OpenRSP Documentation

Release 0.0

OpenRSP developers

August 03, 2015

Contents

OpenRSP is a program for the open-ended calculation of response properties. It connects to response equation solution routines, routines for differentiated one-electron and two-electron integral contributions, and routines for exchange/correlation contributions to enable the calculation of response properties to arbitrary order. The program is not yet ready for public release.

CHAPTER 1

The people behind OpenRSP

1.1 Authors

- Andreas J. Thorvaldsen
- Arnfinn H. Steindal
- Bin Gao
- Dan Jonsson
- Daniel Friese
- Jógvan Magnus Haugaard Olsen
- Kenneth Ruud
- Magnus Ringholm
- Michal Repisky
- Radovan Bast
- Thomas Kjærgaard

1.2 Publications

Several projects where results were obtained using the OpenRSP program have already been carried out, and an overview will be posted.

CHAPTER 2

Documentation for developers

2.1 Compiling the code

2.1.1 Dalton on stallo

First we make sure that we have all required modules loaded:

```
$ module load python/2.7.3 intel/13.0 mkl/11.0.0 cmake/2.8.9 git/1.8.3.4 openmpi/1.6.2
```

We verify that mpif90 really resolves to Intel:

```
$ mpif90 --version
```

Now we can configure the code:

```
$ ./setup --mpi --mkl=parallel
```

And build it:

```
$ cd build
$ make dalton.x
```

2.1.2 BLAS/LAPACK errors

On some occasions (at least experienced when compiled sequentially on Ubuntu), there might be errors with missing references to BLAS or LAPACK libraries (the reason is not fully understood, but it could just be that the default version of these libraries on Ubuntu (14.04) is incomplete). This was resolved by following the steps by user 'treerink' on http://ubuntuforums.org/showthread.php?t=1505249

```
"Go to: System -> Synaptic -> Administration -> Package Manager ->
search on lapack (and/or blas), and mark for installation:

libblas3gf
libblas-doc
libblas-dev
liblapack3gf
liblapack-doc
liblapack-dev

-> Apply "
```

2.2 Testing

Within Dalton:

```
$ ctest [-jN] -L openrsp
```

2.3 Known problems

- Not safe to use "regular" DFT on Dalton's openrsp branch.
- vibgamma test broken on GNU.

2.4 Coding standards

2.4.1 Indentation

2 blanks after module, subroutine, function, program (and contains)

The corresponding 'contains' at same indentation.

3 blanks after if, do, type

No tabs, really. Tab enthusiasts will tell you that they are great but they are a big pain if you copy paste code or want to comment code out.

2.4.2 Do not use "<<<<" or ">>>>" anywhere in the code

These are conflict resolution markers.

2.4.3 Private vs. public

Everything should be private except names explicitly declared as public

This speeds up compilation and avoids bugs and name clashes. It gives more modular and thus better code.

2.4.4 Write end module, end type, end function, end subroutine, end if, end do, without name

subroutine foo end subroutine instead of subroutine foo end subroutine foo

Reason: It is unnecessary and annoying when you want to rename things, you have to rename them in two places.

Also no single-word variants endmodule, endtype, endfunction, endsubroutine, endif or enddo.

2.4.5 Never use F77 style common blocks

They are like nuclear waste.

2.4.6 Don't put several commands on one line

Hard to read and inconvenient for debugging (when you need to uncomment one of the commands): Ef=-Ef; Eff=-Eff; Effff=-Effff; Egff=-Egff

It is also inconvenient from the version control point of view: more commands on one line increase risk of conflicts during merges.

2.4.7 Use module-wide implicit none

Example:

```
module birefring

  use this
  use that

  implicit none

  ...
```

Never use implicit.h. After you spend a day hunting a bug created by using implicit you know why.

2.4.8 Do not leave commented code behind

Also if you replace a routine by a better routine, remove the less better routine.

2.4.9 No program-specific CPP filters

They mean that the library is not general. Program specificity has to go to the interfaces.

2.4.10 Use self-explaining variable names

Also never ever reuse variable names just to save a declaration.

2.4.11 Module naming

Modules and files containing modules should have the same name.

Also we should not let Dalton (or DIRAC) restrict our naming. We had examples where we chose a less ideal name because the better name could upset some developers of Dalton. It is our library and we can dictate naming. Name conflicts can be resolved in interfaces.

2.5 Some equations (just to show how to type equations)

Example ...

$$H = \sum_i h(i) + \frac{1}{2} \sum_{i \neq j} g(i,j) + V_{NN}; \quad V_{NN} = \frac{1}{2} \sum_{A \neq B} \frac{Z_A Z_B}{R_{AB}}$$

Here is some inline example: V_{eN}. And another example:

$$h_D = \beta mc^2 + c(\alpha \cdot \mathbf{p}) + V_{eN}$$

And yet another example:

$$g^{Coulomb}(1,2) = \frac{1}{r_{12}}$$

2.6 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.

2.6.1 How to modify the webpages

You can modify these pages by cloning our public documentation repository:

```
git clone git@gitlab.com:openrsp/website.git
```

Once you commit and push, a post-receive hook updates the documentation on http://openrsp.readthedocs.org. This typically takes less than a minute. Our main page http://openrsp.org redirects to http://openrsp.readthedocs.org.

2.6.2 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 python-sphinx and python-matplotlib. Then build the pages with:

```
make html
```

Then point your browser to _build/html/index.html. The style is not the same but the content is what you would see after the git push.

2.7 Keyword reference

2.7.1 .CUSTOM

Description...

2.7.2 .FREQ

Description...

The keyword is followed by at least 2 lines:

```
.FREQ
1
0.02
```

2.7.3 .SPORDR

Description...

The keyword is followed by 1 line:

```
.SPORDR
2
```

2.7.4 .SPPLAB

Description...

The keyword is followed by at least 1 line:

```
.SPPLAB
EL
GEO
```

2.7.5 .SPRULE

Description...

The keyword is followed by 2 lines:

```
.SPRULE
1
1
```

2.7.6 .THRESH

Description...

The keyword is followed by 1 line:

```
.THRESH
 1.0d-5
```

2.7.7 .XCGRID

Use the XCint grid instead of the default Dalton grid.

The radial grid is generated according to Lindh, Malmqvist, and Gagliardi [TCA 106(3), 178-187 (2001)].

The angular grid is generated according to Lebedev and Laikov [A quadrature formula for the sphere of the 131st algebraic order of accuracy, Russian Academy of Sciences Doklady Mathematics, Volume 59, Number 3, 1999, pages 477-481].

The keyword is followed by 3 lines:

```
.XCGRID
 1.0d-12        # radial precision
 86             # minimum number of angular points per radial shell
 302            # maximum number of angular points per radial shell
```

The smaller the radial precision, the better.

The higher the values for minimum and maximum number of angular points, the better.

For the minimum and maximum number of angular points the code will use the following table and select the closest number with at least the desired precision:

```
  {6,   14,   26,   38,   50,   74,   86,  110,  146,  170,
 194,  230,  266,  302,  350,  434,  590,  770,  974, 1202,
1454, 1730, 2030, 2354, 2702, 3074, 3470, 3890, 4334, 4802,
4934, 5294, 5810}
```

The pruning is a primitive linear interpolation. The full angular grid is reached at 0.2 times the Bragg radius of the center.