Introduction to error-correcting codes.

Maksim Maydanskiy

Summer 2018.

Contents

1 The basics.

1.1 Messages and codewords.

Anyone who has played a game of whispers[1] knows that messages can get garbled in transmission. That's fun in games, but troublesome and sometimes dangerous in communications and computing systems. To counteract, designers of such systems often use *error-correcting codes* – ways of pre-processing data before transmission so that the recipient can recover the original message even if some of it has been corrupted.

Example 1 (Repetition code, version 1). Suppose in the game of whispers you say every word of the message three times. Then even if one of the words gets corrupted, the listener can use "majority vote" to figure out what you said.

To make a mathematical model of this problem, we imagine a set of messages M. We often think of a message as a string of characters drawn from some *alphabet* Σ (its elements are called characters), and take as messages all strings of length k; that is, we pick $m \in M$ to be an ordered k-tuple of elements of Σ.

Remark 1. In the real world, different messages might be more or less likely: If you are expecting a message from a friend "Hello!" is a more likely message than, say, "fsTZYUXQ". Moreover, your transmission channel may transmit some messages more readily than others (for example, in Morse code a dash is the length of 3 dots). You may get an idea to use pre-encoding so that the source messages that are more common correspond to shorter transmitted messages, so that on average your messages go through faster (this is in fact done in Morse code). This is an extremely fruitful idea, developed in Part 1 of famous paper by Claude Shannon "A Mathematical Theory of Communication". This is not, however, directly about error-correction, and we will not deal with it. We will simply assume that whatever necessary pre-processing is done separately, and in our analysis we don't rely on any statistical knowledge about how likely various messages are (essentially thinking that all of them are now equally likely).

Then, to transmit the message, the sender encodes it – maps it to a (generally longer) sting of characters, and then sends that. For the recipient to have any hope of identifying the original message - even without any corruption in transmission - the encoding should be one to one; this means that we can often identify the image of the encoding (set of all transmitted strings) with the message itself. We are led to the following definitions.

Defintion 1. (Encoding function.) An *encoding function* is an injective map $E : M \to \Sigma^n$.

Defintion 2. (Code.) A *code* $\mathcal{C}$ of *block length* n over *alphabet* Σ is a subset of Σ^n. The elements of the code are called *codewords*.

Thus the image of an encoding function is a code.

1.2 Decoding. Hamming noise model and Hamming distance. Hamming bound.

After encoding the message $m \in M$, the sender transmits the resulting codeword $c = E(m)$. Transmission introduces some errors, and the recipient receives some other n-string s. Recipient's task, then, is to recover $c = E(m) \in \mathcal{C}$ and, ultimately, m from s.

[1] Also known as "telephone" in many countries.

$$m \xrightarrow{E} c \xrightarrow{\text{noise}} s$$

Remark 2. Recovering m from $E(m) \in \mathcal{C}$ is the task of inverting the injective function E on its image. For the same code $\mathcal{C}$ this may be more or less difficult, depending on E (this is the whole point of cryptography!), but for error-correcting codes is usually a lesser concern than recovering c from the received message s. We mostly focus on this latter task.

The task of "recovering c from s" is not well posed at this point, because we do not know in what way the transmission channel introduces noise. Clearly if the channel completely ignores the input and just outputs whatever it wants, recovering c will not be possible. Thus we must postulate some kind of *noise model*. One possible option is a probabilistic model – for any codeword c and any possible received string s we specify probability $P(s|c)$ that c will be transmitted as s. "Recovering c from r" becomes a question of finding the code word c with maximal probability $P(c|s)$ that it was the original codeword. This model is introduced in Part 2 of the same famous paper of Shannon (it is famous for a reason!); apart from one optional example right below to illustrate this model's connection with our setup, we shall also not deal with it. We will instead take another option, introduced by Richard Hamming in *his* famous paper "Error detecting and error correcting codes", and will say that the cannel introduces noise by corrupting some bounded number t of entries in the n-tuple. We then ask for an decoding procedure that recovers original c from this corrupted s. The ratio of number of possible errors t from which the code can recover to the total number of transmitted symbols n is called *the relative distance* of the error correcting code.

Remark 3. The two noise models are not completely disconnected, of course. We give one of the simpler illustrations of this connection (which is already a bit involved, and is optional). Here we are considering so-called binary symmetric channel, where we are transmitting bits (that is, the alphabet is $\Sigma = \{0, 1\}$; thus "binary") and the probabilities of erroneously receiving 1 instead of 0 and of erroneously receiving 0 instead of 1 are the same (hence "symmetric"). We want to say that the probability of getting too many errors is not very large. More precisely we have:

Theorem. Suppose the probability of error in every bit is $p < 1/2$ and errors in different bits are independent. Then when sending a message of length n, for any $d > pn$, the probability that we get more than $d = (p + \epsilon)n$ errors occurring is less than $e^{-\epsilon^2 n/2}$.

Since $\lim_{n\to\infty} e^{-\epsilon^2 n/2} = 0$, if we have a family of codes with relative distance $\delta = p + \epsilon > p$ we can use it to communicate over a channel with random noise with arbitrarily low risk of error.

Plausibility arguments for this Theorem. Level 0: The expected number of errors is pn, so getting many more errors than that is unlikely.

Level 1: To get a crude estimate how unlikely it is, we recall that the number of errors follows binomial distribution; normal approximation to binomial (which is an instance of the central limit theorem) is $N(\mu = pn, \sigma^2 = n(p(1-p))$ and being at more than d errors is $d - pn = \epsilon n = \frac{\epsilon\sqrt{n}}{\sqrt{p(1-p)}}\sigma$ above the average, which does not happen with probability of more than $e^{\left(\frac{\epsilon\sqrt{n}}{\sqrt{p(1-p)}}\right)^2/2} = e^{-\frac{\epsilon^2 n}{2p(1-p)}}$. This is off by a factor of $p(1-p)$ in the exponent (and is based on an approximation in any case), but give the right exponential dependency on n and ϵ.

Level 2: The actual proof involves the Chernoff bound in probability. We will not go there. □

Example 2 (Repetition code, version 2). Suppose in our transmission we send every letter of the message three times. Then even if one of the letters gets corrupted, the receipient can use "majority vote" to figure out what the original message was. What happens if you send every letter 4 times? What about 5 times? How many corruptions can you correct?

Mathematically, error-correcting abilities of codes in Hamming model are best described using the notion of Hamming distance.

Defintion 3. (Hamming distance) Given two strings over the same alphabet define the distance to be the number of places where they do not agree. More precisely, if $c = (c_1, c_2, \ldots, c_n)$ and $s = (s_1, s_2, \ldots, s_n)$ then

$$d(c, s) = |\{i \in \{1, \ldots, n\} | c_i \neq s_i\}|.$$

You can think of $d(c, s)$ as the minimal number of "character switching moves" to get from c to s, where a single move is changing one letter.

Exercise 1. Show that d is a *distance metric* on the set of all strings of length n. That is show that:

1. $d(a, b) \geq 0$
2. $d(a, b) = 0 \implies a = b$
3. $d(a, b) = d(b, a)$
4. $d(a, c) \leq d(a, b) + d(b, c)$

Hint. The only non-trivial part is the last one (called the triangle inequality); however for functions defined as "minimal number of steps to get from a to c" the triangle inequality is always true – you can always get to c by first going to b and then continuing to c.

□

Thus in Hamming error model (which we adapt from now on and will stop mentioning by name) we fix the maximal number of errors t and the received message is any message of Hamming distance at most t from the sent code message c.

To be able to uniquely recover c from s, we must have that c is the unique code word at distance t or less from s; this is saying that the (closed) balls of radius t around points of $\mathcal{C}$ do not overlap.

Defintion 4. A code $\mathcal{C} \subset \Sigma^n$ is *t-error correcting* if for any $r \in \Sigma^n$ there is at most one $c \in \mathcal{C}$ with $d(c, s) \leq t$.

Note that this is the same as all the closed balls of radius t centered at codewords being disjoint.

Clearly, there can not be too many disjoint balls in the finite set Σ^n. With view towards the future we denote the size of Σ by q. To see how many can fit at most, we count the points in Σ^n (ok, that's easy: $|\Sigma|^n = q^n$), and in the Hamming balls.

Remark 4. Note that Hamming balls look quite different than "usual" balls. Imagine for a moment that $\Sigma = \mathbb{R}$ (this is not finite, but we ignore this for the moment). Then a closed ball of radius 1 through the origin is simply the union of coordinate lines, while a ball of radius 2 is the union of coordinate planes - etc.

Exercise 2. The number of strings in the closed ball of radius t around any string $s \in \Sigma^n$ is

$$Vol(t, n) = \sum_{j=0}^{t} \binom{n}{i} (q-1)^j$$

Remark 5. (Optional.)

For large n the dominant term in volume $Vol(pn, n)$ is $\binom{n}{np}$. The rough Stirling approximation is $\log_q n! \approx n \log_q n - n \log_q e$. Using this compute for large n, after some algebraic manipulations and cancellations we get:

$$\begin{aligned}\log_q Vol(pn, n) &\approx [n \log_q n - n \log_q e] - [n \log_q np - np \log_q e] \\ &\quad - [n(1-p) \log_q n(1-p) - n(1-p) \log_q e] + np \log_q (q-1) \\ &= n \left[-p \log_q p - (1-p) \log_q (1-p) + p \log_q (q-1) \right] =: n H_q(p)\end{aligned}$$

Here for $q = 2$ we recover $H_2(p) = H(p) = -p \log_2 p - (1-p) \log_2 (1-p)$ the Shannon entropy function, and for other values we get the so-called q-entropy.

We actually computed the volume of the "sphere" or radius pn (assuming this is an integer). The strings in this sphere are "typical" strings output by a source producing 1s with probability p, running for n steps - unless something very unlikely happen, the source will produce a string with number of ones very close to np. This means that the source is basically sending one of $2^{nH(p)}$ roughly equiprobable stings. Following Shannon, it is therefore reasonable to say that the amount of information transmitted (measured in bits) is the $\log_2$ number of possible messages $= \log_2 2^{nH(p)} = nH(p)$. Thus the information capacity of the channel is $H(p)$ bits of information per one transmitted bit. (End of the optional remark).

Since the balls have to be disjoint, we get the following bound on the size of the code $\mathcal{C}$:

Theorem 1 (Hamming bound)**.** *The size of any t-error correcting code $\mathcal{C}$ in Σ^n satisfies*

$$|\mathcal{C}| Vol(t, n) = |\mathcal{C}| \left[\sum_{i=0}^{t} \binom{n}{i} (q-1)^i \right] \leq q^n$$

The inequality in this theorem becomes equality precisely when the Hamming spheres around codewords of $\mathcal{C}$ cover the set of all strings on Σ^k completely. Such codes are called *perfect codes*.

Example 3. Consider repetition code with $\Sigma = \{0, 1\}$ (such codes are called "binary") and odd number of repetitions $n = 2t + 1$. There are two codewords (all-0 and all-1), and the Hamming balls or radius t around them cover all of $\Sigma = \{0, 1\}^n$ without overlapping. Thus this is a perfect code (you can also check algebraically that the Hamming bound holds – summing half of binomial coefficients you get 2^{2t}).
Together with the examples where $t = 0$ and $\mathcal{C} = \Sigma^n$ these codes are called "trivial perfect codes".

Error-correcting ability of a code $\mathcal{C}$ is best formulated in terms of the quantity called minimum distance.

Defintion 5. (Distance of a code.) For a code $\mathcal{C}$, define the it's *minimum distance* (or just *distance*) to be the smallest distance between two codewords in $\mathcal{C}$:

$$d_C = \min\{d(c, c') | c, c' \in \mathcal{C}, c \neq c'\}$$

Relation between this and error-correction ability is as follows:

Theorem 2 (Theorem 3.2.10)**.** *Code $\mathcal{C}$ can correct t errors if and only if $d_C \geq 2t+1$.*

Proof. We prove the contapositive: $d_C \leq 2t$ if and only if some two closed radius t balls centered at some c and c' in $\mathcal{C}$ intersect.

If $d_C \leq 2t$ then exist $c, c' \in \mathcal{C}$ with $d(c,c') \leq 2t$. Then there exists a path of no more than $2t$ coordinate switches from c to c'. At t steps down that path we find s with $d(s,c) \leq t, d(s,c') \leq t$.

Conversely, if $d(s,c) \leq t$ and $d(s,c') \leq t$ for some s and $c, c' \in \mathcal{C}$, then $d(c,c') \leq 2t$. □

1.3 Codes of distance $n-1$ and orthogonal arrays.

Fix q and n. For every d we can ask about the maximal size of a code on n characters from alphabet of size q with distance at least d. People denote this maximal size by $A_q(n,d)$.

Every code has $d \geq 1$, and maximal such is all of Σ^n, $A_q(n,d) = q^n$. We will soon see linear codes, which always achieve distance $d \geq 2$, and will allow us to say that $A_q(n,2) \geq q^{n-1}$. For certain n and q that are prime powers, there are perfect codes with $d = 3$ known as Hamming codes – we sill study them later.

Question 1. Investigate $A_q(n,d)$ for $d = 3$ (without requiring codes to be linear).

On the other end of the spectrum if $d = n$ there can be only one point in $\mathcal{C}$, so $A_q(n,n) = 1$. What about $A_q(n,n-1)$? Let's assume $n \geq 2$.

Theorem 3. *There exists a code in Σ^n of size S and distance $d = n-1$ if and only if there exist n mutually orthogonal arrays of size S (see "Combinatorics" Definition 4, Section 2.1).*

Before we prove the theorem, we state:

Corollary 1. *$A_q(n,n-1) \leq q^2$ and $A_q(n,n-1) = q^2$ if and only if there exists $n-2$ mutually orthogonal latin squares of size q.*

The corollary follows like this: 1) since we can not have a pair of orthogonal arrays of size more than q^2 ("Combinatorics" Proposition 2, Section 2.1), so $A_q(n,n-1) \leq q^2$ and 2) the condition is vacuous if $n = 2$, and $A_q(2,1) = q^2$ indeed, and in other cases having $n-2$ MOLS is of size q is equivalent to having n mutually orthogonal arrays of size q^2 as we saw before ("Combinatorics" Theorem 5, Section 2.1).

Proof of theorem. The proof below is best visualized by imagining a S by n table with all the codewords written out as rows and the arrays forming the columns.

$$\begin{array}{rcccc} & A_1 & A_2 & \dots & A_n \\ & \| & \| & & \| \\ c = & c_1 & c_2 & \dots & c_n \\ c^1 = & c_1^1 & c_2^1 & \dots & c_n^1 \\ & \vdots & \vdots & & \vdots \\ c^S = & c_1^S & c_2^S & \dots & c_n^S \end{array}$$

If we have n mutually orthogonal arrays $L_j(s)$ of size S then to every cell s we associate a codeword $c(s) = (L_1(s), \cdots, L_n(s))$. Any two such codewords $c(s)$ and $c(s')$ agreeing in two places j and j' would mean that the arrays L_j and $L_{j'}$ agree on two distinct cells, and so are not orthogonal. So Hamming distance between any two codewords is at least $n-1$.

Conversely, given a code of size S of $d_c = n-1$, with codewords $c^1, \ldots, c^s$ build n arrays $L_j(s)$ by assigning $L_j(s) = c^s_j$ (this is the reverse of the procedure in the previous paragraph). By the same logic as above, any two L_js being not mutually orthogonal would mean that two of the cs would agree in at two places, which is not possible. So we got our collection of mutually orthogonal arrays.

This completes the proof.

□

1.4 Singleton and Plotkin bounds.

We now present two more bounds on size $|\mathcal{C}|$ of error correcting code $\mathcal{C}$ in Σ^n with distance d.

The first simple bound after Hamming bound (Theorem 1) on the size of $\mathcal{C}$ is obtained by observing that if we forget $d_C - 1$ coordinates of a code word, we should still get distinct strings – any two codewords differ in at least d_C places, so they still differ if we remove any $d_C - 1$ coordinates. The conclusion is that the number of codewords is at most the number of strings of size $n - (d_C - 1)$.

Theorem 4 (Singleton bound)**.** *The size of any t-error correcting code $\mathcal{C}$ in Σ^n satisfies*

$$|\mathcal{C}| \leq q^{n-d_C+1}$$

We note that this is equivalent to $d_C \leq n - \log_q |\mathcal{C}| + 1$.

Remark 6. Note that if $d_c = 2$ this says $|\mathcal{C}| \leq q^{n-1}$. As we mentioned before, this is achievable by a "codimension 1 linear code" (see Section 2 just below), so $A_q(n, 2) = q^{n-1}$.

Remark 7. As we explain later when talking about linear codes, $\log_q |\mathcal{C}|$ is called the dimension of $\mathcal{C}$. This is "approximate number of characters k such that $|\Sigma^k| \approx |C|$". The number $\frac{k}{n}$ then called the *rate* of the code. There is in general a "rate-relative distance tradeoff", and the Singleton bound says basically "rate+ relative distance ≤ 1".

Finally, so-called Plotkin bound is as follows.

Theorem 5. *For any code $\mathcal{C}$ in Σ^n we have*

$$d_C \leq n \frac{|\mathcal{C}|}{|\mathcal{C}| - 1} \frac{q-1}{q}$$

Observe that, since $|\mathcal{C}|$ is the number of codewords and should be quite large, this implies that the relative distance of any code is below something quite close to $\frac{q-1}{q}$, and in particular for binary codes is below something close to $1/2$.

Proof. For the proof, we count in two ways the sum S of all distances $d(c, c')$ for (c, c') an ordered pair of distinct elements in $\mathcal{C}$.

On one hand, we count over all pairs (c, c'). Since for every such pair $d(c, c') \geq d_C$ and there are $|\mathcal{C}|(|\mathcal{C}| - 1)$ such ordered pairs, we get

$$S \geq |\mathcal{C}|(|\mathcal{C}| - 1)d_c$$

On the other hand, we count summing over all coordinates $i \in \{1, 2, \dots n\}$. For any symbol $\sigma \in \Sigma$ define $n_{i,\sigma}$ to be the number of codewords in $\mathcal{C}$ with $c_i = \sigma$. What is the total number (over all pairs (c, c')) of mismatches in the ith coordinate? Well, c has some symbol $c_i = \sigma$ there, and we must have $c'_i \neq \sigma$. For a fixed symbol σ there are $n_{i,\sigma}$ choices for c and $|C| - n_{i,\sigma}$ choices for c'. To get all contributions from ith coordinate, we sum over all $\sigma \in \Sigma$ – and then to get S we sum over all coordinates $i \in \{1, 2, \dots n\}$. Overall we get:

$$S = \sum_i \sum_\sigma n_{i,\sigma}(|\mathcal{C}| - n_{i,\sigma}) = \sum_i |\mathcal{C}||\mathcal{C}| - \sum_i \sum_\sigma n_{i,\sigma}^2 = n|\mathcal{C}|^2 - \sum_i \sum_\sigma n_{i,\sigma}^2$$

Here we used that $\sum_\sigma n_{i,\sigma} = |C|$ – every codeword has some symbol σ in the ith place. Finally, we bound the sum of squares by Cauchy Schwartz: $\sum_\sigma n_{i,\sigma}^2 \leq \frac{1}{q}(\sum_\sigma n_{i,\sigma})^2 = |C|^2/q$ to get overall:

$$S \leq n|\mathcal{C}|^2 - \frac{n}{q}|\mathcal{C}|^2$$

Putting the two inequalities for S together we get

$$|\mathcal{C}|(|\mathcal{C}| - 1)d_c \leq S \leq |\mathcal{C}|^2 n(1 - \frac{1}{q})$$

which gives what we wanted.

□

2 Linear codes.

2.1 Definitions.

Things are always better with linear algebra.

Defintion 6. (Linear code) A *linear code* is a vecor subspace if a vector space F_q^n.

That is, our alphabet is now $\Sigma = F_q$, and the code is a vector space.

Defintion 7. (Dimension of a code) The *dimension* of the linear code is it's dimension as a vector space.

Remark 8. A linear code of dimension k has q^k points, so that $\dim \mathcal{C} = \log_q |\mathcal{C}|$. For this reason the number $\log_q |\mathcal{C}|$ is called "dimension" even when $\mathcal{C}$ is non-linear.

Example 4 (Repetition code, version 3). If $\Sigma = F_q$ then the repetition code from before is a linear code.

The fact that the code is a vector space means that we can add, subtract, and rescale codewords, and that if c and c' are codewords, then so is $c - c'$ (and $c + c'$ and kc for any $k \in F_q$ etc.). This, and the fact that the Hamming distance is translation-invariant, immediately gives:

Proposition 1. *If $\mathcal{C}$ is a linear code then $d_C = \min\{d(0, c)|c \in \mathcal{C}\}$.*

Proof. $d(c, c') = d(0, c - c')$, so the minimum defining d_C and $\min\{d(0, c)|c \in \mathcal{C}\}$ coincide. □

Defintion 8. (Weight of a string) For a string s in Σ^n we define its weight $wt(s)$ to be its distance to the origin: $wt(s) = d(0, s)$. This is clearly the number of non-zero entries of s.

Immediately from this definition and Definition 5 we get:

Corollary 2. *In linear codes, d_C is the smallest weight of a codeword.*

Example 5 (Repetition code version 4). We now immediately see that for repetition code with r repeats $d_C = r$.

Remark 9. So far we have used no field properties of F_q. We could have worked with any group $\mathbb{Z}/n\mathbb{Z}$ as alphabet Σ. However, in a moment we will start solving linear equations and systems and will definitely use the field properties.

Here is another simple linear code:

Example 6 (Naive "parity check"). Given a message $m = (m_1, \ldots, m_k) \in M = F_q^k$ encode it to the codeword $c = E(m) \in F_q^{k+1}$ by computing the parity entry $m_{k+1} = \sum_{i=1}^{k} m_k$ and sending m to $(m_1, \ldots, m_k + 1)$. This code has $d_C = 2$. It can not correct an errors, but can detect a single error (or, more generally, any odd number of errors), by observing that the received message is not a codeword. Such "error detection" codes could be of some use, especially if one is allowed to ask the sender to retransmit a corrupted message, but we focus on error-correction.

2.2 Generator matrices. Parity check matrices.

Every vector subspace of vector space V can be defined as an image of a linear map into V, or as a zero set of a linear map from V. If such description is to be non-redundant, the maps into V should be injective, or, correspondingly the map from V should be surjective. In either case we say that the map is "of full rank". In the context of linear codes, where $V = \mathcal{C}$ we obtain the notions of generating and parity check matrices respectively, which are described below.

Remark 10. In the book we are following, and in most books I have seen on the subject of error correction, the convention is that vectors are by default *row vectors*, of size $1 \times n$, and linear maps are thus given by matrices multiplying ("acting") *on the right* thus $1 \times n \cdot n \times m = 1 \times m$. This is contrary to my personal habits, but I decided to try to follow this convention when talking about coding theory. If you get confused you can try to write out the sizes as above and see if the "same sizes go together". Or simply transpose everything, making vectors into column vectors and making matrices act by multiplying on the left. Good luck!

A $k \times n$ matrix G with entries in F_q encodes a map $F_q^k \to F_q^n$ sending m to mG. If G is of "full" rank k, this map is injective, and so is an encoding function in the sense of Definition 1.

Defintion 9 (Generator matrix). A $k \times n$ matrix G of rank k is called a *generator* of its image code

$$\mathcal{C} = image(G) = \{c | c = mG \text{ for some } m\}.$$

Dually, a $n \times (n-k)$ matrix H^T with entries in F_q encodes a map $F_q^n \to F_q^{n-k}$ sending c to cH^T. If H^T is of "full" rank $n-k$, the null space $\{c | cH^T = 0\}$ (also called kernel) of H^T is a linear code of dimension k. Observe that the same subspace is also the set $\{c | Hc^T = 0\}$.

Defintion 10 (Parity check matrix.). A $(n-k) \times n$ matrix H of rank $n-k$ is called a *parity matrix* of its kernel code

$$\mathcal{C} = kernel H = \{c | H^T c = 0\} = \{c | c^T H = 0\}.$$

Proposition 2. *Matrices H and G of dimensions as above and full rank are parity and generator matrices for the same code $\mathcal{C}$ if and only if*

$$GH^T = 0$$

Here the zero in this equality is the zero $k \times (n-k)$ matrix.

Proof. If $GH^T = 0$ then image of G is in the kernel of H^T, and since by our conditions on the rank they are both of dimension k, they must be equal.

Conversely, if $image G = kernel H^T$ then $GH^T = 0$. □

2.3 Columns of parity check matrix and error correction ability.

It is easy to characterize linear code's correction ability in terms of the parity check matrix.

Proposition 3. *If H is a parity check matrix of the code $\mathcal{C}$ then d_c is the cardinality of the minimal set of linearly dependent columns of H.*

Proof. If $\sum_i c_i \vec{h}_i = \vec{0}$ is a linear combination of columns of H equal to zero if and only if $c = (c_1, \ldots, c_n)$ is in $kernel H^T = \mathcal{C}$. The number of non zero c_i's is the weight of c and so the minimal weight of c in $\mathcal{C}$ is precisely the cardinality of the minimal set of linearly dependent columns of H. By Proposition 2 this is d_C. □

The size of the column $n-k$ is the number of "extra parity check symbols" that you append to your message, so all else being equal you might want to keep it low, and you want to make k large. Thus the game is to keep columns short and numerous, while keeping d_C large.

Example 7. To get $d_c = 2$ you just need to not put any zero vectors as columns. To keep $d_c = 2$ nothing prevents us from taking $n-k = 1$ and any n we want. If we just fill the one row of H with 1s we obtain the "minus parity check" code - the last entry is minus the sum of all the others. This is just as we saw in "Naive parity check" Example 6 - one extra digit is enough to ensure $d_c = 2$.

Example 8 (Hamming codes.). What do we need to get $d_c = 3$? We need to keep columns of H pairwise linearly independent. If we freeze the length $l = n-k$ of these columns, how many pairwise linearly independent column vectors can we get? We have already encountered this problem when constructing Latin squares. In any case, the answer is $n = \frac{q^l-1}{q-1}$, and the resulting code is of dimension $k = \frac{q^l-1}{q-1} - l$ is called the *Hamming code*.

The parity check matrix of Hamming code has particularly simple description when $q = 2$. In this case every "line" (1-d subspace) consists of the origin and just one extra point, and so there is only one choice of non-zero vector in that line. Such a vector is a "binary number", and so the columns of H are "all non-zero binary numbers". This is the description in Hamming's paper. For example, when $l = 3$ we get the matrix

$$H = \begin{pmatrix} 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 0 & 1 & 1 & 0 & 0 & 1 & 1 \\ 1 & 0 & 1 & 0 & 1 & 0 & 1 \end{pmatrix}$$

The corresponding code appears also as an example in Section 17 of Shannon's paper.

Proposition 4. *Hamming codes are perfect.*

Proof. You can of course check that the Hamming inequality in Theorem 1 is equality for Hamming codes. Here is an alternative explanation: every codeword in $\mathcal{C}$ encodes a linear relation between columns of H. Suppose we have any string s in F_q^n. If we interpret it as encoding a relation between columns of H this relation will be false precisely when $s \notin \mathcal{C}$. Instead of summing to zero, the resulting linear combination of columns will some to some non-zero vector e. This e is a multiple of unique column of H, $e = K\vec{h}_j$ for some constant k and column $\vec{h}_j$. If we want to move s to lie in $\mathcal{C}$ by a single Hamming move, we want to add to the relation encoded by s a multiple of one column of H to make it true. But this we can do in precisely unique way by adding $e = K\vec{h}_j$ – adding K to the component s_j of s. Thus every string not in $\mathcal{C}$ lies at distance 1 from exactly one codeword of $\mathcal{C}$, and so $\mathcal{C}$ is perfect. □

2.4 Parity and generator matrices in systematic form.

Perhaps the simplest type of a full rank matrix is the one that has identity matrix as a submatrix. We can get such a matrix for fixed $\mathcal{C}$ by Gaussian elimination, using only row operations for G (correspondingly, only column operations for H). Such forms of generator and parity matrices are called *systematic*.

Proposition 5. *If G and H are systematic generator and parity matrices for $\mathcal{C}$ then*

$$G = \begin{pmatrix} Id_k & P \end{pmatrix} \text{ and } H^T = \begin{pmatrix} -P \\ Id_{n-k} \end{pmatrix}$$

for some matrix P.

Proof. The systematic form dictates

$$G = \begin{pmatrix} Id_k & P \end{pmatrix} \text{ and } H^T = \begin{pmatrix} Q \\ Id_{n-k} \end{pmatrix},$$

and Proposition 2 dictates $Q = -P$.

□

Observe that if G is in systematic form this means that our encoding function simply starts with the message m and then appends some linear combinations of the entries in the end as "check-sums". In this case decoding a valid codeword c to the original message m is trivial – one simply keeps the first k characters of c.

2.5 Dual codes.

The fact that G and H^T play dual roles is formalized in the following proposition.

Proposition 6. *Let $\mathcal{C}$ be a k dimensional code in F_q^n given by a generator matrix G and a parity check matrix H. Then the $n-k$ dimensional codes $\mathcal{C}'$ and $\mathcal{C}''$ in F_q^n given, respectively, by generator matrix H and by parity check matrix G coincide.*

Proof. By Proposition 2 $\mathcal{C}' = \mathcal{C}''$ if and only if $HG^T = 0$. But of course that is true, since $HG^T = (GH^T)^T = 0^T$. □

From this it follows that $\mathcal{C}' = \mathcal{C}''$ actually depends only on $\mathcal{C}$, and not on the choice of G or H (since $\mathcal{C}'$ does not depend on choice of G and $\mathcal{C}''$ does not depend on the choice of H).

In fact we can give direct definition and see that both $\mathcal{C}'$ and $\mathcal{C}''$ coincide with it.

Defintion 11. Given a linear code $\mathcal{C}$, define the dual code $\mathcal{C}^\perp$ to be the vector subspace

$$\mathcal{C}^\perp = \{u \in F_q^n | uc^T = u \cdot c = 0 \text{ for all } c \in \mathcal{C}\}$$

Proposition 7. *For a fixed $\mathcal{C}$, we have $\mathcal{C}^\perp = \mathcal{C}' = \mathcal{C}''$*

Proof. We already know $\mathcal{C}' = \mathcal{C}''$.

We show $\mathcal{C}' \subset \mathcal{C}^\perp \subset \mathcal{C}''$, so the result follows.

Suppose $u \in \mathcal{C}'$. Then $u = mH$, and hence $uC^T = mHC^T = 0$. Hence u is orthogonal to all the rows of G, and thus to their span $\mathcal{C}$. Hence $u \in \mathcal{C}^\perp$.

Suppose $u \in \mathcal{C}^\perp$. Then since all rows of G are in $\mathcal{C}$, $Gu^T = 0$. So $u \in \mathcal{C}''$.

This completes the proof.

□

We conclude that just like in usual linear algebra we have:

Corollary 3. *If dimension of $\mathcal{C}$ is k then dimension of $\mathcal{C}^\perp$ is $n - k$.*

Corollary 4.

$$(\mathcal{C}^\perp)^\perp = C$$

The codes with $\mathcal{C}^\perp = \mathcal{C}$ are called *self-dual*.

Remark 11. Those a more familiar with abstract linear algebra we can also note that $\mathcal{C}^\perp$ can also be described the annihilator of $\mathcal{C}$ in the dual vector space $(F_q^n)^* = F_q^n$ (that is, the set of linear functions in $(F_q^n)^*$ that vanish in $\mathcal{C}$).

2.6 Hadamard codes.

Example 9 (Dual of binary Hamming code aka Hadamard codes.). Consider the code with generator matrix G whose columns are all binary digits of size up to l. This is a code of dimension l and size 2^l. It is easy to see that every row of G has weight 2^{l-1}. Once one thinks for a bit (hah!) about what G does to various basis vectors one concludes that it sends the message vector $\vec{m} = (m_1, \ldots, m_l)$ to the string of values of the linear form $(m_1x_1 + m_2x_2 + \ldots + m_lx_l)$ as $\vec{x} = (x_1, \ldots, x_l)$ varies in F_2^l. Exactly half of the values of this form (for any non-zero $\vec{m}$) are 1. So weight of every non-zero vector in $\mathcal{C}$ is 2^{l-1}. We have a $[2^l, l, 2^{l-1}]$ code.

In fact, the vectors in $\mathcal{C}$ are precisely the rows of the Hadamard matrix H_{2^n}, once we replace 1 by 0 and -1 by 1! (For this reason this is sometimes called the Hadamard code). You can prove this directly by induction, but the "more conceptual high-brow induction" is as follows: replacing 1 by 0 and -1 by 1 turns multiplication into addition in F_2. The original H_2 has rows recording values of the two linear functions $f(x) = x$ and $f(x) = 0$ on F_2. Then, inductively, rows of the tensor product of a pair of matrices correspond to pairs of rows of the original matrices. Via isomorphism $V_1^* \oplus V_2^* = (V_1 \oplus V_2)^*$ sending f_1 and f_2 to $f_1 \oplus f_2(v_1, v_2) = f_1(v_1) + f_2(v_2)$ we see that the tensor product of the matrix recording values of linear functions on V_1 and the matrix recording values of linear functions on V_2 is precisely the matrix recording values of linear functions on $V_1 \oplus V_2$. So

rows of H_{2^n} record values of linear functions on F_2^n in agreement with description above.

If we allow also the constant function $f(\vec{x}) = 1$ this amounts to allowing in addition to rows of H rows of $-H$ (taking x to $-x$ before switching $1, -1$ to $0, 1$ is the operation of taking x to $1 - x$ after this substitution). In vector space description this just means adding vector $(1, \ldots, 1)$ to the span. These rows still all have weight 2^{l-1}, so as far as keeping the same d_C is concerned, this extra bit is "free". We get a linear code with parameters $[2^l, l+1, 2^{l-1}]$, called punctured Hadamard code, or sometimes just Hadamard code. Having very high error-correction ratio, it is suitable for noisy or difficult environments. The $n = 8$ version of this code was used by NASA Mariner missions to Mars.

The (punctured) Hadamard code uses values of multivariable linear polynomials evaluated on the message words to produce a code. Using degree $\leq k$ polynomials one gets so-called Reed-Muller codes $RM(k, l)$ of which Hadamard codes are the spacial case $k = 1$. Similar idea of evaluating polynomials to produce codes will show up again in discussion of Reed-Salomon codes.

2.7 Error detection. Syndromes. Error correction with syndromes.

In this section we will talk about how one might actually perform error correction with a linear code. That is, having received a string s we want to determine the codeword at distance $\leq t$ from it (or reject the received message as having too many errors if no such codeword can be found). To begin, using H we can easily see if s is actually a codeword: we just compute Hs^T (or, equivalently, sH^T). The result is zero precisely when s is in $\mathcal{C}$. If the result is non-zero this is a "syndrome" of errors in transmission. In fact the vector Hs^T is called *the syndrome* of s.

Can this "syndrome" help us "diagnose" what errors were introduced? That is, given $s = c + e$, can we recover c (or, equivalently, e) from Hs^T?

Let's compute:

$$Hs^T = H(c+e)^T = Hc^T + He = 0 + He^T = He^T$$

Exercise 3. Show that if $\mathcal{C}$ is t-error correcting, then $wt(e) \leq t$ then e is the smallest (in terms of weight) among all e' with the same syndrome $He'^T = He^T$. Hint: Show that $(e - e') \in \mathcal{C}$.

So the task of recovering e from the syndrome is that of recovering the smallest element mapped to given $b = Hs^t$ value by a surjective linear map H.

Question 2. When the underlying field is $\mathbb{R}$ and the distance is the Euclidean distance, the solution to this problem is given by Moore-Penrose pseudoinverse, which has formula $H^T(HH^T)^{-1}$. What happens if one tries to use this formula here?

To error-correct an incoming message we can pre-compute syndromes of all errors e of weight at most t. These will al be different by the folowing proposition.

Proposition 8. *For all e and e' with $wt(e) \leq t$, $wt(e') \leq t$ we have $He^T \neq He'^T$.*

Proof. Otherwise $He^T = He'^T$, $H(e-e')^T = 0$, so $e - e' \in \mathcal{C}$ but $wt(e-e') \leq 2t < d_C$, contradiction. □

Then if a string s comes in, we compute Hs^T, compare it with all the syndromes of e's and if we have a match declare $c = s - e$. Otherwsie we decide that we have too many errors to correct confindently.

Remark 12. We could do a bit better, and for all possible syndromes (since H is surjective (being full rank) that means for all possible strings of length $n - k$) find an e with that syndrome and minimal weight (these are called "coset leaders" in the book). Then upon receiving an message with any syndrome we would decide that the error was e and decode to $c = s - e$. This would require finding and then storing all the coset leaders, whereas for the cosets containing e's of weight $\leq t$ those es *are* the coset leaders (this is what Exercise 3 says), and no searching for coset leaders is required.

Example 10 (Error correction in binary Hamming codes). The syndrome of weight 1 error is simply the corresponding column of the parity check matrix H. For binary Hamming code with columns listed "in order", the column in position j contains the binary digits of the number j. Thus for such codes the syndrome of a single bit error is the binary specification of the coordinate where the error occurred. Thats some nice syndrome, telling you exactly where the disease is! Hamming designed his code to have precisely this property.

In general, syndrome decoding is costly, so codes where better decoding methods are available have additional appeal.

3 Some code constructions.

3.1 Random linear codes and GV lower bound.

Suppose we wanted to construct a linear code with $d_c = d$ fixed. We could go about it in various ways. We could fix $l = n - k$ and try to pick as many columns for parity check matrix H as we can, while keeping every $d - 1$ of them independent (this is what is done in the book in the proof of Theorem 3.3.5). Or we could fix n, and start picking rows for a generator matrix G while keeping the distance of every new row from old ones at least d (by simple linear algebra this actually ensures that the whole code had distance at least d). Or we could pick a random generator matrix!

Proposition 9. *If $Vol_q(d-1, n) < q^{n-k}$ then there exists linear code of dimension k in $(F_q)^n$ with $d_c \geq d$.*

Proof. Pick every entry of an $k \times n$ matrix G uniformly and independently from F_q. We will prove the following lemma:

Lemma 1. *For any $m \neq 0$ in $(F_q)^k$, the vector mG has entries that are distributed uniformly and independently in $(F_q)^n$.*

Proof. Different components of $v = mG$ depend on disjoint sets of entries of G, and so are independent. Each one of these components v_j is the same non-zero linear combination of uniform an independent random variables $v_j = \sum_i n_i g_i j$. Since any possible value $v_j = k \in F_q$ is achieved by the same number q^{k-1} assignments of values to g_{ij}s, and each such assignment is equally likely, we conclude that v_j is uniformly distributed in F_q. This completes the proof of the lemma. □

Now the probability that mG has weight $\leq d - 1$ is the volume of Hamming ball $Vol_q(d-1, n)$ divided by the total volume of $(F_q)^n$, i.e. q^n.

So the probability that *any* bad thing happens and some non-zero m gives $wt(mG) \leq d-1$ is at most $(q^k-1)\frac{Vol_q(d-1,n)}{q^n} < Vol_q(d-1,n)q^{k-n}$. As long as that is less than 1, corresponding G will be injective (any m in the kernel will mean $wt(mG) = 0$, which we disallowed), and a generator matrix of a code with $d_c \geq d$. This completes the proof. □

Remark 13. The bounds one gets from other approaches are, while not identical, very similar. The proof above can be strengthened by noticing that we only need to worry about one m in each line (the weights of all multiples are the same), so the probability of a "bad" weight is in fact at most $\frac{(q^k-1)}{q-1}\frac{Vol_q(d-1,n)}{q^n}$.

3.2 Reed-Solomon codes.

There are several ways to think about elements s of $(F_q)^n$: as a sequence $s_1, s_2, \ldots, s_n$ of elements, as a vector tuple $(s_1, s_2, \ldots, s_n)$, or as a function from an n-element set $\{\alpha_1, \alpha_2, \ldots, \alpha_n\}$ to F_q, $s_i = s(\alpha_i)$.

In this language, looking for codes with codewords of high weight (to get high d_c) is like looking for functions that don't vanish a lot. We know some such rigid functions that don't vanish too much: a non-zero polynomial of degree $\leq k-1$ can not vanish at more than $k-1$ points. So if we evaluate such polynomials at more points (say, $n \geq k$) we will get "codewords" that have weight at least $n-k+1$. Of course we must have $n \leq q$ to be able to do this.

Defintion 12. A *Reed-Solomon code* is obtained by picking $n \leq q$ points $\{\alpha_1, \alpha_2, \ldots, \alpha_n\}$ in F_q, and taking

$$\mathcal{C} = \{p(\alpha_1), p(\alpha_2), \ldots, p(\alpha_n) | p \text{ is a polynomial over } F_q, \deg p \leq k\}$$

We get $d_c = n-k+1$ – Reed-Salamon codes meet the Singleton bound (but the alphabet size is constrained to be larger than n).

If we take as αs all elements of F_q we get so called "classical" Reed-Salomon code.

Exercise 4. Pick $1, x, \ldots, x^{k-1}$ as a basis for the space of polynomials of degree $\leq k-1$. Write down the generator matrix of Reed-Solomon code on points $\alpha_1, \ldots, \alpha_n$. Hint: Look up Vandermonde matix (this also appeared briefly in "Combinatorics" notes and will appear again in Section 3.4.1).

In the above exercise, encoding happens by sending the message m into the polynomial which has m as coefficients. There is an alternative encoding procedure which sends m to the unique polynomial of degree $\leq k-1$ taking values m_i at α_i for $i \leq k$ (this polynomial is produces by the so-called Lagrange interpolation formula); the values of resulting polynomial at subsequent αs are used as extra error correction symbols. This produces the same code (the space of values polynomials of degree $\leq k-1$ at the αs), but a different encoding. Notice that with this encoding the generator matrix is in systematic form.

We can generalize this in various ways: over F_2 we can take all multi-variable polynomials in $x_1, \ldots, x_l$ of degree $\leq m$. This is known as Reed-Muller code [2].

Alternatively, instead of, or in addition to, sending p to values at some points, evaluate derivatives of p to some order (this order can in fact vary from α to α); or send p to values and/or derivatives weighted by some constant weights. The resulting generalizations go under name of "Generalized Reed-Salamon codes" and "derivative codes".

[2] A key thing to remember when comparing this definition with those appearing in some other sources is that over F_2 $x^2 = x$, so all higher degree polynomials are sums of terms where each variable appears in at most first power

The most radical generalization is to consider points p not just in F_q but on some algebraic curve over F_q (think one-dimensional curve in the plane or in space specified or "cut out" by polynomial equations, like the circle is "cut out" by $x^2 + y^2 = 1$ – but now in F_q^2 or F_q^3 insetead of $\mathbb{R}^2$ or $\mathbb{R}^3$). And instead of just specifying zeroes and derivatives one can also allow "poles" like $\frac{1}{(x-\alpha)^k}$ or various orders in various points. Such codes are called *algebraic-geometric codes*. The Goppa codes mentioned in the book are of this type. Algebraic-geometric codes are quite powerful, but beyond our present constrains.

Remark 14. Reed-Solomon codes are widely used in practice. For example, Google's file system Colossus uses them to store large amounts of data redundantly without too much space overhead. Apache's Hadoop Distributed File System (HDFS) new version 3.0 released in December of 2017 also allows Reed-Solomon encoding.

One (main?) drawback of Reed-Solomon codes is the constraint of large alphabet size, $q > n$. However, this is less of a disadvantage if the errors come in bursts. What this means is the following: if you want to encode a long bit string, you will not be able to do this with Reed-Solomon code over F_2, since you need $q > n$. Instead, take a Reed-Solomon code over, say $F_{2^4=16}$. Then with classical Reed-Solomon code where $n = 15$ you can encode say, strings of 5 hexadecimal characters with $d = 11$ and error correct up to 5 errors. However, these are 5 errors *in the hexadecimal characters* transmitted. In practice, your 15 hexadecimal characters are 60 bits. If you have 6 bits corrupted – each one in different hexadecimal – your error correction might fail. So in a sense, you are getting up to 5 corrected errors on 60 bits transmitted – not really that great. However, if your corrupted bits tend to clump together (come in "bursts") then you can have many more individual bits affected – nearby bits likely belong to the same hexadecimal, and if so their corruption only counts as one corrupted hexadecimal.

Burst errors are common in various applications – CD and DVD scratches, bar codes and QR codes being damaged or partially occluded etc. This is one reason Reed-Solomon codes are used in these settings.

3.3 Cyclic codes.

3.3.1 Introduction. Ideals in $F_q[x]/(x^n - 1)$ and cyclic codes.

In this section we continue with the theme of viewing n-tuples of elements of F_q as functions with coefficients in F_q, but now instead of thinking about the n-tuple of values on some set of size n, we think about it as coefficients of a degree $\leq n - 1$ polynomial (in the upcoming section 3.4.2 on "general discrete Fourier transform" we will see that these are "Fourier dual" perspectives).

What are some good subspaces in the space of polynomial functions? The best sets of polynomials - the "ideal" ones - are known in algebra by exactly this name "ideal". That is, a (linear) subspace $\mathcal{C}$ of polynomials such that if $p(x) \in \mathcal{C}$ and $g(x)$ any polynomial, then $p(x)g(x) \in \mathcal{C}$. This of course will contain polynomials of arbitrarily large degree. One solution is to take our polynomials "mod x^n. Algebraically speaking, this means working with ideals in the factor ring $F_q[x]/(x^n - 1)$. Note that elements of $F_q[x]/(x^n - 1)$ are equivalence classes of polynomials, but we will be sloppy and call them "polynomials". Every class has a unique representative of degree $\leq n - 1$, so linearly $F_q[x]/(x^n - 1)$ can be identified with the vector space of these (polynomials of degree $\leq n - 1$); we are now looking at special ("ideal") subspaces in that. Note that when we speak about degree of $p \in F_q[x]/(x^n - 1)$ we mean the degree of this lowest degree representative.

What does $\mathcal{C}$ being an ideal mean bit more concretely? If $c = c_0 + c_1 x + \ldots + c_{n-1}x^{n-1} = (c_0, \ldots, c_{n-1})$ is in $\mathcal{C}$, then so should be $xc = c_0 x + c_1 x^2 + \ldots + c_{n-1}x^n = c_n + c_0 x + \ldots + c_{n-2}x^{n-1} = (c_{n-1}, c_0 \ldots, c_{n-2})$.

Defintion 13. (Cyclic linear code) A linear code $\mathcal{C}$ in F_q^n is *cyclic* if whenever $(c_0, \ldots, c_{n-1}) \in \mathcal{C}$ we also have

$(c_{n-1}, c_0 \ldots, c_{n-2}) \in \mathcal{C}$.

So if $\mathcal{C}$ is an ideal in $F_q[x]/(x^n - 1)$ then $\mathcal{C}$ is cyclic. Conversely, suppose $\mathcal{C}$ is cyclic. Then thinking of tuples as polynomials, we see that if $c \in \mathcal{C}$ then $xc \in C$. By induction $x^k c \in \mathcal{C}$ for any power k, and since $\mathcal{C}$ is a vector space, $p(x)c \in \mathcal{C}$ for any (class of) polynomial p in $F_q[x]/(x^n - 1)$. So cyclic codes and ideals correspond.

Every ideal in $F_q[x]/(x^n - 1)$ is principal - generated by a single polynomial $g(x)$. In fact, just like in $F_q[x]$ we can take any non-zero $g(x) \in \mathcal{C}$ with lowest degree and it will be a generator for $\mathcal{C}$.

Exercise 5. Prove this.

Hint. Just like in $F_q[x]$, if $c \in \mathcal{C}$ is not a multiple of g then $c = gf + r$ for r non-zero and of strictly lower degree. □

A warning: Unlike $F_q[x]$, the ring $F_q[x]/(x^n - 1)$ is not an integral domain (there are elements that multiply to zero), and it also has units (divisors of 1) of non-zero degree (any power of x is such) (for this last reason generator of $\mathcal{C}$ as an ideal is also not unique up to scalars, unlike for ideals of $F_q[x]$).

In fact, this forces $g(x)$ to divide $x^n - 1$ – otherwise the reminder of the division would be a polynomial in $\mathcal{C}$ of lower degree. If we assume that $gcd(q, n) = 1$ then all irreducible factors of factorization $x^n - 1 = f_1(x)f_2(x)\ldots f_t(x)$ are distinct. So all cyclic codes are obtained by picking $g(x)$ to be one of the 2^t possible factors of $x^n - 1$ and taking the ideal of all multiples of that $g(x)$.

3.3.2 Generator and parity check matrices for a cyclic code.

If g has degree $n - k$, then $\mathcal{C}$ intersects the $n - k$ dimensional subspace of all polynomials of degree $< n - k$ only at zero, so must have dimension at most k. At the same time the codewords $g, xg, \ldots, x^{n-k}$ form rows of an upper triangular matrix

$$G = \begin{pmatrix} g_0 & g_1 & \ldots & g_{n-k} & 0 & 0 & \ldots & 0 \\ 0 & g_0 & \ldots & g_{n-k-1} & g_{n-k} & 0 & \ldots & 0 \\ \vdots & \vdots & & \vdots & \vdots & \vdots & & \vdots \\ 0 & 0 & \ldots & 0 & g_0 & g_1 & \ldots & g_{n-k} \end{pmatrix}$$

and hence are linearly independent. So $\mathcal{C}$ is of dimension k and G is a generator matrix for $\mathcal{C}$, so that a message m is encoded as mG which is the polynomial $(m_0 + m_1 x + \ldots + m_{k-1}x^{k-1})g(x)$.

The quotient $h(x) = \frac{x^n - 1}{g(x)}$ has $g(x)h(x) = x^n - 1$, which we can write out degree by degree for all degrees $i < n$ to get $g_0 h_i + g_1 h_{i-1} + \ldots + g_{n-k} h_{i-n+k} = 0$ (we take $h_j = 0$ for negative j). This means that the matrix

$$H = \begin{pmatrix} 0 & \dots & 0 & 0 & h_k & \dots & h_1 & h_0 \\ 0 & \dots & 0 & h_k & h_{k-1} & \dots & h_0 & 0 \\ \vdots & & \vdots & \vdots & \vdots & & \vdots & \vdots \\ h_k & \dots & h_1 & h_0 & 0 & \dots & 0 & 0 \end{pmatrix}$$

has $GH^T = 0$ and since it is of rank $n - k$, by Proposition 2 H is a parity check matrix for $\mathcal{C}$.

3.3.3 Zeroes of cyclic codes.

Observation 1. When talking about zeroes of codewords in a cyclic code we have to be a bit careful - after all, code words are equivalence classes, and different representatives may not have the same roots: for example x^2 and 1 don't have the same roots (in any F_q), but are of course the same in $F_q[x]/(x^2 - 1)$. However, and this is going to be used repeatedly in the sequel, if r is a root of $x^n - 1$ – in F_q or in some extension of F_q – then it is either a root of all representatives of $p(x) \in F_q[x]/(x^n - 1)$ or is not a root of any of them. Hence for such n-th roots of unity we can talk about them being "a root" or "not a root" of p. Note also that $g(x)$ divides $x^n - 1$, so all roots of g are n-th roots of unity.

With this proviso, we see that the codewords in $\mathcal{C}$ are polynomials which have roots at all the same n-th roots of unity as g. This gives an alternative description of cyclic codes: start with a collection $\alpha_1, \alpha_2, \dots, \alpha_s$ of n-th roots of unity in some extension of F_q, and take all polynomials in F_q which vanish at these αs. Observe that if minimal polynomial of α_j is f_j, then g for the resulting cyclic code will be the least common multiple of M_js.

3.4 Bose-Chaudhuri-Hocquenghem (BCH) codes.

3.4.1 BCH codes are a special type of cyclic codes.

Special type of cyclic code is obtained when we take as the set of αs a sequential set of powers of a primitive nth root of unity. Here are the details: Still under assumptions of $gcd(n, q) = 1$, let m be the multiplicative order of q mod n; that is m is the smallest such that n is divisor of $q^m - 1$ (all others are multiples). Then F_{q^m} is the smallest extension of F_q in which polynomial $x^n - 1$ splits. Let α be a primitive n-th root of unity in F_{q^m}. Take $\alpha_1 = \alpha^b, \alpha_2 = \alpha^{b+1}, \dots, \alpha_s = \alpha^{b+s-1}$ $(0 < s < n)$. The resulting cyclic code is called *Bose-Chaudhuri-Hocquenghem (BCH) code*.

This code is of interest because it has large distance. In fact, the main result of this subsection is $d_C \geq s + 1$. Thus $s + 1$ is called "designed distance" - the code is designed to have distance at least equal to it, but may end up with higher one.

Example 11. If $s + 1 = n$ the polynomials in $\mathcal{C}$ vanish at all nth roots of unity, so must divide $x^n - 1$. Clearly this means $\mathcal{C} = 0 \subset F_q[x]/(x^n - 1)$ – which is indeed the only code there with distance n!

So why do BCH codes have these large distances? Here is a motivation: if a polynomial has a few non-zero coefficients, it is of low degree, so then it has few roots – most values are non-zero (this is what we used to say that Reed-Solomon codes have high distance). We will introduce a Fourier transform for which, essentially, values of $\hat{p}$ (at nth roots of unity) are coefficients of p and vice versa. Then applying the above logic to $\hat{p}$ we get that if p has a lot of zeroes at nth roots of unity ($\hat{p}$ is of low degree), then p has few vanishing coefficients ($\hat{p}$ has few roots). Translated into "code speak" this says that elements of BCH codes must have high weight!

We shall explain all of this in more detail in the next section, but before let's give a much more "low tech" direct proof. We will need the proposition below, whose rigorous proof in this generality seems to involve some technicalities – which we supply, and which you may take on faith if you prefer.

Proposition 10. *Let $a_1, a_2, \ldots, a_n$ be elements of a ring R. The* Vandermonde matrix of the a_is *is the matrix*

$$V = \begin{bmatrix} 1 & a_1 & a_1^2 & \cdots & a_1^{n-1} \\ 1 & a_2 & a_1^2 & \cdots & a_2^{n-1} \\ \vdots & \vdots & & \vdots & \\ 1 & a_n & a_n^2 & \cdots & a_n^{n-1} \\ & & & & \end{bmatrix}$$

Then $\det V = \prod_{i<j}(a_j - a_i)$.

Proof. We will show that equality we want is true "as a formula", and then argue that this means that it is true in any ring R. In fancy language, this means the following: Let

$$P(x_1, x_2, \ldots, x_n) = \det \begin{bmatrix} 1 & x_1 & x_1^2 & \cdots & x_1^{n-1} \\ 1 & x_2 & x_1^2 & \cdots & x_2^{n-1} \\ \vdots & \vdots & & \vdots & \\ 1 & x_n & x_n^2 & \cdots & x_n^{n-1} \\ & & & & \end{bmatrix}$$

as polynomial in $x_1, x_2, \ldots, x_n$. We argue as follows:

1) We will show the equality $P(x_1, x_2, \ldots, x_n) = \prod_{i<j}(x_j - x_i)$ is true in the polynomial ring $\mathbb{Z}[x_1, x_2, \ldots, x_n]$ (that is, "true as a formula") and 2) for any ring R and $a_1, a_2, \ldots, a_n \in R$, there is a homomorphism $\mathbb{Z}[x_1, x_2, \ldots, x_n] \to R$ that extends the map $x_i \to a_i$. Since the polynomials $P(x_1, x_2, \ldots, x_n$ and $\prod_{i<j}(x_j - x_i)$ are equal in the domain, their images $\det V$ and $\prod_{i<j}(a_j - a_i)$ are also equal.

It remains to show part 1 - the equality in the polynomial ring $\mathbb{Z}[x_1, x_2, \ldots, x_n]$. This consists of four steps:

1. P is divisible by $(x_j - x_i)$ for any pair $i \neq j$.

   Proof: In the "sum over permutations" formula for determinant we can pair up the permutations σ and $\sigma \cdot \tau_{ij}$ (this is a pairing since $\sigma \cdot \tau_{ij} \cdot \tau_{ij} = \sigma$). The corresponding monomial terms will differ by $x_i^{k_i} x_j^{k_j}$ being replaced by $-x_j^{k_i} x_i^{k_j}$. Their sum will be divisible by $x_j^{|k_i - k_j|} - x_i^{|k_i - k_j|}$, which is in turn divisible by $x_j - x_i$; hence the determinant, being the sum of all such pairs, will be divisible by $(x_j - x_i)$ as well.

2. The least common multiple of the collection of $(x_j - x_i)$ is $\prod_{i<j}(x_j - x_i)$ (and hence P is divisible by $\prod_{i<j}(x_j - x_i)$, so that $P(x_1, x_2, \ldots, x_n) = Q(x_1, x_2, \ldots, x_n) \prod_{i<j}(x_j - x_i)$).

   Proof: By a theorem from algebra, $\mathbb{Z}[x_1, x_2, \ldots, x_n]$ is a unique factorization domain, co LCM is the product of all prime divisors of the polynomials $(x_j - x_i)$; but each such polynomial is prime, being of degree 1.

3. Degree of P is the same as degree of $\prod_{i<j}(x_j - x_i)$ (and hence deg $Q = 0$, i.e. $P(x_1, x_2, \ldots, x_n) = C\prod_{i<j}(x_j - x_i)$ for some constant C).

   Proof: Both degrees are equal to $1 + 2 + \ldots + (n-1)$.

4. The constant C is equal to 1.

   Proof: The term coresponding to $\sigma = Id$ in the "sum over permuatations" formula is $x_2 x_3^2 \cdot \ldots \cdot x_n^{n-1}$ and $\sigma = Id$ is the only permutation that gives such a monomial. On the other hand, this monomial appears exactly once in expansion of $\prod_{i<j}(x_j - x_i)$ as well (we must take all the $n-1$ variables x_n that we have, then all the $n-2$ variables x_{n-1} that remain etc.). Equating the coefficients of $x_2 x_3^2 \cdot \ldots \cdot x_n^{n-1}$ on both sides of $P(x_1, x_2, \ldots, x_n) = C\prod_{i<j}(x_j - x_i)$ we see that $C = 1$.

□

Theorem 6. *Suppose $\mathcal{C}$ is the BCH code over F_q given as*

$$\mathcal{C} = \{p | p(\alpha^b) = p(\alpha^{b+1}) = p(\alpha^{b+s-1}) = 0\}$$

for some α primitive root of unity in F_{q^m}, some b and some $s \leq n$. Then $d_c \geq s + 1$

Proof. We want to show that any p in $\mathcal{C}$ has at least $s+1$ non-zero coefficients. Suppose the opposite. Then $p(x) = b_1 x^{n_1} + b_2 x^{n_2} + \ldots + b_s x^{n_s}$. The s equations $p(\alpha^{b+i}) = 0$ can be written in matrix form as:

$$\begin{bmatrix} \alpha^{bn_1} & \alpha^{bn_2} & \ldots & \alpha^{bn_s} \\ \alpha^{(b+1)n_1} & \alpha^{(b+1)n_2} & \ldots & \alpha^{(b+1)n_s} \\ \vdots & \vdots & & \vdots \\ \alpha^{(b+s-1)n_1} & \alpha^{(b+s-1)n_2} & \ldots & \alpha^{(b+s-1)n_s} \end{bmatrix} \begin{bmatrix} b_1 \\ b_2 \\ \vdots \\ b_s \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ \vdots \\ 0 \end{bmatrix}$$

We claim that the matrix on the left is invertible – so that $b_j = 0$ and we get a contradiction. Why is it invertible? We can take out the factor of α^{bn_j} from each column, without affecting invertibility. What remains is the matrix $\begin{bmatrix} 1 & 1 & \cdots & 1 \\ \alpha^{n_1} & \alpha^{n_2} & \ldots & \alpha^{n_s} \\ \vdots & \vdots & & \vdots \\ \alpha^{(s-1)n_1} & \alpha^{(s-1)n_2} & \ldots & \alpha^{(s-1)n_s} \end{bmatrix}$, which is the transpose of the Vandermonde matrix of α^{n_j}, which is invertible (its determinant is $\prod_{i<j}(\alpha^{n_j} - \alpha^{n_i}) \neq 0$).

□

3.4.2 Number theoretic Fourier transforms.

Motivation: "classical" Fourier transformations.

"Classical" Fourier transform decomposes $\mathbb{C}$-valued functions on $\mathbb{R}$ into "continuous linear combinations" of characters $\chi_\xi(x) = e^{2\pi i \xi x}$ (for any $\xi \in \mathbb{R}$) – homomorphisms from $(\mathbb{R}, +)$ to $(S^1 \subset \mathbb{C}, \cdot)$; Fourier series decomposes $\mathbb{C}$-valued functions on $\mathbb{Z}$ into "infinite linear combinations" of characters $\chi_n(x) = e^{2\pi i n x}$ (for any $n \in \mathbb{Z}$) – homomorphisms from $(S^1, +)$ to $(S^1 \subset \mathbb{C}, \cdot)$ (these are the characters of $\mathbb{R}$ that factor through the quotient $\mathbb{R} \to S^1$).

Discrete Fourier transform decomposes $\mathbb{C}$-valued functions on $\mathbb{Z}/n\mathbb{Z} \subset S^1$ into linear combinations of characters $\chi_k(x) = e^{2\pi i \frac{k}{n} x}$ (for $k = 0, \ldots, n-1$) – homomorphisms from $(\mathbb{Z}/n\mathbb{Z}, +)$ to $(S^1 \subset \mathbb{C}, \cdot)$ (these are the restrictions of all the characters from S^1).

General Fourier transformations.

Now if we take functions on $\mathbb{Z}/n\mathbb{Z}$ valued in any field F (we will soon take $F = F_{q^m}$) then a homomorphism from $\mathbb{Z}/n\mathbb{Z}$ to F^* is specified by an element $\beta \in F^*$ of order dividing n (the place where $1\mathbb{Z}/n\mathbb{Z}$ has to go). The n character sending $1\mathbb{Z}/n\mathbb{Z}$ to some primitive root of unity β, and the ones sending $1\mathbb{Z}/n\mathbb{Z}$ to powers of β together suffice to decompose all F valued functions on $\mathbb{Z}/n\mathbb{Z}$. Thus we have a definition.

Defintion 14. Given any field F and an element β of order n in F^*, define a linear map $FT_\beta : F^n \to F^n$ sending $p = (p_0, p_1, \ldots, p_{n-1})$ to $\hat{p}$ with $\hat{p}_j = \sum_i p_i \beta^{-ji}$.

Observation 2. If we think of $p(i)$ as coefficients of F valued polynomial $p(x) = p_0 + p_1 x + \ldots, p_{n-1}x^{n-1}$, then $\hat{p}_j = p(\beta^{-j})$. Thus FT_β relates sequence of coefficients of $p(x)$ to a sequence of its values.

Assume that n is invertible in F (this is always the situation we are in if $F = F_{q^m}$, because otherwise there would be no primitive nth root of unity β in F^* – we must have $n|q^m - 1$). Then our next proposition says that FT_β is invertible, and just like for "classical" Fourier transforms, inverse transform is given by formula very similar to the "direct" transform.

Proposition 11. *The inverse of FT_β is $\frac{1}{n} FT_{\beta^{-1}}$.*

Proof. Method 1: This is true for the same reason as in the usual Fourier transform, namely a kind of "orthogonality of characters". Namely define a dot product of two F valued functions on $\mathbb{Z}/n\mathbb{Z}$ as

$$< f, g >= \sum_{k \in \mathbb{Z}/n\mathbb{Z}} f(k)g(-k)$$

This is not "positive definite" - the notion does not even make sense since the values of the dot product are in F. However it is non-degenerate, since every f has some g with $< f, g >\neq 0$.

Then if f and g are characters (homomorphisms from $\mathbb{Z}/n\mathbb{Z}$ to F^*) then the dot product of fg is equal to its own multiple by $f(1)g(1)^{-1}$:

$$\begin{aligned} < f, g > &= \sum_{k \in \mathbb{Z}/n\mathbb{Z}} f(k)g(-k) = \sum_{(l+1) \in \mathbb{Z}/n\mathbb{Z}} f(l+1)g(-(l+1)) \\ &= \sum_{l \in \mathbb{Z}/n\mathbb{Z}} f(l)f(1)g(-l)g(-1) = f(1)g(1)^{-1} \sum_{k \in \mathbb{Z}/n\mathbb{Z}} f(k)g(-k) = f(1)g(1)^{-1} < f, g > \end{aligned}$$

So for different characters for which $f(1) \neq g(1)$ we get $< f, g >= 0$. Of course one easily computes $< f, f >= n$.

In our situation, the n characters χ_j given by $\chi_j(k) = \beta^{jk}$ pairwise orthogonal of "square length" n. This implies that they are linearly independent, and hence form a basis. In fact $(FT_\beta f)(l) =< f, \chi_l >$ – Fourier coefficients are dot products with characters, aka components in the basis of characters. We then have the the usual way of expressing any f in terms of its components in orthogonal basis: $f = \frac{1}{n} \sum_k < f, \chi_k > \chi_k$. (This holds since $\frac{1}{n} \sum_k < f, \chi_k > \chi_k$ has

the same dot products as f with any basis vector χ_j, so by non-degeneracy of the dot product they must be equal.) This is precisely the formula for inverse Fourier transform we were after.

Method 2, computation: You can compute with summation signs and indexes, but I like to visualize the same computation with matrices. The matrix of FT_β (in the usual convention of matrices acting on column vectors by left multiplication) is

$$\begin{pmatrix} 1 & 1 & 1 & \dots & 1 \\ 1 & \beta^{-1} & \beta^{-2} & \dots & \beta^{-(n-1)} \\ 1 & \beta^{-2} & \beta^{-4} & \dots & \beta^{-2(n-1)} \\ \vdots & \vdots & \vdots & & \vdots \\ 1 & \beta^{-(n-1)} & \beta^{-2(n-1)} & \dots & \beta^{-(n-1)^2} \end{pmatrix}$$

The matrix of $FT_{-\beta}$ is

$$\begin{pmatrix} 1 & 1 & 1 & \dots & 1 \\ 1 & \beta^{1} & \beta^{2} & \dots & \beta^{(n-1)} \\ 1 & \beta^{2} & \beta^{4} & \dots & \beta^{2(n-1)} \\ \vdots & \vdots & \vdots & & \vdots \\ 1 & \beta^{(n-1)} & \beta^{2(n-1)} & \dots & \beta^{(n-1)^2} \end{pmatrix}$$

Their product

$$M = \begin{pmatrix} 1 & 1 & 1 & \dots & 1 \\ 1 & \beta^{-1} & \beta^{-2} & \dots & \beta^{-(n-1)} \\ 1 & \beta^{-2} & \beta^{-4} & \dots & \beta^{-2(n-1)} \\ \vdots & \vdots & \vdots & & \vdots \\ 1 & \beta^{-(n-1)} & \beta^{-2(n-1)} & \dots & \beta^{-(n-1)^2} \end{pmatrix} \begin{pmatrix} 1 & 1 & 1 & \dots & 1 \\ 1 & \beta^{1} & \beta^{2} & \dots & \beta^{(n-1)} \\ 1 & \beta^{2} & \beta^{4} & \dots & \beta^{2(n-1)} \\ \vdots & \vdots & \vdots & & \vdots \\ 1 & \beta^{(n-1)} & \beta^{2(n-1)} & \dots & \beta^{(n-1)^2} \end{pmatrix}$$

clearly has n as each diagonal entry M_{kk}, and in every non-diagonal entry M_{kl} it has a sum of n-term geometric series sith initial term 1 and rate $r = \beta^{k-l} \neq 1$ (β is *primitive* nth root of 1). Thus the entry is $\frac{1-r^n}{1-r}$; but $r^n = \beta^{n(k-l)} = 1$. This concludes the proof.

Remark 15. Behind all the fancy formulas, you can think of these proofs as generalizing the proof that the sum of all nth roots of unity in $\mathbb{C}$ is 0 – either observe that this sum is invariant under rotation by $e^{2\pi i/n}$ and so must be zero; or compute $1 + \rho + \rho^2 + \ldots + \rho^{n-1} = \frac{1-\rho^n}{1-\rho} = 0$.

□

3.4.3 Classical Reed-Solomon code as a BCH code. Design distance of BCH codes.

Example 12 (Classical Reed-Solomon code as BCH code - a Fourier transfrom view). "Classical" Reed-Solomon code is obtained by letting α be the primitive element of F_q^* and taking

$$\mathcal{C} = \{c_p = (p(1), p(\alpha), \ldots, p(\alpha^{q-1}) | \deg p \leq k-1\}$$

Taking $\beta = \alpha^{-1}$ for our FT, we see:

1. The string $c_p = (p(1), p(\alpha), \ldots, p(\alpha^{q-1}))$ is

$$c_p = \left(p(1), p(\beta^{-1}), \ldots, p(\beta^{-(q-1)})\right) = (\hat{p}_0, \hat{p}_1, \ldots, \hat{p}_{q-1}) = \hat{p}$$

2. The condition $\deg p \leq k-1$ is just saying $p_k = p_{k+1} = \ldots, p_{n-1} = 0$. Up to factor of $\frac{1}{n}$, $p_j = \hat{p}(\beta^j) = \hat{p}(\alpha^{-j}) = \hat{p}(\alpha^{n-j})$. So the degree condition on p is the condition

$$\hat{p}(\alpha) = \hat{p}(\alpha^2) = \ldots = \hat{p}(\alpha^{n-k}) = 0$$

So

$$\mathcal{C} = \{\hat{p} | \hat{p}(\alpha) = \hat{p}(\alpha^2), \ldots, \hat{p}(\alpha^{n-k}) = 0$$

This is clearly a BCH code – just one with $n = q-1$ and $b = 1$. Here $s = n-k$ and design distance is $s+1 = n-k+1$, which is indeed the distance for this code.

We now fill in the details of the outline about distance of BCH codes. This is actually quite similar to Example 12.

Theorem 7 (Theorem 6, again)**.** *Suppose $\mathcal{C}$ is the BCH code over F_q given as*

$$\mathcal{C} = \{p | p(\alpha^b) = p(\alpha^{b+1}) = p(\alpha^{b+s-1}) = 0\}$$

for some α primitive root of unity in F_{q^m}, some b and some $s \leq n$. Then $d_c \geq s+1$

Proof. Suppose $p \in \mathcal{C}$. Again, we use $\beta = \alpha^{-1}$ for our Fourier transforms. Then $\hat{p}_j = \frac{1}{n} p(\alpha^j)$, so $\hat{p}$ has at least s zero coefficients. This means that $\hat{p}$ "can be cycled to" a polynomial q of degree $\leq n-s$ (rigorously speaking, $\hat{p}$ is congruent mod $x^n - 1$ to a polynomial q of degree $\leq n-s-1$). So q has no more than $n-s-1$ roots, and so in particular no more than $n-s-1$ of the powers of β are roots of q – and since all such powers are roots of x^{n-1}, the same ones are the roots of $\hat{p}$ (compare with Observation 1). The conclusion is that no more than $n-s-1$ powers of β are roots of $\hat{p}$, which is to say (by inverse FT), no more than $n-s-1$ coefficients of p are zero, at least $s+1$ are non-zero, giving p weight of at least $s+1$. This completes the proof.

□