Torch autodiff DFT-D3

Sebastian Ehlert, Marvin Friede

Jul 25, 2023
# CONTENTS

1 Installation 3
   1.1 PyPI .................................................. 3
   1.2 From Source ......................................... 3

2 Module reference 5
   2.1 DFT-D3 wrapper ...................................... 5
   2.2 Dispersion energy .................................... 6
   2.3 Damping schemes .................................... 8
   2.4 Rational (Becke-Johnson) damping function ........ 8
   2.5 Axilrod-Teller-Muto (ATM) dispersion term ........ 8
   2.6 Dispersion model .................................... 9
   2.7 Coordination number ................................ 10
   2.8 Reference model .................................... 12
   2.9 Atomic data .......................................... 13

Python Module Index 15

Index 17
Implementation of the DFT-D3 dispersion model in PyTorch. This module allows to process a single structure or a batch of structures for the calculation of atom-resolved dispersion energies.

Note: This project is still in early development and the API is subject to change. Contributions are welcome, please checkout our contributing guidelines.

Example

```python
>>> import torch
>>> import tad_dftd3 as d3

... numbers = d3.util.pack((
...     d3.util.to_number("C C N N H H H H O O".split()),
...     d3.util.to_number("C O N H H H".split()),
... ))

... positions = d3.util.pack((
...     torch.tensor([(-3.81469488143921, +0.09993441402912, 0.00000000000000),
...                      (+3.81469488143921, -0.09993441402912, 0.00000000000000),
...                      (-2.66030049324036, -1.5898251533508, 0.00000000000000),
...                      (+2.66030049324036, +1.5898251533508, 0.00000000000000),
...                      (-0.73178529739380, -2.28317795829773, 0.00000000000000),
...                      (+0.73178529739380, +2.28317795829773, 0.00000000000000),
...                      (-3.71254944801331, -3.71254944801331, 0.00000000000000),
...                      (+3.71254944801331, +3.71254944801331, 0.00000000000000),
...                      (-2.74426102638245, +2.16115570068359, 0.00000000000000),
...                      (+2.74426102638245, -2.16115570068359, 0.00000000000000),
...                      (-5.89039325714111, -0.02589114569128, 0.00000000000000),
...                      (+5.89039325714111, +0.02589114569128, 0.00000000000000),
...                      (-2.74426102638245, +2.16115570068359, 0.00000000000000),
...                      (+2.74426102638245, -2.16115570068359, 0.00000000000000),
...                      (-0.55569743203406, +1.09030425468557, 0.00000000000000),
...                      (+0.51473634678469, +3.1512550263611, 0.00000000000000),
...                      (-0.59869696244446, -1.16861263789477, 0.00000000000000),
...                      (+0.45355203669134, -2.74568780438064, 0.00000000000000),
...                      (+2.5271290544999, -1.29200800956867, 0.00000000000000),
...                      (+2.63139587595376, +0.96447869452240, 0.00000000000000),
...                      ],
...     torch.tensor([(-0.55569743203406, +1.09030425468557, 0.00000000000000),
...                      (+0.51473634678469, +3.1512550263611, 0.00000000000000),
...                      (-0.59869696244446, -1.16861263789477, 0.00000000000000),
...                      (+0.45355203669134, -2.74568780438064, 0.00000000000000),
...                      (+2.5271290544999, -1.29200800956867, 0.00000000000000),
...                      (+2.63139587595376, +0.96447869452240, 0.00000000000000),
...                      ]),
...     torch.tensor([0.5660, 0.3908, 3.1280, 0.5660, 0.3908, 3.1280]),
... ))

... param = dict(  
...     B97M-D3(BJ) parameters
...     a1=torch.tensor(0.5660),
...     s8=torch.tensor(0.3908),
...     a2=torch.tensor(3.1280),
... )

... energy = torch.sum(d3.dftd3(numbers, positions, param), -1)
... torch.set_printoptions(precision=7)
... print(energy)  
# Energies in Hartree

... tensor([-0.0124292, -0.0045002])
... print(energy[0] - 2*energy[1])

... tensor(-0.0034288)
```
CHAPTER ONE

INSTALLATION

1.1 PyPI

tad-dftd3 can easily be installed with pip.

\[ \text{pip install tad-dftd3} \]

1.2 From Source

This project is hosted on GitHub at dftd3/tad-dftd3. Obtain the source by cloning the repository with

\[ \text{git clone https://github.com/dftd3/tad-dftd3} \]
\[ \text{cd tad-dftd3} \]

We recommend using a conda environment to install the package. You can setup the environment manager using a mambaforge installer. Install the required dependencies from the conda-forge channel.

\[ \text{mamba env create -n torch -f environment.yml} \]
\[ \text{mamba activate torch} \]

For development, install the following additional dependencies

\[ \text{mamba install black coverage covdefaults mypy pre-commit pylint pytest tox} \]

Install this project with pip in the environment

\[ \text{pip install .} \]

Add the option \text{-e} for installing in development mode.

The following dependencies are required

- numpy
- torch
- pytest (tests only)

You can check your installation by running the test suite with pytest

\[ \text{pytest tests/ --pyargs tad_dftd3} \]

or tox for testing multiple Python versions
Torch autodiff DFT-D3

tox
The following modules are contained with *tad-dftd3*.

### 2.1 DFT-D3 wrapper

```python
tad_dftd3.dftd3(numbers, positions, param, *, ref=None, rcov=None, rvdw=None, r4r2=None, cutoff=None, counting_function=<function exp_count>, weighting_function=<function gaussian_weight>, damping_function=<function rational_damping>)
```

Evaluate DFT-D3 dispersion energy for a batch of geometries.

**Parameters**

- **numbers** (*torch.Tensor*) – Atomic numbers of the atoms in the system.
- **positions** (*torch.Tensor*) – Cartesian coordinates of the atoms in the system.
- **param** (*dict[str, Tensor]*) – DFT-D3 damping parameters.
- **rcov** (*torch.Tensor, optional*) – Covalent radii of the atoms in the system.
- **rvdw** (*torch.Tensor, optional*) – Van der Waals radii of the atoms in the system.
- **r4r2** (*torch.Tensor, optional*) – \( r_4 \) over \( r_2 \) expectation values of the atoms in the system.
- **damping_function** (*Callable, optional*) – Damping function evaluate distance dependent contributions.
- **weighting_function** (*Callable, optional*) – Function to calculate weight of individual reference systems.
- **counting_function** (*Callable, optional*) – Calculates counting value in range 0 to 1 for each atom pair.

**Returns**

Atom-resolved DFT-D3 dispersion energy for each geometry.

**Return type**

*Tensor*
2.2 Dispersion energy

This module provides the dispersion energy evaluation for the pairwise interactions.

Example

```python
>>> import torch
>>> import tad_dftd3 as d3

>>> numbers = torch.tensor([  
...     [8, 1, 1, 8, 1, 6, 1, 1, 1],
...     [0, 0, 0, 8, 1, 6, 1, 1, 1],
...     [8, 1, 1, 0, 0, 0, 0, 0, 0],
... ])

>>> positions = torch.tensor([  
...     [-4.224363834, +0.270465696, +0.527578960],
...     [-5.011768887, +1.780116228, +1.143194385],
...     [-2.468758653, +0.479766200, +0.982905589],
...     [+1.14617671, +0.452771215, +1.257722311],
...     [+1.841554378, -0.628298322, +2.538065200],
...     [+2.024899840, -0.34840095, -1.127412563],
...     [+1.210773578, +0.791908575, -2.550591723],
...     [+4.077073644, -0.342495506, -1.267841745],
...     [+1.404422261, -2.365753991, -1.503620411],
... ]).repeat(numbers.shape[0], 1, 1)

>>> ref = d3.reference.Reference()

>>> param = dict(  
...     a1=torch.tensor(0.49484001),
...     s8=torch.tensor(0.78981345),
...     a2=torch.tensor(5.73083694),
... )

>>> cn = d3.ncoord.coordination_number(numbers, positions)

>>> weights = d3.model.weight_references(numbers, cn, ref)

>>> c6 = d3.model.atomic_c6(numbers, weights, ref)

>>> energy = d3.disp.dispersion(numbers, positions, param, c6)

>>> torch.set_printoptions(precision=7)

>>> print(torch.sum(energy[0] - energy[1] - energy[2]))
# energy in Hartree
tensor(-0.0003964)
```

tad_dftd3.disp.dispersion(numbers, positions, param, c6, rvdw=None, r4r2=None, damping_function=<function rational_damping>, cutoff=None, **kwargs)

Calculate dispersion energy between pairs of atoms.

Parameters

- **numbers** (*Tensor*) – Atomic numbers of the atoms in the system.
- **positions** (*Tensor*) – Cartesian coordinates of the atoms in the system.
- **param** (*dict[str, Tensor]*) – DFT-D3 damping parameters.
- **c6** (*Tensor*) – Atomic C6 dispersion coefficients.
- **rvdw** (*Tensor*) – Van der Waals radii of the atoms in the system.
- **r4r2** (*Tensor*) – $r^4$ over $r^2$ expectation values of the atoms in the system.
• **damping function** (*Callable*) – Damping function evaluate distance dependent contributions. Additional arguments are passed through to the function.

**Returns**

Atom-resolved DFT-D3 dispersion energy for each geometry.

**Return type**

Tensor

tad_dftd3.disp.dispersion2(numbers, positions, param, c6, r4r2, damping_function, cutoff, **kwargs)

Calculate dispersion energy between pairs of atoms.

**Parameters**

- **numbers** (*Tensor*) – Atomic numbers of the atoms in the system.
- **positions** (*Tensor*) – Cartesian coordinates of the atoms in the system.
- **param** (*dict[str, Tensor]*) – DFT-D3 damping parameters.
- **c6** (*Tensor*) – Atomic C6 dispersion coefficients.
- **r4r2** (*Tensor*) – $r^4$ over $r^2$ expectation values of the atoms in the system.
- **damping_function** (*Callable*) – Damping function evaluate distance dependent contributions. Additional arguments are passed through to the function.

**Returns**

Atom-resolved three-body dispersion energy.

**Return type**

Tensor

tad_dftd3.disp.dispersion3(numbers, positions, param, c6, rvdw, cutoff, rs9=tensor(1.3333))

Three-body dispersion term. Currently this is only a wrapper for the Axilrod-Teller-Muto dispersion term.

**Parameters**

- **numbers** (*Tensor*) – Atomic numbers of the atoms in the system.
- **positions** (*Tensor*) – Cartesian coordinates of the atoms in the system.
- **param** (*dict[str, Tensor]*) – Dictionary of dispersion parameters. Default values are used for missing keys.
- **c6** (*Tensor*) – Atomic C6 dispersion coefficients.
- **rvdw** (*Tensor*) – Van der Waals radii of the atoms in the system.
- **cutoff** (*Tensor*) – Real-space cutoff.
- **rs9** (*Tensor, optional*) – Scaling for van-der-Waals radii in damping function. Defaults to $4.0/3.0$.

**Returns**

Atom-resolved three-body dispersion energy.

**Return type**

Tensor
2.3 Damping schemes

Available damping schemes for two- and three-body dispersion terms.

2.4 Rational (Becke-Johnson) damping function

This module defines the rational damping function, also known as Becke-Johnson damping.

\[ f_{\text{damp}}(R_{AB}^0) = \frac{R_{AB}^n}{R_{AB} + (a_1 R_{AB}^0 + a_2)^w} \]

`tad_dftd3.damping.rational.rational_damping(order, distances, qq, param)`

Rational damped dispersion interaction between pairs.

Parameters

- `order (int)` – Order of the dispersion interaction, e.g. 6 for dipole-dipole, 8 for dipole-quadrupole and so on.
- `distances (Tensor)` – Pairwise distances between atoms in the system.
- `qq (Tensor)` – Quotient of C8 and C6 dispersion coefficients.
- `param (dict[str, Tensor])` – DFT-D3 damping parameters.

Returns

Values of the damping function.

Return type

Tensor

2.5 Axilrod-Teller-Muto (ATM) dispersion term

This module provides the dispersion energy evaluation for the three-body Axilrod-Teller-Muto dispersion term.

\[ E_{\text{disp}}^{(3),\text{ATM}} = \sum_{ABC} E^{ABC} f_{\text{damp}}(R_{ABC}) \]

\[ E^{ABC} = C^{ABC}_6 \left( 3 \cos \theta_A \cos \theta_B \cos \theta_C + 1 \right) \]

\[ f_{\text{damp}} = \frac{1}{1 + 6 \left( \frac{1}{R_{ABC}} \right)^{-16}} \]

`tad_dftd3.damping.atm.dispersion_atm(numbers, positions, c6, rvdw, cutoff, s9=tensor(1.), rs9=tensor(1.3333), alp=tensor(14.))`

Axilrod-Teller-Muto dispersion term.

Parameters

- `numbers (Tensor)` – Atomic numbers of the atoms in the system.
- `positions (Tensor)` – Cartesian coordinates of the atoms in the system.
- `c6 (Tensor)` – Atomic C6 dispersion coefficients.
• **rvdw (Tensor)** – Van der Waals radii of the atoms in the system.
• **cutoff (Tensor)** – Real-space cutoff.
• **s9 (Tensor, optional)** – Scaling for dispersion coefficients. Defaults to 1.0.
• **rs9 (Tensor, optional)** – Scaling for van-der-Waals radii in damping function. Defaults to 4.0/3.0.
• **alp (Tensor, optional)** – Exponent of zero damping function. Defaults to 14.0.

**Returns**
Atom-resolved ATM dispersion energy.

**Return type**
Tensor

### 2.6 Dispersion model

Implementation of D3 model to obtain atomic C6 coefficients for a given geometry.

**Examples**

```python
>>> import torch
>>> import tad_dftd3 as d3
>>> numbers = d3.util.to_number(['O', 'H', 'H'])
>>> positions = torch.Tensor([...
...  [+0.00000000000000, +0.00000000000000, -0.73578586109551],
...  [+1.44183152868459, +0.00000000000000, +0.36789293054775],
...  [-1.44183152868459, +0.00000000000000, +0.36789293054775],
... ])
>>> ref = d3.reference.Reference()
>>> cn = d3.ncoord.coordination_number(numbers, positions, rcov, d3.ncoord.exp_count)
>>> weights = d3.model.weight_references(numbers, cn, ref, d3.model.gaussian_weight)
>>> c6 = d3.model.atomic_c6(numbers, weights, ref)
>>> torch.set_printoptions(precision=7)
>>> print(c6)
tensor([[10.4130478, 5.4368815, 5.4368815],
... [ 5.4368811, 3.0930152, 3.0930152],
... [ 5.4368811, 3.0930152, 3.0930152]])
```

tad_dftd3.model.atomic_c6(numbers, weights, reference)

Calculate atomic dispersion coefficients.

**Parameters**

• **numbers (Tensor)** – The atomic numbers of the atoms in the system.
• **weights (Tensor)** – Weights of all reference systems.
• **reference (Reference)** – Reference systems for D3 model.

**Returns**
Atomic dispersion coefficients.
Torch autodiff DFT-D3

Return type
Tensor

tad_dftd3.model.gaussian_weight(dcn, factor=4.0)
Calculate weight of individual reference system.

Parameters
- dcn (Tensor) – Difference of coordination numbers.
- factor (float) – Factor to calculate weight.

Returns
Weight of individual reference system.

Return type
Tensor

tad_dftd3.model.weight_references(numbers, cn, reference, weighting_function=gaussian_weight, **kwargs)
Calculate the weights of the reference system.

Parameters
- numbers (Tensor) – The atomic numbers of the atoms in the system.
- cn (Tensor) – Coordination numbers for all atoms in the system.
- reference (Reference) – Reference systems for D3 model.
- weighting_function (Callable) – Function to calculate weight of individual reference systems.

Returns
Weights of all reference systems

Return type
Tensor

2.7 Coordination number

Evaluates a fractional coordination number for a given geometry or batch of geometries.

Examples

```python
>>> import torch
>>> import tad_dftd3 as d3
>>> numbers = d3.util.pack((
...     torch.tensor([7, 1, 1, 1]),
...     torch.tensor([6, 8, 7, 1, 1]),
...     torch.tensor([6, 8, 1, 1]),
... ))
>>> positions = d3.util.pack((
...     torch.tensor([+0.00000000000000, +0.00000000000000, -0.545283797150],
...     torch.tensor([-0.88451840382282, +1.53203081565085, +0.18174945999050],
...     torch.tensor([-0.88451840382282, -1.53203081565085, +0.18174945999050],
```
torch.tensor([+1.76903680764564, +0.00000000000000, +0.18174945999050],
... }},
... torch.tensor([[-0.555697432303406, +1.09030425468557, +0.00000000000000],
... [-0.51473643678469, +3.15152550263611, +0.00000000000000],
... [-0.59869692444446, -1.16861263789477, +0.00000000000000],
... [-0.4535203669134, -2.74568780438064, +0.00000000000000],
... [-2.5272120544999, -1.29200800956867, +0.00000000000000],
... [+2.09259524590881, +1.87468519515944, +0.00000000000000],
... [-2.63139587595376, +0.96447869452240, +0.00000000000000],
... ]),
... )
>>>
rcov = d3.data.covalent_rad_d3[numbers]
>>>
cn = d3.ncoord.coordination_number(numbers, positions, rcov, d3.ncoord.exp_count)
>>>
torch.set_printoptions(precision=7)
>>>
print(cn)
tensor([[2.9901006, 0.9977214, 0.9977214, 0.9977214, 0.0000000, 0.0000000],
... [3.0059586, 1.0318390, 3.0268824, 1.0061584, 1.0036336, 0.9989871],
... [3.0093639, 2.0046251, 1.0187057, 0.9978270, 1.0069743, 0.0000000]])

---

tad_dftd3.ncoord.coordination_number(numbers, positions, rcov=None, counting_function=<function exp_count>, cutoff=None, **kwargs)

Calculate the coordination number of each atom in the system.

Parameters

- **numbers** (*Tensor*) – The atomic numbers of the atoms in the system.
- **positions** (*Tensor*) – The positions of the atoms in the system.
- **rcov** (*Tensor*) – Covalent radii for all atoms in the system.
- **counting_function** (*Callable*) – Calculates counting value in range 0 to 1 from a batch of distances and covalent radii, additional parameters can be passed through via key-value arguments.
- **cutoff** (*float*) – Real-space cutoff for the evaluation of counting function

Returns

*Tensor*

Return type

The coordination number of each atom in the system.

---

tad_dftd3.ncoord.exp_count(r, r0, kcn=16.0)

Exponential counting function for coordination number contributions.

Parameters

- **r** (*Tensor*) – Current distance.
- **r0** (*Tensor*) – Cutoff radius.

---

2.7. Coordination number
• **kcn** (*float*) – Steepness of the counting function.

**Returns**
Count of coordination number contribution.

**Return type**
Tensor

## 2.8 Reference model

This module defines the reference systems for the D3 model to compute the C6 dispersion coefficients.

```python
class tad_dftd3.reference.Reference(cn=None, c6=None, device=None, dtype=None)
```

Reference systems for the D3 dispersion model

**c6**
C6 coefficients for all pairs of reference systems

**cn**
Coordination numbers for all reference systems

**property device**
The device on which the Reference object resides.

**property dtype**
Floating point dtype used by reference object.

```python
to(device)
```

Returns a copy of the Reference instance on the specified device.

This method creates and returns a new copy of the Reference instance on the specified device “device”.

**Parameters**
device (torch.device) – Device to which all associated tensors should be moved.

**Returns**
A copy of the Reference instance placed on the specified device.

**Return type**
Reference

**Notes**

If the Reference instance is already on the desired device self will be returned.

```python
type(dtype)
```

Returns a copy of the Reference instance with specified floating point type. This method creates and returns a new copy of the Reference instance with the specified dtype.

**Parameters**

dtype (torch.dtype) – Type of the

**Returns**
A copy of the Reference instance with the specified dtype.

**Return type**
Reference
Notes

If the `Reference` instance has already the desired dtype `self` will be returned.

## 2.9 Atomic data

Data arrays for atomic constants like covalent radii or van-der-Waals radii.

```
tad_dftd3.data.covalent_rad_2009 = tensor([0.0000, 0.8063, 1.1590, 3.0236, 2.3685, 1.9401, 1.8897, 1.7889, 1.5874, 1.6126, ...
```


```
tad_dftd3.data.covalent_rad_d3 = tensor([0.0000, 0.8063, 1.1590, 3.0236, 2.3685, 1.9401, 1.8897, 1.7889, 1.5874, 1.6126, ...
```

D3 covalent radii used to construct the coordination number.

```
tad_dftd3.data.r4_over_r2 = tensor([0.0000, 0.80589, 4.7566, 3.8025, 3.1036, 26.1552, 17.2304, 17.2710, 12.7422, 9.5361, ...
```

PBE0/def2-QZVP atomic values calculated by S. Grimme in Gaussian (2010), rare gases recalculated by J. Mewes with PBE0/aug-cc-pVQZ in Dirac (2018). Also new super heavies Cn,Nh,Fl,Lv,Og and Am-Rg calculated at 4c-PBE/Dyall-AE4Z (Dirac 2022)

### 2.9. Atomic data
tad_dftd3, 1
  tad_dftd3.damping, 7
  tad_dftd3.damping.atm, 8
  tad_dftd3.damping.rational, 8
  tad_dftd3.data, 13
  tad_dftd3.disp, 5
  tad_dftd3.model, 9
  tad_dftd3.ncoord, 10
  tad_dftd3.reference, 12
  tad_dftd3.util, 13
Torch autodiff DFT-D3
INDEX

A
atomic_c6() (in module tad_dftd3.model), 9

C
c6 (tad_dftd3.reference.Reference attribute), 12
cn (tad_dftd3.reference.Reference attribute), 12
coordination_number() (in module tad_dftd3.ncoord), 11
covalent_rad_2009 (in module tad_dftd3.data), 13
covalent_rad_d3 (in module tad_dftd3.data), 13

D
device (tad_dftd3.reference.Reference property), 12
dftd3() (in module tad_dftd3), 5
dispersion() (in module tad_dftd3.disp), 6
dispersion2() (in module tad_dftd3.disp), 7
dispersion3() (in module tad_dftd3.disp), 7
dispersion_atm() (in module tad_dftd3.damping.atm), 8
dtype (tad_dftd3.reference.Reference property), 12

E
exp_count() (in module tad_dftd3.ncoord), 11

G
gaussian_weight() (in module tad_dftd3.model), 10

M
module
  tad_dftd3, 1
tad_dftd3.damping, 7
tad_dftd3.damping.atm, 8
tad_dftd3.damping.rational, 8
tad_dftd3.data, 13
tad_dftd3.disp, 5
tad_dftd3.model, 9
tad_dftd3.ncoord, 10
tad_dftd3.reference, 12
tad_dftd3.util, 13

R
rational_damping() (in module
tad_dftd3.damping.rational), 8
Reference (class in tad_dftd3.reference), 12

tad_dftd3
tad_dftd3.damping
tad_dftd3.damping.atm
tad_dftd3.damping.rational
tad_dftd3.data
tad_dftd3.disp
tad_dftd3.model
tad_dftd3.ncoord
tad_dftd3.reference
tad_dftd3.util
to() (tad_dftd3.reference.Reference method), 12
type() (tad_dftd3.reference.Reference method), 12

W
weight_references() (in module tad_dftd3.model), 10