FLAME Documentation

Release 1.0

Maximilian Amsler
S. Alireza Ghasemi

Oct 27, 2019

CONTENTS

CHAPTER

ONE

WELCOME TO FLAME

FLAME – a library of atomistic modeling environments – is a software package to perform a wide range of atomistic simulations for exploring the potential energy surfaces (PES) of complex condensed matter systems. Available methods includes molecular dynamics simulations to sample free energy landscapes, saddle point searches to identify transition states, and gradient relaxations to find dynamically stable geometries. In addition to such common tasks, FLAME implements a structure prediction algorithm based on the minima hopping method (MHM) to identify the ground state structure of any system given solely the chemical composition, and a framework to train a neural network potential to reproduce the PES from *ab initio* calculations. The combination of neural network potentials with the MHM in FLAME allows highly efficient and reliable identification of the ground state as well as metastable structures of molecules and crystals, as well as of nano structures, including surfaces, interfaces, and two-dimensional materials.

CHAPTER TWO

THE FLAME TEAM

The development of methods implemented in FLAME started back in 2007 in Basel, Switzerland, with the implementation of various atomistic simulation tasks in different software packages. These tasks were later merged into two main codes (Alborz and Minhocao), which were independently maintained and developed by S. Alireza Ghasemi and Maximilian Amsler, respectively. In 2018, the FLAME code evolved as a streamlined integration of these two codes into a single package to facilitate atomistic simulation workflows.

The core developer team consists of:

- S Alireza Ghasemi
- Maximilian Amsler
- Samare Rostami
- Hossein Tahmasbi
- Ehsan Rahmatizad
- Somayeh Faraji
- Robabe Rasoulkhani

Following contributors are greatefully acknowledged:

- Luigi Genovese
- Thomas Lenosky
- Li Zhu
- Miguel Marques

CHAPTER THREE

LICENSING AND CITATION

FLAME is distributed under the GPLv3 license.

Please cite reference [1] when publishing results based on using FLAME, which describes the main functionalities of the code.

Additionally, consider citing the following publications when using specific portions of the FLAME code:

- Neural network potential (CENT): [2]
- Minima hopping method: [3][4][5]
- SQNM optimizer: [6]
- Saddle point searches: [7][8]
- Structural fingerprints: [9][10]
- Electrostatic particle-particle, particle-density method: [11][12]
- Interatomic potentials for silicon: [13]

It is the responsibility of the user to follow the respective licensing terms when FLAME is used in conjunction with external (quantum) engines.

CHAPTER FOUR

DOWNLOAD AND INSTALLATION

4.1 Download

The FLAME software is available on the flame website http://flame-code.org, and the latest development version can be found on GitHub https://github.com/flame-code.

Download the current version of FLAME with

```
git clone https://github.com/flame-code/FLAME.git
```

4.2 Prerequisites

FLAME requires **autoconf** and **automake**. IMPORTANT NOTE: currently only **automake** up to version 1.15.1 is supported due to changes introduced in later versions that break the Makefile structure.

Any Fortran and C compiler should in principle work for compiling FLAME. However, we recommend using the Intel Fortran and C compiler.

FLAME has to be linked to Blas, LaPack, and FFT libraries. They can be obtained as part of the Intel Math Kernel Library (MKL), which is the recommended route. In principle, other implementations of the libraries should also work.

Linking to Atsushi Togo's SPGLIB is recommended. The currently supported and well-tested version is 1.6.x and can be found here:

https://sourceforge.net/projects/spglib/files/spglib/

Linking to LAMMPS requires the installation of LAMMPS with the desired potentials. The best upported and well tested version is r12824:

http://lammps.sandia.gov

Futile is required, a library of tools developed as part of the BigDFT project.

http://bigdft.org/

Installation of python is required. Currently, only python 2.7 is supported. Future releases of FLAME will support python 3

4.3 Installing FLAME

4.3.1 On Linux

It is recommended to install FLAME in a different directory than the source code.

1. First, install futile which is a set of utilities from the BigDFT project. Preferably, use the version provided with FLAME to avoid conflicts. Untar the included futile-suite.tar.gz (`tar -zxvf futile-suite.tar.gz`), then create a new build directory (e.g., `mkdir futile-build ; cd futile-build`), and from there run

   for GNU compilers:

   ```
   path_to_futile_source/Installer.py build futile -c CC=gcc CXX=g++
   FC=gfortran F77=gfortran \ --with-ext-linalg="-llapack -lblas"
   ```

   for Intel compilers:

   ```
   path_to_futile_source/Installer.py build futile -c FCFLAGS=-O2
   \ --with-ext-linalg="-L$MKLROOT/lib/intel64 \ -lmkl_rt
   -lmkl_scalapack_lp64 -lmkl_blacs_openmpi_lp64 -liomp5 -lm" \
   CC=icc CXX=icpc FC=ifort F77=ifort
   ```

   for parallel compilation use the corresponding MPI wrappers:

   ```
   path_to_futile_source/Installer.py build futile -c FCFLAGS=-O2
   \ --with-ext-linalg="-L$MKLROOT/lib/intel64 \ -lmkl_rt
   -lmkl_scalapack_lp64 -lmkl_blacs_openmpi_lp64 -liomp5 -lm" \
   CC=mpicc CXX=mpicxx FC=mpif90 F77=mpif90
   ```

   follwed by `make build` if necessary. Make sure to adapt the library locations and the linking flags appropriately. After the installation of futile, we need to link it with FLAME. To display details on the general linking procedure, run:

   ```
   path_to_futile_source/Installer.py link futile
   ```

2. To compile FLAME, change into the main FLAME directory and run:

   ```
   autoreconf -fi
   ```

3. Create a build directory for FLAME (e.g., `mkdir build-FLAME ; cd build-FLAME`). Explicitly replace $FUTILE with the full path of the futile build-directory during `configure`, or define it as an environmental variable:

   ```
   export FUTILE=path_to_futile_build
   ```

   Note that providing the FUTILE variable is required to successfully compile FLAME, and is not optional. Then, run `configure`. For Intel compilers and MPI parallelization:

   ```
   path_to_flame_source/configure FC=mpif90 F77=mpif90 CXX=mpicc CC=mpicc
   \ FCFLAGS="-I$FUTILE/install/include -shared-intel -mcmodel=large
   -mkl=sequential" \ CFLAGS=-mcmodel=large "LIBS=-L$FUTILE/install/lib
   \ -L/$MKLROOT/lib/intel64 -lfutile-1 -lmkl_rt \ -lmkl_scalapack_lp64
   -lmkl_blacs_openmpi_lp64 -liomp5 -lm -lyaml -ldl -lrt -cxxlib"
   ```

   For GNU compilers without MKL:

   ```
   path_to_flame_source/configure FC=mpif90 F77=mpif90 CXX=mpicc CC=mpicc \
   FCFLAGS="-I$FUTILE/install/include -mcmodel=large" \ CFLAGS=-mcmodel=large
   "LIBS=-L$FUTILE/install/lib \ -lfutile-1 -lm -lyaml -llapack -lfftw3 -ldl
   -cxxlib"
   ```

To link with SPGLIB, append `--with-spglib SPGLIB_ROOT=path_to_spglib`

To link with the BigDFT PSolver, append `--with-bps BDIR=path_to_bigdft_root`

To link with LAMMPS, append `--with-lammps LAMMPS_ROOT=path_to_lammps_root`

4. Run `make` to compile the code. Upon successful compilation, the executable can be found in `src/flame`

CHAPTER

FIVE

RUNNING FLAME

FLAME consists of a single executable named `flame`, which resides in the `src` directory upon successful compilation. The main input file *flame_in.yaml* must be provided for all FLAME runs and contains the simulation directives. Most simulation tasks require additonal input files, like the input structure (see *yaml Format*), parameter files for atomic potentials, or run scripts to call external codes.

5.1 Input Parameters

5.1.1 The `flame_in.yaml` file

The contents of *flame_in.yaml* is arranged in labeled blocks. Python-style indentation is used to indicate nesting. Blocks contain keyword–parameter pairs as well as subblocks.

Following top-level blocks are available.

- *main*
- *potential*
- *fingerprint*
- *geopt*
- *geopt_prec*
- *dynamics*
- *ann*
- *single_point*
- *saddle*
- *saddle_opt*
- *minhocao*

Only the *main* block must be present in every FLAME run.

5.1.2 Main Block

The options available in the **main** block of the *flame_in.yaml* file determines the overall task to be performed in FLAME and sets up the atomistic simulation environment.

task: (string) Determines the main FLAME task.

default: `No default value.`

options:

`geopt`: Local geometry optimization. The parameters of this task can be set within the block *geopt*.

`saddle`: Saddle point search. The parameters of this task can be set within the block *saddle*.

`dynamics`: Molecular dynamics simulation. The parameters of this task can be set within the block *dynamics*.

`ann`: All tasks related to the training and evaluation of artificial neural network potentials. Subtasks can be selected in the block *ann*.

`single_point`: Perform a single point calculation to evaluate the energy, forces, and stresses for one or more configurations. Allows linking to external sampling codes. The parameters for this taks can be specified in the block *single_point*.

`minhocao`: Perform a global optimization calculation based on the minima hopping method. The parameters for this task can be set within the block *minhocao*.

two_level_geopt: (logical) Determines if geometry optimizations are performed with two accuracy levels. If `True`, the block **geopt_prec** must be present in *flame_in.yaml*.

default: `False`

verbosity: (integer) Verbosity of output data.

default: `0`

options: `0, 1, 2, 3` Increasing number for higher verbosity.

verbosity_mh: (integer) Verbosity of the output related to the task `minhocao`.

default: `3`

options: `0, 1, 2, 3` Increasing number for higher verbosity.

nat: (integer) Number of atoms involved in the simulation.

default: `0`

types: (list of strings of length [number of atomic types]) Character of the atoms involved in the simulation. Internally, the atomic types are enumerated, starting from 1.

default: `No default value.`

znucl: (list of integers of length [number of atomic types]) The atomic number of the atoms involved in the simulation, charge of the nuclei. Corresponds to an equivalent input of the **types** keyword.

default: `No default value.`

amass: (list of reals of length [number of atomic types]) Overrides the physical atomic masses in dynamics simulations. Particularly useful to reduce the spectrum of the vibrational eigenfrequencies in MD escape trials during minima hopping runs.

default: `No default value.`

typat: (list of integers of length **nat**) Indexes of the atomic types from 1 to number of atomic types.

default: `No default value.`

findsym: (logical) Activates symmetry detection of crystalline solids. FLAME must be compiled and linked with SPGLIB.

default: `False`

rng_type: (string) Not used for production runs, only used for development and regression tests.

seed: (integer) Seed value to initialize the random number generator.

default: `-2`

pressure: (real) External pressure. In units of GPa default: `0.d0`

params_new: (logical) Enables parsing the `params_new.in` for the **task** `minhocao` runs. This functionality is available to allow backwards compatibility with earlier versions of `minhocao`. This option will be removed in the future.

default: `False`

5.1.3 Potential

The interatomic potential parameters are set in this block. Some parameters are used by all methods, while others apply only to specific interatomic potentials.

potential parameters

general `potential` parameters

potential: (string) Method to evaluate the atomic interaction.

default: `No default value.`

options:

Methods implemented in FLAME:

`lj`: Lennard-Jones.

`blj`: Binary Lennard-Jones.

`mlj`: Multicomponent Lennard-Jones with arbitrary number of species.

`ltb`: Lenosky's Tight-Binding method for silicon [13].

`edip`: Bazant's Environment-Dependent Interatomic Potential for silicon [13].

`tersoff`: Tersoff's potential for silicon and sp3 carbon [13].

`ann`: Artificial neural network potential.

`coulomb`: Coulomb potential (electrostatic interactions).

External codes that can be linked to FLAME:

`tinker`: TINKER molecular mechanics code [14].

`lammps`: LAMMPS molecular mechanics code [15].

`mopac`: MOPAC molecular mechanics code [16].

`dftb`: Density functional tight binding as implemented in dftb+ [17].

`bigdft`: BigDFT wavelet DFT code [18].

`vasp`: Plane wave VASP code [19].

`abinit`: Plane wave Abinit code [20].

`espresso`: Plane wave Quantum Espresso code, pw.x [21].

`siesta`: Siesta DFT code [22].

`cp2k`: CP2K DFT/QM/MM code [23].

`msock`: Network socket interface. Uses the i-Pi protocol to interact with external codes [24].

potential_sec: (string) Secondary interatomic potential, usually used to perform a preliminary relaxation or to evaluate an approximate Hessian matrix. All available options are identical to **potential**.

default: None

core_rep: (logical) If selected, a repulsive potential to the **potential** method is added based on the atomic type. Some potentials (like PAW DFT) tend to cause issues if atoms get too close to each other and the core regions start to overlap. To avoid atoms from getting too close, a repulsive $\frac{1}{r^{12}}$ term is added.

default: `False`

kptmesh: (list of three integers) Desired k-points mesh. It will be overruled if **auto_kpt** is `True`. Only relevant for periodic electronic structure codes.

default: `[1, 1, 1]`

auto_kpt: (logical) Activates a scheme to automatically compute the k-points mesh given a predefined density. Only relevant for periodic electronic structure codes.

default: `True`

kptden: (list of two reals) Desired k-points density along every dimension for the fine and the coarse potential settings. In units of the reciprocal lattice vectors, $2\pi/$Bohr. Recommended values are in the range of `0.015` and `0.040` for metals and insulators, respectively. Only relevant for periodic electronic structure codes.

default: `[4.d-2, 6.d-2]`

`msock` parameters

sockinet: (integer) Selects Unix socket or internet (TCP) socket.

default: `0`

options:

`0`: Unix socket

`1`: internet (TCP) socket

sockport: (integer) Socket port number.

default: `3141`

sockhost: (string) Socket address. If **sockinet** is `0`, a string with the **sockhost** name will be created in a temporary directory. Otherwise, a valid IP address must be provided (*127.0.0.1* for localhost).

default: `mh-driver`

sockcutwf: (list of two reals) Plane wave cutoff energies for the fine and coarse settings sent along with the i-Pi protocol. Only relevant for plane wave DFT codes that support this feature (like Quantum Espresso).

default: `[1.d0, 1.d0]`

`confine` parameters

confine: One or more 2D confinement potentials can be imposed based on polynomial functions. The general form of the potential is $P = A(|e - \mathbf{r}_i^\alpha| - r_c)^n$. Where A is the amplitude, e is the equilibrium position along the dimension α, r_c is the cutoff distance, and i runs over all atoms that interact with the potential P.

confinement: (logical) Determines if one or more 2D confinement potentials will be imposed.

default: `False`

nconfine: (integer) Number of confinement potentials.

default: `0`

cartred: (string) Choice of Cartesian or reduced coordinates for setting up the confinement potential. Given as a list of length **nconfine** if more than one confinement potential is imposed.

default: `C`

options:

`C`: Cartesian coordinates

`R`: Reduced coordinates

dim: (integer) Axis along which the confinement potential is applied. Given as a list of length **nconfine** if more than one confinement potential is imposed.

default: `1`

options: `1`, `2`, `3` for the x, y and z directions, respectively.

exp: (integer) Exponent *n* of the potential. Given as a list of length **nconfine** if more than one confinement potential is imposed.

default: `4`

prefac: (real) Prefactor or the amplitude *A* of the potential, in units of eV. Given as a list of length **nconfine** if more than one confinement potential is imposed.

default: `1.d-2`

cut: (real) Cutoff distance r_c of the potential, in units of Angstrom. Given as a list of length **nconfine** if more than one confinement potential is imposed.

default: `1.d0`

av: (integer) Method of defining the equilibrium position e of the potential. Given as a list of length **nconfine** if more than one confinement potential is imposed.

default: `2`

options:

`1`: The equilibrium position is set once during initialization with respect to a predetermined value along the dimension α set in **dim**

`2`: The equilibrium position is set dynamically with respect to the average value of all involved atoms along the dimension α set in **dim**

eq: (real) Equilibrium position e_i of the potential. Only relevant if **av** is set to `1`. The unit depends on the choice of **cartred**: Angstrom for `C`, in reduced units if `R`. Given as a list of length **nconfine** if more than one confinement potential is imposed.

default: `0.d0`

nat: (integer) Number of atoms that are subjected to the potential. Given as a list of length **nconfine** if more than one confinement potential is imposed.

default: `0`

nat: (list of integers and/or strings) The indices of the atoms that are subjected to the potential. If all atoms are affected by the potential, the string `all` can be used instead of listing all atomic indices. Given as a list of length **nconfine** (list of lists) if more than one confinement potential is imposed.

default: `all`

options:

`all`: all atoms are subjected to the potential

`[...]`: list of atomic indices

`ewald` parameters

ewald: If electrostatics is a part of the interactions in any FLAME potential, e.g., in the CENT potential, then the `ewald` key can be used to set the relevant parameters.

ewald: (logical) This subkey determines whether the Ewald method is invoked. If `True`, the Ewald approach is used up to speed up the calculations, especially when the calculations involve localized charge densities, e.g., when the Gaussian width of atomic charge densities in CENT are small.

default: `False`

psolver: (string) Determines the method for the Poisson solver.

default: `No default value.`

options:

`p3d`: The P3D method is used, applicable only for slab boundary conditions.

`kwald`: Fourier summation, applicable only in the CENT potential and for bulk boundary condition.

`bigdft`: The BigDFT PSolver is invoked if FLAME is linked with the BigDFT PSolver. Currently, only applicable for bulk and free boundary conditions.

cell_ortho: (logical) Activates efficient subroutines to place Gaussian charge densities on the grid. `True` can be used only when the simulation cell is orthogonal and the type of simulation does not change the cell variables. If `False`, then generic subroutines are called to put Gaussian charge densities on the grid.

default: `False`

ecut: (real) The cutoff energy that specifies how dense the basis set is when solving the Poisson's equation. The value is used for every non-pairwise method available in FLAME. There is no default value and it must be set. Units in Ha.

default: `No default value.`

ecutz: (real) The cutoff energy that specifies how dense the basis set is in the z-direction when solving the Poisson's equation. The value is used only when the `p3d` method is selected in **psolver**. There is no default value and it must be set. Units in Ha.

default: `No default value.`

rgcut: (real) The cutoff radius beyond which the atomic Gaussian charge densities are assumed to vanish. This parameter is *not* the actual cutoff radius but is a unitless parameter that is multiplied by the Gaussian width value. There is no default value and it must be set. Typically, `6.0` is a reasonable choice, and for very high accuracy one may use values up to `9.0`. Arbitrary units.

default: `No default value.`

bias_type (string) Select schemes to treat external fields or special boundary conditions if the `p3d` method is used.

default: `no`

options:

`p3d_bias`: For modeling conditions in which an ionic material is confined between two parallel metallic plates at two different electric potentials. For more information see [12].

`fixed_efield`: An external uniform electric field is applied along the non-periodic z-direction.

plane_voltageu (real) Voltage of the upper plate used if `p3d_bias` is selected for bias_type. Units in Ha/e.
default: `0.d0`

plane_voltagel (real) Voltage of the lower plate used if `p3d_bias` is selected for bias_type. Units in Ha/e.
default: `0.d0`

external_field (real) Uniform electric field used if `fixed_efield` is selected for bias_type. Units in Ha/e/Bohr.
default: `0.d0`

5.1.4 Fingerprint

The structural fingerprint parameters are set in this block. Some parameters are used by all methods, while others apply only to specific fingerprints. Note that the fingerprint methods discussed here are used in conjunction with the `minhocao` task.

fingerprint parameters

general `fingerprint` parameters

method: (string) Method to evaluate the structural fingerprint.

default: `oganov`

options:

`oganov`: Oganov's fingerprint method for crystalline systems based on radial distribution functions [25].

`gauss`: Gaussian orbital overlap matrix method for clusters and molecules [9].

`gom`: Gaussian orbital overlap matrix method for crystalline system [10].

`molgom`: Gaussian orbital overlap matrix method for molecular crystals [10].

`coganov`: Continuous version of Oganov's fingerprint, without histogram discretization along the radial direction.

`bcm`: Bond characterization matrix method for crystalline systems based on the Calypso method [26].

`atorb`: Atomic orbitals overlap method for crystalline systems.

rcut: (real) Cutoff distance of the fingerprint, in units of Angstrom. Only relevant for periodic, crystalline systems.

default: `15.d0`

`oganov` parameters

dbin: (real) Bin size for the discretization of the Oganov fingerprint, in units of Angstrom.

default: `5.d-2`

`coganov` parameters

atnmax: (integer) Maximum number of neighboring atoms to take into account. Only used to allocate temporary arrays.

default: `10000`

`oganov` & `coganov` parameters

sigma: (real) Broadening of the Gaussian functions placed on the atomic coordinates, in units of Angstrom.

default: `2.d-2`

`bcm` & `atorb` parameters

nl: (integer) Maximum degree l of the spherical harmonics to include.

default: `6`

`molgom` parameters

principleev: (integer) Number of principle eigenvectors for the molecular contraction.

default: `6`

molecules: (integer) Number of molecules in the system.

default: `1`

expa: (integer) Number of neighboring cells to be considered when computing the fingerprint.

default: `1`

molsphere: (integer) Maximum number of molecules taken into account in the cutoff sphere.

default: `50`

widthcut:: (real) Characteristic length scale of the spherical molecular cutoff function. Arbitrary units, scaling factor.

default: `1.d0`

widthover: (real) Characteristic width of the molecular overlap orbitals, which will be scaled by the Van der Waals radii.

default: `1.d0`

`gom` & `molgom` parameters

natx: (integer) Maximum number of neighboring atoms allowed in the cutoff shell, per atom.

default: `75`

nexcut: (integer) Exponent of the spherical cutoff function.

default: `3`

orbital: (string) Degree of Gaussian type orbitals to include.

default: `S`

options:

`S`: s-type orbitals

`P`: p-type orbitals

5.1.5 Optimizer

The geometry optimizer parameters can be set in this block. Some parameters are used by all methods, while others apply only to specific optimizers.

geopt parameters

general `geopt` parameters

method: (string) Optimization method.

default: `No default value.`

options:

`sd`: steepest descent method with energy feedback.

`sdcg`: steepest descent method followed by the conjugate gradient method.

`sddiis`: steepest descent method followed by the direct inversion in the iterative subspace (DIIS) method.

`nlbfgs`: Nocedal's implementation of the limited momory Broyden–Fletcher–Goldfarb–Shanno (LBFGS) method [27].

`bfgs`: Ghasemi's implementation the BFGS method with preferential moves along soft modes.

`qbfgs`: (only withing **task** `minhocao`) Quantum Espresso's version of BFGS for variable cell shapes [21].

`fire`: (only withing **task** `minhocao`) The Fast Inertial Relaxation Engine (FIRE) method [28].

fmaxtol: (real) Convergence parameter. The optimization will terminate as soon as the maximum absolute value of all force vector components falls below this threshold. In units of Ha/Bohr. Must be specified.

default: `No default value.`

alphax: (real) Sandard step size used in most optimizers. Optimal value depends on the system and method.

default: `No default value.`

lprint: (logical) Verbosity setting. If `True`, detailed information at each iteration is printed.

default: `False`

nit: (integer) Maximum number of iterations.

default: `1000`

dxmax: (real) Maximum displacement for each atomic component. Units in Bohr. Not yet implemented for all **methods**.

default: `1.d-1`

funits: (real) Factor to scale energy and force units. Currently only useful for Lennard-Jones potential, where the units can be scaled to be comparable to other potentials.

default: `1.d0`

cellrelax: (logical) Activates variable cell shape relaxations if set to `True`. Not yet implemented in all **methods**.

default: `False`

strfact: (real) The stress tensor is scaled by this factor and treated like forces in the process of optimization. Note that internally the unit of the stress tensor is $\mathrm{Ha/Bohr}^3$. Only relevant for variable cell shape relaxations.

default: `1.d2`

geoext: (logical) Some atomic simulation packages (e.g., LAMMPS, GULP, etc.) come with their own implementations of geometry optimizers. If `True`, these external optimizers will be used. Not yet implemented for all external codes.

default: `False`

`bfgs` parameters

condnum: (real) Predetermined condition number of the system.

default: `1.d1`

precaution: (string) Weighting of the quasi-Hessian. In Ghasemi's controlled BFGS, moves are gradually affected by the quasi-Hessian instead of the initial, diagonal matrix. The parameter *precaution* controls the rate of this gradual transition.

default: `normal`

options:

`high`: A high precaution is employed and the quasi-Hessian is applied with a high rate.

`normal`: A normal precaution is employed and the quasi-Hessian is applied with a medium rate.

`low`: A normal precaution is employed and the quasi-Hessian is applied with a low rate.

`fire` parameters

dt_start: (real) Initial time step. Arbitrary units.

default: `No default value.`

dt_min: (real) Minimal time step. Arbitrary units.

default: `1.d0`

dt_max: (real) Maximal time step. Arbitrary units.

default: `8.d1`

`qbfgs` parameters

qbfgsndim: (integer) Number of old forces and displacements vector used in the PULAY mixing of the residual vectors obtained on the basis of the inverse hessian matrix given by the BFGS algorithm. When bfgs_ndim = 1, the standard quasi-Newton BFGS method is used.

default: `1`

qbfgstri: (real) Initial ionic displacement in the structural relaxation.

default: `5.d-1`

qbfgstrmin: (real) Minimum ionic displacement in the structural relaxation. BFGS is reset when *trust_radius* < *trust_radius_min*.

default: `1.d-3`

qbfgstrmax: (real) Maximum ionic displacement in the structural relaxation.

default: `8.d-1`

qbfgsw1: (real) Parameter used in line search based on the Wolfe conditions.

default: `1.d-2`

qbfgsw2: (real) Parameter used in line search based on the Wolfe conditions.

default: `5.d-1`

5.1.6 Precursor Optimizer

The parameters available in this block are the same as those in *geopt*. The parameters specified here are used during the precursor optimiziation which precedes the main optimizer.

5.1.7 Dynamics

The details of molecular dynamics (MD) simulations are specified here. Not all keywords apply to every available method. Note that there is a fundamental difference in using dynamics to either sample free energies, or as a means to escape local minima in the minima hopping method. For the former, the goal is an accurate and unbiased sampling, while for the latter it is the efficient escape from a catchment basin of the PES. Hence, the available parameters and their suggested values differ significantly between these two classes of dynamics.

dynamics options

md_method: (string) Type of MD simulation ensemble.

default: `No default value.`

options:

`nvt_nose`: NVT ensemble with the Nose-Hoover chain thermostat.

`nvt_langev`: NVT ensemble with the Langevin thermostat.

`nve`: NVE ensemble (under development).

nmd: (integer) Number of MD steps.

default: `300`

nfreq: (integer) The frequency at which detailed output is written to disk.

default: `0`

dt: (real) Time step size. Units in fs.

default: `No default value.`

temp: (real) Target temperature. In units of Kelvin.

default: `0.d0`

init_temp: (real) Initial temperature. In units of Kelvin.

default: `0.d0`

highest_freq: (real) Coupling constant to the extended system. Units in Ha/THz^2.

default: `1.d1`

ntherm: (integer) Number of thermostats used for the Nose-Hoover thermostat chain.

default: `2`

restart: (logical) Activates a restart from a previously interrupted MD run.

default: `False`

Parameters specifically for the MD escape trials in task `minhocao`

algo: (integer) Choice of variable cell shape algorithms.

default: `1`

options:

`1`: Parrinello-Rahman

`2`: Cleveland

`3`: Wentzcovitch

`4`: Andersen (variable cell volume only, fixed cell shape)

integrator: (integer) Integrator for the variable cell shape MD equations of motion.

default: `3`

options:

`1`: Velocity Verlet

`2`: Classic Verlet

`3`: Beeman corrector-predictor scheme

presscomp: (real) Sets a bias pressure for MD runs in **task** `minhocao` simulations. The high temperatures in the `minhocao` cycles can lead to the cell expanding significantly beyond the equlibrium volume. To counteract this thermal expansing, a bias pressure can be imposed during the MD run. In units of GPa.

default: `0.d0`

cellmass: (real) Fictitious cell mass for all variable cell shape MD tasks. Arbitrary units.

default: `1.d0`

mdmin_init: (integer) The initial stopping criterion `mdmin` for an MD run in **task** `minhocao`. The MD run is terminated as soon as `mdmin` minima in the potential energy/enthalpy along the MD trajectory is encountered.

default: `2`

auto_mdmin: (logical) The stopping criterion for an MD run is adjusted dynamically if `True`. `mdmin` will fluctuate between **mdmin_min** and **mdmin_max**. Used for the MD escape steps in **task** `minhocao`.

default: `False`

mdmin_min: (integer) The minimal value of `mdmin`. Used for the MD escape steps in **task** `minhocao` and if **auto_mdmin** is `True`.

default: `2`

mdmin_max: (integer) The maximal value of `mdmin`. Used for the MD escape steps in **task** `minhocao` and if **auto_mdmin** is `True`.

default: `2`

auto_mddt: (logical) Activates the dynamical adjustment of the MD timestep `dt` in **task** `minhocao`. For clusters, `dt` is adjusted such that the total energy is conserved to within 1 %, and has to be accompanied by setting **encon** to `True`. For peridic systems, `dt` is adjusted such that the number of sampling points per oscillation in the energy/enthalpy along an MD trajectory is close to the target value **nit_per_min**.

default: `False`

encon: (logical) Activates the dynamical adjustment of the MD timestep `dt` in **task** `minhocao` for clusters based on energy conservation.

default: `False`

nit_per_min: (integer) Target number of MD samples per energy/enthalpy oscillation. Only used if **auto_mddt** is `True`.

default: `25`

dt_init: (real) Initial MD time step `dt`. In atomic units.

default: `2.d1`

5.1.8 Artificial Neural Network Potentials

FLAME implements a range of artificial neural network (ANN) potentials. In the original Behler-Parrinello approach, the output values of ANNs are considered as atomic energies which sum up to the total energy of a system. In the charge equilibration via neural network technique (CENT), the ANN serve as an intemediate step to include long-range electrostatic interactions. The parameters related to an ANN must be set in the block `ann`, as described below.

ann options

general `ann` parameters

subtask: (string) Sets the task to perform within the ANN schemes.

default: `No default value.`

options:

`train`: routines to perform an ANN training.

approach: (string) Determines the ANN technique to be employed.

default: `atombased`

options:

`atombased`: the original Behler-Parrinello method, where the total energy is the sum of atomic energies [29]

`cent1`: the charge equilibration via neural network technique [2]

`train` parameters

optimizer: (string) Method for the optimization during an ANN training process.

> default: `No default value.`
>
> options:
>
> > `rivals`: Currently the only available method, based on a modification of the extended Kalman filter.

nstep_opt: (integer) Number of extended Kalman filter steps.

> default: `100`

nconf_rmse: (integer) Number of configurations for which energy and force RMSE values will be calculated and reported in the output.

> default: `0`

ampl_rand: (real) The amplitude of random numbers used as the initial values of the ANN weights. Arbitrary units.

> default: `1.d0`

symfunc: (string) Determines how the evaluation of the symmetry functions is treated.

> default: `only_calculate`
>
> options:
>
> > `only_calculate`: values are stored in memory only
> >
> > `write`: values are written to disk
> >
> > `read`: values are read from disk from a previous run

`cent1` parameters

syslinsolver: (string) Determines what method to use to solve the system of linear equations for the charge equilibration process (CEP).

> default: `direct`
>
> options:
>
> > `direct`: direct approach, e.g., Gaussian elemination
> >
> > `operator`: iterative approach by convex optimization

nstep_cep: (integer) Maximum number of steps allowed in the self-consistent CEP required by the CENT approach.

> default: `200`

alphax_q: (real) Stepsize in the CEP optimization.

> default: `1.d0`

qgnrmtol: (real) Convergence criterion for the norm of the gradient in the CEP when **syslinsolver** is set to `operator`. Tight values, e.g. `1.d-5`, may be required for accurate forces. Units in Ha/Bohr.

> default: `5.d-4`

freeBC_direct: (logical) Selectively employs the `direct` method for the **syslinsolver** for cluster structures only during the CEP. This is important when the reference (training or validation) data set contains both periodic and molecular configurations.

default: `False`

Additional files required for ann: `train`

list_posinp_train.yaml: Contains a single keyword, **files**, with a list of configurations for training the ANN.

files: (list of strings) List of filenames of the atomic structure files in `yaml` format. For more information on YAML structure file format, see *yaml Format*.

list_posinp_valid.yaml: Contains a single keyword, **files**, with a list of configurations for validating the ANN.

files: (list of strings) List of filenames of the atomic structure files in `yaml` format. For more information on YAML structure file format, see *yaml Format*.

SE.ann.input.yaml: ANN parameter files for every chemical element `SE` in the system. They contain element-based parameters of the ANN potential as well as parameters of the symmetry functions. They contain two keys, `main` and `symfunc` (see below). All paramters are in atomic units.

`main` parameters

nodes: (list of integers) Determines the architecture of the ANN. For example, `[3, 5]` means that the ANN has two hidden layers (number of elements) with three and five nodes, respectively. Currently, architectures with only one or two hidden layers are implemented, where the latter is well tested and has been employed in several applications.

default: `No default value.`

rcut: (real) The cutoff radius used for the symmetry functions. Units in Bohr.

default: `No default value.`

ampl_chi: (real) Determines the amplitude of the hyperbolic tangent function used to map the value of the ANN output nodes to the atomic electronegativity. We recommend to employ `1.d0`, and using smaller values is strongly discouraged.

default: `No default value.`

prefactor_chi: (real) Determines the prefactor of the argument of the hyperbolic tangent function used to map the value of the ANN output nodes to the atomic electronegativity. We recommend using `1.d0`.

default: `No default value.`

ener_ref: (real) Determines the reference energy value. We recommend to set it to the energy of an isolated atom so that ANN trains indeed the formation energies.

default: `No default value.`

gausswidth: (real) Determines the width of the Gaussian function representing the atomic charge density in CENT. We recommend to try different values in the range between `1.d0` and `3.d0`, which correspond to most atomic radii. Units in Bohr.

default: `No default value.`

hardness: (real) Determines the atomic hardness by which to control how much charge, approximately, to expect to be collected by this type of atom. Similar to **gausswidth**, we recommend using a value in a physically meaningful range given in textbooks. Units in $\mathrm{Ha/Bohr}^3$.

default: `No default value.`

chi0: (real) Determines the offset of atomic electronegativity. Arbitrary units.

default: `No default value.`

method: (string) Determines the type of symmetry functions used as the atomic environment descriptor. Currently, only symmetry functions of type `behler` are implemented.

default: `No default value.`

options:

`behler`: For more details see Ref. [29]

`symfunc` parameters

Two types of symmetry functions, radial and angular, are implemented in FLAME. A complete description and comparison between these symmetry functions is given in Ref.:cite:*Behler2011*. In the parameter file, this information is provided line by line, in a specific format:

The radial symmetry function in FLAME is called *g02* and has two parameters, the exponent to control the broadness of the Gaussian function, and the offset that determines the center of the Gaussian function. Using a finite offset is not well tested and we recommend to set it to zero. A *g02* symmetry function is defined by the key *g02_* appended by a zero-padded enumeration, e.g. *g02_001*. The value of the key is exponent, offset, 0.0, 0.0, and atom type, all separated by spaces. The two zeros at position 2 and 3 are the lower and upper bounds of the symmetry function for all training data sets. The zeros cannot be omitted. The last item is the atomic species of the surrounding atom.

The angular symmetry function is of type *g05* and contains three parameters. The first parameter, the exponent of the Gaussian function, is similar to that of *g02*. The other two parameters are the prefactor of the cosine function and the value of the power. Similar to *g02*, the parameters must be followed by two zero. The last two entries per line indicate the two atomic species surrounding the center atom. An example of an ANN parameter file for type Na in sodium chloride system is given below:

```
main :
    nodes : [5 , 5]     #number of nodes at each hidden layer
    rcut           : 12.0
    ampl_chi       : 10.0
    prefactor_chi  : 0.1
    ener_ref       : -162.15529
    gausswidth     :  1.100
    hardness       :  0.110
    chi0           : -0.220
    qinit          : 1.0
    method         : behler

symfunc:
   g02_001 : 0.0010  0.0000   0.0  0.0   Cl
   g02_002 : 0.0010  0.0000   0.0  0.0   Na
   g02_003 : 0.0100  0.0000   0.0  0.0   Cl
   g02_004 : 0.0100  0.0000   0.0  0.0   Na
   g02_005 : 0.0200  0.0000   0.0  0.0   Cl
   g02_006 : 0.0200  0.0000   0.0  0.0   Na
   g02_007 : 0.0350  0.0000   0.0  0.0   Cl
   g02_008 : 0.0350  0.0000   0.0  0.0   Na
   g02_009 : 0.0600  0.0000   0.0  0.0   Cl
   g02_010 : 0.0600  0.0000   0.0  0.0   Na
   g02_011 : 0.1000  0.0000   0.0  0.0   Cl
   g02_012 : 0.1000  0.0000   0.0  0.0   Na
   g02_013 : 0.2000  0.0000   0.0  0.0   Cl
   g02_014 : 0.2000  0.0000   0.0  0.0   Na
   g02_015 : 0.4000  0.0000   0.0  0.0   Cl
   g02_016 : 0.4000  0.0000   0.0  0.0   Na
```

(continues on next page)

(continued from previous page)

```
    g05_001 : 0.0001  1.0000  1.0000  0.0  0.0   Cl    Cl
    g05_002 : 0.0001  1.0000  1.0000  0.0  0.0   Cl    Na
    g05_003 : 0.0001  1.0000  1.0000  0.0  0.0   Na    Na
    g05_004 : 0.0001  1.0000 -1.0000  0.0  0.0   Cl    Cl
    g05_005 : 0.0001  1.0000 -1.0000  0.0  0.0   Cl    Na
    g05_006 : 0.0001  1.0000 -1.0000  0.0  0.0   Na    Na
    g05_007 : 0.0001  2.0000  1.0000  0.0  0.0   Cl    Cl
    g05_008 : 0.0001  2.0000  1.0000  0.0  0.0   Cl    Na
    g05_009 : 0.0001  2.0000  1.0000  0.0  0.0   Na    Na
    g05_010 : 0.0001  2.0000 -1.0000  0.0  0.0   Cl    Cl
    g05_011 : 0.0001  2.0000 -1.0000  0.0  0.0   Cl    Na
    g05_012 : 0.0001  2.0000 -1.0000  0.0  0.0   Na    Na
    g05_013 : 0.0001  4.0000  1.0000  0.0  0.0   Cl    Cl
    g05_014 : 0.0001  4.0000  1.0000  0.0  0.0   Cl    Na
    g05_015 : 0.0001  4.0000  1.0000  0.0  0.0   Na    Na
    g05_016 : 0.0001  4.0000 -1.0000  0.0  0.0   Cl    Cl
    g05_017 : 0.0001  4.0000 -1.0000  0.0  0.0   Cl    Na
    g05_018 : 0.0001  4.0000 -1.0000  0.0  0.0   Na    Na
    g05_019 : 0.0080  1.0000  1.0000  0.0  0.0   Cl    Cl
    g05_020 : 0.0080  1.0000  1.0000  0.0  0.0   Cl    Na
    g05_021 : 0.0080  1.0000  1.0000  0.0  0.0   Na    Na
    g05_022 : 0.0080  1.0000 -1.0000  0.0  0.0   Cl    Cl
    g05_023 : 0.0080  1.0000 -1.0000  0.0  0.0   Cl    Na
    g05_024 : 0.0080  1.0000 -1.0000  0.0  0.0   Na    Na
    g05_025 : 0.0080  2.0000  1.0000  0.0  0.0   Cl    Cl
    g05_026 : 0.0080  2.0000  1.0000  0.0  0.0   Cl    Na
    g05_027 : 0.0080  2.0000  1.0000  0.0  0.0   Na    Na
    g05_028 : 0.0080  2.0000 -1.0000  0.0  0.0   Cl    Cl
    g05_029 : 0.0080  2.0000 -1.0000  0.0  0.0   Cl    Na
    g05_030 : 0.0080  2.0000 -1.0000  0.0  0.0   Na    Na
    g05_031 : 0.0080  4.0000  1.0000  0.0  0.0   Cl    Cl
    g05_032 : 0.0080  4.0000  1.0000  0.0  0.0   Cl    Na
    g05_033 : 0.0080  4.0000  1.0000  0.0  0.0   Na    Na
    g05_034 : 0.0080  4.0000 -1.0000  0.0  0.0   Cl    Cl
    g05_035 : 0.0080  4.0000 -1.0000  0.0  0.0   Cl    Na
    g05_036 : 0.0080  4.0000 -1.0000  0.0  0.0   Na    Na
    g05_037 : 0.0250  1.0000  1.0000  0.0  0.0   Cl    Cl
    g05_038 : 0.0250  1.0000  1.0000  0.0  0.0   Cl    Na
    g05_039 : 0.0250  1.0000  1.0000  0.0  0.0   Na    Na
    g05_040 : 0.0250  1.0000 -1.0000  0.0  0.0   Cl    Cl
    g05_041 : 0.0250  1.0000 -1.0000  0.0  0.0   Cl    Na
    g05_042 : 0.0250  1.0000 -1.0000  0.0  0.0   Na    Na
    g05_043 : 0.0250  2.0000  1.0000  0.0  0.0   Cl    Cl
    g05_044 : 0.0250  2.0000  1.0000  0.0  0.0   Cl    Na
    g05_045 : 0.0250  2.0000  1.0000  0.0  0.0   Na    Na
    g05_046 : 0.0250  2.0000 -1.0000  0.0  0.0   Cl    Cl
    g05_047 : 0.0250  2.0000 -1.0000  0.0  0.0   Cl    Na
    g05_048 : 0.0250  2.0000 -1.0000  0.0  0.0   Na    Na
    g05_049 : 0.0250  4.0000  1.0000  0.0  0.0   Cl    Cl
    g05_050 : 0.0250  4.0000  1.0000  0.0  0.0   Cl    Na
    g05_051 : 0.0250  4.0000  1.0000  0.0  0.0   Na    Na
    g05_052 : 0.0250  4.0000 -1.0000  0.0  0.0   Cl    Cl
    g05_053 : 0.0250  4.0000 -1.0000  0.0  0.0   Cl    Na
    g05_054 : 0.0250  4.0000 -1.0000  0.0  0.0   Na    Na
```

5.1.9 Singlepoint

The single point task performs a sole energy and force evaluation given a `posinp.yaml` atomic structure file without changing the atomic positions. This task is especially useful to establish a simple interface between an external sampling code and FLAME, which require energy/force evaluation given a atomic structure file. A prominent example for such an application is the computation of force sets of displaced supercell structures in phonon calculations, e.g., using Phonopy or ShengBTE. Further, this task allows to use a potential implemented in the FLAME code as a black-box engine, either by directly linking to the flame library, or by using the socket i-Pi interface [24].

`single_point` parameters

print_force: (logical) The forces will be written into a separate output file if set to `True`.

default: `False`

format: (string) Fortran format string used to print the forces.

default: `No default value.`

usesocket: (logical) Activates the communication scheme over unix or TCP sockets. The i-Pi protocol is used. Note: the server side of the socket communication has to be initialized first before FLAME can connect to the socket. A *posinp.yaml* must be provided that coincides in terms of composition and ordering of the elements with the system run at the i-Pi host.

default: `False`

sockinet: (integer) Selects Unix socket or internet (TCP) socket.

default: `0`

options:

`0`: Unix socket

`1`: internet (TCP) socket

sockport: (integer) Socket port number.

default: `0`

sockhost: (string) Socket address. If **sockinet** is `0`, a string with the **sockhost** name will be created in a temporary directory. Otherwise, a valid IP address must be provided (*127.0.0.1* for localhost).

default: `0`

5.1.10 Saddle

One and two-sided saddle point search methods are implemented in FLAME. Note that not all parameters apply to all available saddle point search methods when **task** is set to `saddle`.

saddle parameters

general `saddle` parameters

method: (string) Method for the saddle point search.

default: `No default value.`

options:

`bar_saddle`: The bar-saddle method from Ref. [8].

`dimer`: The dimer method from Ref. [30].

`splined_saddle`: The splined saddle method from Ref. [7].

`bar_saddle` parameters

dbar: (real) The length of the bar/dimer in the bar-saddle method. In units of Bohr in configurational space.

default: `1.d-1`

nstep: (integer) Maximal number of steps for the saddle search.

default: `1000`

stepsize: (real) The initial step size for the saddle point search iterations. Will be adjusted based on a a feedback mechanism. Arbitratry units.

default: `20.d0`

bar_contract: (logical) Activates the contraction of the bar size during the optimization. The final bar size can be set with **dbar_contracted**. The contraction occurs gradually over **nstep_contract** iterations.

default: `True`

dbar_contracted: (real) The final bar size after contraction. In units of Bohr in configurational space.

default: `1.d-3`

nstep_contract: (integer) Number of iterations to contract the bar from the initial value **dbar** to the final length **dbar_contracted**.

default: `40`

fnrmtol_coarse: (real) The force norm tolerance that has to be reached to commence the contraction of the bar. Units in Ha/Bohr.

default: `1.d-2`

fnrmtol_contracted: (real) The force norm tolerance for the convergence of the final, contracted bar. Units in Ha/Bohr.

default: `5.d-4`

`dimer` parameters

list_random_displace: (list of integers) List of atomic indices that are displaced randomly before initiating the saddle optimization in the dimer method.

default: `No default value.`

ampl: (real) The amplitude of the random displacement applied to the atoms listed by the key **list_random_displace**. In units of Bohr in configurational space.

default: `No default value.`

dimsep: (real) The dimer separation in the dimer method. No default value given, must be set when the dimer method is invoked. In units of Bohr in configurational space.

default: `No default value.`

`splined_saddle` parameters

np_splsad: (integer) Number of anchor points. **np_splsad**-1 is the number of moving anchor points in the splined saddle method. Note: with the default value of `2`, the splined saddle will only have one moving achor point.

default: `2`

np_neb: (integer) Number of NEB images. **np_neb**-1 is the number of moving images in the nudged elastic band (NEB) method. Note: with the default value of `2`, the NEB will run with only one moving image.

default: `2`

ns2: (integer) Number of extra points added to the number of anchor points in order to assist the maximization process, the inner loop of the splined saddle method.

default: `0`

vdtol: (real) The convergence criterion in the maximization process in the inner loop of the splined saddle method. Units in Ha.

default: `1.d-1`

htol: (real) The smallest value of the normalized pathway parameter between two neighboring points in the maximization process of the splined saddle method. Arbitrary units.

default: `2.d-2`

alphax: (real) The step size of the optimizer in the NEB and splined saddle methods. Arbitrary units.

default: `5.d-1`

docineb: (string) Activates the climbing image NEB.

default: `no`

options:

`no`: NEB without climbing image

`yes`: NEB with climbing image

doneb: (string) Activates NEB calculations.

default: `No default value.`

options:

`no`: No NEB calcualtion is performed

`yes`: NEB calculation is performed

pickbestanchorpoints: (string) Activates an automated selection of favored anchor points. This feature is not well tested and we recommend setting this parameter to `no`.

default: `No default value.`

options:

`no`: anchor points are distributed uniformly in the beginning of simulation

`yes`: anchor points are initially selected to favor higher energy points based on estimates obtained by an interpolation.

runstat: (string) Determines whether a new or a restart run is performed.

default: `new`

options:

`new`: New run, the NEB images or splined saddle anchor points are initialized at the beginning of the run.

`restart`: Restart run, the NEB images or splined saddle anchor points are read from a file. Restart runs have not yet been well tested, so we currently do not recommend using it.

typintpol: (string) The type of interpolation in the maximization process of the splined saddle method.

default: `cubic`

options:

`cubic`: Natural cubic splines.

`quintic`: A spline using fifth-order polynomials. Rather unstable except for simple pathways.

fcalls_max: (integer) The maximum number of calls to force evaluation.

default: `100`

fmaxtol_splsad: (real) The convergence criterion for the saddle optimization in the splined saddle method. Units in Ha/Bohr.

default: `2.d-4`

fmaxtol_neb: (real) The convergence criterion for the saddle optimization in the NEB method. Units in Ha/Bohr.

default: `2.d-2`

opt_method: The optimization method used in the saddle point search when using NEB or the splined saddle method.

default: `SD`

options:

`SD`: The steepest descent method.

`BFGS`: The Broyden–Fletcher–Goldfarb–Shanno (BFGS) method.

5.1.11 Optimizer Parameters for Saddle Point Searches

The parameters in this block are essentially identical to the ones for *geopt*. This block must be provided when using the splined saddle method specified by block *saddle*.

5.1.12 Minima Hopping for Crystal Structure Optimization

The minima hopping method implements a global optimization technique to identify the ground state structure of any chemical system.

Input/Output files

In addition to the standard *flame_in.yaml* file, further input files are required to start a MHM run. Also, important information will be written into output files other than *flame_log.yaml*, e.g., *earr.dat* and *global.mon*.

poscur.ascii/poscur.vasp: intitial or current configuration of a MHM run either in the *ascii* or *vasp* format, in Angstrom. It will be overwritten continuously as the simulation progresses with the current local minimum structure.

ioput: contains parameter values which are read and continuously updated throughout an MHM run. In one line, 3 real numbers must be provided, which correspond to *ediff* (in Ha/cell), *temperature* (in K and scaled based on the atomic masses `amass`), *maximum temperature* (in K and scaled based on the atomic masses `amass`).

> Suggested initial choice is `1.d-2 5.d2 1.d5` (for a system with around 16 atoms)

earr.dat: contains information about the initial/current state of the MHM run, and is populated with structural data during the simulation. The first two lines must be always provided:

- Line 1: *nmin_cur*, *nmin_max*
- Line 2: *delta_enthalpy*, *delta_fingerprint*

nmin_cur is the currently known number of local minima, and *nmin_max* is the maximal number of minima to be found in the next MHM run. *nmin_max* must always be larger than *nmin_cur*. Every new MHM run must start with *nmin_cur: 0* In a restart run, *nmin_cur* corresponds to the number of local minima found in the previous run(s).

delta_enthalpy (units of Ha/cell) and *delta_fingerprint* are the minimal enthalpies and structural fingerprint tolerances, respectively, below which two structures will be considered identical.

> Suggested initial choice: `2.d-3 1.d-1` (for a system with around 16 atoms and the Oganov fingerprint method)

During an MHM run, the value *nmin_cur* will incrementally increase, and addtional *nmin_cur* lines will be written to **earr.dat**, each corresponding to a *distinct* local minimum. This list of local minima is constantly sorted based on the enthalpy. Every line consists of 6 entries:

- Line i + 2: *imin*, *enthalpy*, *energy*, *nvisits*, *spg*, *spg_tol*

where *imin* is the index of the local minimum, *enthalpy* and *energy* are its enthalpy/energy (in Ha/cell), *nvisits* is the number of times it has been encountered during the run, *spg* is its space group index (for crystalline systems), and *spg_tol* is the tolerance used in SPGLIB to determine the space group.

poslowXXXXX.ascii/poslowXXXXX.vasp: The structures corresponding to the *nmin_cur* local minima are written into separate files, where *XXXXX* corresponds to the index *imin* in **earr.dat**.

`minhocao` options

nsoften: (integer) Number of softening iterations to eliminate the high-frequency modes from the initial MD velocities.

> default: `7`

alpha_at: (real) Stepsize for the softening algorithm on the atomic degrees of freedom. Arbitrary units.

> default: `5.d-1`

alpha_lat: (real) Stepsize for the softening algorithm on the lattice degrees of freedom. Arbitrary units.

> default: `5.d-1`

auto_soften: (logical) Choice of automatically adjusting the stepsize of the softening iterations based on gradient feedback.

> default: `True`

eref: (real) Reference energy, MHM run will stop as soon a minimum with an energy/enthlapy less than *eref* is found. In units of Ha/cell.

> default: `-1.d50`

5.2 Atomic Structure Files

FLAME can handle various atomic structure file formats. The preferred format is `yaml`, but other formats are supported as well. The subdirectory `utils/python/` contains scripts to convert structural data from a wealth of input formats to a `yaml` file.

5.2.1 `yaml` Format

Keyword-based structural information format. Must start with a block with keyword **conf**, followed by the following parameters.

bc: (string) Boundary condition.

  default: `No default value.`

  options:

    `free`: Free boundary conditions for molecules and clusters

    `slab`: Slab boundary conditions for surfaces, interfaces, and 2D materials

    `bulk`: Periodic boundary conditions for crystals and solids

cell: (list of three lists) Three cell vectors.

  default: `No default value.`

nat: (int) Number of atoms.

  default: `No default value.`

units_length: (string) Length units used in the structure.

  default: `atomic`

  options:

    `angstrom`: units in Angstrom

    `atomic`: units in Bohr

coord: (list of `nat` lists) For every atom in the system, a list of the form `[x, y, z, AT, CCC]` must be given. `x, y, z` are the cartesian coordinates of the atom, while `AT` is the symbol of the particular atomic type, and `CCC` determines if its `x, y, z` degree of freedom are allowed to change during the simulation.

E.g., `[0., 0., 0., Si, TTT]` is a silicon atom at the origin, free to move in all dimensions, while `[1., 1., 1., C, TFF]` is a carbon atom at `(1, 1, 1)`, only allowed to move along the `x` direction.

Example

Calcium fluoride in a crystal lattice.

```
conf:
  nat                                 : 12
  bc                                  : bulk
  units_length                        : angstrom
  cell:
  -  [5.462, 0.0, 0.0]
  -  [0.0, 5.462, 0.0]
  -  [0.0, 0.0, 5.462]
```

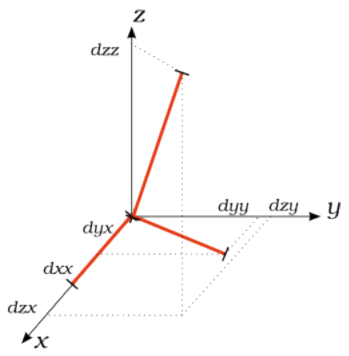
(continues on next page)

(continued from previous page)

```
coord:
-  [0.0, 0.0, 0.0, Ca, TTT]
-  [2.731, 2.731, 0.0, Ca, TTT]
-  [2.731, 0.0, 2.731, Ca, TTT]
-  [0.0, 2.731, 2.731, Ca, TTT]
-  [1.3655, 1.3655, 1.3655, F, TTT]
-  [4.0965, 1.3655, 1.3655, F, TTT]
-  [1.3655, 4.0965, 1.3655, F, TTT]
-  [4.0965, 4.0965, 1.3655, F, TTT]
-  [1.3655, 1.3655, 4.0965, F, TTT]
-  [4.0965, 1.3655, 4.0965, F, TTT]
-  [1.3655, 4.0965, 4.0965, F, TTT]
-  [4.0965, 4.0965, 4.0965, F, TTT]
```

5.2.2 `ascii` Format

The ascii format is described here in detail: http://inac.cea.fr/sp2m/L_Sim/V_Sim/sample.html#sample_ascii It is the native format of the **task** `minhocao` and the only format that currently supports the constraint of individual lattice parameters. Periodic cell vectors have to be provided to describe the simulation cell. The cell consisting of the cell vectors `a, b, c` has to be rotated such that `a` points along the `x` direction, and `b` lies within the `x-y` plane. See below figure for the orientation of the cell and the projections `dxx`, `dyx`, `dyy`, `dzx`, `dzy`, and `dzz`.



line 1: (integer) Number of atoms in the system, `Nat`.

line 2: (real, real, real) `dxx dyx dyy` values

line 3: (real, real, real) `dzx dzy dzz` values

lines 4 – n: Only lines starting with `#keyword:` are allowed and read/interpreted accordingly. The `#keyword:` tag must be followed by the following options.

> `reduced`: the atomic coordinates are treated as reduced coordinates, i.e., with respect to the cell vectors.
>
> `fixlat a b c alpha beta gamma vol`: specific components of the lattice vectors can be fixed. A `True` (or `T`) value fixes the degree of freedom, while `False` (or `F`) does not fix it. The last parameter (`vol`) is used for fixed-cell-shape-variable-volume simulations. I.e., `#keyword: fixlat F F T T T F F` allows the length of the `a` and `b` vectors to change, while keeping the length of the `c` vector constant. At the same time, only the angle between `a` and `b` is allowed to change. This setting is particularly useful for simulations of 2D materials or surfaces.

lines n+1 – Nat: `Nat` lines with the coordinates of every atom of the form `x, y, z, AT, c`. The `x, y, z` coordintes, followed by the chemical symbol `AT`, and optionally `f` for fixed atom. Note that the reduced coordinates will be fixed instead of the Cartesian one if the system is periodic

Example

Calcium fluoride in a crystal lattice, with selectively fixed lattice parameters. The Ca atoms are not allowed to move.

```
12
5.4620E+00   0.0000E+00   5.4620E+00
0.0000E+00   0.0000E+00   5.4620E+00
#keywords: fixlat F F T T T F F
0.0000E+00   0.0000E+00   0.0000E+00   Ca f
2.7310E+00   2.7310E+00   0.0000E+00   Ca f
2.7310E+00   0.0000E+00   2.7310E+00   Ca f
0.0000E+00   2.7310E+00   2.7310E+00   Ca f
1.3655E+00   1.3655E+00   1.3655E+00    F
4.0965E+00   1.3655E+00   1.3655E+00    F
1.3655E+00   4.0965E+00   1.3655E+00    F
4.0965E+00   4.0965E+00   1.3655E+00    F
1.3655E+00   1.3655E+00   4.0965E+00    F
4.0965E+00   1.3655E+00   4.0965E+00    F
1.3655E+00   4.0965E+00   4.0965E+00    F
4.0965E+00   4.0965E+00   4.0965E+00    F
```

BIBLIOGRAPHY

[1] M. Amsler, S. Rostami, H. Tahmasbi, E. Rahmatizad, S. Faraji, R. Rasoulkhani, and S. A. Ghasemi. FLAME: a library of atomistic modeling environments. *Comput. Phys. Commun.*, xxx(xxx):xxx, 2019.

[2] S. A. Ghasemi, A. Hofstetter, S. Saha, and S. Goedecker. Interatomic potentials for ionic systems with density functional accuracy based on charge densities obtained by a neural network. *Phys. Rev. B*, 92:045131, 2015. doi:10.1103/PhysRevB.92.045131.

[3] S. Goedecker. Minima hopping: an efficient search method for the global minimum of the potential energy surface of complex molecular systems. *J. Chem. Phys.*, 120:9911, 2004. doi:10.1063/1.1724816.

[4] M. Amsler and S. Goedecker. Crystal structure prediction using the minima hopping method. *J. Chem. Phys.*, 133:224104, 2010. doi:10.1063/1.3512900.

[5] M. Amsler. Minima Hopping Method for Predicting Complex Structures and Chemical Reaction Pathways. In Wanda Andreoni and Sidney Yip, editors, *Handbook of Materials Modeling: Applications: Current and Emerging Materials*, pages 1–20. Springer International Publishing, Cham, 2018. doi:10.1007/978-3-319-50257-1_77-1.

[6] B. Schaefer, S. A. Ghasemi, S. Roy, and S. Goedecker. Stabilized quasi-newton optimization of noisy potential energy surfaces. *J. Chem. Phys.*, 142(3):034112, 2015. doi:10.1063/1.4905665.

[7] S. A. Ghasemi and S. Goedecker. An enhanced splined saddle method. *J. Chem. Phys.*, 135:014108, 2011. doi:10.1063/1.3605539.

[8] B. Schaefer, S. Mohr, M. Amsler, and S. Goedecker. Minima hopping guided path search: an efficient method for finding complex chemical reaction pathways. *J. Chem. Phys.*, 140:9901, 2014. doi:10.1063/1.4878944.

[9] A. Sadeghi, S. A. Ghasemi, B Schaefer, S. Mohr, M. A. Lill, and S. Goedecker. Metrics for measuring distances in configuration spaces. *J. Chem. Phys.*, 139(18):184118, November 2013. doi:10.1063/1.4828704.

[10] L. Zhu, M. Amsler, T. Fuhrer, B. Schaefer, S. Faraji, S. Rostami, S. A. Ghasemi, A. Sadeghi, M. Grauzinyte, C. Wolverton, and S. Goedecker. A fingerprint based metric for measuring similarities of crystalline structures. *J. Chem. Phys.*, 144:034203, 2016. doi:10.1063/1.4940026.

[11] S. A. Ghasemi, A. Neelov, and S. Goedecker. A particle-particle, particle-density algorithm for the calculation of electrostatic interactions of particles with slablike geometry. *J. Chem. Phys*, 127:224102, 2007. doi:10.1063/1.2804382.

[12] S. Rostami, S. A. Ghasemi, and O. E. Nedaaee. A highly accurate and efficient algorithm for electrostatic interactions of charged particles confined by parallel metallic plates. *J. Chem. Phys.*, 145:124118, 2016. doi:10.1063/1.4963667.

[13] S. A. Ghasemi, M. Amsler, R. G. Hennig, S. Roy, S. Goedecker, T. J. Lenosky, C. J. Umrigar, L. Genovese, T. Morishita, and K. Nishio. Energy landscape of silicon systems and its description by force fields, tight binding schemes, density functional methods, and quantum monte carlo methods. *Phys. Rev. B*, 81:214107, Jun 2010. doi:10.1103/PhysRevB.81.214107.

[14] Tinker. https://dasher.wustl.edu/tinker. Online, accessed: 2019-09-30.

[15] LAMMPS. https://lammps.sandia.gov. Online, accessed: 2019-09-30.

[16] Mopac. http://openmopac.net. Online, accessed: 2019-09-30.

[17] DFTB+. http://www.dftbplus.org. Online, accessed: 2019-09-30.

[18] BigDFT. http://bigdft.org/. Online, accessed: 2019-09-30.

[19] VASP. https://www.vasp.at. Online, accessed: 2019-09-30.

[20] Abinit. https://www.abinit.org. Online, accessed: 2019-09-30.

[21] Quantum Espresso. https://www.quantum-espresso.org. Online, accessed: 2019-09-30.

[22] Siesta. https://departments.icmab.es/leem/siesta/. Online, accessed: 2019-09-30.

[23] CP2K. https://www.cp2k.org. Online, accessed: 2019-09-30.

[24] M. Ceriotti, J. More, and D. E. Manolopoulos. I-PI: A Python interface for ab initio path integral molecular dynamics simulations. *Comput. Phys. Commun.*, 185(3):1019–1026, 2014. doi:10.1016/j.cpc.2013.10.027.

[25] A. R. Oganov and M. Valle. How to quantify energy landscapes of solids. *J. Chem. Phys.*, 130(10):104504–9, 2009. doi:10.1063/1.3079326.

[26] Y. Wang, J. Lv, L. Zhu, and Y. Ma. CALYPSO: a method for crystal structure prediction. *Comput. Phys. Commun.*, 183(1):2063–2070, October 2012. doi:10.1016/j.cpc.2012.05.008.

[27] Nocedal's L-BFGS-B. http://users.iems.northwestern.edu/~nocedal/lbfgsb.html. Online, accessed: 2019-09-30.

[28] E. Bitzek, P. Koskinen, F. Gähler, M. Moseler, and P. Gumbsch. Structural relaxation made simple. *Phys. Rev. Lett.*, 97:170201, Oct 2006. doi:10.1103/PhysRevLett.97.170201.

[29] Jörg Behler. Atom-centered symmetry functions for constructing high-dimensional neural network potentials. *J. Chem. Phys.*, 134(7):074106, Feb 2011. doi:10.1063/1.3553717.

[30] G. Henkelman and H. Jónsson. A dimer method for finding saddle points on high dimensional potential surfaces using only first derivatives. *The Journal of Chemical Physics*, 111(15):7010–7022, 1999. doi:10.1063/1.480097.