XCint Documentation

Release 0.0.0

Radovan Bast

Feb 23, 2020

Contents

CHAPTER 1

What XCint does

XCint integrates the exchange-correlation (XC) energy E_{xc} and the elements of the XC potential matrix V_{xc}, as well as their derivatives with respect to electric field and/or geometric perturbations. The integration is performed on a standard numerical grid.

Before the XC energy can be computed, we require the densities. This is done in two steps:

$$n_b = \sum_k \chi_{kb} \sum_l D_{kl} \chi_{lb} = \sum_k \chi_{kb} X_{kb}$$

$$X_{kb} = \sum_l D_{kl} \chi_{lb}$$

Then the XC energy is computed using the XC energy density ϵ_{xc} evaluated with the help of XCFun:

$$E_{\text{xc}} = \sum_b w_b \epsilon_{\text{xc}}(n_b)$$

A similar strategy is used to compute V_{xc} matrix elements, again in two steps:

$$(V_{\text{xc}})_{kl} = \sum_b w_b \chi_{kb} v_{\text{xc}}(n_b) \chi_{lb} = \sum_b W_{kb} \chi_{lb}$$

$$W_{kb} = w_b \chi_{kb} v_{\text{xc}}(n_b)$$

Note that XCFun is called only once and returns ϵ_{xc} and v_{xc} in one go.

In the above scheme we work with batches of points in order to exploit vectorization and screening, making use of BLAS level 3 libraries to compute X_{kb} and $(V_{\text{xc}})_{kl}$.

CHAPTER 2

System requirements

- CMake above 3.5
- C and C++ compiler
- BLAS library

2.1 Optional requirements

- Fortran compiler (to build the Fortran interface)
- CFFI (to access the Python interface)
- py.test (to test the Python interface)

2.2 External libraries used by XCint

- **XCFun** Copyright: Ulf Ekstrom, licensed under MPL version 2.0
- **Google Test** Copyright: Google Inc.

CHAPTER 3

Building and testing

```
git clone --recursive git@github.com:dftlibs/xcint.git
cd xcint/
./setup [--help]
cd build/
make
make test
```

CHAPTER 4

Interfacing

4.1 General hints

The first is that the density matrix has to be scaled by half before calling the routine, and that is already counting that D is spin-restricted. Similarly, the output F needs to be multiplied times two. This is probably historically related to how DALTON handles data.

The AO primitive weights are expected to contain the spherical harmonic normalization constant. This is essential because there are different ways of absorbing the angular and radial normalizations into different quantities, depending on definitions.

4.2 Where can I find examples?

- https://github.com/dftlibs/xcint/blob/master/test/energy_spherical.cpp
- https://github.com/dftlibs/xcint/blob/master/test/test.f90

4.3 Where is the interface?

The Python interface is currently broken but the Fortran and C interfaces work.

Yes, we need some documentation here but hopefully these files are a good start:

- https://github.com/dftlibs/xcint/blob/master/api/xcint.h
- https://github.com/dftlibs/xcint/blob/master/api/xcint.f90

CHAPTER 5

Status and outlook

5.1 Status

The API is pre-alpha. Expect significant changes.

5.2 Grid generation

The grid generation has been moved outside XCint (the tests employ https://github.com/dftlibs/numgrid). This has the following advantages:

- Gives the caller the possibility to use other grid generators.
- Makes grid-based (MPI) parallelization relatively trivial.
- Moves grid-based (MPI) parallelization outside XCint.

5.3 MPI parallelization

MPI parallelization has been removed (see above section) as it can be introduced by the caller in very few lines. Not having MPI parallelization inside XCint simplifies the code and testing.

5.4 Functional parsing

This is currently done inside the code but should move outside. The reason why the functional is parsed and tracked inside the code is that XCFun does not allow to track the same functional using both the Fortan and C interfaces in the same run. Moving the functional parsing out now would break the Fortran interface of XCint.

5.5 Outlook

The plan is to also move the density evaluation (together with AO evaluation and "Fock"-type matrix distribution routines) outside of XCint to a separate library. This will make XCint very thin and compact. Another advantage is that this step will make XCint basis-set agnostic and point-group symmetry agnostic and therefore more general.