DFT-D4

Grimme group

Jul 17, 2023
# CONTENTS

1 References 3
  1.1 Recipes ......................................................... 3
  1.2 API documentation ............................................ 6

Python Module Index 25

Index 27
This page describes the usage and functionality of the \texttt{dftd4} library. The \texttt{dftd4} project provides an implementation of the generally applicable, charge dependent London-dispersion correction, termed DFT-D4.
1.1 Recipes

This section deals with solving specific task with DFT-D4 by providing step by step recipes.

1.1.1 Installing DFT-D4

This guide will walk you through installing the latest version of DFT-D4.

Installing from conda-forge

This project is packaged for the conda package manager and available on the conda-forge channel. To install the conda package manager we recommend the miniforge installer. If the conda-forge channel is not yet enabled, add it to your channels with

```
conda config --add channels conda-forge
conda config --set channel_priority strict
```

Once the conda-forge channel has been enabled, DFT-D4 can be installed with conda:

```
conda install dftd4
```

or with mamba:

```
mamba install dftd4
```

It is possible to list all of the versions of DFT-D4 available on your platform with conda:
conda search dftd4 --channel conda-forge

or with `mamba`:

mamba search dftd4 --channel conda-forge

Alternatively, `mamba repoquery` may provide more information:

```bash
# Search all versions available on your platform:
mamba repoquery search dftd4 --channel conda-forge

# List packages depending on `dftd4`:
mamba repoquery whoneeds dftd4 --channel conda-forge

# List dependencies of `dftd4`:
mamba repoquery depends dftd4 --channel conda-forge
```

**FreeBSD ports**

A port for FreeBSD is available

```bash
pkg install science/dftd4
```

In case no package is available build the port using

```bash
cd /usr/ports/science/dftd4
make install clean
```

For more information see the `dftd4` port details.

**Building from source**

To build this project from the source code in this repository you need to have

- a Fortran compiler supporting Fortran 2008
- One of the supported build systems
  - `meson` version 0.55 or newer
  - `CMake` version 3.9 or newer

First, get the source by cloning the repository

```bash
git clone https://github.com/dftd4/dftd4
cd dftd4
```
Using Meson

To build this project with meson a build-system backend is required, i.e. ninja version 1.7 or newer. Setup a build with

```bash
meson setup _build --prefix=/path/to/installation
```

You can select the Fortran compiler by the FC environment variable. To compile the project run

```bash
meson compile -C _build
```

DFT-D4 comes with a comprehensive test suite. Run the tests with

```bash
meson test -C _build --print-errorlogs
```

Finally, you can install DFT-D4 with

```bash
meson install -C _build
```

Using CMake

While meson is the preferred way to build this project it also offers CMake support. Configure the CMake build with

```bash
cmake -B _build -GNinja -DCMAKE_INSTALL_PREFIX=/path/to/installation
```

Similar to meson the compiler can be selected with the FC environment variable. You can build the project using

```bash
cmake --build _build
```

DFT-D4 comes with a comprehensive test suite. Run the tests with

```bash
ctest --test-dir _build --output-on-failure
```

Finally, you can install DFT-D4 with

```bash
cmake --install _build
```

1.1.2 Using DFT-D4 in Vasp

To use the D4 dispersion correction in Vasp you need to a compile the dftd4 package using the same Fortran compiler as used for Vasp and enable the API compatibility needed for linking the Vasp-side interface for dftd4. Checkout the instructions for installing dftd4 at Installing DFT-D4. To use dftd4 in Vasp the compatibility layer for the 2.5.x API has to be enable with -Dapi_v2=true (meson) or -DWITH_API_V2=ON (CMake).

**Important:** It is important to build dftd4 with the same Fortran compiler you build Vasp with.

After you completed the installation of dftd4, make sure it is findable by `pkg-config`, you can check by running:

```bash
pkg-config --modversion dftd4
```

If your dftd4 installation is not findable, you have to update your environment variables. One option is to provide a module file for your dftd4 installation. The example module file below can be placed in your MODULEPATH to provide

1.1. Recipes
access to an installation in ~/opt/dftd4/3.5.0. Retry the above comment after loading the dftd4 module and adjust
the module file until pkg-config finds your installation.

```lua
-- dftd4/3.5.0.lua
local name = "dftd4"
local version = "3.5.0"
local prefix = pathJoin(os.getenv("HOME"), "opt", name, version)
local libdir = "lib" -- or lib64

whatis("Name" : ".. name)
whatis("Version" : ".. version")
whatis("Description" : Generally applicable charge dependent London dispersion correction →")
whatis("URL" : https://github.com/dftd4/dftd4")

prepend_path("PATH", pathJoin(prefix, "bin"))
prepend_path("MANPATH", pathJoin(prefix, "share", "man"))
prepend_path("CPATH", pathJoin(prefix, "include"))
prepend_path("LIBRARY_PATH", pathJoin(prefix, libdir))
prepend_path("LD_LIBRARY_PATH", pathJoin(prefix, libdir))
prepend_path("PKG_CONFIG_PATH", pathJoin(prefix, libdir, "pkgconfig"))
```

To enable support for D4 in Vasp add the following lines to the Makefile:

```
CPP_OPTIONS += -DDFTD4
LLIBS += $(shell pkg-config --libs dftd4)
INCS += $(shell pkg-config --cflags dftd4)
```

### 1.2 API documentation

DFT-D4 aims to provide first class API support Fortran, C and Python. Other programming languages should try to
interface via one of those three APIs. To provide first class API support for a new language the interface specification
should be available from the meson build files.

#### 1.2.1 Fortran API

The *dftd4* library seamlessly integrates with other Fortran projects via module interfaces,

**Note:** Generally, all quantities used in the library are stored in atomic units.
Handling of geometries and structure

The basic infrastructure to handle molecular and periodic structures is provided by the modular computation tool chain library. The library provides a structure type which is used to represent all geometry related informations in dftd4. A structure type can be constructed from arrays or read from a file.

The constructor is provided with the generic interface new and takes an array of atomic numbers (integer) or element symbols (character(len=*)) as well as the cartesian coordinates in Bohr. Additionally, the molecular charge and the number of unpaired electrons can be provided the charge and uhf keyword, respectively. To create a periodic structure the lattice parameters can be passed as 3 by 3 matrix with the lattice keyword.

An example for using the constructor is given here

```fortran
subroutine example
  use mctc_env, only : wp
  use mctc_io, only : structure_type, new
  implicit none
  type(structure_type) :: mol
  real(wp), allocatable :: xyz(:, :)
  integer, allocatable :: num(:)

  num = [6, 1, 1, 1, 1]
  xyz = reshape([ &
    0.00000000000000_wp, -0.00000000000000_wp, 0.00000000000000_wp, &
    -1.19220800552211_wp, 1.19220800552211_wp, 1.19220800552211_wp, &
    1.19220800552211_wp, -1.19220800552211_wp, 1.19220800552211_wp, &
    -1.19220800552211_wp, -1.19220800552211_wp, -1.19220800552211_wp, &
    1.19220800552211_wp, 1.19220800552211_wp, -1.19220800552211_wp], &
    [3, size(num)])

  call new(mol, num, xyz, charge=0.0_wp, uhf=0)

  ! ...
end subroutine example
```

To interact with common input file formats for structures the read_structure procedure is available. The file type is inferred from the name of the file automatically or if a file type hint is provided directly from the enumerator of available file types. The read_structure routine can also use an already opened unit, but in this case the file type hint is mandatory to select the correct format to read from.

```fortran
subroutine example
  use mctc_env, only : error_type
  use mctc_io, only : structure_type, read_structure, file_type
  implicit none
  type(structure_type) :: mol
  type(error_type), allocatable :: error
  character(len=:), allocatable :: input

  input = "struc.xyz"

  call read_structure(mol, input, error, file_type%xyz)
  if (allocated(error)) then
    print '(a)', error%message
    stop 1
  end if
end subroutine example
```

(continues on next page)
The structure type as well as the error type are using only allocatable members and can therefore be used without requiring explicit deconstruction.

Certain members of the structure type should be considered immutable, like the number of atoms (nat), the identifiers for unique atoms (id) and the boundary conditions (periodic). To change those specific structure parameters the structure type and all dependent objects should be reconstructed to ensure a consistent setup. Other properties, like the geometry (xyz), molecular charge (charge), number of unpaired electrons (uhf) and lattice parameters (lattice) can be changed without requiring to reconstruct dependent objects like calculators or restart data.

**Error handling**

The basic error handler is an allocatable derived type, available from mctc_env as error_type, which signals an error by its allocation status.

```fortran
use mctc_env, only : error_type, fatal_error
implicit none
type(error_type), allocatable :: error

call always_ok(error)
if (allocated(error)) then
  print '(a)', "Unexpected failure:", error%message
end if

call always_failed(error)
if (allocated(error)) then
  print '(a)', "Error:", error%message
end if
contains
  subroutine always_ok(error)
    type(error_type), allocatable, intent(out) :: error
  end subroutine always_ok

  subroutine always_failed(error)
    type(error_type), allocatable, intent(out) :: error
  call fatal_error(error, "Message associated with this error")
  end subroutine always_failed
end
```

An unhandled error might get dropped by the next procedure call.
1.2.2 C API

The C API bindings are provided by using the iso_c_binding intrinsic module. Generally, objects are exported as opaque pointers and can only be manipulated within the library. The API user is required delete all objects created in the library by using the provided deconstructor functions to avoid memory leaks.

Overall four classes of objects are provided by the library

- error handlers (dftd4_error), used to communicate exceptional conditions and errors from the library to the user
- structure containers (dftd4_structure), used to represent the system specific information and geometry data, only the latter are mutable for the user
- dispersion model objects (dftd4_model), general model for calculating dispersion related properties
- damping function objects (dftd4_param) polymorphic objects to represent the actual method parametrisation

Note: Generally, all quantities provided to the library are assumed to be in atomic units.

Error handling

typedef struct _dftd4_error *dftd4_error;
  Error handle class

The library provides a light error handle type (dftd4_error) for storing error information. The error handle requires only small overhead to construct and can only contain a single error.

The handler is represented by an opaque pointer and can only be manipulated by call from the library. The user of those objects is required to delete the handlers again using the library provided deconstructors to avoid memory leaks.

dftd4_error dftd4_new_error();
  Returns
  New allocation for error handle

Create new error handle object

int dftd4_check_error(dftd4_error error);
  Parameters
    • error – Error handle
  Returns
    Current status of error handle, non-zero in case of error

Check error handle status

void dftd4_get_error(dftd4_error error, char *buffer, const int *buffersize);
  Parameters
    • error – Error handle
    • buffer – Allocation to store error message in
    • buffersize – Maximum length of the buffer (optional)

Get error message from error handle
void dftd4_delete_error(dftd4_error *error);

Parameters
   • error – Error handle

Delete error handle object

Structure data

typedef struct _dftd4_structure *dftd4_structure;
Molecular structure data class

The structure data is used to represent the system of interest in the library. It contains immutable system specific information like the number of atoms, the unique atom groups and the boundary conditions as well as mutable geometry data like cartesian coordinates and lattice parameters.

dftd4_structure dftd4_new_structure(dftd4_error error, const int natoms, const int *numbers, const double *positions, const double *charge, const double *lattice, const bool *periodic);

Parameters
   • natoms – Number of atoms in the system
   • numbers – Atomic numbers of all atoms [natoms]
   • positions – Cartesian coordinates in Bohr [natoms, 3]
   • charge – Total molecular charge (optional)
   • lattice – Lattice parameters in Bohr [3, 3] (optional)
   • periodic – Periodic dimension of the system [3] (optional)

Returns
   New molecular structure data handle

Create new molecular structure data (quantities in Bohr)

void dftd4_delete_structure(dftd4_structure *mol);

Parameters
   • mol – Molecular structure data handle

Delete molecular structure data

void dftd4_update_structure(dftd4_error error, dftd4_structure mol, const double *positions, const double *lattice);

Parameters
   • error – Error handle
   • mol – Molecular structure data handle
   • positions – Cartesian coordinates in Bohr [natoms, 3]
   • lattice – Lattice parameters in Bohr [3, 3] (optional)

Update coordinates and lattice parameters (quantities in Bohr)
**Dispersion model**

typedef struct _dftd4_model *dftd4_model;

Dispersion model class

Instantiated for a given molecular structure type, it carries no information on the geometry but relies on the atomic species of the structure object. Recreating a structure object requires to recreate the dispersion model as well.

\[dftd4_model\ ip dftd4_new_d4_model(dftd4_error\ ip\ error, dftd4_structure\ ip\ mol);\]

**Parameters**

- `error` – Error handle
- `mol` – Molecular structure data handle

**Returns**

New dispersion model handle

Create new D4 dispersion model

\[dftd4_model\ ip\ dftd4_custom_d4_model(dftd4_error\ ip\ error, dftd4_structure\ ip\ mol, double\ ga, double\ gc, double\ wf);\]

**Parameters**

- `error` – Error handle
- `mol` – Molecular structure data handle
- `ga` – Charge scaling height
- `gc` – Charge scaling steepness
- `wf` – Weighting factor for coordination number interpolation

**Returns**

New dispersion model handle

Create new D4 dispersion model with custom parameters

void `dftd4_delete_model(dftd4_model*\ disp);`

**Parameters**

- `disp` – Dispersion model handle

Delete dispersion model

**Damping parameters**

typedef struct _dftd4_param *dftd4_param;

Damping parameter class

The damping parameter object determining the short-range behaviour of the dispersion correction. Standard damping parameters like the rational damping are independent of the molecular structure and can easily be reused for several structures or easily exchanged.

\[dftd4_param\ ip\ dftd4_new_rational_damping(dftd4_error\ ip\ error, double\ s6, double\ s8, double\ s9, double\ a1,\ double\ a2, double\ alp);\]

**Parameters**

- `error` – Error handle
• \( s_6 \) – Scaling factor for C6 contribution
• \( s_8 \) – Scaling factor for C8 contribution
• \( s_9 \) – Scaling factor for C9 contribution
• \( a_1 \) – Scaling factor for critical radii
• \( a_2 \) – Offset distance in Bohr for critical radii

Returns
New damping function parameter handle

Create new rational damping parameters

\[
dftd4_param\ dftd4_load_rational_damping(dftd4_error\ \text{error},\ char*\ \text{method},\ bool\ \text{mdb});
\]

Parameters
• \text{error} – Error handle
• \text{method} – Name of the method to load parameters for
• \text{mdb} – Use three-body specific parametrization

Returns
New damping function parameter handle

Load rational damping parameters from internal storage

\[
void\ dftd4_delete_param(dftd4_param*\ \text{param});
\]

Parameters
• \text{param} – Damping function parameter handle

Delete damping parameters

Calculation entrypoints

To evaluate dispersion energies or related properties the \textit{dftd4_get_dispersion} procedure and similar can be used.

\[
void\ dftd4_get_properties(dftd4_error\ \text{error},\ dftd4_structure\ \text{mol},\ dftd4_model\ \text{disp},\ double*\ \text{cn},\ double*\ \text{charges},\ double*\ \text{c6},\ double*\ \text{alpha});
\]

Parameters
• \text{error} – Error handle
• \text{mol} – Molecular structure data handle
• \text{disp} – Dispersion model handle
• \text{cn} – Coordination number for all atoms [natoms]
• \text{charges} – Partial charges for all atoms [natoms]
• \text{c6} – C6 coefficients for all atom pairs [natoms, natoms]
• \text{alpha} – Static polarizibilities for all atoms [natoms]

Evaluate properties related to the dispersion model
void dftd4_get_dispersion(dftd4_error error, dftd4_structure mol, dftd4_model disp, dftd4_param param, double *energy, double *gradient, double *sigma);

Parameters

- **error** – Error handle
- **mol** – Molecular structure data handle
- **disp** – Dispersion model handle
- **param** – Damping function parameter handle
- **energy** – Dispersion energy
- **gradient** – Dispersion gradient [natoms, 3] (optional)
- **sigma** – Dispersion strain derivatives [3, 3] (optional)

Evaluate the dispersion energy and its derivatives

void dftd4_get_pairwise_dispersion(dftd4_error error, dftd4_structure mol, dftd4_model disp, dftd4_param param, double *pair_energy2, double *pair_energy3);

Parameters

- **error** – Error handle
- **mol** – Molecular structure data handle
- **disp** – Dispersion model handle
- **param** – Damping function parameter handle
- **energy** – Pairwise additive dispersion energies
- **energy** – Pairwise non-additive dispersion energies

Evaluate the pairwise representation of the dispersion energy

1.2.3 Python API

Python API of the dftd4 program package

ASE Support

ASE calculator implementation for the dftd4 program.

This module provides a basic single point calculator implementations to integrate the dftd4 API into existing ASE workflows. To use DFTD4 as dispersion correction the ase.calculators.mixing module can be used to combine DFTD4 with a DFT calculator using the SumCalculator.

Supported properties by this calculator are:

- energy (free_energy)
- forces
- stress
Supported keywords are

<table>
<thead>
<tr>
<th>Keyword</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>method</td>
<td>None</td>
<td>Method to calculate dispersion for</td>
</tr>
<tr>
<td>params_tweaks</td>
<td>None</td>
<td>Optional dict with the damping parameters</td>
</tr>
<tr>
<td>cache_api</td>
<td>True</td>
<td>Reuse generate API objects (recommended)</td>
</tr>
</tbody>
</table>

The params_tweaks dict contains the damping parameters, at least s8, a1 and a2 must be provided

<table>
<thead>
<tr>
<th>Tweakable parameter</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>s6</td>
<td>1.0</td>
<td>Scaling of the dipole-dipole dispersion</td>
</tr>
<tr>
<td>s8</td>
<td>None</td>
<td>Scaling of the dipole-quadrupole dispersion</td>
</tr>
<tr>
<td>s9</td>
<td>1.0</td>
<td>Scaling of the three-body dispersion energy</td>
</tr>
<tr>
<td>a1</td>
<td>None</td>
<td>Scaling of the critical radii</td>
</tr>
<tr>
<td>a2</td>
<td>None</td>
<td>Offset of the critical radii</td>
</tr>
<tr>
<td>alp</td>
<td>16.0</td>
<td>Exponent of the zero damping (ATM only)</td>
</tr>
</tbody>
</table>

Either method or s8, a1 and a2 must be provided, s9 can be used to overwrite the ATM scaling if the method is provided in the model. Disabling the three-body dispersion (s9=0.0) changes the internal selection rules for damping parameters of a given method and prefers special two-body only damping parameters if available!

Example

```python
>>> from ase.build import molecule
>>> from dftd4.ase import DFTD4
>>> atoms = molecule('H2O')
>>> atoms.calc = DFTD4(method='TPSS')
>>> atoms.get_potential_energy()
-0.007310393443152083
>>> atoms.calc.set(method='PBE')
{'method': 'PBE'}
>>> atoms.get_potential_energy()
-0.00535847543239303
>>> atoms.get_forces()
array([[0., 0., 0.00296845],
       [0., 0.00119152, -0.00148423],
       [0., -0.00119152, -0.00148423]])
```

class dftd4.ase.DFTD4(*args, **kwargs)

ASE calculator for DFT-D4 related methods. The DFTD4 class can access all methods exposed by the dftd4 API.
Example

```python
>>> from ase.build import molecule
>>> from ase.calculators.mixing import SumCalculator
>>> from ase.calculators.nwchem import NWChem
>>> from dftd4.ase import DFTD4

>>> atoms = molecule('H2O')
>>> atoms.calc = SumCalculator([DFTD4(method="PBE"), NWChem(xc="PBE")])
```

```plaintext
add_calculator(other)
Convenience function to allow DFTD4 to combine itself with another calculator by returning a SumCalculator:
```

```python
>>> from ase.build import molecule
>>> from ase.calculators.emt import EMT
>>> from dftd4.ase import DFTD4

>>> atoms = molecule("C60")
>>> atoms.calc = DFTD4(method="pbe").add_calculator(EMT())
>>> atoms.get_potential_energy()
6.348142387048062
>>> [calc.get_potential_energy() for calc in atoms.calc.calcs]
[-6.015477436263984, 12.363619823312046]
```

calculate(atomic=None, properties=None, system_changes=ase.calculators.calculator.all_changes)
Perform actual calculation with by calling the dftd4 API

reset()
Clear all information from old calculation

set(**kwargs)
Set new parameters to dftd4

QCSchema Support
Integration with the QCArchive infrastructure.
This module provides a way to translate QCSchema or QCElemental Atomic Input into a format understandable by the dftd4 API which in turn provides the calculation results in a QCSchema compatible format.

Supported keywords are

<table>
<thead>
<tr>
<th>Keyword</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>level_hint</td>
<td>None</td>
<td>Dispersion correction level (allowed: “d4”)</td>
</tr>
<tr>
<td>params_tweaks</td>
<td>None</td>
<td>Optional dict with the damping parameters</td>
</tr>
<tr>
<td>pair_resolved</td>
<td>False</td>
<td>Enable pairwise resolved dispersion energy</td>
</tr>
<tr>
<td>property</td>
<td>False</td>
<td>Evaluate dispersion related properties</td>
</tr>
</tbody>
</table>

The params_tweaks dict contains the damping parameters, at least s8, a1 and a2 must be provided
Either method or s8, a1 and a2 must be provided, s9 can be used to overwrite the ATM scaling if the method is provided in the model. Disabling the three-body dispersion (s9=0.0) changes the internal selection rules for damping parameters of a given method and prefers special two-body only damping parameters if available!

Note: input_data.model.method with a full method name and input_data.keywords[“params_tweaks”] cannot be provided at the same time. It is an error to provide both options at the same time.

Example

```python
>>> from dftd4.qcschema import run_qcschema
>>> import qcelemental as qcel
>>> atomic_input = qcel.models.AtomicInput(...
...    molecule = qcel.models.Molecule(...
...        symbols = ["O", "H", "H"],
...        geometry = [
...            0.00000000000000, 0.00000000000000, -0.73578586109551,
...            1.44183152868459, 0.00000000000000, 0.36789293054775,
...            -1.44183152868459, 0.00000000000000, 0.36789293054775
...        ],
...    ),
...    driver = "energy",
...    model = {
...        "method": "TPSS-D4",
...    },
...    keywords = {},
...)
...)
>>> atomic_result = run_qcschema(atomic_input)
>>> atomic_result.return_result
-0.0002667885779142513
```

dftd4.qcschema.run_qcschema(input_data)

Perform dispersion correction based on an atomic input model
**PySCF Support**

Compatibility layer for supporting DFT-D4 in pyscf.

class dftd4.pyscf.DFTD4Dispersion(*args, **kwargs)

Implementation of the interface for using DFT-D4 in pyscf.

**Examples**

```python
>>> from pyscf import gto
>>> import dftd4.pyscf as disp
>>> mol = gto.M(  
...    atom='''
...    C -0.755422531 -0.796459123 -1.023590391  
...    C 0.634274834 -0.880017014 -1.075233285  
...    C 1.406955202 0.199695367 -0.653144334  
...    C 0.798863737 1.361204515 -0.180597909  
...    C -0.593166787 1.434312023 -0.133597923  
...    I -1.514344238 3.173268101 0.573601106  
...    H 1.110906949 -1.778801728 -1.440619836  
...    H 1.399172302 2.197767355 0.147412751  
...    H 2.486417780 0.142466525 -0.689380574  
...    H -2.454252250 0.422581120 -0.512807958  
...    H -1.362353593 -1.630564523 -1.348743149  
...    S -3.112683203 6.289227834 1.226984439  
...    H -4.328789697 5.797771251 0.973373089  
...    C -2.689135032 6.703163830 -0.489062886  
...    H -1.684433029 7.115457372 -0.460265708  
...    H -2.683867206 5.816530502 -1.115183775  
...    H -3.365330613 7.451201412 -0.890098894  
...  
...'''
...

>>> d4 = disp.DFTD4Dispersion(mol, xc="r2SCAN")
```  

**dump_flags** (verbose=None)

Show options used for the DFT-D4 dispersion correction.

**kernel()**

Compute the DFT-D4 dispersion correction.

The dispersion model as well as the parameters are created locally and not part of the state of the instance.

Returns

The energy and gradient of the DFT-D4 dispersion correction.

**Return type**

float, ndarray

**reset(mol)**

Reset mol and clean up relevant attributes for scanner mode
dftd4.pyscf.energy(mf)
Apply DFT-D4 corrections to SCF or MCSCF methods by returning an instance of a new class built from the original instances class.

Parameters
mf – The method to which DFT-D4 corrections will be applied.

Return type
The method with DFT-D4 corrections applied.

Examples

```python
>>> from pyscf import gto, scf
>>> import dftd4.pyscf as disp
>>> mol = gto.M(  
...     atom='''
N -1.57871857 -0.04661102  0.00000000
N  1.57871857  0.04661102  0.00000000
H -2.15862174  0.13639605  0.80956529
H -0.84947130  0.65819321  0.00000000
H -2.15862174 -0.13639605  0.80956529
H  2.15862174 -0.13639605 -0.80956529
H  0.84947130 -0.65819321  0.00000000
H  2.15862174 -0.13639605 -0.80956529
...''',  
...     )
>>> mf = disp.energy(scf.RHF(mol)).run()  
converged SCF energy = -110.917424528592  
>>> mf.kernel()  
-110.917424528592
```

dftd4.pyscf.grad(mfgrad)
Apply DFT-D4 corrections to SCF or MCSCF nuclear gradients methods by returning an instance of a new class built from the original class.

Parameters
mfgrad – The method to which DFT-D4 corrections will be applied.

Return type
The method with DFT-D4 corrections applied.

Examples

```python
>>> from pyscf import gto, scf
>>> import dftd4.pyscf as disp
>>> mol = gto.M(  
...     atom='''
O -1.65542061 -0.12330038  0.00000000
O  1.24621244  0.10268870  0.00000000
H -0.70409026  0.03193167  0.00000000
H -2.03867273  0.75372294  0.00000000
H  1.57598558 -0.38252146 -0.75856129
H  1.57598558 -0.38252146  0.75856129
...''',  
...     )
>>> mf = disp.energy(scf.RHF(mol)).run()  
converged SCF energy = -110.917424528592  
>>> mf.kernel()  
-110.917424528592
```
Library interface

Wrapper around the C-API of the dftd4 shared library. It provides the definition the basic interface to the library for most further integration in other Python frameworks.

The classes defined here allow a more Pythonic usage of the API object provided by the library in actual workflows than the low-level access provided in the CFFI generated wrappers.

Structure

class dftd4.interface.Structure(numbers, positions, charge=None, lattice=None, periodic=None)

Represents a wrapped structure object in dftd4. The molecular structure data object has a fixed number of atoms and immutable atomic identifiers

Example

```python
>>> from dftd4.interface import Structure
>>> import numpy as np
>>> mol = Structure(  
...     positions=np.array([  
...         [+0.00000000000000, +0.00000000000000, -0.73578586109551],  
...         [+1.44183152868459, +0.00000000000000, +0.36789293054775],  
...         [-1.44183152868459, +0.00000000000000, +0.36789293054775],  
...     ]),  
...     numbers = np.array([8, 1, 1]),
... )
>>> len(mol)
3
```

Raises

- ValueError – on invalid input, like incorrect shape / type of the passed arrays
update(positions, lattice=None)

Update coordinates and lattice parameters, both provided in atomic units (Bohr). The lattice update is optional also for periodic structures.

Generally, only the cartesian coordinates and the lattice parameters can be updated, every other modification, regarding total charge, total spin, boundary condition, atomic types or number of atoms requires the complete reconstruction of the object.

Raises

- ValueError – on invalid input, like incorrect shape / type of the passed arrays

DispersionModel

class dftd4.interface.DispersionModel(numbers, positions, charge=None, lattice=None, periodic=None, **kwargs)

Representation of a dispersion model to evaluate C6 coefficients. The model is coupled to the molecular structure it has been created from and cannot be transferred to another molecular structure without recreating it.

Example

```python
>>> from dftd4.interface import DispersionModel
>>> import numpy as np

>>> disp = DispersionModel(
...     positions=np.array([  # Coordinates in Bohr
...         [+0.00000000000000, +0.00000000000000, -0.73578586109551],
...         [+1.44183152868459, +0.00000000000000, +0.36789293054775],
...         [-1.44183152868459, +0.00000000000000, +0.36789293054775],
...     ]),
...     numbers = np.array([8, 1, 1]),
... )

>>> disp.get_properties()['polarizibilities']
array([6.74893641, 1.33914933, 1.33914933])
```

Raises

- ValueError – on incorrect inputs to for the structure data
- RuntimeError – in case of an error in library

get_dispersion(param, grad)

Perform actual evaluation of the dispersion correction.

Example

```python
>>> from dftd4.interface import DampingParam, DispersionModel
>>> import numpy as np

>>> numbers = np.array([1, 1, 6, 5, 1, 15, 8, 17, 13, 15, 1, 9, 15, 1, 15])

>>> positions = np.array([  # Coordinates in Bohr
...     [+2.79274810283778, +3.82998228828316, -2.79287054959216],
...     [-1.43447454186833, +0.43418729987882, +5.53854345129809],
...     [-3.26268343665218, -2.50644032426151, -1.56631149351046],
... ])
```

(continued from previous page)

... [+2.14548759959147, -0.88798018953965, -2.245923455506187],
... [-4.3023305401858, -0.58105573172764, +5.56854609916143],
... [+4.41363836635653, +0.92515781709283, +2.57961724984000],
... [+1.3370758998700, +1.40194471661647, +1.97530004949523],
... [+3.08342709834868, +1.7252024666801, +4.4266116106828],
... [-3.02346932078505, +0.04438199934191, -0.2763197425010],
... [+1.11508390868455, +2.1790347389232, +6.2127984216963],
... [+0.61938955433011, +2.03175899761859, +1.05038813614845],
... [-2.67491681346835, +0.9271412809198, +6.25462847718180],
... [+0.61938955433011, +2.1790347389232, +6.2127984216963],
... [+1.11508390868455, +2.1790347389232, +6.2127984216963],
... [+0.61938955433011, +2.1790347389232, +6.2127984216963],
... [+1.11508390868455, +2.1790347389232, +6.2127984216963],

... ])

>>> model = DispersionModel(numbers, positions)
>>> res = model.get_dispersion(DampingParam(method="scan"), grad=False)
>>> res.get("energy")  # Results in atomic units
-0.00532888853245093

**Raises**

**RuntimeError** – in case the calculation fails in the library

**get_pairwise_disperslon**

Evaluate pairwise representation of the dispersion energy

```python
>>> from dftd4.interface import DispersionModel, DampingParam
>>> import numpy as np

>>> disp = DispersionModel(
...    numbers=np.array([7, 7, 1, 1, 1, 1, 1, 1]),
...    positions=np.array([[-2.983345508575, -0.88082052767, +0.00000000000],
...                        [+2.983345508575, +0.88082052767, +0.00000000000],
...                        [+4.079203605652, +0.257751166821, +1.529856526214],
...                        [+1.605268001556, +1.23804812431, +0.00000000000],
...                        [+4.079203605652, +0.257751166821, +1.529856526214],
...                        [+1.605268001556, +1.23804812431, +0.00000000000],
...                        [+4.079203605652, -0.257751166821, +1.529856526214],
...                        [+1.605268001556, -1.23804812431, +0.00000000000],
...                        [+4.079203605652, -0.257751166821, +1.529856526214]]),
...    )

>>> res = disp.get_pairwise_disperslon(DampingParam(method="tpss"))
>>> res["additive pairwise energy"].sum()
-0.0023569384234103
>>> res["non-additive pairwise energy"].sum()
8.794562567135391e-08
```

**get_properties**

Evaluate dispersion related properties, like polarizibilities and C6 coefficients. Will also return the coordination numbers and partial charges used to derive the polarizibilities. Only the static polarizibility is return at the moment.

### 1.2. API documentation

21
Example

```python
>>> from dftd4.interface import DispersionModel
>>> import numpy as np

>>> disp = DispersionModel(
...     numbers=np.array([16, 16, 16, 16, 16, 16, 16, 16]),
...     positions=np.array([[-4.15128787379191, +1.71951973863958, -0.93066267097296],
...                         [-4.15128787379191, -1.71951973863958, +0.93066267097296],
...                         [-1.71951973863958, -4.15128787379191, -0.93066267097296],
...                         [+1.71951973863958, -4.15128787379191, +0.93066267097296],
...                         [+4.15128787379191, -1.71951973863958, -0.93066267097296],
...                         [+4.15128787379191, +1.71951973863958, +0.93066267097296],
...                         [+1.71951973863958, +4.15128787379191, -0.93066267097296],
...                         [-1.71951973863958, +4.15128787379191, +0.93066267097296]],
...     )

>>> res = disp.get_properties()

>>> res.get("coordination numbers")
array([1.96273847, 1.96273847, 1.96273847, 1.96273847, 1.96273847, 1.96273847, 1.96273847, 1.96273847])

>>> res.get("polarizibilities").sum()
158.748605606818
```

DampingParam

```python
class dftd4.interface.DampingParam(**kwargs)
```

Rational damping function for DFT-D4.

The damping parameters contained in the object are immutable. To change the parametrization, a new object must be created. Furthermore, the object is opaque to the user and the contained data cannot be accessed directly.

There are two main ways provided to generate a new damping parameter object:

1. a method name is passed to the constructor, the library will load the required data from the `dftd4` shared library.
2. all required parameters are passed to the constructor and the library will generate an object from the given parameters.

**Note:** Mixing of the two methods is not allowed to avoid partial initialization of any created objects. Users who need full control over the creation of the object should use the second method.

**Raises**
- `TypeError` – incorrect input values provided to constructor
- `RuntimeError` – failed to construct damping parameter object in API

```python
static load_param(method, atm=True)
```

Create damping function API object from internal library storage by searching for the provided method name. The method name is case insensitive and hyphens are ignored. In case the method name is unknown an exception is raised.
Example

```python
>>> from dftd4.interface import DampingParam
>>> param = DampingParam(method="pbe", atm=True)
```

Raises

`RuntimeError` – failed to construct damping parameter object in API

```python
static new_param(*, s6=1.0, s8, s9=1.0, a1, a2, alp=16.0)
```

Create damping function API object from user provided parameters. This object represent a rational damping function and requires at least the ‘s8’, ‘a1’, and ‘a2’ parameters as input. Additionally, the parameters ‘s6’, ‘s9’, and ‘alp’ can be overwritten. The user provided damping parameters will be used unchecked.

Example

```python
>>> from dftd4.interface import DampingParam
>>> param = DampingParam(s6=0.6400, s8=1.16888646, a1=0.44154604, a2=4.73114642)
```

Raises

`RuntimeError` – failed to construct damping parameter object in API
d

dftd4, 13
dftd4.ase, 13
dftd4.interface, 19
dftd4.pyscf, 16
dftd4.qcschema, 15
add_calculator() (dftd4.ase.DFTD4 method), 15

calculate() (dftd4.ase.DFTD4 method), 15

DampingParam (class in dftd4.ase), 13
DFTD4 (class in dftd4.ase), 14
dftd4 module, 13
dftd4.ase module, 13
dftd4.interface module, 19
dftd4.pyscf module, 16
dftd4.qcschema module, 15
dftd4_check_error (C function), 9
dftd4_custom_d4_model (C function), 11
dftd4_delete_error (C function), 9
dftd4_delete_model (C function), 11
dftd4_delete_param (C function), 12
dftd4_delete_structure (C function), 10
dftd4_error (C type), 9
dftd4_get_dispersion (C function), 12
dftd4_get_pairwise_dispersion (C function), 9
dftd4.get_properties (C function), 13

dftd4_load_rational_damping (C function), 12
dftd4_model (C type), 11
dftd4_new_d4_model (C function), 11
dftd4_new_error (C function), 9
dftd4_new_rational_damping (C function), 11
dftd4_new_structure (C function), 10
dftd4_param (C type), 11
dftd4_structure (C type), 10
dftd4.update_structure (C function), 10
DFTD4Dispersion (class in dftd4.pyscf), 17
DispersionModel (class in dftd4.interface), 20
dump_flags() (dftd4.pyscf.DFTD4Dispersion method), 17

energy() (in module dftd4.pyscf), 17

get_dispersion() (dftd4.interface.DispersionModel method), 20
get_pairwise Dispersion() (dftd4.interface.DispersionModel method), 21
get_properties() (dftd4.interface.DispersionModel method), 21
grow() (in module dftd4.pyscf), 18

load_param() (dftd4.interface.DampingParam static method), 22

module
dftd4, 13
dftd4.ase, 13
dftd4.interface, 19
dftd4.pyscf, 16
dftd4.qcschema, 15

new_param() (dftd4@interface.DampingParam static method), 23

reset() (dftd4.ase.DFTD4 method), 15
reset() (dftd4.pyscf.DFTD4Dispersion method), 17
run_qcschema() (in module dftd4.qcschema), 16

set() (dftd4.ase.DFTD4 method), 15
DFT-D4

Structure (class in dftd4.interface), 19

U

update() (dftd4.interface.Structure method), 19