

Supplementary Notes on BCH Codes.

We suppose we want to be able to send messages that can be conveniently decoded even in the presence of errors.

First we considered the situation in which we want to be able to correct an error in a single bit.

We want to correct errors by a uniform procedure regardless of what the original message before the error appeared is. To do so, we need to do something to the received message (that has the error) that eliminates the original message. We can then concentrate on the error, fix it, and then, once we have eliminated the error, we can decode the corrected message by inverting the encoding operation.

1. With a matrix code:

In a matrix code in which coding obeys the formula: $c(m) = mE$ where E is the encoding matrix, we can do this by finding an appropriate matrix D obeying $ED = \text{a zero matrix}$. Then if the received message, r , contains one error, it has the form $r = mE + \langle j \rangle$ where $\langle j \rangle$ is a standard basis vector having a '1' in the j -th position, for some as yet unknown j . If we form rD we get $mED + \langle j \rangle D$ which is, by definition, the j -th row of D ,

Once we know $\langle j \rangle D$, we can check each row of D , locate which one is in error, and change the corresponding bit in r ,

2. With a polynomial code using a primitive polynomial

With such a code, defined by polynomial $p(x)$, the encoding rule is $cm(x) = m(x)p(x)$, and every code word has no remainder upon dividing by $p(x)$.

We can decode by computing a "power remainder table" that gives the remainder of each of the powers of x , upon dividing by $p(x)$ up to some appropriate power.

If there is only one error, that error, call it x^e , will come entirely from the (single monomial) error. We will have $r(x) = m(x)p(x) + x^e$. And the remainder of $r(x)$ will be the remainder of x^e . We can compare it to the remainders of powers in the remainder table. The remainder it agrees with will correspond to the power of x that is in error.

This procedure seems quite different from the matrix procedure; instead of multiplying the received message by a decoding matrix, we want to divide its polynomial by $c(x)$ and look at the remainder.

But at the point that we have not corrected errors, we are not interested in the quotient of the division, but only in the remainder. And we can get the remainder of $r(x)$ by adding up the remainders of its 1-bits. (since the remainder of a sum is the sum of remainders).

Furthermore, this sum of reminders of 1-bits consists of the vector whose entries are the dot products of $r(x)$ considered as a vector, with the (remainder) columns of the remainder table. And this is exactly the same as taking the matrix product of the received word with a D matrix that consists of the remainder columns of the power-remainder table.

In other words, the two procedures are really the same thing, and the D matrix for a polynomial single error correcting code is its power-remainder table.

3 Correcting more errors: The plan of BCH codes.

When we want to correct more errors, taking the remainder of $r(x)$ on dividing by $p(x)$ will give us the remainder of the sum of the monomials of the various errors. To locate them we need to be able to extract more information about the errors than merely this remainder of their sum.

Our plan will be to use an encoding polynomial that is the product of $p(x)$ with some other polynomial which latter is chosen to give us additional information which will allow us to deduce exactly what the errors are.

But what polynomial should we multiply p by?

To understand the answer we first ask: how do we hope to decode? What exactly is the information we need to be able to do so?

Well, we hope to mimic the process used in the single error case. In that case we found the remainder of r on dividing by p , and with it we can check each power x^j to see if it has the same remainder. (Just like the prince did with cinderella's slipper.) In general, we want to check each power x^j against something, and have that check tell us if x^j is any one of the error monomials.

We can phrase our single error procedure as solving the equation $\text{rem}(y) = \text{rem}(r)$ among monomials y by checking each one of them, which is the same as checking the equation $\text{rem}(y - \text{rem}(r)) = 0$.

With two errors we want to check a similar equation, but one which gives the location of both of the actual error powers. Suppose the error is $x^{e1} + x^{e2}$. We want to find a polynomial whose remainder vanishes at the monomials x^{e1} and x^{e2} and for no other monomials.

What we want then is a magical polynomial, called the “error locator polynomial” which has the wonderful property that it gives 0 remainder for every monomial that is in error, and gives 0 remainder for no others. If we have it, we can check each monomial in it and change those for which it gives 0 remainder.

And what is this polynomial? It is obviously $(y - x^{e1})(y - x^{e2})$. (And you can write a similar “error locator polynomial” for any number of errors).

(It is obvious that this polynomial gives a 0 remainder for each error monomial, because when we multiply the remainders of the factors one remainder factor will be the all 0 remainder. We will have to, and will, prove that there no other monomial solutions to this is equation, except these two.)

So our task is to find this error locator polynomial. And how can we find it? To determine a polynomial, it is necessary and sufficient to determine its coefficients. Here we may write the polynomial out, using the distributive law, and find it becomes

$$y^2 + (x^{e1} + x^{e2})y + x^{e1+e2}.$$

In the two error case we are considering you can see that the first power coefficient has remainder that is that of r itself. So all we need to do to be able to decode is to find a factor to throw into the encoding polynomial that will allow us to determine the other coefficient, whose remainder on dividing by p will be the that of the monomial that is the product of the two error monomials.

The various terms that occur in the error locator polynomial are given names; they are called the “**elementary symmetric functions**” of the error monomials.

The **k-th elementary symmetric function** consists of the **sum of the products of k distinct monomials, in all possible ways**.

Thus above, there are two terms for $k=1$ and 1 for $k=2$, because there are, we suppose, only two errors. There is no $k=3$ term because there is no product of all three errors. The $k=0$ term is always by definition, 1. We

will denote the k -th elementary symmetric function as $s(k)(x)$ and its remainder as $\text{rem } s_k(x)$ or s_k .

So how can we find the elementary symmetric functions we need to get the coefficients of the error locator polynomial?

Fortunately, there is a second set of symmetric functions of the errors that we can easily get our hands on, (with the right encoding polynomial) and these allow us to determine the elementary symmetric functions.

These are called the **power sum symmetric functions** of the error monomials and the k -th one of these is the sum of the k -th powers of the error monomials.

In the two error correcting case, the one we want is the third power sum, and it is $x^{3e_1} + x^{3e_2}$. We can deduce the remainder of $x^{e_1+r_2}$ from the remainder of this power sum..

In the three error correcting case we want the fifth power sum remainder as well, for 4 error correcting we need the 7th power sum remainder as well, and so on.

We will denote the k th power sum symmetric function as $t(k)(x)$ and its remainder as $\text{rem } t_k(x)$ or t_k .

This leaves us with the following questions:

1. how do we get the elementary symmetric function remainders that we need to find the error locator polynomial from the relevant power sum symmetric function remainders?
2. what factors do we need in the encoding polynomial to be able to deduce the remainders of the relevant power sums of the errors?
3. how do we find the relevant power sum remainders of the errors from the received message?

The answer to question 2 is straightforward. To be able to extract the remainder of the sum of the third powers of the errors, we need a factor in the encoding polynomial that ensures that the remainder (upon dividing by our original polynomial p) of the sum of the third powers of the encoded message monomials always sums to 0. Thus want a factor in the encoding polynomial $q(x)$ such that $\text{rem } q(x^3) = (\text{the all-0 remainder})$, when we divide by $p(x)$.

This will assure that when we take the sums of the third powers of the monomials in our received message, we get contributions only from the errors. Every code word monomial will have a factor in it that has a zero remainder on dividing by $p(x)$ and will therefore itself have a zero remainder on dividing by $p(x)$.

Which leads us to the question: how do we find such factors? And the answer, as we shall see, is that they can be found by row reduction.

And similar remarks hold for fifth seventh and higher odd powers as well.

The answer to question 3 is also straightforward: we can construct a third power remainder table whose entries are the remainder for power z is the remainder of x^{3z} instead of the remainder of x^z as appears in the regular remainder table. Then taking the dot product of $r(x)$ with the columns of this third power remainder table gives us the remainder of the sum of the third powers of the errors.

And the same thing happens for the sum of fifth or higher powers of the error monomials. If $\text{rem } q(x^k) = 0$ then the dot product of $r(x)$ with the remainder columns of a k -th power remainder table will produce the remainder of the sum of the k -th power of the error monomials.

You will see that all of the steps mentioned so far are easily carried out on a spreadsheet.

The first question above also has an elegant answer. There are wonderful relations between the two kinds of symmetric function, which allow us to find elementary symmetric functions from power sums, by solving some simultaneous equations, as we shall see.

To recapitulate what we have said, we have the following tasks, each of which we will learn to perform..

1. Find a suitable primitive polynomial, $p(x)$, of degree k .
2. Multiply a message by an encoding polynomial
3. Create a remainder table.
4. Create a higher power remainder table
5. Find polynomials $q(x)$ such that for appropriate k we have $\text{rem } q(x^k) = 0$.
6. Find the dot products of a received message with the columns of a remainder table.

7. Produce elementary symmetric functions of the error from such dot products, (which give you the error locator polynomial.)
8. Check each monomial to see if it obeys the “error locator equation”. which means that the error locator polynomial evaluated at it gives the 0 polynomial.
9. Divide the corrected message by the encoding polynomial.

We will go over each of these steps and see how to implement it on a spreadsheet, and you will do so so as to be able to encode and decode correcting three errors. At this point every single step mentioned above should appear vague and menacing to you. When you get them to work on a spreadsheet, you will know and love them.

Some folklore on finite fields.

A finite field is a finite set of elements that forms a group under addition, that, with omission of the additive identity (the all 0's vector in our case) forms a group under multiplication, both operations of which are commutative. (This means the product ab is the same as ba)

The key property that distinguishes a field among “commutative rings” is the necessary property that the product of two non-zero elements is not the zero-element. A ring is a set of elements with operations of addition and multiplication that need not be commutative and which

Thus, the numbers mod z for prime z form a field. For $z=2$ the field consists of two elements, 0 and 1, with the usual properties, including $1+1=0$.

The numbers mod z when z is not a prime do not form a field. Why? Consider the cases 4 and 6. Mod 4 we have $2*2=4=0$. Mod 6 we have $2*3=6=0$.

Similar remarks obviously hold for any non-prime.

What is so good about having the product of non-zero elements being non-zero?

It implies that a product can be the zero element only if at least one of its factors is the zero element. This implies (by an easy induction argument) that a polynomial equation of degree k in a field can have at most k distinct solutions. This simple statement (it is the Fundamental Theorem of Algebra applied to finite fields) tells us here that our error locator polynomial with degree given by the number of errors, will give value 0 only for the errors, which means it will be the zero element on the errors and only on the errors, and can be

used to correct them.

Here are some more great properties:

1. If we use addition mod 2, as we do here, then x and x^2 obey the same polynomial equation. (Why? Because when you multiply out the square of the polynomial for which x has the 0 element as remainder, we get two kinds of terms: we get squares of the original terms, and cross terms. The cross terms all have 2's in them and hence are 0. We are left with the squares of the original terms which gives us the same equation for x^2 that we had for x .

Example: $x^3 + x + 1 = 0$, Squaring both sides we get $x^6 + x^2 + 1 + \text{cross terms} = 0$, and the cross terms are all 0 mod 2. If we denote the first equation as $f(x) = 0$, the result of squaring gives us $f(x^2) = 0$.

2. A field with k elements has $k-1$ non-zero elements, and we will necessarily have $x^{k-1} = 1$, for every x in it. We will prove this statement soon.
3. If a polynomial mod 2 has an even number of terms, it is 0 when evaluated at $x=1$, and must therefore have $(x+1)$ as a factor.

Why? If you divide any polynomial, say f by $x+1$, the only possible remainders mod 2 are 0 and 1. We then find $f(x) = (x+1)q(x) + \text{remainder}$, where $q(x)$ is the quotient in this division. Since both f and $x+1$ are the zero element when $x=1$, the remainder must be 0 there and everywhere.

4. If x obeys a polynomial equation $f(x)=0$ in our field, then x^{-1} obeys the equation obtained by reading $f(x)$ backwards. Thus for example, if $x^4 + x + 1 = 0$, we can abbreviate the left side as 10011. Reading this backwards gives us 11001, and we have the equation $x^{-4} + x^{-3} + 1 = 0$. Why? Just divide both sides of the original equation by x^4 and see what happens.
5. This tells us that a polynomial equation that reads the same way backwards and forwards, like $x^4 + x^2 + 1 = 0$ must be obeyed by y^{-1} if it is obeyed by y . If, for example $x^{15} = 1$, then we know from statement 1 above, that x, x^2, x^4, x^8 all obey the same equation as x . And the inverses of these powers cannot obey the same equation of degree 4, since it can have only 4 solutions in this field. On the other hand, the powers 3, 6, 12 and 24 obey the same equation (and the 24th power is the same as the 9th since we have $x^{15} = 1$.) These powers are symmetric, being 3, 6, -3 and -6. We can deduce that these powers obey an equation of degree 4 that is symmetric. .