mlippy - user manual

Konstantin Gubaev (e-mail: yamir4eg@gmail.com)

January 24, 2021

Introduction

This document describes the usage of the `mlippy` (MLIP PYthon) package.

Installation

- `mlippy` should work both for both Python2 and Python3 and requires the following packages: `Ase`, `Cython`, `numpy`, `gfortran`. For parallelization the `mpi4py` package is needed. It is highly recommended to use `Anaconda` as it contains all required packages.
- Run `make libmlip` from the 'mlip-2/' folder to make a library file needed for python version compilations.
- Run `make mlippy` from the 'mlip-2/' folder to make a file `mlippy.so` with `mlippy` python library.
- Copy `mlippy.so` file from 'mlip-2/lib/' folder to your PYTHONPATH directory (for example 'home/username/Python/libs'), or next to your executing py file. You can also add a path to the `mlippy.so` file in the py file like this:
```
import os
import sys
sys.path.insert(0, os.path.abspath(os.path.join(os.path.dirname(__file__),
'../../../lib')))
```
- The `mlippy` examples are located in `/mlip-dev/doc/examples_mlippy/` folder. For some examples binary analogues are available, so you can compare output files for binary and python versions of MLIP.

General tips

To start using mlippy, type:

```
import mlippy
```

and do initizaliation:

```
mlippy.initialize()
```

Note: `mlippy` can also work in parallel, together with `mpi4py`. The right way would be to pass an MPI communicator from python (created with mpi4py) like this:

```
import mpi4py
comm = mpi4py.MPI.COMM_WORLD
mlippy.initialize(comm)
```

1 Reading/writing configurations

The `mlippy` was primarily designed to be used together with Ase, therefore `mlippy` functions work with configurations in `Ase.Atoms` format inside python. But reading from files and writing to files is done in MLIP `cfg` format with implicit conversion during reading/writing. The related functions are listed Table 1. While reading and writing configurations, features are passed from `cfg` files to `Ase.Atoms` and from `Ase.Atoms` to `cfg` files. The features of `Ase.Atoms` objects are dictionaries of strings and they are accessible via the `Atoms.features` field.

Note: MLIP supports only relative (0,1,2...) numeration of atomic types. This corresponds to reading/writing `cfg` files, converting `cfg` to and from VASP, and using MTPR potentials for such configurations.

For MTP-based calculator for ASE the mapping of atomic types (Ase to MTP) is needed. More details can be found in the corresponding section of this manual

Function	Description
`ase_loadcfgs(filename, max_cfgs = None)`	Reads configurations from a file and returns them as an `Ase.Atoms` vector. The last argument limits the number of configurations to be read
`ase_savecfgs(filename, atoms,desc = None)`	Saves `Ase.Atoms` vector to a file with `cfg` configurations with a possibility to assign a feature "From", which can be specified with the last argument

Table 1: Functions for reading/writing/converting configurations.

2 Working with moment tensor potentials

Mlippy works with multicomponent moment tensor potentials (MTPs). They are implemented as a separate `mtp` class. To start working with an MTP create an instance of the `mtp` class:
`mlip = mlippy.mtp()`
and initialize it with the file, containing the MTP:
`mlip.load_potential('pot.mtp')`
For declaration and initialization within one line you can type:
`mlip = mlippy.mtp('pot.mtp')`

You can work with different `mtp` instances simultaneously. Next the functions employing MTPs will are listed:

- `ase_train(pot,train_cfg,options)`

  `pot` - the `mtp` object to train.
  `train_cfg` - `Ase.Atoms` vector to train on.
  `Options` are identical to the ones from the "train" command in the C++ version. In particular, they include:
  `max-iter` - number of BFGS iterations to perform.
  `conv-tol` - convergence tolerance for loss function decrease in 100 steps of BFGS.
  `Returns: void`

  The example of training and checking errors can be found in `/doc/examples_mlippy/train` folder.

- `ase_errors(pot, check_cfgs, on_screen=False)`

  `pot` - the `mtp` object for evaluation.
  `check_cfgs` - `Ase.Atoms` vector with configurations.
  `on_screen` - whether to print the errors on the screen or not.
  `Returns: <string,string> dictionary with elements of kind <'Energy per atom RMSE', '0.02'>`

  The example of processing errors can be found in `/doc/examples_mlippy/filter_stress` folder.

- `ase_select(pot,train_cfg,new_cfg,options)`

  `pot` - the `mtp` object to perform selection for.
  `train_cfg` - `Ase.Atoms` vector with training set.
  `new_cfg` - `Ase.Atoms` vector with new configurations to select from.
  `Returns: Ase.Atoms vector with new selected configurations` `Options` are identical to the ones from the "select˙add" command in the C++ version. Also

they include "select:..." settings from the "ini" files.

The example of selecting configurations can be found in `/doc/examples_mlippy/select` folder.

- `ase_relax(pot,atom_cfgs,options,relax_options)`

  `pot` - the `mtp` object to perform selection for.
  `atom_cfgs` - `Ase.Atoms` vector with training set.
  `Options` include the ones from the "relax" command from the C++ version, plus "select:..." settings from the ".ini" files. `Relax_options` include "relax:..." options from the ".ini" files.

  ```
  Returns: Ase.Atoms vector with aconfigurations,
  which were succesfully relaxed and failed to relax.
  Relaxed configurations have some Energy values.
  Failed to relax configurations have Energy=None.
  ```

  Note: For the active learning scenario, this function requires the `state.als` file in the running directory, which is used for active learning. Also, in the active learning scenario the file with extrapolative configurations, detected by active learning, can be created in the working directory - if the corresponding settings are set.

  The example of relaxing configurations can be found in `/doc/examples_mlippy/relax` folder.

3 MLIP Calculator for Ase

For MTP-based calculator for ASE the mapping of atomic types (Ase to MTP) is needed. This is due to the fact that MTPs work with relative atomic numbers (0,1,2,..) instead of actual ones from Ase.

Mapping is performed for an `mtp` object (representing MTP potential, see Section 2) via command:

```
mlip.add_atomic_type(26)
mlip.add_atomic_type(27)
```

Mapping for current MTP can be accessed by command:

```
mlp.get_types_mapping()
```

which returns 1-D array of Ase atomic numbers. The order of atomic numbers corresponds to 0,1,2.. species in MTPs.

In Ase the energies, forces and stresses in atomistic configurations are calculated via special class: Calculator. In `mlippy` the special type of Calculator, `MLIP_Calculator` is implemented. It uses MTPs for calculation of energies, forces and stresses. `MLIP_Calculator` requires an `mtp` object for initialization:

```
calc = mlippy.MLIP_Calculator(mlip,options)
```

```
a = Ase.Atoms(...)
a.set_calculator(calc)
```

The `mlip` potential in above example should be trained and should have coefficients for the species present in the `a` atomistic system, otherwise calculation will result in an error.

The calculations using active learning (with tracking of extrapolation grades, writing extapolative configurations and breaking when exceeding extrapolation treshold) are also possible with `MLIP_Calculator`. This is controlled by passing the `select:TRUE` option and requires an als-file.

Example of using MLIP˙Calculator is located in `/doc/examples_python/MLIP_Calculator` folder.