Received: 30 January 2024 | Revised: 17 April 2024 | Accepted: 26 April 2024

DOI: 10.1002/jcc.27437

SOFTWARE NOTE

Journal of COMPUTATIONAL CHEMISTRY WILEY

Tinned: A symbolic library for response theory and high-order derivatives

Bin Gao

Hylleraas Centre for Quantum Molecular Sciences, Department of Chemistry, UiT The Arctic University of Norway, Tromsø, Norway

Correspondence
Bin Gao, Hylleraas Centre for Quantum Molecular Sciences, Department of Chemistry, UiT The Arctic University of Norway, N–9037 Tromsø, Norway.
Email: bin.gao@uit.no

Funding information
Norges Forskningsråd; Sigma2, Grant/Award Number: NN14654K

Abstract

A symbolic C++ library—Tinned—has been developed for symbolic differentiation and manipulation in response theory. By recognizing different key building blocks in the density matrix-based (Thorvaldsen *et al.*, J. Chem. Phys. 2008, **129**, 214108) and coupled-cluster response theories, we have implemented their corresponding C++ symbolic classes, including but not limited to one- and two-electron operators, exchange-correlation energy and potential, and coupled-cluster operator. Formulas of response theory can be well expressed in terms of the symbolic classes in the library Tinned. Their high-order perturbation-strength derivatives can be straightforwardly computed and extracted afterwards for numerical evaluation. The library Tinned will greatly facilitate the development work of response theory and may lead to a unified framework for response theory at different levels of electronic structure theory.

KEYWORDS
derivatives of exchange-correlation energy, derivatives of exchange-correlation potential, response theory, symbolic computation, symbolic differentiation

INTRODUCTION

Ever since the pioneering work by Olsen and Jørgensen,[1] response theory has been a valuable tool for computations of various molecular properties. Response functions and residues are two key quantities for the determination of molecular properties in response theory calculations. These calculations involve high-order derivatives with respect to different applied perturbations from the level of the response theory itself and down to a variety of integral evaluations.

Molecular response theory so far has been developed at different levels of electronic structure theory,[2] such as Hartree-Fock self-consistent field (SCF) theory, Kohn–Sham (KS) density functional theory, multiconfigurational SCF theory, coupled-cluster theory and Møller–Plesset theory. In particular, the work by Thorvaldsen et al. [3] has made it possible to study properties of large molecules by using the density matrix-based response theory.

Notice the complexity of response theory as well as its practical implementation, it is definitely valuable that a scheme can automatically calculate molecular properties of arbitrary order without code reimplementation. In a recent work,[4] Ringholm, Jonsson and Ruud have presented a new scheme for implementing the density matrix-based response theory [3] by using the technique of recursive programming. This technique identifies all perturbed overlap, density and Fock matrices in a recursive way from the lower order ones instead of implementing codes order by order. Later on, Friese, Ringholm, Ruud and their co-workers have extended the recursive approach to the calculations of single residues at the level of density matrix-based response theory.[5,6]

A stand-alone library named as OpenRSP [7] has been developed by the author and collaborators based on the recursive programming technique.[4] The library OpenRSP is able to compute molecular properties of arbitrary order automatically at levels of Hartree-Fock and

KS density functional theories. However, it is challenging to extend this library to include response theory for other electronic-structure theories, for example, the coupled cluster theory. One important reason behind the challenge is because most parts of the library OpenRSP are developed for the density matrix-based response theory and cannot be reused for other theoretical levels.

It is interesting to note that the recursive programming technique [4] actually employs symbolic differentiation in an implicit way, that is, it firstly identifies perturbed matrices rather than calculating them in a numerical manner. Obviously, one common and important task for response theory is the symbolic differentiation with respect to different perturbations regardless of the level of electronic-structure theory. It reminds us that a more open-ended and less error-prone library can be developed if one can explicitly separate symbolic computations from numerical evaluations for response theory—so that codes for symbolic computations can be decoupled from the "monolithic" response theory codes and reused for different electronic-structure theories without or with little work.

There are an impressive number of computer algebra systems (CAS) for symbolic calculations, such as Axiom, GiNaC, Maple, Mathematica, Maxima, REDUCE and SymbolicC++, just name a few. However, some of these CASes do not provide an application programming interface so that it is impossible for a response theory library to access the CAS facilities. There are also other issues to be considered for the development and use of symbolic computations for response theory, including but not limited to (i) commutative property of quantum operators, (ii) elimination of response parameters according to the k_{2n+1} and $k_{2n+1,2n+2}$ rules,[8] (iii) derivatives are indeed evaluated at zero perturbation strength and additional frequency factor(s) should be taken care for the time differentiation operator,[3] and (iv) various data types of numerical results evaluated from symbolic expressions—for example, one may use tensors for response functions and residues, and an array (or a vector) of matrices for differentiated operators.

Therefore, it is impossible to directly use existing CASes for the development of response theory. The objective of the current work is to address the issues of symbolic computations in response theory and to develop a stand-alone symbolic library, named as Tinned [9] that is tuned for response theories at different theoretical levels. This library is expected to become the cornerstone of a more open-ended and unified framework for response theory development and computations.

The remainder of this paper is organized as follows: we first give a brief introduction to the quasi-energy formulation of response theory, followed by a formal definition of perturbation-strength derivatives, and key formulas for the density matrix-based [3] and coupled-cluster [2,10] response theories. From these key formulas, we can identify basic symbols and symbolic operations needed for response theory. Next, their design and implementation in the library Tinned will be discussed. In particular, we have also developed a flexible strategy for the (numerical) evaluation of a (differentiated) symbolic expression.

Before making our final remarks, we illustrate the application of the library Tinned by a few snippets to construct the derivative of generalized energy in density matrix-based response theory and the quasi-energy Lagrangian in coupled-cluster response theory, to eliminate response parameters according to the (k,n) rule,[3] and finally to find all (un)perturbed density matrices in the differentiated generalized energy.

THEORETICAL BACKGROUND

In this section, we will briefly summarize the quasi-energy formulation at the levels of density matrix-based and coupled-cluster response theories, from which we can recognize basic symbols for response theory calculations.

In the framework of quasi-energy formulation, the key quantity and starting point is the time-dependent quasi-energy $Q(t)$ [2]

$$Q(t) = \langle \Phi(t)|\hat{H}(t) - \mathrm{i}\frac{\partial}{\partial t}|\Phi(t)\rangle, \tag{1}$$

where $\Phi(t)$ is a phase-isolated state and satisfies [3]

$$\left(\hat{H}(t) - \mathrm{i}\frac{\partial}{\partial t}\right)|\Phi(t)\rangle = Q(t)|\Phi(t)\rangle. \tag{2}$$

The Hamiltonian $\hat{H}(t) = \hat{H}_0 + \hat{V}(t)$ with $\hat{H}_0$ being a time-independent unperturbed Hamiltonian, and $\hat{V}(t)$ a time-dependent operator representing different applied perturbations on the system.

The key point in the quasi-energy formulation is that we require $\hat{V}(t)$ to be time-periodic with a period T, and can be expressed in terms of Fourier series [11]

$$\hat{V}(t) = \sum_{\hat{A}} \varepsilon_A(t)\hat{A} = \sum_{\hat{A}} \left[\sum_{\omega_A} \varepsilon_{\omega_A} \exp(-\mathrm{i}\omega_A t)\right]\hat{A}, \tag{3}$$

where $\varepsilon_A(t)$ is a real-valued time-dependent perturbation strength of a time-independent one-electron operator $\hat{A}$, and the sum in the brackets $[\cdots]$ is the Fourier series of $\varepsilon_A(t)$ because it is also a time-periodic function. The frequencies ω_A can be expressed as an integer times some fundamental frequency, as $z_A\omega_0$ ($z_A \in \mathbb{Z}$) according to the quasi-energy approach.[3] The Fourier series should contain both frequencies $\pm\omega_A$, and the complex Fourier coefficients ε_{ω_A} should satisfy $\varepsilon_{-\omega_A} = \varepsilon^*_{\omega_A}$ for $\varepsilon_A(t)$ to be real-valued. By further requiring the hermiticity of the operator $\hat{A}$ ($\hat{A}^\dagger = \hat{A}$), we ensure $\hat{V}(t)$ is Hermitian.

Due to the periodicity of $\hat{V}(t)$, the Hamiltonian $\hat{H}$ and the state $|\Phi(t)\rangle$ also become time-periodic with the same period T. The time-averaged quasi-energy can then be introduced [3]

$$\{Q(t)\}_T = \frac{1}{T}\int_{-\frac{T}{2}}^{\frac{T}{2}} Q(t)\mathrm{d}t. \tag{4}$$

For exact states $\Phi(t)$, the following variational condition holds [2]

$$\delta\{Q(t)\}_T = 0, \tag{5}$$

from which and the time-averaged Hellmann–Feynmann theorem,[2,3] molecular response functions can be determined as perturbation-strength derivatives of the time-averaged quasi-energy [3,12,13]

$$\langle\langle \hat{A}; \hat{B}_1, \cdots, \hat{B}_n \rangle\rangle_{\omega_{B_1}, \cdots, \omega_{B_n}} = \frac{\mathrm{d}^{n+1}\{Q(t)\}_T}{\mathrm{d}\varepsilon_{\omega_A}\mathrm{d}\varepsilon_{\omega_{B_1}}\cdots\mathrm{d}\varepsilon_{\omega_{B_n}}}\bigg|_{\{\varepsilon\}=0}, \tag{6}$$

where

$$\omega_A = -\sum_{j=1}^{n}\omega_{B_j}, \tag{7}$$

and $\{\varepsilon\}=0$ denotes the evaluation at zero perturbation strength $\varepsilon_{\omega_A}, \varepsilon_{\omega_{B_1}}, \ldots, \varepsilon_{\omega_{B_n}} = 0$.

For approximate states from the density matrix-based and coupled-cluster theories, the time-averaged quasi-energy $\{Q(t)\}_T$ no longer satisfies the variational condition (5) alone. To allow for unconstrained variations, one could use the so-called time-averaged quasi-energy derivative Lagrangian [3] for the density matrix-based response theory and the time-averaged Lagrangian [2] for the coupled-cluster response theory, where $\{Q(t)\}_T$ becomes an integral part of the formulation.

In the following subsections, we will first introduce a formal definition for the perturbation-strength derivatives, and then the key formulas for the density matrix-based and coupled-cluster response theories. Such a formal definition will make the formulas of response theory in a more compact manner.

Last but not least, rules for elimination of response parameters [3,4,8] and residues of response functions [2,3] are both of key importance. But they will not bring any new basic symbols for response theory calculations, so we will focus on the response functions and its $n+1$ formulation in our current work.

Perturbation tuple and perturbation index chain

Hereinafter, we will use small letters $a,b,c,\cdots$ to denote the corresponding Fourier coefficients (or perturbation strength) $\varepsilon_{\omega_A}, \varepsilon_{\omega_B}, \varepsilon_{\omega_C}, \cdots$ for the sake of simplicity. We define "a perturbation tuple" as an ordered list of perturbation strength $a,b,c,\cdots$, and additionally we require that

1. identical perturbation strength (belong to a same operator $\hat{A}$ and with a same frequency ω_A) should be consecutive, and
2. the order of a tuple determines the shape of its corresponding derivatives (usually stored in a tensor).

In the current work, we represent a perturbation tuple in two ways, either putting the perturbation strength $a,b,c,\cdots$ into a parentheses as $(a,b,c,\cdots)$, or in a more compact way $abc\cdots$.

From the above definition, the following remarks are obvious:

1. The length of a perturbation tuple is the order of its corresponding derivative;
2. Multiple instances of a same perturbation strength are allowed so that $(a,b,b,c) \neq (a,b,c)$;
3. A tuple like (a,b,c,b) is illegal because the identical perturbation strength b are not consecutive;
4. Tuples $(a,b,c) \neq (a,c,b)$, because they are in different order and result into derivatives stored in tensors with different shapes.
   For example, let a, b and c be the electric dipole, magnetic dipole and geometrical perturbations. The shapes of derivative tensors are $(3,3,3N)$ and $(3,3N,3)$ respectively for tuples (a,b,c) and (a,c,b), where N is the number of atoms.

To evaluate derivatives of a product, we will also introduce the "sub-perturbation tuple" that is constructed

1. by choosing a few perturbation strength from another perturbation tuple (named as "parent-perturbation tuple"), and
2. by following exactly the same order of the parent-perturbation tuple.

For instance, the following perturbation tuples are the sub-perturbation tuples of abc: $\emptyset$, a, b, c, ab, ac, bc, abc, with two special cases—$\emptyset$ the empty tuple and abc itself. However, tuples like ba, ca and cb are not the sub-perturbation tuples of abc, because they do not follow the same order as abc.

It should be noted that derivatives with respect to the empty tuple $\emptyset$ means zeroth-order derivative throughout the current work.

The formulation and evaluation of derivatives (of a product) can be further facilitated by using the conception of "perturbation index chain", which is a totally ordered set (or a chain) [14] $\{1,\ldots,n\}$ with each element pointing the position of its corresponding perturbation strength appeared in a perturbation tuple $b_1\cdots b_n$.

A special index chain is the empty set $\emptyset$, which represents the empty perturbation tuple and the zeroth-order derivative.

By considering the requirements on perturbation tuples, the following remarks are obvious:

1. The cardinality (denoted as $|\cdots|$) of a perturbation index chain equals to the length of its corresponding perturbation tuple, and the order of the corresponding derivative.
2. Each perturbation index chain $\{1,\ldots,n\}$ has 2^n sub-chains with the same order of elements as the whole chain, and each sub-chain corresponds to a sub-perturbation tuple of the parent-perturbation tuple $b_1\cdots b_n$.
3. Two special sub-chains are $\emptyset$ and $\{1,\ldots,n\}$ itself.

Throughout the current paper, we denote a sub-chain P of $\{1,\ldots,n\}$ as $P \subseteq \{1,\ldots,n\}$. When the sub-chain P is not equal to $\{1,\ldots,n\}$, it is called "a proper sub-chain", and is denoted as $P \subsetneq \{1,\ldots,n\}$.

Except for sub-chains, another important topic related to the perturbation index chain is "a partition of a set". Mathematically, a set

partition π of a set (or a chain) $\{1,...,n\}$ is a collection of k non-empty and disjoint subsets (sub-chains) $\pi_1,...,\pi_k$ of $\{1,...,n\}$, that is [15]

$$\begin{cases} \pi_i \neq \emptyset, \quad 1 \le i \le k, \\ \pi_i \cap \pi_j = \emptyset, \quad 1 \le i \ne j \le k, \\ \cup_{i=1}^{k} \pi_i = \{1,...,n\}, \\ \min(\pi_1) < \cdots < \min(\pi_k), \end{cases} \tag{8}$$

where a subset (sub-chain) π_i ($1 \le i \le k$) is often named as the ith block of the set partition π. The last requirement in the above definition means that the subsets (sub-chains) $\pi_1,...,\pi_k$ are listed in increasing order of their minimal elements, and alternatively, one can require a decreasing order of their maximal elements.[15]

The set of all set partitions of a chain $\{1,...,n\}$ with exactly k ($1 \le k \le n$) blocks is denoted as $\mathcal{P}_{\{1,\cdots,n\},k}$ or simply $\mathcal{P}_{n,k}$, and the number of such set partitions is

$$|\mathcal{P}_{n,k}| = \left\{ {n \atop k} \right\}, \tag{9}$$

where $\left\{ {n \atop k} \right\}$ is the Stirling number of the second kind.[15]

Density matrix-based response theory

In the density matrix-based response theory, molecular response functions can be determined from the time-averaged quasi-energy derivative Lagrangian [3]

$$\tilde{L}^a \overset{\{\mathrm{tr}\}_T}{=} \tilde{Q}^a - \tilde{\lambda}_a \tilde{\mathbf{Y}} - \tilde{\xi}_a \tilde{\mathbf{Z}}, \tag{10}$$

where the superscript a on $\tilde{L}$ and $\tilde{Q}$ indicates the derivative with respect to the perturbation strength a. The notation $\overset{\{\mathrm{tr}\}_T}{=}$ denotes a trace of matrix products on the right-hand side followed by an averaging over a time period T of the applied perturbations. The tildes are, hereafter, to denote quantities at general perturbation strengths $\{\varepsilon\}$ (instead of at zero perturbation strength), which are usually time dependent.

Other quantities on the right-hand side of Equation (10) are [3] the quasi-energy derivative

$$\tilde{Q}^a = \tilde{E}^{0,a} - \tilde{\mathbf{S}}^a \tilde{\mathbf{W}}, \tag{11}$$

the time-dependent self-consistent-field (TDSCF) equation

$$\tilde{\mathbf{Y}} = \tilde{\mathbf{F}}\tilde{\mathbf{D}}\tilde{\mathbf{S}} - \tilde{\mathbf{S}}\tilde{\mathbf{D}}\tilde{\mathbf{F}} - \tilde{\mathbf{S}}\left(\mathrm{i}\frac{\partial \tilde{\mathbf{D}}}{\partial t}\right)\tilde{\mathbf{S}} - \frac{1}{2}\left(\mathrm{i}\frac{\partial \tilde{\mathbf{S}}}{\partial t}\right)\tilde{\mathbf{D}}\tilde{\mathbf{S}} - \frac{1}{2}\tilde{\mathbf{S}}\tilde{\mathbf{D}}\left(\mathrm{i}\frac{\partial \tilde{\mathbf{S}}}{\partial t}\right), \tag{12}$$

the idempotency constraint

$$\tilde{\mathbf{Z}} = \tilde{\mathbf{D}}\tilde{\mathbf{S}}\tilde{\mathbf{D}} - \tilde{\mathbf{D}}, \tag{13}$$

and the Lagrangian multipliers $\tilde{\lambda}_a$ and $\tilde{\xi}_a$ that take the following ansatzes

$$\tilde{\lambda}_a = \tilde{\mathbf{D}}^a\tilde{\mathbf{S}}\tilde{\mathbf{D}} - \tilde{\mathbf{D}}\tilde{\mathbf{S}}\tilde{\mathbf{D}}^a, \tag{14}$$

$$\begin{aligned} \tilde{\xi}_a = {} & \tilde{\mathbf{F}}^a\tilde{\mathbf{D}}\tilde{\mathbf{S}} - \left[\tilde{\mathbf{F}}\tilde{\mathbf{D}} - \frac{1}{2}\left(\mathrm{i}\frac{\partial \tilde{\mathbf{S}}}{\partial t}\right)\tilde{\mathbf{D}} - \tilde{\mathbf{S}}\left(\mathrm{i}\frac{\partial \tilde{\mathbf{D}}}{\partial t}\right)\right]\tilde{\mathbf{S}}^a \\ & + \tilde{\mathbf{S}}\tilde{\mathbf{D}}\tilde{\mathbf{F}}^a - \tilde{\mathbf{S}}^a\left[\tilde{\mathbf{D}}\tilde{\mathbf{F}} + \frac{1}{2}\tilde{\mathbf{D}}\left(\mathrm{i}\frac{\partial \tilde{\mathbf{S}}}{\partial t}\right) + \left(\mathrm{i}\frac{\partial \tilde{\mathbf{D}}}{\partial t}\right)\tilde{\mathbf{S}}\right] - \tilde{\mathbf{F}}^a. \end{aligned} \tag{15}$$

Right-hand side terms of the quasi-energy derivative (11) are the generalized energy $\tilde{E}$, the overlap matrix $\tilde{\mathbf{S}}$ and the generalized energy-weighted density matrix $\tilde{\mathbf{W}}$, which reads [3]

$$\tilde{\mathbf{W}} = \tilde{\mathbf{D}}\tilde{\mathbf{F}}\tilde{\mathbf{D}} + \frac{1}{2}\left(\mathrm{i}\frac{\partial \tilde{\mathbf{D}}}{\partial t}\right)\tilde{\mathbf{S}}\tilde{\mathbf{D}} - \frac{1}{2}\tilde{\mathbf{D}}\tilde{\mathbf{S}}\left(\mathrm{i}\frac{\partial \tilde{\mathbf{D}}}{\partial t}\right). \tag{16}$$

The first superscript on the generalized energy $\tilde{E}$ indicates the order of derivatives with respect to the density matrix $\tilde{\mathbf{D}}$, and [3]

$$\begin{aligned} \tilde{E}^{0,a} = {} & \frac{\partial \tilde{E}}{\partial \varepsilon_{\omega_A}} \\ & \overset{\{\mathrm{tr}\}_T}{=} \left[\tilde{\mathbf{h}}^a + \tilde{\mathbf{V}}^a + \tilde{\mathbf{T}}^a + \frac{1}{2}\tilde{\mathbf{G}}^{\gamma,a}(\tilde{\mathbf{D}})\right]\tilde{\mathbf{D}} + \tilde{E}_{\mathrm{xc}}^{0,a}[\tilde{\rho}(\tilde{\mathbf{D}})] + \tilde{h}_{\mathrm{nuc}}^a + \tilde{v}_{\mathrm{nuc}}^a, \end{aligned} \tag{17}$$

where $\tilde{\mathbf{h}}$ is the one-electron Hamiltonian, and $\tilde{\mathbf{V}}$ the time-periodic perturbation operators. The $\tilde{\mathbf{T}}$ matrix arises due to the use of perturbation- and time-dependent basis sets $\{\tilde{\chi}\}$, and is defined as

$$\tilde{\mathbf{T}}_{\kappa\lambda} = -\frac{1}{2}\left[\left\langle \mathrm{i}\frac{\partial \tilde{\chi}_\kappa}{\partial t} \middle| \tilde{\chi}_\lambda \right\rangle + \left\langle \tilde{\chi}_\kappa \middle| \mathrm{i}\frac{\partial \tilde{\chi}_\lambda}{\partial t} \right\rangle\right], \tag{18}$$

which is different from that of Reference 3, as we have collected the imaginary number $-\frac{\mathrm{i}}{2}$ and the time differentiation together so that the $\tilde{\mathbf{T}}$ matrix here is Hermitian.

The operator $\tilde{\mathbf{G}}^{\gamma}(\tilde{\mathbf{D}})$ is the two-electron matrix with scaled (scaling factor is γ) exchange [3]

$$\tilde{\mathbf{G}}^{\gamma}_{\mu\nu}(\tilde{\mathbf{D}}) = \sum_{\kappa\lambda} \tilde{\mathbf{D}}_{\lambda\kappa}\left[\tilde{g}_{\mu\nu\kappa\lambda} - \gamma \tilde{g}_{\mu\lambda\kappa\nu}\right], \tag{19}$$

$$\tilde{g}_{\mu\nu\kappa\lambda} = \int\int \tilde{\chi}^*_\mu(\boldsymbol{r}_1)\tilde{\chi}_\nu(\boldsymbol{r}_1)\frac{1}{r_{12}}\tilde{\chi}^*_\kappa(\boldsymbol{r}_2)\tilde{\chi}_\lambda(\boldsymbol{r}_2)\mathrm{d}\boldsymbol{r}_1\mathrm{d}\boldsymbol{r}_2. \tag{20}$$

Within the adiabatic approximation, the XC functional $\tilde{E}_{\mathrm{xc}}[\tilde{\rho}(\tilde{\mathbf{D}})]$ depends on time only through the electron density and its Cartesian gradient if using generalized gradient approximation and hybrid functionals [16]

$$\tilde{E}_{\mathrm{xc}}[\tilde{\rho}(\tilde{\mathbf{D}})] = \int \tilde{\epsilon}_{\mathrm{xc}}[\tilde{\rho}(\boldsymbol{r})]\mathrm{d}\boldsymbol{r}, \tag{21}$$

where $\tilde{\epsilon}_{\mathrm{xc}}[\tilde{\rho}(\boldsymbol{r})]$ is a local function of the XC energy density. The notation $\tilde{\rho}(\boldsymbol{r})$ represents a generalized density vector that collects the electron density $\tilde{n}(\boldsymbol{r})$ and/or its Cartesian gradient $\nabla\tilde{n}(\boldsymbol{r})$ [16]

$$\tilde{n}(\boldsymbol{r}) \overset{\{\mathrm{tr}\}_T}{=} \tilde{\boldsymbol{\Omega}}(\boldsymbol{r})\tilde{\mathbf{D}}, \tag{22}$$

$$\nabla\tilde{n}(\boldsymbol{r}) \overset{\{\mathrm{tr}\}_T}{=} [\nabla\tilde{\boldsymbol{\Omega}}(\boldsymbol{r})]\tilde{\mathbf{D}}, \tag{23}$$

where the overlap distribution $\tilde{\boldsymbol{\Omega}}(\boldsymbol{r})$ is

$$\tilde{\boldsymbol{\Omega}}_{\mu\nu}(\boldsymbol{r}) = \tilde{\chi}^*_\mu(\boldsymbol{r})\tilde{\chi}_\nu(\boldsymbol{r}). \tag{24}$$

The last two terms $\tilde{h}_{\mathrm{nuc}}$ and $\tilde{v}_{\mathrm{nuc}}$ in the generalized energy (17) are respectively the nuclear repulsion energy and interaction between perturbations and nuclei [11] that do not depend on the density matrix $\tilde{\mathbf{D}}$.

Finally, we have the generalized Fock matrix $\tilde{\mathbf{F}}$, which can be computed as the first derivative of the generalized energy $\tilde{E}$ with respect to the density matrix $\tilde{\mathbf{D}}$,

$$\tilde{\mathbf{F}} = \tilde{E}^{1,0} = \frac{\partial\tilde{E}}{\partial\tilde{\mathbf{D}}^{\mathsf{T}}} = \tilde{\mathbf{h}} + \tilde{\mathbf{V}} + \tilde{\mathbf{T}} + \tilde{\mathbf{G}}^{\gamma}(\tilde{\mathbf{D}}) + \tilde{\mathbf{F}}_{\mathrm{xc}}(\tilde{\rho}), \tag{25}$$

and $\tilde{\mathbf{F}}_{\mathrm{xc}}(\tilde{\rho})$ is the density functional derivative matrix [3,16]

$$(\tilde{\mathbf{F}}_{\mathrm{xc}})_{\mu\nu}(\tilde{\rho}) = \int \left.\frac{\delta\tilde{\epsilon}_{\mathrm{xc}}[\tilde{\rho}(\boldsymbol{r})]}{\delta\rho}\right|_{\rho=\tilde{\rho}} \frac{\partial\tilde{\rho}}{\partial\tilde{\mathbf{D}}_{\nu\mu}}\mathrm{d}\boldsymbol{r} = \int \tilde{v}_{\mathrm{xc}}[\tilde{\rho}(\boldsymbol{r})](\tilde{\boldsymbol{\Omega}}_\rho)_{\mu\nu}(\boldsymbol{r})\mathrm{d}\boldsymbol{r}, \tag{26}$$

where $\tilde{v}_{\mathrm{xc}}[\tilde{\rho}(\boldsymbol{r})]$ is the XC potential, and $\tilde{\boldsymbol{\Omega}}_\rho(\boldsymbol{r})$ the generalized overlap distribution [16] that collects the overlap distribution $\tilde{\boldsymbol{\Omega}}(\boldsymbol{r})$ and its Cartesian gradient $\nabla\tilde{\boldsymbol{\Omega}}(\boldsymbol{r})$.

In the $n+1$ formulation, the molecular response functions can be computed as the nth order perturbation-strength derivatives of the time-averaged quasi-energy derivative Lagrangian (10) and evaluated at zero perturbation strength [3]

$$\langle\langle\hat{A};\hat{B}_1,\ldots,\hat{B}_n\rangle\rangle_{\omega_{B_1},\ldots,\omega_{B_n}} = \tilde{L}^{ab_1\cdots b_n}\Big|_{\{\varepsilon\}=0}. \tag{27}$$

The key quantities to be solved for the computation of Equation (27) are derivatives of the density matrix in the frequency domain $\mathbf{D}^{b_1}_{\omega_{B_1}}$, $\mathbf{D}^{b_1b_2}_{\omega_{B_1}\omega_{B_2}}$, …, $\mathbf{D}^{b_1\cdots b_n}_{\omega_{B_1}\cdots\omega_{B_n}}$, whose solutions can be obtained from the TDSCF equation $\tilde{\mathbf{Y}}=0$ and the idempotency constraint $\tilde{\mathbf{Z}}=0$. Interested readers are referred to Reference 3 because there are no new basic symbols from those solutions to be presented here.

From the above formulas for the determination of molecular response functions, the following basic symbols and symbolic operations can be identified for the density matrix-based response theory:

1. Perturbations,
2. Electronic state, such as the (perturbed) density matrix,
3. One-electron like operators,
4. Two-electron like operators,
5. XC like energies and potentials,
6. Non-electron like functions, such as the nuclear repulsion energy $\tilde{h}_{\mathrm{nuc}}$ and interaction between perturbations and nuclei $\tilde{v}_{\mathrm{nuc}}$,
7. $\mathrm{i}\frac{\partial}{\partial t}$ and the $\tilde{\mathbf{T}}$ matrix in Equation (18),
8. Symbolic differentiation, arithmetic and trace,
9. Symbolic removal and replacement, for example, used for solving the derivatives of the density matrix in the frequency domain and the computation of residues.[3]

Coupled-cluster response theory

In coupled-cluster response theory, molecular response functions can be determined from the perturbation-strength derivatives of the time average of the following quasi-energy Lagrangian [2,10]

$$L(t) = \langle \mathrm{e}^{-\hat{T}(t)}\hat{H}(t)\mathrm{e}^{\hat{T}(t)}\rangle + \sum\mu\tilde{\lambda}\mu\left\langle\hat{\boldsymbol{\tau}}^\dagger_\mu\mathrm{e}^{-\hat{T}(t)}\left[\hat{H}(t)-\mathrm{i}\frac{\partial}{\partial t}\right]\mathrm{e}^{\hat{T}(t)}\right\rangle, \tag{28}$$

where notations $\langle\hat{A}\rangle$ and $\langle\hat{A}\rangle_\mu$ represent expectation values of an operator $\hat{A}$

$$\langle\hat{A}\rangle = \langle\mathrm{HF}|\hat{A}|\mathrm{HF}\rangle, \tag{29}$$

$$\langle\hat{A}\rangle_\mu = \langle\mathrm{HF}|\hat{\boldsymbol{\tau}}^\dagger_\mu\hat{A}|\mathrm{HF}\rangle, \tag{30}$$

with $|\mathrm{HF}\rangle$ being the Hartree-Fock reference state. The time-dependent cluster operator $\hat{T}(t)$ is

$$\hat{T}(t) = \sum_\nu \tilde{\mathbf{t}}_\nu\hat{\boldsymbol{\tau}}_\nu, \tag{31}$$

where the row vectors $\hat{\boldsymbol{\tau}}$ and $\tilde{\mathbf{t}}$ contain respectively (commuting) excitation operators and time-dependent coupled-cluster amplitudes. The row vector $\tilde{\boldsymbol{\lambda}}$ is introduced as the Lagrangian multipliers.

Key quantites for evaluating response functions are the derivatives of the coupled-cluster amplitudes $\boldsymbol{t}^{b_1\cdots b_n}_{\omega_{B_1}\cdots\omega_{B_n}}$ and the Lagrangian multipliers $\boldsymbol{\lambda}^{b_1\cdots b_n}_{\omega_{B_1}\cdots\omega_{B_n}}$ in the frequency domain. They can be solved respectively from [10,17]

$$\frac{\partial}{\partial\boldsymbol{\lambda}^{b_{n+1}\cdots b_{n+m}}_{\omega,\mu}}\left(\tilde{L}^{b_1\cdots b_{n+m}}\Big|_{\{\varepsilon\}=0}\right) = 0, \tag{32}$$

and

$$\frac{\partial}{\partial\boldsymbol{t}^{b_{n+1}\cdots b_{n+m}}_{\omega,\nu}}\left(\tilde{L}^{b_1\cdots b_{n+m}}\Big|_{\{\varepsilon\}=0}\right) = 0, \tag{33}$$

where $\tilde{L}^{b_1\cdots b_{n+m}}$ is the derivatives of the time average of the quasi-energy Lagrangian (28)

$$\tilde{L}^{b_1\cdots b_{n+m}} = \frac{\partial^{n+m}\{L(t)\}_T}{\partial\varepsilon_{\omega_{B_1}}\cdots\partial\varepsilon_{\omega_{B_{n+m}}}}. \tag{34}$$

Therefore, all basic symbols for coupled-cluster response theory can be recognized by analysing terms of the quasi-energy Lagrangian (28) and their perturbation-strength derivatives. To facilitate our following discussion, we introduce the "exponential map" from Lie algebra [18]

$$\begin{aligned} e^{\mathrm{ad}_X}(Y) &= Y + \mathrm{ad}_X(Y) + \frac{1}{2!}(\mathrm{ad}_X)^2(Y) + \cdots \\ &= Y + [X,Y] + \frac{1}{2!}[X,[X,Y]] + \cdots \\ &= e^{X} Y e^{-X}, \end{aligned} \tag{35}$$

where $\mathrm{ad}_X(Y) = [X,Y]$ is the adjoint map (or adjoint representation).[18]

By using the exponential map, we can rewrite the quasi-energy Lagrangian (28) as

$$L(t) = \langle e^{\mathrm{ad}_{-\hat{T}(t)}}(\hat{H}(t))\rangle + \sum_{\mu} \tilde{\lambda}_{\mu} \left[\langle e^{\mathrm{ad}_{-\hat{T}(t)}}(\hat{H}(t))\rangle_{\mu} - \mathrm{i}\frac{\partial}{\partial t}\tilde{t}_{\mu} \right], \tag{36}$$

from which we can easily recognize the following new basic symbols:

1. Coupled-cluster amplitudes $\tilde{t}$,
2. Lagrangian multipliers $\tilde{\lambda}$, and
3. Expectation values of the exponential map $e^{\mathrm{ad}_{-\hat{T}(t)}}(\hat{H}(t))$.

Derivatives of the operator $e^{\mathrm{ad}_{-\hat{T}(t)}}(\hat{H}(t))$ can be determined as follows: we first note that the Hamiltonian $\hat{H}(t)$ has no more than two-body interactions, so the expansion of $e^{\mathrm{ad}_{-\hat{T}(t)}}(\hat{H}(t))$ can be terminated after fourfold commutators,[19]

$$e^{\mathrm{ad}_{-\hat{T}(t)}}(\hat{H}(t)) = \sum_{j=0}^{4} \frac{1}{j!} \left(\mathrm{ad}_{-\hat{T}(t)}\right)^j (\hat{H}(t)) = \sum_{j=0}^{4} \frac{(-1)^j}{j!} \left(\mathrm{ad}_{\hat{T}(t)}\right)^j (\hat{H}(t)). \tag{37}$$

Secondly, due to the cluster operator $\hat{T}_{\omega}$ and its different derivatives are commutative, we have for $j > 0$,

$$\begin{aligned} \left[\left(\mathrm{ad}_{\hat{T}}\right)^j(\hat{H})\right]_{\omega}^{b_1\cdots b_n} &= \left(\mathrm{ad}_{\hat{T}^{(0)}}\right)^j (\hat{H}_{\omega}^{b_1\cdots b_n}) \\ &+ \sum_{\substack{P \subsetneq \{1,\cdots,n\} \\ Q=\{1,\cdots,n\}-P}} \sum_{k=1}^{\min(j,|Q|)} \sum_{\pi \in \mathcal{P}_{Q,k}} \frac{j!}{(j-k)!} \prod_{q=1}^{j} \left(\mathrm{ad}_{\hat{T}_{\omega}^{b_{\pi_q}}}\right)(\hat{H}_{\omega}^{b_p}), \end{aligned} \tag{38}$$

where the subscript ω indicates quantities in the frequency domain, and $\mathcal{P}_{Q,k}$ stands for all set partitions of the set Q with k blocks. The notation π_q represents the qth block of π, and $\pi_q = \emptyset$ when $q > k$. The operator $\hat{T}^{(0)}$ is defined as

$$\hat{T}^{(0)} = \sum_{\nu} t_{\nu}^{(0)} \hat{\tau}_{\nu}, \tag{39}$$

with $t^{(0)}$ the optimized zeroth-order coupled-cluster amplitudes.

By using Equations (37) and (38), we can get

$$\begin{aligned} \left[e^{\mathrm{ad}_{-\hat{T}(t)}}(\hat{H}(t))\right]_{\omega}^{b_1\cdots b_n} &= e^{\mathrm{ad}_{-\hat{T}^{(0)}}}(\hat{H}_{\omega}^{b_1\cdots b_n}) \\ &+ \sum_{\substack{P \subsetneq \{1,\cdots,n\} \\ Q=\{1,\cdots,n\}-P}} \sum_{k=1}^{\min(4,|Q|)} \sum_{\pi \in \mathcal{P}_{Q,k}} \sum_{j=k}^{4} \frac{(-1)^j}{(j-k)!} \prod_{q=1}^{j} \left(\mathrm{ad}_{\hat{T}_{\omega}^{b_{\pi_q}}}\right)(\hat{H}_{\omega}^{b_p}) \\ &= e^{\mathrm{ad}_{-\hat{T}^{(0)}}}(\hat{H}_{\omega}^{b_1\cdots b_n}) \\ &+ \sum_{\substack{P \subsetneq \{1,\cdots,n\} \\ Q=\{1,\cdots,n\}-P \\ |Q| \geq 4}} \sum_{\pi \in \mathcal{P}_{Q,4}} \prod_{q=1}^{4} \left(\mathrm{ad}_{\hat{T}_{\omega}^{b_{\pi_q}}}\right)(\hat{H}_{\omega}^{b_p}) \\ &+ \sum_{k=1}^{3} \sum_{\substack{P \subsetneq \{1,\cdots,n\} \\ Q=\{1,\cdots,n\}-P \\ |Q| \geq k}} \sum_{\pi \in \mathcal{P}_{Q,k}} (-1)^k e^{\mathrm{ad}_{-\hat{T}^{(0)}}} \left(\prod_{q=1}^{k} \left(\mathrm{ad}_{\hat{T}_{\omega}^{b_{\pi_q}}}\right)(\hat{H}_{\omega}^{b_p}) \right), \end{aligned} \tag{40}$$

where

$$\hat{T}_{\omega}^{b_{\pi_q}} = \sum_{\nu} t_{\omega,\nu}^{b_{\pi_q}} \hat{\tau}_{\nu}. \tag{41}$$

SYMBOLIC COMPUTATIONS—DESIGN AND IMPLEMENTATION FOR RESPONSE THEORY

All our implementation can be found in a new libray named as Tinned,[9] which is built on a modified version of SymEngine library (available at `https://github.com/bingao/symengine`). SymEngine [20] is a standalone fast C++ symbolic manipulation library, and is released under a permissive open source license. Our modified version extends symbolic differentiation to different matrix expressions, such as matrix addition, matrix multiplication, trace and in particular, a class `MatrixSymbol`—from which different electron operators can be derived.

Figure 1 illustrates relationships of classes implemented in Tinned library and their base classes in SymEngine library by using unified modeling language (UML) class diagram. To summarize, all classes developed in Tinned library can be categorized into four groups:

1. The class `Perturbation` derived from the class `Symbol` in SymEngine,
2. Classes derived from the class `FunctionWrapper` to represent scalar functions and functionals,
3. Classes derived from the class `MatrixSymbol` for electron operators, and
4. Different "visitor" classes for symbolic operations like symbolic replacement and removal.

In the following subsections we will discuss these classes in detail, including their design, implementation and application programming interfaces (APIs)—especially those for symbolic differentiation.

Symbolic arithmetic, differentiation and matrix operations

Before presenting our implementation of different electron operators, we will first describe various symbolic operations in

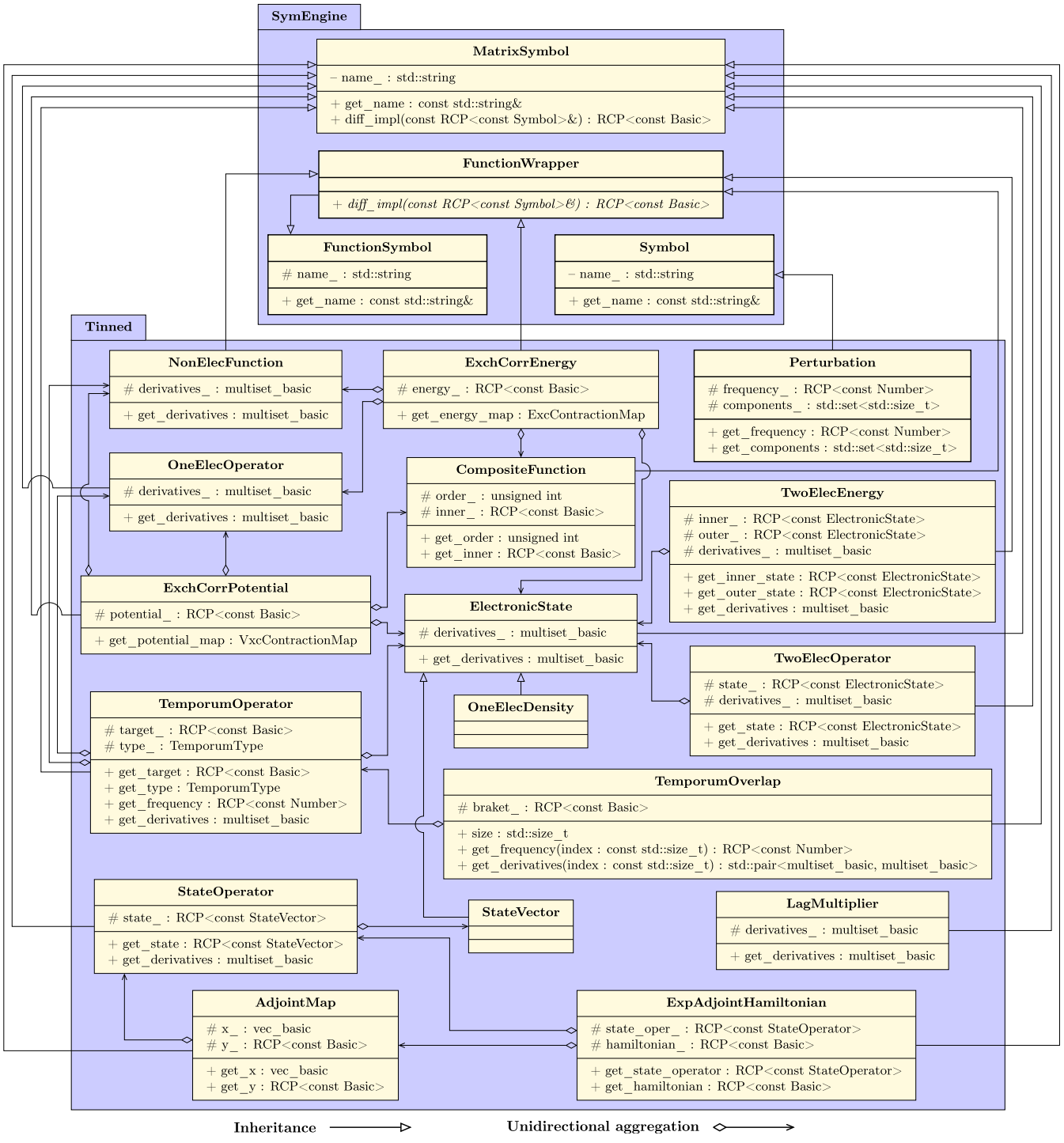

FIGURE 1 Relationships of classes implemented in our Tinned library and their base classes in SymEngine library. Key member variables and functions are highlighted for each class, which are essential for our design and the use of Tinned library. Notations +, # and – respectively represent public, protected and private members of a class. Types of class members–`TemporumType`, `ExcContractionMap` and `VxcContractionMap` are defined in Tinned library and are described in sections of time differentiation i $\frac{\partial}{\partial t}$, and XC energy and potential. Other types are from the C++ standard and SymEngine libraries. The former starts with `std`. The latter includes `RCP` (shared pointer), `Basic` (base class for all types of symbolic representations), `Number` (base class for all types of numbers), `Symbol` (representing symbols or variables in mathematics), `multiset_basic` (multiset of objects of type `Basic`) and `vec_basic` (vector of objects of type `Basic`).

SymEngine that are useful for response theory. In SymEngine, all types of symbolic representations are derived from a base class named as `Basic`, including mathematical symbols, symbolic arithmetic, different mathematicl functions (like trigonometric functions and exponential function), derivatives and so forth.

Matrix expressions and operations are different from scalar symbolic ones. A class `MatrixExpr` (also derived from `Basic`) has been introduced in SymEngine that serves as the base class for all matrix expressions and operations, including but not limited to matrix addition, matrix multiplication and trace. Built on top of SymEngine library, we do not need to reinvent the wheel for different symbolic operations. As aforementioned, we only need to implement symbolic differentiation for different matrix classes. In SymEngine, the symbolic differentiation is realized by using the visitor design pattern.[21]

In Figure 2, we illustrate the visitor pattern for symbolic differentiation in SymEngine and Tinned libraries by using UML class diagram. Every symbolic class derived from the class `Basic` has a member function `accept` that takes an object "v" of the base class `Visitor` as input—obviously, any object of a derived visitor class from `Visitor` can also be the input. The function `accept` "dispatches" the object of the current symbolic class to the visitor "v" by calling its member function `visit` as shown in Figure 2. The visitor object "v" can then perform appropriate operation on the object of the symbolic class.

The visitor design pattern is beneficial when new visitors need to be implemented since no change will be made to symbolic classes. But it can be troublesome when new symbolic classes are introduced as they usually require new `visit` functions. The class `FunctionWrapper` is special in SymEngine. Users can implement their own classes by deriving from it and the member function `diff_impl` will be invoked by the object of the visitor class `DiffVisitor` for symbolic differentiation. As shown in Figures 1 and 2, classes `NonElecFunction` (non-electron like functions), `CompositeFunction` (composite functions) and `ExchCorrEnergy` (XC energy) in Tinned library are all derived from the class `FunctionWrapper`.

In our modified version of SymEngine library, we have also introduced such a member function `diff_impl` for the class `MatrixSymbol`, which can be overridden by derived electron operator classess for their symbolic differentiation.

As shown in Figure 1, derivatives can be readily extracted by using the member function `get_derivatives` of most classes in Tinned library. Exceptions, like the class `ExchCorrEnergy`, will be explained separately in the following subsections.

Perturbations

The class `Perturbation` is derived from the class `Symbol` in SymEngine to represent different perturbations, and objects of the class `Symbol` are distinguished by their names as shown in Figure 1.

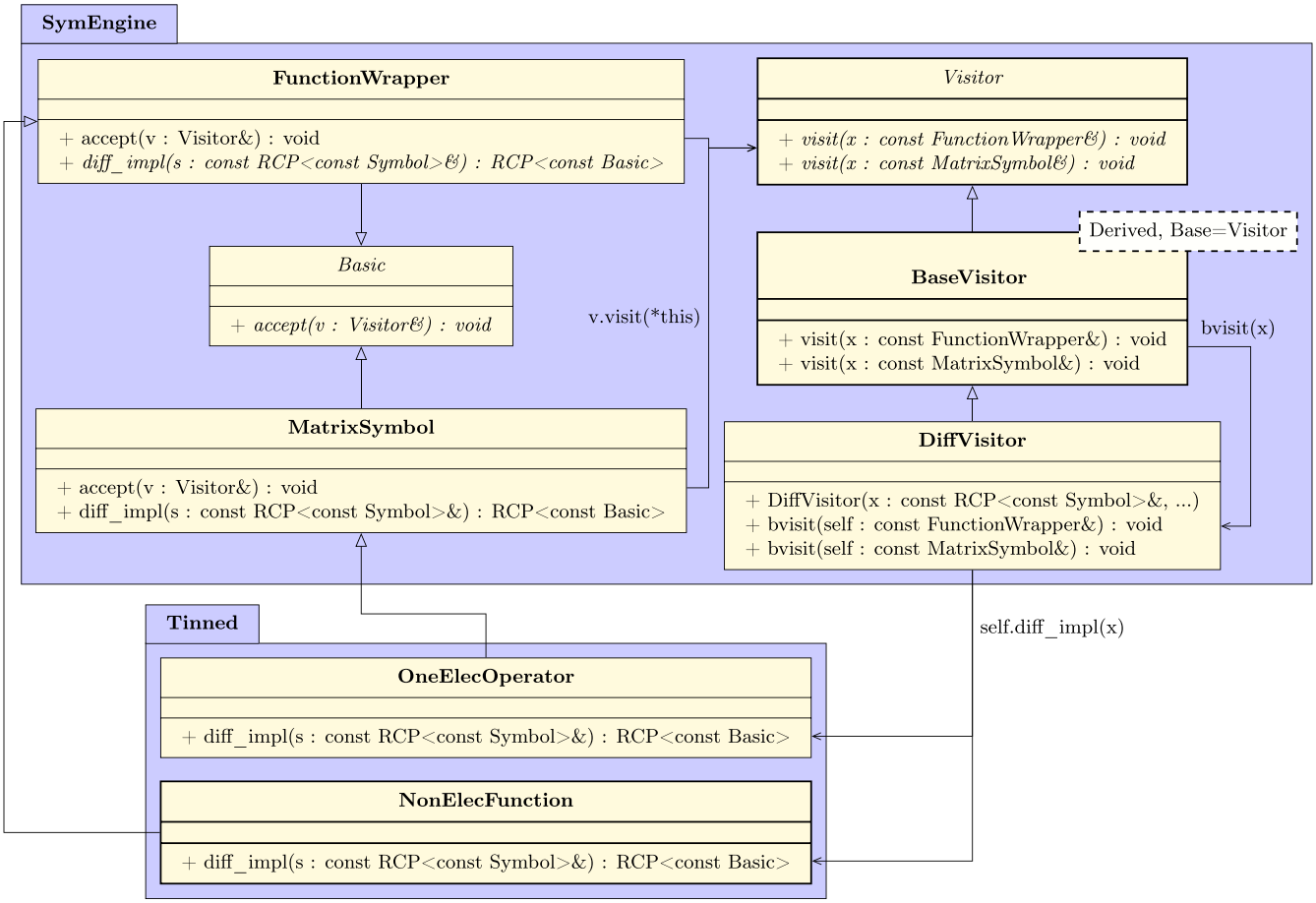

FIGURE 2 UML class diagram for the visitor design pattern of symbolic differentiation in SymEngine and Tinned libraries. Note that classes `FunctionWrapper` and `MatrixSymbol` are further-derived classes of the class `Basic`.

We have further introduced two additional members in the class `Perturbation`: (i) an object of the class `Number` from SymEngine for the frequency of a perturbation, and (ii) a set of integers starting from 0 for components of the perturbation. Default values are 0 for the frequency (a static perturbation) and an empty set for components meaning all components will be considered.

The set of components can be useful if one is interested in specific components of a perturbation. For example, a set {0,2} means only *x* and *z* components will be considered for an electric-field perturbation.

In Tinned library, the class `Perturbation` is mostly used as a variable that other operators and functions can be differentiated with respect to. Although it is not depicted in Figure 1, classes `NonElecFunction`, `OneElecOperator`, `TwoElecEnergy` and `TwoElecOperator` all have a member variable that records their dependencies on different `Perturbation` objects and corresponding maximum orders of differentiation.

Non-electron like functions and one-electron like operators

Non-electron like functions and one-electron like operators are respectively represented by classes `NonElecFunction` and `OneElecOperator` in Tinned. As shown in Figure 1, the former is derived from the class `FunctionWrapper` and the latter from the class `MatrixSymbol`.

The procedure of the member function `diff_impl` is exactly the same for both `NonElecFunction` and `OneElecOperator`. In Figure 3, we illustrate the procedure of differentiation for the class `NonElecFunction` by using UML sequence diagram. Taking into account the perturbation dependencies, we have introduced a type `PertDependency` in Tinned, which is a C++ `std::map` container with each key-value pair as a `Perturbation` object and its corresponding maximum order for differentiation.

Except for perturbation dependencies, both `NonElecFunction` and `OneElecOperator` have a member variable `derivatives_` holding

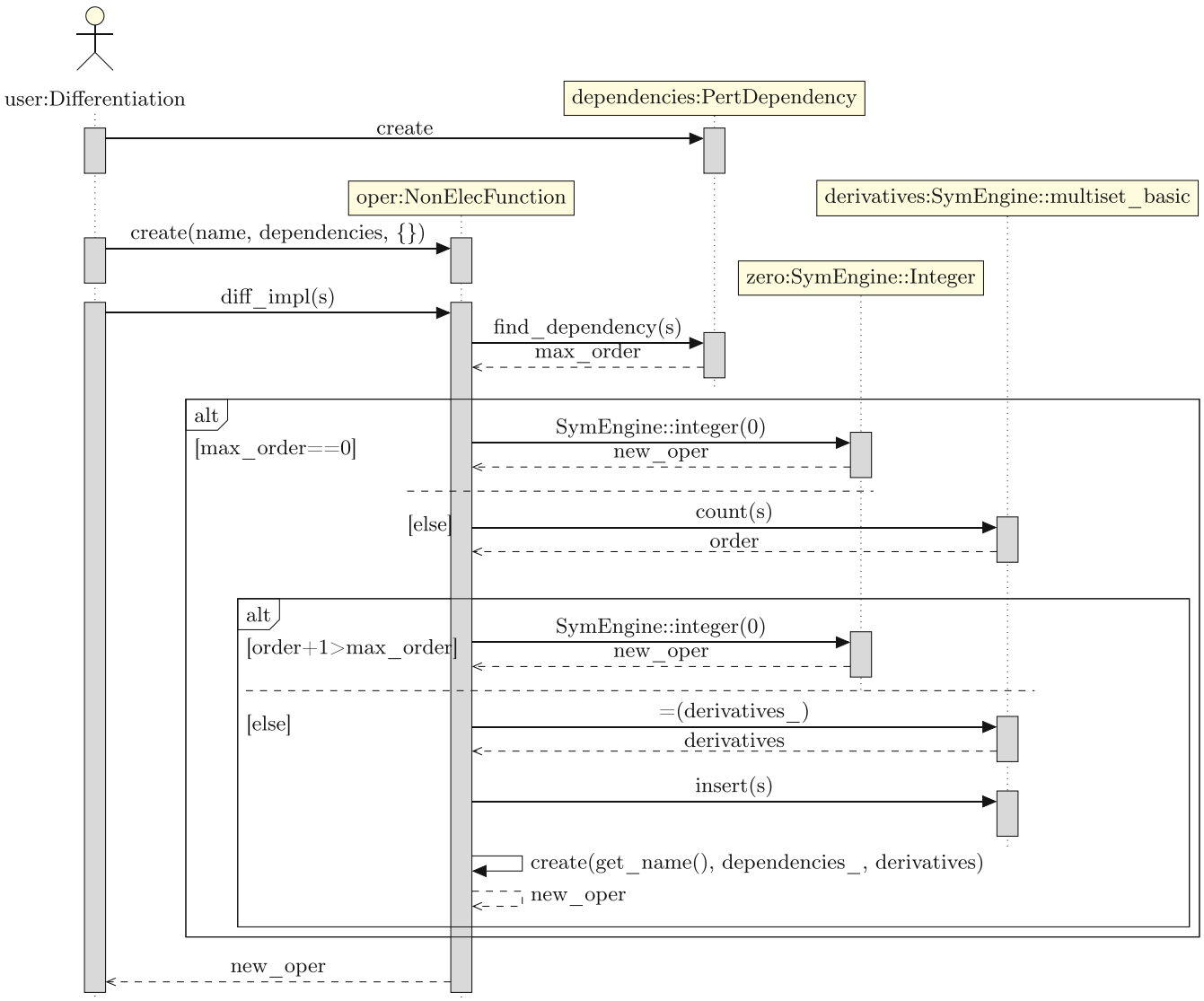

FIGURE 3 UML sequence diagram for the differentiation of the class `NonElecFunction`.

derivatives with respect to perturbations. The variable `derivatives_` stores objects of the class `Perturbation` with respect to which the instance of `NonElecFunction` or `OneElecOperator` will be differentiated. A C++ `std::multiset` container is used for storing these `Perturbation` objects, and equivalent objects are allowed for differentiating as many times as occurrences of the object. The use of `std::multiset` container guarantees that the variable `derivatives_` is a valid perturbation tuple.

When constructing an instance of `NonElecFunction`, the default value for `derivatives_` is an empty container "{}" — representing an undifferentiated instance as shown in Figure 3. When the function `diff_impl` is called to perform differentiation with respect to an object "s" of the type `Symbol`, the function `find_dependency` will first be invoked to return the corresponding maximum order of "s" for differentiation if it exists in the container of perturbation dependencies, 0 otherwise.

For non-zero maximum order of differentiation, we further check the number of occurrences of the object "s" in the member variable `derivatives_` and compare it with the maximum order. For zero derivatives, a symbolic integer zero from SymEngine will be returned from the function `diff_impl`. For non-zero derivatives, a new instance of `NonElecFunction` will be created and returned, which contains derivatives with the object "s" appended.

Electronic state

Notice that response theory calculations can be performed at different theoretical levels, which involve different quantities for the electronic state, such as density matrix, molecular orbital coefficients and coupled-cluster amplitudes. So we have first implemented a base class `ElectronicState` for all different concrete and derived electronic state classes.

The class `ElectronicState` is derived from the class `MatrixSymbol` as shown in Figure 1. Derivatives with respect to perturbations are stored in the member variable `derivatives_`, while derived electronic state classes will implement their own member function `diff_impl` for symbolic differentiation.

So far we have implemented derived classes `OneElecDensity` and `StateVector`, respectively for one-electron spin-orbital density matrix and coupled-cluster amplitudes. Their member function `diff_impl` can be trivially implemented by returning a new `OneElecDensity` or `StateVector`instance with the object "s" appended to the member variable `derivatives_`.

Two-electron like operators and energy

From Equation (20), any two-electron like operator $\tilde{\mathbf{G}}$ can be viewed as a tensor contraction of electron repulsion integrals (ERIs) $\tilde{\mathbf{g}}$ and one-electron spin-orbital density matrix $\tilde{\mathbf{D}}$

$$\tilde{\mathbf{G}}(\tilde{\mathbf{D}}) = \tilde{\mathbf{g}} \otimes \tilde{\mathbf{D}}, \tag{42}$$

and any two-electron like energy becomes

$$\tilde{E}_{2\mathrm{el}} = \mathrm{tr}\left[\tilde{\mathbf{G}}(\tilde{\mathbf{D}}_{\mathrm{inner}})\tilde{\mathbf{D}}_{\mathrm{outer}}\right] = \mathrm{tr}\left[\left(\tilde{\mathbf{g}} \otimes \tilde{\mathbf{D}}_{\mathrm{inner}}\right)\tilde{\mathbf{D}}_{\mathrm{outer}}\right]. \tag{43}$$

In Tinned, we have introduced classes `TwoElecOperator` and `TwoElecEnergy` for the operator $\tilde{\mathbf{G}}$ and the energy $\tilde{E}_{2\mathrm{el}}$, respectively. As shown in Figure 1, they are respectively derived from the class `MatrixSymbol` and the class `FunctionWrapper` in SymEngine.

Member variables `state_`, `inner_` and `outer_` store the one-electron spin-orbital density matrices $\tilde{\mathbf{D}}$, $\tilde{\mathbf{D}}_{\mathrm{inner}}$ and $\tilde{\mathbf{D}}_{\mathrm{outer}}$, respectively. Both `TwoElecOperator` and `TwoElecEnergy` also have member variables `dependencies_` and `derivatives_` for perturbation dependencies and derivatives of the ERIs $\tilde{\mathbf{g}}$.

The differentiation of $\tilde{\mathbf{G}}$ and $\tilde{E}_{2\mathrm{el}}$ with respect to a perturbation "s" can be expressed as

$$\frac{\partial \tilde{\mathbf{G}}}{\partial \mathrm{s}} = \tilde{\mathbf{g}}^{\mathrm{s}} \otimes \tilde{\mathbf{D}} + \tilde{\mathbf{g}} \otimes \tilde{\mathbf{D}}^{\mathrm{s}}, \tag{44}$$

$$\begin{aligned}\frac{\partial \tilde{E}_{2\mathrm{el}}}{\partial \mathrm{s}} &= \mathrm{tr}\left[\left(\tilde{\mathbf{g}}^{\mathrm{s}} \otimes \tilde{\mathbf{D}}_{\mathrm{inner}}\right)\tilde{\mathbf{D}}_{\mathrm{outer}}\right] + \mathrm{tr}\left[\left(\tilde{\mathbf{g}} \otimes \tilde{\mathbf{D}}^{\mathrm{s}}_{\mathrm{inner}}\right)\tilde{\mathbf{D}}_{\mathrm{outer}}\right] \\ &+ \mathrm{tr}\left[\left(\tilde{\mathbf{g}} \otimes \tilde{\mathbf{D}}_{\mathrm{inner}}\right)\tilde{\mathbf{D}}^{\mathrm{s}}_{\mathrm{outer}}\right],\end{aligned} \tag{45}$$

from which their member function `diff_impl` can be developed as the sum of those right-hand-side terms. Derivatives $\tilde{\mathbf{D}}^{\mathrm{s}}$, $\tilde{\mathbf{D}}^{\mathrm{s}}_{\mathrm{inner}}$ and $\tilde{\mathbf{D}}^{\mathrm{s}}_{\mathrm{outer}}$ can be computed from the member function `diff_impl` of the class `OneElecDensity`. Derivatives $\tilde{\mathbf{g}}^{\mathrm{s}}$ can be performed by following the same procedure as that of the class `NonElecFunction` in Figure 3.

Furthermore, the following relationship [3] has been used to collect equivalent terms for the differentiation of $\tilde{E}_{2\mathrm{el}}$,

$$\mathrm{tr}\left[\left(\tilde{\mathbf{g}} \otimes \tilde{\mathbf{D}}_{\mathrm{inner}}\right)\tilde{\mathbf{D}}_{\mathrm{outer}}\right] = \mathrm{tr}\left[\left(\tilde{\mathbf{g}} \otimes \tilde{\mathbf{D}}_{\mathrm{outer}}\right)\tilde{\mathbf{D}}_{\mathrm{inner}}\right]. \tag{46}$$

XC energy and potential

In practice, the XC functional energy and potential matrix are evaluated by using numerical integration techniques [16]

$$\tilde{E}_{\mathrm{xc}} \approx \sum_i w_i \tilde{\epsilon}_{\mathrm{xc}}[\tilde{\rho}(\boldsymbol{r}_i)], \tag{47}$$

$$(\tilde{\mathbf{F}}_{\mathrm{xc}})_{\mu\nu} \approx \sum_i w_i \tilde{v}_{\mathrm{xc}}[\tilde{\rho}(\boldsymbol{r}_i)](\tilde{\boldsymbol{\Omega}}_{\rho})_{\mu\nu}(\boldsymbol{r}_i), \tag{48}$$

where w_i's and $\boldsymbol{r}_i$'s are grid weights and grid points.

In Tinned, we have implemented classes `ExchCorrEnergy` and `ExchCorrPotential` to represent the XC functional energy and potential matrix, respectively. As illustrated in Figure 1, their member variables `energy_` and `potential_` are used to store the above summand of $\tilde{E}_{\mathrm{xc}}$ and $(\tilde{\mathbf{F}}_{\mathrm{xc}})_{\mu\nu}$, i.e.

$$\texttt{energy} = w\tilde{\epsilon}_{\text{xc}}[\tilde{\rho}(\boldsymbol{r})], \tag{49}$$

$$\texttt{potential} \ = w\tilde{v}_{\text{xc}}[\tilde{\rho}(\boldsymbol{r})]\tilde{\boldsymbol{\Omega}}_{\rho}(\boldsymbol{r}) = w\frac{\partial\tilde{\epsilon}_{\text{xc}}[\tilde{\rho}(\boldsymbol{r})]}{\partial\tilde{\rho}(\boldsymbol{r})}\tilde{\boldsymbol{\Omega}}_{\rho}(\boldsymbol{r}), \tag{50}$$

where the subscript i is omitted, and $\tilde{\rho}(\boldsymbol{r}) = \text{tr}\left[\tilde{\boldsymbol{\Omega}}_{\rho}(\boldsymbol{r})\tilde{\mathbf{D}}\right]$.

More explicitly, the grid weight w, the generalized overlap distribution [16] $\tilde{\boldsymbol{\Omega}}_{\rho}(\boldsymbol{r})$ and the one-electron spin-orbital density matrix $\tilde{\mathbf{D}}$ are respectively instances of the classes `NonElecFunction`, `OneElecOperator` and `ElectronicState`. The generalized density vector $\tilde{\rho}(\boldsymbol{r})$ is simply the trace of product of $\tilde{\boldsymbol{\Omega}}_{\rho}(\boldsymbol{r})$ and $\tilde{\mathbf{D}}$. The local function of the XC energy density $\tilde{\epsilon}_{\text{xc}}[\tilde{\rho}(\boldsymbol{r})]$ and the XC potential $\tilde{v}_{\text{xc}}[\tilde{\rho}(\boldsymbol{r})]$ can be treated as composite functions of the generalized density vector $\tilde{\rho}(\boldsymbol{r})$. We further use the class `CompositeFunction` in Tinned for $\tilde{\epsilon}_{\text{xc}}[\tilde{\rho}(\boldsymbol{r})]$ and $\tilde{v}_{\text{xc}}[\tilde{\rho}(\boldsymbol{r})]$, with the member variables `order_` respectively as 0 and 1, and `inner_` as the instance of $\tilde{\rho}(\boldsymbol{r})$.

Next, we consider the implementation of the member function `diff_impl` for `ExchCorrEnergy` and `ExchCorrPotential`. The member variables `energy_` and `potential_` are respectively expressed as multiplication and matrix multiplication in SymEngine. Their derivatives with respect to a perturbation "s" can be performed by using the corresponding differentiation functions in SymEngine, and the only requisite we need to implement is the differentiation of the class `CompositeFunction`. From the knowledge of derivatives of the composition of two functions,[22] we have

$$\frac{\partial\tilde{\epsilon}_{\text{xc}}[\tilde{\rho}(\boldsymbol{r})]}{\partial \text{s}} = \frac{\partial\tilde{\epsilon}_{\text{xc}}[\tilde{\rho}(\boldsymbol{r})]}{\partial\tilde{\rho}(\boldsymbol{r})}\left(\text{tr}\left[\tilde{\boldsymbol{\Omega}}^{\text{s}}_{\rho}(\boldsymbol{r})\tilde{\mathbf{D}}\right] + \text{tr}\left[\tilde{\boldsymbol{\Omega}}_{\rho}(\boldsymbol{r})\tilde{\mathbf{D}}^{\text{s}}\right]\right), \tag{51}$$

which can be represented as multiplication of a new instance of the class `CompositeFunction` (with `order_` increasing by 1) and the derivative of the generalized density vector $\tilde{\rho}(\boldsymbol{r})$. This differentiation procedure has been implemented in the member function `diff_impl` of the class `CompositeFunction`.

Results from the member functions `diff_impl` of `ExchCorrEnergy` and `ExchCorrPotential` are new instances of these two classes with differentiated `energy_` and `potential_`, respectively. The general form of differentiated `energy_` and `potential_` can be obtained by extending the well-known general Leibniz rule in calculus to partial derivatives of the product of multiple functions

$$\begin{aligned}\frac{\partial^n \texttt{energy}}{\partial b_1 \cdots \partial b_n} &= w^{b_1\cdots b_n}\tilde{\epsilon}_{\text{xc}}[\tilde{\rho}(\boldsymbol{r})] \\ &+ \sum_{\substack{P\subseteq\{1,\cdots,n\}\\ Q=\{1,\cdots,n\}-P}} w^{b_P}\sum_{m=1}^{|Q|}\frac{\partial^m\tilde{\epsilon}_{\text{xc}}[\tilde{\rho}(\boldsymbol{r})]}{\partial\tilde{\rho}^m(\boldsymbol{r})}\sum_{\pi\,\in\,\mathcal{P}_{Q,m}}\prod_{j=1}^{m}\tilde{\rho}^{b_{\pi_j}}(\boldsymbol{r}),\end{aligned} \tag{52}$$

$$\begin{aligned}\frac{\partial^n \texttt{potential}}{\partial b_1 \cdots \partial b_n} &= \sum_{\substack{M\subseteq\{1,\cdots,n\}\\ P=\{1,\cdots,n\}-M}} w^{b_P}\frac{\partial\tilde{\epsilon}_{\text{xc}}[\tilde{\rho}(\boldsymbol{r})]}{\partial\tilde{\rho}(\boldsymbol{r})}\tilde{\boldsymbol{\Omega}}^{b_M}_{\rho}(\boldsymbol{r}) \\ &+ \sum_{\substack{M\subseteq\{1,\cdots,n\}\\ P\subseteq\{1,\cdots,n\}-M\\ Q=\{1,\cdots,n\}-M-P}} w^{b_P}\sum_{m=1}^{|Q|}\frac{\partial^{m+1}\tilde{\epsilon}_{\text{xc}}[\tilde{\rho}(\boldsymbol{r})]}{\partial\tilde{\rho}^{m+1}(\boldsymbol{r})} \\ &\times \sum_{\pi\,\in\,\mathcal{P}_{Q,m}}\prod_{j=1}^{m}\tilde{\rho}^{b_{\pi_j}}(\boldsymbol{r})\tilde{\boldsymbol{\Omega}}^{b_M}_{\rho}(\boldsymbol{r}),\end{aligned} \tag{53}$$

where $b_1,\ldots,b_n$ are perturbations and

$$\tilde{\rho}^{b_{j_1}\cdots b_{j_k}}(\boldsymbol{r}) = \sum_{\substack{P\subseteq\{j_1,\cdots,j_k\}\\ Q=\{j_1,\cdots,j_k\}-P}}\tilde{\boldsymbol{\Omega}}^{b_P}_{\rho}(\boldsymbol{r})\tilde{\mathbf{D}}^{b_Q}. \tag{54}$$

Equations (52) and (53) contain all possible differentiated grid weights, XC energy densities, generalized overlap distributions and one-electron spin-orbital density matrices. They will become more complicated for higher-order derivatives of `energy_` and `potential_`. As shown in Figure 1, one can use member functions `get_energy_map` and `get_potential_map` to extract the symbolic expression of Equations (52) and (53), and to convert them to types `ExcContractionMap` and `VxcContractionMap`, respectively.

The type `ExcContractionMap` is a nested map. The key of the outer map is w^{b_P} and the value is the innner map with key-value pair $\left\{\frac{\partial^m\tilde{\epsilon}_{\text{xc}}[\tilde{\rho}(\boldsymbol{r})]}{\partial\tilde{\rho}^m(\boldsymbol{r})},\ \sum_{\pi\,\in\,\mathcal{P}_{Q,m}}\prod_{j=1}^{m}\tilde{\rho}^{b_{\pi_j}}(\boldsymbol{r})\right\}$. The type `VxcContractionMap` is also a nested map and the key of the outermost map is $\tilde{\boldsymbol{\Omega}}^{b_M}_{\rho}(\boldsymbol{r})$ and the value is an object of the type `ExcContractionMap` containing corresponding $\left\{w^{b_P},\ \left\{\frac{\partial^m\tilde{\epsilon}_{\text{xc}}[\tilde{\rho}(\boldsymbol{r})]}{\partial\tilde{\rho}^m(\boldsymbol{r})},\ \sum_{\pi\,\in\,\mathcal{P}_{Q,m}}\prod_{j=1}^{m}\tilde{\rho}^{b_{\pi_j}}(\boldsymbol{r})\right\}\right\}$'s.

Time differentiation $\text{i}\frac{\partial}{\partial t}$ and $\tilde{\mathbf{T}}$ matrix

Derivatives of the time differentiated density matrix in the frequency domain have been given in Reference 3, as

$$\tilde{\mathbf{D}}^{b_1\cdots b_n}_{\omega_{B_1}\cdots\omega_{B_n}} \equiv \left(\text{i}\frac{\partial\tilde{\mathbf{D}}}{\partial t}\right)^{b_1\cdots b_n}\Bigg|_{\varepsilon=0} = \omega_{B_N}\mathbf{D}^{b_1\cdots b_n}_{\omega_{B_1}\cdots\omega_{B_n}}, \tag{55}$$

where

$$\omega_{B_N} = \omega_{B_1} + \cdots + \omega_{B_n}. \tag{56}$$

Equation (55) is also valid for the time differentiated overlap matrix $\text{i}\frac{\partial\tilde{\mathbf{S}}}{\partial t}$ and the time differentiated basis function on ket $\left|\text{i}\frac{\partial\tilde{\chi}}{\partial t}\right\rangle$.

To find out the derivatives of $\tilde{\mathbf{T}}$ matrix, we first notice that $\langle\tilde{\chi}| = \tilde{\chi}^*$ and $\left\langle\text{i}\frac{\partial\tilde{\chi}}{\partial t}\right| = \left(\text{i}\frac{\partial\tilde{\chi}}{\partial t}\right)^*$ so that we have the following expansions

$$\begin{aligned}\langle\tilde{\chi}| &= \langle\chi^{(0)}| + \sum_{\omega_{B_1}}\exp(-\text{i}\omega_{B_1}t)\varepsilon_{\omega_{B_1}}\left\langle\chi^{b_1}_{-\omega_{B_1}}\right| \\ &+ \frac{1}{2}\sum_{\omega_{B_1},\omega_{B_2}}\exp[-\text{i}(\omega_{B_1}+\omega_{B_2})t]\varepsilon_{\omega_{B_1}}\varepsilon_{\omega_{B_2}}\left\langle\chi^{b_1b_2}_{-\omega_{B_1}-\omega_{B_2}}\right| + \cdots \\ &+ \frac{1}{n!}\sum_{\omega_{B_1},\cdots,\omega_{B_n}}\exp(-\text{i}\omega_{B_N}t)\varepsilon_{\omega_{B_1}}\cdots\varepsilon_{\omega_{B_n}}\left\langle\chi^{b_1\cdots b_n}_{-\omega_{B_1}\cdots-\omega_{B_n}}\right| + \cdots,\end{aligned} \tag{57}$$

and

$$\begin{aligned}\left\langle\text{i}\frac{\partial\tilde{\chi}}{\partial t}\right| &= \sum_{\omega_{B_1}}\exp(-\text{i}\omega_{B_1}t)\varepsilon_{\omega_{B_1}}\left\langle\dot{\chi}^{b_1}_{-\omega_{B_1}}\right| \\ &+ \frac{1}{2}\sum_{\omega_{B_1},\omega_{B_2}}\exp[-\text{i}(\omega_{B_1}+\omega_{B_2})t]\varepsilon_{\omega_{B_1}}\varepsilon_{\omega_{B_1}}\left\langle\dot{\chi}^{b_1b_2}_{-\omega_{B_1}-\omega_{B_2}}\right| + \cdots \\ &+ \frac{1}{n!}\sum_{\omega_{B_1},\cdots,\omega_{B_n}}\exp(-\text{i}\omega_{B_N}t)\varepsilon_{\omega_{B_1}}\cdots\varepsilon_{\omega_{B_n}}\left\langle\dot{\chi}^{b_1\cdots b_n}_{-\omega_{B_1}\cdots-\omega_{B_n}}\right| + \cdots,\end{aligned} \tag{58}$$

and from which we can get

$$\left\langle \dot{\chi}^{b_1\cdots b_n}_{-\omega_{B_1}\cdots-\omega_{B_n}} \right| = -\omega_{B_N} \left\langle \chi^{b_1\cdots b_n}_{-\omega_{B_1}\cdots-\omega_{B_n}} \right|. \tag{59}$$

Let $\dot{\tilde{\chi}} \equiv \mathrm{i}\frac{\partial \tilde{\chi}}{\partial t}$, and by using the expansions of $|\tilde{\chi}\rangle$ and $\left|\mathrm{i}\frac{\partial \tilde{\chi}}{\partial t}\right\rangle$, Equations (57)–(59) and the definition (18) of $\tilde{\mathbf{T}}$ matrix, we have

$$\begin{aligned}
\exp(-\mathrm{i}\omega_{B_N} t)\mathbf{T}^{b_1\cdots b_n}_{\omega,\kappa\lambda} &= -\frac{1}{2}\left[\left\langle \dot{\tilde{\chi}}_\kappa \middle| \tilde{\chi}_\lambda \right\rangle + \left\langle \tilde{\chi}_\kappa \middle| \dot{\tilde{\chi}}_\lambda \right\rangle\right]^{b_1\cdots b_n}\Big|_{\{\varepsilon\}=0} \\
&= -\frac{1}{2}\sum_{\substack{P\subseteq\{1,\cdots,n\} \\ Q=\{1,\cdots,n\}-P}} \left[\left\langle \dot{\tilde{\chi}}^{b_P}_\kappa \middle| \tilde{\chi}^{b_Q}_\lambda \right\rangle + \left\langle \tilde{\chi}^{b_P}_\kappa \middle| \dot{\tilde{\chi}}^{b_Q}_\lambda \right\rangle\right]\Big|_{\{\varepsilon\}=0} \\
&= \exp(-\mathrm{i}\omega_{B_N} t)\sum_{\substack{P\subseteq\{1,\cdots,n\} \\ Q=\{1,\cdots,n\}-P}} \frac{\omega_{B_P}-\omega_{B_Q}}{2}\left\langle \chi^{b_P}_{-\omega,\kappa} \middle| \chi^{b_Q}_{\omega,\lambda} \right\rangle.
\end{aligned} \tag{60}$$

By introducing

$$\mathbf{S}^{b_P|b_Q}_{\omega,\kappa\lambda} = \left\langle \chi^{b_P}_{-\omega,\kappa} \middle| \chi^{b_Q}_{\omega,\lambda} \right\rangle \equiv \left\langle \tilde{\chi}^{b_P}_\kappa \middle| \tilde{\chi}^{b_Q}_\lambda \right\rangle\Big|_{\{\varepsilon\}=0}, \tag{61}$$

we finally have

$$\mathbf{T}^{b_1\cdots b_n}_{\omega} = \sum_{\substack{P\subseteq\{1,\cdots,n\} \\ Q=\{1,\cdots,n\}-P}} \frac{\omega_{B_P}-\omega_{B_Q}}{2}\mathbf{S}^{b_P|b_Q}_{\omega}, \tag{62}$$

and obviously

$$\mathbf{S}^{b_Q|b_P}_{\omega} = \left(\mathbf{S}^{b_P|b_Q}_{\omega}\right)^\dagger, \tag{63}$$

can be used to simplify the numerical evaluation of $\mathbf{T}^{b_1\cdots b_n}_{\omega}$.

In Tinned, we have developed classes `TemporumOperator` and `TemporumOverlap` to represent the time differentiation $\mathrm{i}\frac{\partial}{\partial t}$ and $\tilde{\mathbf{T}}$ matrix, respectively. As shown in Figure 1, an operator or a function that the time differentiation acts on is stored in the member variable `target_` of the class `TemporumOperator`. Permitted types of the variable `target_` are `ElectronicState`, `OneElecOperator` and `NonElecFunction` in the current version of Tinned.

The member variable `type_` of the class `TemporumOperator` indicates if the time differentiation acts on bra or ket, which will give opposite signs of frequencies according to Equations (59) and (55).

The member function `diff_impl` can also be implemented by following Equations (55) and (59), that is, the differentiation of a `TemporumOperator` object with respect to a perturbation "s" simply returns a new `TemporumOperator` object whose `target_` is the output of the original `target_`'s member function `diff_impl` with "s" as input.

After differentiation, the sum of frequencies $\pm\omega_{B_N}$, the time differentiated operator or function and its derivatives can be obtained via the class `TemporumOperator`'s member functions `get_frequency`, `get_target` and `get_derivatives`, respectively.

The implementation of $\tilde{\mathbf{T}}$ matrix relies on the class `TemporumOperator`. As shown in Figure 4, the key for representing $\tilde{\mathbf{T}}$ matrix is the class `TemporumOverlap`'s member variable `braket_`, which is a matrix product of time differentiated basis functions on bra and ket. We may symbolically represent `braket_` as follows

$$\texttt{braket} = \langle \dot{\tilde{\chi}}_\kappa | \times | \dot{\tilde{\chi}}_\lambda \rangle, \tag{64}$$

which is different from the definition (18) of $\tilde{\mathbf{T}}$ matrix. But its derivatives with respect to different perturbations are exactly the same as Equation (62) except for the factor $\frac{1}{2}$.

Therefore, the member function `diff_impl` of the class `TemporumOverlap` simply returns a new `TemporumOverlap` object with the variable `braket_` as the output of the differentiation of the original `braket_`, which is clearly illustrated in Figure 4.

Notice that the differentiated result of `braket_` is a sum of several terms as shown in Equation (62). We have therefore implemented two member functions `get_frequency` and `get_derivatives` for the class `TemporumOverlap`, both of which take an "index" as input as shown in Figure 1. The "index" informs which summand in Equation (62) to be considered. The function `get_frequency` will return $\frac{\omega_{B_P}-\omega_{B_Q}}{2}$ of the summand, while the function `get_derivatives` will return derivatives on bra (b_P) and ket (b_Q) of the summand. The number of summands can be straightforwardly got from the member function `size`.

Cluster operator, Lagrangian multipliers and exponential map

As shown in Figure 1, except for the class `StateVector`, we have classes `LagMultiplier`, `StateOperator`, `AdjointMap` and `ExpAdjointHamiltonian` implemented in Tinned library, respectively for the Lagrangian multipliers $\tilde{\lambda}$, the cluster operator (31), the adjoint map $\prod_j\left(\mathrm{ad}_{\tilde{\mathbf{X}}_j}\right)(\tilde{\mathbf{Y}})$ and the exponential map $\mathrm{e}^{\mathrm{ad}_{-\tilde{T}}}(\hat{H}^{b_P})$ or $\mathrm{e}^{\mathrm{ad}_{-\tilde{T}}}\left(\prod_{j=1}^{j_{\max}}\left(\mathrm{ad}_{\tilde{T}^{b_{Q_j}}}\right)(\hat{H}^{b_P})\right)$ with $1 \le j_{\max} \le 3$.

The implementation of the classes `LagMultiplier` and `StateOperator` is relatively simple by noticing their member functions `diff_impl`'s are similar to those of the classes `StateVector` and `TemporumOperator`, respectively.

The classes `AdjointMap` and `ExpAdjointHamiltonian` are developed for the exponential map $\mathrm{e}^{\mathrm{ad}_{-\tilde{T}(t)}}(\hat{H}(t))$ and its perturbation-strength derivatives (40). In the current version of Tinned library, we consider only the case that $\tilde{\mathbf{X}}_j$'s and their derivatives are commutative for the class `AdjointMap`. As shown in Figure 1, member functions `get_x` and `get_y` of the class `AdjointMap` can be used to get $\tilde{\mathbf{X}}_j$'s and $\tilde{\mathbf{Y}}$, while member functions `get_state_operator` and `get_hamiltonian` of the class `ExpAdjointHamiltonian` can return the operator $\hat{T}$ and the "Hamiltonian" $\hat{H}^{b_P}$ or $\prod_{j=1}^{j_{\max}}\left(\mathrm{ad}_{\tilde{T}^{b_{Q_j}}}\right)(\hat{H}^{b_P})$.

Last but not least, the member functions `diff_impl`'s of the classes `AdjointMap` and `ExpAdjointHamiltonian` can be implemented by noticing

$$\left[\prod_j\left(\mathrm{ad}_{\tilde{\mathbf{X}}_j}\right)(\tilde{\mathbf{Y}})\right]^{\mathrm{s}} = \sum_k\left(\mathrm{ad}_{\tilde{\mathbf{X}}_k^{\mathrm{s}}}\right)\prod_{j\neq k}\left(\mathrm{ad}_{\tilde{\mathbf{X}}_j}\right)(\tilde{\mathbf{Y}}) + \prod_j\left(\mathrm{ad}_{\tilde{\mathbf{X}}_j}\right)(\tilde{\mathbf{Y}}^{\mathrm{s}}), \tag{65}$$

$$\left[\mathrm{e}^{\mathrm{ad}_{-\tilde{T}}}(\hat{H}^{b_P})\right]^{\mathrm{s}} = \mathrm{e}^{\mathrm{ad}_{-\tilde{T}}}(\hat{H}^{b_P,\mathrm{s}}) - \mathrm{e}^{\mathrm{ad}_{-\tilde{T}}}\left(\mathrm{ad}_{\tilde{T}^{\mathrm{s}}}(\hat{H}^{b_P})\right), \tag{66}$$

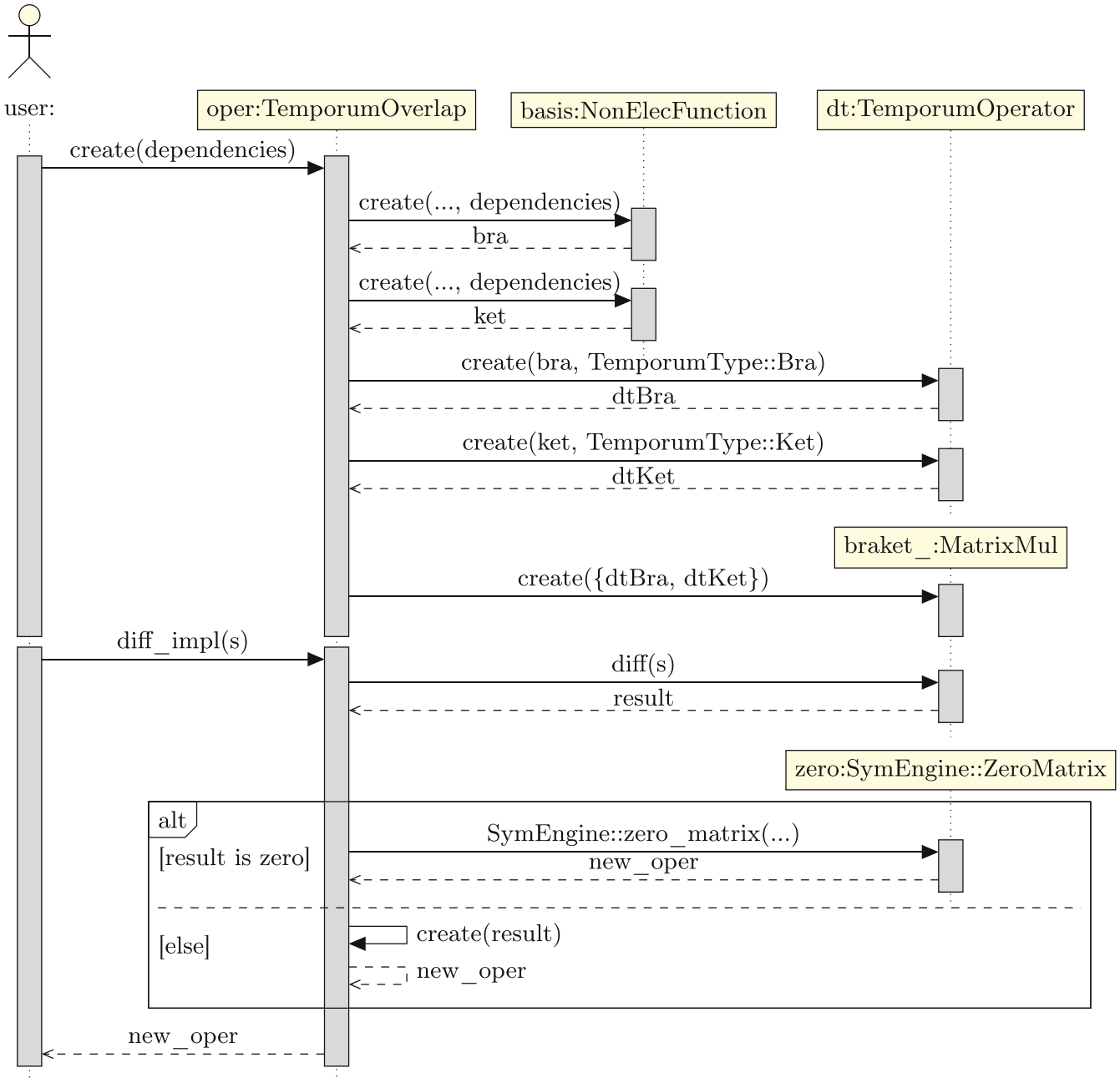

FIGURE 4 UML sequence diagram for the construction and differentiation of the class `TemporumOverlap`.

and

$$\begin{aligned}&\left[e^{\mathrm{ad}_{-\hat{T}}}\left(\prod_{j=1}^{j_{\max}}\left(\mathrm{ad}_{\hat{T}^{b_{Q_j}}}\right)(\hat{H}^{b_P})\right)\right]^s\\&=e^{\mathrm{ad}_{-\hat{T}}}\left(\left[\prod_{j=1}^{j_{\max}}\left(\mathrm{ad}_{\hat{T}^{b_{Q_j}}}\right)(\hat{H}^{b_P})\right]^s\right)-e^{\mathrm{ad}_{-\hat{T}}}\left(\left(\mathrm{ad}_{\hat{T}^s}\right)^{1\le j_{\max}<3}\prod_{j=1}\left(\mathrm{ad}_{\hat{T}^{b_{Q_j}}}\right)(\hat{H}^{b_P})\right)\\&\overset{\text{or}}{=}e^{\mathrm{ad}_{-\hat{T}}}\left(\left[\prod_{j=1}^{j_{\max}}\left(\mathrm{ad}_{\hat{T}^{b_{Q_j}}}\right)(\hat{H}^{b_P})\right]^s\right)-\left(\mathrm{ad}_{\hat{T}^s}\right)^{j_{\max}=3}\prod_{j=1}\left(\mathrm{ad}_{\hat{T}^{b_{Q_j}}}\right)(\hat{H}^{b_P}),\end{aligned}\tag{67}$$

where "s" represents a perturbation.

Symbolic search, removal and replacement

In Tinned library, we currently have (i) `FindAllVisitor`, (ii) `RemoveVisitor` and `KeepVisitor`, and (iii) `ReplaceVisitor` respectively for symbolic search, removal and replacement.

The class `FindAllVisitor` can search a given symbol and all its differentiated ones in a symbolic expression by comparing objects' names, perturbation dependencies, arguments and so forth while neglecting their derivatives. The class `FindAllVisitor` will be useful, for example, when one needs to find all perturbed density matrices, coupled-cluster amplitudes and Lagrangian multipliers.

The classes `RemoveVisitor` and `KeepVisitor` remove symbols from an expression. The former removes given symbols from the expression, while the latter removes other symbols that do not match given ones. Both `RemoveVisitor` and `KeepVisitor` treat an operator (or a function) and its derivatives independently. These two classes can be used for the elimination of response parameters in response theory calculations.[3,8]

Finally, the class `ReplaceVisitor` is derived from the class `MSubsVisitor` in SymEngine library and is used for symbolic replacement of an expression. These two classes also consider an operator (or a function) and its derivatives to be different. The class `ReplaceVisitor` extends the symbolic replacement to include operators and functions implemented in Tinned library, and can be used for, for example, the construction of the right-hand side of the linear response equation of the density matrix.[3]

Bridging symbolic computation and numerical evaluation

So far we have described different classes developed in Tinned library, which can facilitate the response theory programming and

TABLE 1 Helper functions in Tinned library to construct basic symbols for response theory computations, and to perform symbolic search, removal and replacement.

Helper functions	Description
`make_perturbation(name, frequency, components)`	Construct a perturbation with the given `name`. Parameters `frequency` (default value: 0 for a static perturbation) and `components` (default value: an empty set meaning all components will be considered) are optional.
`make_1el_density(name)`	Construct a one-electron spin-orbital density matrix $\tilde{\mathbf{D}}$ with the given `name`.
`make_1el_operator(name, dependencies)`	Construct a one-electron operator with the given `name`. Parameter `dependencies` is optional and specifies the perturbation dependencies of the operator. A perturbation-independent operator will be created when the parameter `dependencies` is not given.
`make_2el_energy(name, G)`	Construct a two-electron energy $\tilde{E}_{2\mathrm{el}} = \mathrm{tr}\left[\tilde{\mathbf{G}}(\tilde{\mathbf{D}}_{\mathrm{inner}})\tilde{\mathbf{D}}_{\mathrm{outer}}\right]$ with the given `name` and two-electron operator $\tilde{\mathbf{G}}(\tilde{\mathbf{D}}_{\mathrm{inner}})$ (specified by the parameter `G`) so that $\tilde{\mathbf{D}}_{\mathrm{inner}} = \tilde{\mathbf{D}}_{\mathrm{outer}}$.
`make_2el_operator(name, state, dependencies)`	Construct a two-electron operator $\tilde{\mathbf{G}}(\tilde{\mathbf{D}}) = \tilde{\mathbf{g}} \otimes \tilde{\mathbf{D}}$ with the given `name` and density matrix $\tilde{\mathbf{D}}$ (specified by the parameter `state`). When the optional parameter `dependencies` is not present, $\tilde{\mathbf{g}}$ will become perturbation independent.
`make_xc_energy(name, state, Omega, weight)`	Construct XC energy $\tilde{E}_{\mathrm{xc}}[\tilde{\rho}(\tilde{\mathbf{D}})] = w\tilde{\epsilon}_{\mathrm{xc}}[\mathrm{tr}_T(\tilde{\mathbf{\Omega}}_\rho\tilde{\mathbf{D}})]$ with the given `name`, $\tilde{\mathbf{D}}$ (specified by the parameter `state`), $\tilde{\mathbf{\Omega}}_\rho$ (the parameter `Omega`) and w (the parameter `weight`). If the optional parameter `weight` is not given, a perturbation-independent one will be used.
`make_xc_potential(name, state, Omega, weight)`	Construct XC potential $\tilde{\mathbf{F}}_{\mathrm{xc}}[\tilde{\rho}(\tilde{\mathbf{D}})] = w\tilde{v}_{\mathrm{xc}}[\mathrm{tr}_T(\tilde{\mathbf{\Omega}}_\rho\tilde{\mathbf{D}})]\tilde{\mathbf{\Omega}}_\rho$. Parameters are the same as those of the helper function `make_xc_energy`.
`make_nonel_function(name, dependencies)`	Construct a non-electron like function with the given `name`. Parameter `dependencies` is optional and a perturbation-independent function will be returned if it is not given.
`make_dt_operator(target, type)`	Construct $\langle \mathrm{i}\frac{\partial}{\partial t}\texttt{target}\rangle$ or $\vert \mathrm{i}\frac{\partial}{\partial t}\texttt{target}\rangle$ according to the given `type` (`TemporumType::Bra` or `TemporumType::Ket`, default value is the latter).
`make_t_matrix(dependencies)`	Construct the $\tilde{\mathbf{T}}$ matrix. Parameter `dependencies` is optional and $\tilde{\mathbf{T}}$ will become perturbation independent if it is absent.
`make_state_vector(name)`	Construct a state vector with the given `name`, such as the coupled-cluster amplitude vector $\tilde{\boldsymbol{t}}$.
`make_lagrangian_multiplier(name)`	Construct a Lagrangian multiplier vector with the given `name`.
`make_state_operator(name, state)`	Construct a state operator with the given `name` and state vector (the parameter `state`), such as the coupled-cluster operator $\hat{T}$ in Equation (31).
`make_adjoint_map(name, x, y)`	Construct an adjoint map $\prod_j\left(\mathrm{ad}_{\tilde{\mathbf{X}}_j}\right)(\tilde{\mathbf{Y}})$ with the given `name`. Parameters `x` and `y` represent respectively the vector of operators $\tilde{\mathbf{X}}_j$'s and the operator $\tilde{\mathbf{Y}}$.
`make_eadj_hamiltonian(name, operator, hamiltonian)`	Construct an exponential map with the given `name` and in the form of $\mathrm{e}^{\mathrm{ad}_{-\hat{T}}}(\hat{H}^{b_P})$ or $\mathrm{e}^{\mathrm{ad}_{-\hat{T}}}\left(\prod_{j=1}^{j_{\max}}\left(\mathrm{ad}_{\hat{T}^{b_{Q_j}}}\right)(\hat{H}^{b_P})\right)$ with $1 \le j_{\max} \le 3$. The operator $\hat{T}$ is given by the parameter `operator`, and $\hat{H}^{b_P}$ or $\prod_{j=1}^{j_{\max}}\left(\mathrm{ad}_{\hat{T}^{b_{Q_j}}}\right)(\hat{H}^{b_P})$ is given by the parameter `hamiltonian`.
`find_all(x, symbol)`	Find a given `symbol` and all its differentiated ones in `x`.
`keep_if(x, symbols)`	Keep given `symbols` in `x` while removing others.
`remove_if(x, symbols)`	Remove given `symbols` from `x`.
`replace(x, subs_dict)`	Replace some symbols by others (specified by the parameter `subs_dict`) in `x`.

computations to a great extent. The last component to be addressed is the numerical evaluation of a (differentiated) symbolic expression, which can also be implemented by using the visitor design pattern.[21]

There are two issues to be resolved for the development of the numerical evaluator. First, there are two different types of symbolic expressions—scalar and matrix that require different numerical data types. Secondly, users can choose various data types for numerical results, for example, arrays, vectors, matrices, or any user defined class. To greatly decouple symbolic and numerical computations, and to be flexible over data types, we have developed two template visitor classes in Tinned library: `FunctionEvaluator`and `OperatorEvaluator`. The former is used for evaluating scalar symbolic expressions including the matrix trace, and the latter is for evaluating different matrix expressions. Both of them require one template parameter specified by users for the data type of numerical evaluation.

To use either of the two template visitor classes, one needs to implement a derived class from the corresponding template class and to provide dispatch functions for a few classes in SymEngine and Tinned libraries. For further information, interested readers are referred to Tinned library.[9]

One benefit of the above design is that the numerical evaluation is almost performed outside the scope of Tinned library, and users will have complete control of the evaluation procedure. For instance, users can evaluate symbolic expressions sequentially or in parallel, and can cache necessary data inside derived evaluation classes for later use.

Last but not least, data types of the evaluation are not necessarily restricted to be numerical. Any C++ type can in principle be the data type of the evaluation. In particular, one can still evaluate an expression symbolically so that results are still in terms of symbols. This can be useful for studying model systems where analytical solutions are available.

LIBRARY APIS AND EXAMPLES

In the above section, we have presented all classes in Tinned library that can be used for the development of response theory. Moreover, there is a corresponding helper function for each class as shown in Table 1. Users are encouraged to use these helper functions for the construction of basic symbols and for the symbolic search, removal and replacement.

LISTING 1 **Snippet for constructing $\widetilde{E}^{0,a}$ in Equation (17).**

```
#include <utility>
#include <string>
#include <symengine/dict.h>
#include <symengine/constants.h>
#include <symengine/add.h>
#include <symengine/mul.h>
#include <symengine/matrices/matrix_mul.h>
#include <symengine/matrices/trace.h>
#include "Tinned.hpp"
using namespace Tinned;
auto a = make_perturbation(std::string("a"));
auto b = make_perturbation(std::string("b"));
auto c = make_perturbation(std::string("c"));
auto dependencies = PertDependency({
  std::make_pair(a, 99), std::make_pair(b, 99), std::make_pair(c, 99)
});
auto D = make_1el_density(std::string("D"));
auto h = make_1el_operator(std::string("h"), dependencies);
auto V = make_1el_operator(std::string("V"), dependencies);
auto T = make_t_matrix(dependencies);
auto E_2el = make_2el_energy(std::string("G"), D, dependencies);
auto weight = make_nonel_function(std::string("weight"));
auto Omega = make_1el_operator(std::string("Omega"), dependencies);
auto Exc = make_xc_energy(std::string("Exc"), D, Omega, weight);
auto hnuc = make_nonel_function(std::string("hnuc"), dependencies);
auto E = SymEngine::add(SymEngine::vec_basic({
  SymEngine::trace(SymEngine::matrix_mul(SymEngine::vec_basic({h, D}))),
  SymEngine::trace(SymEngine::matrix_mul(SymEngine::vec_basic({V, D}))),
  SymEngine::trace(SymEngine::matrix_mul(SymEngine::vec_basic({T, D}))),
  SymEngine::div(E_2el, SymEngine::two),
  Exc,
  hnuc
}));
auto D_a = D->diff(a);
auto E_a = E->diff(a);
auto E_0a = remove_if(E_a, SymEngine::set_basic({D_a}));
```

In Listing 1, we illustrate how to construct the generalized energy derivative $\tilde{E}^{0,a}$ in Equation (17) by using Tinned library. The snippet is fairly self-explanatory and it strictly follows Equation (17) by creating different terms of $\tilde{E}^{0,a}$ from lines 18 to 25. Lines 26–33 are used to make the generalized energy $\tilde{E}$ by using functions from SymEngine library for symbolic addition (`SymEngine::add`) and division (`SymEngine::div`) as well as matrix multiplication (`SymEngine::matrix_mul`) and trace (`SymEngine::trace`).

Except for the class `Perturbation`, other classes in Tinned library has a member function `diff` that can be called to perform differentiation with respect to a given perturbation. So, we get $\tilde{\mathbf{D}}^a$ and $\frac{\partial}{\partial a}\tilde{E}$ at lines 34 and 35. Finally, $\tilde{E}^{0,a}$ is obtained at line 36 by removing $\tilde{\mathbf{D}}^a$ from $\frac{\partial}{\partial a}\tilde{E}$ according to Equation (17).

Next, we consider the construction of the coupled-cluster quasi-energy Lagrangian (36) in Listing 2. One may note that the codes are not mathematically correct to represent Equation (36). For example, we create the Hamiltonian $\hat{H}$ as a one-electron operator at line 16, but with dependencies on all declared perturbations. We also use trace and multiplication instead of expectation value and dot product at lines 24–28, because the first two operations do obey the "same" rules of symbolic arithmetic and differentiation as their counterparts in the context of the coupled-cluster response theory outlined in the current work.

The third example is the use of (k,n) rule [3,4] for density-matrix based response theory computations. In Listing 3, we show the computation of $(\tilde{E}^{0,a})^{bc}_{k,n}$ with $k=n=1$, in which terms containing

LISTING 2 **Snippet for constructing the coupled-cluster quasi-energy Lagrangian (36).**

```
#include <utility>
#include <string>
#include <symengine/dict.h>
#include <symengine/constants.h>
#include <symengine/add.h>
#include <symengine/mul.h>
#include <symengine/matrices/trace.h>
#include "Tinned.hpp"
using namespace Tinned;
auto a = make_perturbation(std::string("a"));
auto b = make_perturbation(std::string("b"));
auto c = make_perturbation(std::string("c"));
auto dependencies = PertDependency({
  std::make_pair(a, 99), std::make_pair(b, 99), std::make_pair(c, 99)
});
auto H = make_1el_operator(std::string("H"), dependencies);
auto amplitudes = make_state_vector(std::string("amplitudes"));
auto multipliers = make_lagrangian_multiplier(std::string("multipliers"));
auto T = make_state_operator(std::string("T"), amplitudes);
auto eadj_H = make_eadj_hamiltonian(std::string("eadj(H)"), T, H);
auto mu_eadj_H = make_eadj_hamiltonian(std::string("mu-eadj(H)"), T, H);
auto dt_amplitudes = make_dt_operator(amplitudes);
auto L = SymEngine::add(SymEngine::vec_basic({
  SymEngine::trace(eadj_H),
  SymEngine::mul(multipliers, SymEngine::trace(mu_eadj_H)),
  SymEngine::mul(SymEngine::vec_basic({
    SymEngine::minus_one, multipliers, dt_amplitudes
  }))
}));
```

LISTING 3 **Snippet for constructing $(\tilde{E}^{0,a})^{bc}_{k,n}$ with $k=n=1$.**

```
auto D_bc = (D->diff(b))->diff(c);
auto E_0a_bc = (E_0a->diff(b))->diff(c);
auto E_0a_kn = remove_if(E_0a_bc, SymEngine::set_basic({D_bc}));
```

LISTING 4 **Snippet for finding all (un)perturbed density matrices in $(\tilde{E}^{0,a})^{bc}_{k,n}$ ($k = n = 1$).**

```
1  auto all_Ds = find_all (E_0a_kn , D);
```

perturbed density matrices at a higher order than k (involving the perturbation a) and n (perturbation a is not involved) will be removed,[3,4] that is, terms containing $\tilde{\mathbf{D}}^{bc}$ will be removed from $(\tilde{E}^{0,a})^{bc}$ as shown at line 3 of the snippet.

Our last example is to find all (un)perturbed density matrices in $(\tilde{E}^{0,a})^{bc}_{k,n}$ ($k = n = 1$) from the previous case. As illustrated in Listing 4, it only uses one line to perform such a search by using the helper function template `find_all` in the library Tinned.

From these examples, it can readily conclude that the use of SymEngine and Tinned libraries will enable one to develop response theory codes in a more compact and less error-prone manner. Symbolic computations and numerical evaluations are completely or almost decoupled for response theory. It can thus reduce the effort for the development and maintenance of response theory codes.

CONCLUSIONS

Built on top of the C++ symbolic library SymEngine, we have developed a new symbolic library Tinned for computational chemistry and in particular for response theory. As demonstrated in the previous section, it will become straightforward to develop the density matrix-based and coupled-cluster response theories by using the library Tinned. We will report this work in the near future, and a unified framework can be expected for response theory at different levels of electronic structure theory.

Furthermore, the outcome of the current work can also be helpful for other area relevant to response theory. For example, it could facilitate the development of high-order derivatives of XC energies and potentials by using the differentiated results of classes `ExchCorrEnergy` and `ExchCorrPotential`.

ACKNOWLEDGMENTS

This work was partially supported by the Research Council of Norway through its Centres of Excellence scheme, project number 262695. This work also received support from the Sigma2 through a grant of computer time (Grant no. NN14654K).

DATA AVAILABILITY STATEMENT

The data that support the findings of this study are openly available in tinned at https://github.com/bingao/tinned.

ORCID

Bin Gao https://orcid.org/0000-0003-1092-7785

REFERENCES

[1] J. Olsen, P. Jørgensen, *J. Chem. Phys.***1985**, *82*, 3235.
[2] T. Helgaker, S. Coriani, P. Jørgensen, K. Kristensen, J. Olsen, K. Ruud, *Chem. Rev.* **2012**, *112*, 543.
[3] A. J. Thorvaldsen, K. Ruud, K. Kristensen, P. Jørgensen, S. Coriani, *J. Chem. Phys.* **2008**, *129*, 214108.
[4] M. Ringholm, D. Jonsson, K. Ruud, *J. Comput. Chem.* **2014a**, *35*, 622.
[5] D. H. Friese, M. T. P. Beerepoot, M. Ringholm, K. Ruud, *J. Chem. Theory Comput.* **2015a**, *11*, 1129.
[6] D. H. Friese, M. Ringholm, B. Gao, K. Ruud, *J. Chem. Theory Comput.* **2015b**, *11*, 4814.
[7] R. Bast, D. H. Friese, B. Gao, D. J. Jonsson, M. Ringholm, S. S. Reine, K. Ruud, OpenRSP: open-ended response theory. **2020** https://doi.org/10.5281/zenodo.3923836
[8] K. Kristensen, P. Jørgensen, A. J. Thorvaldsen, T. Helgaker, *J. Chem. Phys.***2008**, *129*, 214103.
[9] B. Gao, Tinned (2023), a set of nonnumerical routines for computational chemistry. **2023** https://github.com/bingao/tinned
[10] C. Hattig, O. Christiansen, P. Jørgensen, *J. Chem. Phys.* **1998**, *108*, 8331.
[11] R. Bast, U. Ekstrom, B. Gao, T. Helgaker, K. Ruud, A. J. Thorvaldsen, *Phys. Chem. Chem. Phys.* **2011**, *13*, 2627.
[12] O. Christiansen, P. Jørgensen, *Int. J. Quantum Chem.* **1998**, *68*, 1.
[13] P. Norman, *Phys. Chem. Chem. Phys.***2011**, *13*, 20519.
[14] R. Nederpelt, F. Kamareddine, Logical Reasoning: A First Course, *Texts in computing*, King's College Publications, London **2004**.
[15] T. Mansour, Combinatorics of Set Partitions, *Discrete Mathematics and Its Applications*, Taylor & Francis, Boca Raton **2012**.
[16] M. Ringholm, D. Jonsson, R. Bast, B. Gao, A. J. Thorvaldsen, U. Ekstrom, T. Helgakero, K. Ruud, *J. Chem. Phys.* **2014b**, *140*, 034103.
[17] T. B. Pedersen, H. Koch, *J. Chem. Phys.* **1997**, *106*, 8059.
[18] B. C. Hall, Lie Groups, Lie Algebras, and Representations: An Elementary Introduction, *Graduate Texts in Mathematics*, Springer, Heidelberg **2015**.
[19] R. J. Bartlett, M. Musiał, *Rev. Mod. Phys.* **2007**, *79*, 291.
[20] SymEngine, a standalone fast C++ symbolic manipulation library. https://github.com/symengine/symengine
[21] E. Gamma, R. Helm, R. Johnson, J. Vlissides, *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley Professional Computing Series, Addison-Wesley, Boston **1994**.
[22] J. Riordan, *Bull. Amer. Math. Soc.* **1946**, *52*, 664.

How to cite this article: B. Gao, *J. Comput. Chem.* **2024**, *45*(25), 2136. https://doi.org/10.1002/jcc.27437