Code Guide

nadav.carmel1

July 2021

1 About

The simulation of quantum circuit comprising of individual qubits is a subject under study. A tool for simulating quantum circuits with decoherence (energy relaxation and dephasing) is hard to find, especially with precise control over the qubit's decoherence times T_1, related to energy relaxation, and T_2, related to dephasing. This is a documentation on the basic tool written for my thesis research, with a few examples on how to use it. This package includes the basic simulation tool for noisy or ideal circuits, and notebooks that re-create the results of my research. The documentations includes some basic examples of how to use the tool.

2 Installation

2.1 required packages

- Numpy
- QisKit (will not be required in future work)
- QuTiP
- SciPy

2.2 Download

All relevant code is open for use in this GitHub repository or in the following url: `https://github.com/nadavcarmel40/thesis`.

To install and use the package, just install QuTiP (`https://qutip.org/docs/latest/installation.html`) by following the instructions on the above web page.

The basic tool enabling the simulations can be found under the 'simulators' folder in the attached GitHub repository. 'BigStepSimulator' is a state-vector simulator and 'SmallStepSimulator' is a density matrix simulator. The state vector simulator is fast and can simulate quantum circuits without noise, and the density matrix simulator is slower and can simulate noisy circuits.

3 The Simulators

The basic tools created for my project are the simulators. They are constructed with the same user interface, so this documentation will show examples with the density matrix simulator but is relevant for both.

3.1 Creating a Quantum Register

The first and most basic step of every simulation is the creation of the underlying quantum register. This subsection is a walk-through of how to create this register.

Here we will introduce the set of parameters that define the quantum register:

- The register is built from N qubits, that can interact and be entangled with each other.
- The register starts from some state (which is a density matrix - a positive semi-definite hermitian matrix with trace one) in the N-qubit Hilbert-Space, $\rho_0 \in M_{2^N \cdot 2^N}(\mathbb{C})$.
- Each qubit q has the amplitude-damping time T_1^q. If all qubits have the same T_1, we can just pass the parameter $T1$. Else, we set $T1$ to some unimportant constant and pass the Python list of amplitude damping times, ordered as are the qubits, as the parameter $T1s = [T_1^0, ..., T_1^N]$.
- Each qubit q has the pure dephasing time T_2^q. If all qubits have the same T_2, we can just pass the parameter $T2$. Else, we set $T2$ to some unimportant constant and pass the Python list of pure dephasing times, ordered as are the qubits, as the parameter $T2s = [T_2^0, ..., T_2^N]$.
- The simulation works with finite size time steps, each of length dt. Default is $10^{-4}T_1$ assuming all qubits have the same T_1.
- Another important parameter is the time it takes to apply a single gate, $Tgate$. We usually take this time to be '1' in arbitrary units, and work in the units of this time. default is $20 * dt$.
- The last parameter required to define a quantum register is a list of all of the qubit's frequencies. Default is 6[GHZ] for all qubits, and this essentially has no impact on the simulation results. The difference is whether or not to work in the qubit's rotating frame.

So, if we want to create a quantum register, we first need to import the relevant classes:

```
from qutip import *
from simulators.BigStepSimulation import EfficientQuantumRegister
from simulators.SmallStepSimulation import InCoherentQuantumRegister
```

And to create a register comprised of $N = 2$ qubits starting from the state $\rho_0 = |+1\rangle$ with $T_{gate} = 1$, $T_1/T_{gate} = 10^3$ and $T_2/T_{gate} = 10^4$ for both qubits, with $dt = T_{gate}/20$, we can run the following code block:

```
plus = 1/np.sqrt(2)*(fock_dm(2,0) + fock_dm(2,1))
rho0 = tensor([plus,fock_dm(2,1)])
register = InCoherentQuantumRegister(2,rho0,1e3,1e4,Tgate=1,dt=1/20)
```

Or if we want to initialize the register such that qubit 2 has different lifetimes from qubit 1, we can initialize it like so:

```
T1s = [1e3,2e3]
T2s = [1e4,3e4]
qr = InCoherentQuantumRegister(2,rho0,None,None,T1s=T1s,T2s=T2s,\
    Tgate=1,dt=1/20)
```

We can also control all decoherence related parameters of the register via the function 'setError(self, dephase=True, amplitude_damp=True, T1s=None, T2s=None)'. It enables us to switch T_1, T_2 processes on and off, and updates the qubit's lifetimes. For example, if we want to switch amplitude damping off, we can just run:

```
qr.setError(amplitude_damp=False)
```

Some important and useful attributes of the quantum register are:

- self.state - The state of the register. Can be accessed any time.
- self.qI - The identity matrix of the register
- self.dt, self.Tgate - The times defining how big are the time steps of the simulation. Can be changed, for example, before and after the activation of some gates if one wants to make them last a different amount of time, or have better or worse trotterization.
- self.dephase, self.amplitude_damp - Boolian values that can be changed if we want to switch T_1, T_2 processes on and off.
- self.Sx, self.Sy, self.Sz - Lists of the Pauli operators acting on the register's qubits. For example, self.Sx[q] is the Pauli-X operator acting on the qubit 'q'.

3.2 Simulating a Quantum Circuit

Now, we can run a simple quantum circuit on the quantum register we have just created. All quantum circuits are passed to the register as a list of commands, each command is a physical command acting on a physical qubit. The list of commands is built in a logical way and can be understood easily by looking at figure 4, describing a general simulation of gates.

The commands is a list containing all big time steps T, in each one a number of different gates can be acted upon the register. Each of these lists is a list itself, contaning all gates acting at that moment. Each gate can be represented as a tuple object, in the form ('c',q$_1$, q_2, *operator*).

'c' is the command. It can be one of the Strings:
$C \in \{i, X, Y, Z, H, CNOT, CZ, Rx, Ry, Rz, SingleQubitOperator, m\}$
Note that every controlled operator can be constructed as a series of CNOTs and single qubit gates be Neilsen and Chaung.

'q_1' is the qubit the operator is acted upon.

'q_2' is the control qubit in a two-quit gate (Else, None).

'operator' is some extra information needed for the gate. It can be:

- None for defined gates (H,X,Y,Z,CNOT,CZ).
- Angle for Rotations (Rx,Ry,Rz) in Radians.
- Number of gates to wait (int) for the identity gate 'c'='i'.
- A 2x2 Numpy or QuTiP matrix for general single qubit operator.

For example, creating a quantum register of two qubits starting in the $|00\rangle$ state, and creating the bell state 1, can be simulated using the simple code block:

```
T1s = [1e3,2e3]
T2s = [1e4,3e4]
rho0 = tensor([fock_dm(2,0),fock_dm(2,0)]
qr = InCoherentQuantumRegister(2,rho0,None,None,T1s=T1s,T2s=T2s,\
    Tgate=1,dt=1/20)
qr.run([[('H',0,None,None)],[('CNOT',1,0,None)]])
```

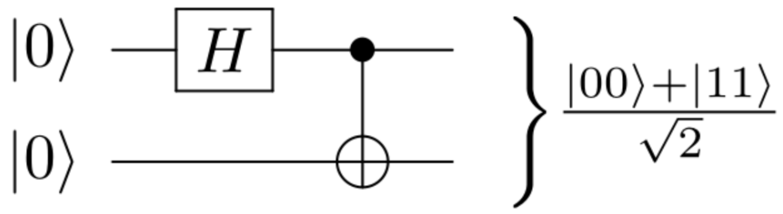


Figure 1: The Bell State

3.3 Visualizing The State History of a Qubit

Here we will show two sainity checks for the simulation, confirming that the decoherence is correct and that all defined gates act as we expect them to. For this purpose we will simulate a noisy register of only one qubit, and we will introduce some new attributes of the quantum register.

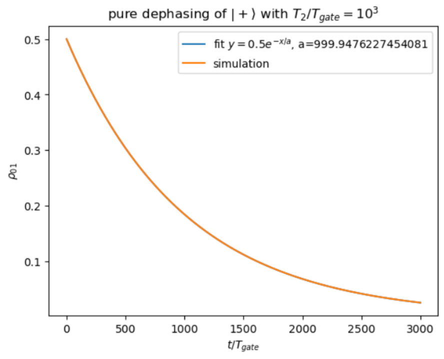
3.3.1 Confirming Decoherence

Here we do T_2 relaxation on the state $|+\rangle\langle+| = \frac{1}{2}\left(\begin{smallmatrix} 1 & 1 \\ 1 & 1 \end{smallmatrix}\right)$, and expect to see that the state evolves as $\rho(t) = \frac{1}{2}\left(\begin{smallmatrix} 1 & e^{-t/T_2} \\ e^{-t/T_2} & 1 \end{smallmatrix}\right)$. we plot (fig.2 (a)) the coherence and see how close is the graph of $\rho_{10}(t)$ to e^{-t/T_2}. The simulation is done be excecuting the code block below. A similar code, setting 'dephase'=False and 'amplitude_damp'=True can simulate energy relaxation on the state $|+\rangle\langle+| = \frac{1}{2}\left(\begin{smallmatrix} 1 & 1 \\ 1 & 1 \end{smallmatrix}\right)$, and we expect to see that the state evolves as $\rho(t) = \frac{1}{2}\left(\begin{smallmatrix} 2-e^{-t/T_1} & e^{-t/2T_1} \\ e^{-t/2T_2} & e^{-t/T_1} \end{smallmatrix}\right)$. we plot (fig.2 (b,c,d)) all elements of the density matrix to see that they are well-behaved. Overall the decoherence simulation is as expected and plotted in figure 2

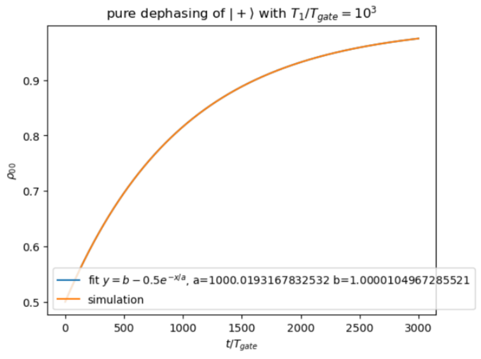
```
 def get_rho_from_bloch(x,y,z):
     """
     (x,y,z) is point in bloch sphere.
     returns density matrix
     """
     return 0.5*qeye(2) + x/2*sigmax() + y/2*sigmay() + z/2 * sigmaz()

import matplotlib.pyplot as plt
from scipy.optimize import curve_fit

plus = 1/np.sqrt(2)*(basis(2,0)+basis(2,1))
T2 = 1000
qubit = InCoherentQuantumRegister(1,plus*plus.dag()\
     , T1=1, T2 = T2, Tgate=1, dt = 1/20)
qubit.setError(dephase = True, amplitude_damp=False)
qubit.setCollectData(data = True, bloch=True)
qubit.run([[('i', 0, None, 3000)]])
xs, ys, zs = qubit.history[0][0], qubit.history[0][1], qubit.history[0][2]
time = qubit.times
print('constructing coherencses')
coherence = []
for i in range(len(xs)):
     rho = get_rho_from_bloch(xs[i], ys[i], zs[i])
     coherence.append(rho[0,1])

def func(x,a):
     return 1/2*np.exp(-x/a)
popt, pcov = curve_fit(func, time, coherence)
print(popt)
plt.plot(np.array(time), func(np.array(time),*popt),\
     label = 'fit $y=0.5e^{-x/a}$, a='+str(popt[0]))
plt.style.use('default')
plt.title('pure dephasing of $|+\\rangle$ with $T_2/T_{gate}=10^3$')
plt.xlabel('$t/T_{gate}$')
plt.ylabel('$\\rho_{01}$')
plt.plot(time, coherence, label='simulation')
plt.legend()
plt.show()
```

So, what's new in that code?

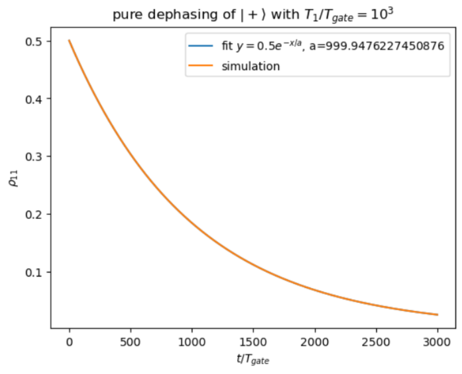
- self.setCollectData() - Gets the two boolian parameters 'data' and 'bloch' (with default values False). 'data' controlls the collection 'self.history' and 'self.purities'. 'bloch' controlles the collection of bloch sphere figures for each qubit, 'self.blochs' and 'self.bloch3ds'.
  - self.history - List of lists. Each list is of the form [xs,ys,zs] such that (xs[i],ys[i],zs[i]) is the position of the qubit's state on bloch sphere in time $i * dt$.
  - self.purities - If the register has N qubits, then this is a list of length $N + 1$. the first N elements are lists such as the i'th list contains the purity as a function of time dt of the i'th qubit, and the last list (in place $N + 1$) is the purity of the entire register as a function of time. Purity is calculated as $Tr(\rho^2)$.
  - self.blochs - A list of QuTiP's bloch spheres, one for each qubit, with the qubit's trajectory.
  - self.bloch3ds - A three dimentional version of the above.
- self.times - a list of dt times the qubit had lived through.



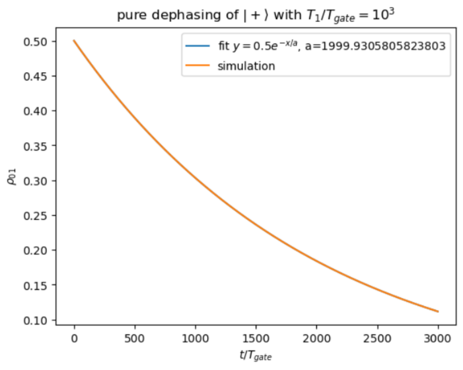
(a) ρ_{01} dephasing



(b) ρ_{00} amplitude damping



(c) ρ_{11} amplitude damping



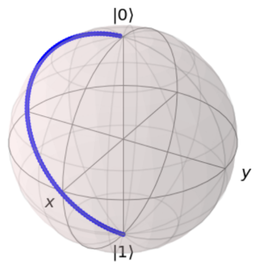
(d) ρ_{01} amplitude damping

Figure 2: Relaxation for three decoherence times. Fit and simulation are the same.

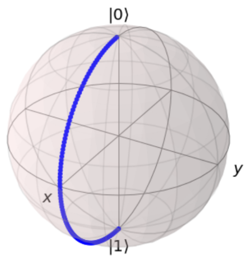
3.3.2 Confirming Defined Gates

Here we show the action of all previously defined gates on one (or two) qubits, on the Bloch Sphere. One can reproduce these figures by running the code blocks similar to the one below:

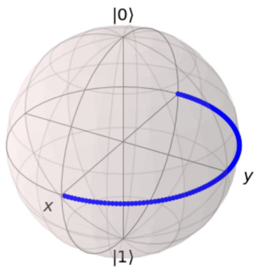
```
qubit.setCollectData(data = True, bloch=False)
qubit.run([[('Rz',0,None,np.pi/4)]])
bloch = Bloch()
bloch.add_points([qubit.history[0][0],\
qubit.history[0][1], qubit.history[0][2]])
bloch.show()
```



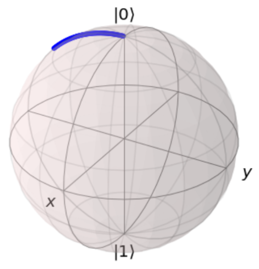
(a) $X|0\rangle = |1\rangle$



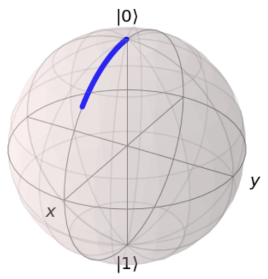
(b) $Y|0\rangle = |1\rangle$



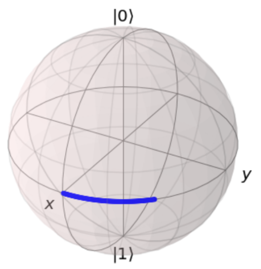
(c) $Z|+\rangle = |-\rangle$



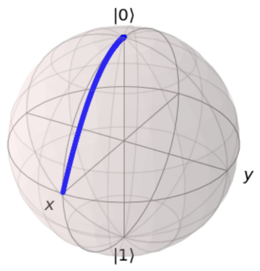
(d) $Rx(\frac{\pi}{4})|0\rangle$



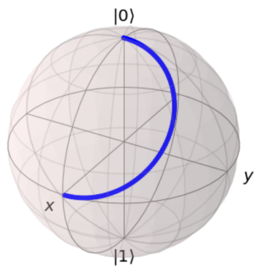
(e) $Ry(\frac{\pi}{4})|0\rangle$



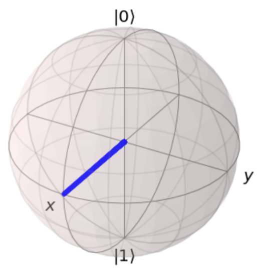
(f) $Rz(\frac{\pi}{4})|0\rangle$



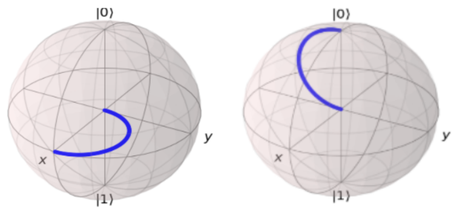
(g) T_1 on $|+\rangle$ for 30 T_{gate} with $T_1/T_{gate} = 1$



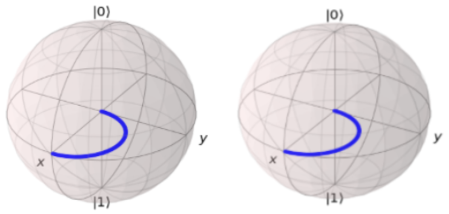
(h) $H|+\rangle = |0\rangle$



(i) T_2 on $|+\rangle$ for 30 T_{gate} with $T_2/T_{gate} = 1$



(j) $CX|+0\rangle = \frac{|00\rangle+|11\rangle}{\sqrt{2}}$



(k) $CZ|++\rangle = \frac{|00\rangle+|01\rangle+|10\rangle-|11\rangle}{2}$

Figure 3: All Defined Gates

4 Theory

In my Work, I use a full density matrix simulation, saving the quantum state of n qubit register as a 2-d matrix of dimensions $[2^n, 2^n]$. Operators are saved as a $[2^n, 2^n]$ matrix, and if the quantum state was initially ρ then performing the operation U on the density matrix is equivalent to updating the density matrix $\rho \to U\rho U^\dagger$.

The noise in the simulation is based on Krauss operators (more of them in section 4.2).Thus, the simulation is made up of many small time steps, with repeated application of Krauss based decoherence in one small time-step and gate-based evolution in the next small time-step. A description of the simulation is given in figure 4 along with algorithm 1. In the algorithm, we take the generator of each unitary gate G_i to be H_i, such that $G_i = e^{i\frac{dt}{T}H_i}$.

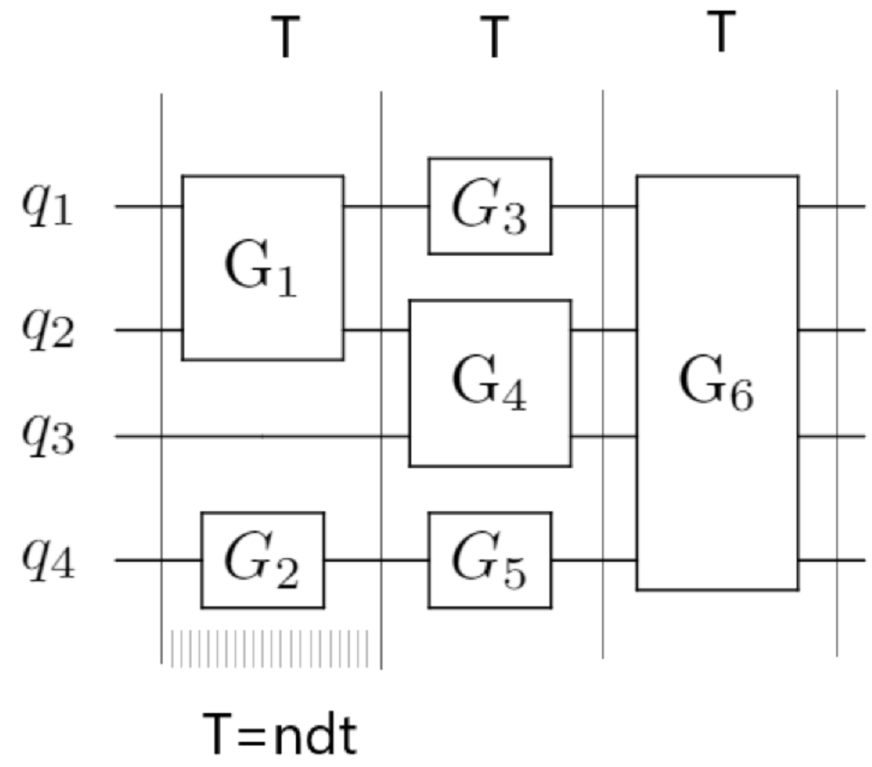


Figure 4: Simulation of a general circuit. Each gate-step T has length T_{gate} and is made up of $n = 20$ small time-steps of size T_{gate}/n.

Algorithm 1: Noisy Circuit Simulation

```
Result: Density matrix of the register after a noisy quantum circuit.
for gate-step $T$ do
    $U^T_{dt} = e^{i\frac{dt}{T}\sum_i H_i}$ ;
    for $t$ do
        $\rho_{t+1} = U^T_{dt}\rho_t(U^T_{dt})^\dagger$ ;
        for qubit do
            amplitude damping ;
            dephasing ;
        end
    end
end
```

4.0.1 Main Parameters

The main parameters used in each simulation are the number of qubits N, the time $T_{gate} = n \cdot dt$ ($n = 20$) of each gate-step, the dephasing time of qubit q, T^q_2 and the energy relaxation time T^q_1 of the same qubit. From these parameters we define the error rates for each process and qubit:

$p^q_{decay} = 1 - e^{-\frac{dt}{T^q_1}}$, and $p^q_{dephase} = 1 - e^{-\frac{dt}{T^q_2}}$

Defining the pauli operator σ^q_i acting on qubit q as a tensor product of σ_i in the q index and Identity operators in all other indexes, we take the base Hamiltonian $H_0 = \bigotimes^N_{q=1} \frac{\hbar\omega_{01}}{2}\sigma^q_z$ to represent the free evolution of the quantum register, with $\omega_{01} = 6[GHZ]$.

4.1 Gate-Based Evolution

In each gate-step, possibly many gates act upon the register. Thus, we start with the base Hamiltonian $H = H_0$ and for each gate G in the gate-step we find it's corresponding Hamiltonain H_G using table 1 and update the Hamiltonian to be $H \rightarrow H + H_G$.

Now, we define the evolution operator U to be $e^{iH\frac{dt}{T_{gate}}}$, and we apply this evolution as in step 4 of algorithm 1 for a total of T_{gate}/dt times with decoherence step between each application of U.

4.2 Krauss-Based Decoherence

In each small time-step dt we do the Krauss-based decoherence on every qubit in the register. There are three options:

- We are given only decay time T^q_1, assuming the pure dephasing time $(T^q_2)^*$ is infinite, for each qubit q.

Gate	Hamiltonian
P^q	$-\frac{\pi}{2}\sigma_P^q$
$R_P^q(\theta)$	$-\frac{\theta}{2}\sigma_P^q$
H^q	$\frac{\pi}{2}\cdot\frac{\sigma_X^q+\sigma_Z^q}{\sqrt{2}}$
$CNOT(q_1,q_2)$ - q_2 is control	$\ln\left(\frac{1}{2}\cdot((I-\sigma_Z^{q_2})\sigma_X^{q_1}+I+\sigma_Z^{q_2})\right)$
$CZ(q_1,q_2)$ - q_2 is control	$\ln\left(\frac{1}{2}\cdot((I-\sigma_Z^{q_2})\sigma_Z^{q_1}+I+\sigma_Z^{q_2})\right)$
$\left(\begin{smallmatrix} a & b \\ c & d \end{smallmatrix}\right)$ - single qubit (q) gate	$\ln\left(\frac{a+d}{2}I+\frac{a-d}{2}\sigma_Z^q+\frac{c+b}{2}\sigma_X^q+\frac{c-b}{2i}\sigma_Y^q\right)$

Table 1: Gate and Gate Hamiltonian table for the simulation, with $P \in \{X,Y,Z\}$

- We are given only pure dephasing time T_2^q, assuming the decay time T_1^q is infinite, for each qubit q.
- We are given both the decay time T_1 and the pure dephasing time T_2 for each qubit q.

Using the relation between the decay time T_1, the dephasing time T_2 and the pure dephasing time T_2^*, $T_2 = (\frac{1}{2T_1}+\frac{1}{T_2^*})^{-1}$, we can see that if the pure dephasing time is assumed to be infinite then we have $T_2 = 2T_1$. This is embedded in the amplitude damping process, with it's Kraus operators.

So, for the second option (given T_2^q (pure dephasing time) assuming T_1^q is infinite) we iterate over the qubit q and update the register state to be:

$$\rho \to (1-\frac{P_{dephase}^q}{2})\rho + \frac{P_{dephase}^q}{2}\sigma_Z^q\rho\sigma_Z^q$$

This is true because if we take the Krauss operators of the phase damping (=phase flip) channel, $K_1 = \sqrt{p}\left(\begin{smallmatrix} 1 & 0 \\ 0 & 0 \end{smallmatrix}\right)$, $K_2 = \sqrt{p}\left(\begin{smallmatrix} 0 & 0 \\ 0 & 1 \end{smallmatrix}\right)$, $K_3 = \sqrt{1-p}\left(\begin{smallmatrix} 1 & 0 \\ 0 & 1 \end{smallmatrix}\right)$, We get $\sum_i K_i\rho K_i^\dagger = (1-\frac{p}{2})\rho + \frac{p}{2}\sigma_Z\rho\sigma_Z$.

For the first option (given T_1^q assuming $(T_2^q)^*$ is infinite) we define the Krauss operators of the amplitude damping channel $M_1 = \left(\begin{smallmatrix} 1 & 0 \\ 0 & \sqrt{1-p} \end{smallmatrix}\right)$, $M_2 = \left(\begin{smallmatrix} 0 & \sqrt{p} \\ 0 & 0 \end{smallmatrix}\right)$ Which, expressed by pauli operators, look like

$M_1^q = \frac{\sqrt{1-P_{decay}^q}}{2}(I-\sigma_Z^q)+\frac{1}{2}(I+\sigma_Z^q)$,

$M_2^q = \frac{\sqrt{P_{decay}^q}}{2}(\sigma_X^q + i\sigma_Y^q)$

And we iterate over the qubit q, updating the register to be:

$$\rho = M_1^q\rho(M_1^q)^\dagger + M_2^q\rho(M_2^q)^\dagger$$

Finally, for the third option - given both pure dephasing rate and energy relaxation rate, we update the state of the register as follows:

$$\rho_{temp} = M_1^q\rho(M_1^q)^\dagger + M_2^q\rho(M_2^q)^\dagger$$

$$\rho = (1-\frac{P_{dephase}^q}{2})\rho_{temp} + \frac{P_{dephase}^q}{2}\sigma_Z^q\rho_{temp}\sigma_Z^q$$

4.3 Measurements

There are two kinds of measurements I perform - Post Selection measurements and probabilistic measurements. Here, I first Refer to the post selection measurements.

Post selection measurements are done on the sensor qubit for SPS, on flag qubits for fault-tolerance, and once again on the sensor qubit (that acts as an ancilla qubit for syndrome measurement) for LPS. To perform post selection measurements, I collapse the register state as defined below according to my preferred measurement outcomes.

First, to add flag qubits to my simulation, I expand the register state with a tensor product to the two additional qubit sub-spaces. Next, I perform the entangling operations with the flags, and finally I project on the trivial flag state $|0..0\rangle$ using the operator defined below.

For probabilistic measurements (e.g. Error correction measurements), to decide measurement outcome on the qubit group $A = \{q_{k_1}, ..., q_{k_n}\}$, remaning with $B = \{q_1, ..., q_N\}/A$, I trace out B to get $\rho^A = Tr_B(\rho)$. Then, I define P as the diagonal of ρ^A and P' as the comulative sum of P. I take a random number $0 < x < 1$ and find the first index i such that $x < P'[i]$. The result of the measurement is the binary string of $i - 1$.

To collapse the quantum state to a state after measuring qubits in the group A, I use the following projector:

$$P_m = \bigotimes_{q=1}^{N} \begin{cases} I_2 & \text{if } q \notin A \\ \frac{I+\sigma_Z^q}{2} & \text{if } q \in A \text{ and measurement result is } |0\rangle \\ \frac{I-\sigma_Z^q}{2} & \text{if } q \in A \text{ and measurement result is } |1\rangle \end{cases}$$

And after this projection operation $\rho \to P_m \rho P_m^\dagger$ I trace out the flag qubits.

The procedure described above can cause numeric errors when the state decohere for a long time, becuase the projection operation as I described it is not trace preserving. To have a valid density matrix, for each projection, say the k'th projection, I first save the state's trace as Ps_k and then normalize the state. To calculate the portion of information i have lost due to post selection, I use the following reasoning:

- after one projection, I have lost $1-Ps_1$ information and remain with a state with trace Ps_1.
- after the second projection, I have lost $Ps_1 \cdot (1 - Ps_2)$ more information.
- after the third projection, I have lost $Ps_1 \cdot Ps_2 \cdot (1 - Ps_3)$ more information.
- after the k'th projection, I have lost $Ps_1 \cdot ... \cdot Ps_{k-1} \cdot (1 - Ps_k)$ more information.

Overall, this is the amount of *lost information*:

$$lostinformation = \sum_{i}((\prod_{k}^{i-1} Ps_k) \cdot (1 - Ps_i))$$