Estimation of Error Probabilities for the Quantum Repetition Code

by

Jasper Mulder

to obtain the degree of Bachelor of Science in Applied Physics and Applied Mathematics
at the Delft University of Technology.

Student number: 4830709

Thesis committee:

Prof. dr. B.M. Terhal,	TU Delft, supervisor Applied Physics
Prof. dr. A.W. van der Vaart,	TU Delft, supervisor Applied Mathematics
Dr. J.A.M. de Groot,	TU Delft
Dr. A.R. Akhmerov,	TU Delft

TU Delft

Acknowledgements

I would like to thank my supervisors Prof. dr. Barbara Terhal and Prof. dr. Aad van der Vaart for their time, advice and feedback throughout the course of this research. I have learned a lot from both of you. I especially want to thank Boris Varbanov for all the times he helped me when I got stuck on a difficult subject or got stuck in not working code. Thank you for making me enthusiastic for the field of quantum error correction and learning me how to set up a proper project in Python. Lastly, I want to thank Juliette for her patience and support. She was there to listen and help me, even when I was working late at night.

Jasper Mulder
Delft, June 2022

Abstract

In this thesis, the repetition code for bit flip errors is examined. Based the stabilizer measurements outcome of a run of the repetition code, one does not know exactly which errors have occurred. Statistics can be used to estimate the probability of all possible error events. This probability estimation is investigated for simulated data assuming phenomenological and circuit level noise. Experiments on the repetition code are performed by the DiCarlo group at the Delft University of Technology on a superconducting quantum computer. For this data, some error probabilities are estimated to be nonphysical. The goal of this thesis is to investigate these nonphysical probabilities and to determine whether they result from sampling noise or a problem with the assumed error model that determines the estimation of the probabilities. By estimating the standard deviation using the bootstrap method it is shown that the nonphysical probabilities are not due to sampling noise. Therefore, it is possible that non-conventional errors, such as leakage or crosstalk, are affecting these estimates.

In a further analysis on the error probabilities and their standard deviation, it is shown that the standard error for space and time edges in the experiment is consistent with the standard error from phenomenological noise initialised with the average error data and ancilla error probabilities, even when these may be affected by non-conventional errors.

Contents

1

Introduction

Quantum computing promises exponential speedup for certain tasks compared to a classical computer (Feynman, 1982). One example is Shor's algorithm that could solve RSA encryption exponentially faster than any classical algorithm available. However, a practical large-scale quantum computer has not been realized yet. One major problem is that quantum states are vulnerable to environmental interactions, leading to errors on the quantum states. To extend the time over which quantum information can be stored and used to perform fault-tolerant computations, quantum error correction has to be applied continuously. The surface code is currently a popular quantum error correcting code due to its practical implementations as it requires only nearest-neighbor connectivity, can be realized in planar 2D architectures and has demonstrated high fault-tolerant thresholds (Raussendorf and Harrington, 2007; Fowler et al., 2012)

In this thesis, the repetition code for bit flip errors will be examined. The repetition code can be seen as a 1D version of the surface code that corrects only X or Z errors, while the surface code is able to correct both. For quantum error correction with CSS codes, like the surface code or the repetition code, one measures a set of stabilizers. An error on the qubit states leads to a change in one or more stabilizer eigenvalues. Based on the measurement outcome of a run of the repetition code, one does not know exactly which errors have occurred. It is the task of a decoder to find the most likely set of errors that have occurred. Given that errors lead to changes in the stabilizer eigenvalues, one way of estimating the error probabilities is by attempting to extract these values from the statistics on the stabilizer measurements. This research focuses on the estimation of the probabilities for possible error events.

Experiments on the repetition code are performed by the DiCarlo group at the Delft University of Technology on a superconducting quantum computer. When estimating the error probabilities from the measured syndromes, some error probabilities were observed to be nonphysical. The goal of this thesis is to investigate these nonphysical probabilities and to determine whether they result from sampling noise or a problem with the assumed error model that determines the estimation of the probabilities.

First, in Chapter 2, the most important theory necessary for understanding this thesis will be presented. Here, the basic principles of classical and quantum error correction will be introduced. The noise models that will be analysed, are introduced as well. This includes the error rates and coherence times characterized for the quantum chip that is used for the experiments as well. Finally, two statistical methods to estimate the standard deviation of some estimator are presented. In Chapter 3, the equations for estimating the error probabilities will be derived based on the model assumptions. These equations are applied to numerically simulated data and experimental data. Chapter 4 includes further analysis on the error probabilities and their standard deviations. Finally the results will be concluded and discussed in Chapter 5.

This bachelor thesis has been written as part of the double degree Applied Physics and Applied Mathematics at Delft University of Technology.

2

Theory

This chapter provides all theory that is necessary for understanding the research of this thesis. First, an introduction will be given on the repetition code for the classical and the quantum case. Then, the formalism for the quantum repetition code will be made more precise using the stabilizer formalism. Furthermore, it will be explained how the measurements of an experiment are converted into defects. Also, two error models are introduced. Assuming these error models, we have that an error is defined by an edge that connects two defects. In later chapters, the probability of such an edge occurring will be estimated. Finally, two statistical methods are presented to estimate the standard deviation of a certain estimator. These methods will later be used to estimate the standard deviation of the edge probabilities.

2.1. The Repetition Code

In this section, the classical error correction and quantum error correction will be introduced. The repetition code is the simplest example of a classical error correcting code. It can also be applied for quantum error correction, but it can not correct each type of error. Here, the repetition code will be explained for both the classical and quantum setting.

2.1.1. The Classical Model

Say we want to send some information to a receiver. When the data is transmitted through a noisy channel, the receiver could obtain different data than was initially sent. How could the receiver know that the information is wrong? And is it possible to obtain the initial information from this possibly corrupted information? Error correcting codes are a solution to this. The idea is to add redundant information before sending the data. This redundancy allows the receiver to see that an error has occurred and possibly also what the initial information was. We already know this principle from our natural language. When the word ”maphemotics” is read, we know that this word is incorrect. This is because of redundancy. Not every combination of letters is an existing word. We, as humans, are able to see that this word does not exist and we can probably also guess what the word is most likely supposed to be: ”mathematics”. Error correcting codes aim to do the same thing, but now computers have to recognise and correct the errors. In this subsection, the basics for classical error correction will be introduced.

To start, we will describe the mathematical model of a noisy channel. Digital data is encoded in 0’s and 1’s: binary information. A common way to model the noise is to say that there is a probability p for a bit to flip, leading to $0 \to 1$ and $1 \to 0$. This model is called the Binary Symmetric Channel (BSC). The input of the BSC is the encoded message from the sender and the output is the data that will be decoded by the receiver. Note that the reciever in this case does not know the information that is initially sent by the sender. Ideally, the decoded message is equal to the message that is initially sent. This structure is summarized in Figure 2.1.

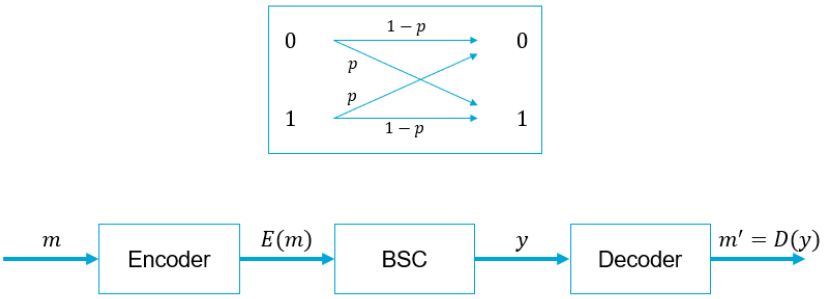


Figure 2.1: Structure of sending and receiving a message m over a Binary Symmetric Channel with error probability p. The message m that will be sent, is first encoded to $E(m)$. This $E(m)$ is then sent through the BSC. The output y is decoded to a message $D(y) = m'$. Ideally, $m' = m$.

Now, there are countless ways to encode information. The most basic error correcting code is the repetition code. As the name suggests, for a repetition code of length d, the information is simply repeated d times. For a repetition code of length 3, every 0 will be encoded as 000 and every 1 as 111. After the data is transmitted through the BSC, there may have occurred some errors on the data. For decoding of the data majority voting is commonly used. Majority voting essentially means that we choose for the option that is most likely to be the case, i.e. the option where the least amount of error have occurred. This means that, for example, 010 will be decoded to a 0 and 110 to a 1. As can be seen, our repetition code with length 3 can correct any corrupted data when at most one error has occurred. When two errors occur on different bits, majority voting results in the wrong outcome. This is called a logical error. For example, when 000 is sent and 101 is received, the message will be corrected to 111. In general, a repetition code of length d can correct up to $\lfloor \frac{d-1}{2} \rfloor$ errors.

Example *Probability of a logical error for the repetition code of length* 3. Say there is a probability p that a bit flip occurs. The chance that there occurs no error on all three bits is then $(1-p)^3$. The probability of one error on any bit is $3p(1-p)^2$, since there are three possibilities of one error occurring. Similarly, the probability of two errors is $3p^2(1-p)$. Finally, the probability that three errors occur is p^3. Therefore, the chance that majority voting gives the wrong outcome, and thus results in a logical error, is $3p^2(1-p) + p^3$. For an error probability of $p = 10\%$, the probability of a logical error for the repetition code of length 3 is 2.8%.

The repetition code is an example of a linear code. Linear error correcting codes are characterised as a $[n, k, d]$-code, where n is the length, k is the dimension and d is the distance of the code. For a repetition code, the dimension is always 1 as the original message consists of 1 bit. The distance of the code is defined as the minimal distance between the code words. For the classical repetition code, this is always equal to the length of the code.

2.1.2. Quantum Model

In the classical world, errors occur during the transmission of data. Quantum systems are sensitive to interactions with the environment. The quantum states of 0 and 1 are defined as two different eigenstates. One example of noise is that the higher energy state can decay into the lower energy state. The process where quantum information gets lost due to the environmental interactions is called decoherence. Due to this continuous decoherence, error correction has to be applied continuously to store quantum information and do calculations with it (Rieffel and Polak, 2011).

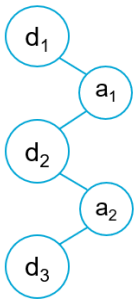
Due to the fundamentals of the quantum world, classical error correction cannot be directly applied to quantum systems. First we have the notion of superposition. A quantum bit, or qubit, can be a 0 and 1 simultaneously. Mathematically, we describe this as a qubit being in a state $|\psi\rangle = \alpha|0\rangle + \beta|1\rangle$, where $\alpha, \beta \in \mathbb{C}$ and $|\alpha|^2 + |\beta|^2 = 1$. Now if we would measure this qubit, it will collapse to $|0\rangle$ with probability $|\alpha|^2$ or to $|1\rangle$ with probability $|\beta|^2$. This directly describes a problem with quantum states. When a

qubit is measured, the state is collapsed to the observed outcome, i.e. the superposition is destroyed. Therefore, information about its true state is lost. For this reason, we do not want to measure the qubits while performing error correction.

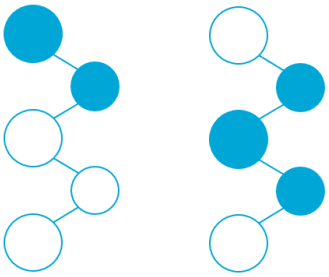
The most simple error correcting code to correct any single bit-flip error is the 3 qubit repetition code, similar to the classical case. The state $|0\rangle$ is encoded as $|000\rangle$ and $|1\rangle$ as $|111\rangle$. Now, as described above, we cannot measure the qubits itself to apply the majority voting. What is allowed, however, is to measure the parity between two qubits. The parity of two qubits can be projected onto an additional qubit, called an ancilla qubit. More details about this indirect measurement will be discussed in Subsection 2.2.3. The qubits that hold the initial data will now be called data qubits. The value of the ancilla qubit will then be the sum, modulo 2, of the states of two adjacent data qubits. So, we have

- ancilla qubit 1 = data qubit $1 \oplus$ data qubit 2, and
- ancilla qubit 2 = data qubit $2 \oplus$ data qubit 3.

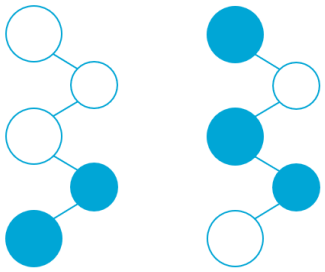
Note that the number of ancilla qubits is one less than the number of data qubits. Figure 2.2 visually explains the idea of the ancilla measurement.



(a) Notation of data qubits d_i and ancilla qubits a_i.



(b) Two different data errors and their corresponding ancilla measurements.



(c) One ancilla measurement corresponds to two different data qubit configurations.

Figure 2.2: Ancilla measurement for the 3 qubit repetition code. Here, a blue dot means a 1 and a white dot means 0. Note, however, that it does not matter whenever a blue/white dot for the data qubits means $1/0$ or defect/no defect, because the ancilla measures only the difference between two data qubits.

Now if we would measure the ancilla qubits, we don't know in which state the data qubits are. However, from the measurement of the ancilla bits, it is possible to apply the majority voting principle. Table 2.1 describes the required decoding step, dependent on the ancilla qubit measurements.

ancilla qubit$_1$	ancilla qubit$_2$	Error correction
0	0	do nothing
0	1	flip data qubit 3
1	0	flip data qubit 1
1	1	flip data qubit 2

Table 2.1: Decoding steps for the 3 qubit repetition code, depending on the outcome of the ancilla measurement.

As described, the repetition code of length 3 can correct any single bit-flip error. This is however not a very practical quantum error correcting code as it cannot correct phase-flip errors. As described before, a qubit can be in a superposition of two states. Two different superposition states are, for example, $|\psi_1\rangle = \frac{1}{\sqrt{2}}|0\rangle + \frac{1}{\sqrt{2}}|1\rangle$ and $|\psi_2\rangle = \frac{1}{\sqrt{2}}|0\rangle - \frac{1}{\sqrt{2}}|1\rangle$. A transition between these two states is what we call a phase flip. The repetition code that is presented above does not recognise these phase-flips. A similar 3 qubit repetition code can however be constructed to correct only phase-flip errors. More

complex error correcting codes are necessary to correct both bit- and phase-flip errors. The simplest of these is the 9-qubit Shor's code where the 3-qubit repetition code for bit-flips is concatenated with the 3-qubit repetition code for phase-flips (Terhal, 2015). For now, it is only necessary to understand the repetition code that corrects bit-flips.

Example *Bit-flip error correction on a superposition.* The superposition $|\psi\rangle = \alpha|0\rangle+\beta|1\rangle$ is encoded as $|\overline{\psi}\rangle = \alpha|000\rangle + \beta|111\rangle$. Now if a bit-flip error occurs on the first data qubit, we obtain the state $\alpha|100\rangle + \beta|011\rangle$. In both cases, the ancilla qubits will measure a difference between the first and second data qubits. Therefore, without knowing the superposition state, we still know that the first data qubit needs to be flipped.

Example *Phase-flip error on a superposition.* Suppose we again have the state $|\overline{\psi}\rangle = \alpha|000\rangle + \beta|111\rangle$. Now if a phase-flip error occurs on the first data qubit, we obtain the state $\alpha|000\rangle - \beta|111\rangle$. Now, the ancilla qubits will not measure a difference between the first and second data qubits. The ancilla measurement therefore suggests that no error has occurred.

2.2. Stabilizer Formalism

In this section, the mathematical model of the class of stabilizer codes will be described. This section is not necessary for the general understanding of the experiment but provides a more formal framework. Additionally, examples for the distance 3 quantum repetition code are provided to explain some abstract concepts. First, a small introduction into quantum computation will be given. Then, the stabilizer formalism will be introduced. This subsection is based on Terhal's "Quantum Error Correction for Quantum Memories" (Terhal, 2015). Finally, the indirect parity measurement will be explained.

2.2.1. Introduction to Quantum Computation

Qubit states are represented as vectors in a two-dimensional Hilbert space, as qubits ideally are two-state quantum systems. The two states form an orthonormal basis for the vector space of the qubit. The general state of a qubit is $|\psi\rangle = \begin{pmatrix}\alpha\\ \beta\end{pmatrix} = \alpha|\phi\rangle + \beta|\phi^{\perp}\rangle$, with $|\alpha|^2 + |\beta|^2 = 1$. Unless mentioned otherwise, we will use the $\{|0\rangle, |1\rangle\}$ basis. Some common qubit states in the $\{|0\rangle, |1\rangle\}$ basis are

$$|0\rangle = \begin{pmatrix}1\\0\end{pmatrix}, \quad |1\rangle = \begin{pmatrix}0\\1\end{pmatrix}, \quad |+\rangle = \frac{1}{\sqrt{2}}\begin{pmatrix}1\\1\end{pmatrix}, \quad \text{and} \quad |-\rangle = \frac{1}{\sqrt{2}}\begin{pmatrix}1\\-1\end{pmatrix}.$$

Here, the ket $|\psi\rangle$, is a column vector using Dirac notation. On the contrary, we have the bra vector $\langle\psi|$, which is the conjugate transpose of the ket vector. An inner product between two states $|\phi\rangle$ and $|\psi\rangle$ is defined to be $\langle\phi|\psi\rangle$. Similarly, an outer product between two states $|\phi\rangle$ and $|\psi\rangle$ is defined to be $|\phi\rangle\langle\psi|$. Now if we would measure a qubit with state $|\psi\rangle$ in the $\{|\phi\rangle, |\phi^{\perp}\rangle\}$ basis, $|\phi\rangle$ is measured with probability $|\langle\phi|\psi\rangle|^2 = |\alpha|^2$. Similarly, $|\phi^{\perp}\rangle$ is measured with probability $|\langle\phi^{\perp}|\psi\rangle|^2 = |\beta|^2$.

Operations that are used to manipulate a qubit can be represented as an unitary matrix[1]. Measurements work slightly different and are represented by a Hermitian matrix[2]. The resulting qubit state, $|\rho\rangle$, after operation O is applied on $|\psi\rangle$ can simply be calculated using standard matrix multiplication, i.e. $O|\psi\rangle = |\rho\rangle$. The Pauli operators describe the errors that are typically modelled to occur on a qubit state: a bit-flip, phase-flip or both a bit- and phase-flip. The Pauli operators, excluding the identity matrix, are given by

$$X = \begin{pmatrix}0 & 1\\ 1 & 0\end{pmatrix}, \quad Z = \begin{pmatrix}1 & 0\\ 0 & -1\end{pmatrix}, \quad \text{and} \quad Y = \begin{pmatrix}0 & -i\\ i & 0\end{pmatrix} = iXZ.$$

Note that for all Pauli operators P, we have $P^2 = I$, where I is the identity matrix. Also, for all Pauli operators P and Q, we have $PQ = \pm QP$. The eigenvalues of all Pauli operators are ± 1.

[1] A unitary matrix U is a matrix such that the conjugate transpose (Hermitian transpose) $U^{\dagger}$ is equal to the inverse of U. So by definition, $UU^{\dagger} = UU^{-1} = I$.

[2] A Hermitian matrix is a matrix Q such that it is equal to the conjugate transpose, so $Q^{\dagger} = Q$.

Example Pauli operators applied on the state $|\psi\rangle = \alpha|0\rangle + \beta|1\rangle$:
- $X|\psi\rangle = \beta|0\rangle + \alpha|1\rangle$,
- $Z|\psi\rangle = \alpha|0\rangle - \beta|1\rangle$,
- $Y|\psi\rangle = i(-\beta|0\rangle + \alpha|1\rangle)$.

When we have a system of k qubits, states are given in a 2^k-dimensional vector space. The combined state can sometimes be represented as a tensor product of the individual qubit states[3]. For example, the product state of three qubits in the $|0\rangle$-state is given as $|000\rangle = |0\rangle \otimes |0\rangle \otimes |0\rangle$. When we now want to apply a Pauli-X gate on the third qubit and identity on the other ones, we take the tensor product of the individual gates. We therefore would apply the operation $O = I_1 I_2 X_3 = I_1 \otimes I_2 \otimes X_3$. This can also be abbreviated as IIX.

2.2.2. Formalism

The idea of stabilizer codes is to encode states $|\psi\rangle$ of k logical qubits into states $|\overline{\psi}\rangle$ of n qubits that are "stabilized" by operators in the stabilizer group $\mathcal{S}$. Here, $|\overline{\psi}\rangle$ denotes the encoded state. For the 3 qubit repetition code we have $|\overline{0}\rangle = |000\rangle$ and $|\overline{1}\rangle = |111\rangle$. An operator is called a stabilizer if it doesn't change the state of a code word, when applied. The code space of a stabilizer code with stabilizer group $\mathcal{S}$ is therefore defined to be

$$C = \{|\psi\rangle : P|\psi\rangle = |\psi\rangle \quad \forall P \in \mathcal{S}\}.$$

Here, $\mathcal{S}$ is an Abelian subgroup of the Pauli group $\mathcal{P}_n$ with $-I^{\otimes n} \notin \mathcal{S}$. The Pauli group is defined to be the group generated by the Pauli operators and $\pm i$, i.e.

$$\mathcal{P}_n = \langle i, I_1, X_1, Z_1, \ldots, I_n, X_n, Z_n \rangle.$$

A stabilizer group $\mathcal{S}$ consists of $n - k$ linearly independent operators that generate the group. This means that instead of defining a code by its code space in a 2^n dimensional vector space, the code can be defined by at most $n - k$ linearly independent operators.

Example *Stabilizers of the repetition code with length* 3. For the repetition code of length 3, we can find that the states in the code space $C = \{|000\rangle, |111\rangle\}$ are invariant under the action of the stabilizers

$$\mathcal{S} = \{III, IZZ, ZIZ, ZZI\} = \langle IZZ, ZZI \rangle.$$

Note that the stabilizer group $\mathcal{S}$ can be generated by two operators. We have $(I_1 Z_2 Z_3)(Z_1 Z_2 I_3) = Z_1 I_2 Z_3$ and $(I_1 Z_2 Z_3)(I_1 Z_2 Z_3) = I_1 I_2 I_3$, since $Z_i Z_i = I_i$. For this repetition code we have $n = 3$ and $k = 1$ because one qubit is encoded as three qubits. Therefore, it is true that $\mathcal{S}$ is generated by $n - k = 2$ generators.

Logical operators are the Pauli operators $\overline{X}, \overline{Z} \in \mathcal{P}_n$ such that $\overline{X}|\overline{0}\rangle \leftrightarrow |\overline{1}\rangle$ and $\overline{Z}|\overline{+}\rangle \leftrightarrow |\overline{-}\rangle$. For a code encoding k qubits, there are k pairs of logical operators $\overline{X}_i, \overline{Z}_i$ with $i = 1, \ldots, k$. Logical operators applied on states in the code space, return states that are in the code space as well. This means that logical operators commute with the stabilizers $s \in \mathcal{S}$. For example $s\overline{X}|\overline{0}\rangle = s|\overline{1}\rangle = |\overline{1}\rangle$ and $\overline{X}s|\overline{0}\rangle = \overline{X}|\overline{0}\rangle = |\overline{1}\rangle$. Due to the fact that logical operators preserve the code space, we have that errors that are logical operators cannot be detected. We define the centralizer $\mathcal{C}(\mathcal{S})$ to be the set Pauli operators in $\mathcal{P}_n$ that commute with $\mathcal{S}$. So, we have

$$\mathcal{C}(\mathcal{S}) = \{P \in \mathcal{P}_n : Ps = sP \quad \forall s \in \mathcal{S}\}.$$

The logical operators are the operators in the set $\mathcal{C}(\mathcal{S}) \setminus \mathcal{S}$. All the other Pauli operators that are not in $\mathcal{C}(\mathcal{S})$, anti-commute with at least one element in $\mathcal{S}$ and therefore map a code word outside the code space. These Pauli operators $P \in \mathcal{P}_n \setminus \mathcal{C}(\mathcal{S})$ are called the detectable Pauli errors E.

[3] A combined state can not be represented as a tensor product of the individual qubit states if the states entangled. For example, $|000\rangle + |111\rangle$.

Example *Logical operators of the repetition code with length* 3. For the repetition, we have $k = 1$. Therefore we have one pair of logical operators; $\overline{X}$ and $\overline{Z}$. For $\overline{X}$ we need to have $\overline{X}|000\rangle = |111\rangle$, so $\overline{X} = X_1X_2X_3$. For $\overline{Z}$ we need to have $\overline{Z}(|000\rangle + |111\rangle) = |000\rangle - |111\rangle$. Therefore $\overline{Z}$ can be $Z_1I_2I_3$, $I_1Z_2I_3$, $I_1I_2Z_3$ or $Z_1Z_2Z_3$. Here we see that logical operators don't have to be unique.

The weight $|P|$ of a Pauli operator $P \in \mathcal{P}_n$ is the number of nontrivial single-qubit Pauli operators. Now, the distance of a stabilizer code is defined to be the minimum weight of the logical operators, i.e.

$$d = \min_{P \in \mathcal{C}(\mathcal{S}) \setminus \mathcal{S}} |P|.$$

Example *Distance of the* 3 *qubit repetition code.* We have seen that the logical operators are $\overline{X} = X_1X_2X_3$ and $\overline{Z} = Z_1$. The minimum weight of these operators is 1. Therefore the distance of the quantum repetition with length 3 is $d = 1$. Note that this is different from the classical repetition code with length 3, where $d = 3$.

The class of stabilizer codes is attractive in use because (1) they are the quantum generalization of classical binary linear codes, (2) their logical operators and distance are easily determined, and it is relatively simple to (3) understand how to construct universal sets of logical gates and (4) execute a numerical analysis of the code performance (Terhal, 2015).

2.2.3. Parity Checks

In the previous subsection it was described that a stabilizer code is defined by the operators in the stabilizer group. These operators are the parity checks that are used to determine the error. The eigenvalues of the parity checks are called the error syndrome and will be projected on the ancilla qubits. A general circuit for measuring a parity check with stabilizer S looks like Figure 2.3a.

Now, why is it 'allowed' to measure the parity where it was not 'allowed' to measure the qubits directly? Measuring a quantum state is equivalent to projecting the state onto one of the two vectors of the measurement basis. So, when a qubit is the superposition $|\psi\rangle = \alpha|0\rangle + \beta|1\rangle$ and we measure it in the $\{|0\rangle, |1\rangle\}$ basis, we essentially force it to be in one of those two states. This means that the information about α and β is lost. When the parity is measured, we also perform a projective measurement. However, by making sure the states from the different qubits remain indistinguishable, the qubit states are not destroyed. So by measuring the parity, just enough information is revealed such that error correction can be applied without losing the information about the encoded state.

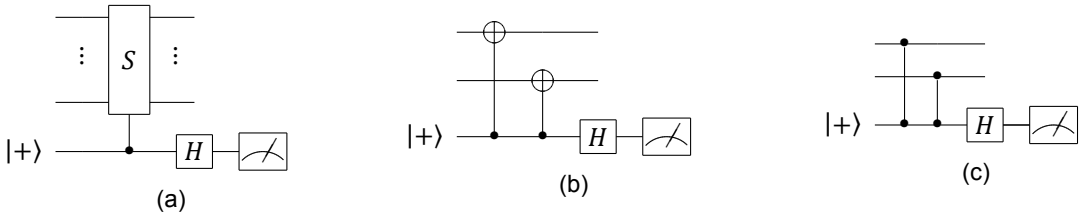


Figure 2.3: Measuring the parity checks in a quantum circuit. (a) Circuit to measure a general stabilizer S using a controlled-S gate. The controlled gate applies I if the control qubit is $|0\rangle$ and applies S when the control qubit is $|1\rangle$. (b) Circuit to apply an XX-check. (c) Circuit to apply a ZZ-check.

What happens to the combined quantum state when measuring a stabilizer using the circuit from Figure 2.3a? Because all stabilizers consists of Pauli operators, the stabilizers also have eigenvalues ± 1. We can decompose the incoming quantum state into the two eigenvectors of S, namely $|\psi_{in}\rangle = \alpha|\psi_+\rangle + \beta|\psi_-\rangle$. Note that $S|\psi_\pm\rangle = \pm|\psi_\pm\rangle$. We will go through the circuit from left to right. The input state, combined with the ancilla, is given by

$$\frac{1}{\sqrt{2}}|0\rangle\big[\alpha|\psi_+\rangle + \beta|\psi_-\rangle\big] + \frac{1}{\sqrt{2}}|1\rangle\big[\alpha|\psi_+\rangle + \beta|\psi_-\rangle\big].$$

Now, the controlled-S gate applies stabilizer S on the input state if the ancilla qubit is $|1\rangle$. Therefore, the state becomes

$$\frac{1}{\sqrt{2}}|0\rangle\big[\alpha|\psi_+\rangle + \beta|\psi_-\rangle\big] + \frac{1}{\sqrt{2}}|1\rangle\big[\alpha|\psi_+\rangle - \beta|\psi_-\rangle\big].$$

Finally, the Hadamard gate changes the ancilla qubit such that $|0\rangle \to |+\rangle$ and $|1\rangle \to |-\rangle$. When also reorganising the terms, we obtain.

$$\frac{1}{\sqrt{2}}|+\rangle\big[\alpha|\psi_+\rangle + \beta|\psi_-\rangle\big] + \frac{1}{\sqrt{2}}|-\rangle\big[\alpha|\psi_+\rangle - \beta|\psi_-\rangle\big] = |0\rangle\alpha|\psi_+\rangle + |1\rangle\beta|\psi_-\rangle.$$

Here, we can see that when the ancilla is measured, the state collapses to one of the eigenvectors of S. This means that a superposition between the two states is lost. However, this is now actually a good thing. With this projection we get rid of a possible superposition of having an error/no error.

Example *Direct measurement of the ZZ-check.* Say we have the following input of the data qubits:

- $|\psi_{d1}\rangle = a|0\rangle + b|1\rangle$, and
- $|\psi_{d2}\rangle = c|0\rangle + d|1\rangle$.

Of course, it holds that $|a|^2+|b|^2 = |c|^2+|d|^2 = 1$. First, say we don't apply the ancilla measurement and just do a direct measurement. When measuring $|\psi_{d1}\rangle$, there is a $|a|^2$ chance the state collapses to $|0\rangle$ and a $|b|^2$ chance the state collapses to $|1\rangle$. The same holds for $|\psi_{d2}\rangle$ with $|c|^2$ and $|d|^2$. Therefore, the probability to measure $|00\rangle$ is $|a|^2|c|^2 = |ac|^2$. Similarly, the probability to measure $|01\rangle$ is $|ad|^2$, etc. Therefore, if the parity is measured by measuring the qubits individually, the outcome would be even ($|00\rangle$ or $|11\rangle$) with probability $|ac|^2 + |bd|^2$ and odd ($|01\rangle$ or $|10\rangle$) with probability $|ad|^2 + |bc|^2$. Note that after this direct measurement, $|\psi\rangle$ has collapsed to one of the four states $|00\rangle$, $|01\rangle$, $|10\rangle$ or $|11\rangle$.

Example *Indirect measurement of the ZZ-check.* Now, say we measure the parity indirectly using the ancilla qubit according to Figure 2.3c. Assume the same input as for the direct measurement. The input state, including the ancilla qubit, is given by

$$\frac{1}{\sqrt{2}}|0\rangle \otimes \big((a|0\rangle + b|1\rangle) \otimes (c|0\rangle + d|1\rangle)\big) + \frac{1}{\sqrt{2}}|1\rangle \otimes \big((a|0\rangle + b|1\rangle) \otimes (c|0\rangle + d|1\rangle)\big),$$

where the first qubit denotes the ancilla qubit and the other two denote the data qubits. The controlled-Z gates now give the state

$$\frac{1}{\sqrt{2}}|0\rangle \otimes \big((a|0\rangle + b|1\rangle) \otimes (c|0\rangle + d|1\rangle)\big) + \frac{1}{\sqrt{2}}|1\rangle \otimes \big((a|0\rangle - b|1\rangle) \otimes (c|0\rangle - d|1\rangle)\big).$$

Finally, the Hadamard gate rotates the ancilla qubit such that

$$\frac{1}{\sqrt{2}}|+\rangle \otimes \big((a|0\rangle + b|1\rangle) \otimes (c|0\rangle + d|1\rangle)\big) + \frac{1}{\sqrt{2}}|-\rangle \otimes \big((a|0\rangle - b|1\rangle) \otimes (c|0\rangle - d|1\rangle)\big).$$

Reorganising some terms gives

$$|0\rangle \otimes (ac|00\rangle + bd|11\rangle) + |1\rangle \otimes (ad|01\rangle + bc|10\rangle).$$

Note that for this state it holds that the ancilla qubit is measured to be $|0\rangle$ with a probability $|ac|^2 + |bd|^2$ and $|1\rangle$ with a probability $|ad|^2 + |bc|^2$. This is the same result as for the direct measurement. However, in this case we do not know whether the individual qubits are in $|0\rangle$ or $|1\rangle$.

2.3. Defect Analysis for Different Noise Models

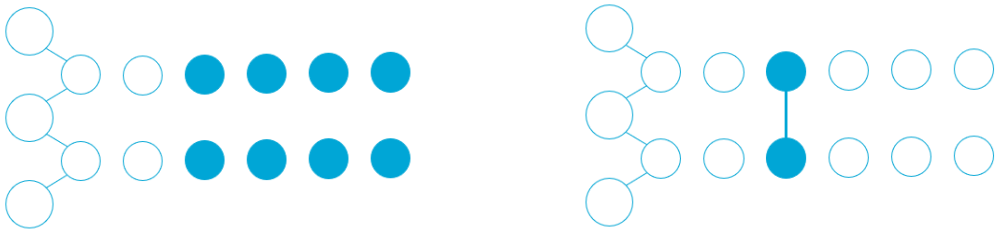
In Subsection 2.1.2, the idea of ancilla measurement is introduced in the context of the repetition code. Implicitly it was assumed that errors only occur on the data qubits. However, the ancillas are also qubits. Therefore, errors could occur on them as well. Now if an ancilla is measured to be 1, how can we know if the the corresponding data qubit(s) have an error or if the ancilla qubit has an error? To find out what kind of error has occurred, we will measure the parity multiple times. Each cycle the parity of the data qubits is measured, is called a quantum error correction round, or QEC round.

The measurements of the error syndromes over multiple QEC rounds are converted into a so-called defect matrix. This defect matrix will later be used to estimate the probability of certain error occurring. In this section, the defect matrix will be introduced on the basis of two different error models. For the first model it is assumed that there may occur errors at the start of each QEC round with some probability. For the second model it is assumed that error may occur after the logic gates performing the operations to measure the syndrome. For both models, only Pauli errors are taken into account. This means that there is some probability that one of the Pauli operators is applied on a qubit.

2.3.1. Phenomenological Noise

A simple error model one can think of is to assume that bit-flip (Pauli-X) errors may occur on the data and ancilla qubits at the start of each QEC round with some probability p[4]. We also assume that the quantum gates performing the operations during the QEC round work perfectly. This noise model is called phenomenological noise.

When the parity is measured multiple times, and the error probability is low, we would expect to measure the same outcomes multiple times in the event of a data error. Since we are only interested in the change an error makes in the measurements, it makes sense to take the time difference of the measurements. We call these time differences the defects syndromes. The defects are computed by $d_{a,r} = m_{a,r} \oplus m_{a,r-1}$, where a indicates the ancilla qubit and r indicates the QEC round. $m_{a,-1}$ is defined to be the parity of the initial data qubit state (Varbanov et al., 2020). Here, it is assumed that the ancilla qubits are reset after each QEC round. Figure 2.4 demonstrates the ancilla measurement over multiple QEC rounds and the corresponding defect syndrome of a single data qubit error. Figure 2.5 demonstrates the ancilla measurement over multiple QEC rounds and the corresponding defect syndrome of a single ancilla qubit error.



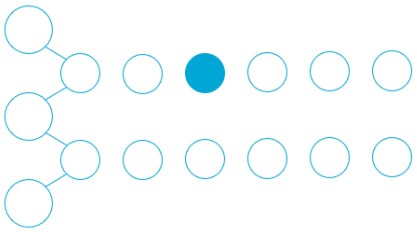
(a) Ancilla measurement over multiple QEC rounds of a single data error at data qubit 2.

(b) Defect syndrome of a data error. The data error corresponds to a 'space edge'.

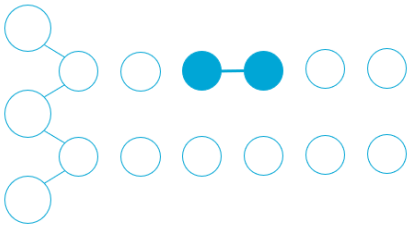
Figure 2.4: Ancilla measurement and defect syndrome for a data error at data qubit d_2 before QEC round 3. For the ancilla measurement it is assumed that the qubits are reset after each QEC round.

It can be seen in Figure 2.4 and 2.5 that the defects come in pairs. These pairs of defect can happen in the same round on neighbouring qubits, leading a space edge, or on the same qubit at two following QEC rounds, leading to a time edge. One exception is when an error occurs on a data qubit that is located at the boundary. An example of this is shown in Figure 2.6. In this case, the defect will be connected to the boundary. In total we therefore have three types of edges: space and time edges, which together are called bulk edges, and boundary edges.

[4] An error at the start of a QEC round means that the error occurs before a single gate of the QEC round has been applied.

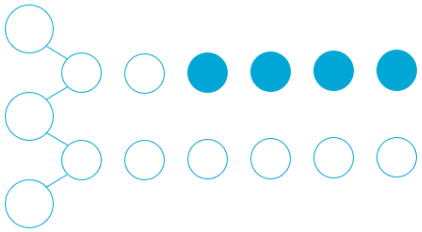


(a) Ancilla measurement over multiple QEC rounds of a single ancilla error at ancilla qubit 1.

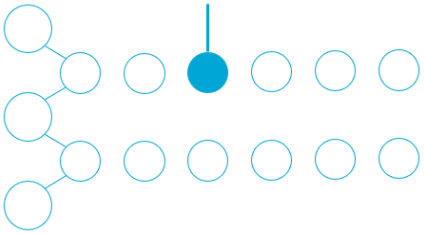


(b) Defect syndrome of an ancilla error. The ancilla error corresponds to a 'time edge'.

Figure 2.5: Ancilla measurement and defect syndrome for an ancilla error at ancilla qubit a_1 before QEC round 3. For the ancilla measurement it is assumed that the qubits are reset after each QEC round.



(a) Ancilla measurement over multiple QEC rounds of a single data error at data qubit 1.



(b) Defect syndrome of an data error at the boundary. The ancilla error corresponds to a boundary space edge.

Figure 2.6: Ancilla measurement and defect syndrome for an data error at data qubit d_1 before QEC round 3. For the ancilla measurement it is assumed that the qubits are reset after each QEC round.

When constructing the defect syndrome, it was assumed that the ancilla qubits were reset after every QEC round. However, in practice, superconducting qubits are hard to reset quickly with high-fidelity (McEwen et al., 2021; Egger et al., 2018; Magnard et al., 2018). Therefore, an extra calculation step is required to construct the same defect syndromes. A sequence of ancilla measurements "1111" will look like "1010" when the qubits are not reset after a measurement. Now, the syndrome can be calculated[5] by $s_{a,i} = m_{a,i-1} \oplus m_{a,i}$. Then the defects can be calculated again using $d_{a,i} = s_{a,i-i} \oplus s_{a,i} = m_{a,i-2} \oplus m_{a,i}$. Here, $m_{a,-1}$ is set to be 0 and $s_{a,-1}$ is set to be the parity of the initial state of the data qubits (Varbanov et al., 2020). After N_r quantum error correction rounds, the data qubits are measured to construct the final syndrome. Therefore, the syndrome is calculated $N_r + 1$ times. Note that the final measurement can obtain errors as well. This is modeled using a last bit-flip channel with probability p on the data qubits. The QEC experiment model with phenomenological noise is summarized in Figure 2.7.

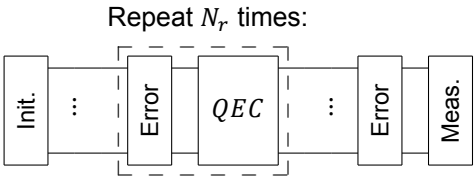


Figure 2.7: QEC experiment for phenomenological noise. The experiment starts with initialising all the qubits in a certain state. Then, N_r rounds of quantum error correction are applied. Because of the phenomenological noise model, there are incoming errors to a perfect QEC circuit. Finally, the data qubits are measured to give the final syndrome.

[5] If the ancilla qubits would be reset after each round, then the syndrome would be equal to the measurement.

For one experiment of a repetition code of length d and N_r QEC rounds will now return a defect matrix of size $(d-1) \times (N_r+1)$. During the experiment, multiple data and ancilla errors will occur. All those errors that occur will result in an edge between two vertices or an edge from a vertex to the boundary. However, from the experiment we only obtain the defect syndromes. The goal will therefore be to use statistics to estimate the probability of all the possible edges. These probabilities can then be used to guess what errors actually occurred on the qubits. Before the formulas for the edge probabilities will be derived, a more accurate noise model will be introduced.

2.3.2. Circuit Level Noise

With phenomenological noise, it was assumed that errors may occur before a QEC round and that the operations in the QEC round work perfectly. In reality, however, the latter is not true. It is therefore possible to construct a noise model that better represents reality. Errors will now be assumed to occur with some probability at the end of each gate, based on the actual performance of operations on the individual qubits. Still, only Pauli errors are assumed. This model is called circuit level noise. The error probabilities can be based on the probabilities of the individual qubits.

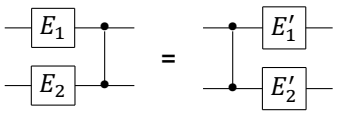
Before the circuit of a QEC round is introduced, it will be explained how errors propagate through a circuit. When an error occurs an a certain qubit, it is possible that the error will also propagate to another qubit via the applied two-qubit gates. This also holds for classical data. An easy example is the controlled NOT gate. A controlled NOT gate flips the output of the second bit if and only if the input of the first bit is a 1, which is also shown in Table 2.2. Here, we can see that an error on the target bit does not change the output of the control bit. However, when an error occurs on the control bit, then the output of the target bit also obtains the error. The error is propagated onto the other bit as well.

Input		Output	
Control	Target	Control	Target
$\|0\rangle$	$\|0\rangle$	$\|0\rangle$	$\|0\rangle$
$\|0\rangle$	$\|1\rangle$	$\|0\rangle$	$\|1\rangle$
$\|1\rangle$	$\|0\rangle$	$\|1\rangle$	$\|1\rangle$
$\|1\rangle$	$\|1\rangle$	$\|1\rangle$	$\|0\rangle$

Table 2.2: Input/Output table of a controlled-NOT gate.

In the case of the repetition code that corrects bit flips, the only two-qubit gates used are controlled-Z gates. For the controlled-Z gates, only X errors propagate onto the other qubit in the form of a Z error. With the bit-flip repetition code, we cannot detect Z errors. Therefore, error propagation does not have to be taken into account for this error correcting code. Circuit level noise does however give other types of errors than the phenomenological noise model. To understand these type of errors, the QEC circuit has to be explained first.

Proof *Errors propagating through a controlled-Z gate.* We want to know how errors propagate through a certain two-qubit gate. To find this, we can say Pauli errors E1 and E2 appear before the two qubit gate. We then want to know what the errors E1' and E2' are after the gate. We can set the following two circuits to be equivalent:



Now, we will determine E_1' and E_2' for certain inputs E1 and E2. First, note that a controlled-Z gate (CZ) applied twice is equivalent to the identity operation. Therefore, we have

$$CZ \cdot E1E2 = E1'E2' \cdot CZ \quad \Longrightarrow \quad E1'E2' = CZ \cdot E1E2 \cdot CZ.$$

For convenience, the matrix representation for the operators will be used. The operators we need are represented as

- $I = |0\rangle\langle 0| + |1\rangle\langle 1|$
- $X = |0\rangle\langle 1| + |1\rangle\langle 0|$
- $Z = |0\rangle\langle 0| - |1\rangle\langle 1|$
- $CZ = |0\rangle\langle 0| \otimes I + |1\rangle\langle 1| \otimes Z = I \otimes |0\rangle\langle 0| + Z \otimes |1\rangle\langle 1|$.

Now assume the input to be $E_1 E_2 = IX$. Then

$$\begin{aligned} E_1' E_2' = CZ \cdot IX \cdot CZ &= (|0\rangle\langle 0| \otimes I + |1\rangle\langle 1| \otimes Z)(I \otimes X)(|0\rangle\langle 0| \otimes I + |1\rangle\langle 1| \otimes Z) \\ &= (|0\rangle\langle 0| \otimes I + |1\rangle\langle 1| \otimes Z)(|0\rangle\langle 0| \otimes X + |1\rangle\langle 1| \otimes XZ) \\ &= |0\rangle\langle 0| \otimes X + |1\rangle\langle 1| \otimes ZXZ \\ &= |0\rangle\langle 0| \otimes X - |1\rangle\langle 1| \otimes X = Z \otimes X. \end{aligned}$$

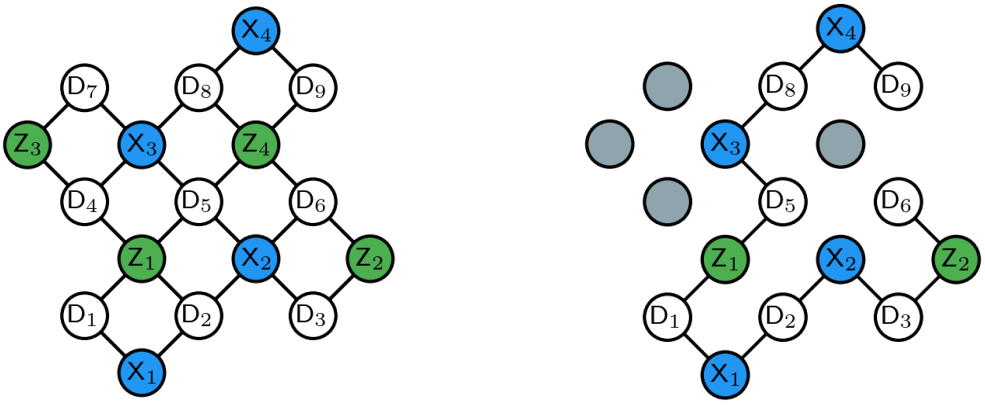
So the X error is unchanged on the second qubit and the first qubit obtains a Z error. Since the controlled-Z gate is symmetric, we also have that $E_1 E_2 = XI$ results in $E_1' E_2' = XZ$.

Now assume the input to be $E_1 E_2 = IZ$. Using similar calculation we obtain

$$\begin{aligned} E_1' E_2' = CZ \cdot IZ \cdot CZ &= (|0\rangle\langle 0| \otimes I + |1\rangle\langle 1| \otimes Z)(I \otimes Z)(|0\rangle\langle 0| \otimes I + |1\rangle\langle 1| \otimes Z) \\ &= (|0\rangle\langle 0| \otimes I + |1\rangle\langle 1| \otimes Z)(|0\rangle\langle 0| \otimes Z + |1\rangle\langle 1| \otimes I) \\ &= |0\rangle\langle 0| \otimes Z + |1\rangle\langle 1| \otimes Z = I \otimes Z. \end{aligned}$$

Therefore, the Z error does not propagate through a controlled-Z gate. Again, by symmetry, an input of $E_1 E_2 = ZI$ results in $E_1' E_2' = ZI$. Since Y is a combination of X and Z, we do not need to work out the propagation of a Y error. ⊠

The quantum chip that will be used for the repetition code experiment, is from the DiCarlo Lab at Delft University of Technology. This chip is initially is designed to run the surface code. This is a 2D extension of the repetition code that is able to correct both bit- and phase-flip errors. The layout of the chip is shown in Figure 2.8a. The data qubits are labeled by D_i and the ancilla qubits are labeled X_i and Z_i. The X/Z label of an ancilla indicate whether the ancilla performs an X- or Z-check on the data qubits. To run the repetition code, some data and ancilla qubits can be connected in a specific way to construct a string of qubits. This setup is shown in Figure 2.8b. Note that the ancilla qubits are labeled X/Z just to indicate their parity check for the surface code. This does not mean a Z-ancilla cannot perform an X-check.



(a) (b)

Figure 2.8: Layout of the quantum chip from the DiCarlo group for (a) the surface-17 code and (b) the repetition code of length 7. The labels indicate the data qubits D_i and ancilla qubits X_i/Z_i.

For this experiment we look at the repetition with length 7, as can be seen in Figure 2.8b. So there are 7 data qubits and 6 ancilla qubits. The stabilizer group of this error correcting code is given by

$$\langle Z_1Z_2, Z_2Z_3, Z_3Z_4, Z_4Z_5, Z_5Z_6, Z_6Z_7 \rangle.$$

The circuit of a single ZZ-check was shown in Figure 2.3c. The full circuit is given in Figure 2.9. One can check that this layout corresponds with Figure 2.8b. For now, we will only focus on the ZZ-checks, not the X gates. If an error appears on a data qubit, say on D_1 before QEC round n (but after $n-1$), then the ancilla qubits X_1 and Z_1 detect that error in round n, which results in a space edge at round n. Now assume an error occurs on D_1, during QEC round n after the first ZZ-check ($Z_{D_1}Z_{D_5}$), but before the second ZZ-check ($Z_{D_2}Z_{D_1}$). In this case, X_1 detects the error in round n but Z_1 detects the error in round $n+1$ since the error happened after the check of round n. The edges that represent these errors are called space-time edges, since these edges connect nodes of a different ancilla and different QEC round. The direction of the space-time edges is determined by the order of the ZZ-checks in the circuit. For the circuit level noise we now have 4 types of edges: space, time and space-time edges (called the bulk edges) and boundary edges. These edges are illustrated in Figure 2.10.

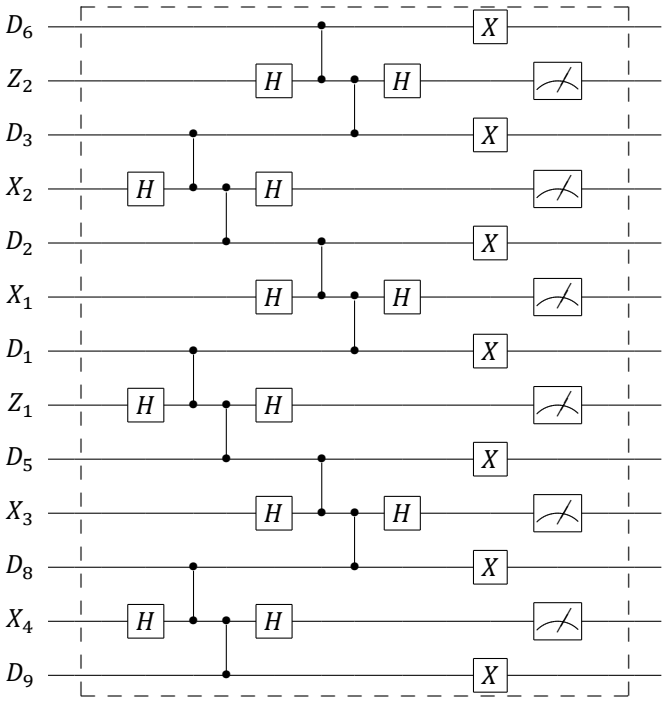
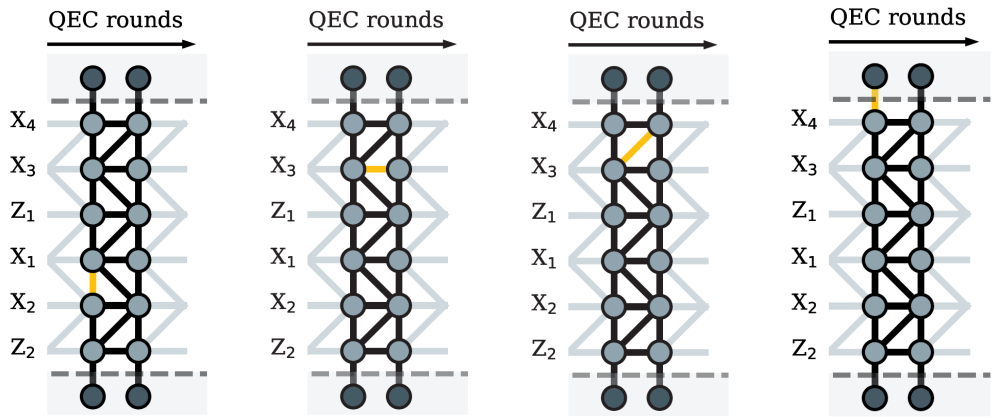


Figure 2.9: Quantum circuit for one QEC round. Each round, the parity of a ZZ-check between two data qubits is projected on the ancilla between them. At the end of each round, the ancilla is measured. An X-gate is applied to change the data qubits states.



(a) Space edge between ancilla X_1 and X_2.

(b) Time edge on ancilla X_3.

(c) Space-time edge between ancilla X_4 and X_3.

(d) Boundary edge on ancilla X_4.

Figure 2.10: The grid shows the possible edges between nodes for circuit level noise. The four edge types that appear from errors are highlighted to be yellow.

The error rates of the qubit gates are initialised to be the values obtained from randomized benchmark experiments. The gate error rates p_{gate} for single qubit gates can be found in Table 2.3. The error rates of all the two-qubit gates, p_{gate}, that are used can be found in Table 2.4. This information is provided by the DiCarlo Lab. The gate errors are modelled using a depolarizing channel with parameter p after the gate. The parameter is determined by $p = \frac{d}{d-1} p_{gate}$, where d is the dimension of the Hilbert space (Magesan et al., 2012). For a single qubit depolarizing channel, Pauli errors X, Y and Z occur each with probability $p/3$. For a two qubit depolarizing channel, each non-identity two qubit Pauli error occurs with probability $p/15$.

The assignment error p_{assign} is the probability of observing an incorrect measurement outcome and the post measurement error p_{QND} is the probability of an error after the measurement (which is only relevant for the ancilla qubits). The gate errors are modeled using a bit-flip channel with parameter p_{assign} after the readout and a bit-flip channel with parameter p_{QND} after the measurement.

Qubit	T_1 (ns)	T_2 (ns)	Gate error p_{gate}	Assignment error p_{assign}	Post meas. error p_{QND}
D_1	13973	7181	0.0011	0.0281	-
D_2	14460	12197	0.0016	0.1182	-
D_3	10244	15769	0.0061	0.0275	-
D_5	11508	10684	0.0009	0.0135	-
D_6	5858	9737	0.0033	0.0216	-
D_8	11273	14264	0.0024	0.0502	-
D_9	18161	14397	0.0041	0.0563	-
X_1	9902	6587	0.0027	0.031	0.0906
X_2	12423	13410	0.0050	0.0156	0.0914
X_3	8717	11483	0.0032	0.0182	0.0601
X_4	7058	11125	0.0033	0.0302	0.0349
Z_1	5105	4471	0.0093	0.0198	0.0533
Z_2	10612	18038	0.0022	0.0381	0.0383

Table 2.3: Characteristics of the qubits used in the repetition code experiment, provided by the DiCarlo Lab.

qubit 1	qubit 2	Gate error p_{gate}
D_1	X_1	0.0179
D_2	X_1	0.0216
D_2	X_2	0.0259
D_3	X_2	0.0143
D_3	Z_2	0.0124
D_6	Z_2	0.0334
D_1	Z_1	0.0270
D_5	Z_1	0.0256
D_5	X_3	0.0340
D_8	X_3	0.0149
D_8	X_4	0.0180
D_9	X_4	0.0290

Table 2.4: Error probabilities of gates between the pairs of qubits used in the repetition code experiment, provided by the DiCarlo Lab.

A qubit can, over time, decay from the excited state $|1\rangle$ to $|0\rangle$. Furthermore, it can acquire random phase shifts. This relaxation and dephasing are described by the decoherence times T_1 and T_2. These values can also be found in Table 2.3. We can describe this process by the amplitude and phase damping channels, corresponding to the Pauli transfer matrices (O'Brien et al., 2017)

$$R_{T_1} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & \sqrt{1-p_1} & 0 & 0 \\ 0 & 0 & \sqrt{1-p_1} & 0 \\ p_1 & 0 & 0 & 1-p_1 \end{pmatrix}, \tag{2.1}$$

$$R_{T_\phi} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & \sqrt{1-p_\phi} & 0 & 0 \\ 0 & 0 & \sqrt{1-p_\phi} & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}. \tag{2.2}$$

The probabilities p_1 and p_ϕ are defined to be $p_1 = 1 - e^{-t/T_1}$ and $p_\phi = 1 - e^{-t/T_\phi}$, where

$$\frac{1}{T_\phi} = \frac{1}{T_2} - \frac{1}{2T_1}.$$

Now, the amplitude and phase damping is approximated as a depolarizing channel using the Pauli twirling approximation (Geller and Zhou, 2013). This approximation essentially takes the diagonal elements of the R_{T_1} and R_{T_ϕ} matrices. The parameters p_X, p_Y, p_Z of the depolarizing channel are given by

$$p_X = \frac{p_1}{4}, \quad p_Y = \frac{p_1}{4}, \quad \text{and} \quad p_Z = \frac{1}{2} - \frac{p_1}{4} - \frac{\sqrt{1-p_1-\lambda}}{2}. \tag{2.3}$$

Here, λ is defined to be $\lambda = e^{-t/T_1}\left(1 - e^{-2(t/T_\phi)^2}\right)$. We can now define the depolarizing channel in terms of the T_1 and T_2 times and the running duration of certain gates.

The duration of a Hadamard gate and an X-gate are approximated to be $t = 20$ ns, the duration of a controlled-Z gate is approximated to be $t = 40$ ns and the duration of a measurement is approximated to be $t = 800$ ns. These times are based on experimental performance.

Since the amplitude damping is asymmetric, an X gate is applied on all data qubits each round to prevent the qubits from decaying to $|0\rangle$ as time goes on. Note that the X gates do not change the parity of the data qubits (when of course applied on all data qubits), so they do not make a difference in the ancilla measurements.

As a final note, it should be mentioned that the logical states $|\overline{0}\rangle$ and $|\overline{1}\rangle$ are often not encoded as $|00\cdots0\rangle$ and $|11\cdots1\rangle$. To symmetrize the results, balanced stings of 0's and 1's can be used, e.g. $|\overline{0}\rangle = |0101101\rangle$ and $|\overline{1}\rangle = |1010010\rangle$. In this case, a nontrivial parity of the initial state is used for computing the defect syndrome at the first QEC round, as described in Subsection 2.3.1.

2.3.3. Other Noise

In the phenomenological and circuit level noise models, only Pauli errors were assumed to happen on the data and ancilla qubits. With this assumption, it is therefore assumed that a single error results in at most 2 defects. In reality, more complex errors may also occur like leakage, crosstalk or cosmic rays that cannot be modelled using Pauli errors (Google Quantum AI, 2021). These errors will not be investigated in this thesis. Non-Pauli errors can lead to unconventional edges that do not result 3 or more defects. These edges are called hyperedges (Google Quantum AI, 2021; Chen et al., 2022). The presence of non-conventional errors in the experiment may then affect the estimated edge probabilities and lead to the observation of non-zero probabilities for some edges that should otherwise be zero in the absence of these non-Pauli errors.

2.4. Statistics

In statistics, we often want to estimate an unknown parameter θ with estimator T_N. The question will now be: how good or how accurate is that estimator? In this section, two methods will be introduced to estimate the standard deviation of an estimator. First, an analytical method will be presented which is based on the central limit theorem; the delta method. Secondly, the bootstrap method will be presented, which is a computer-based method.

2.4.1. Delta Method

Theorem 2.1 *Suppose we have an estimator T_N of parameter θ and we are interested in $g(\theta)$ for some function g that is differentiable at θ. Now, when we assume that*

$$\sqrt{N}(T_N - \theta) \rightsquigarrow \mathcal{N}(0, \sigma^2)$$

as N approaches infinity. Then

$$\sqrt{N}(g(T_N) - g(\theta)) \rightsquigarrow g'(\theta)\mathcal{N}(0, \sigma^2) = \mathcal{N}(0, \sigma^2 g'(\theta)^2).$$

Here, $\rightsquigarrow$ denotes a convergence in distribution. This result follows from writing out the first order approximation of the Taylor expansion of g. In greater generality, the distribution doesn't have to be a normal distribution (van der Vaart, 1995).

Proof We assume that g' is a continuous function and that T_N converges in probability to θ, i.e. $T_N \xrightarrow{P} \theta$. Now, by the mean value theorem, we have

$$g'(\tilde{\theta}) = \frac{g(T_N) - g(\theta)}{T_N - \theta} \quad \Leftrightarrow \quad g'(\tilde{\theta})(T_N - \theta) = g(T_N) - g(\theta)$$

where $\tilde{\theta}$ is in between T_N and θ. Therefore we also have $|T_N - \theta| > |\tilde{\theta} - \theta|$, so $\mathbb{P}(|T_N - \theta| > \varepsilon) \geq \mathbb{P}(|\tilde{\theta} - \theta| > \varepsilon)$. By definition of $T_N \xrightarrow{P} \theta$,

$$\mathbb{P}(|T_N - \theta| > \varepsilon) \to 0.$$

Therefore, $\tilde{\theta} \xrightarrow{P} \theta$ as well. By the continuous mapping theorem we then also have $g'(\tilde{\theta}) \xrightarrow{P} g'(\theta)$. To conclude, we find

$$\sqrt{N}(g(T_N) - g(\theta)) = g'(\tilde{\theta})\sqrt{N}(T_N - \theta) \rightsquigarrow g'(\theta)\mathcal{N}(0, \sigma^2).$$

⊠

2.4.2. Bootstrap

The delta method that was presented, is an analytical method to estimate the standard deviation. The bootstrap method, on the other hand, replaces the analytical derivations with an extensive amount of computations. The advantage of the bootstrap method is that it can handle functions that are analytically too complex to analyse for the delta method. The downside of bootstrap is that the amount of calculations is rather computer-intensive, even for relatively simple problems (Efron and Tibshirani, 1986).

Suppose that the observed data $\mathbf{x} = (x_1, x_2, \dots, x_N)$ is independent and identically distributed according to an unknown distribution F. We are interested in the standard deviation of the estimate T_N. Note that when we are given a certain data set, we are not able generate more samples, since F is unknown. The idea of the bootstrap method is to replace the distribution F with the empirical distribution function $\hat{F}$. Since we know this empirical distribution function, it is possible to sample from this distribution as often as desired.

A bootstrap sample is defined to be

$$\mathbf{x}^* = (x_1^*, x_2^*, \dots, x_N^*),$$

where each x_i^* is drawn randomly from the original sample $(x_1, x_2, \dots, x_N)$, with replacement. From each bootstrap sample it is possible to compute a bootstrap replication of the estimator T_N, denoted as T_N^*. Now, the bootstrap estimate of the standard deviation for T_N is given by

$$\hat{\sigma}_B = \left[\sum_{b=1}^{B} \left(T_N^{*b} - T_N^{*\cdot}\right)^2 / (B-1)\right]^{1/2}, \quad \text{with } T_N^{*\cdot} = \sum_{b=1}^{B} T_N^{*b}/B,$$

(Efron and Hastie, 2016). Since $\hat{F}$ approaches F for large N, $\hat{\sigma}_B$ correctly approaches the standard deviation of T_N in most cases (Efron and Hastie, 2016). To estimate the standard deviation, a B in the range of 50 to 200 is sufficient (Efron and Tibshirani, 1986). Under the assumption that the data is normally distributed, a 95% confidence interval can be constructed by $T_n \pm 1.96\sigma$. To construct confidence intervals using bootstrap, B in the order of 2000 is desirable (Efron and Hastie, 2016). Note that the bootstrap sample size is equal to the original sample size. If they are not equal, then the standard deviation $\hat{\sigma}_B$ will not converge to the standard deviation of T_N (Bickel and Freedman, 1981).

3

Estimating Edge Probabilities

In Section 2.3, it is described how a QEC experiment returns a matrix of defect syndromes. Now, we are interested in the probabilities of errors that may have occurred. An error is characterised by an edge between two defects or between a defect and the boundary. However, there may be countless ways to connect the defects. It is therefore not possible to know for sure which errors did occur, when the only data available is the defect matrix. It is however possible to assign a probability to every edge. With these probabilities, the most likely combination of errors can be found using a decoder. This research only focuses on estimating the probabilities, so not the decoding process itself.

This chapter will start with the derivation of edge probabilities for bulk and boundary edges. In this derivation it is assumed that every error leads to at most 2 defects. These estimations will then be tested for simulated data. Data will be simulated for the two presented error models: phenomenological noise and circuit level noise. Finally, the estimation formulas will be tested for experimental data.

3.1. Derivation of Probabilities

In this section, it will be described how the probabilities of edges can be estimated from the defect syndromes. Furthermore, the probabilities are estimated from simulated and experimental data.

3.1.1. Bulk edges

An error event at edge e is denoted as the binary random variable y_e where $y_e = 1$ means that an error happens at edge e and $y_e = 0$ means that no error happens at edge e. Note that y_e is a hidden random variable we cannot measure. However, y_e influences the binary random d_i that can be measured. These d_i's are the defects in the defect matrix. Here, each node $\{a, r\}$ is indicated with one index number $i = r + a \cdot (N_r + 1)$. We are interested in the probability $p_e = \mathbb{P}(y_e = 1)$ that an error happened at edge e between node i and j, as illustrated in Figure 3.1.

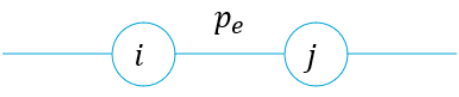


Figure 3.1: Bulk edge e occurring with probability $p_e = \mathbb{P}(y_e = 1)$ between nodes i and j.

From experimental data we can estimate the quantities $\langle d_i \rangle$ and $\langle d_i d_j \rangle$ by $\overline{d_i}$ and $\overline{d_i d_j}$ respectively. Here, $\overline{d_i}$ means an average of d_i over the number of experiments. Theoretically, these values are equal to

$$\langle d_i \rangle = \mathbb{E}[d_i] = \sum_{x_i \in \{0,1\}} x_i \mathbb{P}(d_i = x_i) = \mathbb{P}(d_i = 1), \tag{3.1}$$

$$\langle d_i d_j \rangle = \mathbb{E}[d_i d_j] = \sum_{x_i \in \{0,1\}, x_j \in \{0,1\}} x_i x_j \mathbb{P}(d_i = x_i, d_j = x_j) = \mathbb{P}(d_i = 1, d_j = 1). \tag{3.2}$$

For a defect to become $d_i = 1$, there needs to be an error event $y_e = 1$ on an edge that is connected to node i. However, for two error events $y_{e_1} = 1$ and $y_{e_2} = 1$ connected to node i, we have $d_i = 0$. This is because an error event ”flips” the connected nodes, i.e. $0 \to 1$ and $1 \to 0$. Therefore, there needs to be an odd number of error events connected to node i to have $d_i = 1$. Let A^i denote the set of edges connected to node i. We then have

$$\mathbb{P}(d_i = 1) = \mathbb{P}(\sum_{e \in A^i} y_e = \textit{odd}).$$

Since we are interested in the probability of edge e, we will ”extract” p_e from this $\mathbb{P}(d_i = 1)$. Let A_e^i denote set of edges connected to node i, excluding edge e. So $A_e^i = A^i \setminus \{e\}$. We can now write $\mathbb{P}(d_i = 1)$ as a function of p_e to get

$$\mathbb{P}(d_i = 1) = p_e \mathbb{P}(\sum_{e' \in A_e^i} y_{e'} = \textit{even}) + (1 - p_e)\mathbb{P}(\sum_{e' \in A_e^i} y_{e'} = \textit{odd}).$$

A similar expression can be constructed for $\langle d_i d_j \rangle$. We have

$$\mathbb{P}(d_i = 1, d_j = 1) = p_e \mathbb{P}(\sum_{e' \in A_e^i} y_{e'} = \textit{even})(\sum_{e' \in A_e^j} y_{e'} = \textit{even}) + (1 - p_e)\mathbb{P}(\sum_{e' \in A_e^i} y_{e'} = \textit{odd})(\sum_{e' \in A_e^j} y_{e'} = \textit{odd}).$$

Define the quantity $q_{i,e}^{\textit{eff}} = \mathbb{P}(\sum_{e' \in A_e^i} y_{e'} = \textit{odd})$. Using this $q_{i,e}^{\textit{eff}}$ and $q_{j,e}^{\textit{eff}}$, we can write the system of equations for $\langle d_i \rangle$, $\langle d_j \rangle$ and $\langle d_i d_j \rangle$ as

$$\langle d_i \rangle = p_e(1 - q_{i,e}^{\textit{eff}}) + (1 - p_e) q_{i,e}^{\textit{eff}}, \tag{3.3}$$

$$\langle d_j \rangle = p_e(1 - q_{j,e}^{\textit{eff}}) + (1 - p_e) q_{j,e}^{\textit{eff}}, \tag{3.4}$$

$$\langle d_i d_j \rangle = p_e(1 - q_{i,e}^{\textit{eff}})(1 - q_{j,e}^{\textit{eff}}) + (1 - p_e) q_{i,e}^{\textit{eff}} q_{j,e}^{\textit{eff}}. \tag{3.5}$$

We can now solve $q_{i,e}^{\textit{eff}}$ and $q_{j,e}^{\textit{eff}}$ from Equation (3.3) and (3.4), and substitute these in Equation (3.5). Finally, we can solve Equation (3.5) for the edge probability p_e to find

$$p_e = p_{ij} = \frac{1}{2} - \sqrt{\frac{1}{4} - \frac{\langle d_i d_j \rangle - \langle d_i \rangle \langle d_j \rangle}{1 - 2\langle d_i \rangle - 2\langle d_j \rangle + 4\langle d_i d_j \rangle}}. \tag{3.6}$$

The full calculations can be found in Appendix A. This means that the edge probability can now be determined from only the estimates of $\langle d_i \rangle$, $\langle d_j \rangle$ and $\langle d_i d_j \rangle$. This is actually a great result because $q_{i,e}^{\textit{eff}}$ is quite a complex expression.

3.1.2. Boundary edges

Note that Equation (3.6) cannot be used for an edge between a node i and the boundary as there is no data available for $\langle d_j \rangle$ and $\langle d_i d_j \rangle$. The boundary edge probability from node i to the boundary B can still be estimated using Equation (3.3). Solving for p_e gives the boundary edge probability

$$p_e = p_{iB} = \frac{\langle d_i \rangle - q_{i,e}^{\textit{eff}}}{1 - 2q_{i,e}^{\textit{eff}}}. \tag{3.7}$$

So in order to estimate a boundary edge probability, the term $q_{i,e}^{\textit{eff}}$ is required. $q_{i,e}^{\textit{eff}}$ was defined to be the probability that an odd number of adjacent edges e', excluding e, have $y_{e'} = 1$. This results in a sum

over the different combinations with an odd number of events $y_{e'} = 1$. The full expression is given by

$$q_{i,e}^{eff} = \sum_{b \in \{0,1\}^{|A_e^i|} : \sum b_{e'} = odd} \left(\prod_{e' \in A_e^i} p_{e'}^{b_{e'}} (1 - p_{e'})^{(1-b_{e'})} \right). \tag{3.8}$$

Now there are two approaches in calculating $q_{i,e}^{eff}$. The difference is in defining the adjacency set A_e^i. For the first approach, we assume to know the correct error model. This means that edges only exist between certain pairs. For the phenomenological noise model, only space and time edges between direct neighbouring nodes are expected. The adjacent edges for the two boundary qubits are represented in Figure 3.2 by blue edges. Here, we can see that the size of the adjacency set of a boundary edge for phenomenological noise is given by $|A_e^i| = 3$. For the circuit level noise, space-time edges are also allowed. These are represented as grey lines in Figure 3.2. In this case we have $|A_e^i| = 4$ for a boundary edge.

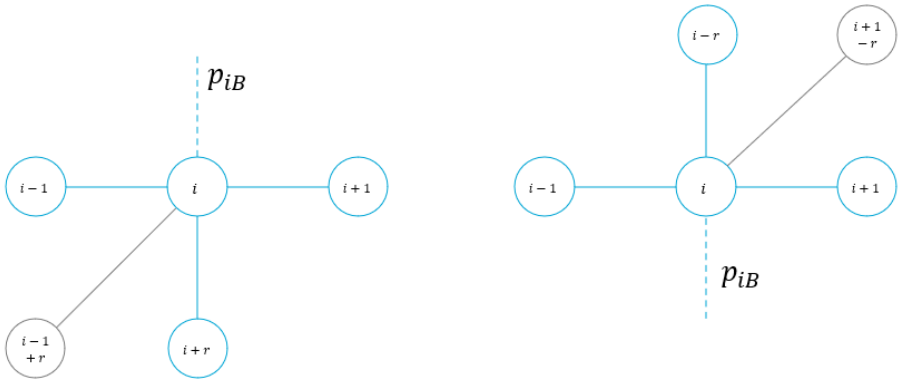


Figure 3.2: Adjacent edges for the boundary nodes of the two data qubits at the boundary. Here, the horizontal axis is the time axis and the vertical axis is the space axis. Considering phenomenological noise, only the blue lines form the adjacent edges. The grey lines (space-time edges) are also adjacent edges when considering circuit level noise.

For the second approach we assume we do not know the error model. This means that edges may exist between any pair of nodes. Therefore, the size of the adjacency set is given by the total number of other nodes, so $|A_e^i| = (d-1) \times (N_r + 1) - 1$. For this approach, every node may be connected to the boundary instead of just the data qubits that are located at the boundary.

Instead of calculating the full sum of products in Equation (3.8), which is computationally expensive, one can implement an equivalent equation presented by Google Quantum AI, 2021. The $q_{i,e}^{eff}$ term can also be calculated by[1]

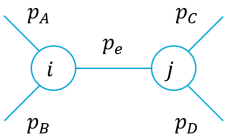
$$q_{i,B}^{eff} = g(p_{ij_k}, \dots g(p_{ij_3}, g(p_{ij_2}, p_{ij_1})) \dots), \tag{3.9}$$

$$g(p,q) = p(1-q) + (1-p)q = p + q - 2pq, \tag{3.10}$$

Here k denotes the number of edges connected to i. Equation (3.9) is computationally faster because it can used in combination with the numpy.reduce function over the whole matrix of p_{ij} elements to allow vectorized computation.

[1] Equation (3.9) may look a bit confusing. Here, the function g is applied each time on g itself with a new edge probability. For 5 elements p_{ij_k}, the function looks like $q_{i,B}^{eff} = g(p_{ij_5}, g(p_{ij_4}, g(p_{ij_3}, g(p_{ij_2}, p_{ij_1}))))$.

Example *Calculating the probability $q_{i,e}^{eff}$ for a small graph.* Suppose we have the following graph and we want to determine $q_{i,e}^{eff}$ and $q_{j,e}^{eff}$.



First, $q_{i,e}^{eff}$ is the probability of $d_i = 1$ when considering edges $e' \in A_e^i = \{A, B\}$. This happens when $y_A = 1$ and $y_B = 0$ or when $y_A = 0$ and $y_B = 1$. Therefore the probability is given by

$$q_{i,e}^{eff} = p_A(1 - p_B) + (1 - p_A)p_B.$$

The expression for $q_{j,e}^{eff}$ can be constructed in a similar manner, with $A_e^j = \{C, D\}$, resulting in

$$q_{j,e}^{eff} = p_C(1 - p_D) + (1 - p_C)p_D.$$

3.2. Simulations

In this section, the estimated probabilities will be presented for numerically simulated data. First, phenomenological noise is assumed. The bulk and boundary edges are estimated using the equations derived in Section 3.1. Also, the two different approaches for estimating the boundary edges are compared. Thereafter, the analysis will be repeated for the circuit level noise model.

3.2.1. Phenomenological Noise

With the assumption of phenomenological noise, only space and time edges are expected to occur with probability p, as described in Subsection 2.3.1. Remember that a space edge exists between nodes of two neighboring ancillas that are connected to the same data qubit and that are in the same QEC round. A time edges exists between nodes of the same ancilla with adjacent QEC rounds.

A simulation is run with error probability $p = 0.05$ for both data and ancilla qubits, with $N_r = 7$ QEC rounds and is repeated $N_{exp} = 200,000$ times. Each run is called a 'shot'. The code for running the simulations can be found in Appendix C. The probabilities of all bulk edges are estimated using Equation (3.6), where the terms $\langle d_i \rangle$, $\langle d_j \rangle$ and $\langle d_i d_j \rangle$ are replaced by the observed averages. These estimated probabilities are shown in Figure 3.3. In this figure, the x- and y-axis represent the nodes i and j. The major ticks represent the different ancilla qubits and the minor ticks represent the QEC rounds. Note that this matrix has 8 QEC rounds, this is because the syndrome of the last round is obtained from the data measurement. Each square in the graph is now the probability between nodes i and j. For illustrative purposes, the diagonal is set to be zero. Because $p_{ij} = p_{ji}$, the matrix is symmetric about the diagonal. The upper half and lower half are represented on different scales, which will become relevant if different edges have different probabilities. The estimated probabilities for the space and time edges are $\hat{p}_e \approx 0.05$.

The boundary edges are estimated using the two approaches, described in Subsection 3.1.2. For the first approach, only the direct adjacent edges are taken into account when calculating the boundary edge probability. The estimated probabilities are shown in Figure 3.4a. Here, the squares represent the probability of an edge from node i to the boundary. The boundary edge probabilities for the ancilla qubits at the boundary are correctly estimated to be $\hat{p}_e \approx 0.05$. For the other ancilla qubits, the boundary edge probability is initialised to be 0 because we assume no edge exists to the boundary.

For the second approach, all edges are taken into account when calculating the boundary probability. The estimated probabilities using this approach are shown in Figure 3.4b. Note that some probabilities here are estimated to be negative. We want to better understand these negative probabilities. Therefore, the estimated boundary edge probabilities are also plotted as a function of QEC

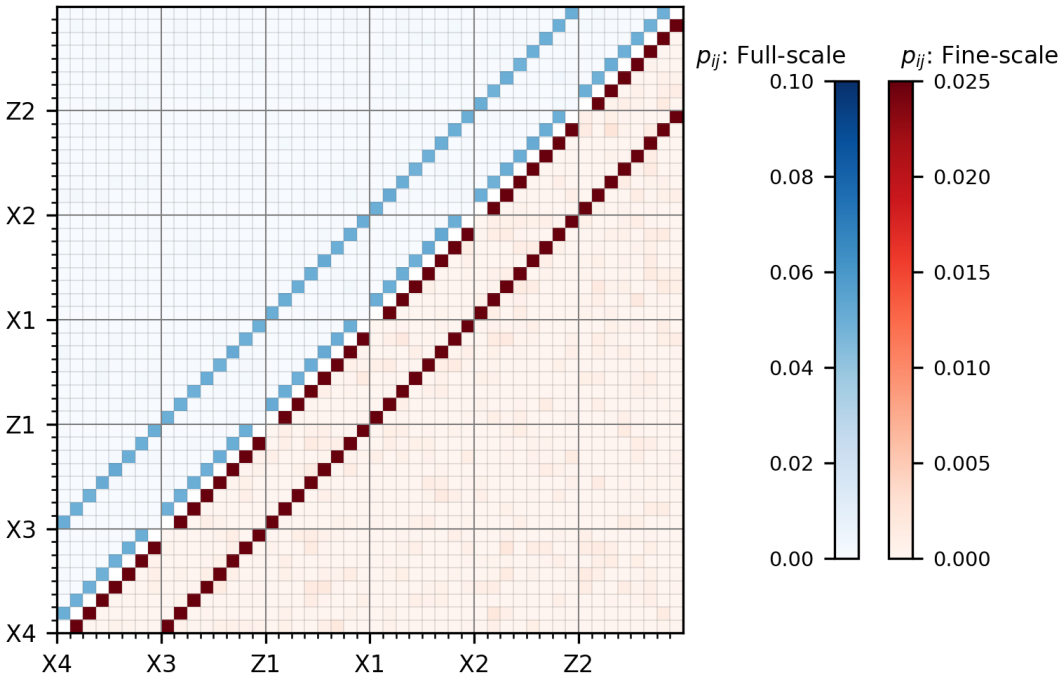


Figure 3.3: Probability matrix for the bulk edges of a simulation with phenomenological noise. The major ticks represent the different ancilla qubits and the minor ticks represent the QEC rounds. Each square represents the probability of an edge between the nodes i and j. The matrix is symmetric about the diagonal. The upper half and lower half are represented on different scales, which is not relevant for this specific figure. The outer diagonals represent the space edges, since they connect nodes between two different ancillas. The inner diagonals represent the time edges, since they connect nodes for the same ancilla but a different QEC round.

rounds in Figure 3.5. To also provide a standard error[2], the boundary edge estimation is repeated using the bootstrap method with 200 resamples, like explained in Section 2.4.2. The values for the estimated probabilities of the boundary edges with their standard error averaged over the QEC rounds are given in Table 3.1. Here, the values are averaged over the QEC rounds[3]. We can see that some probabilities, which are expected to be 0, are estimated to be negative. Note however that estimated edge probabilities are within the range of the standard error around the expected value of $p_e = 0$. This means that a sampling error can explain the observed nonphysical probabilities.

[2]The standard error is the standard deviation of the sample distribution.

[3]Note that the standard deviation cannot simply be averaged. The average standard deviation is found by taking the square root of the average variance. We therefore have $\overline{\sigma} = \frac{1}{k^2}\sqrt{\sigma_1^2 + \cdots + \sigma_k^2}$.

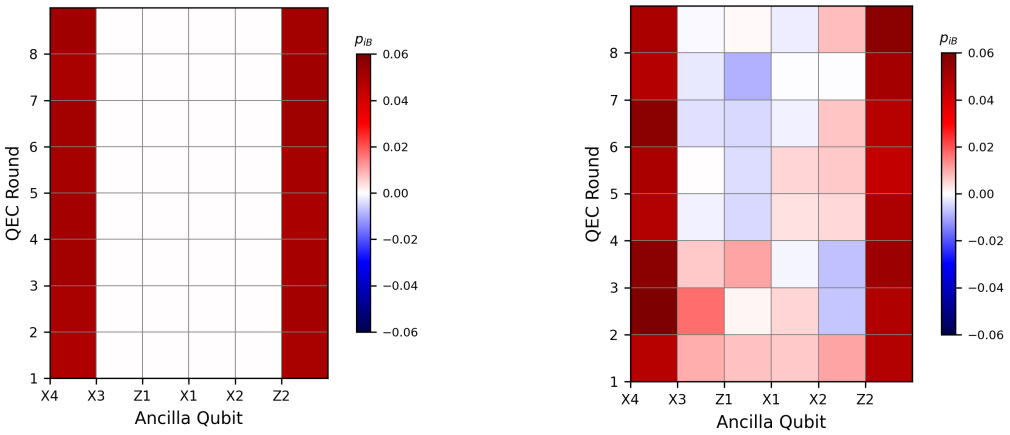


Figure 3.4: Boundary edge matrices for phenomenological noise. Each square represents the probability of an edge from that node i to the boundary. (a) The boundary edges are estimated using only the edges to the adjacent nodes. (b) The boundary edges are estimated considering every other edges.

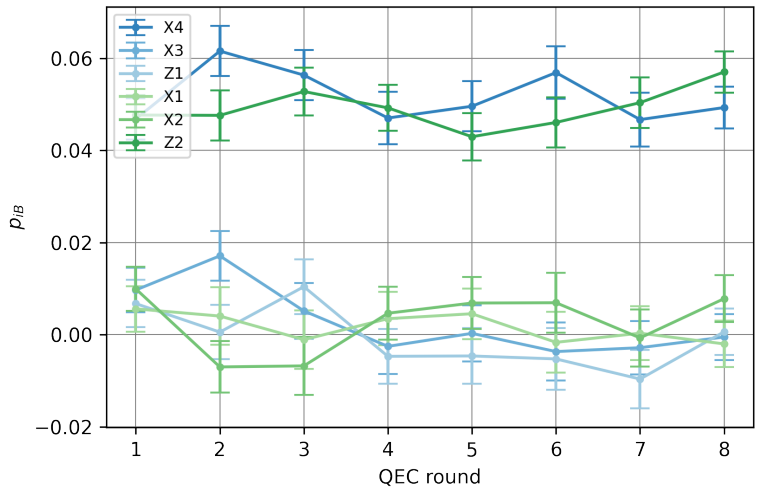


Figure 3.5: Estimated boundary edge probabilities for the ancilla qubits over QEC rounds. The probabilities are estimated by considering all other edges. The error bars represent the standard errors and are determined using bootstrap with 200 resamples. It can be seen that the edges for the ancillas at the boundary, X_4 and Z_2, are estimated to be $p_e = \approx 0.05$. The other edges are estimated to be $p_e \approx 0$.

In Figure 3.6, the two approaches for estimating the boundary edge probability for ancilla qubits X_4 and Z_2 are compared. Here it becomes clear that both methods estimate about the same probability. However, when taking only the edges with adjacent nodes into account, the probability is estimated to be more accurate and with a lower standard error (about 5 times smaller).

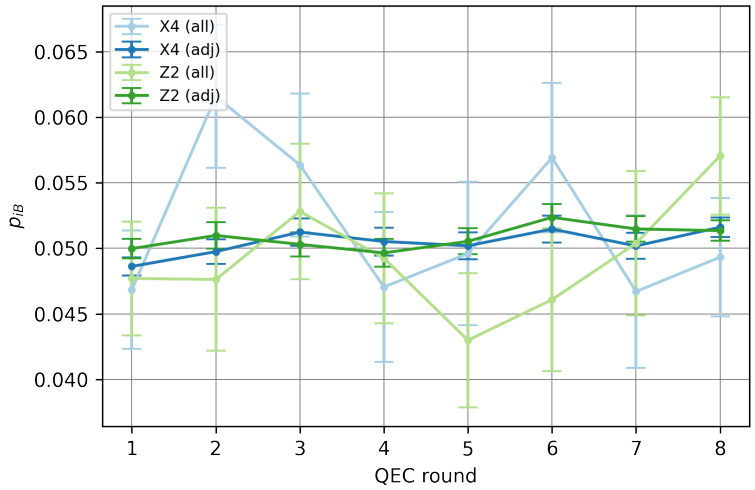


Figure 3.6: Estimated boundary edge probabilities over QEC rounds for simulated data of the phenomenological noise model. The two approaches for estimating the boundary edge probability are compared for the two boundary ancillas: considering all or only the adjacent edges. Both methods estimate about the same probabilities. The error bars represent the standard error and are determined using bootstrap with 200 resamples. Considering all other edges when estimating the boundary edges shows a larger standard error.

	adj		all	
	$\overline{\hat{p}_e}$	$\overline{SE}$	$\overline{\hat{p}_e}$	$\overline{SE}$
X_4	0.0504	0.000948	0.0518	0.00535
X_3	0	0	0.00283	0.00570
Z_1	0	0	-0.000745	0.00590
X_1	0	0	0.00164	0.00583
X_2	0	0	0.00271	0.00575
Z_2	0.0508	0.000944	0.0492	0.00507

Table 3.1: Estimated boundary probabilities and their standard errors for simulated data with phenomenological noise, averaged over the QEC rounds, calculated using bootstrap with 200 resamples.

3.2.2. Absolute Error for Phenomenological Noise

For the phenomenological noise model, we do know the true value of the error probability. Therefore, we are able calculate the difference $|\hat{p}_e - p_e|$ for each edge type, where $\hat{p}_e$ denotes the estimated edge probability p_e. This quantity is called the absolute error of the edge probability p_e. The absolute errors are averaged per edge type. This mean absolute error can be analysed as a function of N, number of shots, and p, initialised error probability. The results are shown Figure 3.7 and 3.8.

For the mean absolute error over N_{exp}, the results are fitted to the function $a \cdot N_{exp}^b$. It follows that the absolute error has $1/\sqrt{N_{exp}}$ behaviour for all edge types. It can also be seen the absolute error for the boundary edges is almost a factor of 10 higher if all edges are taken into account.

The absolute error over p also shows that the error is a lot larger for the boundary edges if all other edges are taken into account. This suggests that, if possible, one should only look at edges with adjacent nodes to reduce the error of edge estimation. This is of course only possible if the edges with adjacent nodes are the only expected errors to affect the boundary node.

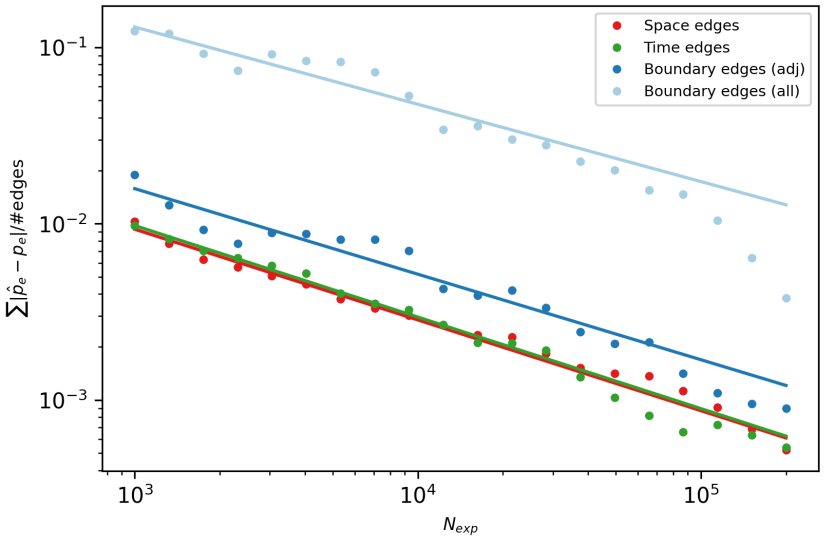


Figure 3.7: Average absolute error per edge type as a function of number of shots. The simulation is run for phenomenological noise with error probability $p = 0.05$ and $N_r = 7$ QEC rounds.

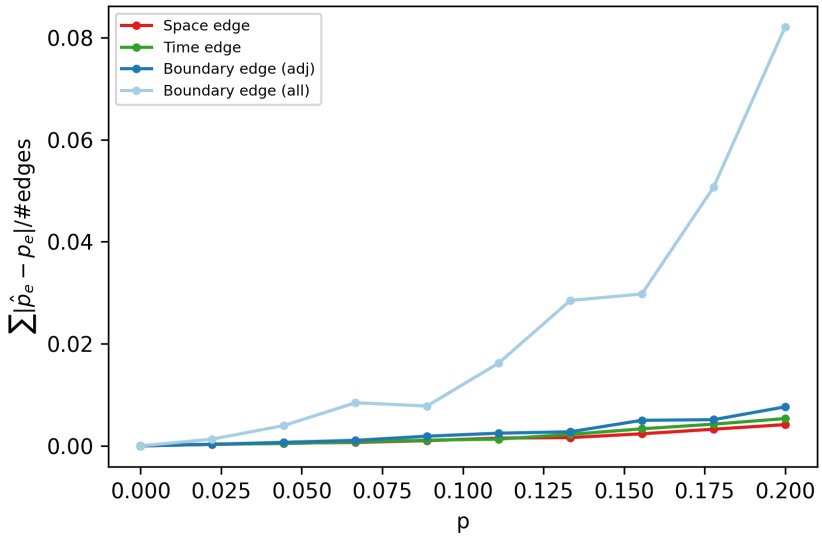


Figure 3.8: Average absolute error per edge type as a function of initialised error probability. The simulation is run for phenomenological noise with $N_r = 7$ QEC rounds and $200,000$ shots.

3.2.3. Circuit Level Noise

The analysis we did for the phenomenological noise model will now be repeated for circuit level noise. In this case, the probability of an edge cannot be initialised directly because the gate errors are initialised using the actual qubit characteristics, explained in Subsection 2.3.2. The provided experimental data has $N_{exp} = 163,840$ runs for $N_r = 7$ QEC rounds. Since we want to try to replicate this data with the circuit level noise model, we will also initialise the numerical simulation of circuit level noise with $N_r = 7$ QEC rounds and $N_{exp} = 163,840$ runs.

Figure 3.9 shows the bulk edge probabilities estimated from the simulation with circuit level noise. We can again see the space and time edges, like for the phenomenological noise. However, now we can also see the space-time edges. These edges follow the structure of Figure 2.10c. It can also be seen that for some ancilla qubits, the last round has a higher probability for the space edges. Remember that after the last QEC round, the data qubits are measured. From this measurement, the

last syndrome is obtained. A relatively high assignment error for some data qubits results in the higher space edge probability for the last round. One can also see that the space edge probability for the first round is lower. This is because the syndrome of the initialised state, for which it is assumed that it is prepared perfectly, was used to construct the defects of the first round. This results in a lower space edge probability for the first round.

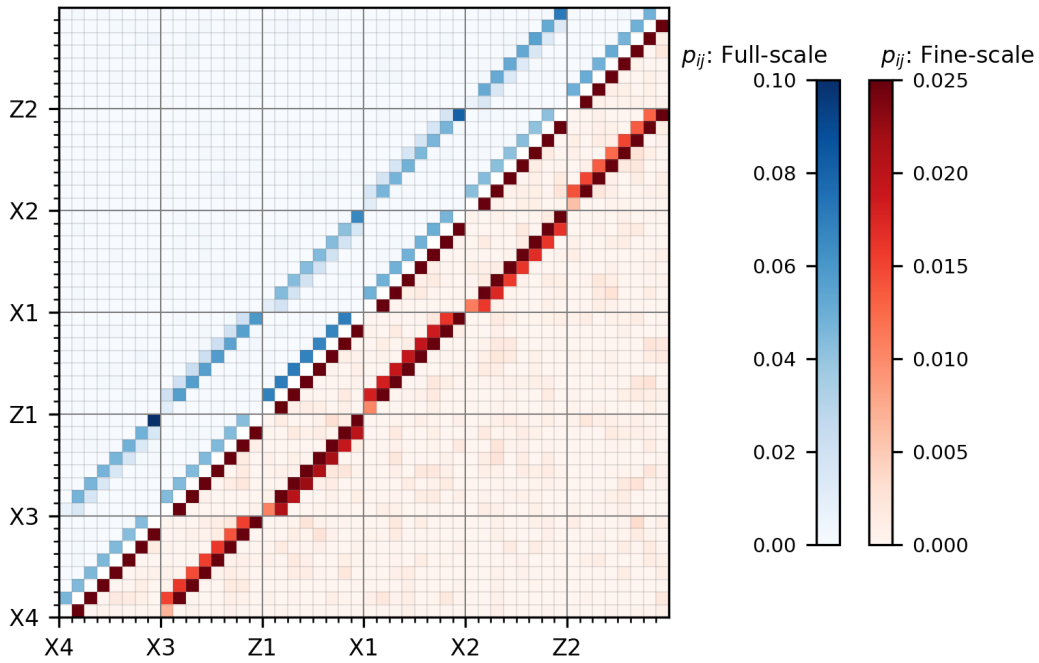


Figure 3.9: Probability matrix for the bulk edges of a simulation with circuit level noise. The major ticks represent the different ancilla qubits and the minor ticks represent the QEC rounds. Each square represents the probability of an edge between the nodes i and j. The matrix is symmetric about the diagonal. The upper half and lower half are represented on different scales. The space and time edges are similar to Figure 3.3. The new edges that appear here are the space-time edges.

The estimated boundary edge probabilities are shown in Figure 3.10. Similar as before, the two different approaches are used to estimate the boundary edge probabilities. The only difference with phenomenological noise is that now also space-time edges are taken into account when considering only adjacent nodes. Note that the ancilla qubits X_4 and Z_2 have different error probabilities, this is because of different qubit characteristics mentioned in Table 2.3. One can also see the lower/higher probability for the first/last QEC round. Again, the bootstrap method is used with 200 resamples to plot the boundary edge probabilities with their standard error over the QEC rounds. The results are shown in Figure 3.11 and the values, averaged over the QEC rounds, are given in Table 3.2.

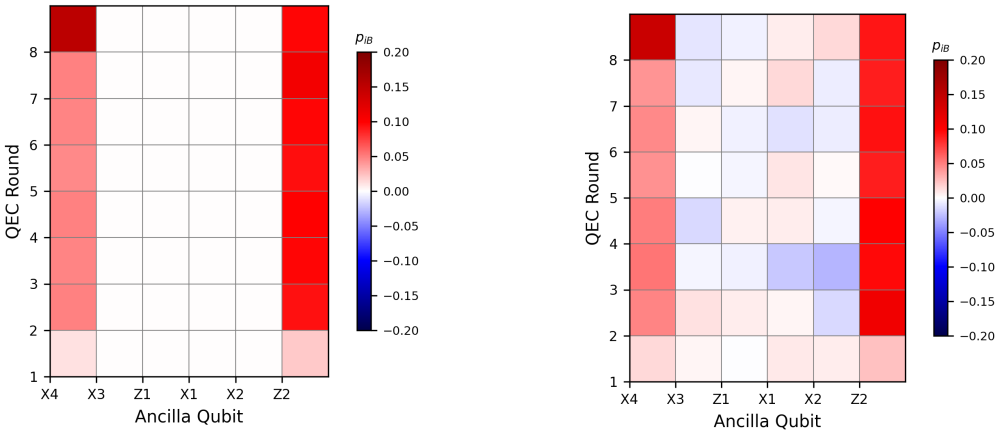


Figure 3.10: Boundary edge matrices for circuit level noise. Each square represents the probability of an edge from that node i to the boundary. (a) The boundary edges are estimated using only the edges to the adjacent nodes. (b) The boundary edges are estimated considering every other edges.

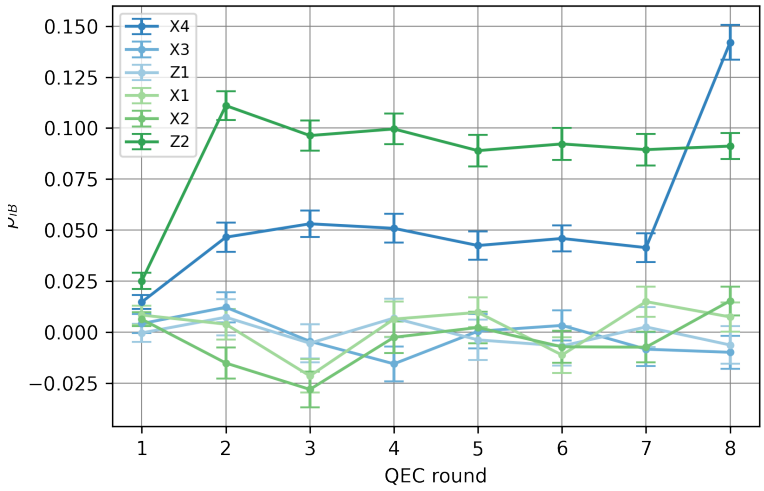


Figure 3.11: Estimated boundary edge probabilities for the ancilla qubits over QEC rounds. The probabilities are estimated by considering all other edges. The error bars represent the standard errors and are determined using bootstrap with 200 resamples. It can be seen that the edges for the ancillas at the boundary, X_4 and Z_2, are estimated to be $\hat{p}_e \approx 0.05$ and $\hat{p}_e \approx 0.1$ respectively when not considering the first and final round. The other edges are estimated to be $\hat{p}_e \approx 0$.

The two approaches for estimating the boundary edge probability for ancilla X_4 and Z_2 are compared in Figure 3.12. Here, we can see a similar result as for the phenomenological noise. Both approaches estimate about the same probability but when taking more edges into account, the standard error increases.

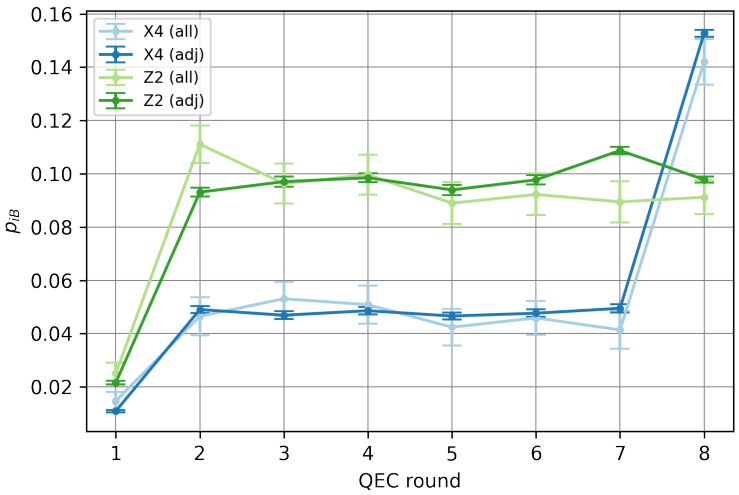


Figure 3.12: Estimated boundary edge probabilities over QEC rounds. The two approaches for estimating the boundary edge probability are compared for the two boundary ancillas: considering all or only the adjacent edges. Both methods estimate about the same probabilities. The error bars represent the standard error and are determined using bootstrap with 200 resamples. Considering all other edges when estimating the boundary edges shows a larger standard error.

	adj		all	
	$\overline{\hat{p}_e}$	$\overline{SE}$	$\overline{\hat{p}_e}$	$\overline{SE}$
X_4	0.0565	0.00131	0.0546	0.00678
X_3	0	0	-0.00233	0.00776
Z_1	0	0	-0.000804	0.00899
X_1	0	0	0.00220	0.00755
X_2	0	0	-0.00461	0.00738
Z_2	0.0885	0.00158	0.0867	0.00709

Table 3.2: Estimated boundary probabilities and their standard error for numerically simulated data with circuit level noise, averaged over the QEC rounds, calculated using bootstrap with 200 resamples.

3.3. Experiment

Now that we have seen how the edge estimation performs on simulations, we will try to do the same for the experimental data. Figure 3.13 show the estimated edge probabilities. Here, the space and time edges are actually quite visible. The space-time edges can also be recognised for some qubits, like for X_3. There is however also a lot of other noise, which we did not see in the simulations. Note also that the color scales are higher than for Figure 3.9. This might suggests that the noise parameters in the circuit level noise model are not correctly describing reality. Besides the additional noise we can also see some patterns of higher probability appearing, like the red coloured triangle under the time edges of ancilla X_3 or the coloured regions between ancilla qubits X_2/Z_1 and Z_2/X_1. This suggests that there are other errors occurring than our noise models allowed, like leakage or cross-talk. Note that these other errors could also be the reason for the larger color scales.

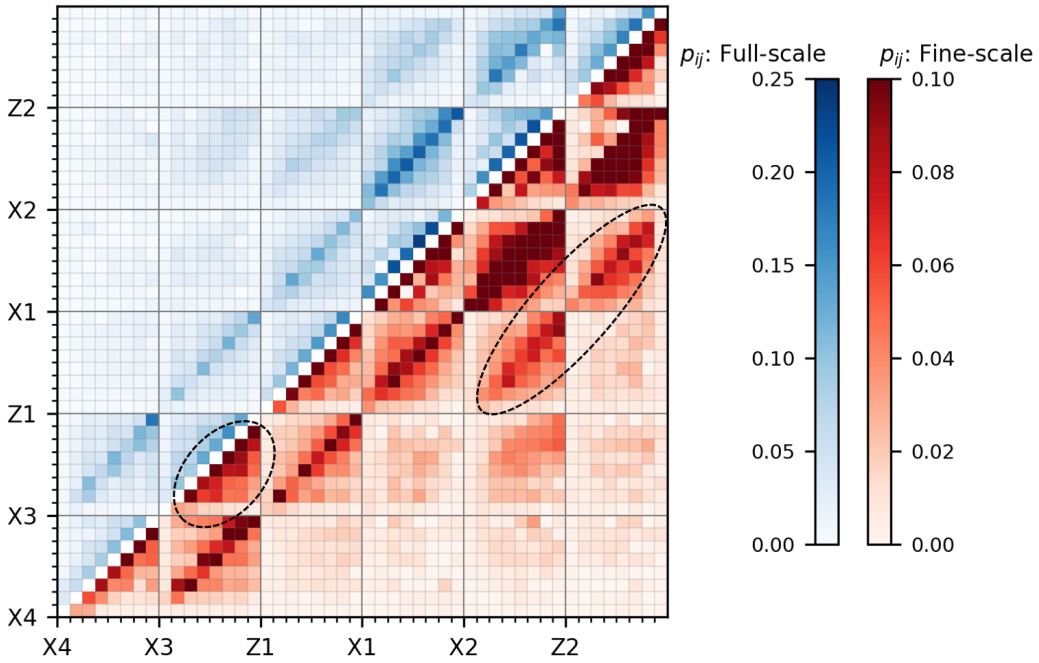


Figure 3.13: Probability matrix for the bulk edges of the experimental data. The major ticks represent the different ancilla qubits and the minor ticks represent the QEC rounds. Each square represents the probability of an edge between the nodes i and j. The matrix is symmetric about the diagonal. The upper half and lower half are represented on different scales. The marked areas show non-conventional errors.

Even though there may be unexpected errors, it is still possible to try to estimate the boundary probabilities. Remember that the second approach of choosing an adjacency set allowed all other edges between any pair of nodes. The estimated boundary edge probabilities, using the two approaches, are shown in Figure 3.14. One can easily see that these probabilities are nonphysical.

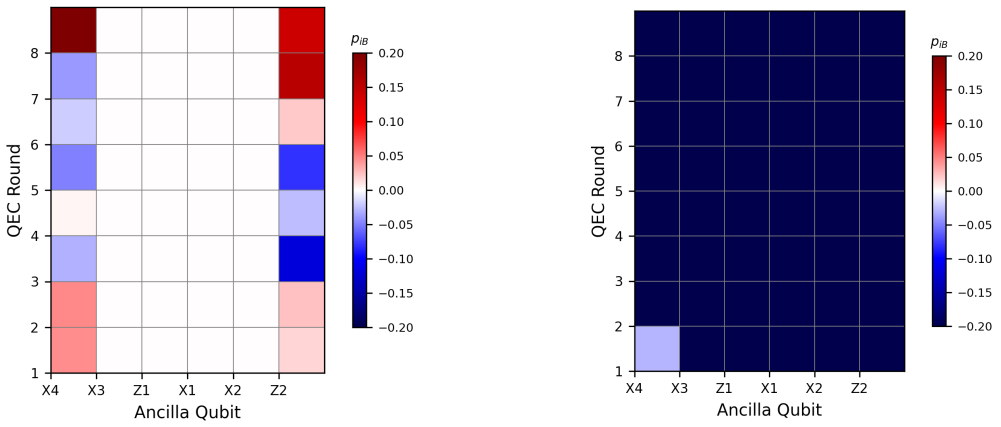


Figure 3.14: Boundary edge matrices for the experimental data. Each square represents the probability of an edge from that node i to the boundary. (a) The boundary edges are estimated using only the edges to the adjacent nodes. (b) The boundary edges are estimated considering every other edges. It can be seen that (a) shows a few negative probabilities. However, (b) shows that all probabilities are estimated to be negative with a probability that is out of the color scale.

To estimate the standard deviation of these probabilities, bootstrap is again applied with 200 resamples. This is to check if the standard error may be even larger than the negative probabilities. The results are shown in Figure 3.15, where the boundary probability was estimated by taking all edges into account, also when considering the standard error. This shows that the probabilities are not estimated to be negative due to some undersampling, which suggests that the error model that was assumed when constructing the equations for edge estimations might not be correct. The averaged values for the probability and standard error over QEC rounds are shown in 3.3.

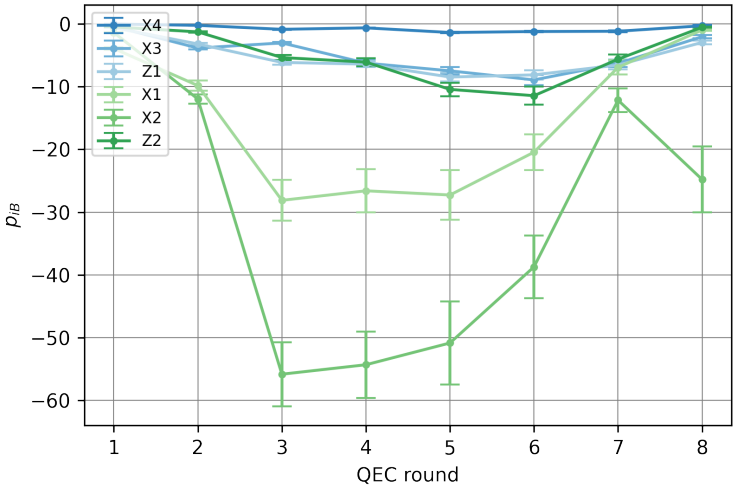


Figure 3.15: Estimated boundary edge probabilities for the ancilla qubits over QEC rounds. The probabilities are estimated by considering all other edges. The error bars represent the standard error and are determined using bootstrap with 200 resamples. All probabilities are estimated to be negative, even when taking the standard error into account.

It can be seen in Table 3.3 that the average values turn out to be positive when the boundary probabilities are calculated considering only the edges to adjacent nodes. In Figure 3.16, the estimated boundary edge probabilities are shown for this approach. It can be seen, however, that some edges are still estimated to be negative, even when also considering the standard error. The average probabilities turn out to be positive due to the higher probability at later QEC rounds. The edge probability is not consistent over the QEC rounds. Therefore, even though the probabilities are way less negative compared to Figure 3.15, this approach still does not give physical probabilities. We can conclude that the assumed error model does not correctly describe all errors that occur during the experiment.

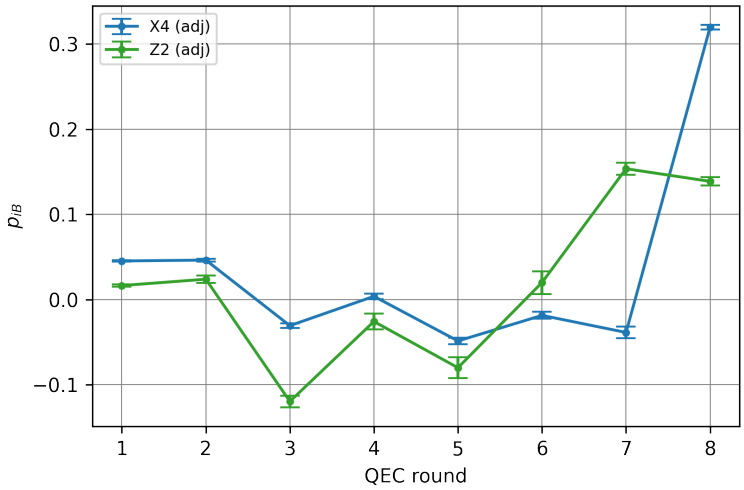


Figure 3.16: Estimated boundary edge probabilities for the ancilla qubits over QEC rounds. The probabilities are estimated by considering only the edges to adjacent nodes. The error bars represent the standard error and are determined using bootstrap with 200 resamples. Some probabilities are still estimated to be negative, even when taking the standard error into account.

	adj		all	
	$\overline{\hat{p}_e}$	$\overline{SE}$	$\overline{\hat{p}_e}$	$\overline{SE}$
X_4	0.0347	0.00372	-0.739	0.102
X_3	0	0	-4.81	0.447
Z_1	0	0	-5.347	0.479
X_1	0	0	-15.4	2.33
X_2	0	0	-31.1	4.30
Z_2	0.0159	0.00831	-5.17	0.777

Table 3.3: Estimated boundary probabilities and their standard error for experimental data, averaged over the QEC rounds, calculated using bootstrap with 200 resamples.

4

Further Analysis on Bulk Edges

In the previous chapter we have seen that it is not possible to estimate the probability of the boundary edges from the experimental data. There may be multiple solutions to better estimate the boundary probabilities, but what if the boundary edges can be removed? Just like an ancilla measures the parity between two adjacent data qubits, we are also allowed to add an additional ancilla that measures the parity between the two boundary qubits. Note that in this case there is no boundary edge at all. We call this the ”circular repetition code”. More details about this circular repetition code can be found in Appendix B. Currently, there is no data available on the circular repetition code. What can be investigated further, however, are the bulk edges and their standard deviation. It is expected that the circular repetition code returns similar results for the bulk as the ”normal” repetition code, for which we do have data.

We will start this chapter by investigating the standard deviation of bulk edges for different numbers of shots and different initialised error probabilities. The standard deviation is calculated using three analytic approximation formulas and using the bootstrap method. These methods are compared. Furthermore, the standard deviations for the bulk edges of the experimental data are analysed. Finally, the bulk edges are estimated and compared to the circuit level noise model.

4.1. Standard Deviation of Bulk Edges for Simulated Data

In this section, the standard deviation of bulk edges will be evaluated. Since bulk edges have a relatively easy expression for the probability of that edge, multiple approximation formulas can be constructed. First, two approximations for the standard deviation will be presented, that were derived by Google Quantum AI, 2021. Secondly, the delta method from Subsection 2.4.1 will be applied to Equation (3.6) to also find an approximation for the standard deviation of the edge probability. Finally, these expressions will be compared with the standard deviation from the bootstrap method as a function of number of shots and initialised edge probability.

4.1.1. Approximation Formulas

In 2021 Google Quantum AI presented their results on a similar experiment of the repetition code. In addition, they presented two analytic expressions for the standard deviation of the bulk edges, σ_{p_e}. Under the assumption that $p_e \ll \frac{1}{4}$. Equation (3.6) can then be approximated as

$$p_e \approx \frac{\langle d_i d_j \rangle - \langle d_i \rangle \langle d_j \rangle}{(1 - 2\langle d_i \rangle)(1 - 2\langle d_j \rangle)}.$$

The two analytic expressions for the standard deviation will be presented here as Theorem 4.1 and Corollary 4.1.1.

Theorem 4.1 *(Google Quantum AI, 2021: S19) Assume that* $\langle d_i \rangle \ll 1$ *and* $p_e \ll 1$*. Then the standard deviation of the bulk edge probability* p_e *from Equation (3.9) can be approximated by*

$$\sigma_{p_e} \approx \frac{1}{\sqrt{N_{exp}}} \sqrt{p_e(1-p_e) + \frac{\langle d_i \rangle \langle d_j \rangle (1-\langle d_i \rangle)(1-\langle d_j \rangle)}{(1-2\langle d_i \rangle)^2 (1-2\langle d_j \rangle)^2}}. \tag{4.1}$$

Corollary 4.1.1 *(Google Quantum AI, 2021: S20) Under the assumption of* $\langle d_i \rangle \ll 1$ *and* $p_e \ll 1$*, Equation (4.1) can be further simplified by neglecting the factors* $(1-p_e)$ *and* $(1-\langle d_i \rangle)(1-\langle d_j \rangle)$*. Neglecting these factors should increase* σ_{p_e}*. We obtain*

$$\sigma_{p_e} \approx \frac{1}{\sqrt{N_{exp}}} \sqrt{p_e + \frac{\langle d_i \rangle \langle d_j \rangle}{(1-2\langle d_i \rangle)^2 (1-2\langle d_j \rangle)^2}}. \tag{4.2}$$

4.1.2. Delta Method

The delta method, which was introduced in Subsection 2.4.1, provided an expression to estimate the variance of a function of a random variable satisfying a known distribution. In our case we have the estimator $\overline{d_i}$ for the parameter $\langle d_i \rangle$, $\overline{d_j}$ for $\langle d_j \rangle$ and $\overline{d_i d_j}$ for $\langle d_i d_j \rangle$. We are interested in $p_e = f(\langle d_i \rangle, \langle d_j \rangle, \langle d_i d_j \rangle)$, where

$$f(x,y,z) = \frac{1}{2} - \sqrt{\frac{1}{4} - \frac{z - x \cdot y}{1 - 2x - 2y + 4z}}.$$

Let $Y_1, Y_2, \dots, Y_n, \dots$ be independent and identically distributed random vectors such that

$$Y_n = \begin{pmatrix} d_{i,n} \\ d_{j,n} \\ d_{i,n}d_{j,n} \end{pmatrix}, \quad \overline{Y}_N = \begin{pmatrix} \overline{d_i} \\ \overline{d_j} \\ \overline{d_i d_j} \end{pmatrix} \quad \text{and} \quad \mu = \begin{pmatrix} \langle d_i \rangle \\ \langle d_j \rangle \\ \langle d_i d_j \rangle \end{pmatrix}.$$

Here, $d_{i,n}$ denotes the measurement of d_i in experiment n and $\overline{d_i}$ denotes the average of d_i over N experiments. Now, by the multivariate central limit theorem (van der Vaart, 1995), we have

$$\sqrt{N}(\overline{Y}_N - \mu) \rightsquigarrow \mathcal{N}_3(0, \Sigma),$$

where Σ is the covariance matrix

$$\Sigma = \begin{pmatrix} \mathrm{cov}(d_i, d_i) & \mathrm{cov}(d_i, d_j) & \mathrm{cov}(d_i, d_i d_j) \\ \mathrm{cov}(d_j, d_i) & \mathrm{cov}(d_j, d_j) & \mathrm{cov}(d_j, d_i d_j) \\ \mathrm{cov}(d_i d_j, d_i) & \mathrm{cov}(d_i d_j, d_j) & \mathrm{cov}(d_i d_j, d_i d_j) \end{pmatrix}.$$

Since d_i holds binary values, we have $d_i^2 = d_i$ and $(d_i d_j)^2 = d_i d_j$. This simplifies the expressions in the covariance matrix. The multivariate central limit theorem may now be written as

$$\sqrt{N}\left(\begin{pmatrix} \overline{d_i} \\ \overline{d_j} \\ \overline{d_i d_j} \end{pmatrix} - \begin{pmatrix} \langle d_i \rangle \\ \langle d_j \rangle \\ \langle d_i d_j \rangle \end{pmatrix}\right) \rightsquigarrow \mathcal{N}_3\left(\begin{pmatrix} 0 \\ 0 \\ 0 \end{pmatrix}, \begin{pmatrix} \langle d_i \rangle - \langle d_i \rangle^2 & \langle d_i d_j \rangle - \langle d_i \rangle \langle d_j \rangle & \langle d_i d_j \rangle (1 - \langle d_i \rangle) \\ \langle d_i d_j \rangle - \langle d_i \rangle \langle d_j \rangle & \langle d_j \rangle - \langle d_j \rangle^2 & \langle d_i d_j \rangle (1 - \langle d_j \rangle) \\ \langle d_i d_j \rangle (1 - \langle d_i \rangle) & \langle d_i d_j \rangle (1 - \langle d_j \rangle) & \langle d_i d_j \rangle - \langle d_i d_j \rangle^2 \end{pmatrix}\right).$$

Let $\hat{p}_e = f(\overline{d_i}, \overline{d_j}, \overline{d_i d_j})$ and $p_e = f(\langle d_i \rangle, \langle d_j \rangle, \langle d_i d_j \rangle)$. Now, for the multivariate case of the delta method, we have that

$$\sqrt{N}(\hat{p}_e - p_e) \rightsquigarrow \mathcal{N}(0, \sigma^2), \quad \text{where} \quad \sigma^2 = f'(\mu) \Sigma (f'(\mu))^T.$$

If we now take the variance on both sides, and divide by N, we obtain

$$\mathrm{var}(\hat{p}_e) \approx \sigma^2 / N,$$

since $\mathrm{var}(p_e) = 0$. The derivative of $f(x,y,z)$ at μ is given by

$$f'(\mu) = \begin{pmatrix} \frac{\partial f}{\partial x}(\mu) & \frac{\partial f}{\partial y}(\mu) & \frac{\partial f}{\partial z}(\mu) \end{pmatrix},$$

with

$$\frac{\partial f}{\partial x} = \frac{y - 2y^2 + 4zy - 2z}{\sqrt{(1-2x)(1-2y)}(1-2x-2y+4z)^{3/2}},$$
$$\frac{\partial f}{\partial y} = \frac{x - 2x^2 + 4zx - 2z}{\sqrt{(1-2x)(1-2y)}(1-2x-2y+4z)^{3/2}},$$
$$\frac{\partial f}{\partial z} = -\frac{(1-2x)(1-2y)}{\sqrt{(1-2x)(1-2y)}(1-2x-2y+4z)^{3/2}}.$$

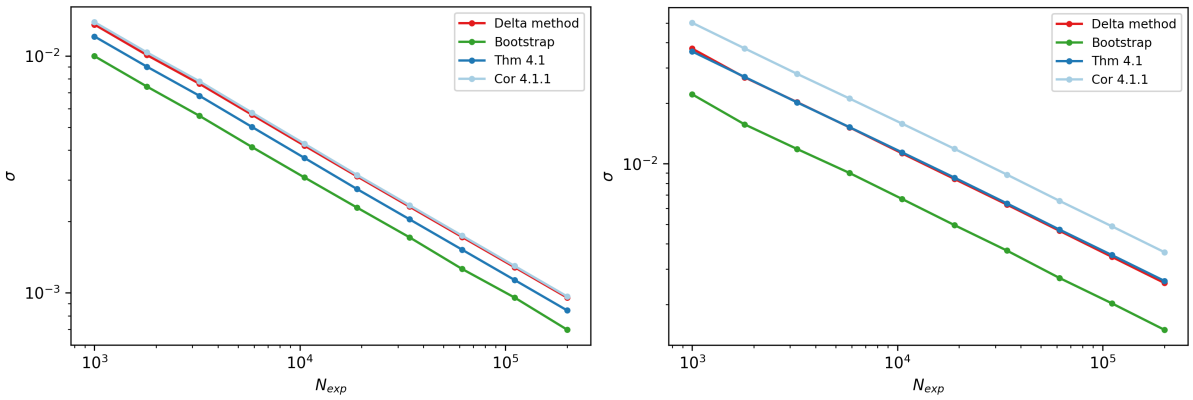
We now have an expression for the variance of the edge probabilities. We can take the square root to find the standard deviation. It is possible to work out the matrix multiplications of $f'(\mu)\Sigma(f'(\mu))^T$. This does however not lead to a nice simple expression. Therefore, the matrix multiplications will be worked out using a numerically.

4.1.3. Standard Deviation over N_{exp}

In Section 3.2 we have seen that the phenomenological noise model and the circuit level have about the same expected behaviour for edge estimation, i.e. no unexpected errors. To analyse the standard deviation, simulations of phenomenological noise are used to find the estimates since it is more convenient to initialise simulations with this noise model. We will first analyse the standard deviation as a function of number of shots.

The three analytic expressions for the standard deviation from the delta method, Theorem 4.1 and Corollary 4.1.1 are compared with the estimated standard deviation of space edge probabilities using the bootstrap method with 200 resamples. The results for initalised error probabilities $p = 0.05$ and $p = 0.10$ are shown on a log-log scale in Figure 4.1. Note that the $1/\sqrt{N_{exp}}$ behaviour is linear on a log-log scale. Also note that a constant difference between two lines on the log-log scale means an decreasing difference on a linear scale. From this we can conclude that all methods agree on the standard deviation as $N_{exp} \to \infty$. A similar plot for time edges shows the same results.

From the difference between Figure 4.1a and 4.1b we can already see that a higher error probability results in a higher standard deviation.



(a) (b)

Figure 4.1: Average standard deviation σ of space edges as a function of number of experiments N_{exp}. The data is obtained from a simulation assuming phenomenological noise with error probability (a) $p = 0.05$ and (b) $p = 0.10$. Time edges show a similar plots.

4.1.4. Standard Deviation over p

Equation (4.1) and (4.2) suggest that the standard deviation also depends on the edge probability. Therefore, we will also analyse the standard deviation as a function of error probability p. The simulations are again based on phenomenological noise to easily be able to initialise the simulation with a certain error probability. The results for the three analytical expressions are again compared to the bootstrap method. The results are shown in Figure 4.2. It can be seen that the difference between the methods increases as the probability increases. Theorem 4.1 and Corollary 4.1.1, seem to estimate the highest standard deviation. This can be attributed to the assumption that $p \ll 1/4$.

It can also be seen that the standard deviation does not behave like a $\sqrt{p}$ function as Equation (4.1) and (4.2) might suggest, but shows an exponential dependence on p. This means that a more complex dependence of p is indirectly incorporated in the $\langle d_i \rangle$ terms of these equations.

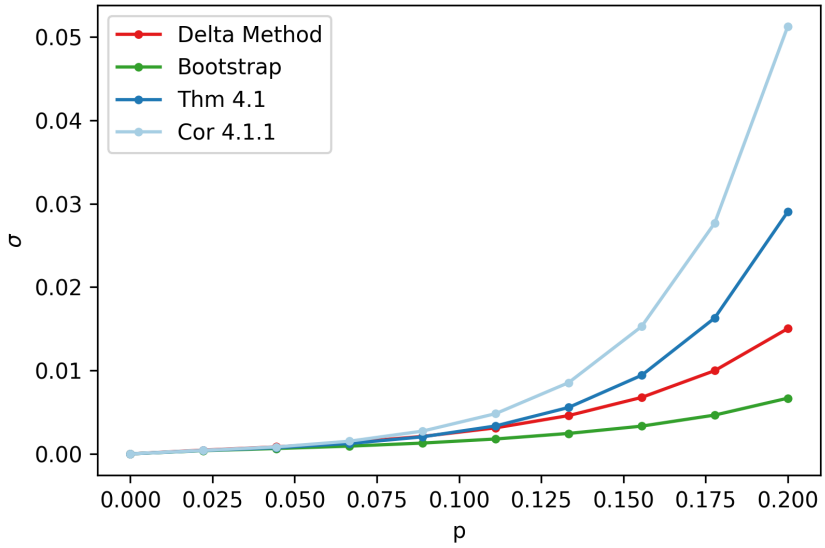


Figure 4.2: Average standard deviation σ for space edge probability as a function of initialised error probability p. The data is obtained from a simulation assuming phenomenological noise with $N_{exp} = 200,000$. The average standard deviation σ for time edge probability shows a similar plot.

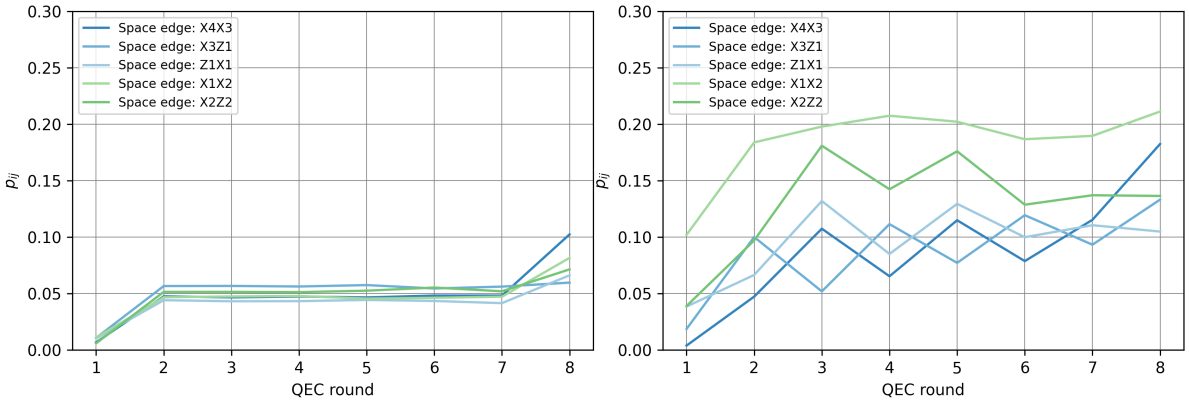
It should be mentioned that the probability estimation for bulk edges resulted in negative values under the square root of Equation (3.6) for high error probabilities combined with a low number of experiments. For $N_{exp} \approx 10^5$, the edge probability could be calculated without problems up to $p \approx 0.2$. For $N_{exp} \approx 10^6$, the edge probability could be calculated up to $p \approx 0.3$. This seems to be a sampling noise error. This error is not further investigated because higher values of p do not have any practical meaning. Due to the computational time as N_{exp} increases, the standard deviation is estimated from simulations up to a initialised error probability of $p = 0.2$.

4.2. Bulk Edge Estimation for Experimental Data

There is no experimental data for the circular repetition code. However, we can expect that the bulk edges behave similarly as the bulk edges for the normal repetition code, since those edges are the same. Therefore, we will analyse the probability of the different type of bulk edges for the experimental data of the normal repetition code. These estimated probabilities will be compared to the estimated probabilities from the circuit level noise model to check this noise model[1].

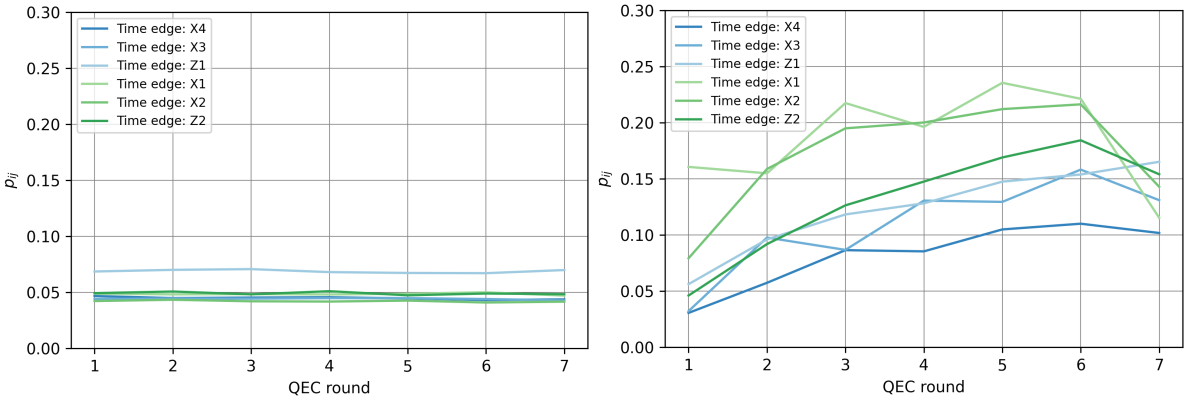
[1] Note that this information could also be obtained by comparing Figure 3.9 with Figure 3.13. However, these are shown on different color scales and there is a lot of other information in the figures. This makes the comparison less convenient.

Figure 4.3 shows the estimated probabilities of space edges for simulated data and the experimental data. It can be seen that the probabilities for experimental data are two to four times higher. Similar results can be seen in Figure 4.4 where the estimated probabilities are shown for time edges. Figure 4.5 shows the estimated probabilities of space-time edges. Here it can be seen that the probabilities for experimental data are two to eight times higher. These three figures might suggests that the initialised values for error probability on the data and ancilla qubits are incorrect. Note that difference can also be caused by errors we did not model with circuit level noise.



(a) (b)

Figure 4.3: Estimated probabilities of space edges for each QEC round. (a) Probabilities for simulations of the circuit level noise model. (b) Probabilities for experimental data.



(a) (b)

Figure 4.4: Estimated probabilities of time edges for each QEC round. A time edge is defined on a certain ancilla qubit from QEC round i to round $i+1$. (a) Probabilities for simulations of the circuit level noise model. (b) Probabilities for experimental data.

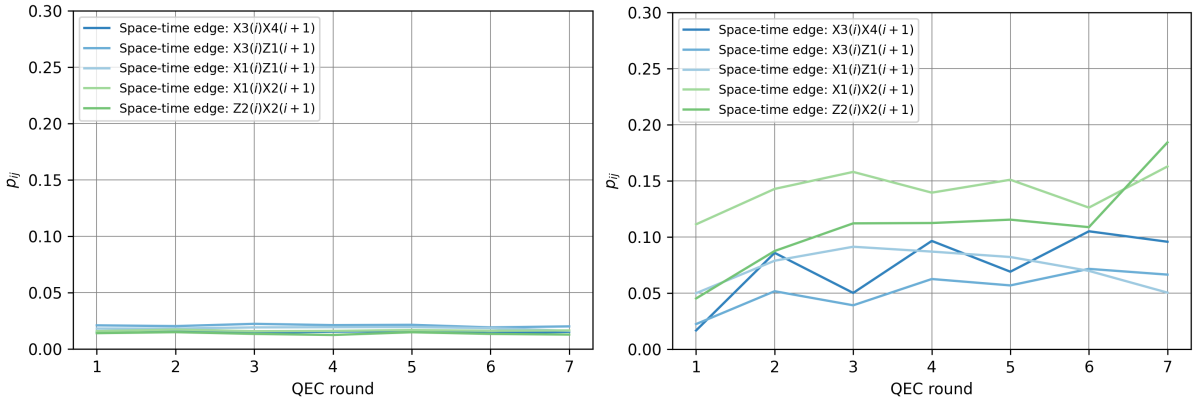


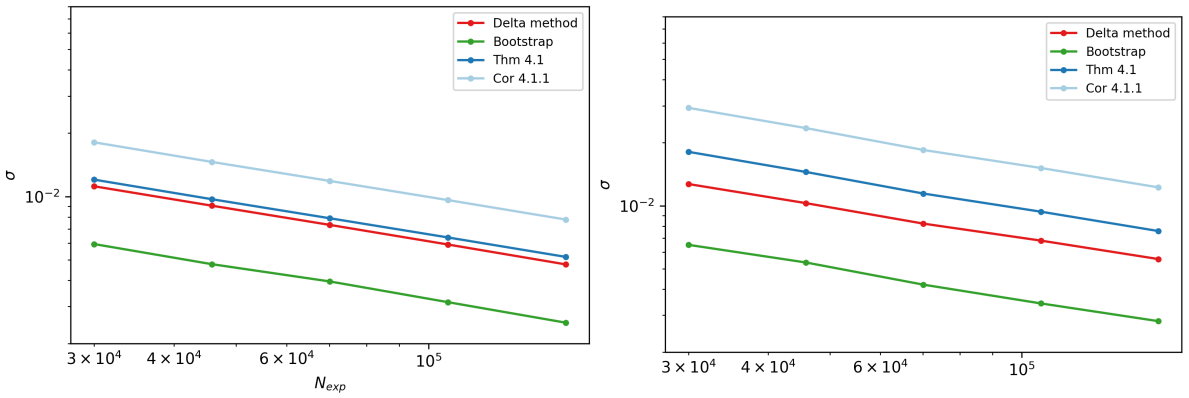
Figure 4.5: Estimated probabilities of space-time edges for each QEC round. (a) Probabilities for simulations of the circuit level noise model. (b) Probabilities for experimental data.

4.3. Standard deviation of Bulk Edges for Experimental Data

In Section 4.1, the standard deviation of the space and time edges are investigated for simulated data. In this section, the standard deviations will be analysed for experimental data. Note that for the experimental data, the error probability p is not a parameter we can change. Also, the experimental data that is provided for $N_r = 7$ QEC rounds is limited to $163,840$ shots. As a final note, the minimal number shots to not obtain the errors described in Subsection 4.1.4 is $30,000$. Therefore, it is only possible to analyse the standard deviation for experimental data as a function of number of shots in the range $[30000, 163840]$.

In Section 4.2 it is shown that the circuit level noise model is not correctly simulating the bulk edges for experimental data. We still want to compare the standard deviation of the edge probabilities with simulated data. Therefore, the standard deviation is again estimated for the phenomenological noise model. The noise model is initialised with two parameters: the error probability for data qubits and for ancilla qubits. For these values we take the average probabilities of space and time edges respectively from experimental data: $p_{data} = 0.1176$ and $p_{ancilla} = 0.1341$. Again, the standard deviation is calculated using the four different methods described in Section 4.1. Note that we are not able to estimate the standard deviation of space-time edges when using the phenomenological noise model. Therefore, we fill focus on the space and time edges only.

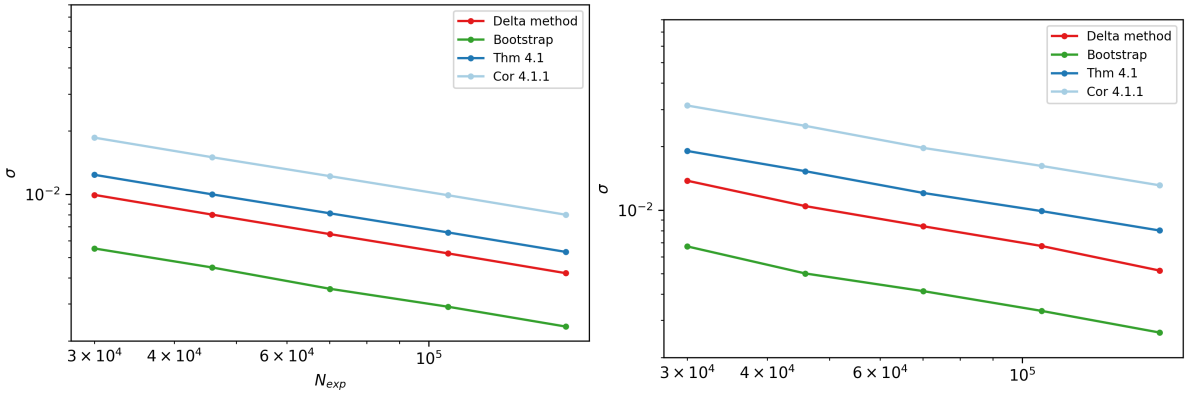
The average standard deviation of the space edge probabilities are shown in Figure 4.6. It can be seen that the delta method and bootstrap show similar results for the simulated and experimental data. The approximation formulas from Theorem 4.1 and Corollary 4.1.1 show a slightly larger difference between the results from simulated and experimental data. They do however still approximate the standard deviation to be in the same order of magnitude.



(a) (b)

Figure 4.6: Average standard deviation σ of space edges as a function of number of experiments N_{exp}. (a) The data is obtained from a numerical simulation assuming phenomenological noise with error probabilities $p_{data} = 0.1176$ and $p_{ancilla} = 0.1341$. (b) The data is obtained from the experiment of the repetition code on the superconducting quantum computer.

The average standard deviation of the time edge probabilities are shown in Figure 4.7. Here, we can see similar results as for the space edges. The standard deviation for the experimental data is estimated to be slightly higher than for the simulated data.



(a) (b)

Figure 4.7: Average standard deviation σ of time edges as a function of number of experiments N_{exp}. (a) The data is obtained from a numerical simulation assuming phenomenological noise with error probabilities $p_{data} = 0.1176$ and $p_{ancilla} = 0.1341$. (b) The data is obtained from the experiment of the repetition code on the superconducting quantum computer.

We conclude that, for similar N_{exp}, the standard error for space and time edges in the experiment is consistent with the standard error from phenomenological noise initialised with the average error data and ancilla error probabilities, even when these may be affected by non-conventional errors.

5

Conclusion

The goal of this thesis was to investigate the nonphysical error probabilities estimated from the experimental data. We have seen that some boundary edges are also estimated to be negative for numerically simulated data. These values are however within the range of the standard error around the expected value $p_e = 0$. Therefore, the nonphysical probabilities can be explained by sampling noise. This was not the case for the experimental data. Here, some edges are estimated to have a negative probability that is way outside of the range of the standard error around $p_e = 0$. Therefore, we conclude that the current error model does not model all errors that occur in reality.

The error model assumed only Pauli errors, which implies that every error results in at most 2 defects. Since we have shown that the nonphysical probabilities are not due to sampling noise, it is possible that non-conventional errors, such as leakage or crosstalk, are affecting these estimates. The inclusion of these errors, by for example also considering edges of higher order, is a subject of future work. Note that, instead of improving the error model, it is also possible to improve the experiments. If the error rates of non-conventional errors are much lower than the error rates of conventional errors, we can expect that the non-conventional errors would not affect the estimated error probabilities as much. Reducing non-Pauli errors like leakage or crosstalk is therefore also of great importance for improving quantum error correction.

Additionally, two approaches for estimating the boundary edge probability are compared. For the first approach, only edges to adjacent nodes are considered. For the second approach all other edges are considered. We have seen that the absolute error for boundary edges is significantly higher for the second approach. For phenomenological noise with error probability $p = 0.05$, the difference was already almost a factor of 10. The difference increases even more for higher error probabilities. We can conclude that the first approach results in a more accurate estimate for the boundary edge probabilities. The problem with this approach, however, is that one needs to have sufficient understanding of the errors occurring during the circuit and the edges they lead to.

We have also seen that the boundary edge probabilities for experimental data are way less negative for the first approach then for the second approach. Moreover, the average boundary edge probabilities over QEC rounds are positive for the first approach. Some edges, however, still show nonphysical probabilities that cannot be explained by sampling noise. Therefore, the boundary edges can still be incorrectly estimated via this procedure.

Furthermore, the circular repetition code gave the motivation to further investigate the bulk edge probabilities because this code does not have any boundary edges. First, the average standard deviation of the bulk edges are analysed as a function of N_{exp} and p for phenomenological noise. The standard deviations are estimated using bootstrap, the delta method and two approximation formulas from Google Quantum AI, 2021. It is shown that all methods agree when $N_{exp} \to \infty$. We have also seen that for a constant N_{exp}, the approximation formulas diverge from the bootstrap estimate for standard deviation as p increases.

The standard deviation is also analysed from the experimental data as a function of N_{exp}. Since the circuit level noise model, using the characterized error rates, estimates much lower error prob-

abilities compared to what is observed in the experiment, we instead performed a comparison with a phenomenology model parameterized by the average data and ancilla qubit error probabilities extracted from experiment. It is shown that the phenomenological noise model estimates similar standard errors compared to the experimental data. So, for similar N_{exp}, the standard error for space and time edges in the experiment is consistent with the standard error from phenomenological noise initialised with the average error data and ancilla error probabilities, even when these may be affected by non-conventional errors. The reason why both show similar results has not yet been investigated and would be interesting for further research.

To improve this report, it would be interesting to have a more in-depth analysis on the edge probabilities and standard deviations over QEC rounds instead of averaging them, because we have seen that for experimental data, the probabilities are not constant over the QEC rounds. Also note that it was not appropriate to average over all QEC rounds, since the first and last round are calculated differently from the other rounds.

In addition, for the bootstrap method it was chosen to estimate the standard deviation with a maximum of 200 samples due to computation times. If one wants to construct a confidence interval, a more efficient code would be desirable for running bootstrap with 2000 samples.

Finally, for the bootstrap method, it is assumed that the QEC experiments are independent and identically distributed. This is however not further investigated, but should be confirmed in order to make sure that the estimated standard deviations are more reliable.

Bibliography

Bickel, P. J., & Freedman, D. A. (1981). Some Asymptotic Theory for the Bootstrap. *The Annals of Statistics*, *9*(6), 1196–1217. https://doi.org/10.1214/aos/1176345637

Chen, E. H., Yoder, T. J., Kim, Y., Sundaresan, N., Srinivasan, S., Li, M., Córcoles, A. D., Cross, A. W., & Takita, M. (2022). Calibrated decoders for experimental quantum error correction. *Phys. Rev. Lett.*, *128*, 110504. https://doi.org/10.1103/PhysRevLett.128.110504

Efron, B., & Tibshirani, R. (1986). Bootstrap Methods for Standard Errors, Confidence Intervals, and Other Measures of Statistical Accuracy. *Statistical Science*, *1*(1), 54–75. https://doi.org/10.1214/ss/1177013815

Efron, B., & Hastie, T. (2016). *Computer age statistical inference*. Cambridge University Press.

Egger, D. J., Werninghaus, M., Ganzhorn, M., Salis, G., Fuhrer, A., Mueller, P., & Filipp, S. (2018). Pulsed reset protocol for fixed-frequency superconducting qubits. https://doi.org/10.48550/ARXIV.1802.08980

Feynman, R. P. (1982). Simulating physics with computers. *International Journal of Theoretical Physics*, *21*(6-7), 467–488. https://doi.org/10.1007/bf02650179

Fowler, A. G., Mariantoni, M., Martinis, J. M., & Cleland, A. N. (2012). Surface codes: Towards practical large-scale quantum computation. *Physical Review A*, *86*(3). https://doi.org/10.1103/physreva.86.032324

Geller, M. R., & Zhou, Z. (2013). Efficient error models for fault-tolerant architectures and the pauli twirling approximation. *Physical Review A*, *88*(1). https://doi.org/10.1103/physreva.88.012314

Google Quantum AI. (2021). Exponential suppression of bit or phase errors with cyclic error correction. *Nature*, *595*(7867), 383–387. https://doi.org/10.1038/s41586-021-03588-y

Magesan, E., Gambetta, J. M., & Emerson, J. (2012). Characterizing quantum gates via randomized benchmarking. *Physical Review A*, *85*(4). https://doi.org/10.1103/physreva.85.042311

Magnard, P., Kurpiers, P., Royer, B., Walter, T., Besse, J.-C., Gasparinetti, S., Pechal, M., Heinsoo, J., Storz, S., Blais, A., & Wallraff, A. (2018). Fast and unconditional all-microwave reset of a superconducting qubit. https://doi.org/10.48550/ARXIV.1801.07689

McEwen, M., Kafri, D., Chen, Z., Atalaya, J., Satzinger, K. J., Quintana, C., Klimov, P. V., Sank, D., Gidney, C., Fowler, A. G., Arute, F., Arya, K., Buckley, B., Burkett, B., Bushnell, N., Chiaro, B., Collins, R., Demura, S., Dunsworth, A., … Barends, R. (2021). Removing leakage-induced correlated errors in superconducting quantum error correction. *Nature Communications*, *12*(1). https://doi.org/10.1038/s41467-021-21982-y

O'Brien, T. E., Tarasinski, B., & DiCarlo, L. (2017). Density-matrix simulation of small surface codes under current and projected experimental noise. *npj Quantum Information*, *3*(1). https://doi.org/10.1038/s41534-017-0039-x

Raussendorf, R., & Harrington, J. (2007). Fault-tolerant quantum computation with high threshold in two dimensions. *Physical Review Letters*, *98*(19). https://doi.org/10.1103/physrevlett.98.190504

Rieffel, E., & Polak, W. (2011). *Quantum computing: A gentle introduction*. MIT Press.

Terhal, B. M. (2015). Quantum error correction for quantum memories. *Reviews of Modern Physics*, *87*(2), 307–346. https://doi.org/10.1103/revmodphys.87.307

van der Vaart, A. W. (1995). *Mathematische statistiek*.

Varbanov, B. M., Battistel, F., Tarasinski, B. M., Ostroukh, V. P., O'Brien, T. E., DiCarlo, L., & Terhal, B. M. (2020). Leakage detection for a transmon-based surface code. *npj Quantum Information*, *6*(1). https://doi.org/10.1038/s41534-020-00330-w

A

Derivation of the Edge Probability

In Section 3.1, the system of equations for $\langle d_i \rangle$, $\langle d_j \rangle$ and $\langle d_i d_j \rangle$ was written as

$$\begin{aligned}\langle d_i \rangle &= p_e(1 - q_{i,e}^{\mathit{eff}}) + (1 - p_e)q_{i,e}^{\mathit{eff}},\\ \langle d_j \rangle &= p_e(1 - q_{j,e}^{\mathit{eff}}) + (1 - p_e)q_{j,e}^{\mathit{eff}},\\ \langle d_i d_j \rangle &= p_e(1 - q_{i,e}^{\mathit{eff}})(1 - q_{j,e}^{\mathit{eff}}) + (1 - p_e)q_{i,e}^{\mathit{eff}}q_{j,e}^{\mathit{eff}}.\end{aligned}$$

We can solve the first two equations for $q_{i,e}^{\mathit{eff}}$ and $q_{j,e}^{\mathit{eff}}$ to find

$$q_{i,e}^{\mathit{eff}} = \frac{\langle d_i \rangle - p_e}{1 - 2p_e}, \quad \text{and} \quad q_{j,e}^{\mathit{eff}} = \frac{\langle d_j \rangle - p_e}{1 - 2p_e}.$$

These can be substituted in the equation for $\langle d_i d_j \rangle$ to obtain

$$\langle d_i d_j \rangle = p_e\left(1 - \frac{\langle d_i \rangle - p_e}{1 - 2p_e}\right)\left(1 - \frac{\langle d_j \rangle - p_e}{1 - 2p_e}\right) + (1 - p_e)\left(\frac{\langle d_i \rangle - p_e}{1 - 2p_e}\right)\left(\frac{\langle d_j \rangle - p_e}{1 - 2p_e}\right).$$

Multiplying both sides by $(1 - 2p_e)^2$ gives

$$\langle d_i d_j \rangle(1 - 2p_e)^2 = p_e(1 - 2p_e - \langle d_i \rangle + p_e)(1 - 2p_e - \langle d_j \rangle + p_e) + (1 - p_e)(1 - \langle d_i \rangle - p_e)(1 - \langle d_j \rangle - p_e).$$

Writing out all terms, we are left with a simple quadratic equation for p_e,

$$(4\langle d_i d_j \rangle - 2\langle d_i \rangle - 2\langle d_j \rangle + 1)p_e^2 + (-4\langle d_i d_j \rangle + 2\langle d_i \rangle + 2\langle d_j \rangle - 1)p_e + (\langle d_i d_j \rangle - \langle d_i \rangle\langle d_j \rangle) = 0.$$

Finally, we can solve this equation for the edge probability p_e to find

$$p_e = \frac{1}{2} - \sqrt{\frac{1}{4} - \frac{\langle d_i d_j \rangle - \langle d_i \rangle\langle d_j \rangle}{1 - 2\langle d_i \rangle - 2\langle d_j \rangle + 4\langle d_i d_j \rangle}}.$$

B

Circular Repetition Code

In this Section, the idea behind the circular repetition code will be explained. Just like an ancilla can measure the parity between two adjacent data qubits, an ancilla can also measure the parity between the two boundary qubits. This is only possible if the 1D string of data qubits can be folded in a 2D plane such that the boundary qubits become adjacent qubits as well. Note that this is exactly what is happening at our experiment. The 1D sting is connected on the layout of the surface code, which is 2D. One may have already seen how the boundary qubits can be connected in Figure 2.8b. The setup for the circular repetition code of length 7 on the surface code quantum chip is shown in Figure B.1. Note that the boundary qubits can not be connected for all lengths of a repetition code.

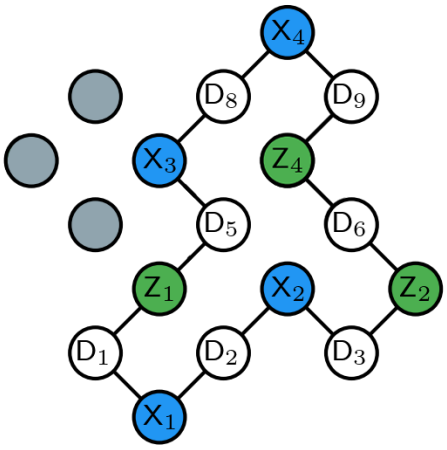


Figure B.1: Possible layout for the circular repetition code with length 7 on the quantum chip for the surface code.

Now what happens to the edge probability estimations for this circular repetition code? First, we assume that the bulk edges that are similar to the normal repetition code experiment. The only difference is that the boundary edges for the qubits at the boundary will be replaced by (bulk) space edges. In the probability matrix of all bulk edges we would therefore expect to find additional space edges between ancilla X_4 and Z_4 and between Z_2 and Z_4. The results for simulated data of the circular repetition code with phenomenological noise are shown in Figure B.2. Here, the expected additional space edges can be found in the upper left and lower right corner.

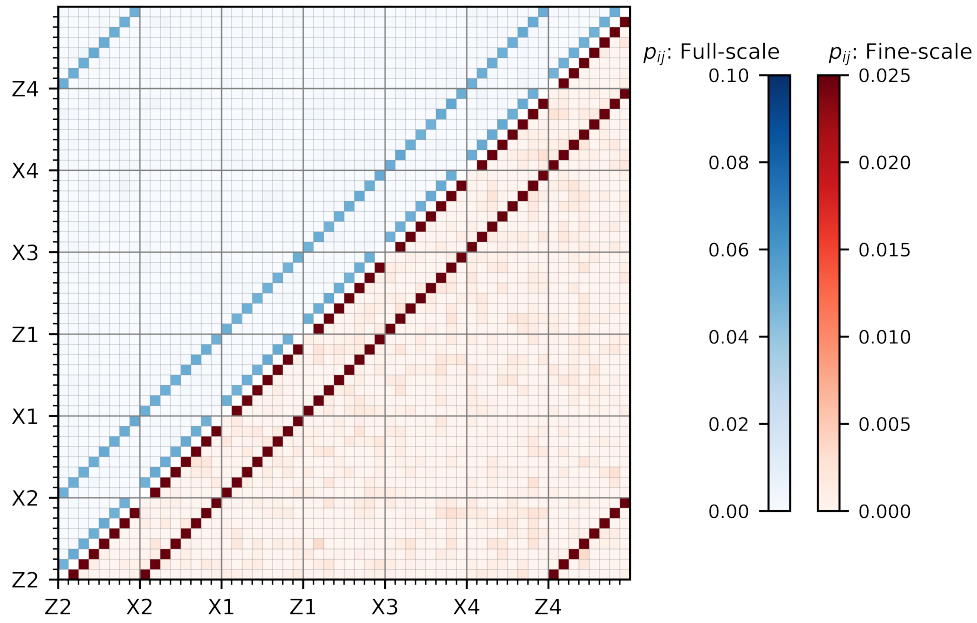


Figure B.2: Probability matrix for the bulk edges of a simulation of the circular repetition code with phenomenological noise. The major ticks represent the different ancilla qubits and the minor ticks represent the QEC rounds. Each square represents the probability of an edge between the nodes i and j. The matrix is symmetric about the diagonal. The upper half and lower half are represented on different scales, which is not relevant for this specific figure.

For now, there are no experimental results available for the circular repetition code.

C

Code and Data availability

The data and code that produce the plots for this thesis are available upon reasonable request. The Python function that are used for the simulations and analysis are presented in the following sections. The experimental data is provided by the DiCarlo group. The simulated data of the circuit level noise model is provided by Boris Varbanov.

C.1. Obtaining Data

C.1.1. Simulation of QEC Circuit

```
import stim

def generate_circuit(
    data_qubits,
    anc_qubits,
    qec_round,
    error_model,
    p_error,
    final_measurement,
    num_rounds,
):
    qubits = data_qubits + anc_qubits
    circuit = stim.Circuit()

    if error_model == "incoming_error":
        qubit_inds = [qubits.index(qubit_label) for qubit_label in data_qubits]
        circuit.append_operation("X_ERROR", qubit_inds, p_error[0])
        qubit_inds = [qubits.index(qubit_label) for qubit_label in anc_qubits]
        circuit.append_operation("X_ERROR", qubit_inds, p_error[1])

    for operation in qec_round:
        gate, target_qubits = operation
        qubit_inds = [qubits.index(qubit_label) for qubit_label in target_qubits]
        if error_model == "error_after_gate":
            if gate == "M":
                circuit.append_operation("X_ERROR", qubit_inds, p_error)
                circuit.append_operation(gate, qubit_inds)
            elif gate == "CZ":
                circuit.append_operation(gate, qubit_inds)
                circuit.append_operation("DEPOLARIZE2", qubit_inds, p_error)
            else:
                circuit.append_operation(gate, qubit_inds)
                circuit.append_operation("DEPOLARIZE1", qubit_inds, p_error)
        else:
            circuit.append_operation(gate, qubit_inds)

    circuit *= num_rounds

    gate, target_qubits = final_measurement
    qubit_inds = [qubits.index(qubit_label) for qubit_label in target_qubits]
```

```
        circuit.append_operation("X_ERROR", qubit_inds, p_error[0])
        circuit.append_operation(gate, qubit_inds)

    return circuit

# ===== Initialisation =====
DATA_QUBITS = ["D9", "D8", "D5", "D1", "D2", "D3", "D6"]
NUM_DATA = len(DATA_QUBITS)
ANC_QUBITS = ["X4", "X3", "Z1", "X1", "X2", "Z2"]
NUM_ANC = len(ANC_QUBITS)
QUBITS = DATA_QUBITS + ANC_QUBITS

INIT_ANC_STATE = xr.DataArray(
    data=np.zeros(NUM_ANC, dtype=int),
    dims=["anc_qubit"],
    coords=dict(anc_qubit=ANC_QUBITS),
)

p_data = 0.05
p_ancilla = 0.05

error_model = "incoming_error"
p_error = (p_data, p_ancilla)

QEC_ROUND = [
    ("H", ["X2"] + ["Z1"] + ["X4"]),
    ("CZ", ["D3", "X2"] + ["D1", "Z1"] + ["D8", "X4"]),
    ("TICK", []),
    ("CZ", ["D2", "X2"] + ["D5", "Z1"] + ["D9", "X4"]),
    ("H", ["X2"] + ["Z1"] + ["X4"]),
    ("H", ["Z2"] + ["X1"] + ["X3"]),
    ("CZ", ["D6", "Z2"] + ["D2", "X1"] + ["D5", "X3"]),
    ("TICK", []),
    ("CZ", ["D3", "Z2"] + ["D1", "X1"] + ["D8", "X3"]),
    ("H", ["Z2"] + ["X1"] + ["X3"]),
    ("M", ANC_QUBITS),
    ("X", DATA_QUBITS),
]
FINAL_MEAS = ("M", DATA_QUBITS)

# ===== Define circuit and obtain Defects =====
NUM_ROUNDS = 7
QEC_ROUNDS = np.arange(NUM_ROUNDS)

NUM_SHOTS = 200000
SHOTS = np.arange(NUM_SHOTS)

circuit = generate_circuit(
    DATA_QUBITS, ANC_QUBITS, QEC_ROUND, error_model, p_error, FINAL_MEAS, NUM_ROUNDS
)

sampler = circuit.compile_sampler()

samples = sampler.sample(shots=NUM_SHOTS)
anc_outcomes, data_outcomes = np.split(samples, [-NUM_DATA], axis=1)
anc_outcomes = anc_outcomes.reshape(NUM_SHOTS, NUM_ROUNDS, NUM_ANC)

anc_meas = xr.DataArray(
    data=anc_outcomes,
    dims=["shot", "qec_round", "anc_qubit"],
    coords=dict(shot=SHOTS, qec_round=QEC_ROUNDS, anc_qubit=ANC_QUBITS),
)
data_meas = xr.DataArray(
    data=data_outcomes,
    dims=["shot", "data_qubits"],
    coords=dict(shot=SHOTS, data_qubits=DATA_QUBITS),
)

syndrome_proj = syndrome_projection(data_meas, SHOTS, ANC_QUBITS)
syndrome_proj["qec_round"] = NUM_ROUNDS
```

```
syndrome_meas = get_syndrome(anc_meas)
syndromes = xr.concat([syndrome_meas, syndrome_proj], "qec_round")
NUM_ROUNDS += 1

defect_matrix = get_defects(syndromes, INIT_ANC_STATE)
```

C.1.2. Experiment

```
import numpy as np
import netCDF4 as nc
import xarray as xr

# ===== Loading data for circuit level noise, qec_rounds=7, init_state = 0101101 =====
nc_file = nc.Dataset("../data/circuit_level/Init_0101101/bitflip_code_mem_exp_nrounds_7_data.nc",'r')
dataset = xr.open_dataset(xr.backends.NetCDF4DataStore(nc_file))

NUM_ROUNDS = np.size(dataset.qec_round)
QEC_ROUNDS = np.arange(1,NUM_ROUNDS+1)

NUM_SHOTS = 163840 #np.size(dataset.shot)
SHOTS = np.arange(1,NUM_SHOTS+1)

initial_syndrome = dataset.init_state.data[:-1]^dataset.init_state.data[1:]
INIT_ANC_STATE = xr.DataArray(
    data=np.array(initial_syndrome),
    dims=["anc_qubit"],
    coords=dict(anc_qubit=ANC_QUBITS),
)

anc_meas = dataset.anc_meas[:NUM_SHOTS]
data_meas = dataset.data_meas[:NUM_SHOTS]

# ===== Loading experimental data, qec_rounds=7, init_state = 0101101 =====
nc_file = nc.Dataset("../data/experiment/Init_0101101/bitflip_code_mem_exp_nrounds_7_data.nc",'r')
dataset = xr.open_dataset(xr.backends.NetCDF4DataStore(nc_file))

NUM_ROUNDS = np.size(dataset.qec_round)
QEC_ROUNDS = np.arange(1,NUM_ROUNDS+1)

NUM_SHOTS = np.size(dataset.shot)
SHOTS = np.arange(1,NUM_SHOTS+1)

initial_syndrome = dataset.init_state.data[:-1]^dataset.init_state.data[1:]
initial_syndrome ^= dataset.par_corrections
INIT_ANC_STATE = xr.DataArray(
    data=np.array(initial_syndrome),
    dims=["anc_qubit"],
    coords=dict(anc_qubit=ANC_QUBITS),
)

anc_meas = dataset.anc_meas[:NUM_SHOTS]
data_meas = dataset.data_meas[:NUM_SHOTS]

anc_meas = dataset.anc_meas.transpose('shot','qec_round','anc_qubit')
anc_meas = xr.where(anc_meas < dataset.anc_threshold, 0,1)


data_meas = dataset.data_meas.transpose('shot','data_qubit')
data_meas = xr.where(data_meas < dataset.data_threshold, 0, 1)
```

C.2. Decoding

C.2.1. Samples to Defects

```
import xarray as xr

def get_syndrome(meas_matrix: xr.DataArray):
    syndrome_matrix = meas_matrix ^ meas_matrix.shift(qec_round=1, fill_value=0)
```

```
    return syndrome_matrix

def get_defects(syndrome_matrix: xr.DataArray, initial_state: xr.DataArray):
    syndrome_matrix_copy = syndrome_matrix.copy()
    syndrome_matrix_copy.data[:, -1, :] = initial_state
    defect_matrix = syndrome_matrix ^ syndrome_matrix_copy.roll(qec_round=1)
    return defect_matrix

def syndrome_projection(data_meas: xr.DataArray, shots, anc_qubits):
    syndrome_proj = data_meas.data[:, :-1] ^ data_meas.data[:, 1:]
    syndrome_proj = xr.DataArray(
        data=syndrome_proj,
        dims=["shot", "anc_qubit"],
        coords=dict(shot=shots, anc_qubit=anc_qubits),
    )
    return syndrome_proj
```

C.2.2. Probability Estimations

```
import numpy as np
import xarray as xr
from itertools import product

from numba import njit, prange
from numba import vectorize, float64

from .indices import index

# ===== Estimate <di> and <didj> =====
def estimate_di(defect_matrix: xr.DataArray):
    num_shots = np.size(defect_matrix.data, axis=0)
    avg_di_array = np.sum(defect_matrix.data, axis=0) / num_shots
    return avg_di_array


def estimate_didj_slow(defect_matrix: xr.DataArray, num_rounds, num_anc):
    num_shots = np.size(defect_matrix.data, axis=0)
    avg_didj_array = np.zeros((num_rounds**2, num_anc**2))
    for shot in range(num_shots):
        avg_didj_array += np.kron(
            defect_matrix.data[shot, :, :], defect_matrix.data[shot, :, :]
        )
    avg_didj_array = avg_didj_array / num_shots
    avg_didj_array = avg_didj_array.reshape(num_rounds, num_rounds, num_anc, num_anc)
    return avg_didj_array


@njit(fastmath=True, parallel=False)
def tensor_prod(A, B, out):
    for i in prange(A.shape[0]):
        for j in range(B.shape[0]):
            for k in range(A.shape[1]):
                for l in range(B.shape[1]):
                    out[i, j, k, l] = A[i, k] * B[j, l]
    return out


@njit(parallel=True)
def estimate_didj(defect_matrix, out, num_shots, num_rounds, num_anc):
    for shot in prange(num_shots):
        out_float64 = np.empty(
            (num_rounds, num_rounds, num_anc, num_anc), dtype=np.float64
        )
        out += tensor_prod(defect_matrix[shot], defect_matrix[shot], out_float64)
    return out / num_shots


# ===== Estimate probabilities =====
def prob_bulk_edge(edge, avg_di_array, avg_didj_array):
    i, j = edge
```

```
    round_i, anc_i = i
    round_j, anc_j = j

    di = avg_di_array[round_i, anc_i]
    dj = avg_di_array[round_j, anc_j]
    didj = avg_didj_array[round_i, round_j, anc_i, anc_j]

    num = didj - di * dj
    denom = 1 - 2 * di - 2 * dj + 4 * didj

    if num / denom > 1 / 4:
        frac = 1 / 4
        print("negative number in sqrt.")
    else:
        frac = num / denom

    return 1 / 2 - np.sqrt(1 / 4 - frac)


def adjacency_set(
    node, num_rounds, num_anc, exclude_edge=None, include_spacetime=False
):
    """
                        | (i-1)
    (i-num_rounds)    - i -   (i+num_rounds)
                        | (i+1)
    """
    round_i, anc_i = node
    adj_set = [
        (node, (round_i - 1, anc_i)),
        (node, (round_i + 1, anc_i)),
        (node, (round_i, anc_i - 1)),
        (node, (round_i, anc_i + 1)),
    ]
    if anc_i == 0:
        adj_set.remove((node, (round_i, anc_i - 1)))
        adj_set.append("boundary")
        if (
            (include_spacetime == True)
            and (round_i != 0)
            and (round_i != num_rounds - 1)
        ):
            adj_set.append((node, (round_i - 1, anc_i + 1)))  # X4
    if anc_i == num_anc - 1:
        adj_set.remove((node, (round_i, anc_i + 1)))
        adj_set.append("boundary")
        if (
            (include_spacetime == True)
            and (round_i != num_rounds - 2)
            and (round_i != num_rounds - 1)
        ):
            adj_set.append((node, (round_i + 1, anc_i - 1)))  # Z2
    if round_i == 0:
        adj_set.remove((node, (round_i - 1, anc_i)))
    if round_i == num_rounds - 1:
        adj_set.remove((node, (round_i + 1, anc_i)))


    if exclude_edge == None:
        pass
    else:
        adj_set.remove(exclude_edge)

    return adj_set


def odd_permuations(dim):
    iter = product([0, 1], repeat=dim)
    b_list = [b for b in iter if sum(b) % 2 == 1]
    return b_list
```

```
def prob_boundary_edge(
    node, avg_di_array, avg_didj_array, num_rounds, num_anc, include_spacetime
):
    round_i, anc_i = node
    di = avg_di_array[round_i, anc_i]

    assert anc_i == 0 or anc_i == num_anc - 1, "Node is not a boundary node."

    adjacent_edges = adjacency_set(
        node, num_rounds, num_anc, "boundary", include_spacetime
    )
    prob_adj_edges = np.zeros(len(adjacent_edges))
    for e in range(len(adjacent_edges)):
        prob_adj_edges[e] = prob_bulk_edge(
            adjacent_edges[e], avg_di_array, avg_didj_array
        )

    qi_eff = 0
    b_list = odd_permuations(len(adjacent_edges))
    for b in b_list:
        prod = 1
        for j in range(len(adjacent_edges)):
            prod *= (prob_adj_edges[j] ** b[j]) * (
                (1 - prob_adj_edges[j]) ** (1 - b[j])
            )
        qi_eff += prod

    return (di - qi_eff) / (1 - 2 * qi_eff)


@vectorize([float64(float64, float64)])
def g(p, q):
    return p * (1 - q) + (1 - p) * q


# ===== Probabilities in Array =====
def edge_prob_array(avg_di_array, avg_didj_array, num_rounds, num_anc):
    """
    returns matrix with pij elements
    """
    prob_array = np.zeros((num_anc * num_rounds, num_anc * num_rounds))
    for round_i in range(num_rounds):
        for anc_i in range(num_anc):
            for round_j in range(num_rounds):
                for anc_j in range(num_anc):
                    i = index(round_i, anc_i, num_rounds)
                    j = index(round_j, anc_j, num_rounds)
                    prob_array[i][j] = prob_bulk_edge(
                        ((round_i, anc_i), (round_j, anc_j)),
                        avg_di_array,
                        avg_didj_array,
                    )

    np.fill_diagonal(prob_array, 0)
    return prob_array


def bound_prob_array_adj(
    avg_di_array, avg_didj_array, num_rounds, num_anc, include_spacetime=False
):
    """
    returns vector with piB elements using only adjacent nodes
    """
    prob_array = np.zeros((num_rounds, num_anc))
    for round_i in range(num_rounds):
        for anc_i in range(num_anc):
            if (anc_i == 0) or (anc_i == num_anc - 1):
                i = index(round_i, anc_i, num_rounds)
                prob_array[round_i][anc_i] = prob_boundary_edge(
                    (round_i, anc_i),
```

```
                    avg_di_array,
                    avg_didj_array,
                    num_rounds,
                    num_anc,
                    include_spacetime,
                )
            else:
                pass
    return prob_array


def bound_prob_array_all(avg_di_array, pij_array, num_rounds, num_anc):
    """
    returns array with piB elements using google function S14, S15
    """
    np.fill_diagonal(pij_array, 0)
    qie_eff = np.zeros(np.shape(pij_array)[0])

    qie_eff = g.reduce(pij_array, axis=0)

    qie_eff = np.reshape(qie_eff, (num_anc, num_rounds))
    qie_eff = qie_eff.T

    piB = (avg_di_array - qie_eff) / (1 - 2 * qie_eff)

    return piB
```

C.2.3. Indices

```
def index(round, ancilla, num_rounds):
    return round + num_rounds * ancilla


def index_inv(index, num_rounds):
    round = index % num_rounds
    ancilla = index // num_rounds
    return round, ancilla


def edge_lists(num_rounds, num_anc, include_spacetime=False):
    space_edges = []
    time_edges = []
    bound_edges = []
    space_time_edges = []

    for round_i in range(num_rounds):
        for anc_i in range(num_anc):
            # space edges
            if anc_i != num_anc - 1:
                space_edges.append(((round_i, anc_i), (round_i, anc_i + 1)))

            # time edges
            if round_i != num_rounds - 1:
                time_edges.append(((round_i, anc_i), (round_i + 1, anc_i)))

            # boundary edges
            if (anc_i == 0) or (anc_i == num_anc - 1):
                bound_edges.append((round_i, anc_i))
    for anc in range(1,num_anc):
        if anc%2==0:
            for round in range(num_rounds-1):
                space_time_edges.append( ((round+1,anc),(round,anc-1)) )
        else:
            for round in range(num_rounds-1):
                space_time_edges.append( ((round,anc),(round+1,anc-1)) )

    if include_spacetime==True:
        return space_edges, time_edges, bound_edges, space_time_edges
    else:
        return space_edges, time_edges, bound_edges
```

C.3. Statistics

C.3.1. Bootstrap

```
import numpy as np
from decoder.estimations import estimate_di, estimate_didj, edge_prob_array
from decoder.estimations import bound_prob_array_adj, bound_prob_array_all


def bootstrap_std_bulk(defect_matrix, num_resamples, num_shots, num_rounds, num_anc):
    sample_size = num_shots
    bootstrap_pij = np.zeros(
        (num_resamples, num_rounds * num_anc, num_rounds * num_anc)
    )

    for b in range(num_resamples):
        sample_indices = np.random.choice(num_shots, size=sample_size, replace=True)
        sample_def_mat = defect_matrix[sample_indices]

        avg_di_array = estimate_di(sample_def_mat)
        out_float64 = np.zeros(
            (num_rounds, num_rounds, num_anc, num_anc), dtype=np.float64
        )
        avg_didj_array = estimate_didj(
            sample_def_mat.data, out_float64, sample_size, num_rounds, num_anc
        )

        bootstrap_pij[b] = edge_prob_array(
            avg_di_array, avg_didj_array, num_rounds, num_anc
        )

    return np.mean(bootstrap_pij, axis=0), np.std(bootstrap_pij, axis=0)


def bootstrap_std_bound_adj(
    defect_matrix,
    num_resamples,
    num_shots,
    num_rounds,
    num_anc,
    include_spacetime=False,
):
    sample_size = num_shots
    bootstrap_piB = np.zeros((num_resamples, num_rounds, num_anc))

    for b in range(num_resamples):
        sample_indices = np.random.choice(num_shots, size=sample_size, replace=True)
        sample_def_mat = defect_matrix[sample_indices]

        avg_di_array = estimate_di(sample_def_mat)
        out_float64 = np.zeros(
            (num_rounds, num_rounds, num_anc, num_anc), dtype=np.float64
        )
        avg_didj_array = estimate_didj(
            sample_def_mat.data, out_float64, sample_size, num_rounds, num_anc
        )

        bootstrap_piB[b] = bound_prob_array_adj(
            avg_di_array, avg_didj_array, num_rounds, num_anc, include_spacetime
        )

    return np.mean(bootstrap_piB, axis=0), np.std(bootstrap_piB, axis=0)


def bootstrap_std_bound_all(
    defect_matrix, num_resamples, num_shots, num_rounds, num_anc
):
    sample_size = num_shots
    bootstrap_piB = np.zeros((num_resamples, num_rounds, num_anc))

    for b in range(num_resamples):
```

```
        sample_indices = np.random.choice(num_shots, size=sample_size, replace=True)
        sample_def_mat = defect_matrix[sample_indices]

        avg_di_array = estimate_di(sample_def_mat)
        out_float64 = np.zeros(
            (num_rounds, num_rounds, num_anc, num_anc), dtype=np.float64
        )
        avg_didj_array = estimate_didj(
            sample_def_mat.data, out_float64, sample_size, num_rounds, num_anc
        )
        pij_array = edge_prob_array(avg_di_array, avg_didj_array, num_rounds, num_anc)

        bootstrap_piB[b] = bound_prob_array_all(
            avg_di_array, pij_array, num_rounds, num_anc
        )

    return np.mean(bootstrap_piB, axis=0), np.std(bootstrap_piB, axis=0)
```

C.3.2. Delta Method

```
import numpy as np

# ===== Delta Method - Bulk edge =====
def covariance_matrix_bulk(di, dj, didj):
    cov_matrix = np.array(
        [
            [di - di**2, didj - di * dj, didj * (1 - di)],
            [didj - di * dj, dj - dj**2, didj * (1 - dj)],
            [didj * (1 - di), didj * (1 - dj), didj - didj**2],
        ]
    )
    return cov_matrix


def gradient_bulk(x, y, z):
    denom = np.sqrt((1 - 2 * x) * (1 - 2 * y)) * (1 - 2 * x - 2 * y + 4 * z) ** (3 / 2)
    grad = (
        np.array(
            [
                (y - 2 * y**2 + 4 * y * z - 2 * z),
                (x - 2 * x**2 + 4 * x * z - 2 * z),
                (1 - 2 * x) * (1 - 2 * y),
            ]
        )
        / denom
    )
    return grad


def delta_variance_bulk_edge(edge, avg_di_array, avg_didj_array, num_shots):
    i, j = edge
    round_i, anc_i = i
    round_j, anc_j = j

    di = avg_di_array[round_i, anc_i]
    dj = avg_di_array[round_j, anc_j]
    didj = avg_didj_array[round_i, round_j, anc_i, anc_j]

    grad = gradient_bulk(di, dj, didj)
    cov_matrix = covariance_matrix_bulk(di, dj, didj)

    var_normal = (grad @ cov_matrix) @ grad
    var_pe = var_normal / num_shots

    return var_pe
```

C.3.3. Approximation Formulas

```
import numpy as np
```

```
from decoder.indices import index

def google_std_bulk_edge_S19(edge, avg_di_array, pij_matrix, num_shots, num_rounds):
    i, j = edge
    round_i, anc_i = i
    round_j, anc_j = j

    xi = avg_di_array[round_i, anc_i]
    xj = avg_di_array[round_j, anc_j]
    pij = pij_matrix[index(round_i, anc_i, num_rounds)][
        index(round_j, anc_j, num_rounds)
    ]

    frac = (xi * xj * (1 - xi) * (1 - xj)) / ((1 - 2 * xi) ** 2 * (1 - 2 * xj) ** 2)

    return np.sqrt(pij * (1 - pij) + frac) / np.sqrt(num_shots)


def google_std_bulk_edge_S20(edge, avg_di_array, pij_matrix, num_shots, num_rounds):
    i, j = edge
    round_i, anc_i = i
    round_j, anc_j = j

    xi = avg_di_array[round_i, anc_i]
    xj = avg_di_array[round_j, anc_j]
    pij = pij_matrix[index(round_i, anc_i, num_rounds)][
        index(round_j, anc_j, num_rounds)
    ]

    frac = xi * xj / ((1 - 2 * xi) ** 2 * (1 - 2 * xj) ** 2)

    return np.sqrt(pij + frac) / np.sqrt(num_shots)
```

C.3.4. Absolute Error

```
import numpy as np

from decoder.estimations import estimate_di, estimate_didj
from decoder.estimations import (
    edge_prob_array,
    bound_prob_array_adj,
    bound_prob_array_all,
)
from decoder.indices import index


def total_diff_space(prob_array, p_error, num_rounds):
    """
    input: prob_array = prob matrix of bulk edges with dims (num_rounds*num_anc , num_rounds*num_anc).
    """
    p_est = np.diag(prob_array, k=num_rounds)
    return np.abs(p_est - p_error)


def total_diff_time(prob_array, p_error, num_rounds):
    """
    input: prob_array = prob matrix of bulk edges with dims (num_rounds*num_anc , num_rounds*num_anc).
    """
    p_est = np.diag(prob_array, k=1)
    p_est = np.delete(p_est, np.s_[num_rounds - 1 :: num_rounds])
    return np.abs(p_est - p_error)


def total_diff_bound(prob_array, p_error, num_anc):
    """
    input: prob_array = prob matrix of boundary edges with dims (num_rounds, num_anc).
    """
    p_est = prob_array[:, :: num_anc - 1]
    return np.abs(p_est - p_error)
```

```
def total_diff_spacetime():
    # what will be p_error? You can't initialise circuit_level noise with a certain p_st = x.
    return


def average_differences(defect_matrix, p_error, num_shots, num_rounds, num_anc):
    p_data, p_ancilla = p_error
    avg_di_array = estimate_di(defect_matrix)

    out_float64 = np.zeros((num_rounds, num_rounds, num_anc, num_anc), dtype=np.float64)
    avg_didj_array = estimate_didj(
        defect_matrix.data, out_float64, num_shots, num_rounds, num_anc
    )

    pij_array = edge_prob_array(avg_di_array, avg_didj_array, num_rounds, num_anc)
    piB_array_adj = bound_prob_array_adj(
        avg_di_array, avg_didj_array, num_rounds, num_anc
    )
    piB_array_all = bound_prob_array_all(avg_di_array, pij_array, num_rounds, num_anc)

    avg_diff_space = np.mean(total_diff_space(pij_array, p_data, num_rounds))
    avg_diff_time = np.mean(total_diff_time(pij_array, p_ancilla, num_rounds))
    avg_diff_bound_adj = np.mean(total_diff_bound(piB_array_adj, p_data, num_anc))
    avg_diff_bound_all = np.mean(total_diff_bound(piB_array_all, p_data, num_anc))

    return avg_diff_space, avg_diff_time, avg_diff_bound_adj, avg_diff_bound_all


def func(N, a, b):
    return a * N**b


# ===== Difference over N =====
def average_difference_over_N(
    defect_matrix, p_error, N_min, N_max, num_rounds, num_anc
):
    num_edge_types = 4
    N_range = np.logspace(np.log10(N_min), np.log10(N_max), 20, dtype=int)
    total_difference_array = np.zeros((len(N_range), num_edge_types))
    for i in range(len(N_range)):
        print(N_range[i])
        total_difference_array[i] = average_differences(
            defect_matrix, p_error, N_range[i], num_rounds, num_anc
        )

    return N_range, total_difference_array


# ===== Difference over p =====
def average_difference_over_p(
    big_defect_matrix, p_range, num_edge_types, num_shots, num_rounds, num_anc
):
    """
    Input: big_defect_matrix with dims [p_list, NUM_SHOTS, NUM_ROUNDS, NUM_ANC]
    """

    total_difference_array = np.zeros((len(p_range), num_edge_types))

    for i in range(len(p_range)):
        print(p_range[i])
        p_error = (p_range[i], p_range[i])
        total_difference_array[i] = average_differences(
            big_defect_matrix[i], p_error, num_shots, num_rounds, num_anc
        )

    return total_difference_array
```

C.3.5. Standard Deviations

```
import numpy as np
import matplotlib.pyplot as plt

from decoder.estimations import estimate_di, estimate_didj, edge_prob_array
from decoder.indices import index

from .google import google_std_bulk_edge_S19, google_std_bulk_edge_S20
from .delta_method import delta_variance_bulk_edge

# ===== Std over N =====
def prob_edge(edge, pij_matrix, num_rounds):
    i, j = edge
    round_i, anc_i = i
    round_j, anc_j = j

    return pij_matrix[index(round_i, anc_i, num_rounds)][
        index(round_j, anc_j, num_rounds)
    ]


def std_over_N(edge_list, defect_matrix, N_min, N_max, num_steps, num_rounds, num_anc):
    N_range = np.logspace(np.log10(N_min), np.log10(N_max), num_steps, dtype=int)
    delta_var_space_edge = np.zeros((num_steps, len(edge_list)))
    google_var_space_edgeS19 = np.zeros((num_steps, len(edge_list)))
    google_var_space_edgeS20 = np.zeros((num_steps, len(edge_list)))

    for n in range(len(N_range)):
        print(n)
        avg_di_array = estimate_di(defect_matrix[: N_range[n]])
        out_float64 = np.zeros(
            (num_rounds, num_rounds, num_anc, num_anc), dtype=np.float64
        )
        avg_didj_array = estimate_didj(
            defect_matrix.data[: N_range[n]],
            out_float64,
            N_range[n],
            num_rounds,
            num_anc,
        )
        pij_matrix = edge_prob_array(avg_di_array, avg_didj_array, num_rounds, num_anc)
        for e in range(len(edge_list)):
            google_var_space_edgeS19[n][e] = (
                google_std_bulk_edge_S19(
                    edge_list[e], avg_di_array, pij_matrix, N_range[n], num_rounds
                )
                ** 2
            )
            google_var_space_edgeS20[n][e] = (
                google_std_bulk_edge_S20(
                    edge_list[e], avg_di_array, pij_matrix, N_range[n], num_rounds
                )
                ** 2
            )
            delta_var_space_edge[n][e] = delta_variance_bulk_edge(
                edge_list[e], avg_di_array, avg_didj_array, N_range[n]
            )
    delta_std_over_N = np.sqrt(np.mean(delta_var_space_edge, axis=1))
    google_std_over_NS19 = np.sqrt(np.mean(google_var_space_edgeS19, axis=1))
    google_std_over_NS20 = np.sqrt(np.mean(google_var_space_edgeS20, axis=1))

    return N_range, delta_std_over_N, google_std_over_NS19, google_std_over_NS20


# ===== Std over p =====
def std_over_p(edge_list, big_defect_matrix, p_range, num_shots, num_rounds, num_anc):
    num_steps = len(p_range)
    delta_var_space_edge = np.zeros((num_steps, len(edge_list)))
    google_var_space_edgeS19 = np.zeros((num_steps, len(edge_list)))
    google_var_space_edgeS20 = np.zeros((num_steps, len(edge_list)))

    for p in range(num_steps):
```

```
        avg_di_array = estimate_di(big_defect_matrix[p])
        out_float64 = np.zeros(
            (num_rounds, num_rounds, num_anc, num_anc), dtype=np.float64
        )
        avg_didj_array = estimate_didj(
            big_defect_matrix[p].data, out_float64, num_shots, num_rounds, num_anc
        )
        pij_matrix = edge_prob_array(avg_di_array, avg_didj_array, num_rounds, num_anc)
        for e in range(len(edge_list)):
            google_var_space_edgeS19[p][e] = (
                google_std_bulk_edge_S19(
                    edge_list[e], avg_di_array, pij_matrix, num_shots, num_rounds
                )
                ** 2
            )
            google_var_space_edgeS20[p][e] = (
                google_std_bulk_edge_S20(
                    edge_list[e], avg_di_array, pij_matrix, num_shots, num_rounds
                )
                ** 2
            )
            delta_var_space_edge[p][e] = delta_variance_bulk_edge(
                edge_list[e], avg_di_array, avg_didj_array, num_shots
            )

    delta_std_over_p = np.sqrt(np.mean(delta_var_space_edge, axis=1))
    google_std_over_pS19 = np.sqrt(np.mean(google_var_space_edgeS19, axis=1))
    google_std_over_pS20 = np.sqrt(np.mean(google_var_space_edgeS20, axis=1))

    return delta_std_over_p, google_std_over_pS19, google_std_over_pS20
```