EZFF Documentation

Release 0.9 Beta

Aravind Krishnamoorthy

Aug 19, 2023

Contents:

EZFF is a Python-based library for quick and easy parameterization of forcefields and interatomic potentials for molecular dynamics simulations. EZFF provides interfaces to popular atomistic simulation software, GULP, LAMMPS, VASP, RXMD, and QChem and uses Platypus for solving multi-objective optimization problems. Use the links below to get started.

CHAPTER 1

Basic Usage

You will need five pieces of information to get started with focefield optimization using EZFF.

1. Ground truths - Physical properties to parameterize the forcefield against
2. Serial MD executable (either GULP or LAMMPS or RXMD)
3. Forcefield template
4. Maximum and minimum values of decision variables
5. A master python script (*run.py* in examples) that defines different errors, handles GULP jobs and optimization parameters
6. (**Optional**) A working installation of MPI and mpi4py for parallel optimization

Ground truths Ground truth values (for lattice constant, elastic constants, energies, phonon dispersion curves etc) can either be provided by hand in the master script, or can be calculated using methods provided in the different ezff.interfaces modules

Serial MD executable A working serial MD engine. Options available are:

1. Serial GULP executable built from source code available from http://gulp.curtin.edu.au/gulp/
2. Serial LAMMPS executable built from https://www.lammps.org/#gsc.tab=0

3. Serial RXMD executable built from https://magics.usc.edu/rxmd/ Parallel optimization jobs simply launch multiple copies of the serial executable.

Forcefield template A forcefield template file is used to designate which variables must be considered for optimization. The forcefield template is constructed from a functioning GULP-readable forcefield by replacing the parameters that need to be optimized with variable names enclosed within dual angle-brackets. For example, the following Lennard-Jones forcefield for solid Neon (from examples lj-serial and lj-parallel)

```
lennard epsilon 12 6  # Tell GULP that the next line will contain LJ␣
↪parameters
Ne Ne 1.0 1.5  # Format: atom1 atom2 epsilon sigma
```

can be converted to a template by replacing epsilon and sigma by optimizable decision variables:

```
lennard epsilon 12 6  # Tell GULP that the next line will contain LJ␣
↪parameters
Ne Ne <<eps>> <<sgma>>  # Format: atom1 atom2 epsilon sigma
```

This fill will instruct EZFF to optimize the **eps** and **sgma** variables. Decision variables are assumed to be real-valued by default. Any decision variable beginning with an underscore (_) is assumed to be integer-valued.

The template forcefield must be paired with an appropriate file specifying the permissible ranges of these decision variables.

Decision variable ranges This file lists the permissible ranges of decision variables (i.e. minimum and maximum value that the variable can take) during forcefield optimization. This text file is written in the following format:

```
Decision_variable_1(without the angle brackets)    Minimum_value ␣
↪Maximum_value
Decision_variable_2(without the angle brackets)    Minimum_value ␣
↪Maximum_value
Decision_variable_3(without the angle brackets)    Minimum_value ␣
↪Maximum_value
.
.
Decision_variable_n(without the angle brackets)    Minimum_value ␣
↪Maximum_value
```

A valid variable range file for the Lennard-Jones template file above is given below:

```
eps 0.5 2.5
sgma 0.1 0.4
```

Warning: Please ensure that the template and variable_ranges file are compatible. Specifically,

1. Every variable defined in the forcefield template must have one (and only one) corresonding entry in the variable ranges file
2. The variable ranges file should not refer to variables not present in the template file

Python script This python script should include, at the very least, your custom function to calculate the error (i.e. deviation of the forcefield from ground-truths), an ezff.FFParam object, a call to ezff.set_algorithm and a call to ezff.parameterize.

The custom error function should be written to accept one input – a dictionary of decision_variable-value pairs (e.g. {'eps': 1.273, 'sgma': 0.12} for example above) and should return a list of all computed objectives. The length of this returned list should equal the number of errors.

CHAPTER 2

Algorithms

EZFF comes with several algorithms for gradient-free single- and multi-objective parameterization of forcefields. Algorithms are provided from one of four optimization frameworks - Nevergrad, Platypus, PyMOO, and MOBOpt. The following algorithms are available in EZFF v1.0.

Table 1: Available EZFF Algorithms

Algorithm name	Algorithm type	Framework	Number of Objecives
ngopt_so	Adaptible meta-optimizer	Nevergrad	Single
twopointsde_so	Differential Evolution with 2-points crossover	Nevergrad	Single
portfoliodiscreteoneplusone_so	Genetic Algorithm for mixed discrete/continuous search spaces	Nevergrad	Single
oneplusone_so	One Plus One	Nevergrad	Single
twopointsde_so	Differential Evolution with 2-points crossover	Nevergrad	Single
cma_so	Covariance Matrix Adaptation Evolution Strategy	Nevergrad	Single
tbpsa_so	Test-based population size adaptation	Nevergrad	Single
pso_so	Particle Swarm Optimization	Nevergrad	Single
scrhammersleysearchplusmiddlepoint_so	Scrambled Hammersley plus middle point single-shot optimization	Nevergrad	Single
randomsearch_so	Random sampling	Nevergrad	Single
nsga2_mo_pymoo	Nondominated Sorting Genetic Algorithm II	pymoo	Multiple
nsga3_mo_pymoo	Nondominated Sorting Genetic Algorithm III	pymoo	Multiple
unsga3_mo_pymoo	Unified Nondominated Sorting Genetic Algorithm III with tournament pressure	pymoo	Multiple
ctaea_mo_pymoo	Constrained Two-Archive Evolutionary Algorithm	pymoo	Multiple
smsemoa_mo_pymoo	S-Metric Selection Evolutionary Multiobjective Optimization Algorithm	pymoo	Multiple
rvea_mo_pymoo	Reference Vector Guided Evolutionary Algorithm	pymoo	Multiple
es_so_pymoo	Evolutionary Strategy	pymoo	Single
neldermead_so_pymoo	Nelder Mead	pymoo	Single
cmaes_so_pymoo	Covariance Matrix Adaptation Evolution Strategy	pymoo	Single
nsga2_mo_platypus	Nondominated Sorting Genetic Algorithm II	Platypus	Multiple
nsga3_mo_platypus	Nondominated Sorting Genetic Algorithm III	Platypus	Multiple
gde3_mo_platypus	Generalized Differential Evolution 3	Platypus	Multiple
mobo	Multi-objective Bayesian Optimization	MOBOpt	Multiple

CHAPTER 3

Examples

The following forcefield optimization examples showcase the features of EZFF

3.1 lj-gulp-serial

Optimization of a Lennard-Jones forcefield for FCC Neon against 2 objectives – **bulk modulus** and **lattice constant**

Features demonstrated in this example

1. Basic use of forcefield templates and variable_range files
2. Reading-in elastic modulus tensor from GULP run
3. Reading-in lattice constants after a GULP run

3.2 lj-lammps-serial

Optimization of a Lennard-Jones forcefield for FCC Neon against 2 objectives – **bulk modulus** and **lattice constant**

Features demonstrated in this example

1. Reading-in elastic modulus tensor from LAMMPS run
2. Reading-in lattice constants after a LAMMPS run

3.3 sw-gulp-serial

Optimization of a Stillinger-Weber forcefield for the 2H-MoSe2 monolayer system against 3 objectives – **Lattice constant** (a), **Elastic modulus** (C_{11}) and **Phonon dispersion**

Features demonstrated in this example

1. Reading-in phonon dispersion from GULP and VASP data files

2. Calculating error between phonon dispersions
3. Calculating error between computed and ground-truth phonon dispersions
4. Reading-in elastic modulus tensor from GULP run
5. Usage of the Multi-objective Bayesian Optimizer

3.4 sw-gulp-multialgo

Parallel optimization of a Stillinger-Weber forcefield for the 1T' monolayer system against 6 objectives – **Two lattice constants** (a and b), **One elastic modulus** (C_{11}) and **Three phonon dispersion curves** (one each for compressed, relaxed and expanded crystals) using a sequence of multiple multi-objective genetic algorithms. Here, the population from the last epoch of optimization with a single algorithm is used as the initial population for the next algorithm in the sequence.

Features demonstrated in this example

1. Using multiple genetic algorithms in sequence for a single problem
2. Use of different population sizes and epochs for each algorithm in the sequence

3.5 sw-gulp-parallel-multi

Parallel optimization of a Stillinger-Weber forcefield for the 1T' monolayer system against 6 objectives – **Two lattice constants** (a and b), **One elastic modulus** (C_{11}) and **Three phonon dispersion curves** (one each for compressed, relaxed and expanded crystals)

Features demonstrated in this example

1. Spawning and using Multiprocessing pools for optimization
2. Non-uniform weighting schemes for calculating phonon dispersion errors

3.6 sw-gulp-parallel-mpi

Parallel optimization of a Stillinger-Weber forcefield for the 1T' monolayer system against 6 objectives – **Two lattice constants** (a and b), **One elastic modulus** (C_{11}) and **Three phonon dispersion curves** (one each for compressed, relaxed and expanded crystals)

Features demonstrated in this example

1. Spawning and using MPI pools for optimization

3.7 vashishta-lammps-serial

Optimization of a Stillinger-Weber forcefield for SiC crystal against 2 objectives – **Lattice constant** (a) and **Elastic modulus** (C_{11})

Features demonstrated in this example

1. Reading-in elastic modulus tensor from LAMMPS run
2. Optimization of Vashishta potential

3.8 reaxff-charge-gulp-serial

Optimization of charge parameters in the ReaxFF forcefield for a thio-ketone monomer against 1 objective – **atomic charges**

Features demonstrated in this example

1. Use of make_template_qeq
2. Use of ezff.error_atomic_charges
3. Use of nevergrad single-objective optimizers

3.9 reaxff-distortion-gulp-serial

Optimization of charge and bond parameters in the ReaxFF forcefield for a thio-ketone monomer against 2 objective – **atomic charges** and **structural distortion**

Features demonstrated in this example

1. Use of ezff.error_structure_distortion

3.10 reaxff-lammps-parallel-multi

Parallel optimization of ReaxFF forcefield for a thio-ketone monomer against 2 objectives – **Dissociation energy** of the C-S bond and C-S **vibrational frequency**

Features demonstrated in this example

1. Using QChem interface to read-in QM energies
2. Using LAMMPS interface to perform single-point calculations and read-in energy
3. Using utils.reaxff methods for generating forcefields templates and variable range files
4. Heterogeneous weighting scheme for calculating errors from potential energy surface scans

3.11 lj-gulp-save-restart

Serial optimization of Lennard Jones forcefield for solid Neon against 2 objectives – **Lattice constant** (a) and **Elastic modulus** (C_{11})

Features demonstrated in this example

1. Save evaluated variables as numpy arrays
2. Continue optimization after loading pre-evaluated variables

3.12 pedone-lammps-parallel-multi

Serial optimization of the Pedone forcefield (hybrid mixture of Coulombic + Morse + Repulsive interactions) for amorphous SiO2 against 2 objectives – **Lattice constant** (a) and **Elastic modulus** (C_{11})

Features demonstrated in this example

1. Parameterization of hybrid forcefields (containing 2 or more forcefield types) in LAMMPS

CHAPTER 4

Code Documentation

The following links provide a detailed description of individual classes and methods in the EZFF package.

4.1 EZFF - Easy forcefield fitting

4.2 ezff.ffio - Methods to read/write forcefield files

This module provide methods to handle reading and writing forcefields

ffio.**generate_forcefield**(*template_string*, *parameters*, *FFtype=None*, *outfile=None*, *MD='GULP'*)

Generate a new forcefield from the template by replacing variables with numerical values

Parameters

- **template_string** (*str*) – Text of the forcefield template
- **parameters** (*dict*) – Numerical value of all decision variables in the form of variable:value pairs
- **FFtype** (*string*) – Type of forcefield being optimized. (e.g. reaxff, sw, lj, etc.)
- **outfile** (*string*) – Optional filename to write out the forcefield
- **MD** (*string*) – MD Engine used for parameterization

ffio.**read_forcefield_template**(*template_filename*)

Read-in the forcefield template. The template is constructed from a functional forcefield file by replacing all optimizable numerical values with variable names enclosed within dual angled brackets << and >>.

Parameters **template_filename** (*str*) – Name of the forcefield template file to be read-in

ffio.**read_variable_bounds**(*filename*, *verbose=False*)

Read permissible lower and upper bounds for decision variables used in forcefields optimization

Parameters

- **filename** (*str*) – Name of text file listing bounds for each decision variable that must be optimized
- **verbose** (*bool*) – Print all variables read-in

4.3 ezff.errors - Error calculation modules

4.4 ezff.interfaces.vasp - Interface to VASP

4.5 ezff.interfaces.qchem - Interface to QChem

4.6 ezff.interfaces.gulp - Interface to GULP

4.7 ezff.utilities.reaxff - Utility to generate forcefield templates for ReaxFF

CHAPTER 5

Installing

Install from PyPI using the command

```
pip install EZFF
```

Alternatively, you can install the latest developmental version from GitHub via

```
git clone https://github.com/arvk/EZFF.git
cd EZFF
python setup.py install
```

CHAPTER 6

Contributing

1. Please make sure to submit only passing builds
2. Adhere to PEP8 where you can
3. Submit a pull request

CHAPTER 7

License

EZFF source code and documentation is released under the MIT License

CHAPTER 8

Indices and tables

- genindex
- modindex
- search

Python Module Index

f

Index

F

G

R