III.2 Polynomial Codes

III.2.1 Single-Error-Correcting Polynomial Codes. If we only wanted to correct single errors, the matrix codes already discussed would be fairly satisfactory. Though vastly more convenient than utterly unstructured codes, they still have some defects: they require specification of at least a K matrix to define them, and require multiplying the message or received word by a matrix in order to code or decode. Can we do better?

We can. So far we have treated our messages and code words as vectors, and have gotten mileage out of our ability to add them. We shall now interpret them as polynomials. Polynomials can be added (and we will do so mod 2 again) and they can be multiplied as well. This additional structure will simplify encoding and decoding, will allow us to form codes that correct several errors, and allow us to correct them with relative ease.

Our message is again assumed to be a sequence of m binary bits. We now interpret it as a polynomial of degree $m-1$, whose jth bit, for each j, is the coefficient of x^{j-1} in the polynomial. (Notice, that the jth bit is the coefficient of x^{j-1} and not x^j. This is an easy thing to get confused and can be the source of many errors.)

Thus 1011 becomes the polynomial: $1 + x^2 + x^3$.

We multiply polynomials by the ordinary rules of multiplication; we still use binary addition, so that $2 = 0$ holds, but multiplication will be what it always has been.

For example, 101 + 110 gives

$$(1 + x^2) + (1 + x) = x + x^2 \text{ or } 011,$$

while 1011 × 11 gives

$$(1 + x^2 + x^3) \times (1 + x) = 1 + x + x^2 + x^4 \text{ or } 11101.$$

Given some particular polynomial $m(x)$ (whose bits form the plaintext message), the sender encodes $m(x)$ by multiplying it by some fixed polynomial $P(x)$, obtaining thereby the polynomial $s(x)$ whose bits are sent across the communication channel. These bits are received at the other end of the channel (possibly with some errors) and are put together to form the polynomial $r(x)$. In equation form:

$$\begin{aligned} s(x) &= m(x)P(x) \\ r(x) &= s(x) + e(x) \end{aligned}$$

where $e(x)$ is the "noise", expressed in polynomial form. Combining:

$$r(x) = m(x)P(x) + e(x). \tag{1}$$

Note that the transformation that turns $m(x)$ into $s(x)$ is indeed a linear transformation; this is an easy consequence of the fact that $(m(x)+m'(x))P(x) = m(x)P(x)+m'(x)P(x)$. It follows that our code is a linear code, and that its behavior is determined once we know how to encode a basis for the space. In this case, the basis is the set of monomials $1, x, x^2, x^3, \ldots$; in terms of binary sequences, these are the weight-1 code words $1, 01, 001, 0001, \ldots$. To encode such a monomial x^k, we simply multiply it by $P(x)$, which amounts to increasing all the exponents in $P(x)$ by k. Putting it differently, the encoding of the weight-1 code word associated with the monomial x^k is just the code word associated with the polynomial $P(x)$, with k 0's put at the front of it.

Let us illustrate this with an example. With 4 message bits (the message polynomials of degree ≤ 3) and using as encoding polynomial $P(x) = 1 + x^2 + x^3$, we obtain code polynomials of degree 6, or seven-bit code words. The encoding matrix for this particular polynomial has rows given by the code words of the weight-1 message polynomials $1, x, x^2$, and x^3, namely $P(x), xP(x), x^2P(x)$, and $x^3P(x)$:

power of x	0	1	2	3	4	5	6
matrix:	1	0	1	1	0	0	0
	0	1	0	1	1	0	0
	0	0	1	0	1	1	0
	0	0	0	1	0	1	1

This is the typical matrix form of a polynomial code: each row is a cyclic shift (one step to the right) of the previous row, because the basis vector for the lower row is x times that for the previous row, and the result of multiplying by x is to raise all exponents in the result by one. Thus we can uniquely specify the matrix $P(x)$, and hence the entire code, merely by giving the first row.

Let us look at the structure of code words in a polynomial code which detects *and* corrects single errors. As we have observed, the code words in a polynomial code are all multiples of the encoding polynomial $P(x)$; i.e. they have $P(x)$ as a factor. If we receive a code polynomial with a single error, therefore, any non-zero remainder when we divide the received polynomial by $P(x)$ *must come from the error*. That is, equation (1) tells us that $r(x)$ and $e(x)$ must leave the same remainder when divided by $P(x)$. Since we are assuming only a single error, the error must be in only one bit position. Therefore $e(x)$ will be a *monomial*, i.e.

$$e(x) = x^i$$

for some i, where i is the bit position in which $r(x)$ is in error.

So in order to be able to correct the error in $r(x)$, we need to *find the i such that there exists some $m(x)$ such that $r(x) = m(x)P(x) + x^i$*. This is easy enough to do:

we simply divide the received word $r(x)$ by $P(x)$, and compare the remainder to the remainder we get when we divide each possible monomial by $P(x)$. That is, if

$$r(x) = m(x)P(x) + e(x)$$

then, dividing through by $P(x)$ and examining the remainder, we find that

$$\text{remainder}(r(x)) = \text{remainder}(e(x)) = \text{remainder}(x^i).$$

Here remainder$(\cdot)$ denotes the remainder that results from dividing a polynomial by $P(x)$; it is always a polynomial of degree strictly less than $P(x)$.

We can use long division to compute the message from the code word, once errors have been corrected (which is easier here than in grade school since we have $0 = 2$).

Let us, as an example, do a single-error correction with the coding polynomial $P(x) = 1 + x^2 + x^3$. It is helpful here to keep handy a table of the remainders of all the monomials upon dividing by our polynomial. We compute the following table:

monomial	remainder mod $1 + x^2 + x^3$
1	1
x	x
x^2	x^2
x^3	$1 + x^2$
x^4	$1 + x + x^2$
x^5	$1 + x$
x^6	$x + x^2$

We can get each successive row of this table by multiplying its predecessor by x and substituting $x^3 \equiv 1 + x^2$ when x^3 appears in the result. Thus, starting from $x^3 \equiv 1 + x^2$ and multiplying by x yields $x^4 \equiv x + x^3 \equiv x + 1 + x^2$. Continuing:

$$\begin{array}{lll} x^4 \equiv 1 + x + x^2 & \Rightarrow & x^5 \equiv x + x^2 + x^3 \equiv x + x^2 + 1 + x^2 \equiv 1 + x \\ x^5 \equiv 1 + x & \Rightarrow & x^6 \equiv x + x^2 \\ x^6 \equiv x + x^2 & \Rightarrow & x^7 \equiv x^2 + x^3 \equiv x^2 + 1 + x^2 \equiv 1. \end{array}$$

Of course, the table could be generated by simply dividing and finding each remainder. The reader should verify the entries in this table by doing several of the divisions.

Suppose, for example, that the word 1101101 is received. We add up the remainders of the powers present in the word according to our monomial remainder table, or by dividing $r(x)$ by $1 + x^2 + x^3$ and looking at the remainder. Either way, we get $1 + x + x^2$, which tells us (again by the table) that the error is in the 4th power, or

5th digit, so that the real code word is 1101001 (assuming that only one error was made). By long division, we get:

$$\begin{array}{cccc|ccccccc}
 & & & & & & & 1 & 1 & 1 & 1 \\ \cline{5-11}
1 & 0 & 1 & 1 & 1 & 1 & 0 & 1 & 0 & 0 & 1 \\
 & & & & 1 & 0 & 1 & 1 & & & \\ \cline{5-8}
 & & & & & 1 & 1 & 0 & 0 & & \\
 & & & & & 1 & 0 & 1 & 1 & & \\ \cline{6-9}
 & & & & & & 1 & 1 & 1 & 0 & \\
 & & & & & & 1 & 0 & 1 & 1 & \\ \cline{7-10}
 & & & & & & & 1 & 0 & 1 & 1
\end{array}$$

which tells us the message was 1111.

Note that the above calculation is really a short-hand for the division

$$\begin{array}{cccc|ccccccc}
 & & & & & & & 1 & +1x & +1x^2 & +1x^3 \\ \cline{5-11}
1 & +0x & +1x^2 & +1x^3 & 1 & +1x & +0x^2 & +1x^3 & +0x^4 & +0x^5 & +1x^6 \\
 & & & & 1 & +0x & +1x^2 & +1x^3 & & & \\ \cline{5-8}
 & & & & & 1x & +1x^2 & +0x^3 & +0x^4 & & \\
 & & & & & 1x & +0x^2 & +1x^3 & +1x^4 & & \\ \cline{6-9}
 & & & & & & 1x^2 & +1x^3 & +1x^4 & +0x^5 & \\
 & & & & & & 1x^2 & +0x^3 & +1x^4 & +1x^5 & \\ \cline{7-10}
 & & & & & & & 1x^3 & +0x^4 & +1x^5 & +1x^6
\end{array}$$

Normally we do division of polynomials the other way around (that is, putting the terms in order of *decreasing* exponents), but this way works too, provided that we know ahead of time that the division will come out exact, i.e. with no remainder. If on the other hand we need to perform division with remainder, we must either revert to the customary form of long division of polynomials (in which higher-order terms appear first) or else use a remainder table, as described above.

We have been assuming so far that our code can correct single errors, but when will this really be the case? Clearly, by the above discussion, we will be able to correct single errors if and only if *each monomial which can appear in the code has a* unique *remainder* when divided by $P(x)$. Clearly, if we look at all possible remainders (just by the pigeonhole principle!) for x, x^2, x^3, ... letting the exponents get arbitrarily large, at some point we will have two different monomials with the same remainder. This is why our error-correcting codes can't be arbitrarily long, and encode arbitrarily long messages, for some fixed polynomial. For example, we only looked at 7-bit code words (4-bit messages) in the above example. Could we have handled 8-bit code words and still been able to correct a single error? No, because if we look at the remainder of x^7 when we divide by $P(x) = 1 + x^2 + x^3$, we get 1, which is the remainder of

$x^0 = 1$ as well. (If we divided $r(x)$ by $P(x)$ and got a remainder of 1, we wouldn't know if the error was in the first or the last bit.)

In general, if the polynomial $P(x)$ has a multiple $m(x)P(x)$ of the form $1 + x^j$ (where j is small enough so that this is a legal code word), then $1 + x^j$ is in fact a weight-2 code word, and the code cannot then be single-error-correcting. The length of the longest single-error-correcting code obtainable from a given polynomial P is given by the smallest value of j for which $1 + x^j$ is a multiple of $P(x)$. (Check that in the example above, $j = 7$.)

III.2.2 Constructing Perfect Polynomial Codes. A set of objects (numbers, polynomials, whatever) on which addition and multiplication are defined, so that they commute and associate and distribute the way ordinary numbers do, and there are both additive and multiplicative inverses the way rational numbers have them, is called a *field*. The simplest example of a field are the elements 0 and 1 with usual binary arithmetic. Other common examples of fields are the rational numbers, the real numbers, and the complex numbers, but we will be more concerned in this chapter with the 2-element field. In particular, we will be studying polynomials whose coefficients belong to the 2-element field.

A polynomial that cannot be factored into two polynomials each of smaller degree is said to be *prime*. A polynomial $p(x)$ of degree k (with coefficients in a field F) which has the property that the remainders of monomials upon dividing by $p(x)$ include every non-zero polynomial of degree up to $k-1$ is said to be *primitive*. It is not hard to show that in order to be primitive, a polynomial must be prime.

To forestall confusion, you should take notice of the fact that the primality or lack thereof of a polynomial depends on the field in which its coefficients live. Thus, $1 + x^2$, viewed as a polynomial with *rational* coefficients, is prime; but $1 + x^2$, viewed as a polynomial with coefficients in the 2-element field, factors as $(1 + x)(1 + x)$.

With F the field whose elements are 0 and 1, there are $2^k - 1$ non-zero polynomials of degree up to $k - 1$. If a primitive polynomial is used to generate a code having $2^k - 1$ bits, then every possible non-zero remainder will occur as the remainder of some monomial. This means that any word of length $2^k - 1$ produces the same remainder as some monomial; it is therefore within distance 1 of a code word, which means that the code is perfect. Notice we have just showed (yet again) that we can't do any better than code words of length $2^k - 1$!

We have reduced the problem of finding a perfect code to the problem of finding a primitive kth degree polynomial.

Consider the various low-degree polynomials. The only polynomials of *degree 1* are x and $1 + x$, and multiplying a message by x merely shifts the message by one location (or equivalently, sticks a 0 in front of it). The polynomial $1 + x$ is a factor of any polynomial with an even number of non-vanishing terms. (A polynomial with an even number of terms vanishes mod 2 for $x = 1$; therefore 1 is a root of the polynomial, and $x - 1$, which is the same as $1 + x$, is a factor of the polynomial. See section III.4.) The one-check-bit code whose polynomial is $1+x$ separates code words so that their minimum distance is 2, but cannot separate them enough to correct errors. It generates a *parity check* code: if there is one error, $r(x)$ will have an odd number of terms and this will be noticed.

A polynomial of *degree 2* that is prime must have a quadratic term to be of degree 2, have a constant term to avoid divisibility by x and have an odd number of terms to avoid divisibility by $1 + x$. There is only one such polynomial: $1 + x + x^2$. Its remainder table is

$$\text{rem}(1) = 1,\ \text{rem}(x) = x,\ \text{rem}(x^2) = 1 + x,\ \text{rem}(x^3) = 1.$$

This is a single-error-correcting code only if there are at most 3 bits in the code word. This allows only one message bit, since there must be $k = 2$ check bits. The code corresponding to this polynomial encodes the message 0 as 000 and 1 as 111.

The polynomials of *degree 3* that are prime are, as can be deduced by similar reasoning, $1 + x + x^3$ and $1 + x^2 + x^3$. Each of these polynomials generates a single-error-correcting code having a total of 7 bits, encoding $7 - 3 = 4$ message bits, as we have already seen.

Prime polynomials of *degree 4* are those polynomials of degree 4 having a constant term and an odd number of terms, other than the polynomial $(1 + x + x^2)^2$ which is $(1 + x^2 + x^4)$. There are three of them. One of them $(1 + x + x^2 + x^3 + x^4)$ is not primitive. The other two $(1 + x + x^4$ and $1 + x^3 + x^4)$ generate perfect codes having $2^4 - 1 = 15$ bits.

It is quite easy though tedious to find prime and primitive polynomials of degree 5,6,... by the simple procedure we have just used for degree 4: write down all polynomials of appropriate degree having a constant term and an odd number of terms; remove any divisible by lower degree polynomials; and then form the remainder tables of the polynomials left, to see if they are primitive.

III.2.3 More on polynomial arithmetic. If we do polynomial arithmetic mod $p(x)$, we in effect impose the condition $p(x) \equiv 0$. A question that will be of interest to us soon is: if $p(x) = 0$, what kind of equation will be obeyed by higher powers of x, such as $x^2, x^3, \ldots$? What polynomials $q(x)$ satisfy $q(x^3) = 0$, for example?

Fact: Since we have $2 = 0$ in the 2-element field, x^2 will satisfy all the equations that x does.

Proof. If $p(x) = 0$, then $[p(x)]^2 = 0$; but $[p(x)]^2 = 0$ consists of two types of terms: terms that are squares of terms in $p(x)$, and cross terms. Since the cross terms have coefficients of 2's, and $2 = 0$, we have no cross terms. Also the square terms are the terms in $p(x)$ with all the exponents doubled; thus we have $[p(x)]^2 \equiv p(x^2) \equiv 0 \bmod 2$ which has the interpretation that x and x^2 obey the same equations. □

We can find an equation which x^3 obeys (when $p(x) = 0$) by forming the powers x^{3j}, for j up to the degree of $p(x)$, computing their remainders, and then eliminating all other powers in those remainders; this leaves an equation obeyed by x^3. (We illustrate this below.) A similar procedure can be used to find the equation obeyed by any other power of x.

Example. Suppose $p(x) = 1 + x + x^4$. The remainder table is:

Power	Remainder mod $1 + x + x^4$
0	1
1	x
2	x^2
3	x^3
4	$1 + x$
5	$x + x^2$
6	$x^2 + x^3$
7	$1 + x + x^3$
8	$1 + x^2$
9	$x + x^3$
10	$1 + x + x^2$
11	$x + x^2 + x^3$
12	$1 + x + x^2 + x^3$
13	$1 + x^2 + x^3$
14	$1 + x^3$
15	1

We therefore have: $\text{rem}(x^6) = x^2+x^3$, $\text{rem}(x^9) = x+x^3$, $\text{rem}(x^{12}) = 1+x+x^2+x^3$ from which we can eliminate x and x^2 to get $\text{rem}(x^6+x^9+x^{12}) = 1+x^3$, which means $\text{rem}(1 + x^3 + x^6 + x^9 + x^{12}) = 0$ so that y with $y = x^3$ obeys the equation

$$1 + y + y^2 + y^3 + y^4 = 0.$$

For another example, consider the problem of finding an equation that x^5 satisfies. Proceeding as above, we get $\text{rem}(x^5) = x + x^2$ and $\text{rem}(x^{10}) = 1 + x + x^2$, so that $1 + x^5 + x^{10}$ has 0 remainder. This can be rephrased as the statement: $z = x^5$ obeys the equation $1 + z + z^2 = 0$.

We have seen that x^2 always obeys the same equation as x; by the same reasoning, x^4 and x^8 and x^{16} do likewise; in our example $x^{16} = x$ again so we may stop there. Similarly x^3, x^6, x^{12} and $x^{24} = x^9$ all obey the same equation, and so do x^5 and x^{10}. In general, we also have that x^{-j} (in our example x^{-j} is x^{15-j}) obeys the equation obtained by reading the equation for x^j backwards (i.e., reversing the order of all the coefficients).

We therefore have that of the powers of x: the first, second, fourth, and eighth satisfy $1 + y + y^4 = 0$, and further one can show:

power:	equation satisfied:
1,2,4,8	$1 + y + y^4 = 0$
3,6,9,12	$1 + y + y^2 + y^3 + y^4 = 0$
5,10	$1 + y + y^2 = 0$
7,11,13,14	$1 + y^3 + y^4 = 0$
0	$1 + y = 0$

It is straightforward to compute a similar table for remainders with respect to any primitive polynomial. We call the table of remainders the *equation table* for our primitive polynomial $p(x)$.

III.2.4 Further remarks. We have seen that any binary polynomial $P(x)$ of degree k defines a code with $N - m = k$ by the rule $s(x) = m(x)P(x)$. If the polynomial is primitive, its code will be perfect single-error-correcting for $N = 2^k - 1$ ($m = 2^k - 1 - k$). The code words will be all N-bit strings that correspond to polynomials divisible by $P(x)$.

We have also noted that it is easy to determine the error if only one error is made. The error is then a monomial, and by comparing the remainder of the received polynomial $r(x)$ divided by $P(x)$ with the remainders of the various mononomials we can observe which monomial it is.

Since code words are divisible by $P(x)$, they contribute nothing to remainders. Looking at remainders is therefore a way of focusing our attention on the errors. The following two fundamental facts about remainders are worth noting, for they are what make everything work:

$$\begin{aligned}\mathrm{rem}(a(x) + b(x)) &= \mathrm{rem}(a(x)) + \mathrm{rem}(b(x))\\ \mathrm{rem}(a(x)b(x)) &= \mathrm{rem}(\mathrm{rem}(a(x))\mathrm{rem}(b(x)))\end{aligned}$$