piqs Documentation

Release 1.2

Nathan Shammah, Shahnawaz Ahmed

Jul 17, 2020

CONTENTS:

CHAPTER

ONE

INTRODUCTION

1.1 Permutational Invariant Quantum Solver (PIQS)

PIQS is an open-source Python solver to study the exact Lindbladian dynamics of open quantum systems consisting of identical qubits.

In the case where local processes are included in the model of a system's dynamics, numerical simulation requires dealing with density matrices of size 2^N. This becomes infeasible for a large number of qubits. We can simplify the calculations by exploiting the permutational invariance of indistinguishable quantum particles which allows the user to study hundreds of qubits.

1.2 Integrated with QuTiP

A major feature of PIQS is that it allows to build the Liouvillian of the system in an optimal way. It uses Cython to optimize performance and by taking full advangtage of the sparsity of the matrix it can deal with large systems. Since it is compatible with the *quantum object* class of [QuTiP] one can take full advantage of existing features of this excellent open-source library.

PIQS is integrated inside QuTiP, from QuTiP version 4.3.1, as the `qutip.piqs` module. A list of tutorials can be found on the QuTiP website. For issues related to `qutip.piqs` open an issue on QuTiP's Github page.

1.3 A wide range of applications

- The time evolution of the total density matrix of quantum optics and cavity QED systems for permutationally symmetric initial states (such as the GHZ state, Dicke states, coherent spin states).
- Phase transitions of driven-dissipative out-of-equilibrium quantum systems.
- Correlation functions of collective systems in quantum optics experiments, such as the spectral density and second-order correlation functions.
- Various quantum optics phenomena such as steady-state superradiance, superradiant light emission, superradiant phase transition, spin squeezing, boundary time crystals, resonance fluorescence.

## Installation ### Linux and MacOS We have made the package available in [conda-forge](https://conda-forge.org/). Download and install Anaconda or Miniconda from [https://www.anaconda.com/download/{]}(https://www.anaconda.com/download/) and then install *piqs* from the conda-forge repository with ` conda config --add channels conda-forge conda install conda-forge piqs ` Then you can fire up a Jupyter notebook and run *piqs* out of your browser.

### Windows Windows is not officially supported, but if you are on Windows, you can build piqs from source by first installing conda. If you have a python environment with *cython*, *numpy*, *scipy* and *qutip* installed then simply download the [source code](https://github.com/nathanshammah/piqs/archive/v1.2.tar.gz), unzip it and run the setup file inside with

` python setup.py install ` If you have any problems installing the tool, please open an issue or write to us.

### Source Alternatively, you can download the package from the [source](https://github.com/nathanshammah/piqs/archive/v1.2.tar.gz) and install with

` python setup.py install `

### QuTiP A version of PIQS is already included in the excellent QuTiP library, from version 4.3. If you have QuTiP (>4.2), you can just import the module as *qutip.piqs*.

CHAPTER

TWO

USER GUIDE

The *Permutational Invariant Quantum Solver (PIQS)* is an open-source Python solver to study the exact Lindbladian dynamics of open quantum systems consisting of identical qubits. It is integrated in QuTiP and can be imported as as a model.

Using this library, the Liouvillian of an ensemble of N qubits, or two-level systems (TLSs), $\mathcal{D}_{TLS}(\rho)$, can be built using only polynomial – instead of exponential – resources. This has many applications for the study of realistic quantum optics models of many TLSs and in general as a tool in cavity QED [1].

Consider a system evolving according to the equation

$$\begin{aligned}\dot{\rho} = \mathcal{D}_\text{TLS}(\rho) = -\frac{i}{\hbar}[H,\rho] + \frac{\gamma_\text{CE}}{2}\mathcal{L}_{J_{-}}[\rho] + \frac{\gamma_\text{CD}}{2}\mathcal{L}_{J_{z}}[\rho] + \frac{\gamma_\text{CP}}{2}\mathcal{L}_{J_{+}}[\rho] \\ + \sum_{n=1}^{N}\left(\frac{\gamma_\text{E}}{2}\mathcal{L}_{J_{-,n}}[\rho] + \frac{\gamma_\text{D}}{2}\mathcal{L}_{J_{z,n}}[\rho] + \frac{\gamma_\text{P}}{2}\mathcal{L}_{J_{+,n}}[\rho]\right)\end{aligned}$$

where $J_{\alpha,n}=\frac{1}{2}\sigma_{\alpha,n}$ are SU(2) Pauli spin operators, with $\alpha=x,y,z$ and $J_{\pm,n}=\sigma_{\pm,n}$. The collective spin operators are $J_{\alpha}=\sum_{n}J_{\alpha,n}$. The Lindblad super-operators are $\mathcal{L}_{A}=2A\rho A^\dagger - A^\dagger A\rho - \rho A^\dagger A$.

The inclusion of local processes in the dynamics lead to using a Liouvillian space of dimension 4^N. By exploiting the permutational invariance of identical particles [2-8], the Liouvillian $\mathcal{D}_\text{TLS}(\rho)$ can be built as a block-diagonal matrix in the basis of Dicke states $|j,m\rangle$.

The system under study is defined by creating an object of the `Dicke` class, e.g. simply named `system`, whose first attribute is

- `system.N`, the number of TLSs of the system N.

The rates for collective and local processes are simply defined as

- `collective_emission` defines γ_CE, collective (superradiant) emission
- `collective_dephasing` defines γ_CD, collective dephasing
- `collective_pumping` defines γ_CP, collective pumping.
- `emission` defines γ_E, incoherent emission (losses)
- `dephasing` defines γ_D, local dephasing
- `pumping` defines γ_P, incoherent pumping.

Then the `system.lindbladian()` creates the total TLS Linbladian superoperator matrix. Similarly, `system.hamiltonian` defines the TLS hamiltonian of the system H_TLS.

The system's Liouvillian can be built using `system.liouvillian()`. The properties of a Piqs object can be visualized by simply calling `system`. We give two basic examples on the use of *PIQS*. In the first example the incoherent emission of N driven TLSs is considered.

```
from piqs import Dicke
from qutip import steadystate
N = 10
system = Dicke(N, emission = 1, pumping = 2)
L = system.liouvillian()
steady = steadystate(L)
```

2.1 Superradiant Light Emission

We consider a system of N two-level systems (TLSs) with identical frequency ω_0, which can emit collectively at a rate γ_{CE}, and suffer from dephasing and local losses at rates γ_{D} and γ_{E}, respectively. The dynamics can be written as

$$\dot{\rho} = -i[\omega_0 J_z, \rho] + \frac{\gamma_{\text{CE}}}{2}\mathcal{L}_{J_-}[\rho] + \sum_{n=1}^{N}\frac{\gamma_{\text{D}}}{2}\mathcal{L}_{J_{z,n}}[\rho] + \frac{\gamma_{\text{E}}}{2}\mathcal{L}_{J_{-,n}}[\rho].$$

When $\gamma_{\text{E}} = \gamma_{\text{D}} = 0$ this dynamics is the classical superradiant master equation. In this limit, a system initially prepared in the fully-excited state undergoes superradiant light emission whose peak intensity scales proportionally to N^2.

```
from qutip import *
from piqs import *
import matplotlib.pyplot as plt

N = 20
[jx, jy, jz] = jspin(N)
jp = jspin(N, "+")
jm = jp.dag()

# spin hamiltonian
w0 = 1
H = w0 * jz

# dissipation
gCE, gD, gE = 1, 1, 0

# set initial conditions for spins
system = Dicke(N=N, hamiltonian=h, dephasing=gD,
               collective_emission=gCE)

# build the Liouvillian matrix
liouv = system.liouvillian()
```

Now that the system Liouvillian is defined, we can use QuTiP to solve the dynamics. We use as integration time a multiple of the superradiant delay time, $t_{\text{D}} = \log(N)/(N\gamma_{\text{CE}})$. We specify the operators for which the expectation values should be calculated to *mesolve* with the keyword argument *e_ops*. In this case, we are interested in J_x, J_+J_-, J_z^2.

```
nt = 1001
td0 = np.log(N)/(N*gCE)
tmax = 10 * td0

t = np.linspace(0, tmax, nt)

```

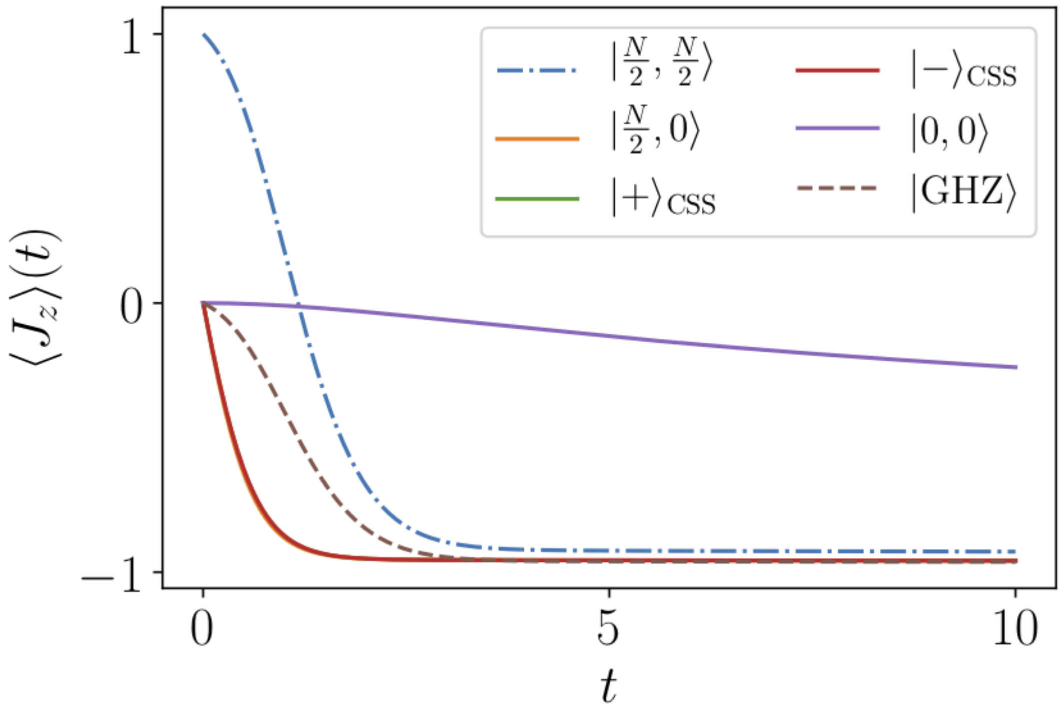
(continues on next page)

(continued from previous page)

```
# initial state
excited_rho = excited(N)

# alternative states
superradiant_rho = dicke(N, N/2, 0)
subradiant_rho = dicke(N, 0, 0)
css_symmetric = css(N)

a = 1/np.sqrt(2)
css_antisymmetric = css(N, a, -a)
ghz_rho = ghz(N)
rho0 = excited_rho
result = mesolve(liouv, rho0, t, [], e_ops = [jz, jp*jm, jz**2],
                 options = Options(store_states=True))
rhot = result.states
```

We can then plot the results of the time evolution of the expectation values of the collective spin operators for different initial states.

```
jz_t = result.expect[0]
jpjm_t = result.expect[1]
jz2_t = result.expect[2]

jmax = (0.5 * N)
fig1 = plt.figure()
plt.plot(t/td0, jz_t/jmax)
plt.show()
plt.close()
```



References:

2.2 Superradiance: Open Dicke Model

We consider a system of N two-level systems (TLSs) coupled to a cavity mode. This is known as the Dicke model

$$H = \omega_0 J_z + \omega_c a^\dagger a + g\left(a^\dagger + a\right)\left(J_+ + J_-\right)$$

where each TLS has identical frequency ω_0. The light matter coupling can be in the ultrastrong coupling (USC) regime, $g/\omega_0 > 0.1$.

If we study this model as an open quantum system, the cavity can leak photons and the TLSs are subject to local processes. For example the system can be incoherently pumped at a rate γ_P, the TLSs are subject to dephaisng at a rate γ_D, and local incoherent emission occurs at a rate γ_E. The dynamics of the coupled light-matter system is governed by

$$\begin{aligned}\dot{\rho} &= -i[\omega_0 J_z + \omega_a a^\dagger a + g\left(a^\dagger + a\right)\left(J_+ + J_-\right), \rho] \\ &+\frac{\kappa}{2}\mathcal{L}_a[\rho] + \sum_{n=1}^{N}\left(\frac{\gamma_\text{P}}{2}\mathcal{L}_{J_{+,n}}[\rho] + \frac{\gamma_\text{E}}{2}\mathcal{L}_{J_{+,n}}[\rho] + \frac{\gamma_\text{D}}{2}\mathcal{L}_{J_{+,n}}[\rho]\right)\end{aligned} \quad (1)$$

```
import matplotlib.pyplot as plt
from qutip import *
from piqs import *

#TLS parameters
N = 6
nds = num_dicke_states(N)
[jx, jy, jz] = jspin(N)
jp, jm = jspin(N, "+"), jspin(N, "-")
w0 = 1
gE, gD = 0.1, 0.01

# Hamiltonian
h = w0 * jz

#photonic parameters
nphot = 20
wc = 1
kappa = 1
ratio_g = 2
g = ratio_g/np.sqrt(N)
a = destroy(nphot)
```

After defining all the parameters, we can build a Liouvillian for the TLS ensemble and the photonic cavity. In order to study this system using QuTiP and $PIQS$, we will first build the TLS Liouvillian, then we will build the photonic Liouvillian and finally we will build the light-matter interaction. The total dynamics of the system is thus defined in a Liouvillian space that has both TLS and photonic degrees of freedom.

```
#TLS liouvillian
ensemble = dicke(N = N, hamiltonian=h, emission=gE, dephasing=gD)
liouv = ensemble.liouvillian()

#photonic liouvilian
h_phot = wc * a.dag() * a
c_ops_phot = [np.sqrt(kappa) * a]
liouv_phot = liouvillian(h_phot, c_ops_phot)
```

We can then make a light-matter superoperator to address the total system of N spins and the photonic cavity by the *super_tensor* function in QuTiP. Similarly, the Liouvillian for the interaction Hamiltonian can be constructed with the *spre* and *spost* functions representing pre and post multiplication super-operators to finally construct the total Liouvillian of the combined light-matter system.

```
# identity operators
id_tls = to_super(qeye(nds))
id_phot = to_super(qeye(nphot))

# light-matter superoperator
liouv_sum = super_tensor(liouv_phot, id_tls) + super_tensor(id_phot, liouv)

# total liouvillian
h_int = g * tensor(a + a.dag(), jx)
liouv_int = -1j* spre(h_int) + 1j* spost(h_int)
liouv_tot = liouv_sum + liouv_int
```

A similar treatment is possible for any operator and we construct the total $J_z, J_+ J_-$ superoperators.

```
#total operators

jz_tot = tensor(qeye(nphot), jz)
jpjm_tot = tensor(qeye(nphot), jp*jm)
nphot_tot = tensor(a.dag()*a, qeye(nds))
```

When only the dissipation of the cavity is present, beyond a critical value of the coupling g, the steady state of the system becomes superradiant. This is visible by looking at the Wigner function of the photonic part of the density matrix, which displays two displaced lobes in the x and p plane.

```
rho_steady_state = steadystate(liouv_tot)
jz_steady_state = expect(jz_tot, rho_steady_state)
jpjm_steady_state = expect(jpjm_tot, rho_steady_state)

nphot_steady_state = expect(nphot_tot, rho_steady_state)
psi = rho_steady_state.ptrace(0)
xvec = np.linspace(-6, 6, 100)
W = wigner(psi, xvec, xvec)

wmap = wigner_cmap(W)  # Generate Wigner colormap
nrm = mpl.colors.Normalize(0, W.max())
plt.contourf(xvec, xvec, W, 100, cmap=wmap, norm=nrm)
plt.show()
```

As it has been shown in Ref. [1], the presence of dephasing suppresses the superradiant phase transition, while the presence of local emission restores it [2].

References:

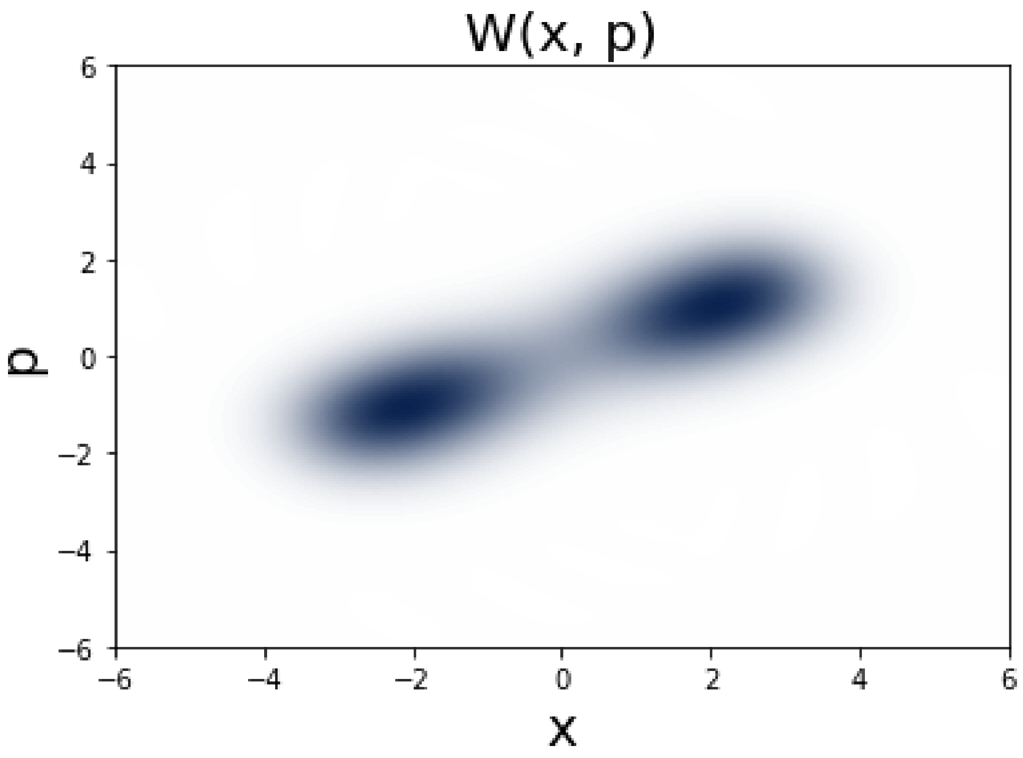
2.3 Spin Squeezing

PIQS can be used to study spin squeezing and the effect of collective and local processes on a spin squeezing Hamiltonian such as:

$$H = -i\Lambda\left(J_+^2 - J_-^2\right)$$

which evolves under the dynamics given by:

$$\dot{\rho} = -\frac{i}{\hbar}[H,\rho] + \frac{\gamma_{\text{CE}}}{2}\mathcal{L}_{J_-} + \frac{\gamma_{\text{E}}}{2}\sum_{n=1}^{N}\mathcal{L}_{J_{-,n}}[\rho].$$



In [1] it has been shown that the collective emmission ($\gamma_{\rm CE}$) affects the spin squeezing in a system in a different way than the homogeneous local emission ($\gamma_{\rm E}$). In PIQS, we can study these effects easily by adding these rates to an ensemble constructed as a *Dicke* object.

```python
from qutip import *
from piqs import *
import matplotlib.pyplot as plt

# general parameters
N = 20
nds = num_dicke_states(N)
[jx, jy, jz] = jspin(N)
jp, jm = jspin(N, "+"), jspin(N, "-")
jpjm = jp*jm

lam = 1
# spin hamiltonian
h = -1j*lam*(jp**2-jm**2)

gamma = 0.2

# Ensemble with collective emission only
ensemble_ce = Dicke(N=N, hamiltonian=h, collective_emission=gamma)

# Ensemble with local emission only
ensemble_le = Dicke(N=N, hamiltonian=h, emission=gamma)

# Build the Liouvillians for both ensembles
liouv_collective = ensemble_ce.liouvillian()
liouv_local = ensemble_le.liouvillian()
```

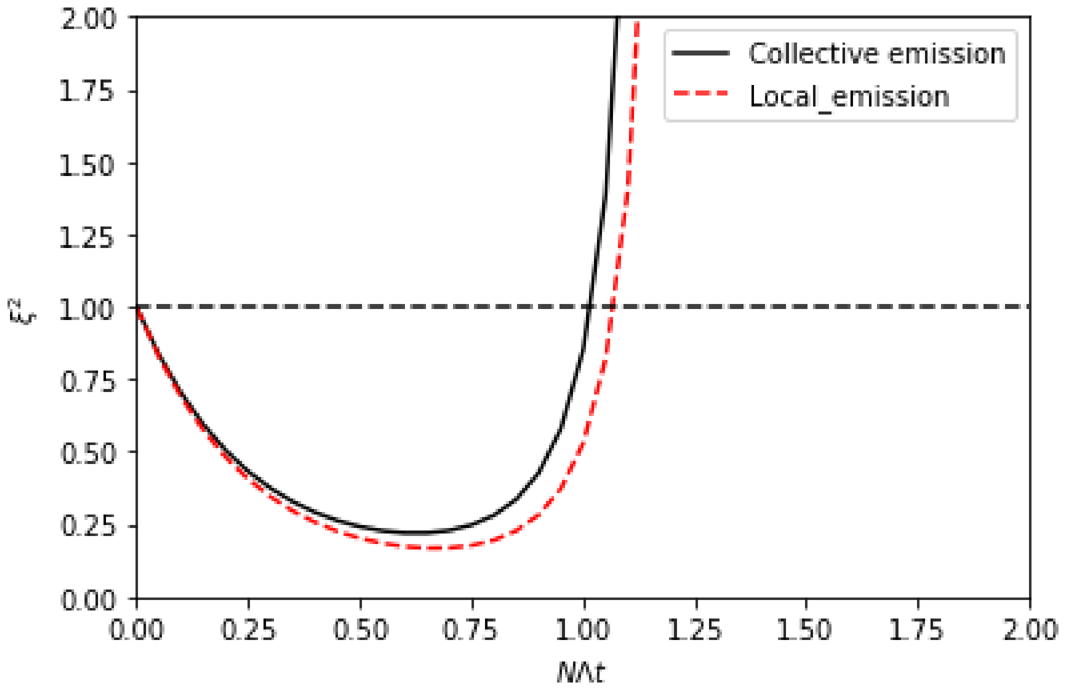
Once we have defined our ensembles and constructed their Liouvillians, we can plot the time evolution of the spin squeezing parameter given by $\xi^2 = \frac{N\langle\Delta J_y^2\rangle}{\langle J_z\rangle^2}$ starting from any initial state.

```
# set initial state for spins (Dicke basis)
rho0 = dicke(N, 10, 10)
t = np.linspace(0, 2.5, 1000)

result_collective = mesolve(liouv_collective, excited(N), t, [],
                  e_ops = [jz, jy, jy**2,jz**2, jx])
result_local = mesolve(liouv_local, excited(N), t, [],
                  e_ops = [jz, jy, jy**2,jz**2, jx])

# Get the expectation values
jzt_c, jyt_c, jy2t_c, jz2t_c, jxt_c = result_collective.expect
jzt_l, jyt_l, jy2t_l, jz2t_l, jxt_l = result_local.expect

del_jy_c = jy2t_c - jyt_c**2
del_jy_l = jy2t_l - jyt_l**2

xi2_c = N * del_jy_c/(jzt_c**2 + jxt_c**2)
xi2_l = N * del_jy_l/(jzt_l**2 + jxt_l**2)

# Generate the plots
plt.plot(t*N*lam, xi2_c, 'k-', label="Collective emission")
plt.plot(t*N*lam, xi2_l, 'r--', label="Local_emission")
plt.plot(t*N*lam, 1+0*t, '--k')
plt.ylabel(r'$\xi^2$')
plt.xlabel(r'$ N \Lambda t$')
plt.legend()

plt.xlim([0, 2])
plt.ylim([0, 2])
plt.show()
```

References:

Table 1: PIQS functions

Operators	Command	Description
Collective spin Jx	`jspin(N, "x")`	The collective spin operator Jx. Requires N number of TLS
Collective spin J+	`jspin(N, "+")`	The collective spin operator J+.
Collective spin J-	`jspin(N, "-")`	The collective spin operator Jz.
Collective spin Jz in uncoupled basis	`jspin(N, "z", basis='uncoupled')`	The collective spin operator Jz in the uncoupled basis
Dicke state |j, m>	`dicke(N, j, m)`	A Dicke state given by |j, m>
Excited state in uncoupled basis	`excited(N, basis="uncoupled")`	The excited state in the uncoupled basis
GHZ state in the Dicke basis	`ghz(N)`	The GHZ state in the Dicke (default) basis for N number of TLS
Collapse operators of the ensemble	`Dicke.c_ops()`	The collapse operators for the ensemble can be called by the *c_ops* method of the dicke class.


Collective emission
Local_emission
ξ^2
$N\Lambda t$
2.00
1.75
1.50
1.25
1.00
0.75
0.50
0.25
0.00
0.00
0.25
0.50
0.75
1.00
1.25
1.50
1.75
2.00

CHAPTER THREE

DOCUMENTATION

# Developers

- Nathan Shammah

nathan.shammah@gmail.com

- Shahnawaz Ahmed

shahnawaz.ahmed95@gmail.com

# References

The code and an introductory notebook can be found in Ref. [1]. A paper detailing the theoretical aspects and illustrating many applications is in Ref. [2]. Related open-source libraries for open quantum dynamics that exploit permutational invariance are *Permutations* [3] by Peter Kirton and *PsiQuaSP* by Michael Gegg [4].

N. Shammah, S. Ahmed, N. Lambert, S. De Liberato, and F. Nori, *Phys. Rev. A* **98**, 063815 (2018). Open quantum systems with local and collective incoherent processes: Efficient numerical simulation using permutational invariance. arXiv preprint arXiv:1805.05129.