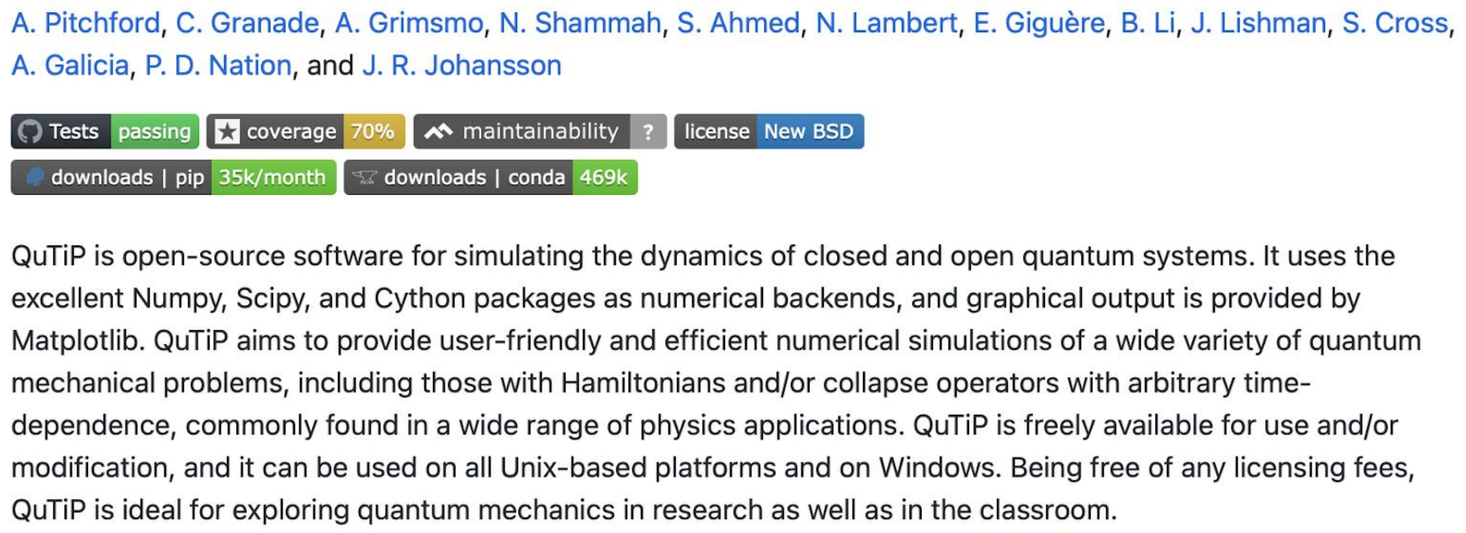
What is QuTiP

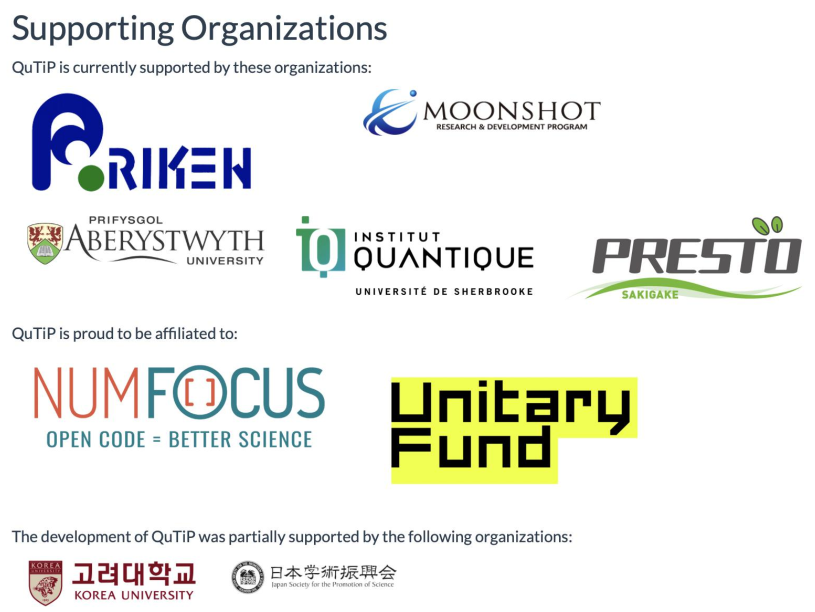
QuTiP: Quantum Toolbox in Python



A. Pitchford, C. Granade, A. Grimsmo, N. Shammah, S. Ahmed, N. Lambert, E. Giguère, B. Li, J. Lishman, S. Cross, A. Galicia, P. D. Nation, and J. R. Johansson

QuTiP is open-source software for simulating the dynamics of closed and open quantum systems. It uses the excellent Numpy, Scipy, and Cython packages as numerical backends, and graphical output is provided by Matplotlib. QuTiP aims to provide user-friendly and efficient numerical simulations of a wide variety of quantum mechanical problems, including those with Hamiltonians and/or collapse operators with arbitrary time-dependence, commonly found in a wide range of physics applications. QuTiP is freely available for use and/or modification, and it can be used on all Unix-based platforms and on Windows. Being free of any licensing fees, QuTiP is ideal for exploring quantum mechanics in research as well as in the classroom.

https://qutip.org



Supporting Organizations

QuTiP is currently supported by these organizations:

QuTiP is proud to be affiliated to:

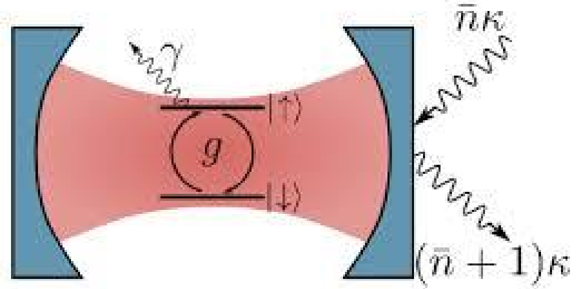
The development of QuTiP was partially supported by the following organizations:

Paper references

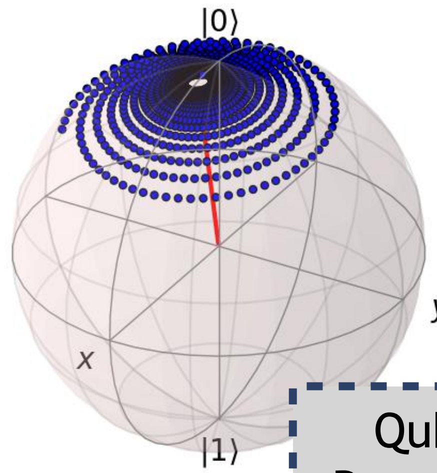
J. R. Johansson, P. D. Nation, and F. Nori, *Comp. Phys. Comm.* **183**, 1760 (2012)
J. R. Johansson, P. D. Nation, and F. Nori, *Comp. Phys. Comm.* **184**, 1234 (2013)

What can be simulated in QuTiP

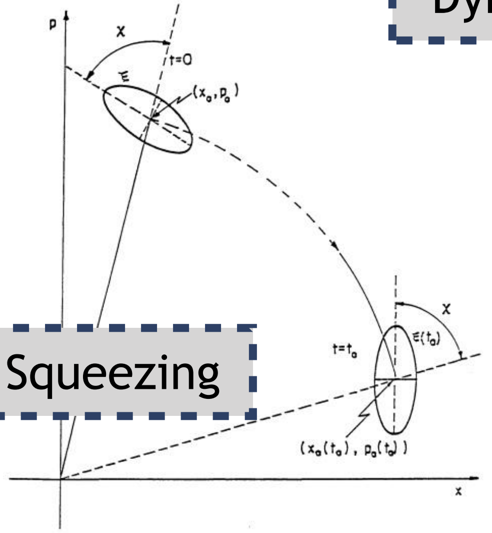
Open Quantum System



Jaynes-Cumming Model: Atom inside cavity

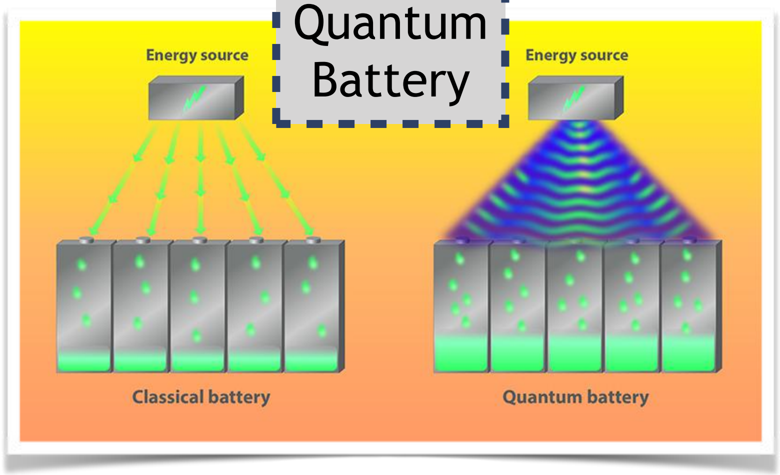


Qubit Dynamics



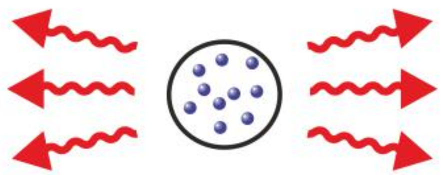
Squeezing

Sonar, Sameet, et al., *PRL* **120**, 163601 (2018)



Quantum Battery

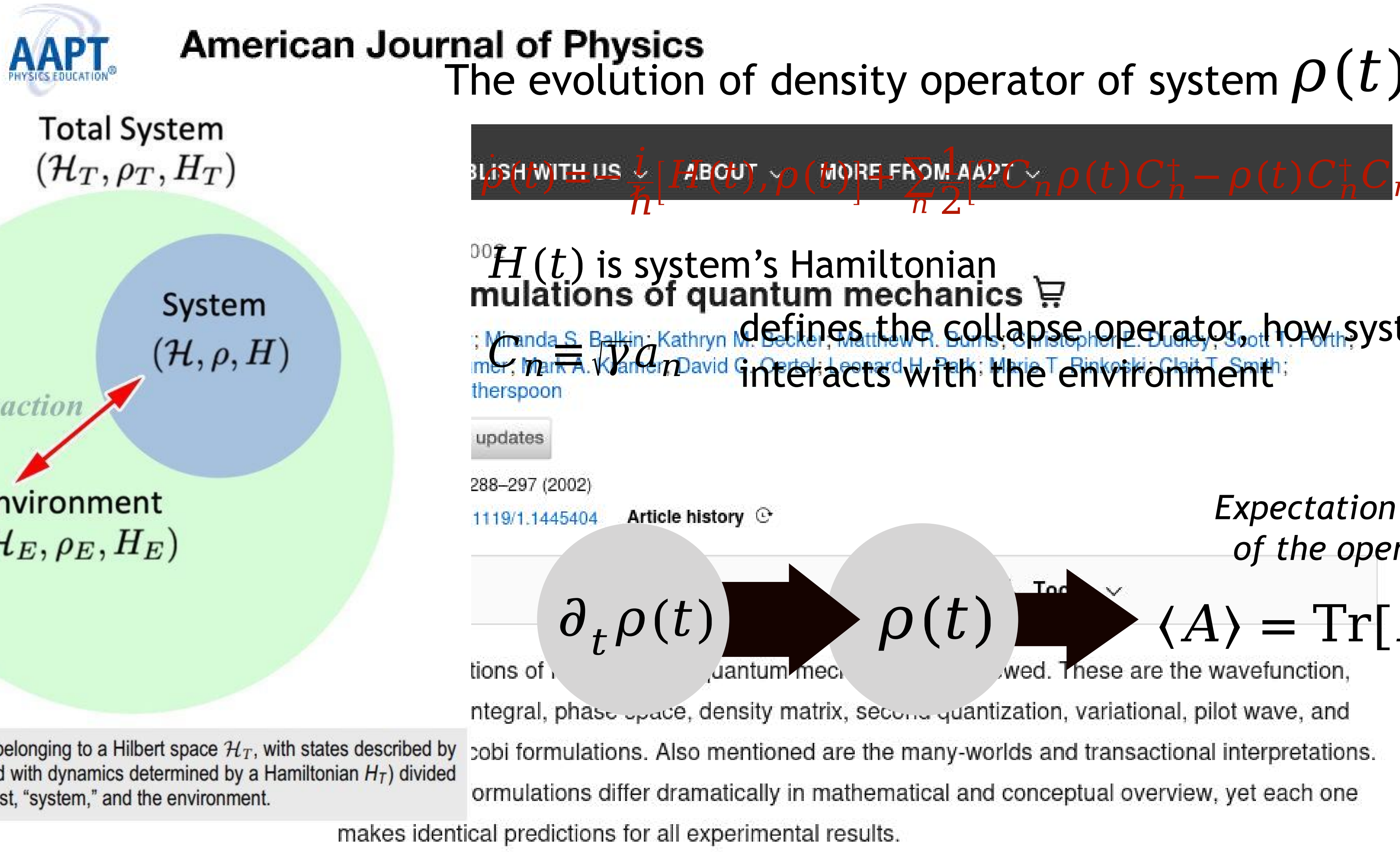
Andolina, Gian Marcello, et al., *PRB* **99**, 205437 (2019)



Dicke Superradiance

Master equation approach

Open Quantum System


American Journal of Physics

BLISH WITH US ⌄ ABOUT ⌄ MORE FROM AAPT ⌄

002

mulations of quantum mechanics

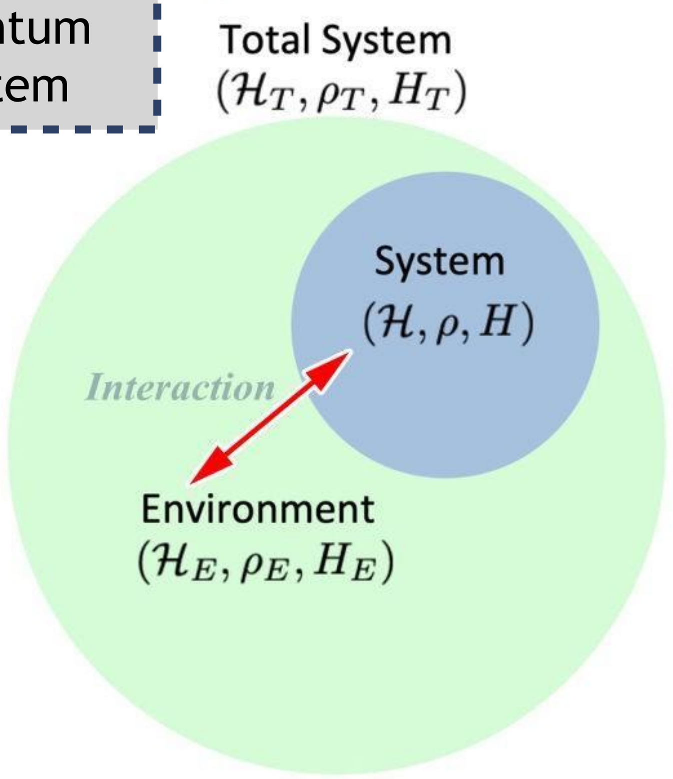
; Miranda S. Balkin; Kathryn M. Becker; Matthew R. Burns; Christopher E. Dudley; Scott T. Forth; mer; Mark A. Kramer; David C. Oertel; Leonard H. Park; Marie T. Rinkoski; Clait T. Smith; therspoon

updates

288–297 (2002)

1119/1.1445404 Article history

tions of quantum mech… …wed. These are the wavefunction, ntegral, phase space, density matrix, second quantization, variational, pilot wave, and cobi formulations. Also mentioned are the many-worlds and transactional interpretations. ormulations differ dramatically in mathematical and conceptual overview, yet each one makes identical predictions for all experimental results.



A total system (belonging to a Hilbert space $\mathcal{H}_T$, with states described by density matrices ρ_T, and with dynamics determined by a Hamiltonian H_T) divided into the system of interest, "system," and the environment.

The evolution of density operator of system $\rho(t)$

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

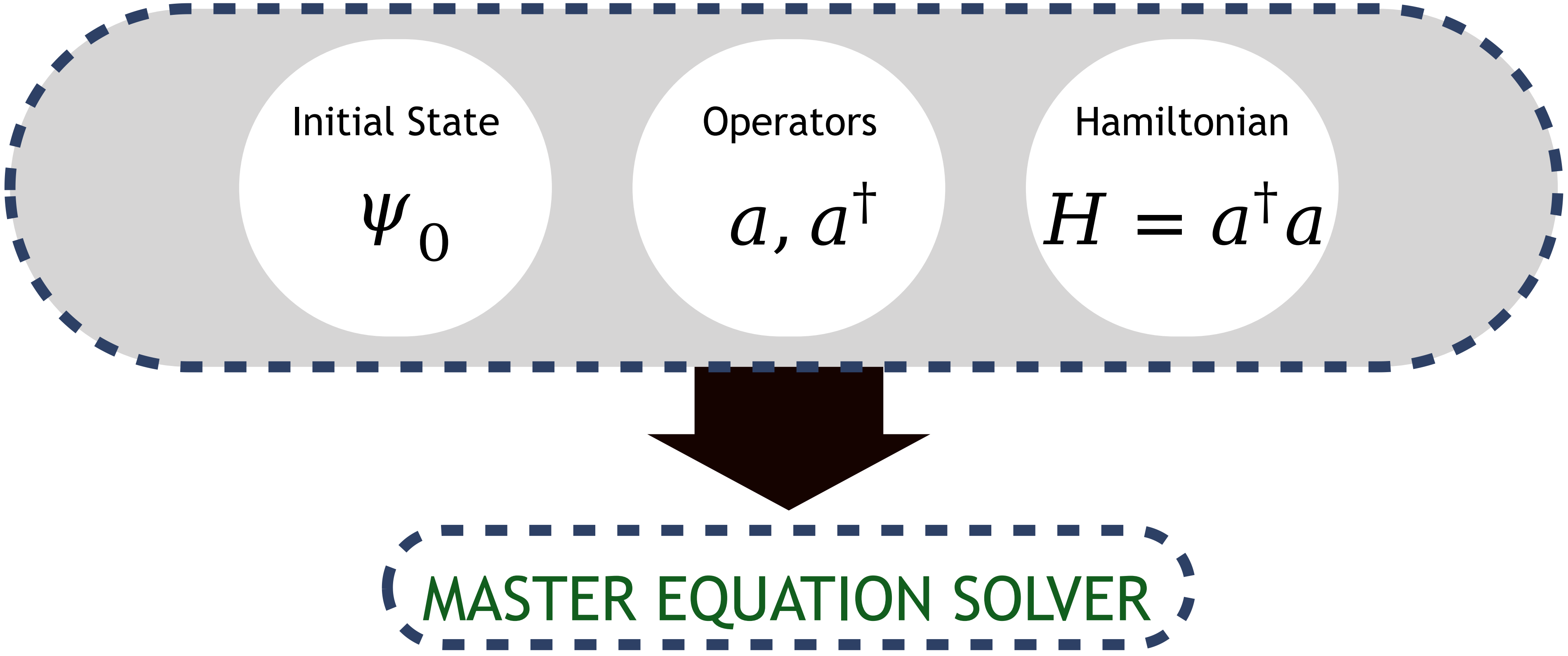
$H(t)$ is system's Hamiltonian

$C_n \equiv \sqrt{\gamma} a_n$ defines the collapse operator, how system interacts with the environment

Expectation value of the operator

$$\begin{tikzcd} \partial_t \rho(t) \arrow[r] & \rho(t) \arrow[r] & \langle A \rangle = \mathrm{Tr}[A\rho(t)] \end{tikzcd}$$

Basic input for QuTiP



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

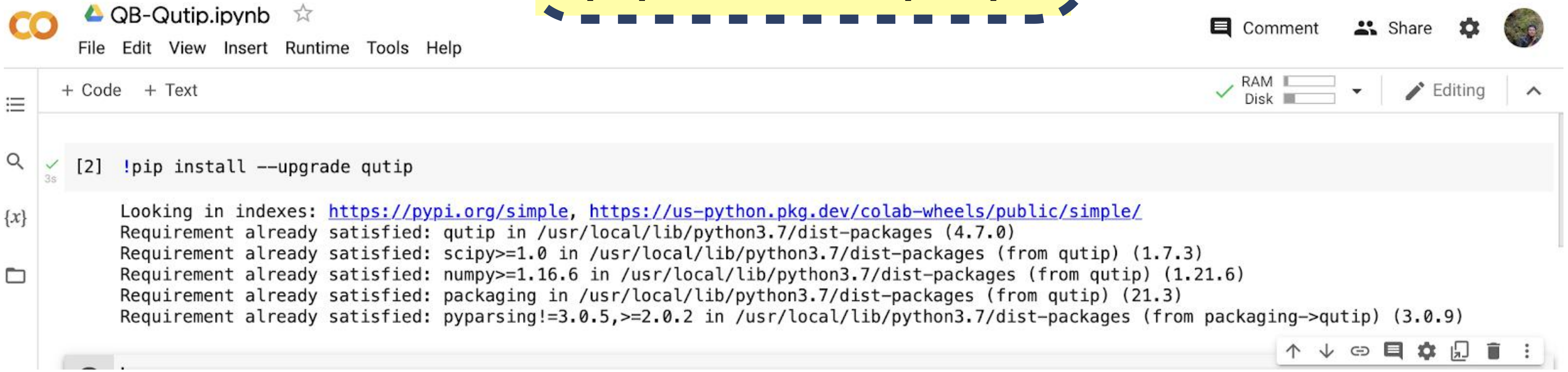
QuTiP Installation

www.anaconda.com

Python command

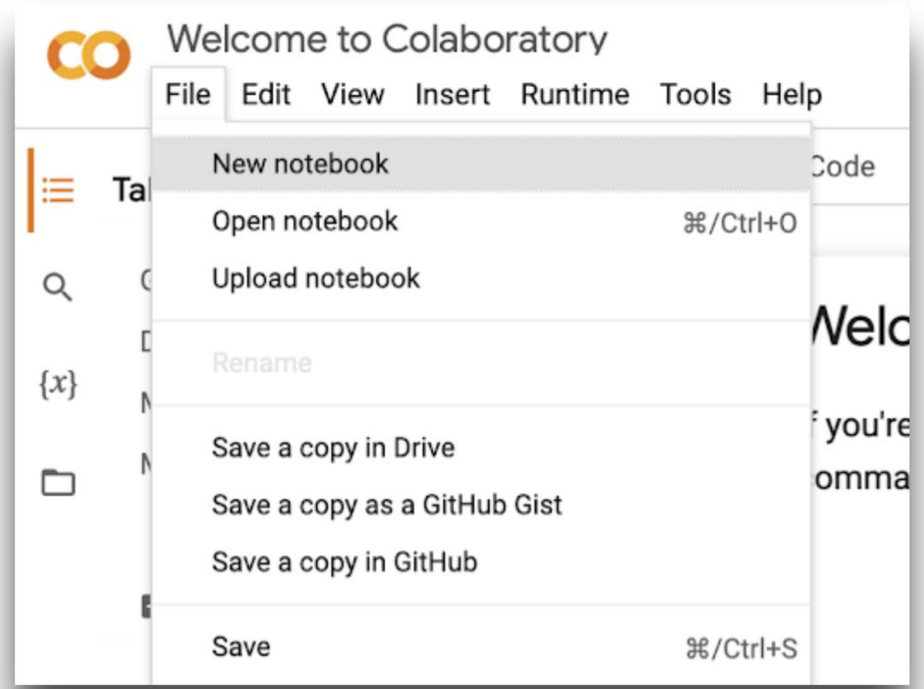
pip install qutip

colab.research.google.com



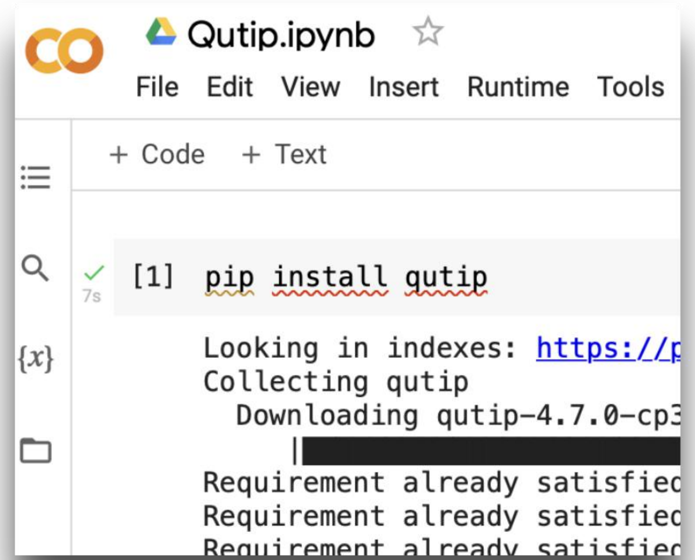
```
[2] !pip install --upgrade qutip

Looking in indexes: https://pypi.org/simple, https://us-python.pkg.dev/colab-wheels/public/simple/
Requirement already satisfied: qutip in /usr/local/lib/python3.7/dist-packages (4.7.0)
Requirement already satisfied: scipy>=1.0 in /usr/local/lib/python3.7/dist-packages (from qutip) (1.7.3)
Requirement already satisfied: numpy>=1.16.6 in /usr/local/lib/python3.7/dist-packages (from qutip) (1.21.6)
Requirement already satisfied: packaging in /usr/local/lib/python3.7/dist-packages (from qutip) (21.3)
Requirement already satisfied: pyparsing!=3.0.5,>=2.0.2 in /usr/local/lib/python3.7/dist-packages (from packaging->qutip) (3.0.9)
```

QuTiP with Google Colab



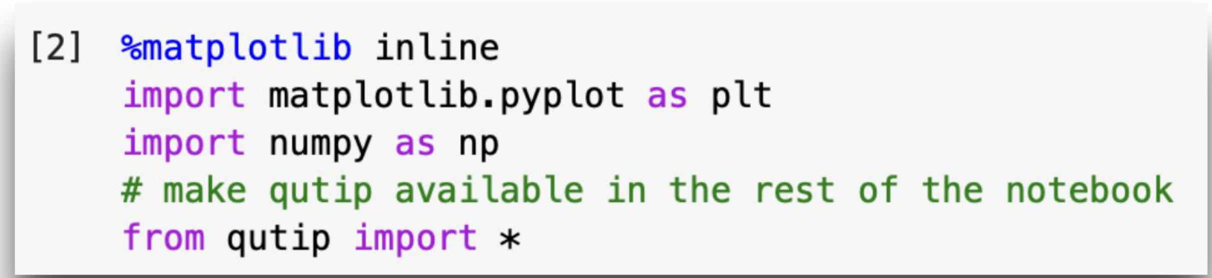
colab.research.google.com

Installation



```
[1]  pip install qutip

     Looking in indexes: https://p
     Collecting qutip
       Downloading qutip-4.7.0-cp3
     Requirement already satisfied
     Requirement already satisfied
     Requirement already satisfied
```

Calling QuTiP
and other packages

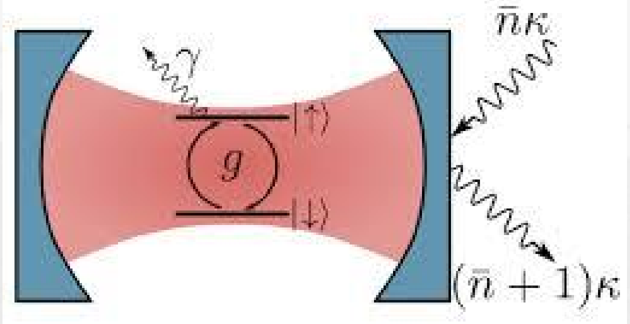


```
[2]  %matplotlib inline
     import matplotlib.pyplot as plt
     import numpy as np
     # make qutip available in the rest of the notebook
     from qutip import *
```

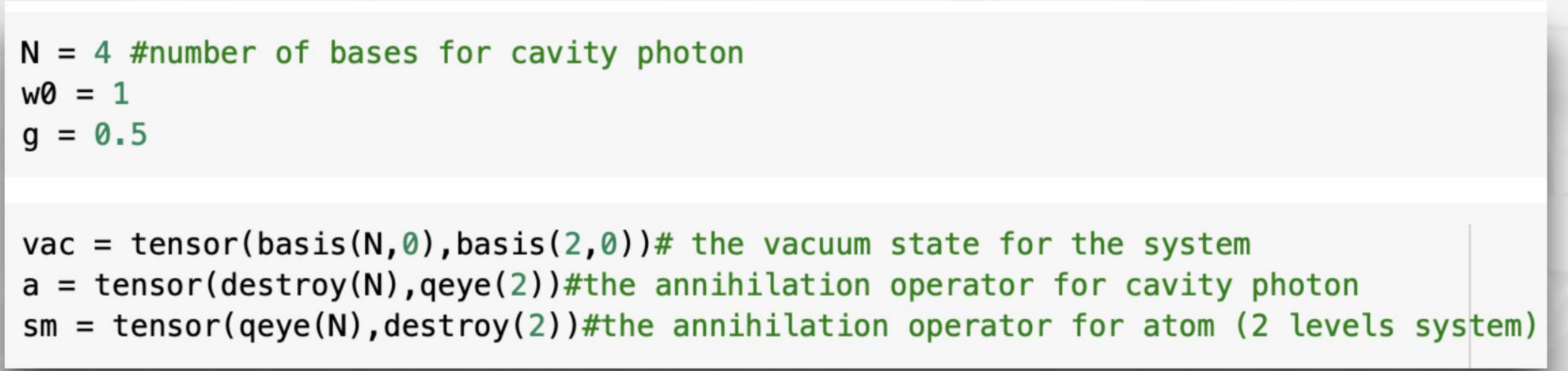
Let us simulate a quantum system!

$$H = \underbrace{\omega_0 a^\dagger a}_{\text{Cavity Photon}} + \underbrace{\omega_0 \sigma^+\sigma^-}_{\text{Atom}} + \underbrace{g(a^\dagger\sigma^- + a\sigma^+)}_{\text{Interaction}}$$

Open Quantum System



Jaynes-Cumming Model: Atom inside cavity



```
N = 4 #number of bases for cavity photon
w0 = 1
g = 0.5
```

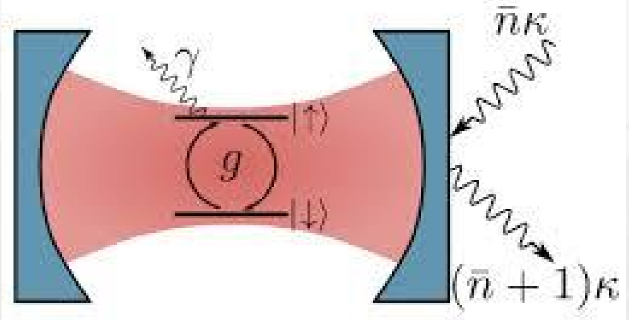
```
vac = tensor(basis(N,0),basis(2,0))# the vacuum state for the system
a = tensor(destroy(N),qeye(2))#the annihilation operator for cavity photon
sm = tensor(qeye(N),destroy(2))#the annihilation operator for atom (2 levels system)
```

Since our system is a composite system made of **cavity photon and atom**, the operators and state are given by tensor product.

Let us simulate a quantum system! (2)

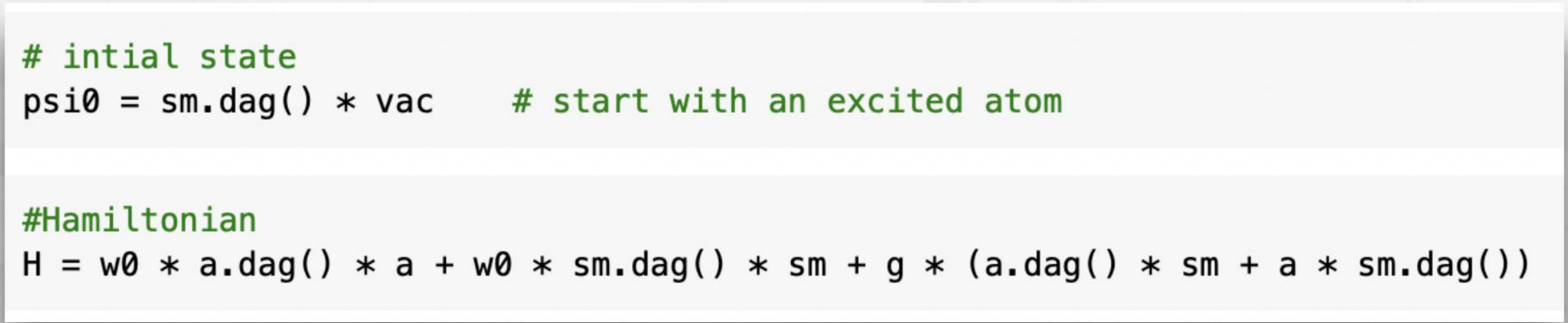
$$H = \underbrace{\omega_0 a^\dagger a}_{\text{Cavity Photon}} + \underbrace{\omega_0 \sigma^+ \sigma^-}_{\text{Atom}} + \underbrace{g(a^\dagger \sigma^- + a\sigma^+)}_{\text{Interaction}}$$

Open Quantum System

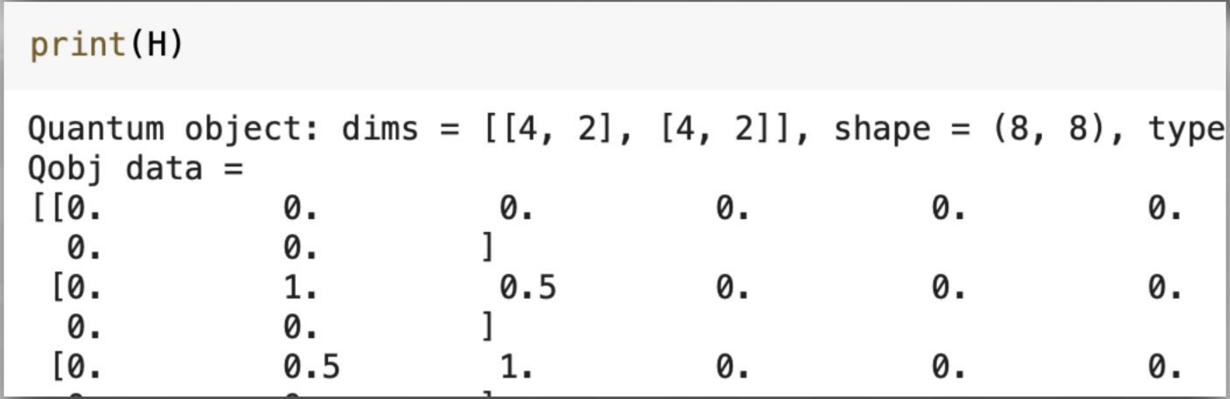


Jaynes-Cumming Model:
Atom inside cavity

H is $2N \times 2N$ matrix

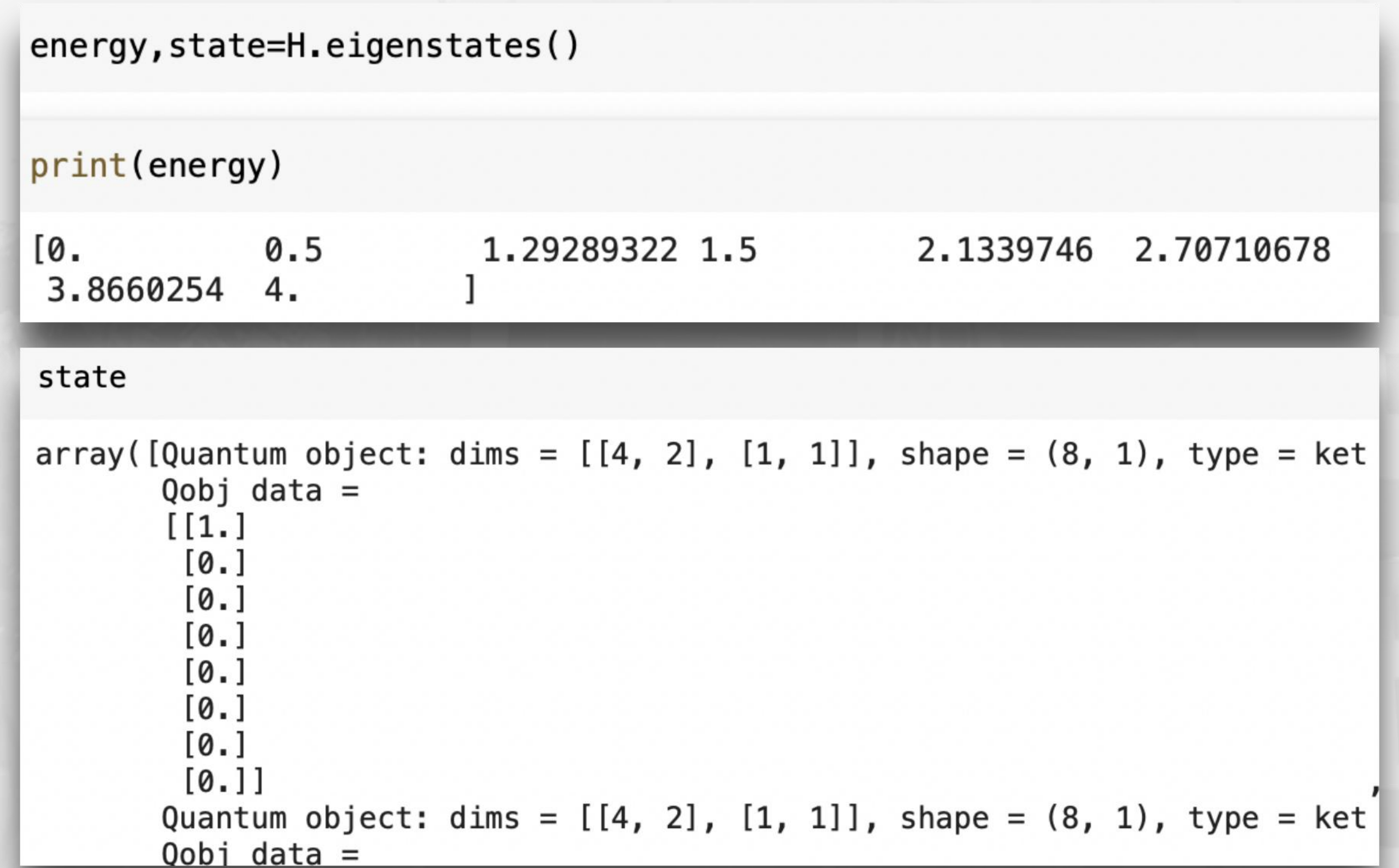


```
# intial state
psi0 = sm.dag() * vac     # start with an excited atom

#Hamiltonian
H = w0 * a.dag() * a + w0 * sm.dag() * sm + g * (a.dag() * sm + a * sm.dag())
```



```
print(H)

Quantum object: dims = [[4, 2], [4, 2]], shape = (8, 8), type
Qobj data =
[[0.          0.          0.          0.          0.          0.
  0.          0.         ]
 [0.          1.          0.5         0.          0.          0.
  0.          0.         ]
 [0.          0.5         1.          0.          0.          0.
```

Let us simulate a quantum system! (3)



```
energy,state=H.eigenstates()
```

```
print(energy)
```

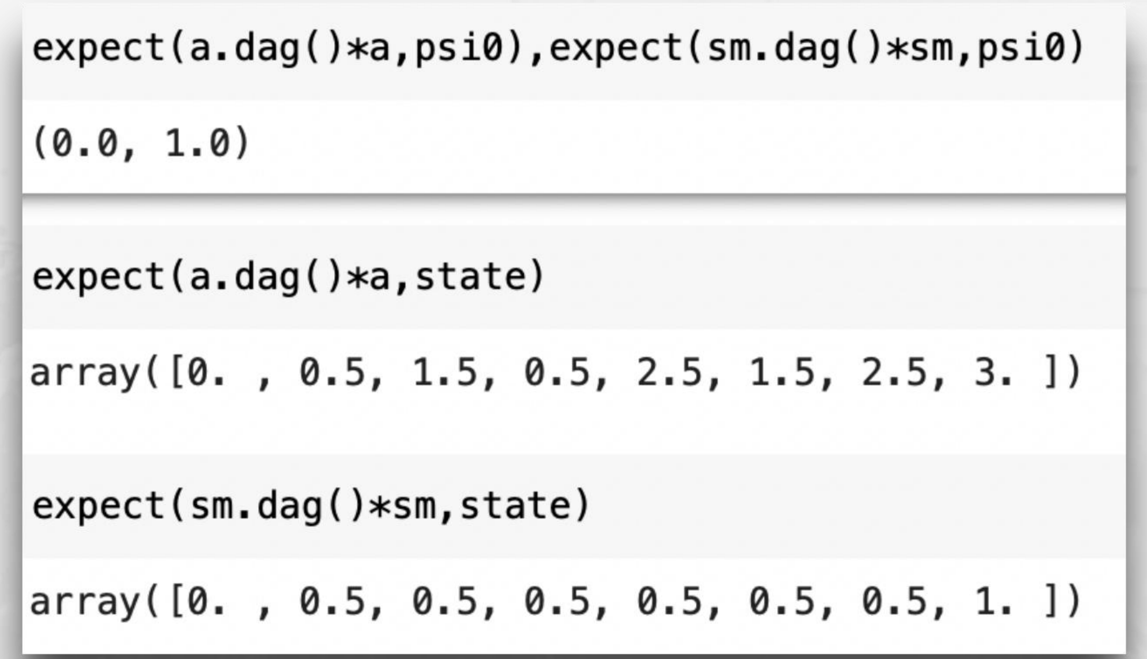
```
[0.         0.5        1.29289322 1.5        2.1339746  2.70710678
 3.8660254  4.        ]
```

```
state
```

```
array([Quantum object: dims = [[4, 2], [1, 1]], shape = (8, 1), type = ket
       Qobj data =
       [[1.]
        [0.]
        [0.]
        [0.]
        [0.]
        [0.]
        [0.]
        [0.]]
                                                                          ,
       Quantum object: dims = [[4, 2], [1, 1]], shape = (8, 1), type = ket
       Qobj data =
```

`H.eigenstates()`
gives the **energy**
and **wave function**

Let us simulate a quantum system! (4)



```
expect(a.dag()*a,psi0),expect(sm.dag()*sm,psi0)
```

```
(0.0, 1.0)
```

```
expect(a.dag()*a,state)
```

```
array([0.  , 0.5, 1.5, 0.5, 2.5, 1.5, 2.5, 3. ])
```

```
expect(sm.dag()*sm,state)
```

```
array([0.  , 0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 1. ])
```

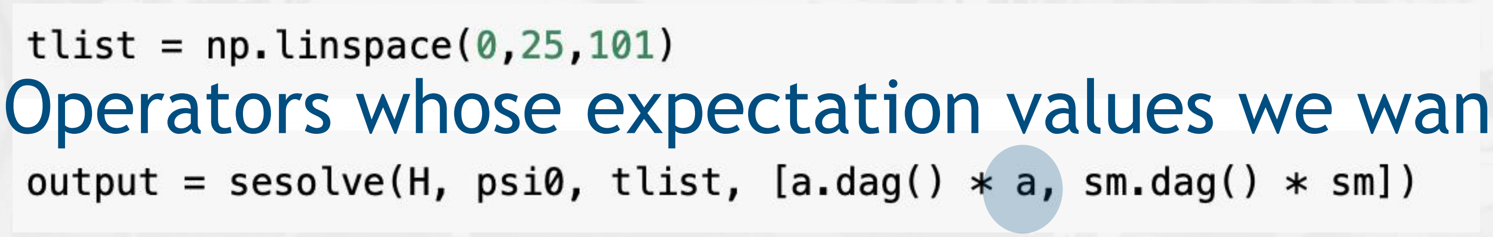
Expect $\left(\hat{O},\psi\right)$ gives expectation value of operator $\hat{O}$ for state ψ

$$\langle\hat{O}\rangle = \langle\psi\left|\hat{O}\right|\psi\rangle$$

Let us evolve the system with time!

$$\dot{\rho}(t) = -\frac{i}{\hbar}[H(t),\rho(t)]$$

$$\langle\hat{O}\rangle = \mathrm{Tr}\left(\hat{O}\rho(t)\right)$$

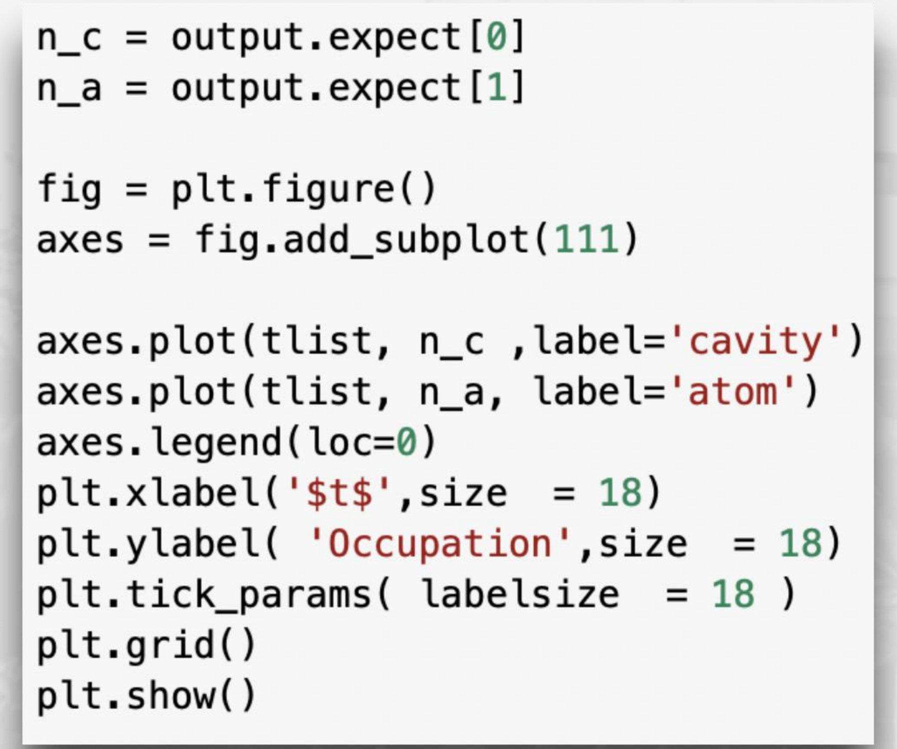


```
tlist = np.linspace(0,25,101)
```

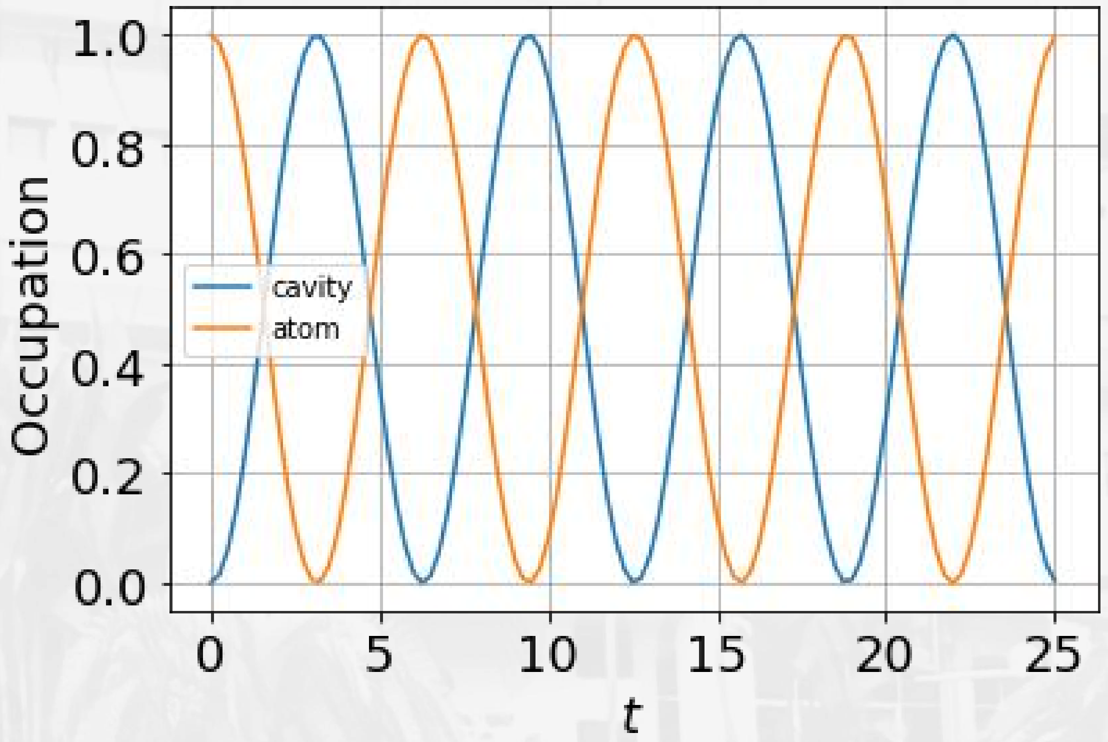
Operators whose expectation values we want to consider

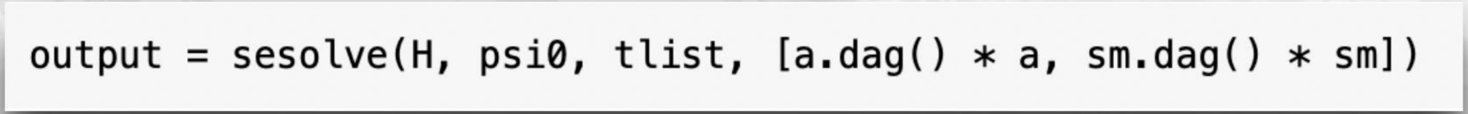
```
output = sesolve(H, psi0, tlist, [a.dag() * a, sm.dag() * sm])
```

Let us simulate a quantum system! (5)



```
n_c = output.expect[0]
n_a = output.expect[1]

fig = plt.figure()
axes = fig.add_subplot(111)

axes.plot(tlist, n_c ,label='cavity')
axes.plot(tlist, n_a, label='atom')
axes.legend(loc=0)
plt.xlabel('$t$',size  = 18)
plt.ylabel( 'Occupation',size  = 18)
plt.tick_params( labelsize  = 18 )
plt.grid()
plt.show()
```





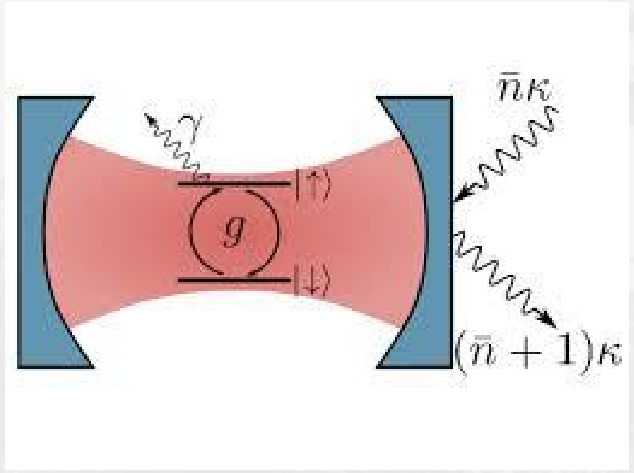
```
output = sesolve(H, psi0, tlist, [a.dag() * a, sm.dag() * sm])
```

`output.expect` gives an array of expectation values

Let us simulate a quantum system! (6)

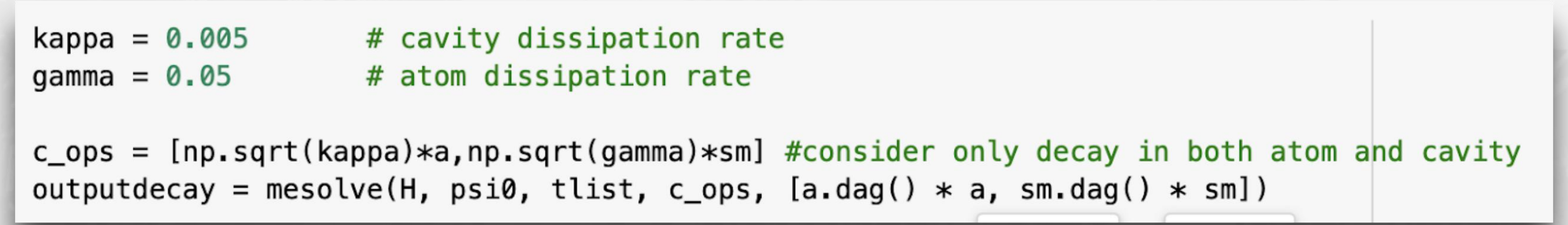
Let us **add environment**!

$$\dot{\rho}(t) = -\frac{i}{\hbar}[H(t), \rho(t)] + \sum_n \frac{1}{2}\left[2C_n\rho(t)C_n^\dagger - \rho(t)C_n^\dagger C_n - C_n^\dagger C_n \rho(t)\right]$$



Interaction with environment appears in the collapse operator C_n

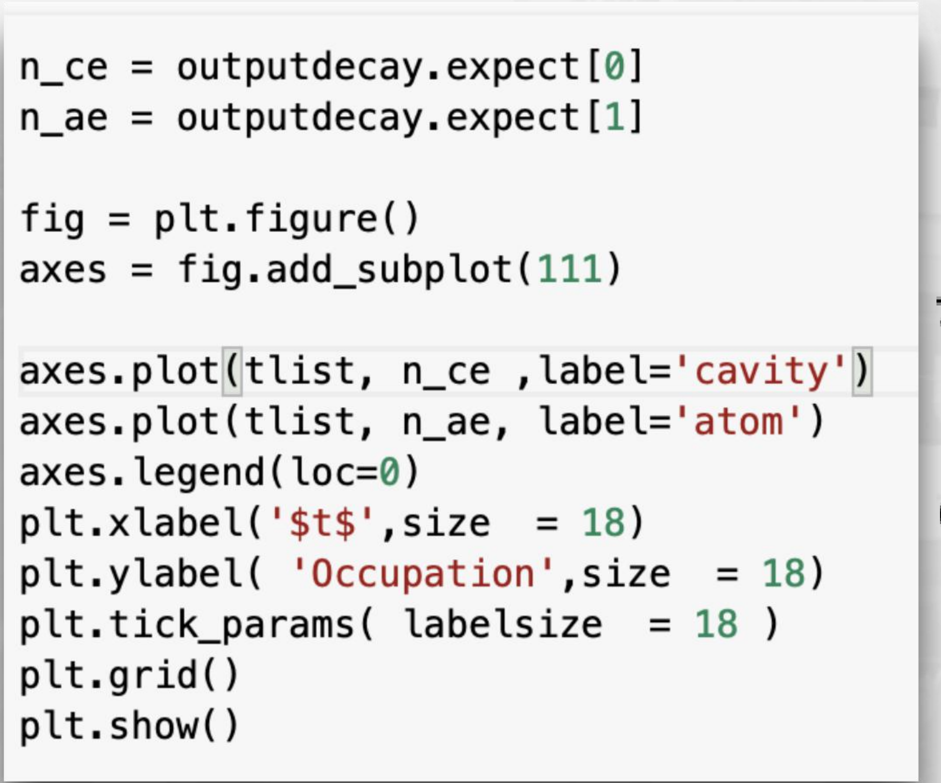
To decay to the environment, collapse operators are $\sqrt{\kappa}a$ and $\sqrt{\gamma}\sigma^-$



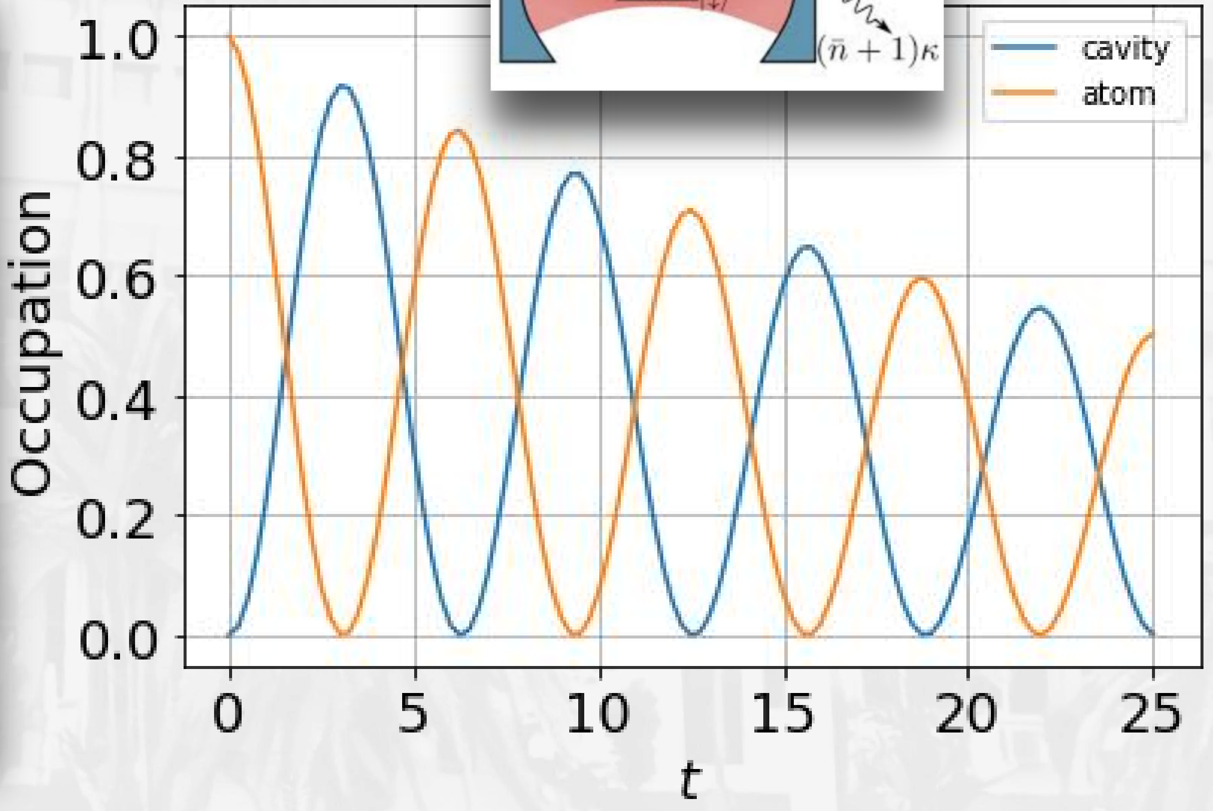
```
kappa = 0.005        # cavity dissipation rate
gamma = 0.05         # atom dissipation rate

c_ops = [np.sqrt(kappa)*a,np.sqrt(gamma)*sm] #consider only decay in both atom and cavity
outputdecay = mesolve(H, psi0, tlist, c_ops, [a.dag() * a, sm.dag() * sm])
```

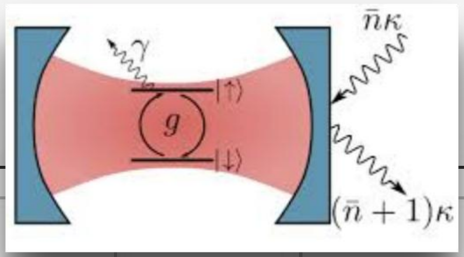
Here, we use `mesolve`, instead of `sesolve`

Let us simulate a quantum system! (7)



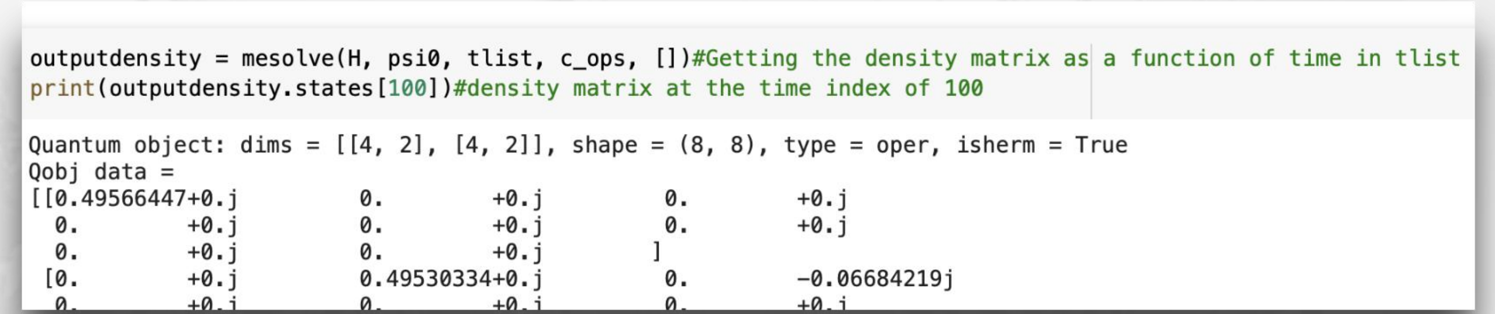
```
n_ce = outputdecay.expect[0]
n_ae = outputdecay.expect[1]

fig = plt.figure()
axes = fig.add_subplot(111)

axes.plot(tlist, n_ce ,label='cavity')
axes.plot(tlist, n_ae, label='atom')
axes.legend(loc=0)
plt.xlabel('$t$',size  = 18)
plt.ylabel( 'Occupation',size  = 18)
plt.tick_params( labelsize  = 18 )
plt.grid()
plt.show()
```





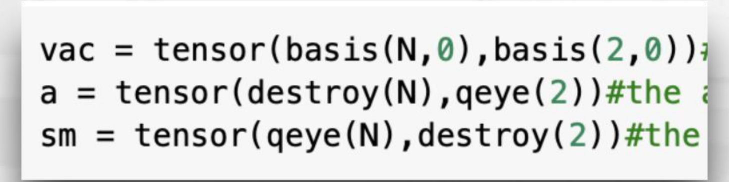
Let us simulate a quantum system! (8)

Can we obtain the density matrix $\rho(t)$? Yes!



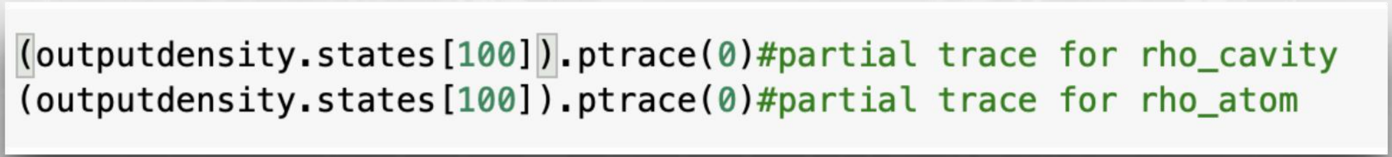
```
outputdensity = mesolve(H, psi0, tlist, c_ops, [])#Getting the density matrix as a function of time in tlist
print(outputdensity.states[100])#density matrix at the time index of 100
```

```
Quantum object: dims = [[4, 2], [4, 2]], shape = (8, 8), type = oper, isherm = True
Qobj data =
[[0.49566447+0.j          0.        +0.j          0.        +0.j
  0.        +0.j          0.        +0.j          0.        +0.j
  0.        +0.j          0.        +0.j        ]
 [0.        +0.j          0.49530334+0.j          0.        -0.06684219j
  0.        +0.j          0.        +0.j          0.        +0.j
```



```
vac = tensor(basis(N,0),basis(2,0))#
a = tensor(destroy(N),qeye(2))#the
sm = tensor(qeye(N),destroy(2))#the
```

We can do partial trace



```
(outputdensity.states[100]).ptrace(0)#partial trace for rho_cavity
(outputdensity.states[100]).ptrace(0)#partial trace for rho_atom
```

$$\rho_{\text{cav}} = \text{Tr}_{\text{atom}}[\rho(t)]$$

$$\rho_{\text{atom}} = \text{Tr}_{\text{cav}}[\rho(t)]$$

Plotting energy levels

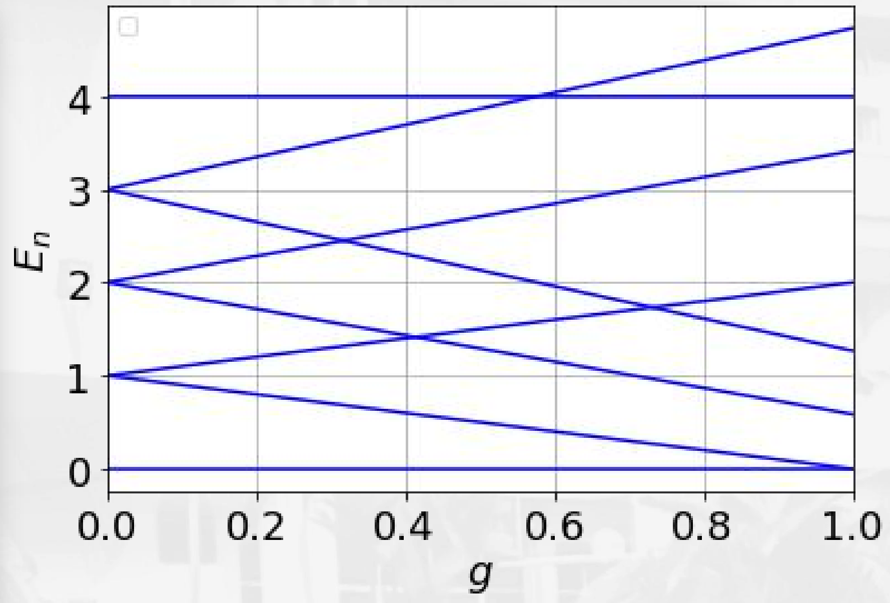
Jaynes-Cumming Model:
Atom inside cavity

$$H = \omega_0 a^\dagger a + \omega_0 \sigma^+\sigma^- + g(a^\dagger\sigma^- + a\sigma^+)$$

```
def energy(gg):
    i  = 0
    en = np.zeros((len(gg),N*2))
    for k in gg:
        # evaluate the Hamiltonian
        H = w0 * a.dag() * a + w0 * sm.dag() * sm + k * (a.dag() * sm + a * sm.dag())
        ener, ekets = H.eigenstates()
        en[i,:] = np.real(ener)
        i += 1

    return en
```

```
gg = np.linspace(0,1,100)
output_energy = energy(gg)
```

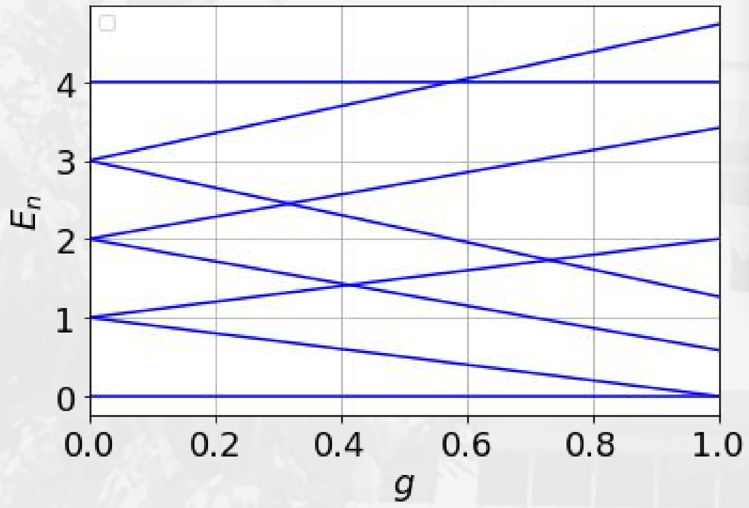
```
fig = plt.figure()
axes = fig.add_subplot(111)
for n in [0,1,2,3,4,5,6,7]:
     axes.plot(gg, output_energy[:,n] , 'b')
axes.legend(loc=0)
plt.xlabel('$g$',size  = 18)
plt.ylabel( '$E_n$',size  = 18)
plt.tick_params( labelsize  = 18 )
plt.grid()
plt.xlim([0,1])
plt.show()
```



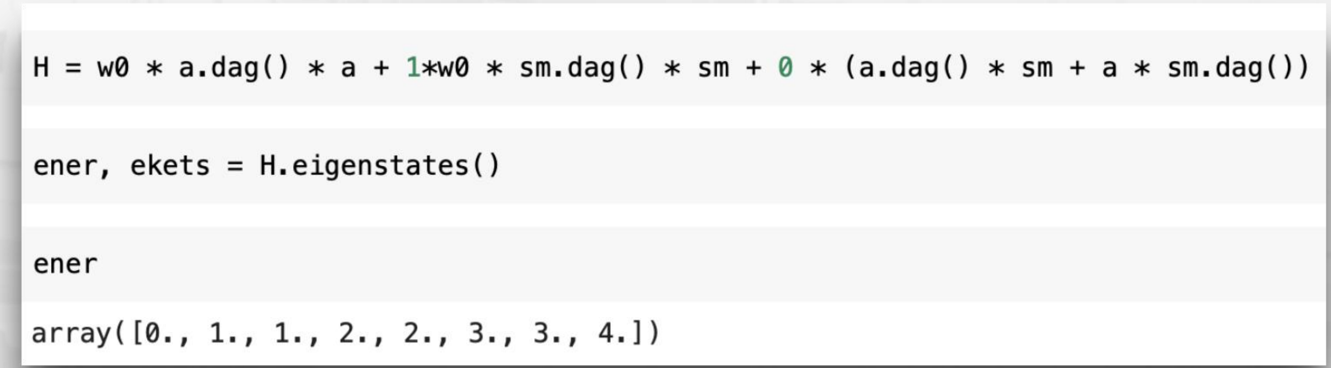
Plotting energy levels (2)

How to understand the eigenstate of each level?

$$H = \omega_0 a^\dagger a + \omega_0 \sigma^+\sigma^- + g\,(a^\dagger\sigma^- + a\sigma^+)$$



Takes first $g = 0$

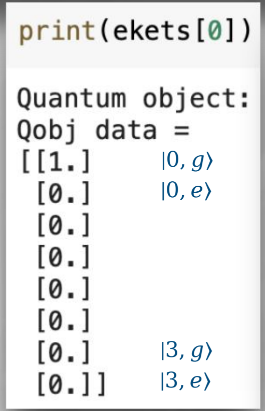


```
H = w0 * a.dag() * a + 1*w0 * sm.dag() * sm + 0 * (a.dag() * sm + a * sm.dag())
```

```
ener, ekets = H.eigenstates()
```

```
ener
```

```
array([0., 1., 1., 2., 2., 3., 3., 4.])
```

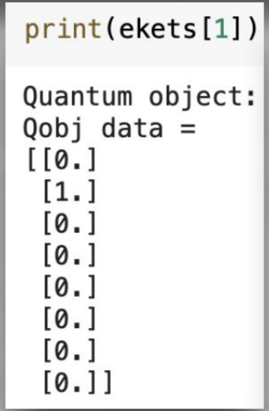


```
print(ekets[0])
```

```
Quantum object:
Qobj data =
[[1.]      |0, g⟩
 [0.]      |0, e⟩
 [0.]
 [0.]
 [0.]
 [0.]
 [0.]      |3, g⟩
 [0.]]     |3, e⟩
```

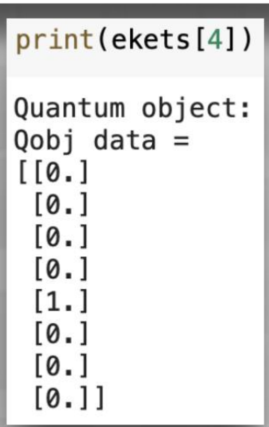
$n = 0$ {

$n = 3$ {



```
print(ekets[1])
```

```
Quantum object:
Qobj data =
[[0.]
 [1.]
 [0.]
 [0.]
 [0.]
 [0.]
 [0.]
 [0.]]
```



```
print(ekets[4])
```

```
Quantum object:
Qobj data =
[[0.]
 [0.]
 [0.]
 [0.]
 [1.]
 [0.]
 [0.]
 [0.]]
```

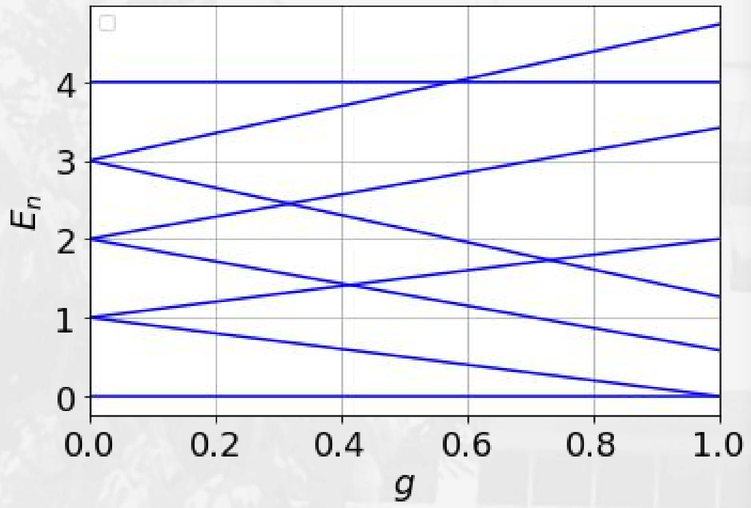
$|0, g\rangle$ $|0, e\rangle$ $|2, g\rangle$

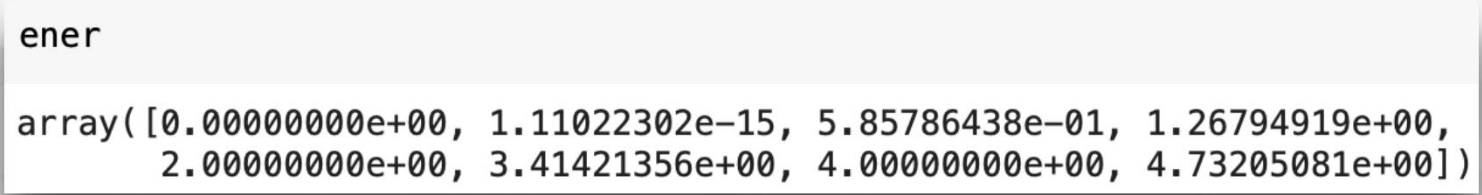
Plotting energy levels (3)

How to understand the eigenstate of each level?

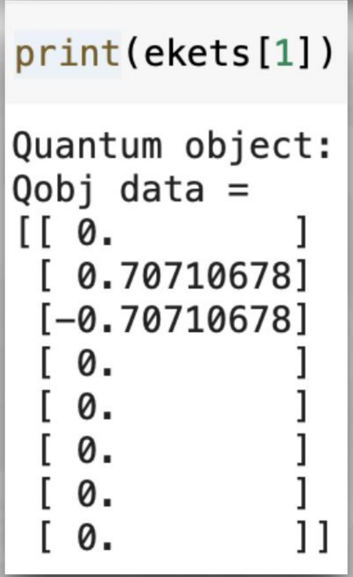
Now, we $g = 1$

$H = \omega_0 a^\dagger a + \omega_0 \sigma^+ \sigma^- + g\ (a^\dagger \sigma^- + a\sigma^+)$

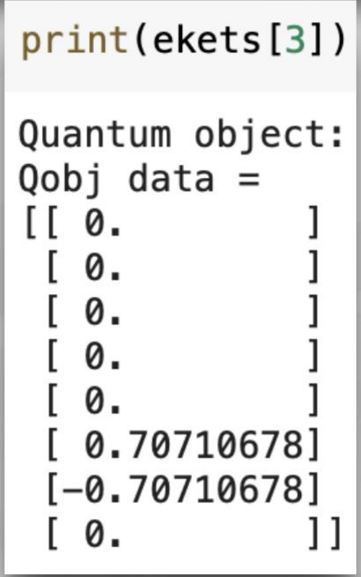




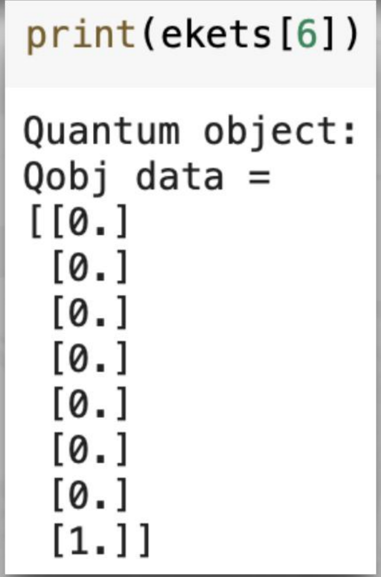
```
ener

array([0.00000000e+00, 1.11022302e-15, 5.85786438e-01, 1.26794919e+00,
       2.00000000e+00, 3.41421356e+00, 4.00000000e+00, 4.73205081e+00])
```



```
print(ekets[1])

Quantum object:
Qobj data =
[[ 0.        ]
 [ 0.70710678]
 [-0.70710678]
 [ 0.        ]
 [ 0.        ]
 [ 0.        ]
 [ 0.        ]
 [ 0.        ]]
```



```
print(ekets[3])

Quantum object:
Qobj data =
[[ 0.        ]
 [ 0.        ]
 [ 0.        ]
 [ 0.        ]
 [ 0.        ]
 [ 0.70710678]
 [-0.70710678]
 [ 0.        ]]
```



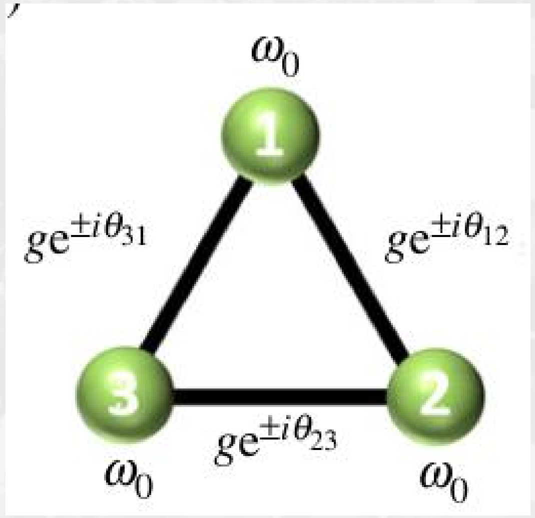
```
print(ekets[6])

Quantum object:
Qobj data =
[[0.]
 [0.]
 [0.]
 [0.]
 [0.]
 [0.]
 [0.]
 [1.]]
```

$\frac{1}{\sqrt{2}}(|0,e\rangle - |1,g\rangle)$ $\frac{1}{\sqrt{2}}(|2,e\rangle - |3,g\rangle)$ $|3,e\rangle$

Exercise (1)

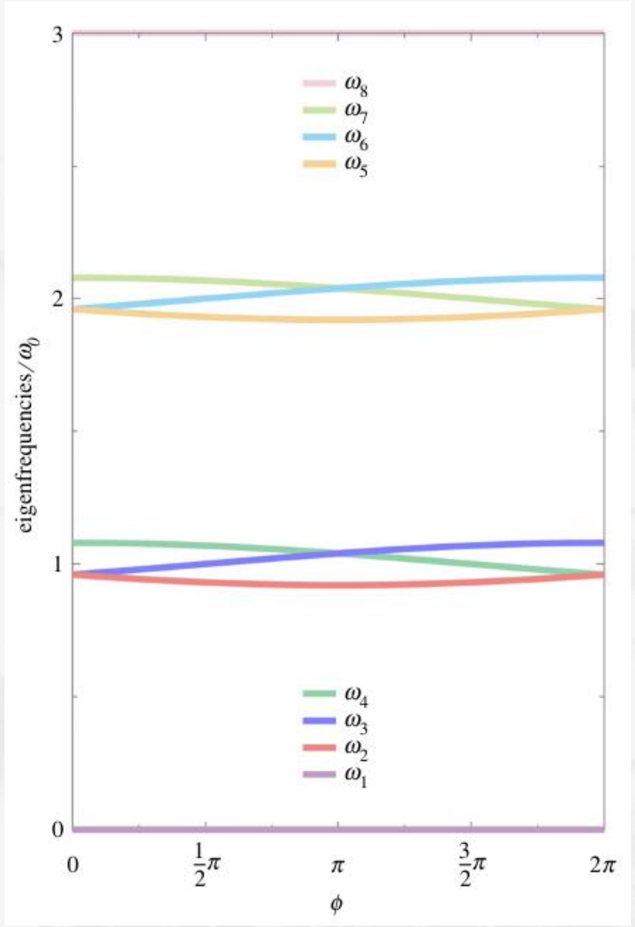
Two level system in triangular structure

$$\hat{H} = \omega_0\left(\sigma_1{}^\dagger\sigma_1 + \sigma_2{}^\dagger\sigma_2 + \sigma_3{}^\dagger\sigma_3\right) + g\left(e^{i\theta_{12}}\sigma_1{}^\dagger\sigma_2 + e^{i\theta_{23}}\sigma_2{}^\dagger\sigma_3 + e^{i\theta_{31}}\sigma_3{}^\dagger\sigma_1 + h.c.\right)$$



$$\phi = \theta_{12} + \theta_{23} + \theta_{31}$$

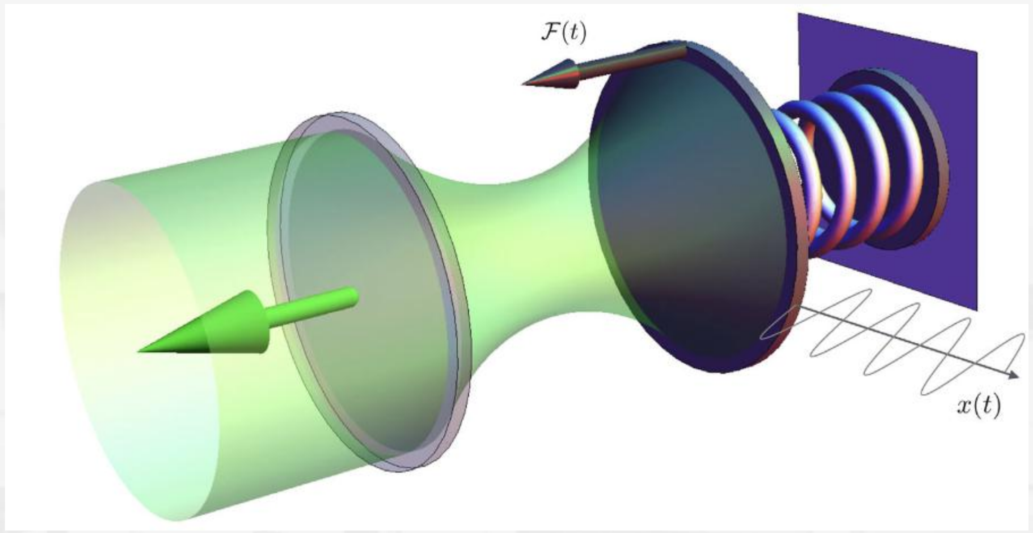
$$g = \frac{\omega_0}{25}$$

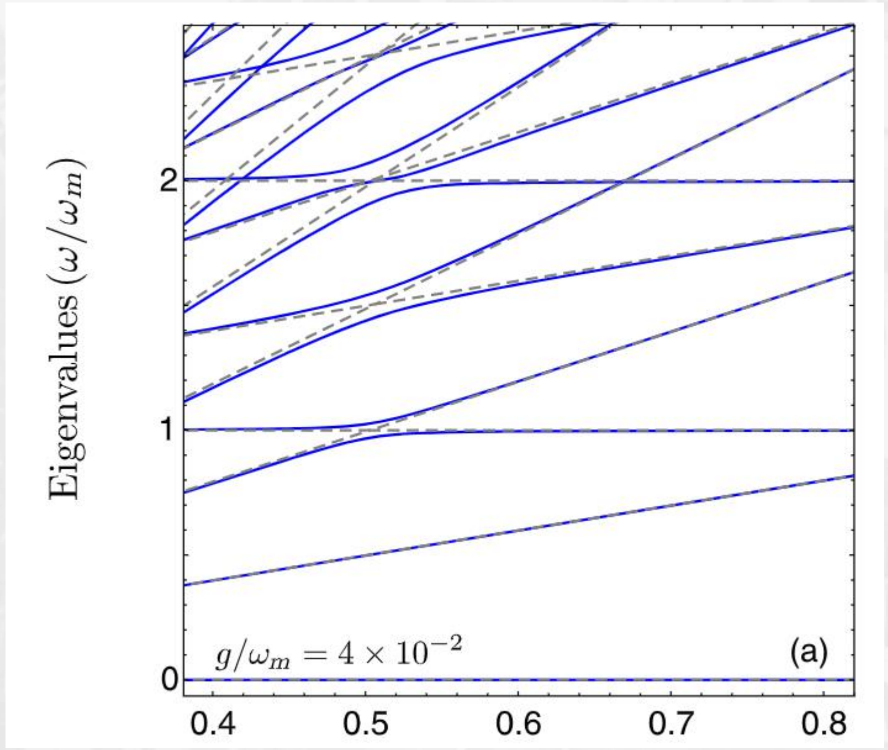


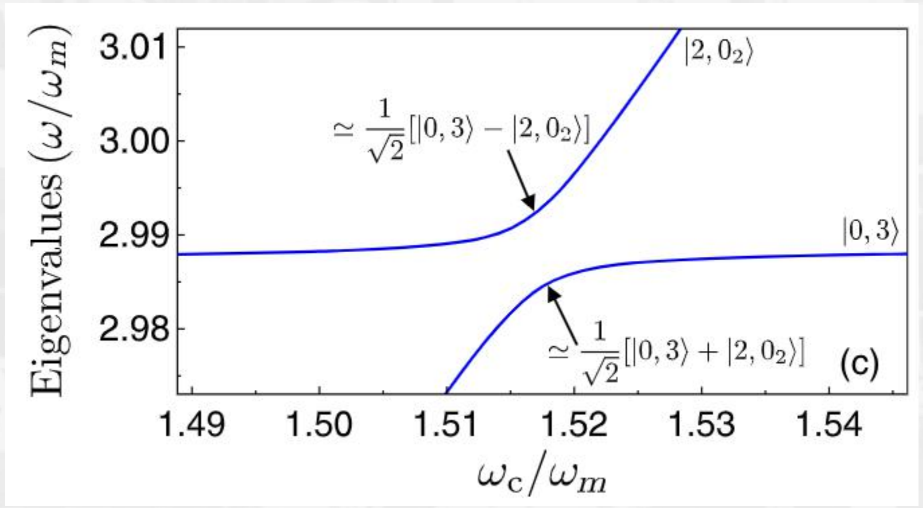
Downing, C. A., & Zueco, D. (2021). Non-reciprocal population dynamics in a quantum trimer. *Proceedings of the Royal Society A: Mathematical, Physical and Engineering Sciences*, **477**(2255).

Exercise (2)

Dynamical Casimir effect



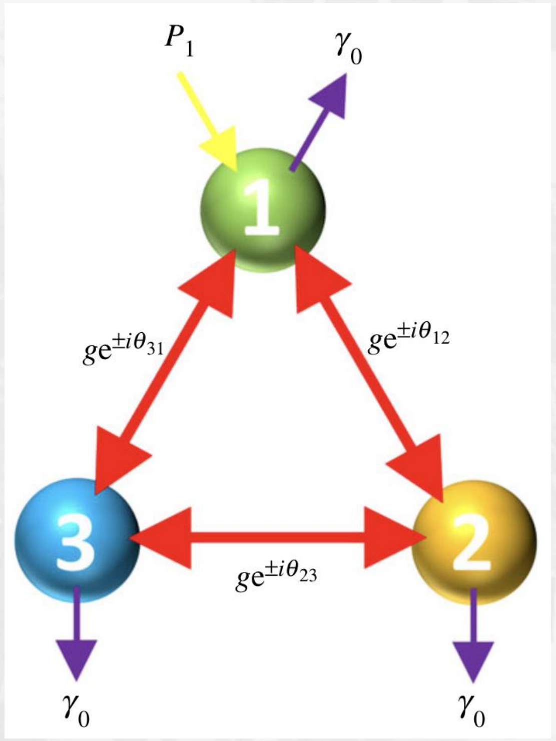




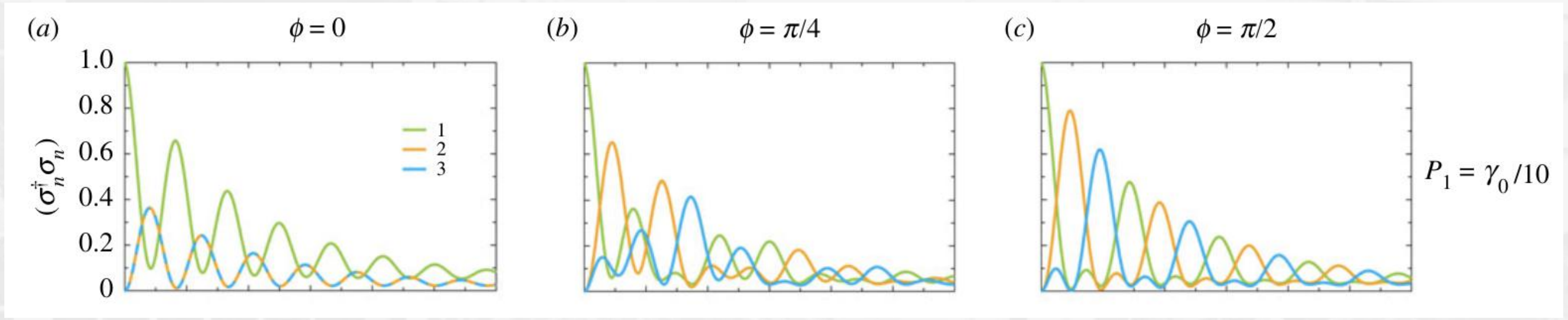
Macrì, V., Ridolfo, A., di Stefano, O., Kockum, A. F., Nori, F., & Savasta, S. (2018). Nonperturbative Dynamical Casimir Effect in Optomechanical Systems: Vacuum Casimir-Rabi Splittings. *Physical Review X*, 8(1), 11031

Exercise (3)

Two level system in triangular structure



$$\hat{H} = \omega_0\left(\sigma_1{}^\dagger\sigma_1 + \sigma_2{}^\dagger\sigma_2 + \sigma_3{}^\dagger\sigma_3\right) + g(e^{i\theta_{12}}\sigma_1{}^\dagger\sigma_2 + e^{i\theta_{23}}\sigma_2{}^\dagger\sigma_3 + e^{i\theta_{31}}\sigma_3{}^\dagger\sigma_1 + h.c.)$$



$g = 5\gamma_0$ Initial state: Particle 1 is excited

Downing, C. A., & Zueco, D. (2021). Non-reciprocal population dynamics in a quantum trimer. *Proceedings of the Royal Society A: Mathematical, Physical and Engineering Sciences*, **477**(2255).

Quantum Battery

“Quantum-mechanical systems that can store energy”

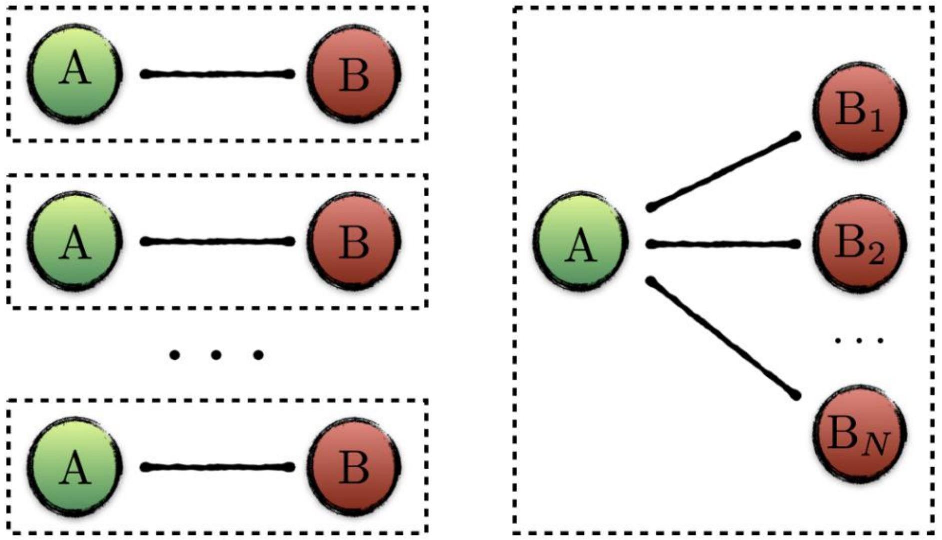


FIG. 1. A sketch of the parallel (left) versus collective (right) charging schemes.

Collective charging gives faster charging which scales with number of *B*

Andolina, G. M, et al. (2019). Quantum versus classical many-body batteries. *Phys. Rev. B*, 99(20), 1–7.

Quantum Battery: Spin Battery

N-Qubits of **charger**

$$H_A = \omega_0 \left(J_z^{(a)} + \frac{N}{2} \right)$$

Interaction between A and B

$$H_1 = 4g \left(J_x^{(a)} J_x^{(b)} + J_y^{(a)} J_y^{(b)} \right)$$

N-Qubits of **battery**

$$H_B = \omega_0 \left(J_z^{(b)} + \frac{N}{2} \right)$$

Collective spin operator with length $N/2$

$$J_z = \sum_{i=1}^{N} \sigma_z^{(i)}$$

Spin of each qubit

Figure of merit:

$$E_{\mathrm{B}}^{(N)}(\tau) \equiv \mathrm{Tr}\left[H_{\mathrm{B}} \rho_{\mathrm{B}}(\tau) \right]$$

Total energy stored at batteries

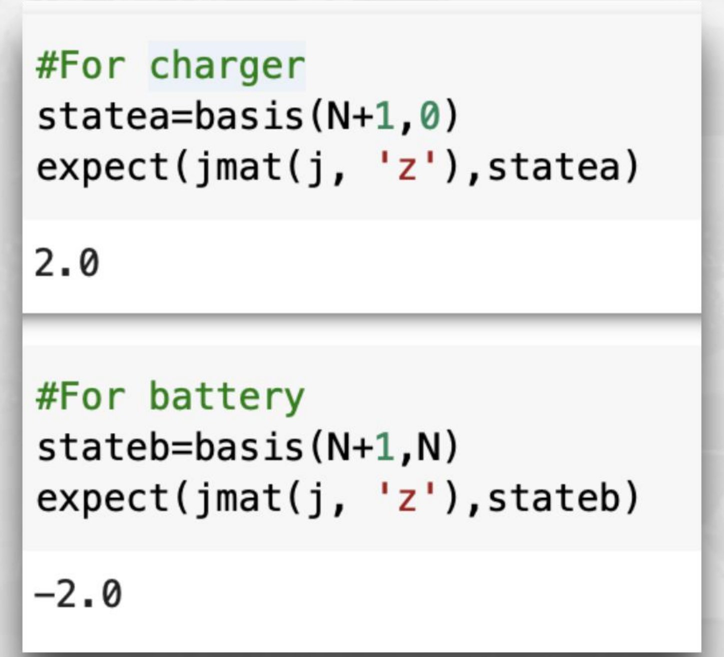
$$P_{\mathrm{B}}^{(N)}(\tau) = \frac{E_{\mathrm{B}}^{(N)}}{\tau}$$

Charging power

Let us simulate it with QuTiP!

Quantum Battery: Spin Battery (2)

	Real	In QuTiP
Initial State	All **chargers** are **excited states**	All spins in **A** are **spin-up**
	All **batteries** are **ground states**	All spins in **B** are **spin-down**



```
#For charger
statea=basis(N+1,0)
expect(jmat(j, 'z'),statea)
```

```
2.0
```

```
#For battery
stateb=basis(N+1,N)
expect(jmat(j, 'z'),stateb)
```

```
-2.0
```

Collective spin operator with length $N/2$

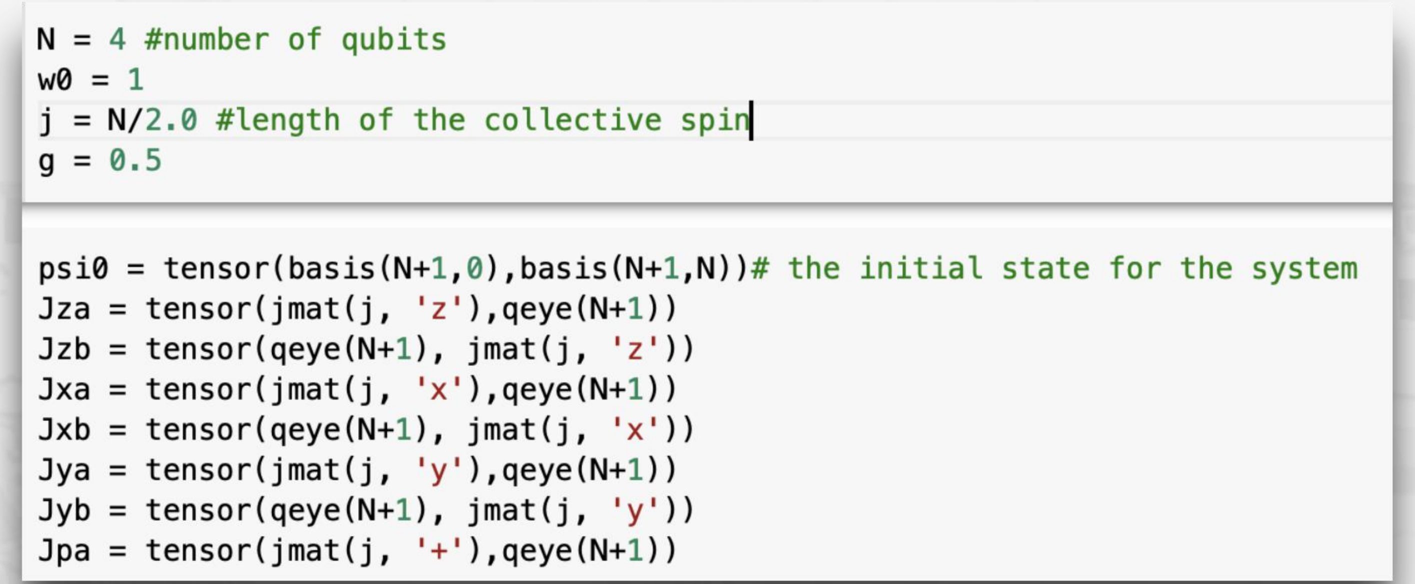
$$J_z = \sum_{i=1}^{N} \sigma_z^{(i)}$$

Spin of each qubit

```
jmat(N/2, 'z')
```

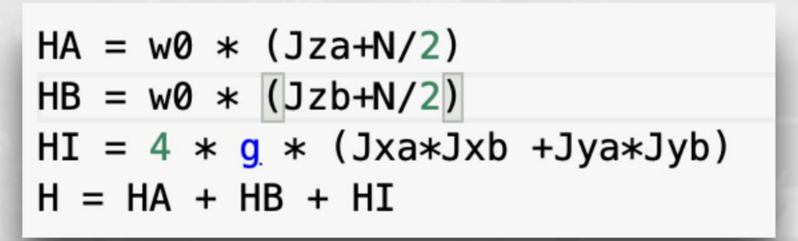
```
psi0=tensor(basis(N+1,0),basis(N+1,N))
# the initial state for the system
```

Quantum Battery: Spin Battery (3)

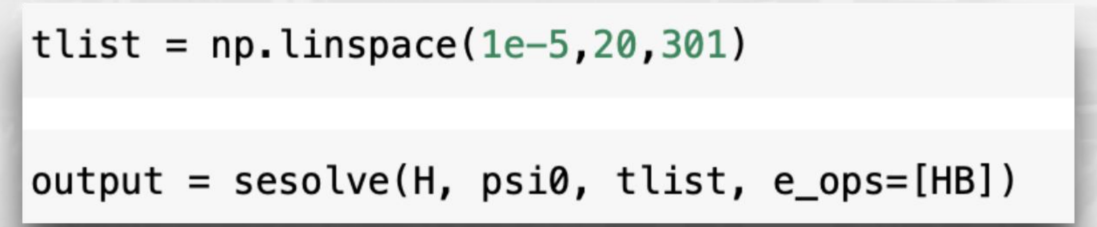


```
N = 4 #number of qubits
w0 = 1
j = N/2.0 #length of the collective spin
g = 0.5
```

```
psi0 = tensor(basis(N+1,0),basis(N+1,N))# the initial state for the system
Jza = tensor(jmat(j, 'z'),qeye(N+1))
Jzb = tensor(qeye(N+1), jmat(j, 'z'))
Jxa = tensor(jmat(j, 'x'),qeye(N+1))
Jxb = tensor(qeye(N+1), jmat(j, 'x'))
Jya = tensor(jmat(j, 'y'),qeye(N+1))
Jyb = tensor(qeye(N+1), jmat(j, 'y'))
Jpa = tensor(jmat(j, '+'),qeye(N+1))
```



```
HA = w0 * (Jza+N/2)
HB = w0 * (Jzb+N/2)
HI = 4 * g * (Jxa*Jxb +Jya*Jyb)
H = HA + HB + HI
```



```
tlist = np.linspace(1e-5,20,301)
```

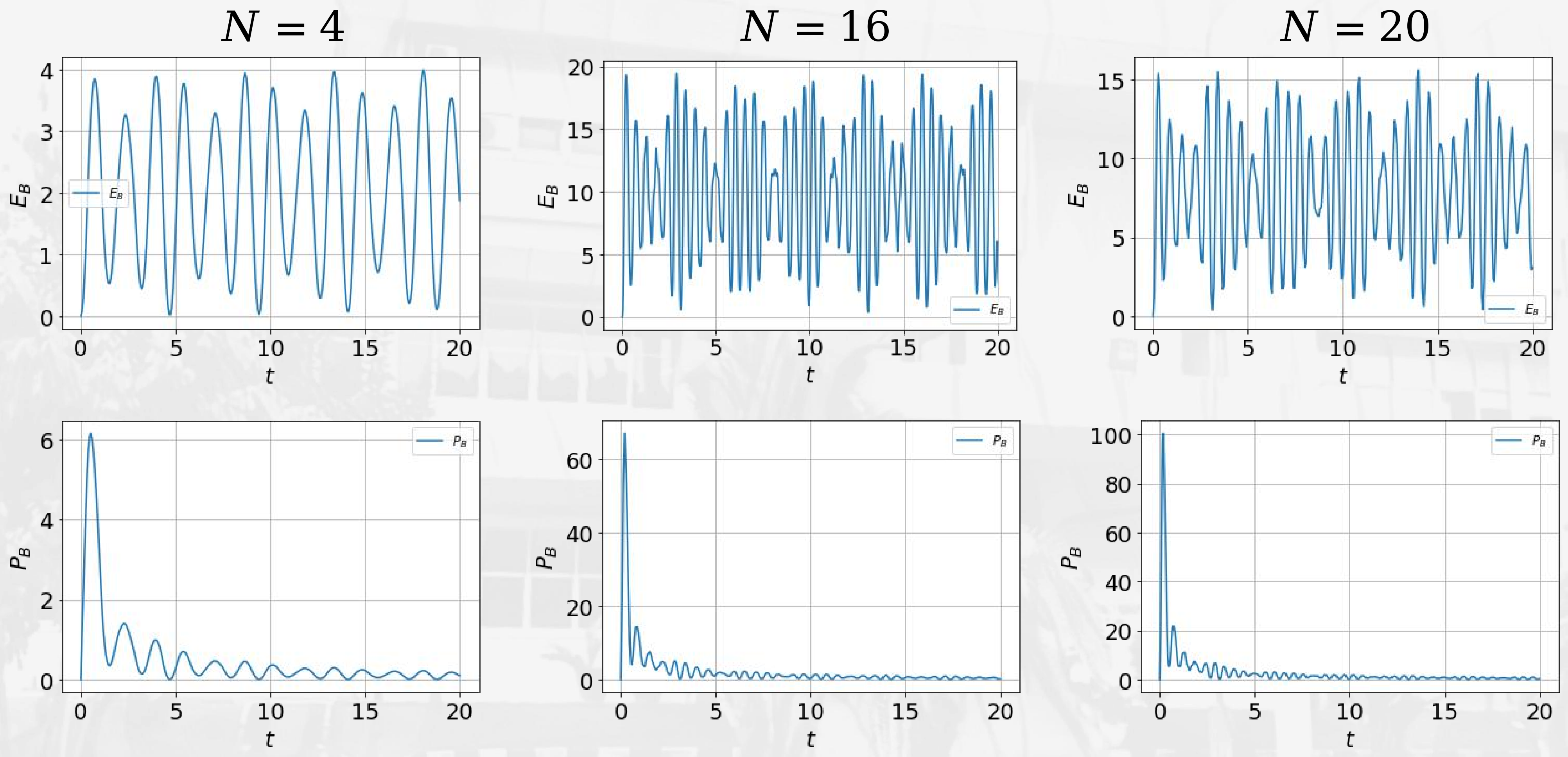
```
output = sesolve(H, psi0, tlist, e_ops=[HB])
```

$$\omega_0 = 1$$

$$E_{\mathrm{B}}^{(N)}(\tau) \equiv \mathrm{Tr}[H_{\mathrm{B}}\rho_{\mathrm{B}}(\tau)]$$

Total Energy at B

Quantum Battery: Spin Battery (4)



$N = 4$
$N = 16$
$N = 20$
E_B
P_B
t