A Tutorial on Quantum Dynamics Simulations on Quantum Computers. Part II: Open Systems

Yuchen Wang,[†] Xiaohan Dan,[‡] Delmar G. A. Cabral,[‡] Saurabh Shivpuje,[†] Zixuan Hu,[†] Ningyi Lyu,[‡] Eitan Geva,[¶] Victor S. Batista,[‡] and Sabre Kais[*,†]

†*Department of Chemistry, and Purdue Quantum Science and Engineering Institute, Purdue University, West Lafayette, Indiana 47907, USA*

‡*Department of Chemistry, Yale Quantum Institute, Yale University, New Haven, CT 06511, USA*

¶*Department of Chemistry, University of Michigan, Ann Arbor, MI 48109, USA*

E-mail:

Contents

Abstract

Simulating the open quantum system dynamics is indispensable to a wide range of studies of physical and chemical systems. Its applications include understanding the energy and charge transport within the electronic and nuclear processes, investigating the mechanism in the field of photochemistry, condensed-phase physics, nanoscience, molecular electronics, quantum optics, spectroscopy and quantum information processing and more. Quantum computer, with the ability to do parallel processing and the fact that their computational space grows exponentially to the number of the qubits being used, acts as a good candidate for performing such simulations of open systems. This article is Part II of a series of tutorials about quantum dynamics simulations on quantum computers. In this part we explore open quantum systems, employing the widely-used Lindblad master equation within the framework of the Markovian approximation. With the Kraus representation of the Lindblad equation, we propose a quantum algorithm utilizing the Sz.-Nagy dilation theorem and parallelization techniques for the linear expansions of non-unitary propagators. We provide the coding and simulation details of our algorithm and apply it to simulate a 2-level atom modeled by amplitude-channel damping. We also include a quantum circuit generator that makes use of the Group Leader Optimization Algorithm. It serves as an alternative and useful tool in the quantum circuit compiling process of the proposed quantum algorithm. This article serves as a beginner's guide to simulating open quantum system dynamics with quantum computers.

1 Lindblad master equation for the open system quantum dynamics

In Part II of the quantum dynamics tutorial paper, we expand the discussion on quantum dynamics to open systems. The open system refers to a system that interacts with its surrounding environment. In contrast to the deterministic evolution of the state vector in a closed quantum system, the presence of the environment introduces stochastic processes, resulting in stochastic evolution. The state of an open quantum system is therefore described in terms of ensemble-averaged states using the density matrix formalism. The system density matrix ρ (also called "reduced density matrix" in the sense that it only contains the information of the system) describes a probability distribution of quantum states $|\psi_n\rangle$, in a matrix representation $\rho = \sum_n p_n |\psi_n\rangle\langle\psi_n|$, where p_n is the classical probability that the system is in the quantum state $|\psi_n\rangle$. The time evolution of the system density matrix ρ is the topic of the remaining portions of this paper.

The standard approach for deriving the equations of motion for a system interacting with its environment is to expand the scope of the system to include the environment. The combined quantum system is then closed, and its evolution is governed by the Liouville (Liouville–von Neumann) equation[1]

$$\dot{\rho}_{tot}(t) = -\frac{i}{\hbar}[H_{tot}, \rho_{tot}(t)], \tag{1}$$

which is the equivalent of the Schrödinger equation in the density matrix formalism. Here, ρ_{tot} is the density matrix of the combined quantum system, and the total Hamiltonian

$$H_{tot} = H_{sys} + H_{env} + H_{int} \tag{2}$$

includes the original system Hamiltonian H_{sys}, the Hamiltonian for the environment H_{ev}, and a term representing the interaction between the system and its environment H_{int}.

Since we are only interested in the dynamics of the system, we can at this point perform a partial trace over the environmental degrees of freedom in Eq. (1), and thereby obtain a master equation for the evolution of the reduced system density matrix $\rho = \mathrm{Tr}_{env}[\rho_{tot}]$. The most general trace-preserving and completely positive form of this evolution is the Lindblad master equation[1,2] for the reduced density matrix

$$\dot{\rho}(t) = -\frac{i}{\hbar}[H, \rho(t)] + \sum_n \frac{1}{2}\gamma_n[2L_n\rho(t)L_n^\dagger - \rho(t)L_n^\dagger L_n - L_n^\dagger L_n\rho(t)] \tag{3}$$

where H is the system Hamiltonian H_{sys} in Eq. (2), L_n are jump operators describing the dissipative part of the dynamics, and γ_n are the corresponding damping rates characterizing the rates of dissipation induced by the environment. L_n and γ_n are determined by the environment H_{env} and the system-environment coupling H_{int}. The derivation of Eq. (3) may be found in several sources[1,3] and will not be reproduced here. Instead, we emphasize the approximations that are required to arrive at the master equation in the form of Eq. (3) from physical arguments, and hence perform a calculation in QuTiP:[4,5]

Separability: At $t = 0$ there are no correlations between the system and its environment such that the total density matrix can be written as a tensor product $\rho_{tot}(0) = \rho(0) \otimes \rho_{env}(0)$.

Born approximation: (1) The state of the environment does not significantly change as a result of the interaction with the system. (2) The system and the environment remain separable throughout the evolution. Hence, $\rho_{tot}(t) \approx \rho(t) \otimes \rho_{env}(t)$. These assumptions are justified when the interaction is weak and when the environment is significantly larger than the system's size.

Markov approximation: The time-scale of decay for the environment τ_{env} is much shorter than the smallest time-scale of the system dynamics $\tau_{sys} >> \tau_{env}$. This approximation is often deemed a "short-memory environment" as it requires rapid decay of environmental correlation functions compared to the system's time scale.

Secular approximation: Stipulates all fast-rotating terms in the master equation can be neglected. It also ignores terms that lead to a small renormalization of the system energy levels. This approximation is not strictly necessary for all master-equation formalisms (e.g., the Block-Redfield master equation), but it is required for arriving at the Lindblad form Eq. (3).

For systems with environments satisfying the conditions outlined above, the Lindblad master equation Eq. (3) governs the time evolution of the system density matrix, giving an ensemble average of the system dynamics. To ensure that these approximations are not violated, it is important that the damping rates γ_n be smaller than the minimum energy splitting in the system Hamiltonian. Situations that demand treatments beyond the above approximations therefore include, for example, systems strongly coupled to their environment and systems with degenerate or nearly degenerate energy levels.[1,6,7]

2 Simulations of Lindblad Master Equation on Digital Computers

In this section, we demonstrate the simulation of the Lindblad master equation [i.e. Eq. (3)] on a digital computer. We introduce two methods to simulate the Lindblad master equation: matrix exponential propagation and QuTiP[4,5] method. We will simulate the dynamics of a spin-1/2 system and then proceed to simulate the dynamics of a larger spin-chain system.

2.1 Matrix Exponential Propagation

We demonstrate how to perform the Lindblad equation by solving a matrix-vector multiplication problem. To do so, we start by vectorizing the density matrix,

$$\rho \to \mathbf{v}_\rho = [\rho_{11}, ..., \rho_{1N}, \rho_{21}, ..., \rho_{2N}, ..., \rho_{N1}, ..., \rho_{NN}]^T \tag{4}$$

Where N is the dimension of the system Hilbert space (i.e. dimension of Hamiltonian H).

Then, we recast the Lindblad equation in Eq. (3) into the equivalent matrix-vector form

$$\frac{d}{dt}\mathbf{v}_\rho(t) = [\mathbf{A}_C + \mathbf{A}_D]\mathbf{v}_\rho(t) \tag{5}$$

with the matrix form of the commutator $\mathbf{A}_C$ and the matrix form of Lindbladian dissipator $\mathbf{A}_D$ as

$$\begin{aligned}\mathbf{A}_C &= -i(H \otimes \mathbb{I} - \mathbb{I} \otimes H^T) \\ \mathbf{A}_D &= \sum_n \frac{1}{2}\gamma_n \left[2L_n \otimes L_n^* - \mathbb{I} \otimes L_n^T L_n^* - L_n^\dagger L_n \otimes \mathbb{I}\right]\end{aligned} \tag{6}$$

where $\mathbb{I}$ is the identity matrix in the Hilbert space of H. By integrating Eq. 18, the density matrix at time t can be expressed according to the action of the exponential of the Lindbladian dissipator on the vectorized density matrix at $t = 0$,

$$\mathbf{v}_\rho(t) = \mathbf{G}(t)\mathbf{v}_\rho(0) = e^{[\mathbf{A}_C + \mathbf{A}_D]t}\mathbf{v}_\rho(0) \tag{7}$$

where $\mathbf{G}(t)$ is called the propagator.

2.2 QuTiP method

We can also compute the Lindblad dynamics using QuTiP's `mesolve` function. When using `qutip.mesolve`, the following things needs to be provided:

1. The system Hamiltonian H
2. Initial density matrix $\rho(0)$
3. Time list for propagation
4. Collapse operators defined by $\sqrt{\gamma_n}L_n$. if no collapse operators are given, it will propagate the Liouville equation of the system.
5. The expectation values are to be calculated.

With the above quantities defined, `qutip.mesolve` can generate the time-dependent expectation values with respect to Liouville equation (if no collapse operators are given) or Lindblad master equation (if collapse operators were given).

2.3 Spin-1/2 System

The spin-1/2 system Hamiltonian is

$$H = \Delta\sigma^x \tag{8}$$

with tunneling rate $\Delta = 0.1 \times 2\pi$, and σ^x denotes the Pauli-X matrix. For the jump operators in Eq. (3), we chose the single jump operator as $L = \sigma_x$ and the damping rate $\gamma = 0.05$.

We define the spin-up state $|\uparrow\rangle$ and spin-down state $|\downarrow\rangle$ as

$$|\uparrow\rangle = |0\rangle = \begin{bmatrix} 1 \\ 0 \end{bmatrix}, \qquad |\downarrow\rangle = |1\rangle = \begin{bmatrix} 0 \\ 1 \end{bmatrix}; \tag{9}$$

The initial state and the density matrix are set at the spin-up state

$$|\psi(0)\rangle = |0\rangle, \quad \rho(0) = |0\rangle\langle 0| \ . \tag{10}$$

Script 2.1: Importing Libraries

```python
!pip install qiskit
!pip install qiskit_aer
!pip install qutip
import numpy as np
import scipy.linalg as sp
import matplotlib.pyplot as plt
import qiskit
from qiskit import *
from qiskit_aer import AerSimulator
from qiskit.quantum_info.operators import Operator
```

Script 2.2: Spin-1/2 Parameters

```python
#Pauli matries
sigmax = np.array([[0, 1], [1, 0]],dtype=np.complex_)
sigmaz = np.array([[1, 0], [0, -1]],dtype=np.complex_)
sigmay = np.array([[0, -1j], [1j, 0]],dtype=np.complex_)
ident  = np.eye(2,dtype=np.complex_)

#The Spin-1/2 system Hamiltonian
H_1spin = 2*np.pi * 0.1 * sigmax

#The jump operator and damping rate of Spin-1/2 system
gamma_1spin = 0.05
L_1spin = sigmax.copy()

#spin_up and down state
spin_up = np.array([1.0,0.0],dtype=np.float_)
spin_down = np.array([0.0,1.0],dtype=np.float_)

#initial density matrix
rho0_1spin = np.outer(spin_up,spin_up.conj())

#time step
dt = 0.1
nsteps = 250
```

With the parameters and Hamiltonian defined in the above code, we can simulate the dynamics. The following code using matrix exponential propagation to propagate the dynamics.

Script 2.3: Spin-1/2 Parameters

```python
Nsystem = H_1spin.shape[0] #dimension of Hilbert space
vec_rho0_1spin = rho0_1spin.reshape(Nsystem**2)

#derivation matrix
ident_h = np.eye(Nsystem, dtype = np.complex_)
Amat = -1j*(np.kron(H_1spin,ident_h)-np.kron(ident_h,H_1spin.T))

# the lindblad term
Amat += (2.0*(np.kron(L_1spin,L_1spin.conj())) - np.kron(ident_h,
↪   L_1spin.T@L_1spin.conj()) -
↪   np.kron(L_1spin.T.conj()@L_1spin,ident_h))*0.5*gamma_1spin

result_1spin_matprop = []
for i in range(nsteps):
  Gt = sp.expm(Amat*dt*i)
  vec_rhot_1spin = Gt@vec_rho0_1spin
  rhot_1spin = vec_rhot_1spin.reshape(Nsystem,Nsystem)
  #the expection value of sigmaz
  result_1spin_matprop.append(np.trace(rhot_1spin@sigmaz))
```

The following code using QuTiP to propagate the dynamics, we also propagate the Liouville equation for comparison.

Script 2.4: Spin-1/2 Parameters

```python
from qutip import mesolve, Qobj

times = np.linspace(0,(nsteps-1)*dt,nsteps)

#not give collapse operator: Liouville equation
result_qutip_Liouv = mesolve(Qobj(H_1spin), Qobj(rho0_1spin), times, e_ops =
↪   Qobj(sigmaz))
#QuTiP for Lindblad equation
result_qutip = mesolve(Qobj(H_1spin), Qobj(rho0_1spin), times, c_ops = np.sqrt(0.05)
↪   * Qobj(sigmax), e_ops = Qobj(sigmaz))


fig, ax = plt.subplots()
ax.plot(times, result_1spin_matprop, label = "Matrix Exponential Propagation")
ax.plot(times, result_qutip.expect[0],'--',label = "QuTiP_Lindblad")
ax.plot(times, result_qutip_Liouv.expect[0],'*',label = "QuTiP_Liouville")
ax.set_xlabel('Time')
ax.set_ylabel('Sigma-Z')
ax.legend()
plt.show()
```

The result shows in Fig. 1. The two methods shows the same result, and compare to the continuous oscillation of the Liouville results, the Lindblad equation introduce dissipation and the spin will relax...

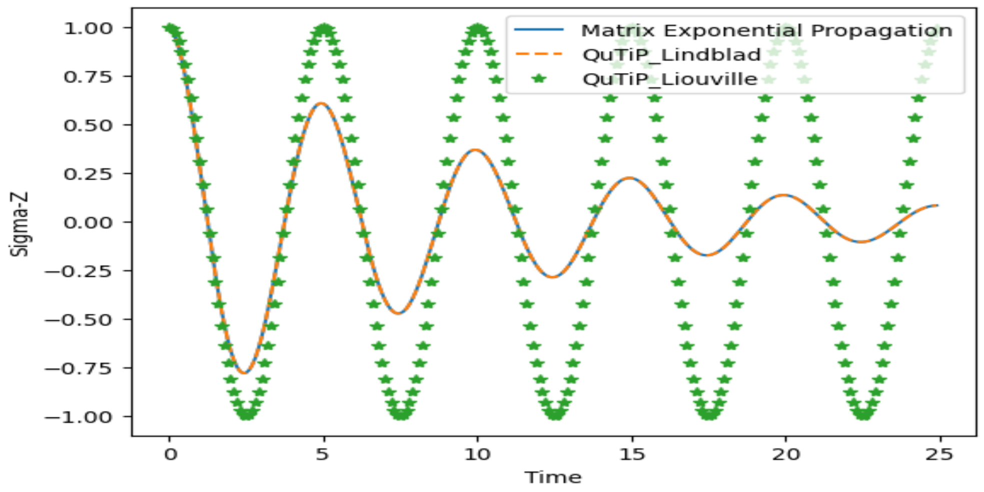

Figure 1: Lindblad dynamics of the spin-1/2 system coupled to dissipator. Two methods are used. Compared to the Liouville equation without bath.

2.4 Spin Chain

As another example, we choose the Heisenberg spin-chain model, which is widely used to study properties of radical and magnetic materials.[8] DC has references for this; adding soon. The Hamiltonian for the spin-chain is defined as follows:[9]

$$H = \sum_{n=0}^{N-1} \Omega_n \sigma_n^z - \frac{1}{2} \sum_{n=0}^{N-2} \left(J_{n,n+1}^x \hat{\sigma}_n^x \hat{\sigma}_{n+1}^x + J_{n,n+1}^y \hat{\sigma}_n^y \hat{\sigma}_{n+1}^y + J_{n,n+1}^z \hat{\sigma}_n^z \hat{\sigma}_{n+1}^z \right) \tag{11}$$

where N denotes the number of spins in the model, σ_n^i denotes a Pauli matrix acting on the n-th ($n \in \{0, ..., N-1\}$) spin site with $i \in \{x, y, z\}$. Ω_n is the local potential in the n-th spin site, $J_{n,n+1}^i$ with $i \in \{x, y, z\}$ denotes the coupling between n-th and $n+1$-th site.

For illustrative purposes we consider the parameters as used in the Ref.[9] but with three spin sites (N=3) rather than twenty.

Table 1: Hamiltonian parameters used in the spin chain simulation, from Fiori *et al*[9]

Parameter	$n = 0$	$n \neq 0$
Ω_n	0.65	1.0
$J^x_{n,n+1}$	0.75	1.0
$J^y_{n,n+1}$	0.75	1.0
$J^z_{n,n+1}$	0.0	0.0

We use the following code to setup the necessary operators and parameters for the open quantum dynamics simulation of the spin chain Hamiltonian using a digital computer.

Script 2.5: Spin Chain Hamiltonian parameter

```python
from qutip import mesolve, Qobj
import numpy as np

#the system Hamiltonian parameter
nsite = 3 #this states how many spins in the simulation
ndvr = 2**nsite #this is the dimension of the Hilbert space
Omegai_list = [0.65, 1.0, 1.0]
Jix_list = [0.75, 1.0]
Jiy_list = [0.75, 1.0]
Jiz_list = [0.0, 0.0]

spin_up = np.array([1.0,0.0],dtype=np.float_)
spin_down = np.array([0.0,1.0],dtype=np.float_)

#Pauli matries
sigmax = np.array([[0, 1], [1, 0]],dtype=np.complex_)
sigmaz = np.array([[1, 0], [0, -1]],dtype=np.complex_)
sigmay = np.array([[0, -1j], [1j, 0]],dtype=np.complex_)
ident  = np.eye(2,dtype=np.complex_)
```

The initial state and the density matrix are defined as

$$|\psi(0)\rangle = |\uparrow\downarrow\downarrow\rangle, \quad \rho(0) = |\psi(0)\rangle\langle\psi(0)| \ . \tag{12}$$

Likewise, we set up the initial state wavefunction by an outer product of the spin-up (first site) and spin-down vectors, and then the initial density matrix by an outer product of the initial wavefunction vector (since it is a pure state).

Script 2.6: Spin Chain Hamiltonian parameter

```python
#set up the initial state at [up,down,down...]
init_state = spin_up.copy()
for i in range(nsite-1):
  init_state = np.kron(init_state,spin_down)

#ste up the initial density matrix according to initial state
rho0 = np.zeros((ndvr,ndvr),dtype=np.complex_)
rho0 += np.outer(init_state,init_state.conj())
```

With the parameters above, we can define the spin chain Hamiltonian through the outer product of the Pauli matrices constituting the Hamiltonian operators.

Script 2.7: Spin Chain Hamiltonian

```python
#the diagnoal part of the Hamiltonian
H_diag = np.zeros((ndvr,ndvr),dtype=np.complex_)
for n in range(nsite):
    tmp = 1.0
    for i in range(nsite):
      if(i==n):
        tmp = np.kron(tmp,sigmaz)
      else:
        tmp = np.kron(tmp,ident)
    H_diag += Omegai_list[n]*tmp
#the non-diagnoal (coupling) part of the Hamiltonian
H_coup = np.zeros((ndvr,ndvr),dtype=np.complex_)
XX = np.kron(sigmax,sigmax)
YY = np.kron(sigmay,sigmay)
ZZ = np.kron(sigmaz,sigmaz)
for n in range(nsite-1):
  coup_tmp = Jix_list[n]*XX+Jiy_list[n]*YY+Jiz_list[n]*ZZ
  tmp = 1.0
  for i in range(nsite-1):
    if(n==i):
      tmp = np.kron(tmp,coup_tmp)
    else:
      tmp = np.kron(tmp,ident.copy())
  H_coup += tmp
hsys = H_diag - 0.5 * H_coup
hsys_qobj = Qobj(hsys)
```

(Maybe remove the Lindblad equation since we have defined it in Eq 3) The Lindblad equation for the spin-chain was chosen to model the effect of dissipation of an analogous spin system on a quantum device[10]

$$\dot{\rho}(t) = -i[H, \rho(t)] + \sum_{m=1}^{2} \sum_{n=0}^{N-1} \frac{1}{2} \gamma_{m,n} \left(2L_{m,n}\rho(t)L_{m,n}^{\dagger} - \rho(t)L_{m,n}^{\dagger}L_{m,n} - L_{m,n}^{\dagger}L_{m,n}\rho(t)\right) \tag{13}$$

Here m denotes two different noise channels, the amplitude damping noise ($m = 1$) and the dephasing noise ($m = 2$), acting on the n-th spin site. The collapse operators $L_{m,n}$ for the Lindblad equation is chosen as[10]

$$L_{1,n} = \hat{\sigma}_n^- \tag{14}$$

$$L_{2,n} = \hat{\sigma}_n^+ \hat{\sigma}_n^- \quad . \tag{15}$$

with $\hat{\sigma}_n^{\pm} \equiv (\hat{\sigma}_n^x \pm \hat{\sigma}_n^y)/2$. The damping rate $\gamma_{1,n}$ and $\gamma_{2,n}$ can be determined experimentally by measuring the spin relaxation process.[10] Here, we choose $\gamma_{1,n}$ and $\gamma_{2,n}$ corresponds to the average noise determined by Dang and coworkers in Ref.[10] Where they determined the average noise for the second processor is $T_{1,n} = 24.9 * 5$ and $T_{2,n} = 15.3 * 5$ (in units of $1/J$), where J denotes nearest-neighbor coupling. Using J has a scale of 1.0 in our Table 1, and relate $T_{1,n}$ and $T_{2,n}$ to dampling rate through $\gamma_{1,n} = 2/T_{1,n}$, $\gamma_{2,n} = 4/T_{2,n}$, then we have $\gamma_{1,n} = 0.016$ and $\gamma_{2,n} = 0.0523$.

With this Lindbladian operator in mind, we implement it in a manner analogous to the Hamiltonian above. For convenience, we store two copies of the Lindbladian operator, one for the digital computer simulation (`Lindbladian_qobj`) and another for the qubit-device simulation (`Lindbladian`).

Script 2.8: Spin Chain Lindbladian Operators

```python
#The lindblad damping rate
Gamma1 = [2/24.9]*nsite
Gamma2 = [4/15.3]*nsite

Lindbladian = []
Lindbladian_qobj = []

sigmap = (sigmax+1j*sigmay)*0.5
sigmam = (sigmax-1j*sigmay)*0.5
sigma2 = sigmap@sigmam

for isite in range(nsite):
  #Lindbladian for type 1
  res = 1.0
  for j in range(nsite):
    if(j==isite):
      res = np.kron(res,sigmam)*np.sqrt(Gamma1[isite])
    else:
      res = np.kron(res,ident)
  Lindbladian.append(res)
  Lindbladian_qobj.append(Qobj(res))

  #Lindbladian for type 2
  res = 1.0
  for j in range(nsite):
    if(j==isite):
      res = np.kron(res,sigma2)*np.sqrt(Gamma2[isite])
    else:
      res = np.kron(res,ident)
  Lindbladian.append(res)
  Lindbladian_qobj.append(Qobj(res))
```

The observable of interest for our example is the time-evolved survival amplitude, defined as

$$P_s(t) = \sqrt{\mathrm{Tr}[\rho(t)\rho(0)]}\ . \tag{16}$$

When starting from a pure state $|\psi(0)\rangle$, this expression is equal to $|\langle\psi(0)|\psi(t)\rangle|$.

Using Matrix exponential propagation, the Lindblad dynamics for the spin chain can be simulated as follows

Script 2.9: Spin Chain dynamics using Matrix exponential propagation

```python
vec_rho0 = rho0.reshape(Nsystem**2)

#derivation matrix
ident_h = np.eye(Nsystem, dtype = np.complex_)
Amat = -1j*(np.kron(hsys,ident_h)-np.kron(ident_h,hsys.T))

# the lindblad term
for i in range(len(Lindbladian)):
  Amat += (2.0*(np.kron(Lindbladian[i],Lindbladian[i].conj())) - np.kron(ident_h,
  ↪  Lindbladian[i].T@Lindbladian[i].conj()) -
  ↪  np.kron(Lindbladian[i].T.conj()@Lindbladian[i],ident_h))*0.5

# the propagator
Gprop = []
Prob_exp = np.zeros(nsteps,dtype=np.float_)
for i in range(nsteps):
  Gt = sp.expm(Amat*dt*i)
  Gprop.append(Gt)
  vec_rhot = Gt@vec_rho0
  survi_prob = np.sqrt((np.dot(vec_rho0.conj(),vec_rhot)).real)
  Prob_exp[i] = survi_prob
```

Using QuTiP, the Lindblad dynamics for the spin chain can be simulated as follows,

Script 2.10: Spin Chain dynamics using QuTiP

```python
from qutip import mesolve, Qobj

#time step
dt = 0.1
nsteps = 250

time = np.linspace(0,dt*(nsteps-1),nsteps)

rho0_qobj = Qobj(rho0)
#using QuTip to simulate the spin-chain Lindblad equation
result = mesolve(hsys_qobj, rho0_qobj, time, c_ops=Lindbladian_qobj, e_ops=rho0_qobj)
#compare to spin-chain pure system dynamics without Lindbladian
result_liouv = mesolve(hsys_qobj, rho0_qobj, time, e_ops=rho0_qobj)

Prob = np.sqrt(result.expect[0][:])
Prob_liouv = np.sqrt(result_liouv.expect[0][:])

plt.plot(time,Prob_exp, label="Matrix Exponential Prop")
plt.plot(time,Prob,'--',label="QuTiP_Lindblad")
plt.plot(time,Prob_liouv,'*',label="Pure system evolution")
plt.legend()
```

The results shown in Fig. 2 showcase the dynamics of the pure system without dissipation, with the system oscillating between different spin configurations and recovering the initial state after some time. However, the inclusion of dissipation leads to coherence loss and the initial state is not recovered.

2.5 Quantum Simulation of Spin Chain Dynamics

We demonstrate how to perform the qubit-based simulation of the open quantum dynamics of the spin chain Hamiltonian by solving a matrix-vector multiplication problem. To do so, we start by vectorizing the density matrix,

$$\rho \rightarrow \mathbf{v}_\rho = [\rho_{11}, ..., \rho_{1n}, \rho_{21}, ..., \rho_{2n}, ..., \rho_{n1}, ..., \rho_{nn}]^T \tag{17}$$

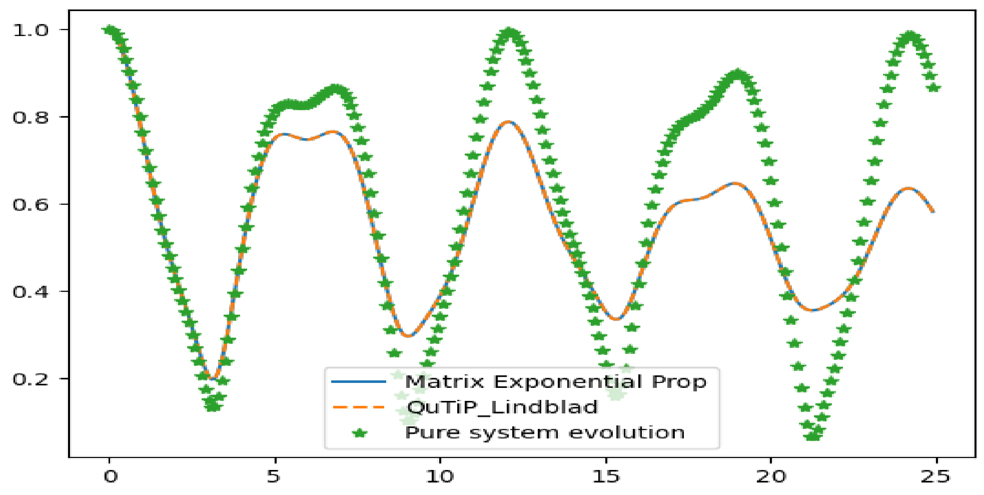

Figure 2: Quantum dynamics of the spin-chain for the closed system (dashed orange line) and the open system described by the Lindbladian dissipator described in the text (blue continuous line). The closed system oscillates between the available spin configurations of the system, while the open system gradually loses energy and resemblance with the initial state with increased simulation time.

Then, we recast the Lindblad equation in Eq. (13) into the equivalent matrix-vector form

$$\frac{d}{dt}\mathbf{v}_\rho(t) = [\mathbf{A}_C + \mathbf{A}_D]\mathbf{v}_\rho(t) \tag{18}$$

with the matrix form of the commutator $\mathbf{A}_C$ and the matrix form of Lindbladian dissipator $\mathbf{A}_D$ as

$$\begin{aligned} \mathbf{A}_C &= -i(H \otimes \mathbb{I} - \mathbb{I} \otimes H^T) \\ \mathbf{A}_D &= \sum_{m=1}^{2}\sum_{n=0}^{N-1} \frac{1}{2}\gamma_{m,n}\left[2L_{m,n} \otimes L^*_{m,n} - \mathbb{I} \otimes L^T_{m,n}L^*_{m,n} - L^\dagger_{m,n}L_{m,n} \otimes \mathbb{I}\right] \end{aligned} \tag{19}$$

where $\mathbb{I}$ is the identity matrix in the Hilbert space of H. By integrating Eq. 18, the density matrix at time t can be expressed according to the action of the exponential of the Lindbladian dissipator on the vectorized density matrix at $t = 0$,

$$\mathbf{v}_\rho(t) = \mathbf{G}(t)\mathbf{v}_\rho(0) = e^{[\mathbf{A}_C+\mathbf{A}_D]t}\mathbf{v}_\rho(0) \tag{20}$$

where $\mathbf{G}(t)$ is called the propagator. In our particular example, $\mathbf{v}_\rho(0) = |\uparrow\downarrow\downarrow\uparrow\downarrow\downarrow\rangle = |011011\rangle$ is the vectorized form of $\rho(0)$.

Script 2.11: Vectorized Lindblad equation

```python
vec_rho0 = rho0.reshape(ndvr**2)

#derivation matrix
ident_h = np.eye(ndvr, dtype = np.complex_)
Amat = -1j*(np.kron(hsys,ident_h)-np.kron(ident_h,hsys.T))

# the lindblad term
for i in range(len(Lindbladian)):
  Amat += (2.0*(np.kron(Lindbladian[i],Lindbladian[i].conj())) - np.kron(ident_h,
  ↪   Lindbladian[i].T@Lindbladian[i].conj()) -
  ↪   np.kron(Lindbladian[i].T.conj()@Lindbladian[i],ident_h))*0.5

# the propagator
Gprop = []
for i in range(nsteps+1):
  Gprop.append(sp.expm(Amat*dt*i))
```

Since the propagator $\mathbf{G}(t)$ for the Lindblad equation is non-unitary, we apply the Sz.-Nagy's dilation in Eq. (31) to convert $\mathbf{G}(t)$ into the unitary $\mathbf{U}_G(t)$. Application of the dilation procedure requires doubling each dimension of the matrix, which in practice means adding an ancilla qubit for quantum simulation.

Script 2.12: Sz.-Nagy 1-dilation of Lindbladian Propagator

```python
from numpy import linalg as la
import scipy.linalg as sp

def dilate(array):

  # Normalization factor of 1.5 to ensure contraction
  norm = la.norm(array,2)*1.5
  array_new = array/norm

  ident = np.eye(array.shape[0])

  # Calculate the conjugate transpose of the G propagator
  fcon = (array_new.conjugate()).T

  # Calculate the defect matrix for dilation
  fdef = sp.sqrtm(ident - np.dot(fcon, array_new))

  # Calculate the defect matrix for the conjugate of the G propagator
  fcondef = sp.sqrtm(ident - np.dot(array_new, fcon))

  # Dilate the G propagator to create a unitary operator
  array_dilated = np.block([[array_new, fcondef], [fdef, -fcon]])

  return array_dilated, norm
```

Thus, the dynamics in the dilated space are given by the expression

$$\tilde{\mathbf{v}}_\rho(t) = \mathbf{U}_G(t)\tilde{\mathbf{v}}_\rho(0) \tag{21}$$

with $\tilde{\mathbf{v}}_\rho(0) = [\mathbf{v}_\rho(0), 0, \cdots, 0] = |0011011\rangle$ being the zero-padded input vector to match the dimensionality of the expanded Hilbert space. Since $\tilde{\mathbf{v}}_\rho(t) = [\mathbf{v}_\rho(t), \mathbf{v}_N(t)]$, the relevant information regarding the time evolved density vector, $\mathbf{v}_\rho(t)$, can be extracted from $\tilde{\mathbf{v}}_\rho(t)$ while the remaining $\mathbf{v}_N(t)$ in the extend space can be discarded.

To calculate the survival amplitude in Eq. (16) in this dilated formalism, we use the transformed expression

$$P_s(t) = \sqrt{\langle \tilde{\mathbf{v}}_\rho(t)|\tilde{\mathbf{v}}_\rho(0)\rangle} \tag{22}$$

whre $\langle \tilde{\mathbf{v}}_\rho(t)|\tilde{\mathbf{v}}_\rho(0)\rangle$ can be obtained by taking the square root of the probability of measuring the $\tilde{\mathbf{v}}_\rho(0) = |0011011\rangle$ component in the quantum circuit and multiplying by the normalization factor n_c (explain?...this seems defined in the Yuchen's section...) utilized in the dilation process.

Script 2.13: Quantum Device Simulation of Spin Chain Dynamics

```python
# initial state in the dilated space
rho0_dilated = np.concatenate((vec_rho0,np.zeros(ndvr**2)))

Prob_qc = np.zeros(nsteps+1,dtype=np.float_)

#aersim=AerSimulator(method='statevector')
aersim=AerSimulator()
shots = 10000
for i in range(nsteps):
    if(i%10==0):print('istep',i)
    qr = QuantumRegister(nsite*2+1)  # Create a quantum register
    cr = ClassicalRegister(nsite*2+1)  # Create a classical register to store
    ↪  measurement results
    qc = QuantumCircuit(qr, cr)  # Combine the quantum and classical registers to
    ↪  create the quantum circuit

    # Initialize the quantum circuit with the initial state
    qc.initialize(rho0_dilated, qr)

    # Dilated propagator
    U_G, norm = dilate(Gprop[i])

    # Create a custom unitary operator with the dilated propagator
    U_G_op = Operator(U_G)

    # Apply the unitary operator to the quantum circuit's qubits
    qc.unitary(U_G_op, qr)

    qc.measure(qr, cr)

    counts1 = aersim.run(qc,shots=shots).result().get_counts()
    if '0011011' in counts1:
      aa = np.sqrt(counts1['0011011']/shots)*norm
      #print(aa)
      Prob_qc[i] = np.sqrt(aa)

plt.plot(time,Prob_qc,'*',label="quantum")
plt.plot(time,Prob,'--',label="QuTip benchmark")
plt.legend()
```

It is noteworthy to mention that the simulation needs to be executed many times for each time point to obtain statistically meaningful results after averaging. To demonstrate this importance, we showcase results obtained for the open quantum dynamics by performing the simulation for 10000 shots (Fig. 3, blue stars).

times for each time point. It can be observed that while the the results agree with the classical computer benchmark for short propagation time, more shots are necessary to obtain agreement at longer times

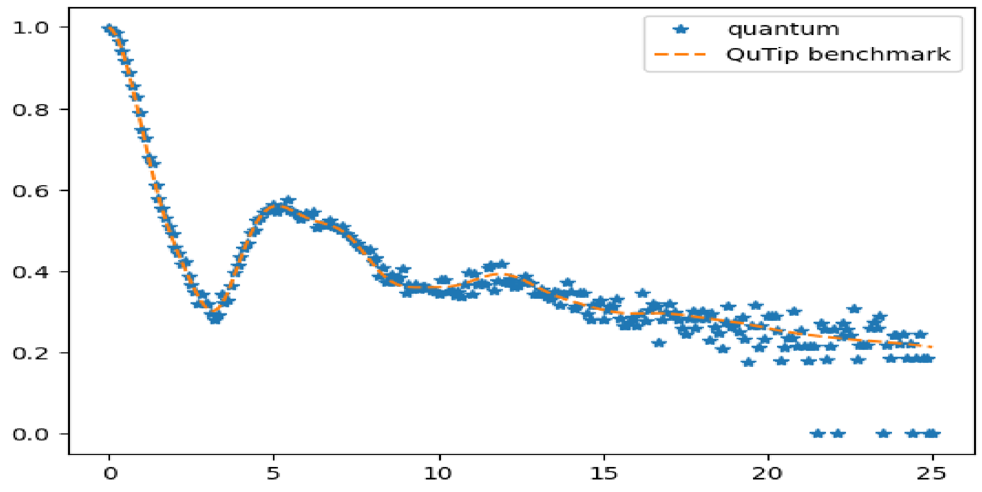

Figure 3: Quantum simulation of the Lindbladian dynamics of the spin-chain model system using a quantum device simulator (blue stars) as compared to those obtained by a digital computer simulation with QuTip.

3 Quantum simulation of open system described by the Lindblad master equation

In Sec. 1 we introduce the open quantum dynamics governed by the Lindblad equation:

$$\frac{d}{dt}\rho(t) = -\mathrm{i}[H, \rho(t)] + \mathcal{L}_D[\rho(t)], \tag{23}$$

in which $\rho(t)$ is the density matrix of the system. We trace out the environment density matrix in the derivation of the Lindblad equation and represent the environmental interactions with the dissipative term $\mathcal{L}_D[\rho(t)]$ as follows:

$$\mathcal{L}_D[\rho(t)] = \sum_{k=1}^{K} \gamma_k \left(L_k \rho(t) L_k^\dagger - \frac{1}{2}\{L_k^\dagger L_k, \rho(t)\} \right). \tag{24}$$

The Lindblad operators are shown as L_k and are taken to be dimensionless. As the Lindblad equation models a dissipative system, the rates of dissipation or decay, represented as γ_k, are non-negative quantities with the dimension of inverse time.

3.1 Quantum algorithm utilizing the Sz.-Nagy dilation theorem

In this subsection, we introduce a general algorithm utilizing the Sz.-Nagy dilation theorem for open quantum system dynamics that is described by the Lindblad equation Eq.(23). [11,12] To start with, the time-evolution for such open systems can be given in terms of Kraus operators

$$\rho(t) = \sum_k M_k(t)\rho(0)M_k^\dagger(t). \tag{25}$$

To relate the Kraus operator sum representation to the dissipative terms of the Lindblad equation Eq.(24) we associate each of L_k with a Kraus operator $M_k(\delta t)$:

$$M_k(\delta t) = e^{-iH\delta t}\sqrt{\gamma_k \delta t} L_k, \tag{26}$$

and set $M_0(\delta t) = e^{-iH\delta t}(\mathbf{I} - \frac{1}{2}\delta t \sum_{k>0} \gamma_k L_k^\dagger L_k)$. Starting with $\rho(t)$ and incrementing time by a infinitesimal δt we get:

$$\begin{aligned} e^{iH\delta t}\rho(t+\delta t)e^{-iH\delta t} &= M_0(\delta t)\rho(t)M_0^\dagger(\delta t) + \sum_{k>0} M_k(\delta t)\rho(t)M_k^\dagger(\delta t) \qquad (27) \\ &= \rho(t) - \frac{1}{2}\delta t \sum_{k>0} \gamma_k\{L_k^\dagger L_k, \rho(t)\} + \mathcal{O}(\delta t^2) + \delta t \sum_{k>0} \gamma_k L_k \rho(t) L_k^\dagger \end{aligned}$$

It is easy to see that when $\delta t \to 0$ Eq.(27) is converging to Eq.(23). Meanwhile

$$\begin{aligned}\sum_k M_k^\dagger(\delta t)M_k(\delta t) &= M_0^\dagger(\delta t)M_0(\delta t) + \sum_{k>0} M_k^\dagger(\delta t)M_k(\delta t) \\ &= \mathbf{I} - \delta t\sum_{k>0}\gamma_k L_k^\dagger L_k + \mathcal{O}(\delta t^2) + \delta t\sum_{k>0}\gamma_k L_k^\dagger L_k \\ &= \mathbf{I} + \mathcal{O}(\delta t^2).\end{aligned} \tag{28}$$

Therefore the condition of $\sum_k M_k^\dagger(\delta t)M_k(\delta t) = \mathbf{I}$ in the Kraus' theorem is satisfied if the higher order terms of δt are ignored. In our quantum algorithm, we redefine

$$M_0(\delta t) = \sqrt{\mathbf{I} - \sum_{k>0} M_k^\dagger(\delta t)M_k(\delta t)} \tag{29}$$

in order to avoid any problems caused by the approximation. Notice that the Lindblad equation which governs the dynamics of the system is a differential equation on the density operator. Now that this dynamics is described by the Kraus operators, the sum representation essentially evolves the differential equation with the Euler method. For example, the evolution after the first 3 time steps looks like (omitted the δt dependence of M_k for simplicity)

$$\begin{aligned}\rho(\delta t) &= \sum_k M_k\rho(0)M_k^\dagger \\ \rho(2\delta t) &= \sum_k M_k\rho(\delta t)M_k^\dagger = \sum_j\sum_k M_jM_k\rho(0)M_k^\dagger M_j^\dagger \\ \rho(3\delta t) &= \sum_k M_k\rho(2\delta t)M_k^\dagger = \sum_i\sum_j\sum_k M_iM_jM_k\rho(0)M_k^\dagger M_j^\dagger M_i^\dagger\end{aligned} \tag{30}$$

The full dynamics can be obtained by dividing the time length into small enough steps and applying the corresponding Kraus operators of each time step to the density function ρ. In other words, the dynamic of the system is calculated in a step-by-step manner for each of the designated time steps.

To simulate this process and transform the Kraus operators into quantum gates we need to convert the non-unitary process into a unitary operator. The conversion of a non-unitary process to a unitary operation is facilitated by Sz.-Nagy's unitary dilation procedure.[13] This procedure starts with a non-unitary *contraction* T, which means the operator norm of T is less than or equal to 1, i.e. $||T||_{\mathrm{op}} = \sup \frac{||T\boldsymbol{v}||}{||\boldsymbol{v}||} \leq 1$. Since the Kraus operators M are contractions themselves by definition which is demonstrated in Ref. 11, we replace T with M for the rest of the tutorial for simplicity. With Sz.-Nagy's unitary dilation procedure, we can convert M into a unitary U_M of the form:

$$U_M = \begin{pmatrix} M & D_{M^\dagger} \\ D_M & -M^\dagger \end{pmatrix} \text{ Where } D_M = \sqrt{I - M^\dagger M},\ D_{M^\dagger} = \sqrt{I - MM^\dagger}. \tag{31}$$

Here D_M is called the defect operator of M and the operation to get U_M is called the 1-dilation. Recently, this 1-dilation technique is used in simulating open system dynamics.[14–16] The circuit depth of the dilated U_M can be further reduced by utilizing the singular-value decomposition of the operator M, as detailed in Ref.[15]

The 1-dilation procedure can be generalized to convert a sequence composed by k number of $\{M_i | i = 1, \cdots, k\}$ to a sequence of unitary operations via the k-dilation for each of the M_i as follows:

$$U_{M_i} = \begin{pmatrix} M_i & & & & D_{M_i^\dagger} \\ D_{M_i} & & & & -M_i^\dagger \\ & I & & & \\ & & \ddots & & \\ & & & I & \end{pmatrix}. \tag{32}$$

The empty slots in Eq.(32) should be filled with 0s, and the sub-diagonal dots are filled with identity matrices I of the same dimension as M_i and there are $k-1$ of Is. Given the dimension of M_i to be $n \times n$, the k-dilation creates a unitary matrix of the dimension $(k+1)n \times (k+1)n$. This k-dilation has the minimal dimension possible compared to other schemes of dilation which is proved by the *Sz.-Nagy's Dilation Theorem.*[13] The physical explanation of the k-dilation is that applying k-number of contractions M_i on a smaller Hilbert space can be replicated by applying a list of unitary U_{M_i} where $i = 1, \cdots, k$ on a $(k+1)n$ times larger Hilbert space and projected the output vector of the larger Hilbert space into the smaller Hilbert space. It is worth mentioning that the input state in the dilated space should match the dimension of the dilated operators by adding zeros in the ancillary space.

Next, we show the details of the quantum algorithm.[11,14] For simplicity, the notation of time dependency is omitted hereafter for superoperators M and superoperators derived from it.In the first step, the density matrix is flattened to vector form: $\rho \rightarrow \mathbf{v}_\rho = (\rho_{11}, ..., \rho_{1n}, \rho_{21}, ..., \rho_{2n}, ..., \rho_{n1}, ..., \rho_{nn})^T$. We calculate the Frobenius norm of $\mathbf{v}_\rho$ as $\|\mathbf{v}_\rho\|_F = \sqrt{\sum\limits_{ij} |\rho_{ij}|^2}$ and divide $\mathbf{v}_\rho$ by $\|\mathbf{v}_\rho\|_F$ to normalize $\mathbf{v}_\rho$. The operator M_k is then transformed into $\tilde{M}_k = M_k \otimes I$; similarly, the $M_k^\dagger$ is transformed into $\tilde{N}_k = I \otimes \bar{M}_k$. The $\otimes$ stands for the Kronecker product and the bar over M_k indicates complex conjugation. The new equivalent form for the Kraus representation is:

$$M_k \rho M_k^\dagger \overset{\text{equivalent}}{\longleftrightarrow} \tilde{N}_k \tilde{M}_k \mathbf{v}_\rho. \tag{33}$$

To build the quantum circuit of $\tilde{N}_k \tilde{M}_k \mathbf{v}_\rho$ with unitary gates, we need two separate 2-dilations as there are 2 operators multiplying to $\mathbf{v}_\rho$:

$$\tilde{N}_k \tilde{M}_k \mathbf{v}_\rho \xrightarrow{\text{unitary dilation}} \mathbf{U}_{N_k} \mathbf{U}_{M_k} \left(\mathbf{v}_\rho^T, 0, ..., 0\right)^T. \tag{34}$$

For M_k of dimension $n \times n$, $\tilde{M}_k$ and $\tilde{N}_k$ are $n^2 \times n^2$; and consequently, the 2-dilation $\mathbf{U}_{M_k}$ and $\mathbf{U}_{N_k}$ are $3n^2 \times 3n^2$. All the evolved density matrices in the circuit calculated at each time-step are obtained as the output vector $\mathbf{v}_k(t) = \tilde{N}_k \tilde{M}_k \mathbf{v}_\rho$. The desired information to be collected from the density matrix is extracted by applying projection measurements on $\mathbf{v}_k(t)$.

3.2 Quantum simulation of the amplitude-damping channel with QASM simulator

In this subsection, we show the example Python code of the algorithm with an example of the spontaneous emission of a 2-level atom modeled by amplitude-channel damping. The corresponding Lindblad master equation is:

$$\dot{\rho}\,(t) = \gamma \left[\sigma^{+}\rho(t)\sigma^{-} - \frac{1}{2}\{\sigma^{-}\sigma^{+}, \rho(t)\} \right],$$

where the spontaneous emission rate is $\gamma = 1.52 \times 10^{9}$ s^{-1}, and the $\sigma^{+} = |0\rangle\langle 1|$ and $\sigma^{-} = (\sigma^{+})^{\dagger}$ are Pauli raising and lowering operators, respectively. The density matrix $\rho(t)$ in the Kraus representation is as follows:

$$\begin{aligned} \rho(t) &= M_0(t)\rho M_0(t)^{\dagger} + M_1(t)\rho M_1(t)^{\dagger}, \\ M_0(t) &= \tfrac{1+\sqrt{e^{-\gamma t}}}{2}\mathbf{I} + \tfrac{1-\sqrt{e^{-\gamma t}}}{2}\sigma_z = \begin{pmatrix} 1 & 0 \\ 0 & \sqrt{e^{-\gamma t}} \end{pmatrix}, \\ M_1(t) &= \sqrt{1-e^{-\gamma t}}\sigma^{+} = \begin{pmatrix} 0 & \sqrt{1-e^{-\gamma t}} \\ 0 & 0 \end{pmatrix}. \end{aligned} \tag{35}$$

For M_k of dimension 2×2, $\tilde{M}_k$, $\tilde{N}_k$, and D_A are 4×4 matrices, as given in Eq. (36). In this way, the 2-dilations $\mathbf{U}_{M_k}$ and $\mathbf{U}_{N_k}$ are 12×12 following the k-dilation.[11,13] Although these superoperators are time dependent, we omitted the notation of time dependency for simplicity. Since the dimension of n-qubits is $2^n \times 2^n$, we append the dilated matrix with an ancillary 12×4 zero matrix on the right and 4×12 at the bottom, and an 4×4 identity matrix along the diagonal. The resulting dilated superoperator matrix is 16×16, requiring 4 qubits for the quantum implementation.

Below we list the explicit matrix forms of each of the operators we mentioned:

$$\tilde{M}_0 = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & \sqrt{e^{-\gamma t}} & 0 \\ 0 & 0 & 0 & \sqrt{e^{-\gamma t}} \end{pmatrix}, \quad \tilde{N}_0 = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & \sqrt{e^{-\gamma t}} & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & \sqrt{e^{-\gamma t}} \end{pmatrix},$$

$$\mathbf{D}_{\tilde{M}_0} = \begin{pmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & \sqrt{1-e^{-\gamma t}} & 0 \\ 0 & 0 & 0 & \sqrt{1-e^{-\gamma t}} \end{pmatrix}, \quad \mathbf{D}_{\tilde{N}_0} = \begin{pmatrix} 0 & 0 & 0 & 0 \\ 0 & \sqrt{1-e^{-\gamma t}} & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & \sqrt{1-e^{-\gamma t}} \end{pmatrix}.$$

$$\tilde{M}_1 = \begin{pmatrix} 0 & 0 & \sqrt{1-e^{-\gamma t}} & 0 \\ 0 & 0 & 0 & \sqrt{1-e^{-\gamma t}} \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix}, \; \tilde{N}_1 = \begin{pmatrix} 0 & \sqrt{1-e^{-\gamma t}} & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & \sqrt{1-e^{-\gamma t}} \\ 0 & 0 & 0 & 0 \end{pmatrix},$$

$$\mathbf{D}_{\tilde{M}_1} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & \sqrt{e^{-\gamma t}} & 0 \\ 0 & 0 & 0 & \sqrt{e^{-\gamma t}} \end{pmatrix}, \; \mathbf{D}_{\tilde{N}_1} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & \sqrt{e^{-\gamma t}} & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & \sqrt{e^{-\gamma t}} \end{pmatrix}. \tag{36}$$

For an initial density $\rho(0) = \frac{1}{4}\begin{pmatrix} 1 & 1 \\ 1 & 3 \end{pmatrix}$, we calculate the populations in the basis $\{|0\rangle, |1\rangle\}$ from $t = 0$ to $t = 1000$ ps with a time step of 10 ps. With $\|\rho\|_F = \frac{\sqrt{3}}{2}$, the input state is:

$$\mathbf{v}_0 = \frac{1}{\|\rho\|_F}\left(\mathbf{v}_\rho^T, \overbrace{0, ..., 0}^{m}\right)^T = \frac{1}{2\sqrt{3}}\left(1, 1, 1, 3, \overbrace{0, ..., 0}^{m}\right)^T, \tag{37}$$

where $m = 12$ for the vector $\mathbf{v}_\rho^T$ to be of length 16. After extracting the output $\mathbf{v}_k(t)$, the ground state and excited state populations are obtained as the first and fourth entry of the vector, respectively. More technical details of the quantum algorithm and the amplitude-channel damping model can be found in the Appendix of Refs.[11,14]

Next, we include a Python coding example of the quantum algorithm working for the amplitude-channel damping model. Qiskit package from IBM quantum are required for the execution of the code.[17] We start with importing essential packages and the four major libraries imported for this code are: ”numpy” for all matrix related numerical calculations, ”scipy” for calculating square root of matrix, ”matplotlib” for generating plots from results obtained, ”qiskit” for all types of quantum implementation. If one faces any issues importing these libraries, one must check their respective official user-manuals for the recent changes made to them.

Script 3.1: Importing Qiskit packages

```python
#These libraries might vary depending on the version of qiskit.
import numpy as np
import scipy.linalg as sp
import matplotlib.pyplot as plt
import qiskit
from qiskit import *
from qiskit_aer import AerSimulator
from qiskit.quantum_info.operators import Operator
```

Next we defining the density matrix and parameters used to build the Kraus operators. There are in total two Kraus operators used in this calculation and we encode them as k_0 and k_1.

Script 3.2: Setting up the input parameters

```python
#defining the initial density matrix rho
rho = np.zeros((2,2),'complex')
rho[0,0]=1/4
rho[0,1]=1/4
rho[1,0]=1/4
rho[1,1]=3/4
gamma=1.52e9# gamma is the spontaneous emission rate

iden=np.eye(2)
iden2= np.eye(4)

#flattening or vectorize the density matrix
rho_norm=sp.norm(rho)
rho_flat= rho.flatten()/rho_norm
#defining the Kraus operators
def k_0(x):
      k = np.zeros((2,2),'complex')
      k[0,0]=1
      k[0,1]=0
      k[1,0]=0
      k[1,1]= np.sqrt(np.exp(-gamma*x))
      return k
def k_1(x):
      k = np.zeros((2,2),'complex')
      k[0,0]=0
      k[0,1]=np.sqrt(1-np.exp(-gamma*x))
      k[1,0]=0
      k[1,1]=0
      return k
```

Then we defining the function that does the 2-dilation process and transforms the Kraus operators into unitary operators of a larger dimension.

Script 3.3: Functions for dilation

```python
#defining the function for dilation
def udil(k):
#first and second stands for the M and N, i.e. Kraus and its complex conjugate
    first= np.kron(k,iden)
    kc=k.conjugate()  #complex conjugate of k
    second= np.kron(iden,kc)
    fcon = (first.conjugate()).T
    scon = (second.conjugate()).T
    #calculate the defect operators of each Kraus operators
    fdef = sp.sqrtm(iden2-np.dot(fcon,first))
    sdef = sp.sqrtm(iden2 -np.dot(scon,second))
    fcondef = sp.sqrtm(iden2-np.dot(first,fcon))
    scondef = sp.sqrtm(iden2-np.dot(second,scon))
    #2-dilation process
    Ufirst= np.block([[first, np.zeros((4,4)),fcondef,np.zeros((4,4))],
    [fdef,np.zeros((4,4)),-fcon,np.zeros((4,4))],
    [np.zeros((4,4)),np.eye(4),np.zeros((4,4)),np.zeros((4,4))],
    [np.zeros((4,4)),np.zeros((4,4)),np.zeros((4,4)),np.eye(4)]])

    Usecond=np.block([[second, np.zeros((4,4)),scondef,np.zeros((4,4))],
    [sdef,np.zeros((4,4)),-scon,np.zeros((4,4))],
    [np.zeros((4,4)),np.eye(4),np.zeros((4,4)),np.zeros((4,4))],
    [np.zeros((4,4)),np.zeros((4,4)),np.zeros((4,4)),np.eye(4)]])

    ufre=np.reshape(Ufirst,(1,256))
    usec=np.reshape(Usecond,(1,256))
    #returning the unitaries for both the Kraus operator and its complex conjugate
    return ufre,usec
```

In the following code we assigning the time steps and defining various lists used to store the simulation results.

Script 3.4: Defining Python lists to record the results

```python
#defining lists used to store the results
rho_at_t=np.zeros((1000,2), "complex")
time=np.zeros((1000,1),)
udil01=np.zeros((1000,256),"complex")
udil02=np.zeros((1000,256),"complex")
udil11=np.zeros((1000,256),"complex")
udil12=np.zeros((1000,256),"complex")
# assigning the time-steps
for t in range(0,1000):
    tt=t*10**-12
    k0=k_0(tt)
    k1=k_1(tt)
    udil01[t],udil02[t] = udil(k0)
    udil11[t],udil12[t] = udil(k1)

shots = 2000
#create dictionaries to store the results. three binary digits due to 8*8 matrix needs
↪ three qubits
result = {'0000': 0, '0001': 0, '0010': 0, '0011': 0,'0100': 0, '0101': 0, '0110': 0,
↪ '0111': 0, '1000': 0, '1001': 0, '1010': 0, '1011': 0,'1100': 0, '1101': 0,
↪ '1110': 0, '1111': 0}
result2 = {'0000': 0, '0001': 0, '0010': 0, '0011': 0,'0100': 0, '0101': 0, '0110': 0,
↪ '0111': 0, '1000': 0, '1001': 0, '1010': 0, '1011': 0,'1100': 0, '1101': 0,
↪ '1110': 0, '1111': 0}
p_excited = []# create list to store probability for acceptor state
p_ground = []# create list to store probability for donor state
```

Below is the code used to perform the iterations of the quantum algorithm that goes over all 1000 times steps used in this specific case. Noticed that the two Kraus operators are implemented separately and sum up at the end to generate the results. In the case of a more general Lindblad equation, all the derived Kraus operators can be calculated in parallel to reduce the total computational time.

Script 3.5: QASM simulations

```python
# Below are the loop of the QASM simulation
aersim=AerSimulator()
for i in range (0,1000):
   #for Kraus operator 1
   U1first=np.reshape(udil01[i],(16,16))
   U1second=np.reshape(udil02[i],(16,16))
   # qiskit quantum circuit generation process
   initial_state = np.concatenate([rho_flat,np.zeros(12)])
   qr = QuantumRegister(4)
   cr = ClassicalRegister(4)
   qc = QuantumCircuit(qr, cr)
   qc.initialize(initial_state, qr)
   # generating self-defined quantum gates with dilated unitaries
   A = Operator(U1first)
   B = Operator(U1second)
   U_G_op = A.compose(B)
   qc.unitary(U_G_op, qr)
   qc.measure(qr, cr)
   counts1 = aersim.run(qc,shots=shots).result().get_counts()
   for x in counts1:
      result[x] = counts1[x]
   # multiply the norm factor to retrieve the original data, sqrt because the results
   ↪  is probability not prob density
   pg1 = np.sqrt(result['0000'] / 2000)*rho_norm
   pe1 =  np.sqrt(result['0011'] / 2000)*rho_norm
   #for Kraus operator 2
   U2first=np.reshape(udil11[i],(16,16))
   U2second=np.reshape(udil12[i],(16,16))
   initial_state2 = np.concatenate([rho_flat,np.zeros(12)])
   qr2 = QuantumRegister(4)
   cr2 = ClassicalRegister(4)
   qc2 = QuantumCircuit(qr2, cr2)
   qc2.initialize(initial_state, qr2)
   C = Operator(U2first)
   D = Operator(U2second)
   U_G_op2 = C.compose(D)
   qc2.unitary(U_G_op2, qr2)
   qc2.measure(qr2, cr2)
   counts2 = aersim.run(qc2, shots=shots).result().get_counts()
   for x in counts2:
      result2[x] = counts2[x]
   # multiply the norm factor to retrieve the original data, sqrt because the results
   ↪  is probability not prob density
   pg2 = np.sqrt(result2['0000'] / 2000)*rho_norm
   pe2 =  np.sqrt(result2['0011'] / 2000)*rho_norm
   pe=pe1+pe2
   pg=pg1+pg2
   p_excited.append(pe)
   p_ground.append(pg)
```

The distribution probability of each states is used to calculate the population of the donor state and acceptor state and the following code is used to plot the results for a better visualization of the system dynamics. The plot of the simulation is shown as Fig. 4.

Script 3.6: Plotting the results

```python
#plotting the QASM simulation results
time_array_QASM = np.arange(0, 1000, 1)

plt.plot(time_array_QASM, p_ground, 'r-', label="Ground state")
plt.plot(time_array_QASM, p_excited, 'b-', label='Excited state')
```

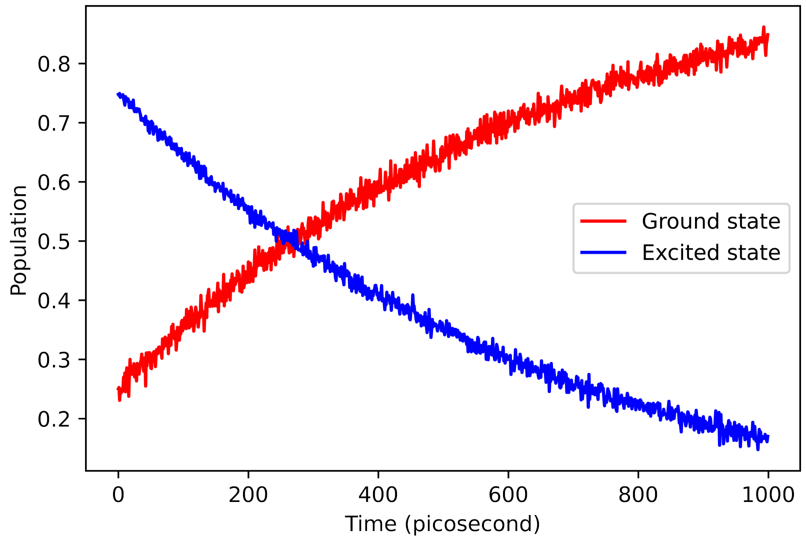

Figure 4: Population of ground state and excited state for the amplitude-damping model obtained by the quantum implementation on the QASM simulator.The population of the ground state is denoted in red and the population of the excited state is denoted in blue. The population distributions are simulated for 1000 time steps. The number of projection measurements applied by the QASM simulator to obtain a single time step is 2000 shots.

3.3 Quantum circuit generator using the Group Leader Optimization Algorithm

In this section, we introduce a heuristic method that generates a quantum circuit for any given unitary matrix.[18] The goal of this subsection is to present an alternative way to generate a quantum circuit. It is worth pointing out that this method does not guaranteed to produce the optimal quantum circuit compares to other commonly used circuit generator such as the 'transpile' function in Qiskit. Instead we just provide a perspective on how this specific circuit generator works and hopefully this can enhance the reader's understanding of this important process in the realm of quantum computing.

This quantum circuit generator utilizes the Group Leader Optimization Algorithm (GLOA) which takes its inspiration from the influence of the leaders to other members of social groups.[19] GLOA is an evolutionary algorithm that decomposes the unitary matrices into a set of quantum gates by randomly generating a large amounts of circuit candidates, dividing them into groups and doing optimization in parallel under the guidance of the group leaders within each groups. Here the group leader is the member with best performance and is constantly updating after each iteration.

The goal of the optimization process is to generate and compile the quantum circuits with minimum costs and errors. So we variate and optimize the generated quantum circuits to have the lowest cost in terms of the number of the gate and the lowest fidelity error in terms of the distance between the target and the generated quantum circuits in the form of unitary matrix. In our case, one qubit gate has a cost of 1 and two-qubit gate has a cost of 2. It is worth noting that the minimization of the error to an acceptable level has more priority than lowering the cost because we prefer to get more accurate and reliable results before reducing the number of gates in the circuit. In the optimization process, the balance of the weight of the cost and the error in the approximated circuit can be adjusted by an objective function. Here the trace fidelity is given by:

$$\mathcal{F} = \frac{1}{N}|\mathrm{Tr}(U_a U_t^\dagger)|, \tag{38}$$

where $N = 2^n$ (n is the number of qubits); the dagger symbol $\dagger$ represents the complex conjugate transpose of a matrix; $\mathrm{Tr}(\cdot)$is the trace of a matrix; $|\cdot|$ means taking the absolute value and U_a is the matrix representation of the approximated circuit and U_t is the target unitary matrix. The fidelity error used in the optimization is defined as:

$$\epsilon = 1 - \mathcal{F}^2 \tag{39}$$

where $\mathcal{F}$ is squared to amplify the effects of small fidelity changes in the error.

In the optimization we utilize the method of the Cartesian genetic programming so that each quantum gate within the generated circuit is represented as integer strings that include information of the position, free parameters (if applicable) and controllability if it is a controlled gate. The variation of the group member following the mutation rule which is:

$$\begin{aligned} \text{new member} =& \, r1 \text{ portion of old member} \\ & \cup \, r2 \text{ portion of group leader} \\ & \cup \, r3 \text{ portion of random} \end{aligned}$$

In our case this mutation rule is apply to the free parameters of the applicable quantum gates. In addition to the mutation, after a given number of iterations, one-way-crossover (also called the parameter transfer) is done between a member from the target group and a member from a different group all chosen at random. The full process of the iteration is showing as a flowchart in Fig. 5. The replacement criteria in our optimization is as follow: if a new formed (or mutated) member results in less error compared to the old member, or they have the same error values but the cost of the new member is less than the old member, then the new member survives and replaces the old member; otherwise, the old member is kept for the next iteration.

In the rest of this section we demonstrate a demo code that allows you to utilize this circuit generator after you have all the necessary packages installed and configured. To begin with, we need to import the necessary packages and functions required for the procedure. The definition of all the functions used can be found in the qtoolkit package and the brief explanation of each called function is listed as comments. Noticed that it is important to install and configure the Qiskit package as there is a function in the qtoolkit to transform the quantum circuit generated into a Qiskit quantum circuit object.

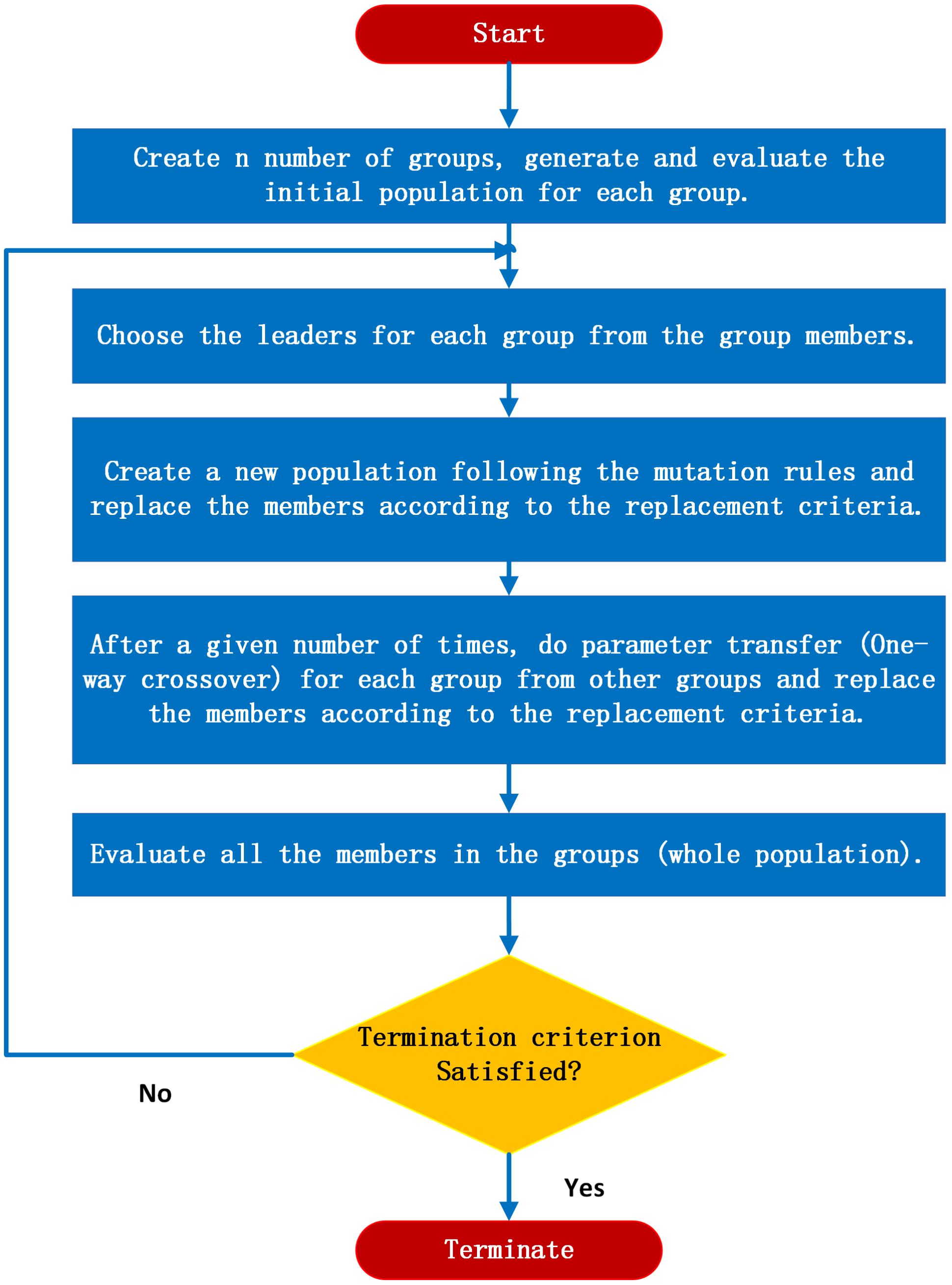
Start
Create n number of groups, generate and evaluate the initial population for each group.
Choose the leaders for each group from the group members.
Create a new population following the mutation rules and replace the members according to the replacement criteria.
After a given number of times, do parameter transfer (One-way crossover) for each group from other groups and replace the members according to the replacement criteria.
Evaluate all the members in the groups (whole population).
Termination criterion Satisfied?
No
Yes
Terminate

Figure 5: The flow chart of the Group Leaders Optimization Algorithm.

Script 3.7: Importing Functions from qtoolkit

```python
import numpy as np
import qtoolkit.gloa.group_leader as gloa #core function of gloa
import qtoolkit.solovay_kitaev.maths.distances as qdists
#measuring distance between the target and generated as parts of cost function
import qtoolkit.core.utils.constants.operations as qopconsts# all the default
↪  quantum gate
import qtoolkit.core.utils.timeit as qtimeit # run time tracking
import qtoolkit.core.data_structures.quantum_circuit as qcirc #building the
↪  quantum circuit
import qtoolkit.core.data_structures.quantum_operation as qop # operations to the
↪  quantum gates
import qtoolkit.core.utils.types as qtypes # datatype used in this program
```

With all the necessary package imported, we first declare the the basis quantum gates to be used in the algorithm and then include the unitary matrix we want to build the circuit for.

Script 3.8: Declaring basis quantum gate set and the unitary matrix

```python
# 1.Define the basis quantum gates to be used in the algorithm
# supported gates includes: X Y Z S(phase) H (Hadamard)T(pi/8) ID CX(control-not) Rx
↪  Ry Rz Sx(sqrt X)
basis = [
    qopconsts.SX,
    qopconsts.Rz,
    qopconsts.CX
]
# enter the boundary for the gates with parameters such as rotation gates. use None for
↪  those don't have parameters
bounds = [
    None,
    np.array([[0], [2 * np.pi]]),
    None,
]
timer = qtimeit.Timer()

# 2. U is the unitary we want to approximate.
# Make sure the dimension of the matrix is to the power of 2 i.e. dim=2^n
U = np.array(
    [
        [ 0.59416024+3.91441004e-16j,  0.13258395-4.03129458e-16j,
   0.78700436+8.27518375e-33j, -0.10009615+6.08222076e-18j],
 [ 0.13258395-3.89089610e-16j , 0.59416024+4.00729831e-16j,
  -0.10009615-6.08222076e-18j , 0.78700436+1.23259516e-32j],
 [ 0.78700436-1.03520189e-35j, -0.10009615+4.51736357e-18j,
  -0.59416024+3.91441004e-16j, -0.13258395-3.89089610e-16j],
 [-0.10009615-4.51736357e-18j , 0.78700436+1.54074396e-33j,
  -0.13258395-4.03129458e-16j, -0.59416024+4.00729831e-16j],
    ]
)
```

Once we have all the input setting up we can start the GLOA iteration. After each iteration the decomposition characteristics will be printed for your reference and the "U_approx" object returned is the quantum circuit object used by the qtoolkit package.

Script 3.9: Starting the GLOA process with the user-defined parameters

```python
# 3. Starting the GLOA process.
timer.tic()

cost, U_approx = gloa.group_leader(
      U, length=40, n=10, p=20, basis=basis, max_iter=100, parameters_bound=bounds,r
       ↪  = [0.6, 0.2, 0.2]
)
'''
Parameter meaning:
length: number of gates allowed, not necessarily used all of them
n: number of groups
p: number of members inside each groups
max_iter: maximum iteration
r: rates determining the portion of old (r[0]), leader (r[1]) and
      random (r[2]) that are used to generate new candidates. If None, the
      default value of the GLOA article is used: r = [0.8, 0.1, 0.1].
'''

timer.toc("GLOA algorithm")

print("Decomposition characteristics:")
print(f"Fowler error: {qdists.fowler_distance(U, U_approx.matrix)}.")
print(f"Trace error:  {qdists.trace_distance(U, U_approx.matrix)}.")
print(f"GLOA cost:    {cost}.")
print(f"Gate count:   {len(list(U_approx.operations))}")
```

Finally we show a useful function that transform the resulting circuit object into a Qiskit circuit object. Qiskit package have many built in functions that work with this object. For example, there is a function that can draw the resulting circuit into a fancy figure that allows us to visualized the circuit in an direct way as shown in Fig. 6. It is worth noting that the resulting circuit can be different after each execution due to the randomness in the members creation process. The form of the circuit also depends on all the user-defined parameters given in the code such as the length of the circuit and types of the quantum gates used. The readers are encouraged to refer to Ref.[18] for more detailed evaluation and discussion of this method used for generating quantum circuit for the propagator of the Hamiltonian of hydrogen molecule, water molecule and more. Furthermore, the readers are welcomed to compare the results generating from GLOA to that from the Qiskit package and to explore the optimal parameters sets that gives the best approximation of the given matrix in a case by case manner.

Script 3.10: Visualization of the generated circuit

```python
#U_approx.to_qiskit() is a qiskit circuit objects so .draw() can be called to draw it.
U_approx.to_qiskit().draw(output='mpl')
```

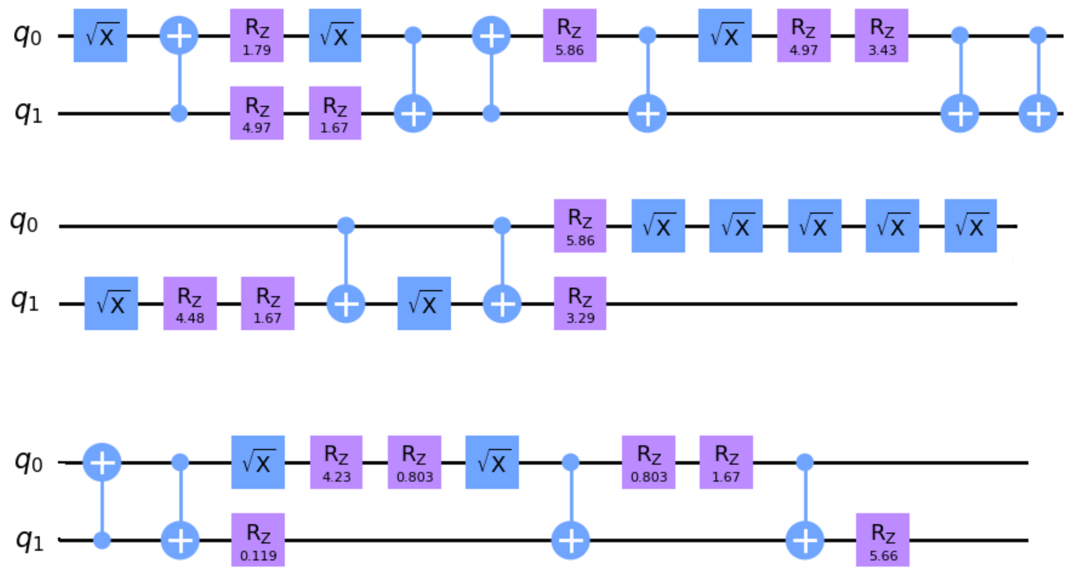

Figure 6: The quantum circuit generated by the Group Leaders Optimization Algorithm. The blue squre is the SX gate, the purple squre is the RZ gate with the rotation parameter listed below and the two-qubit gate is the CX gate. The circuit figure is rescaled and adjusted to show more details.

4 Conclusion

In part II of the whole series we delve into the realm of open quantum system, described by the popular methods like the Lindblad equation that rely on the Markovian approximation. We first introduce the Lindblad master equation and the conditions this equation is applicable and provide a classical simulation example with QuTiP. Next, we introduce a quantum algorithm that can simulate the open quantum system governed by the Lindblad equation with the help of the Kraus representation and dilation. We provide an coding example of the algorithm that works for the amplitude damping model. Lastly we introduce a quantum circuit generator that allows one to approximate a general unitary matrix with a variationally derived quantum circuit. We hope this part serves as a good guidance to the simulations of open quantum systems following the Markovian approximation with quantum computers.

Acknowledgement

We acknowledge the financial support of the National Science Foundation under award number 2124511, CCI Phase I: NSF Center for Quantum Dynamics on Modular Quantum Devices (CQD-MQD).

References

(1) Breuer, H.-P.; Petruccione, F. *The Theory of Open Quantum Systems*; Oxford University Press, 2007.

(2) Milz, S.; Pollock, F. A.; Modi, K. An introduction to operational quantum dynamics. *Open Systems & Information Dynamics* **2017**, *24*, 1740016.

(3) Manzano, D. A short introduction to the Lindblad master equation. *AIP Adv.* **2020**, *10*, 025106.

(4) Johansson, J.; Nation, P.; Nori, F. QuTiP: An open-source Python framework for the dynamics of open quantum systems. *Computer Physics Communications* **2012**, *183*, 1760–1772.

(5) Johansson, J.; Nation, P.; Nori, F. QuTiP 2: A Python framework for the dynamics of open quantum systems. *Comput. Phys. Commun.* **2013**, *184*, 1234–1240.

(6) Nitzan, A. *Chemical Dynamics in Condensed Phases*; Oxford University Press: New York, 2006.

(7) Jang, S. *Dynamics of Molecular Excitons*; Nanophotonics; Elsevier Science, 2020.

(8) Wang, T.; Sanz, S.; Castro-Esteban, J.; Lawrence, J.; Berdonces-Layunta, A.; Mohammed, M. S. G.; Vilas-Varela, M.; Corso, M.; Peña, D.; Frederiksen, T.; de Oteyza, D. G. Magnetic Interactions Between Radical Pairs in Chiral Graphene Nanoribbons. *Nano Letters* **2022**, *22*, 164–171, PMID: 34936370.

(9) Fiori, E. R.; Pastawski, H. Non-Markovian decay beyond the Fermi Golden Rule: Survival collapse of the polarization in spin chains. *Chem. Phys. Lett.* **2006**, *420*, 35–41.

(10) Dong, H. et al. Measuring Spectral Form Factor in Many-Body Chaotic and Localized Phases of Quantum Processors. 2024.

(11) Hu, Z.; Xia, R.; Kais, S. A quantum algorithm for evolving open quantum dynamics on quantum computing devices. *Sci. Rep.* **2020**, *10*, 1–9.

(12) Hu, Z.; Head-Marsden, K.; Mazziotti, D. A.; Narang, P.; Kais, S. A general quantum algorithm for open quantum dynamics demonstrated with the Fenna-Matthews-Olson complex. *Quantum* **2022**, *6*, 726.

(13) Levy, E.; Shalit, O. M. Dilation theory in finite dimensions: the possible, the impossible and the unknown. *Rocky Mt. J. Math.* **2014**, *44*, 203–221.

(14) Wang, Y.; Mulvihill, E.; Hu, Z.; Lyu, N.; Shivpuje, S.; Liu, Y.; Soley, M. B.; Geva, E.; Batista, V. S.; Kais, S. Simulating Open Quantum System Dynamics on NISQ Computers with Generalized Quantum Master Equations. *J. Chem. Theory Comput.* **2023**, *19*, 4851–4862.

(15) Schlimgen, A. W.; Head-Marsden, K.; Sager-Smith, L. M.; Narang, P.; Mazziotti, D. A.

Quantum state preparation and nonunitary evolution with diagonal operators. *Phys. Rev. A* **2022**, *106*, 022414.

(16) Seneviratne, A.; Walters, P. L.; Wang, F. Exact Non-Markovian Quantum Dynamics on the NISQ Device Using Kraus Operators. *ACS Omega* **2024**, *9*, 9666–9675.

(17) Aleksandrowicz, G. et al. Qiskit: An Open-source Framework for Quantum Computing. 2019; .

(18) Daskin, A.; Kais, S. Decomposition of unitary matrices for finding quantum circuits: application to molecular Hamiltonians. *J. Chem. Phys* **2011**, *134*.

(19) Daskin, A.; Kais, S. Group leaders optimization algorithm. *Molecular Physics* **2011**, *109*, 761–772.