in conjunction with the two–stage demodulator of Chapter 3. Since the 13 operations required by these decoder look–up tables is fewer than the number of operations in the L_9 algorithm, the lexicographic ordering provides a useful solution to the mapping problem where other approaches would not. We would therefore contend that the combination of speed and efficient use of memory compensates for the slight loss in power efficiency due to the shape of the constellation.

The simulation results that were presented in Chapter 5 indicate that 8 bits of quantization is sufficient to obtain the error performance predicted for the lattice code. Although a 6 bit quantizer suffers only a minimal amount of degradation in error performance over an 8 bit quantizer, there is substantially more degradation in the probability of an erasure. It is thus recommended that a minimum of an 8 bits A/D be used if at all possible.

As a final note, it should be stated that at the time this thesis was undertaken, there did not exist a decoder that had sufficient speed for a lattice code

to be used in a multi–megabit data transmission system. Our initial goal was to develop a decoder that could provide these high data rates, with an eye towards an eventual application in the digital microwave radio channel. We chose, however, not to develop a single fixed decoder but rather to develop the demodulator and the decoder look–up tables as separate, independent entities. The development was kept as general as possible to accomodate a large variety of codes when these individual parts were later integrated into a complete decoder structure. By taking this approach, we are not limited to a specific set of applications, nor is there anything to prevent one from using our demodulator design with a different decoder mapping process or our decoder look–up tables with a different demodulator. The approach that we have tried to maintain is that, since this is a theoretical study, there should be enough flexibilty for future applications of our results to be easily tailored to suit a wide range of individual needs.

6.2 Suggestions For Future Work

There is a common adage that for every question one is able to answer, several new and more involved questions are created. Although we have endeavoured to present a complete, self–contained piece of work, there are still a number of open problems which could be explored. Perhaps the most interesting theoretical problem is the extension of our results to higher dimensional lattices that offer greater amounts of coding gain. Coxeter [19] has pointed out that the series of lattices A_n^r becomes less and less dense as the dimension n increases above eight leaving little promise for more significant gains from this direction. However, it may be possible to find "partial response" forms of other well known lattices such as the Barnes–Wall lattices Λ_{16}, Λ_{32} and the Leech lattice Λ_{24} which offer asymptotic coding gains of 4.5 dB, 6 dB, and 6 dB, respectively [27]. The basic concept behind

the laminated lattice constructions of Conway and Sloane [17] is that many dense lattices can be shown to exist as planar cross sections in a lattice of the next higher dimension. The lattices A_n^r exhibit a spectral null because they are n–dimensional lattices embedded in a hyperplane in a space of $n+1$ dimensions. It would therefore seem that if one is able to find the appropriate planar cross section of Λ_{16}, Λ_{24} and Λ_{32} in 17, 25 and 33 dimensions, respectively, one could obtain constructions of these lattices that exhibit a spectral null. How easy it would be to define such constructions and the complexity of the demodulation problem are not known and would be worth studying.

At the outset of this thesis, we stated that we were not primarily concerned with the intersymbol interference problem or partial response signaling. However, because A_8^3 Gosset lattice codes fit into the $1-D$ partial response channel, they should perform well in an intersymbol interference environment. How well they will perform relative to other partial response signaling schemes, and relative to their own performance on the Gaussian channel, has still to be determined. It is felt that one should be able to sustain a great deal more closure in the eye pattern, (a general measure of performance in an intersymbol interference environment), for a given level of error performance but how much more we cannot say. As far as we know, no such study has been undertaken for the trellis codes with spectral nulls of [10] either. An interesting research problem would then be to do a comparative study of the two types of dc free codes in a severe intersymbol interference environment.

Finally, we have stated in several places that this decoder was developed for application in multi–megabit data transmissions. We feel that it should be possible to develop a hardware implementation of the decoder for use in a 90 megabit digital radio but cannot say this with absolute certainty. It is left for future development to see whether this is indeed feasible.

REFERENCES

[1] M. Abramowitz and I. Stegun, *Handbook of Mathematical Functions*, Dover Publications, 1970.

[2] J.P. Adoul, C. Lambelin and A. Leguyader, *Baseband speech coding at 2400 bps using spherical vector quantization*, IEEE Conf. Rec. CH1945–5, ICASSP'84, pp. 1.12.1–1.12.4, 1984.

[3] I.F. Blake, *The Leech lattice as a code for the Gaussian Channel*, Infor. and Control, vol. 19, pp. 66–74, 1971.

[4] P. de Buda, *Encoding and decoding algorithms for an optimal lattice–based code*, IEEE Conf. Rec. ICC'81, pp. 65.3.1–65.3.5, Denver, June 1981.

[5] P. de Buda and R. de Buda, *Basis signals for a lattice code*, Communications Research Laboratory, McMaster University, Hamilton, Ontario, Internal Report No. 131, October 1984.

[6] R. de Buda, *The upper error bound of a new near optimal code*, IEEE Trans, Info. Theory, vol. IT–21, pp. 441–445, 1975.

[7] R. de Buda, *Some optimal codes have structure*, to appear IEEE J. Sel. Areas Comm., 1989.

[8] R. de Buda and W. Kassem, *About lattices and the random coding theorem*, in Abstracts of Papers, IEEE Inter. Symp. Info. Theory, February 1981; also available as Internal Report No. 84, Communications Research Laboratory, McMaster University, Hamilton, Ontario, December 1980.

[9] A.R. Calderbank, *Multilevel codes and multistage decoding*, IEEE Trans. Comm., vol. COM–37, pp. 222–229, March 1989.

[10] A.R. Calderbank. T.A. Lee and J.E. Mazo, *Baseband trellis codes with a spectral null at zero*, IEEE Trans. Info. Theory, vol. IT–34, pp. 425–434, May 1988.

[11] A.R. Calderbank and N.J.A. Sloane, *New trellis codes based on lattices and cosets*, IEEE Trans. Info. Theory, vol. IT–33, pp. 177–195, March 1987.

[12] G.L. Cariolaro and G.P. Tronca, *Correlation and spectral density of multilevelt (M,N) coded digital signals with applications to pseudoternary (4,3) codes*, Alta Frequenza, vol. 43, pp. 2–15, January 1974.

[13] J.H. Conway and N.J.A. Sloane, *Voronoi regions of lattices, second moments of polytopes and quantization*, IEEE Trans. Info. Theory, vol. IT–28, pp. 211–226, March 1982.

[14] J.H. Conway and N.J.A. Sloane, *Fast quantizing and decoding algorithms for lattice quantizers and codes*, IEEE Trans. Info. Theory, vol. IT–28, pp. 227–232, March 1982.

[15] J.H. Conway and N.J.A. Sloane, *A fast encoding method for lattice codes and quantizers*, IEEE Trans. Infor. Theory, vol. IT–29, pp. 820–824, November 1983.

[16] J.H. Conway and N.J.A. Sloane, *Soft decoding techniques for codes and lattices, including the Golay code and the Leech lattice*, IEEE Trans. Info. Theory, vol. IT–32, pp. 41–50, January 1986.

[17] J.H. Conway and N.J.A. Sloane, *Sphere Packings, Lattices and Groups*, Springer–Verlag, New York, 1988.

[18] H.S.M. Coxeter, *Integral Cayley Numbers*, Duke Math. J., vol. 13, No. 4, pp. 561–578, 1946; reprinted as Chapter 2, *Twelve Geometric Essays*, Southern Illinois University Press, 1968.

[19] H.S.M. Coxeter, *Extreme forms*, Can. J. Math., vol. 3, pp. 391–441, 1951.

[20] H.S.M. Coxeter, *An upper bound for the number of equal nonoverlapping spheres that can touch another of the same size*, Proc. Symp. Pure Math., vol. 7, pp. 53–72, 1963; reprinted as Chapter 9, *Twelve Geometric Essays*, Southern Illinois University Press, 1968.

[21] H.S.M. Coxeter, *Regular Polytopes*, Dover Publications, 3rd ed., New York, 1973.

[22] E.L. Cusack, *Error control codes for QAM signalling*, Electron. Lett., vol. 20, pp. 62–63, January 1984.

[23] ESE Ltd., *Block coding for improved modem performance*, Contribution to CCITT Study Group COM–XVII, No. 112, March 1983.

[24] G.D. Forney Jr., *Generalized minimum distance decoding*, IEEE Trans. Info. Theory, vol. IT–12, pp. 125–131, April 1966.

[25] G.D. Forney Jr., *Exponential error bounds for erasure, list and decision feedback schemes*, IEEE Trans. Info. Theory, vol. IT–14, pp. 206–220, March 1968.

[26] G.D. Forney Jr., *Maximum likelihood sequence estimation in the presence of intersymbol interference*, IEEE Trans. Info. Theory, vol. IT–18, pp. 363–378, May 1972

[27] G.D. Forney Jr., *Coset Codes I: Introduction and geometric classification*, IEEE Trans. Info. Theory, vol IT–34, pp. 1123–1151, September 1988.

[28] G.D. Forney Jr., *Coset Codes II: Binary lattices and related codes*, IEEE Trans. Info. Theory, vol IT–34, pp. 1152–1187, September 1988.

[29] G.D. Forney Jr. and A.R. Calderbank, *Coset codes for partial response channels; or coset codes with spectral nulls*, submitted to IEEE Trans. Info. Theory.

[30] G.D. Forney Jr., R.G. Gallager, G.R. Lang, F.M. Longstaff and S.U. Qureshi, *Efficient modulation for band–limited channels*, IEEE J. Sel. Areas Commun., vol. SAC–2, pp. 632–647, Sept. 1984.

[31] A. Gersho, *Principles of quantization*, IEEE Trans. Circuits and Syst., vol. CAS–25, pp. 427–436, 1978; reprinted in Quantization, Peter Swazek ed., Van Nostrand Reinhold Co., New York, 1985, pp. 7–16.

[32] A. Gersho and V.B. Lawrence, *Multidimensional signal constellations for voiceband data transmission*, IEEE J. Sel. Areas Commun., vol. SAC–2, pp. 687–702, Sept. 1984.

[33] T. Gosset, *On the regular and semiregular figures in space of n dimensions*, Messenger of Mathematics, vol. 29, pp. 43–48, 1900.

[34] S. Haykin, Communications Systems, 2nd ed., Wiley, New York, 1983.

[35] T.C. Hu, Combinatorial Algorithms, Addison–Wesley, Reading MA, 1982.

[36] P. Kabal and S.Pasupathy, *Partial response signaling*, IEEE Trans. Commun., vol. COM–23, pp. 921–934, Sept. 1975.

[37] W. Kassem, Optimal Lattice Codes for the Gaussian Channel, Ph D. dissertation, McMaster University, Hamilton, Ontario, 1981.

[38] D.E. Knuth, The Art of Computer Programming, Volume 3: Sorting and Searching, Addison–Wesley, Reading MA, 1973.

[39] H. Kobayashi, *A survey of coding schemes for transmission or recording of digital data*, IEEE Trans. Commun. Tech., vol. COM–19, pp. 1087–1100, December 1971.

[40] J. Leech, *Some sphere packings in higher space*, Can. J. Math, vol. 16, pp. 657–682, 1964.

[41] J. Leech, *Notes on sphere packings*, Can. J. Math, vol. 19, pp. 251–267, 1967.

[42] J. Leech and N.J.A. Sloane, *Sphere packings and error correcting codes*, Can. J. Math., vol. 23, pp. 718–745, 1971.

[43] S. Lin and D.J. Costello Jr., *Error Control Coding: Fundamentals and Applications*, Prentice–Hall, Englewood Cliffs NJ, 1983.

[44] P.A.P. Moran, *An Introduction to Probability Theory*, Oxford University Press , London, 1968.

[45] A. Papoulis, *Probability, Random Variables and Stochastic Processes*, 2nd ed., McGraw–Hill, New York, 1984.

[46] G. Pierobon, *Codes for zero spectral density at zero frequency*, IEEE Trans. Infor. Theory, vol. IT–30, pp. 425–429, March 1984.

[47] G.J. Pottie and D.P. Taylor, *Multilevel codes based on partitioning*, IEEE Trans. Info. Theory, vol. IT–35, pp. 87–98, January 1989.

[48] W.H. Press, B.P Flannery, S.A. Teukolsky and W.T. Vetterling, *Numerical Recipes in C, The Art of Scientific Computing*, Oxford University Press, New York, 1988.

[49] N. Secord and R. de Buda, *Microprocessor decoding of a lattice code*, Communications Research Lab. McMaster University, Hamilton, Ontario, Internal Report No. 130, February 1986.

[50] N. Secord and R. de Buda, *Demodulation of a Gosset lattice code having a spectral null at dc*, to appear IEEE Trans. Info. Theory.

[51] C.E. Shannon, *A mathematical theory of communication*, Bell Sys. Tech. J., vol. 27, pp. 378–423 and 623–656, 1948.

[52] C.E. Shannon, *Probability of error for optimal codes in a Gaussian Channel*, Bell Sys. Tech. J., vol. 38, pp. 611–656, 1959.

[53] D. Slepian, *Permutation modulation*, Proc. IEEE, vol. 53, pp. 228–236, March 1965.

[54] N.J.A. Sloane, *Binary codes, lattices and sphere packings*, in *Combinatorial Surveys*, ed. P.J. Cameron, pp. 117–164, Academic Press, New York, 1977.

[55] N.J.A. Sloane, *Tables of sphere packings and spherical codes*, IEEE Trans. Info. Theory, vol. IT–27, pp. 327–338, May 1981.

[56] N.J.A. Sloane, *The packing of spheres*, Scientific American, pp. 116–125, January 1984.

[57] T.M. Thompson, *From Error-Correcting Codes Through Sphere Packings To Simple Groups*, Carus Math. Monograph No. 21, Math. Assoc. of America, 1983.

[58] G. Ungerboeck, *Channel coding with multilevel/phase signals*, IEEE Trans. Infor. Theory, vol. IT-28, pp. 55–67, January 1982.

[59] J.M. Wozencraft and I.M. Jacobs, *Principles of Communication Engineering*, Wiley, New York, 1965.

[60] W.R. Bennett, *Statistics of regerative digital transmission*, B.S.T.J., vol. 37, no. 6, pp. 1501–1542, Nov. 1958

APPENDIX A

LINEAR TRANSFORMATIONS LINKING GOSSET LATTICE CONSTRUCTIONS

In this appendix, we redefine the three Gosset lattice constructions E_8, Λ_8 and A_8^3, and provide transformations that allow one to map the vectors defined under one construction into the vectors of another construction. These transformations make it theoretically possible to encode using one construction and decode using another, although from a practical point of view this is not a very efficient method of encoding and decoding. The main interest in such transformations lies in showing how the three different lattice constructions are related.

A construction of the Gosset lattice that appears quite often in the literature on lattices [42],[55], as well as applications of lattice codes [23],[32],[49] and lattice quantizers [2],[13],[14], is the following.

Definition 1: The Gosset lattice in 8–dimensional Euclidean space $\mathbb{R}^8$ is the set of all vectors whose eight components are

i) either all even or all odd integers, and

ii) whose sum is a multiple of 4.

This construction of the Gosset lattice is commonly given the symbolic notation E_8 and expressed by the formula

$$E_8 = 2D_8 \cup 2D_8 + (1,1,1,1,1,1,1,1) \qquad (A.1)$$

where D_8 is the lattice of all 8–tuples of integers with an even sum [13]. More recently, Forney [27] has identified this construction of the Gosset lattice as a decomposable modulo–4 lattice with code formula[7]

$$E_8 = 4\mathbb{Z}^8 + 2(8,7) + (8,1) \qquad (A.2)$$

where (8,7) and (8,1) are binary block codes of length 8, the former being a single parity check code and the latter a repetition code.

The beauty of the Gosset lattice lies in its structure and symmetry. The vectors of the lattice fall into discrete energy shells[8] that are multiples of the energy of the first shell. Within this first energy shell are the 240 minimal vectors that join any point within the lattice with its 240 nearest neighbours. For the above E_8 construction, these 240 minimal vectors consist of 112 vectors obtained through all permutations and sign changes made to the vector (2,2,0,0,0,0,0,0) and the 128 vectors obtained through the permutations of an even number of sign changes made to the vector (1,1,1,1,1,1,1,1). These 240 minimal vectors all have squared magnitude 8 and this is the minimum squared distance between any two points within this construction of the Gosset lattice.

It is also possible to construct the Gosset lattice using error correcting codes.

Definition 2: The Gosset lattice in $\mathbb{R}^8$ is the set of all vectors whose components are congruent modulo 2 to one of the sixteen codewords of the (8,4) first order Reed–Muller code.

This is what Leech and Sloane [42] refer to as the *Construction A* form of the Gosset lattice. This construction has the symbolic notation Λ_8 and can be expressed by the formula [16]

---

[7]Note that Forney refers to this construction of the lattice as RE_8.

[8]An energy shell is the set of all vectors of the same squared magnitude.

$$\Lambda_8 = c + 2z \qquad (A.3)$$

where c is a codeword of the (8,4) Reed–Muller code and $z \in \mathbb{Z}^8$. Slightly different versions of this definition result when one notes that the (8,4) first order Reed–Muller code is equivalent to the (8,4) extended Hamming code [55] and can also be identified with the Walsh functions of order 8 [4],[5].

The 240 minimal vectors of this Λ_8 construction consist of the 16 vectors obtained by all permutations and sign changes made to the vector (2,0,0,0,0,0,0,0) and the 224 vectors obtained by making all possible sign changes to the 14 vectors whose components form a codeword of Hamming weight 4 in the (8,4) Reed–Muller code.[9] These 240 vectors all have a squared magnitude or energy of 4 which is also the minimum squared distance between any pair of points within this construction of the Gosset lattice. Note that this is twice the energy of the minimal vectors in the E_8 construction, however for any two sets of vectors taken from the same region in $\mathbb{R}^8$, the ratio of the average energy of these vectors to the minimum squared distance of the construction must be the same. This is in fact true for all constructions of the Gosset lattice and is what ensures that codes of equal size taken from equivalent regions in $\mathbb{R}^8$ have equivalent error performance.

Leech [40] has given transformations which allow one to map the vectors of the E_8 construction into the vectors of the Λ_8 construction and vice versa. If $x = (x_1, x_2, \ldots, x_8)$ is a vector of the E_8 construction and $y = (y_1, y_2, \ldots, y_8)$ is a vector of the Λ_8 construction, then x and y are related by [40],[57,p.78]

$$\begin{aligned} x_{2i} &= y_{2i-1} - y_{2i}, \qquad y_{2i} = \tfrac{1}{2}(x_{2i-1} - x_{2i}), \\ x_{2i-1} &= y_{2i-1} + y_{2i}, \qquad y_{2i-1} = \tfrac{1}{2}(x_{2i-1} + x_{2i}), \end{aligned} \qquad (A.4)$$

where $i = 1,2,3,4$. The transformation which takes every vector $y \in \Lambda_8$ to a vector

---

[9]The Hamming weight of a binary codeword is the number of 1's within the codeword.

$x \in E_8$ is referred to alternatively as the *norm doubling map* [11] or the *rotation operator R* [26] because it rotates the lattice by 45 degrees and doubles the norm or squared magnitude of each vector in the lattice.

A third construction of the Gosset lattice, the one which the work of this thesis is based upon, has the following definition due to Coxeter [19], (see also [5],[18],[20], [50])

Definition 3: The Gosset lattice in $\mathbb{R}^8$ is the lattice whose vertices, in an orthonormal coordinate system in $\mathbb{R}^9$, are the set of vectors whose 9 integer components

i) are all congruent modulo 3 to each other, and

ii) have a sum of zero.

Coxeter [19] refers to this construction of the Gosset lattice as ${A_8}^3$ because it is the union of three scaled copies of the lattice A_8, the lattice of all 9–tuples of integers with a zero sum [13],[551], that is

$$ {A_8}^3 = 3A_8 \cup 3A_8 + (2^3,-1^6) \cup 3A_8 + (-2^3,1^6) \qquad (A.5) $$

where $(2^3,-1^6)$ is a compact notation for (2,2,2,–1,–1,–1,–1,–1,–1).

The 240 minimal vectors of this ${A_8}^3$ construction consist of the 72 vectors obtained by permuting the components of (3,–3,0,0,0,0,0,0,0) and the 168 vectors obtained through all permutations of the vector (2,2,2,–1,–1,–1,–1,–1,–1) and its negative (–2,–2,–2,1,1,1,1,1,1). From these vectors it is easily seen that the minimum squared distance between any pair of points in the ${A_8}^3$ construction of the Gosset lattice is 18.

As with the E_8 and Λ_8 constructions, there exists transformations to map every vector from one of these constructions to a vector of the ${A_8}^3$ construction and

vice versa. For the E_8 and A_8^3 constructions, a transformation which maps every $x \in E_8$ to a vector $w \in A_8^3$ is the following [5]

$$\begin{aligned} w_i &= \tfrac{3}{2} x_i + \tfrac{1}{2} x_8 - \tfrac{1}{4} s, && i = 1,2,\ldots 7 \\ w_8 &= -x_8 - \tfrac{1}{4} s \\ w_9 &= -x_8 - \tfrac{1}{2} s \end{aligned} \qquad (A.6)$$

where s is the sum of the components of x, i.e., $s = \sum_{i=1}^{8} x_i$. The inverse transformation which takes every $w \in A_8^3$ to a vector $x \in E_8$ is

$$\begin{aligned} x_i &= \tfrac{2}{3} w_i - \tfrac{1}{3} w_9, && i = 1,2,\ldots,7 \\ x_8 &= \tfrac{-2}{3} w_8 - \tfrac{1}{3} w_9 . \end{aligned} \qquad (A.7)$$

By substituting the equations for the x_i of (A.8) into the equations of (A.4), a transformation can be found to map every $w \in A_8^3$ into a vector $y \in \Lambda_8$.

APPENDIX B

REDUCTION OF THE CODE SPECTRUM EQUATION USING CHEBYSHEV POLYNOMIALS

In Section 2.2, the power spectrum of an A_8^3 Gosset lattice code was given as

$$\mathcal{C}(f) = \frac{E_a}{T_n}\left[\,1 - \frac{1}{9}\sum_{k=1}^{8}\frac{(9-k)}{4}\cos(2k\pi fT_n/9)\,\right], \qquad |f| \le \frac{9}{2T_n} \tag{B.1}$$

and it was stated that the summation of cosine terms could be reduced using Chebyshev polynomials of first and second kind. This reduction is done as follows.

Using the Chebyshev polynomial of the first kind [1, p.776],

$$T_k(\cos\theta) = \cos k\theta\,, \tag{B.2}$$

the summation on the right of (*B*.1) can be rewritten as

$$\sum_{k=1}^{8}\frac{(9-k)}{4}\cos(2k\pi fT_n/9) = \sum_{k=1}^{8}\frac{(9-k)}{4}T_k(\cos\theta) \tag{B.3}$$

where $\theta = 2\pi fT_n/9$. Using the identity [1, p.778]

$$T_k(x) = \tfrac{1}{2}[U_k(x) - U_{k-2}(x)]\,, \tag{B.4}$$

with the change of variable $x = \cos\theta$, and where $U_k(\cos\theta)$ is the Chebyshev polynomial of the second kind [1, p.776],

$$U_k(\cos\theta) = \frac{\sin(k+1)\theta}{\sin\theta}, \tag{B.5}$$

the summation on the right of (*B*.3) is rewritten as

$$\sum_{k=1}^{8}\frac{(9-k)}{4}T_k(x) = -\frac{9}{8} + \frac{1}{4}\sum_{k=1}^{8}U_k(x) - \frac{1}{8}U_8(x)\,. \tag{B.6}$$

The summation formulas [1, p.785],

$$\sum_{m=0}^{n}U_{2m}(x) = \frac{1 - T_{2n+2}(x)}{2(1 - x^2)} \tag{B.7}$$

and

$$\sum_{m=0}^{n-1} U_{2m+1}(x) = \frac{x - T_{2n+1}(x)}{2(1 - x^2)}, \tag{B.8}$$

and the identity [1, p.778],

$$U_{k-1}(x) = \frac{1}{1 - x^2}[xT_k(x) - T_{k+1}(x)] \tag{B.9}$$

can then be used to obtain

$$\begin{aligned}\sum_{k=1}^{8} \frac{(9-k)}{4} T_k(x) &= -\frac{9}{8} + \frac{1 + x - T_9(x) - T_{10}(x)}{8(1 - x^2)} - \frac{1}{8} U_8(x) \\ &= -\frac{9}{8} + \frac{1 + x - T_9(x) - T_{10}(x)}{8(1 - x^2)} - \frac{xT_9(x) - T_{10}(x)}{8(1 - x^2)} \\ &= -\frac{9}{8} + \frac{1}{8}\frac{1 - T_9(x)}{1 - x}. \end{aligned} \tag{B.10}$$

Finally, using equations (*B*.2), (*B*.3) and (*B*.10), the equation for the power spectrum of the code reduces to

$$\begin{aligned} C(f) &= \frac{E_a}{T_n}\left[\frac{9}{8} - \frac{1}{72}\frac{1 - cos(2\pi f T_n)}{1 - cos(2\pi f T_n/9)}\right] \\ &= \frac{E_a}{T_n}\left[\frac{9}{8} - \frac{1}{72}\left[\frac{sin(\pi f T_n)}{sin(\pi f T_n/9)}\right]^2\right] \end{aligned} \tag{B.11}$$