Preface

This document is written for readers who are non-experts in modern algebra and coding theory. It is mainly a "theory" document, but still contains many down-to-earth examples. There are no discussions of detailed error-correction implementations as are found in books on Error Correction, but the tight connection between Galois fields and cyclic codes is hopefully made very clear.

In some coding texts, the review of modern algebra is so brief and dense that comprehension is quite difficult, especially for someone totally unfamiliar with the subject. Conversely, in some math books the discussion of modern algebra is so comprehensive that one is forced to invest in many concepts unnecessary for Galois field applications We have attempted somewhat to bridge this gap.

The wonderfully efficient and dense mathematical notations like $\in$ $\forall$ | $\exists$ $\Leftrightarrow$ iff are generally replaced with words, certain theorems are not proved, and in general a medium-to-low level of mathematical rigor permeates the presentation. On the other hand, lots of "words" are used to reinforce the various concepts, examples are provided, and there is constant (perhaps excessive) repetition to grind in definitions and "facts".

After doing manual examples, it is often shown how algorithms can be automated using very simple Maple programs, Maple being a commercial symbolic computer algebra system. Freeware systems exist (see wiki) and our short Maple programs can easily be translated into other languages which support the constructs used.

The structure of a document like this one involves certain design issues.

On the one hand, if a simple Fact applies to a larger class of objects than we are really interested in, but if proving the fact for that larger class of objects is no harder than for the smaller class, one might as well prove the Fact in its more general application. The reader is then forced to learn somewhat more than he or she needs to know, but this seems fairly harmless. Some Facts are so simple to prove, and perhaps so interesting, that they are included in the forest of Facts, even though they are not directly needed on this particular voyage through the forest.

On the other hand, one might argue that the forest then becomes so cluttered with trees that a voyager loses track of where he or she is going, and which trees are important and which are not. The relative importance of various trees only becomes clear later in the trip when certain applications of the Facts are considered. It is useful to pause from time to time and review the trip up to the current point of rest, and that is done in several places in the document.

In an earlier version of this document, there were no equation numbers but the Facts in each chapter had Fact numbers starting with Fact 1. Although now redundant, some of these Fact numbers have been retained. One might then see Fact 4 (7.35) as a cross-reference. Equation numbers of the form (3.14) are applied to equations, Facts and certain definitions. When a numbered item is quoted later in the document, the equation number is put in italics. Longer proofs end with the letters **QED** so the reader knows where the text flow continues.

Summary

This is a fairly detailed summary. The reader is directed to the Table of Contents for a more concise overview.

Chapter 1 [**Algebra**] is a partial review of Modern Algebra which includes only concepts that will be needed in later sections. The basic subjects here are groups, fields, rings and ideals. The so-called residue class ring is formed as R/I where R is a ring and I is an ideal. In particular, Z/(n) is such a residue class ring where Z are the integers and (n) is the ideal which consists of integers which are multiplies of n. It is shown that this residue class ring is isomorphic to the ring of integers mod n, called Z_n. It is then shown that if n is a prime number p, the rings Z/(n) and Z_n are fields. Many "facts" are accumulated in this chapter, and most of them have analogs in the polynomial world introduced in Chapter 3.

Chapter 2 [GF(p)] provides more information about Z_n with a few examples. It then shows that the fields Z/(p) and Z_p are isomorphic to the Galois Field GF(p). The field operations of GF(p) = Z_p = Z/(p) are here called • and + and we learn how to construct the addition and multiplication tables for GF(p). Various facts about GF(p) are then developed. Some authors refer to Galois Fields simply as finite fields, since that is what they in fact are. The argument of GF(*) denotes the number of elements in the finite field. Two of these elements are always 0 and 1.

Chapter 3 [**Polynomials**] discusses another ring, the ring of polynomials R whose coefficients lie in Z_p= GF(p). In this chapter, the operations of GF(p) are called $\oplus$ and $\otimes$. Basic facts concerning such polynomials are developed in analogy with similar properties of integers presented back in Chapter 1. Within the polynomial world, the notion of an irreducible polynomial is introduced and is seen to be analogous to the notion of a prime number in the integer world. An ideal within the polynomial ring R, called (f(x)), consists of all polynomials which are multiples of f(x). Then, just as in Chapter 1 for integers, here the residue class ring R/(f(x)) is shown to be a field when f(x) is an irreducible polynomial in R.

In **Chapter 4** [**GF(q)**] it is shown that, if irreducible polynomial f(x) is of degree m, then the field R/(f(x)) is in fact isomorphic to Galois Field GF(p^m). Each element of GF(p^m) can be associated with a possible remainder polynomial which is obtainable when a polynomial is divided by f(x) whose leading power x^m has coefficient 1 ("monic"). Since there are m coefficients of f(x) each lying in GF(p), there are then p^m such remainder polynomials and this matches the number of element sin GF(p^m). Each remainder polynomial, and thus each element of GF(q), can be represented as an m-tuple of Z_p elements which are the remainder polynomial coefficients.

Since the only finite fields that exist are these GF(p^m) where p is a prime number and m a positive integer, we have at this point a "model" (realization, representation) for *all* the Galois Fields. A method for determining the + and • tables for any GF(q = p^m) is then developed based on this model.

There follows a lengthy discussion of the notion of cyclic groups with respect to the GF(q) fields, and after much work it is shown that the non-zero elements of every Galois Field GF(q) form a cyclic group with respect to the • operator. This means that it is possible to find some element in GF(q) (a generator) whose powers enumerate all non-zero elements of the field, an extremely useful fact. This is the "power basis" and serves as a second method of labeling elements of GF(q), the first being the m-tuple basis mentioned above. Each basis leads to different forms of the + and • tables for GF(q).

It is then shown that the polynomial $x^q - x$ can be written as a product of q factors of the form $(x-a_i)$ where the a_i are the q elements of GF(q):

$$(x^q - x) = (x - a_1)\bullet(x - a_2)\bullet(x - a_3)\bullet(x - a_4)\bullet.....(x - a_q) \ .$$

In other words, $x^q - x$ can be fully factored in GF(q). Another way to say this is that the q elements of GF(q) are all roots of the polynomial $x^q - x$. This implies that $\alpha^q = \alpha$ for any α in GF(q).

The chapter then closes with a few facts about GF(q) similar to those presented at the end of Chapter 2 for GF(p). It is shown that GF(p) is a subfield of GF(p^m) and both these fields have the same 0 and 1 elements. In some ways, this is similar to the real numbers being a subfield of the complex numbers, those two fields of course being infinite fields and thus not Galois fields.

Chapter 5 [Minimum and Primitive Polynomials] broaches the topic of the minimum polynomial m(x) of an element α of GF(q). Such a polynomial is simply a portion of the above displayed product of factors $(x-a_i)$ which includes $(x-\alpha)$ and includes the smallest set of other $(x-a_i)$ factors which, when multiplied out, results in m(x) having coefficients all lying in Z_p = GF(p). In general, some *arbitrary* product of the $(x-a_i)$ factors will form a polynomial with coefficients in GF(q) which contains GF(p). Since the factor $(x-\alpha)$ is included, $m(\alpha) = 0$. It is shown that any minimum polynomial is irreducible in GF(p). It turns out that a given minimum polynomial m(x) is the minimum polynomial of all the GF(q) elements a_i that appear in those other $(x-a_i)$ factors which make up its portion of the fully factored $x^q - x$. The set of GF(q) elements for which some m(x) is the minimum polynomial is called a conjugate set. If element α of GF(q) is a primitive element of GF(q), meaning its powers can enumerate all the non-zero elements of GF(q) as noted above, then the minimum polynomial of α is called a primitive polynomial of GF(q). We determine exactly how many primitive polynomials GF(q) has. The question of how minimum and primitive polynomials are determined is then discussed with various examples. A primitive polynomial f(x) of GF($q=p^m$) is always of degree m, and can therefore serve as the f(x) in the residue class ring R / (f(x)) which represents GF(q). It is shown that the conjugate sets partition the elements of GF(q), and this is then related to the subject of cyclotomic cosets. The final subsection discusses certain products of minimum polynomials which will appear later in the theory of BCH codes.

Chapter 6 [GF(q) Enumeration Table] shows how one determines the "enumeration table" of any Galois Field GF($q=p^m$). This table is based on a selected primitive element α and its corresponding primitive polynomial m(x). Since $m(\alpha) = 0$, this equation gives a way to express α^m as a sum of lower powers of α times coefficients. One first enumerates all the non-zero elements of GF(q) as powers of primitive element α up to power α^{q-2}, and then one uses this α^m reduction equation to express the higher powers in terms of lesser powers, and the result is the "table" for GF(q). The table is useful because, once formed, one can immediately obtain the addition table for the field GF(q). The multiplication table in this "powers basis" is completely trivial.

In section (c) a very simple Maple program is used to directly construct enumeration tables for several GF(q) fields. Given the enumeration table for a field, section (d) shows how one can then expand the factored minimum polynomials of Chapter 5 to verify that they do indeed have coefficients in GF(p). Along the way we show how Maple can be used to accomplish various tasks like factoring a polynomial in GF(q), finding roots, and multiplying and dividing polynomials in the ring of polynomials R whose coefficients lie in Z_p= GF(p). The final section (e) provides an example of how the + and • tables for GF(2^2) are computed using the GF(2^2) enumeration table.

With **Chapter 7 [Block Codes]**, there is a rather sudden shift in topic. This chapter provides the basic facts of the linear block codes (n,k) which are used in forward-error-correction systems. In each block of such a code, k data symbols are combined with n-k parity check symbols to form a code word of n

symbols (symbols could be bits or bytes or something else). The codes are linear because the n code word symbols in a block, treated as a vector **c**, are generated by the application of a generator matrix G to the vector **d** of data symbols, **c** = G**d**. When a transmitted code word **c** is received at the end of a "transmission", one or more of the symbols might have been damaged by the effect of "noise". The damaged code word **c'** will normally not be in the allowed "code book" of legal code vectors **c**, and then the parity check symbols can possibly be used to correct **c'** back to **c**. A certain parity check matrix H which is related to the generator matrix G is used to check incoming code words for errors. If H**c'** = **0**, then **c'** is a legal code word. If H**c'** = **s** ≠ 0, there has been some kind of error. The vector **s** is called the syndrome, and it can be used in various schemes to correct the error. Various drawings are used to illustrate how code words appear as points in an embedding vector space V^n which contains many non-code words. Each code word is surrounded by a sphere of protection of radius d, the Hamming distance of the code. If a received bad code word lies in this sphere, it is corrected to the legal code word at the center of the sphere. The chapter concludes with a brief history of coding theory including mention of non-block codes.

In **Chapter 8 [Cyclic Codes]** we have a mighty confluence of the flowing river of Chapters 1-6 with the tributary of Chapter 7. The signpost overlooking this confluence reads "Galois Cyclic Codes". It is shown how the data and code words of the block codes of Section 7 appear as coefficients of polynomials of the type described in Chapter 3, but now the coefficients are in general elements of GF(q=p^m) instead of GF(p). The ring of such polynomials is called R_q. The full power of the theory of Galois Fields is brought to bear in the theory of cyclic codes; almost every concept of all earlier chapters makes an appearance. The codes are cyclic because a rotation of any code word's symbols by any number of places generates a new code word, but this fact lays hidden in the background until section (f).

In the cyclic code world, the generator block-code matrix G is replaced by a generator polynomial g(x) of degree n-k, and the formal encoding process is c(x) = g(x)d(x) where the data and code words are encoded into d(x) and c(x). In this so-called cyclic basis, the parity check symbols are convoluted into the c(x) coefficients. An equivalent basis called the systematic basis encodes a little differently and has implementation benefits since the data symbols are exposed in the code words. Section (c) discussed how encoders and decoders are implemented, but only at a very high level. Section (d) shows how CRC works in terms of a cyclic code which in fact is not really cyclic. Section (e) ferrets out the fact that the two bases just mentioned are rearrangements of each other. Then section (f) shows that the cyclic codes as defined in this chapter are in fact cyclic.

At this point, the dormant residue-class-ring machinery introduced in Chapter 1 and used in Chapter 3 is powered up again, this time in double overdrive. First, a nameless ring A_n is defined as $R_q/(x^n-1)$ and its elements are associated with remainder polynomials of degree less than n. This ring A_n contains the q^k code word polynomials c(x) along with the q^n-q^k "illegal" code word polynomials, all mixed together. In a second residue-class-ring application, the elements of A_n are used in $A_n/(g(x))$ to form what is called the standard array. In this array, all the legal code words are rounded up into one bin which is the ideal (g(x)). This is all the content of section (g). Then in section (h) it is shown how this standard array "does" error correction. It is a rather amazing logical thread.

Section (i) then shows the form of the parity check matrix H for a cyclic code, and gives a top level picture of the code families associated with the names BCH, Hamming and Reed-Solomon. The first and last code families can be designed to correct an arbitrary number of symbol errors in a code word and are very efficient at doing it. Finally, section (j) explains why one might want the coefficients of the generator polynomial g(x) to lie in G(p) rather than G(q). The reason is that this makes the design of hardware

polynomial multipliers and dividers extremely simple. This desire to have g(x) have coefficients in GF(p) is a major driver for all the work of Chapter 5 on minimal polynomials, as is seen in Chapter 9

Chapter 9 [Small Code Survey] states the BCH Bound Theorem and defines the BCH cyclic codes as those which optimize this theorem. The theorem provides a *floor* for distance d and therefore error-correcting ability t. The order of a BCH code with respect to GF(q) is always n = q-1 = p^m-1, whereas the k value of the (n,k) code designation is an integer forced by the code. The so-called narrow-sense BCH codes are a special case which have the coefficients of g(x) in GF(p). The generators $g_N(x)$ of such codes are the "least common multiples" of N of the minimum polynomials described in Chapter 5. An example with GF(2^5) is worked out in some detail, providing several BCH codes with code length n = 31. A simple subset of the narrow-sense BCH codes have g(x) = a single minimum polynomial, and from this subset various Hamming codes are derived, though historically the Hamming codes were known before the BCH codes revealed their framework. Finally it is shown how the Reed-Solomon codes have g(x) coefficients in GF(q) rather than GF(p) and, despite increased implementation costs, are able to correct the most possible errors that an (n,k) code can correct ; they are maximum-distance separable. The final section (e) comments on how Galois logic uses the algebra of GF(q), whereas digital filters use that of Z_q.

Chapter 10 [Matrix Representation of GF(q)] is another change of topic. It is shown how one can easily construct a representation of any GF(q=p^m) field as a set of q mxm matrices whose elements lie in GF(p). Two critical ingredients are the Cayley-Hamilton Theorem and the notion of a Companion Matrix. A very simple Maple program is then presented which generates a set of q matrices which represent any GF(q) Galois field.

Appendix A gives a proof of a certain claim (4.29) made in Chapter 4 upon which is based the derivation of the important fact that all Galois fields are cyclic.

Appendix B explains the basic nature of the conjugate set of α as encountered in Chapter 5.

Appendix C evaluates the polynomial product a(x)b(x).

Appendix D is a brief matrix review in support of Chapter 10.

A few **References** are then provided.