Quantum Coding Theory **(UC Berkeley CS294, Spring 2024)**

Lecture 14: Chain Complexes

March 8, 2024

Lecturer: John Wright *Scribe: Angelos Pelecanos*

In the last lecture, we saw surface codes, which allow us to build a quantum code from an arbitrary surface. This is done by choosing some cellulation of the surface and then associating the qubits to the edges, the Z parity checks to the faces, and the X parity checks to the vertices.

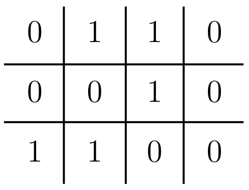
We also saw how some properties of the resulting code were related to the topology of the surface and not to the exact cellulation used. For example, the number of logical qubits was equal to twice the genus of the surface.

In this lecture, we will see how we can use chain complexes, an object from algebraic topology, to study surface codes, but also quantum codes in general.

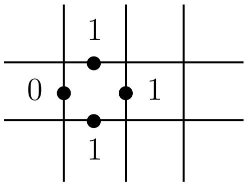
1 Introduction to Chain Complexes

Let us again consider a surface code, and zoom into a small portion of its grid. Then what are objects that we care about?

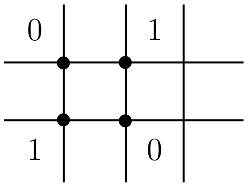
1. General Z parity checks: They correspond to subsets of plaquettes,



2. Codewords: They specify 0 or 1 on the edges,



3. General X parity checks: They correspond to subsets of the vertices,



Turns out that all 3 of these objects are chains.

1.1 Chains and boundary maps

Definition 1.1. Let V be the set of vertices, E the set of edges, and F the set of faces. Then

- A 0-chain is an element $v \in \mathbb{Z}_2^V$. In particular, it assigns to every vertex a 0 or a 1.
- A 1-chain is an element $e \in \mathbb{Z}_2^E$. In particular, it assigns to every edge a 0 or a 1.
- A 2-chain is an element $f \in \mathbb{Z}_2^F$. In particular, it assigns to every face a 0 or a 1.

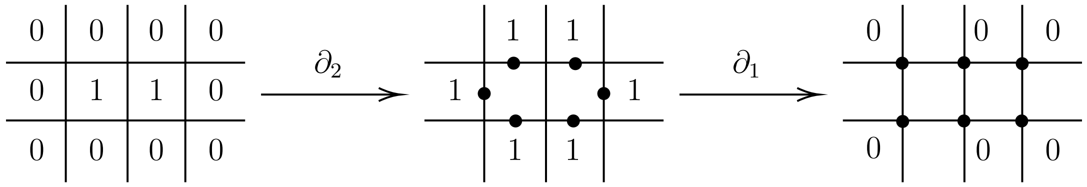
One may wonder, why did we assign the numbers $0, 1$, and 2 to each specific object? It turns out that each number represents the dimensionality of the object. For example, a vertex is a 0-dimensional object and thus belongs to a 0-chain. Similarly, edges are 1-dimensional objects, and faces are 2-dimensional.

We will denote by C_i the set of i-chains. The relation between these chains is captured by the *boundary map.*

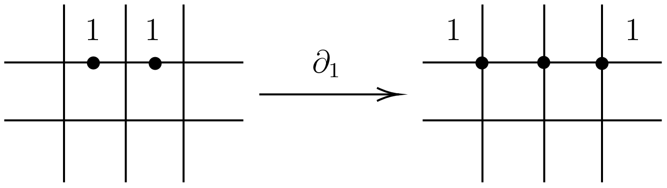
Definition 1.2. The boundary map ∂_2 maps a 2-chain to its 1-chain boundary. Similarly, ∂_1 maps a 1-chain to its 0-chain boundary.

Example 1.3. *Consider the following* 2*-chain* f*. We can compute its boundary map* ∂_2 *to obtain the* 1*-chain boundary* $\partial_2(f)$*, which assigns the value* 1 *to the edges that surround the* 2*-chain faces. We can then apply the boundary map* ∂_1 *to obtain the* 0*-chain boundary* $\partial_1\partial_2(f)$*.*

The boundary of a 1*-chain includes the vertices that only touch one edge. In our case, since* $\partial_2(f)$ *is a cycle, it has no boundary. Thus* $\partial_1\partial_2 f = 0$*.*



Example 1.4. *Consider the following* 1*-chain. We apply* ∂_1 *to get its boundary, which is non-zero.*



If we want to formally define these boundary maps, first let f be the indicator for one face. Then $\partial_2(f) = e$, where e is the indicator of the edges that make up f's boundary. For general f, we extend the definition of ∂_2 via linearity.

We can see that the boundary map depends on the cellulation, which defines the boundary of a face.

Remark. Even though these objects are called *chains*, as we saw in Example 1.4 they do not have to be continuous as our intuition may suggest!

Matrix representations. Since ∂_2 is a linear map, it can be represented by a matrix

$$\partial_2 = \begin{array}{cc} & f\downarrow \\ e\rightarrow & \begin{bmatrix} & & \\ & c_{ef} & \\ & & \end{bmatrix} \end{array},$$

where the entry c_{ef} is 1 if the edge e borders face f in the cellulation. In particular, the column under face f is the indicator of f's boundary.

It is not hard to see that this matrix is the Z parity check matrix H_Z^T. This is because a codeword $c \in C_Z$ is defined on edges and thus $c \in \mathbb{Z}_2^E$. The parity checks imply that for every face, the sum of the bits on its boundary is equal to 0 in $\mathbb{Z}_2$. Note also that this matrix contains a redundant parity check (since the sum of the parity checks of all faces always succeed), but our observation is still valid.

We can similarly represent the linear map ∂_1 as a matrix

$$\partial_1 = \begin{array}{cc} & e\downarrow \\ v\rightarrow & \begin{bmatrix} & & \\ & d_{ve} & \\ & & \end{bmatrix} \end{array},$$

where the entry d_{ve} is 1 if the vertex v is one of the endpoints of edge e. Again, we observe that this matrix is exactly the matrix that contains the X parity checks, H_X.

A fundamental fact in topology is the following:

Fact 1.5. *The boundary of a boundary is zero.*

In the Example 1.3 we saw above, the boundary of a boundary is obtained by applying the maps $\partial_1\partial_2$. Indeed, we saw that this maps to the empty set, zero. This fact should hold for every cellulation of a surface.

More importantly, Fact 1.5 implies something very useful for our code construction. If we replace the maps by their matrix representations, we get

$$\partial_1\partial_2 = 0 \implies H_X H_Z^T = 0 \implies C_Z^\perp \subseteq C_X$$

thus we have a valid CSS code.

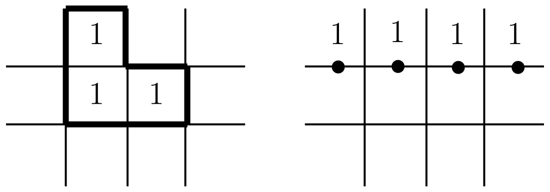


Figure 1: On the left we have the boundary of f, which is the sum of 3 plaquettes / faces. On the right there is a 1-cycle, which is a 1-chain with no boundary

1.2 The homology group

In the past two lectures, we spent some time studying equivalent cycles and cycles that are not boundaries, as they determined the logical operators and the distance of a code. In the framework of chain complexes, we will see how both the logical operators and the distance of a code are captured by the homology group.

Definition 1.6. A 1-boundary is a 1-chain that is of the form $\partial_2(f)$. We denote the set of all 1-boundaries by:

$$B_1 = \{e \mid e \text{ is a 1-boundary}\}.$$

A 1-cycle e is a 1-chain with no boundary, that is $\partial_1(e) = 0$. We denote the set of all 1-cycles by:

$$Z_1 = \{e \mid \partial_i(e) = 0\}.$$

Definition 1.7. Two 1-cycles $e, e' \in Z_1$ are homologically equivalent if $e = e' + b$, for $b \in B_1$ being a boundary. In words, two 1-cycles are equivalent if they only differ by a boundary.

Definition 1.8. The 1^{st} homology group contains the 1-cycles modded out by 1-boundaries, i.e.

$$H_1 = Z_1/B_1.$$

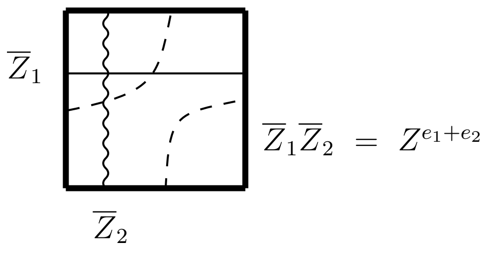
Recall that we can represent a Z error by Z^{e_1}, where e_1 is the 1-chain that specifies which edges contain the error. Then two errros are

$$Z^{e_1}, Z^{e_2} = \begin{cases} \text{same} & e_1, e_2 \in \text{same equivalence class} \\ \text{different} & \text{otherwise} \end{cases}.$$

Example 1.9. *Recall the toric code and the 4 errors that you can apply. Then we can write the elements of the* 1^{st} *homology group as*

$$H_1 = \{0, e_1, e_2, e_1 + e_2\}.$$

Observe that H_1 is an additive group since when you apply two errors, you get a third error that is applied to the "sum" of the edges of the two errors, i.e. $Z^{e_1}Z^{e_2} = Z^{e_1+e_2}$.



In conclusion, the homology group H_1 determines some properties of our code:

- The size of H_1 is the number of distinct Z operations.
- The distance is the minimum weight of a non-zero element of H_1.

2 Analysis of the X parity checks

So far we have spent a lot of time studying the Z parity checks of our CSS code. A very attractive feature of the toric code was that we could replicate our Z parity check analysis to the X parity checks by considering the dual lattice. Could we hope to do something similar?

The definition of a dual can be extended to cellulations other than a square lattice. Taking the dual of an arbitrary cellulation of a $2D$ surface corresponds to introducing a vertex for each face, and connecting two faces that share an edge. The resulting faces then correspond to the vertices of the primal cellulation.

Armed with the dual, let us introduce the chain on the dual of our cellulation. This is called a co-chain, or a dual chain.

Remark. The square lattice on a torus was a particularly nice cellulation, whose dual was the same square lattice on a torus. This may not always be the case, as we saw with the square lattice on the disk. Thus it makes sense to introduce different notation for the dual.

Definition 2.1. Let V be the set of vertices, E the set of edges, and F the set of faces of the dual. Then

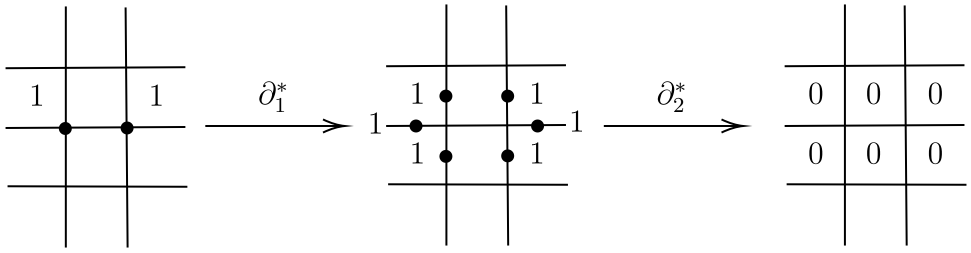
- A 0-cochain is an element $v \in \mathbb{Z}_2^V$.
- A 1-cochain is an element $e \in \mathbb{Z}_2^E$.
- A 2-cochain is an element $f \in \mathbb{Z}_2^F$.

These are the same definitions as in the primal. We denote by C_i^* the set of i-cochains.

We define the analog of the boundary map in the dual to be ∂_i^* that maps an $(i-1)$-cochain to its i-cochain boundary. Note that now we are mapping to a higher-dimensional object.

Notation. In the literature it is typical for people to write ∂^*_{i-1} instead of ∂^*_i.

Example 2.2 (Boundary of a boundary is zero in the dual)**.** *Consider the following* 0*-cochain. Its boundary map will send it to the set of edges that the vertex parity checks involve. Namely, any edge that is incident to an odd number of vertices of the* 0*-cochain will be in its boundary. The boundary of a* 1*-cochain is the set of faces that are incident to an odd number of edges. We can thus verify that even in the dual, the boundary of a boundary is zero.*



Since the boundary map is linear, we can write it as a matrix

$$\partial_1^* = \begin{array}{cc} & d\downarrow \\ e\rightarrow & \left[\begin{array}{c} c_{ev} \end{array}\right] \end{array} = H_X^T = (\partial_1)^T,$$

where c_{ev} is 1 if vertex v is one of e's endpoints. Similarly,

$$\partial_2^* = \begin{array}{cc} & e\downarrow \\ f\rightarrow & \left[\begin{array}{c} d_{fe} \end{array}\right] \end{array} = H_Z = (\partial_2)^T,$$

with d_{fe} being 1 if edge e borders face f.

Notation. The "$*$" symbol on the boundary map may seem like the conjugate transpose. Turns out this is *almost* the case, since the dual boundary maps are just transposes of the boundary maps in the primal.

Like in the Z parity check case,

$$\partial_2^*\partial_1^* = H_Z H_X^T = 0,$$

indeed, the boundary of a boundary is zero even in the dual.

We can repeat the same analysis we did in the previous section and define coboundaries, cocyles, and cohomology groups and establish their relations with the properties of our CSS code.

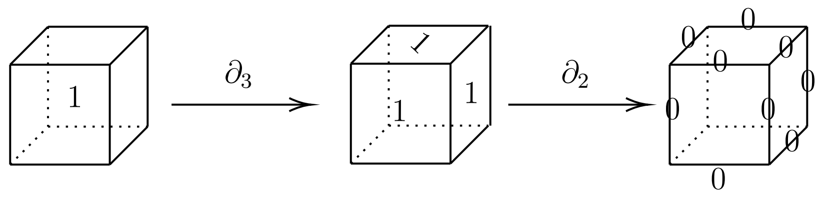


Figure 2: An example of a 3-dimensional chain complex. We first consider the indicator vector for a single cube (3-chain). Apply its boundary map ∂_3 will assign 1 to faces (2-chains) that are adjacent to it. Applying again its boundary map ∂_2 will assign 1 to edges (1-chains) that are adjacent to an odd number of faces. Since every edge is adjacent to 2 faces in this example, we see again that the boundary of a boundary is zero.

3 Extending to Higher Dimensions

So far we have focused on two-dimensional surfaces, however, the chain machinery that we have developed can be very easily extended to higher dimensions. Extending to higher dimensions will enable us to study more complicated objects that may have good error-correction properties.

One can take a d-dimensional surface and cellulate it to d-dimensional cells. Two d-dimensional cells will intersect on a $(d-1)$-dimensional object. These $(d-1)$-dimensional objects will intersect on a $(d-2)$-dimensional object, and so on.

As a result, we get a chain complex $\mathcal{C} = (C_0, C_1, \ldots, C_d)$ with boundary maps $\partial_1, \partial_2, \ldots, \partial_d$. The boundary maps satisfy $\partial_i : C_i \to C_{i-1}$ and $\partial_i\partial_{i+1}$ (the boundary of a boundary) is the zero map.

These properties are enough for (C_{i-1}, C_i, C_{i+1}) to define a CSS code! Namely, we will place Z parity checks on C_{i+1}, X parity checks on C_{i-1}, and the qubits on C_i. The toric code is just a special case with $i = 1$.

This already suggests a concrete direction towards getting better codes, by finding a high-dimensional chain and looking at its resulting CSS code.

Example 3.1. *In the last lecture, we considered the CSS code of the* 3*D torus with an* $L \times L \times L$ *lattice and agreed that it did not give a significant improvement over the toric code. However, the CSS code we constructed last time placed the qubits on the* 1*-chains and the parity checks on the* 0*- and* 2*-chains. Using chains, we can try placing our code on a higher dimension.*

In particular, let's place our qubits on the faces (2-chains). The Z parity checks will correspond to cubes, and each cube checks that its 6 *neighboring faces sum to* 0*. The X parity checks are now edges. Each edge has* 4 *incident faces and it checks whether they also sum to* 0*.*

Is this a good code? It is not hard to compute that the number of qubits is $\approx 3L^3$*, whereas its distance is* $\approx L$ *(since any straight "tube" of length* L *that cycles around the torus satisfies all edge parity checks, or a sequence of* L *parallel edges satisfies all cube parity checks). Thus the distance is the cubic root of the number of qubits, which is worse than the toric code. The* *number of logical qubits can also be computed and is equal to* 3[1]*.*

To deal with the X parity checks, it is useful for one to consider the dual of our $3D$ torus. In this case, the cubes (Z parity checks) become vertices, the faces (qubits) become edges, and the edges (X parity checks) become faces. Thus the dual of this surface is the $3D$ torus we considered last class.

4 Conclusion

It turns out that the higher dimension of the $3D$ torus did not give any improvement to the toric code. This raises the natural question of whether one can come up with a surface that gives a better code than the toric code. Moreover, due to the complexity of implementing a high-dimensional surface code in a quantum machine (how would you arrange the qubits that live on a $4D$ object in the real world?), we would like this surface to be of a low dimension.

The answer for $2D$ surfaces turns out to be negative, as shown by Bravyi, Poulin, and Terhal [BPT10].

Theorem 4.1 ([BPT10])**.** *Any $[[n, k, d]]$-QECC on a $2D$ lattice with local checks satisfies $kd^2 \leq O(n)$.*

Here "local checks" means that the parity checks only involve a constant region of qubits[2].

Since k is at least 1, this means that $d^2 \leq O(n) \implies d \leq O(\sqrt{n})$. Thus the distance of any such code cannot be more than the square root of the number of physical qubits, which is the distance of the toric code. Additionally, if we want our code to handle more qubits, the distance will have to decrease. This is something we saw last time, where one could increase the number of logical qubits by increasing the number of handles in the surface code, at the expense of decreasing the distance relative to the number of physical qubits.

In conclusion, it seems that surface codes cannot improve on the toric code. But the question remains: How can we generalize the toric code to get better parameters? Next class, we will learn about hypergraph product codes (a different generalization of the toric code) that allow us to get codes with good parameters.

[1]The number of logical qubits can be computed by carefully counting the number of independent parity checks. If we ignore independence, the number of qubits is $3L^3$ and the number of X and Z parity checks is $3L^3$ and L^3 respectively. By carefully counting the number of independent X parity checks, we find that there are only $\approx 2L^3$ non-redundant ones. The number of independent Z parity checks is also $\approx L^3$. Thus the number of logical qubits is a constant, 3.

[2]As mentioned in class, one may want to "abuse" this model by adding many qubits in a small region of the surface to make a parity check involve more qubits. Then, to keep the region small w.r.t. the entire surface, the rest of the qubits will have to be very sparse, thus limiting the power of the remaining parity checks.

References

[BPT10] Sergey Bravyi, David Poulin, and Barbara Terhal. Tradeoffs for reliable quantum information storage in 2d systems. *Physical Review Letters*, 104(5), February 2010. 4, 4.1