aiida-tbmodels

Release 0.1.0

Jan 23, 2019

Contents:

1 Installation **3**

2 Examples **5**
- 2.1 Eigenvals Calculation . . . 5
- 2.2 Parse Calculation . . . 7
- 2.3 Slice Calculation . . . 8
- 2.4 Symmetrize Calculation . . . 9

3 Reference **13**
- 3.1 Calculation classes . . . 13
- 3.2 Parser class . . . 13

4 Indices and tables **15**

`aiida-tbmodels` is a plugin for the AiiDA framework to run calculations with the TBmodels code.

CHAPTER 1

Installation

You can install `aiida-tbmodels` with

```
python -m pip install aiida-tbmodels
```

where `python` is the interpreter for which you installed AiiDA.

To run TBmodels calculations, you will also have to set up the `tbmodels` CLI as an AiiDA code. After installing TBmodels, you can run `which tbmodels` to find the full path of the executable. Make sure to put `unset PYTHONPATH` into the prepend text of the code, since the `PYTHONPATH` set by AiiDA might interfere with your TBmodels installation.

CHAPTER 2

Examples

The following examples show how each of the TBmodels commands can be run with `aiida-tbmodels`. The source and input files of the examples can be found on GitHub. The TBmodels commands themselves are described in more detail in the TBmodels documentation

2.1 Eigenvals Calculation

This example runs the `tbmodels eigenvals` command, which calculated the eigenvalues of a tight-binding model for a given set of k-points.

```
#!/usr/bin/env runaiida
# -*- coding: utf-8 -*-

# © 2017-2019, ETH Zurich, Institut für Theoretische Physik
# Author: Dominik Gresch <greschd@gmx.ch>
"""
Runs a 'tbmodels eigenvals' calculation.
"""

from __future__ import division, print_function, unicode_literals

import os

from aiida.orm import DataFactory, Code
from aiida.orm.querybuilder import QueryBuilder
from aiida.orm.data.singlefile import SinglefileData
from aiida.work.launch import run_get_pid

from aiida_tbmodels.calculations.eigenvals import EigenvalsCalculation


def get_singlefile_instance(description, path):
    """
```

(continues on next page)

(continued from previous page)

```
    Retrieve an instance of SinglefileData with the given description, loading it␣
↪from ``path`` if it does not exist.
    """
    query_builder = QueryBuilder()
    query_builder.append(
        SinglefileData, filters={'description': {
            '==': description
        }}
    )
    res = query_builder.all()
    if len(res) == 0:
        # create archive
        res = SinglefileData()
        res.add_path(os.path.abspath(path))
        res.description = description
        res.store()
    elif len(res) > 1:
        raise ValueError(
            'Query returned more than one matching SinglefileData instance.'
        )
    else:
        res = res[0][0]
    return res


def run_eigenvals():
    """
    Creates and runs the eigenvals calculation.
    """
    builder = EigenvalsCalculation.get_builder()
    builder.code = Code.get_from_string('tbmodels')

    builder.tb_model = get_singlefile_instance(
        description=u'InSb TB model', path='./reference_input/model.hdf5'
    )

    # single-core on local machine
    builder.options = dict(
        resources=dict(num_machines=1, tot_num_mpiprocs=1), withmpi=False
    )

    builder.kpoints = DataFactory('array.kpoints')()
    builder.kpoints.set_kpoints_mesh([4, 4, 4], offset=[0, 0, 0])

    result, pid = run_get_pid(builder)
    print('\nRan calculation with PID', pid)
    print('Result:', result)


if __name__ == '__main__':
    run_eigenvals()
```

2.2 Parse Calculation

This example runs the `tbmodels parse` command, which creates a TBmodels HDF5 tight-binding model from a Wannier90 output folder.

```
#!/usr/bin/env runaiida
# -*- coding: utf-8 -*-

# © 2017-2019, ETH Zurich, Institut für Theoretische Physik
# Author: Dominik Gresch <greschd@gmx.ch>
"""
Runs a 'tbmodels parse' calculation.
"""

from __future__ import division, print_function, unicode_literals

import os

from aiida.orm import Code
from aiida.orm.querybuilder import QueryBuilder
from aiida.orm.data.folder import FolderData
from aiida.work.launch import run_get_pid

from aiida_tbmodels.calculations.parse import ParseCalculation


def get_input_folder():
    """
    Gets or creates the input folder containing the Wannier90 output.
    """
    folder_description = u'Bi Wannier90 output'
    query_builder = QueryBuilder()
    query_builder.append(
        FolderData, filters={'description': {
            '==': folder_description
        }}
    )
    res = query_builder.all()
    if len(res) == 0:
        # create archive
        res = FolderData()
        input_folder = './reference_input'
        for filename in os.listdir(input_folder):
            res.add_path(
                os.path.abspath(os.path.join(input_folder, filename)), filename
            )
        res.description = folder_description
        res.store()
    elif len(res) > 1:
        raise ValueError(
            'Query returned more than one matching FolderData instance.'
        )
    else:
        res = res[0][0]
    return res
```

(continues on next page)

(continued from previous page)

```
def run_parse():
    """
    Creates and runs the parse calculation.
    """
    builder = ParseCalculation.get_builder()
    builder.code = Code.get_from_string('tbmodels')

    # single-core on local machine
    builder.options = dict(
        resources=dict(num_machines=1, tot_num_mpiprocs=1),
        withmpi=False,
    )

    builder.wannier_folder = get_input_folder()

    result, pid = run_get_pid(builder)
    print('\nRan calculation with PID', pid)
    print('Result:', result)


if __name__ == '__main__':
    run_parse()
```

2.3 Slice Calculation

This example runs the `tbmodels slice` command, which can slice or re-order the orbitals in a tight-binding model.

```
#!/usr/bin/env runaiida
# -*- coding: utf-8 -*-

# © 2017-2019, ETH Zurich, Institut für Theoretische Physik
# Author: Dominik Gresch <greschd@gmx.ch>
"""
Runs a 'tbmodels slice' calculation.
"""

from __future__ import division, print_function, unicode_literals

import os

from aiida.orm import Code
from aiida.orm.data.base import List
from aiida.orm.data.singlefile import SinglefileData
from aiida.orm.querybuilder import QueryBuilder
from aiida.work.launch import run_get_pid

from aiida_tbmodels.calculations.slice import SliceCalculation


def get_singlefile_instance(description, path):
    """
    Retrieve an instance of SinglefileData with the given description, loading it␣
↪from ``path`` if it does not exist.
    """
```

(continues on next page)

(continued from previous page)

```
    query_builder = QueryBuilder()
    query_builder.append(
        SinglefileData, filters={'description': {
            '==': description
        }}
    )
    res = query_builder.all()
    if len(res) == 0:
        # create archive
        res = SinglefileData()
        res.add_path(os.path.abspath(path))
        res.description = description
        res.store()
    elif len(res) > 1:
        raise ValueError(
            'Query returned more than one matching SinglefileData instance.'
        )
    else:
        res = res[0][0]
    return res


def run_slice():
    """
    Creates and runs the slice calculation.
    """
    builder = SliceCalculation.get_builder()
    builder.code = Code.get_from_string('tbmodels')

    builder.tb_model = get_singlefile_instance(
        description=u'InSb TB model', path='./reference_input/model.hdf5'
    )

    # single-core on local machine
    builder.options = dict(
        resources=dict(num_machines=1, tot_num_mpiprocs=1), withmpi=False
    )

    builder.slice_idx = List(list=[0, 3, 2, 1])

    result, pid = run_get_pid(builder)
    print('\nRan calculation with PID', pid)
    print('Result:\n', result)


if __name__ == '__main__':
    run_slice()
```

2.4 Symmetrize Calculation

This example runs the `tbmodels symmetrize` command, which applies a group average to symmetrize a tight-binding model, given a symmetry-representation symmetries file.

```
#!/usr/bin/env runaiida
# -*- coding: utf-8 -*-

# © 2017-2019, ETH Zurich, Institut für Theoretische Physik
# Author: Dominik Gresch <greschd@gmx.ch>
"""
Runs a 'tbmodels symmetrize' calculation.
"""

from __future__ import division, print_function, unicode_literals

import os

from aiida.orm import Code
from aiida.orm.querybuilder import QueryBuilder
from aiida.orm.data.singlefile import SinglefileData
from aiida.work.launch import run_get_pid

from aiida_tbmodels.calculations.symmetrize import SymmetrizeCalculation


def get_singlefile_instance(description, path):
    """
    Retrieve an instance of SinglefileData with the given description, loading it␣
↪from ``path`` if it does not exist.
    """
    query_builder = QueryBuilder()
    query_builder.append(
        SinglefileData, filters={'description': {
            '==': description
        }}
    )
    res = query_builder.all()
    if len(res) == 0:
        # create archive
        res = SinglefileData()
        res.add_path(os.path.abspath(path))
        res.description = description
        res.store()
    elif len(res) > 1:
        raise ValueError(
            'Query returned more than one matching SinglefileData instance.'
        )
    else:
        res = res[0][0]
    return res


def run_symmetrize():
    """
    Creates and runs the symmetrize calculation.
    """
    builder = SymmetrizeCalculation.get_builder()

    builder.code = Code.get_from_string('tbmodels')

    # single-core on local machine
```

(continues on next page)

(continued from previous page)

```
    builder.options = dict(
        resources=dict(num_machines=1, tot_num_mpiprocs=1), withmpi=False
    )

    builder.tb_model = get_singlefile_instance(
        description=u'InAs unsymmetrized TB model',
        path='./reference_input/model_nosym.hdf5'
    )

    builder.symmetries = get_singlefile_instance(
        description=u'InAs symmetries',
        path='./reference_input/symmetries.hdf5'
    )

    result, pid = run_get_pid(builder)
    print('\nRan calculation with PID', pid)
    print('Result:', result)


if __name__ == '__main__':
    run_symmetrize()
```

CHAPTER 3

Reference

- *Calculation classes*
- *Parser class*

3.1 Calculation classes

class aiida_tbmodels.calculations.eigenvals.**EigenvalsCalculation**(***kwargs*)
Calculation class for the 'tbmodels eigenvals' command, which computes the eigenvalues from a given tight-binding model.

class aiida_tbmodels.calculations.parse.**ParseCalculation**(***kwargs*)
Calculation plugin for the 'tbmodels parse' command, which creates a TBmodels tight-binding model from the Wannier90 output.

class aiida_tbmodels.calculations.slice.**SliceCalculation**(***kwargs*)
Calculation plugin for the 'tbmodels slice' command, which re-orders or slices orbitals of a tight-binding model.

class aiida_tbmodels.calculations.symmetrize.**SymmetrizeCalculation**(***kwargs*)
Calculation class for the 'tbmodels symmetrize' command, which creates a symmetrized tight-binding model from a tight-binding model and symmetry representations.

3.2 Parser class

class aiida_tbmodels.parsers.model.**ModelParser**(*calc*)
Parse TBmodels output to a SinglefileData containing the model file.

Getting in touch

The development version of aiida-tbmodels is hosted on GitHub . Post an issue there or contact me directly with questions / suggestions / feedback.

CHAPTER 4

Indices and tables

- genindex
- modindex
- search

Index

E

EigenvalsCalculation (class in aiida_tbmodels.calculations.eigenvals), 13

M

ModelParser (class in aiida_tbmodels.parsers.model), 13

P

ParseCalculation (class in aiida_tbmodels.calculations.parse), 13

S

SliceCalculation (class in aiida_tbmodels.calculations.slice), 13
SymmetrizeCalculation (class in aiida_tbmodels.calculations.symmetrize), 13