MCMD

Generated by Doxygen 1.8.15

Chapter 1

MCMD

This piece of C++ code reads some dump files which are the output of an atomistic simulation, creates atomic environment boxes for each atom, and relaxes the structures with a hybrid Monte Carlo - Molecular Dynamics scheme. The relaxed structures are aimed to have minimum value for the maximum energy atom in the system with the ultimate goal of applying further Quantum Simulations on the output data as an integral part of Machine Learning potential development.

Chapter 2

Class Index

2.1 Class List

Here are the classes, structs, unions and interfaces with brief descriptions:

Chapter 3

File Index

3.1 File List

Here is a list of all documented files with brief descriptions:

Chapter 4

Class Documentation

4.1 Monte_Carlo Class Reference

Public Member Functions

- void Write_Lammps_In (int atom, int file, int atomrem, int flag, int count)
- int Atom_Remove (vector< double > &maxen, vector< int > &numofa)
- bool MC_Energy (vector< double > &lenx, double &mce)
- void MC_Perturbation (int atom, int file, int atomrem, int flag, int count, vector< double > &maxen, vector< int > &numofa)
- void Delete_Files (int atom, int file, int count, int minmc)
- void MC_Iteration (int atom, int file, int atomrem, vector< double > &lx, int &count, vector< double > &maxen, vector< int > &numofa, double r_cut, int check)

Public Attributes

- int count

  Counter for the MC perturbation.
- int atomrem

  Atom ID to be removed in a MC perturbation.
- int check

  Boolen value to accept/reject the perturbation.
- int flag

  Flag to stop the MC perturbations.
- double mce

  Per-atom energy.
- double minmc

  The MC perturbation step with the minimum energy value of maximum-energy-atom.

4.1.1 Member Function Documentation

4.1.1.1 Atom_Remove()

```
int Monte_Carlo::Atom_Remove (
            vector< double > & maxen,
            vector< int > & numofa )
```

/ Find the atom with maximum energy to remove.

4.1.1.2 Delete_Files()

```
void Monte_Carlo::Delete_Files (
            int atom,
            int file,
            int count,
            int minmc )
```

/ Delete the .dat files that are not needed.

4.1.1.3 MC_Energy()

```
bool Monte_Carlo::MC_Energy (
            vector< double > & lenx,
            double & mce )
```

/ Read the file including average atomic potential energy and the length of the simulation box and determine the acceptance/removal with a MC formulation.

4.1.1.4 MC_Iteration()

```
void Monte_Carlo::MC_Iteration (
            int atom,
            int file,
            int atomrem,
            vector< double > & lx,
            int & count,
            vector< double > & maxen,
            vector< int > & numofa,
            double r_cut,
            int check )
```

/ Iterative Monte-Carlo moves with atom removal/acceptance flag.

4.1.1.5 MC_Perturbation()

```
void Monte_Carlo::MC_Perturbation (
            int atom,
            int file,
            int atomrem,
            int flag,
            int count,
            vector< double > & maxen,
            vector< int > & numofa )
```

/ Single Monte-Carlo move.

1) Write LAMMPS in-script

2) Call LAMMPS

3) Find atom to remove

4) Write LAMMPS in-script

5) Call LAMMPS

4.1.1.6 Write_Lammps_In()

```
void Monte_Carlo::Write_Lammps_In (
            int atom,
            int file,
            int atomrem,
            int flag,
            int count )
```

/ Write LAMMPS input script depending on the MC flag that is to accept/reject the atom removal.

The documentation for this class was generated from the following files:

- monte_carlo.h
- monte_carlo.cpp

4.2 Sim_Box Class Reference

Public Member Functions

- void set_atm_coors (mat Coors)
- void set_box_props (vec bp, mat bv)
- vec calc_num_imgs (double r_cut)
- mat create_img_coors (double r_cut)
- int calc_nproj (vec vec1, vec vec2, vec vec3, double r_cut)

Public Attributes

- mat atm_coors
  Atom coordinates.
- int num_atoms
  Number of atoms.
- vec box_period
  The periodicity flags for each direction.
- mat box_vecs
  The supercell vectors.

4.2.1 Member Function Documentation

4.2.1.1 calc_nproj()

```
int Sim_Box::calc_nproj (
            vec vec1,
            vec vec2,
            vec vec3,
            double r_cut )
```

/ Calculate the number of box-images required.

4.2.1.2 calc_num_imgs()

```
vec Sim_Box::calc_num_imgs (
            double r_cut )
```

/ Calculate the number of minimum number of images along box dimensions.

4.2.1.3 create_img_coors()

```
mat Sim_Box::create_img_coors (
            double r_cut )
```

/ Create Images (3D, 2D or 1D depends on the values of nx, ny, nz).

4.2.1.4 set_atm_coors()

```
void Sim_Box::set_atm_coors (
            mat Coors )
```

/ Get coordinates of the atoms in the simulation box.

4.2.1.5 set_box_props()

```
void Sim_Box::set_box_props (
            vec bp,
            mat bv )
```

/ Set box periodicity and box vectors (origin is (0,0,0))

The documentation for this class was generated from the following files:

- sim_box.h
- sim_box.cpp

Chapter 5

File Documentation

5.1 mlpack_util.h File Reference

Neighbor atom calculations for the atomic environment.

Functions

- void Find_Neighbors_mlpack (mat Img_Coors, uvec atm_inds, double r_cut, vector< vector< size_t > > &resultingNeighbors, vector< vector< double > > &resultingDistances)
- mat Find_Neighbors_mlpack (mat Img_Coors, rowvec pt, double r_cut)

5.1.1 Detailed Description

Neighbor atom calculations for the atomic environment.

These functions find the neighbor atoms and the corresponding coordinates for an atomic environment using RangeSearch function of MLpack library.

5.1.2 Function Documentation

5.1.2.1 Find_Neighbors_mlpack() [1/2]

```
void Find_Neighbors_mlpack (
            mat Img_Coors,
            uvec atm_inds,
            double r_cut,
            vector< vector< size_t > > & resultingNeighbors,
            vector< vector< double > > & resultingDistances )
```

/ Find all the neighbors for the atomic environment.

5.1.2.2 Find_Neighbors_mlpack() [2/2]

```
mat Find_Neighbors_mlpack (
             mat Img_Coors,
             rowvec pt,
             double r_cut )
```

/ Find the atomic environment coordinates.

5.2 monte_carlo.h File Reference

Class for executing hybrid Monte Carlo-Molecular Dynamics relaxation.

Classes

- class Monte_Carlo

5.2.1 Detailed Description

Class for executing hybrid Monte Carlo-Molecular Dynamics relaxation.

This class executes the Monte-Carlo algorithm for relaxing the given atomic structure by removing the highest energy atoms. It writes input scripts for LAMMPS and runs it, and decides on atom removal in a stochastic-iterative manner until the system is relaxed.

5.3 sim_box.h File Reference

Class for calculating the box geometry and image coordinates.

Classes

- class Sim_Box

5.3.1 Detailed Description

Class for calculating the box geometry and image coordinates.

This class reads the box properties from given dump files and and calculates the resulting image coordinates for the atomic environment calculations.

5.4 test.cpp File Reference

Main function for executing the code.

Functions

- int main ()

5.4.1 Detailed Description

Main function for executing the code.

Hybrid Monte Carlo-Molecular Dynamics relaxation scheme is executed in this function by calling all the other member (in classes) or non-member functions with the steps given below.

5.4.2 Function Documentation

5.4.2.1 main()

```
int main ( )
```

1) Read the dump files in the directory and store them in a string-vector.

3) Read the dump file and store the data.

4) Create the image coordinates.

5) Calculate the neighbor IDs and distances for the atomic environment of each atom.

6) Find the atomic environment coordinates for each atom.

7) Write the initial data file.

8) Start with the initial MC move.

9) Boolen check for accept/reject the move and increase the MC counter.

10) Execute the MC iterations until it converges.

11) Find the MC move number for the minimum energy configuration among all the moves.

12) Delete the unused .dat files that are used during the relaxation.

13) Write the .cfg file for MLIP in the desired file structure.

5.5 util_functions.h File Reference

Other functions.

Functions

- void read_dump_file (string fname, int &num_atoms, int &num_attr, vec &box_period, mat &box_vecs, vector< string > &attribute_labels, mat &atm_coor, mat &Atm_attr, uvec &atm_ids)
- mat Find_Box_Neighbors (mat nn_coors, rowvec pt, double r_cut)
- vector< string > **split** (string str, char delimiter)
- vector< string > **globVector** (const string &pattern)
- mat Add_ID_Type (mat Box_Coors, double r_cut)
- void Write_Data_File (mat Box_Coors, double r_cut, int atom, int file, string df_name)
- void Write_ML (int atom, int file, int minmc, string fdel)

5.5.1 Detailed Description

Other functions.

This header consists of functions with different purposes as will be mentioned below.

5.5.2 Function Documentation

5.5.2.1 Add_ID_Type()

```
mat Add_ID_Type (
             mat Box_Coors,
             double r_cut )
```

/ Determines the inside (rigid) and outside atoms for the relaxation. / Eliminates the atoms which are very close to box boundary to avoid overlapping. / Prepares the atom IDs, types and coordinates for the .dat file

5.5.2.2 Find_Box_Neighbors()

```
mat Find_Box_Neighbors (
             mat nn_coors,
             rowvec pt,
             double r_cut )
```

/ Calculate the atomic environment for a given atom.

5.5.2.3 read_dump_file()

```
void read_dump_file (
             string fname,
             int & num_atoms,
             int & num_attr,
             vec & box_period,
             mat & box_vecs,
             vector< string > & attribute_labels,
             mat & atm_coor,
             mat & Atm_attr,
             uvec & atm_ids )
```

/ Read the dump file for all the atom attributes and the coordinates.

5.5.2.4 Write_Data_File()

```
void Write_Data_File (
             mat Box_Coors,
             double r_cut,
             int atom,
             int file,
             string df_name )
```

/ Write the data file of atomic environment for LAMMPS. Add atom IDs and types to the data, and trim the boundary atoms at the edges

5.5.2.5 Write_ML()

```
void Write_ML (
             int atom,
             int file,
             int minmc,
             string fdel )
```

/ Write the resulting (relaxed) configurations in a .cfg file for MLIP.

1) Take the selected .dat file and write the configurations, energy and stress in a new dump file using LAM↩ MPS.

2) Read the data from that dump file.

3) Calculate the new supercell vectors.

4) Shift the xyz coordinates with respect to the supercell vectors defined

5) Read the energy and stress for the cfg. file.

6) Write the .cfg file.