Intro to Quantum Error Correction

Ben Criger

October 26, 2016

1 Introduction

The purpose of this note is to help introduce quantum error correction [QEC] and (especially) fault-tolerance to people who are not familiar with quantum mechanics, or quantum computing, such as:

- undergraduates,
- electrical/computer/software engineers,
- experimental physicists now trying to implement QEC, who have missed such an introduction.

My goals in writing this note are:

- to keep it short, at the cost of completeness,
- to refer to longer/superior works where possible (often),
- to frequently compare "scary quantum" processes with their classical equivalents, to help de-mystify QEC.

[Insert usual caveat about imprecise notation, i.e.: these notes are free, and you get what you pay for.]

I begin in the following section with an explanation of why error correction is necessary, moving on to describe the difference between error-correction and fault-tolerance, before moving on to the most popular elements of QEC theory (stabilizer codes $\mapsto$ CSS codes $\mapsto$ homological codes $\mapsto$ the surface code).

2 Why Error Correction?

If you meet a software engineer in the street, the need for error correction may not be obvious. In the life of a computer programmer, errors are things caused exclusively by people, and careful thinking and good practices are required to prevent, detect and correct them. [Maybe there's a repetition code at work here, especially with `diff`, but that is an analogy for another day.] However, if you scratch the surface, you find instances of error correction and even (what I'm going to call) fault-tolerance at work.

Take, for example, satellite/spacecraft communications. These spacecraft are very far away, and there are bits that don't make it between the spacecraft and the ground station. Actually, that example is too far outside our daily experience to take, so let's take flash drives, AKA USB keys. The job of these drives is to store bits (0's and 1's), which they manage to do most of the time. However, one bit in every 10^{4-9} is randomly flipped, which in the worst case scenario would lead to an ASCII typo per page of text [or something].

To compensate for this, the USB key records the information redundantly, using more bits than are strictly necessary. When reading, the CPU can then check that different decodings of the redundant information are consistent, and if they're not, can infer a decoding that's likely to be correct. The fact that the CPU is not itself (assumed to be) error-prone is a great boon to us in reading USB keys, otherwise we couldn't trust that any decoding was correct.

There's a reason that CPUs are not considered noisy, by the way. Transistor-transistor logic (TTL for short) defines a standard voltage for its 0 and 1 states, usually $0\,\mathrm{V}$ and $5\,\mathrm{V}$, respectively. These circuits can be designed to output $0\,\mathrm{V}$ for any input voltage between $0\,\mathrm{V}$ and $0.4\,\mathrm{V}$, likewise to output $5\,\mathrm{V}$ whenever a voltage of $2.6\,\mathrm{V}$ or higher is input. This is a (very powerful) error correction mechanism. If noise spreads voltages out according to some unimodal distribution, we can tighten that distribution at every TTL element, paying only a small energy cost to do so. In theory, we could also design an element which outputs a third value whenever the input voltage doesn't correspond to a

logical 0 or 1, so that we can detect errors. In practice, we would just increase the supply voltage until we don't have to worry.

The availability of reliable components which can be used to store and process classical information is a great benefit to computer engineers. In the following sections, we'll see which of these properties can also be achieved for quantum systems. But first, some definitions.

2.1 What is a Code?

A code is a subset.

If you have a system, classical quantum or other, which can assume states from some set, you can define any subset of these states to be a code. The states in the code are called the *codewords*.

Most of the time, we will also constrain our codes to be *linear* with respect to some operation (usually denoted $+$), which means that for any codewords a and b, $a + b$ is also a codeword. This is pretty common, and it means we can normally consider state sets which are vector spaces, and codes which are subspaces. This is true in both the classical and quantum case.

A code can be used to fulfil a variety of purposes. Compression, for example, involves codes whose codewords are shorter than the logical messages they represent.

We will mostly concern ourselves with *error-correcting* codes, which can only be analysed using two other mathematical objects as tools. The first of these is the *error model*, the set of possible transformations which can act on a state (codeword or otherwise) without our knowledge. The second is the *recovery map*, which takes as input states from the full set and outputs corresponding codewords. To solidify these ideas, let's consider an example.

2.1.1 Example: 3-Bit Repetition Code

Suppose Alice wants to tell Bob the value of a bit, 0 or 1, but that she has to use a channel which is subject to symmetric bit-flip:

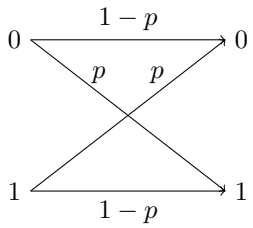

Figure 1: Graphical representation of an error map on a classical bit which flips the bit (taking 0 to 1 and vice versa) with probability p.

Alice, then, would like to send messages from $\mathbb{Z}_2$, so she should design codes that have two codewords. Supposing that Alice has repeated access to the channel, and knows the probability p, she can design codes which use $\mathbb{Z}_2^n$ as the full state space. A simple thing that Alice can do is repeat herself three times (taking $n = 3$) to make herself understood. Bob, then, should infer that 0 is the desired message when he receives a state containing mostly 0s and 1 when he receives a state containing mostly 1s. Let's express these ideas mathematically, so that we can generalize later.

Codewords : $000, 111$

Error Model :

$$000 \mapsto \begin{array}{cc} 000 & (1-p)^3 \\ 001,\,010,\,100 & p(1-p)^2 \\ 110,\,101,\,011 & p^2(1-p) \\ 111 & p^3 \end{array} \qquad 111 \mapsto \begin{array}{cc} 111 & (1-p)^3 \\ 110,\,101,\,011 & p(1-p)^2 \\ 001,\,010,\,100 & p^2(1-p) \\ 000 & p^3 \end{array} \tag{1}$$

Recovery Map : $(000,\,001,\,010,\,100) \mapsto 000$, $(111,\,110,\,101,\,011) \mapsto 111$

We can see that, if there is a weight-two or weight-three error, Bob will infer the incorrect state. The resulting probability of error is $3p^2(1-p)+p^3$.

Exercise: When does this protocol amplify errors? How should we remedy this?

We will go on to construct quantum codes in later sections. But first, I should confess something.

2.2 Fault Tolerance

Let's begin with an overall reminder of what we're trying to accomplish. We want to replicate the properties of TTL logic for quantum states and operations. Not only are we able to interpret the output of a TTL gate using a perfect computer, we also know that TTL operations suppress errors that occur *during the operations themselves*. We therefore have to *encode operations* in some way, using a small subset of operations drawn from some large set to ensure that errors from the operations used to correct the code can themselves be corrected. For the first few examples, we won't focus on this, since it's useful to see that we can correct 'vanilla' errors before moving on to attempt fault tolerance.

3 Quantum Codes

If fault tolerance wasn't a big enough problem, we have another one. Error models for quantum systems are much more complex than those for classical bits, because they're *continuous*. They don't map $|\psi\rangle$ to $|\psi_\perp\rangle$, but instead to $U|\psi\rangle$ for U some unknown unitary, for example. There are even some channels for which no mixture of unitaries can describe the channel's effects. For this general set of quantum channels, we can represent their effects mathematically using a set of operators. This is frequently called the *Kraus representation*, after Karl Kraus. Given these operators (let's call them E_j), we calculate the effect of the channel on a density matrix as follows:

$$\rho \mapsto \sum_j E_j \rho E_j^\dagger \tag{2}$$

There's also a constraint on these operator sets. In order to be valid transformations on quantum states, they have to map density matrices to density matrices. Since density matrices are positive-semidefinite with trace one, the action of the operator set must be *completely positive* (preserving positivity, even when it acts on *part of a larger system*), and trace-preserving. This is true iff $\sum_j E_j^\dagger E_j = \hat{\mathbb{1}}$. Let's get this clearer with an example. Let's do bit-flip again, calling the channel B:

$$B_0 = \sqrt{1-p}\begin{bmatrix}1 & 0\\ 0 & 1\end{bmatrix}, \quad B_1 = \sqrt{p}\begin{bmatrix}0 & 1\\ 1 & 0\end{bmatrix} \tag{3}$$

We can calculate the effect on a one-qubit density matrix with ground state population q and coherence c:

$$\rho = \begin{bmatrix}q & c\\ c^* & 1-q\end{bmatrix} \tag{4}$$

$$\sum_j B_j \rho B_j^\dagger = (1-p)\rho + pX\rho X \tag{5}$$

$$= (1-p)\begin{bmatrix}q & c\\ c^* & 1-q\end{bmatrix} + p\begin{bmatrix}1-q & c^*\\ c & q\end{bmatrix} \tag{6}$$

$$= \begin{bmatrix}q+p-2pq & c-p(c-c^*)\\ c^*-p(c^*-c) & 1-p-q+2pq\end{bmatrix} \tag{7}$$

Exercise: Sub in the $|0\rangle$ and $|1\rangle$ states. What happens to them?

If we allow generic operations, this looks similar to the central problem of analog computing, that it's far easier to force an input from a continuous space to a few discrete values (oh, say, $0\,\mathrm{V}$ and $5\,\mathrm{V}$) than it is to force noisy values to stay close to the value that some arbitrary computation should be at ($x\,\mathrm{V}$). For this reason, people with experience in analog computing are wary of quantum computing when they hear about it. All is not lost, though. In the next subsection, we'll see that these analog errors can be *digitized*, but first, we have to define what it means to correct an error in the quantum case.

3.1 The Knill-Laflamme Criterion and Noise Digitization

To talk about which quantum errors can be detected or corrected, we should first make clear that in the typical case, not all of them can be corrected. To see this, consider the quantum bit-flip map defined above, and its action on the quantum bit-flip code $|000\rangle$, $|111\rangle$. With probability p^3, the channel maps one logical state directly to the other. Any sane recovery map would include $|111\rangle \mapsto |111\rangle$, so that there's always some probability of error. However, we can always try to take a subset of the operators E_j and declare it to be exactly detectable or correctable. If our (assumed orthogonal) codewords are labeled $|\psi_j\rangle$, then this happens whenever

$$\langle\psi_j |E_k| \psi_m\rangle = C_k\delta_{jm} \text{ or } \left\langle\psi_j \left|E_k^\dagger E_l\right| \psi_m\right\rangle = C_{kl}\delta_{jm} \tag{8}$$

Due to limited time, we won't go through the proof, but a detailed and accessible proof can be found in section 2.3 of Lidar and Brun. I would like to focus on the interpretation of this formula instead. First, let's note that it implies that any time two different codewords are acted upon by correctable Kraus operators, the resulting states will still be orthogonal. This means that, in theory, a recovery map can be designed which maps them back to the original code states, though this may be difficult in practice.

Second, and more importantly, it implies that if a code can correct two Kraus operators E_a and E_b, it can correct any linear combination of the two. I will also not prove this, but it's in section 2.6 of Lidar/Brun. If we can correct an X, Y or Z error on an arbitrary qubit, though, it follows that we can correct any single-qubit error, since X, Y, and Z form a basis for the set of possible operators on one qubit. It takes a little work to get such a code, we'll see some examples in the next section.

3.2 The Nine-Qubit Code

First, let's make a note of what happens if a Z error acts on the bit-flip code.

$$\alpha|000\rangle + \beta|111\rangle \mapsto \alpha|000\rangle - \beta|111\rangle. \tag{9}$$

This is the equivalent of a logical Z operation, so this code cannot correct Z errors. There is, however, a code which corrects Z errors, it consists of the states $|+++\rangle$ and $|---\rangle$. When Z acts on these states, they get mapped to $|++-\rangle$, et cetera, exactly as would happen if we acted X on the states $|000\rangle$ and $|111\rangle$. Now is as good a time as any to put in some circuits for bit-flip, phase-flip and Shor encoding/decoding (from Chapter 2 of Lidar/Brun by Dave Bacon, used without permission$_{\text{yolo}}$):

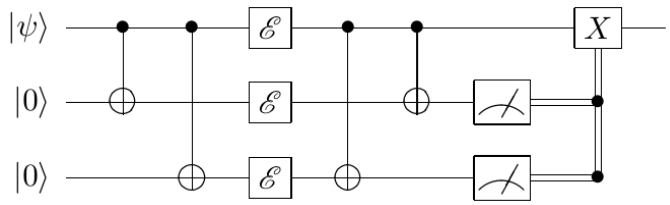

Figure 2: Typical circuit for an error correction scenario with only one noisy channel. In this case, the channel is a bit-flip, so we encode $\alpha|0\rangle + \beta|1\rangle$ into $\alpha|000\rangle + \beta|111\rangle$ for redundancy.

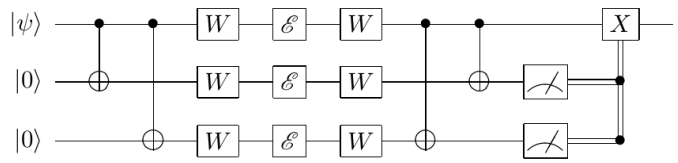

Figure 3: Same deal as before, but with phase-flip, so we encode $\alpha|0\rangle+\beta|1\rangle$ into $\alpha|+++\rangle+\beta|---\rangle$ for redundancy. Bacon uses a W for the Hadamard gate. Maybe this stands for Walsh, maybe not. We may never know . . .

Given these two circuits, we can take a look at *concatenation*, where we encode the logical qubits of one code into another code:

Exercise: Show that this code can also correct a single Y error on any qubit.

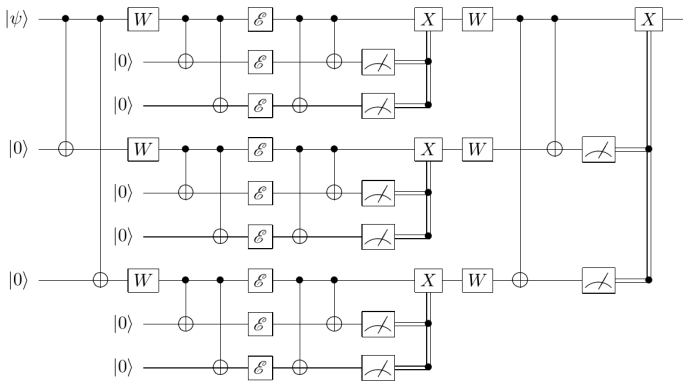

Figure 4: Each physical bit of the three-qubit code versus phase flip is encoded into a three-qubit code versus bit-flip to construct a nine-qubit *Shor code*.

Even though we get Y error correction 'for free' since it's the product of an X and a Z, this is still a little clunky. In the next section, we take a look at a code which can correct an arbitrary single-qubit error using only five qubits.

3.3 The Five-Qubit Code

With the tools we have so far, we can't do much more than state the codewords of this code:

$$\begin{aligned}|0_L\rangle = \frac{1}{4}\Big[&|00000\rangle + |10010\rangle + |01001\rangle + |10100\rangle \\ &+|01010\rangle - |11011\rangle - |00110\rangle - |11000\rangle \\ &- |11101\rangle - |00011\rangle - |11110\rangle - |01111\rangle \\ &- |10001\rangle - |01100\rangle - |10111\rangle + |00101\rangle\Big] \qquad (10.104)\\ |1_L\rangle = \frac{1}{4}\Big[&|11111\rangle + |01101\rangle + |10110\rangle + |01011\rangle \\ &+|10101\rangle - |00100\rangle - |11001\rangle - |00111\rangle \\ &- |00010\rangle - |11100\rangle - |00001\rangle - |10000\rangle \\ &- |01110\rangle - |10011\rangle - |01000\rangle + |11010\rangle\Big] \qquad (10.105)\end{aligned}$$

Figure 5: Some codewords, straight jacked from Nielsen/Chuang. I'm pretty sure this code was derived from some obscure research in codes over $GF(4)$, an abstract field that acts like the Paulis in some way.

Exercise for Masochists: Show that this code can correct an arbitrary single-qubit error using the Knill-Laflamme criteria.

The Shor code may have seemed clunky, but the fact that it could be decomposed into other codes meant we didn't really have to write out the codewords in the computational basis. This is valuable, since there can be exponentially many terms in such a decomposition. This motivates the introduction of some new formalism, to make sure that the codes we study don't require us to write down an exponential number of parameters.

4 The Stabilizer Formalism in a Nutshell

There is at least one well-studied way to do this (there are a few others as well, but they're not as popular). This theory uses sets of operators instead of basis vectors as its representation of a subspace, because tensor-product operators can give rise to states with non-trivial entanglement, which it turns out is necessary to do error correction [Prove this?]. This is the *Heisenberg picture*.

It's also best to use operators that form a group to do this, that way we don't have to keep track of any matrices during a calculation, we can just use the group operation. If we're bound and determined to use qubits at the physical level to store and process qubits at the logical level, we might as well use a group that has a matrix representation that's

2-by-2, like the Pauli group. To write out tensor products of Paulis, we just put them in a string. These operators have $+1$ and -1 eigenspaces of equal size. We can see this from the fact that each single-qubit Pauli has a $+1$ and a -1 eigenvalue, and that the eigenvalues of a tensor product of two matrices are the Cartesian product of the sets of eigenvalues of the original matrices. This means that, if we want to specify an $n-1$-qubit subspace of a space, we only have to write down one n-qubit operator, given as an $\mathcal{O}(n)$-letter string. This means that, in order to define a subspace with k "logical" or "effective" qubits, we specify $n-k$ stabilisers, using a total of $\mathcal{O}(n^2)$ letters (assuming k is constant with respect to n). A rigourous expression of the codespace is as follows:

$$C(S) = \{|\psi\rangle \,|\, s|\psi\rangle = |\psi\rangle \forall s \in S\} \tag{10}$$

Exercise: $-\hat{\mathbb{1}}$ is a Pauli, but it can never be a stabiliser. Why not?

Exercise: In order for a list of r Paulis to generate a stabiliser group with $n-r$ logical qubits, they have to commute and be independent under multiplication. Why?

end of day 1
start of day 2

Since this is the beginning of the lecture, let's get into a bit of a digression.

Last lecture, we saw that we can represent subspaces of a Hilbert space using lists of Pauli operators. We're going to use these spaces as error-correcting codes in a little while, but let's take a look at other things we can do with them:

4.0.1 Study Entangled States

Recall that the dimension of the subspace described by some stabiliser group is 2^{n-r}, where r is the number of generators and n is the number of qubits. If $n = r$, then we're describing a unique state, one which is usually pretty entangled. We can express this state as a rank-one projector which we derive from the stabiliser generators in the following fashion. First, we note that each Pauli can be expressed as the difference between two projectors, one onto its $+1$ eigenspace and one onto its -1 eigenspace:

$$p = \Pi_{p,+1} - \Pi_{p,-1} \forall p \in P \tag{11}$$

Also, we can express the identity as a sum over the same projectors:

$$\hat{\mathbb{1}} = \Pi_{p,+1} + \Pi_{p,-1} \forall p \in P \tag{12}$$

Therefore, we can write the projector $\Pi_{p,+1}$ as $1/2(p + \hat{\mathbb{1}})$ for any p, and express the projector onto the mutual $+1$ eigenspace as the product of all these projectors:

$$\Pi_S = \frac{1}{2^r} \prod_{s \in S} (\hat{\mathbb{1}} + s) = \frac{1}{2^r} \sum_{s' \in \langle S \rangle} s' = |\psi\rangle\langle\psi| \text{ (if } n = r) \tag{13}$$

where the last step is a little combinatorics, and I've put the generators of the stabiliser in the set S, with $\langle S \rangle$ being the generated group.

This is kind of neat, it gives us a direct mapping to projectors onto subspaces, which are states when $n = r$. It's not useless, either, since it makes taking the partial trace really easy. Let's divide the qubits into two regions R and its complement $\bar{R}$. The partial trace over $\bar{R}$ can be expressed as follows:

$$\mathrm{tr}_{\bar{R}} \left[|\psi\rangle\langle\psi|\right] = \mathrm{tr}_{\bar{R}} \left[\frac{1}{2^n} \sum_{s' \in \langle S \rangle} s'\right] = \frac{1}{2^n} \sum_{s' \in \langle S \rangle} \mathrm{tr}_{\bar{R}} \left[s'\right] = \frac{1}{2^n} \sum_{s' \in \langle S \rangle} s'_R \times \mathrm{tr}\left(s'_{\bar{R}}\right), \tag{14}$$

where p_r is the Pauli p restricted to the region r (e.g. $(X_1 Y_2 Z_3)_{23} = Y_2 Z_3$). If $s'_{\bar{R}}$ is anything other than the identity, its trace is 0 (neat property of Pauli matrices), so the sum simplifies to:

$$\mathrm{tr}_{\bar{R}} \left[|\psi\rangle\langle\psi|\right] = \frac{1}{2^n} \sum_{s' \in \langle S \rangle, s'_{\bar{R}} = \hat{\mathbb{1}}} s'_R, \tag{15}$$

a uniformly mixed state over a different stabiliser subspace. The ease with which we can take this partial trace looks like it makes the calculation of the Von Neumann entropy easier, but I haven't checked.

4.0.2 Express Commuting Hamiltonian Ground States

Actual Physicists® like to calculate the ground states of local Hamiltonians. For spin systems, the terms in these Hamiltonians are sometimes tensor products of Pauli operators. When these terms commute, the Hamiltonian can be expressed as

$$H = -E \sum_{s \in \tilde{S} \subset \langle S \rangle} s, \tag{16}$$

and the ground state (or states) are in the stabiliser subspace of the generator S. This is pretty interesting, because we can try to design codes that *correct themselves* when exposed to thermal noise, provided that the amount of energy needed to produce a logical error scales with the size of the system (this is different than just saying that the code has a distance that scales with the size of the system, as we will see later).

4.1 Actual Error Correction

It remains to be seen that subspaces defined in this way can detect/correct errors, though it is sufficient to detect/correct Pauli errors. Let's show that Pauli errors can be detected if they anticommute with some stabiliser.

First, let's note that, for stabiliser codes, logicals commute with the stabiliser, but are not part of it:

$$L|\psi\rangle = |\phi\rangle,\ |\phi\rangle, |\psi\rangle \in C(S), |\phi\rangle \neq |\psi\rangle \tag{17}$$

$$s|\phi\rangle = |\phi\rangle \forall s \tag{18}$$
$$\therefore sL|\psi\rangle = L|\psi\rangle \forall s \tag{19}$$
$$\therefore sL|\psi\rangle = Ls|\psi\rangle \forall s \tag{20}$$

Since Paulis either commute or anticommute, this means that $[s,\, L] = 0 \forall s$.

An error is undetectable when it can change $\langle \psi \, | \phi \rangle$, this is exactly what logicals do. Given a set of potential Pauli errors, the product of any two has to be a logical for the error to not be correctable. Let's go for an example of errors which are detectable but not correctable for the 5-qubit code. This code has $2^4 = 16$ possible syndromes, and $1 + 5 \cdot 3 = 16$ correctable errors ($\hat{\mathbb{1}}$ and $X/Y/Z$ on every qubit), so any error outside of this set ought to do, let's try $XXIII$. Sure enough, it anticommutes with $XZZXI$ and $ZXIXZ$, but commutes with $IXZZX$ and $XIXZZ$, which is the exact thing that $IIIZI$ does, so this error will be decoded incorrectly, even though it can be detected.

It's all well and good to say that there are some codes that can be *analysed* efficiently, but what about designing them? If we select a set of stabilisers (which seems easy enough to do), how many independent one-qubit errors can be corrected? The answer to this question is the *distance* of the code, and it is the minimum weight (expressed in terms of one-qubit errors) of a logical operator.
Exercise: A naïve but commonly-used method for finding the distance of a code is to obtain a set of logical operators and multiply them by all possible elements of the stabiliser group, keeping track of the minimum weight encountered. Is there a better method? How about for classical codes?

As a side note, I'll mention here that classical linear codes with n physical bits, k logical bits and distance d are often referred to as $(n,\, k,\, d)$ codes, and quantum codes with the same parameters are called $[\![n,\, k,\, d]\!]$ codes.

This question drives us to design quantum codes using classical codes as components, since there are many large families of classical codes whose distances are known. We consider this in the following section.

4.2 CSS Codes

We saw, in our discussion of the Shor nine-qubit code, that we can correct all possible one-qubit errors as long as we can correct an X and a Z error which occur on any qubit, since $Y_j \sim X_j Z_j$. Our recipe for codes that achieve this was based on a few ideas:

Classical-to-Quantum We waved our hands pretty hard on an example of the three-qubit repetition code to come up with a circuit that produces a quantum version of this code, we will reveal the ideas behind this in this section.

X/Z **Equivalence** We took that same repetition code and Hadamard-transformed all the bits, producing a code that corrects Z errors ("phase flips") given a code that corrects X errors ("bit flips"). This is also general, and we will take advantage of this further.

Concatenation To unite two codes and obtain the error correcting properties of each, we encoded a code in a code. This can be accomplished with any two stabiliser codes, but if the two codes C and C' use n and n' qubits, respectively, then the concatenated code uses $n \times n'$ qubits, which is pretty high overhead just to be able to correct single-qubit errors. We will see in this section that this is not necessary, and an alternate construction can yield smaller codes.

To do this, we use a construction that takes classical codes (on $\mathbb{Z}_2$) to quantum codes that correct one error type.

This construction is based on the *parity check matrix* of a classical code, which is a matrix whose kernel is the codespace. We usually keep track of this kernel by writing it out as a matrix of column vectors called the *generator matrix*. This is best understood by example:

$$G_3 = \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix}, \; H_3 = \begin{bmatrix} 1 & 1 & 0 \\ 0 & 1 & 1 \end{bmatrix} \qquad G_7 = \begin{bmatrix} 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 1 & 1 & 1 \\ 1 & 0 & 0 & 0 \\ 1 & 0 & 1 & 1 \\ 1 & 1 & 0 & 1 \end{bmatrix}, \; H_7 = \begin{bmatrix} 1 & 1 & 1 & 1 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & 1 & 1 & 0 \\ 1 & 0 & 1 & 0 & 1 & 0 & 1 \end{bmatrix}$$

Repetition Code Hamming Code

Figure 6: Some parity check and generator matrices.

There's a convenient feature of classical linear codes that allows them to be translated into quantum codes, which is the relationship between inner products of vectors over $\mathbb{Z}_2$ and commutation of certain Pauli operators. Consider a Pauli operator which is of the form $X_S \otimes \hat{\mathbb{1}}_{\bar{S}}$, and another which is of the form $Z_{S'} \otimes \hat{\mathbb{1}}_{\bar{S}'}$, and suppose for the sake of the example that $S \cup \bar{S} = S' \cup \bar{S}' = \mathbf{S}$. We recall that

$$(a \otimes b)(c \otimes d) = ac \otimes bd. \tag{21}$$

This means that our giant tensor product decomposes as follows:

$$\left(\bigotimes_{j \in S} X_j \otimes \bigotimes_{j \in \bar{S}} \hat{\mathbb{1}}_j \right) \left(\bigotimes_{j \in S'} Z_j \otimes \bigotimes_{j \in \bar{S}'} \hat{\mathbb{1}}_j \right) = \bigotimes_{j \in S \triangle (S \cap S')} X_j \otimes \bigotimes_{j \in (S \cap S')} X_j Z_j \otimes \bigotimes_{j \in S' \triangle (S \cap S')} Z_j \otimes \bigotimes_{j \notin (S \cup S')} \hat{\mathbb{1}}_j \tag{22}$$

$$= (-1)^{|S \cap S'|} \bigotimes_{j \in S \triangle (S \cap S')} X_j \otimes \bigotimes_{j \in (S \cap S')} Z_j X_j \otimes \bigotimes_{j \in S' \triangle (S \cap S')} Z_j \otimes \bigotimes_{j \notin (S \cup S')} \hat{\mathbb{1}}_j, \tag{23}$$

therefore a big X Pauli and a big Z Pauli commute if the set $S \cap S'$ is even size. If we represent these sets as bit vectors by putting a 1 on any element in the set, and a 0 on any element outside the set, this is exactly the idea that the inner product of these two vectors should be 0 in order for these Paulis to commute.

If I take some classical code, then, and I replace the 1s of the parity check matrix with Xs or Zs, I get a code that corrects Zs or Xs with the same distance, and has logical Zs or Xs that I can derive from the generator matrix. This construction doesn't always lead to a useful code, like if we tried this with the repetition code (do this in the lecture). If we try this with the Hamming code, though, we get a useful code.

4.3 Homological Codes

Access to 70 years' worth of coding literature is nice, but our technology is pretty limited in terms of what stabilisers we can implement. Also, we'd like to search for Hamiltonians that correct quantum errors when you let them thermalize, and these Hamiltonians ought to be few-body, translationally invariant and geometrically local. This drives us to try to construct CSS codes that incorporate these ideas of locality.

We can start by taking a look at *graphs*, which are mathematical descriptions of networks. The math language for these things is that they are made of two sets, one labeled V (the set of *vertices*, also called *nodes*) and another one E, (the set of *edges*, AKA *links*, each of which is of the form (v, v') with v and v' being elements of V). We can construct commuting X and Z Paulis on networks by putting attaching a qubit to each edge, and associating a Z Pauli

with the edges in each cycle, and an X pauli with the edges which touch a given vertex, and obtain a commuting state. Why? Because every cycle has two edges in common with every vertex: If we construct stabilisers in this

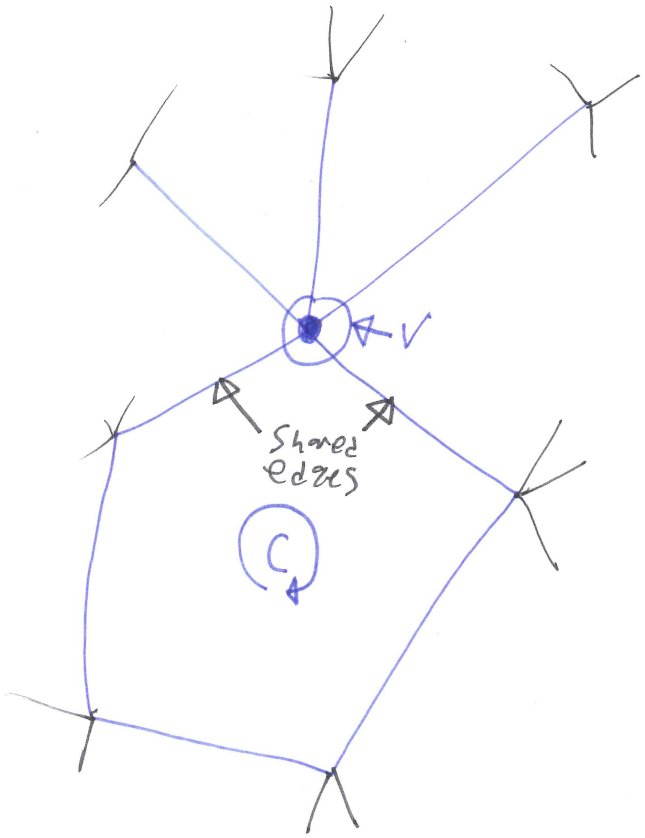

Figure 7: Every cycle shares either 0 or 2 edges with the 'star' associated with a vertex.

fashion, then the 'star' stabilisers are small if the maximum degree of the graph is small, and we can construct large graphs that have small degree. These stabilisers also have a notion of locality baked in. The Z stabilisers are not yet local or small, since I can draw a gigantic cycle over the whole graph if I want. To solve this problem, we choose a specific set of small cycles and label them *faces*. This adds structure to the graph, turning it into a *tiling*. This is even more physical, you have probably actually touched a tiling today.

There's a lot of math literature about tilings, especially *regular* and *semi-regular* tilings, where the tiles are all similar shapes. These tilings are nice to focus on, since their tiles have bounded size, which keeps the stabilisers small and local. The only decision that remains is what kind surface to tile. Here, we can also pick up a result from the math literature, the *Euler characteristic*.

The number of qubits in some tiling-based code is $|E|$. The number of stabilisers is $|V| + |F|$. WAIT! We have to check how many of these stabilisers are *independent*. Each edge has two neighbouring faces and two neighbouring vertices. We know that $\hat{\mathbb{1}}$ is in the stabiliser. However, if we take the product of all the stars, or all the faces, each qubit is hit exactly twice by an X or Z, so there are two products of stabilisers that are 'redundant', they produce something that is already in the stabiliser. Therefore, the number of logical qubits in the code is

$$|E| - |V| - |F| + 2 \tag{24}$$

There's an old result in the theory of tilings that tells us that this number is exactly twice the number of 'holes' or 'handles' in the surface we tile, completely independent of the tiling we chose$_{\text{amazing}}$. The logical operators are just the cycles which are not constructed from tiles, and the length of the minimum size one is called the *systole* (there are algorithms for calculating this). There's also lots of research into tilings of hyperbolic spaces, where the number of holes actually does grow with the size of the tiling, so that the number of logical qubits increases when we increase the size of the code.

4.4 The Surface Code

Let's begin by picking the most boring possible tiling (L-by-L square) of the most boring non-trivial surface (2D torus), and see what we get:

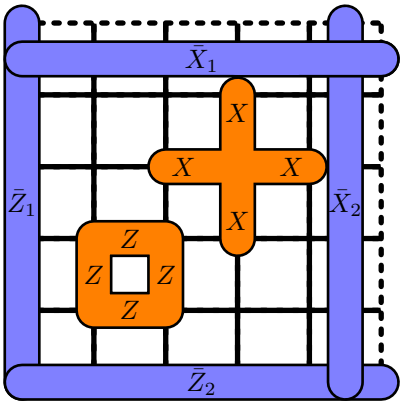

Figure 8: What we get.

This is pretty convenient, it's got small stabilisers that are arranged in an almost planar pattern. We can make it planar by cutting the torus, either in half or just into a rectangle:

Exercise: Draw this.

This is the one code that everyone's trying to optimize and use, because it's the most well-studied. It's also the most well-studied because everyone's trying to optimize it and use it. I hope that the idea gets across that these codes are a subset of a subset of a subset of a subset of the possible codes.

5 What I Haven't Covered

How To Encode A Qubit in Your Experiment : You might have a large spin system, many-level transmon, trapped ion or some other exotic Hilbert space in your laboratory. It comes with its own set of noise maps and its own allowed operations at the physical level (maybe these form a universal gate set, maybe not). What's the best way to encode a qubit into it? Nobody really knows. Most of the time, we make some arbitrary decision, like picking two low-energy levels. It would be interesting to pursue this, though.