

Modern Quantum Error Correction

Exercise Solutions

ICTP–SAIFR Summer School · São Paulo · 2026

July 2026

Table of contents

1	Lecture 1 — Introduction to quantum error correction	1
1.1	Easy	1
1.2	Medium	1
2	Lecture 2 — Pauli and Clifford structure	3
2.1	Easy	3
2.2	Medium	3
3	Lecture 3 — Physical noise to Pauli channels	4
3.1	Easy	4
3.2	Medium	4
4	Lecture 4 — Detecting regions and detector error models	6
4.1	Easy	6
4.2	Medium	6
5	Lecture 5 — Decoding as probabilistic inference	7
5.1	Easy	7
5.2	Medium	7
6	Lecture 6 — Hands-on QEC with Stim and PyMatching	8
6.1	Easy	8
6.2	Medium	8

1 Lecture 1 — Introduction to quantum error correction

1.1 Easy

An error flips a syndrome bit exactly when it anticommutes with the corresponding check. Therefore

Error	s_1 for Z_1Z_2	s_2 for Z_2Z_3
I	0	0
X_1	1	0
X_2	1	1
X_3	0	1

Both codewords $|000\rangle$ and $|111\rangle$ have eigenvalue $+1$ under both checks. Hence every superposition $\alpha|000\rangle + \beta|111\rangle$ also has the same check values. The check measurements reveal only whether the state lies in a syndrome subspace, not the logical amplitudes α, β .

1.2 Medium

At $p = 0.02$,

$$\begin{aligned}
p_L &= 3(0.02)^2(0.98) + (0.02)^3 \\
&= 0.001184.
\end{aligned}$$

Thus the encoded failure probability is about 5.92% of the unencoded value $p = 0.02$. Equivalently, it is reduced from 2% to 0.1184% under the stated idealized assumptions.

The phase error Z_1 commutes with both Z -type checks and therefore has syndrome $(0, 0)$. Nevertheless,

$$Z_1 (\alpha|000\rangle + \beta|111\rangle) = \alpha|000\rangle - \beta|111\rangle.$$

This is the logical operation $\bar{Z}$. The shortest undetectable X -type logical operator has weight three, so $d_X = 3$ and one X error can be corrected. But $\bar{Z} = Z_1$ has weight one, so the distance against arbitrary Pauli errors is $d = 1$.

2 Lecture 2 — Pauli and Clifford structure

2.1 Easy

Conjugation by $\text{CNOT}_{c \rightarrow t}$ gives

$$X_c \mapsto X_c X_t, \quad Z_c \mapsto Z_c,$$

$$X_t \mapsto X_t, \quad Z_t \mapsto Z_c Z_t.$$

Propagating the final ancilla observable backward through $\text{CNOT}_{2 \rightarrow a}$ and $\text{CNOT}_{1 \rightarrow a}$ gives

$$Z_a \leftarrow Z_2 Z_a \leftarrow Z_1 Z_2 Z_a.$$

The ancilla starts in the $+1$ eigenstate of Z_a , so its final Z measurement has the eigenvalue of $Z_1 Z_2$.

2.2 Medium

The middle data qubit $q1$ controls one CNOT into each check ancilla. Therefore

$$X_1 \mapsto X_1 X_{q3} X_{q4}.$$

The X factors on the ancillas flip both final Z measurements, so the syndrome is $(1, 1)$. The data error remains on $q1$.

In contrast, a Z on a CNOT control remains on the control:

$$Z_1 \mapsto Z_1.$$

It does not propagate to either ancilla, so it produces syndrome $(0, 0)$ while acting as a logical phase error. A stabilizer simulator obtains both results by updating Pauli strings under Clifford conjugation and testing commutation with the measured Paulis. It never needs the 2^n state amplitudes.

3 Lecture 3 — Physical noise to Pauli channels

3.1 Easy

Using

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

we find

$$\frac{1}{T_\phi} = \frac{1}{60\,\mu\text{s}} - \frac{1}{160\,\mu\text{s}} = \frac{1}{96\,\mu\text{s}}, \quad \boxed{T_\phi = 96\,\mu\text{s}}.$$

The consistency condition holds because $T_2 = 60\,\mu\text{s} < 2T_1 = 160\,\mu\text{s}$. With zero-temperature relaxation and dephasing acting together, the long-time state is

$$\rho_{\text{ss}} = |0\rangle\langle 0|.$$

3.2 Medium

In inverse nanoseconds,

$$\Omega = \frac{\pi}{40} \simeq 0.0785398, \quad \kappa = \frac{1}{80000} = 1.25 \times 10^{-5}.$$

In the ordered basis (I, X, Y, Z) ,

$$G = \begin{pmatrix} 0 & 0 & 0 & 0 \\ 0 & -\kappa/2 & 0 & 0 \\ 0 & 0 & -\kappa/2 & -\Omega \\ \kappa & 0 & \Omega & -\kappa \end{pmatrix}.$$

For example, the matrix exponential can be evaluated with

```
import numpy as np
from scipy.linalg import expm

tau = 40.0 # ns
omega = np.pi / tau
kappa = 1.0 / 80_000.0
G = np.array(
    [
        [0, 0, 0, 0],
        [0, -kappa / 2, 0, 0],
        [0, 0, -kappa / 2, -omega],
        [kappa, 0, omega, -kappa],
    ]
)
R_x = expm(tau * G)
```

Numerically,

$$R_X \simeq \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 0.9997500 & 0 & 0 \\ -0.0003183 & 0 & -0.9996251 & -2.5 \times 10^{-9} \\ 2.5 \times 10^{-8} & 0 & 2.5 \times 10^{-9} & -0.9996251 \end{pmatrix}.$$

When $\kappa = 0$, the first two coordinates are unchanged and the Y - Z plane rotates by $\Omega\tau = \pi$, giving $\text{diag}(1, 1, -1, -1)$.

The entries below R_{II} in the first column encode non-unital drift; here the dominant one is $R_{YI} \simeq -3.18 \times 10^{-4}$ because relaxation acts while the drive rotates the Bloch vector. Pauli twirling removes the off-diagonal process-matrix terms, including coherent interference and non-unital information, and keeps only the diagonal Pauli probabilities.

4 Lecture 4 — Detecting regions and detector error models

4.1 Easy

Both regions have a Z -type label. Since $XZ = -ZX$ and $YZ = -ZY$, either an X or a Y error flips both D_0 and D_1 . Since Z commutes with itself, a Z error flips neither detector.

4.2 Medium

The DEM instructions are

```
error(0.01) D0 D1
error(0.02) D1 L0
error(0.005) D0 D1 D2
```

The first mechanism is an ordinary edge between D_0 and D_1 . The second is a boundary edge incident on D_1 and is also logical-labeled because it contains $L0$. The third mechanism is a three-detector hyperedge.

5 Lecture 5 — Decoding as probabilistic inference

5.1 Easy

The first fault vector selects the first column of H :

$$H(1, 0, 0, 0, 0)^T = (1, 0, 1)^T.$$

One second solution is

$$\mathbf{e}' = (0, 1, 1, 0, 0)^T.$$

It selects the XOR of columns two and three:

$$\begin{pmatrix} 0 \\ 1 \\ 1 \end{pmatrix} \oplus \begin{pmatrix} 1 \\ 1 \\ 0 \end{pmatrix} = \begin{pmatrix} 1 \\ 0 \\ 1 \end{pmatrix}.$$

Thus distinct fault configurations can produce the same syndrome.

5.2 Medium

The largest individual weight is 0.30, so a most-likely-error decoder chooses e_1 and therefore logical class zero.

The total weights of the two logical classes are

$$W_0 = 0.30 + 0.08 = 0.38, \quad W_1 = 0.24 + 0.23 = 0.47.$$

Maximum-likelihood logical decoding therefore chooses class one. The answers differ because the most likely individual fault belongs to class zero, while the combined weight of all class-one explanations is larger. The decoder is scored on its logical prediction, so the correct objective is the marginalized probability of each logical class, not identification of a microscopic fault.

6 Lecture 6 — Hands-on QEC with Stim and PyMatching

6.1 Easy

The following helper evaluates either circuit after `build_repetition_memory` has been defined:

```
def evaluate(circuit, shots=1_000, seed=10):
    detector_samples, actual_observables = (
        circuit.compile_detector_sampler(seed=seed).sample(
            shots=shots,
            separate_observables=True,
        )
    )
    dem = circuit.detector_error_model(decompose_errors=True)
    matching = pymatching.Matching.from_detector_error_model(dem)
    predicted_observables = matching.decode_batch(detector_samples)
    failed = np.any(
        predicted_observables != actual_observables,
        axis=1,
    )
    return {
        "detector_shape": detector_samples.shape,
        "observable_shape": actual_observables.shape,
        "detector_fraction": detector_samples.mean(),
        "logical_failure_fraction": failed.mean(),
    }

ideal = build_repetition_memory(rounds=3)
noisy = build_repetition_memory(
    rounds=3,
    p_data=0.01,
    p_meas=0.01,
)
print(evaluate(ideal))
print(evaluate(noisy))
```

Both cases have detector shape (1000,8) and observable shape (1000,1). The ideal case has detector-event fraction zero and logical failure fraction zero. The noisy numerical fractions are stochastic and can vary with the software version and seed; the printed values are the reproducible answer for the stated environment.

6.2 Medium

Sample once from the true circuit, then reuse exactly those samples with each assumed DEM:

```
shots = 20_000
true_circuit = build_repetition_memory(
    rounds=3,
    p_data=0.02,
    p_meas=0.02,
)
detector_samples, actual_observables = (
    true_circuit.compile_detector_sampler(seed=11).sample(
        shots=shots,
        separate_observables=True,
    )
)

for assumed_p in [0.005, 0.02, 0.05]:
    assumed_circuit = build_repetition_memory(
        rounds=3,
        p_data=assumed_p,
        p_meas=assumed_p,
```



```

)
assert assumed_circuit.num_detectors == true_circuit.num_detectors
assumed_dem = assumed_circuit.detector_error_model(
    decompose_errors=True
)
matching = pymatching.Matching.from_detector_error_model(assumed_dem)
prediction = matching.decode_batch(detector_samples)
failure_fraction = np.any(
    prediction != actual_observables,
    axis=1,
).mean()
print(assumed_p, failure_fraction)

```

For an exact maximum-likelihood decoder, the true model minimizes the expected logical error under data generated by that model. In this experiment, PyMatching performs minimum-weight matching rather than exact logical-class marginalization, and the three assumed probabilities may preserve the same ordering of candidate corrections. Consequently, the matched model should be the best-informed choice, but the measured rates can tie or a mismatched model can appear slightly better because of approximation, unchanged decisions, or finite-shot fluctuations. The experiment demonstrates sensitivity to decoder weights only if the resulting decisions actually change.