XCFun

Release 2.1.1

Ulf Ekström and contributors

Nov 30, 2022

CONTENTS

XCFun is a library of exchange-correlation (XC) functionals to be used in density-functional theory (DFT) codes. XCFun follows a unique implementation strategy which enables the computation of derivatives of the XC functional kernel up to arbitrary order. It does so by relying on forward-mode automatic differentiation.

Given a new XC functional kernel, its implementation with all its derivatives only requires to write code for the undifferentiated kernel. This implementation strategy is very powerful and allows:

1. Faster implementation of new functionals: you write the kernel, the compiler does the rest.
2. Introduction of new variables, for example current densities, in the parametrization of new or existing XC kernels.
3. Testing for numerical stability of XC kernels, using arbitrary precision arithmetic libraries.

Contents:

CHAPTER

ONE

BUILDING XCFUN

1.1 Dependencies

- A C++ compiler compliant with the C++11 standard. See here for a list of compatible compilers.
- The CMake build system generator. Version 3.11 or later is required. To install a recent version of CMake locally:

```
$ CMAKE_VERSION=3.14.7
$ target_path=$HOME/Deps/cmake/$CMAKE_VERSION
$ cmake_url="https://cmake.org/files/v${CMAKE_VERSION%.*}/cmake-${CMAKE_VERSION}-
↪Linux-x86_64.tar.gz"
$ mkdir -p "$target_path"
$ curl -Ls "$cmake_url" | tar -xz -C "$target_path" --strip-components=1
$ export PATH=$HOME/Deps/cmake/$CMAKE_VERSION/bin${PATH:+:$PATH}
```

1.1.1 Optional dependencies

To compile the standalone examples:

- A Fortran compiler with complete `iso_c_binding` support.
- A C compiler compliant with the C99 standard.

To compile the Python bindings:

- Python 3.6+ and its development libraries and headers.
- pybind11. This will be automatically downloaded if not available.
- Other dependencies listed in the *requirements.txt* or *environment.yml* files.

To compile the documentation:

- Doxygen
- Sphinx
- The Breathe Sphinx extension.
- The `recommonmark` Sphinx extension.

1.2 Configuring, building, testing

1. Clone the repository from GitHub or download a tarball with the sources.
2. Configure:

```
$ cmake -H. -Bbuild -DCMAKE_INSTALL_PREFIX=<install-prefix>
```

   We also provide a Python script as front-end to CMake, see *Compilation options*.

3. Build:

```
$ cd build
$ make
```

4. Test:

```
$ ctest
```

5. Install:

```
$ make install
```

Congratulations, you are all set to use XCFun! Read on for details on *Using XCFun*.

1.3 Compilation options

A Python script called `setup` is made available as a front-end to CMake. The basic configuration command:

```
$ cmake -H. -Bbuild -DCMAKE_INSTALL_PREFIX=<install-prefix>
```

translates to the following invocation of the `setup` script:

```
$ python setup --prefix=<install-prefix>
```

The script's options mirror exactly the options you can set by directly using CMake.

- `--cxx` / `CMAKE_CXX_COMPILER`. The C++ compiler to use to compile the library.
- `--type` / `CMAKE_BUILD_TYPE`. Any of the build types recognized by CMake, *i.e.* debug, release, and so forth.
- `<build-dir>` / `-B<build-dir>`. The location of the build folder.
- `--xcmaxorder` / `XCFUN_MAX_ORDER`. Maximum derivative order, defaults to 6.
- `--pybindings` / `XCFUN_PYTHON_INTERFACE`. Enable compilation of Python bindings, defaults to `OFF`.
- `--static` / `BUILD_SHARED_LIBS`. Compile only the static library, defaults to `OFF`, building the shared library only.
- `ENABLE_TESTALL`. Whether to compile unit tests. `ON` by default. To toggle it `OFF` when using the `setup` script use `--cmake-options="-DENABLE_TESTALL=OFF"`.

1.4 Building the documentation

To build the documentation:

```
$ cd docs
$ make html
```

or:

```
$ sphinx-build docs _build -t html
```

1.5 Bumping versions

To bump a version you should edit the `cmake/custom/xcfun.cmake`, `src/version_info.hpp`, and `docs/conf.py` files.

CHAPTER

TWO

USING XCFUN

To use the library, you will need to:

- Link your executable to it. Either using the static, `libxcfun.a` or shared, `libxcfun.so`, version.
- For C/C++ hosts, include the header file `xcfun.h` where appropriate:

```
#include "XCFun/xcfun.h"
```

- For Fortran hosts, compile the `xcfun.f90` source file together with your sources. This will allow using the Fortran/C interoperability layer with:

```
use xcfun
```

2.1 Installing using Spack

XCFun can be installed in a Spack environment with:

```
spack env create myenv
spack env activate myenv
spack install xcfun
```

2.2 Installing using Conda

XCFun can be installed in a Conda environment with:

```
conda create -n myenv xcfun -c conda-forge
conda activate myenv
```

The package is built with derivatives up to 8th order and includes the Python bindings.

2.3 Integration with your build system

The set up of the build system for you code will change the details on how to achieve the points above. In the following, we provide minimalistic instructions for codes that use either CMake as their build system generator or plain `Makefile`.

2.3.1 CMake as build system

Note: You can find **complete**, **standalone** examples for C, C++, and Fortran in the `examples` folder.

If you use CMake as your build system, adding the command:

```
find_package(XCFun CONFIG)
```

in your `CMakeLists.txt` will let CMake search for an XCFun installation. CMake will honor the hint variable:

```
-DXCFun_DIR=<install-prefix>/share/cmake/XCFun
```

and set up the target `XCFun::xcfun` for you to link your target against:

```
target_link_libraries(<your-target-name>
  PRIVATE
    XCFun::xcfun
  )
```

For Fortran hosts the `xcfun.f90` will have to be compiled too. The following addition suffices:

```
target_sources(<your-target-name>
  PRIVATE
    ${XCFun_Fortran_SOURCES}
  )
```

2.3.2 Other build systems

You will need to set:

- The linker path:

```
-L<install-prefix>/lib64 -lxcfun
```

note that on some systems it might be `lib` rather than `lib64`.

- For C/C++ codes, the include path:

```
-I<install-prefix>/include
```

- For Fortran codes, the location of the Fortran/C interoperability source file `xcfun.f90`:

```
<install-prefix>/include/XCFun/xcfun.f90
```

2.4 Writing an interface

Note: Please, read the full *XCFun's application programming interface* documentation for a complete overview.

The library exposes an opaque type, *xcfun_t*, through which you can obtain the exchange-correlation functional derivatives to the desired order. To do so:

1. Create one *xcfun_t* object. There should be **only one** such object per thread and per XC functional. In C/C++ this is achieved with:

   ```
   xcfun_t * fun = xcfun_new();
   ```

   whereas in Fortran:

   ```
   use, intrinsic :: iso_c_binding

   type(c_ptr) :: fun

   fun = xcfun_new()
   ```

2. The *xcfun_t* object is now a blank slate. You will need to set the exchange-correlation admixture, *i.e.* which functional and which amount to use for exchange and correlation. This is achieved with calls to *xcfun_set()*:

   ```
   int ierr = 0;
   ierr = xcfun_set(fun, "blyp", 0.9);
   ierr = xcfun_set(fun, "pbec", 0.1);
   ```

   We have now set up the BLYP GGA functional.

3. Next, you will have to set up the evaluation strategy, *i.e.* which variables will be passed in as input to the functional, which outputs are expected, and the order of the derivatives to return upon evaluation. This can be done by calling *xcfun_eval_setup()*:

   ```
   ierr = xcfun_eval_setup(fun, XC_A_B_AX_AY_AX_BX_BY_BZ, XC_PARTIAL_DERIVATIVES, 1);
   ```

   The convenience function *xcfun_user_eval_setup()* is also available. With this set up, we will obtain functional derivatives of the BLYP functional up to first order, using α and β variables and partial derivatives.

4. We are now ready to run the evaluation and for this you will have to allocate a properly sized chunk of memory. The function *xcfun_output_length()* will return how large such a scratch array has to be:

   ```
   int nout = xcfun_output_length(fun);

   double * output = malloc(sizeof(double) * nout);
   ```

5. Finally, we proceed to the evaluation. We call *xcfun_eval()* with an array of density values:

   ```
   xcfun_eval(fun, d_elements, output);
   ```

6. The important last step is to clean up the used heap memory. *xcfun_delete()* is the function to call:

   ```
   free(output);
   xcfun_delete(fun);
   ```

2.4.1 Input, output and units

The library uses atomic units for all input and output variables.

The XC energy density and derivatives can be evaluated using local spin-up (α) and spin-down (β) quantities. In the most general case these are:

- n_α The spin-up electron number density.
- n_β The spin-down density.
- $\sigma_{\alpha\alpha} = \nabla n_\alpha . \nabla n_\alpha$ The square magnitude of the spin-up density gradient.
- $\sigma_{\alpha\beta} = \nabla n_\alpha . \nabla n_\beta$ The dot product between the spin-up and spin-down gradient vectors.
- $\sigma_{\beta\beta} = \nabla n_\beta . \nabla n_\beta$ The square magnitude of the spin-down density gradient.
- $\tau_\alpha = \frac{1}{2}\sum_i |\psi_{i\alpha}|^2$ The spin-up Kohn-Sham kinetic energy density.
- τ_β The spin-down Kohn-Sham kinetic energy density.

Alternatively you can use total density ($n = n_\alpha + n_\beta$) and spin density ($s = n_\alpha - n_\beta$) variables. These also have corresponding gradient and kinetic energy components. See *xcfun_set()* below for more information.

The output is given in graded reverse lexicographical order. For example a spin-polarized second order GGA functional will give 21 output elements, starting with the XC energy density. Symbolically we may write this as a list starting with the energy E, followed by five gradient elements $E_\alpha E_\beta E_{\sigma_{\alpha\alpha}} E_{\sigma_{\alpha\beta}} E_{\sigma_{\beta\beta}}$ and 15 second derivatives $E_{\alpha\alpha} E_{\alpha\beta} E_{\alpha\sigma_{\alpha\alpha}} ... E_{\beta\beta} E_{\beta\sigma_{\alpha\alpha}} ... E_{\sigma_{\beta\beta}\sigma_{\beta\beta}}$.

CHAPTER

THREE

MIGRATING TO THE NEW APPLICATION PROGRAMMERS' INTERFACE

This is a short guide to migrating to the new application programmers' interface (API) and build system for XCFun. We assume that you have successfully built and tested XCFun and installed it to `<install-prefix>`. The layout of the install tree will be as follows:

```
<install-prefix>/
                  ├── include
                  │   └── XCFun
                  │       ├── config.hpp
                  │       ├── densvars.hpp
                  │       ├── functional.hpp
                  │       ├── functionals
                  │       ├── specmath.hpp
                  │       ├── XCFunctional.hpp
                  │       ├── XCFunExport.h
                  │       ├── xcfun.f90
                  │       ├── xcfun.h
                  │       └── xcint.hpp
                  ├── lib
                  │   └── python
                  │       └── xcfun
                  ├── lib64
                  │   ├── libxcfun.a
                  │   ├── libxcfun.so -> libxcfun.so.2
                  │   └── libxcfun.so.2
                  └── share
                      └── cmake
                          └── XCFun
```

3.1 C/C++ host programs

Types and function signatures are in the header file `xcfun.h`.

In your source code, apply the following changes:

- Remove any of the calls to the functions that have been removed from the API. **Open an issue** if these functions are essential to your workflow and you would like them to be reinstated.
- Replace `xc_functional` with `xcfun_t *`.
- Replace `xc_new_functional` with `xcfun_new`.

- Replace `xc_enumerate_parameters` with `xcfun_enumerate_parameters`.
- Replace `xc_enumerate_aliases` with `xcfun_enumerate_aliases`.
- Replace `xc_set` with `xcfun_set`.
- Replace `xc_get` with `xcfun_get`.
- Replace `xc_describe_short` with `xcfun_describe_short`.
- Replace `xc_describe_long` with `xcfun_describe_long`.
- Replace `xc_is_gga` with `xcfun_is_gga`.
- Replace `xc_is_metagga` with `xcfun_is_metagga`.
- Replace `xc_eval_setup` with `xcfun_eval_setup`.
- Replace `xc_user_eval_setup` with `xcfun_user_eval_setup`.
- Replace `xc_input_length` with `xcfun_input_length`.
- Replace `xc_output_length` with `xcfun_output_length`.
- Replace `xc_eval` with `xcfun_eval`.
- Replace `xc_eval_vec` with `xcfun_eval_vec`.

3.2 Fortran host programs

The Fortran/C interoperability layer for types and function signatures is in the source file `xcfun.f90`.

In your source code, apply the following changes:

- Use the intrinsic `iso_c_binding` module: `use, intrinsic :: iso_c_binding`.
- Remove any of the calls to the functions that have been removed from the API. **Open an issue** if these functions are essential to your workflow and you would like them to be reinstated.
- You should call the intrinsic `trim` on functions returning strings: `xcfun_version`, `xcfun_splash`, `xcfun_authors`, `xcfun_enumerate_paramters`, `xcfun_enumerate_aliases`, `xcfun_describe_short`, `xcfun_describe_long`.
- Replace the type for the `xc_functional` object (now `xcfun_t *`) from `integer` to `type(c_ptr)`.
- Replace `xc_new_functional` with `xcfun_new`.
- Replace `xc_enumerate_parameters` with `xcfun_enumerate_parameters`.
- Replace `xc_enumerate_aliases` with `xcfun_enumerate_aliases`.
- Replace `xc_set` with `xcfun_set`.
- Replace `xc_get` with `xcfun_get`.
- Replace `xc_describe_short` with `xcfun_describe_short`.
- Replace `xc_describe_long` with `xcfun_describe_long`.
- Replace `xc_is_gga` with `xcfun_is_gga`.
- Replace `xc_is_metagga` with `xcfun_is_metagga`.
- Replace `xc_eval_setup` with `xcfun_eval_setup`.
- Replace `xc_user_eval_setup` with `xcfun_user_eval_setup`.

- Replace `xc_input_length` with `xcfun_input_length`.
- Replace `xc_output_length` with `xcfun_output_length`.
- Replace `xc_eval` with `xcfun_eval`.
- Replace `xc_eval_vec` with `xcfun_eval_vec`.

CHAPTER

FOUR

XCFUN'S APPLICATION PROGRAMMING INTERFACE

The library is written in C++, but can also be directly used in a C or Fortran project through its application programming interface. The C interface is exposed described in the `api/xcfun.h`, while the Fortran interface is described in the module file `api/xcfun.f90`. This documentation describes the C API. The Fortran API is written as a wrapper to the C API and has the same behavior.

4.1 Types and type definitions

struct **XCFunctional**

Exchange-correlation functional.

typedef struct xcfun_s **xcfun_t**

Opaque handle to a *XCFunctional* object.

Note: This type definition is a workaround to have the opaque `xcfun_t` `struct` available to C.

4.2 Functions

const char ***xcfun_version**()

The version of XCFun in use.

Returns
the version of XCFun

const char ***xcfun_splash**()

The XCFun splash screen.

Return a multi-line string describing the library. This functions shows the code attribution and literature citation. It should be called when initializing XCFun in client code, so that your users find the right citation for the library.

Returns
A `char` array with the XCFun splash screen.

const char ***xcfun_authors**()

The XCFun splash screen.

Returns
A `char` array with the current list of XCFun authors.

int **xcfun_test**()

Test XCFun.

Run all internal tests and return the number of failed tests.

Returns
the number of failed tests.

bool **xcfun_is_compatible_library**()

Whether the library is compatible with the header file Checks that the compiled library and header file version match. Host should abort when that is not the case.

> **Warning:** This function should be called **before** instantiating any *XCFunctional* object.

xcfun_vars **xcfun_which_vars**(const unsigned int func_type, const unsigned int dens_type, const unsigned int laplacian, const unsigned int kinetic, const unsigned int current, const unsigned int explicit_derivatives)

Obtain correct value of `xcfun_vars enum`.

This routine encodes the different options bitwise. Each legitimate combination is then converted to the corresponding enum value.

7	6	5	4	3	2	1	0	
0	0							LDA
0	1							GGA
1	0							metaGGA
1	1							Taylor
		0	0					ρ_α
		0	1					ρ
		1	0					ρ_α and ρ_β
		1	1					ρ and s
				0				no laplacian
				1				laplacian required
					0			no kinetic energy density
					1			kinetic energy density required
						0		no current density required
						1		current density required
							0	γ-type partial derivatives
							1	explicit partial derivatives

Parameters

- **func_type** – **[in]** LDA (0), GGA (1), metaGGA (2), taylor (3)
- **dens_type** – **[in]** Alpha (A,0), Rho (N,1), Alpha&Beta (A_B,2), Rho&Spin (N_S,3)

- **laplacian** – **[in]** (0 not required / 1 required)
- **kinetic** – **[in]** (0 not required / 1 required)
- **current** – **[in]** (0 not required / 1 required)
- **explicit_derivatives** – **[in]** (0 not required / 1 required)

Returns
XC functional variables to use

xcfun_mode **xcfun_which_mode**(const unsigned int mode_type)

Obtain correct value of `xcfun_mode` `enum`.

Parameters
mode_type – **[in]** Partial derivatives (1), Potential (2), Contracted (3)

Returns
The XC functional evaluation mode

const char ***xcfun_enumerate_parameters**(int param)

Describe XC functional parameters.

Parameters
param – **[in]** the parameter to describe. `param` >= 0.

Returns
description of the given parameter, or `NULL` is `param` is too large.

const char ***xcfun_enumerate_aliases**(int n)

Describe XC functional aliases.

Parameters
n – **[in]** the alias to describe. `n` >= 0.

Returns
description of the given alias, or `NULL` is `n` is too large.

const char ***xcfun_describe_short**(const char *name)

Short description of the XC functional.

Parameters
name – **[in]**

Returns
short description of the functional.

const char ***xcfun_describe_long**(const char *name)

Long description of the XC functional.

Parameters
name – **[in]**

Returns
long description of the functional.

xcfun_t ***xcfun_new**()

Create a new XC functional object.

Create a new functional object. The creation of this object may be rather slow; create an object once for each calculation, not once for each grid point.

Returns
A `xcfun_t` object.

void **xcfun_delete**(*xcfun_t* *fun)

Delete a XCFun functional.

Parameters
fun – **[inout]** the XCFun functional to be deleted

int **xcfun_set**(*xcfun_t* *fun, const char *name, double value)

Set a parameter in the XC functional.

Parameters

- **fun** – **[inout]**
- **name** – **[in]**
- **value** – **[in]**

Returns
error code (0 means normal exit)

int **xcfun_get**(const *xcfun_t* *fun, const char *name, double *value)

Get weight of given functional in the current setup.

Parameters

- **fun** – **[in]** the functional object
- **name** – **[in]** functional name to test, aliases not supported
- **value** – **[out]** weight of functional

Returns
`0` if `name` is a valid functional, `-1` if not. See `list_of_functionals.hpp` for valid functional names.

bool **xcfun_is_gga**(const *xcfun_t* *fun)

Is the XC functional GGA?

Parameters
fun – **[inout]**

Returns
Whether `fun` is a GGA-type functional

bool **xcfun_is_metagga**(const *xcfun_t* *fun)

Is the XC functional GGA?

Parameters
fun – **[inout]**

Returns
Whether `fun` is a metaGGA-type functional

int **xcfun_eval_setup**(*xcfun_t* *fun, *xcfun_vars* vars, *xcfun_mode* mode, int order)

Set up XC functional evaluation variables, mode, and order.

Parameters

- **fun** – **[inout]** XC functional object
- **vars** – **[in]** evaluation variables

- **mode** – **[in]** evaluation mode
- **order** – **[in]** order of the derivative requested (order=1 is the xc potential)

Returns
some combination of XC_E* if an error occurs, else 0

int **xcfun_user_eval_setup**(*xcfun_t* *fun, const int order, const unsigned int func_type, const unsigned int dens_type, const unsigned int mode_type, const unsigned int laplacian, const unsigned int kinetic, const unsigned int current, const unsigned int explicit_derivatives)

Host program-friendly set up of the XC functional evaluation variables, mode, and order.

Parameters

- **fun** – **[inout]** XC functional object
- **order** – **[in]** order of the derivative requested (order 0 (functional), 1 (potential), 2 (hessian), ...)
- **func_type** – **[in]** LDA (0), GGA (1), metaGGA (2), taylor (3)
- **dens_type** – **[in]** Alpha (A,0), Rho (N,1), Alpha&Beta (A_B,2), Rho&Spin (N_S,3)
- **mode_type** – **[in]** Partial derivatives (1), Potential (2), Contracted (3)
- **laplacian** – **[in]** (0 not required / 1 required)
- **kinetic** – **[in]** (0 not required / 1 required)
- **current** – **[in]** (0 not required / 1 required)
- **explicit_derivatives** – **[in]** (0 not required / 1 required)

Returns
some combination of XC_E* if an error occurs, else 0

int **xcfun_input_length**(const *xcfun_t* *fun)

Length of the density[] argument to xcfun_eval

Parameters
fun – **[inout]** XC functional object

Returns
some combination of XC_E* if an error occurs, else 0

int **xcfun_output_length**(const *xcfun_t* *fun)

Length of the result[] argument to xcfun_eval

Note: All derivatives up to order are calculated, not only those of the particular order.

Parameters
fun – **[inout]** XC functional object

Returns
Return the number of output coefficients computed by xc_eval().

void **xcfun_eval**(const *xcfun_t* *fun, const double density[], double result[])

Evaluate the XC functional for given density at a point.

Note: In contracted mode density is of dimension $2^{\text{order}} * N_{\text{vars}}$

Parameters

- **fun** – **[inout]** XC functional object
- **density** – **[in]**
- **result** – **[inout]**

void **xcfun_eval_vec**(const *xcfun_t* *fun, int nr_points, const double *density, int density_pitch, double *result, int result_pitch)

Evaluate the XC functional for given density on a set of points.

Note: In contracted mode density is of dimension $2^{\text{order}} * N_{\text{vars}}$

Parameters

- **fun** – **[inout]** XC functional object
- **nr_points** – **[in]** number of points in the evaluation set.
- **density** – **[in]**
- **density_pitch** – **[in]** `density[start_of_second_point] - density[start_of_first_point]`
- **result** – **[inout]**
- **result_pitch** – **[in]** `result[start_of_second_point] - result[start_of_first_point]`

4.3 Enumerations

enum **xcfun_mode**

Evaluation mode for functional derivatives.

Values:

enumerator **XC_MODE_UNSET**

Need to be zero for default initialized structs

enumerator **XC_PARTIAL_DERIVATIVES**

???

enumerator **XC_POTENTIAL**

???

enumerator **XC_CONTRACTED**

???

enumerator **XC_NR_MODES**

???

enum **xcfun_vars**

Types of variables to define a functional.

The XC energy density and derivatives can be evaluated using a variety of variables and variables combinations. The variables in this `enum` are named as:

- `XC_` prefix
- Tag for density variables.
- Tag for gradient variables.
- Tag for Laplacian variables.
- Tag for kinetic energy density variables.
- Tag for current density variables.

XCFun recognizes the following basic variables:

- `A`, the spin-up electron number density: n_α
- `B`, the spin-down electron number density: n_β
- `GAA`, the square magnitude of the spin-up density gradient: $\sigma_{\alpha\alpha} = \nabla n_\alpha . \nabla n_\alpha$
- `GAB`, the dot product of the spin-up and spin-down density gradients: $\sigma_{\alpha\beta} = \nabla n_\alpha . \nabla n_\beta$
- `GBB`, the square magnitude of the spin-down density gradient: $\sigma_{\beta\beta} = \nabla n_\beta . \nabla n_\beta$
- `LAPA`, the Laplacian of the spin-up density: $\nabla^2 n_\alpha$
- `LAPB`, the Laplacian of the spin-down density: $\nabla^2 n_\beta$
- `TAUA`, the spin-up Kohn-Sham kinetic energy density: $\tau_\alpha = \frac{1}{2}\sum_i |\psi_{i\alpha}|^2$
- `TAUB`, the spin-down Kohn-Sham kinetic energy density: $\tau_\beta = \frac{1}{2}\sum_i |\psi_{i\beta}|^2$
- `JPAA`, the spin-up current density: $\mathbf{j}_{\alpha\alpha}$
- `JPBB`, the spin-down current density: $\mathbf{j}_{\beta\beta}$

The following quantities are also recognized:

- `N`, the number density: $n = n_\alpha + n_\beta$
- `S`, the spin density: $s = n_\alpha - n_\beta$
- `GNN`, the square magnitude of the density gradient: $\sigma_{nn} = \nabla n . \nabla n$
- `GSS`, the dot product of the number and spin density gradients: $\sigma_{ns} = \nabla n . \nabla s$
- `GNS`, the square magnitude of the spin density gradient: $\sigma_{ss} = \nabla s . \nabla s$

- LAPN, the Laplacian of the density: $\nabla^2 n$
- LAPS, the Laplacian of the spin density: $\nabla^2 s$
- TAUN, the Kohn-Sham kinetic energy density: τ_n
- TAUS, the spin Kohn-Sham kinetic energy density: τ_s

XC functionals depending on the gradient of the density can furthermore be defined to use the (x, y, z) components of the gradient explicitly.

Values:

enumerator **XC_VARS_UNSET**

Not defined

enumerator **XC_A**

LDA with n_α

enumerator **XC_N**

LDA with n

enumerator **XC_A_B**

LDA with n_α and n_β

enumerator **XC_N_S**

LDA with n and s

enumerator **XC_A_GAA**

GGA with grad^2 alpha

enumerator **XC_N_GNN**

GGA with grad^2 rho

enumerator **XC_A_B_GAA_GAB_GBB**

GGA with grad^2 alpha & beta

enumerator **XC_N_S_GNN_GNS_GSS**

GGA with grad^2 rho and spin

enumerator **XC_A_GAA_LAPA**

metaGGA with grad^2 alpha laplacian

enumerator **XC_A_GAA_TAUA**

metaGGA with grad^2 alpha kinetic

enumerator **XC_N_GNN_LAPN**

metaGGA with grad^2 rho laplacian

enumerator **XC_N_GNN_TAUN**
metaGGA with grad^2 rho kinetic

enumerator **XC_A_B_GAA_GAB_GBB_LAPA_LAPB**
metaGGA with grad^2 alpha & beta laplacian

enumerator **XC_A_B_GAA_GAB_GBB_TAUA_TAUB**
metaGGA with grad^2 alpha & beta kinetic

enumerator **XC_N_S_GNN_GNS_GSS_LAPN_LAPS**
metaGGA with grad^2 rho and spin laplacian

enumerator **XC_N_S_GNN_GNS_GSS_TAUN_TAUS**
metaGGA with grad^2 rho and spin kinetic

enumerator **XC_A_B_GAA_GAB_GBB_LAPA_LAPB_TAUA_TAUB**
metaGGA with grad^2 alpha & beta laplacian kinetic

enumerator **XC_A_B_GAA_GAB_GBB_LAPA_LAPB_TAUA_TAUB_JPAA_JPBB**
metaGGA with grad^2 alpha & beta laplacian kinetic current

enumerator **XC_N_S_GNN_GNS_GSS_LAPN_LAPS_TAUN_TAUS**
metaGGA with grad^2 rho and spin laplacian kinetic

enumerator **XC_A_AX_AY_AZ**
GGA with gradient components alpha

enumerator **XC_A_B_AX_AY_AZ_BX_BY_BZ**
GGA with gradient components alpha & beta

enumerator **XC_N_NX_NY_NZ**
GGA with gradient components rho

enumerator **XC_N_S_NX_NY_NZ_SX_SY_SZ**
GGA with gradient components rho and spin

enumerator **XC_A_AX_AY_AZ_TAUA**
metaGGA with gradient components alpha

enumerator **XC_A_B_AX_AY_AZ_BX_BY_BZ_TAUA_TAUB**
metaGGA with gradient components alpha & beta

enumerator **XC_N_NX_NY_NZ_TAUN**
metaGGA with gradient components rho

enumerator **XC_N_S_NX_NY_NZ_SX_SY_SZ_TAUN_TAUS**

metaGGA with gradient components rho and spin

enumerator **XC_A_2ND_TAYLOR**

2nd order Taylor coefficients of alpha density, 1+3+6=10 numbers, rev gradlex order

enumerator **XC_A_B_2ND_TAYLOR**

2nd order Taylor expansion of alpha and beta densities (first alpha, then beta) 20 numbers

enumerator **XC_N_2ND_TAYLOR**

2nd order Taylor rho

enumerator **XC_N_S_2ND_TAYLOR**

2nd order Taylor rho and spin

enumerator **XC_NR_VARS**

Number of variables

4.4 Preprocessor definitions and global variables

XCFUN_API_VERSION

Version of the XCFun API

XCFUN_MAX_ORDER

Maximum differentiation order for XC kernels

constexpr auto `xcfun::XCFUN_TINY_DENSITY` = 1e-14

Used for regularizing input

constexpr auto `xcfun::XC_EORDER` = 1

Invalid order for given mode and vars

constexpr auto `xcfun::XC_EVARS` = 2

Invalid vars for functional type (ie. lda vars for gga)

constexpr auto `xcfun::XC_EMODE` = 4

Invalid mode for functional type (ie. potential for mgga)

CHAPTER

FIVE

EXCHANGE-CORRELATION FUNCTIONALS

The following functionals are implemented within XCFun

SLATERX	Slater LDA exchange
PW86X	PW86 exchange
VWN5C	VWN5 LDA Correlation functional
PBEC	PBE correlation functional
PBEX	PBE Exchange Functional
BECKEX	Becke 88 exchange
BECKECORRX	Becke 88 exchange correction
BECKESRX	Short range Becke 88 exchange
LDAERFX	Short-range spin-dependent LDA exchange functional
LDAERFC	Short-range spin-dependent LDA correlation functional
LDAERFC_JT	Short-range spin-unpolarized LDA correlation functional
LYPC	LYP correlation
OPTX	OPTX Handy & Cohen exchange
REVPBEX	Revised PBE Exchange Functional
RPBEX	RPBE Exchange Functional
SPBEC	sPBE correlation functional
VWN_PBEC	PBE correlation functional using VWN LDA correlation.
KTX	KT exchange GGA correction
TFK	Thomas-Fermi Kinetic Energy Functional
PW91X	Perdew-Wang 1991 GGA Exchange Functional
PW91K	PW91 GGA Kinetic Energy Functional
PW92C	PW92 LDA correlation
M05X	M05 exchange
M05X2X	M05-2X exchange
M06X	M06 exchange
M06X2X	M06-2X exchange
M06LX	M06-L exchange
M06HFX	M06-HF exchange
M05X2C	M05-2X Correlation
M05C	M05 Correlation
M06C	M06 Correlation
M06LC	M06-L Correlation
M06X2C	M06-2X Correlation
TPSSC	TPSS original correlation functional
TPSSX	TPSS original exchange functional
REVTPSSC	Revised TPSS correlation functional

continues on next page

Table 1 – continued from previous page

REVTPSSX	Reviewed TPSS exchange functional
PZ81C	PZ81 LDA correlation
P86C	P86C GGA correlation
RANGESEP_MU	Range separation inverse length [1/a0]
EXX	Amount of exact (HF like) exchange (must be provided externally)

5.1 Implementing a new XC functional

Warning: To be written

5.2 Introducing new variables

Warning: To be written

CHAPTER

SIX

CHANGE LOG

6.1 Version 2.1.1 - 2020-11-12

6.1.1 Changed

- Linux and macOS continuous integration testing is run on GitHub actions. See PR #145

6.1.2 Fixed

- We polished the installation of header files, CMake target export files, and Python module. These are especially relevant for Conda packaging XCFun. See PR #143
- A numerical issue with SCAN functionals and small density gradients was fixed by James Furness (@JFurness1). See issue #144 reported by Xing Zhang (@fishjojo) and subsequent PR #146 for the fix.

6.2 Version 2.1.0 - 2020-09-18

- Many new functionals in the SCAN family have been added. Thanks to James Furness for the contribution. See PR #140
- The library is now available both as a Spack and a Conda package.
- The library can now be *natively* compiled on Linux, macOS, and Windows.

6.2.1 Changed

- **BREAKING** CMake >= 3.14 is required to configure the code.

6.3 Version 2.0.2 - 2020-07-15

6.3.1 Fixed

- VWN3 functional has been fixed for the spin-polarized case. It previously gave wrong results when alpha and beta densities differed. Thanks to Zhenyu Zhu for reporting the problem and also suggesting the solution. See PR #134 and issue #132.

6.4 Version 2.0.1 - 2020-05-06

6.4.1 Fixed

- We removed the `DEBUG_POSTFIX` property from the properties on the `xcfun` target. This was leading to build failures when using the library through CMake `FetchContent` with mixed release/debug mode.

6.5 Version 2.0.0 - 2020-04-14

6.5.1 Changed

- **BREAKING** The build system will only produce a shared (default) or static library. Compilation of the static library can be requested by setting `BUILD_SHARED_LIBS` to `OFF`.
- macOS CI testing was moved to Azure Pipelines.
- The dependency on pybind11 was bumped to v2.5.0

6.5.2 Fixed

- We corrected a number of wrinkles in the handling of symbol visibility in the shared library.

6.6 Version 2.0.0a7 - 2020-04-10

6.6.1 Fixed

- Address warnings from compilers. Fix #90.

6.7 Version 2.0.0a6 - 2020-02-23

6.7.1 Fixed

- Compilation with GCC 5.4.0.

6.8 Version 2.0.0a5 - 2020-02-20

6.8.1 Fixed

- Handling of 64-bit integers in the Fortran interface.

6.9 Version 2.0.0a4 - 2020-02-02

6.9.1 Fixed

- The API function `xcfun_get` accepts a single in-out `double` parameter. It was erroneously declared to accept an array of `double`-s instead.

6.10 Version 2.0.0a3 - 2020-01-31

We have introduced a number of breaking changes, motivated by the need to modernize the library. See the migration guide.

6.10.1 Added

- Up-to-date API documentation generated with Doxygen, breathe, and Sphinx.
- Up-to-date documentation on how to build and develop XCFun.
- Up-to-date documentation on how to use XCFun in your code.
- API functions `xcfun_which_vars` and `xcfun_which_mode`.
- A full example, based on CMake as build system generator, showing how to use the library from a C++ host. Thanks @stigrj!
- A full example, based on CMake as build system generator, showing how to use the library from a C host.
- A full example, based on CMake as build system generator, showing how to use the library from a Fortran host.

6.10.2 Changed

- **BREAKING** All API functions are uniformly namespaced with the `xcfun_` prefix.
- **BREAKING** The Fortran interface has been completely rewritten using `iso_c_binding`: the library can now be compiled without the use of neither a C nor a Fortran compiler. :confetti_ball:
- **BREAKING** CMake option `XCFun_XC_MAX_ORDER` renamed to `XCFUN_MAX_ORDER`. New default value of 6.
- **BREAKING** CMake option `XCFun_ENABLE_PYTHON_INTERFACE` renamed to `XCFUN_PYTHON_INTERFACE`.

6.10.3 Deprecated

6.10.4 Removed

- **BREAKING** API functions `xc_serialize`, `xc_deserialize`, `xc_set_fromstring`, and `xc_derivative_index`.
- **BREAKING** The CMake options `ENABLE_FC_SUPPORT` and `ENABLE_64BIT_INTEGERS`.

6.10.5 Fixed

6.10.6 Security

6.11 Version 2.0.0a2 - 2020-01-21

6.12 Version 2.0.0a1 - 2019-12-15

6.12.1 Added

- A user-friendly API function to set up functional evaluation `xc_user_eval_setup`. Thanks @ilfreddy.

6.12.2 Changed

- **BREAKING** A compiler compliant with the C++11 (or later) standard is required.
- **BREAKING** CMake >= 3.11 is required to configure the code.
- **BREAKING** The Python bindings are now generated using pybind11 instead of SWIG. The dependency will be fetched at configuration time if not found on your system.
- **BREAKING** The Fortran interface is no longer build with the code, but shipped as a separate file to be compiled within your own Fortran code.

CHAPTER

SEVEN

INDICES AND TABLES

- genindex
- modindex
- search

INDEX

X