

## 12. Correcting Errors in BCH Codes

Our polynomial code is now a product of a primitive polynomial,  $p(x)$ , and a second polynomial  $p_3(x)$  and perhaps a third,  $p_5(x)$ , and even more. These polynomials are defined so that the **remainder of  $p_j(x^j)$  is 0** upon dividing by  $p(x)$ .

If we evaluate our received message polynomial  $r(y)$  at  $y=x^j$  we will find that the message part of that received polynomial will have 0 remainder, since it will have  $p_j(x^j)$  as a factor and that factor will have 0 remainder, so we will **obtain the remainder of the sum of the  $j$ -th powers of the errors.**

We will illustrate the situation when there are two errors, but the general case can be treated in exactly the same way. The code will have more factors when you want to have the power to correct more errors; however the factors are determined by the same procedure, and the *syndrome* of the message is computed in exactly the same way, though there will be more of it.

When there are two errors, suppose they are in powers  $e_1$  and  $e_2$ . Our task is to determine these two unknown values.

The code in the two error case is  $p(x)p_3(x)$ , and we have

$$r(x) = m(x)p(x)p_3(x) + x^{e_1} + x^{e_2}.$$

By evaluating the remainder of  $r(x)$  and of  $r(x^3)$  on dividing by  $p(x)$  we obtain

$$\text{rem } r(x) = \text{rem } (x^{e_1} + x^{e_2})$$

$$\text{rem } r(x^3) = \text{rem } (x^{3e_1} + x^{3e_2}).$$

**Our task is to find the two error powers,  $e_1$  and  $e_2$ , from these two pieces of information.**

**We will do so by finding an (almost magical) *error locator polynomial*, which will have 0 remainder when evaluated at powers  $e_1$  and  $e_2$ , and non zero remainder otherwise.**

Finding the errors then involves evaluating the remainder of this error locator polynomial at each of the powers of  $x$  in turn. If we change each bit for which this polynomial has 0 remainder, we will correct the errors. We can then retrieve the original message (whose binary bits are those of  $m(2)$  in reverse order) by dividing the corrected message by  $p(x)$  and by  $p_3(x)$  and reading off the coefficients of the various powers in the result.

If we let  $y$  be our variable, the polynomial which will vanish (that is, have vanishing remainder) for  $y=x^{e_1}$  and  $y=x^{e_2}$ , will have to obey

$$\text{rem } (y - x^{e_1})(y - x^{e_2}) = 0$$

or

$$\text{rem } y^2 - (x^{e_1} + x^{e_2})y + x^{e_1}x^{e_2} = 0.$$

Notice that the coefficients in this equation will necessarily be powers of  $x$ , and not just 0 or 1.

**Notice also that we can determine this equation by determining the two coefficients ( $x^{e_1} + x^{e_2}$ ) and  $x^{e_1} x^{e_2}$ , that is, by determining their remainders.**

Finally, notice that the remainder of the first of these two coefficients is exactly what we do determine when we find the remainder of  $r(x)$  on dividing by  $p(x)$ .

**So all we need beyond what is necessary to correct one error in order to find this error locator equation, is to determine the remainder of  $x^{e_1} x^{e_2}$ , from the information we have. This information consists of the remainders of the sums of the first and third powers of the error monomials.**

We now introduce some definitions. Notice that here and in general, a coefficient of the error locator polynomial will be characterized by its degree. There will be a linear term, (which will be the coefficient of  $y^{n-1}$  which here is  $y$ ) a quadratic term, (a term of degree three and so on)

**The linear term is always just the sum of the error monomials. The quadratic term will always be the sum of all products  $x^{e_i} x^{e_j}$  with  $i < j$ ; the next term will consist of the sum of all products  $x^{e_i} x^{e_j} x^{e_k}$  for  $i < j < k$ , and so on.**

These coefficients are called the **elementary symmetric functions** of the error monomials. **We will denote these symmetric functions by  $s_1, s_2, \dots$**  Thus, our error locator polynomial can be written, when there are two errors, as

$$\text{rem } (y^2 + s_1(x) y + s_2(x)) = 0,$$

or, with 5 errors

$$\text{rem } (y^5 + s_1 y^4 + s_2 y^3 + s_3 y^2 + s_4 y + s_5) = 0.$$

On the other hand, the information that we do have, that we get from  $r(x)$ , are the **power sums** of the error monomials, or at least the odd **power sums** of the remainders. **We will denote these as  $t_j$** . That is,  $t_j$  is the sum over all errors  $e_k$  of  $(x^{e_k})^j$ .

**So our task in general is to go from the sums of powers of our error monomials to the elementary symmetric functions of these same monomials, which are the coefficients in the error locator polynomial.**

By the way, because we are using binary addition with  $1+1=2$ , we have  $t_{2j} = t_j^2$ , and so we get the even power sums by squaring odd ones, perhaps more than once.

We already know some things about the relations between the  $s$ 's and the  $t$ 's, which is the key to our error correcting.

In particular, we know  $s_1 = t_1$  since both are the sum of the error monomials. In general, both  $s$ 's and  $t$ 's with the same index  $j$  have degree  $j$  in the error monomials; a single term in  $t_j$  has one such monomial raised to the  $j^{\text{th}}$  power, while a single term in  $s_j$  is a product of  $j$  different error monomials.

When there are no more than  $j-1$  errors then  $s_j$  is 0.

**To correct two errors we need a second relation between the s's and t's, and we can get one as follows.** Take the defining equation for  $s_1$  and  $s_2$ :

$$\text{rem}(y^2 + s_1 y + s_2) = 0$$

**when  $y$  is an error monomial;**

**multiply through by  $y$ , and evaluate the resulting polynomial at the two values for which it is 0, namely at  $y=x^{e_1}$  and  $y=x^{e_2}$ .** Then sum both statements.

If you do so, you get

$$\text{rem}(x^{3e_1} + x^{3e_2} + s_1(x^{2e_1} + x^{2e_2}) + s_2(x^{e_1} + x^{e_2})) = 0,$$

or

$$\text{rem}(t_3 + s_1 t_2 + s_2 t_1) = 0.$$

This equation, with the facts that we already know. ( $t_2 = t_1^2$ ,  $s_1 = t_1$ ), gives us

$$\text{rem}(t_1 s_2) = \text{rem}(t_3 + t_1^3).$$

**This equation tells us all we need to know to correct up to two errors. The procedure is as follows.**

Use  $r(x)$  to find  $\text{rem}(t_1)$  (which is the same as  $\text{rem}(s_1)$ ) and  $\text{rem}(t_3)$  as previously discussed. **Then find  $\text{rem } s_2$**

**by using this last equation, and then check each power to see if it obeys the error locator equation.**

We still have to describe how to do these two things. We will proceed as follows:

First we will discuss how we can find  $s_2$  from the equation just above.

Then we discuss how to set up the power tester to find and correct errors.

## 11.2 Finding $s_2$ .

The equation for  $s_2$  just obtained requires that we be able to add and multiply and divide out remainders. We can do these things, but you should not take that for granted.

We add remainders by treating them as vectors and adding with  $2=0$ , as we have been doing.

We must also **multiply** our remainders. One way we could do this is to multiply the remainders (the way we usually multiply polynomials) and then divide by  $p(x)$  to get the remainder. There is an easier way: we can multiply remainders  $A$  and  $B$  by consulting our remainder table for the polynomial  $p(x)$ , and determining which powers have these remainders; if they are powers  $a$  and  $b$ , so that  $\text{rem}(x^a) = A$  and  $\text{rem}(x^b) =$

B, we determine  $A \cdot B$  by finding  $c = a + b \pmod{2^k - 1}$ . Then, the product of A and B will be  $\text{rem}(x^c)$ . Our remainder table is like a table of logarithms. To multiply you need only add the powers.

There is another very good reason to consult the remainder table to find products when we are making a spreadsheet for BCH codes. The reason is that our products are defined by remainders obtained when dividing by a specific primitive polynomial. If we divide by a different primitive polynomial, the rules for multiplying remainders will be different. Thus, the rule cannot be deduced from the remainders themselves, but requires us to know our primitive polynomial. If we multiply by consulting our remainder table, all the information we need to know about the primitive polynomial is contained in our remainder table, so we can easily change our primitive polynomial by changing our remainder table without making a whole new spreadsheet.

So what should we actually do to find errors, given remainders  $t_1$  and  $t_3$ ?

STEP 1: find  $t_1^3$ ; to do this, form the identifier for  $t_1$ ;  
then find the power that has remainder with this identifier;  
then triple that power mod  $(2^k - 1)$  (here  $k$  is the degree of  $p(x)$ .)  
then find the identifier for this power.  
And then find the remainder from this identifier.

(There is another way to do this when you have an ordinary (powers go up by one from row to row) remainder table and a (powers up by 3 from row to row) table side by side. Then when you find the identifier for  $t_1$  you can **find the identifier in the up by 3 table in the same row**. That will be the identifier for  $t_1^3$ .)

STEP 2: find  $t_3 + t_1^3$ .

To do this, add the remainders of  $t_3$  and  $t_1^3 \pmod{2}$ . That will be the remainder of  $t_3 + t_1^3$ .

STEP 3: find  $(t_3 + t_1^3)/t_1$ .

To do this, find the identifier of  $(t_3 + t_1^3)$  and its power.

If  $t_1$  has identifier 0 then there are no errors, and we can stop.

Add  $2^k - 1$  and subtract the power having remainder  $t_1$  to the power of  $(t_3 + t_1^3)$  and find the result mod  $(2^k - 1)$ . That will be the power of the indicated combination. Find the indicator and remainder of this power.

**Actually, there is a clever way of finding the errors that avoids all division, which is the hardest part of the above steps.**

This makes constructing the spreadsheets quite a bit easier. Multiply everything by  $t_1$ , and find the solutions of the resulting equation, as is described below. This was also described in class, but I may try to alter the notes this weekend to explain this in some more detail.

You can see from this description that performing additions and multiplications here involves repeated application of the following steps

**Going from a remainder to its identifier**; (if the remainder is  $(a,b,c,d,e)$  this consists of forming  $a + b*2 + c*4 + d*8 + e*16$ .)

**Going from an identifier id to a power p(id);** this involves a look up in the remainder table. You can do this on a spreadsheet by forming a column anywhere to the right of the remainder table with the entry =if(id=id(j),j,0) (by the way I think (id=id(j))\*j will work) and summing this over the column.

**The sum will be p(id). This method however will fail if id=0. So the sum should really be replaced by if(id=0,no power, sum over column)**

**Going from a power p to an identifier id(p);** This is the opposite lookup in the remainder table. You can do it on a spreadsheet by forming a column to the right of the remainder table with the entry =if(p=j,id(j),0) and summing it. **The sum will be id(p). If power =no power then the identifier id(no power) will be 0.**

**Going from an identifier id to a remainder.**

This can be done as follows

With id in any location put under it, =mod(id,2) and next to id on the right put =(id-mod(id,2))/2

And copy those two instructions to the right a so that there are k columns all together.

**The lower row will be the desired remainder.**

(Please note here that when we write something like =mod(id,2) on a spreadsheet you should always enter the location of id on the spreadsheet and not the symbols id.)

### 11.3 Having formed $s_1$ and $s_2$ what then?

The next task is to set up a power tester. You can do this by setting up **three remainder- like tables, one for each term in the error locator polynomial. Each has a column entry for each power that occurs in the remainder, 0 up to k-1 if your primitive polynomial has degree k. There is a row for every power from 0 up to  $2^k-2$ .**

The first, the  $y^2$  term table, starts with 1000... which is the remainder of the 0th power and **powers go up by 2** from row to row.

In the second, or  $s_1y$  term table, you insert the remainder of  $s_1$  rather than 1000..., and **powers go up by 1 as in an ordinary remainder table.**

In the third or  $s_2$  table the  $0^{\text{th}}$  power entry is the remainder of  $s_2$ , and it stays the same throughout the table.

**You next form the sum mod 2 of these three tables forming a new table with an entry for each row and column of the others which is the sum mod 2 of the corresponding element of all three. And you form an identifier column for it.**

**The rows for which the identifier is 0 correspond to the error powers.** Switching 0 and 1 in these rows of the received message will correct it..

There are two problems with what we have been doing. The first is that there may have been more than two errors, and the received word is not Hamming distance 2 from any code word.

If there are more than two errors, our second equation which relates s's and t's will not be correct

(an  $s^3$

term is missing from it when there are three or more errors), and any corrections we find using it will be wrong.

Sometimes, by using the computation described we will correct to the wrong message when there are 3 or more errors, and we lose. But some of the time we will end up with a received word that is not a distance 2 from any message word. In that case we will try to correct, but our corrections will not work. So our "corrected message," computed as described above, will not be divisible by  $p \cdot p_3$ .

Which means that our procedure fails.

(This will happen for roughly half of all possible 3-error combinations.)

We can check for this by taking the remainder of the computed corrected message. If it has non-zero remainder, our corrections will have failed. We can recognize that and detect our failure.

A second problem is that we left the no-error case dangling a bit. We can handle this in either of two ways.

One is to check early if  $\text{rem } r(x) = \text{rem } r(x^3) = 0$ , and see that we do not try to correct when this happens, (by making all corrections contingent on this not happening).

Actually, if  $r(x)$  has 0 remainder and  $r(x^3)$  does not, there must be at least 3 errors. The reason for this is that no two distinct error remainders can add up to  $0 \pmod 2$ , so whether there is one error or two  $r(x)$  cannot be 0. The same reasoning holds for  $r(x^3)$  only when  $k$  (the degree of the primitive polynomial) is odd. When the degree is even,  $r(x^3)$  can be 0 because the polynomial  $p_3$  is not primitive when  $k$  is even.

There is actually an easier way to set up the error checking and correcting step here, and that is to check the equation  $t_1 y^2 + t_1^2 y + (t_1^3 + t_3) = 0$ , obtained by multiplying every term in the error locator polynomial by  $t_1$ . This only alters what has to be put into the 0 power row of these three tables, and the simplification is that you can read all three of these coefficients from the remainders  $t_1$   $t_3$  and from  $t_1^2$  and  $t_1^3$  which can be obtained directly from remainders up by 1 2 and 3 tables, without ever having to convert to powers.

This only causes trouble if you have  $t_1=0$ . Then, after this multiplication, all terms in all these tables will be 0 (assuming that there are at most two errors) and there actually are no errors. In that case if you change all bits in rows in which the sum of the entries in the three tables is (0) then you will change them all, and therefore you should add a term to the decoded message that changes them all back. (add  $=(\text{sum of all indicators for final chart } =0)$  to each bit of the message.

The original computational method described required dividing by  $t_1$  to form  $s_2$  and this is a bit tricky if  $t_1 = 0$ . The way we indicated above to divide by  $t_1$  involves converting the (0) remainder to a power; but when  $t_1$  is (0) it is not the remainder of any power. If you are not careful, the method you use for converting to a power will convert it to the power that will be the sum of all 0's, and that will come out to be the  $0^{\text{th}}$  power, when it should really be no power at all)

### 11.4 Can This really be Done?

The following illustrates the error correcting procedure with tricks that can be used, here starting with the primitive polynomial  $1+x+x^4$ . First we compute  $p_3$  which involves row reducing in the following matrix.

computation of  $p_3$

power	0	1	2	3
0	1	0	0	0
3	0	0	0	1
6	1	0	0	1
9	0	1	0	1
12	1	1	1	1

Then we compute the row to row transitions with powers increasing by 1 2 and 3.

transition in remainder table				transition in up by 3 table			
a	b	c	d	a	b	c	d
d	a+d	b	c	b	b+c	c+d	a+d
transition in up by 2 table							
a	b	c	d				
c	c+d	a+d	b				

Then we compute the remainder table and up by two and three tables:

rem table					
power	0	1	2	3	id
0	1	0	0	0	1
1	0	1	0	0	2
2	0	0	1	0	4
3	0	0	0	1	8
4	1	1	0	0	3
5	0	1	1	0	6
6	0	0	1	1	12
7	1	1	0	1	11
8	1	0	1	0	5
9	0	1	0	1	10
10	1	1	1	0	7
11	0	1	1	1	14
12	1	1	1	1	15
13	1	0	1	1	13
14	1	0	0	1	9

up by 3 table					
power	0	1	2	3	id
0	1	0	0	0	1
3	0	0	0	1	8

6	0	0	1	1	12
9	0	1	0	1	10
12	1	1	1	1	15
0	1	0	0	0	1
3	0	0	0	1	8
6	0	0	1	1	12
9	0	1	0	1	10
12	1	1	1	1	15
0	1	0	0	0	1
3	0	0	0	1	8
6	0	0	1	1	12
9	0	1	0	1	10
12	1	1	1	1	15

up by 2 table

power	0	1	2	3	id
0	1	0	0	0	1
2	0	0	1	0	4
4	1	1	0	0	3
6	0	0	1	1	12
8	1	0	1	0	5
10	1	1	1	0	7
12	1	1	1	1	15
14	1	0	0	1	9
1	0	1	0	0	2
3	0	0	0	1	8
5	0	1	1	0	6
7	1	1	0	1	11
9	0	1	0	1	10
11	0	1	1	1	14
13	1	0	1	1	13

Next we take the dot product of  $r(x)$  with each one of these tables. Here is the result for  $t_1$

R dot rem table				get t1
0	1	2	3	
1	0	0	0	0
0	0	0	0	0
0	0	1	0	12
0	0	0	0	0
0	0	0	0	0
0	0	0	0	0
0	0	0	0	0
0	0	0	0	0
0	0	0	0	0
0	0	0	0	0
1	0	1	0	0
0	0	0	0	0
0	0	0	0	0
0	0	0	0	0
0	0	0	0	0
1	0	1	1	0
1	0	0	1	0

4	0	3	2	0	12	0
0	0	1	0	4	0	0

And here are the formulae that led to it

r dot rem table

0	1	2	3		get t1
=E10*H10	=E10*I10	=E10*J10	=E10*K10		=IF(L
=E11*H11	=E11*I11	=E11*J11	=E11*K11		=IF(L
=E12*H12	=E12*I12	=E12*J12	=E12*K12		=IF(L
=E13*H13	=E13*I13	=E13*J13	=E13*K13		=IF(L
=E14*H14	=E14*I14	=E14*J14	=E14*K14		=IF(L
=E15*H15	=E15*I15	=E15*J15	=E15*K15		=IF(L
=E16*H16	=E16*I16	=E16*J16	=E16*K16		=IF(L
=E17*H17	=E17*I17	=E17*J17	=E17*K17		=IF(L
=E18*H18	=E18*I18	=E18*J18	=E18*K18		=IF(L
=E19*H19	=E19*I19	=E19*J19	=E19*K19		=IF(L
=E20*H20	=E20*I20	=E20*J20	=E20*K20		=IF(L
=E21*H21	=E21*I21	=E21*J21	=E21*K21		=IF(L
=E22*H22	=E22*I22	=E22*J22	=E22*K22		=IF(L
=E23*H23	=E23*I23	=E23*J23	=E23*K23		=IF(L
=E24*H24	=E24*I24	=E24*J24	=E24*K24		=IF(L
=SUM(AD10:AD24)	=SUM(AE10:AE24)	=SUM(AF10:AF24)	=SUM(AG10:AG24)	=SUM(AH9:AH24)	=SUM
=MOD(AD25,2)	=MOD(AE25,2)	=MOD(AF25,2)	=MOD(AG25,2)	=AD26+AE26*2+AF26*4+AG26*8	=MOI

We omit the similar tables for t<sub>2</sub> and t<sub>3</sub>

We then compute t<sub>1</sub><sup>3</sup> + t<sub>3</sub>: after extracting rem t<sub>1</sub><sup>3</sup> from its identifier. It is the third line in this table.

0	t3	0	1	0	0
	t1^3	0	0	1	1
	t1s2	0	1	1	1
		12	6	3	1

The formulae for this table are, the last line being used to get rem t<sub>1</sub>#.

t3	=MOD(AL25,2)	=MOD(AM25,2)	=MOD(AN25,2)	=MOD(AO25,2)
t1^3	=MOD(AL29,2)	=MOD(AM29,2)	=MOD(AN29,2)	=MOD(AO29,2)
t1s2	=MOD(AL26+AL27,2)	=MOD(AM26+AM27,2)	=MOD(AN26+AN27,2)	=MOD(AO26+AO27,2)
	=AI25	=(AL29-AL27)/2	=(AM29-AM27)/2	=(AN29-AN27)/2

Finally we construct the tables of the error locator polynomial.

We illustrate the last two tables, namely those for the constant term and the sum

(t3+t1^3)*t <sub>1</sub> table				sum table			id	
0	1	1	1	1	0	0	1	9
0	1	1	1	1	1	0	1	11
0	1	1	1	0	1	1	1	14

0	1	1	1	0	0	0	0	0
0	1	1	1	0	0	1	1	12
0	1	1	1	1	1	0	1	11
0	1	1	1	0	0	0	0	0
0	1	1	1	0	1	0	0	2
0	1	1	1	1	0	0	1	9
0	1	1	1	1	0	1	0	5
0	1	1	1	0	0	1	1	12
0	1	1	1	1	0	1	0	5
0	1	1	1	0	1	0	0	2
0	1	1	1	1	1	1	0	7
0	1	1	1	1	1	1	0	7

106

The first two terms had the following tables

error locator polynomial term 1

t1\*up by 2 table

t2\*up by 1 table

0	0	1	0	t2	1	1	0	0
1	1	0	0		0	1	1	0
0	0	1	1		0	0	1	1
1	0	1	0		1	1	0	1
1	1	1	0		1	0	1	0
1	1	1	1		0	1	0	1
1	0	0	1		1	1	1	0
0	1	0	0		0	1	1	1
0	0	0	1		1	1	1	1
0	1	1	0		1	0	1	1
1	1	0	1		1	0	0	1
0	1	0	1		1	0	0	0
0	1	1	1		0	1	0	0
1	0	1	1		0	0	1	0
1	0	0	0		0	0	0	1

The remaining steps are checking, and dividing.

We now discuss generalizing this procedure to correct more errors. (It can be accomplished in exactly the same way) and then considering why it works, and properties of these codes and further generalizations.

**Exercise 1: Construct a coder and error corrector that corrects two errors with your favorite degree 5 or degree 6 primitive polynomial, (degree 6 is no harder than degree 5 or degree 4).**

1. choose a primitive polynomial p and construct its remainder table
2. construct p3 from p
3. construct an encoder that multiplies first by p and then by p3 (or however you choose to multiply)

4. **construct up by two and up by three power-remainder tables**
5. **take dot products of the received word with each of your tables.**
6. **find  $t_1^3+t_3$ .**
7. **set up the tables for the error locator polynomial terms and their sum.**
8. **correct the errors.**
9. **divide twice to get the message back.**

**You don't have to do it exactly this way but it seems easiest to me.**

2. Find a primitive polynomial of degree 8.

3. Extra Credit: Suppose you choose as your coding polynomial a primitive polynomial  $p$  of degree 5 multiplied by its reverse polynomial, (which is gotten by switching its  $j^{\text{th}}$  power term with power  $5-j$  for each  $j$ .) Can you correct up to two errors with this code? (Such a code allows you to compute  $t_1$  and  $t_{-1}$  as a syndrome). Hint: find an equation for  $s_2$

in terms of these two  $t$ 's assuming that there are two errors. (Do the same if there is only one error as well) As long as there is no confusion between one error and two errors the code will work if you find these. Can it work if your primitive polynomial has even degree? Set up an encoder, error supplier, error corrector, decoder for a degree 5 primitive polynomial.