

10. Polynomial Codes and some lore about Polynomials

10.1 Introduction:

We will now introduce more structure for our message words and code words, and use it to get single error correcting codes that can be described more easily, as well as codes that can correct several errors.

We do this by treating **our sequences as polynomials. This allows us a natural way to define multiplication and division for them.**

Given a sequence, say **1011**, we can make it into a polynomial by converting each bit a_j into $a_j x^{j-1}$ and adding the results, For the given sequence we get $1 + x^2 + x^3$.

We multiply our sequences by the usual laws of multiplication, again with the $2=0$ rule. Thus, in multiplying (111) by (1011), we multiply $1+x+x^2$ by $1+x^2+x^3$ which using the distributive law comes to $1 + x + 2x^2 + 2x^3 + 2x^4 + x^5$ which we take to be $1 + x + x^5$.

We will describe encoding by use of an encoding polynomial $p(x)$ and our rule for encoding will be

$$c_m(x) = p(x) m(x).$$

Now, if our rule for encoding is multiplication, what is our rule for decoding? How do you undo multiplication? Division!

We want codes that will correct one error. What does a single error look like in a polynomial code? It looks like a monomial, x^t . And when we undo our multiplication by dividing by $p(x)$, we will recognize this error by looking at the **remainder** of x^t divided by $p(x)$. We will explain this in much more detail later, but first, we need to know some more properties of polynomials.

Thus the task of finding useful codes is, in this context, the task of finding useful polynomials. In order to find them we first look at some properties of polynomials.

10.2 Binary Polynomials

We will find, that if you want to find efficient single error correcting polynomial codes, you can do so by choosing an encoding polynomial that is "**primitive**". **Primitive** means prime, plus one other property we shall soon describe. So we will begin by looking at low degree prime polynomials.

Recall that binary numbers obey these rules for addition:

$$\begin{aligned} 0+0 &= 1+1 = 0 \\ 1+0 &= 0+1 = 1 \end{aligned}$$

and these for multiplication:

$$0*0 = 0*1 = 1*0 = 0$$

$$1*1 = 1.$$

A polynomial defined over this "field" (the field is called GF(2)) is no more than a polynomial whose coefficients are each 0 or 1. (A field is a set of entities that can be added, subtracted, multiplied, and divided while obeying the same rules as ordinary numbers obey. Examples are the real numbers, the rational numbers, the complex numbers, and 0 and 1 subject to the rules for multiplication and addition just stated.)

Here is a polynomial of degree five: $1 + x^2 + x^5$.

Polynomials have the wonderful property that you can add and subtract them and also multiply and divide them using all the standard commutative, distributive and associative laws of arithmetic.

When you divide one polynomial, say $p(x)$, by another, say $q(x)$, you can get both a quotient polynomial (whose degree will be the degree of p minus that of q) and perhaps a remainder (whose degree will be strictly less than that of q).

The rules for dividing are the rules for long division. (actually, when you write a number as say 543 you are treating it as a sort of polynomial, namely $5*10^2 + 4*10 + 3$, with 10 here replacing the variable x .) The rules you learned as a child for dividing numbers having several digits are exactly the rules to be used for dividing polynomials, except here $2=0$ holds and there is no carrying. polynomials, except for carrying, which you do with numbers and not with polynomials. (in our case, given $2=0$, there is no carrying to do)

With our polynomials life is much more pleasant than what you encountered in your youth when you want to do arithmetic since the only possible non-zero coefficient of our polynomials is 1.

We give an illustration of division of polynomials.

Suppose we want to divide x^8 by $1 + x^2 + x^5$.

The latter goes x^3 times into the former, with a remainder of $x^5 + x^3$. It goes once into this remainder with $x^3 + x^2 + 1$ left over. The resultant quotient is therefore $x^3 + 1$ with remainder $x^3 + x^2 + 1$.

A polynomial is said to be prime if it cannot be factored into polynomials of lower degree.

The polynomial of degree five above happens to be prime. On the other hand the polynomial $(1 + x^2 + x^4)$ is not prime, being $(1 + x + x^2)^2$.

We will now look at polynomials defined over our field GF(2) of low degree (there aren't many) and identify which are primes.

A monomial corresponds to a bit stream of Hamming weight 1. We can immediately see that x is the only monomial that is prime; the rest have x as a factor.

Any polynomial, $p(x)$ with an even number of terms has $(1 + x)$ as a factor.

Why? Because if you set $x = 1$ you find $p(1)=0$, when p has an even number of terms. And when you divide $p(x)$ by $(1+x)$ you get a remainder that must have 0 degree and so must be 0 or 1.

But we have $p(x)=q(x)(1+x) + r(x)$ with $q(x)$ the quotient upon dividing p by $(1+x)$, and $r(x)$ the remainder. Evaluating this at $x=1$ we get $0 = p(1) = q(1)*0 + r(1)$.

We conclude that $r=r(1) = 0$, and this means $(1+x)$ divides any polynomial with an even number of terms without a remainder.

Any polynomial without a 1 term is divisible by x .

Further,

Any polynomial $p(x)$ whose terms all have even degree is the square of the polynomial whose terms have degree half of the terms of $p(x)$, so $p(x)$ cannot be prime.

This is true because the ugly cross terms which normally occur when you square a polynomial all have factors of 2 multiplying them so they are all 0 here. The resulting terms of a square are then only the squares of the terms in the original polynomial.

There is one more fact that is useful when examining polynomials. If we take a sequence like 1011 and make it into a polynomial, we can do so in two ways. The way we have chosen is to start from the **left**, and have successive bits correspond to increasing powers.

We could just as well have done the same thing starting from the **right**.

But this gives us a different polynomial, which we call the "**reverse polynomial**" to the first. In our example, our polynomial is $1 + x^2 + x^3$ and the reverse one is $1 + x + x^3$.

Now no important property of our code can depend on whether we started from the left or the right in turning our sequences into polynomials.

We conclude from this that the properties of a polynomial and its reverse such as **primitivity and primality must be the same for each polynomial and its reverse**. This conclusion will be justified by what we see very soon: that these properties have a direct effect on the error correcting properties of the code a polynomial generates.

So let us explore the polynomials of low degree and classify them as prime or composite. Prime means cannot be factored into polynomials of lower degree.

There are two polynomials of degree one, x and $1+x$, and both are prime

The only possible prime polynomial of degree two must, by the discussion above, have terms 1 (so as not to be divisible by x) and x^2 (in order to have degree two) and must have an odd number of terms (so as not to be divisible by $1+x$). There is one and only one and it is $1 + x + x^2$. It is prime, as you can see because it is not the product of any combination of the degree 1 primes. It is symmetric on reversal of bits

The only possible primes of degree three with odd number of terms having 1 and x^3 as terms are $1 + x + x^3$ and $1 + x^2 + x^3$, which are reverses of one another. If they were not primes they would have to have a factor of degree one, but they do not, because they have an odd number of terms.

Among polynomials of degree four the only polynomials with a constant term, with a term of degree four, with an odd number of terms, and not having all terms of even degree, are $1 + x + x^4$, $1 + x^3 + x^4$, which are reverses of one another and $1+x+x^2+x^3+x^4$, which is symmetric. By the reasoning used for polynomials of degree three in the last paragraph, all of these are prime. (The square of $(1+x+x^2)$ is $1+x^2+x^4$)

By similar reasoning the candidates for primes of degree five are $1+x+x^5$, $1+x^2+x^5$, all terms **except** x , all terms except x^2 and the reverses of these 4 polynomials. Of these only $(1+x+x^2)(1+x+x^3)$ and its reverse are not prime; all the others are prime.

You could, if you wanted to, continue this investigation to degree six through ten. For example, for degree 8, by my calculation, there are 64 polynomials with an odd number of terms including 1 and x^8 , which form 28 pairs of a polynomial and its reverse, and 8 symmetric polynomials. Of these about half are primes.

10.3 Which Polynomials Make Good Codes?

The polynomials we want are those that are not only prime, but **primitive** as well. **A primitive polynomial is one such that every possible remainder is a remainder of a power of x .**

Consider the simplest non-trivial example, which is the polynomial $1+x+x^3$.

What are the possible remainders?

Because our polynomial has degree 3, **a remainder is** a polynomial of the form $a+bx+cx^2$; that is, **a polynomial of degree one less than that of our polynomial, such that a , b and c are each chosen from $(0,1)$, and not all are 0.** (If x^k were to have 0 remainder that means our polynomial is a factor of a monomial so it would have to be a monomial, and that is too silly for us to consider)

If our polynomial has degree k , the number of such possible remainders is 2^k-1 , which for $k=3$ is 7. The possible remainders are **$1, x, x^2, 1+x, 1+x^2, 1+x+x^2$. and $x+x^2$.**

It is slightly more convenient to represent these remainders as 3-, or more generally, k -component vectors, in which form we would write them as **$100, 010, 001, 110, 101, 111$, and 011 .**

Sometimes, when you want to recognize them easily you can interpret them as numbers. The way I like to do it is perhaps not the best but it works. I treat them as binary numbers **read backwards**; they then read **$1,2,4,3,5,7$, and 6** in the order given.

Our polynomial will be primitive if every one of these remainders is the remainder of some power of x .

We can test for this by writing out the remainders of powers in ascending order. We know that $x^0=1$ has remainder 1.

We will be able to write out the remainders of all powers very conveniently on a spreadsheet. We do this by giving a **rule for finding the remainder of x^{j+1} given the remainder of x^j** . We do this on a spreadsheet by applying the rule to go from the remainder of 1 to the remainder of x , and copying that rule down the columns which contain the remainders. This will allow us to test a polynomial for primitivity very quickly using a spreadsheet.

So what is the rule for finding the remainder of x^{j+1} from that for x^j ?

Suppose we have Remainder of $x^j = \mathbf{rem}(x^j) = \mathbf{a + bx + cx^2 + \dots + sx^{k-2} + tx^{k-1}}$. We get x^{j+1} from x^j by multiplying it by x . We know that $x^j = p(x)q(x) + \mathbf{rem}(x^j)$ for some $q(x)$. We therefore have $x^{j+1} = p(x) [xq(x)] + x\mathbf{rem}(x^j)$, the first term of which has no remainder so we have $\mathbf{rem}(x^{j+1}) = \mathbf{rem}(x\mathbf{rem}(x^j))$.

If the coefficient, t , of x^{k-1} , is 0, the effect of multiplying $\mathbf{rem}(x^j)$ by x is to increase each power by 1, so that we get $\mathbf{rem}(x^{j+1}) = \mathbf{ax + bx^2 + cx^3 + \dots + sx^{k-1}}$ when $t=0$.

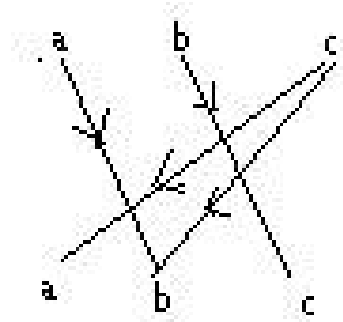
If instead, $t=1$, the last term in $\mathbf{rem}(x^j)$ is x^{k-1} . When multiplied by x this becomes x^k , whose remainder is $x^{k-p(x)}$.

We conclude, that the general form for the remainder of x^{j+1} here is

$$\mathbf{rem}(x^{j+1}) = \mathbf{ax + bx^2 + cx^3 + \dots + sx^{k-1} + tp(x) + tx^k}.$$

Suppose we consider the polynomial p given by $p(x) = x^3 + x + 1$. Its remainder has three coefficients and let us denote them as a , b and c as above.

The rule for going from x^j to x^{j+1} in is then (a,b,c) becomes $(c, a+c, b)$. The a and b terms move over one place to the right and c goes to the 1 and the x places. The following pictures show how bits propagate as we increase powers of x by 1 among the bits of the remainders.



successive power rule

$$1 + x + xxx$$

The remainders of powers, according to this rule, are

Power	Remainder			Binary Representation
	a (1)	b (x)	c (x ²)	
0	1	0	0	1
1	0	1	0	2
2	0	0	1	4
3	1	1	0	3
4	0	1	1	6
5	1	1	1	7
6	1	0	1	5
7	1	0	0	1

We call this a remainder table for the polynomial $p(x)$. Notice that we have added a column on the right which converts the information contained in the binary bits a, b, and c, into a single number, using the formula $a+2b+4c$.

You will notice that every **remainder appears on this list**, which means that 1 appears as a remainder for the first time as the 0th power of x and for the second time at the 7th power of x. More

generally, for a primitive power, the remainder 1 first reappears at the $(2^k-1)^{\text{st}}$ power. **This behavior characterizes a primitive polynomial. A non-primitive polynomial has the first occurrence of 1 as a remainder at a smaller power.**

Notice that the matrix appearing in the middle of the chart from the 3rd power row to the 6th power row is essentially a Z type matrix with an identity matrix on top. The 7 column, 3 row matrix from rows corresponding to powers 0 through 6 is actually the decoding matrix D (the message destroyer) for the code obtained by multiplying by this polynomial, as we shall soon see.

How can we test a polynomial for primitivity?

We can set up a spreadsheet to generate the remainders of powers, and locate the first power for which the binary number is 1. If that power is 2^k-1 we have a primitive polynomial, and otherwise not.

Once you have set up a test spreadsheet this way, you can change the polynomial to be tested by changing the line on which you enter its succession rule, and copy it down to apply it everywhere. This involves modifying at most $k-1$ entries and copying. (the a entry is always t so it never changes).

You can test polynomials for primitivity without a spreadsheet by using tricks. Thus the remainder of x^{2^j} is the square of the remainder of x^j . If you only write the remainders of the even powers from k to $2k-2$, you can compute all the remainders of powers of the form 2^j from these, and if you find that the remainder of the 2^k power is not x , you know that the polynomial is not primitive. It can happen that the remainder of 2^k is x and the code is not primitive, if the remainder of some lower power is also x . You can check this by checking whether the remainder powers that are factors of 2^k-1 are 1. If not, the polynomial is primitive.

In our example, $1+x+x^3$, we have $\text{rem}(x^4) = x+x^2$. Squaring, we get $\text{rem}(x^8) = \text{rem}(x^2 + x^4) = x$, and our code is a candidate for primitivity. Since $2^k-1=7$ is a prime, this implies that the polynomial is primitive.

Why is primitivity important to us?

Our plan for decoding is to find the remainder of the received message, $r(x)$, on dividing by $p(x)$. Since the encoded message, $m(x)p(x)$, has no remainder, this will be our message killer. The remainder will be caused entirely by the error or errors, and will be independent of the message.

If the error was in one bit only, suppose the error bit occurs in the e^{th} power. Then the error has to be the error of the monomial x^e .

We can find the error location e by finding the power of x that has remainder $\text{rem}(r(x))$, something that can be read off from the remainder table.

We can do this, so long as no two rows of the remainder table are the same, which means that no two powers have the same remainder.

A primitive polynomial will have the first repetition in the remainder table occur at the highest possible power for its degree. Repetition is inevitable if every non-zero remainder has appeared already, and

for a primitive polynomial all non-zero remainders do appear in its remainder table before any repetition.

Notice that once there is a repetition in the remainder table, the table cycles. The remainder of the next power depends only on the remainder of the present one. Therefore, if the remainder of x^a is the same as that of x^b , then the remainder of x^{a+s} will be the same as that of x^{b+s} for all s .

10.4 Decoding a Single Error Correcting Code Generated by a Primitive Polynomial

As outlined above, to decode we first find the remainder of our received message, $r(x)$, upon dividing by the primitive polynomial $p(x)$.

The obvious way to do this is to divide and look at the remainder, as we have said. But there is an easier way which is based upon the fact that the remainder of a sum of polynomials is the sum of their remainders:

$$\text{rem}(a(x) + b(x)) = \text{rem}(a(x)) + \text{rem}(b(x)).$$

This means we can find the remainder of $r(x)$ by summing up the remainders of the individual bits (monomials) in it. This can be achieved very easily with a spreadsheet from the remainder table. The procedure is to take the dot product of r with each of the columns of the remainder table. The procedure is the same as multiplying the received message by the matrix formed by the remainder table.

(You construct a second table whose rows are the entries of the remainder table each multiplied by the bit of r corresponding to that row. You then sum these rows mod 2, compute the binary number corresponding to that sum, and identify the row of the remainder table having that binary number as its identifier. Switching that bit corresponding to that power will produce the sent message, $m(x)p(x)$.)

Once you have found the remainder of the error and located it, you must then divide the sent message $m(x)p(x)$ by $p(x)$ to find $m(x)$. This can also be accomplished on a spreadsheet, without a huge amount of effort. Dividing, it turns out, is just like multiplying except sort of upside down, as you will see

Here is a somewhat confusing description of how to divide. To do so, write the message to be divided **from right to left** as a column, (say it is 0111001), and then write the code polynomial (say 1101) similarly **from right to left** as a column (that is, in reverse), to its left next to it. You construct columns to the right of the upside down message as follows.

(Why upside down? When we divide we usually put the highest power first. In our encoding we have put the lowest powers first. To get from one of these to the other we must reverse order. We really do not have to do this reversal to find the quotient message but if there were a remainder we have to reverse like this to get the remainder right. Otherwise you will get the remainder expressed as a sum of the highest power remainders rather than the lowest power remainders, which is its usual form.)

At each stage there are two possibilities: either the top bit in the previous column is 1 or 0. If it is 0 you shift the column upward by 1 into the next column; if it is 1 you add the code column except for its top entry to it, and shift the sum you have obtained up by one. That is what long division does. The entry in the j^{th} row of the next column is the sum of the entry in the $j-1$ row of the present column and the top row of the present column multiplied by the entry of the code polynomial reversed in the $j-1$ row, mod 2.

Shifting the column over corresponds to looking at the next lower power. Adding the code polynomial except for its top entry corresponds to replacing the highest power in the column by the corresponding lower powers of the code polynomial, and entering the fact that you have done so in the next column.

Here is what you get for the example given: dividing $(x^6 + x^3 + x^2 + x)$ by $(x^3 + x + 1)$. The first column here is the polynomial, the second column would not appear in the spreadsheet; I put it there to show you which cells correspond to which powers of x ; hopefully this will illustrate better how the procedure works. The mp column is the product of the polynomial with the message. If there are no errors, this is the same as the received message. The answer m appears in the row at the top, and the remainder (0 in our example) appears in the last column.

p		mp	1	0	1	0	= quotient message highest powers first
1		1	0	1	0	0	Last entry is x^2 term of remainder; previous entries are quotient highest powers first
0	$x^6 \nearrow$	0	1	0	0	0	Last entry is x term of remainder
1	$x^5 \nearrow$	0	0	1	0	0	Last entry is constant term of remainder
1	$x^4 \nearrow$	1	1	1	0		
0	$x^3 \nearrow$	1	1	0			
0	$x^2 \nearrow$	1	0				
0	$x \nearrow$	0					
	$1 \nearrow$						

The result of this division is x^3+x .

If you look at the x^j terms (or the x^j terms for any constant j), they move diagonally upward as indicated by the arrow in the second column (except for the top row, which is the quotient).

What appears in the last column here is the remainder with the most significant bit at the top. Thus if the top bit were a , the next bit b , and the bottom one c , this would correspond to the remainder cba in our previous notation, or $c + bx + ax^2$.

Notice that you can get rid of the received word $r(x)$ by division as done here, and that is also the obvious method for finding the remainder. However, division is really mostly useful in finding the quotient rather than the remainder. It is actually easier to find the remainder by using the remainder table. You will see that the remainder table has other uses.

10.4 Comments

So far, we have introduced multiplication of polynomials as a way to code, and seen a way to find and describe Hamming codes with k check bits using one polynomial of degree k , instead of filling in a $2^k - k - 1$ by k matrix of check bits.

The method of finding errors that we have described here, by finding the remainder of $r(x)$ and discovering which power has that remainder, seems quite different from the method used for general matrix codes. That consisted of matrix multiplying the remainder r vector by the message killing D matrix and seeing what row of D the answer agreed with.

If we find the remainder by long division, the methods of error location really seem quite different.

However if you compute the remainder by adding up the remainders of the powers in $r(x)$ you are actually matrix multiplying the $r(x)$ row vector by the remainder table matrix, and comparing the result with the rows of the remainder table matrix. Since the remainder table matrix when multiplied on the left by any code word will give the 0 vector, the procedure from this point of view is identical to the previous one. The remainder table provides the message killing matrix D for the code.

The code generated by a polynomial p can be considered a matrix code, in which the first row contains the encoding polynomial, written lowest power first, followed by a string of 0's, and successive rows have that polynomial shifted by 1 to the right from their immediate predecessors; in the final row the last polynomial bit reaches the last column.

The remainders we encounter when dividing by a primitive polynomial can be added in the usual way. But if we have our remainder table, we can also multiply them.

Each non-zero remainder is remainder of a power of x , which power can be determined from the remainder table.

We define the product of two remainders to be the remainder corresponding to the power that is the sum of their powers. When this sum exceeds $2^k - 1$ we can use the fact that

$$x^{2^k - 1} = 1$$

to subtract $2^k - 1$ from that sum of powers.

For example, with our old remainder table:

Power	Remainder			Binary Representation
	a (1)	b (x)	c (x ²)	
0	1	0	0	1
1	0	1	0	2
2	0	0	1	4
3	1	1	0	3
4	0	1	1	6
5	1	1	1	7
6	1	0	1	5

we define the product of 110 and 111, which are the third and fifth powers of x , to be x^8 , which (subtracting 7) is x or 010.

The addition and multiplication defined here for a primitive polynomial satisfy all the standard laws that addition and multiplications of numbers obey; the 0 polynomial plays the role of 0, and the polynomial 1 plays the role of 1. Obviously the product of two powers is another power and that is never 0, and the sum of two remainders is another remainder.

Such remainders therefore form what mathematicians call a **field**, which implies that they can be considered as number systems, like the real numbers or the rational numbers or the complex numbers or $GF(2)$. This field is called $GF(2^k)$.

This is not true for remainders upon dividing by a factorable polynomial, like $p(x)q(x)$ with each factor of degree at least 1. The problem with factorable polynomials is that the remainder of the product $p(x)$ times $q(x)$ is the 0 polynomial. This means that two non-zero remainders can have product 0. This is unacceptable for numbers in a field.

A very important fact about real and complex numbers, which we want all number systems to preserve is: (It is a part of the fundamental theorem of algebra.)

A polynomial equation of degree k can have at most k solutions.

This statement is true if the coefficients of the polynomial are elements of a field and generally false otherwise.

This statement will be important to us so **we now prove it.**

Suppose we have a polynomial equation of degree k with coefficients in a field.

If k is 1 the equation takes the form $ax-b=0$ for non-zero a . If c is a solution, we must have $ac=b$ which implies $c = b/a$, which makes c a unique element of the field.

We suppose now that the statement (that an equation of given degree d has at most d solutions) is true for all polynomials of degree $k-1$, and prove that it is true for any polynomial $p(x)$ of degree k .

Suppose c is a solution to the equation $p(x)=0$. Then $(x-c)$ must divide p without a remainder. (Otherwise we would have $p(x) = (x-c)q(x) + r$ and $p(c) = r$ and not $p(c)=0$.)

This means that we can write $p(x)$ as $(x-c)q(x)$, where $q(x)$ has degree $k-1$ and q has at most $k-1$ solutions by our induction hypothesis..

Any solution to $p(x)=0$ therefore obeys $(x-c)q(x)=0$.

x-c

has only one solution and $q(x)$, being of degree $k-1$, has at most $k-1$ of them by the induction hypothesis.

Since no two non zero field elements have product 0, the only solutions occur where one or another of the factors of p is 0.

So $p(x)=0$ can have at most k solutions, as we set out to prove.

Now we ask: **what can we possibly do to correct more errors in a polynomial code?**

Using a primitive polynomial, $p(x)$ as encoding polynomial works splendidly to correct one error, and we certainly want to maintain this power. **To correct two errors we will use a polynomial that is the product of a primitive polynomial and a second polynomial which we choose to provide the additional information that we need to locate two errors.**

What second polynomial do we choose here?

We will choose a polynomial $p_3(x)$ which obeys the condition

$$\text{rem}(p_3(x^3)) = 0$$

where we take the remainder on dividing by our primitive polynomial p as before.

And it will turn out that we can correct three errors by having a third factor, p_5 in the encoding polynomial that obeys

$$\text{rem}(p_5(x^5)) = 0,$$

(again on dividing by the primitive polynomial) and so on, to correct more errors.

In the next chapter we address the two questions that arise here.

- 1. How do we find polynomials p_3 and p_5 and p_7 and so on?**
- 2. How can we locate and correct two or more errors if our encoding polynomial has such factors in it?**

The answer to this question also answers the question: Why do we want such factors?

Codes of the form outlined here are called BCH codes, after the three people who discovered them: Bose, Ray-Chaudhuri, and Hocquenghem.

Exercises:

- 1. Find all primitive polynomials of degree six (over the two element field $GF(2)$ defined by $2=0$.)**
- 2. Pick a primitive polynomial of degree five. Construct a spreadsheet encoder for it, that takes any binary message of length 26 and converts it into a coded message using that polynomial as encoding polynomial.**
- 3. Construct a spreadsheet single error corrector for it, that starting with any received message**

of length 31 takes its matrix product with the remainder table, locates the error and corrects it.

4. Construct a spreadsheet divider that takes the corrected code word and finds the message that was encoded to it.