Christian B. Mendl, Martina Nibbi, Pedro Hack, Irene López Gutiérrez due: 30 May 2023, 08:00 on Moodle

Tutorial 5 (Quantum circuit implementation of bit flip error detection)
Recall the bit flip code introduced in the lecture, encoding a logical qubit by three physical qubits:

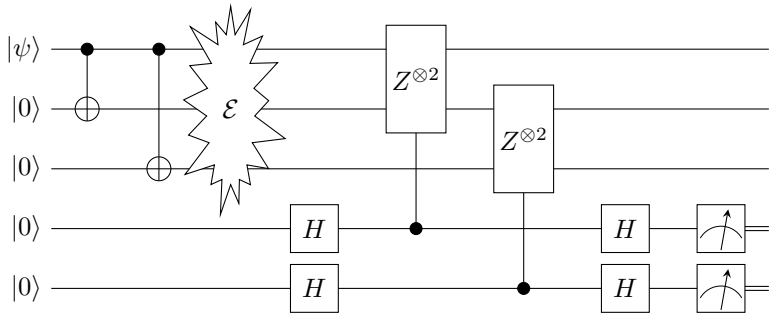
$$|0_L\rangle = |000\rangle$$
$$|1_L\rangle = |111\rangle$$

Here we study syndrome diagnosis for the bit flip code using the quantum circuit representation.

(a) The following circuit first encodes a single qubit $|\psi\rangle = \alpha\,|0\rangle + \beta\,|1\rangle$ via the first two CNOT gates, resulting in

$$|\psi_L\rangle = \alpha\,|0_L\rangle + \beta\,|1_L\rangle\,.$$

Then a bit flip error $\mathcal{E}$ affects the physical qubits storing $|\psi_L\rangle$. Here we separately consider the cases that none or exactly one qubit flipped. The remaining components of the circuit realize error detection by measuring the observables Z_1Z_2 and Z_2Z_3. Step through the circuit and determine that the measurement results indeed allows us to diagnose a bit flip on a single qubit.



(b) Can you design a simpler circuit, equivalent to (a), for performing the error detection?

Exercise 5.1 (Three qubit bit flip code analysis)
We analytically study the effects of bit flip errors and possible recovery using the three qubit bit flip code in detail. First recall the definition of the bit flip channel (acting on the j-th qubit):

$$\mathcal{E}_j(\rho) = (1-p)\rho + pX_j\rho X_j,$$

i.e., a flip $|0\rangle \leftrightarrow |1\rangle$ of the j-th qubit occurs with probability $p \in [0,1]$.

(a) In general, any of the qubits could flip. Given an initial three qubit state $|\psi\rangle$, the density matrix after such potential errors thus reads

$$\rho_{\text{err}} = \mathcal{E}_1(\mathcal{E}_2(\mathcal{E}_3(|\psi\rangle\langle\psi|))).$$

Expand this expression in terms of the X_j operators. (You can retain $|\psi\rangle$ symbolically.)

(b) We now assume that $|\psi\rangle$ is an encoded logical qubit, $|\psi\rangle = \alpha\,|0_L\rangle + \beta\,|1_L\rangle = \alpha\,|000\rangle + \beta\,|111\rangle$. Compute the density matrix ρ' after the error correction procedure has been applied to ρ_{err}.
Hint: First determine what the error detection step (measurement of the projection operators $P_0, \dots P_3$ defined in the lecture) yields for each possible case of $0, \dots, 3$ bit flips. The second step, recovery, then flips the qubit back (if any) which the error detection diagnosed to have flipped. However, note that this does not always correspond to the actual error. You should arrive at $\rho' = (1-r)\,|\psi\rangle\langle\psi| + r\,X_1X_2X_3\,|\psi\rangle\langle\psi|\,X_1X_2X_3$ with a to-be determined coefficient $r \in [0,1]$.

(c) The capability of the error correction procedure can be quantified by the fidelity, defined as

$$F = \sqrt{\langle\psi|\rho'|\psi\rangle}.$$

In general, $F \le 1$, and perfect recovery corresponds to $F = 1$. Compute F for the setup in (b). What is the minimal and maximal fidelity (depending on α and β)?

5.1 :

(a) $\rho_{out} =$

$= \mathcal{E}_1(\mathcal{E}_2(\mathcal{E}_3(|\psi\rangle\langle\psi|))) = \mathcal{E}_1(\mathcal{E}_2((1-p_3)\rho + p_3 X_3 \rho X_3) =$

$= \mathcal{E}_1\big((1-p_2)(1-p_3)\rho + p_2(1-p_3) X_2 \rho X_2 + (1-p_2)p_3 X_3 \rho X_3 + p_2 \cdot p_3 X_2 X_3 \rho X_3 X_2\big) =$

$= (1-p_1)(1-p_2)(1-p_3)\rho + p_1(1-p_2)(1-p_3) X_1 \rho X_1 +$
$(1-p_1)p_2(1-p_3) X_2 \rho X_2 + p_1 \cdot p_2 \cdot (1-p_3) X_1 X_2 \rho X_2 X_1 +$
$(1-p_1)(1-p_2)p_3 X_3 \rho X_3 + p_1(1-p_2)p_3 X_1 X_3 \rho X_3 X_1 +$
$(1-p_1)p_2 p_3 X_2 X_3 \rho X_3 X_2 + p_1 p_2 p_3 X_1 X_2 X_3 \rho X_3 X_2 X_1$

(b) $|\psi\rangle = \alpha|0000\rangle + \beta|1111\rangle$ initial $\rho = |\psi\rangle\langle\psi| = |\alpha|^2|0000\rangle\langle 0000| + \alpha\beta^*|0000\rangle\langle 1111| +$
$+ \beta\alpha^*|1111\rangle\langle 0000| + |\beta|^2|1111\rangle\langle 1111|$

$\rho' = ?$ after error correction.

$p_1 = p_2 = p_3 = p$

$\rho' = (1-p)^3\rho + (1-p)^2 p\, X_1 \rho X_1 + (1-p)^2 p \cdot \rho + (1-p)^2 p \cdot X_2 \rho X_2$
$+ (1-p)p^2 X_1 X_2 \rho X_2 X_1 + (1-p)^2 p\, X_3 \rho X_3 + (1-p)p^2 X_1 X_3 \rho X_3 X_1$
$+ (1-p)p^2 X_2 X_3 \rho X_3 X_2 + p^3 X_1 X_2 X_3 \rho X_3 X_2 X_1$

$p_0 = |0000\rangle\langle 0000| + |1111\rangle\langle 1111|$
$p_1 = |1000\rangle\langle 1000| + |0111\rangle\langle 0111|$
$p_2 = |0100\rangle\langle 0100| + |1011\rangle\langle 1011|$
$p_3 = |0010\rangle\langle 0010| + |1101\rangle\langle 1101|$

$p_0 \rho p_0 = (1-p)^3\big[|\alpha|^2|0000\rangle\langle 0000| + |\beta|^2|1111\rangle\langle 1111|\big] +$
$p^3\big[|\alpha|^2|1111\rangle\langle 1111| + |\beta|^2|0000\rangle\langle 0000|\big] =$

$p_1 \rho p_1 = (1-p)^2 p\big[|\alpha|^2|1000\rangle\langle 1000| + |\beta|^2|0111\rangle\langle 0111|\big] +$
$(1-p)p^2\big[|\alpha|^2|0111\rangle\langle 0111| + |\beta|^2|1000\rangle\langle 1000|\big]$

$p_2 \rho p_2 = (1-p)^2 p\big[|\alpha|^2|0100\rangle\langle 0100| + |\beta|^2|1011\rangle\langle 1011|\big] +$
$(1-p)p^2\big[|\alpha|^2|1011\rangle\langle 1011| + |\beta|^2|0100\rangle\langle 0100|\big]$

$p_3 \rho p_3 = (1-p)^2 p\big[|\alpha|^2|0010\rangle\langle 0010| + |\beta|^2|1101\rangle\langle 1101|\big] +$
$(1-p)p^2\big[|\alpha|^2|1101\rangle\langle 1101| + |\beta|^2|0010\rangle\langle 0010|\big] +$

$$\rho' = \left[(1-\rho)^3 + |1-\rho|^2\rho + (1-\rho)^2\rho + (1-\rho)\rho\right](|\alpha|^2+|\beta|^2)^2)\,|\psi\rangle\langle\psi|$$
$$+\left[\rho^3 + (1-\rho)\rho^2 + |1-\rho|\rho^2 + (1-\rho)\rho^2\right](|\alpha|^2+|\beta|^2)\,X_1X_2X_3\,|\psi\rangle\langle\psi|\,X_3X_2X_1$$

$$\lambda = \rho^3 + \rho^2 - \rho^3 + \rho^2 - \rho^3 + \rho^2 - \rho^3 = 3\rho^2 - 2\rho^3 = \boxed{\rho^2(3-2\rho)}$$

$$(1-\lambda) = 1 - 3\rho + 3\rho^2 + \rho^3 + 3(\rho - 2\rho^2 + \rho^3)$$
$$= 1 - \cancel{3\rho} + 3\rho^2 - \rho^3 + \cancel{3\rho} - 6\rho^2 + 3\rho^3$$
$$= 1 - 3\rho^2 + 2\rho^3 = 1 - (3\rho^2 - 2\rho^3)$$

(c).

$$F = \sqrt{\langle\psi|\rho'|\psi\rangle} = \sqrt{(1-\lambda) + \lambda(\alpha^*\beta + \beta^*\alpha)^2} = \sqrt{1 - 3\rho^2 + 2\rho^3 + \rho^2(3-2\rho)(\alpha^*\beta + \beta^*\alpha)^2}$$

$|\psi\rangle = \alpha|0000\rangle + \beta|1111\rangle$

$X_1X_2X_3|\psi\rangle = \alpha|1111\rangle + \beta|0000\rangle$

$\langle\psi|X_1X_2X_3|\rho|\psi\rangle\langle\psi|X_3X_2X_1|\psi\rangle$

$= \alpha\langle 0000| + \beta\langle 1111|\;\big|\;\alpha|1111\rangle + \beta|0000\rangle\big\rangle\big\langle\alpha^*\langle 1111| + \beta^*\langle 0000|\;\big|\;\alpha|0000\rangle + \beta|1111\rangle$

$= (\alpha^*\beta + \beta^*\alpha)(\beta^*\alpha + \beta\alpha^*) = (\alpha^*\beta + \beta^*\alpha)^2$

using the following identity
$z_1\cdot\bar{z}_2 + \bar{z}_1\cdot z_2 = 2\mathrm{Re}(\bar{z}_1 z_2)$

max fidelity = 1 is $\alpha = 1$ or $\beta = 1$

min fidelity = $\sqrt{1 - 3\rho^2 + 2\rho^3 + \rho^2(3-2\rho)\,4\underbrace{\mathrm{Re}^2(\alpha\bar{\beta})}_{\geq 0}} = \sqrt{1 - 3\rho^2 + 2\rho^3}$ — min value.

Exercise 5.2 (Monte Carlo simulation of quantum bit flip errors)
In this exercise we explore Monte Carlo simulations of noise effects. The simulation algorithm repeats the following steps n times, with n the number of Monte Carlo samples (or "realizations"). In each realization, one keeps track of the quantum statevector $|\psi\rangle$. The effect of a noise operation is applied to $|\psi\rangle$ at random, with probability p corresponding to the noise strength. For example, in case of bit flip noise, one applies the Pauli-X gate to $|\psi\rangle$ with probability p. The final density matrix is then approximated as

$$\rho_{\mathsf{mc}} = \frac{1}{n}\sum_{i=1}^{n} |\psi_i\rangle \langle\psi_i| ,$$

where $|\psi_i\rangle$ is the output quantum state of the i-th realization.

As before we consider bit flip errors in this exercise, described by the bit flip channel (on the j-th qubit)

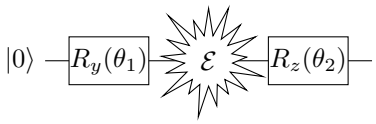
$$\mathcal{E}_j(\rho) = (1-p)\rho + pX_j\rho X_j,$$

i.e., a flip $|0\rangle \leftrightarrow |1\rangle$ occurs with probability $p \in [0,1]$. For the following numerical simulations, set $p = 0.2$.

(a) Implement the described algorithm for the case of a single qubit $|\psi\rangle$ initially in state $|0\rangle$ and affected by bit flip noise. Compare the resulting Monte Carlo density matrix ρ_{mc} with the exact solution $\rho_{\mathsf{ref}} = \mathcal{E}(|0\rangle\langle 0|)$ by computing the distance $\epsilon = \|\rho_{\mathsf{mc}} - \rho_{\mathsf{ref}}\|$ for $n = 100, 1000, 10000$ samples. To visualize the accuracy of the Monte Carlo method, plot ϵ in dependence of n on a log-log scale. For comparison, also show the (theoretically predicted) $1/\sqrt{n}$ curve in the plot.

Hint: You can execute a code block with probability p via `if np.random.rand() < p: ...`. The NumPy equivalent of the outer product $|\psi\rangle\langle\psi|$ is `np.outer(psi, psi.conj())`. The norm function in NumPy is `np.linalg.norm(...)`.

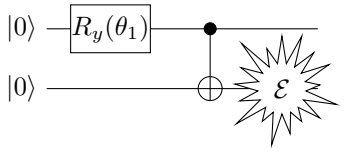
(b) A useful property of the algorithm is the fact that quantum circuit gates can be included as well, simply by applying them to the statevector $|\psi\rangle$. Extend the simulation from (a) to the following scenario: starting from $|0\rangle$, the gate $R_y(\theta_1)$ with $\theta_1 = 2\pi/5$ acts on $|\psi\rangle$, then a bit flip occurs with probability p, and finally $|\psi\rangle$ is transformed by $R_z(\theta_2)$ with $\theta_2 = 2\pi \cdot \frac{2}{7}$.



Use $n = 10000$ samples for this simulation, and evaluate the deviation from the reference density matrix ρ_{ref}: $\epsilon = \|\rho_{\mathsf{mc}} - \rho_{\mathsf{ref}}\|$.

Hint: To compute ρ_{ref}, recall that applying a gate U to a density matrix ρ is the conjugation $\rho \mapsto U\rho U^\dagger$. It is sufficient to calculate ρ_{ref} numerically (instead of symbolically). You should find that $\epsilon \lesssim 0.01$.

(c) We now consider a two qubit setup: starting with $|00\rangle$, the gate $R_y(\theta_1)$ with $\theta_1 = 2\pi/5$ acts on the first qubit, followed by a CNOT gate, and subsequently a bit flip error affects the second qubit.



Simulate this scenario analogously to (b), including the deviation from the reference density matrix.

Hint: You can obtain the matrix representation of a single-qubit gate U acting on the j-th qubit (using zero-based indexing for j) via $U_j = I_{2^j} \otimes U \otimes I_{2^{\ell-j-1}}$, where I_d is the $d \times d$ identity matrix and ℓ denotes the overall number of qubits.

hw5

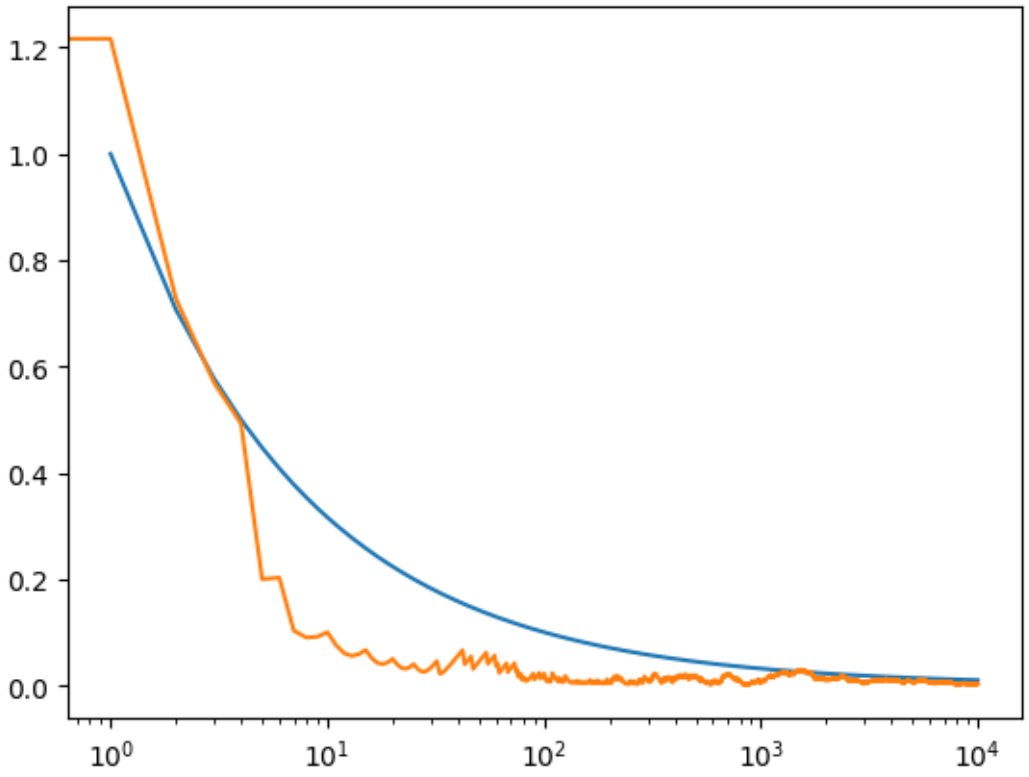
May 28, 2023

[ ]:
```
#Ex 5.2 a
# import qiskit
import numpy as np
import matplotlib.pyplot as plt

p = 0.2
N = 10000
sigmaX = np.array([[0., 1.],[1., 0.]])
state_vec = np.array([1., 0.])
state_ground = np.tensordot(state_vec.conj().T, state_vec,0)
state_dens = np.tensordot(state_vec.conj().T, state_vec,0)
state_teoretical = (1-p)*state_ground + p*(np.tensordot(np.tensordot(sigmaX,␣
 ↪state_ground,1),sigmaX,1))
print(state_teoretical)

states = []
states.append(np.linalg.norm(state_dens - state_teoretical))[0,1,0,0],
                [0,0,0,1],
                [0,0,1,0]])

for n in range(N):
    if (np.random.rand() < p):
        state_dens += np.array([[0., 0.],[0., 1.]])
    else:
        state_dens += np.array([[1., 0.],[0., 0.]])

    states.append(np.linalg.norm(1./(n+1)*state_dens - state_teoretical))



print(state_dens/(N+1))


fig = plt.figure()
ax = fig.add_subplot(1, 1, 1)

values = []
```

```
for n in range(N):
    values.append(1/np.sqrt(n))

plt.plot(values)
plt.plot(states)

plt.xscale('log')
```

```
[[0.8 0. ]
 [0.  0.2]]
[[0.80211979 0.        ]
 [0.         0.19788021]]
```

```
/tmp/ipykernel_6696/1336586889.py:36: RuntimeWarning: divide by zero encountered
in double_scalars
  values.append(1/np.sqrt(n))
```

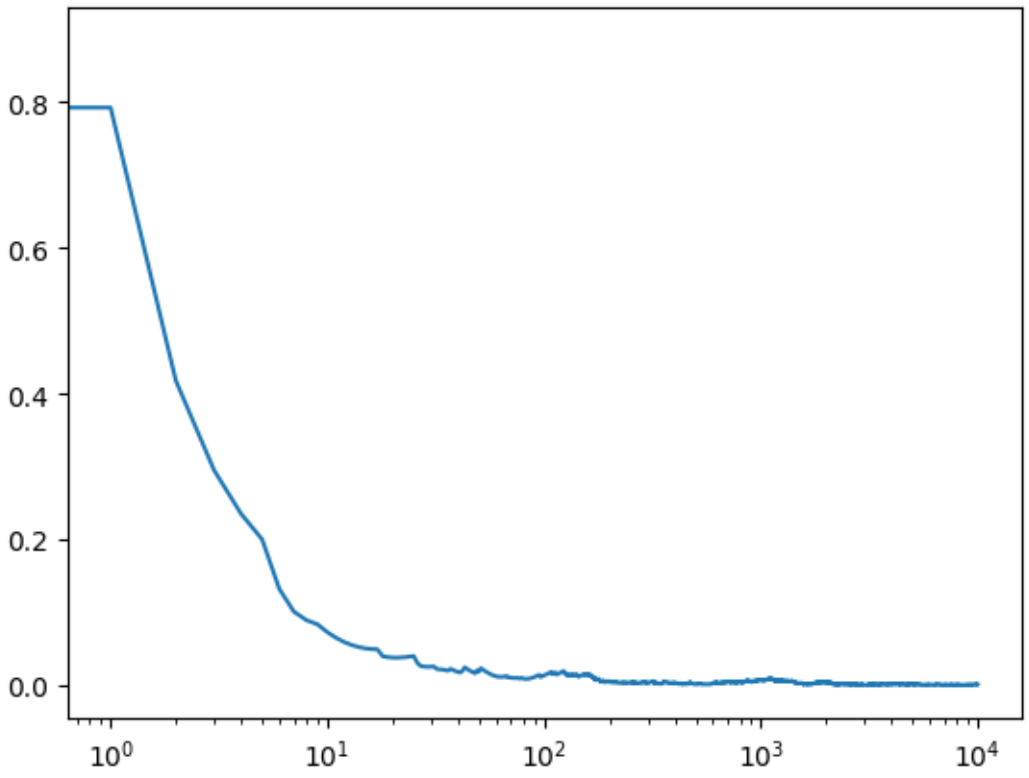


[ ]:
```
#EX 5.2 b
def rotY(theta):
    return np.array([[np.cos(theta/2), -1* np.sin(theta/2)],[np.sin(theta/2),␣
 ↪np.cos(theta/2)]])
```

```
def rotZ(theta):
    return np.array([[np.cos(theta/2) - 1j*np.sin(theta/2), 0],[0, np.cos(theta/
 ↪2) + 1j*np.sin(theta/2)]])

sigmaX = np.array([[0., 1.],[1., 0.]])


theta1 = 2*np.pi/5
theta2 = 2*np.pi*2/7


# import qiskit
import numpy as np
import matplotlib.pyplot as plt

p = 0.2
N = 10000

state_vec = np.array([1., 0.])
state_ground = np.tensordot(state_vec.conj().T, state_vec,0)
state_ground = np.tensordot(np.tensordot(rotY(theta1), state_ground, [1,␣
 ↪0]),rotY(theta1).conj().T, [1, 0])

state_teoretical = (1-p)*state_ground + p*(np.tensordot(np.tensordot(sigmaX,␣
 ↪state_ground,1),sigmaX.conj().T,1))
state_teoretical = np.tensordot(np.tensordot(rotZ(theta2), state_teoretical,␣
 ↪[1, 0]),rotZ(theta2).conj().T, [1, 0])


print(state_teoretical)

state_dens = np.array([[1. ,0.],[0., 0.]], dtype='complex128')

states = []
states.append(np.linalg.norm(state_dens - state_teoretical))

for n in range(N):
    new_state = state_ground
    if (np.random.rand() < p):
        new_state = np.tensordot(np.tensordot(sigmaX, state_ground, [1,␣
 ↪0]),sigmaX.conj().T, [1, 0])
    new_state = np.tensordot(np.tensordot(rotZ(theta2), new_state, [1,␣
 ↪0]),rotZ(theta2).conj().T, [1, 0])
    state_dens += new_state

    states.append(np.linalg.norm((1/(n+1))*state_dens - state_teoretical))
```

```
print(state_dens)

fig = plt.figure()
ax = fig.add_subplot(1, 1, 1)
plt.plot(states)
plt.xscale('log')

assert states[N] < 0.01
print(states[N])
```

```
[[ 0.5927051 +0.00000000e+00j -0.10581499-4.63605772e-01j]
 [-0.10581499+4.63605772e-01j  0.4072949 -2.77555756e-17j]]
[[ 5920.94359225+5.61495295e-14j -1058.14992126-4.63605772e+03j]
 [-1058.14992126+4.63605772e+03j  4080.05640775-2.21406227e-13j]]
0.0009370977975735105
```



[ ]:
```
#Ex 5.2 c
def rotY(theta):
```

```
    return np.array([[np.cos(theta/2), -1* np.sin(theta/2)],[np.sin(theta/2),␣
 ↪np.cos(theta/2)]])

def rotZ(theta):
    return np.array([[np.cos(theta/2) - 1j*np.sin(theta/2), 0],[0, np.cos(theta/
 ↪2) + 1j*np.sin(theta/2)]])

sigmaX = np.kron(np.eye(2), np.array([[0., 1.],[1., 0.]]))
# print(np.eye(2))

theta1 = 2*np.pi/5
theta2 = 2*np.pi*2/7


# import qiskit
import numpy as np
import matplotlib.pyplot as plt

p = 0.2
N = 10000

state_vec = np.kron(np.array([1., 0.]), np.array([1., 0.]))
print(state_vec)
G1 = np.kron(rotY(theta1), np.eye(2))
G2 = np.kron(rotZ(theta2), np.eye(2))

CNOT = np.array([[1,0,0,0],
                 [0,1,0,0],
                 [0,0,0,1],
                 [0,0,1,0]])

print(state_dens)



p = 0.2
N = 10000

state_ground = np.tensordot(state_vec.T, state_vec,0)
state_ground = np.tensordot(np.tensordot(G1, state_ground, 1),G1.conj().T, 1)
state_ground = np.tensordot(np.tensordot(CNOT, state_ground, 1),CNOT.conj().T,␣
 ↪1)

state_teoretical = (1-p)*state_ground + p*(np.tensordot(np.tensordot(sigmaX,␣
 ↪state_ground,1),sigmaX.conj().T,1))
state_teoretical = np.tensordot(np.tensordot(G2, state_teoretical, [1, 0]),G2.
 ↪conj().T, [1, 0])
```

```
print(state_teoretical)

state_dens = np.array(np.tensordot(state_vec.T, state_vec,0),␣
 ↪dtype='complex128')

states = []
states.append(np.linalg.norm(state_dens - state_teoretical))

for n in range(N):
    new_state = np.tensordot(state_vec.T, state_vec,0)
    if (np.random.rand() < p):
        new_state = np.tensordot(np.tensordot(sigmaX, new_state, [1, 0]),sigmaX.
 ↪conj().T, [1, 0])
    new_state = np.tensordot(np.tensordot(G2, new_state, [1, 0]),G2.conj().T,␣
 ↪[1, 0])
    state_dens += new_state

    states.append(np.linalg.norm((1/(n+1))*state_dens - state_teoretical))



print(state_dens)

fig = plt.figure()
ax = fig.add_subplot(1, 1, 1)
plt.plot(states)
plt.xscale('log')

assert states[N] < 0.01
print(states[N])
```

```
[1. 0. 0. 0.]
[[1. 0. 0. 0.]
 [0. 0. 0. 0.]
 [0. 0. 0. 0.]
 [0. 0. 0. 0.]]
[[ 0.5236068 +5.55111512e-17j  0.        +0.00000000e+00j
   0.        +0.00000000e+00j -0.08465199-3.70884618e-01j]
 [ 0.        +0.00000000e+00j  0.1309017 +1.38777878e-17j
  -0.021163  -9.27211544e-02j  0.        +0.00000000e+00j]
 [ 0.        +0.00000000e+00j -0.021163  +9.27211544e-02j
   0.0690983 -6.93889390e-18j  0.        +0.00000000e+00j]
 [-0.08465199+3.70884618e-01j  0.        +0.00000000e+00j
   0.        +0.00000000e+00j  0.2763932 -2.77555756e-17j]]
[[8006.+0.j    0.+0.j    0.+0.j    0.+0.j]
```

```
[   0.+0.j 1995.+0.j    0.+0.j    0.+0.j]
[   0.+0.j    0.+0.j    0.+0.j    0.+0.j]
[   0.+0.j    0.+0.j    0.+0.j    0.+0.j]]
```

```
---------------------------------------------------------------------------
AssertionError                            Traceback (most recent call last)
Cell In[136], line 72
     69 plt.plot(states)
     70 plt.xscale('log')
---> 72 assert states[N] < 0.01
     73 print(states[N])

AssertionError:
```