Clearing up the CRC business PhL 7.3.13

PART I : CORRECTING WRONG STATEMENT ABOUT ERROR DETECTION OF CRC. 1
PART II : CORRECTING CONFUSING COMMENTS ABOUT N,K AND CYCLICNESS. 6

Part I : Correcting wrong statement about error detection of CRC.

Before I do this, I have to get straight these three words

period of some h(x) min poly in Galois theory where smallest n such that $(x^n-1)/h(x)$ is even
order of an element α of GF(q)
exponent of GF(q) as I used it in Galois (order of largest cyclic subgroup of GF(q) - 0.
exponent as used in the CRC original paper below. (where it means period!)

Here was my **erroneous** CRC statement

So imagine a packet of 100,000 bytes followed by 16 parity check bytes all stored on some medium. In theory, the code distance could be d = 17 in this example, according to (7.34), so in the long data packet (code word) there would have to be a 17-symbol (17-byte) error in order that the code word be interpreted as some other valid code word. Since symbol errors are presumably rare, perhaps on the order of 10^{-6}, an error of 16 or less bad symbols would be reliably detected by this CRC scheme.

July 3, 2013. I am preparing to fix my CRC problem. I have started reading (proofing) at Section 7 (f) which is the first discussion of distance. I am making tiny edits along the way, cosmetic really.

(f) is excellent
(g) is excellent
(h) just fine
(i) also just fine, mention of turbo codes, and here Chapter 7 ends.

There is really not much about "distance" other than definition and a few facts. If you know the H matrix, then you might know the distance!

The CRC packet has k = 1000 data symbols and perhaps n-k = 16 parity check bits. I will continue reading into Chapter 8.

opening -- good example, fine

(a) OK, but reader still has no idea what the cyclic systematic basis is. That comes next

(b) I claim that for the cyclic systematic basis, you can find a matrix G that does the same thing in a regular block code, but I never pursue this idea. This section is OK

(c) encoders and decoders and syndrome in the systematic basis. Maybe I should have called this thing S(x), since it is no doubt different from the **s** mentioned in the previous block code chapter. But that was a vector, and now we have s(x), I think it is OK to stay with s(x).

(d) is the CRC section!

OK, I am now ready to start working on the problem. I have put in red my wrong statements.

We know that $d \le n-k+1$ so $d \le 17$ for the CRC situation. But no one says this is a maximal length code, so no one can say that d = 17 for example. So do *I* know anything about d in this situation?

d = smallest distance between any pair of code words

d = min weight of all the code words = the min number of non-zero symbols in a code word!

Suppose the entire data packet was zero. Then the parity check bits would be 0 I am pretty sure from (8.8) and then result is an all zero code word and d = 0 ! But we could rule out such a data packet.

Suppose that for an ensemble of k = 1000 long data packets, each packet has at most 50 zeros. Then the parity checks won't all be zeros, but suppose they were. Then all n packets have $\le$ 66 zeros, so they all have weight > 1016-66 = 1000 - 50 = 950 so d = 950! If we scramble the data stream, then the probability of having a lot of zero bytes can be statistically computed and is small. So this is some kind of statistical problem.

So now I will go to the web and see what I can find. I found the original CRC paper!

The CRC was invented by W. Wesley Peterson in 1961. Wow, this is the author of the book, I never knew this connection to CRC. The first edition was 1961. I wonder what he called CRC at that time, and maybe that is where it is located in the book!

Proceedings of the IRE (Volume:49 , Issue: 1) "Cyclic Codes for Error Detection"

I have the paper now as a PDF from somewhere. Let's lock onto **his conventions** before trying to decode the theorems claimed.

n, k same as me
m-tuple ordered with first symbol the lowest power of X
highest order sent first same as me, but his argument seems wrong so I ignore it
P(x) generator degree n-k same as my g(x) generator degree n-k
m = degree
code polynomials divisible by P(x) check
G(x) = message poly = my d(x) data poly
H(x) = received message
F(x) = code poly, has degree < n he says (does not use "code word")
E(x) = difference H(x)-F(x)
t = number of errors detectable
m = maximum possible value of exponent e = my q-1 = p^m-1 = 2^m - 1 for him (page 1961 left)
n = length of code words = same as mine = q-1 for a Galois Induced Code.
e = exponent to which any P(x) belongs means smallest e such $(x^e-1)/P(x)$ has no remainder.

So his exponent = my period!! For sure, here it is stated:

A polynomial $P(X)$ is said to belong to an exponent e if e is the least positive integer such that $P(X)$ evenly divides $X^e-1(=X^e+1 \bmod 2)$.

Theorem 1: If g(x) has more than one term, all single errors will be detected. Also, all odd numbers of errors. I think this is the same as parity.

Theorem 3: A code generated by the polynomial $P(X)$ detects all single and double errors if the length n of the code is no greater than the exponent e to which $P(X)$ belongs.

Example: If g(x) is a prim poly for GF(2^{16}) which has q = 2^{16} = 65,536, it has degree n-k = 16 and period = exponent = q-1 = 65535. Code words have n = q-1 = 2^{16}-1 = 65535 symbols. So this is indeed a pretty long packet and we get single and double detection!

Theorem 4: A code generated by $P(X)=(1+X)P_1(X)$ detects all single, double, and triple errors if the length n of the code is no greater than the exponent e to which $P_1(X)$ belongs.

My comment: Such a P(X) cannot be a primitive polynomial since 1 is one of the roots.

Theorem 5: Any cyclic code generated by a polynomial of degree $n-k$ detects any burst-error of length $n-k$ or less.

Theorem 6: The fraction of bursts of length $b>n-k$ that are undetected is

$2^{-(n-k)}$ if $b > n-k+1$, $2^{-(n-k-1)}$ if $b = n-k+1$.

Here is the most impressive claim, since it makes no requirement of the length of the code. I think this is the benefit you get over parity. Notice the definition of burst as just the range between first and last error. This is a theorem I could quote, and I should add this paper as a reference.

Detection of a Burst-Error

A burst-error of length b will be defined as any pattern of errors for which the number of symbols between the first and last errors, including these errors, is b.

Example:

$$\text{The } E(X) = X^3 + X^6 + X^7$$
$$= 0\ 0\ 0\ 1\ 0\ 0\ 1\ 1\ 0\ 0\ 0\ 0\ 0\ 0\ 0$$

of the previous example is a burst of length 5.

Theorem 5: Any cyclic code generated by a polynomial of degree $n-k$ detects any burst-error of length $n-k$ or less.

Proof: Clearly, any burst-error polynomial can be

OK, I think I have extracted everything useful from this original paper, it took a while, so I now want to look at a more modern source.

I have found another good reference but don't have it yet. Numerical recipes third edition.
Numerical Recipes: The Art of Scientific Computing (3rd ed.). New York: Cambridge University Press. ISBN 978-0-521-88068-8.

But first lets look at **another source** crc.pdf which I have:

- If G contains two or more terms, all single-bit errors are detected.
- If G is not divisible by x (that is, if the last term is 1), and e is the least positive integer such that G evenly divides $x^e + 1$, then all double errors that are within a frame of e bits are detected. A particularly good polynomial in this respect is $x^{15} + x^{14} + 1$, for which $e = 32767$.
- If $x + 1$ is a factor of G, all errors consisting of an odd number of bits are detected.
- An r-bit CRC checksum detects all burst errors of length $\leq r$. (A burst error of length r is a string of r bits in which the first and last are in error, and the intermediate $r - 2$ bits may or may not be in error.)

The first items reassert single and double symbol errors detected (but called bits not symbols).
The third item excludes G from being a prim poly, but gets all odd bit errors detected. So nothing really new here compared to the original paper.

Realization: Assume GF(p^m) is used, and in these cases GF(2^m). The period max is q-1 = 2^m - 1 and this is the size of the code packet protected by the checksum. Larger m means longer protected packet!

My CRC paper quotes some standards

TABLE 14–1. GENERATOR POLYNOMIALS OF SOME CRC CODES

Common Name	r	Generator	
		Polynomial	Hex
CRC-12	12	$x^{12} + x^{11} + x^3 + x^2 + x + 1$	80F
CRC-16	16	$x^{16} + x^{15} + x^2 + 1$	8005
CRC-CCITT	16	$x^{16} + x^{12} + x^5 + 1$	1021
CRC-32	32	$x^{32} + x^{26} + x^{23} + x^{22} + x^{16} + x^{12} + x^{11} + x^{10} + x^8 + x^7 + x^5 + x^4 + x^2 + x + 1$	04C11DB7

and says the first three polys have (x-1) as a factor, which to me means they are not primitive. They do this to be able to detect all odd-bit errors. Perhaps the last one is primitive. Let's see if it is in the P&W list. First, here is their limited data on m = 32,

```
DEGREE 32
          5 50521021747B
         11 40035532523F
         17 44165166133B
         51 73274317525C
        257 52213142567B
       3855 47254550703B
      21845 65636126613D
          1 40020000007F
          7 40460216667F
         13 42003247143F
         19 41760427607F
         85 55255004227B
        771 46633742135A
       4369 45052437233B
      65535 57410204175A
          3 40001114005A
          9 40220536125A
         15 42644424505A
         21 56032357221A
        255 60537314115C
       1285 53046115123B
      13107 71265756301C
```

where recall that EFGH are primitive.

Fact: The Hex notation for the CRC=32 case does not include the leading power! Horrors. here is why

The IEEE 802.3 standard adopts the CRC polynomial:
$x^{32}+x^{26}+x^{23}+x^{22}+x^{16}+x^{12}+x^{11}+x^{10}+x^8+x^7+x^5+x^4+x^2+x+1$
(this is irreducible, but not primitive). We represent this polynomial as a 32-bit hexadecimal number 0x82608EDB. The leading "8" of this number corresponds to the top four (x^{32} through x^{29}) polynomial coefficients, with lower order bits corresponding to lower order coefficients, down to the trailing "B," which refers to the terms (x^4+x^2+x). The "+1" term is implicit in this representation, permitting representing a polynomial of degree 32 using a 32-bit integer as is common practice in software CRC implementations.

So it takes me a lot of work to fix this up and then compare it to the P&W table. But I see what I wanted to know, it is irreducible but not primitive!

```
convert(`04C11DB7`,decimal,hex):convert(%,octal,decimal);
                                  460216667
```

OK, I have run out of steam on the above path. But I think I know enough now about the error detection ability of a CRC scheme and I can alter Galois accordingly.

Part II : Correcting confusing comments about n,k and cyclicness.

This is a separate topic, also confusing. My "scream out" comment is (8.29), very hard to find. But hold n that for moment and go back to MY definition of a cyclic code. I say to start by picking an integer n and then "find" some g(x) that divides into $x^n - 1$.

Stop right there, I have lots of questions already.

Question 1. Can you find such a g(x) of *any* degree you want that is less than n ?

The answer is no. Even for GF(2) the answer is no, but in general there are many degrees of g(x) that do work. See counter examples below etc etc.

I interpret this question to say " The polynomial x^n-1 lies in my polynomial space R with coeffs in GF(p). Can you find a g(x) for any degree you want where g(x) lies in R and g(x) divides $x^n - 1$? "

In general I don't know how to answer that question. But if it happens that $n = q-1 = p^m - 1$ for prime p and positive integer m, then I know one possible answer: any g(x) which is a minimum polynomial of GF(q) works. This is because such a g(x) contains a subset of the factors of $x^n- 1$ and is in R. In this specialized case, I don't think g(x) exists for *any* degree < n. For example, for GF(2^5) all min polys have

degree 5. So if you choose n = 2^5-1 = 31, then the only possible g(x) have degree 5 ! In the case GF(2^4) I tried all possible g(x) and classified each of them A,B,C,D,E. I found 4 min polys. Three had degree 4 and one had degree 3. Here n = 2^4-1 = 15. So for x^{15}- 1 I think the only g(x) in R that divide evenly are these min polys. But that is wrong as next paragraph shows!

I guess I could just try all possible polys and see if this is true. I don't really have a proof. I just did this for GF(2) case that n = 2^4 - 1 = 15. I divide (x^{15}-1) by all possible polys in R of degree 15 and less, and I do in fact find a g(x) that divides evenly for every single degree ≤15. I am surprised. All the successful g(x) have a "+1" which I guess makes sense since x does not divide x^{15}-1 .

```
for N from 1 to 2^16-1 do
  # compute polynomial for this value of N
  c := convert(N,base,2):
  L := nops(c):
  p := sum(c[i]*x^(i-1),i=1..L):
  # see if it has a remainder
  r := Rem(x^15-1,p,x) mod 2;
    # f := Factor(p,a) mod 2:
  if type(r,numeric) and r = 0 then print(p, r) fi;
  # print(p,r);
od:
```

$$1, 0$$
$$1+x, 0$$
$$x^2+x+1, 0$$
$$x^3+1, 0$$
$$1+x+x^4, 0$$
$$x^4+x^3+1, 0$$
$$1+x+x^2+x^3+x^4, 0$$
$$x^5+1, 0$$
$$x^5+x^3+x+1, 0$$
$$x^5+x^4+x^2+1, 0$$
$$1+x+x^2+x^3+x^6, 0$$
$$x^6+x^4+x^3+x^2+1, 0$$
$$1+x^3+x^4+x^5+x^6, 0$$
$$1+x+x^3+x^7, 0$$
$$1+x^4+x^6+x^7, 0$$
$$1+x+x^2+x^5+x^6+x^7, 0$$
$$1+x+x^2+x^4+x^8, 0$$
$$1+x+x^3+x^4+x^5+x^7+x^8, 0$$
$$1+x+x^3+x^4+x^5+x^7+x^8, 0$$
$$1+x^4+x^6+x^7+x^8, 0$$
$$1+x+x^4+x^5+x^6+x^9, 0$$
$$1+x^2+x^3+x^6+x^7+x^9, 0$$
$$1+x^3+x^4+x^5+x^8+x^9, 0$$
$$1+x^5+x^{10}, 0$$
$$1+x+x^2+x^4+x^5+x^8+x^{10}, 0$$
$$1+x^2+x^5+x^6+x^8+x^9+x^{10}, 0$$
$$1+x+x^2+x^3+x^5+x^7+x^8+x^{11}, 0$$
$$1+x+x^5+x^6+x^{10}+x^{11}, 0$$
$$1+x^3+x^4+x^6+x^8+x^9+x^{10}+x^{11}, 0$$
$$1+x^3+x^6+x^9+x^{12}, 0$$
$$1+x+x^3+x^4+x^6+x^7+x^9+x^{10}+x^{12}+x^{13}, 0$$
$$1+x+x^2+x^3+x^4+x^5+x^6+x^7+x^8+x^9+x^{10}+x^{11}+x^{12}+x^{13}+x^{14}, 0$$
$$x^{15}+1, 0$$

Some degrees have several candidate g(x). I tried the same program for $x^{14}-1$ and get the same kind of result.

BUT, it does not work in the general case. Here is a **counterexample**:

```
n := 5;
for N from 1 to 2^(n+1)-1 do   # go only as far as needed
 # compute polynomial for this value of N
  c := convert(N,base,2):
  p := sum(c[i]*x^(i-1),i=1..nops(c)):
# see if it has a remainder
  r := Rem(x^n-1,p,x) mod 2;
  if type(r,numeric) and r = 0 then print("N=",N,"degree=",nops(c)-1,"p(x)=",p, r) fi;
od:
```

$$n := 5$$

$$\text{"N=", 1, "degree=", 0, "p(x)=", 1, 0}$$

$$\text{"N=", 3, "degree=", 1, "p(x)=", } 1+x\text{, 0}$$

$$\text{"N=", 31, "degree=", 4, "p(x)=", } 1+x+x^2+x^3+x^4\text{, 0}$$

$$\text{"N=", 33, "degree=", 5, "p(x)=", } x^5+1\text{, 0}$$

You see that degree = 2 and 3 are missing, there is no g(x) that works for these degrees of g(x).

I first made sure that the code really does sequence through all possibly polys in ever increasing degrees. This example shows that (x^5-1) cannot be divided by any g(x) of degree 2 or 3 ! I did this by hand and confirmed the result. I tried to find

$$(x^2+bx+1)(x^3+\beta x^2+\delta x+1) = (x^5-1)$$

You get a set of equations in variables b,β,δ which cannot be solved. You get b + b + 1 = 0 which has **no solution** for b, for example. So I am sure this is a true fact.

There are lots of **counterexamples**: Not all are prime, not all are even.

n = 5	no degree 2 or 3
n = 7	no degree 2 or 5
n = 9	no degree 4 or 5
n = 10	no degree 3 or 7
n = 14	OK, all degrees present
n = 15	OK, all degrees present

So it just happened that the first two cases I tried worked.

Idea: Suppose we want g(x) to divide x^n-1. Suppose g(x) factors into g_1g_2 because it is reducible. Then we want (g_1g_2) to divide x^n-1. That requires that each of g_1 and g_2 divide x^n-1 !!!

$(x^n-1)/(g_1g_1) = h \quad \Rightarrow \quad (x^n-1)/g_1 = g_2h$ so g_1 must divide x_n-1.

Example: If $g = xg_2$, then x must divide x^n-1, but it clearly does not, so this form can never work, and that is why there is always a "+1" in the above.

Conclusion: I think that for any n, you can find one or more g(x) in R for each degree ≤n that divide x^n-1 [wrong!] . If you restrict to the case that $n = 2^m - 1$, *some* of these g(x) will be min polys of GF(2^m). The above example had m = 4 since I picked n = 15. Here are the min polys for GF(2^4):

$$\begin{aligned}
p_1(x) &= (x-\alpha)(x-\alpha^2)(x-\alpha^4)(x-\alpha^8) && = x^4 + x + 1 && 10011 \\
p_7(x) &= (x-\alpha^7)(x-\alpha^{14})(x-\alpha^{13})(x-\alpha^{11}) && = x^4 + x^3 + 1 && 11001 \\
m_3(x) &= (x-\alpha^3)(x-\alpha^6)(x-\alpha^{12})(x-\alpha^9) && = x^4 + x^3 + x^2 + x + 1 && 11111 \\
m_5(x) &= (x-\alpha^5)(x-\alpha^{10}) && = x^2 + x + 1 && 111
\end{aligned} \qquad (6.21)$$

and yes, they all appear on the above list.

Resume now: So I first pick an integer n as in (8.2). I now know I can next pick some k < n, any value I want. There will be some g(x) of degree n-k that divides $x^n - 1$. [This is true, but only after I did a lot of work elsewhere!!] So I can make a cyclic code for any (n,k) pair that I want! I am still working inside GF(2) of course, that is my R world [true for GF(p)]. So starting off in Chapter 8, I really have no Galois connection at all except GF(2) is a GF. I can talk about irreducible with respect to GF(2), that is true. But there is no min poly connection, nothing like that.

This is the context of things when I have my little CRC subsection. So given this context, I can probably pick any values of n and k and m = n-k that I want. If I pick n-k = 16, then I can pick any k that I want, and have n = k+16. So my comment seems way off base. For any n I can even find a g(x) of any degree m ≤n which divides $x^n - 1$, so my CRC g(x) can always then be cyclic.

More questions:

Question 2: Do the theorems of the **CRC Peterson paper** require that code by cyclic?

I suspect yes. Let's see how he defines his stuff. On his page 228,229 nothing about x^n-1. Key point: if the received codeword cannot be divided by g(x) due to error, *then neither can the error codeword*. So if you have a 1 bit error, the error codeword is just x^i and certainly you cannot divide this by any g(x)! That is why it detects single bit errors! Good.

OK, I understand better the opening discussion:

Theorem 2: Every polynomial divisible by $1+X$ has an even number of terms.

Proof: Let $F(X)=X^a+X^b+X^c+\cdots=(1+X)Q(X)$. Substituting $X=1$ gives

$$F(1) = 1 + 1 + 1 + \cdots = (1 + 1)Q(1) = 0.$$

There is one "1" in $F(1)$ for each term, and since the sum is zero, there must be an even number of terms. Q.E.D.

It follows that the code generated by $P(X)=1+X$ detects not only any single error, but also any odd number of errors. In fact, the check symbol must simply be an over-all parity check, chosen to make the number of ones in the code polynomial even.

Any polynomial of the form $1+X^e$ contains a factor $1+X$ since $1+X^e=(1+X)(X^{e-1}+X^{e-2}+\cdots+1)$. Therefore, if $P(X)$ contains a factor $1+X^e$, any odd number of errors will be detected.

Now he continues

Double and Triple Error Detecting Codes (Hamming Codes)

A polynomial $P(X)$ is said to belong to an exponent e if e is the least positive integer such that $P(X)$ evenly divides $X^e-1(=X^e+1 \bmod 2)$.

If you pick an arbitrary P(x) = g(x), must it have an exponent e ? He is quiet on this.

His double correction comment also seems good.

It can be shown that for any m there exists at least one polynomial $P(X)$ of degree m that belongs to $e=2^m-1$. This is the maximum possible value of e. Polynomials with this property (usually called primitive polynomials) are always irreducible. A few such polynomials are listed in Appendix II, and more extensive tables are available.[12,16] Thus for any m there is a double-error detecting code of length $n=2^m-1$ generated by a polynomial $P(X)$ of degree m, which therefore, has m check symbols and 2^m-1-m information symbols. These codes can be shown to be completely equivalent to Hamming single-error correcting codes.[2,3,12,17]

OK, now he is more explicit concerning exponent e. We are no longer talking about a general e, we are talking about e having the form e = q-1 = 2^m -1 and we are dealing now with GF(2^m). For such a value of e, I know that I can find a prim poly g(x) and its period will be q-1 so that will be the lowest e. So for code words not too long, you get double-bit error detection. He really is setting n = 2^m - 1 = e, so this is the same as my n in my cyclic code definition

Conclusion: You must have some exponent for your generator, or Theorem 3 does not fly.

Question 3: Is it possible that some given g(x) won't divide x^e-1 no matter how large e is allowed to be?

I don't know. But if it *is* possible, then we might say e = ∞. In this case, look at the Theorem 3. The proof is still flawless and it seems we can have n then be any size we want (code word length). Pete is interested in those prim polys because we get at least some value for e. I suspect there is always some finite e for the GF(2) case.

Idea: Suppose we start with some n for our x^n - 1 and we define everything over GF(p). We can then start running through all the extension fields GF(p^m) for m = 1,2,3....∞. For each m, we have q-1 = p^m-1 being the largest possible value for which a min poly divides x^e - 1, that is to say, q-1 is the largest possible value of order = period of a min poly. So here is a question:

Question: For GF(p^m), is there at least one element of each order from the set divisors(q-1) ? [yes!]

Suppose this were true. Then we systematically try m = 1,2,3... and for each m, we list off the divisors of q-1 and then we know that for each such divisor we have an α of that order and thus a min poly of that period = order. We do this until we eventually hit the value n that we started with.

1. For GF(2^3) I find

$$\{ \alpha, \alpha^2, \alpha^4 \}, \{ \alpha^3, \alpha^6, \alpha^5 \}, \{1\} \qquad (5.23)$$
prim prim

In this example, q = 2^3 = 8 and so q-1 = 7. Both polys are order 7, there are no divisors, boring.

2. For GF(2^4) $\alpha^{15} = 1$ and we get q-1 = 15 and

$$\{ \alpha, \alpha^2, \alpha^4, \alpha^8 \}, \{ \alpha^3, \alpha^6, \alpha^{12}, \alpha^9 \}, \{ \alpha^5, \alpha^{10} \}, \{ \alpha^7, \alpha^{14}, \alpha^{13}, \alpha^{11} \}, \{1\} \qquad (5.26)$$
prim prim
order = 15 order = 5 order = 3 order = 15

All these min polys divide x^{15}- 1. Do I know the order of each conjugate set's elements? The size of the sets are associated with the degree of the min poly, not the order. For each set, I have to do this test:

test first element:	is $(\alpha^3)^5 = 1$?	Yes, so 5 is a candidate order of α^3
	is $(\alpha^3)^3 = 1$?	No, since $\alpha^9 \neq 1$ so 3 not a candidate
	Therefore order $\alpha^3 = 5$.	

You have to try out each divisor d of q-1 and test to see if $(\alpha^b)^d = 1$, and pick the smallest d. The statement $(\alpha^b)^d = 1$ means bd = multiple of 15.

In the above example we DO have min pols of order 3, 5 and 15. Well consider

```
p := 2;
                                p := 2
for m from 1 to 10 do divisors(p^m-1)od;
                                {1}
                                {1, 3}
                                {1, 7}
                             {1, 3, 5, 15}
                                {1, 31}
                           {1, 3, 7, 9, 21, 63}
                                {1, 127}
                        {1, 3, 5, 15, 17, 51, 85, 255}
                             {1, 7, 73, 511}
                     {1, 3, 11, 31, 33, 93, 1023, 341}
```

We can never hit an even integer since q = 2^p = even and q-1 = odd. So for example, we are never going to find a min poly that divides x^4 - 1. So n = 4 has no g(x). But for other values of p, this conclusion does not follow. I might comment on this somewhere.

Are all odd values hit? I think eventually they will be, but how would you prove that? I wrote a little test program for p = 2 and seems true.

In general I think p^m- 1 can never be a multiple of p. Yes, this is special case of my lemma

Lemma: Show that GCD(p^i, (p^m-1)/K) = 1 for i = 0,1....m (note that m ≥ i) . (5.35a)

GCD(p, (p^m-1)) = 1

This says that (p^m-1) is not a multiple of p. That means that (p^m-1) cannot have p in its set of divisors. That means that no multiple of p is in the set of divisors. Ie, if A/p ≠ int, then A/np ≠ = (1/n)A/p ≠ int. This is born out for p = 3 by my same code.

Fact: When you select your value of n, n cannot be a multiple of p. The reason is that any multiple of p is never in the set divisors(p^m-1) for any m, so you can never have an α which has an order which is a multiple of p, and thus you can never have period p for a min poly. I should get this stated correctly and then add it somewhere. Probably every other value of n is OK.

Conclusion: If we pick any n not a multiple of p, then our plan of scanning all the GF(p^m) for m = 1,2,... will eventually produce a min poly which divides x^n-1. We will eventually find among the divisors of p^m-1 then umber n and for the winning m, that means that n is a possible value for the order of an element of that GF. If we knew that a GF has at least one element of each possible order, then we have it. But I have not shown that part yet.

Idea on showing that in any GF(q) there is an element whose order is each divisor of q-1

Form the set of divisors(q-1). We know order(α^k) = (q-1)/gcd(k,q-1). For each divisor element d, try to find a k that satisfies this equation. I think I can make this fly.

d = (q-1)/I = (q-1)/ gcd(k,q-1) => I = gcd(k,q-1)

So for a given q-1, and a given divisor I, is there always a value of k in the set 1..q-2 so I = gcd(k,q-1) ? Could write that as I = gcd(k,dI). I think this needs a Maple run before I do more. Maple suggests that k = I is always a solution. So is this true

I = gcd(I, q-1) where I is a divisor of q-1 ?

We know that I divides q-1 and I divides I, and no *greater* integer could divide I do no greater integer could divide both, so I is in fact the gcd, simple!

Claim: For any divisor I in the set divisors(q-1) there exists an element of GF(q) whose order is I.

Proof: We know from (4.21) that [order(α^k)] = (q-1)/gcd(k,q-1). Suppose I is a divisor of q-1. Then we can write IM= q-1 where M is the other factor. M then also divides q-1. We know that

[order(α^M)] = (q-1)/gcd(M,q-1)

But we claim that gcd(M,q-1) = M. The reason is that M is a candidate GCD since it divides M and q-1, and no larger integer can divide M, so M is it. Thus we have shown that [order(α^M)] = (q-1)/M = I. Thus we have found an element of GF(q), namely α^M , which has order I, for any I in divisors(q-1). Therefore, the conjugate set of α^M has order I and corresponds to a min poly of order I which has period I!

Conclusion: For every integer I in divisors(q-1), there exists at least one min poly with period I.

I think I have a solid proof of this claim.

The subgroup is what has order n, and this has to divide q-1. But who says we have a subgroup for each candidate order?

So I come back to the other claim: // stop

Question 4: Using my definition, what bad thing perhaps happen if n is not the period of g(x)? Do you still get a cyclic code? We could compare $n_1$and n_2 . [this is now well answered]

Question 5: Pete talks about code words of length n ≤ e, whereas I seem to assign n to be e. For the prim poly situation, we do know that e = q-1. Otherwise we are not really sure how low e might be.
[this is now explained in the CRC section and yes, n < e is needed for double-bit errors.]

I think question 4 is the only one I need to look at. I think it causes code words to have repeated sections or something like that, which is probably OK. I could add a comment that if n is the period, then we get exhaustion. But does that proof work without regard to Galois theory?