OpenRSP Documentation

Release 1.0.0-alpha

OpenRSP authors

Jan 31, 2020

About the project

Welcome to the website of OpenRSP - a program library for the open-ended, analytic calculation of molecular properties! Please choose a topic to learn more about what OpenRSP is, who is involved it it, how you can use it or how you can get involved.

CHAPTER 1

What is OpenRSP?

OpenRSP is a program library that uses recursive routines to identify and assemble contributions to response properties - that is, molecular properties as they are expressed in the theory called "response theory" from theoretical chemistry.

The name of OpenRSP reflects the following features:

- It is a library for the **Open**-ended calculation of **ReSP**onse properties: It can be used for the calculation of reponse properties to arbitrary order.
- It is **Open**-source and is publicly available under the LGPL v2.1 license.
- It has an application programming interface that **Open**s it to connection with other programs that wish to make use of its functionality.

1.1 What are response properties?

Response properties describe how fundamental properties of a molecular system *respond* to external influences like subjection to an electromagnetic field or displacement of the atomic nuclei. They and related properties are essential for the description of spectroscopic processes and molecular characteristics like infrared spectroscopy, Raman scattering, multiphoton absorption and vibrational energy levels. If you have ever done computational work on the molecular level for phenomena in this category, chances are that response properties were involved at some stage of the calculation.

Response properties can be categorized by their *order*, that is, the "number of influences" that were taken into consideration for a given property. The first such order is called *linear response* and contains much-used properties like the electric dipole polarizability - i.e. the first-order change to the molecular dipole moment in the presence of an electric field - or the Hessian matrix of nuclear geometric displacements - i.e. the change in the molecular gradient that would result from displacing each coordinate of the molecular geometry.

Higher orders of response properties describe the changes that the fundamental molecular property would undergo upon subjection to more than one external influence, or upon higher-order interactions with the same influence. Examples of such properties are the geometric gradient of the electric dipole polarizability - essential for the description of vibrational Raman spectra - or the cubic and quartic force constants, i.e. the third- and fourth-order derivatives of the molecular energy with respect to geometrical displacements - which may be used to calculate corrections to a description of the vibrational energy levels stemming from the geometric Hessian.

1.2 Why use OpenRSP?

By its recursive structure, OpenRSP makes it possible to calculate response properties of arbitrary complexity in an analytical manner, not resorting to numerical schemes like finite difference methods in the calculation. Compared to analytical methods, numerical approaches may be associated with a greater degree of uncertainty related to accuracy and practical feasibility of the calculation, and we therefore think that analytical calculation should be used whenever it is practical.

Today's programs written for the calculation of response properties may either not have a recursive structure, or may use numerical methods to different extents, or both. In the cases where existing programs use an analytical approach, they may either be not recursive (which typically means that a new program routine must be written for each new property for which calculation is desired), or they may only be usable for a limited category of properties. As the complexity of the expressions that must be evaluated in an analytical approach to yield the desired response property increases rapidly with the order of response, such analytic calculation of high-order response properties can quickly become a very complicated task and the implementation of *ad hoc* program routines for their calculation may be intractable at higher orders.

The structure of OpenRSP, using recursion as a core tool, solves the task of identifying and assembling contributions to response properties "once and for all". When combined with program libraries that can provide the contributions that OpenRSP identifies, any response property can be calculated fully analytically as long as those libraries can provide the necessary contributions. We note, however, that the present version of the code is still awaiting the completion of functionality to handle perturbations that both change the basis set and have a nonzero frequency associated with them, but that such extension is within the scope of the present underlying theory.

CHAPTER 2

Authors

This table lists the main developers of OpenRSP and their current affiliation:

Table 1: OpenRSP authors (alphabetical order of surname)

Name	Affiliation
Radovan Bast	UiT The Arctic University of Norway
Daniel H. Friese	Heinrich-Heine-Universität Düsseldorf
Bin Gao	UiT The Arctic University of Norway
Dan J. Jonsson	UiT The Arctic University of Norway
Magnus Ringholm	UiT The Arctic University of Norway
Simen S. Reine	University of Oslo
Kenneth Ruud	UiT The Arctic University of Norway

Requests or comments should primarily be directed to authors listed in **boldface** whose e-mail addresses are all in the format *firstname.lastname@uit.no*.

CHAPTER 3

Citation guide

All published results obtained with OpenRSP are expected to cite the following references:

- Andreas J. Thorvaldsen, Kenneth Ruud, Kasper Kristensen, Poul Jørgensen and Sonia Coriani, *J. Chem. Phys.* **129**, 214108 (2008).
- Magnus Ringholm, Dan Jonsson and Kenneth Ruud, *J. Comput. Chem.* **35**, 622-633 (2014).
- [OpenRSP] OpenRSP, an open-ended response property library, www.openrsp.org

If the use of OpenRSP involved the calculation of single residues of response properties, citation of the following reference is expected in addition to the references listed above:

- Daniel H. Friese, Maarten T. P. Beerepoot, Magnus Ringholm and Kenneth Ruud, *J. Chem. Theory Comput.* **11**, 1129-1144 (2015).

Please also note that host programs into which OpenRSP is incorporated may have their own citation guidelines or requirements to be observed if such programs are used.

CHAPTER 4

A brief history

4.1 The OpenRSP core functionality

Work on the OpenRSP project began in the mid-2000's when the first work started on what has become the present version of the program. During this time, the theoretical foundation of response theory on which OpenRSP was based was developed and used to create program routines that were connected to the Dalton quantum chemistry program. This version of the code was used to compute several response properties for which analytic calculation had not been carried out before.

In 2008, we have generalized OpenRSP for the DIRAC program package which enabled us to access a wealth of response properties at the 4-component relativistic level.

In 2011, work was started on a version - then still a part of Dalton - where recursion was used to achieve an open-ended implementation of the theory, so that one set of routines could be used to manage the calculation of any response property. This version forms the basis of the present-day core functionality of the OpenRSP, but was since developed further to include features such as calculation of single residues of response properties (of use in the calculation of multiphoton strengths), calculation of multiple properties in one invocation with reuse of common intermediate results, and restructuring of calls to external routines to reduce recalculation of various contributions such as perturbed one- and two-electron integrals.

4.2 OpenRSP as a modular library with an API

In order to make OpenRSP into a modular library that was not tied to any one particular quantum chemistry program - or *host program* - work began in 2013 on developing an application programming interface (API) for OpenRSP, involving the creation of clearly defined interfaces between OpenRSP and other codes, the use of callback routines in order to abstract the way in which the OpenRSP core asks for contributions from external libraries, and the development of the QcMatrix library to abstract and mediate matrix operations so that OpenRSP is agnostic to the underlying implementation of such operations. The first host program to make use of this modular functionality is the LSDalton <http://daltonprogram.org/>`_ quantum chemistry program.

4.3 Libraries for external contributions

During the course of its execution, OpenRSP identifies various contributions that it must get from libraries external to it in order to be able to assemble the response property or properties to be calculated, such as perturbed one- and two-electron integral contributions, exchange-correlation contributions if a density-functional theory calculation is requested, or solution of so-called response equations. Therefore, the development of libraries that can provide such functionality at a sufficient level of generality - although not necessarily driven by the demands of OpenRSP - has nevertheless been an important concurrent task, and has resulted in the creation of sophisticated software without which OpenRSP would not be able to do what it does best. Some of the libraries that are presently used or have been used by OpenRSP are listed below:

- Gen1Int for the calculation of perturbed one-electron integrals
- cgto-diff-eri for the calculation of perturbed two-electron integrals
- HODI for the calculation of perturbed integrals
- XCint and XCfun for the calculation of exchange-correlation contributions
- A linear response equation solver by Sonia Coriani et al.
- FraME for a polarizable embedding description of molecular surroundings
- PCMSolver for a polarizable continuum description of molecular surroundings

CHAPTER 5

Programs where OpenRSP is used

The following programs feature or have featured OpenRSP in some version:

Dalton has featured OpenRSP in a private version at an earlier stage of development, but that version of OpenRSP is now outdated.

LSDalton will soon feature a new and public version of OpenRSP that is in active development.

CHAPTER 6

Papers involving OpenRSP

This is a list of scientific articles where OpenRSP is involved in some capacity, either pertaining to theoretical development related to the core functionality or related functionality, or as having been applied to produce computational results.

6.1 2018

6.2 2017

6.3 2016

6.4 2015

6.5 2014

6.6 2008

CHAPTER 7

Theoretical background

We are working on a documentation of the OpenRSP core routines and its application programming interface (API), and this, together with a introduction of the underlying theory of OpenRSP intended to be accessible, will be made available on this website once ready.

In the meantime, for a technical explanation of the theoretical background of OpenRSP, the following references may prove informative:

The paper describing the version of response theory upon which OpenRSP is based.

- Andreas J. Thorvaldsen, Kenneth Ruud, Kasper Kristensen, Poul Jørgensen and Sonia Coriani, *J. Chem. Phys.* **129**, 214108 (2008)

Describes a recursive algorithmic approach for the calculation of response properties:

- Magnus Ringholm, Dan Jonsson and Kenneth Ruud, *J. Comput. Chem.* **35**, 622-633 (2014)

Describes a recursive algorithmic approach for the calculation of single residues of response functions that can be used to obtain multiphoton absorption matrix elements:

- Daniel H. Friese, Maarten T. P. Beerepoot, Magnus Ringholm and Kenneth Ruud, *J. Chem. Theory Comput.* **11**, 1129-1144 (2015)

Describes a recursive algorithmic approach for the calculation of single residues of response functions that contain perturbations which affect the basis set (please note that this functionality is not yet implemented in the latest version of OpenRSP):

- Daniel H. Friese, Magnus Ringholm, Bin Gao and Kenneth Ruud, *J. Chem. Theory Comput.* **11** (10), 4814 (2015)

CHAPTER 8

Get and run OpenRSP

If you want to get OpenRSP and use it for calculations, then please make note of the following: OpenRSP is a program library that manages the calculation of response properties, and it **cannot** calculate these properties without getting contributions like perturbed one- and two-electron integrals or solutions of response equations from other codes to which it connects through the application programming interface (API). This means that if you download and build OpenRSP from its GitHub repository, the compiled product will not on its own be able to calculate response properties. A set of API connections to enable OpenRSP to manage response property calculations can for example be made in quantum chemistry programs the where necessary routines for these contributions are implemented.

Consequently, in order to use OpenRSP for calculations, it is necessary to use it in a *host program* into which OpenRSP has been incorporated in this way, and a list of such programs is kept at the *Programs where OpenRSP is used* page. The specific way in which OpenRSP is invoked in a host program - i.e. the way that you can make OpenRSP calculate something in that program - is a feature of each such program, and you must therefore follow the relevant instructions to achieve this, as may for example be shown in the user manual for the host program that you want to use.

CHAPTER 9

Add OpenRSP to a quantum chemistry program

If you want to add OpenRSP to a quantum chemistry program, then you are free to do so provided that you do not violate OpenRSP's LGPL v2.1 software license as described on OpenRSP's GitHub repository.

In order to enable OpenRSP to work as intended, you must provide routines that connect to the OpenRSP application programming interface (API) to give OpenRSP access to contributions such as perturbed one- and two electron integrals, exchange-correlation contributions if calculations at the density-functional theory (DFT) level is desired, or solution routines for response equations.

Please note that OpenRSP is a program library that manages the calculation of response properties, and it **cannot** carry out actual such calculations without getting contributions like the ones mentioned here from program routines that are external to OpenRSP.

9.1 Compile OpenRP

Before compiling OpenRSP, you need to make sure the following programs are installed on your computer:

1. Git,
2. CMake (≥ 2.8),
3. C, C++ (if C++ APIs built) and/or Fortran 2003 (if Fortran APIs built) compilers,
4. HDF 5 (≥ 1.8) if it is enabled in QcMatrix library,
5. BLAS and LAPACK libraries, and
6. QcMatrix library.

For the time being, only CMake can be used to compile OpenRSP. In general, OpenRSP should be compiled together with the host programs. See for example the LSDalton program.

You can also compile OpenRSP alone to be familiar with how it works. But no real calculations will be performed, all the callback functions in the OpenRSP unit testing only return pre-defined data or read data from file. Let us assume that you want to compile the library in directory `build`, you could invoke the following commands:

```
mkdir build
cd build
ccmake ..
make
```

During the step `ccmake`, you need to set some parameters appropriately for the compilation. For instance, if you enable `OPENRSP_TEST_EXECUTABLE`, some executables for the test suite will be built and can run after compilation. So that you are able to check if OpenRSP has been successfully compiled. A detailed list of the parameters controlling the compilation is given in the following table:

Table 1: OpenRSP CMake parameters

CMake parameters	Description	Default
`OPENRSP_BUILD_WEB`	Build OpenRSP from WEB files (only useful for developers)	`OFF`
`OPENRSP_FORTRAN_API`	Build Fortran 2003 API	`OFF`
`OPENRSP_PERT_LABEL_BIT`	Number of bits for a perturbation label (used for perturbation free scheme)	10
`OPENRSP_TEST_EXECUTABLE`	Build test suite as excutables (otherwise, as functions in the library)	`ON`
`OPENRSP_USER_CONTEXT`	Enable user context in callback functions	`OFF`
`OPENRSP_ZERO_BASED`	Zero-based numbering	`ON`
`QCMATRIX_HEADER_DIR`	Directory of header files of QcMatrix library	`None`
`QCMATRIX_LIB`	Name of QcMatrix library with absolute path	`None`
`QCMATRIX_MODULE_DIR`	Directory of Fortran modules of QcMatrix library	`None`

9.2 OpenRSP Notations and Conventions

The following notations and conventions will be used through the OpenRSP program and the documentation:

Perturbation is described by a label, a complex frequency and its order. Any two perturbations are different if they have different labels, and/or frequencies, and/or orders.

Perturbation label An integer distinguishing one perturbation from others; all *different* perturbation labels involved in the calculations should be given by calling the application programming interface (API) `OpenRSPSetPerturbations()`; OpenRSP will stop if there is any unspecified perturbation label given afterwards when calling the APIs `OpenRSPGetRSPFun()` or `OpenRSPGetResidue()`.

Perturbation order Each perturbation can acting on molecules once or many times, that is the order of the perturbation.

Perturbation components and their ranks Each perturbation may have different numbers of components for their different orders, the position of each component is called its rank.

For instance, there will usually be x, y, z components for the electric dipole perturbation, and their ranks are `{0,1,2}` in zero-based numbering, or `{1,2,3}` in one-based numbering.

The numbers of different components of perturbations and their ranks are totally decided by the host program. OpenRSP will get such information from callback functions, that is OpenRSP itself is a perturbation free library.

NOTE: the above perturbtion free scheme is however not implemented for the current release so that OpenRSP will use its own internal representations for different perturbations.

Perturbation tuple An ordered list of perturbation labels, and in which we further require that *identical perturbation labels should be consecutive*. That means the tuple (a, b, b, c) is allowed, but (a, b, c, b) is illegal because the identical labels b are not consecutive.

As a tuple:

1. Multiple instances of the same labels are allowed so that $(a, b, b, c) \neq (a, b, c)$, and
2. The perturbation labels are ordered so that $(a, b, c) \neq (a, c, b)$ (because their corresponding response functions or residues are in different shapes).

We will sometimes use an abbreviated form of perturbation tuple as, for instance $abc \equiv (a, b, c)$.

Obviously, a perturbation tuple + its corresponding complex frequencies for each perturbation label can be viewed as a set of perturbations, in which the number of times a label (with the same frequency) appears is the order of the corresponding perturbation.

Category of perturbation frequencies We use different integers for distinguishing different values of frequencies within a frequency configuration. The category arrary is determined by:

1. For each frequency configuration, we start at the first perturbation and let its frequency value be designated number 1, then
2. For the next perturbation,
    1. If its frequency value corresponds to a frequency value encountered previously in this frequency, then use the same designation as for that previously encountered frequency value, or
    2. If its frequency value has not been encountered before, then let that frequency value be designated with the first unused number;
3. Continue like this until the end of the perturbation tuple;
4. Start the numbering over again at the next frequency configuration.

Canonical order

1. In OpenRSP, all perturbation tuples are canonically orderd according to the argument `pert_tuple` in the API `OpenRSPGetRSPFun()` or `OpenRSPGetResidue()`. For instance, when a perturbation tuple (a, b, c) given as `pert_tuple` in the API `OpenRSPGetRSPFun()`, OpenRSP will use such order ($a > b > c$) to arrange all perturbation tuples inside and sent to the callback functions.
2. Moreover, a collection of several perturbation tuples will also follow the canonical order. For instance, a collection of all possible perturbation tuples of labels a, b, c, d are $(0, a, b, ab, c, ac, bc, abc, d, ad, bd, abd, cd, acd, bcd, abcd)$, where 0 means unperturbed quantities that is always the first one in the collection.

    The rules for generating the above collection are:

    1. When taking a new perturbation into consideration, always do so in alphabetical order (and begin with the empty set);
    2. When taking a new perturbation into consideration, the new subsets are created by making the union of all previous subsets (including the empty set) and the new perturbation (putting the new perturbation at the end).

Perturbation a The first perturbation label in the tuple sent to OpenRSP APIs `OpenRSPGetRSPFun()` or `OpenRSPGetResidue()`, are the perturbation a [Thorvaldsen2008].

Perturbation addressing

1. The addressing of perturbation labels in a tuple is decided by (i) the argument `pert_tuple` sent to the API `OpenRSPGetRSPFun()` or `OpenRSPGetResidue()`, and (ii) the canonical order that OpenRSP uses.
2. The addressing of components per perturbation (several consecutive identical labels with the same complex frequency) are decided by the host program (**NOTE**: as mentioned above, the perturbtion free scheme is

not implemented for the current release so that OpenRSP will use its own internal subroutines to compute the address of components per perturbation).

3. The addressing of a collection of perturbation tuples follows the canonical order as aforementioned.

Therefore, the shape of response functions or residues is mostly decided by the host program. Take $\mathcal{E}^{abbc}$ as an example, its shape is (N_a, N_{bb}, N_c), where N_a and N_c are respectively the numbers of components of the first order of the perturbations a and c, and N_{bb} is the number of components of the second order of the perturbation b, and

1. In OpenRSP, we will use notation `[a][bb][c]` for $\mathcal{E}^{abbc}$, where the leftmost index (`a`) runs slowest in memory and the rightmost index (`c`) runs fastest. However, one should be aware that the results are still in a one-dimensional array.
2. If there two different frequencies for the perturbation b, OpenRSP will return `[a][b1][b2][c]`, where `b1` and `b2` stand for the components of the first order of the perturbation b.
3. The notation for a collection of perturbation tuples (still in a one-dimensional array) is `{1,[a],[b],[a][b],[c],[a][c],[b][c],[a][b][c]}` for $(0, a, b, ab, c, ac, bc, abc)$, where as aforementioned the first one is the unperturbed quantities.

9.3 API Reference

In order to use OpenRSP, C users should first include the header file of OpenRSP in their codes:

```
#inclde "OpenRSP.h"
```

while Fortran users should use the OpenRSP module:

```
use OpenRSP_f
```

In this chapter, we will describe all the functions defined in OpenRSP API for users. These functions should be invoked as:

```
ierr = OpenRSP...(...)
```

where `ierr` contains the error information. Users should check if it equals to `QSUCCESS` (constant defined in QcMatrix library). If not, there was error happened in the invoked function, and the calculations should stop.

9.3.1 Functions of OpenRSP API (C version)

QErrorCode **`OpenRSPCreate`**(open_rsp, num_atoms)
Creates the context of response theory calculations, should be called at first.

Var open_rsp context of response theory calculations

Vartype open_rsp OpenRSP* (struct*)

Parameters

- **`num_atoms`** (*`const QInt`*) – number of atoms (**to be removed after perturbation free scheme implemented**)

Return type QErrorCode (error information)

QErrorCode **`OpenRSPSetLinearRSPSolver`**(open_rsp, user_ctx, get_linear_rsp_solution)
Sets the context of linear response equation solver.

Var open_rsp context of response theory calculations

Vartype open_rsp OpenRSP*

Parameters

- **user_ctx** (*void**) – user-defined callback function context
- **get_linear_rsp_solution** (*const GetLinearRSPSolution (function pointer void (*)(...))*) – user-specified callback function of linear response equation solver, see the callback function `get_linear_rsp_solution()`

Return type QErrorCode

QErrorCode **OpenRSPSetPerturbations**(open_rsp, num_pert_lab, pert_labels, pert_max_orders, pert_num_comps, user_ctx, get_pert_concatenation)

Sets all perturbations involved in response theory calculations.

Var open_rsp context of response theory calculations

Vartype open_rsp OpenRSP*

Parameters

- **num_pert_lab** (*const QInt*) – number of all *different* perturbation labels involved in calculations
- **pert_labels** (*const QcPertInt**) – all the *different* perturbation labels involved
- **pert_max_orders** (*const QInt**) – allowed maximal order of a perturbation described by exactly one of the above different labels
- **pert_num_comps** (*const QInt**) – number of components of a perturbation described by exactly one of the above different labels, up to the allowed maximal order, size is therefore the sum of `pert_max_orders`
- **user_ctx** (*void**) – user-defined callback function context
- **get_pert_concatenation** (*const GetPertCat (function pointer void (*)(...))*) – user specified function for getting the ranks of components of sub-perturbation tuples (with the same perturbation label) for given components of the corresponding concatenated perturbation tuple

Return type QErrorCode

NOTE: `get_pert_concatenation()` will not be invoked in the current release; OpenRSP will use it after the perturbation free scheme implmented.

QErrorCode **OpenRSPSetOverlap**(open_rsp, num_pert_lab, pert_labels, pert_max_orders, user_ctx, get_overlap_mat, get_overlap_exp)

Sets the overlap operator.

Var open_rsp context of response theory calculations

Vartype open_rsp OpenRSP*

Parameters

- **num_pert_lab** (*const QInt*) – number of all *different* perturbation labels that can act on the overlap operator
- **pert_labels** (*const QcPertInt**) – all the *different* perturbation labels involved
- **pert_max_orders** (*const QInt**) – allowed maximal order of a perturbation described by exactly one of the above different labels
- **user_ctx** (*void**) – user-defined callback function context

- **get_overlap_mat** (*const GetOverlapMat (function pointer void (*) (...))*) – user-specified callback function to calculate integral matrices of overlap operator as well as its derivatives with respect to different perturbations, see the callback function `get_overlap_mat()`
- **get_overlap_exp** (*const GetOverlapExp (function pointer void (*) (...))*) – user-specified callback function to calculate expectation values of overlap operator as well as its derivatives with respect to different perturbations, see the callback function `get_overlap_exp()`

Return type QErrorCode

QErrorCode **OpenRSPAddOneOper**(open_rsp, num_pert_lab, pert_labels, pert_max_orders, user_ctx, get_one_oper_mat, get_one_oper_exp)

Adds a one-electron operator to the Hamiltonian.

Var open_rsp context of response theory calculations

Vartype open_rsp OpenRSP*

Parameters

- **num_pert_lab** (*const QInt*) – number of all *different* perturbation labels that can act on the one-electron operator
- **pert_labels** (*const QcPertInt**) – all the *different* perturbation labels involved
- **pert_max_orders** (*const QInt**) – allowed maximal order of a perturbation described by exactly one of the above different labels
- **user_ctx** (*void**) – user-defined callback function context
- **get_one_oper_mat** (*const GetOneOperMat (function pointer void (*) (...))*) – user-specified callback function to calculate integral matrices of one-electron operator as well as its derivatives with respect to different perturbations, see the callback function `get_one_oper_mat()`
- **get_one_oper_exp** (*const GetOneOperExp (function pointer void (*) (...))*) – user-specified callback function to calculate expectation values of one-electron operator as well as its derivatives with respect to different perturbations, see the callback function `get_one_oper_exp()`

Return type QErrorCode

QErrorCode **OpenRSPAddTwoOper**(open_rsp, num_pert_lab, pert_labels, pert_max_orders, user_ctx, get_two_oper_mat, get_two_oper_exp)

Adds a two-electron operator to the Hamiltonian.

Var open_rsp context of response theory calculations

Vartype open_rsp OpenRSP*

Parameters

- **num_pert_lab** (*const QInt*) – number of all *different* perturbation labels that can act on the two-electron operator
- **pert_labels** (*const QcPertInt**) – all the *different* perturbation labels involved
- **pert_max_orders** (*const QInt**) – allowed maximal order of a perturbation described by exactly one of the above different labels
- **user_ctx** (*void**) – user-defined callback function context

- **get_two_oper_mat** (*const GetTwoOperMat (function pointer void (*)(...))*) – user-specified callback function to calculate integral matrices of two-electron operator as well as its derivatives with respect to different perturbations, see the callback function `get_two_oper_mat()`
- **get_two_oper_exp** (*const GetTwoOperExp (function pointer void (*)(...))*) – user-specified callback function to calculate expectation values of two-electron operator as well as its derivatives with respect to different perturbations, see the callback function `get_two_oper_exp()`

Return type QErrorCode

QErrorCode **OpenRSPAddXCFun**(open_rsp, num_pert_lab, pert_labels, pert_max_orders, user_ctx, get_xc_fun_mat, get_xc_fun_exp)

Adds an exchange-correlation (XC) functional to the Hamiltonian.

Var open_rsp context of response theory calculations

Vartype open_rsp OpenRSP*

Parameters

- **num_pert_lab** (*const QInt*) – number of all *different* perturbation labels that can act on the XC functional
- **pert_labels** (*const QcPertInt**) – all the *different* perturbation labels involved
- **pert_max_orders** (*const QInt**) – allowed maximal order of a perturbation described by exactly one of the above different labels
- **user_ctx** (*void**) – user-defined callback function context
- **get_xc_fun_mat** (*const GetXCFunMat (function pointer void (*)(...))*) – user-specified callback function to calculate integral matrices of XC functional as well as its derivatives with respect to different perturbations, see the callback function `get_xc_fun_mat()`
- **get_xc_fun_exp** (*const GetXCFunExp (function pointer void (*)(...))*) – user-specified callback function to calculate expectation values of XC functional as well as its derivatives with respect to different perturbations, see the callback function `get_xc_fun_exp()`

Return type QErrorCode

QErrorCode **OpenRSPAddZeroOper**(open_rsp, num_pert_lab, pert_labels, pert_max_orders, user_ctx, get_zero_oper_contrib)

Adds a zero-electron operator to the Hamiltonian.

Var open_rsp context of response theory calculations

Vartype open_rsp OpenRSP*

Parameters

- **num_pert_lab** (*const QInt*) – number of all *different* perturbation labels that can act on the zero-electron operator
- **pert_labels** (*const QcPertInt**) – all the *different* perturbation labels involved
- **pert_max_orders** (*const QInt**) – allowed maximal order of a perturbation described by exactly one of the above different labels
- **user_ctx** (*void**) – user-defined callback function context

- **get_zero_oper_contrib** (*const GetZeroOperContrib (function pointer void (*)(...))*) – user-specified function to calculate contributions from the zero-electron operator, see the callback function `get_zero_oper_contrib()`

Return type QErrorCode

QErrorCode **OpenRSPAssemble**(open_rsp)

Assembles the context of response theory calculations and checks its validity, should be called before any function `OpenRSPGet...()`, otherwise the results might be incorrect.

Var open_rsp context of response theory calculations

Vartype open_rsp OpenRSP*

Return type QErrorCode

QErrorCode **OpenRSPWrite**(open_rsp, fp_rsp)

Writes the context of response theory calculations.

Parameters

- **open_rsp** (*const OpenRSP*) – context of response theory calculations
- **fp_rsp** (*FILE*) – file pointer

Return type QErrorCode

QErrorCode **OpenRSPGetRSPFun**(open_rsp, ref_ham, ref_state, ref_overlap, num_props, len_tuple, pert_tuple, num_freq_configs, pert_freqs, kn_rules, r_flag, write_threshold, size_rsp_funs, rsp_funs)

Gets the response functions for given perturbations.

Parameters

- **open_rsp** (*OpenRSP*) – context of response theory calculations
- **ref_ham** (*const QcMat*) – Hamiltonian of referenced state
- **ref_state** (*const QcMat*) – electronic state of referenced state
- **ref_overlap** (*const QcMat*) – overlap integral matrix of referenced state
- **num_props** (*const QInt*) – number of properties to calculate
- **len_tuple** (*const QInt*) – length of perturbation tuple for each property, size is the number of properties (`num_props`)
- **pert_tuple** (*const QcPertInt*) – ordered list of perturbation labels (perturbation tuple) for each property, size is `sum(len_tuple)`, the first label of each property is the perturbation a
- **num_freq_configs** (*const QInt*) – number of different frequency configurations for each property, size is `num_props`
- **pert_freqs** (*const QReal*) – complex frequencies of each perturbation label (except for the perturbation a) over all frequency configurations, size is `2` × `(dot_product(len_tuple, num_freq_configs)-sum(num_freq_configs))`, and arranged as `[num_freq_configs[i]][len_tuple[i]-1][2]` (`i` runs from `0` to `num_props-1`) and the real and imaginary parts of each frequency are consecutive in memory
- **kn_rules** (*const QInt*) – number k for the (k, n) rule[1] for each property (OpenRSP will determine the number n), size is the number of properties (`num_props`)

[1] The description of the (k, n) rule can be found, for instance, in [Ringholm2014].

- **r_flag** (*const QInt*) – flag to determine the restarting setup; *0* means "do not load/use any existing restarting data and do not save any new restarting data", and *3* means "use any existing restarting data and extend existing restarting data with all new restarting data"
- **write_threshold** (*const QReal*) – tensor elements with absolute value below write_threshold will not be output by OpenRSP
- **size_rsp_funs** (*const QInt*) – size of the response functions, equals to the sum of the size of each property to calculate—which is the product of the size of added perturbations (specified by the perturbation tuple pert_tuple) and the number of frequency configurations num_freq_configs for each property

Var rsp_funs the response functions, size is 2 × size_rsp_funs and arranged as [num_props][num_freq_configs][pert_tuple][2], where the real and imaginary parts of the response functions are consecutive in memory

Vartype rsp_funs QReal*

Return type QErrorCode

QErrorCode **OpenRSPGetResidue**(open_rsp, ref_ham, ref_state, ref_overlap, order_residue, num_excit, excit_energy, eigen_vector, num_props, len_tuple, pert_tuple, residue_num_pert, residue_idx_pert, num_freq_configs, pert_freqs, kn_rules, r_flag, write_threshold, size_residues, residues)

Gets the residues for given perturbations.

Parameters

- **open_rsp** (*OpenRSP**) – context of response theory calculations
- **ref_ham** (*const QcMat**) – Hamiltonian of referenced state
- **ref_state** (*const QcMat**) – electronic state of referenced state
- **ref_overlap** (*const QcMat**) – overlap integral matrix of referenced state
- **order_residue** (*const QInt*) – order of residues, that is also the length of each excitation tuple
- **num_excit** (*const QInt*) – number of excitation tuples that will be used for residue calculations
- **excit_energy** (*const QReal**) – excitation energies of all tuples, size is order_residue × num_excit, and arranged as [num_excit][order_residue]; that is, there will be order_residue frequencies of perturbation labels (or sums of frequencies of perturbation labels) respectively equal to the order_residue excitation energies per tuple excit_energy[i][:] (i runs from 0 to num_excit−1)
- **eigen_vector** (*QcMat*[]*) – eigenvectors (obtained from the generalized eigenvalue problem) of all excitation tuples, size is order_residue × num_excit, and also arranged in memory as [num_excit][order_residue] so that each eigenvector has its corresponding excitation energy in excit_energy
- **num_props** (*const QInt*) – number of properties to calculate
- **len_tuple** (*const QInt**) – length of perturbation tuple for each property, size is the number of properties (num_props)
- **pert_tuple** (*const QcPertInt**) – ordered list of perturbation labels (perturbation tuple) for each property, size is sum(len_tuple), the first label of each property is the perturbation a

- **residue_num_pert** (*const QInt**) – for each property and each excitation energy in the tuple, the number of perturbation labels whose sum of frequencies equals to that excitation energy, size is `order_residue` × `num_props`, and arragned as `[num_props][order_residue]`; a negative `residue_num_pert[i][j]` (`i` runs from `0` to `num_props-1`) means that the sum of frequencies of perturbation labels equals to `-excit_energy[:][j]`
- **residue_idx_pert** (*const QInt**) – for each property and each excitation energy in the tuple, the indices of perturbation labels whose sum of frequencies equals to that excitation energy, size is `sum(residue_num_pert)`, and arranged as `[residue_num_pert]`
- **num_freq_configs** (*const QInt**) – number of different frequency configurations for each property, size is `num_props`
- **pert_freqs** (*const QReal**) – complex frequencies of each perturbation label (except for the perturbation a) over all frequency configurations and excitation tuples, size is `2` × `(dot_product(len_tuple, num_freq_configs)-sum(num_freq_configs))` × `num_excit`, and arranged as `[num_excit][num_freq_configs[i]][len_tuple[i]-1][2]` (`i` runs from `0` to `num_props-1`) and the real and imaginary parts of each frequency are consecutive in memory; notice that the (sums of) frequencies of perturbation labels specified by `residue_idx_pert` should equal to the corresponding excitation energies for all frequency configurations and excitation tuples
- **kn_rules** (*const QInt**) – number k for the (k, n) rule for each property (OpenRSP will determine the number n), size is the number of properties (`num_props`)
- **r_flag** (*const QInt*) – flag to determine the restarting setup; *0* means "do not load/use any existing restarting data and do not save any new restarting data", and *3* means "use any existing restarting data and extend existing restarting data with all new restarting data"
- **write_threshold** (*const QReal*) – tensor elements with absolute value below `write_threshold` will not be output by OpenRSP
- **size_residues** (*const QInt*) – size of the residues, equals to the sum of the size of each property to calculate—which is the product of the size of added perturbations (specified by the perturbation tuple `pert_tuple`), the number of excitation tuples (`num_excit`) and the number of frequency configurations `num_freq_configs` for each property

Var residues the residues, size is `2` × `size_residues` and arranged as `[num_props][num_excit][num_freq_configs][pert_tuple][2]`, where the real and imaginary parts of the residues are consecutive in memory

Vartype residues QReal*

Return type QErrorCode

QErrorCode **OpenRSPDestroy**(open_rsp)

Destroys the context of response theory calculations, should be called at the end.

Var open_rsp context of response theory calculations

Vartype open_rsp OpenRSP*

Return type QErrorCode

9.3.2 Functions of OpenRSP API (Fortran version)

Functions of OpenRSP API (Fortran) are similar to those of the C version, except that an extra `_f` should be appended to each function. Other differences are the (ii) argument types and (iii) callback functions (subroutines for Fortran). The latter will be described in Chapter subsection_callback_functions. The former relates to the convention of types in Fortran, please refer to the manual of QcMatrix library and the following table for the convention:

Type in OpenRSP	Fortran
`struct OpenRSP`	`type(OpenRSP)`
`void* user_ctx`	`type(C_PTR) user_ctx`
callback functions	external subroutines

We also want to mention that users can also pass their own defined Fortran type as the user-defined callback function context to OpenRSP, by encapsulated into the `type(C_PTR) user_ctx`.

9.4 Callback Function Scheme

To use OpenRSP, users should also prepare different callback functions needed by OpenRSP. These callback functions will be invoked by OpenRSP during calculations to get integral matrices or expectation values of different one- and two-electron operators, exchange-correlation functionals and nuclear contributions, or to solve the linear response equation. The callback functions are slightly different for C and Fortran users, which will be described separately in this chapter.

It should be noted that the arguments in the following callback functions are over complete. For instance, from the knowledge of perturbations (`oper_num_pert`, `oper_pert_labels` and `oper_pert_orders`), the dimension of integral matrices `num_int` in the callback function `get_one_oper_mat()` can be computed.

Last but not least, users should be aware that:

1. OpenRSP always ask for **complex expectation values** for different one- and two-electron operators, exchange-correlation functionals and nuclear contributions, and these values are presented in memory that the real and imaginary parts of each value are consecutive. This affects:
    1. `get_overlap_exp()`
    2. `get_one_oper_exp()`
    3. `get_two_oper_exp()`
    4. `get_xc_fun_exp()`
    5. `get_zero_oper_contrib()`
2. In order to reduce the use of temporary matrices and values, OpenRSP requires that calculated integral matrices and expectation values should **be added to the returned argument**. OpenRSP will zero the entries of these matrices and expectation values at first. This requirement affects the callback functions of one- and two-electron operators, exchange-correlation functionals and nuclear contributions:
    1. `get_overlap_mat()` and `get_overlap_exp()`
    2. `get_one_oper_mat()` and `get_one_oper_exp()`
    3. `get_two_oper_mat()` and `get_two_oper_exp()`
    4. `get_xc_fun_mat()` and `get_xc_fun_exp()`
    5. `get_zero_oper_contrib()`

9.4.1 OpenRSP Callback Functions (C version)

Examples of C callback functions can be found in these files `tests/OpenRSP*Callback.c`. The detailed information of these callback functions are given as follows.

void **get_pert_concatenation**(pert_label, first_cat_comp, num_cat_comps, num_sub_tuples, len_sub_tuples, user_ctx, rank_sub_comps)

User specified function for getting the ranks of components of sub-perturbation tuples (with the same perturbation label) for given components of the corresponding concatenated perturbation tuple, the last argument for the function `OpenRSPSetPerturbations()`.

Parameters

- **pert_label** (*const QcPertInt*) – the perturbation label
- **first_cat_comp** (*const QInt*) – rank of the first component of the concatenated perturbation tuple
- **num_cat_comps** (*const QInt*) – number of components of the concatenated perturbation tuple
- **num_sub_tuples** (*const QInt*) – number of sub-perturbation tuples to construct the concatenated perturbation tuple
- **len_sub_tuples** (*const QInt**) – length of each sub-perturbation tuple, size is `num_sub_tuples`; so that the length of the concatenated perturbation is `sum(len_sub_tuples)`
- **user_ctx** (*void**) – user-defined callback function context

Var rank_sub_comps ranks of components of sub-perturbation tuples for the corresponding component of the concatenated perturbation tuple, i.e. `num_cat_comps` components starting from the one with rank `first_cat_comp`, size is therefore `num_sub_tuples` × `num_cat_comps`, and arranged as `[num_cat_comps][num_sub_tuples]`

Vartype rank_sub_comps QInt*

Return type void

NOTE: `get_pert_concatenation()` will not be invoked in the current release so that users can use a "faked" function for it.

void **get_overlap_mat**(bra_num_pert, bra_pert_labels, bra_pert_orders, ket_num_pert, ket_pert_labels, ket_pert_orders, oper_num_pert, oper_pert_labels, oper_pert_orders, user_ctx, num_int, val_int)

User-specified callback function to calculate integral matrices of overlap operator as well as its derivatives with respect to different perturbations, the second last argument for the function `OpenRSPSetOverlap()`.

Parameters

- **bra_num_pert** (*const QInt*) – number of perturbations on the bra center
- **bra_pert_labels** (*const QcPertInt**) – labels of perturbations on the bra center, size is `bra_num_pert`
- **bra_pert_orders** (*const QInt**) – orders of perturbations on the bra center, size is `bra_num_pert`
- **ket_num_pert** (*const QInt*) – number of perturbations on the ket center
- **ket_pert_labels** (*const QcPertInt**) – labels of perturbations on the ket center, size is `ket_num_pert`

- **ket_pert_orders** (*const QInt**) – orders of perturbations on the ket center, size is ket_num_pert
- **oper_num_pert** (*const QInt*) – number of perturbations on the overlap operator[2]
- **oper_pert_labels** (*const QcPertInt**) – labels of perturbations on the overlap operator, size is oper_num_pert
- **oper_pert_orders** (*const QInt**) – orders of perturbations on the overlap operator, size is oper_num_pert[3]
- **user_ctx** (*void**) – user-defined callback function context
- **num_int** (*const QInt*) – number of the integral matrices, as the product of the sizes of perturbations on the bra, the ket and the overlap operator

Var val_int the integral matrices to be added, size is num_int, and arranged as [oper_pert][bra_pert][ket_pert]

Vartype val_int QcMat*[]

Return type void

void **get_overlap_exp**(bra_num_pert, bra_pert_labels, bra_pert_orders, ket_num_pert, ket_pert_labels, ket_pert_orders, oper_num_pert, oper_pert_labels, oper_pert_orders, num_dmat, dens_mat, user_ctx, num_exp, val_exp)

User-specified function for calculating expectation values of the overlap operator and its derivatives, the last argument for the function OpenRSPSetOverlap().

Parameters

- **bra_num_pert** (*const QInt*) – number of perturbations on the bra center
- **bra_pert_labels** (*const QcPertInt**) – labels of perturbations on the bra center, size is bra_num_pert
- **bra_pert_orders** (*const QInt**) – orders of perturbations on the bra center, size is bra_num_pert
- **ket_num_pert** (*const QInt*) – number of perturbations on the ket center
- **ket_pert_labels** (*const QcPertInt**) – labels of perturbations on the ket center, size is ket_num_pert
- **ket_pert_orders** (*const QInt**) – orders of perturbations on the ket center, size is ket_num_pert
- **oper_num_pert** (*const QInt*) – number of perturbations on the overlap operator[4]
- **oper_pert_labels** (*const QcPertInt**) – labels of perturbations on the overlap operator, size is oper_num_pert
- **oper_pert_orders** (*const QInt**) – orders of perturbations on the overlap operator, size is oper_num_pert
- **num_dmat** (*const QInt*) – number of atomic orbital (AO) based density matrices
- **dens_mat** (*QcMat*[]*) – the AO based density matrices

[2] Here perturbations on the overlap operator represent those acting on the whole integral of the overlap operator, i.e. they can act on either the bra center or the ket center by applying the rule of derivatives of a product.

[3] Only overlap integrals perturbed on the bra and/or the ket, and those perturbed on the whole integral are needed in the calculations. It means that, OpenRSP will only ask for overlap integrals either with perturbations on the bra and/or ket (oper_num_pert=0), or with perturbations on the whole overlap integral (bra_num_pert=0 and ket_num_pert=0).

[4] Similar to the callback function get_overlap_mat(), OpenRSP will only ask for expectation values either with perturbations on the bra and/or ket (oper_num_pert=0), or with perturbations on the whole overlap integral (bra_num_pert=0 and ket_num_pert=0).

- **user_ctx** (*void**) – user-defined callback function context
- **num_exp** (*const QInt*) – number of the expectation values, as the product of sizes of perturbations on the bra, the ket, the overlap operator and the number of density matrices (num_dmat)

Var val_exp the expectation values to be added, size is 2 × num_exp, and arranged as [num_dmat][oper_pert][bra_pert][ket_pert][2]

Vartype val_exp QReal*

Return type void

void **get_one_oper_mat**(oper_num_pert, oper_pert_labels, oper_pert_orders, user_ctx, num_int, val_int)

User-specified function for calculating integral matrices of the one-electron operator and its derivatives, the second last argument for the function OpenRSPAddOneOper().

Parameters

- **oper_num_pert** (*const QInt*) – number of perturbations on the one-electron operator
- **oper_pert_labels** (*const QcPertInt**) – labels of perturbations on the one-electron operator, size is oper_num_pert
- **oper_pert_orders** (*const QInt**) – orders of perturbations on the one-electron operator, size is oper_num_pert
- **user_ctx** (*void**) – user-defined callback function context
- **num_int** (*const QInt*) – number of the integral matrices, as the size of perturbations that are specified by oper_num_pert, oper_pert_labels and oper_pert_orders

Var val_int the integral matrices to be added, size is num_int

Vartype val_int QcMat*[]

Return type void

void **get_one_oper_exp**(oper_num_pert, oper_pert_labels, oper_pert_orders, num_dmat, dens_mat, user_ctx, num_exp, val_exp)

User-specified callback function to calculate expectation values of one-electron operator as well as its derivatives with respect to different perturbations, the last argument for the function OpenRSPAddOneOper().

Parameters

- **oper_num_pert** (*const QInt*) – number of perturbations on the one-electron operator
- **oper_pert_labels** (*const QcPertInt**) – labels of perturbations on the one-electron operator, size is oper_num_pert
- **oper_pert_orders** (*const QInt**) – orders of perturbations on the one-electron operator, size is oper_num_pert
- **num_dmat** (*const QInt*) – number of AO based density matrices
- **dens_mat** (*QcMat*[]*) – the AO based density matrices
- **user_ctx** (*void**) – user-defined callback function context
- **num_exp** (*const QInt*) – number of expectation values, as the product of the size of perturbations on the one-electron operator (specified by oper_num_pert, oper_pert_labels and oper_pert_orders) and the number of density matrices (num_dmat)

Var val_exp the expectation values to be added, size is `2 × num_exp`, and arranged as `[num_dmat][oper_pert][2]`

Vartype val_exp QReal*

Return type void

void **get_two_oper_mat**(oper_num_pert, oper_pert_labels, oper_pert_orders, num_dmat, dens_mat, user_ctx, num_int, val_int)

User-specified function for calculating integral matrices of the two-electron operator and its derivatives, the second last argument for the function `OpenRSPAddTwoOper()`.

Parameters

- **oper_num_pert** (*const QInt*) – number of perturbations on the two-electron operator
- **oper_pert_labels** (*const QcPertInt**) – labels of perturbations on the two-electron operator, size is `oper_num_pert`
- **oper_pert_orders** (*const QInt**) – orders of perturbations on the two-electron operator, size is `oper_num_pert`
- **num_dmat** (*const QInt*) – number of AO based density matrices
- **dens_mat** (*QcMat*[]*) – the AO based density matrices ($\boldsymbol{D}$) for calculating $\boldsymbol{G}^{\texttt{perturbations}}(\boldsymbol{D})$, where `perturbations` are specified by `oper_num_pert`, `oper_pert_labels` and `oper_pert_orders`.
- **user_ctx** (*void**) – user-defined callback function context
- **num_int** (*const QInt*) – number of the integral matrices, as the product of the size of perturbations on the two-electron operator (specified by `oper_num_pert`, `oper_pert_labels` and `oper_pert_orders`) and the number of AO based density matrices (`num_dmat`)

Var val_int the integral matrices to be added, size is `num_int`, and arranged as `[num_dmat][oper_pert]`

Vartype val_int QcMat*[]

Return type void

void **get_two_oper_exp**(oper_num_pert, oper_pert_labels, oper_pert_orders, dmat_len_tuple, num_LHS_dmat, LHS_dens_mat, num_RHS_dmat, RHS_dens_mat, user_ctx, num_exp, val_exp)

User-specified callback function to calculate expectation values of two-electron operator as well as its derivatives with respect to different perturbations, the last argument for the function `OpenRSPAddTwoOper()`.

Parameters

- **oper_num_pert** (*const QInt*) – number of perturbations on the two-electron operator
- **oper_pert_labels** (*const QcPertInt**) – labels of perturbations on the two-electron operator, size is `oper_num_pert`
- **oper_pert_orders** (*const QInt**) – orders of perturbations on the two-electron operator, size is `oper_num_pert`
- **dmat_len_tuple** (*const QInt*) – length of different perturbation tuples of the left-hand-side (LHS) and right-hand-side (RHS) AO based density matrices passed; for instance, if the LHS density matrices passed are ($\boldsymbol{D}$, $\boldsymbol{D}^a$, $\boldsymbol{D}^b$, $\boldsymbol{D}^{ab}$), and the RHS density matrices passed are ($\boldsymbol{D}^b$, $\boldsymbol{D}^c$, $\boldsymbol{D}^{bc}$, $\boldsymbol{D}^d$), then `dmat_len_tuple` equals to *4*, and that means we want to calculate $\mathrm{Tr}[\boldsymbol{G}^{\texttt{perturbations}}(\boldsymbol{D})\boldsymbol{D}^b]$, $\mathrm{Tr}[\boldsymbol{G}^{\texttt{perturbations}}(\boldsymbol{D}^a)\boldsymbol{D}^c]$, $\mathrm{Tr}[\boldsymbol{G}^{\texttt{perturbations}}(\boldsymbol{D}^b)\boldsymbol{D}^{bc}]$,

and $\mathrm{Tr}[\boldsymbol{G}^{\texttt{perturbations}}(\boldsymbol{D}^{ab})\boldsymbol{D}^{d}]$, where `perturbations` are specified by `oper_num_pert`, `oper_pert_labels` and `oper_pert_orders`.

- **num_LHS_dmat** (*const QInt**) – number of LHS AO based density matrices passed for each LHS density matrix perturbation tuple, size is `dmat_len_tuple`; sticking with the above example, `num_LHS_dmat` will be `{1, N_a, N_b, N_ab}` where `N_a`, `N_b` and `N_ab` are respectively the numbers of density matrices for the density matrix perturbation tuples `a`, `b` and `ab`
- **LHS_dens_mat** (*QcMat*[]*) – the LHS AO based density matrices ($\boldsymbol{D}_{\mathrm{LHS}}$) for calculating $\mathrm{Tr}[\boldsymbol{G}^{\texttt{perturbations}}(\boldsymbol{D}_{\mathrm{LHS}})\boldsymbol{D}_{\mathrm{RHS}}]$, size is the sum of `num_LHS_dmat`
- **num_RHS_dmat** (*const QInt**) – number of RHS AO based density matrices passed for each RHS density matrix perturbation tuple, size is `dmat_len_tuple`; sticking with the above example, `num_RHS_dmat` will be `{N_b, N_c, N_bc, N_d}` where `N_b`, `N_c` `N_bc` and `N_d` are respectively the numbers of density matrices for the density matrix perturbation tuples `b`, `c`, `bc` and `d`
- **RHS_dens_mat** (*QcMat*[]*) – the RHS AO based density matrices ($\boldsymbol{D}_{\mathrm{RHS}}$) for calculating $\mathrm{Tr}[\boldsymbol{G}^{\texttt{perturbations}}(\boldsymbol{D}_{\mathrm{LHS}})\boldsymbol{D}_{\mathrm{RHS}}]$, size is the sum of `num_RHS_dmat`
- **user_ctx** (*void**) – user-defined callback function context
- **num_exp** (*const QInt*) – number of expectation values, as the product of the size of perturbations on the two-electron operator (specified by `oper_num_pert`, `oper_pert_labels` and `oper_pert_orders`) and the number of pairs of LHS and RHS density matrices, and the number of pairs of LHS and RHS density matrices can be computed as the dot product of `num_LHS_dmat` and `num_RHS_dmat`

Var val_exp the expectation values to be added, size is `2` $\times$ `num_exp`, and arranged as `[dmat_len_tuple][num_LHS_dmat][num_RHS_dmat][oper_pert][2]`

Vartype val_exp QReal*

Return type void

void **get_xc_fun_mat**(xc_len_tuple, xc_pert_tuple, num_freq_configs, pert_freq_category, dmat_num_tuple, dmat_idx_tuple, num_dmat, dens_mat, user_ctx, num_int, val_int)

User-specified function for calculating integral matrices of the XC functional and its derivatives, the second last argument for the function `OpenRSPAddXCFun()`.

Parameters

- **xc_len_tuple** (*const QInt*) – length of the perturbation tuple on the XC functional
- **xc_pert_tuple** (*const QcPertInt**) – perturbation tuple on the XC functional, size is `xc_len_tuple`
- **num_freq_configs** (*const QInt*) – the number of different frequency configurations to be considered for the perturbation tuple specified by `xc_pert_tuple`
- **pert_freq_category** (*const QInt**) – category of perturbation frequencies, size is `[num_freq_configs][xc_len_tuple]`. Take $\mathcal{E}^{gfff}$ as an example, suppose we have four different frequency configurations: "0.0,0.0,0.0,0.0" ($3N \times 10$ unique elements), "0.0,-0.2,0.1,0.1" ($3N \times 18$ unique elements), "0.0,-0,3,0.1,0.2" ($3N \times 27$ unique elements) and "0.0,-0.1,0.1,0.0" ($3N \times 27$ unique elements), the `pert_freq_category` argument would then be `(1,1,1,1, 1,2,3,3, 1,2,3,4, 1,2,3,1)`.
- **dmat_num_tuple** (*const QInt*) – the number of different perturbation tuples of the AO based density matrices passed; for instance, the complete density matrix perturbation tuples (canonically ordered) for a property $\mathcal{E}^{abc}$ (i.e. the perturbation tuple `xc_pert_tuple`

is abc) are ($\boldsymbol{D}$, $\boldsymbol{D}^a$, $\boldsymbol{D}^b$, $\boldsymbol{D}^{ab}$, $\boldsymbol{D}^c$, $\boldsymbol{D}^{ac}$, $\boldsymbol{D}^{bc}$), and with the $(0, 2)$ rule, the relevant density matrix perturbation tuples become ($\boldsymbol{D}$, $\boldsymbol{D}^b$, $\boldsymbol{D}^c$, $\boldsymbol{D}^{bc}$) that gives the dmat_num_tuple as *4*

- **dmat_idx_tuple** (*const QInt**) – indices of the density matrix perturbation tuples passed (canonically ordered), size is dmat_num_tuple; sticking with the above example, the density matrix perturbation tuples passed are ($\boldsymbol{D}$, $\boldsymbol{D}^b$, $\boldsymbol{D}^c$, $\boldsymbol{D}^{bc}$) and their associated indices dmat_idx_tuple is {1, 3, 5, 7} because these numbers correspond to the positions of the "(k, n)-surviving" perturbation tuples in the canonically ordered complete density matrix perturbation tuples
- **num_dmat** (*const QInt*) – number of collected AO based density matrices for the passed density matrix perturbation tuples (specified by dmat_idx_tuple) and all frequency configurations, that is num_freq_configs $\times \sum_{\mathrm{i}} N_{\mathrm{i}}$, where N_{i} is the number of density matrices for the density matrix perturbation tuple dmat_idx_tuple[i] for a frequency configuration
- **dens_mat** (*QcMat*[]*) – the collected AO based density matrices, size is num_dmat, and arranged as [num_freq_configs][dmat_idx_tuple]
- **user_ctx** (*void**) – user-defined callback function context
- **num_int** (*const QInt*) – number of the integral matrices, equals to the product of the size of perturbations on the XC functional (specified by the perturbation tuple xc_pert_tuple) and the number of different frequency configurations num_freq_configs

Var val_int the integral matrices to be added, size is num_int, and arranged as [num_freq_configs][xc_pert_tuple]

Vartype val_int QcMat*[]

Return type void

void **get_xc_fun_exp**(xc_len_tuple, xc_pert_tuple, num_freq_configs, pert_freq_category, dmat_num_tuple, dmat_idx_tuple, num_dmat, dens_mat, user_ctx, num_exp, val_exp)

User-specified function for calculating expectation values of the XC functional and its derivatives, the last argument for the function OpenRSPAddXCFun().

Parameters

- **xc_len_tuple** (*const QInt*) – length of the perturbation tuple on the XC functional
- **xc_pert_tuple** (*const QcPertInt**) – perturbation tuple on the XC functional, size is xc_len_tuple
- **num_freq_configs** (*const QInt*) – the number of different frequency configurations to be considered for the perturbation tuple specified by xc_pert_tuple
- **pert_freq_category** (*const QInt**) – category of perturbation frequencies, size is [num_freq_configs][xc_len_tuple].
- **dmat_num_tuple** (*const QInt*) – the number of different perturbation tuples of the AO based density matrices passed
- **dmat_idx_tuple** (*const QInt**) – indices of the density matrix perturbation tuples passed (canonically ordered), size is dmat_num_tuple
- **num_dmat** (*const QInt*) – number of collected AO based density matrices for the passed density matrix perturbation tuples (specified by dmat_idx_tuple) and all frequency configurations, that is num_freq_configs $\times \sum_{\mathrm{i}} N_{\mathrm{i}}$, where N_{i} is the number of

density matrices for the density matrix perturbation tuple `dmat_idx_tuple[i]` for a frequency configuration

- **dens_mat** (*QcMat*[]*) – the collected AO based density matrices, size is `num_dmat`, and arranged as `[num_freq_configs][dmat_idx_tuple]`
- **user_ctx** (*void**) – user-defined callback function context
- **num_exp** (*const QInt*) – number of the expectation values, equals to the product of the size of perturbations on the XC functional (specified by the perturbation tuple `xc_pert_tuple`) and the number of different frequency configurations `num_freq_configs`

Var val_exp the expectation values to be added, size is $2 \times$ `num_exp`, and arranged as `[num_freq_configs][xc_pert_tuple][2]`

Vartype val_exp QReal*

Return type void

void **get_zero_oper_contrib**(oper_num_pert, oper_pert_labels, oper_pert_orders, user_ctx, size_pert, val_oper)

User-specified callback function to calculate contributions from the zero-electron operator, the last argument for the function `OpenRSPAddZeroOper()`.

Parameters

- **oper_num_pert** (*const QInt*) – number of perturbations on the zero-electron operator
- **oper_pert_labels** (*const QcPertInt**) – labels of perturbations on the zero-electron operator, size is `oper_num_pert`
- **oper_pert_orders** (*const QInt**) – orders of perturbations on the zero-electron operator, size is `oper_num_pert`
- **user_ctx** (*void**) – user-defined callback function context
- **size_pert** (*const QInt*) – size of the perturbations on the zero-electron operator

Var val_oper contributions from the zero-electron operator to be added, arranged as `[size_pert][2]`

Vartype val_oper QReal*

Return type void

void **get_linear_rsp_solution**(num_pert, num_comps, num_freq_sums, freq_sums, RHS_mat, user_ctx, rsp_param)

User-specified callback function of linear response equation solver, the last argument for the function `OpenRSPSetLinearRSPSolver()`.

Parameters

- **num_pert** (*const QInt*) – number of different perturbations on the right hand side of the linear response equation
- **num_comps** (*const QInt**) – number of components of each perturbation, size is `num_pert`
- **num_freq_sums** (*const QInt**) – for each perturbation, number of complex frequency sums on the left hand side of the linear response equation, size is `num_pert`

- **freq_sums** (*const QReal**) – the complex frequency sums on the left hand side of the linear response equation, size is twice of the sum of num_freq_sums, the real and imaginary parts of each frequency sum are consecutive in memory
- **RHS_mat** (*QcMat*[]*) – RHS matrices, size is the dot product of num_comps and num_freq_sums, and index of num_freq_sums runs faster in memory
- **user_ctx** (*void**) – user-defined callback function context

Var rsp_param solved response parameters, size is the dot product of num_comps and num_freq_sums, and index of num_freq_sums runs faster in memory

Vartype rsp_param QcMat*[]

Return type void

9.4.2 OpenRSP Callback Subroutines (Fortran version)

The callback subroutines of Fortran codes take almost the exact arguments as the callback functions of C codes. One difference is the type convention between C and Fortran, which has been discussed in Secion subsubsection_fortran_convention. Moreover, the pointers of basic types (integer and real numbers) in the C codes should be converted to corresponding array in Fortran. The array of QcMat pointers should be converted to an array of type(QcMat) in Fortran. Last, the user-defined callback function/subroutine context should be replaced by type(C_PTR).

We will develop Fortran unit testing in next release. For the time being, interested users can refer to LSDalton for examples of Fortran callback subroutines.

9.5 Limitations or Known Problems

- "T matrix contributions" - i.e. contributions from the perturbed "half-time-differentiated" overlap matrix - are not yet supported in the newest version of the code. These contributions are only nonzero for perturbations that both a) affect the basis set and b) have frequencies other than zero. The most relevant such kind of perturbation is the magnetic dipole perturbations using London atomic orbitals. Properties consisting of only other kinds of perturbations - such as geometric displacement of the nuclei or electric dipole perturbations - are unaffected by the lack of T matrix contributions.
- Currently we use QcPertInt (defined as QInt type in include/RSPPerturbation.h, and src/fortran/RSPPertBasicTypes.F90 for Fortran APIs) to reprenset several perturbation labels (see subsection_notations_and_conventions), in which one label is described by OPENRSP_PERT_LABEL_BIT bits (that can be modified during the step ccmake, see subsection_compile).

  For the time being, we do not suggest that users change the type of QcPertInt, because other integer types are not supported by OpenRSP yet.
- The current implementation of residues is just tested for electric field perturbations and single residues.

9.6 Unit Testing

After successfully building OpenRSP (see subsection_compile), we recommend users perform the unit testing of OpenRSP.

If OPENRSP_TEST_EXECUTABLE is enabled, you will have an executable openrsp_c_test after successfully building OpenRSP. Run this executable for unit testing.

If `OPENRSP_TEST_EXECUTABLE` is disabled, you will need to call the function

QErrorCode **`OpenRSPTest`**(FILE **fp_log*)

to perform the unit testing.

CHAPTER 10

Get involved with development

We welcome your participation if you want to become involved with the development of OpenRSP! Our code is hosted on GitHub and is publicly available under the LGPL v2.1 software license. You may freely obtain and use this code provided that you do not violate this software license, but us present *Authors* would of course would of course also like to get in touch with you.

We are still working on a documentation of the OpenRSP code and API, and this will be made available on this website when ready. A style guide and contribution guidelines are also under development.

CHAPTER 11

How Sphinx works

These pages are generated using Sphinx. If you want to find out more about RST/Sphinx, please read http://sphinx-doc.org/rest.html. RST is a subset of Sphinx. Sphinx is RST with some extensions.

11.1 How to modify the website

The website is generated from RST sources under `doc/`. Once a pull request is merged, a post-receive hook updates the documentation on https://openrsp.readthedocs.io. This typically takes less than a minute. Our main page http://openrsp.org redirects to https://openrsp.readthedocs.io.

11.2 How to locally test changes

You don't have to push to see and test your changes. You can test them locally. For this install the Python packages `sphinx` and `sphinx_rtd_theme`. Then build the pages with:

```
$ sphinx-build doc/ _build
```

Then point your browser to `_build/html/index.html`. The style is not the same but the content is what you would see after a successful pull request merge.

CHAPTER 12

Tentative Rules for Developers

12.1 Short-version

1. First analyze the problem, then design the code (data and algorithm structures), prepare test suite. Last, write the code.
2. Make everything as simple as possible.
3. Do your best to prepare a readable document.

12.2 Long-version

1. First of all, please write explicitly what you would like to implement in `doc`! Describe your idea using formulas and/or words. Then translate them into algorithms and data structure. Please do write what objects/types/variables you will define and their corresponding public and private functions (including detailed descriptions of the input and output arguments). It would be better if you could write down the framework of your implementation using figures. Please also write down the limitations or risks of your code, for instance, does it stable or have some numerical error? If yes, how to prevent or how to know if the results are reasonable?

   In this stage, you may refer to some rules in object-oriented programming (OOP). For instance, when you define a module/class etc.:

   1. it should be open for extension but closed for modification (Open Closed Principle, OCP),
   2. subclasses should be substitutable for their base classes (Liskov Substitution Principle, LSP),
   3. depend upon abstractions, do not depend upon concretions (Dependency Inversion Principle, DIP),
   4. many client specific interfaces are better than one general purpose interface (Interface Segregation Principle, ISP),
   5. In other words: low coupling, high cohesion, open for extension, and closed for changes (from "Developing Chemical Information Systems: An Object-Oriented Approach Using Enterprise Java", Fan Li).

2. Write the codes. During this stage, we would be happy if you could:

    1. write comments (in english, one line for each 10-20 line of codes at least),
    2. try to use descriptive names for your classes and methods,
    3. do your best to avoid global variables,
    4. try to re-use code and try to use libraries,

3. This is very important, and should be considered and implemented during the aforementioned two steps:

   Always provide a test suite for each function/subroutine/module etc., unless you are 100% sure what you did is right. Integration testing will also be required in some cases.

Bibliography

[Morgan2018] Geometric Energy Derivatives at the Complete Basis Set Limit: Application to the Equilibrium Structure and Molecular Force Field of Formaldehyde, Morgan, W. James; Matthews, Devin A.; Ringholm, Magnus; et al. J. Chem. Theory Comput. 14 (3), 1333 (2018)

[DiRemigio2017] Open-ended formulation of self-consistent field response theory with the polarizable continuum model for solvation Di Remigio, Roberto; Beerepoot, Maarten T. P.; Cornaton, Yann; et al. PCCP 19 (1), 366 (2017)

[Anelli2017] Gauge-origin independent calculations of electric-field-induced second-harmonic generation circular intensity difference using London atomic orbitals Anelli, Marco; Ringholm, Magnus; Ruud, Kenneth Mol. Phys. 115 (1-2), 241 (2017)

[Steindal2016] Open-ended response theory with polarizable embedding: multiphoton absorption in biomolecular systems Steindal, Arnfinn Hykkerud; Beerepoot, Maarten T. P.; Ringholm, Magnus; et al. PCCP 18 (40), 28339 (2016)

[Cornaton2016-2] Complete analytic anharmonic hyper-Raman scattering spectra Cornaton, Yann; Ringholm, Magnus; Ruud, Kenneth PCCP 18 (32), 22331 (2016)

[Cornaton2016] Analytic calculations of anharmonic infrared and Raman vibrational spectra Cornaton, Yann; Ringholm, Magnus; Louant, Orian; et al. PCCP 18 (5) 4201 (2016)

[Friese2015-2] Open-Ended Recursive Calculation of Single Residues of Response Functions for Perturbation-Dependent Basis Sets Friese, Daniel H.; Ringholm, Magnus; Gao, Bin; et al. J. Chem. Theory Comput. 11 (10), 4814 (2015)

[Friese2015] Open-Ended Recursive Approach for the Calculation of Multiphoton Absorption Matrix Elements Friese, Daniel H.; Beerepoot, Maarten T. P.; Ringholm, Magnus; et al. J. Chem. Theory Comput. 11 (3), 1129 (2015)

[Ringholm2014-3] Analytic calculations of hyper-Raman spectra from density functional theory hyperpolarizability gradients Ringholm, Magnus; Bast, Radovan; Oggioni, Luca; et al. J. Chem. Phys. 141 (13), 134107 (2014)

[Ringholm2014-2] Analytic cubic and quartic force fields using density-functional theory Ringholm, Magnus; Jonsson, Dan; Bast, Radovan; et al. J. Chem. Phys. 140 (3), 034103 (2014)

[Gao2014] Analytic Density Functional Theory Calculations of Pure Vibrational Hyperpolarizabilities: The First Dipole Hyperpolarizability of Retinal and Related Molecules Gao, Bin; Ringholm, Magnus; Bast, Radovan; et al. J. Phys. Chem. A 118 (4), 748 (2014)

[Ringholm2014] A General, Recursive, and Open-Ended Response Code Ringholm, Magnus; Jonsson, Dan; Ruud, Kenneth J. Comput. Chem. 35 (8), 622 (2014)

[Thorvaldsen2008] A density matrix-based quasienergy formulation of the Kohn–Sham density functional response theory using perturbation- and time-dependent basis sets Thorvaldsen, Andreas J.; Ruud, Kenneth; Kristensen, Kasper; et al. J. Chem. Phys. 129 (21), 214108 (2008)

Index

G

O