QCS: Quantum Correlation Solver

Release 1.0.5

A Practical Guide

Zhi-Guang Lu

youngqlzg@gmail.com

Updating

Jun 8, 2023

Contents

1. Frontmatter

1.1 About This Documentation

This document contains a user guide and automatically generated API documentation for QCS. A PDF version of this text only is available at the Github.

1.2 Citing This Project

If you find this project useful, then please cite:

xxx

which may also be download from `http://arxiv.org/abs/2305.08923`.

1.3 About QCS

The correlation function plays a key role in quantum physics. For example, the statistical properties of light could be acquired by the second-order correlation function, and the photon transport can be characterized by single-photon transimission and reflection. Consequently, analytical or numerical calculating the higher-order correlation function is very important in quantum physics, which is the reason that we develop the tool.

In general, the calculation of higher-order correlation function, whether numerical or analytical, is extremely difficult for any open quantum systems. For example, multi-cavity or nonidentical multi-atom systems will face dimensional exponential growth of Hilbert space with the size of the cavity or atom. However, under one or multiple weak coherent inputs, the computation complexity of higher-order equal-time correlation function (ETCF) will be reduced from exponential to polynomial when the system Hamiltonian satisfies U(1) symmetry.

The Quantum Correlation Solver, or QCS, is an open-source framework written in the Python programming language, designed for calculating the higher-order ETCF of systems that satisfy the abovementioned criterion. This framework distinguishes itself from other available software solutions in providing the following advantages:

- QCS relies entirely on open-source software. You are free to modify and use it as you wish with no licensing fees or limitations.
- QCS is based on the Python scripting language, providing easy to read, fast code generation without the need to compile after modification.

- QCS allows for solving the higher-order ETCF, cross-correlation function, transimission, and reflection with arbitrary multiple weak coherent driven cases.
- QCS allows for solving the large size systems, such as multi-cavity and multi-atom.

For detailed information about new features of each release of QCS, see the *Change Log*.

1.4 Contributing to QCS

We welcome anyone who is interested in helping us perfected QCS. If you are interested, please contact me.

2. Installation

2.1 General Requirements

QCS mainly depends on two open-source libraries for scientific computing in the Python programming language. The following packages are currently required:

Package	Version	Details
Python	3.0+	Not tested on lower versions.
NumPy	1.8+	Not tested on lower versions.
SciPy	0.15+	Lower versions have missing features.

2.2 Installing via pip

It is often easiest to use the Python package manager pip.

```
pip install qcs_phy
```

Note that our Python package name is `qcs_phy`.

3. Users Guide

3.1 Guide Overview

The goal of this guide is to introduce you to the basic structures and functions that make up QCS. This guide is divided up into several sections, each highlighting a specific set of functionalities. In combination with the examples that can be found on the project web page https://github.com/ZhiGuangLu/qcs/tree/main/examples, this guide should provide a more or less complete overview. In addition, the *API documentation* for each function is located at the end of this guide.

3.2 Basic Operations on Quantum Objects

3.2.1 First thing first

To load the qutip modules, we must first call the import statement:

```
In [1]: from qcs_phy import qcs
```

that will load the kernel class `qcs`. Often, we also need to import the NumPy and Matplotlib libraries with:

```
In [2]: import numpy as np
In [3]: import matplotlib.pyplot as plt
```

Note that the class `qcs` contains all of the user available functions.

3.2.2 The symbol stipulations of quantum operators and Hamiltonian

Here, we must have to define a class of creation and annihilation operators about boson and fermion. For the sake of simplicity, we only consider three physical modes, i.e., the cavity mode, the spin-1/2 mode, and the collective spin mode. The specific representations are shown below:

Physical Mode	Symbol: annihilation	Symbol: creation	Python Code: `str`
cavity	a	$a^\dagger$	`"a", "ad"`
spin-1/2	σ	$\sigma^\dagger$	`"sm", "sp"`
n collective spins	$S=\sum_{i=1}^{n}\sigma_i$	$S^\dagger=\sum_{i=1}^{n}\sigma_i^\dagger$	`"Sm N=n", "Sp N=n"`
	$S_z=\sum_{i=1}^{n}\sigma_i^\dagger\sigma_i$		`"Sz N=n"`

Based on the definition above, we could construct Hamiltonian of any form. However, in order to obtain the matrix form of Hamiltonian at the subspace of certain excitation number, we must have to number each physical mode, i.e., existing in the form of `tuple` in Python, such as

$a_k^\dagger \leftrightarrow$("ad", k), $a_k \leftrightarrow$("a", k), $\sigma_k \leftrightarrow$("sm", k), $S_k \leftrightarrow$("Sm N=n", k), $\cdots$

In general, each Hamiltonian carries the corresponding coefficients in addition to containing various physical modes, and the form exists in the form of `list` in Python. Now, we consider a kind of Hamiltonian that only contains one term, and the specific representations are shown in the table below:

One term	Python Code: `list`
$\omega_c a_1^\dagger a_1$	[ω_c, ("ad", 1), ("a", 1)]
$\omega_e \sigma_1^\dagger \sigma_1$	[ω_e, ("sp", 1), ("sm", 1)]
$g a_1^\dagger \sigma_1$	[g, ("ad", 1), ("sm", 1)]
$g a_1 \sigma_1^\dagger$	[g, ("a", 1), ("sp", 1)]
$U a_1^\dagger a_1^\dagger a_1 a_1$	[U, ("ad", 1), ("ad", 1), ("a", 1), ("a", 1)]

Notably, the Python code of this Hamiltonian is described by the `list` form:

`Hj = [coff, ope1, ope2, ···]` $\Leftrightarrow$ `Hj = coff` $\times$ `ope1` $\times$ `ope2` $\times \cdots$

Here, `ope1` and `ope2` are denote as the `tuple` above, such as `ope1` $= a_1 \leftrightarrow$ `ope1 = ("a", 1)`. Note that `coff` can only be a number, rather than list and array, e.g.,

`coff = 0.5` ✔, `coff = [1, 2, 3]` ✘, `coff = numpy.linspace(0, 1, 10)` ✘

Similarly, for the Hamiltonian containing multiple term, we use `list[list]` Python code to represent it. Here, we take the Hamiltonian containing two terms as example, i.e.,

Two terms	Python Code: `list[list]`
$\omega_c a_1^\dagger a_1 + \omega_e \sigma_1^\dagger \sigma_1$	[[ω_c, ("ad", 1), ("a", 1)], [ω_e, ("sp", 1), ("sm", 1)]]
$g_1 a_1^\dagger \sigma_1 + g_2 a_1 \sigma_1^\dagger$	[[g_1, ("ad", 1), ("sm", 1)], [g_2, ("a", 1), ("sp", 1)]]

Obviously, it is also true for multiple terms, and the Python code of this Hamiltonian is described by the `list` form:

`H = [H1, H2, H3, ···]` $\Leftrightarrow$ `H = H1 + H2 + H3 +` $\cdots$

Here, `H1`, `H2`, and `H3` have the `list` form: `Hj = [coff, ope1, ope2, ...]`.

Note: For the Hamiltonian containing one term, we also use the `list[list]` form in order to ensure consistency.

Example: Constructing effective Hamiltonian

Here, we take four effective Hamiltonian as examples, i.e.,

$$H_1 = (\omega_c - i\kappa/2)a_1^\dagger a_1 + (\omega_e - i\gamma/2)\sigma_1^\dagger \sigma_1 + g(a_1^\dagger \sigma_1 + a_1 \sigma_1^\dagger), \tag{3.1}$$

$$H_2 = (\omega_c - i\kappa/2)a_1^\dagger a_1 + (\omega_e - i\gamma/2)\sigma_1^\dagger \sigma_1 + g(a_1^{\dagger 2} \sigma_1 + a_1^2 \sigma_1^\dagger), \tag{3.2}$$

$$H_3 = (\omega_c - i\kappa/2)a_1^\dagger a_1 + (\omega_e - i\gamma/2)\sum_{j=1}^{N} \sigma_j^\dagger \sigma_j + \sum_{j=1}^{N} g_j(a_1^\dagger \sigma_j + a_1 \sigma_j^\dagger), \tag{3.3}$$

$$H_4 = \sum_{j=1}^{N} [(\omega_c - i\kappa/2)a_j^\dagger a_j + U a_j^\dagger a_j^\dagger a_j a_j] + J \sum_{j=1}^{N-1} (a_j^\dagger a_{j+1} + a_j a_{j+1}^\dagger). \tag{3.4}$$

The corresponding Python codes are shown below:

```
# construct H1
H1 = []
H1.append([ωc-1j*κ/2, ("ad", 1), ("a", 1)])
H1.append([ωe-1j*γ/2, ("sp", 1), ("sm", 1)])
H1.append([g, ("ad", 1), ("sm", 1)])
H1.append([g, ("a", 1), ("sp", 1)])

# construct H2
H2 = []
H2.append([ωc-1j*κ/2, ("ad", 1), ("a", 1)])
H2.append([ωe-1j*γ/2, ("sp", 1), ("sm", 1)])
H2.append([g, ("ad", 1), ("ad", 1), ("sm", 1)])
H2.append([g, ("a", 1), ("a", 1), ("sp", 1)])

# construct H3
H3 = []
H3.append([ωc-1j*κ/2, ("ad", 1), ("a", 1)])
for k in range(N):
    H3.append([ωe-1j*γ/2, ("sp", k), ("sp", k)])
    H3.append([g[k], ("ad", 1), ("sm", k)])
    H3.append([g[k], ("a", 1), ("sp", k)])

# construct H4
H4 = []
for k in range(N):
    H4.append([ωe-1j*γ/2, ("ad", k), ("a", k)])
    H4.append([U, ("ad", k), ("ad", k), ("a", k), ("a", k)])
    if k != N-1:
        H4.append([J, ("ad", k), ("a", k+1)])
        H4.append([J, ("a", k), ("ad", k+1)])
```

3.2.3 The input and output channels

According to the scattering theory, we must have to define the input and output channels in order to obtain the statistical properties of outgoing light. To proceed, we consider a standard input-output formalism that relates μ_{in}, μ_{out}, and o_μ as

$$\mu_{\text{out}}(t) = \mu_{\text{in}}(t) - io_\mu(t). \tag{3.5}$$

Obviously, the quantum operator o_μ also satisfies the symbol stipulations 3.2.2 above.

For example, we assume that the name of input and output channels are $\mu = b$ and $\mu = c$, respectively, and the quantum operators are $o_b = \sqrt{\kappa}a_1$ and $o_c = \sqrt{\gamma}\sigma_1$. Here, we consider an incoming coherent state from the input channel, which the amplitude and frequency are β and ω_d, respectively. This process is equivalent with a coherent drive, and the Hamiltonian could be written as

$$H_d = \frac{\beta a_1 \exp(i\omega_d t) + \beta^* a_1^\dagger \exp(-i\omega_d t)}{\sqrt{2\pi/\kappa}}. \tag{3.6}$$

```
In [1]: from qcs_phy import qcs
In [2]: import numpy as np
In [3]: o_b = [np.sqrt(κ), ("a", 1)]
In [4]: o_c = [np.sqrt(γ), ("sm", 1)]
In [5]: Input = qcs.Input_channel("b", o_b, ω_d)
In [6]: Output = qcs.Output_channel("c", o_c)
```

Here, we must use the two functions of the class `qcs` to construct the `Input` and `Output`, and the two functions are given by

```
qcs.Input_channel(para1, para2, para3), qcs.Output_channel(para1, para2)
```

- The first parameter (`para1`) represents the name of channel in the form of `str`, such as `"b"` and `"c"`. Please ensure each channel name is unique, and this implies that we should rule out the possibility of like `"b"` and `"b1"` simultaneously as channel names.
- The second parameter (`para2`) represents the quantum operator o_μ in the form of `list`, e.g., `para2 = [np.sqrt(κ), ("a", 1)]`, `para2 = [np.sqrt(γ), ("sm", 1)]`.
- The third parameter (`para3`) represents the driving frequency ω_d, and it could be [number, list, or array], e.g., `para3 = 0.5`, `para3 = [1, 2, 3]`, `para3 = np.linspace(0, 1, 10)`.

Multiple coherent state inputs

As expected, the case of multiple coherent state inputs is equivalent with the multi-mode coherent drives. For simplicity, we consider the case of two coherent state inputs, and the corresponding input channels are $\mu = b_1$ and $\mu = b_2$, respectively. Meanwhile, we assume the quantum operators are $o_{b_1} = \sqrt{\kappa_1}a_1$ and $o_{b_2} = \sqrt{\kappa_2}a_2$; the amplitudes and driving frequencies are (β_1, ω_1) and (β_2, ω_2), respectively. Thus, the driven Hamiltonian could be written as

$$H_d = \frac{\beta_1 a_1 \exp(i\omega_1 t) + \beta_1^* a_1^\dagger \exp(-i\omega_1 t)}{\sqrt{2\pi/\kappa_1}} + \frac{\beta_2 a_2 \exp(i\omega_2 t) + \beta_2^* a_2^\dagger \exp(-i\omega_2 t)}{\sqrt{2\pi/\kappa_2}}. \tag{3.7}$$

Note: For the multiple coherent state inputs, although the individual amplitude could be ignored, the ratio between amplitudes are not negligible. Therefore, we must provide a new parameter in the form of `list`, i.e., `ratio=[`β_1`,` β_2`,` $\cdots$`]`. Actually, the specific values of β_1 and β_2 are unimportant, but we only care about their relative magnitudes. Thus, the parameter also could be equal to `[1,` β_2/β_1`,` $\cdots$`]`.

```
In [6]: o_b1 = [np.sqrt(κ1), ("a", 1)]
In [7]: o_b2 = [np.sqrt(κ2), ("a", 2)]
In [8]: Input_1 = qcs.Input_channel("b1", o_b1, ω1) # The first input
In [9]: Input_2 = qcs.Input_channel("b2", o_b2, ω2) # The second input
In [10]: Input = [Input_1, Input_2] # Two inputs
In [11]: ratio = [β1, β2] # The ratio between amplitudes
```

Note that the indexes of element in `ratio` must be a one-to-one correspondence with the indexes of element in `Input`. For the case of multiple outputs, the usage is completely identical with the multiple inputs.

3.2.4 Main body

When we construct the effective Hamiltonian `Heff`, the input channel `Input`, and the output channel `Output`, the main body has finished, and the Python code is shown below:

```
In [1]: from qcs_phy import qcs

In [2]: ... Heff
In [3]: ... Input
In [4]: ... Output
In [5]: system = qcs(Heff, Input, Output)
```

If the `Input` contains multiple coherent state inputs, we will provide the new parameter `ratio`, i.e.,

```
In [1]: from qcs_phy import qcs

In [2]: ... Heff
In [3]: ... Input
In [4]: ... ratio
In [5]: ... Output
In [6]: system = qcs(Heff, Input, Output, ratio)
```

3.3 The Higher-Order Equal-Time Correlation Function

In Sec. 3.2, we have provided the steps how to construct the effective Hamiltonian, Input, and Output. Meanwhile, we also define the name of output channel in the form of `str`, such as `"c1"` or `"c2"`. For the sake of simplicity, we assume the output channel modes are $c_1, c_2, \ldots, c_n$, and the corresponding channel names are `"c1"`, `"c2"`, ..., `"cn"`, respectively. Subsequently, we will introduce the function on `qcs` class that calculates physical quantities, i.e.,

```
system.calculate_quantity(Quantity, tlist=0, zp=0)
```

- The first parameter (`Quantity`) represents the physical quantity in the form of `str`.
- The second parameter (`tlist`) represents time variable, which is only given in multiple unequal driving frequencies (→ Dynamical ETCF 3.3.2). Default is zero (→ Steady ETCF 3.3.2).
- The third parameter (`zp`) represents the coefficient 0^+, which is only given when the system has multiple steadystates. Default is zero.

3.3.1 Steady ETCF

Here, the criteria of steady ETCF is denoted as the system reduced density matrix is time-independent after long-time evolution. In other words, if there is single coherent state input or multiple coherent state inputs with the identical driving frequencies, the system reduced density matrix will satisfy the criteria. Meanwhile, the usage of the parameter `Quantity` is shown in the table below:

`Quantity`	Equation	Name
`"c1c1"`	$g^{(2)}(0) = \frac{\langle\Psi_{\rm out}\|c_1^\dagger(t)c_1^\dagger(t)c_1(t)c_1(t)\|\Psi_{\rm out}\rangle}{\langle\Psi_{\rm out}\|c_1^\dagger(t)c_1(t)\|\Psi_{\rm out}\rangle^2}$	2nd-order ETCF
`"c1c1c1"`	$g^{(3)}(0) = \frac{\langle\Psi_{\rm out}\|c_1^\dagger(t)c_1^\dagger(t)c_1^\dagger(t)c_1(t)c_1(t)c_1(t)\|\Psi_{\rm out}\rangle}{\langle\Psi_{\rm out}\|c_1^\dagger(t)c_1(t)\|\Psi_{\rm out}\rangle^3}$	3rd-order ETCF
`"c1c2"`	$g_c^{(2)}(0) = \frac{\langle\Psi_{\rm out}\|c_1^\dagger(t)c_2^\dagger(t)c_1(t)c_2(t)\|\Psi_{\rm out}\rangle}{\langle\Psi_{\rm out}\|c_1^\dagger(t)c_1(t)\|\Psi_{\rm out}\rangle\langle\Psi_{\rm out}\|c_2^\dagger(t)c_2(t)\|\Psi_{\rm out}\rangle}$	2nd-order cross ETCF
`"c1c1c2"`	$g_c^{(3)}(0) = \frac{\langle\Psi_{\rm out}\|c_1^\dagger(t)c_1^\dagger(t)c_2^\dagger(t)c_1(t)c_1(t)c_2(t)\|\Psi_{\rm out}\rangle}{\langle\Psi_{\rm out}\|c_1^\dagger(t)c_1(t)\|\Psi_{\rm out}\rangle^2\langle\Psi_{\rm out}\|c_2^\dagger(t)c_2(t)\|\Psi_{\rm out}\rangle}$	3nd-order cross ETCF
⋮		

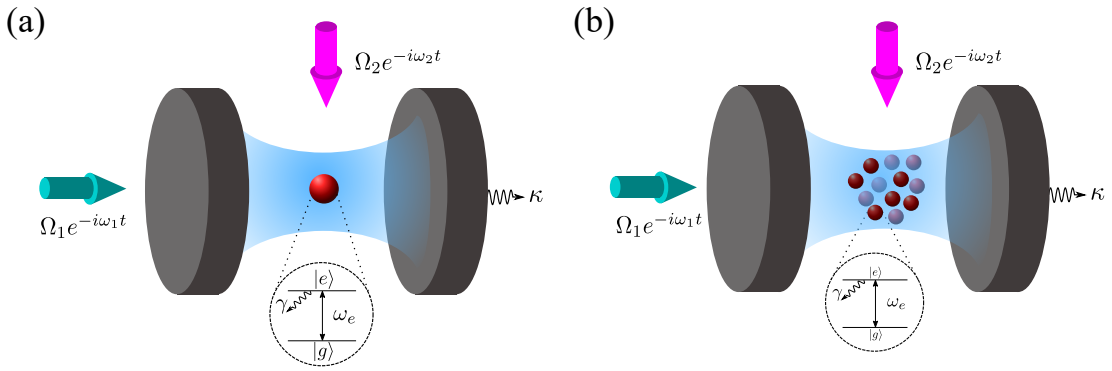

Figure 3.1: (a) Jaynes-Cummings Model. (b) Tavis-Cummings Model.

Subsequently, we will apply the method to two typical examples in quantum optics and provide the corresponding Python code.

Example-1: Jaynes-Cummings Model

A schematic of the considered system is shown in Fig. 3.1(a). Here, the system Hamiltonian is $H = \omega_c a^\dagger a + \omega_e \sigma^\dagger \sigma + g\left(a^\dagger \sigma + \sigma^\dagger a\right)$, and the corresponding Lindblad master equation is

$$\dot{\rho} = -i[H,\rho] + \kappa \mathscr{D}[a]\rho + \gamma \mathscr{D}[\sigma]\rho = -i[H_{\text{eff}}\rho - \rho H_{\text{eff}}^\dagger] + \kappa a \rho a^\dagger + \gamma \sigma \rho \sigma^\dagger, \tag{3.8}$$

where $\mathscr{D}[A]\rho = A\rho A^\dagger - \frac{1}{2}\{A^\dagger A, \rho\}$ and $H_{\text{eff}} = H - i\frac{\kappa}{2}a^\dagger a - i\frac{\gamma}{2}\sigma^\dagger\sigma$. Meanwhile, the coherently driven Hamiltonian is

$$H_d = \Omega_1 e^{i\omega_1 t} a + \Omega_2 e^{i\omega_2 t}\sigma + \text{H.c.}, \tag{3.9}$$

where $\Omega_1 \to 0$, $\Omega_2 = \eta\Omega_1$, and $\omega_1 = \omega_2$.

Based on the discussions above, the input-output formalism of cavity and atom could be written as

$$b_{1,\text{out}}(t) = b_{1,\text{in}}(t) - i\sqrt{\kappa/2}a(t),\ b_{2,\text{out}}(t) = b_{2,\text{in}}(t) - i\sqrt{\gamma/2}\sigma(t) \tag{3.10}$$

and

$$c_{1,\text{out}}(t) = c_{1,\text{in}}(t) - i\sqrt{\kappa/2}a(t),\ c_{2,\text{out}}(t) = c_{2,\text{in}}(t) - i\sqrt{\gamma/2}\sigma(t), \tag{3.11}$$

which correspond to the input and output channels, respectively.

To proceed, we will calculate some important physical quantities, i.e., second-order equal-time correlation function and second-order equal-time cross correlation function, and these formulas are as follows:

$$g^{(2)}(0) = \frac{\langle\Psi_{\text{out}}|c_1^\dagger(t)c_1^\dagger(t)c_1(t)c_1(t)|\Psi_{\text{out}}\rangle}{\langle\Psi_{\text{out}}|c_1^\dagger(t)c_1(t)|\Psi_{\text{out}}\rangle^2} = \frac{\text{Tr}\left[a^\dagger a^\dagger a a \rho_{\text{ss}}\right]}{\text{Tr}[a^\dagger a \rho_{\text{ss}}]^2}, \tag{3.12}$$

$$g_c^{(2)}(0) = \frac{\langle\Psi_{\text{out}}|c_1^\dagger(t)c_2^\dagger(t)c_1(t)c_2(t)|\Psi_{\text{out}}\rangle}{\langle\Psi_{\text{out}}|c_1^\dagger(t)c_1(t)|\Psi_{\text{out}}\rangle\langle\Psi_{\text{out}}|c_2^\dagger(t)c_2(t)|\Psi_{\text{out}}\rangle} = \frac{\text{Tr}\left[a^\dagger\sigma^\dagger a\sigma\rho_{\text{ss}}\right]}{\text{Tr}[a^\dagger a\rho_{\text{ss}}]\times\text{Tr}[\sigma^\dagger\sigma\rho_{\text{ss}}]}, \tag{3.13}$$

where ρ_{ss} represents the steady state of the system.

Obviously, the dynamical ETCF appears only when the two driving frequencies (i.e., ω_1 and ω_2) are not identical. Subsequently, we will discuss two cases: One is the cavity-driven case, and the other is the cavity-atom-driven case, which correspond to $\eta = 0$ and $\eta \neq 0$, respectively. Note for $\eta \neq 0$ that we need to provide an additional parameter, i.e., `ratio=[1, β2/β1]`, where $\beta_1 = \Omega_1\sqrt{2\pi/\kappa}$ and $\beta_2 = \Omega_2\sqrt{2\pi/\gamma}$, which imply $\beta_2/\beta_1 = \eta\sqrt{\kappa/\gamma}$.

The Python code of the cavity-driven case:

```
from qcs_phy import qcs
import matplotlib.pyplot as plt
import numpy as np

# The system parameters
ome_c = 0 # the frequency of cavity
ome_e = 0 # the frequency of atom
kap = 1   # the decay rate of cavity
gam = 1   # the decay rate of atom
g = 0.7   # the coupling strength between cavity and atom
ome_1 = np.linspace(-2, 2, 1000) # the driving frequency of cavity mode

# construct the effective Hamiltonian
Heff = []
Heff.append([ome_c-1j*kap/2, ("ad", 0), ("a", 0)])
Heff.append([ome_e-1j*gam/2, ("sp", 0), ("sm", 0)])
Heff.append([g, ("ad", 0), ("sm", 0)])
Heff.append([g, ("a", 0), ("sp", 0)])

# construct input and output channels
o_b1 = [np.sqrt(kap/2), ("a", 0)]
o_c1 = o_b1
o_c2 = [np.sqrt(gam/2), ("sm", 0)]

Input = qcs.Input_channel("b1", o_b1, ome_1)
Output_1 = qcs.Output_channel("c1", o_c1)
Output_2 = qcs.Output_channel("c2", o_c2)
Output = [Output_1, Output_2]

# construct main body
system = qcs(Heff, Input, Output)

# calculate physical quantities
g2_0 = system.calculate_quantity("c1c1")
g2_c_0 = system.calculate_quantity("c1c2")

# drawing
plt.plot(ome_1, g2_0, label=r'$g^{(2)}(0)$')
plt.plot(ome_1, g2_c_0, label=r'$g^{(2)}_c(0)$')
plt.xlabel(r"$\omega_1$", size=15)
plt.legend(prop={'size': 15})
plt.show()
```

The result is shown in Fig. 3.2(a).

The Python code of the cavity-atom-driven case:

```
from qcs_phy import qcs
import matplotlib.pyplot as plt
import numpy as np

# The system parameters
ome_c = 0 # the frequency of cavity
ome_e = 0 # the frequency of atom
kap = 1   # the decay rate of cavity
gam = 0.1   # the decay rate of atom
g = 0.7   # the coupling strength between cavity and atom
eta_list = [0, 1, 2, 3] # the ratio of the driving strengths, Ω2/Ω1 = η
ome_d = np.linspace(-6, 2, 1000) # the driving frequency of
                                 # cavity and atom modes, ω1 = ω2 = ωd
# construct the effective Hamiltonian
Heff = []
Heff.append([ome_c-1j*kap/2, ("ad", 0), ("a", 0)])
Heff.append([ome_e-1j*gam/2, ("sp", 0), ("sm", 0)])
Heff.append([g, ("ad", 0), ("sm", 0)])
Heff.append([g, ("a", 0), ("sp", 0)])

# construct input and output channels
o_b1 = [np.sqrt(kap/2), ("a", 0)]
o_b2 = [np.sqrt(gam/2), ("sm", 0)]
o_c1 = o_b1

Input_1 = qcs.Input_channel("b1", o_b1, ome_d)
Input_2 = qcs.Input_channel("b2", o_b2, ome_d)
Input = [Input_1, Input_2]
Output = qcs.Output_channel("c1", o_c1)

for eta in eta_list:
    ratio = [1, eta * np.sqrt(kap / gam)] # the additional parameter
    # construct main body
    system = qcs(Heff, Input, Output, ratio)
    # calculate 2nd-order ETCF
    g2_0 = system.calculate_quantity("c1c1")
    # drawing
    plt.semilogy(ome_d, g2_0, label = r"$\eta=%s$" % eta)
plt.xlabel(r"$\omega_1$", size=15)
plt.ylabel(r"$g^{(2)}(0)$", size=15)
plt.legend(prop={'size': 10})
plt.show()
```

The result is shown in Fig. 3.2(b).

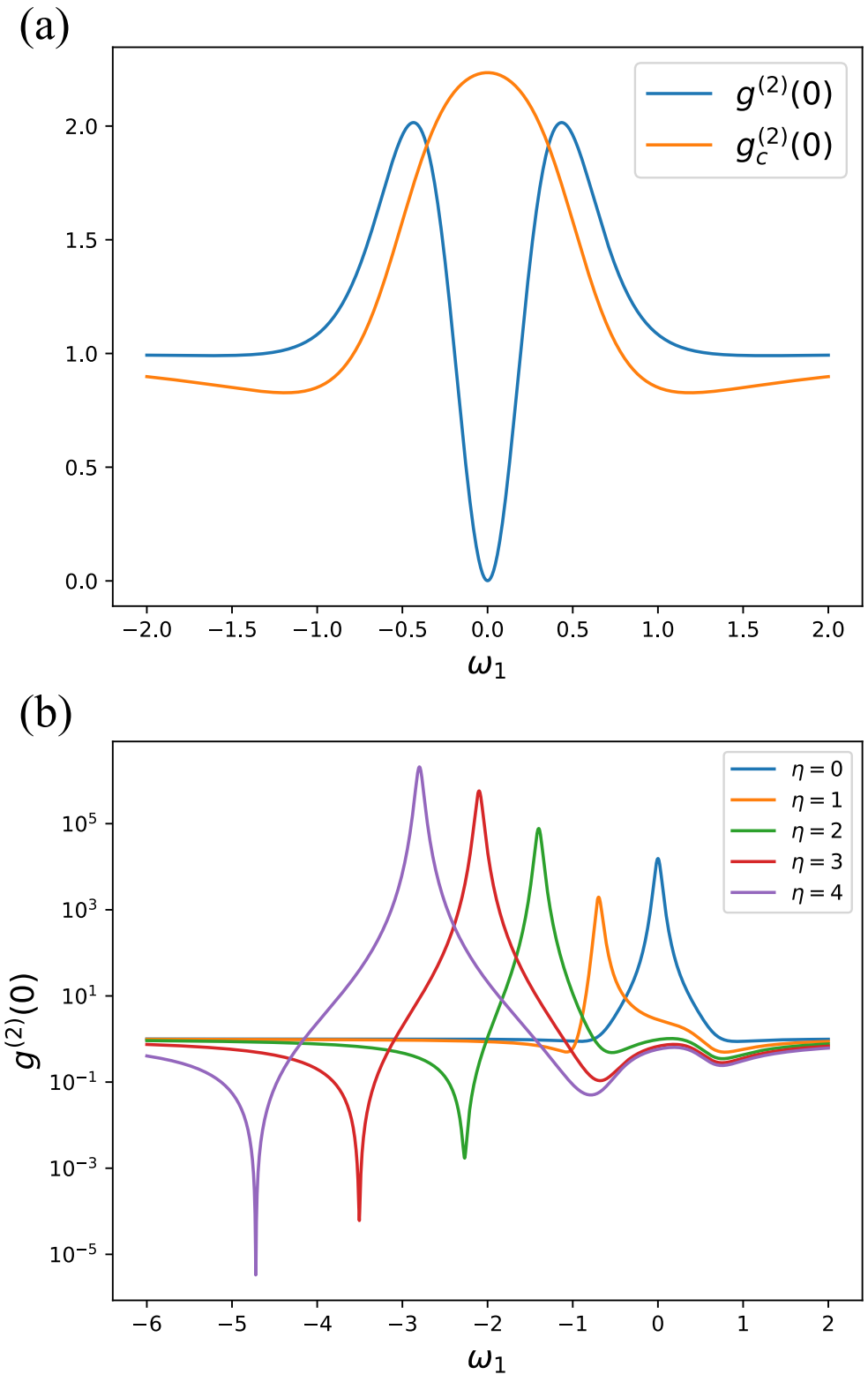

Figure 3.2: (a) The cavity-driven case. (b) The cavity-atom-driven case.

Example-2: Tavis-Cummings Model

A schematic of the considered system is shown in Fig. 3.1(b). Here, the system Hamiltonian is

$$H = \omega_c a^\dagger a + \sum_{n=1}^{N} \omega_n \sigma_n^\dagger \sigma_n + \sum_{n=1}^{N} g_n \left(a^\dagger \sigma_n + \sigma_n^\dagger a\right), \tag{3.14}$$

and the corresponding Lindblad master equation is

$$\dot{\rho} = -i[H,\rho] + \kappa \mathscr{D}[a]\rho + \sum_{n=1}^{N} \gamma_n \mathscr{D}[\sigma_n]\rho = -i[H_{\rm eff}\rho - \rho H_{\rm eff}^\dagger] + \kappa a \rho a^\dagger + \sum_{n=1}^{N} \gamma_n \sigma_n \rho \sigma_n^\dagger, \tag{3.15}$$

where $H_{\rm eff} = H - i\frac{\kappa}{2} a^\dagger a - i\sum_{n=1}^{N} \frac{\gamma_n}{2} \sigma_n^\dagger \sigma_n$.

Here, we assume that all the emitters are identical ($\omega_n = \omega_e$, $g_n = g$, and $\gamma_n = \gamma$ for all $n \in \{1,2,\ldots,N\}$), and the effective Hamiltonian can be written as:

$$H_{\rm eff} = \left(\omega_c - \frac{i\kappa}{2}\right) a^\dagger a + \left(\omega_e - \frac{i\gamma}{2}\right) S_z + g\left(a^\dagger S + S^\dagger a\right), \tag{3.16}$$

where $S_z = \sum_{n=1}^{N} \sigma_n^\dagger \sigma_n$ and $S = \sum_{n=1}^{N} \sigma_n$. Meanwhile, the coherently driven Hamiltonian is

$$H_d = \Omega_1 e^{i\omega_1 t} a + \Omega_2 e^{i\omega_2 t} S + \text{H.c.}, \tag{3.17}$$

where $\Omega_1 \to 0$, $\Omega_2 = \eta\Omega_1$, and $\omega_1 = \omega_2$.

Based on the discussions above, the input-output formalism of cavity and atom could be written as

$$b_{1,\rm out}(t) = b_{1,\rm in}(t) - i\sqrt{\kappa/2}a(t),\ b_{2,\rm out}(t) = b_{2,\rm in}(t) - i\sqrt{\gamma/2}S(t) \tag{3.18}$$

and

$$c_{1,\rm out}(t) = c_{1,\rm in}(t) - i\sqrt{\kappa/2}a(t),\ c_{2,\rm out}(t) = c_{2,\rm in}(t) - i\sqrt{\gamma/2}S(t), \tag{3.19}$$

which correspond to the input and output channels, respectively.

To proceed, we will calculate some important physical quantities, i.e., second-order equal-time correlation function and second-order equal-time cross correlation function, and these formulas are as follows:

$$g_1^{(2)}(0) = \frac{\langle\Psi_{\rm out}|c_1^\dagger(t)c_1^\dagger(t)c_1(t)c_1(t)|\Psi_{\rm out}\rangle}{\langle\Psi_{\rm out}|c_1^\dagger(t)c_1(t)|\Psi_{\rm out}\rangle^2} = \frac{{\rm Tr}\left[a^\dagger a^\dagger a a \rho_{\rm ss}\right]}{{\rm Tr}[a^\dagger a \rho_{\rm ss}]^2}, \tag{3.20}$$

$$g_2^{(2)}(0) = \frac{\langle\Psi_{\rm out}|c_2^\dagger(t)c_2^\dagger(t)c_2(t)c_2(t)|\Psi_{\rm out}\rangle}{\langle\Psi_{\rm out}|c_2^\dagger(t)c_2(t)|\Psi_{\rm out}\rangle^2} = \frac{{\rm Tr}\left[S^\dagger S^\dagger S S \rho_{\rm ss}\right]}{{\rm Tr}[S^\dagger S \rho_{\rm ss}]^2}, \tag{3.21}$$

$$g_c^{(2)}(0) = \frac{\langle\Psi_{\rm out}|c_1^\dagger(t)c_2^\dagger(t)c_1(t)c_2(t)|\Psi_{\rm out}\rangle}{\langle\Psi_{\rm out}|c_1^\dagger(t)c_1(t)|\Psi_{\rm out}\rangle\langle\Psi_{\rm out}|c_2^\dagger(t)c_2(t)|\Psi_{\rm out}\rangle} = \frac{{\rm Tr}\left[a^\dagger S^\dagger a S \rho_{\rm ss}\right]}{{\rm Tr}[a^\dagger a \rho_{\rm ss}] \times {\rm Tr}[S^\dagger S \rho_{\rm ss}]}. \tag{3.22}$$

Finally, we will calculate three cases: cavity-driven case, atoms-driven case, and both. Note that the atoms-driven case is equivalent to driving a big spin, i.e., spin $N/2$.

The Python code of the cavity-driven case:

```
from qcs_phy import qcs
import matplotlib.pyplot as plt
import numpy as np

# The system parameters
ome_c = 0     # the frequency of cavity
ome_e = 0     # the frequency of atom
kap = 1       # the decay rate of cavity
gam = 0.01    # the decay rate of atom
g = 2         # the coupling strength between cavity and atom
ome_1 = np.linspace(-25, 25, 4000) # the driving frequency of cavity mode
N_list = [2, 10, 20, 50, 100] # the number of emitters

for N in N_list:
    # construct the effective Hamiltonian
    Heff = []
    Heff.append([ome_c-1j*kap/2, ("ad", 0), ("a", 0)])
    Heff.append([ome_e-1j*gam/2, ("Sz N=%s" % N, 0)])
    Heff.append([g, ("ad", 0), ("Sm N=%s" % N, 0)])
    Heff.append([g, ("a", 0), ("Sp N=%s" % N, 0)])

    # construct input and output channels
    o_b1 = [np.sqrt(kap/2), ("a", 0)]
    o_c1 = o_b1
    o_c2 = [np.sqrt(gam/2), ("Sm N=%s" % N, 0)]

    Input = qcs.Input_channel("b1", o_b1, ome_1)
    Output_1 = qcs.Output_channel("c1", o_c1)
    Output_2 = qcs.Output_channel("c2", o_c2)
    Output = [Output_1, Output_2]

    # construct main body
    system = qcs(Heff, Input, Output)
    # calculate physical quantities
    g2_1_0 = system.calculate_quantity("c1c1")
    # g2_2_0 = system.calculate_quantity("c2c2")
    # g2_c_0 = system.calculate_quantity("c1c2")

    # drawing
    plt.plot(ome_1, g2_1_0, label=r'$N = %s$' % N)
    # plt.plot(ome_1, g2_2_0, label=r'$N = %s$' % N)
    # plt.plot(ome_1, g2_c_0, label=r'$N = %s$' % N)
plt.xlabel(r"$\omega_1$", size=15)
plt.ylabel(r"$g_1^{(2)}(0)$", size=15)
plt.legend(prop={'size': 15})
plt.ylim([0, 3])
plt.show()
```

The result is shown in Fig. 3.3.

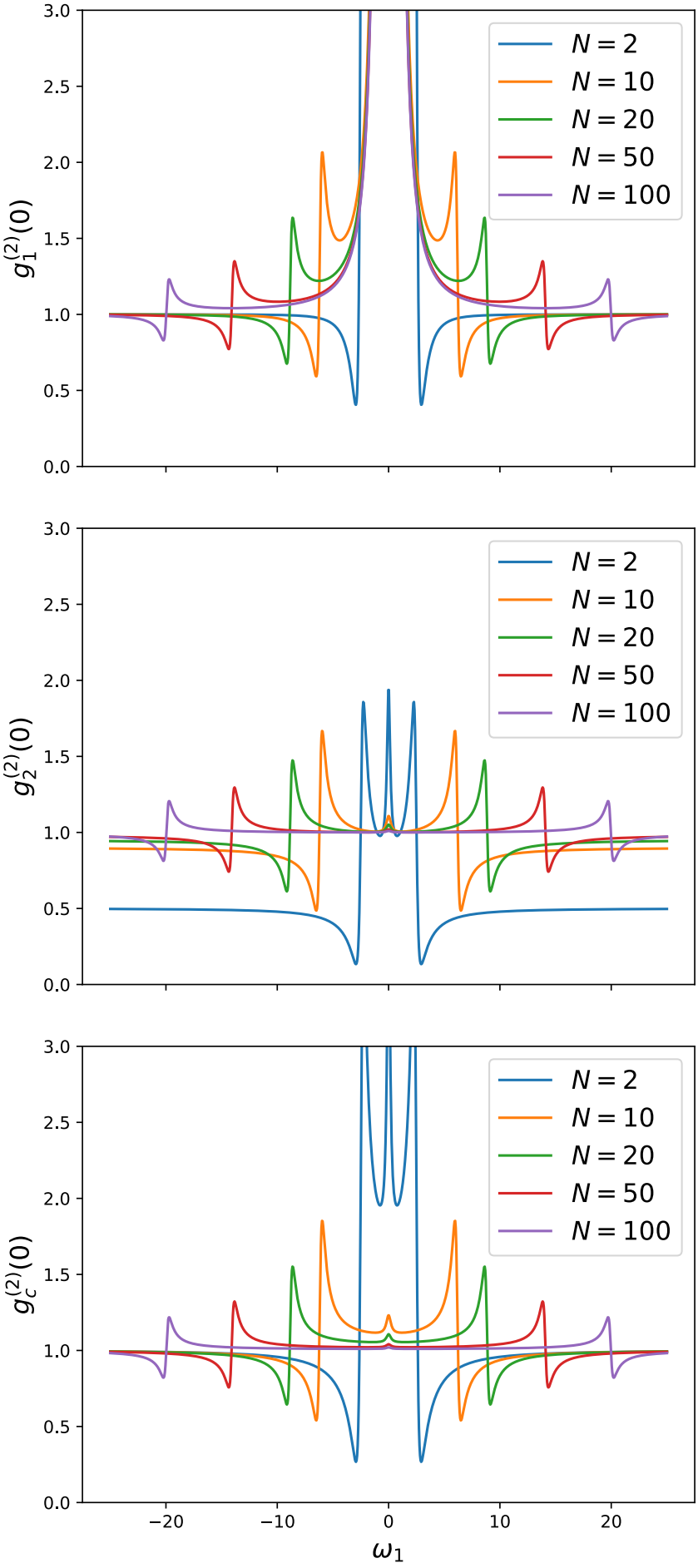

Figure 3.3: The cavity-driven case.

The Python code of the atoms-driven case:

```
from qcs_phy import qcs
import matplotlib.pyplot as plt
import numpy as np

# The system parameters
ome_c = 0     # the frequency of cavity
ome_e = 0     # the frequency of atom
kap = 1       # the decay rate of cavity
gam = 0.01    # the decay rate of atom
g = 2         # the coupling strength between cavity and atom
ome_1 = np.linspace(-25, 25, 4000) # the driving frequency of cavity mode
N_list = [2, 10, 20, 50, 100] # the number of emitters

for N in N_list:
    # construct the effective Hamiltonian
    Heff = []
    Heff.append([ome_c-1j*kap/2, ("ad", 0), ("a", 0)])
    Heff.append([ome_e-1j*gam/2, ("Sz N=%s" % N, 0)])
    Heff.append([g, ("ad", 0), ("Sm N=%s" % N, 0)])
    Heff.append([g, ("a", 0), ("Sp N=%s" % N, 0)])

    # construct input and output channels
    o_b1 = [np.sqrt(gam/2), ("Sm N=%s" % N, 0)]
    o_c1 = [np.sqrt(kap/2), ("a", 0)]
    o_c2 = o_b1

    Input = qcs.Input_channel("b1", o_b1, ome_1)
    Output_1 = qcs.Output_channel("c1", o_c1)
    Output_2 = qcs.Output_channel("c2", o_c2)
    Output = [Output_1, Output_2]

    # construct main body
    system = qcs(Heff, Input, Output)
    # calculate physical quantities
    g2_1_0 = system.calculate_quantity("c1c1")
    # g2_2_0 = system.calculate_quantity("c2c2")
    # g2_c_0 = system.calculate_quantity("c1c2")

    # drawing
    plt.plot(ome_1, g2_1_0, label=r'$N = %s$' % N)
    # plt.plot(ome_1, g2_2_0, label=r'$N = %s$' % N)
    # plt.plot(ome_1, g2_c_0, label=r'$N = %s$' % N)
plt.xlabel(r"$\omega_1$", size=15)
plt.ylabel(r"$g_1^{(2)}(0)$", size=15)
plt.legend(prop={'size': 15})
plt.ylim([0, 3])
plt.show()
```

The result is shown in Fig. 3.4.

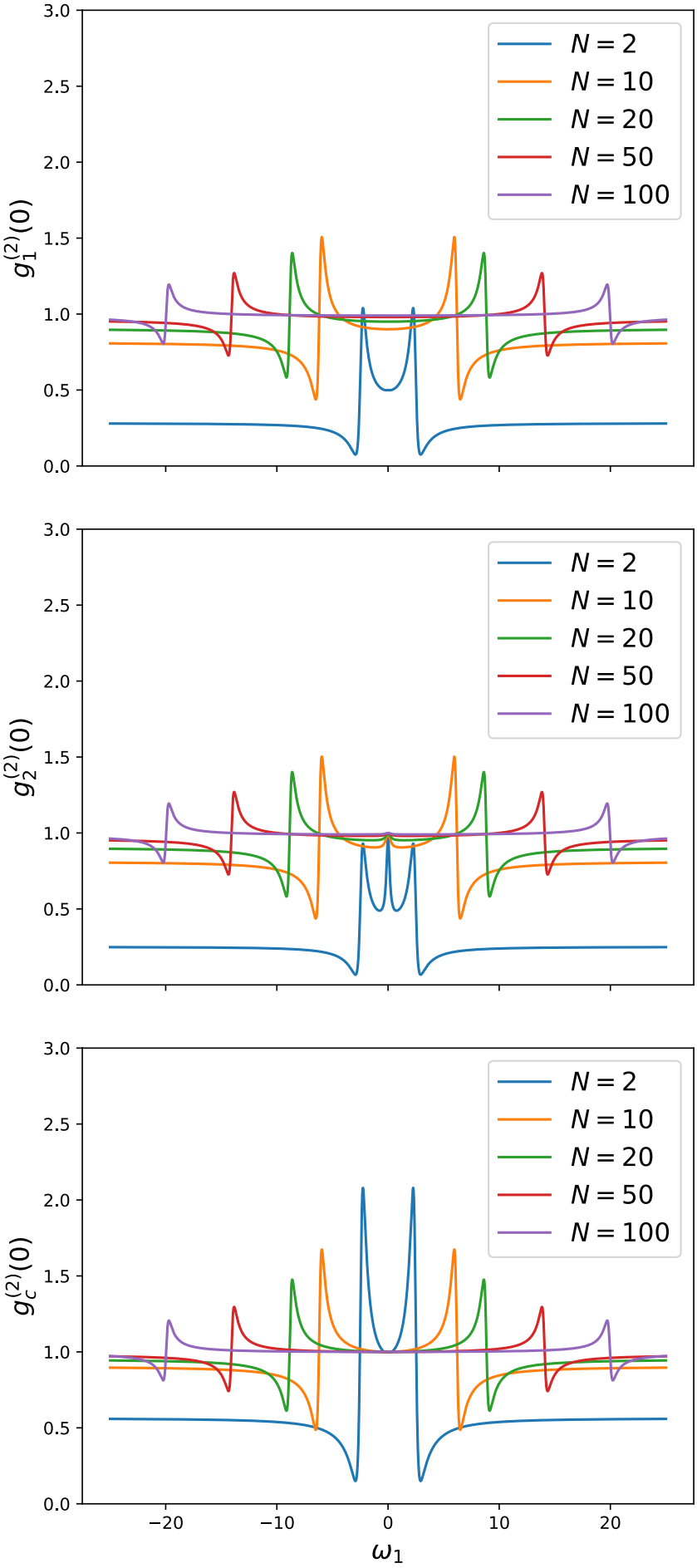

Figure 3.4: The atoms-driven case.

The Python code of the cavity-atoms-driven case:

```
from qcs_phy import qcs
import matplotlib.pyplot as plt
import numpy as np

# The system parameters
ome_c = 0     # the frequency of cavity
ome_e = 0     # the frequency of atom
kap = 1       # the decay rate of cavity
gam = 1    # the decay rate of atom
g = 1         # the coupling strength between cavity and atom
eta = 1      # the ratio of the two driving strengths
N_list = [18, 32, 50, 72, 98]  # the number of emitters
color = ['#1f77b4', '#ff7f0e', '#2ca02c', '#d62728', '#9467bd'] # color
for k, N in enumerate(N_list):
    # the driving frequency of both cavity and atoms mode: ω_1 = ω_2 = ω_d
    ome_d = - g * eta * N + np.linspace(-10, 10, 10000)
    # construct the effective Hamiltonian
    Heff = []
    Heff.append([ome_c-1j*kap/2, ("ad", 0), ("a", 0)])
    Heff.append([ome_e-1j*gam/2, ("Sz N=%s" % N, 0)])
    Heff.append([g, ("ad", 0), ("Sm N=%s" % N, 0)])
    Heff.append([g, ("a", 0), ("Sp N=%s" % N, 0)])
    # construct input and output channels
    o_b1 = [np.sqrt(kap/2), ("a", 0)]
    o_b2 = [np.sqrt(gam/2), ("Sm N=%s" % N, 0)]
    o_c1 = o_b1

    Input_1 = qcs.Input_channel("b1", o_b1, ome_d)
    Input_2 = qcs.Input_channel("b2", o_b2, ome_d)
    Output = qcs.Output_channel("c1", o_c1)
    Input = [Input_1, Input_2]
    # construct main body
    ratio = [1, eta * np.sqrt(kap / gam)]
    system = qcs(Heff, Input, Output, ratio)
    # calculate physical quantities
    g2_1_0 = system.calculate_quantity("c1c1")
    # drawing
    plt.semilogy(ome_d + g * eta * N, g2_1_0, label=r'$N = %s$' % N)
    plt.axvline(g * eta * np.sqrt(N / 2), linestyle='--', color=color[k])
    plt.axvline(-g * eta * np.sqrt(N / 2), linestyle='--', color=color[k])
plt.xticks([-10, -7, -6, -5, -4, -3, 0, 3, 4, 5, 6, 7, 10])
plt.xlabel(r"$\omega_1 + g\eta N$", size=15)
plt.ylabel(r"$g_1^{(2)}(0)$", size=15)
plt.legend(prop={'size': 15}, framealpha=1)
plt.show()
```

The result is shown in Fig. 3.5. Actually, we could obtain the analytical expression of the second-order ETCF, and then calculate the optimal driving frequency of strong bunching and antibunching effects for different number of atoms N. For these antibunching points as shown in these dashed

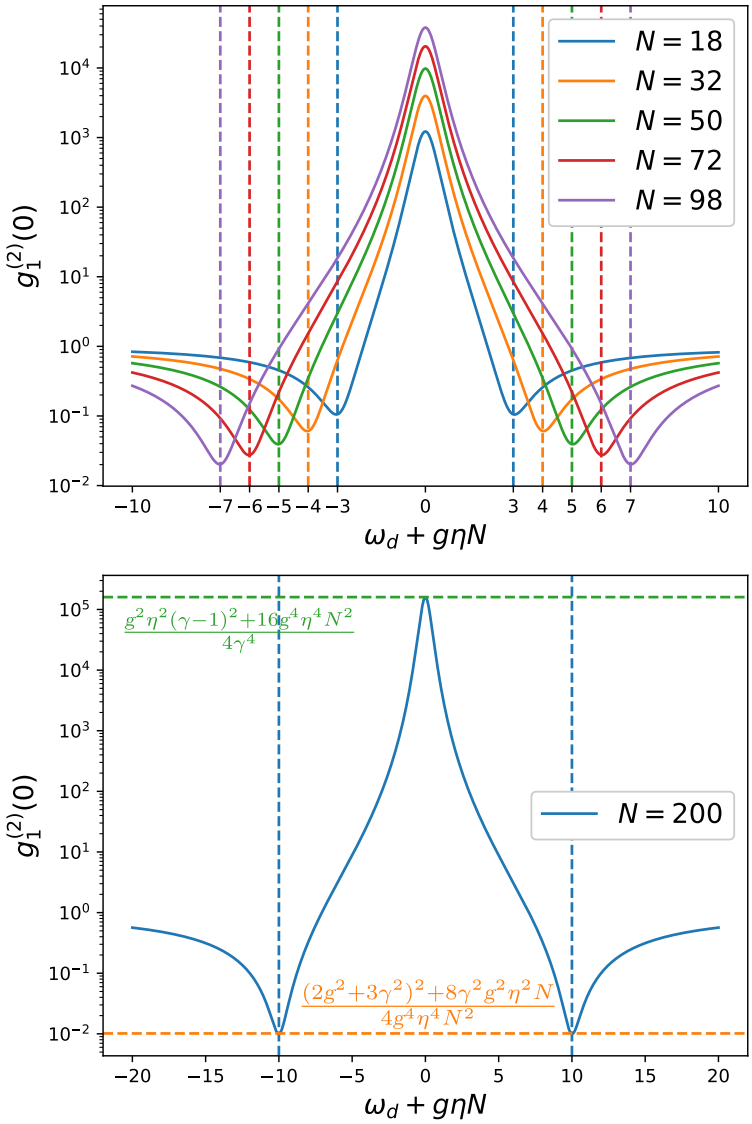

Figure 3.5: The cavity-atoms-driven case for different number of atoms.

lines in Fig. 3.5 [top], the optimal driving frequency is $\omega_d^{\text{opt}} \approx -g\eta N \pm g\eta\sqrt{N/2}$, and the corresponding second-order ETCF has

$$g^{(2)}_{1,\text{opt}}(0;\omega_d^{\text{opt}}) \approx \frac{(2g^2+3\gamma^2)^2+8\gamma^2g^2\eta^2N}{4g^4\eta^4N^2}, \tag{3.23}$$

as shown in the yellow dashed line in Fig. 3.5 [bottom]. Similarly, for the strong bunching point, the driving frequency is $\omega_d^{\text{bun}} = -g\eta N$, and the corresponding second-order ETCF has

$$g^{(2)}_{1,\text{bun}}(0;\omega_d^{\text{bun}}) \approx \frac{g^2\eta^2(\gamma-1)^2+16g^4\eta^4N^2}{4\gamma^4}, \tag{3.24}$$

as shown in the green dashed line in Fig. 3.5 [bottom]. However, these results above are based on $\eta \gg 1$ or $N \gg 1$, and this implies that the two antibunching points ($\omega_d^{\text{opt}} \approx -g\eta \pm g\eta/\sqrt{2}$) still exist for big enough η even in JC model.

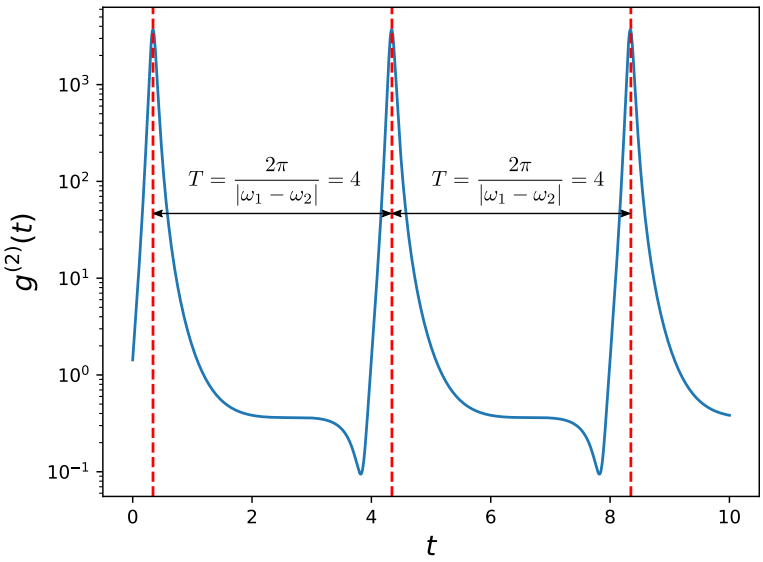

Figure 3.6: The cavity-atom-driven case with different driving frequencies.

3.3.2 Dynamical ETCF

Here, the criteria of dynamical ETCF is denoted as the system reduced density matrix is time-dependent even after long-time evolution. In other words, if there is multiple coherent state inputs with the unequal driving frequencies, the system reduced density matrix will satisfy the criteria. Meanwhile, the usage of the parameter `Quantity` is shown in the table below:

`Quantity`	Equation	Name
`"c1c1"`	$g^{(2)}(t) = \frac{\langle\Psi_{\text{out}}\|c_1^\dagger(t)c_1^\dagger(t)c_1(t)c_1(t)\|\Psi_{\text{out}}\rangle}{\langle\Psi_{\text{out}}\|c_1^\dagger(t)c_1(t)\|\Psi_{\text{out}}\rangle^2} \neq g^{(2)}(0)$	2nd-order ETCF
`"c1c1c1"`	$g^{(3)}(t) = \frac{\langle\Psi_{\text{out}}\|c_1^\dagger(t)c_1^\dagger(t)c_1^\dagger(t)c_1(t)c_1(t)c_1(t)\|\Psi_{\text{out}}\rangle}{\langle\Psi_{\text{out}}\|c_1^\dagger(t)c_1(t)\|\Psi_{\text{out}}\rangle^3}$	3rd-order ETCF
`"c1c2"`	$g_c^{(2)}(t) = \frac{\langle\Psi_{\text{out}}\|c_1^\dagger(t)c_2^\dagger(t)c_1(t)c_2(t)\|\Psi_{\text{out}}\rangle}{\langle\Psi_{\text{out}}\|c_1^\dagger(t)c_1(t)\|\Psi_{\text{out}}\rangle\langle\Psi_{\text{out}}\|c_2^\dagger(t)c_2(t)\|\Psi_{\text{out}}\rangle}$	2nd-order cross ETCF
`"c1c1c2"`	$g_c^{(3)}(t) = \frac{\langle\Psi_{\text{out}}\|c_1^\dagger(t)c_1^\dagger(t)c_2^\dagger(t)c_1(t)c_1(t)c_2(t)\|\Psi_{\text{out}}\rangle}{\langle\Psi_{\text{out}}\|c_1^\dagger(t)c_1(t)\|\Psi_{\text{out}}\rangle^2\langle\Psi_{\text{out}}\|c_2^\dagger(t)c_2(t)\|\Psi_{\text{out}}\rangle}$	3nd-order cross ETCF
⋮		

Example: Jaynes-Cummings Model

Seeing the section 3.3.1 about detailed description. Here, the only difference is the driving frequencies, i.e., $\omega_1 \neq \omega_2$. Thus, the higher-order ETCFs are time-dependent, and these formulas are as follows:

$$g^{(2)}(t) = \frac{\langle\Psi_{\text{out}}|c_1^\dagger(t)c_1^\dagger(t)c_1(t)c_1(t)|\Psi_{\text{out}}\rangle}{\langle\Psi_{\text{out}}|c_1^\dagger(t)c_1(t)|\Psi_{\text{out}}\rangle^2} = \frac{\text{Tr}\left[a^\dagger a^\dagger a a \rho_s(t+\infty)\right]}{\text{Tr}[a^\dagger a \rho_s(t+\infty)]^2}, \tag{3.25}$$

$$g_c^{(2)}(t) = \frac{\langle\Psi_{\text{out}}|c_1^\dagger(t)c_2^\dagger(t)c_1(t)c_2(t)|\Psi_{\text{out}}\rangle}{\prod_{k=1}^2\langle\Psi_{\text{out}}|c_k^\dagger(t)c_k(t)|\Psi_{\text{out}}\rangle} = \frac{\text{Tr}\left[a^\dagger\sigma^\dagger a\sigma\rho_s(t+\infty)\right]}{\text{Tr}[a^\dagger a\rho_s(t+\infty)] \times \text{Tr}[\sigma^\dagger\sigma\rho_s(t+\infty)]}, \tag{3.26}$$

where ρ_s represents the reduced density matrix of the system.

The Python code of the cavity-atoms-driven case with different driving frequencies:

```
from qcs_phy import qcs
import matplotlib.pyplot as plt
import numpy as np

# The system parameters
ome_c = 0 # the frequenc of cavity
ome_e = 0 # the frequenc of atom
kap = 1   # the decay rate of cavity
gam = 0.1 # the decay rate of atom
g = 0.7   # the coupling strength between cavity and atom
eta = 1 # the ratio of the driving strengths, Ω2/Ω1 = η
ome_1 = np.pi / 4  # the driving frequency of cavity
ome_2 = -np.pi / 4 # the driving frequency of atom

# construct the effective Hamiltonian
Heff = []
Heff.append([ome_c-1j*kap/2, ("ad", 0), ("a", 0)])
Heff.append([ome_e-1j*gam/2, ("sp", 0), ("sm", 0)])
Heff.append([g, ("ad", 0), ("sm", 0)])
Heff.append([g, ("a", 0), ("sp", 0)])

# construct input and output channels
o_b1 = [np.sqrt(kap/2), ("a", 0)]
o_b2 = [np.sqrt(gam/2), ("sm", 0)]
o_c1 = o_b1
Input_1 = qcs.Input_channel("b1", o_b1, ome_1)
Input_2 = qcs.Input_channel("b2", o_b2, ome_2)
Input = [Input_1, Input_2]
Output = qcs.Output_channel("c1", o_c1)

# construct main body
ratio = [1, eta * np.sqrt(kap / gam)] # the additional parameter
system = qcs(Heff, Input, Output, ratio)
# calculate 2nd-order ETCF
tlist = np.linspace(0, 10, 1000) # the time variable
g2_t = system.calculate_quantity("c1c1", tlist)
# drawing
plt.semilogy(tlist, g2_t)
plt.xlabel(r"$t$", size=15)
plt.ylabel(r"$g^{(2)}(t)$", size=15)
plt.show()
```

The result is shown in Fig. 3.6. Obviously, the dynamical second-order ETCF is a periodic function, and the period is equal to $2\pi/|\omega_1 - \omega_2|$. The reason is that the driven Hamiltonian also is a periodic function.

Note: In the current 1.0.4 release, we have not provided the function calculating many-to-many case, such as Eq. (3.26).

3.3.3 The single-photon transimission and reflection

Single coherent state input

Here, we only consider a coherent state input, and the corresponding channel mode is set to b_1, which the name could be represented by `"b1"` in Python. Thus, we have the following table, i.e.,

Quantity	Equation	Name
`"cj"`	$T_j = \frac{\langle\Psi_{\text{out}}\|c_j^\dagger(t)c_j(t)\|\Psi_{\text{out}}\rangle}{\langle\Psi_{\text{in}}\|b_1^\dagger(t)b_1(t)\|\Psi_{\text{in}}\rangle}$	Transimission
`"b1"`	$R = \frac{\langle\Psi_{\text{out}}\|b_1^\dagger(t)b_1(t)\|\Psi_{\text{out}}\rangle}{\langle\Psi_{\text{in}}\|b_1^\dagger(t)b_1(t)\|\Psi_{\text{in}}\rangle}$	Reflection
$\vdots$		

Multiple coherent state inputs

Here, we consider m coherent state inputs, and the corresponding channel modes are $b_1, b_2, \ldots, b_m$, which their names could be represented by `"b1"`, `"b2"`, `...`, `"bm"`, respectively. Thus, we have the following table, i.e.,

Quantity	Equation	Name
`"cj"`	$T_j = \frac{\langle\Psi_{\text{out}}\|c_j^\dagger(t)c_j(t)\|\Psi_{\text{out}}\rangle}{\sum_{k=1}^{m}\langle\Psi_{\text{in}}\|b_k^\dagger(t)b_k(t)\|\Psi_{\text{in}}\rangle}$	Transimission
`"b1"`	$R = \frac{\langle\Psi_{\text{out}}\|b_1^\dagger(t)b_1(t)\|\Psi_{\text{out}}\rangle}{\sum_{k=1}^{m}\langle\Psi_{\text{in}}\|b_k^\dagger(t)b_k(t)\|\Psi_{\text{in}}\rangle}$	Reflection
$\vdots$		

Note: For the Hamiltonian driving a collective spins like Eq. (3.17), the corresponding input-output formalism like Eq. (3.18) should not be treated as single input channel, rather than N (the number of atoms) input channels, and this only influences the denominator of transimission and reflection equations above, as shown in Eq. (3.27) and Eq. (3.28).

Finally, the description of the steady or dynamical single-photon transimission and reflection is in agreement with the two sections (3.3.1, 3.3.2) above.

Example1: Tavis-Cummings Model

Seeing the section 3.3.1 about detailed description. Here, the single-photon transimission expression of the cavity is as followss:

$$T_c = \frac{\langle\Psi_{\text{out}}|c_1^\dagger(t)c_1(t)|\Psi_{\text{out}}\rangle}{\langle\Psi_{\text{in}}|b_1^\dagger(t)b_1(t)|\Psi_{\text{in}}\rangle}, \quad T_a = \frac{\langle\Psi_{\text{out}}|c_1^\dagger(t)c_1(t)|\Psi_{\text{out}}\rangle}{\langle\Psi_{\text{in}}|Nb_2^\dagger(t)b_2(t)|\Psi_{\text{in}}\rangle}, \tag{3.27}$$

$$T_{ca} = \frac{\langle\Psi_{\text{out}}|c_1^\dagger(t)c_1(t)|\Psi_{\text{out}}\rangle}{\langle\Psi_{\text{in}}|[b_1^\dagger(t)b_1(t)+Nb_2^\dagger(t)b_2(t)]|\Psi_{\text{in}}\rangle}, \tag{3.28}$$

where the subscripts c, a, and ca represent the cavity-driven, atoms-driven, and cavity-atoms-driven cases, respectively. Meanwhile, the channel modes b_1, b_2, and c_1 are described by Eq. (3.18) and Eq. (3.19).

The Python code of the cavity-driven, atoms-driven, and cavity-atoms-driven (with identical driving frequencies) cases:

```
from qcs_phy import qcs
import matplotlib.pyplot as plt
import numpy as np

# The system parameters
ome_c = 0 # the frequency of cavity
ome_e = 0 # the frequency of atom
kap = 1 # the decay rate of cavity
gam = 0.01 # the decay rate of atom
g = 2 # the coupling strength between cavity and atom
ome_1 = np.linspace(-25, 25, 4000) # the driving frequency of cavity mode
N_list = [2, 10, 20, 50, 100] # the number of emitters
for N in N_list:
    # construct the effective Hamiltonian
    Heff = []
    Heff.append([ome_c-1j*kap/2, ("ad", 0), ("a", 0)])
    Heff.append([ome_e-1j*gam/2, ("Sz N=%s" % N, 0)])
    Heff.append([g, ("ad", 0), ("Sm N=%s" % N, 0)])
    Heff.append([g, ("a", 0), ("Sp N=%s" % N, 0)])

    # construct input and output channels
    o_b1 = [np.sqrt(kap/2), ("a", 0)]
    o_c1 = o_b1
    Input_1 = qcs.Input_channel("b1", o_b1, ome_1)
    Output = qcs.Output_channel("c1", o_c1)
    # construct main body
    system = qcs(Heff, Input_1, Output)
    # calculate physical quantities
    T = system.calculate_quantity("c1")
    # drawing
    plt.semilogy(ome_1, T, label=r'$N = %s$' % N)
plt.xlabel(r"$\omega_1$", size=15)
plt.ylabel(r"$T$", size=15)
plt.legend(prop={'size': 15})
plt.show()
```

For the atoms-driven case, we need to make one adjustment in the Python code above, i.e.,

```
o_b2 = [np.sqrt(gam/2), ("Sm N=%s" % N, 0)]
Input_2 = qcs.Input_channel("b2", o_b2, ome_1)
system = qcs(Heff, Input_2, Output)
```

Similarly, for the cavity-atoms-driven case, we also have

```
ome_1 = np.linspace(-125, 50, 4000) # the driving frequency of
                                    # the cavity and atom modes
eta = 0.5 # the ratio of the driving strengths, Ω2/Ω1 = η
ratio = [1, eta * np.sqrt(kap / gam)] # the additional parameter
system = qcs(Heff, [Input_1, Input_2], Output, ratio)
```

These results are shown in Fig. 3.7.

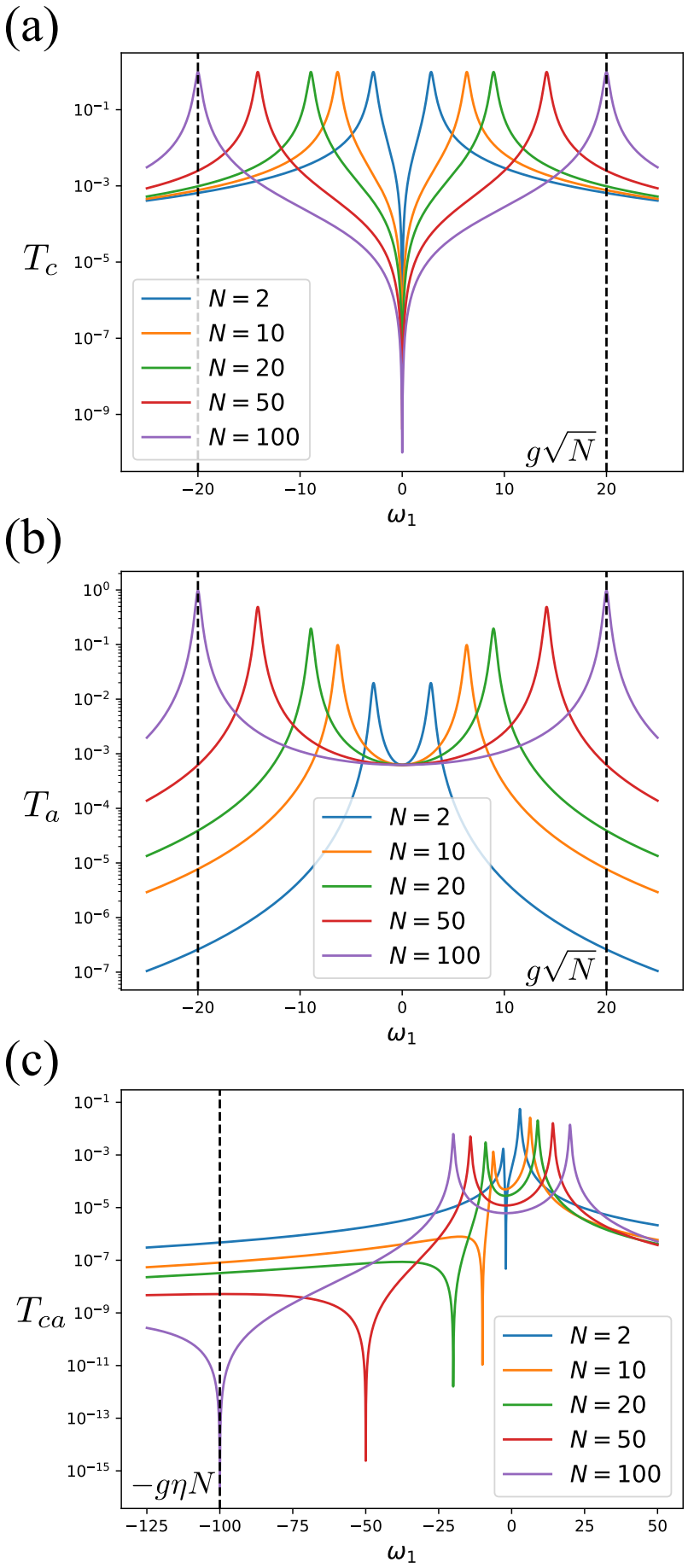

Figure 3.7: (a) The cavity-driven case. (b) The atoms-driven case. (c) The cavity-atoms-driven case with identical driving frequencies.

Example2: The Waveguide Quantum Electrodynamics System

A schematic of the considered system is shown in Fig. 3.8. Here, the system Hamiltonian is $H_{\rm sys} = \omega_e \sum_{j=1}^{N} \sigma_j^\dagger \sigma_j$, and the corresponding Lindblad master equation is

$$\begin{aligned}\dot{\rho} &= -i[H_{\rm sys} + H_{\rm coh}, \rho] + \gamma \sum_{j,l} \cos(k|x_j - x_l|)(\sigma_l \rho \sigma_j^\dagger - \frac{1}{2}\{\sigma_j^\dagger \sigma_l, \rho\}) \\ &= -i[H_{\rm eff}\rho - \rho H_{\rm eff}^\dagger] + \gamma \sum_{j,l} \cos(k|x_j - x_l|)\sigma_l \rho \sigma_j^\dagger, \end{aligned} \tag{3.29}$$

where $H_{\rm coh} = (\gamma/2)\sum_{j,l} \sin(k|x_j - x_l|)\sigma_j^\dagger \sigma_l$ and $H_{\rm eff} = H_{\rm sys} - i\frac{\gamma}{2}\sum_{j,l} e^{ik|x_j - x_l|}\sigma_j^\dagger \sigma_l$. Now, we consider an coherent state entering from the left side of waveguide, and the process is equivalent to a coherently driven Hamiltonian, i.e.,

$$H_d = \sum_{j=1}^{N} [\beta\sqrt{\gamma/4\pi}\exp(i\omega_d t - ikx_j)\sigma_j + \beta^*\sqrt{\gamma/4\pi}\exp(-i\omega_d t + ikx_j)\sigma_j^\dagger] \tag{3.30}$$

where $|\beta| \to 0$. Based on the discussions above, the input-output formalism of the right (b_r) and left (b_l) moving waveguide modes could be written as

$$b_{r,\rm out}(t) = b_{r,\rm in}(t) - i\sqrt{\gamma/2}\sum_{j=1}^{N} e^{-ikx_j}\sigma_j(t),\ b_{l,\rm out}(t) = b_{l,\rm in}(t) - i\sqrt{\gamma/2}\sum_{j=1}^{N} e^{+ikx_j}\sigma_j(t), \tag{3.31}$$

which implies $o_{b_r} = \sqrt{\gamma/2}\sum_{j=1}^{N} e^{-ikx_j}\sigma_j$ and $o_{b_l} = \sqrt{\gamma/2}\sum_{j=1}^{N} e^{+ikx_j}\sigma_j$.

Due to $N \gg 1$ and long-range interactions, constructing the effective Hamiltonian and input-output channels need to use for-loop structure in Python code, and this will improve our efficiency in writing Python code. Here, we will show the specific code to construct $H_{\rm eff}$, o_{b_r}, and o_{b_l}, i.e.,

```
# construct the effective Hamiltonian
Heff = []
for j in range(N):
    Heff.append([ω_e, ("sp", j), ("sm", l)])
    for l in range(N):
        phi = np.exp(1j * np.abs(x[j] - x[l]) * k) # conciseness
        Heff.append([-1j * phi * γ / 2, ('sp', j), ('sm', l)])
```

```
# construct the input-output modes, o_{b_r} and o_{b_l}
# Plan-1
o_br = [[np.sqrt(γ/2) * np.exp(-1j*x[j]*k), ('sm', j)] for j in range(N)]
o_bl = [[np.sqrt(γ/2) * np.exp(+1j*x[j]*k), ('sm', j)] for j in range(N)]
# Plan-2
o_br = []
o_bl = []
for j in range(N):
    o_br.append([np.sqrt(γ/2) * np.exp(-1j * x[j] * k), ("sm", j)])
    o_bl.append([np.sqrt(γ/2) * np.exp(+1j * x[j] * k), ("sm", j)])
```

To proceed, we will calculate the single-photon transimission and reflection, i.e.,

$$T = \frac{\langle\Psi_{\rm out}|b_r^\dagger(t)b_r(t)|\Psi_{\rm out}\rangle}{\langle\Psi_{\rm in}|b_r^\dagger(t)b_r(t)|\Psi_{\rm in}\rangle},\ R = \frac{\langle\Psi_{\rm out}|b_l^\dagger(t)b_l(t)|\Psi_{\rm out}\rangle}{\langle\Psi_{\rm in}|b_r^\dagger(t)b_r(t)|\Psi_{\rm in}\rangle}. \tag{3.32}$$

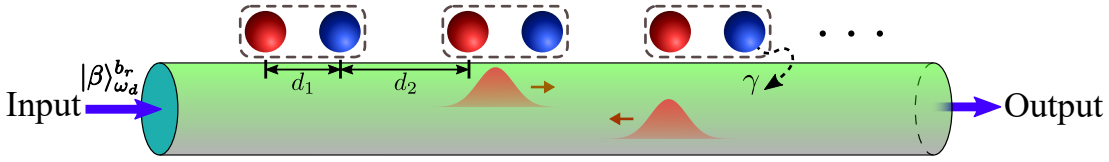

Figure 3.8: Schematic of a dimer atom chain coupled to an one-dimensional waveguide.

The Python code of the single-photon transimission and reflection:

```
from qcs_phy import qcs
import matplotlib.pyplot as plt
import numpy as np

N = 20     # the number of atoms
ome_e = 0 # the transition frequency of atom
gam = 1    # the decay rate of atom
k = 2 * np.pi # the wave vector of the resonant waveguide modes
d_1, d_2 = 1/4, 3/4 # the intra-cell and inter-cell distances, d_1, d_2
# the locations of atoms
x = [j*d_1 + i*(d_1+d_2) for i in range(int(N / 2)) for j in range(2)]
ome_d = np.linspace(-1, 1, 4000) # the driving frequency
# construct the effective Hamiltonian
Heff = []
for i in range(N):
    Heff.append([ome_e, ("sp", i), ("sm", i)])
    for j in range(N):
        phi = np.exp(1j * np.abs(x[i] - x[j]) * k)
        Heff.append([-1j * phi * gam / 2, ('sp', i), ('sm', j)])
# construct input and output channels
o_br = [[np.sqrt(gam/2)*np.exp(-1j*x[j]*k), ('sm', j)] for j in range(N)]
o_bl = [[np.sqrt(gam/2)*np.exp(+1j*x[j]*k), ('sm', j)] for j in range(N)]
Input = qcs.Input_channel('br', o_br, ome_d)
Output_1 = qcs.Output_channel('br', o_br)
Output_2 = qcs.Output_channel('bl', o_bl)
Output = [Output_1, Output_2]
# construct main body
system = qcs(Heff, Input, Output)
# calculate physical quantities
T = system.calculate_quantity("br")
R = system.calculate_quantity("bl")
# calculate eigenvalues of the effective Hamiltonian
Heff_matrix = system.print_Heff(1)
lm, _ = np.linalg.eig(Heff_matrix)
# drawing
plt.plot(ome_d, T, label=r'$T$')
plt.plot(ome_d, R, label=r'$R$')
plt.plot(np.real(lm), np.ones((N, 1))*1.02,'.',label=r'Re($\lambda_n$)')
plt.xlabel(r"$\omega_d$", size=15)
plt.legend(prop={'size': 15})
plt.show()
```

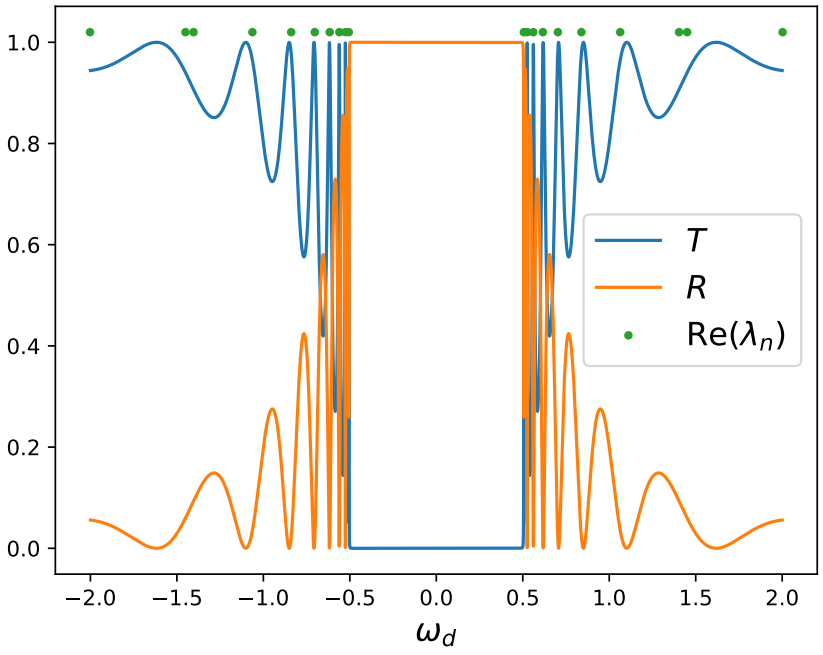

Figure 3.9: The single-photon transimission and reflection in an one-dimensional waveguide.

The result is shown in Fig. 3.9. Firstly, the transimission and reflection have a common window, i.e., $T = 0$ and $R = 1$ within the window. Secondly, the real part of all eigenvalues of the effective Hamiltonian also has a construction similar to band gap, and the width of the band gap almost is equal to the window. Thus, when the frequency ω_d of incoming photon is within the range, i.e., $[-\min|\mathrm{Re}(\lambda_n)| + \omega_e,\ \min|\mathrm{Re}(\lambda_n)| + \omega_e]$, the transmission and reflection are almost zero and one, respectively, as shown in Fig. 3.9. Note that λ_n represents all the eigenvalues of the effective Hamiltonian. For a detailed discussion about the width of the window, please refer to our paper: http://arxiv.org/abs/2305.08923.

3.4 The Other Helpful Functions On Qcs Class

3.4.1 The second-order unequal-time correlation function

The usage of this function is similar to the higher-order ETCF 3.3, i.e.,

```
system.calculate_2nd_uETCF(Quantity, tau, zp=0)
```

- The first parameter (`Quantity`) represents the physical quantity in the form of `str`.
- The second parameter (`tau`) represents the delay time τ.
- The third parameter (`zp`) represents the coefficient 0^+. Default is zero.

Meanwhile, the usage of the parameter `Quantity` is shown in the table below:

Quantity	Equation	Name
`"c1c1"`	$g_1^{(2)}(\tau) = \frac{\langle\Psi_{\mathrm{out}}\|c_1^\dagger(t)c_1^\dagger(t+\tau)c_1(t+\tau)c_1(t)\|\Psi_{\mathrm{out}}\rangle}{\langle\Psi_{\mathrm{out}}\|c_1^\dagger(t)c_1(t)\|\Psi_{\mathrm{out}}\rangle^2}$	The 2nd-order uETCF
⋮	⋮	
`"cncn"`	$g_n^{(2)}(\tau) = \frac{\langle\Psi_{\mathrm{out}}\|c_n^\dagger(t)c_n^\dagger(t+\tau)c_n(t+\tau)c_n(t)\|\Psi_{\mathrm{out}}\rangle}{\langle\Psi_{\mathrm{out}}\|c_n^\dagger(t)c_n(t)\|\Psi_{\mathrm{out}}\rangle^2}$	

Subsequently, we take the same example 3.3.1 [Jaynes-Cummings Model] to illustrate the specific usage, and the 2nd-order uETCF of the cavity mode satisfies the relation below:

$$g^{(2)}(\tau)=\frac{\langle\Psi_{\text{out}}|c_1^{\dagger}(t)c_1^{\dagger}(t+\tau)c_1(t+\tau)c_1(t)|\Psi_{\text{out}}\rangle}{\langle\Psi_{\text{out}}|c_1^{\dagger}(t)c_1(t)|\Psi_{\text{out}}\rangle^2}=\frac{\text{Tr}\left[a^{\dagger}(t)a^{\dagger}(t+\tau)a(t+\tau)a(t)\rho_{\text{ss}}\right]}{\text{Tr}[a^{\dagger}(t)a(t)\rho_{\text{ss}}]^2}. \quad (3.33)$$

The Python code of the cavity-driven case:

```
from qcs_phy import qcs
import matplotlib.pyplot as plt
import numpy as np

# The system parameters
ome_c = 0 # the frequency of cavity
ome_e = 0 # the frequency of atom
kap = 1   # the decay rate of cavity
gam = 1   # the decay rate of atom
g = 0.7   # the coupling strength between cavity and atom
ome_1 = 0 # the driving frequency of cavity mode

# construct the effective Hamiltonian
Heff = []
Heff.append([ome_c-1j*kap/2, ("ad", 0), ("a", 0)])
Heff.append([ome_e-1j*gam/2, ("sp", 0), ("sm", 0)])
Heff.append([g, ("ad", 0), ("sm", 0)])
Heff.append([g, ("a", 0), ("sp", 0)])

# construct input and output channels
o_b1 = [np.sqrt(kap/2), ("a", 0)]
o_c1 = o_b1

Input = qcs.Input_channel("b1", o_b1, ome_1)
Output = qcs.Output_channel("c1", o_c1)

# construct main body
system = qcs(Heff, Input, Output)

# calculate physical quantities
tau = np.linspace(-10, 10, 1000) # the delay time
g2_tau = system.calculate_2nd_uETCF("c1c1", tau)

# drawing
plt.plot(tau, g2_tau)
plt.xlabel(r"$\tau$", size=15)
plt.ylabel(r"$g^{(2)}(\tau)$", size=15)
plt.show()
```

The result is shown in Fig. 3.10.

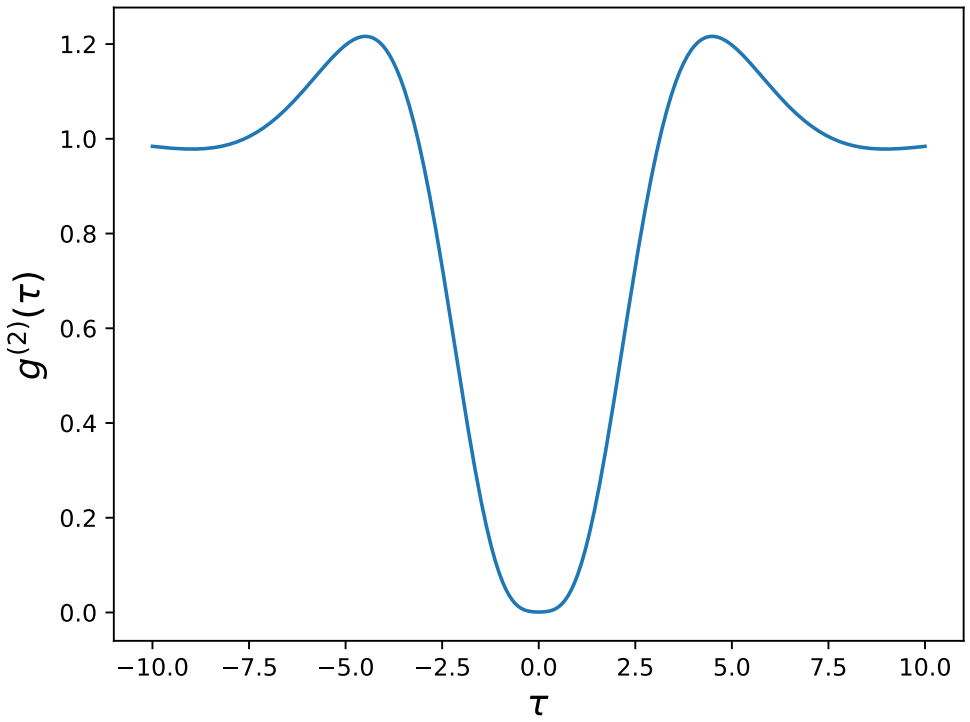

Figure 3.10: The second-order unequal-time correlation function of cavity mode

3.4.2 Print all basis vectors under a specific excitation number subspace

Here, the function, `print_basis`, could generate automatically all basis vectors for any excitation number subspace after we provided the effective Hamiltonian. Thus, the information of input and output channels is not necessary if we only want to obtain the basis vector and the effective Hamiltonian. Subsequently, the specific usage about this function is shown below:

```
system = qcs(Heff, [], [])
inf, basis = system.print_basis(n_exc)
```

- `n_exc` [`int`]: the excitation number.
- `basis` [`list`]: all basis vectors.
- `inf` [`list`]: the information of basis vector.

Then, we take a simple example (Jaynes-Cummings Model) to illustrate its meaning, i.e.,

```
In [1]: Heff = []
In [2]: Heff.append([g, ("ad", 0), ("sm", 0)])
In [3]: Heff.append([g, ("a", 0), ("sp", 0)])
In [4]: system = qcs(Heff, [], [])
In [5]: inf, basis = system.print_basis(1)

Out [1]: inf
[("a", 0), ("sm", 0)]
Out [2]: basis
[[1, 0], [0, 1]]
```

Under the single excitation number subspace, there are two basis vectors, `[1, 0]` and `[0, 1]`. For any basis vector, the i-th element represents the occupation number of the corresponding i-th element in `inf`. More specifically, we have

`[1, 0]` $\equiv a_0^\dagger \lvert vacuum\rangle$, `[0, 1]` $\equiv \sigma_0^\dagger \lvert vacuum\rangle$

Note that the sequence of basis vector in `basis` will influence matrix form (the sequences of rows and colums) of the effective Hamiltonian.

3.4.3 Print the effective Hamiltonian in the form of matrix

```
system = qcs(Heff, [], [])
H = system.print_Heff(n_exc)
```

- `n_exc [int]`: the excitation number.
- `H [ndarray]`: the projection of the effective Hamiltonian on the `n_exc`-th excitation subspace.

For the sake of convenience, we use the same example as above 3.4.2, and we have

```
In [6]: H = system.print_Heff(1)
Out [3]: H
array([[0, g]
       [g, 0]])
```

3.4.4 Print the input/output modes in the form of matrix

Here, we must simultaneously provide the effective Hamiltonian and input/output channels, i.e.,

```
system = qcs(Heff, Input, Output)
O = system.print_InOutput(n_exc, channel_name)
```

- `n_exc [int]`: the excitation number.
- `channel_name [str]`: the channel name, such as `"b"`, `"c"`.
- `O [ndarray]`: the projection of the input/output modes onto the direct sum of the (`n_exc`-1)-th and `n_exc`-th excitation subspace.

For the sake of convenience, we still use the Jaynes-Cummings model as above 3.4.2, and consider the input-output formalism below:

$$b_{\text{out}}(t) = b_{\text{in}}(t) - io_b(t),\ c_{\text{out}}(t) = c_{\text{in}}(t) - io_c(t), \tag{3.34}$$

Here, $o_b = \sqrt{\kappa}a_0$ and $o_c = \sqrt{\gamma}\sigma_0$, which represent the input mode and output mode, respectively. Thus, we have

```
In [7]: o_b = [√κ, ("a", 0)]
In [8]: o_c = [√γ, ("sm", 0)]
In [9]: Input = qcs.Input_channel("b", o_b, ω_d)
In [10]: Output = qcs.Output_channel("c", o_c)
In [11]: system = qcs(Heff, Input, Output)
In [12]: O_b = system.print_InOutput(1, "b")
In [13]: O_c = system.print_InOutput(1, "c")

Out [4]: O_b
array([[√κ, 0]])
Out [5]: O_c
array([[0, √γ]])
```

4. API documentation

This chapter contains automatically generated API documentation, including a complete list of QCSs public classes and functions. However, the API document in this PDF consists of a series of pictures, which originate from Github, and you can preview the API.html file, i.e., API.

Module **qcs_phy**

Quantum Correlation Solver (QCS) This module calculates the nth-order equal-time correlation functions, single-photon transmission and reflection in open quantum systems. First, the effective Hamiltonian must satisfy U(1) symmetry, namely the total excitation number conservation. Second, the incoming coherent state amplitude must be small enough, namely the weak driving approximation. Finally, this module allows multiple incoming coherent states no matter their frequencies are identical or not, namely the multi-drive case, and could be also used to calculate the cross-correlation function and the 2nd-order unequal-time correlation function.

▶ EXPAND SOURCE CODE

Functions

```
def basis_dot(m1: list, m2: numpy.ndarray, coff: numpy.ndarray) -> scipy.sparse._csr.csr_matrix
```

:param `m1`: fixed basis vectors

:param `m2`: updated basis vectors after acted by all possible modes

:param `coff`: the coefficient created by all possible modes acted on fixed basis vectors

:return a "dot" product between fixed basis vectors and updated basis vectors

▶ EXPAND SOURCE CODE

```
def compare_dicts(dict_a: dict, dict_b: dict) -> bool
```

In order to compare two dicts

:param `dict_a`: dict a

:param `dict_b`: dict b

:return True or False

▶ EXPAND SOURCE CODE

```
def covert_to_decimals(num_list: list, m: int) -> list
```

:param `num_list`: list

:param `m`: the number of input modes

:return list

▶ EXPAND SOURCE CODE

```
def create_basis(n_exc: int, max_dims: list, k: list) -> list
```

:param `n_exc`: the excitation number

:param `max_dims`: the maximum dimension of each mode

:param `k`: the corresponding coefficient of each mode in the total excitation number operator

:return all the possible basis vectors

▶ EXPAND SOURCE CODE

```
def dagger(ope: str) -> str
```

:param `ope` : an operator o

:return $o^{\dagger}$

▶ EXPAND SOURCE CODE

```
def left_prod(C: list) -> numpy.ndarray
```

:param `C` : a list including multiple matrices, corresponding to the output operators in different excitation numbers

:return the matrices product of all matrices in the list

▶ EXPAND SOURCE CODE

```
def n_m_ary(n: int, m: int) -> list
```

:param `n` : the total excitation number

:param `m` : the number of the input modes

:return list

▶ EXPAND SOURCE CODE

```
def right_prod(H: list, B: list, I: list, ome: float, zp: float) -> numpy.ndarray
```

:param `H` : a list including multiple matrices, corresponding to the effective Hamiltonian in different excitation numbers

:param `B` : a list including multiple matrices, corresponding to the input operators in different excitation numbers

:param `I` : a list including multiple matrices, corresponding to the identity operators in different excitation numbers

:param `ome` : the frequency of the incoming photon

:param `zp` : a coefficient for ensuring the invertibility of effective Hamiltonian

:return a matrix acquired by a series of matrices multiplicatopm above

▶ EXPAND SOURCE CODE

```
def sum_frequencies(omega: list, n: int) -> list
```

:param `omega` : the frequencies of incoming photons, the corresponding type: list or np.ndarray

:param `n` : the total excitation number

:return all possible frequencies combination in excitation number n

▶ EXPAND SOURCE CODE

```
def sum_sparse(m: list) -> numpy.ndarray
```

:param `m` : a list including multiple sparse matrices

:return the summation of all sparse matrices in the list

▶ EXPAND SOURCE CODE

```
def update_H(H: list) -> list
```

:param `H`: the effective Hamiltonian, list

:return the nonredundant Hamiltonian terms

▶ EXPAND SOURCE CODE

Classes

```
class qcs (Heff: list, Input: list, Output: list, ratio=None)
```

Here, for each term in the effective Hamiltonian, such as $H_{eff} = Ea_1^\dagger a_1$, we use a list to represent it, i.e.,

```
Heff = [E, ("ad", 1), ("a", 1)],
```

where the first element represents the corresponding coefficient, and the last two elements represent the operator $a_1^\dagger$ and a_1, respectively.

For example, $H_{eff} = Ea_1^\dagger a_1 + Ua_1^\dagger a_1^\dagger a_1 a_1$, and the python code is

```
Heff = [[E, ("ad", 1), ("a", 1)], [U, ("ad", 1), ("ad", 1), ("a", 1), ("a", 1)]].
```

Note that the first and second elements in `("ad", 1)` represent the operator and the corresponding subscript, respectively.

More importantly, we only give three symbols to represent the corresponding system's operators:

```
"ad" ==> bosonic creation operator, such as cavity field mode

"a"  ==> bosonic annilihlation operator, such as cavity field mode

"sp" ==> raising operator: |e><g|, such as two-level spin

"sm" ==> lowering operator: |g><e|, such as two-level spin

"Sp N=M" ==> Sp_M = \sum_{i=1}^{M}{sp_i}, M collective two-level spins

"Sm N=M" ==> Sm_M = \sum_{i=1}^{M}{sm_i}, M collective two-level spins

"Sz N=M" ==> Sz_M = \sum_{i=1}^{M}{sp_i*sm_i}, M collective two-level spins
```

Meanwhile, the Input and Output variables must be acquired by the two functions `Input_channel` and `Output_channel`, respectively.

And the ratio variable represents that each input channel has a corresponding coherent amplitude, e.g.,

$$b_1 \Rightarrow \beta_1, b_2 \Rightarrow \beta_2, b_3 \Rightarrow \beta_3,$$

and we have

$$ratio = [\beta_1, \beta_2, \beta_3],$$

where $\eta_k = \beta_k/\beta_1$. Of course, if there was only one input channel, we can ignore this variable, and the default value is `[1,1,…,1]`.

:param `Heff`: The effective Hamiltonian in the form of List[list].

:param `Input`: The input form is acquired by Input_channel function.

:param `Output`: The output form is acquired by Output_channel function.

:param `ratio`: The ratio between all input coherent amplitudes.

▶ EXPAND SOURCE CODE

Methods

```
def Input_channel(channel_name: str, mode: list, frequency) -> dict
```

Assuming `channel_name = "b1"`, the corresponding input-output formalism is

$$b_{1,out}(t) = b_{1,in}(t) - io_{b_1}.$$

For example, when $o_{b_1} = \sqrt{\kappa}a_1$, it corresponds to

```
o_b1 = [np.sqrt(κ), ("a", 1)].
```

Obviously, if o_{b_1} could consist of multiple system's operators, such as

$$o_{b_1} = \sqrt{\kappa_1}a_1 + \sqrt{\kappa_2}a_2,$$

which corresponds to

```
o_b1 = [[np.sqrt(κ_1), ("a", 1)], [np.sqrt(κ_2), ("a", 2)]]
```

:param `channel_name`: The input channel name, such as `'b1', 'b2'`, and etc.

:param `mode`: it consists of system's annihilation operator, e.g., `o_b1`.

:param `frequency [number, list, array]`: driving frequency or incoming photon frequency.

:return `dict`

▶ EXPAND SOURCE CODE

```
def Output_channel(channel_name: str, mode: list) -> dict
```

Assuming `channel_name = "c1"`, the corresponding input-output formalism is

$$c_{1,out}(t) = c_{1,in}(t) - io_{c_1}.$$

For example, when $o_{c_1} = \sqrt{\kappa}a_1$, it corresponds to

```
o_c1 = [np.sqrt(κ), ("a", 1)].
```

Obviously, if o_{c_1} could consist of multiple system's operators, such as

$$o_{c_1} = \sqrt{\kappa_1}a_1 + \sqrt{\kappa_2}a_2,$$

which corresponds to

```
o_c1 = [[np.sqrt(κ_1), ("a", 1)], [np.sqrt(κ_2), ("a", 2)]]
```

:param `channel_name`: The output channel name, such as `'c1', 'c2'`, and etc.

:param `mode`: It consists of system's annihilation operator, e.g., `o_c1`.

:return `dict`

▶ EXPAND SOURCE CODE

`def calculate_2nd_uETCF(self, Quantity: str, tau=0, zp=0)`

Calculating the 2nd-order unequal-time coreelation function . For example,

```
Quantity = "c1c1" ==> The 2nd-order unequal-time correlation function
```

:param `Quantity`: physical quantity

:param `tau [number, list, array]`: the delay time

:param `zp`: an infinitely small quantity

:return `[number, list]` the 2nd-order unequal-time coreelation function (uETCF). When the variable `frequency` or `tau` is `[number, list/array]`, the output also is `[number, list]`.

▶ EXPAND SOURCE CODE

`def calculate_quantity(self, Quantity: str, tlist=0, zp=0)`

Calculating a series of physical quantities. For example,

```
Quantity = "c1"     ==> The single-photon transmission

Quantity = "c1c1"   ==> The 2nd-order equal-time correlation function

Quantity = "c1c2"   ==> The 2nd-order equal-time cross-correlation function

Quantity = "c1c1c1" ==> The 3rd-order equal-time correlation function
```

Note that these physical quantities describe the statistical properties of output light in the output channel.

When input channel is different from the output channel, e.g., input channel `"b1"` and output channel `"c1"`, the physical quantity can represent the correlation function about system's modes based on the input-output formalism. For example, the input-output relation about output channel `"c1"` is given by

$$c_{1,out} = c_{1,in} - io_{c_1},$$

and we assume that $o_{c_1} = \sqrt{\kappa} a_1$ and `Quantity = "c1c1"`. The 2nd-order equal-time correlation function is equivalent to the correlation function of mode a_1.

Here, we consider the presence of tlist only when the frequencies of incoming coherent states are not identical, and consider `zp` only when the effective Hamiltonian is irreversible, i.e.,

$$[H_{eff}^{(n)} - \omega - i0^+]^{-1} \neq 0 \Rightarrow zp = i0^+$$

:param `Quantity` : physical quantity

:param `tlist [number, list, array]` : a time variable

:param `zp` : an infinitely small quantity

:return `[number, list]` the corresponding physical quantity. When the variable `frequency` or `tlist` is `[number, list/array]`, the output also is `[number, list]`

▶ EXPAND SOURCE CODE

```
def print_Heff(self, n_exc: int)
```

This function is used to input the correspondint effective Hamiltonian in the excitation subspace `n_exc`.

:param `n_exc` : the excitation number

:return the effective Hamiltonian in the excitation number (`n_exc`) subspace.

▶ EXPAND SOURCE CODE

```
def print_InOutput(self, n_exc: int, channel_name: str)
```

This function is used to print the matrix, which corresponds to the projections of the input/output mode onto the direct sum of the (`n_exc-1`)-th abd (`n_exc`)-th excitation subspace.

:param `n_exc` : the excitation number

:param `channel_name` : the name of channel

:return the input and output modes onto the direct sum of the (`n_exc-1`)-th abd (`n_exc`)-th excitation subspace.

▶ EXPAND SOURCE CODE

```
def print_basis(self, n_exc: int)
```

This function is used to print the basis vectors, which corresponds to excitation number `n_exc`.

:param `n_exc` : the excitation number

:return the information about the basis vector and all basis vectors in the excitation number (`n_exc`) subspace

▶ EXPAND SOURCE CODE

5. Change Log

5.1 Version 1.0.0 (May 10,2023)

First release

5.2 Version 1.0.1 (May 25,2023)

5.2.1 Improvements

- Functions `print_Dim`, `print_basis`, `print_InOutput`,`print_Heff` do not need the parameter p.

5.3 Version 1.0.2 (Jun 01,2023)

5.3.1 Bug Fixes

- Fixed the problem of the denominator about calculating the single-photon transmission and reflection when the collective spins are coherently driven.

5.4 Version 1.0.3 (Jun 02,2023)

5.4.1 Bug Fixes

- Fixed the formula of the single-photon transmission and reflection.

5.5 Version 1.0.4 (Jun 04,2023)

5.5.1 Improvements

- Updated the function `calculate_2nd_uETCF` in order to unify with the `calculate_Quantity`.

5.6 Version 1.0.5 (Jun 07,2023)

5.6.1 Improvements

- Deleted the function `print_Dim`, and updated the function `print_basis`.