ONETEP Tutorials

7.0.0

Tutorial 13

Tutorial 13: ASE ONETEP interface

Version: all versions Tom Demeyere

Preamble & Download links

Here is the tutorial for the ONETEP interface in ASE. It is a work in progress, but if you wish to give it a try, please find the two files:

- learn_ase_in_y_minutes.py
- onetep_interface.py

The first file is a general tutorial for people who are not used to ASE. The second file is a small file that explains how to use the ONETEP interface with ASE.

Below is the documentation for the ONETEP interface in ASE, as plan to be included in the ASE documentation.

Introduction

ONETEP is a linear-scaling density functional theory code which exploit the near-sightness of the electronic density. It uses a set of atom-centered local orbitals (denoted NGWFs) which are optimised in situ to enable calculations with a minimal number of orbitals.

This interface makes it possible to use ONETEP as a calculator in ASE. You need to have a copy of the ONETEP code (and an appropriate license) to use this interface.

Environment variables

The environment variable `ASE_ONETEP_COMMAND` must hold the command to invoke the ONETEP calculation. The variable must be a string with a link to the ONETEP binary, and any other specific settings required for your environment (srun, mpirun, ...)

You can setup this environment variable in your shell configuration file:

```
  $ export ASE_ONETEP_COMMAND="export OMP_NUM_THREADS=4; mpirun -n 6
~/onetep/bin/onetep.arch"
```

Or within python itself:

```
  >>> environ["ASE_ONETEP_COMMAND"]="export OMP_NUM_THREADS=4; mpirun -n 6
~/onetep/bin/onetep.arch"
```

ASE will automatically redirect stdout and stderr to the appropriate files, namely "$LABEL.out"and"$ LABEL.err" where label is the name used for your ONETEP calculations

Pseudopotentials

ONETEP accepts PAW datasets in the abinit format, and NCP pseudpotentials with formats USP and recpot. Support has recently been added for the upf format, for both PAW and NCPP potentials. Pseudopotentials are passed directly to the Onetep calculator as a dictionary definition. If no pseudopotentials are passed ASE will try to guess the files based on the element used and the pseudo_path variable.

```
    # Explicitly providing each path
    calc = Onetep(pseudopotentials = {'H': '/path/to/pseudos/H.usp', 'O':
'/path/to/pseudos/O.usp'})
    # Using pseudo_path
    calc = Onetep(pseudo_path = '/path/to/pseudos', pseudopotentials = {'H':
'H.usp', 'O': 'O.usp'})
    # ASE will try to guess them
    calc = Onetep(pseudo_path = '/path/to/pseudos')
```

For ASE to correctly guess the pseudopotentials, it is best to use a pseudo_path that contains only one pseudopotential file for each element.

ONETEP Calculator

Simple calculations can be setup calling the Onetep calculator without any parameters, in this case ONETEP's default parameters will be used. For more complex cases using the keywords parameters is necessary. The 'keywords' parameters is a dictionary, in which each of the keys is a string that should be a ONETEP keyword, and the corresponding value is what you want to set that keyword to in the input.

Examples

Here is an example python script which sets up a calculation on a water molecule:

```
    from ase.build import molecule
    from ase.calculators.onetep import Onetep
    from os import environ

    # water molecule from ASE database, centered in a ~ 24 Å box
    wat = molecule('H2O')
    wat.center(12)
    environ["ASE_ONETEP_COMMAND"]="export OMP_NUM_THREADS=8; mpirun -n 2
~/onetep/bin/onetep.arch"
    # Ouput will be in "water.out"
    calc = Onetep(label = 'water', xc = 'PBE', paw = True, pseudo_path =
'/path/to/pseudos')
    wat.calc = calc
    wat.get_potential_energy()
```

Here is a more complex example, this time setting up a Pt13 cluster and running a geometry optimisation on 64 cores:

```
    from os import environ

    import numpy as np

    from ase.build import molecule
    from ase.calculators.onetep import Onetep
    from ase.cluster import Octahedron
    from ase.optimize.sciopt import SciPyFminBFGS
    # Pt13 from ase.cluster
    nano = Octahedron('Pt', 3, 1)
    nano.set_cell(np.eye(3)*12)
    nano.center()

    label = 'pt13'

    environ["ASE_ONETEP_COMMAND"]="export OMP_NUM_THREADS=8; mpirun -n 8
~/onetep/bin/onetep.arch"

    # ONETEP default are atomic units, one can specify 'cutoff_energy' : '600
eV' if needed.
    keywords = {
        'xc' : 'rpbe',
        'do_properties' : True,
        'cutoff_energy' : 35,
        'output_detail': 'verbose',
        'elec_energy_tol': 1.0e-5/len(atoms),
    }

    # Ouput will be in "pt13.out",
    # append = True will not overwrite file at each step
    calc = Onetep(
        label = label,
        edft = True,
        append = True,
        pseudo_path = '/path/to/pseudos',
        keywords = keywords)

    nanoparticle.calc = calc

    opt = SciPyFminBFGS(atoms = nano, trajectory = label + ".traj", logfile =
label + ".log")
    opt.run(fmax=0.01)
```

Here is an example of setting up an EELS and LDOS calculation on an N-substituted graphene sheet, demonstrating several more advanced functionalities (eg tags, species groups, and overrides to pseudopotentials and atomic solver strings):

Quickly restart with solvation effect using the soft sphere model

```python
from ase.io import read
from ase.io.onetep import get_onetep_keywords

# Read from the previous run...
optimized_sheet = read("N_doped_graphene_001.out")

# Function to retrieve keywords dict from input file...
keywords = get_onetep_keywords('N_doped_graphene_001.dat')

# We add solvation keywords
keywords.update(
    {
    'is_implicit_solvent': True,
    'is_include_apolar': True,
    'is_smeared_ion_rep': True,
    'is_dielectric_model': 'fix_cavity',
    'is_dielectric_function' : 'soft_sphere'
    }
)

optimized_sheet.calc = Onetep(...)

...
```