pyBoLaNO: A Python symbolic package for normal ordering involving bosonic ladder operators

Hendry M. Lim[a,b,*], Donny Dwiputra[c,a], M. Shoufie Ukhtary[a], Ahmad R. T. Nugraha[a,d]

[a] *Research Center for Quantum Physics, National Research and Innovation Agency (BRIN), South Tangerang 15314, Indonesia*
[b] *Department of Physics, Faculty of Mathematics and Natural Sciences, Universitas Indonesia, Depok 16424, Indonesia*
[c] *Asia Pacific Center for Theoretical Physics, Pohang 37673, Korea*
[d] *Department of Engineering Physics, Telkom University, Bandung 40257, Indonesia*

Abstract

We present pyBoLaNO, a Python symbolic package based on SymPy to quickly normal-order any polynomial in bosonic ladder operators regarding the canonical commutation relations, using Blasiak's formulae. By extension, this package offers the normal ordering of commutators of any two polynomials in bosonic ladder operators and the evaluation of the normal-ordered expectation value evolution in the Lindblad master equation framework for open quantum systems. The package supports multipartite descriptions and multiprocessing. We describe the package's workflow, show examples of use, and discuss its computational performance. All codes and examples are available on our GitHub repository.

Keywords: bosonic ladder operators, Blasiak's formulae, normal ordering, commutator, Lindblad master equation

PROGRAM SUMMARY

Program Title: pyBoLaNO
Developer's repository link: https://github.com/hendry24/pyBoLaNO
Licensing provisions: MIT License
Programming language: Python

*Corresponding author.
E-mail address: hendry01@ui.ac.id

Nature of problgem: Normal ordering involving bosonic ladder operators regarding the canonical commutation relations.
Solution method: Blasiak's formulae for the normal ordering of an arbitrary monomial in bosonic ladder operators regarding the canonical commutation relations. Symbolic programming is fully provided by `SymPy`.

1. Introduction

Ladder operators arise in the study of the quantum simple harmonic oscillator. They consist of the annihilation/lowering operator $\hat{b}$ and its Hermitian conjugate, the creation/raising operator $\hat{b}^\dagger$. They allow for the algebraic treatment of the problem, giving a simple relation between the system's eigenstates. The eigenstate $|n\rangle$ corresponding to the nth energy level E_n is obtained by applying $\hat{b}^\dagger$ to the vacuum state $|0\rangle$ a total of n times, $|n\rangle \propto \hat{b}^{\dagger n}|0\rangle$ [1, 2]. Taking into account quantum statistics, the ladder operators are defined differently for bosons and fermions—usually denoted $\hat{a}$ and $\hat{a}^\dagger$ in the latter case. Furthermore, the eigenvectors of the bosonic annihilation operator are quantum states that most closely resemble the classical harmonic oscillator, that is, the coherent states [3]. The formulation is far-reaching in quantum mechanics. The ladder operators form the basis for quantum field theory, in which a particle is considered as the excitation of the underlying quantum field [2]. They can also be found in the theoretical description of physical systems in atomic, molecular, and optical physics [3] that extends beyond simple harmonic oscillators.

We are interested in obtaining the normal-ordered equivalent of a polynomial in the ladder operators—a process we shall call "normal ordering" herein. This is useful in quantum optics as physically relevant expectation values involve normal-ordered monomials in the ladder operators. Matrix elements in the coherent-state basis can be straightforwardly evaluated for normal-ordered monomials. Furthermore, given the system's Glauber-Sudarshan P function, the normal-ordered form is convenient via the optical equivalence theorem [3, 4]. It is also quite common in the literature to present the evolution equation for a given expectation value in a normal-ordered form [5–19]. As the system description becomes more complex, the algebraic manipulation of the dynamical equations becomes more tedious and prone to errors, making automation desirable.

The `SymPy` package [20] provides a symbolic computation framework in the `Python` programming language. At the time of writing this work,

SymPy is on release ver. 1.13.3 which supports the ladder operators via the sympy.physics.secondquant and sympy.physics.quantum submodules. In particular, normal ordering is implemented as the function normal_ordered_form. *Unfortunately*, the algorithm implemented in the package quickly slows down as we increase the complexity of the expression to normal-order.

Motivated by a more efficient method and the need for a convenient tool to obtain the expectation value evolution given the equation of motion, we develop the Python package pyBoLaNO that offers fast symbolic normal ordering of expressions involving bosonic ladder operators, which extends to fast normal ordering of commutators and expectation value evolution from the Lindblad master equation. The package is fully built on the core functionalities of Sympy and supports multipartite descriptions as well as multiprocessing for each additive term in the input(s).

The remainder of this paper is structured as follows. In Section 2, we elaborate on the theoretical basis for the features presented in this package. Section 3 describes the package's functionalities. The package usage is shown through some selected examples in Section 4. The computational performance of the package is discussed in Section 5. Finally, Section 6 concludes this paper.

2. Theoretical Considerations

Here, we present a brief overview of the formulations of the ladder operators available in quantum mechanics textbooks. The readers interested in only the essentials of the package may skip to Section 2.1.

We consider the classical Hamiltonian of the simple harmonic oscillator under Hooke's law [1]:

$$H = \frac{p^2}{2m} + \frac{m\omega_0^2 x^2}{2}, \tag{1}$$

where m is the oscillator's mass and ω_0 is its natural angular frequency. The quantization of this Hamiltonian is achieved by replacing the position x and momentum p with the corresponding Hilbert space operators. We obtain

$$\hat{H} = \frac{\hat{p}^2}{2m} + \frac{m\omega_0^2 \hat{x}^2}{2}. \tag{2}$$

The time-independent Schrödinger equation (TISE) is given by

$$\hat{H}\psi = E\psi, \tag{3}$$

where ψ is the system's wave function and E is its energy. In the algebraic treatment of the problem (the other being the analytic method involving Hermite polynomials), we define the operators

$$\hat{b} = \frac{(m\omega_0 \hat{x} + i\hat{p})}{\sqrt{2\hbar m\omega_0}}, \tag{4a}$$

$$\hat{b}^\dagger = \frac{(m\omega_0 \hat{x} - i\hat{p})}{\sqrt{2\hbar m\omega_0}}, \tag{4b}$$

satisfying the commutation relations

$$\left[\hat{b}, \hat{b}^\dagger\right] = 1, \tag{5}$$

where $\left[\hat{A}, \hat{B}\right] = \hat{A}\hat{B} - \hat{B}\hat{A}$. The Hamiltonian becomes

$$\hat{H} = \hbar\omega_0 \left(\hat{b}^\dagger \hat{b} + \frac{1}{2}\right). \tag{6}$$

Considering the time-independent Schrödinger equation (TISE),

$$\hat{H}\left|\psi\right\rangle = E\left|\psi\right\rangle, \tag{7}$$

where ψ is the wave function and E is the energy, it can be shown that

$$\hat{H}\left(\hat{b}^\dagger\left|\psi\right\rangle\right) = (E + \hbar\omega_0)\left(\hat{b}^\dagger\left|\psi\right\rangle\right), \tag{8a}$$

$$\hat{H}\left(\hat{b}\left|\psi\right\rangle\right) = (E - \hbar\omega_0)\left(\hat{b}\left|\psi\right\rangle\right), \tag{8b}$$

meaning that operating on the system $\left|\psi\right\rangle$ with $\hat{b}^\dagger$ raises its energy by a quantum of $\hbar\omega_0$, while operating with $\hat{b}$ lowers its energy by $\hbar\omega_0$. This is akin to moving up and down the rung of a ladder, hence the name "ladder operators". By definition, the lowest energy state is called the vacuum state $\left|0\right\rangle$, which satisfies $\hat{H}\left(\hat{b}\left|0\right\rangle\right) = 0$. The nth eigenstate is given by

$$\left|n\right\rangle = \frac{1}{\sqrt{n!}}\hat{b}^{\dagger n}\left|0\right\rangle. \tag{9}$$

The formulation of ladder operators is not limited to a mechanical oscillator. In general, it can be formulated from any system described by a

Hamiltonian resembling Eq. (2) where the *canonical* position q and momentum p take the roles of the real position and momentum considered above (see, for example Chap. 2 of Ref. [3] for the treatment of electromagnetic waves). In some cases, they are called the "raising" and "lowering" operators. In some others, they are called the "annihilation" and "creation" operators. In this paper, we shall use the term "ladder operators" as a reference to both operators, while the "annihilation" and "creation" operators refer to $\hat{b}$ and $\hat{b}^\dagger$, respectively.

The formulation can also be applied to an ensemble of many simple harmonic oscillators. In this case, we have pairs of ladder operators $\left(\hat{b}_j, \hat{b}_j^\dagger\right)$, each associated with a single uncoupled oscillator in the ensemble indexed by j. They satisfy the commutation relations

$$\left[\hat{b}_j, \hat{b}_k\right] = \left[\hat{b}_j^\dagger, \hat{b}_k^\dagger\right] = 0, \tag{10a}$$

$$\left[\hat{b}_j, \hat{b}_k^\dagger\right] = \delta_{jk}, \tag{10b}$$

where δ_{jk} is the Kronecker delta.

The indices also appear in the occupation number representation (see, for example, Chap. 3 of Ref. [2]). Given $|n_0, n_1, n_2, \ldots\rangle$, where n_0 denotes the number of particles in the quantum state ψ_0, n_1 is the number of particles in the state ψ_1, etc. [2], to decrease and increase the number of particles, we use the ladder operators with the corresponding index: $\hat{b}_j$ and $\hat{b}_j^\dagger$, respectively. We consider $|0, 0\rangle$ and try to increase each count by one. This can be done in two ways:

$$\hat{b}_1^\dagger \hat{b}_0^\dagger |0, 0\rangle \propto |1, 1\rangle \tag{11}$$

or

$$\hat{b}_0^\dagger \hat{b}_1^\dagger |0, 0\rangle \propto |1, 1\rangle . \tag{12}$$

Since we end up in the same state, we must have

$$\hat{b}_0^\dagger \hat{b}_1^\dagger = \lambda \hat{b}_1^\dagger \hat{b}_0^\dagger \tag{13}$$

where λ is a scalar. By quantum statistics, in our three-dimensional world we have $\lambda = \pm 1$. We can similarly consider other processes, such as adding one particle in the state ψ_0 and removing one in the state ψ_1. In general, the $\lambda = 1$ case corresponds to bosons and gives the commutation relations

shown in Eqs. (10a) and (10b). Meanwhile, the $\lambda = -1$ case corresponds to fermions and gives the (anti)commutation relations

$$\{\hat{a}_j, \hat{a}_k\} = \left\{\hat{a}_j^\dagger, \hat{a}_k^\dagger\right\} = 0, \tag{14a}$$

$$\left\{\hat{a}_j, \hat{a}_k^\dagger\right\} = \delta_{jk}, \tag{14b}$$

where we have used a instead of b for fermionic ladder operators. In the following, we focus only on the bosonic ladder operators and drop the adjective "bosonic" for brevity.

2.1. Normal ordering and Blasiak's formulae

Let us consider a monomial in the ladder operators, such as $\hat{b}^\dagger\hat{b}\hat{b}^\dagger$ or $\hat{b}^2\hat{b}^{\dagger 2}$. A ladder monomial is said to be *normal-ordered* if all creation operators are positioned to the left of all annihilation operators, i.e., a monomial of the form $\hat{b}^{\dagger p}\hat{b}^q; p, q \in \mathbb{Z}$. Conventionally, the term "normal ordering" refers to Wick ordering, where a given monomial is replaced with an *inequivalent* normal-ordered monomial containing the same number of creation and annihilation operators. It is represented by the "double dot enclosure", e.g. $:\hat{b}^\dagger\hat{b}\hat{b}^\dagger: = \hat{b}^{\dagger 2}\hat{b}$ and $:\hat{b}^2\hat{b}^{\dagger 2}: = \hat{b}^{\dagger 2}\hat{b}^2$. In other words, the monomial has been reordered with $\left[\hat{b}_j, \hat{b}_k^\dagger\right] = 0$ (instead of δ_{jk}). In quantum field theory, Wick ordering is useful to avoid infinite self-energy and to develop the Wick's theorem (see, for example, Chaps. 4 and 18 in Ref. [2]). In quantum optics, it is useful to simplify equations in quadrature squeezing (see, for example, Chapter 7 in Ref. [3]).

Throughout this paper, we define "normal ordering" as rewriting the monomial into an *equivalent* expression using the commutation relations. The normal ordering of a ladder monomial $\hat{X}$ is denoted $\mathcal{N}\left(\hat{X}\right)$ herein, for example,

$$\mathcal{N}\left(\hat{b}^\dagger\hat{b}\hat{b}^\dagger\right) = \hat{b}^\dagger + \hat{b}^{\dagger 2}\hat{b} \tag{15}$$

This is useful in quantum optics when dealing with coherent states of the harmonic oscillator, i.e. the state $|\beta\rangle$ satisfying $\hat{b}\,|\beta\rangle = \beta\,|\beta\rangle\,, \beta \in \mathbb{Z}$, which implies that the matrix element $\left\langle\beta\middle|\hat{X}\middle|\beta\right\rangle$ of some monomial $\hat{X}$ in the coherent state basis can be straightforwardly evaluated if $\hat{X}$ is normal-ordered. It means if $g_{\mathcal{N}}\left(\hat{b}, \hat{b}^\dagger\right)$ is a normal-ordered polynomial in the ladder operators,

then

$$\left\langle \beta \middle| g_{\mathcal{N}}\left(\hat{b},\hat{b}^{\dagger}\right) \middle| \beta \right\rangle = g_{\mathcal{N}}\left(\beta,\beta^{*}\right). \tag{16}$$

One interesting property is the optical equivalence theorem. Let $P(\beta)$ be the Glauber-Sudarshan P function, one of the possible phase-space representations of a quantum system; then,

$$\left\langle g_{\mathcal{N}}\left(\hat{b},\hat{b}^{\dagger}\right)\right\rangle = \int \mathrm{d}^{2}\beta \; P\left(\beta\right) g_{\mathcal{N}}\left(\beta,\beta^{*}\right), \tag{17}$$

which means that we can conveniently obtain the expectation value of any ladder polynomial by normal-ordering the operator and replacing $\left(\hat{b},\hat{b}^{\dagger}\right)$ by (β,β^{*}), turning the expectation value integral into a simpler weighted average(see for example, Chap. 3 of Ref. [3]).

A useful result is given by Blasiak [21, 22] for the normal ordering of a ladder monomial. Let us now define

$$\hat{X} = \hat{b}^{\dagger r_M}\hat{b}^{s_M}\ldots\hat{b}^{\dagger r_2}\hat{b}^{s_2}\hat{b}^{\dagger r_1}\hat{b}^{s_1}, \tag{18}$$

$\boldsymbol{r} = (r_1, r_2, \ldots, r_M)$, and $\boldsymbol{s} = (s_1, s_2, \ldots, s_M)$. Also, let

$$d_l = \sum_{m=1}^{l}\left(r_m - s_m\right) \tag{19}$$

be the lth excess of creation operators in the monomial. Given the generalized Stirling numbers,

$$S_{\boldsymbol{r},\boldsymbol{s}}(k) = \frac{1}{k!}\sum_{j=0}^{k}\binom{k}{j}(-1)^{k-j}\prod_{m=1}^{M}\left(d_{m-1}+j\right)_{s_m}, \tag{20}$$

where $\binom{a}{b} = \frac{a!}{b!(a-b)!}$ is the binomial coefficient and $(m)_n = \frac{m!}{(m-n)!}$ is the falling factorial, we can obtain the normal-ordered expression for $\hat{X}$ as

$$\mathcal{N}\left(\hat{X}\right) = \begin{cases} \hat{b}^{\dagger d_M}\displaystyle\sum_{k=s_1}^{s_1+s_2+\cdots+s_M} S_{\boldsymbol{r},\boldsymbol{s}}(k)\hat{b}^{\dagger k}\hat{b}^{k}, & d_M \geq 0 \\ \displaystyle\sum_{k=r_M}^{r_1+r_2+\cdots+r_M} S_{\overline{\boldsymbol{s}},\overline{\boldsymbol{r}}}(k)\hat{b}^{\dagger k}\hat{b}^{k}\;\hat{b}^{-d_M}, & d_M < 0 \end{cases} \tag{21}$$

where $\overline{\boldsymbol{r}} = (r_M, \ldots, r_2, r_1)$ and $\overline{\boldsymbol{s}} = (s_M, \ldots, s_2, s_1)$.

2.2. The Lindblad master equation

Normal ordering is practically used in open quantum systems to obtain expressions that are convenient to interpret and use in other calculations, e.g., the Q-parameter for photon statistics [3]. A widely used formalism is the Lindblad master equation [23, 24]. For a system described by the density matrix ρ, the evolution of the system is given by

$$\frac{\mathrm{d}\rho}{\mathrm{d}t} = -\frac{i}{\hbar}\left[\hat{H},\rho\right] + \sum_j \gamma_j \mathcal{D}\left(\hat{O}_j,\hat{P}_j\right)[\rho]\,. \tag{22}$$

The Hamiltonian $\hat{H}$ describes the closed system dynamics, while the Lindblad dissipators

$$\mathcal{D}\left(\hat{O}_j,\hat{P}_j\right)[\rho] = \hat{O}_j\rho\hat{P}_j^\dagger - \frac{1}{2}\left\{\hat{P}_j^\dagger\hat{O}_j,\rho\right\} \tag{23}$$

describe the open system dynamics. Each dissipator is defined by the operators $\hat{O}_j, \hat{P}_j$, which describe the open system process. The multiplying scalar $\gamma_j \geq 0$ can be interpreted as the process rate.

Theoretical treatment of open quantum systems often deals with calculating the evolution of expectation values for some physical quantities. Let a quantity A be represented by the operator $\hat{A}$. Given the density matrix ρ, the expectation value of A is given by $\mathrm{tr}\left(\rho\hat{A}\right)$. Multiplying Eq. (22) by $\hat{A}$ (from both sides of the equation), we have

$$\begin{aligned}\frac{\mathrm{d}\left\langle A\right\rangle}{\mathrm{d}t} =& -\frac{i}{\hbar}\mathrm{tr}\left(\left[\hat{H},\rho\right]\hat{A}\right)\\ &+\sum_j \gamma_j \mathrm{tr}\left(\mathcal{D}\left(\hat{O}_j,\hat{P}_j\right)[\rho]\,\hat{A}\right)\end{aligned} \tag{24}$$

We call the trace containing $\hat{H}$ the "Hamiltonian trace", and those containing $\mathcal{D}$ the "dissipator traces". For arbitrary Hamiltonian and dissipator operators, they are given by

$$\mathrm{tr}\left(\left[\hat{H},\rho\right]\hat{A}\right) = \left\langle\left[\hat{A},\hat{H}\right]\right\rangle \tag{25}$$

and

$$\begin{aligned}\mathrm{tr}\left(\mathcal{D}\left(\hat{O}_j,\hat{P}_j\right)[\rho]\,\hat{A}\right) =& \frac{1}{2}\left\langle\left[\hat{P}_j^\dagger,\hat{A}\right]\hat{O}_j\right\rangle\\ &+\frac{1}{2}\left\langle\hat{P}_j^\dagger\left[\hat{A},\hat{O}_j\right]\right\rangle.\end{aligned} \tag{26}$$

Table 1: SymPy objects relevant to pyBoLaNO . SymPy is ver. 1.13.3 .

SymPy object	Description
Number	Atomic expression for numbers (Integer , Float , Rational). The args attribute is an empty tuple.
Symbol	Symbol expression. Useful for variables. The args attribute is a tuple containing the string input to the object's constructor.
Add	The sum object. The args attribute is a tuple of its summands.
Mul	The product object. The args attribute is a tuple of its factors.
Pow	The exponentiation object. The args attribute is a tuple (b,e) of its base and exponent.

These expressions become more cumbersome and error-prone to normal-order as the operators involved become more complex. This motivates us to automate the process with a computer program, pyBoLaNO, that we develop in this work.

3. Package Description

The core workflow of pyBoLaNO is shown in Fig. 1. We start with a brief overview of the SymPy objects relevant to our package. Then, we describe the workflow of the package's main functions: ops , normal_ordering / NO , NO_commutator , and LME_expval_evo ; as well as the multiprocessing configuration dictionary: mp_config .

3.1. Relevant objects in SymPy

All algebraic expressions in SymPy are based on the Expr or AtomicExpr class. The former is used for expressions with arguments (composite expressions), such as Symbol . The latter is used for expressions without arguments (atomic expressions), such as Number . The tuple of arguments of an expression object can be accessed from its args attribute. For AtomicExpr , an empty tuple is returned.

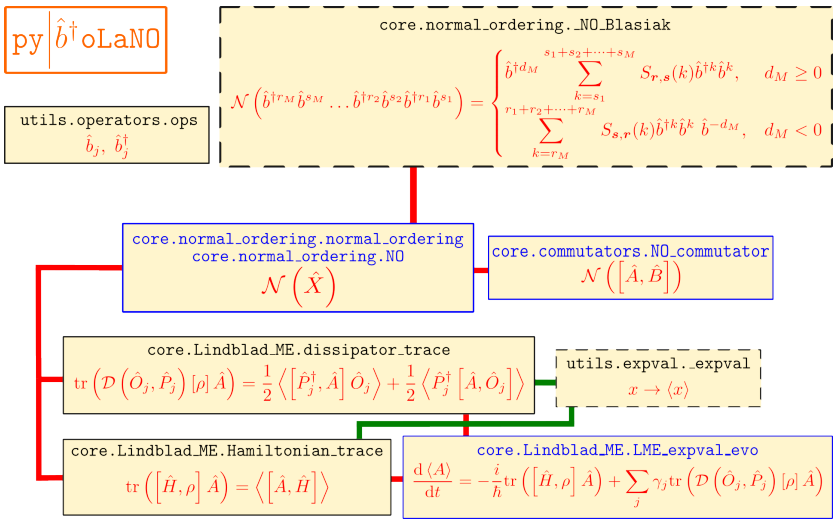

Figure 1: Core functionalities of `pyBoLaNO` . The main functions are shown in blue, while the core utility functions are colored black. Meanwhile, the purple-colored `mp_config` is a `Python` variable. Red connectors show symbolic workflows, while green connectors show visual workflows. Dashed rectangles indicate functionalities that are not accessible by the user.

Table 1 briefly describes `SymPy` objects relevant to our package. Each mathematical operation returns the corresponding operation class: `Add` for summation, `Mul` for multiplication, and `Pow` for exponentiation, to name a few. These are all composite objects and may have each other as arguments. `Add` and `Mul` may have any number of arguments. Meanwhile, the first argument of `Pow` is the base and the second argument is the exponent. For example, given that `x=Symbol("x")` , then `x**2+2*x` is constructed as `Add(Pow(Symbol("x"),Integer(2)),Mul(Integer(2), Symbol("x")))` . This representation can be accessed using the `srepr` function. We can also check whether a given expression is contained using the `has` method.

3.2. Constructing the ladder operators

The ladder operator objects are implemented as `BosonicAnnihilationOp` and `BosonicCreationOp` . Instead of initializing both objects, the package provides a convenient call through the function `ops` . By calling

```
ops(k)
```

we can initialize both annihilation and creation operators with the same subscript. The subscript `k` is optional and can be `Symbol` or any `Python` object convertible to a string. This function preprocesses `k` into `Symbol` to be used as the subscript of ladder operators.

3.3. Normal ordering

The normal ordering of a polynomial `q` in ladder operators is done by the `normal_ordering` function, with syntax

```
normal_ordering(q)
```

We also offer a shorthand

```
NO(q)
```

for the same purpose.

Blasiak's formulae, given by Eq. (21), are implemented in the `_NO_Blasiak` function, which evaluates the normal ordering of the given monomial in the ladder operators. Extra algorithms are executed to handle multipartite descriptions. The general workflow is described below.

1. Make a list of summands of `q`.
2. Make each summand of `q` into a list of factors, separated by the ladder operator subscripts. Scalars are put in a separate group.
3. Normal-order each factor using `_NO_Blasiak`.
4. Multiply the sums obtained by normal-ordering each factor to give the almost-normal-ordered summand of `q`.
5. Sum all the almost-normal-ordered summands of `q` to give the almost-normal-ordered `q`.
6. Due to how `SymPy` handles noncommuting objects, the `Mul` of factors obtained in step 4 are yet to be normal-ordered. Instead, each term goes like $\ldots \hat{b}_j^{\dagger a} \hat{b}_j^b \hat{b}_{k\neq j}^{\dagger c} \hat{b}_{k\neq j}^d \ldots$. Do the final sorting as follows for each summand:
   (a) Collect scalars into one list.
   (b) Collect creation operators as dictionary entries whose keys are their subscripts.

(c) Collect annihilation operators similarly.
(d) Sort both dictionaries by key, then return a list of their values.
(e) Multiply all items in each list to give a `Mul` of scalars, creation operators, and annihilation operators.
(f) Multiply the three resulting `Mul` objects to give the fully normal-ordered summand.

Sum all summands and return the resulting `Add`.

3.4. Normal-ordered commutators

The normal ordering of a commutator between two polynomials `A` and `B` in ladder operators can be evaluated using the `do_commutator` function. The syntax is given by

```
do_commutator(A, B)
```

This function simply calls `normal_ordering(A*B-B*A)`.

3.5. Expectation value evolution in the Lindblad master equation framework

Equation (24) can be evaluated by calling `LME_expval_evo`. The syntax is

```
LME_expval_evo(H, D, A, hbar_is_one)
```

Here, `H` is the Hamiltonian of the system, written as a `SymPy` expression. Meanwhile, `D` is a list that specifies the dynamics of the open system. Each element in `D` is a list `[gamma_j, O_j, P_j]` containing the process rate γ_j and the operators $\hat{O}_j, \hat{P}_j$ defining the dissipator. If $\hat{O}_j = \hat{P}_j$, the user may omit the third element from the list. If no dissipation is present, the user can input an empty list, `D=[]`. The operator corresponding to the quantity whose expectation value evolution is calculated is input to `A`. Finally, `hbar_is_one` is a Boolean to set $\hbar = 1$. This option is `True` by default. We note that `Symbol("hbar")` is used instead of `sympy.physics.quantum.hbar` to ensure that it behaves well with the package functionalities. The following happens when this function is called.

1. Call `Hamiltonian_trace(H, A)` to calculate the Hamiltonian trace with Eq. (25) in normal-ordered form. Add the result to the output.

2. For each entry in `D`, call `dissipator_trace(O_j,P_j,A)` to calculate the dissipator trace given by Eq. (26) in normal-ordered form. Multiply the result by the corresponding process rate `gamma_j` and add to the output.
3. Return the output `Add`.

The output is generally a sum that contains `_expval` objects. The `_expval` class wraps the output operators inside bra-kets. The class inherits `SymPy`'s `Symbol` class and is used only for result visualization.

3.6. Mutiprocessing configurations

Multiprocessing is used in `normal_ordering` to handle each summand of the input. The user may control the multiprocessing behavior by modifying the `mp_config` variable after importing the package. The syntax is

```
mp_config["enable"] = True
mp_config["num_cpus"] = os.cpu_count()
mp_config["min_num_args"] = 2
```

where the values assigned above are the default values of the package. Here, `"enable"` specifies whether multiprocessing is enabled, `"num_cpus"` specifies the number of CPU threads to use, and `"min_num_args"` specifies the minimum number of summands mentioned above (arguments of the `Add` object). Any integer less than `2` will cause the package to set `mp_config["min_num_args"]=2`.

4. Some Use Cases of the Package

The examples provided in this section are available in the package tutorial on our GitHub repository, which uses `Jupyter Notebook` to render the LaTeX output. The following snippets show an exemplary setup of the package.

```
import pybolano as bl
bl.mpconfig["num_cpus"] = 4
```

```
import sympy as sm
latex = sm.latex # to print as LaTeX code.
```

```
b, bd = bl.ops()
b_1, bd_1 = bl.ops(1)
b_2, bd_2 = bl.ops(sm.Symbol("2"))

print(latex(
b_1
))

Out: b_{1}
```

4.1. `normal_ordering`

```
print(latex(
bl.normal_ordering(b * bd * b)
))

Out: b_{} + {b^\dagger_{}} b_{}^{2}
```

Multipartite input:

```
print(latex(
bl.normal_ordering(b_2 * b_1 * bd_2**2 * bd_1)
))

Out: 2 {b^\dagger_{1}} {b^\dagger_{2}} b_{1} + {b^\dagger_{1}}
↪  {b^\dagger_{2}}^{2} b_{1} b_{2} + 2 {b^\dagger_{2}} +
↪  {b^\dagger_{2}}^{2} b_{2}
```

Polynomial input:

```
print(latex(
bl.normal_ordering(b_1*bd_2 + 5*b_2**2*bd_1*b_1 + b_2)
))

Out: b_{2} + 5 {b^\dagger_{1}} b_{1} b_{2}^{2} +
↪  {b^\dagger_{2}} b_{1}
```

Input with symbols:

```
x = sm.Symbol("x")

print(latex(
bl.normal_ordering(x * b_1 * x**2 * bd_1**2)
))

Out: 2 x^{3} {b^\dagger_{1}} + x^{3} {b^\dagger_{1}}^{2} b_{1}
```

4.2. `NO_commutator`

```
A = bd*b
B = b

print(latex(
bl.NO_commutator(A, B)
))

Out: - b_{}
```

Multipartite input:

```
A = bd_1*bd_2
B = b_1*b_2

print(latex(
bl.NO_commutator(A, B)
))

Out: -1 - {b^\dagger_{1}} b_{1} - {b^\dagger_{2}} b_{2}
```

Polynomial input:

```
A = b_1 + 2*b_2**2
B = bd_1**3 + 2*bd_2*b_2
```

```
print(latex(
bl.NO_commutator(A, B)
))

Out: 8 b_{2}^{2} + 3 {b^\dagger_{1}}^{2}
```

Input with symbols:

```
x = sm.Symbol("x")
A = x*b_1
B = x**(0.5)*bd_1*b

print(latex(
bl.NO_commutator(A, B)
))

Out: x^{1.5} b_{}
```

4.3. `LME_expval_evo`

Now we consider some evolution equations in the literature as a way to validate this package, highlighting its potential usage for describing various quantum systems. The following examples also illustrate problems where it is desirable to compute the evolution of the expectation values of some observables for which `pyBoLaNO` may be useful.

4.3.1. The quantum simple harmonic oscillator

It is well-known that the evolution of the expected value of the annihilation operator $\hat{b}$ in a simple harmonic Hamiltonian $\hat{H} = \hbar\omega_0 \left(\hat{b}^\dagger \hat{b} + 1/2\right)$ is given by

$$\frac{\mathrm{d}\left\langle \hat{b} \right\rangle}{\mathrm{d}t} = -i\hbar\omega_0 \left\langle \hat{b} \right\rangle \tag{27}$$

Here is how it is obtained with `pyBoLaNO` :

```
hbar, omega_0 = sm.symbols(r"hbar omega_0")
b, bd = bl.ops()
```

```
H = hbar*omega_0*bd*b
D = []
A = b

print(latex(
bl.LME_expval_evo(H, D, A, hbar_is_one=False)
))

Out: \frac{d}{d t} {\left\langle b_{} \right\rangle} = - i
↪  \omega_{0} {\left\langle b_{} \right\rangle}
```

Additionally, we can show that the energy is conserved, i.e. that

$$\frac{\mathrm{d}\left\langle \hat{b}^\dagger \hat{b} \right\rangle}{\mathrm{d}t} = 0 \tag{28}$$

Assuming the variables from the previous code block, we have

```
A = bd*b

print(latex(
bl.LME_expval_evo(H, D, A)
))

Out: \frac{d}{d t} {\left\langle {b^\dagger_{}} b_{}
↪  \right\rangle} = 0
```

4.3.2. The quantum Rayleigh oscillator

A limit cycle can be defined as a stable oscillation to which the system is always attracted [25]. This is a consequence of nonlinearity and can be observed in nonlinear oscillators. The Rayleigh oscillator is one such exemplary model. We consider the evolution of the expected phase point $\left\langle \hat{b} \right\rangle$ for a quantum Rayleigh oscillator, a nonlinear quantum oscillator exhibiting a quantum limit cycle. From Ref. [5], the equation of motion is specified by

(with $\hbar = 1$ and $\hat{b} = [\hat{x} + i\hat{p}]/2$, cf. Eqs. [14–16] of Ref. [5]):

$$\begin{aligned}\hat{H} = \omega_0 \hat{b}^\dagger \hat{b} + i\frac{\mu}{12}\left(\hat{b}^\dagger \hat{b}^3 - \hat{b}^{\dagger 3}\hat{b}\right) \\ + i\frac{\mu}{24}\left(\hat{b}^4 - \hat{b}^{\dagger 4}\right) - i\frac{\mu\left(q_0^2 - 1\right)}{4}\left(\hat{b}^2 - \hat{b}^{\dagger 2}\right)\end{aligned} \tag{29}$$

and

$$\gamma_1 = \mu\left(q_0^2 - 1\right), \quad \hat{O}_1 = \hat{P}_1 = \hat{b}^\dagger \tag{30a}$$

$$\gamma_2 = \frac{3\mu}{4}, \quad \hat{O}_2 = \hat{P}_2 = \hat{b}^2 \tag{30b}$$

$$\gamma_3 = \mu, \quad \hat{O}_3 = \hat{P}_3 = \hat{b}^\dagger \hat{b} - \frac{\hat{b}^{\dagger 2}}{2} \tag{30c}$$

where μ is the nonlinearity parameter and q_0 is one of the Rayleigh parameters. The evolution of $\left\langle \hat{b} \right\rangle$ is governed by (cf. Eqs. [13, 17–25] of Ref. [5])

$$\begin{aligned}\frac{\mathrm{d}\left\langle \hat{b} \right\rangle}{\mathrm{d}t} = -i\omega_0 \left\langle \hat{b} \right\rangle + \frac{\mu}{2}\left(q_0^2 - 1\right)\left[\left\langle \hat{b} \right\rangle + \left\langle \hat{b}^\dagger \right\rangle\right] \\ - \frac{\mu}{6}\left[\left\langle \hat{b}^3 \right\rangle + \left\langle \hat{b}^{\dagger 3} \right\rangle\right] - \frac{\mu}{2}\left[\left\langle \hat{b}^\dagger \hat{b}^2 \right\rangle + \left\langle \hat{b}^{\dagger 2}\hat{b} \right\rangle\right]\end{aligned} \tag{31}$$

Here is how Eq. (31) can be obtained with `pyBoLaNO` :

```
omega_0, mu, q_0 = sm.symbols(r"omega_0 mu q_0")
b, bd = bl.ops()

H = omega_0*bd*b \
    + sm.I*mu/12 * (bd*b**3 - bd**3*b) \
    + sm.I*mu/24 * (b**4-bd**4) \
    - sm.I*mu*(q_0**2-1)/4 * (b**2-bd**2)

D = [[mu*(q_0**2-1), bd],
     [3*mu/4, b**2],
     [mu, bd*b-bd**2/2]]

A = b
```

```
print(latex(
bl.LME_expval_evo(H, D, A)
))

Out: \frac{d}{d t} {\left\langle b_{} \right\rangle} =
↪  \frac{\mu q_{0}^{2} {\left\langle b_{} \right\rangle}}{2}
↪  + \frac{\mu q_{0}^{2} {\left\langle {b^\dagger_{}}
↪  \right\rangle}}{2} - \frac{\mu {\left\langle b_{}
↪  \right\rangle}}{2} - \frac{\mu {\left\langle b_{}^{3}
↪  \right\rangle}}{6} - \frac{\mu {\left\langle
↪  {b^\dagger_{}} \right\rangle}}{2} - \frac{\mu
↪  {\left\langle {b^\dagger_{}} b_{}^{2} \right\rangle}}{2} -
↪  \frac{\mu {\left\langle {b^\dagger_{}}^{2} b_{}
↪  \right\rangle}}{2} - \frac{\mu {\left\langle
↪  {b^\dagger_{}}^{3} \right\rangle}}{6} - i \omega_{0}
↪  {\left\langle b_{} \right\rangle}
```

4.3.3. A bipartite quantum battery with quadratic driving

A quantum battery is a quantum system that can store energy and whose energy can be harvested for useful work [7]. Recently, Downing and Ukhtary [7] proposed a setup for a bipartite quantum battery. The system contains a charger and a holder coupled to each other, with a short quadratic pulse driving the charger. The equation of motion is specified by (with $\hbar = 1$, cf. Eqs. [1–6] of Ref. [7])

$$\begin{aligned}\hat{H} = {} & \omega_c \hat{b}_c^\dagger \hat{b}_c + \omega_h \hat{b}_h^\dagger \hat{b}_h \\ & + g\left(\hat{b}_c^\dagger \hat{b}_h + \hat{b}_h^\dagger \hat{b}_c\right) + \frac{\Omega}{2}\delta(t)\left(\hat{b}_c^{\dagger 2} + \hat{b}_c^2\right)\end{aligned} \tag{32}$$

and

$$\gamma_1 = \gamma, \quad \hat{O}_1 = \hat{O}_1 = \hat{b}_c \tag{33}$$

where g is the coupling strength, Ω is the pulse strength, and $\delta(t)$ is the Dirac delta function. The quantities $\left\langle \hat{b}_c^\dagger \hat{b}_c \right\rangle$ and $\left\langle \hat{b}_h^\dagger \hat{b}_h \right\rangle$ are proportional to the energy of the charger and the holder, respectively. After $t = 0$, the

driving is off, and they evolve according to (cf. Eq. [12] of Ref. [7])

$$\frac{\mathrm{d}\left\langle \hat{b}_c^\dagger \hat{b}_c \right\rangle}{\mathrm{d}t} = -\gamma \left\langle \hat{b}_c^\dagger \hat{b}_c \right\rangle - ig \left[\left\langle \hat{b}_c^\dagger \hat{b}_h \right\rangle - \left\langle \hat{b}_h^\dagger \hat{b}_c \right\rangle \right] \tag{34a}$$

$$\frac{\mathrm{d}\left\langle \hat{b}_h^\dagger \hat{b}_h \right\rangle}{\mathrm{d}t} = ig \left[\left\langle \hat{b}_c^\dagger \hat{b}_h \right\rangle - \left\langle \hat{b}_h^\dagger \hat{b}_c \right\rangle \right] \tag{34b}$$

Here are the same equations obtained using our package:

```
omega_c, omega_h, g, gamma = \
    sm.symbols(r"omega_c omega_h g gamma")
b_c, bd_c = bl.ops("c")
b_h, bd_h = bl.ops("h")

H = omega_c * bd_c*b_c \
    + omega_h * bd_h*b_h \
    + g*(bd_c*b_h + bd_h*b_c)

D = [[gamma, b_c]]

A = bd_c*b_c

print(latex(
bl.LME_expval_evo(H,D,A)
))
```

```
Out: \frac{d}{d t} {\left\langle
 ↪  {b^\dagger_{\mathtt{\text{c}}}} b_{\mathtt{\text{c}}}
 ↪  \right\rangle} = - i g {\left\langle
 ↪  {b^\dagger_{\mathtt{\text{c}}}} b_{\mathtt{\text{h}}}
 ↪  \right\rangle} + i g {\left\langle
 ↪  {b^\dagger_{\mathtt{\text{h}}}} b_{\mathtt{\text{c}}}
 ↪  \right\rangle} - \gamma {\left\langle
 ↪  {b^\dagger_{\mathtt{\text{c}}}} b_{\mathtt{\text{c}}}
 ↪  \right\rangle}
```

Assuming the variables from the previous block,

```
A = bd_h*b_h

print(latex(
bl.LME_expval_evo(H,D,A)
))

Out: \frac{d}{d t} {\left\langle
↪  {b^\dagger_{\mathtt{\text{h}}}} b_{\mathtt{\text{h}}}
↪  \right\rangle} = i g {\left\langle
↪  {b^\dagger_{\mathtt{\text{c}}}} b_{\mathtt{\text{h}}}
↪  \right\rangle} - i g {\left\langle
↪  {b^\dagger_{\mathtt{\text{h}}}} b_{\mathtt{\text{c}}}
↪  \right\rangle}
```

4.3.4. A $\mathcal{PT}$-symmetric trimer of harmonic oscillators

Bender and Boettcher [26] show that a Hamiltonian does not need to be Hermitian to have a real, nonnegative spectrum. The parity-time ($\mathcal{PT}$) symmetry can be viewed as a generalization to the hermicity condition—this validates the non-Hermitian quantum mechanics formulation. In non-Hermitian quantum mechanics, exceptional points (EPs) mark the crossover between the unbroken and broken $\mathcal{PT}$ phase. Exceptional points are useful. For example, at its EPs, a system may gain enhanced sensitivity, making it desirable for sensing applications [27]. As another example, it is observed that the fluorescence rates of certain single-photon sources reach their quantum limit upon crossing an EP [28].

Downing and Saroka [11] formulate a simple model of short oligomer chains of harmonic oscillators with a $\mathcal{PT}$-symmetric Hamiltonian in the Lindblad master equation framework, showing the emergence of EPs. We consider a trimer system whose dynamics is specified by ($\hbar = 1$, cf. Eqs. [4, 5a, 5b, 10] of Ref. [11])

$$\begin{aligned}\hat{H} = &\left(\omega_0 + i\frac{\kappa}{2}\right)\hat{b}_1^\dagger\hat{b}_1 + \omega_0\hat{b}_2^\dagger\hat{b}_2 + \left(\omega_0 - i\frac{\kappa}{2}\right)\hat{b}_3^\dagger\hat{b}_3 \\ &+ g\left(\hat{b}_1^\dagger\hat{b}_2 + \hat{b}_2^\dagger\hat{b}_3 + \text{h.c.}\right)\end{aligned} \tag{35}$$

and

$$\gamma_1 = \gamma_1, \quad \hat{O}_1 = \hat{P}_1 = \hat{b}_1 \tag{36a}$$
$$\gamma_2 = \gamma_2, \quad \hat{O}_2 = \hat{P}_2 = \hat{b}_2 \tag{36b}$$
$$\gamma_3 = \gamma_3, \quad \hat{O}_3 = \hat{P}_3 = \hat{b}_3 \tag{36c}$$
$$\gamma_4 = p_1, \quad \hat{O}_4 = \hat{P}_4 = \hat{b}_1^\dagger \tag{36d}$$
$$\gamma_5 = p_2, \quad \hat{O}_5 = \hat{P}_5 = \hat{b}_2^\dagger \tag{36e}$$
$$\gamma_6 = p_3, \quad \hat{O}_6 = \hat{P}_6 = \hat{b}_3^\dagger \tag{36f}$$

where ω_0 is the natural frequency of all oscillators, g is the coupling strength, and κ specifies both the gain rate of oscillator 1 and loss rate of oscillator 3. Meanwhile, γ_k and p_k, $k = 1, 2, 3$, specify the gain and loss rates from incoherent processes for the oscillator k. The non-Hermitian parts of $\hat{H}$ are consequences of these processes. The evolution of $\left\langle \hat{b}_k^\dagger \hat{b}_k \right\rangle$ are given by (cf. Eqs.[9, 14–19] of Ref. [11])

$$\begin{aligned}\frac{\mathrm{d}\left\langle \hat{b}_1^\dagger \hat{b}_1 \right\rangle}{\mathrm{d}t} &= p_1 - (\gamma_1 - p_1)\left\langle \hat{b}_1^\dagger \hat{b}_1 \right\rangle \\ &\quad - ig\left\langle \hat{b}_1^\dagger \hat{b}_2 \right\rangle + ig\left\langle \hat{b}_2^\dagger \hat{b}_1 \right\rangle\end{aligned} \tag{37a}$$

$$\begin{aligned}\frac{\mathrm{d}\left\langle \hat{b}_2^\dagger \hat{b}_2 \right\rangle}{\mathrm{d}t} &= p_2 - (\gamma_2 - p_2)\left\langle \hat{b}_2^\dagger \hat{b}_2 \right\rangle \\ &\quad + ig\left\langle \hat{b}_1^\dagger \hat{b}_2 \right\rangle - ig\left\langle \hat{b}_2^\dagger \hat{b}_1 \right\rangle \\ &\quad - ig\left\langle \hat{b}_2^\dagger \hat{b}_3 \right\rangle + ig\left\langle \hat{b}_3^\dagger \hat{b}_2 \right\rangle\end{aligned} \tag{37b}$$

$$\begin{aligned}\frac{\mathrm{d}\left\langle \hat{b}_3^\dagger \hat{b}_3 \right\rangle}{\mathrm{d}t} &= p_3 - (\gamma_3 - p_3)\left\langle \hat{b}_3^\dagger \hat{b}_3 \right\rangle \\ &\quad + ig\left\langle \hat{b}_2^\dagger \hat{b}_3 \right\rangle - ig\left\langle \hat{b}_3^\dagger \hat{b}_2 \right\rangle\end{aligned} \tag{37c}$$

Here are the equations obtained using `pyBoLaNO` :

```
omega_0, kappa = sm.symbols("omega_0 kappa")
gamma_1, gamma_2, gamma_3 = sm.symbols("gamma_1 gamma_2
↪  gamma_3")
```

```
p_1, p_2, p_3 = sm.symbols("p_1 p_2 p_3")

b_1, bd_1 = bl.ops(1)
b_2, bd_2 = bl.ops(2)
b_3, bd_3 = bl.ops(3)

H = (omega_0 + sm.I*kappa/2)*bd_1*b_1 \
    + omega_0*bd_2*b_2 \
    + (omega_0 - sm.I*kappa/2)*bd_3*b_3 \
    + g*(bd_1*b_2+bd_2*b_1 + bd_2*b_3 + bd_3*b_2)

D = [[gamma_1, b_1],
     [gamma_2, b_2],
     [gamma_3, b_3],
     [p_1, bd_1],
     [p_2, bd_2],
     [p_3, bd_3]]

A = bd_1*b_1

print(latex(
bl.LME_expval_evo(H, D, A)
))

Out: \frac{d}{d t} {\left\langle {b^\dagger_{1}} b_{1}
↪  \right\rangle} = - i g {\left\langle {b^\dagger_{1}} b_{2}
↪  \right\rangle} + i g {\left\langle {b^\dagger_{2}} b_{1}
↪  \right\rangle} - \gamma_{1} {\left\langle {b^\dagger_{1}}
↪  b_{1} \right\rangle} + p_{1} {\left\langle {b^\dagger_{1}}
↪  b_{1} \right\rangle} + p_{1}
```

Assuming the variables from the previous block,

```
A = bd_2*b_2

print(latex(
bl.LME_expval_evo(H, D, A)
```

```
))

Out: \frac{d}{d t} {\left\langle {b^\dagger_{2}} b_{2}
↪  \right\rangle} = i g {\left\langle {b^\dagger_{1}} b_{2}
↪  \right\rangle} - i g {\left\langle {b^\dagger_{2}} b_{1}
↪  \right\rangle} - i g {\left\langle {b^\dagger_{2}} b_{3}
↪  \right\rangle} + i g {\left\langle {b^\dagger_{3}} b_{2}
↪  \right\rangle} - \gamma_{2} {\left\langle {b^\dagger_{2}}
↪  b_{2} \right\rangle} + p_{2} {\left\langle {b^\dagger_{2}}
↪  b_{2} \right\rangle} + p_{2}
```

and

```
A = bd_3*b_3

print(latex(
bl.LME_expval_evo(H, D, A)
))

Out: \frac{d}{d t} {\left\langle {b^\dagger_{3}} b_{3}
↪  \right\rangle} = i g {\left\langle {b^\dagger_{2}} b_{3}
↪  \right\rangle} - i g {\left\langle {b^\dagger_{3}} b_{2}
↪  \right\rangle} - \gamma_{3} {\left\langle {b^\dagger_{3}}
↪  b_{3} \right\rangle} + p_{3} {\left\langle {b^\dagger_{3}}
↪  b_{3} \right\rangle} + p_{3}
```

4.3.5. A pair of nonreciprocal driven-dissipative quantum resonators

So far, we have considered the cases where $\hat{O}_j = \hat{P}_j$. For the last example, we will discuss a system in which this is not the case.

The notion of reciprocity, i.e., when the interaction between two objects is identical if they are swapped, is encountered in various topics in physics, such as Newton's third law of motion, electromagnetic reciprocity [29], and scattering reciprocity [30]. The lack of reciprocity (in other words: nonreciprocity) can lead to interesting phenomena and practical advantages [12]. A physical model that gives rise to nonreciprocity in open quantum systems is given by Downing and Sturges [12]. The model consists of a pair of driven-dissipative quantum resonators, where the asymmetry arises from

the relative phase difference between the coherent and incoherent couplings. The dynamics is described within the Lindblad master equation framework, with ($\hbar = 1$, cf. Eqs. [1, 2] of Ref. [12])

$$\hat{H} = \Delta \sum_{k=1,2} \hat{b}_k^\dagger \hat{b}_k + \Omega \left(\hat{b}_1 + \hat{b}_1^\dagger \right) + g \left(e^{i\theta} \hat{b}_1^\dagger \hat{b}_2 + e^{-i\theta} \hat{b}_2^\dagger \hat{b}_1 \right) \tag{38}$$

and

$$\gamma_1 = \gamma, \quad \hat{O}_1 = \hat{P}_1 = \hat{b}_1 \tag{39a}$$
$$\gamma_2 = \gamma, \quad \hat{O}_2 = \hat{P}_2 = \hat{b}_2 \tag{39b}$$
$$\gamma_3 = \Gamma e^{i\phi}, \quad \hat{O}_3 = \hat{b}_2, \quad \hat{P}_3 = \hat{b}_1 \tag{39c}$$
$$\gamma_4 = \Gamma e^{-i\phi}, \quad \hat{O}_4 = \hat{b}_1, \quad \hat{P}_4 = \hat{b}_2 \tag{39d}$$

in the rotating reference frame of the laser driving oscillator 1. Here, Δ is the detuning of the oscillators with respect to the laser, Ω is the driving strength, g is the (coherent) coupling strength, and θ is the coupling phase. The third and fourth dissipators describe the incoherent coupling between the two oscillators sharing the same bath, whose rate is taken to be complex with amplitude $0 \leq \Gamma \leq \gamma$ and phase ϕ. The quantities $\left\langle \hat{b}_1 \right\rangle$ and $\left\langle \hat{b}_2 \right\rangle$ evolve according to (cf. Eqs. [3, 4] of Ref. [12])

$$\frac{\mathrm{d} \left\langle \hat{b}_1 \right\rangle}{\mathrm{d}t} = -i \left(\Delta - i\frac{\gamma}{2} \right) \left\langle \hat{b}_1 \right\rangle - \left(ige^{i\theta} + \frac{\Gamma e^{i\phi}}{2} \right) \left\langle \hat{b}_2 \right\rangle - i\Omega \tag{40a}$$
$$\frac{\mathrm{d} \left\langle \hat{b}_2 \right\rangle}{\mathrm{d}t} = -\left(ige^{-i\theta} + \frac{\Gamma e^{-i\phi}}{2} \right) \left\langle \hat{b}_1 \right\rangle - i \left(\Delta - i\frac{\gamma}{2} \right) \left\langle \hat{b}_2 \right\rangle \tag{40b}$$

Here are Eqs. (40a) and (40b) obtained using the package:

```
Delta, Omega, g, theta = sm.symbols("Delta Omega g theta")
gamma, Gamma, phi = sm.symbols("gamma Gamma phi")

b_1, bd_1 = bl.ops(1)
b_2, bd_2 = bl.ops(2)

H = Delta*(bd_1*b_1 + bd_2*b_2) \
```

```
    + Omega*(b_1+bd_1) \
    + g*(sm.E**(sm.I*theta)*bd_1*b_2 +
     ↪  sm.E**(-sm.I*theta)*bd_2*b_1)

D = [[gamma, b_1],
     [gamma, b_2],
     [Gamma*sm.E**(sm.I*phi), b_2, b_1],
     [Gamma*sm.E**(-sm.I*phi), b_1, b_2]]

A = b_1

print(latex(
bl.LME_expval_evo(H, D, A)
))

Out: \frac{d}{d t} {\left\langle b_{1} \right\rangle} = - i
 ↪  \Delta {\left\langle b_{1} \right\rangle} - \frac{\Gamma
 ↪  {\left\langle b_{2} \right\rangle} e^{i \phi}}{2} - i
 ↪  \Omega - i g {\left\langle b_{2} \right\rangle} e^{i
 ↪  \theta} - \frac{\gamma {\left\langle b_{1}
 ↪  \right\rangle}}{2}
```

Assuming variables from the previous block,

```
A = b_2

print(latex(
bl.LME_expval_evo(H, D, A)
))

Out: \frac{d}{d t} {\left\langle b_{2} \right\rangle} = - i
 ↪  \Delta {\left\langle b_{2} \right\rangle} - \frac{\Gamma
 ↪  {\left\langle b_{1} \right\rangle} e^{- i \phi}}{2} - i g
 ↪  {\left\langle b_{1} \right\rangle} e^{- i \theta} -
 ↪  \frac{\gamma {\left\langle b_{2} \right\rangle}}{2}
```

5. Performance

Since multiprocessing is implemented for each summand in the input expression, the speedup gained is approximately linear (the evaluation of each summand may take different durations). Meanwhile, the evaluation of a single monomial is optimal because explicit formulae are used. Considering Eq. (21), we see that the number of terms in the normal-ordered equivalent of the input expression depends on the powers $\{s_k\}$ of $\hat{b}$ for nonnegative excess (more $\hat{b}^\dagger$ than $\hat{b}$), and $\{r_k\}$ for negative excess. Furthermore, from Eq. (20) we can see that the calculation of the generalized Stirling number $S_{\boldsymbol{r},\boldsymbol{s}}(k)$ becomes more costly as we have more $\hat{b}$ or $\hat{b}^\dagger$ (more indices to sum over) in the expression and for larger M (more indices to multiply over for the given term in the sum).

In comparison to our package, the original `SymPy` implements what we call the "recursive flatten-and-swap" algorithm, which can be roughly described as follows:

1. For each summand in the input expression `expr`, make a list of its factors where `Pow` objects are flattened into multiple ladder operator objects.
2. Iterate through each item in the list. If the sequence $(\hat{b}_j, \hat{b}_k^\dagger)$ is found, swap their position using the commutation relations.
3. Multiply together the resulting factors to generally get an `Add` object.
4. Repeat steps 1–3 recursively until the resulting expression is not an `Add`, in which case it is added as an output summand.
5. Add together output summands to get the normal-ordered equivalent of `expr`.

The algorithm creates a recursion tree where factor-listing and operator swapping are done at all nodes except the leaves. This quickly increases the computational cost as the input expression becomes more complex.

The algorithm implemented by `pyBoLaNO`, on the other hand, provides a significant speedup over the conventional implementation. This is evident in our benchmark results as shown in Fig. 2. Our benchmarks are done in one node of our homebuilt computer cluster, Quasi Lab, running an `Intel i9-13900K` with 64 GB of RAM on a `Debian GNU/Linux 12 (bookworm) x86_64` operating system.

In our first benchmark, we time the normal ordering of 1000 random monomials containing 10 ladder operators of 2 subsystems. The result in

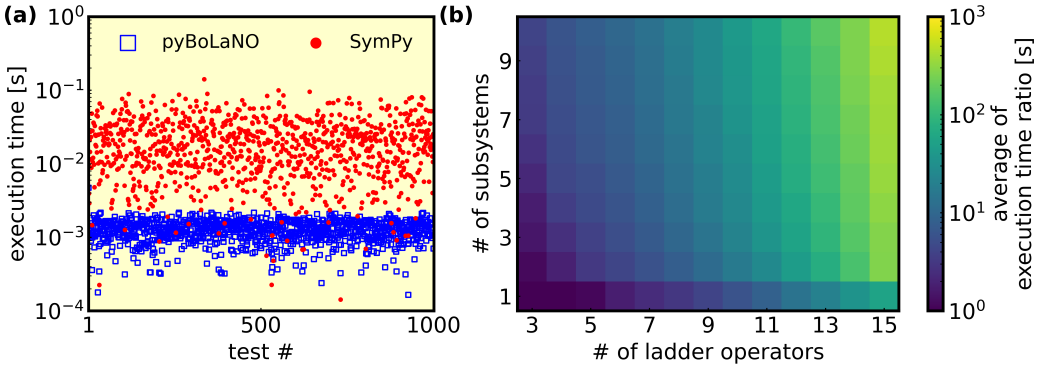

Figure 2: Benchmarks of `pyBoLaNO`'s normal ordering algorithm against `sympy.physics.operatorordering.normal_ordererd_form` involving random ladder operator monomials. **Benchmark (a):** execution time of the normal ordering of 1000 monomials containing 10 ladder operators of 2 subsystems. **Benchmark (b):** Average of the ratio between the average execution time of the normal ordering of 1000 monomials by `SymPy` and `pyBoLaNO`, for varying numbers of ladder operators and subsystems. Note: these results may not be exactly reproducible due to the random nature of the benchmarks.

Fig. 2(a) shows that our algorithm can normal-order the given input about an order of magnitude faster than that of `SymPy`. There are cases where `SymPy` is faster. Upon further inspection, we find that these are the cases where the input monomial is already normal-ordered. Our algorithm does not bypass the normal ordering process if the input is normal-ordered; adding this feature would introduce additional computational costs for other inputs. On the other hand, `SymPy` does this as a consequence of the algorithm it implements. The different spreads of the execution times for both algorithms are characteristic of them for the inputs used and are irrelevant to this discussion.

In our second benchmark, we take the average of the ratio between `SymPy`'s and our package's execution time over 1000 random ladder monomials, observing how the value varies with the number of ladder operators in the input and the number of subsystems involved. Figure 2 (b) shows a log-scale tile plot of our result. Evidently, as the numbers of ladder operators and subsystems increase, `SymPy`'s average execution time quickly grows to become multiple orders of magnitude above our package. This illustrates the superiority of the explicit formula which we implement. We finally note

that our package's multiprocessing does not help it perform better in these benchmarks since the input is a monomial.

6. Conclusion

We have developed a symbolic package `pyBoLaNO` for the normal ordering of polynomials in bosonic ladder operators utilizing Blasiak's formulae. By extension, it allows the user to obtain the normal-ordered equivalent of a commutator between two polynomials in bosonic ladder operators, as well as the normal-ordered expectation value evolution equation for a system described in the Lindblad master equation framework. We described the package workflow in detail and provided the syntaxes for the core functionalities. We have exhibited some examples of use by taking recent results from the literature which also serve to validate the package. We have discussed the computational cost of the normal ordering algorithm, showing its superiority against the conventional implementation of `SymPy`. This package is aimed at quantum physics theorists who desire fast, error-free normal ordering for bosonic ladder algebra, in particular when dealing with expectation value evolution in the Lindblad master equation framework. We welcome suggestions and constructive criticism to improve this package through its public repository.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Data availability

The package's source code and instructions for installation are publicly available at https://github.com/hendry24/pyBoLaNO. The code used for Section 4 is compiled into a `Jupyter Notebook` available at https://github.com/hendry24/pyBoLaNO/blob/main/tutorial.ipynb. The code used for Section 5 is compiled into a `Jupyter Notebook` available at https://github.com/hendry24/pyBoLaNO/blob/main/benchmarks.ipynb.

Acknowledgments

H.M.L., under the supervision of A.R.T.N., is supported by a research assistantship from the BRIN Directorate for Talent Management. All authors acknowledge Quasi Lab and Mahameru BRIN for their mini-cluster and HPC facilities.

References

[1] D. J. Griffiths, D. F. Schroeter, Introduction to quantum mechanics, 3rd Edition, Cambridge University Press, 2018.

[2] T. Lancaster, S. Blundell, Quantum field theory for the gifted amateur, 1st Edition, Oxford Univ. Press, 2014.

[3] C. C. Gerry, P. L. Knight, Introductory quantum optics, Cambridge University Press, 2005.

[4] M. Fox, Quantum Optics: An Introduction, Oxford Master Series in Physics, OUP Oxford, 2006.

[5] A. Chia, L. C. Kwek, C. Noh, Relaxation oscillations and frequency entrainment in quantum mechanics, Phys. Rev. E 102 (4) (2020) 042213.

[6] Y. Shen, W.-K. Mok, C. Noh, A. Q. Liu, L.-C. Kwek, W. Fan, A. Chia, Quantum synchronization effects induced by strong nonlinearities, Phys. Rev. A 107 (5) (2023) 053713.

[7] C. A. Downing, M. S. Ukhtary, Hyperbolic enhancement of a quantum battery, Phys. Rev. A 109 (5) (2024) 052206.

[8] D. O. Krimer, M. Zens, S. Rotter, Critical phenomena and nonlinear dynamics in a spin ensemble strongly coupled to a cavity. I. semiclassical approach, Phys. Rev. A 100 (1) (2019) 013855.

[9] M. Zens, D. O. Krimer, S. Rotter, Critical phenomena and nonlinear dynamics in a spin ensemble strongly coupled to a cavity. II. semiclassical-to-quantum boundary, Phys. Rev. A 100 (1) (2019) 013856.

[10] B. Ahmadi, P. Mazurek, P. Horodecki, S. Barzanjeh, Nonreciprocal quantum batteries, Phys. Rev, Lett. 132 (21) (2024) 210402.

[11] C. A. Downing, V. A. Saroka, Exceptional points in oligomer chains, Commun. Phys. 4 (1) (2021) 254.

[12] C. A. Downing, T. J. Sturges, Directionality between driven-dissipative resonators, Europhys. Lett. 140 (3) (2022) 35001.

[13] C. A. Downing, A. Vidiella-Barranco, Parametrically driving a quantum oscillator into exceptionality, Sci. Rep. 13 (1) (2023) 11004.

[14] L. Ben Arosh, M. C. Cross, R. Lifshitz, Quantum limit cycles and the rayleigh and van der pol oscillators, Phys. Rev. Res. 3 (2021) 013130.

[15] E. Amitai, M. Koppenhöfer, N. Lörch, C. Bruder, Quantum effects in amplitude death of coupled anharmonic self-oscillators, Phys. Rev. E 97 (2018) 052203.

[16] F. Minganti, A. Miranowicz, R. W. Chhajlany, F. Nori, Quantum exceptional points of non-hermitian hamiltonians and liouvillians: The effects of quantum jumps, Phys. Rev. A 100 (6) (2019) 062131.

[17] G. Chimczak, A. Kowalewska-Kudłaszyk, E. Lange, K. Bartkiewicz, J. Peřina, The effect of thermal photons on exceptional points in coupled resonators, Sci. Rep. 13 (1) (2023) 5859.

[18] D. Farina, G. M. Andolina, A. Mari, M. Polini, V. Giovannetti, Charger-mediated energy transfer for quantum batteries: An open-system approach, Phys. Rev. B 99 (3) (2019) 035421.

[19] G.-Q. Zhang, Z. Chen, W. Xiong, C.-H. Lam, J. Q. You, Parity-symmetry-breaking quantum phase transition via parametric drive in a cavity magnonic system, Phys. Rev. B 104 (6) (2021) 064423.

[20] A. Meurer, C. P. Smith, M. Paprocki, O. Čertík, S. B. Kirpichev, M. Rocklin, A. Kumar, S. Ivanov, J. K. Moore, S. Singh, T. Rathnayake, S. Vig, B. E. Granger, R. P. Muller, F. Bonazzi, H. Gupta, S. Vats, F. Johansson, F. Pedregosa, M. J. Curry, A. R. Terrel, Š. Roučka, A. Saboo, I. Fernando, S. Kulal, R. Cimrman, A. Scopatz, SymPy: symbolic computing in Python, PeerJ Comput. Sci. 3 (2017) e103.

[21] P. Blasiak, Combinatorics of boson normal ordering and some applications (2005). arXiv:quant-ph/0507206.

[22] M. A. Méndez, P. Blasiak, K. A. Penson, Combinatorial approach to generalized bell and stirling numbers and boson normal ordering problem, J. Math. Phys. 46 (8) (2005) 083511.

[23] M. Schlosshauer, Decoherence and the quantum-to-classical transition, Springer, 2007.

[24] H.-P. Breuer, F. Petruccione, The theory of open quantum systems, Oxford University Press, 2002.

[25] A. Pikovsky, M. Rosenblum, J. Kurths, Synchronization: a universal concept in nonlinear sciences, Cambridge University Press, 2001.

[26] C. M. Bender, S. Boettcher, Real spectra in non-hermitian hamiltonians having $\mathcal{PT}$ symmetry, Phys. Rev. Lett. 80 (24) (1998) 5243.

[27] J. Wiersig, Prospects and fundamental limits in exceptional point-based sensing, Nat. Commun. 11 (1) (2020) 2454.

[28] W. Zhou, J. Liu, J. Zhu, D. Gromyko, C. Qiu, L. Wu, Exceptional points unveiling quantum limit of fluorescence rates in non-hermitian plexcitonic single-photon sources, APL Quantum 1 (1) (2024) 016110.

[29] C. Caloz, A. Alù, S. Tretyakov, D. Sounas, K. Achouri, Z.-L. Deck-Léger, Electromagnetic nonreciprocity, Phys. Rev. Appl. 10 (4) (2018) 047001.

[30] L. Deák, T. Fülöp, Reciprocity in quantum, electromagnetic and other wave scattering, Ann. of Phys. 327 (4) (2012) 1050.