South Tangerang, April 16, 2025

Dear Editor of Computer Physics Communications (CPC),

Thank you for sending us the referee reports and the invitation to submit a revised version of our article titled "pyBoLaNO: A Python symbolic package for normal ordering involving bosonic ladder operators" (COMPHY-D-25-00012).

We have carefully read the reports of the two reviewers and thank them for kindly evaluating the manuscript.

We fully address all the comments and questions from the reviewers by revising some parts of the manuscript, as well as the source code of our package. The revised parts of the manuscript are indicated in red in the compiled PDF file, while in the LaTeX source we use a self-defined environment `\begin{revision}...\end{revision}`, which can be safely deleted by the copy-editor once this manuscript can go to the production stage.

Enclosed with this response letter, we provide more detailed answers to each comment/question, alongside the corresponding changes made to the manuscript. We have also included some addenda to the manuscript upon checking our manuscript and finding some errors. We describe the changes after we address the reviewer's reports in this message.

We thank you for your consideration and look forward to hearing from you.

Corresponding author:

Hendry Minfui Lim

hendry01@ui.ac.id

Research Center for Quantum Physics, National Research and Innovation Agency (BRIN), South Tangerang 15314, Indonesia

Department of Physics, Faculty of Mathematics and Natural Sciences, Universitas Indonesia, Depok 16424, Indonesia

REPLY TO REPORT OF REVIEWER 1

REVIEWER'S REMARK:

We have attached a file with feedback.

AUTHORS:

We thank you for reviewing our manuscript and providing insightful responses. Below are our responses to your questions/comments. Unfortunately, the attached file seems to be missing from the review report sent to us. Since we received this report, we have mailed the Editor to obtain the file but have yet to receive any response upon completing this round of revision. As such, we are unable to address the feedback given in the file. Nevertheless, at this moment we can at least respond to your other comments present in the report. If you are not satisfied with our answers, we are happy to continue to the next revision stage.

QUESTION/COMMENT 1:

It appears that some (or all) of the functionality is available elsewhere (https://github.com/sympy/sympy/tree/master/sympy/physics/quantum/operatorordering.py).

REPLY:

We thank the reviewer for pointing this out. We admit that we are unaware of `sympy.physics.quantum.operatorordering`, as probably evident from the commit history of our GitHub repository. Our package, formerly named `boson_ladder`, used the same algorithm as that in SymPy, where the commutation relations are recursively used to reorder the operators (see, for example, `Commit a234f48`, which is before the addition of Blasiak's formulae). Then, we found one critical drawback to this method: *the resulting recursion tree is highly inefficient*.

Similar to normal-ordering by hand, the method currently applied in SymPy grows quickly in computation cost as the number of ladder operators increases and the excess (the difference between the number of annihilation operators and creation operators in the expression) decreases. For example, the normal ordering of $\hat{b}^{10}\hat{b}^{\dagger 10}$ takes more than 10 seconds on the corresponding author's personal computer. This limitation is overcome by Blasiak's formulae, which provide an explicit formula for the normal-ordered equivalent of any given expression involving the ladder operators.

We further realized that the claim that normal ordering functionalities are absent in SymPy `ver.1.13.3` was also incorrect. **However**, we believe that the **more efficient method implemented in our package**, as well as the **convenient way for computing the evolution of any expectation value** within the Lindblad master equation framework, can serve as **novelties worth publishing**.

To address this comment, we have made several significant changes to the manuscript:

1. We deleted our incorrect claim and reworded our motivation for writing the package, highlighting its novelties over the conventional SymPy.
2. We have added a benchmark against SymPy's implementation in Section 5 (Performance) to demonstrate the superiority of the explicit formulae.
3. As we are uncertain of how SymPy's developers will resolve this conflict (were it be one), we have decided to not use the ladder operator objects in SymPy and instead wrote our package's own base for the ladder operator object, `pybolano.utils.operators.pybolanoOp`, which are similar to `sympy.physics.secondquant.SqOperator` in construction. It is added with `Commit 8838d27`.

BEFORE REVISION:

(paragraphs 2–3 of Section 1 of the original manuscript)

As the system description becomes more complex, the algebraic manipulation of the dynamical equations becomes more tedious and prone to errors. As such, a calculator that does the algebra is desirable. The `SymPy` package [1] 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` submodule. However, the support is limited to arithmetic operations. The function `wicks` for normal (Wick) ordering is not implemented for bosonic ladder operators. In practice, we often desire to evaluate the normal ordering of operators involving bosonic ladder operators. An example of such cases, which motivates us for this work, is the evolution of the expectation values for open quantum systems described by the Lindblad master equation [2, 3]. If analytical expressions are not the concern, several packages are available to solve the problem numerically, e.g., `QuTiP` [4, 5] and `QuantumOptics.jl` [6]. Otherwise, to our knowledge [7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21], there is yet a symbolic computational package that facilitates the solution to the problem.

In this work, 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 and compatible with `SymPy`. It supports multipartite descriptions and multiprocessing for each additive term in the input(s).

(paragraphs 1 of Section 3.2 of the revised manuscript)

Instead of calling `AnnihilateBoson(k)` and `CreateBoson(k)` from `sympy.physics.secondquant`, where `k` is the subscript, we recommend using `pyBoLaNO` 's `ops`:

```
ops(k)
```

that can be used to construct the annihilation and creation operators. The subscript `k` is optional and can be `Symbol` or any `Python` object convertible to a string. This function preprocesses `k` into `Symbol`, which is the most compatible with `AnnihilateBoson` and `CreateBoson`. In the current version of `SymPy`, it also avoids the printing error that occurs when nonzero integers are used as subscripts (which is likely a bug).

AFTER REVISION:

(paragraphs 3–4 of Section 1 of the revised manuscript)

The SymPy package [1] 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 implemented algorithm 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).

(paragraphs 1 of Section 3.2 of the revised manuscript)

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 the ladder operators.

QUESTION/COMMENT 2:

We also think the program can only do relatively simple explicit operations, and is less useful in a research setting in which an operator might have a sum over many bosons operators corresponding to different normal modes.

REPLY:

Thank you for your comment. We kindly disagree with this assessment. As pointed out in the manuscript (and in the GitHub `README`), our package **supports the normal ordering of any polynomial containing the ladder operators, including one in a many-body setting**. The algorithm is described in Section 3.3, which we believe is computationally inexpensive considering how the conventional `SymPy` package works. The computational performance in multipartite settings is discussed in the revised **Section 5: Performance** and we show how our package is superior to the original `SymPy`.

REPLY TO REPORT OF REVIEWER 2

REVIEWER'S REMARK:

In this work, the authors put forth a method for symbolically computing the normal-ordering of Bosonic operators. The authors detail the method behind the work, outline the package organization, and give a set of usage examples. This work is largely based on existing functionality in the SymPy package, and appears to fill a gap in that programs implementation of Wick ordering.

AUTHORS:

We thank you for reviewing our manuscript and providing insightful responses. Below are our responses to your questions/comments.

QUESTION/COMMENT 1:

I am curious as to why this is a separate package and not a simple Pull-Request to the SymPy repository? The package does appear to be useful, but I believe the authors should address why their work needs to be a separate installation.

REPLY:

Thanks for the question. We opted for a separate symbolic package as we prefer a simple package that does specific jobs and desire the freedom to add new relevant features to the package. It is not guaranteed that SymPy will accept every pull request we will be making in the future, as their design philosophy may differ from ours. As an example, `SymPy` does not natively support multiprocessing, which greatly benefits our normal ordering algorithm. However, we acknowledge the potential value of integration and remain open to collaborating with the `SymPy` community in the future to explore merging our work if there is sufficient interest. This standpoint is added and described in the second-to-last paragraph of **Section 1** of the revised manuscript:

—

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).

—

QUESTION/COMMENT 2:

The authors do not have any unit-tests in their repository. As this package is supposed to be used by other researchers, I would expect some level of testing that gives me confidence in its usage.

REPLY:

We thank you for your criticism. We have added unit testing to the package you can find at `pybolano/testing`, added with `Commit 8838d27`.

QUESTION/COMMENT 3:

The performance section needs to show actual data. In my execution of their code, some cells took ~ 1 sec to run, indicating that more complicated examples might take substantially longer. Some kind of scalable example, such as scaling the number of subsystems in a problem, would greatly help in understanding the performance of this work

REPLY:

We thank you for your criticism. While using explicit formulae is arguably the most efficient method, it is indeed good to know for what expressions the evaluation is more computationally costly. We have rewritten Section 5 to discuss the performance of our algorithm in more detail, additionally comparing it to the "conventional" method `SymPy` currently implements. The conclusion has been adjusted accordingly, and we have added a link to the benchmark file in **Data Availability**.

BEFORE REVISION:

(Section 5 of the original manuscript)

Since it is implemented for each summand in the input expression, the speedup gained by using multiprocessing 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.

(Section 6 of the original manuscript)

... 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. ...

(Data Availability)

... The code used for Section 4 is compiled into a `Jupyter Notebook` available at https://github.com/hendry24/pyBoLaNO/blob/main/tutorial.ipynb.

AFTER REVISION:

(Section 5 of the revised manuscript)

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 $S_{\mathbf{r},\mathbf{s}}(k)$ 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 originalSymPy 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 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

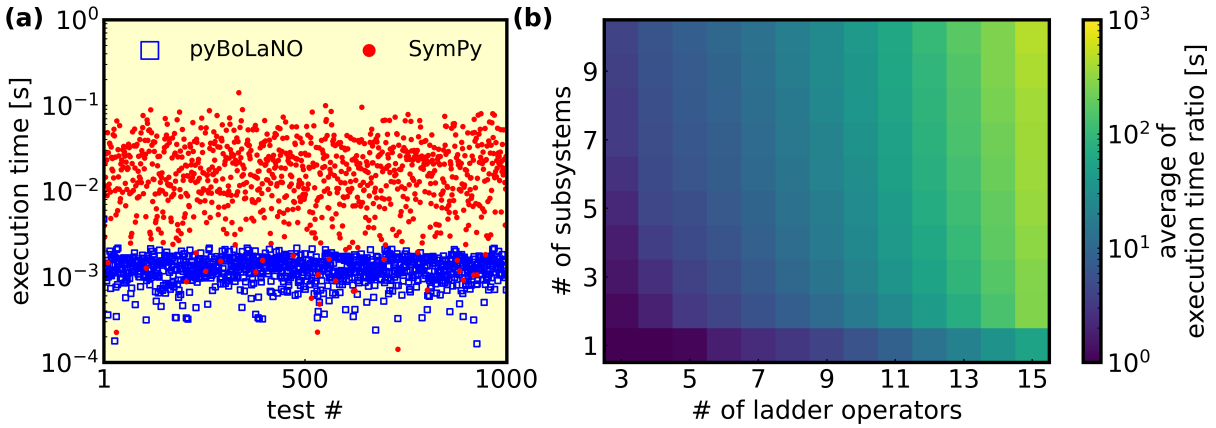

Figure 2: Benchmarks of pyBoLaNO's normal ordering algorithm against sympy.physics.operatorordering.normal_ordererd_form involving random ladder operator monomials. **Benchmark (a):** execution time of the normal ordering of 1000 monomials containing 10 ladder operators of 2 subsystems. **Benchmark (b):** 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.

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.

(Section 6 of the revised manuscript)

... 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. ...

(Data Availability)

... 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.

QUESTION/COMMENT 4:

This work is focused on analytical evaluation, but more often than not one needs to numerically solve for the dynamics of open quantum systems. It would be nice if the authors showed how to take their output and solve the resulting equations of motion numerically to obtain an output, e.g. a plot of expectation values over time for some system. This would give the reader an end-to-end example that they could extend in their own research.

REPLY:

We thank you for the suggestion. While this is an interesting idea, we believe that with the differential equations obtained, running a numerical simulation with them is a *trivial* matter. This is especially true for our targeted audience, who we believe have taken at least one course in numerical methods for differential equations. Arguably, adding a discussion about numerical methods would break the flow of the manuscript, which is focused on symbolic evaluation.

ADDENDUM

Upon further inspection, we have realized that the term "normal ordering" is conventionally used to mean rewriting a given ladder monomial into a normal-ordered monomial $\hat{b}^{\dagger p}\hat{b}^{q}$ containing the same number of creator and annihilator operators, i.e. Wick ordering. Since this does not regard the commutation relations, the resulting expression is not equivalent to the original. This is *not* what we want to do with our package. Instead, our definition of "normal ordering" is to rewrite a given ladder monomial into an equivalent normal-ordered polynomial. We have fixed this misunderstanding in our manuscript by revising the abstract, keywords, program summary, Section 1, and Section 2.1.

BEFORE REVISION:

(Abstract)

We present `pyBoLaNO`, a `Python` symbolic package based on `SymPy` to quickly normal-order (Wick-order) any polynomial in bosonic ladder operators. ...

Keywords: bosonic ladder operators, normal ordering, Wick ordering, commutator, Lindblad master equation

(Program Summmary)

...

Nature of problem: Normal (Wick) ordering involving bosonic ladder operators.

Solution method: Blasiak's formulae for the normal ordering of an arbitrary monomial in bosonic ladder operators. Symbolic programming is fully provided by `SymPy`.

(Paragraph 2 of Section 1 of the original manuscript)

As the system description becomes more complex, the algebraic manipulation of the dynamical equations becomes more tedious and prone to errors. As such, a calculator that does the algebra is desirable. The `SymPy` package [1] 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` submodule. However, the support is limited to arithmetic operations. The function `wicks` for normal (Wick) ordering is not implemented for bosonic ladder operators. In practice, we often desire to evaluate the normal ordering of operators involving bosonic ladder operators. An example of such cases, which motivates us for this work, is the evolution of the expectation values for open quantum systems described by the Lindblad master equation [2, 3]. If analytical expressions are not the concern, several packages are available to solve the problem numerically, e.g., `QuTiP` [4, 5] and `QuantumOptics.jl` [6]. Otherwise, to our knowledge [7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21], there is yet a symbolic computational package that facilitates the solution to the problem.

(Paragraph 1 of Section 2.1. in the original manuscript)

Normal ordering (or Wick ordering) means using commutation relations to rewrite the given expression so that all creation operators are written to the left of all annihilation operators. This is important in quantum field theory to avoid divergent integrals and convenient for calculations via Wick's theorem [22]. In quantum optics, it is convenient for calculations via the optical equivalence theorem [23]. The normal ordering of an operator $\hat{A}$ may be denoted by $:\hat{A}:$, for example,

$$:\hat{b}_j\hat{b}_j^\dagger\hat{b}_j:=\hat{b}_j+\hat{b}_j^\dagger\hat{b}_j^2 \tag{15}$$

AFTER REVISION:

(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. ...

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

(Program Summmary)

...

Nature of problem: 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`.

(Paragraph 2–3 of Section 1 of the revised manuscript)

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 [23, 24]. It is also quite common in the literature to present the evolution equation for a given expectation value in a normal-ordered form [7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21]. 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 [1] 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.

(Paragraph 1–2 of Section 2.1. in the revised manuscript)

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. [22]). In quantum optics, it is useful to simplify equations in quadrature squeezing (see, for example, Chapter 7 in Ref. [23]).

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 [23, Chapter 3].

References

[1] Aaron Meurer, Christopher P. Smith, Mateusz Paprocki, Ondřej Čertík, Sergey B. Kirpichev, Matthew Rocklin, AMiT Kumar, Sergiu Ivanov, Jason K. Moore, Sartaj Singh, Thilina Rathnayake, Sean Vig, Brian E. Granger, Richard P. Muller, Francesco Bonazzi, Harsh Gupta, Shivam Vats, Fredrik Johansson, Fabian Pedregosa, Matthew J. Curry, Andy R. Terrel, Štěpán Roučka, Ashutosh Saboo, Isuru Fernando, Sumith Kulal, Robert Cimrman, and Anthony Scopatz. SymPy: symbolic computing in Python. *PeerJ Comput. Sci.*, 3:e103, January 2017.

[2] Maximilian Schlosshauer. *Decoherence and the quantum-to-classical transition*. Springer, 2007.

[3] Heinz-Peter Breuer and F. Petruccione. *The theory of open quantum systems*. Oxford University Press, 2002.

[4] J.R. Johansson, P.D. Nation, and Franco Nori. QuTiP: An open-source Python framework for the dynamics of open quantum systems. *Comput. Phys. Commun.*, 183(8):1760, August 2012.

[5] J.R. Johansson, P.D. Nation, and Franco Nori. QuTiP 2: A Python framework for the dynamics of open quantum systems. *Comput. Phys. Commun.*, 184(4):1234, April 2013.

[6] Sebastian Krämer, David Plankensteiner, Laurin Ostermann, and Helmut Ritsch. QuantumOptics.jl: A Julia framework for simulating open quantum systems. *Comput. Phys. Commun.*, 227:109, 2018.

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

[8] Yuan Shen, Wai-Keong Mok, Changsuk Noh, Ai Qun Liu, Leong-Chuan Kwek, Weijun Fan, and Andy Chia. Quantum synchronization effects induced by strong nonlinearities. *Phys. Rev. A*, 107(5):053713, May 2023.

[9] Charles Andrew Downing and Muhammad Shoufie Ukhtary. Hyperbolic enhancement of a quantum battery. *Phys. Rev. A*, 109(5):052206, May 2024.

[10] Dmitry O. Krimer, Matthias Zens, and Stefan Rotter. Critical phenomena and nonlinear dynamics in a spin ensemble strongly coupled to a cavity. I. semiclassical approach. *Phys. Rev. A*, 100(1):013855, 2019.

[11] Matthias Zens, Dmitry O. Krimer, and Stefan 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):013856, 2019.

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

[13] Charles Andrew Downing and Vasil Arkadievich Saroka. Exceptional points in oligomer chains. *Commun. Phys.*, 4(1):254, 2021.

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

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

[16] Lior Ben Arosh, M. C. Cross, and Ron Lifshitz. Quantum limit cycles and the rayleigh and van der pol oscillators. *Phys. Rev. Res.*, 3:013130, February 2021.

[17] Ehud Amitai, Martin Koppenhöfer, Niels Lörch, and Christoph Bruder. Quantum effects in amplitude death of coupled anharmonic self-oscillators. *Phys. Rev. E*, 97:052203, 2018.

[18] Fabrizio Minganti, Adam Miranowicz, Ravindra W. Chhajlany, and Franco Nori. Quantum exceptional points of non-hermitian hamiltonians and liouvillians: The effects of quantum jumps. *Phys. Rev. A*, 100(6):062131, 2019.

[19] Grzegorz Chimczak, Anna Kowalewska-Kudłaszyk, Ewelina Lange, Karol Bartkiewicz, and Jan Peřina. The effect of thermal photons on exceptional points in coupled resonators. *Sci. Rep.*, 13(1):5859, 2023.

[20] Donato Farina, Gian Marcello Andolina, Andrea Mari, Marco Polini, and Vittorio Giovannetti. Charger-mediated energy transfer for quantum batteries: An open-system approach. *Phys. Rev. B*, 99(3):035421, 2019.

[21] Guo-Qiang Zhang, Zhen Chen, Wei Xiong, Chi-Hang Lam, and J. Q. You. Parity-symmetry-breaking quantum phase transition via parametric drive in a cavity magnonic system. *Phys. Rev. B*, 104(6):064423, 2021.

[22] Tom Lancaster and Stephen Blundell. *Quantum field theory for the gifted amateur*. Oxford Univ. Press, 1. ed. edition, 2014.

[23] Christopher C. Gerry and Peter L. Knight. *Introductory quantum optics*. Cambridge University Press, 2005.

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