# Solution Set #2

CO 639: Quantum Error Correction
Instructor: Daniel Gottesman

Problem 1. Standard Form for Stabilizers

a) When we represent the stabilizer as an $r \times 2n$ matrix, adding rows of the matrix corresponds to multiplying together generators of the stabilizer; replacing a row with its sum with another row corresponds to replacing the generator with its product with another generator. Since the product is also an element of the stabilizer, and it is independent of the other generators, this leads to exactly the same stabilizer — it is merely presented in a different way.

Rearranging qubits of the code — switching the first and third qubits, for instance — corresponds to making the same rearrangement on the columns of the $r \times 2n$ matrix, only we have to rearrange both the $X$ part of the matrix (the left half) and the $Z$ part (the right half). Between this ability and the ability to perform row additions, Gaussian reduction gives us an equivalent QECC to the one we started with.

Let us for the moment restrict attention to the $X$ part of the matrix, and let $s$ be its rank. Then we can choose $s$ linearly independent vectors, which we will put in the first $s$ rows of the matrix. We rearrange columns so that the first row has a 1 in the first column, and then add it to other rows so that all of the others have a 0 in the first column. Then we rearrange columns again so that the second row has a 1 in the second column. We can always do this because it is linearly independent from the first row. We add the second row to other rows (including, if necessary, the first) to make sure all of them have 0s in the second column. We repeat this procedure for all of the $s$ linearly independent rows. Now, since $s$ is the rank of the $X$ part of the matrix, all of the remaining rows are linearly dependent on the first $s$ rows, so the process of making them 0 in the first $s$ columns has actually made them 0 everywhere. (Note that if $s = r$, there might not be any all-0s rows.)

This is the Gaussian elimination procedure, and applying the column rearrangements and row additions to both halves of the matrix can therefore always produce a matrix of the desired form:

$$\left( \begin{array}{cc|cc} I & A & B & C \\ 0 & 0 & D & E \end{array} \right) \tag{1}$$

b) The rows of the full $r \times 2n$ matrix are all linearly independent, so the matrix $(D\ E)$ has maximum rank. Suppose $E$ did not have maximum rank; then by Gaussian elimination, we could make a row of it all 0s. The corresponding row of $D$ is $\vec{d}$. But the stabilizer is Abelian, so we should have 0 for the symplectic inner product of this row $\vec{v}$ with the $j$th row $\vec{m}_j$ of the overall matrix ($j \leq s$). Use the notation $\vec{w}_X$ and $\vec{w}_Z$ to refer to the $X$ and $Z$ parts of a vector. Then we get

$$0 = \vec{v}_X \cdot \vec{m}_{jZ} + \vec{v}_Z \cdot \vec{m}_{jX}. \tag{2}$$

Now, $\vec{v}_X = \vec{0}$, $\vec{v}_Z = (\vec{d}\ \vec{0})$, and $\vec{m}_{jX} = (\vec{e}_j\ \vec{a})$, where $\vec{a}$ is some vector and $\vec{e}_j$ is a vector which is 1 in the $j$th position and 0 elsewhere. Thus, we have that $\vec{d} \cdot \vec{e}_j = 0$, so the $j$th coordinate of $\vec{d}$ is 0. But this is true for $j = 1, \ldots, s$, so $\vec{d} = \vec{0}$, contradicting the fact that $(D\ E)$ has maximum rank.

Thus, $E$ has maximum rank, and by following the same Gaussian reduction procedure as in part a, we can convert it to the matrix $(I\ E')$. This requires column rearrangements and row additions on the $X$ part of the matrix as well, but the row additions do nothing, as the $X$ part of these rows is all 0, and the column rearrangements alter $A$ but do not change the basic form (1). We can then add rows of $E$

to rows of $C$ in order to reduce $C$ to the form $(0\ C')$. These row additions alter $B$, but again do not change the basic structure of the matrix.

c) Suppose we want to find a Pauli matrix with error syndrome $\vec{s}$. Then for coordinates $j = 1, \ldots, s$, we take a $Z$ iff the $j$th coordinate of $\vec{s}$ is 1. For coordinates $j = s+1, \ldots, r$, we take an $X$ iff the $j$th coordinate of $\vec{s}$ is 1. A Pauli matrix of this form will automatically anticommute with the $j$th row iff the $j$th coordinate of $\vec{s}$ is 1, as the standard form of the stabilizer has $X$s or $Z$s (for $j \leq s$ and $j > s$, respectively) in those locations.

Problem 2. Error Syndromes and Cosets

a) Suppose $E$ and $F$ are in the same coset of $N(S)$. Then $E = FN$, with $N \in N(S)$. Let $M \in S$, so $MN = NM$. Then

$$EM = FNM = FMN = (-1)^f MFN = (-1)^f ME, \tag{3}$$

where $FM = (-1)^f MF$. Therefore $E$ and $F$ have the same commutation or anticommutation relationship with all the elements of the stabilizer and therefore have the same error syndrome.

Conversely, if $E$ and $F$ have the same error syndrome, they have the same commutation/anticommutation relationship with all elements of the stabilizer: Let $N = F^\dagger E$, and for some given $M \in S$, suppose $FM = (-1)^f MF$, so $EM = (-1)^f ME$ also. Then

$$NM = F^\dagger EM = (-1)^f F^\dagger ME = (-1)^{2f} MF^\dagger E = MN. \tag{4}$$

Since $M$ was arbitrary, this means that $N \in N(S)$, and therefore that $E = FN$ is in the same coset of $N(S)$ as $F$.

b) Note that when error $F$ occurs and we correct by $E$, we end up with an error syndrome equal to the sum of the two error syndromes. When $E$ and $F$ are in the same coset, the overall error syndrome is therefore 0, meaning we have returned to the code. However, we may have changed the encoded state. Indeed, we have performed $EF \in N(S)$; this is in some coset in $N(S)/S$, which corresponds to some logical Pauli operation on the encoded state.

c) Recall we have the stabilizer

$$\begin{array}{ccccc} X & Z & Z & X & I \\ I & X & Z & Z & X \\ X & I & X & Z & Z \\ Z & X & I & X & Z \end{array} \tag{5}$$

and the logical $\overline{X}$ and $\overline{Z}$ operations can be chosen to be $X \otimes X \otimes X \otimes X \otimes X$ and $Z \otimes Z \otimes Z \otimes Z \otimes Z$.

The error syndrome of $X_1 Z_3$ is 0011, which is the same as the error syndrome of the one-qubit Pauli operation $X_5$. Therefore, the overall operation is $X_1 Z_3 X_5$. We need to figure out which coset $N(S)/S$ this is in. We could solve this systematically using linear algebraic methods (we want to write the vector corresponding to $X_1 Z_3 X_5$ as a sum of the vectors corresponding to the generators of $S$, $\overline{X}$, and $\overline{Z}$), but for a problem this size, it is probably just as easy to figure it out just by looking at it. We note that $X_1 Z_3 X_5$ times $\overline{Z}$ has the form $-Y \otimes Z \otimes I \otimes Z \otimes Y$, which does have the same sort of structure as elements of $S$. Indeed, we can multiply together the first, second, and fourth generators to get $Y \otimes Z \otimes I \otimes Z \otimes Y$. The overall minus sign between the two has no physical significance, so we can conclude that the error $X_1 Z_3$ results in an overall $\overline{Z}$ after correction.

The error syndrome of $Y_2 X_4 Z_5$ is 1111, which is the same as the error syndrome of $Y_4$. The overall operation is thus $Y_2 Z_4 Z_5$, with an overall phase that has no physical significance. Multiplying this by $\overline{X}$ and $\overline{Z}$, we get $Y \otimes I \otimes Y \otimes X \otimes X$, again with some overall phase. We can just get this by multiplying together the second, third, and fourth generators of $S$, so the net operation is a $\overline{Y}$ after correction.

Problem 3. Logical $\overline{X}$ and $\overline{Z}$

a) We can choose $\overline{X}_i$ to be $X_iX_5$, and we can choose $\overline{Z}_i$ to be $Z_iZ_6$ (both for $i = 1, \ldots, 4$. Then $[\overline{X}_i, M] = 0$ and $[\overline{Z}_i, M] = 0$ for $M \in S$, $[\overline{X}_i, \overline{X}_j] = [\overline{Z}_i, \overline{Z}_j] = [\overline{X}_i, \overline{Z}_j] = 0$ for $i \neq j$, and $\{\overline{X}_i, \overline{Z}_i\} = 0$.

b) For a CSS code built from the classical codes $C_1$ and $C_2$, the stabilizer is formed from generators where the 1s in the parity check matrix of $C_1$ have been replaced by $X$s and generators where the 1s in the parity check matrix of $C_2$ have been replaced by $Z$s. Also, for a CSS code, $C_1^\perp \subseteq C_2$ and $C_2^\perp \subseteq C_1$. Finally, the CSS code encodes $k = k_2 - (n - k_1) = k_1 - (n - k_2)$ encoded qubits. That is, there are exactly $k$ independent vectors $\vec{x}_j$ of $C_2$ that are not in $C_1^\perp$ and $k$ independent vectors $\vec{z}_j$ of $C_1$ that are not in $C_2^\perp$.

   We therefore choose $\overline{X}_j$ to be an operator with the 1s of $\vec{x}_j$ replaced by $X$s. Since all these vectors are in $C_2$, the corresponding operators commute with the $Z$ generators formed out of $C_2^\perp$, and none is in $C_1^\perp$, so the operators are outside $S$.

   We actually have some freedom to choose the $\vec{z}_j$s, as we can take any linear combination of them, provided the full set is linearly independent. Since the $\vec{z}_j$s are outside of $C_2^\perp$, they all have nontrivial dot product with at least one of the $\vec{x}_i s$, and indeed, linear independence implies that the dot products $\vec{z}_j \cdot \vec{x}_i$ form a matrix of full rank. Therefore, we can take a linear combination of the $\vec{z}_j$s to product vectors $\vec{z}_j'$ that gives us the identity matrix $\vec{z}_j' \cdot \vec{x}_i = \delta_{ij}$, and we can then choose $\overline{Z}_j$ to be an operator with the 1s of $\vec{z}_j'$ replaced by $Z$s. By construction of the $\vec{z}_j'$s, these operators have the right commutation relationship with everything else, meaning this is a consistent construction of the $\overline{X}_j$s and the $\overline{Z}_j$s.

Problem 4. Creating CSS Codes: Reed-Muller codes

a) Each of the vectors $v_i$ has weight exactly $n/2$, as exactly half the numbers $0, \ldots, n-1$ will be 1 in any given bit location. We can also easily show that any sum of the $v_i$s will have weight exactly $n/2$: The sum will be 0 or 1 in the $j$th location iff the XOR of the corresponding bits of $j$ is 0 or 1. If we look at the XOR of any number of binary variables and run over all possible inputs, we get 0 and 1 equally. But as $j$ runs from $0, \ldots, n-1$, the appropriate bits of $j$ take on all possible sets of values an equal number of times, so exactly half the bits of the sum will be 0 and half will be 1.

   Adding the all-1s vector to any other vector in the code therefore also gives us a vector of weight $n/2$. Thus, the code has distance $n/2 = 2^{m-1}$.

b) $\mathcal{R}(r, m)$ is spanned by a product of vectors for each subset of up to $r$ elements of the numbers $1, \ldots, m$. There are thus

$$1 + \binom{m}{1} + \binom{m}{2} + \cdots + \binom{m}{r} \tag{6}$$

   different products. We do need to show that these are linearly independent to see that this is also the number of encoded bits. We can see this by considering the extreme case of $\mathcal{R}(m, m)$. There are, in this case, $2^m = n$ different products, so to be linearly independent, we should be able to get any vector at all. Indeed, we can imagine arbitrary vectors on $2^m$ bits as functions from $m$ bits to one bit. The product of multiple vectors $v_{i_1}, \ldots, v_{i_s}$ is the AND of the inputs $i_1, \ldots, i_s$, and the sum of products is the XOR of these ANDs. We can write an arbitrary function as the XOR of ANDs of up to all $m$ bits, so all possible vectors are in the code $\mathcal{R}(m, m)$. Therefore all products of the $v_i$s are linearly independent.

   Now we wish to look at the distance. It is easy to see that the product of any $r$ vectors $v_i$ will have weight $n/2^r = 2^{m-r}$, as the product is 1 in the $j$th location iff the AND of the corresponding bits of $j$ is 1, which happens for only a fraction $1/2^r$ of the possible values of $j$. We also have to check the distance, however, for the sums of these products, and it is not clear that taking the sum cannot cause the weight to decrease.

   In fact, I claim the distance of $\mathcal{R}(r, m)$ is indeed $2^{m-r}$. We will show this by induction on $r$ and $m$. As $m \geq r$, we will first show it for a given $r$ for the smallest possible value of $m$, namely $\mathcal{R}(r, r)$. In this case, there is nothing really to show, as the distance from the formula is 1, which is indeed the weight of the product of all $r$ $v_i$ vectors, and there is no possibility of having a shorter distance. We have also shown the formula already for the base cases of $\mathcal{R}(1, m)$.

Suppose now we have shown the above formula for the distance of $\mathcal{R}(r-1, m)$ and $\mathcal{R}(r, m)$, and we wish to show it for $\mathcal{R}(r, m+1)$. We can consider any given sum of basis vectors, and break it up into a term where none of the products includes the vector $v_{m+1}$ and a term where all of the products include $v_{m+1}$. Now, if we restrict attention to the first $2^m$ coordinates, the second term is uniformly 0 and the first term is a vector from $\mathcal{R}(r, m)$, which we already know has weight at least $2^{m-r}$ unless it is the 0 vector.

If we restrict attention to the last $2^m$ coordinates, $v_{m+1}$ is always 1, so it can be ignored, and the second term is the sum of products of at most $r-1$ vectors, and is thus a vector from $\mathcal{R}(r-1, m)$. The first term is an identical copy of the vector on the first $2^m$ coordinates, and is again a vector $\mathcal{R}(r, m)$. But $\mathcal{R}(r-1, m) \subseteq \mathcal{R}(r, m)$, so the sum of a term from $\mathcal{R}(r-1, m)$ and a term from $\mathcal{R}(r, m)$ is in $\mathcal{R}(r, m)$, and therefore has weight at least $2^{m-r}$. If the first term is the 0 vector, we actually have a vector from $\mathcal{R}(r-1, m)$, which therefore has weight at least $2^{m-r+1}$.

That is, the first $2^m$ coordinates of an arbitrary vector can either be all 0, in which case the last $2^m$ coordinates will have weight $2^{m-r+1}$, or the first and last $2^m$ coordinates will each have weight at least $2^{m-r}$. Either way, we know that the overall vector has weight at least $2^{m+1-r}$, completing the induction for the distance.

c) Let us take the dot product of two basis vectors $w \in \mathcal{R}(r, m)$ and $w' \in \mathcal{R}(r', m)$. The dot product is the parity of the pointwise product $ww'$. But $w$ is the pointwise product of up to $r$ of the $v_i$ vectors, and $w'$ is the pointwise product of up to $r'$ of the $v_i$ vectors, so $ww'$ is the pointwise product of up to $r + r'$ of the $v_i$ vectors. Suppose we eliminate redundant $v_i$s that appear twice, and we are left with $s \leq r + r'$ vectors $v_i$ that appear in at least one of the two products $w$ and $w'$. We already know from part b that such a product has weight exactly $2^{m-s}$. Therefore the dot product of $w$ and $w'$ is 0 unless $s = m$, which is only possible if $r + r' \geq m$. Therefore, $\mathcal{R}(m-r-1, m)$ is orthogonal to $\mathcal{R}(r, m)$, and is contained in its dual.

Now, $\mathcal{R}(r, m)$ encodes $N(r, m) = \binom{m}{0} + \binom{m}{1} + \cdots + \binom{m}{r}$ bits by part b, so its dual encodes $2^m - N(r, m)$ bits. But $\mathcal{R}(m-r-1, m)$ encodes

$$\binom{m}{0} + \binom{m}{1} + \cdots + \binom{m}{m-r-1} = \binom{m}{m} + \binom{m}{m-1} + \cdots + \binom{m}{r+1} \tag{7}$$

bits. Therefore, $N(r, m) + N(m-r-1, m) = 2^m$, and $N(m-r-1, m)$ is not only contained in the dual of $N(r, m)$, it is the same size as the dual, and therefore equals the dual.

d) After the previous parts, this one should be easy. We can create a CSS code out of $C_1$ and $C_2$ iff $C_1^{\perp} \subseteq C_2$, so we can have $C_1 = C_2 = \mathcal{R}(r, m)$ iff $m - r - 1 \leq r$. That is, if $2r \geq m - 1$. In that case, $n = 2^m$, $d = 2^{m-r}$, and $k = 2N(r, m) - 2^m$. (We know $d = 2^{m-r}$ instead of being something greater because this code is nondegenerate: the stabilizer is based on the parity check matrix of $\mathcal{R}(r, m)$, which is the generator matrix of $\mathcal{R}(m-r-1, m)$, which has distance $2^{r+1} \geq 2^{m-r}$. Therefore the stabilizer has no elements of low weight.)

Problem 5. Stabilizer Codes via GF(4) Codes

a) To write down the parity check matrix for the 21-register Hamming code over GF(4), we should have three rows and write down as many vectors as we can which are not scalar multiples of each other:

$$\begin{matrix} 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 0 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & \omega & \omega & \omega & \omega & \omega^2 & \omega^2 & \omega^2 & \omega^2 \\ 1 & 0 & 1 & \omega & \omega^2 & 0 & 1 & \omega & \omega^2 & 0 & 1 & \omega & \omega^2 & 0 & 1 & \omega & \omega^2 & 0 & 1 & \omega & \omega^2 \end{matrix} \tag{8}$$

We can easily check that this parity check matrix has symplectic inner product 0 between any two rows: The first two rows differ in 8 places, the first and third rows differ in 8 places, and the second and third rows also differ in 8 places.

b) We must take the above three vectors and $\omega$ times the above three vectors and convert them into Pauli operators using the rule $1 \mapsto X$, $\omega \mapsto Z$, $\omega^2 \mapsto Y$. (We need not take $\omega^2$ times the vectors because, as with the 5-qubit code, that will be the sum of the original plus the $\omega$ multiple, as $1 + \omega = \omega^2$.) We get

$$\begin{array}{ccccccccccccccccccccc}
I & I & I & I & I & X & X & X & X & X & X & X & X & X & X & X & X & X & X & X & X \\
I & I & I & I & I & Z & Z & Z & Z & Z & Z & Z & Z & Z & Z & Z & Z & Z & Z & Z & Z \\
I & X & X & X & X & I & I & I & I & X & X & X & X & Z & Z & Z & Z & Y & Y & Y & Y \\
I & Z & Z & Z & Z & I & I & I & I & Z & Z & Z & Z & Y & Y & Y & Y & X & X & X & X \\
X & I & X & Z & Y & I & X & Z & Y & I & X & Z & Y & I & X & Z & Y & I & X & Z & Y \\
Z & I & Z & Y & X & I & Z & Y & X & I & Z & Y & X & I & Z & Y & X & I & Z & Y & X
\end{array} \tag{9}$$

This is a $[[21, 15, 3]]$ code. It is perfect because there are $1 + 21 \times 3 = 64$ possible errors, and $2^6 = 64$ possible error syndromes.

c) To write down a maximal list of $r$-component vectors in GF(4) with the property that none is a scalar multiple of another, we can follow the pattern of the 21-register Hamming code. The first vector is 0 everywhere but the last component, where it is 1. The next four vectors are 0 everywhere but the last two components. They are 1 in the next-to-last component, and the last component runs over all four possibilities 0, 1, $\omega$, $\omega^2$. In general, we have $4^s$ vectors which are 0 in the first $r - s - 1$ positions, 1 in the $(r - s)$th position, and run over all possibilities in the last $s$ positions. This gives us $1 + 4 + 4^2 + \cdots + 4^{r-1}$ different vectors, and none is a scalar multiple of any other. We can perform the sum

$$\sum_{j=0}^{r-1} 4^j = (4^r - 1)/(4 - 1) = (4^r - 1)/3. \tag{10}$$

Thus, a GF(4) Hamming code exists whenever $n = (4^r - 1)/3$. Since no two of the columns satisfy a linear dependency, the distance of the classical code for which these vectors give the parity check matrix is 3. The classical code is dual using the classical notion, not the quantum notion (the symplectic inner product), but recall that for a linear GF(4) code, the two are equivalent. Therefore, provided the code defines a stabilizer $S$, $N(S)$ corresponds exactly to the classical code with this parity check matrix, and thus contains no nonidentity operators of weight less than 3.

We now need only show that these vectors are symplectically dual to each other. To show that the $i$th row is orthogonal to the $j$th row under the symplectic inner product (with $i < j$), we can break down the $i$th row into three pieces: an initial piece where it is all 0s, a middle piece where it is all 1s, and a final piece where it varies over all possibilities in succession. The first piece occupies the coordinates $1, \ldots, a_i$, with $a_i = 1 + 4 + \cdots + 4^{r-i-1}$. The second piece occupies the coordinates $a_i + 1, \ldots, a_i + 4^{r-i}$. The third piece occupies the remaining coordinates. Recall that when two GF(4) elements are the same, or when one is 0, the symplectic dual will be 0; and when they are different and nonzero, the symplectic dual will be 1. Also, the overall symplectic dual is the sum of the symplectic duals of the coordinates. Below, I will talk of "commuting" and "anticommuting" vectors if the symplectic dual is 0 or 1, respectively, as if the vectors were replaced by the corresponding Pauli operators.

On the first piece of the $i$th row, it is always 0, so will automatically commute with the corresponding coordinates in the $j$th row. Since $j > i$, on the second piece, the $i$th row is always 1 and the $j$th row runs over all possibilities 0, 1, $\omega$, and $\omega^2$ an integral number of times. When the $j$th row is 0 or 1, it therefore commutes with the 1 in the $i$th row, and when the $j$th row is $\omega$ or $\omega^2$, it anticommutes with the 1 in the $i$th row. Since $\omega$ and $\omega^2$ appear an equal number of times, there are an even number of places where the second piece of the $i$th row anticommutes with the corresponding piece of the $j$th row, and therefore those pieces commute overall.

On the third piece, the $j$th row also runs over all possibilities, and does so independently of the $i$th row. In particular, they take on all 2-register possibilities an integral number of times; there are 10 possibilities where the two coordinates will commute (e.g., 01 or $\omega\omega$), and 6 where they anticommute (e.g., $1\omega$). Therefore, the third segment of the $i$th row commutes with the corresponding piece of the $j$th row.

Since all three segments of the rows commute, the complete rows commute, and the parity check matrix can be converted to a stabilizer. As we have argued above, the distance of this quantum code is 3 and it uses $n = (4^r - 1)/3$ qubits. The classical parity check matrix has $r$ rows, so the stabilizer has $2r$ generators (the original rows and the rows multiplied by $\omega$), so the code encodes $n - 2r$ qubits. This is a perfect quantum code: there are $4^r$ error syndromes, and there are $3n + 1 = 4^r$ possible single-qubit errors. As we argued in class, these are the only possible parameters for a single-error-correcting perfect QECC.