# Symbolic Indices and Summations in QuantumCumulants.jl

**Masterarbeit**

zur Erlangung des akademischen Grades
Master of Science
(MSc)

eingereicht an der
Fakultät für Mathematik, Informatik und Physik
der Universität Innsbruck

von
**Julian Moser, BSc**

Betreuer:
Univ. Prof. Dr. Helmut Ritsch

Mitwirkender Betreuer:
Christoph Hotter, PhD

Institut für Theoretische Physik

Innsbruck, Juni 2023

# Abstract

Numerical simulation of the full quantum dynamics of open quantum systems is often limited by the sheer size of the underlying Hilbert space. Commonly, numerical approaches use matrix representations for operators and states, which grow exponentially in size with the number of subsystems involved. Hence, often approximation methods are needed to calculate important properties and the dynamics of wavefunctions of large quantum systems. Alternatively, one can work in the Heisenberg picture, and aim to determine the time evolution of desired operator expectation values via corresponding systems of coupled ordinary differential equations. In many cases, such an approach does not directly lead to a closed set of equations and therefore is not directly applicable. However, approximations to higher-order products of operators can be used in the form of the well-known cumulant expansion method. This allows truncating these infinite sets into smaller solvable ones by approximating higher-order quantum correlations by products of lower order.

Depending on the complexity of the system, deriving the required equations of motion for operator averages can be a difficult and laborious task still. A helpful tool to automate this process for quantum optics type systems is the open-source Julia package QuantumCumulants.jl. It is capable of automatic generation of the Quantum Langevin Equations for general system operator averages and application of the aforementioned cumulant expansion method up to a desired order. For this, it requires the user to define the Hamiltonian and the dissipative processes of the open quantum system by using the available building blocks in the package. Furthermore, one can directly obtain numerical solutions of the analytically derived equations using standard differential equation solvers.

QuantumCumulants.jl thus is a powerful tool to simulate open quantum systems. However, for the simulation of large systems, the prime time-limiting process has been the composition of the large number of required symbolic equations for each subsystem. So far, for quantum systems consisting of multiple equivalent subsystems, redundant calculus has been performed. Ensembles of atoms coupled to a cavity or a dipole-dipole interacting atomic array are common examples of such systems. In this work, we introduce a newly developed algorithm to avoid this redundancy and speed up the process by introducing symbolic summations and indices to the QuantumCumulants.jl toolbox. This allows to efficiently handle systems composed of many equivalent subsystems and generate the final equations in a much more compact form. To this end, we first introduce the basic underlying theoretical and computational concepts and then, we showcase the effectiveness of this extension and its practical use in several examples.

# Zusammenfassung

Die numerische Simulation der vollständigen Quantendynamik offener Quantensystemen ist meist limitiert aufgrund der Größe des unterliegenden Hilbertraums. Üblicherweise werden bei numerischen Ansätzen Matrizen für die Darstellung von Operatoren und Zustände verwendet, welche exponentiell mit der Anzahl an Untersystemen wachsen. Aus diesem Grund werden oft Näherungen angewandt, um wichtige Eigenschaften und die Dynamik von Wellenfunktionen von großen Systemen berechnen zu können. Als Alternative kann man auch im Heisenberg Bild Berechnungen durchführen und dadurch die zeitliche Entwicklung von gewünschten Operator-Erwartungswerten mittels Systemen von gewöhnlichen gekoppelten Differentialgleichungen darstellen. In vielen Fällen führt dies nicht zu einer geschlossenen Menge an Gleichungen und ist dadurch nicht direkt anwendbar. Allerdings lassen sich Näherungen für Operator Produkte von höherer Ordnung, wie beispielsweise die Cumulantenentwicklung, anwenden. Dies ermöglicht eine unendliche Menge an Gleichungen in eine kleinere lösbare Menge zu zerlegen, indem Quantenkorrelationen höherer Ordnung durch Produkte niedrigerer Ordnung genähert werden.

Je nach Komplexität des betrachteten Systems kann das Herleiten der Bewegungsgleichungen von Operator-Erwartungswerten schwer und mühsam sein. Ein Hilfsmittel, welches diesen Prozess für Quantenoptisch ähnlichen Systeme automatisiert, ist das Open-Source Julia Paket QuantumCumulants.jl. Jenes ist in der Lage, die Quantum Langevin Gleichungen für allgemeine Operatoren des Systems automatisch zu generieren und die zuvor genannte Cumulantenentwicklung bis zu einer gewissen Ordnung anzuwenden. Um dies zu erfüllen, muss ein Benutzer des Pakets zuerst mithilfe von den zur Verfügung gestellten Bausteinen einen Hamiltonian und dissipative Prozesse definieren. Zusätzlich ist es ebenfalls möglich, direkt eine numerische Lösung zu erhalten, durch Anwendung von standardmäßigen Differenzialgleichungs-Rechnern.

QuantumCumulants.jl ist daher ein nützliches Werkzeug für schnelle Berechnungen von offenen Quantensystemen. Allerdings war der Zeit limitierende Prozess das Herleiten der großen Anzahl an benötigten symbolischen Gleichungen für jedes Untersystem. Bisher wurden für Systeme, welche aus mehreren ähnlichen Untersystemen bestehen, redundante Berechnungen durchgeführt. Bekannte Beispiele für solche Systeme sind Ensembles von Atomen, welche mit einer Cavity interagieren, oder auch Dipol-Dipol wechselwirkende atomare Gitter. In dieser Arbeit stellen wir einen neuen Algorithmus vor, welcher diese Redundanz umgeht und die Geschwindigkeit des Herleitungsprozesses erhöht, indem symbolische Summen und Indexe in QuantumCumulants.jl eingeführt werden. Dies ermöglicht Systeme, welche aus mehreren gleichartigen Untersystemen bestehen, effizient

zu verarbeiten und die endgültigen Gleichungen in einer kompakteren Form darzustellen. Zu diesem Zweck werden wir zunächst die zugrundeliegenden theoretischen Konzepte vorstellen und zeigen dann die Effektivität dieser Erweiterung und deren praktische Anwendung in mehreren Beispielen.

# Danksagung

Als allererstes möchte ich bei Prof. Helmut Ritsch für die Möglichkeit, an diesem Projekt arbeiten zu können, bedanken. Zugleich möchte ich mich bei Christoph Hotter für die ausgezeichnete Betreuung und für das stetige Feedback bedanken. Ein besonderer Dank gilt auch allen anderen Mitgliedern der Arbeitsgruppe von Prof. Ritsch, welche mich mit offenen Armen aufgenommen haben.

Zusätzlich möchte ich mich bei allen meinen Freunden bedanken, welche ich während meines Studiums kennenlernen konnte: Francesco, Katharina, Lorenz, Martin, Phillip, Regina, Stefan, Thomas, Verena und Viktor, um nur einige beim Namen zu nennen. Ohne euch wäre das Studium nicht halb so toll gewesen. Im gleichen Zuge möchte ich noch meinen Freunden in Vorarlberg und dem Musikverein Satteins danken, spezieller Dank gilt hierbei Andreas, David, Manuel und Wolfgang. Auf euch ist immer Verlass.

Natürlich möchte ich auch noch meiner Familie danken, welche mir während des Studiums immer beiseite standen. Besonderer Dank gilt hierbei meiner Mutter, welche mir das Studium finanziell überhaupt ermöglicht hat.

# Contents

# Chapter 1.

# Introduction

The growing interest in the computational simulation of quantum systems gives rise to the development of different computational toolboxes. Multiple frameworks aim to solve or simulate elaborate systems in a reasonable timeframe. Several modern programming languages and packages provide a possibility for the numeric calculation of quantum optical systems, for example, QuTiP [1], the Quantum Optics toolbox in MATLAB [2] or QuantumOptics.jl [3]. These programs ease the simulation of different systems by providing a user with basic structures of quantum-mechanical calculus. They describe the system dynamics in the Schrödinger picture via matrix representation of operators and state vectors. Therefore, they have an intrinsic limit to system size due to the exponential growth of the matrix dimensions with the number of subsystems. Consequently, common packages which use this matrix representation are usually only applicable to describe the dynamics of rather small quantum systems. Typically, several different approximations are used to handle larger ones. As an alternative to the description in the Schrödinger picture, one can formulate the dynamics of open quantum systems with differential equations of quantum operators in the Heisenberg picture. Commonly, this description leads to a large, often even infinite, set of equations and therefore does not directly reduce the exponential growth of the Hilbert space. To this end, approximations like the cumulant expansion method are used to truncate the number of differential equations. This reliable practice is applicable in many cases [4–6].

Typically one derives a differential equation of a single Heisenberg operator average in an open quantum system with the so-called Quantum Langevin Equation (QLE) [7, 8]. When several expectation values of quantum operators are needed to describe the desired system of interest, many iterations of this process are necessary. Furthermore, to end up with a closed and complete set of differential equations, we perform a cumulant expansion on every higher-order operator product, which approximates the system by neglecting higher-order quantum correlations. Since this procedure can be very cumbersome and prone to errors, QuantumCumulants.jl was developed to automate it. The package is a helpful tool to quickly elaborate open quantum optical systems and was already used in several publications [4, 5, 9, 10]. It derives the differential equations by automatically calculating the Quantum Langevin Equation for user-defined systems specified by a Hamiltonian and dissipative processes. After deriving an initial set of

equations, an automatic completion and the cumulant expansion to a defined order is performed, to end up with a complete and solvable set. In particular, the program uses the cumulant expansion to rewrite higher-order products of quantum operators as summations of lower-order ones by neglecting higher quantum correlations. Even though QuantumCumulants.jl is a handy and powerful tool, it lacks optimisation for systems consisting of several similar subsystems, such as, for example, an ensemble of multiple atoms coupled to a cavity field. Such systems are commonly encountered in quantum optics [11, 12]. For these kinds of arrangements, the toolbox has so far performed redundant calculations as an equation for every operator acting on each subsystem is required. In principle, one can derive equations for systems, which can be described by multiple subsystems using indexed operators instead of individual ones. For example, equations for the operators $\sigma_1$, $\sigma_2$, ..., $\sigma_N$ can be expressed in a single equation for the operator $\sigma_i$ together with indexed variables and summations storing the different information of the subsystems. However, implementing such an abstract way of writing operators and equations into an algorithm using indices and summations is not as straightforward as one might think. As quantum operators inherit non-trivial rules of calculus, performing multiplications of different indexed operators requires the treatment of several special cases. This becomes especially clear when products also include summations. Therefore, deriving such indexed equations requires several complex and laborious implementations, which we will highlight in this work. To this end, we will first underline the basic physical concepts needed and give an introduction to Julia and QuantumCumulants.jl. Furthermore, we will showcase several examples and benchmark tests of this symbolic indexing extension.

# Chapter 2.

# Basic Concepts

In this section, we want to give a short introduction into the relevant theoretical concepts and definitions needed to understand the underlying functionalities of the framework. First, we define the quantum operators used to describe atomic and bosonic systems. Following up, we briefly summarize a derivation of the Quantum Langevin Equation (QLE) for open quantum systems. We continue by describing how one can construct a closed set of equations of motion to simulate the dynamics of an open quantum system. In this description, we will introduce the so-called cumulant expansion method, which is needed to truncate the possibly infinite set of equations to a finite solvable one. Finally, we will also give a quick overview of the physics of dipole-dipole interacting systems.

## 2.1. Atoms and light

In this section, we will introduce the concepts of quantum operators describing N-level atoms and quantum harmonical oscillators similar as in [7, 8]. Although one cannot represent all quantum systems with only these operators, a substantial amount of problems in quantum optics can be covered. The simplest case of a multi-level atom is the two-level atom, described by the ground state $|g\rangle$, excited state $|e\rangle$ and a dipole transition moment $\vec{d}$. A schematic drawing of such a two-level atom, which closely relates to a Spin 1/2 system, with its transition operators is shown in Fig. 2.1.

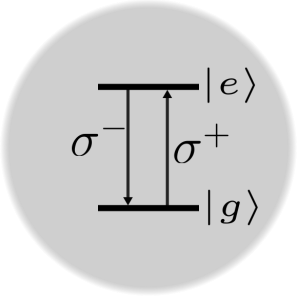


**Figure 2.1.:** *Schematic drawing of a two-level atom with the corresponding transition operators* $\sigma^+$ *and* $\sigma^-$.

Let us first define the atomic transition operators $\sigma^+$ and $\sigma^-$. The operator $\sigma^+$ corresponds to the excitation from the ground state $|g\rangle$ to the excited state $|e\rangle$ and one can therefore write $\sigma^+ = |e\rangle\langle g|$. The lowering operator $\sigma^-$ describes the inverse process, $\sigma^- = (\sigma^+)^\dagger = |g\rangle\langle e|$. When describing multi-level atoms, one would define such quantum operators for each possible transition within the atomic levels. By enumerating the atomic states in such a system with $\{|1\rangle, |2\rangle, ..., |N\rangle\}$, we can relate which transition an operator describes. For example, $\sigma^{21} = |2\rangle\langle 1|$ describes the transition from state $|1\rangle$ to state $|2\rangle$. When we encounter several transition operators acting at the same time, we can use the orthonormality of the atomic states to conduct

$$\sigma^{ij}\sigma^{kl} = |i\rangle\langle j|k\rangle\langle l| = \delta_{jk}\sigma^{il}, \tag{2.1.1}$$

where $\delta_{jk}$ is the Kronecker delta. A specific state $|i\rangle$ of a multi-level atom has the energy $E_i = \hbar\omega_i$, with $\omega_i$ being the transition frequency between the atomic ground state and the state $|i\rangle$. In this description, we choose, without loss of generality, that the ground state of an atom has zero energy, i.e. the frequency of the ground state $\omega_1$ is set to 0. The Hamiltonian describing the energy of the entire N-level atom is then a summation of all populated states with their energies and is given as

$$H = \hbar\sum_{i=2}^{N}\omega_i\sigma^{ii}. \tag{2.1.2}$$

Similarly, different states of a light field can be expressed by a number or Fock state $|n\rangle$. In this case, $n \in \mathbb{N}_0$ represents the number of photons within the field mode described by a wave vector $\vec{k}$ and polarisation $\lambda$. Equivalent to the general quantum harmonic oscillator, such a light field can be described with the raising-operator $a^\dagger_{\vec{k},\lambda}$ and lowering-operator $a_{\vec{k},\lambda}$. These have the following properties when acting on a number state $|n\rangle$

$$a|n\rangle = \sqrt{n}\,|n-1\rangle \tag{2.1.3}$$

$$a^\dagger|n\rangle = \sqrt{n+1}\,|n+1\rangle \tag{2.1.4}$$

with their commutator relation

$$[a_{\vec{k},\lambda}, a^\dagger_{\vec{k}',\lambda'}] = \delta_{\vec{k}\vec{k}'}\delta_{\lambda\lambda'}\mathbb{1}. \tag{2.1.5}$$

From this, one can deduct the so-called number operator $N = a^\dagger a$. The eigenstates of this operator are the Fock states with

$$\hat{N}|n\rangle = a^\dagger a|n\rangle = n|n\rangle. \tag{2.1.6}$$

Following up, since the number of photons in a mode is defined by the expectation value of the operator product $a^\dagger a$, the energy of a single mode and polarisation with frequency $\omega_{\vec{k}}$ is given as $E_{\vec{k}} = \hbar\omega_{\vec{k}} a^\dagger_{\vec{k}} a_{\vec{k}}$. Therefore the Hamiltonian describing the energy of a system consisting of multiple light field modes with mode vectors $\vec{k}$ and polarisations $\lambda$ then takes the form:

$$H = \sum_{\vec{k},\lambda} \hbar\omega_{\vec{k},\lambda} a^\dagger_{\vec{k},\lambda} a_{\vec{k},\lambda}. \tag{2.1.7}$$

Here we neglect the energy of the vacuum field mode, which would result in an additional $\hbar\omega_{\vec{k}}/2$ term. This, however, does not change the general dynamics of systems of interest. With the operators for atomic and light field configurations defined, we can now continue to define the interaction Hamiltonian between these two systems. Within the electric dipole approximation, the interaction between an atom and an electromagnetic field is determined by the Hamiltonian [7]:

$$H_{\text{int}} = -\vec{d} \cdot \vec{E}. \tag{2.1.8}$$

With $\vec{d}$ being the dipole moment of the atom and $\vec{E}$ being the quantized electric field. Using the defined transition operators in Eq. (2.1.1), the dipole moment of a two-level atom can be written as

$$\vec{d} = \vec{d}_{21}(\sigma^{21} + \sigma^{12}), \tag{2.1.9}$$

where $\vec{d}_{21}$ is the dipole matrix element and is assumed to be real. The quantized electric field $\vec{E}$ has a positional dependency, and neglecting the polarization of different modes takes the form of [7]:

$$\vec{E}(\vec{r}) = i \sum_{\vec{k}} E_{\vec{k}} (a_{\vec{k}} \vec{f}_{\vec{k}}(\vec{r}) - a^\dagger_{\vec{k}} \vec{f}^*_{\vec{k}}(\vec{r})). \tag{2.1.10}$$

Here, $E_{\vec{k}} = (\hbar\omega_k/2\epsilon_0)^{\frac{1}{2}}$ is the field amplitude regarding the corresponding mode frequency $\omega_{\vec{k}}$ and $\vec{f}_{\vec{k}}$ is the positional depending mode function. Inserting these quantities into Eq. (2.1.8), results in the Hamiltonian

$$H_{\text{int}} = i\hbar \sum_{\vec{k}} g_{\vec{k}} (\sigma^{21} + \sigma^{12})(a_{\vec{k}} f(\vec{r}) - a^\dagger_{\vec{k}} f^*(\vec{r})), \tag{2.1.11}$$

with $g_{\vec{k}} = -E_{\vec{k}} \vec{d}_{21} \vec{e}_{\vec{k}}/\hbar$ the coupling strength, and $\vec{e}_{\vec{k}}$ being the polarisation direction of the electric field, which we extracted from the mode function $\vec{f}_{\vec{k}}$. Using the rotating wave approximation [13] and adding the atomic (2.1.2) and field Hamiltonian (2.1.7) we end up with the Jaynes-Cummings model [14] for a single mode

$$H_{JC} = \hbar\omega_a \sigma^{22} + \hbar\omega_f a^\dagger a + i\hbar g(\sigma^{21} a f(\vec{r}) - \sigma^{12} a^\dagger f^*(\vec{r})). \tag{2.1.12}$$

Here, we denote the frequency of the atoms as $\omega_a$ and the ones for the field as $\omega_f$. A generalisation of this model for $N$ identical atoms leads to the well-known Tavis-Cummings Hamiltonian, given as [15]

$$H_{TC} = \sum_i \hbar\omega_i \sigma_i^{22} + \hbar\omega_f a^\dagger a + i\hbar \sum_i g_i(\sigma_i^{21} a f(\vec{r}_i) - \sigma_i^{12} a^\dagger f^*(\vec{r}_i)). \tag{2.1.13}$$

This Hamiltonian is frequently encountered in quantum optical systems, and we will also use it in the examples later in this work.

## 2.2. Heisenberg Picture

Before deriving the Quantum Langevin Equation, which is used to determine the differential equations of operator averages, we will now give a short introduction into the Heisenberg picture. The time evolution of a system, described by a quantum state $|\Psi(t)\rangle$, is represented by the time-dependent Schrödinger equation. This well-known equation relates the time derivation of a state using a Hamiltonian $H$ and takes the form:

$$i\hbar \frac{\mathrm{d}}{\mathrm{d}t} |\Psi(t)\rangle = H |\Psi(t)\rangle . \tag{2.2.1}$$

Using the definition of the density operator $\rho = \sum_i p_i |\Psi_i\rangle \langle\Psi_i|$, with $p_i$ being the normalised state probabilities, one can also conduct the so-called von Neumann equation. For this, the time derivative of the density operator $\rho$ of a system relates to the Hamiltonian and is

$$\frac{\partial}{\partial t}\rho = \frac{i}{\hbar}[\rho, H]. \tag{2.2.2}$$

In contrast to the Schrödinger picture, where a quantum state is described as being time-dependent, in the Heisenberg picture, the operators inherit the time dependence. A Heisenberg operator, i.e. a system operator $a$ represented in the Heisenberg picture, $a(t)$ is given as

$$a(t) = U(t)^\dagger a U(t), \tag{2.2.3}$$

where $U(t)$ is the time-evolution operator for a system described by the Hamiltonian $H$. If $H$ itself is a time-independent quantity, one can write the time-evolution operator in the general form as

$$U(t) = e^{-i\frac{H}{\hbar}t}. \tag{2.2.4}$$

From the above two equations, one can then derive the so-called Heisenberg equation, which describes the time evolution of a Heisenberg operator, evolving under the time-independent Hamiltonian $H$ as

$$\frac{\mathrm{d}}{\mathrm{d}t} O(t) = \frac{i}{\hbar}[H, O(t)]. \tag{2.2.5}$$

The expectation value of an arbitrary operator regarding a time-dependent quantum state $|\Psi(t)\rangle$ is specified by $\langle O\rangle = \langle\Psi(t)|\, O\, |\Psi(t)\rangle$. Since the state $|\Psi(t)\rangle$ can also be expressed using the time evolution operator given in Eq. (2.2.4) as $|\Psi(t)\rangle = U(t)\,|\Psi(0)\rangle$, it follows that

$$\langle\Psi(t)|\, O\, |\Psi(t)\rangle = \langle\Psi(0)|\, U(t)^\dagger O U(t)\, |\Psi(0)\rangle = \langle\Psi(0)|\, O(t)\, |\Psi(0)\rangle\,. \tag{2.2.6}$$

Moreover, one can show that the time derivative of an operator average in the Schrödinger picture leads to the same as applying the expectation value on Eq. (2.2.5). For a general, non-pure quantum state described by the density matrix $\rho$, the expectation value of an operator is then given as

$$\langle O\rangle(t) \equiv \mathrm{Tr}\{U(t)^\dagger O U(t)\rho\} = \mathrm{Tr}\{O U(t)\rho U(t)^\dagger\}, \tag{2.2.7}$$

where we use the cyclic properties of the trace to obtain the right most relation. Following this, it is also apparent to see, that the time derivative of said operator average results in the relation

$$\frac{\mathrm{d}}{\mathrm{d}t}\langle O\rangle \equiv \frac{\mathrm{d}}{\mathrm{d}t}\mathrm{Tr}\{O(t)\rho\} = \mathrm{Tr}\{O\frac{\mathrm{d}}{\mathrm{d}t}\rho(t)\}. \tag{2.2.8}$$

## 2.3. Quantum Langevin Equation

In this section, we will give a short introduction into the derivation of the Quantum Langevin Equation for a system coupled to a bath consisting of harmonic oscillators similar as described in [7, 8]. Considering a system linked to a bath, the Hamiltonian $H$, specifying the whole super-system, then consists of a system term, describing the dynamics of the system of interest $H_{\mathrm{sys}}$, as well as a term representing the bath $H_{\mathrm{b}}$ and the interaction between them $H_{\mathrm{int}}$. Using this total Hamiltonian and the Heisenberg equation (2.2.5), the time derivative of an operator of the system coupled with the bath reads

$$\frac{\mathrm{d}}{\mathrm{d}t}O(t) = \frac{i}{\hbar}\left([H_{\mathrm{sys}}, O(t)] + [H_{\mathrm{b}}, O(t)] + [H_{\mathrm{int}}, O(t)]\right). \tag{2.3.1}$$

Furthermore, the bath consists of infinitely many harmonic oscillator modes, describing an external field with the Hamiltonian given in Eq. (2.1.7). Continuing, we use a simplified version of the bath system, where we neglect different polarisations. Therefore, we have the interaction Hamiltonian between the system and the bath

$$H_{\mathrm{int}} = i\hbar\sum_{\vec{k}}\left(\kappa^*_{\vec{k}} c^\dagger b_{\vec{k}} - \kappa_{\vec{k}} c b^\dagger_{\vec{k}}\right), \tag{2.3.2}$$

where we require, that $\kappa_{\vec{k}}$ is a weak coupling strength between the system and the mode with wave vector $\vec{k}$. The dissipative coupling operator $c$ depends on the system of interest. Common examples of these coupling operators would include systems like:

- *Two level atom:* $c = \sigma^-$, describing spontaneous emission of the excited state.
- *Single mode cavity:* $c = a$, describing photon losses of the cavity.

With the bath Hamiltonian given in Eq. (2.1.7) and the interaction Hamiltonian in Eq. (2.3.2), the whole Hamiltonian of the system coupled to the bath reads

$$H = H_{\text{sys}} + \sum_{\vec{k}} \hbar\omega_{\vec{k}} b_{\vec{k}}^\dagger b_{\vec{k}} + i\hbar \sum_{\vec{k}} \left(\kappa_{\vec{k}}^* c^\dagger b_{\vec{k}} - \kappa_{\vec{k}} c b_{\vec{k}}^\dagger\right). \tag{2.3.3}$$

Following this, we can use the Heisenberg equation (2.2.5) to calculate the time evolution for our bath $b(t)$ and system operators $A(t)$. As all bath operators commute with the system Hamiltonian, we conclude the time derivative for the bath annihilation operator $b_{\vec{k}}$ to

$$\dot{b}_{\vec{k}} = \frac{i}{\hbar}[H, b_{\vec{k}}] = -i\omega_{\vec{k}} b_{\vec{k}} + \kappa_{\vec{k}}^* c. \tag{2.3.4}$$

By formal integration of Eq. (2.3.4), one can therefore obtain

$$b_{\vec{k}}(t) = b_{\vec{k}}(t=0)e^{-i\omega_{\vec{k}}t} + \kappa_{\vec{k}}^* \int_0^t dt' \left(e^{-i\omega_{\vec{k}}(t-t')} c(t')\right). \tag{2.3.5}$$

Continuing, for a general system operator $A$ we derive

$$\dot{A} = \frac{i}{\hbar}[H, A] = \frac{i}{\hbar}[H_{\text{sys}}, A] + \sum_{\vec{k}} \left(\kappa_{\vec{k}}^* b_{\vec{k}}^\dagger [A, c] - \kappa_{\vec{k}} [A, c^\dagger] b_{\vec{k}}\right). \tag{2.3.6}$$

Inserting Eq. (2.3.5) into Eq. (2.3.6), we arrive at an equation for $A(t)$ in the form

$$\begin{aligned}\dot{A} = \frac{i}{\hbar}[H_{\text{sys}}, A] + \int_0^t dt' \Big(\gamma^*(t-t') c^\dagger(t')[A, c] - \gamma_n(t-t')[A, c^\dagger] c(t')\Big) \\ + \eta^\dagger(t)[A, c] - [A, c^\dagger]\eta(t).\end{aligned} \tag{2.3.7}$$

In this step we introduce the time-dependant functions $\gamma(t)$ and $\eta(t)$. These are defined as

$$\gamma(t-t') = \sum_{\vec{k}} e^{-i\omega_{\vec{k}}(t-t')} |\kappa_{\vec{k}}|^2 \tag{2.3.8}$$

$$\eta(t) = \sum_{\vec{k}} e^{-i\omega_{\vec{k}}t} \kappa_{\vec{k}} b_{\vec{k}}(t=0) \tag{2.3.9}$$

We assume that the coupling $\kappa_{\vec{k}}$ is constant for all $\vec{k}$, i.e. $\kappa_{\vec{k}} = \kappa$. For this assumption we can simplify $\gamma(t-t')$ to

$$\gamma(t-t') \approx |\kappa|^2 \sum_{\vec{k}} e^{-i\omega_{\vec{k}}(t-t')}. \tag{2.3.10}$$

Furthermore, we use the Markov approximation and use the fact that $\gamma$ is a function that peaks very sharply around $t' = t$ and is close to zero otherwise. We exchange for this step $\tau = t - t'$, and since $\gamma$ is close to zero for $\tau > t$, we can rewrite the upper limit as infinite. With this rewriting, we arrive at the relation:

$$\int_0^\infty d\tau \gamma(\tau) c(\tau) \to c(0) \int_0^\infty d\tau \gamma(\tau) e^{i\omega_0 \tau} \tag{2.3.11}$$

For the integration of this function over $t'$, we use the Cauchy principal value and rewrite the summation over $\vec{k}$ into an integral over $\omega_{\vec{k}}$ by considering the density of states $g(\omega_{\vec{k}})$. By doing so, we acquire

$$\int_0^\infty d\tau \gamma(\tau) = \lim_{\epsilon \to 0} \int_0^\infty d\tau \int_0^\infty d\omega_{\vec{k}} g(\omega_{\vec{k}}) |\kappa(\omega_{\vec{k}})|^2 e^{-i(\omega_{\vec{k}} - \omega_0 - i\epsilon)\tau} \tag{2.3.12}$$

$$\int_0^\infty d\tau \gamma(\tau) = \lim_{\epsilon \to 0} \int_0^\infty d\omega_{\vec{k}} \frac{g(\omega_{\vec{k}}) |\kappa(\omega_{\vec{k}})|^2}{i(\omega_{\vec{k}} - \omega_0 - i\epsilon)}. \tag{2.3.13}$$

And for the integral over $\omega_{\vec{k}}$ we calculate

$$\int_0^\infty d\tau \gamma(\tau) = \lim_{\epsilon \to 0} \int_0^\infty d\omega_{\vec{k}} g(\omega_{\vec{k}}) |\kappa(\omega_{\vec{k}})|^2 \left[ \frac{\epsilon}{(\omega_{\vec{k}} - \omega_0)^2 + \epsilon^2} - \frac{i(\omega_{\vec{k}} - \omega_0)}{(\omega_{\vec{k}} - \omega_0)^2 + \epsilon^2} \right] \tag{2.3.14}$$

For the first part of this integral we then obtain

$$\lim_{\epsilon \to 0} \int_0^\infty d\omega_{\vec{k}} g(\omega_{\vec{k}}) |\kappa(\omega_{\vec{k}})|^2 \frac{\epsilon}{(\omega_{\vec{k}} - \omega_0)^2 + \epsilon^2} = \pi g(\omega_0) |\kappa(\omega_0)|^2, \tag{2.3.15}$$

and for the second part

$$\lim_{\epsilon \to 0} \int_0^\infty d\omega_{\vec{k}} g(\omega_{\vec{k}}) |\kappa(\omega_{\vec{k}})|^2 \frac{i(\omega_{\vec{k}} - \omega_0)}{(\omega_{\vec{k}} - \omega_0)^2 + \epsilon^2} = i \frac{\mathrm{P}}{\omega_{\vec{k}} - \omega_0}. \tag{2.3.16}$$

Here, P denotes the principal value integral. Finally, we end up with the relation for $\gamma(\tau)$ with

$$\int_0^\infty d\tau \gamma(\tau) = \frac{1}{2}\Gamma + i\delta_l, \tag{2.3.17}$$

where $\Gamma = 2\pi g(\omega_0)|\kappa|^2$ is a damping rate and $\delta_l$ an induced line-shift, which one can eliminate using a renormalization of the transition frequency $\omega_0$, i.e. $\omega_0 = \omega_0' + \delta_l$, where $\omega_0'$ is the initial transition frequency. With the use of Eq. (2.3.17) we can conclude that

$$\int_0^t dt' \gamma(t-t') c(t') \to c(t) \int_0^t dt' \gamma(t-t') e^{i\omega_0(t-t')} = \left(\frac{1}{2}\Gamma + i\delta_l\right) c(t). \tag{2.3.18}$$

Inserting this relation into Eq. (2.3.7) leads us to the Quantum Langevin Equation for a system operator $A$ in the form of

$$\dot{A} = \frac{i}{\hbar}[H_{\text{sys}}, A] - \left[[A, c^\dagger]\left(\frac{1}{2}\Gamma c + \eta(t)\right) - \left(\frac{1}{2}\Gamma c^\dagger + \eta^\dagger(t)\right)[A, c]\right]. \tag{2.3.19}$$

The function $\eta(t)$ depends purely on the bath and its initial state. It can be related to an input field from the bath via $\eta(t) = \sqrt{\Gamma} b_{\text{in}}(t)$[8]. In the following, we assume that the bath is initially in a vacuum state and any noise coming from the bath operator $b_{\text{in}}(t)$ is white noise. Following this, $b_{\text{in}}(t)$ vanishes after averaging any operator equations. Therefore, we neglect $\eta(t)$ in Eq. (2.3.19) in the upcoming steps since we are only interested in time derivatives of operator averages. Calculating the expectation value of Eq. (2.3.19) and also allowing multiple dissipative processes leads us now to the final equation of the expectation value of the time derivative of a system operator

$$\langle \dot{A} \rangle = \frac{i}{\hbar}\langle [H_{\text{sys}}, A] \rangle + \frac{1}{2}\sum_n \Gamma_n \left( \langle c_n^\dagger [A, c_n] \rangle + \langle [c_n^\dagger, A] c_n \rangle \right). \tag{2.3.20}$$

When the dissipative operators $c_n$ with their rates $\Gamma_n$ are known, we can calculate the dynamics of all system operators we need. However, these equations of motion are not necessarily complete since time derivatives of operator averages usually depend on other operator averages with higher-order products. To numerically solve them, the dynamics of all the encountered expectation values need to be known. To solve this problem, we can approximate higher-order averages by already known lower-order ones. A method that achieves this truncation is the cumulant expansion, where one neglects higher-order quantum correlations to represent higher-order operator product averages as lower-order ones.

## 2.4. The Cumulant expansion method

When deriving differential equations for operator averages using the Quantum Langevin Equation, one commonly encounters several operator averages containing higher order products than the differentiated one. Therefore, to end up with an equation set containing a differential equation for every average, one has to calculate the Quantum Langevin Equation many, if not an infinite amount, of times. To this end, approximations like the cumulant expansion method [16] are used to truncate the equation set therefore completing the set in a reasonable amount of iterations, for the cost of neglecting higher order quantum correlations. This method was first introduced in [17] for general random

variables $X$, in which the cumulant average is defined as

$$\langle X_1 X_2 ... X_n \rangle_c := \sum_{p \in P(\mathcal{I})} (|p|-1)!(-1)^{|p|-1} \prod_{B \in p} \langle \prod_{i \in B} X_i \rangle. \tag{2.4.1}$$

In the above equation, $n$ is the so-called order of the cumulant average, and $P(\mathcal{I})$ is a set of all possible partitions of the index-set $\mathcal{I} = \{1, 2, ..., n\}$. One such partition is denoted by $p$, with its length $|p|$, and $B$ is one block of a given partition, i.e. a set of indices of $p$. The explicit form of a cumulant average expansion is for up to order three random variables $X_i$:

$$\langle X \rangle_c = \langle X \rangle \tag{2.4.2}$$

$$\langle X_1 X_2 \rangle_c = \langle X_1 X_2 \rangle - \langle X_1 \rangle \langle X_2 \rangle \tag{2.4.3}$$

$$\begin{aligned} \langle X_1 X_2 X_3 \rangle_c = \langle X_1 X_2 X_3 \rangle - \langle X_1 X_2 \rangle \langle X_3 \rangle - \langle X_1 X_3 \rangle \langle X_2 \rangle \\ - \langle X_1 \rangle \langle X_2 X_3 \rangle + 2 \langle X_1 \rangle \langle X_2 \rangle \langle X_3 \rangle. \end{aligned} \tag{2.4.4}$$

Theorem 1 in Ref. [17] states that a cumulant average is zero when the elements are dividable in at least two statistically independent groups, i.e. $\langle X_1 X_2 ... X_n \rangle_c = 0$ if at least one of the elements in $\{X_1, X_2, ..., X_n\}$ is statistically independent of the others. We can use this Theorem to create an approximation. Let us assume that at least one of the variables is statistically independent of the others so that the cumulant average is zero. Therefore, we can set the left-hand side of Eq. (2.4.1) to zero and rearrange it to end up with

$$\langle X_1 X_2 ... X_n \rangle = \sum_{p \in P(\mathcal{I}) \setminus \mathcal{I}} (|p|-1)!(-1)^{|p|} \prod_{B \in p} \langle \prod_{i \in B} X_i \rangle. \tag{2.4.5}$$

Here, the summation runs over all possible partitions except the one that is equal to the index-set $\mathcal{I}$ itself, implying that no average on the right-hand side of Eq. (2.4.5) has the same or higher order than the left-hand side. With this, one can approximate any average as summations of products of lower-order averages.

This assumption allows us to introduce an upper limit to the number of operators in averages encountered throughout the derivation of equations of motion. This upper bound limit will be denoted as the "order of expansion" and is equivalent to the maximum number of operators allowed in any average. As for the example referenced above, one can now calculate an average of orders two and three to:

$$\langle X_1 X_2 \rangle \to \langle X_1 \rangle \langle X_2 \rangle \tag{2.4.6}$$

$$\langle X_1 X_2 X_3 \rangle \to \langle X_1 X_2 \rangle \langle X_3 \rangle + \langle X_1 X_3 \rangle \langle X_2 \rangle + \langle X_1 \rangle \langle X_2 X_3 \rangle - 2 \langle X_1 \rangle \langle X_2 \rangle \langle X_3 \rangle. \tag{2.4.7}$$

Such an approximation for the order one cumulant expansion (2.4.6) is often referenced as the mean-field approximation. The relationship described in Eq. (2.4.1) can also be used recursively, meaning that one can for example represent an average of order four

in terms of order two and lower, by first performing a third order expansion and then a second order one. Substituting operator averages as shown in (2.4.6) and in (2.4.7) is, in general, an approximation. However, if the joint cumulant of the expanding average is indeed zero, this substitution is exact.

## 2.5. Dipole-Dipole Interactions

In this section, we build upon the previous derivation of the Quantum Langevin Equation to include systems with close-range dipole-dipole interactions. For simplicity, we will consider an ensemble of identical two-level atoms, with a ground state $|g\rangle$ and excited state $|e\rangle$, interacting with a single external field bath. A simplified depicture of such a system is shown in Fig. 2.2. In contrast to the previous derivation, where atoms coupled individually to the bath, this time, we also describe the interaction between the systems via the bath. We will take a similar approach as provided in [18] and rely on the derivation given in [7, 8].

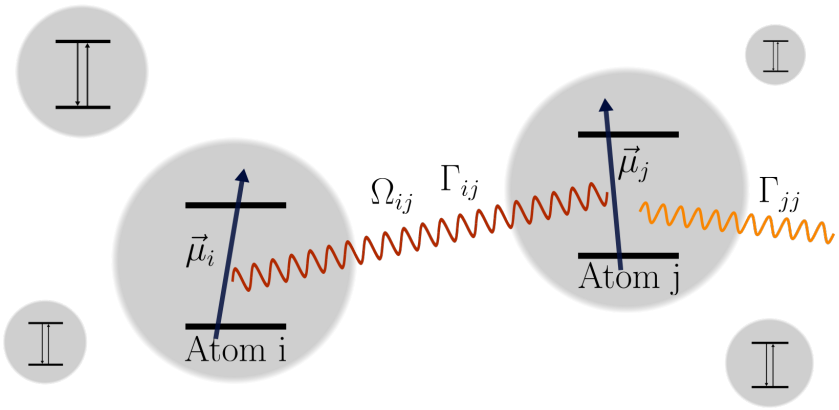


**Figure 2.2.:** *Illustration of the coherent and dissipative coupling of an atomic ensemble.* The coherent coupling $\Omega_{i,j}$ and dissipative coupling $\Gamma_{i,j}$ are highlighted by the red curve. Individual decay with the natural decay rate $\Gamma_0$ into the bath is shown by the orange curve. Atom $i$ and $j$ are also depicted with their dipole moment $\vec{\mu} = \vec{d}_{eg}$.

The bath Hamiltonian is similar to the previous one in Eq. (2.1.7), whereas the interaction Hamiltonian now reads [8]:

$$H_{\text{int}} = i\hbar \sum_{\vec{k},i} \left( \kappa^*_{\vec{k}} b^\dagger_{\vec{k}} e^{-i\vec{k}\vec{r}_i} \sigma_i^- - \kappa_{\vec{k}} b_{\vec{k}} e^{i\vec{k}\vec{r}_i} \sigma_i^+ \right), \tag{2.5.1}$$

where $\sigma_i^- = |g_i\rangle \langle e_i|$, $\sigma_i^+ = |e_i\rangle \langle g_i|$ and $\vec{r}_i$ is the positional vector of the $i$-th atom in the ensemble. Following the previous procedure, we arrive at the Heisenberg equation for a system operator $A(t)$ as

$$\dot{A} = \frac{i}{\hbar}[H, A] + \sum_{i,j}\Bigg[\int_0^t dt' \left(\gamma^*_{i,j}(t-t')\sigma_j^+(t')[A,\sigma_i^-] - \gamma_{i,j}(t-t')[A,\sigma_i^+]\sigma_j^-(t')\right) \\ \delta_{i,j}\eta^\dagger(t)[A,\sigma_i^-] - \delta_{i,j}[A,\sigma_i^+]\eta(t)\Bigg], \tag{2.5.2}$$

where this time

$$\gamma_{i,j}(t-t') = \sum_{\vec{k}} e^{-i\omega_{\vec{k}}(t-t')+i\vec{k}(\vec{r}_i-\vec{r}_j)}|\kappa_{\vec{k}}|^2 \tag{2.5.3}$$

$$\eta(t) = \sum_{\vec{k}} e^{-i\omega_{\vec{k}}t+i\vec{k}\vec{r}_j}\kappa_{\vec{k}}b_{\vec{k}}(t=0). \tag{2.5.4}$$

Applying the same approximations, which we used in Sec. 2.3, we obtain

$$\int_0^\infty d\tau\gamma_{i,j}(\tau)F(\tau) \to F(0)\int_0^\infty d\tau\gamma_{i,j}(\tau)e^{i\omega_0\tau} = \left(\frac{1}{2}\Gamma_{i,j} + i\Omega_{i,j}\right)F(0), \tag{2.5.5}$$

and with that, we conclude that

$$\dot{A} = \frac{i}{\hbar}[H + H_{\text{dip}}, A] - \sum_{i,j}\Bigg[[A,\sigma_i^+]\left(\frac{1}{2}\Gamma_{i,j}\sigma_j^- + \delta_{i,j}\eta(t)\right) \\ - \left(\frac{1}{2}\Gamma_{i,j}\sigma_j^+ + \delta_{i,j}\eta^\dagger(t)\right)[A,\sigma_i^-]\Bigg]. \tag{2.5.6}$$

In Eq. (2.5.6) we combined terms proportional to $\Omega_{i,j}$ into the Hamiltonian $H_{\text{dip}}$ for $i \neq j$ and absorbed $\Omega_{i,i}$ into $\omega$. Therefore, the dipole-dipole interaction Hamiltonian $H_{\text{dip}}$ is given as

$$H_{\text{dip}} = \hbar\sum_{i\neq j}\Omega_{i,j}\sigma_i^+\sigma_j^-. \tag{2.5.7}$$

Using again, that $\eta(t)$ depends only on the initial bath state and that any noise generated by the bath is white noise, we end up with the time derivative of an operator average of our system as

$$\langle\dot{A}\rangle = \frac{i}{\hbar}\langle[H_{\text{sys}} + H_{\text{dip}}, A]\rangle + \frac{1}{2}\sum_{i,j}\Gamma_{i,j}\left(\langle\sigma_i^+[A,\sigma_j^-]\rangle + \langle[\sigma_i^+,A]\sigma_j^-\rangle\right). \tag{2.5.8}$$

The quantities calculated by the integral in Eq. (2.5.5) are the so called coherent coupling $\Omega_{i,j}$ and the dissipative coupling $\Gamma_{i,j}$. They are given as [18]

$$\Omega_{i,j} = \Gamma_0 G(k_0 r_{i,j}) \tag{2.5.9}$$
$$\Gamma_{i,j} = \Gamma_0 F(k_0 r_{i,j}), \tag{2.5.10}$$

where $\Gamma_0 = \Gamma_{i,i} = 2\pi g(\omega)|\kappa|^2$ is the natural decay rate, $k_0 = \omega/c$ the associated wave vector of the decay channel and $r_{i,j} = |\vec{r}_i - \vec{r}_j|$ is the distance between atom $i$ and atom $j$. Furthermore the functions $G$ and $F$ are defined as

$$G(\xi) = -\frac{3}{4}\Big[(1-\cos^2\theta)\frac{\cos\xi}{\xi} - (1-3\cos^2\theta)(\frac{\sin\xi}{\xi^2} + \frac{\cos\xi}{\xi^3})\Big] \tag{2.5.11}$$
$$F(\xi) = -\frac{3}{2}\Big[(1-\cos^2\theta)\frac{\sin\xi}{\xi} - (1-3\cos^2\theta)(\frac{\cos\xi}{\xi^2} + \frac{\sin\xi}{\xi^3})\Big], \tag{2.5.12}$$

where $\theta$ is the angle between the vector $\vec{r}_{i,j}$ and the dipole moment of the emitter $\vec{d}_{eg}$. From these functions, one can also determine the behaviour in the limiting cases $r_{i,j} \gg \lambda$ and $r_{i,j} \ll \lambda$, where $\lambda$ is the wavelength of the emitted light. If the atoms are largely separated, i.e. $r_{i,j} \to \infty$, they behave like completely separated emitters and the decay rates $\Gamma_{i,j}$ approach the one of a single emitter meaning that $\Gamma_{i,j} \xrightarrow{r_{i,j}\to\infty} \Gamma_0\delta_{ij}$, as one would intuitively expect. On the other hand, if the atoms are very close to each other, all decay rates approach the same value of the single-emitter emission rate. In this case, one can conduct that $\Gamma_{i,j} \xrightarrow{r_{i,j}\to 0} \Gamma_0$ for all decay rates.

# Chapter 3.

# QuantumCumulants.jl

QuantumCumulants.jl is a framework written in the Julia programming language and designed for the automatic symbolic derivation of equations of motion for quantum operator averages in the Heisenberg picture. Time derivatives of expectation values of open system operators are calculated using the Heisenberg equations for operators including damping via a QLE-based approach given in Eq. (2.3.20). For any nontrivial Hamiltonian the commutator introduces new and often more complex operator products and thus this approach commonly leads to a large, or even infinite number, of differential equations needed to fully describe the system of interest. To this end, QuantumCumulants.jl uses a truncation of the fast-growing set of expectation values of the differential equations using the cumulant expansion method described in Sec. 2.4 [16].

The framework enables a user to do quick simulations of quantum optical systems. For this purpose, it consists of multiple methods and constructors designed to automate the calculation of commutator relations of bosonic and transition operators. In difference to other frameworks, QuantumCumulants.jl uses solely symbolic operators. No in-detail knowledge of the inner structure of the operator is therefore required. Calculating commutator relations is done by using predefined rules of calculus for different operators. Other functionalities like completing a set of equations to a closed one or calculation of Correlation-Functions are integrated. Outputs of these methods are composed in a specific way so that the differential equations solver implemented in Julia, i.e. the package DifferentialEquations.jl [19], can interpret them. Further functionalities, such as calculating the spectrum of a cavity output using the two-time correlation-function or defining an initial state using QuantumOptics.jl, are also possible.

## 3.1. The Julia Language

Julia is a programming language created by Jeff Bezanson et al. [20] with the goal to achieve both the speed of efficient compiled languages such as C and the usability of dynamic languages like Python. One of the benefits of the Julia programming language is the intuitive way of programming, which other dynamic languages commonly achieve. Despite being one of these high-dynamic programming languages, Julia is not suffering from performance issues. It achieves performances similar to compiled languages, such

as C or Rust. The reasons for this is its just-in-time (JIT) compilations [21]. As of this, the language uses dynamic, multiple dispatches for different code selections, meaning that Julia chooses methods for encountered types at run time. In return, this requires a strict declaration of types. To this end, a dynamic type system is implemented, meaning the type of a value is not known during compilation times. However, an explicitly declared hierarchy of types is still present. As an example, every value is an `Any` type, which is the universal type in Julia, but not every `Number` type is a `Float`. With such type declarations, one can create methods that are distinguished by the types of their inputs, meaning that one can write two functions with the same name but different input types.

Another advantageous aspect of the Julia programming language is that it is an open-source project. Therefore, it also has a large number of contributors and many available packages. Installing and loading packages is simple and doable with only a few commands. Most of them are designed, so one can easily apply their core functionalities and even build upon them. Multiple of them are also easily combinable. As an example, QuantumCumulants.jl banks on functionalities of the package Symbolics.jl [22]. Detailed information about the programming language can be found in its extensive documentary [23].

## 3.2. Usage

In this section, we want to give a quick insight into the fundamental workflow when using QuantumCumulants.jl. To this end we show an example, where we derive the equations of motion for a small system containing two atoms. We look at a setup similar to the one described in the later cavity anti-resonance example [24]. In particular, we have an extended Jaynes-Cummings model, analogous to the one in Eq. (2.1.12), describing the interaction between two atoms and a lossy cavity. We include in this system the dipole-dipole interaction between the two atoms and a laser drive with the corresponding rate $\eta$. In the rotating frame of the driving laser the Hamiltonian of the system is given by

$$H = \Delta_c a^\dagger a + \eta(a^\dagger + a) + \Delta_a \sum_i^N \sigma_i^{22} + \sum_{i\neq j}^N \sum_{j\neq i}^N \Omega_{ij} \sigma_i^{21} \sigma_j^{12} + \sum_i^N g_i (a^\dagger \sigma_i^{12} + a\sigma_i^{21}), \quad (3.2.1)$$

with $\Delta_c$ and $\Delta_a$ being the detuning of the cavity and atoms, respectively, to the laser frequency. We also include in this Hamiltonian dipole-dipole interaction similar to the one seen in Eq. (2.5.1) with $\Omega_{ij}$ being the coherent coupling.

To derive the equations of motion, we first need to import the package functionalities into our program. We do so by writing `using QuantumCumulants.jl`, as seen in code sample 1. Next, we define the parameters $g_i$, $\Gamma_{ij}$ and $\Omega_{ij}$, as well as $\Delta_c$, $\Delta_a$, $\eta$ and $\kappa$.

For this step, we use both the macro `@cnumbers` to create single parameters and also the function call `cnumbers()` to generate functions, which depend on the numeric indices $i$ and $j$, representing the other parameters needed.

```
using QuantumCumulants
order = 2
@cnumbers Δc η Δa κ

g(j) = cnumbers(Symbol(g, j))[1]
Γ(i,j) = cnumbers(Symbol(Γ, i, j))[1]
Ω(i,j) = cnumbers(Symbol(Ω, i, j))[1]

N = 2 #number of atoms
hc = FockSpace(:cavity)
ha = ⊗([NLevelSpace(Symbol(:atom,i),2) for i=1:N]...)
h = hc ⊗ ha

@qnumbers a::Destroy(h)
σ(i,j,k) = Transition(h,Symbol("σ_{$k}"),i,j,k+1)
```

Code sample 1: *Including the package and defining Hilbert spaces, parameters and operators.*

Additionally, we define in code sample 1 a `FockSpace` and also a combination of several `NLevelSpace` entities. These definitions are used to distinguish the rules of calculus for operators acting on different Hilbert spaces. The whole Hilbert space of the system `h` is then defined as a tensor product of the former two. As a last step, we introduce the operators `a` and $\sigma$ as callable functions, which act on the previously defined `FockSpace` and $k$-th `NLevelSpace`, respectively. At this point, we can construct the Hamiltonian from Eq. (5.3.1) describing the system of interest using the previously defined operators, as seen in code sample 2.

```
# Hamiltonian
Hc = Δc*a'a + η*(a' + a)
Ha = Δa*sum(σ(2,2,k) for k=1:N) +
    sum(Ω(i,j)*σ(2,1,i)*σ(1,2,j)*(i≠j) for i=1:N for j=1:N)
Hi = sum(g(k)*(a'σ(1,2,k) + a*σ(2,1,k)) for k=1:N)
H = Hc + Ha + Hi

# Jump operators & and rates
J = [a, [σ(1,2,k) for k=1:N]]
rates = [κ, [Γ(i,j) for i=1:N, j=1:N]]

ops = [a, σ(2,2,1), σ(1,2,1)]
eqs = meanfield(ops,H,J;rates=rates,order=order)
complete!(eqs)
```

Code sample 2: *Constructing the Hamiltonian, which describes the system, and deriving the associated equations of motion.*

Furthermore, we can also specify the dissipative processes described by their jump operators and corresponding rates with the vectors `J` and `rates`, respectively. After

defining everything we need, we can create an initial set of equations for the operator averages defined in the vector `ops` by calling the function `meanfield`. The output of this function is a set of equations of motion for the averages of the operators given in `ops`, calculated by the Quantum Langevin Equation (2.5.8). To this end, the function uses the coupling operators from the vector `J` with their corresponding dissipative rates in the argument `rates`. Averages in the equations with a larger order than the one specified in the keyword `order` are approximated using the cumulant expansion, as described in Sec. 2.4. As a final step, we complete the equations using the in-place method `complete!`. An in-place function modifies its input to correspond to the output, meaning that after the call `complete!`, `eqs` is a complete set of equations. This completion of the equations calculates the QLE for every unknown average encountered within the equations, which is not a conjugate of already-known ones. This process is repeated until there are no unknown expectation values left. As a result, we obtain a set of equations which is complete and has in this particular case the following form:

$$\begin{aligned}
\frac{d}{dt}\langle a\rangle &= -1i\eta - 1i\Delta_c\langle a\rangle - 0.5\kappa\langle a\rangle - 1ig_1\langle {\sigma_1}^{12}\rangle - 1ig_2\langle {\sigma_2}^{12}\rangle \\
\frac{d}{dt}\langle {\sigma_1}^{12}\rangle &= \Gamma_{12}\langle {\sigma_1}^{22}{\sigma_2}^{12}\rangle - 0.5\Gamma_{11}\langle {\sigma_1}^{12}\rangle - 1ig_1\langle a\rangle - 0.5\Gamma_{12}\langle {\sigma_2}^{12}\rangle + ... \\
\frac{d}{dt}\langle {\sigma_2}^{12}\rangle &= \Gamma_{21}\langle {\sigma_1}^{12}{\sigma_2}^{22}\rangle - 1ig_2\langle a\rangle + 2ig_2\langle a{\sigma_2}^{22}\rangle - 0.5\Gamma_{21}\langle {\sigma_1}^{12}\rangle + ... \\
\frac{d}{dt}\langle {\sigma_1}^{21}{\sigma_2}^{12}\rangle &= \Gamma_{21}\left(-0.5\langle {\sigma_2}^{22}\rangle - 0.5\langle {\sigma_1}^{22}\rangle\right) + 1ig_1\langle a^\dagger {\sigma_2}^{12}\rangle + ... \\
&\vdots
\end{aligned}$$

At this point, it is clear that for a higher number of atoms in the cavity, the number of equations needed to describe the systems grows quickly. Not only does the number of equations grow fast, but also the number of calculations since terms in the Hamiltonian grow. However, there are certain symmetries in the equations for different averages, which we can use to simplify both, calculus and equations. One can express these symmetries using a recent extension to QuantumCumulants.jl, which introduces symbolic indices and summations. This extension allows one to efficiently calculate systems which consist of similar subsystems by reducing the number of redundant calculations and equations. To showcase the usage of these symbolic indices and summations and also their applications is the core of this thesis.

# Chapter 4.

# The Extension

In this chapter, we give an overview of the extension to symbolic sums developed in this thesis work. We construct this overview by showing how one can instantiate the newly implemented objects and how they interact. Furthermore, we will introduce the most important features for reliable usage.

## 4.1. Goals

The main goal of this new utility package is to reduce redundant calculus by collecting similar differential equations as equations of averages specified with symbolic indices. Solving this problem also requires an implementation of a symbolic summation, with which a user can reduce the individual additions of many different terms to a simple symbolic object. To illustrate this, one can look at the following example: Imagine a user wants to calculate the sum over many operators $a_i$, where $i = \{1, 2, ..., N\}$. Such a summation would result in the series of addition operations $a_1 + a_2 + ...a_N$, whereas one can represent the same addition in just one single symbolic object, $\sum_{i=1}^{N} a_i$. Such summations, including their respective operators, underlie specific calculation rules.

It is clear that an immense series of additions quickly lead to lots of memory allocations and a high number of needed computations. In contrast, when we represent the additions as a symbolic summation, only a few operations need to be calculated for the same effective outcome. Similarly, when deriving the equations of motion for similar subsystems, many have some symmetry, which one can use to simplify and truncate them even further. Let us consider the case where we want to derive the equations of motions for operator averages for a system containing $N$ atoms coupled to a single-mode electromagnetic field. In such a system, we would need to derive equations for operators $\sigma_1^{xy}$ up to $\sigma_N^{xy}$. These operator equations all conclude in the same equation up to their numeric index. Therefore, one needs to store at least $N$ equations when deriving a complete set of equations. We can, however, as an alternative, write equations for an indexed operator $\sigma_i^{xy}$, where $i$ ranges from 1 to $N$. This indexing immensely reduces the size of our equation set. It also allows us to derive a complete set of equations in a time frame, which is only proportional to the order of the cumulant expansion. Because of that, the time frame does not depend on the system size. However, these equations

contain symbolic indices and summations and are not interpretable by differential equations solvers, since they consist of abstract symbolic index objects. However, we can still transform said equations into a solvable form. This transformation is possible in a time and resource-efficient way, as we only need to express the symbolic objects with actual numbers and additions. We will explain this and its increases in computation speed in the later examples and benchmarks.

## 4.2. Indices and Summations

Let us start by examining the implementation of a so-called `Index` entity. An index is an object consisting of the full system `HilberSpace`, a name, a range, and the specific Hilbert space or a number defining the Hilbert space the index acts on. The latter is needed to ensure proper calculation relations between operators with different indices. One can use these index objects combined with operator entities defined in the base QuantumCumulants.jl package to construct a `IndexedOperator` object as shown in code sample 3.

```
@cnumbers N

# Hilbert space
hc = FockSpace(:cavity)
ha = NLevelSpace(:atom,2)

h = hc ⊗ ha

i = Index(h,:i,N,2) # index acts on second subspace of h
j = Index(h,:j,N,ha) # equivalent definition as Index(h,:j,N,2)

# define σ as a callable function for creating operators
σ(x,y,z) = IndexedOperator(Transition(h,:σ,x,y),z)

σ(2,1,i)*σ(1,2,j) # σᵢ²¹σⱼ¹²
```

Code sample 3: *Example for defining indices and operators.*

In code sample 3, we define $N$, the number of two-level atoms, as a c-number, meaning it is a general complex number with no numerical value yet. We create this instance of $N$ using the macro `@cnumbers`. It is still possible to define the upper limit of any index with this parameter, even though it has no numeric value. Constructing an index with a number instead of this parameter is also possible. It is important to note that two indices are equal only if every single attribute of them is equivalent. So if two indices have the same Hilbert spaces and range, but distinct names, they are considered different. The same is true for `IndexedOperator` Instances. These rules are fundamental features of the package. Most of the calculation rules implemented rely on equality checks of `Index` objects. Therefore, careful naming and creations of indices are essential

for correct usage. To complete code sample 3, the last line of code in the above example naturally gives the symbolic output $\sigma_i^{21}\sigma_j^{12}$.

When two operators are multiplied together outside of any symbolic summations, they are first analyzed, whether or not their indices match. If both operators have the same index, a simplification is applied if possible. For an example $\sigma_i^{21}\sigma_i^{12}$ would result in the single operator $\sigma_i^{22}$. When the operators have different indices, they are assumed to commute even though they might act on the same specific Hilbert space. This rule is applied for operators if they are not within a summation. However, different rules are applied if they occur inside a symbolic summation with a running index equal to one of the operators. For example, if an operator, which has an index with the same specific Hilbert space as the summation running index, is multiplied by the sum, an immediate simplification is applied. What simplification occurs depends on the operators inside the summation and the one which is multiplied by the sum. Different types of such simplification processes are shown in Fig. 4.1.

```
∑(σ(2,1,i),i)
```

a) $\sum_{i}^{N}\sigma_i^{21}$

```
∑(σ(2,1,i)*σ(1,2,j),i)
```

b) $\sum_{i\neq j}^{N}\sigma_i^{21}\sigma_j^{12}+\sigma_j^{22}$

```
∑(σ(2,1,i)*σ(1,2,j),i,[j])
```

c) $\sum_{i\neq j}^{N}\sigma_i^{21}\sigma_j^{12}$

```
∑(σ(2,1,i)*σ(1,2,j),i,j)
```

d) $\sum_{j\neq i i\neq j}^{N}\sum^{N}\sigma_i^{21}\sigma_j^{12}+\sum_{j}^{N}\sigma_j^{22}$

**Figure 4.1.:** *Possible different constructions of summations with their corresponding outputs.*

As shown in Fig. 4.1, a simplification occurs in different cases. In example a) of the above figure, we define a regular summation with only one indexed operator entity. This example needs no simplification since there are no multiplications of operators involved. As in example b), we create the additional operator $\sigma_j^{22}$. This operator is calculated from the product $\sigma_i^{21}\sigma_j^{12}$ where $i = j$. This simplification rule is applied because the index $j$ acts on the same Hilbert space as the running index $i$ of the summation. If $j$ acts on a different Hilbert space, the operator gets multiplied into the summation term. Such a special case is calculated by swapping the index of all operators inside the summation that have the running index of the sum with the index of the additionally multiplied operator. After that swapping, a possible simplification is applied to the resulting term. This rule also implies that the upper left example multiplied with an operator $\sigma_j^{12}$ leads to the same result as the upper right code sample. In instantiation c), the index $j$ is excluded by the summation definition, as it is the third argument

of the function call. As a last example, in d), a summation with two running indices, $i$ and $j$, is defined. Again this leads to an additional term, where $i = j$ is still inside the outermost summation. However, the outer summation with running index $j$ is still applied to the additional term, leading to a new creation of a summation. Even though we only use transition operators in these examples, it is also possible to create symbolic summations using indexed bosonic operators. For these, simplifications are applied according to their commutator relations given in Eq. (2.1.5).

It is also possible to instantiate so-called `NumberedOperator` directly by defining `IndexedOperators` with symbolic numbers instead of symbolic indices. These operators are similar to their indexed counterparts regarding their computation rules and commutator relations. What is different, however, is that the numbered variant acts strictly on the Hilbert space identified by its number. These operators are created in the process of transforming a set of indexed average operator equations into an interpretable form for differential equation solver.

## 4.3. Variables

One can also combine variables with indices in the same way as for operators. The command `IndexedVariable` allows a user to create such variables. When doing so, they act as a symbolic placeholder for numerical values. For example, $g_i$ is a placeholder for a variable $g$ with a value for the $i$-th subsystem or operator. What is special about them is that they also underlie exchanges of indices when they are within summations. As for an example, the multiplication $\sigma_j^{22}\left(\sum_i^N g_i\sigma_i^{22}\right)$ results in $\sum_{i\neq j}^N g_i\sigma_i^{22}\sigma_j^{22} + g_j\sigma_j^{22}$. The same is true for Variables with two indices. Fig. 4.2 shows examples of variable definitions.

```
IndexedVariable(:g,i)
```

$g_i$

```
IndexedVariable(:Γ,i,j)
```

$\Gamma_{i,j}$

**Figure 4.2.:** *Different constructions of variables with indices.* On the left definition, a single indexed variable is created, while on the right side, we have a symbolic variable $\Gamma$ with two indices.

When creating variables with two indices, one can use the keyword argument `identical` to simplify multiplications even further. This keyword specifies if the two indices of the variable can have the same symbolic value. If it is set to `true`, instances with the same value for the first and second index are created. If the keyword is `false`, such instances are immediately replaced by zero.

As mentioned, these variables created with the function call `IndexedVariable` do not have any numerical value but can be easily associated with values. One can do this using the `value_map` function, which creates a dictionary of variables given, mapping them to their corresponding value using a given matrix in the second argument. We will show this mapping functionality in several examples where we solve systems containing indexed variables. Variables of any kind get created as Symbols using the Symbolics.jl [22] package. Symbols created by this package act as regular c-numbers. Following this, symbolic variables also undergo simplifications when encountered. For example $g_i + g_i$ results in $2g_i$ and $g_i - g_i$ results in 0. Any Symbol created with this package underlies such simplification rules.

## 4.4. Averages and Equations

Deriving the equations of motions for operator averages is the most crucial feature of the framework. Doing so should be a simple and user-friendly task, which should not require too many input steps. Therefore, we give in this section a quick overview of the few commands and functions needed to derive the equations of motion for different open quantum systems.

First, creating an initial set of equations for operators requires a specification of the Hamiltonian and dissipative processes in the system. For the later, one has to define both, dissipative operators, which we will also refer to as so-called jump-operators, and their corresponding dissipative rates. In code sample 4, these operators are collected in the vector `J` and the rates in `rates`. Both vectors have in this example only one entry. Following up, one can call the function `meanfield` as shown in the code sample below.

```
H = ∑(σ(2,2,i),i) + ∑(σ(2,1,i)*a + σ(1,2,i)*a',i)
ops = [σ(2,2,j)]
J = [σ(2,1,i)]; rates = [Γ]
eqs_mf = meanfield(ops,H,J;rates=rates,order=1)
eqs_complete = complete(eqs_mf)
evaluate(eqs_complete;limits=(N=>3))
scale(eqs_complete)
```

Code sample 4: *Example for creating equations of motions and completing them.* We use the Hamiltonian $H$ and operator $\sigma_j^{22}$ to instantiate a set of equations, using the dissipative operator $\sigma_i^{21}$ with its rate $\Gamma$. The function call `complete` completes the system, and we use `evaluate` and `scale` to transform them into a solvable set.

The function `meanfield` creates an initial set of equations for operator averages defined in the `ops` argument, using either Eq. (2.3.20) or Eq. (2.5.8). Which equation is calculated depends on the given `rates` and `J` argument. The indexed operators in the `ops` vector can not have any index, which is already used in the Hamiltonian of the system since an equivalently indexed operator commutes with the corresponding term in the Hamiltonian. The dissipative process operators $c_i$ get replaced by the

ones defined in the `J` argument. The keyword argument `rates` is a vector, with entries equivalent to either $\Gamma_{i,j}$ seen in Eq. (2.5.8) or $\Gamma_i$ seen in Eqs. (2.3.20). It is also possible to combine both approaches for different operators. It is important to note that the number of rates defined in the `rates` keyword coincides with the number of jump operators defined in `J`. If this condition is false, the algorithm to create an initial set will throw an error. Dissipative rates can be numerical values, symbols or `IndexedVariable` entities when used in this method call. As an example, code sample 5 shows different instantiations of the `rates` vector.

```
@cnumbers Γ
Γᵢ = IndexedVariable(:Γ,i)
J = [σ(2,1,i)]

rates_1 = [Γ] # symbolic
rates_2 = [Γᵢ] # indexed variable
rates_3 = [1.0] # numerical value
```

Code sample 5: *Example for different definitions of* `rates`. All three rates vectors are valid examples when used in the `meanfield` function.

When one uses a `IndexedVariable` with two indices, equations according to Eq. (2.5.8) are created instead. Creating these equations requires both coupling operators $c_i$ and $c_j$ to be specified. These operators can either be submitted to the function as a vector with two elements or by a single operator. When only a single operator is present, the second operator is assumed to be the same. Furthermore, the operators corresponding to the double-indexed rate must have the same indices as the variable. The `order` keyword attribute specifies the order of the cumulant expansion, and if neglected, automates to the highest order found in `ops`.

The initial set of equations created by the `meanfield` method does not necessarily need to be a closed one, this means that the equations might depend on averages with unknown values. To create a complete set, one calls the `complete` function or its in-place variant `complete!`. These functions iterate over all already calculated equations and collect any averages where the time derivatives are unknown. For any of these collected averages, the function adds a newly created equation using Eq. (2.3.20) and Eq. (2.5.8). Whether or not a derivative of an operator average is already known does not depend on index names, implying that the program can detect whether two equations are the same under exchanging indices. As an example, if the equation for $\langle\sigma_i^{12}\rangle$ is already known, no equation for $\langle\sigma_j^{12}\rangle$ will be created. The same is true if two averages contain the complex conjugate of each other's operator products. So if an equation for $\langle\sigma_i^{12}\rangle$ already exists, no equation for $\langle\sigma_i^{21}\rangle$ or $\langle\sigma_j^{21}\rangle$ is created.

As a result, we arrive at a closed set of equations, with all needed but no redundant equations. When using indices and symbolic summations, the individual system sizes do not matter when creating this complete set since the number of subsystems is

treated as a symbolic number. However, this changes when we insert numerical values for the symbolic indices. Such a step is necessary to solve the equations numerically. QuantumCumulants.jl implements two methods to achieve such a transformation. One is the `evaluate` function, which exchanges each index found with an explicit corresponding number and writes symbolic summations as additions of terms. This method creates equations proportional to the actual system size. When calling `evaluate`, it is necessary to specify any index ranges which are symbols rather than numbers. One can achieve this by using the `limits` keyword, which takes a dictionary mapping each symbolic parameter to its numerical value. We show one way of defining this keyword in code sample 4. Another way to transform the equations is the method `scale`. With `scale`, we assume that all constituents in the subsystem are indistinguishable, meaning that all of them are described by the same numerical value. This implies that for example, all $\langle \sigma_i^{22} \rangle$ have the same value. This approach allows a description of systems with enormous numbers of identical subsystems since it only enters as a numerical factor. This implemented scaling functionality is a key feature of the package and has many applications, as we will see in the later examples. Both functionalities `evaluate` and `scale` can be used on specific subsystems independently of each other. For example, consider a system with $N$ atoms interacting with $M$ cavity fields. One can `scale` the atoms and `evaluate` the cavity fields. Using the implemented keyword `h`, one can specify the Hilbert space upon which functionality gets applied.

To summarize, several steps are necessary for a user to obtain a set of differential equations for the corresponding system of interest. We give here a quick overview:

1. Hilbert spaces and indices are defined using the `Index` function.
2. Indexed operators are created by calling `IndexedOperator` and using the previously defined indices.
3. The Hamiltonian with symbolic summations, created by the $\sum$ function, is defined.
4. Jump operators and rates are defined to call the `meanfield` function.
5. The cumulant expansion and a `complete` algorithm are applied to the Equations created by this `meanfield` function.
6. Either `evaluate` or `scale` is called to the resulting set of equations.
7. The final result can be numerically solved using a differential equation solver such as DifferentialEquations.jl.

Furthermore, as a quick overview, we provide a list of all essential functionalities with a short description. For a full explanation of all the exported methods and additional examples of using the package, we refer to the current QuantumCumulants.jl documentary.

- `Index(hilb,name,range,aon)`
  The constructor of an Index entity, using a Hilbert space `hilb`, a `name`, a `range` and either a sub-Hilbert space or a number specifying the Hilbert space `aon`.

- `IndexedOperator(op,ind)`
  The constructor for an indexed operator, using a QuantumCumulants operator `op` and an index `ind`.

- `IndexedVariable(name,ind1,ind2)`
  A method for creating an indexed variable with up to two indices.

- `∑(term,ind1,ind2,excluded_indices)`
  A function one can use to create a symbolic summation over a term with up to two running indices. One can also specify an optional vector of indices to exclude them from the summation.

- `meanfield(ops,H,J;rates,order)`
  A function one can use to create a first initial set of equations derived for operators in `ops` using the Hamiltonian `H` together with the Jump-operators `J` with their rates up to the given expansion order.

- `complete(equations)`
  A method which automatically completes a given initial set of equations to its closed set under the cumulant-expansion method.

- `evaluate(equations;limits,h)`
  A function to evaluate a given set of equations containing symbolic summations and indices. If a maximum range of an index is a symbol, it will get replaced by a numerical value given in the `limits` keyword argument. Additionally, it is possible to evaluate only a subset of the Hilbert spaces of the system using the `h` keyword argument.

- `scale(equations;h)`
  A method designed to simplify and evaluate a subset of the system or the whole Hilbert space, depending on the keyword `h`. The simplification uses the approximation that all the operators of the chosen Hilbert space act with the same effective action, meaning that averages of two different operators result in the same outcome.

- `value_map(ps,p0;limits)`
  This method is designed to ease the process of generating a dictionary, which maps each symbolic character given in `ps` to its corresponding numerical value `p0`. The resulting dictionary is used for solving the equations. If a symbolic parameter is a `IndexedVariable` entity with a symbol as upper index limit, the `limits` keyword

must be set comparable to the case in `evaluate`. Indexed variables with one index are mapped to a vector of numerical values, and a variable with two indices to a two dimensional matrix.

# Chapter 5.

# Examples and Benchmarking

In this chapter, we demonstrate how to use the package and its most relevant usage of the package and most of its relevant features. We present a step-by-step guide through several examples available on the QuantumCumulants.jl documentary. We start with a simple example highlighting most of the necessary features.

## 5.1. Important Features

Let us now look at the code sample below, to see how these previous definitions of operators and indices are helpful when deriving equations of motion for the averages of such operators. First, we define our Hilbert space consisting of a `NLevelSpace` and a `FockSpace`. After that, we define as many indices as needed. Furthermore, we also create additional variables to describe our system in a more specific way. Using these variables, we continue by defining a Hamiltonian. We consider the Tavis-Cummings Hamiltonian for multiple atoms in a single mode cavity, similar to the one in Eq. (2.1.13). Finishing up the code example, we define `ops`, `J`, and `rates` using again the previously constructed operators and indexed variables. The last line of the code sample then creates an initial set of equations, according to the `meanfield` function, which we already described in Sec. 4.4.

```
using QuantumCumulants

@cnumbers N Δ κ

#define Hilbert space
ha = NLevelSpace(:atom,2)   # HilbertSpace of Atoms, in this case 2-Level-Atoms
hc = FockSpace(:field)   # HilbertSpace of the Field
h = ha ⊗ hc     # combined HilbertSpace

# Define some Indices
i = Index(h, :i, N, ha)
j = Index(h, :j, N, ha)
k = Index(h, :k, N, ha)

a = Destroy(h,:a)
σ(x,y,z) = IndexedOperator(Transition(h,:σ,x,y),z)

# Create indexed Variables
gi = IndexedVariable(:g,i)
γj = IndexedVariable(:γ,j)
νj = IndexedVariable(:ν,j)

# Defining the Hamiltonian
H_TC = Δ*a'*a + Σ(gi*(a'*σ(1,2,i) + a*σ(2,1,i)),i)

# Define rates and Jump-operators for derivation of the equations
J = [a,σ(1,2,j),σ(2,1,j)]
rates = [κ, γj, νj]

# automatic derivation of the second-order equation
eqs_mf = meanfield([σ(2,2,k)],H_JC,J;rates=rates,order=2)
```

Code sample 6: *Example for deriving equations of motion using indexed variables and operators.*

The output of code sample 6 is the equation of motion in a second order for $\sigma_k^{22}$. In particular, the exact output of this code reads:

$$\frac{d}{dt}\langle\sigma_k^{22}\rangle = \nu_k - 1ig_k\langle\sigma_k^{21}a\rangle + 1ig_k\langle\sigma_k^{12}a^\dagger\rangle - 1.0\gamma_k\langle\sigma_k^{22}\rangle - 1.0\nu_k\langle\sigma_k^{22}\rangle$$

The indexed variables defined in the previous code sample are used in this system to represent the interaction between the light field and the atoms in $g_i$ and the decay rates of the atomic operators $\gamma_i$ and their decoherence $\nu_i$. Here one can see another feature of the package, a variable only needs to be created a single time for all $N$ possible values it can have. They do not have a numeric value yet, but they can be assigned easily using vectors after the equations of motion have been evaluated.

```julia
# complete a given set of equations to a complete set
eqs_comp = complete(eqs_mf);

# evaluate the summations and create equations for all individual atoms
# in this case we calculate the equations for 5 atoms,
# which is specified in the limits keyword of the function
eqs = evaluate(eqs_comp;limits=(N=>5));

# solve the system by creating an initial state and construct a ODESystem
# Generate an ODESystem
using ModelingToolkit
@named sys = ODESystem(eqs)

# Solve the system using the OrdinaryDiffEq package
using OrdinaryDiffEq
u0 = zeros(ComplexF64,length(eqs))
p = [Δ, gi, γj, κ, νj, ωi]
g = [0.1,0.3,0.5,0.75,1.0] # different coupling strengths for different atoms
p0 = [0,g,0.25,1,1.5,1.0] # we set all γj and νj to the same values

# construct parameter-values for symbolic variables
p_ = value_map(p,p0;limits=(N=>5))

prob = ODEProblem(sys,u0,(0.0,10.0),p_)
sol = solve(prob,RK4()); #solve the problem
```

Code sample 7: *Example for completing and evaluating the previously derived equations.* The derived equations are first completed using the cumulant expansion method to order two and evaluated for $N = 5$ atoms. Additionally, we numerically solve the equations using the package OrdinaryDiffEq.jl [19].

In code sample 7 we use the previously derived equation of motion of the average of $\sigma_k^{22}$ to complete our system of equations. As a result of this automatic completion, we arrive at a total of 13 equations, which take the form

$$
\begin{aligned}
\langle \sigma_k^{22} \rangle &= \nu_k - i g_k \langle \sigma_k^{21} a \rangle + i g_k \langle \sigma_k^{12} a^\dagger \rangle - \gamma_k \langle \sigma_k^{22} \rangle - \nu_k \langle \sigma_k^{22} \rangle \\
\langle \sigma_k^{21} a \rangle &= -i \sum_{i \neq k}^{N} (g_i \langle \sigma_i^{12} \sigma_k^{21} \rangle) - i g_k \langle \sigma_k^{22} \rangle + i g_k \langle a^\dagger a \rangle + \dots \\
\langle a^\dagger a \rangle &= -i \sum_{i}^{N} (g_i \langle \sigma_i^{12} a^\dagger \rangle) + i \sum_{i}^{N} (g_i \langle \sigma_i^{21} a \rangle) - \kappa \langle a^\dagger a \rangle \\
&\vdots \\
\langle \sigma_k^{22} \sigma_l^{22} \rangle &= (-\gamma_k - \gamma_l - \nu_k - \nu_l) \langle \sigma_k^{22} \sigma_l^{22} \rangle + \nu_l \langle \sigma_k^{22} \rangle + \dots
\end{aligned}
$$

Since these equations still depend on symbolic indices and summations, we can not derive a numerical solution. To this end, we use the `evaluate` function, which converts the complete set of equations to a numerically solvable form for a specific atom number, in this case, for $N = 5$. In code sample 7, we also specify the number of atoms with five inside the `limits` keyword argument of the function. The output for the first couple of operator averages after calling this evaluate function is

$$\begin{aligned}
\langle \sigma_1^{22} \rangle &= \nu_1 - ig_1 \langle \sigma_1^{21} a \rangle + ig_1 \langle \sigma_1^{21} a \rangle^* - \gamma_1 \langle \sigma_1^{22} \rangle - \nu_1 \langle \sigma_1^{22} \rangle \\
\langle \sigma_2^{22} \rangle &= \nu_2 - ig_2 \langle \sigma_2^{21} a \rangle + ig_2 \langle \sigma_2^{21} a \rangle^* - \gamma_2 \langle \sigma_2^{22} \rangle - \nu_2 \langle \sigma_2^{22} \rangle \\
&\vdots \\
\langle \sigma_4^{22} \sigma_5^{22} \rangle &= ig_5 \langle a^\dagger \rangle \langle \sigma_4^{22} \sigma_5^{12} \rangle + i \langle \sigma_4^{22} \rangle \langle \sigma_5^{21} a \rangle^* + \dots
\end{aligned}$$

These equations are now solvable using a Differential Equations solver since all operator averages are defined with numerical indices. We use the packages OrdinaryDiffEq.jl and ModelingToolkit.jl to create a system and then a so-called `ODEProblem` with the previous equations. Before defining our problem, we construct numerical values for all the used variables using the vector `p0`. With those, we create a dictionary, which maps the symbolic variables to their numerical values using the function `value_map`, where we also use the keyword argument `limits` to set $N = 5$. This method creates a mapping for each interaction coupling $g$ for each individual atom. One can then use this dictionary with a vector stating the initial values of the operator averages in `u0` to construct a `ODEPorblem`. The range `(0.0,10.0)` used in the same function call is the time interval for which the solution is calculated. The problem can now be solved using the `solve` function exported by the OrdinaryDiffEq.jl package. In the final step, we plot both the photon number and the average excitation of each atom. We access the particular solutions for the averages using `NumberedOperators` as indices of the `sol` object.

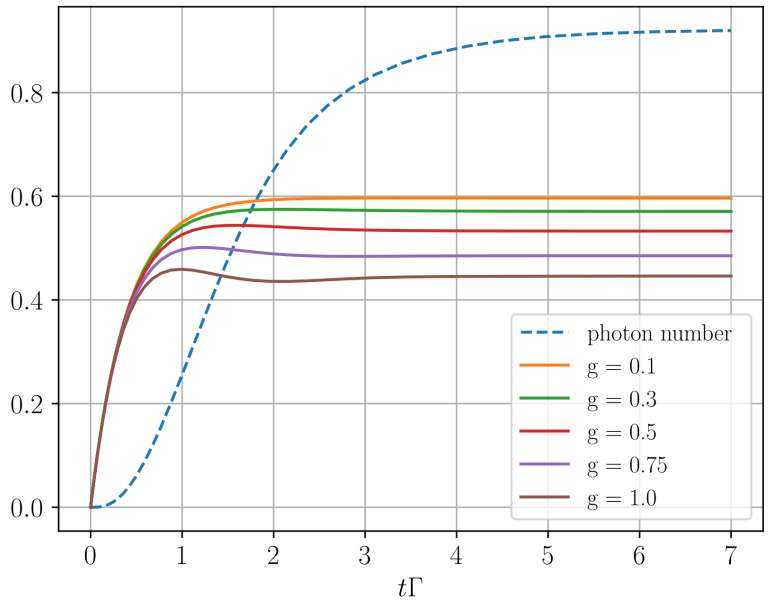


**Figure 5.1.:** *Output plot of the previous code sample 7.* The excited state population $\langle \sigma_i^{22} \rangle$ of the atoms inside the cavity is plotted against the evolution time of the system and is labeled according to their numerical interaction strength $g$. Additionally, the average photon number inside the cavity is shown by the dashed line.

## 5.2. Example A: A Superradiant Laser

Using these newly implemented functionalities, one can simulate various systems which consist of an arbitrary composition of multi-level atoms and light-field modes. As a compulsive example, we give a simulation of a superradiant laser, as described in [25]. A crucial feature of this model is that all averages are assumed to be indistinguishable, meaning that $\langle \sigma_i^{xy} \sigma_j^{xy} \rangle = \langle \sigma_1^{xy} \sigma_2^{xy} \rangle$ for all $i \neq j$. This symmetry is applied when using the `scale` function to a set of indexed equations of motions for operator averages. Fig. 5.2 shows a simplified illustration of the system.

Let us start by introducing the Hamiltonian of the system of interest

$$H = -\Delta a^\dagger a + \sum_{i=1}^{N} g_i (a^\dagger \sigma_i^{12} + a \sigma_i^{21}), \tag{5.2.1}$$

where $\Delta$ is the detuning between cavity- and atom-frequency and $g_j$ the coupling strength of the $j$-th atom. We can now use this Hamiltonian to derive a complete set of equations for operator averages by using the previously described methods. We start by importing the package and defining the Hilbert space and Hamiltonian, as seen in

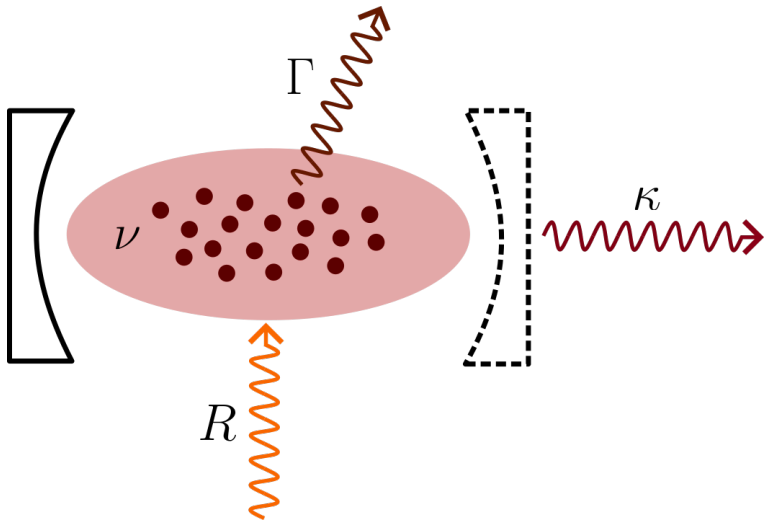


**Figure 5.2.:** *Schematic of the superradiant laser setup.* An ensemble of atoms is depicted in a lossy cavity, characterized by the photon decay rate $\kappa$. The atoms are driven by a laser with a rate $R$ and individually decay with a rate $\Gamma$. Additionally, the atoms are affected by dephasing, represented by the rate $\nu$.

code sample 8.

```
using QuantumCumulants

order = 2 #order of the cumulant expansion
@cnumbers N Δ κ Γ R ν

hc = FockSpace(:cavity)
ha = NLevelSpace(:atom,2)
h = hc ⊗ ha

# Indices and Operators
i = Index(h,:i,N,ha); k = Index(h,:k,N,ha); l = Index(h,:l,N,ha)

@qnumbers a::Destroy(h)
σ(i,j,k) = IndexedOperator(Transition(h,:σ,i,j),k)
g(i) = IndexedVariable(:g,i)

# Define the Hamiltonian
H = -Δ*a'a + (Σ(g(i)*a'*σ(1,2,i),i) + Σ(g(i)*a*σ(2,1,i),i))
```

Code sample 8: *Defining the Hilbert space and Hamiltonian for the superradiant laser example.*

The order of expansion for this example is set to two, but one can alter this easily at the cost of computation time. We also define the index $i$ again with a symbolic upper limit $N$ and use an indexed variable $g_i$ to use different couplings for later calculation. In the very last step of the upper code, we define our Hamiltonian according to Eq. (5.2.1).

We continue by constructing the Jump-operators with their rates and a set of operators for which we derive the equations.

```
J = [a,σ(1,2,l),σ(2,1,l),σ(2,2,l)]
rates = [κ, Γ, R, ν]
ops = [a'*a,σ(2,2,k)]

# create Meanfield-Equations with given order for the given operators
eqs = meanfield(ops,H,J;rates=rates,order=order)

# custom filter function
φ(x::Average) = φ(x.arguments[1])
φ(::Destroy) = -1
φ(::Create) =1
φ(x::QTerm) = sum(map(φ, x.args_nc))
φ(x::Transition) = x.i - x.j
φ(x::IndexedOperator) = φ(x.op)
φ(x::SingleSum) = φ(x.term)
φ(x::AvgSums) = φ(arguments(x))
phase_invariant(x) = iszero(φ(x))

eqs_c = QuantumCumulants.complete(eqs;filter_func=phase_invariant)
eqs_sc = scale(eqs_c)
```

Code sample 9: *Creating a complete set of equations of motions with a filter function.*

Additionally, we also define a filter function. Since the entire system is phase invariant, we can neglect terms with a phase. One can do this by setting the keyword argument of the complete function `filter_func` to the pre-defined `phase_invariant` function, which checks if a term has zero phases. In the last step of code sample 9, we call the `scale` function to scale up the set of equations, applying the before-mentioned symmetry rules onto the equations. The resulting complete set of equations then takes the form

$$
\begin{aligned}
\frac{d}{dt}\langle a^\dagger a\rangle &= -1.0\kappa\langle a^\dagger a\rangle - 1iNg_1\langle a^\dagger\sigma_1^{12}\rangle + 1iNg_1\langle a\sigma_1^{21}\rangle \\
\frac{d}{dt}\langle \sigma_1^{22}\rangle &= R - 1.0R\langle\sigma_1^{22}\rangle - 1.0\Gamma\langle\sigma_1^{22}\rangle + 1ig_1\langle a^\dagger\sigma_1^{12}\rangle - 1ig_1\langle a\sigma_1^{21}\rangle \\
\frac{d}{dt}\langle a^\dagger\sigma_1^{12}\rangle &= -0.5R\langle a^\dagger\sigma_1^{12}\rangle - 0.5\Gamma\langle a^\dagger\sigma_1^{12}\rangle - 0.5\kappa\langle a^\dagger\sigma_1^{12}\rangle - 0.5\nu\langle a^\dagger\sigma_1^{12}\rangle + \dots \\
\frac{d}{dt}\langle \sigma_1^{12}\sigma_2^{21}\rangle &= \left(-1.0R - 1.0\Gamma\right)\langle\sigma_1^{12}\sigma_2^{21}\rangle + 1ig_1\langle a^\dagger\sigma_1^{12}\rangle - 1ig_1\langle a\sigma_1^{21}\rangle + \dots
\end{aligned}
$$

Continuing, we define initial values and solve the equations before finally plotting the average photon number inside the cavity and the excitation of one atom.

```
# define the ODE System and Problem
@named sys = ODESystem(eqs_sc)
# Initial state
u0 = zeros(ComplexF64, length(eqs_sc))
# System parameters
N_ = 2e5; Γ_ = 1.0; Δ_ = 2500Γ_; g_ = 1000Γ_
κ_ = 5e6*Γ_; R_ = 1000Γ_; ν_ = 1000Γ_
ps = [N, Δ, g(1), κ, Γ, R, ν]
p0 = [N_, Δ_, g_, κ_, Γ_, R_, ν_]
prob = ODEProblem(sys,u0,(0.0, 1.0/50Γ_), ps.=>p0);
# Solve the Problem
sol = solve(prob,Tsit5(),maxiters=1e7)
# Plot time evolution
t = sol.t; n = real.(sol[a'a]); s22 = real.(sol[σ(2,2,1)])
```

Code sample 10: *Solving the previously derived equations.*

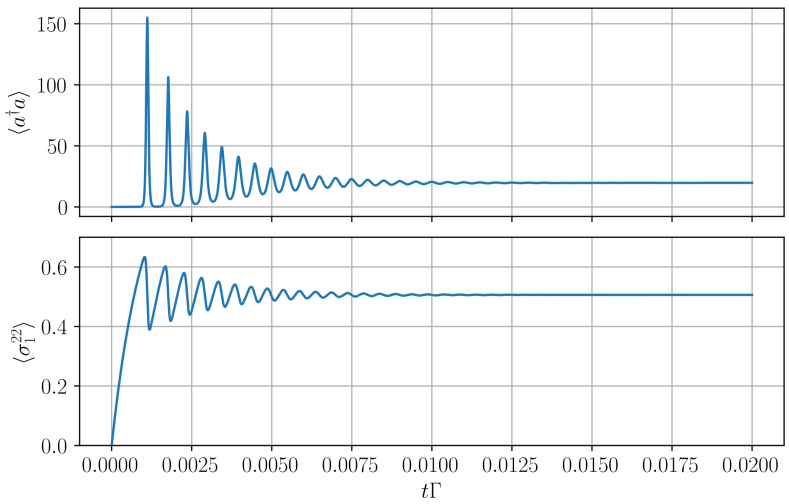


**Figure 5.3.:** *Plots of the solution of the previous code sample 10 for the number of photons* $\langle a^\dagger a \rangle$ *and average excitation of the atoms* $\langle \sigma_1^{22} \rangle$.

To show more functionalities of the package, we also include a calculation of the spectrum using two-time correlation functions. One can do so by calling the functions `CorrelationFunction` and `Spectrum` as shown in code sample 11.

```
corr = CorrelationFunction(a', a, eqs_c; steady_state=true,
    filter_func=phase_invariant)
corr = scale(corr)
S = Spectrum(corr, ps);

prob_ss = SteadyStateProblem(prob)
sol_ss = solve(prob_ss, DynamicSS(Tsit5(); abstol=1e-8, reltol=1e-8),
    reltol=1e-14, abstol=1e-14, maxiters=5e7);

ω = [-10:0.01:10;]Γ_
spec = S(ω,sol_ss.u,p0)
spec_n = spec ./ maximum(spec)
δ = abs(ω[(findmax(spec)[2])]);
```

Code sample 11: *Calculating the correlation function and the spectrum of the emitted steady state output.*

We use here the function `CorrelationFunction`, to derive $\langle a^\dagger(t)a(\tau)\rangle$ for our system using the previously derived complete set of equations `eqs_c`. This method returns a complete set of equations of correlation functions between system operators and $a(\tau)$. Here the keyword `steady_state` is set to `true` since we are interested in the output spectrum of the system after it reaches its steady state. We then continue by scaling these equations and defining our spectrum `S`. At this point, `S` is an abstract function that can be called using a range of desired frequencies, in this case, $\omega$. The steady-state solution for the operator averages of our system `sol_ss.u` and the numerical values for parameters encountered in the correlation function `p0` are also provided to the spectrum function. After calculating the spectrum, we normalize it in the last few steps and plot the result in Fig. 5.4.

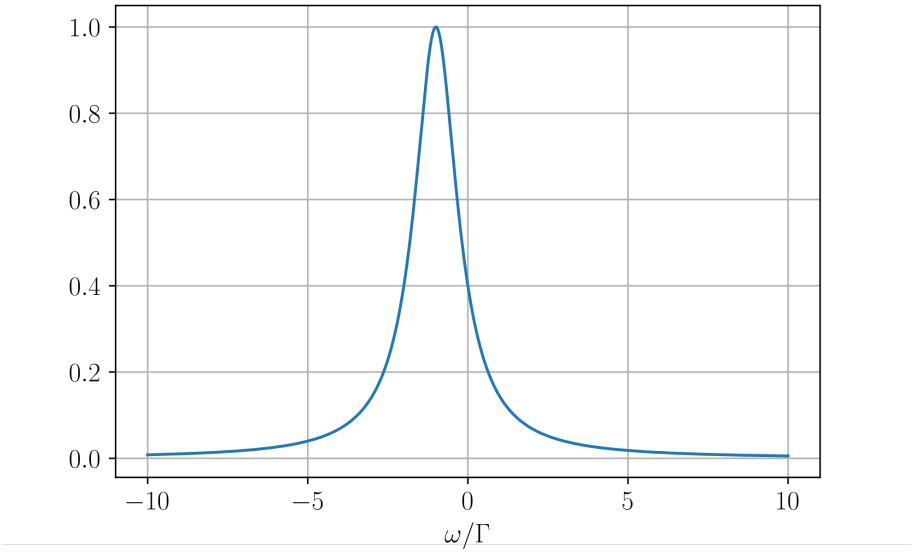


**Figure 5.4.:** *Normalised spectrum of the laser for the above example.* The spectrum is calculated by Fourier transforming the two-time correlation function.

## 5.3. Example B: Cavity Anti-resonance

In this example, we want to highlight further advantages of the indexing extension for QuantumCumulants.jl. It is also possible within the package to create summations with up to two running indices, saving even more derivation time. We give an example of a system where we construct summations with two indices, similar to the model described in [24]. The system contains a cavity field interacting with atoms, including dipole-dipole interaction. Additionally, a laser drives the system through one of the cavity mirrors. The Hamiltonian of the system is similar to the Tavis-Cummings Hamiltonian given in Eq. (2.1.13)

$$H = \Delta_c a^\dagger a + \eta(a^\dagger + a) + \Delta_a \sum_i^N \sigma_i^{22} + \sum_{i\neq j}^N \sum_{j\neq i}^N \Omega_{ij} \sigma_i^{21} \sigma_j^{12} + \sum_i^N g_i (a^\dagger \sigma_i^{12} + a\sigma_i^{21}). \quad (5.3.1)$$

Here, $\Delta_c$ and $\Delta_a$ are the detunings of the cavity and atoms to the gain laser, respectively. We see that this Hamiltonian consists of several summations, one of which is a summation over two running indices $i$ and $j$. This term corresponds to the dipole-dipole interactions between the atoms. We also encounter here a variable with two indices, namely $\Omega_{ij}$. This double-indexed variable behaves in the first place like a symbolic complex number. However, as in the example below, we defined the variable to have the keyword `identical` set to `false`, meaning that $\Omega_{ii} = 0$ for all $i$. One can use this small optimization to

neglect terms that contain variables like $\Omega_{ii}$ immediately.

```
using QuantumCumulants
order = 2; @cnumbers Δc η Δa κ N
# define Hilbert space
hc = FockSpace(:cavity); ha = NLevelSpace(Symbol(:atom),2); h = ha ⊗ hc
# define indices
i = Index(h,:i,N,ha); j = Index(h,:j,N,ha); k = Index(h,:k,N,ha)
# define indexed variables
gi = IndexedVariable(:g,i); Γ_ij = IndexedVariable(:Γ,i,j);
Ω_ij = IndexedVariable(:Ω,i,j;identical=false)
@qnumbers a::Destroy(h)
σ(x,y,k) = IndexedOperator(Transition(h,:σ,x,y),k)
# Hamiltonian
Hc = Δc*a'a + η*(a' + a)
Ha = Δa*Σ(σ(2,2,i),i) + Σ(Ω_ij*σ(2,1,i)*σ(1,2,j),j,i;non_equal=true)
Hi = Σ(gi*(a'*σ(1,2,i) + a*σ(2,1,i)),i)
H = Hc + Ha + Hi

# Jump operators & and rates
J = [a, σ(1,2,i) ]
rates = [κ,Γ_ij]
ops = [a, σ(2,2,k), σ(1,2,k)]
eqs = meanfield(ops,H,J;rates=rates,order=order)
eqs_comp = complete(eqs);

eqs_ = evaluate(eqs_comp;limits=(N=>2));
```

Code sample 12: *Code sample for deriving equations of motions for a cavity with 2 atoms and dipole-dipole coupling.*

As mentioned, we can create a summation with two running indices using the same methods one uses to instantiate single-indexed summations. We also set the keyword `non_equal` to `true`, indicating that the two running indices of the summation can never be equal. In code sample 12, we also define a dissipative rate $\Gamma_{ij}$ with two indices. This rate corresponds to the collective decay rate of the closely spaced atoms, and when deriving the equations of motion, it is used equivalently as in Eq. (2.3.20). In this particular example, the second dissipative coupling operator is automatically derived from the first. Therefore, for the rate $\Gamma_{ij}$ with the corresponding coupling operator $c_i = \sigma_i^{12}$, the second operator $c_j^\dagger$ is automatically set to be $\sigma_j^{21}$.

We continue the example by deriving a set of equations and transforming them into a solvable form using the `evaluate` function. We call the method in this example using the keyword `limits` to set the number of atoms to two. We use a steady-state solver for our equations to determine the photon number in the cavity $|\langle a \rangle|^2$ of the system. To resolve the transmission of the gain Laser through the cavity, we scan the laser detuning. As a last step, we normalize the photon number by their maximum and plot the result. One finds an anti-resonance feature of the transmitted light with a sharp phase response as shown in Fig. 5.5.

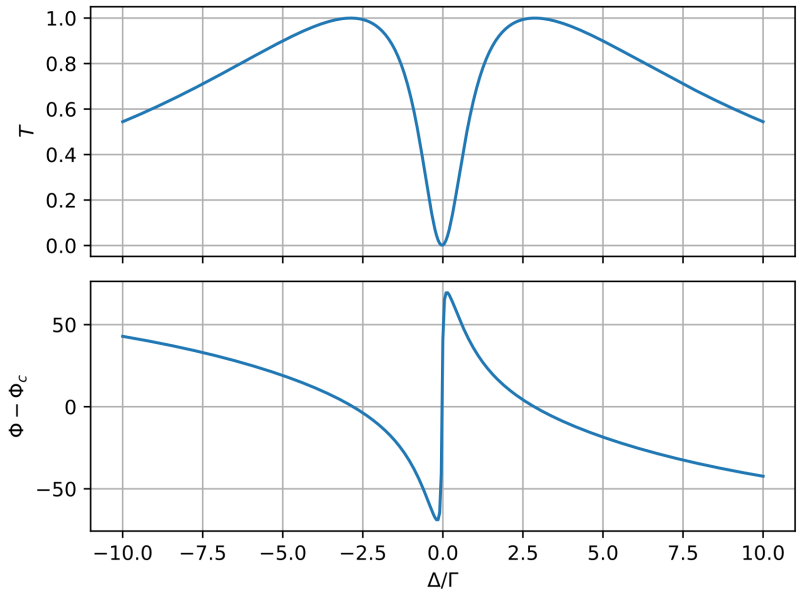


**Figure 5.5.:** *The transmission coefficient $T$ of the cavity and the phase $\Phi$ in degrees are plotted against the laser detuning in units of* $\Gamma$. The transmission coefficient $T$ is calculated by normalizing the photon number in the cavity in the steady state.

## 5.4. Example C: Ensemble of Two-Level Atoms in an optical Array

For this example, we present a system of several two-level atoms interacting via the dipole-dipole interaction. The atoms are arranged in a two-dimensional optical array with variable lattice parameters. In this example, we are interested in the case where the polarizations of the atom-transition dipoles $\vec{\mu}$ are perpendicular to the lattice plane. The atoms are excited by a $\pi/2$ laser-pulse from a specific angle. Fig. 5.6 depicts a schematic drawing of the system of interest.

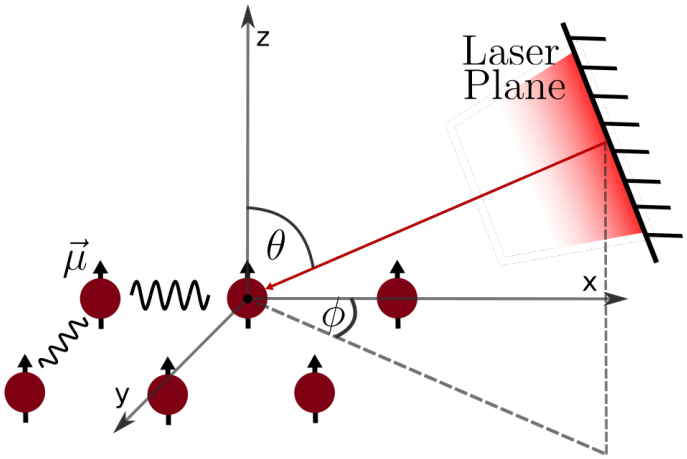


**Figure 5.6.:** *Schematic of the atomic array model.* The figure depicts the atoms with their corresponding dipole moment $\vec{\mu}$ as well as the angles $\phi$ and $\theta$ specifying the direction of the laser plane.

Since we are interested in the dipole-dipole interactions and the collective decay of the atomic array, we are considering the Hamiltonian

$$H = -\Delta_a \sum_i^N \sigma_i^{22} + \sum_{i \neq j}^N \Omega_{ij} \sigma_i^{12} \sigma_j^{21} + \frac{i}{2} \Omega_{\mathrm{D}}(t) \sum_i^N \left( \Phi_i \sigma_i^{21} - \Phi_i^* \sigma_i^{12} \right) \tag{5.4.1}$$

with $\Delta_a$ being the atom-laser detuning, and $\Omega_{ij}$ the coherent coupling between atom $i$ and $j$. We introduce here a time-dependent pump laser $\Omega_{\mathrm{D}}(t)$, which is essentially a rectangular function with a set width and amplitude. We use this function to initiate the system in a preferable state and to simulate an initial $\pi/2$-laser pulse. $\Phi_i$ is a parameter describing the phase of the laser pulse at the position of atom $i$. It is a geometric property, depending on the setup of our system and is calculated via $\Phi_i = e^{-i\vec{k}\vec{r_i}}$, where $\vec{k}$ is the wave vector of the incoming laser-light and $\vec{r_i}$ the positional vector of atom $i$, based on the laser-plane. We also choose the relative position of the array corresponding to the laser plane to be at zero phase in the middle of the lattice. To simulate the system with QuantumCumulants.jl, we start by loading the packages and defining the Hilbert space, see code sample 13.

```
using QuantumCumulants, Symbolics
@syms t::Real
@cnumbers Δ_a Ω_R N
@register Ω(t)
Nx=3; Ny=3; N_N = Nx*Ny; order_=1;
h = NLevelSpace(Symbol(:atom),2);
i = Index(h,:i,N,h); j = Index(h,:j,N,h); k = Index(h,:k,N,h);
σ(i,j,k) = IndexedOperator(Transition(h,:σ,i,j),k)
Γ(i,j) = IndexedVariable(:Γ,i,j)
Ω(i,j) = IndexedVariable(:Ω,i,j);
Φ(i) = IndexedVariable(:Φ,i); Φd(i) = IndexedVariable(:Φd,i) ;
Hₐ = -Δ_a*Σ(σ(2,1,i)*σ(1,2,i),i)
Hₑₗ = Σ(Σ((Ω(i,j)*σ(2,1,i)*σ(1,2,j)),j,[i]),i,[j])
Hₚᵤₘₚ =  im*(1/2)*Ω(t)*Ω_R*Σ((Φ(i)*σ(2,1,i) - Φd(i)*σ(1,2,i)),i)
H = Hₐ + Hₑₗ + Hₚᵤₘₚ
J = [σ(1,2,i)]; rates = [Γ(i,j)]; ops = [σ(2,2,k)]
eqs = meanfield(ops,H,J;rates=rates,iv=t,order=order_)
eqs_complete = complete(eqs);
eqs_eval = evaluate(eqs_complete;limits=(N=>N_N));
```

Code sample 13: *Deriving equations of motions for a three-by-three array of two-level atoms.* We use here the package Symbolics.jl [22] to define the invariant variable $t$, to simulate the time dependency of the function $\Omega(t)$, which we introduced in the system Hamiltonian from Eq. (5.4.1).

Following up, we calculate the numerical values for all the parameters needed. For this, we choose the distance between two neighbouring atoms to be not too far apart from each other so that we can still observe collective behaviour. To be precise, the lattice constant is in both dimension set to $a = 0.1\lambda$ with $\lambda$ being the wavelength of the emitters. Furthermore, we calculate the laser phase, having the zenith angle $\theta$ and azimuth angle $\phi$, as seen from the array centre. Using all of these, we can simulate the collective behaviour of the array in different orders of the expansion and extract the expectation value of a quantum mechanical operator after some evolution time $T$.

Let us first consider the case where we have a three-by-three array. This array size allows us to do calculations in the second order of the cumulant expansion, as the number of equations needed to describe the system is not too large for solving it efficiently. For example, we want to know how the expectation value of the excited state population of the middlemost atom changes over time compared to an atom at the corner of the array for a laser positioned at the angles $\theta = \pi/2$ and $\phi = 0$. We plot the time evolution in first-order and second-order expansion for both atoms of interest and for a free single atom which is not interacting via the dipole-dipole interaction, see Fig. 5.7.

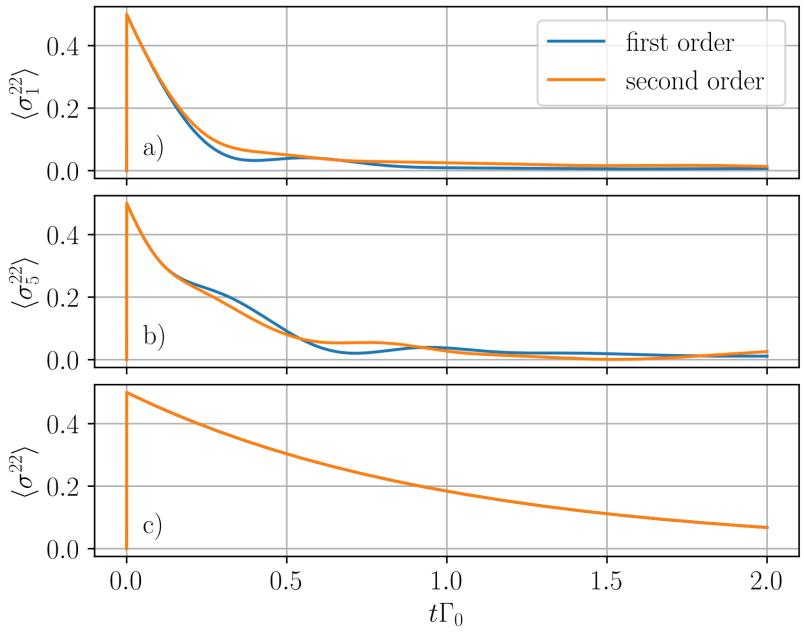


**Figure 5.7.:** *Comparison of the excited state population expectation value between an atom at the corner (a) of the array to the centre atom (b) and a free single atom (c).* The simulation was made for a three-by-three array in first and second-order expansion, using a laser at angular positions $\theta = \pi/2$, $\phi = 0$ and a lattice constant of $a = 0.1\lambda$. In the case of a single free atom (c), the first and second-order expansion lead to the same result. The reason is that the system is describable solely by using the equations of motions for first-order averages since no higher-order products can occur.

Continuing the example, we increase the system size of the array to a nine-by-nine array. For this, we use the first-order expansion, commonly known as the mean-field approximation, where we reduce all averages of operator multiplications as a product of the individual operator averages. As mentioned, we do not need to specify an entirely new system. We only need to vary a numerical value in our code and calculate additional parameters. For deriving the equations, we change the numerical value in the `limits` keyword of the `evaluate` function. As output, we plot the average population of the excited level for an atom in the middle of the array, an atom at the corner and a free atom. The result is depicted in Fig. 5.8.

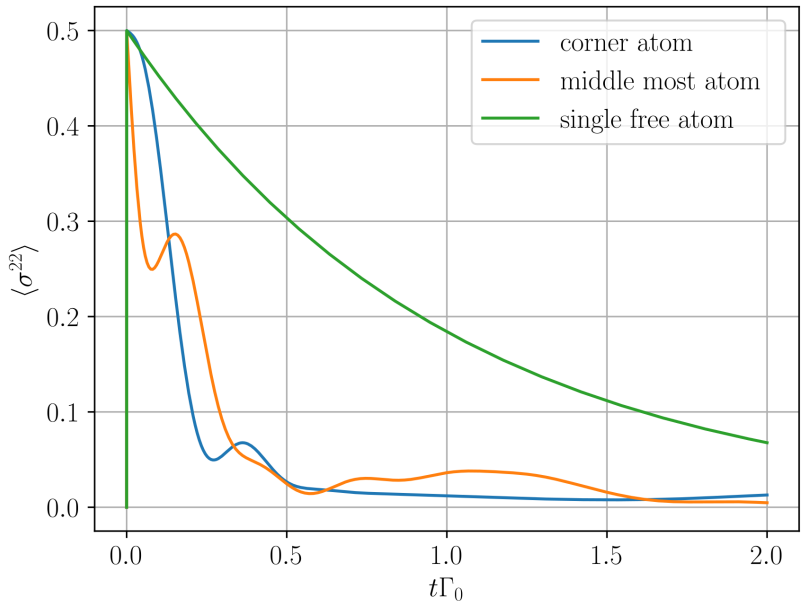


**Figure 5.8.:** *Comparison of the excited state population expectation value between different atoms confined in a nine-by-nine optical array construction.* The simulation was made for a nine-by-nine array in first-order expansion, using a laser at angular positions $\theta = \pi/2$, $\phi = 0$ and a lattice constant of $a = 0.1\lambda$.

As a final examination of this example system, we analyse how the angular dependency of the laser impacts the decay of the excited state population of the atoms in the array. We do so by iterating over the angular positions between the gain laser and the array. We then continue by solving the equations of motion for each iteration individually. For this, we calculate for each angular step in $\theta$ and $\Phi$ the excited population average of each atom and take the mean value of the whole array after a set evolution time of $t = 1/\Gamma_0$.

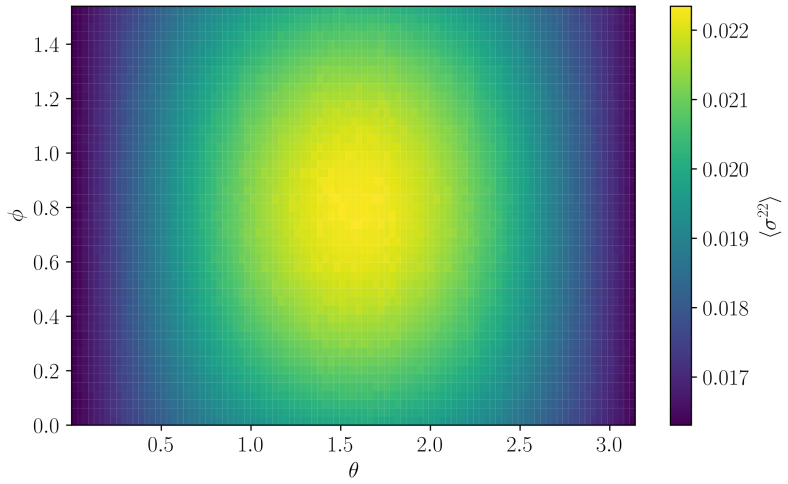


**Figure 5.9.:** *Average excitation of each atom in the array depending on the laser angular positions after an evolution time of* $t = 1/\Gamma_0$. Both angles $\phi$ and $\theta$ are represented in units of $\pi$. The lattice constant is in both directions set to $a = 0.1\lambda$, with $\lambda$ being the wavelength of the emitters. Brighter colours (yellow) represent a higher average excited state population, and darker colours (blue) refer to a lower average excited state population.

As we see in Fig. 5.9, the average excitation of the lattice is at its maximum after $t = 1/\Gamma_0$, when the wave vector $\vec{k}$ of the incoming laser light is perpendicular to the dipole moment $\vec{\mu}$ of the individual atoms. The value for a free atom at the chosen point in time of $t = 1/\Gamma_0$ is $\langle\sigma^{22}\rangle = 0.196$.

## 5.5. Example D: Laser with Filter Cavities

In this final example, we want to show the combined usage of both functionalities `evaluate` and `scale`. Furthermore, in this example, we use symbolic indices on bosonic operators and on atomic ones. Let us first start by introducing the system and its Hamiltonian. We are considering multiple cavities with different detunings coupled to a system similar to the one described in the Superradiant Laser example in Sec. 5.2. Such a setup is used to calculate the spectrum of a laser by filtering the emitted light using filter cavities. One can do so by measuring the photon number within each filter cavity, as seen in [26]. With the additional cavities the Hamiltonian has the form

$$H = -\Delta a^\dagger a + g\sum_{j=1}^{N}(a^\dagger\sigma_j^{12} + a\sigma_j^{21}) - \sum_{i=1}^{M}\delta_i b_i^\dagger b_i + g_f\sum_{i=1}^{M}(a^\dagger b_i + a b_i^\dagger). \tag{5.5.1}$$

In this equation, $\delta_i$ is the detuning of the $i$-th filter cavity, and $g_f$ is the coupling strength between the gain cavity and the filter cavities. Each filter cavity has the same coupling strength $g_f$ and photon decay rate $\kappa_f$. In this setup, the symbolic index $j$ describes the atomic Hilbert space and the symbolic index $i$ is the bosonic one of the filter cavities. In code sample 14, this Hamiltonian is defined and the equations of motion are derived.

```
# Paramters
@cnumbers κ g gf κf R Γ Δ ν N M
δ(i) = IndexedVariable(:δ, i)
# Hilbertspace
hc = FockSpace(:cavity)
hf = FockSpace(:filter)
ha = NLevelSpace(:atom, 2)
h = hc ⊗ hf ⊗ ha
# Indices and Operators
i = Index(h,:i,M,hf)
j = Index(h,:j,N,ha)
@qnumbers a::Destroy(h,1)
b(k) = IndexedOperator(Destroy(h,:b,2), k)
σ(α,β,k) = IndexedOperator(Transition(h,:σ,α,β,3), k)
# Hamiltonian
H = Δ*Σ(σ(2,2,j),j) + Σ(δ(i)*b(i)'b(i),i) +
    gf*(Σ(a'*b(i) + a*b(i)',i)) + g*(Σ(a'*σ(1,2,j) + a*σ(2,1,j),j))
# Jumps & rates
J = [a, b(i), σ(1,2,j), σ(2,1,j), σ(2,2,j)]
rates = [κ, κf, Γ, R, ν]
eqs = meanfield(a'a,H,J;rates=rates,order=2)
eqs_c = complete(eqs)
M_ = 20 # 20 cavities
eqs_sc = scale(eqs_c;h=[ha]) # scale only atomic Hilbert space
eqs_eval = evaluate(eqs_sc; limits=Dict(M=>M_)) # evaluate the rest
```

Code sample 14: *Derivation of a closed set of equation for the filter cavities.* The equations are first derived using the Hamiltonian in Eq. (5.5.1) and then both scaled and evaluated. The `scale` function is used on the atomic Hilbert space by setting the keyword `h` to `[ha]` and the `evaluate` function on the Hilbert space describing the filter cavities.

In code sample 14, we use `scale` with the keyword `h` set to the atomic Hilbert space. After the scaling of the equations, we evaluate them with the number of filter cavities $M$ set to 20. The detunings of the cavities $\delta_i$ range from $\delta_1 = 0$ up until $\delta_{20} = 9.5\Gamma_0$ in $0.5\Gamma_0$ steps. Solving these equations numerically for $g = \Gamma_0$, $g_f = 0.1\Gamma_0$, $\kappa_f = 0.1\Gamma_0$, $R = 10\Gamma_0$, $\kappa = 100\Gamma_0$, $\nu = \Gamma_0$ and $N = 200$ atoms in the main cavity gives the results shown in Fig. 5.10.

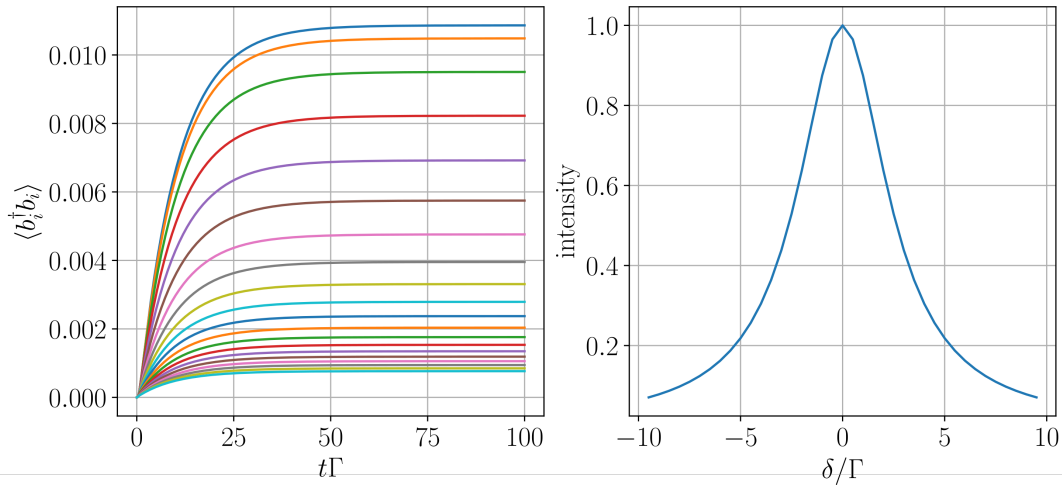


**Figure 5.10.:** *Average photon number of the filter cavities (left) and the normalized intensity depending on the detuning from the main cavity (right).* The uppermost filter cavity (blue) corresponds to the one with the lowest detuning $\delta_1 = 0$ and the lowest one (dark green) to the highest detuning $\delta_{20} = 9.5\Gamma_0$. The normalized intensity curve (right) is extrapolated from the steady state photon number of the filter cavities depending on their detuning.

## 5.6. Benchmarking

To test the performance of the package regarding different system sizes, we vary the number of atoms included in two examples, a linear chain and an optical array, similar to the one in Sec. 5.4. We compare calculation times and resources for these systems. Furthermore, we compare the brute force algorithm to the newly implemented one using indexed operators and equations. For this, we compare the resources needed to transform the equations of motion into a solvable set, i.e. time and memory allocations for the function call `evaluate`, and for completion, i.e. the function `complete`. For the first example, we derive the equations of motions in a second order for the system, explained in Sec. 5.4 for a $N$-by-$N$ array, where we continuously increase $N$ and measure both calculation time and memory allocations. We do so by using another Julia package called TimerOutputs.jl. Fig. 5.11 shows the results for this benchmark.

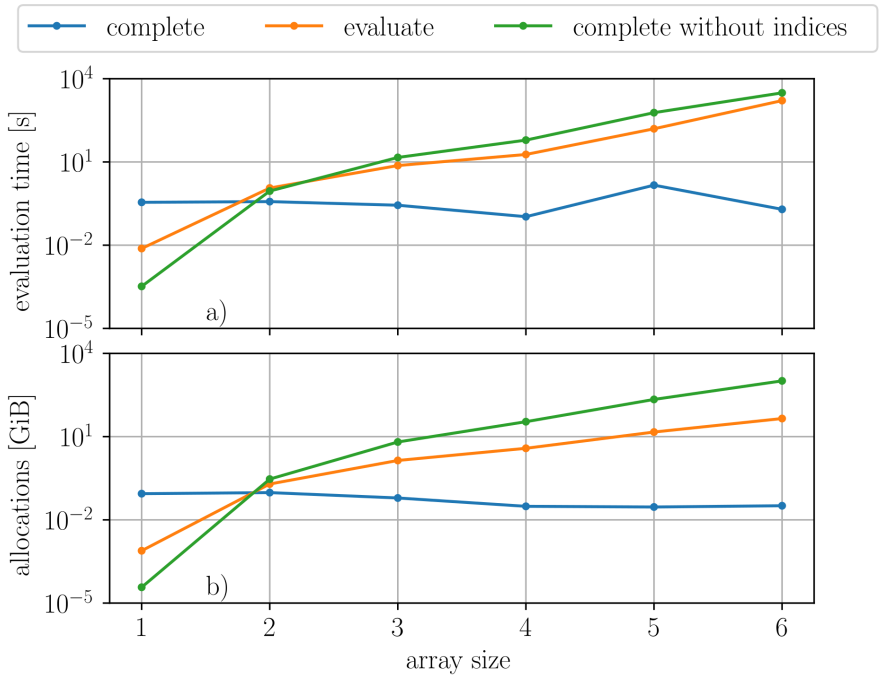


**Figure 5.11.:** *Comparison of derivation times and memory allocations needed between different algorithms.* In (a), calculation times of different function calls are plotted against the number of atoms included in the array on a logarithmic scale. In (b), memory allocations of the same function calls are shown. The blue graph is associated with the resources needed for the `complete` function call, and the orange one with the `evaluate` function. The green graph shows the resources for the same system, where one does not use symbolic indexing.

As shown in Fig. 5.11, both time needed to complete the equations and memory allocations are constant over the system size. What is also notable about both graphs is that for sizable systems, the total time and allocations needed for the `evaluate` function is lower than the total for a `complete` call without using indices, as one would expect. Similarly, for a comparison in first-order expansion, we vary a system consisting of a linear atom chain, including dipole-dipole interaction terms. We measure once again derivation time and memory allocations. Fig. 5.12 displays the results of the benchmarking.

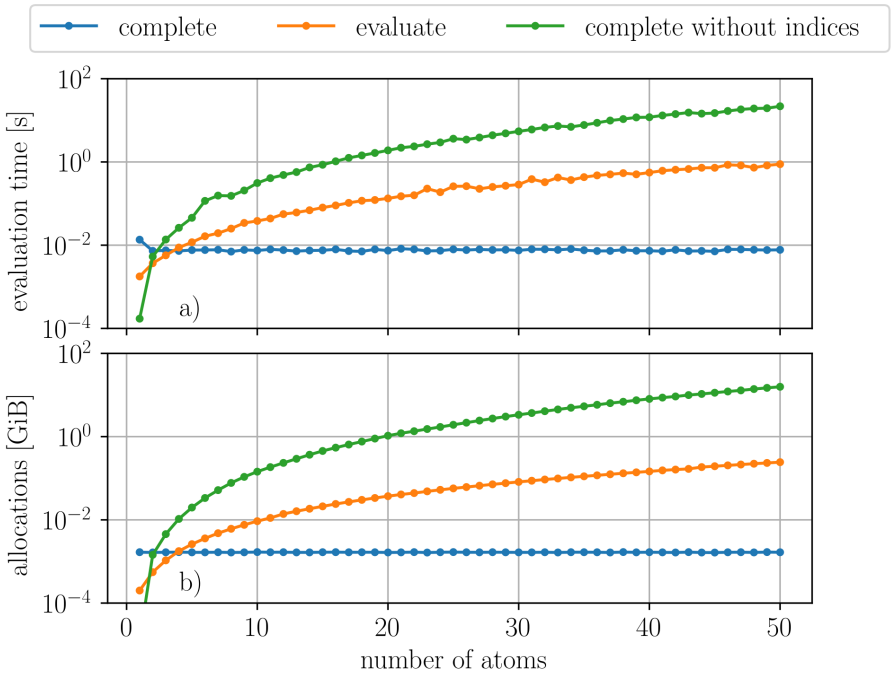


**Figure 5.12.:** *Comparison of calculation times and memory allocations needed between different algorithms for a linear atomic chain in first-order expansion.* In (a), the calculation times of different function calls are plotted against the number of atoms included. In (b), memory allocations of the same function calls are shown. Both plots display the results on a logarithmic scale. The blue graph is associated with the resources needed for the `complete` function call, and the orange one with the `evaluate` function. The green graph shows the resources for the same system, where one does not use symbolic indexing.

When using symbolic indexing, resources needed for the `complete` call are again constant over different system sizes. The blue graph in Fig. 5.12 shows this feature. Furthermore, it is visible that using `evaluate` is more efficient in time and allocations than not using indices and summations at all. For a sizable system with 50 atoms, the `evaluate` function is significantly more efficient then the algorithm without symbolic indexing.

All testing regarding benchmarking was done using an AMD Ryzen 5 3400G CPU, an NVIDIA GeForce GTX960 GPU and 16GB of DDR4 2666MHz RAM on QuantumCumulants.jl version v0.2.13.

# Chapter 6.

# Conclusion and Outlook

We added the possibility to use symbolic indices and summations to the Julia package QuantumCumulants.jl. With this implementation, it is possible to implement and simulate quantum-optical systems, which consist of several similar subsystems, efficiently. One can use these symbolic summations and indices to ease the creation of systems of interest and reduce both memory allocations required and calculation time for deriving equations. By representing additions as symbolic summations, we reduce redundant calculus. In addition, by rewriting similar differential equations using indexed operator expectation values, we also reduce the number of equations needed to describe the quantum dynamics of the system of interest. Furthermore, we added new possibilities to transform these equations with the `scale` and `evaluate` functionalities. The output of these functions is interpretable for differential equation solvers like DifferentialEquations.jl. In addition to indexed operators, we also implemented `IndexedVariables` to help the user construct additional parameters in an intuitive way. We showcased the potential of symbolic indexing in various examples, mostly building upon the Tavis-Cummings model. Highlighted examples include recently investigated systems such as the cavity-anti-resonance [24] and the superradiant laser [25]. Finally, we show that using symbolic summations reduces the computational resources needed to calculate specific systems by doing benchmark testing on two different example models.

## 6.1. Current Limitations

For now, one of the extensive limitations is that systems requiring more than two indices to describe a single parameter are not yet possible to simulate. The same limitation also holds for models defined by summations with three or more indices. For higher orders of indices, the calculation rules for these objects are progressively more complicated. Since there are already a lot of systems that can be effectively described by only using summations and parameters using two or fewer indices, this was not a focus point in the initial development.

## 6.2. Further Development

Since QuantumCumualants.jl is under current development, more extensions will likely get created for this package to handle more use cases. For example, it is not yet possible for the toolbox to simulate fermionic operator or corresponding Hilbert spaces. One of the primary goals for the indexing extension is to further decrease computation times for larger systems by increasing the efficiency and speed of the methods provided. On the other hand, a problem one might want to tackle in the future is the creation of multi-indexed summations and variables. Although one can calculate many systems using only two indices, more cumbersome systems with multiple indices and summations should also profit from the extension.

# Appendices

# Appendix A.

# Source code of important functions

Here, we show the source code of the most critical implemented methods. We start with the `IndexedOperator` and the `SingleSum` objects.

```
struct IndexedOperator <: QSym
    op::IndexableOps
    ind::Index
    function IndexedOperator(op::IndexableOps,ind::Index)
        @assert isequal(ind.hilb,hilbert(op))
        isa(ind.hilb, ProductSpace) && (@assert isequal(acts_on(op),ind.aon))
        return new(op,ind)
    end
end
const Summable = Union{<:QNumber,<:CNumber,
    <:SymbolicUtils.Sym{Parameter,IndexedVariable},
    <:SymbolicUtils.Sym{Parameter,DoubleIndexedVariable}}
const IndexedObSym = Union{IndexedOperator,
    SymbolicUtils.Sym{Parameter,IndexedVariable},
    SymbolicUtils.Sym{Parameter,DoubleIndexedVariable}}
struct SingleSum{M} <: QTerm #Sum with an index
    term::Summable
    sum_index::Index
    non_equal_indices::Vector{IndexInt}  #indices not equal to the summation index
    metadata::M
    function SingleSum(term::Summable,sum_index::Index,
            non_equal_indices::Vector,metadata)
        SymbolicUtils._iszero(term) ? 0 : new{typeof(metadata)}
            (term,sum_index,sort(non_equal_indices,by=getIndName),metadata)
    end
end
function SingleSum(term::IndexedObSym,
        sum_index, non_equal_indices;metadata=NO_METADATA)
    term_indices = get_indices(term)
    if sum_index in term_indices
        return SingleSum(term,sum_index,non_equal_indices,metadata)
    else
        return (sum_index.range - length(non_equal_indices)) * term
    end
end
```

Code sample 15: *Source code of the implementation of the* `IndexedOperator` *and* `SingleSum`.

Continuing, we show how the multiplication of a summation with an indexed operator is implemented.

```
function *(sum::SingleSum,elem::IndexedObSym)
    NEIds = copy(sum.non_equal_indices)
    if !((elem.ind == sum.sum_index) && (elem.ind ∉ NEIds)
            && (sum.sum_index.aon == elem.ind.aon))
        qaddterm = nothing
        term = sum.term
        if length(NEIds) == 0
            extraterm = change_index(term,sum.sum_index,elem.ind)
            qaddterm = extraterm*elem
        else
            specNEIs = Tuple{Index,Index}[]
            for ind in NEIds
                tuple = (elem.ind,ind)
                push!(specNEIs,tuple)
            end
            extraterm_ = change_index(term,sum.sum_index,elem.ind)
            qaddterm = reorder(extraterm_*elem,specNEIs)
        end
        push!(NEIds,elem.ind)
        qmul = sum.term*elem
        if qmul isa QMul
            qmul = order_by_index(qmul,[sum.sum_index])
        end
        if (qmul isa QMul && (isequal(qmul.arg_c,0)
                || SymbolicUtils._iszero(qmul.args_nc)))
            return 0
        end
        sort!(NEIds,by=getIndName)
        newsum = SingleSum(qmul,sum.sum_index,NEIds)
        if SymbolicUtils._iszero(newsum)
            return qaddterm
        elseif SymbolicUtils._iszero(qaddterm)
            return newsum
        end
        return QAdd([newsum,qaddterm])
    else
        qmul = sum.term*elem
        if qmul isa QMul
            qmul = order_by_index(qmul,[sum.sum_index])
        end
        qmul isa QMul && (isequal(qmul.arg_c,0)
            || SymbolicUtils._iszero(qmul.args_nc)) && return 0
        return SingleSum(qmul,sum.sum_index,NEIds)
    end
end
```

Code sample 16: *Implementation of a multiplication of a summation and an operator.* As a final example we show the implementation of the `meanfield` function.

```
function indexed_meanfield(a::Vector,H,J;
    Jdagger::Vector=adjoint.(J),rates=ones(Int,length(J)),
    multithread=false,simplify::Bool=true,order=nothing,
    mix_choice=maximum,iv=SymbolicUtils.Sym{Real}(:t))
    for ind in get_indices(a)
        if ind in get_indices(H)
            error("Index $(ind.name) in operator-vector is already used in H!")
        end
    end
    rhs = Vector{Any}(undef, length(a))
    imH = im*H
    for i=1:length(a)
        try
            rhs_ = commutator(imH,a[i])
            rhs_diss = indexed_master_lindblad(a[i],J,Jdagger,rates)
            indices = get_indices(a[i])
            if length(indices) <= 1
                rhs[i] = rhs_ + rhs_diss
            else
                mapping = Tuple{Index,Index}[]
                for j = 1:length(indices)
                    for k = 1:j
                        if k != j
                            push!(mapping,(indices[k],indices[j]))
                        end
                    end
                end
                rhs[i] = reorder((rhs_+rhs_diss),mapping)
            end
        catch err
            println("could not calculate meanfield-equations for operator $(a[i])")
            rethrow(err)
        end
    end
    vs = map(average, a)
    rhs_avg = map(average, rhs)
    if simplify
        rhs_avg = map(SymbolicUtils.simplify, rhs_avg)
    end
    rhs = map(undo_average, rhs_avg)
    if order !== nothing
        rhs_avg = [cumulant_expansion(r, order; simplify=simplify,
             mix_choice=mix_choice) for r∈rhs_avg]
    end
    eqs_avg = [Symbolics.Equation(l,r) for (l,r)=zip(vs,rhs_avg)]
    eqs = [Symbolics.Equation(l,r) for (l,r)=zip(a,rhs)]
    varmap = make_varmap(vs, iv)
    me = IndexedMeanfieldEquations(eqs_avg,eqs,vs,a,H,J,Jdagger,rates,iv,varmap,order)
    return me
end
```

Code sample 17: *Source code of the implementation of the* `meanfield` *function when called with* `IndexedOperator` *entities.*

The whole source code of the package is available on the official QuantumCumulants.jl GitHub repository [27].

# Chapter 7.

# Bibliography


[1] J.R. Johansson, P.D. Nation, and Franco Nori. Qutip: An open-source python framework for the dynamics of open quantum systems. *Computer Physics Communications*, 183(8):1760–1772, 2012.

[2] Sze M Tan. A computational toolbox for quantum and atomic optics. *Journal of Optics B: Quantum and Semiclassical Optics*, 1(4):424, aug 1999.

[3] Sebastian Krämer, David Plankensteiner, Laurin Ostermann, and Helmut Ritsch. Quantumoptics. jl: A julia framework for simulating open quantum systems. *Computer Physics Communications*, 227:109–116, 2018.

[4] Christoph Hotter, David Plankensteiner, and Helmut Ritsch. Continuous narrowband lasing with coherently driven v-level atoms. *New Journal of Physics*, 22(11):113021, nov 2020.

[5] Christoph Hotter, Laurin Ostermann, and Helmut Ritsch. Cavity sub- and superradiance for transversely driven atomic ensembles. *Phys. Rev. Res.*, 5:013056, Jan 2023.

[6] Oriol Rubies-Bigorda, Stefan Ostermann, and Susanne F. Yelin. Characterizing superradiant dynamics in atomic arrays via a cumulant expansion approach. *Phys. Rev. Res.*, 5:013091, Feb 2023.

[7] Crispin Gardiner and Peter Zoller. *The Quantum World of Ultra-Cold Atoms and Light Book I: Foundations of Quantum Optics*. IMPERIAL COLLEGE PRESS, 2014.

[8] Crispin Gardiner and Peter Zoller. *The Quantum World of Ultra-Cold Atoms and Light Book II: The Physics of Quantum-Optical Devices*. IMPERIAL COLLEGE PRESS, 2015.

[9] Yuan Zhang, Qilong Wu, Shi-Lei Su, Qing Lou, Chongxin Shan, and Klaus Mølmer. Cavity quantum electrodynamics effects with nitrogen vacancy center spins coupled to room temperature microwave resonators. *Phys. Rev. Lett.*, 128:253601, Jun 2022.

[10] Christoph Hotter, David Plankensteiner, Georgy Kazakov, and Helmut Ritsch. Continuous multi-step pumping of the optical clock transition in alkaline-earth atoms with minimal perturbation. *Opt. Express*, 30(4):5553–5568, Feb 2022.

[11] Kathrin Henschel, Johannes Majer, Jörg Schmiedmayer, and Helmut Ritsch. Cavity qed with an ultracold ensemble on a chip: Prospects for strong magnetic coupling at finite temperatures. *Phys. Rev. A*, 82:033810, Sep 2010.

[12] Thomas Maier, Sebastian Kraemer, Laurin Ostermann, and Helmut Ritsch. A superradiant clock laser on a magic wavelength optical lattice. *Opt. Express*, 22(11):13269–13279, Jun 2014.

[13] Serge Haroche and Jean Michel Raimond. *Exploring the Quantum: Atoms, Cavities, and Photons*. Oxford Univ. Press, Oxford, 2006.

[14] E.T. Jaynes and F.W. Cummings. Comparison of quantum and semiclassical radiation theories with application to the beam maser. *Proceedings of the IEEE*, 51(1):89–109, 1963.

[15] Michael Tavis and Frederick W. Cummings. Exact solution for an $n$-molecule—radiation-field hamiltonian. *Phys. Rev.*, 170:379–384, Jun 1968.

[16] David Plankensteiner, Christoph Hotter, and Helmut Ritsch. QuantumCumulants.jl: A Julia framework for generalized mean-field equations in open quantum systems. *Quantum*, 6:617, January 2022.

[17] Ryogo Kubo. Generalized cumulant expansion method. *Journal of the Physical Society of Japan*, 17(7):1100–1120, 1962.

[18] R. H. Lehmberg. Radiation from an $n$-atom system. i. general formalism. *Phys. Rev. A*, 2:883–888, Sep 1970.

[19] Christopher Rackauckas and Qing Nie. Differentialequations.jl–a performant and feature-rich ecosystem for solving differential equations in julia. *Journal of Open Research Software*, 5(1), 2017.

[20] Jeff Bezanson, Alan Edelman, Stefan Karpinski, and Viral B. Shah. Julia: A fresh approach to numerical computing. *SIAM Review*, 59(1):65–98, 2017.

[21] Jeff Bezanson, Stefan Karpinski, Viral B. Shah, and Alan Edelman. Julia: A fast dynamic language for technical computing, 2012.

[22] Shashi Gowda, Yingbo Ma, Alessandro Cheli, Maja Gwóźzdź, Viral B. Shah, Alan Edelman, and Christopher Rackauckas. High-performance symbolic-numerics via multiple dispatch. *ACM Commun. Comput. Algebra*, 55(3):92–96, jan 2022.

[23] Documentary of the Julia language https://docs.julialang.org/en/v1/.

[24] David Plankensteiner, Christian Sommer, Helmut Ritsch, and Claudiu Genes. Cavity antiresonance spectroscopy of dipole coupled subradiant arrays. *Phys. Rev. Lett.*, 119:093601, Aug 2017.

[25] D. Meiser, Jun Ye, D. R. Carlson, and M. J. Holland. Prospects for a millihertz-linewidth laser. *Phys. Rev. Lett.*, 102:163601, Apr 2009.

[26] Kamanasish Debnath, Yuan Zhang, and Klaus Mølmer. Lasing in the superradiant crossover regime. *Phys. Rev. A*, 98:063837, Dec 2018.

[27] GitHub repository of QuantumCumulants.jl https://github.com/qojulia/QuantumCumulants.jl.

# Eidesstattliche Erklärung

Ich erkläre hiermit an Eides statt durch meine eigenhändige Unterschrift, dass ich die vorliegende Arbeit selbständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel verwendet habe. Alle Stellen, die wörtlich oder inhaltlich den angegebenen Quellen entnommen wurden, sind als solche kenntlich gemacht.

Die vorliegende Arbeit wurde bisher in gleicher oder ähnlicher Form noch nicht als Magister- /Master-/Diplomarbeit/Dissertation eingereicht.

| | |
|---|---|
| Datum | Unterschrift |