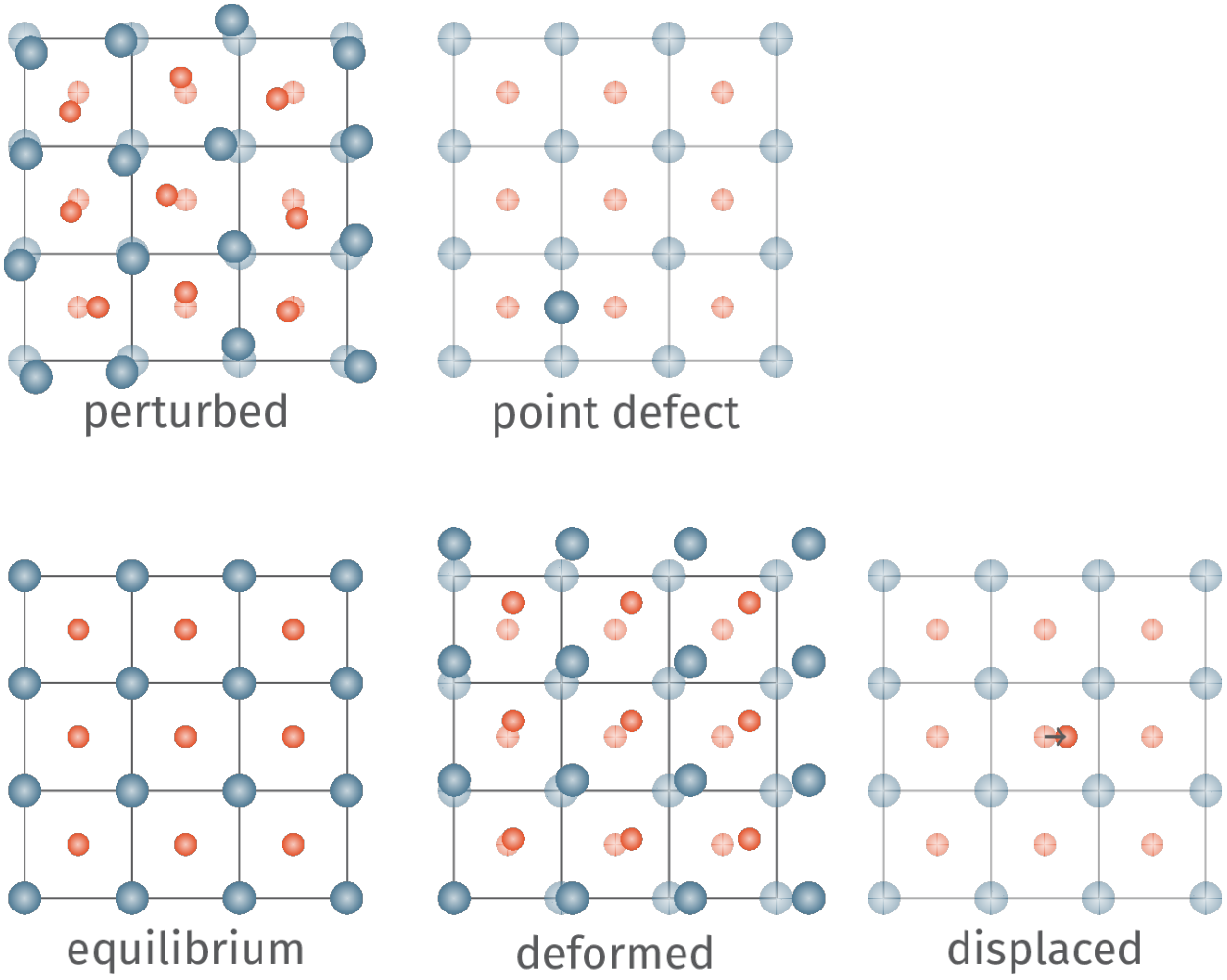
dftfit Documentation

Release 0.3.2

Chris Ostrouchov

Sep 20, 2018

Contents:



DFTFIT is a python code that used Ab Initio data from DFT calculations such as VASP, Quantum Espresso, and Siesta to develop molecular dynamic potentials. Our package differs from other similar codes in that we leverage LAMMPS as a calculator enabling a wide variety of potentials. The potentials include custom python functions and a wide variety or three-body interactions including the Tersoff, Stillinger-Weber, Gao-Weber, Vashishta, and COMB Potentials. All of which can be combined to have for example a Buckingham + Coulomb + ZBL potential. We also have an extensive set of multi-objective and single-objective optimizers that can evaluate a potential for many properties including energy, forces, stress, lattice constants, elastic constants, bulk modulus, and shear modulus.

In general three things are required from the user.

- Ab-Initio Training Data includes VASP, Siesta, and Quantum Espresso Calculations. Additionally the user may supply measured properties such as lattice constants, elastic constants, bulk modulus, and shear modulus.
- configuration specifies optimization algorithm and number of steps, sqlite database to store results, MD calculator to use, weights to give for each property.
- Potential among a rich set of two and three body potentials. Including a custom python function.

CHAPTER 1

Installation

For `pypi` installation. Note that installation of *lammps-cython* may fail and is required. You will need to install `LAMMPS` as documented here.

```
pip install dftfit
```

For *conda* installation

```
conda install -c costrouc -c matsci -c conda-forge dftfit
```

For *docker* installation

```
docker pull costrouc/dftfit
docker run -it costrouc/dftfit /bin/bash
```

CHAPTER 2

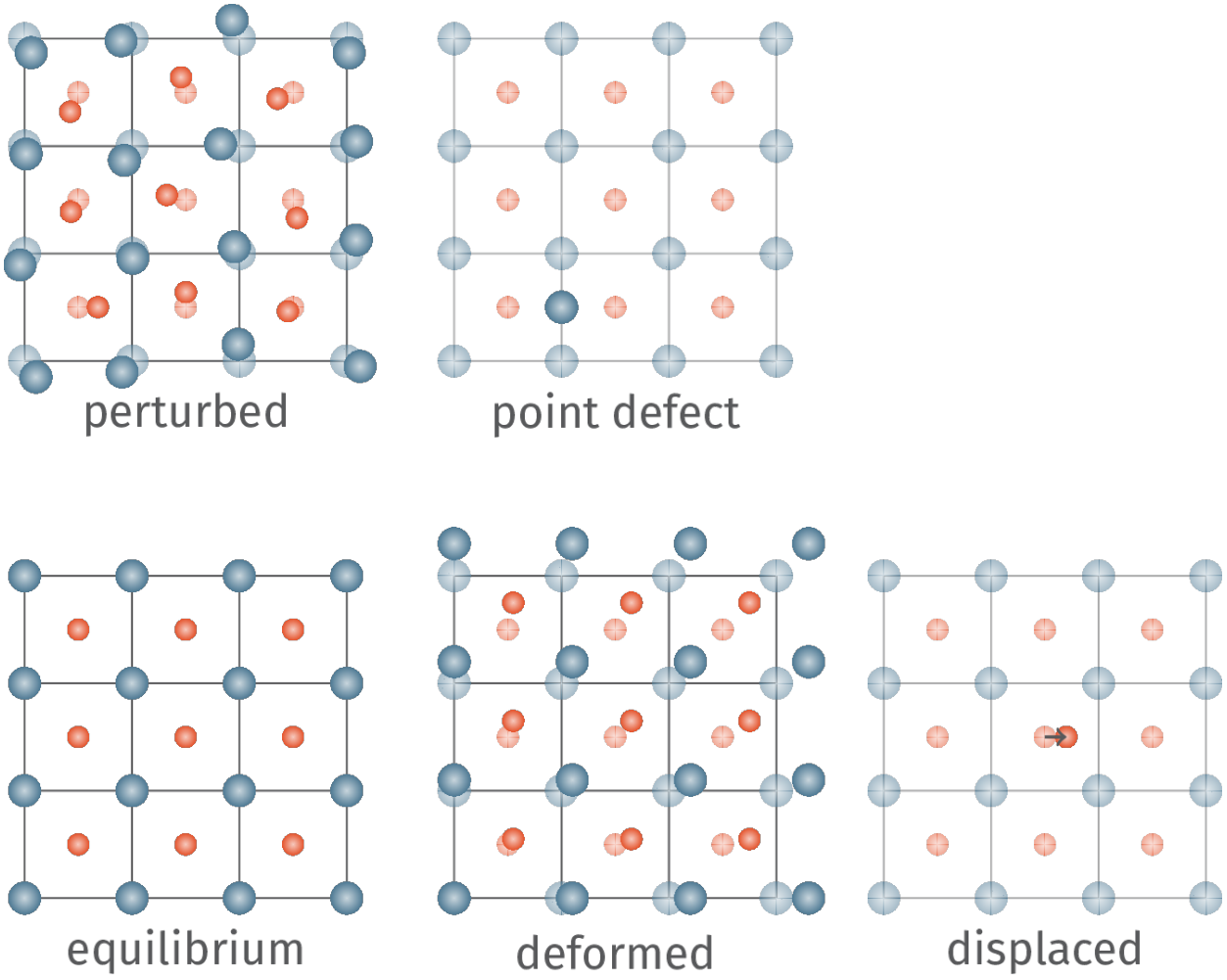
Tutorial

2.1 Simple MgO Example

DFTFIT is a software that is used to produce interatomic potentials for molecular dynamics simulations. The goal of the code is to enable scientists to use more accurate simulation methods to guide the construction of these potentials. For this example we will be using VASP calculations.

The dft training data consists of:

- equilibrium structure
- displaced structures $0.01x212B - 0.04x212B$
- deformed structures $\pm\%2$ normal, $\pm\%8$ shear
- random perturbed structures $0.04x212B$



For this example all the training structures are stored in a cached database. DFTFIT uses a cache so that it does not have to reparse all input vasp and quantum espresso input files. From this cache we get 131 training structures.

- 1 equilibrium structure
- 6 x 10 deformed structures
- 2 x 5 displaced structures
- 60 perturbed structures

Now that we have all of our input files we now need to configure our dftfit potential fitting. We are studing MgO which most commonly uses a buckingham potential with each atom having a charge.

$$U_{Mg-Mg}(A_1, \rho_1, C_1) \quad U_{Mg-O}(A_2, \rho_2, C_2) \quad U_{O-O}(A_3, \rho_3, C_3)$$
$$q_{Mg} = -q_O$$

This leads to 10 free parameters.

$$A_1, \rho_1, C_1, A_2, \rho_2, C_2, A_3, \rho_3, C_3, q_{Mg}$$

DFTFIT configuration is intentionally separated into 3 parts: potential, training structures, dftfit configuration. Let us look at an example potential file. In this case we will be looking at the lewis-catlow potential for MgO.

```
version: v1
kind: Potential
spec:
  constraint:
    charge_balance: MgO
  charge:
    Mg: {'initial': 2.0, 'bounds': [1.3, 2]}
    O: {'initial': -2.0, 'bounds': [-2, -1.3]}
  kspace:
    type: pppm
    tollerance: 1e-5
  pair:
    type: buckingham
    cutoff: 10.0
    parameters:
      - elements: ['Mg', 'Mg']
        coefficients:
          - {'initial': 0.1, 'bounds': [1e-6, 1000]}
          - {'initial': 0.1, 'bounds': [1e-6, 1e2]}
          - 0.0
      - elements: ['Mg', 'O']
        coefficients:
          - {'initial': 821.61, 'bounds': [10, 1e4]}
          - {'initial': 0.324199, 'bounds': [1e-6, 1e2]}
          - 0.0
      - elements: ['O', 'O']
        coefficients:
          - {'initial': 22764.915, 'bounds': [1e3, 1e6]}
          - {'initial': 0.14899, 'bounds': [1e-3, 1e2]}
          - {'initial': 20.3705, 'bounds': [1, 1e3]}
```

Currently DFTFIT supports any float value in the input file to be a parameter for the optimization. This even means that the buckingham cutoff radius can be an optimization parameter. Additionally, DFTFIT has been designed such that it is possible to determine if two potential parameter files are using the same potential form (I am quite proud of this feature). It is not perfect but will work for %90 of all cases. The form for a parameter is `{'initial': <value>, 'bound': [<lower>,<upper>]}`. There is planned support in the future for spline function parameters. This would be useful for example in constructing EAM potentials.

Next the training structures need to be defined for input. The current input file uses calculations generated with my own framework `mattoolkit` however vasp and quantum espresso are also supported. The actual input file in the examples is much longer. We can see that it is a list of calculations to include. In the case of mattoolkit all calculations are chosen using a selector. All calculations that match the labels `project:potential_fitting`, `structrue:MgO`, `calculation_type:static`, `calculation_group:lattice_constant`. If you haven't done it already, when you accumulate a lot of data it is important to assign metadata to all of your calculations. This makes the calculations much easier to select.

```
version: v1
kind: Training
spec:
  - type: mattoolkit
    selector:
      labels:
        - project:potential_fitting
        - structrue:MgO
        - calculation_type:static
        - calculation_group:lattice_constant
```

Finally we have the DFTFIT configuration. The configuration is quite flexible.

```
version: v1
kind: Configuration
metadata:
  name: testing
  labels:
    algorithm: "pygmo.sade"
    test: test
spec:
  logging: INFO
  database:
    interval: 10
    filename: "/tmp/dftfit/database.db"
  algorithm:
    name: 'pygmo.sade'
    steps: 10
    population: 10
  problem:
    calculator: 'lammps_cython'
    num_workers: 3
    weights:
      force: 0.3
      stress: 0.6
      energy: 0.1
  training:
    cache_filename: "./cache/cache.db"
```

The `metadata` section is for providing information about the calculation. The `name` property needs to be a string. While the `labels` are key value pairs that must both be strings. The key value pairs can be anything. Now we have the parameters that actually affect the calculation.

logging *default is WARNING* it is used to print information during the run. Stick with WARNING for a much much cleaner stdout

database if specified provide the location for the sqlite3 database

steps *required* number of optimization steps

population *required* number of potential parameters sets to solve at each iteration

algorithm *required* the optimization algorithm to use a good one to start using is SADE. See pagmo2 algorithmn documentation.

problem.calculator molecular dynamics calculator to use lammps is the only one for now

problem.command command to run to start calculator *make sure command matches calculator!*

problem.num_workers determines the parallelism. Currently DFTFIT doesn't scale well past 8 processors. Each processor does about 150 calculations/second.

weights features to calculate and the associated weights. Can be `None` for value.

training.cache_filename where to store the caches parsed training calculations

If is a global optimization algorithm is chosen random population points will be chosen. After the configuration file has been setup you are ready to go! The optimization can be simply run using the `dftfit` command installed when installing the python package. Run the command within the `examples/mgo` folder.

```
dftfit train -c configuration.yaml -p potential.yaml -t training.yaml
```

Since the example configuration only run 10 * 10 = 100 optimization steps the potential really will not improve. For my calculations I do 100,000 optimization steps with each step taking less than a fraction of a second. In total 100,000 steps takes about 10-12 hours.

DFTFIT comes with tools for investigating the results from the optimization.

TODO add more.

CHAPTER 3

Potentials

Read a potential from a python `dict`

Read a potential from `json` or `yaml` file.

```
from dftfit.potential import Potential
potential = Potential.from_file(filename)
```

DFTFIT can define very complex potentials. Unlike similar potential fitting software, DFTFIT allows any combination of potentials defined bellow. This allows a user for example to mix a ZBL and buckingham with a coulombic interaction seen here. Even though it may not make sense a user can mix a Tersoff, ZBL, Stillinger Weber, and python custom pair potential. The performance impact of mixing several potentials is almost negligible for small systems of less than 1000 atoms.

DFTFIT uses a `json` schema to represent any potential. To make DFTFIT optimize any float value in the potential replace the float value for with somthing similar to `{"initial": 1.0, "bounds ": [2.0, 3.0]}`. This tells DFTFIT that the initial guess should be `1.0` and to restrict the optimization values between `2.0` and `3.0`. An example is shown here for MgO.

Note that the yaml schema is not the only way to provide a potential. `json` can be used to represent any DFTFIT yaml specification. Additionally they can be represented with normal python datastructures `dict` and `list`.

Bellow is a list of all the supported potentials. Soon EAM and arbitrary splines for potentials will be supported see issue 17

3.1 Two Body Potentials

3.1.1 Python Functions

DFTFIT allows for arbitrary python functions to be used for pair potentials. The only requirement is that you define a function named `potential` with the last arguments being the *r* that will be supplied by dftfit. All the other parameters to the function will be optimized. See this stack-overflow question if you need clarification on what

numpy function vectorization is. Under the covers DFTFIT evaluates the function at a set number of points (`np.linspace(cutoff[0], cutoff[1], samples)`) to calculate the energies and forces (via the finite centered difference).

An example of the buckingham potential is below. You are free to import and call any functions within the block.

```
import numpy as np

def potential(A, p, C, r):
    return A * np.exp(-r/p) - C / (r**6)
```

yaml schema

```
pair:
 - type: python-function
   cutoff: [1.0, 10.0]
   samples: 1000
   function: |
     import numpy as np

     def potential(A, p, C, r):
         return A * np.exp(-r/p) - C / (r**6)
   parameters:
     - elements: ['Mg', 'Mg']
       coefficients: [1309362.2766468062, 0.104, 0.0]
     - elements: ['Mg', 'O']
       coefficients: [9892.357, 0.20199, 0.0]
     - elements: ['O', 'O']
       coefficients: [2145.7345, 0.3, 30.2222]
```

3.1.2 Coloumbic Interaction Potential

- lammps documentation
- example potential
- n parameters: where n is number of elements - 1

The coloumbic interaction potential has more knobs that other pair potentials to allow for how the long range interactions are integrated. Additionally a `constraint` allows for ensure that the total charge of the system is balanced. Complex forumla can be used e.g. `LiTaO3`.

$$E = \frac{Cq_iq_j}{\epsilon r}$$

yaml schema

```
spec:
  constraint:
    charge_balance: MgO
  charge:
    Mg: 1.4
    O: -1.4
  kspace:
    type: pppm
    tollerance: 1e-5
```

3.1.3 ZBL Potential

- lammps documentation
- example potential
- 2 parameters: Z_1, Z_2

$$E_{ij} = \frac{Z_i Z_j e^2}{4\pi\epsilon_0 r_{ij}} \phi(r_{ij}/a)$$

$$a = \frac{0.46850}{Z_i^{0.23} + Z_j^{0.23}}$$

$$\phi x = 0.18175e^{-3.19980x} + 0.50986e^{-0.94229x} + 0.28022e^{-0.40290x} + 0.02817e^{-0.20162x}$$

yaml schema

```
pair:
 - type: zbl
   cutoff: [3.0, 4.0]
   parameters:
     - elements: ['Mg', 'Mg']
       coefficients: [12, 12]
     - elements: ['Mg', 'O']
       coefficients: [12, 8]
     - elements: ['O', 'O']
       coefficients: [8, 8]
```

3.1.4 Lennard Jones Potential

- lammps documentation
- example potential
- parameters 2: ϵ, σ

$$E = 4\epsilon \left[\left(\frac{\sigma}{r}\right)^{12} - \left(\frac{\sigma}{r}\right)^{6} \right]$$

yaml schema

```
pair:
 - type: lennard-jones
   cutoff: [10.0]
   parameters:
     - elements: ['Ne', 'Ne']
       coefficients: [33.921, 2.801]
```

3.1.5 Beck Potential

- lammps documentation
- example potential
- 5 parameters: A, B, a, α, β

$$E(r) = A\exp\left[-\alpha r - \beta r^6\right] - \frac{B}{(r^2+a^2)^3}\left(1 + \frac{2.709 + 3a^2}{r^2+a^2}\right)$$

yaml schema

```
pair:
 - type: beck
   cutoff: [8.0]
   parameters:
     - elements: ['He', 'He']
       coefficients: [399.671876712, 0.0000867636112694, 0.675, 4.390, 0.0003746]
```

3.1.6 Buckingham Potential

- lammps documentation
- example potential
- 3 parameters: A, ρ, C

$$\psi(r) = A\exp^{-\frac{r}{\rho}} - \frac{C}{r^6}$$

yaml schema

```
pair:
 - type: buckingham
   cutoff: [10.0]
   parameters:
     - elements: ['Mg', 'Mg']
       coefficients: [1309362.2766468062, 0.104, 0.0]
     - elements: ['Mg', 'O']
       coefficients: [9892.357, 0.20199, 0.0]
     - elements: ['O', 'O']
       coefficients: [2145.7345, 0.3, 30.2222]
```

3.2 Three Body Potentials

Three Body potentials tend to have **many** more parameters. Because of this there are often mixing rules that help to reduce the number of parameters. They define some rules such that given interaction element_{i, i} ∇_i and element_{j,j} ∇_i the potential for intercation element_{ij} can be calculated via $f(_i, \nabla_j)$.

Currently defined mixes:

- tersoff-2

3.2.1 Tersoff Potential

- lammmps documentation
- example potential
- 14 parameters: $m, \gamma, \lambda_3, c, d, \cos(\theta_0), n, beta, \lambda_2, B, R, D, \lambda_1, A$

yaml schema

```
pair:
 - type: tersoff
   parameters:
     - elements: ['C', 'C', 'C']
       coefficients: [3.0, 1.0, 0.0, 38049, 4.3484, -0.57058, 0.72751, 0.00000015724,␣
↪2.2119, 346.7, 1.95, 0.15, 3.4879, 1393.6]
     - elements: ['Si', 'Si', 'Si']
       coefficients: [3.0, 1.0, 0.0, 100390, 16.217, -0.59825 ,0.78734, 0.0000011, 1.
↪73222, 471.18, 2.85, 0.15, 2.4799, 1830.8]
     - elements: ['Si', 'Si', 'C']
       coefficients: [3.0, 1.0, 0.0, 100390, 16.217, -0.59825, 0.0, 0.0, 0.0, 0.0, 2.
↪36, 0.15, 0.0, 0.0]
     - elements: ['Si', 'C', 'C']
       coefficients: [3.0, 1.0, 0.0, 100390, 16.217, -0.59825, 0.787340, 0.0000011, 1.
↪97205, 395.126, 2.36, 0.15, 2.9839, 1597.3111]
     - elements: ['C', 'Si', 'Si']
       coefficients: [3.0, 1.0, 0.0, 38049, 4.3484, -0.57058, 0.72751, 0.00000015724,␣
↪1.97205, 395.126, 2.36, 0.15, 2.9839, 1597.3111]
     - elements: ['C', 'Si', 'C']
       coefficients: [3.0, 1.0, 0.0, 38049, 4.3484, -0.57058, 0.0, 0.0, 0.0, 0.0, 1.
↪95, 0.15, 0.0, 0.0]
     - elements: ['C', 'C', 'Si']
       coefficients: [3.0, 1.0, 0.0, 38049, 4.3484, -0.57058, 0.0, 0.0, 0.0, 0.0, 2.
↪36, 0.15, 0.0, 0.0]
     - elements: ['Si', 'C', 'Si']
       coefficients: [3.0, 1.0, 0.0, 100390, 16.217, -0.59825, 0.0, 0.0, 0.0, 0.0, 2.
↪85, 0.15, 0.0, 0.0]
```

Equations

$$E = \frac{1}{2}\sum_i \sum_{j \neq i} V_{ij}$$

$$V_{ij} = f_c\left(r_{ij}\right)\left[f_R(r_{ij}) + b_{ij} f_A(r_{ij})\right]$$

$$f_c(r_{ij}) = \begin{cases} 1 & r_{ij} < R_{ij} - D_{ij} \\ \frac{1}{2} - \frac{1}{2}\sin\left[\frac{\pi}{2}(r_{ij} - R_{ij})/D_{ij}\right] & R_{ij} - S_{ij} < r_{ij} < R_{ij} + D_{ij} \\ 0 & r_{ij} > R_{ij} + D_{ij} \end{cases}$$

$$f_R(r) = A_{ij}\exp(-\lambda_{1,ij} r)$$

$$f_A(r) = -B_{ij}\exp(-\lambda_{2,ij} r)$$

$$b_{ij} = (1 + \beta_i^{n_i}\zeta_{ij}^{n_i})^{-\frac{1}{2n_i}}$$

$$\zeta_{ij} = \sum_{k \neq i,j} f_c(r_{ik}) g(\theta_{ijk}) \exp[\lambda_{3,ij}^m (r_{ij} - r_{ik})^m]$$

$$g(\theta_{ijk}) = \gamma_{ik}\left(1 + \frac{c_i^2}{d_i^2} - \frac{c_i^2}{[d_i^2 + (\cos\theta_{0,i} - \cos\theta_{ijk})^2]}\right)$$

Variables: $R_{ij}, D_{ij}, A_{ij}, \lambda_{1,ij}, B_{ij}, \lambda_{2,ij}, \beta_i, n_i, \gamma_{ik}, c_i, d_i, m_i, \lambda_{3,ij}, \theta_{0,i}$

Two body terms (6): $n_i, \beta_i, \lambda_{2,ij}, B_{ij}, \lambda_{1,ij}, A_{ij}$

Three body terms (6): $m_i, \gamma_{ik}, \lambda_{3,ij}, c_i, d_i, \theta_{0,i}$

Terms that only depend on primary atom (6): $n_i, \beta_i, m_i, c_i, d_i, \theta_{0,i}$

Usually Fixed Terms $‘m, gamma, beta‘$

Mixing Terms λ, A, B, R, D

__m must be 3 or 1__

Original tersoff [1] form achieved when $m = 3$ and $\gamma = 1$

Tersoff [2] has the the following contstraints:

$\lambda_{3,i} = 0$ thus m has not effect. In original paper $\gamma_{ik} = 1$.

Additional assumptions are the following: $\lambda_3 = 0$, `m = 3`, and `gamma = 1` thus these parameters are not included.

The order of the parameters are $c, d, \cos(\theta_0), n, \beta, \lambda_2, B, R, D, \lambda_1, A$. Additional models may be added if necessary.

$$\lambda_{ij} = \frac{1}{2}(\lambda_i + \lambda_j)$$

$$A_{ij} = \sqrt{A_i A_j}$$

$$B_{ij} = \chi_{ij}\sqrt{B_i B_j}$$

A mixing parameter is required for elements (N -1) see paper

$$R_{ij} = \sqrt{R_i R_j}$$

$$D_{ij} = \sqrt{D_i D_j}$$

Albe [3] when $\beta = 1$ and $m = 1$.

From [4] an R is 1.95, 2.85 for C-C-C and Si-Si-Si respectively and 0.15 for D (units Angstroms). R and D are chosen so as to include the first neighbor shell only.

1. Tersoff Original Paper J. Tersoff, Phys Rev B, 37, 6991 (1988).
2. Albe Form
3. Tersoff 2
4. Lammps Implementation

3.2.2 Stillinger Weber Potential

- lammps documentation
- example potential
- 11 parameters: $\epsilon, \sigma, a, \lambda, \gamma, \cos(\theta_0), A, B, p, q, tol$

yaml schema

```
pair:
  - type: stillinger-weber
    parameters:
      - elements: ["Cd", "Cd", "Cd"]
        coefficients: [1.03, 2.51, 1.80, 25.0, 1.20, -0.333333333333, 5.1726, 0.8807,␣
↪4.0, 0.0, 0.0]
      - elements: ["Te", "Te", "Te"]
        coefficients: [1.03, 2.51, 1.80, 25.0, 1.20, -0.333333333333, 8.1415, 0.6671,␣
↪4.0, 0.0, 0.0]
      - elements: ["Cd", "Cd", "Te"]
        coefficients: [1.03, 0.0 , 0.0, 25.0, 0.0, -0.333333333333, 0.0, 0.0, 0.0, 0.
↪0, 0.0]
```

(continues on next page)

(continued from previous page)

```
      - elements: ["Cd", "Te", "Te"]
        coefficients: [1.03, 2.51, 1.80, 25.0, 1.20, -0.333333333333, 7.0496, 0.6022,␣
↪4.0, 0.0, 0.0]
      - elements: ["Te", "Cd", "Cd"]
        coefficients: [1.03, 2.51, 1.80, 25.0, 1.20, -0.333333333333, 7.0496, 0.6022,␣
↪4.0, 0.0, 0.0]
      - elements: ["Te", "Cd", "Te"]
        coefficients: [1.03, 0.0, 0.0, 25.0, 0.0, -0.333333333333, 0.0, 0.0, 0.0, 0.0,
↪ 0.0]
      - elements: ["Te", "Te", "Cd"]
        coefficients: [1.03, 0.0, 0.0, 25.0, 0.0, -0.333333333333, 0.0, 0.0, 0.0, 0.0,
↪ 0.0]
      - elements: ["Cd", "Te", "Cd"]
        coefficients: [1.03, 0.0, 0.0, 25.0, 0.0, -0.333333333333, 0.0, 0.0, 0.0, 0.0,
↪ 0.0]
```

Equations

$$E = \sum_i \sum_{j>i} \phi_2(r_{ij}) + \sum_i \sum_{j\neq i} \sum_{k>j} \phi_3(r_{ij}, r_{ik}, \theta_{ijk})$$

$$\phi_2(r_{ij}) = A_{ij}\epsilon_{ij}\left[B_{ij}\left(\frac{\sigma_{ij}}{r_{ij}}\right)^{p_{ij}} - \left(\frac{\sigma_{ij}}{r_{ij}}\right)^{q_{ij}}\right] \exp\left(\frac{\sigma_{ij}}{r_{ij} - a_{ij}\sigma_{ij}}\right)$$

$$\phi_3(r_{ij}, r_{ik}, \theta_{ijk}) = \lambda_{ijk}\epsilon_{ijk}\left[cos\theta_{ijk} - cos\theta_{0ijk}\right]^2 exp\left(\frac{\gamma_{ij}\sigma_{ij}}{r_{ij} - a_{ij}\sigma_{ij}}\right) exp\left(\frac{\gamma_{ij}\sigma_{ik}}{r_{ik} - a_{ik}\sigma_{ik}}\right)$$

Parameters: $\epsilon, \sigma, a, \lambda, \gamma, \cos(\theta_0), A, B, p, q, tol$

Mixing terms: σ, ϵ

Mixing Rules

Analysis of the mixing rules for the Stillinger–Weber potential: a case-study of Ge–Si interactions in the liquid phase

https://doi.org/10.1016/j.jnoncrysol.2006.07.017

With such systems, however, there arises a problem of choosing suitable parameters for unlike-species interactions, i.e. devising $\sigma_{ij}, \epsilon_{ij} from \sigma_i, sigma_j, \epsilon_i, \epsilon_j$ (for the two-body term) and $\epsilon_{ijk}, \lambda_{ijk}$ from $\epsilon_i, \epsilon_j, \lambda_i, and \lambda_j$, where i, j, and k label the species of atoms in bond pairs and triplets. The two-body parameters were usually approximated using the geometric mean for the energy parameter and the arithmetic mean for the length parameter (the so-called Lorentz–Berthelot mixing rules). This had no rigoristic justification in first principles, but was analogous to what was usually done for other potentials. - page 4233

Choosing mixed-species paramters $\epsilon_{ijk}, \lambda_{ijk}$ for the three-body part is less obvious. Usually the choice of $\epsilon_{ijk} = \sqrt{\epsilon_{ij}\epsilon_{ik}} = \epsilon_j^{\frac{1}{4}}\epsilon_j^{\frac{1}{2}}\epsilon_j^{\frac{1}{4}}$ and $\lambda_{ijk} = \sqrt{\lambda_{ij}\lambda_{ik}} = \lambda_j^{\frac{1}{4}}\lambda_j^{\frac{1}{2}}\lambda_j^{\frac{1}{4}}$, first made by Grabow and Gilmer in [1] was iterated, even though the original authors had not justified it in any way.

In our study we decided to further test this traditional choice against other ways of constructing the parameters, eg. $\lambda_{SiSiGe} = \sqrt[3]{\lambda_{Si}\lambda_{Si}\lambda_{Ge}}$.

Since the resultant parameters differed by only a few percent, we expected to obtain similar results, regardless of the type of the mixing rule employed, which would then confirm the validity of the Grabow–Gilmer mixing as one of several that work. Surprisingly, this was not the case. It turned out that the simulations performed with only slightly different parameters resulted in radically different final atomic configurations.

1. M.H. Grabow, G.H. Gilmer, Surf. Sci. 194 (1987) 333

3.2.3 Gao Weber Potential

- [lammps documentation](https://lammps.sandia.gov/doc/pair_gw.html)
- [example potential](https://gitlab.com/costrouc/dftfit/blob/master/test_files/potential/SiC-gao-weber.yaml)
- 14 parameters: $‘m, gamma, lambda_3, c, d, h, n, beta ,lambda_2, B, R, D, lambda_1, A‘$

yaml schema

```
pair:
  - type: gao-weber
    parameters:
      - elements: ['Si', 'Si', 'Si']
        coefficients: [1, 0.013318, 0, 14, 2.1, -1, 0.78000, 1, 1.80821400248640, 632.
↪658058300867, 2.35, 0.15, 2.38684248328205, 1708.79738703139]
      - elements: ['Si', 'Si', 'C']
        coefficients: [1, 0.013318, 0, 14, 2.1, -1, 0.78000, 1, 1.80821400248640, 632.
↪658058300867, 2.35, 0.15, 2.38684248328205, 1708.79738703139]
      - elements: ['Si', 'C', 'Si']
        coefficients: [1, 0.013318, 0, 14, 2.1, -1, 0.78000, 1, 1.96859970919818, 428.
↪946015420752, 2.35, 0.15, 3.03361215187440, 1820.05673775234]
      - elements: ['C', 'Si', 'Si']
        coefficients: [1, 0.011304, 0, 19, 2.5, -1, 0.80468, 1, 1.96859970919818, 428.
↪946015420752, 2.35, 0.15, 3.03361215187440, 1820.05673775234]
      - elements: ['C', 'C', 'Si']
        coefficients: [1, 0.011304, 0, 19, 2.5, -1, 0.80469, 1, 1.76776695296637, 203.
↪208547714849, 2.35, 0.15, 2.54558441227157, 458.510465798439]
      - elements: ['C', 'Si', 'C']
        coefficients: [1, 0.011304, 0, 19, 2.5, -1, 0.80469, 1, 1.96859970919818, 428.
↪946015420752, 2.35, 0.15, 3.03361215187440, 1820.05673775234]
      - elements: ['Si', 'C', 'C']
        coefficients: [1, 0.013318, 0, 14, 2.1, -1, 0.78000, 1, 1.96859970919818, 428.
↪946015420752, 2.35, 0.15, 3.03361215187440, 1820.05673775234]
      - elements: ['C', 'C', 'C']
        coefficients: [1, 0.011304, 0, 19, 2.5, -1, 0.80469, 1, 1.76776695296637, 203.
↪208547714849, 2.35, 0.15, 2.54558441227157, 458.510465798439]
```

Equations

Not documented see publication: Gao and Weber, Nuclear Instruments and Methods in Physics Research B 191 (2012) 504.

3.2.4 Vashishta Potential

- lammps documentation
- example potential
- 14 parameters: $H, \eta, Z_i, Z_j, \lambda_1, D, \lambda_4, W, rc, B, \gamma, r_0, C, \cos(\theta)$

yaml schema

```
pair:
 - type: vashishta
   parameters:
     - elements: ['C', 'C', 'C']
       coefficients: [471.74538, 7, -1.201, -1.201, 5.0, 0.0, 3.0, 0.0, 7.35, 0.0, 0.
↪0, 0.0, 0.0, 0.0]
```

(continues on next page)

(continued from previous page)

```
    - elements: ['Si', 'Si', 'Si']
      coefficients: [23.67291, 7, 1.201, 1.201, 5.0, 15.575, 3.0, 0.0, 7.35, 0.0, 0.
↪0, 0.0, 0.0, 0.0]
    - elements: ['C', 'Si', 'Si']
      coefficients: [447.09026, 9, -1.201, 1.201, 5.0, 7.7874, 3.0, 61.4694, 7.35, 9.
↪003, 1.0, 2.90, 5.0, -0.333333333333]
    - elements: ['Si', 'C', 'C']
      coefficients: [447.09026, 9, 1.201, -1.201, 5.0, 7.7874, 3.0, 61.4694, 7.35, 9.
↪003, 1.0, 2.90, 5.0, -0.333333333333]
    - elements: ['C', 'C', 'Si']
      coefficients: [0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0,
↪ 0.0]
    - elements: ['C', 'Si', 'C']
      coefficients: [0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0,
↪ 0.0]
    - elements: ['Si', 'C', 'Si']
      coefficients: [0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0,
↪ 0.0]
    - elements: ['Si', 'Si', 'C']
      coefficients: [0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0,
↪ 0.0]
```

Equations

$$U = \sum_{i}^{N} \sum_{j>i}^{N} U_{ij}^{(2)}(r_{ij}) + \sum_{i}^{N} \sum_{j \neq i}^{N} \sum_{k>j, k \neq i}^{N} U_{ijk}^{(3)}(r_{ij}, r_{ik}, \theta_{ijk})$$

$$U_{ij}^{(2)}(r) = \frac{H_{ij}}{r^{eta_{ij}}} + \frac{Z_i Z_j}{r} \exp(-r/\lambda_{1,ij}) - \frac{D_{ij}}{r^4} \exp(-r/\lambda_{4,ij}) - \frac{W_{ij}}{r^6}, r < r_{c,ij}$$

$$U_{ijk}^{(3)}(r_{ij}, r_{ik}, \theta_{ijk}) = B_{ijk} \frac{[\cos\theta_{ijk} - \cos\theta_{0ijk}]^2}{1 + C_{ijk}[\cos\theta_{ijk} - \cos\theta_{0ijk}]^2} \times \exp\left(\frac{\gamma_{ij}}{r_{ij} - r_{0,ij}}\right) \exp\left(\frac{\gamma_i k}{r_{ik} - r_{0,ik}}\right), r_{ij} < r_{0,ij}, r_{ik} < r_{0,ik}$$

3.2.5 COMB Potential

- lammps documentation
- example comb potential
- example comb3 potential
- 46 parameters (comb), 71 parameters (comb3)

Spec not provided here because is so large. See examples.

Equations

$$E = \sum_i [E_i^{self}(q_i) + \sum_{j>i} [E_{ij}^{short}(r_{ij}, q_i, q_j) + E_{ij}^{Coul}(r_{ij}, q_i, q_j)] + E^{polar}(q_i, r_{ij}) + E^{vdw}(r_{ij}) + E^{barr}(q_i) + E^{corr}(r_{ij}, \theta_{jik})]$$

See publication for full parameter list.

- COMB - T.-R. Shan, B. D. Dvine, T. W. Kemper, S. B. Sinnott, and S. R. Phillpot, Phys. Rev. B 81, 125328 (2010)
- COMB3 - T. Liang, T.-R. Shan, Y.-T. Cheng, B. D. Devine, M. Noordhoek, Y. Li, Z. Lu, S. R. Phillpot, and S. B. Sinnott, Mat. Sci. & Eng: R 74, 255-279 (2013).

CHAPTER 4

Training Sets

DFTFIT uses Ab-Inito calculation to guide the optimization of potentials along with measured properties. The goal is to make it easier for users to include vasp calculations in their potential fitting. Thus DFTFIT has support for reading VASP `vasprun.xml`, Quantum Espresso `*.out`, and Siesta `*.xml` output files. The parsers will read as many sets that contain the structure, energy, stress, and forces. These output files may be the result of a relaxation, SCF, of BOMD, etc. calculation. If DFTFIT does not have support for the output format that you supply please submit an issue. Additionally measured properties include: lattice_constants, elastic_constants, bulk_modulus, and shear_modulus.

4.1 Measured Properties

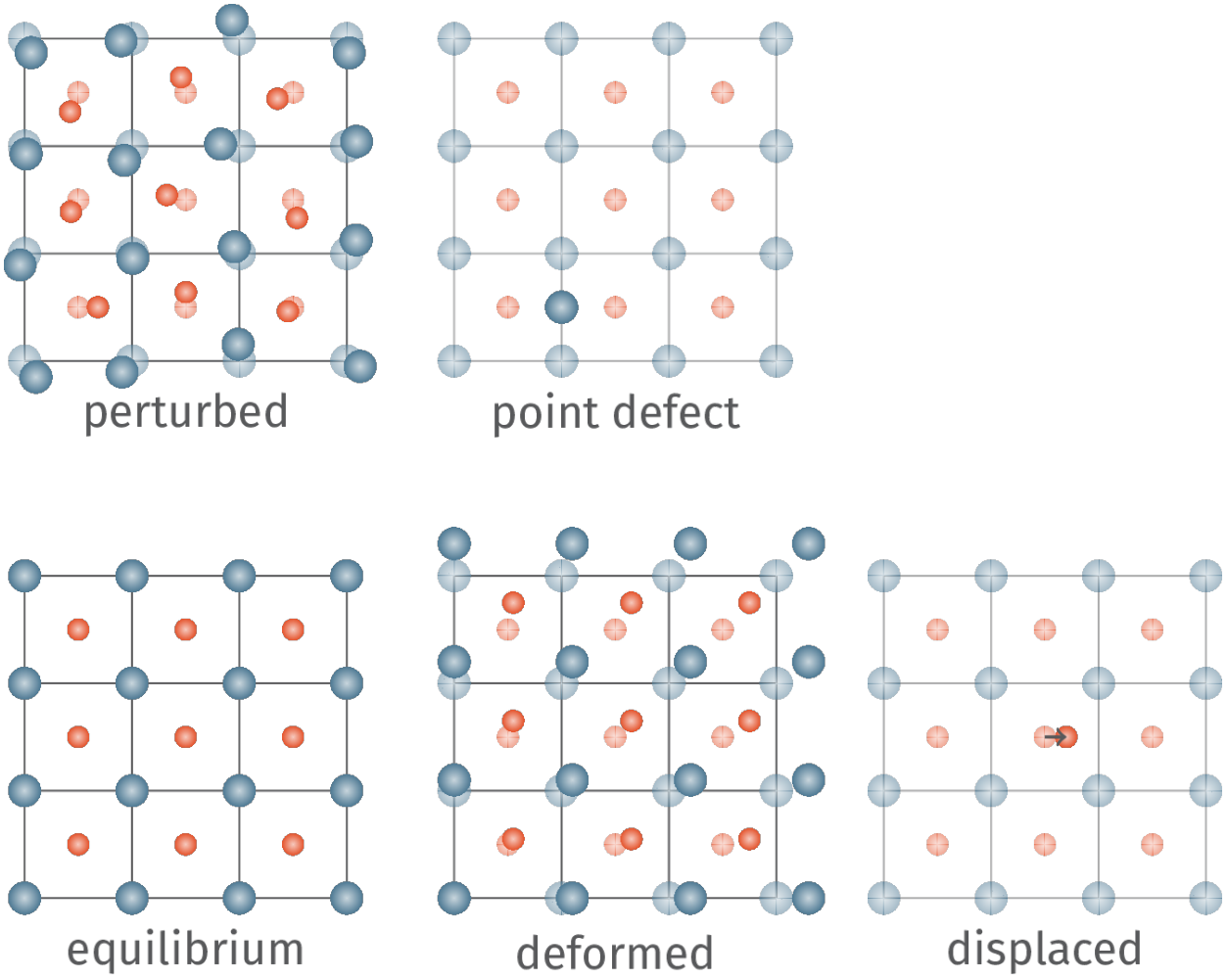
Recently DFTFIT has added support for experimental properties and other measured quantities. These include: lattice_constants, elastic_constants, bulk_modulus, and shear_modulus. In order to use one you must include a ground_state for an example see this input training file.

- lattice constants (lengths)
- elastic constants (voigt)
- bulk modulus
- shear modulus

4.2 Ab-Initio Training Sets

Example of training sets include:

- equilibrium structure
- displaced structures $0.01x212B - 0.04x212B$
- deformed structures $\pm\%2$ normal, $\pm\%8$ shear
- random perturbed structures $0.04x212B$



Be aware the DFT calculations that change the unit cell result in less accurate energy, stress, and forces. Thus an additional SCF calculation will be necessary.

In general a `selector` is used to get the input files.

- `selector.filename` select a specific output filename of `type`
- `selector.fileglob` select a specific set of output files that match glob of `type`.
- `selector.num_samples` for each matching file choose num_samples with maximum separation

An example Siesta training set is included below.

```
version: v1
kind: Training
spec:
  - type: Siesta
    selector:
      filename: test_files/siesta/d1_li_20ev.xml
      num_samples: 3
  - type: Siesta
    selector:
      filename: test_files/siesta/d1_o_30ev.xml
      num_samples: 4
  - type: Siesta
```

(continues on next page)

(continued from previous page)

```
    selector:
      filename: test_files/siesta/d1_ta_20ev.xml
      num_samples: 5
```

4.3 VASP

```
spec:
 - type: VASP
   selector:
     filename: test_files/vasp/vasprun.xml.mgo
```

4.4 Quantum Espresso

```
spec:
 - type: QE
   selector:
     filename: test_files/espresso/...
```

4.5 Siesta

```
spec:
 - type: Siesta
   selector:
     filename: test_files/siesta/d1_o_30ev.xml
     num_samples: 4
```

CHAPTER 5

Configuration

The configuration schema is where you specify all optimization settings and general DFTFIT settings such as which `sqlite` database to write to. See bellow for an example configuration file.

```
version: v1
kind: Configuration
metadata:
  name: simple test
  labels:
    test: simple
    hello: world
spec:
  logging: INFO
  database:
    filename: "test.db"
    interval: 10
  algorithm:
    name: 'pygmo.de'
    steps: 10
    population: 5
    include_initial_guess: False
  problem:
    calculator: 'lammps'
    command: 'lammps_serial'
    weights:
      force:  0.8
      stress: 0.1
      energy: 0.1
```

5.1 Metadata

DFTFIT allows a user to assign a name to an optimization run `metadata.name` along with arbitrary key value strings to the run. This metadata will be included in the SQLite database.

- `metadata.name` string name to assign to run
- `metadata.labels` key, value strings to assign to run

5.2 Optimization

DFTFIT gives the user explicit control over the optimization procedure. In general the number of potential evaluations is equal to `spec.population * (spec.steps + 1)`. This is because DFTFIT does one initial evaluations of guessed parameters. Note that optimization is broken into two parts. The problem is how DFTFIT evaluates the objective function. The algorithm is control over the optimization algorithm used on the objective function.

5.2.1 Problem

DFTFIT uses the problem to specify how it evaluates the objective function.

- `spec.problem.weights` weights to use in addition to the features to calculate. Available options include: force, stress, energy, lattice_constants, elastic_constants, bulk_modulus, shear_modulus. Note that even for multiobjective optimization functions a single objective value can be computed.

5.2.2 Algorithms

DFTFIT is unique in that it allows for both single and multi objective optimization. By using pagmo2 for optimization DFTFIT is able to offer 20+ single objective and several multi-objective algorithms. A list of some of the notable algorithms include.

- SADE
- LBFGS
- Bee Colony
- MOEAD

Values

- `spec.algorithm.name` pagmo2 optimization algorithm to use
- `spec.algorithm.steps` number of steps to take in optimization
- `spec.algorithm.population` number of guesses per optimization step
- `spec.algorithm.include_initial_guess` whether to include the initial values from the potential schema

5.3 SQLite Database

Most scientific software writes output to a custom binary output file or `json` files. DFTFIT writes all optimization information to an SQLite database. This provides MANY benefits.

- several concurrent runs can write to the same file
- since sqlite is a database you can evaluate the progress of the optimization in realtime
- sqlite is fault tollerant meaning that the change of corruption is very low

For easily viewing the results DFTFIT provides serveral methods in `dftfit.db_actions`. Also you may use any available SQLite viewer such as the free sqlitebrowser.

- `spec.database.filename` controls the sqlite filename that all information is written to
- `spec.database.interval` controls how dftfit batches writes of evaluation information. Each write takes around 1-20 ms depending on system.

5.4 MD Calculator

DFTFIT originally only had one MD calculator `lammps`. However it worked by writting input files and then telling lammps to run them. This was not ideal so a new calculator was written that uses lammps-cython. This calculator integrated *LAMMPS* within the python process.

It is at least 5X-10X faster and is the recommended calculator.

- `spec.problem.calculator` set that DFTFIT calculator to use. Recommended `lammps_cython`. Available: "lammps", "lammps_cython"
- `spec.problem.command` only used by "lammps" calculator to specify the executable path.
- `spec.problem.num_workers` allows for parallelism of DFTFIT optimization. Does not scale well past 6 workers (1500 lammps calculations/second).

5.5 Miscellaneous

- `spec.logging` controls the verbosity of DFTFIT (DEBUG, INFO, WARNING, CRITICAL)

CHAPTER 6

Commands

6.1 Merging Databases

Everything in DFTFIT is backed by an SQLite3 database. Often times when running calcluations on supercomputers your calculations will be stored in several databases. In fact this is recommended because SQLite3 does not handle concurrency and distributed file systems well. `dftfit db merge` will combine several databases into one. It can also de-duplicate calculations.

```
dftfit db merge database1.db  database2.db -o database.db
```

6.2 Evaluating Potentials

A challenging area of using potentials for molecular dynamics calculations is checking that the potential is stable for your work. DFTFIT makes this easy by taking in a potential plus the ground state structure. It will predict the:

- lattice constants
- elastic constants
- energy, forces, and stress of configuration

```
dftfit test properties -p test_files/potential/mgo.yaml \
                       -s test_files/structure/MgO.cif
```

Output:

```
Lattice Constants:
      a:  4.199    b:  4.199     c:  4.199
  alpha: 90.000 beta: 90.000 gamma: 90.000

Elastic:
Stiffness Tensor
```

(continues on next page)

(continued from previous page)

```
  -313.8   -148.5   -179.1     -0.0     -0.0     -0.0

  -117.1   -333.8   -179.1     -0.0     -0.0     -0.0

  -117.1   -148.6   -353.2     -0.0     -0.0     -0.0

    -0.0     +0.0     -0.0   -120.7     -0.0     +0.0

    -0.0     +0.0     -0.0     -0.0   -120.7     +0.0

    -0.0     -0.0     -0.0     +0.0     -0.0   -120.7



Shear Modulus G_V -105.36002528878238
Shear Modulus G_R -106.1332136372622
Shear Modulus G_vrh -105.74661946302228
Bulk Modulus K_V -210.04961372048925
Bulk Modulus K_R -211.15489789675007
Bulk Modulus K_vrh -210.60225580861965
Elastic Anisotropy -0.04165984250465549
Poisons Ration 0.284937698051096

Static:
    Energy: [eV]
               -14.559
    Forces: [eV/Angstrom]
                 8.500           20.370           -9.638
                -8.500          -20.370            9.638
    Stress: [bars]
               118.599          211.146          -29.249
               211.146          399.328          -53.051
               -29.249          -53.051           33.288
```

DFTFIT will perform the calculation on the input structure so it may be important to relax it with the potential beforehand.

```
dftfit test relax -p test_files/potential/mgo.yaml \
                  -s test_files/structure/MgO.cif \
                  -o MgO-relaxed.cif
```

6.3 Summarizing DFTFIT runs

After performing many calculations you may want to quickly look through the database and look at the progress. It will print interesting features that indicate the success of the optimization.

```
dftfit db summary test_files/database/database.db
```

Output:

```
run: 1
      algo: pygmo.sade       steps:         21
      stats:
            mean:    1.0
```

(continues on next page)

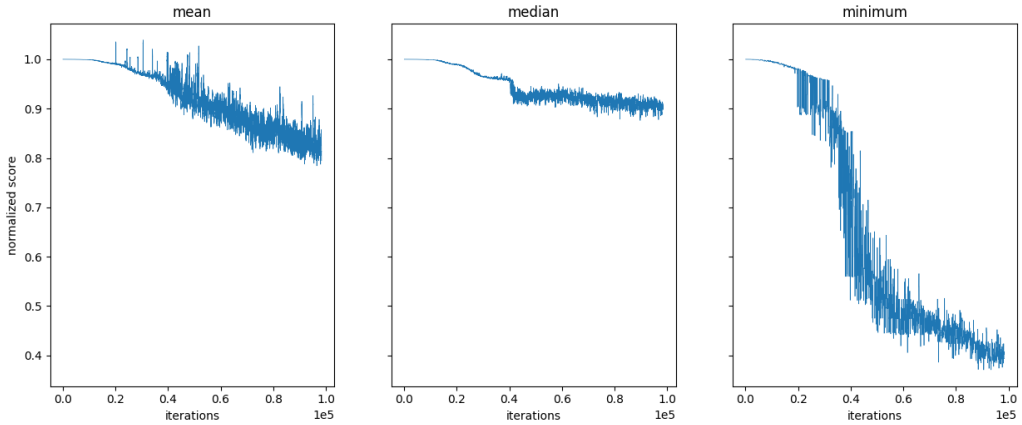
(continued from previous page)

```
            median:  1.0
            min:     1.0
      final:   -1.98,  177, 1.92,  871, 74.6, 1.82e+05, 88.2,  976
      score:   0.9999780011307999

run: 2
        algo: pygmo.sade       steps:      26248
        stats:
              mean:   0.932
              median: 0.925
              min:    0.86
        final:   -1.31, 0.741, 3.31, 12.8, 0.26, 1.47e+03, 0.404,  640
        score:   0.8522755981712459

run: 3
        algo: pygmo.sade       steps:      26248
        stats:
              mean:   0.977
              median: 0.973
              min:    0.956
        final:   -1.31, 46.6, 0.143, 20.4, 0.614, 3.67e+03, 0.397,  995
        score:   0.8908135790668292

run: 4
        algo: pygmo.sade       steps:      98366
        stats:
              mean:   0.795
              median: 0.902
              min:    0.393
        final:   1.77,  163, 0.395,  348, 0.38, 1.88e+03, 0.382,  811
        score:   0.37092898795248247

run: 5
        algo: pygmo.sade       steps:      26220
        stats:
              mean:   0.909
              median: 0.898
              min:    0.823
        final:   -1.69,  2.5, 96.5, 4.66e+03, 0.229, 1.12e+03, 0.411,  773
        score:   0.7009948887639249
```

6.4 Visualize Progress of Run

The DFTFIT summary command only gives a some information about the convergence but cannot show the full progress through time. To address this you can visualize the convergence of a single run.

```
dftfit db progress test_files/database/database.db --run-id=4
```



6.5 Training Set Radial Distribution

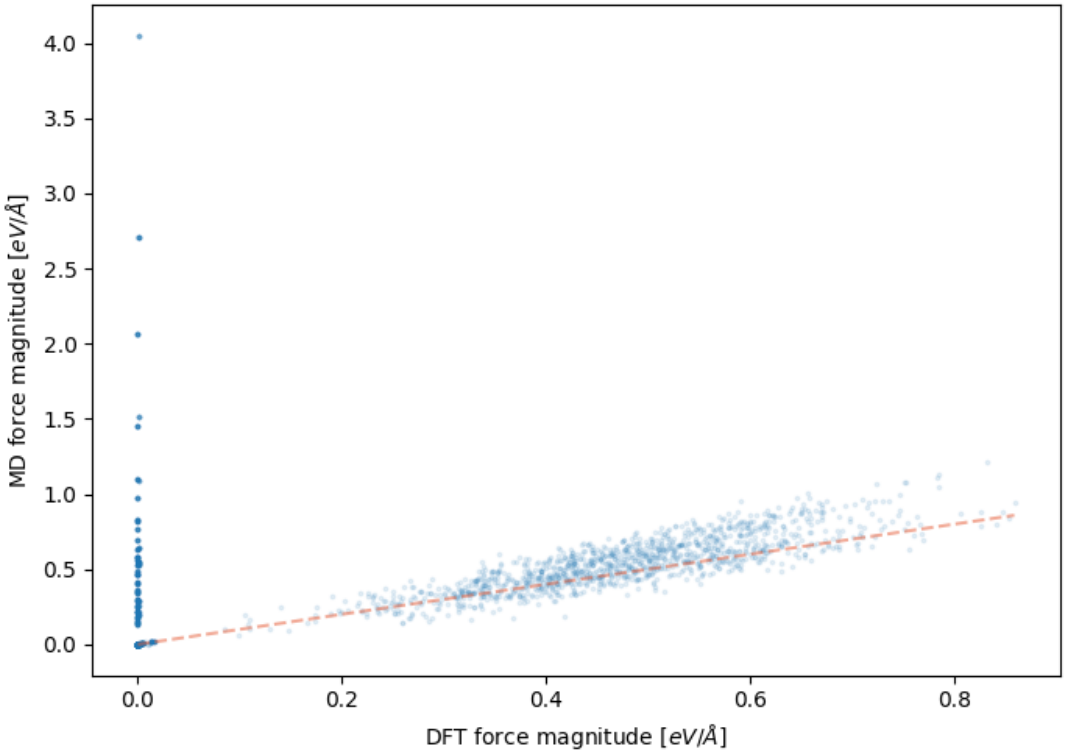
One of DFTFIT's main goals is to produce more transferable potentials. To enable this you can visualize the pair distribution of each atom combination. The picture below show training from 140 structures close to equilibrium indicated by the sharp peaks.

```
dftfit test radial -t test_files/training/training-full-mgo.yaml
```

6.6 Visualize Potential Error on Training Set

So far the only indicator for progress was $\Delta E, \Delta F, \Delta S$. It is not easy to visualize how the potential is fitting the DFT data. For this DFTFIT provides methods to visualize the error or the energy, stresses, and forces. Bellow is an example of visualizing the force error. In this image we can see that the MD predicted forces tend to be higher than the DFT forces. The red dashed line indicates a one to one match between the data.

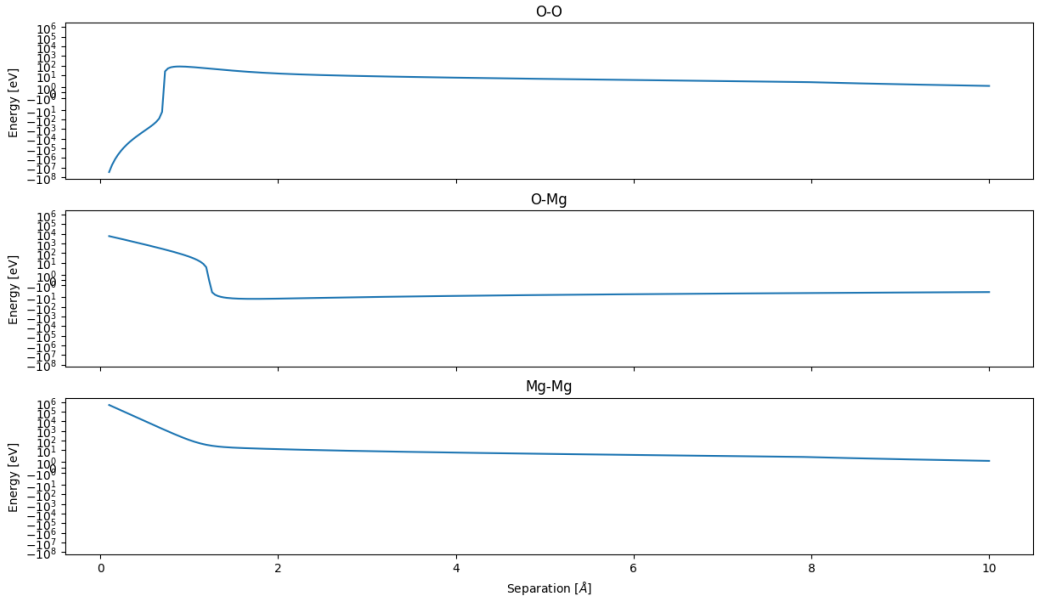
```
dftfit test training forces -p test_files/potential/mgo.yaml \
                            -t test_files/training/training-full-mgo.yaml
```



6.7 Visualize Pair Potential

Often times complex pair potentials are used in combination with three body and n body terms. This leads to a complicated pair potential interaction. This command will visualize the resulting pair potential from the model for each pair of atoms. This method just does a simple evaluation of the pair of atoms at different separations in a large periodic box (much larger than separation).

```
dftfit test pair -p test_files/potential/mgo.yaml
```


O-O
O-Mg
Mg-Mg
Energy [eV]
Separation [Å]
0
2
4
6
8
10

CHAPTER 7

Visualization

DFTFIT has many tools for visualizing the progress and results from potential fitting.

For now the visualization is mostly in the commands documentation.

TODO

CHAPTER 8

Improving Performance

DFTFIT's performance is predictable. A great amount of effort has been put into ensuring that the code is benchmarked. See the benchmark tests to view the time it takes for certain methods to complete.

The optimization is limited by the time each `LAMMPS` calculator evaluation takes. For configurations of around 100-200 atoms this takes around 1-10 ms. DFTFIT provides a method to parallelize these calculations among processors `lammps.problem.num_workers`.

So for example if you have 100 training images. You can expect without parallelism you will achieve around 5 iterations per seconds. The code scales almost ideally with more processors.

CHAPTER 9

Indices and tables

- genindex
- modindex
- search