**­­­­Limitations/To do:**

* Overbonding won’t work right for non-H2O and when multiple “to” overbonding atom types are requested
* MD code functionality when parameter files has USEPOVER true but FITPOVER false is not working. This needs to be fixed.

**Compiling/running the code:**

The md code now supports both serial and MPI functionality. To switch between the two, change the “CXX” command in the Makefile. Refer to the examples therein.

* The lsq and md codes both share source files, and have several compilation options.
* Before compiling, navigate to the src directory
* Makefile options include the following tabulated values. Note that “xx” stands for either “lsq” or “md.” The make command is executed as make <options>. The resulting executables are of the form house\_xx.

|  |  |
| --- | --- |
| house\_xx | Compile the xx code. |
| clean\_xx | Remove .o files related to xx code. Remember, some .o files are shared. |
| realclean\_xx | Remove xx code executable. |
| build\_tests\_xx | Generate fresh “correct” files for test suite . |
| travis | Compile Travis trajectory analyzer. See the .pdf file in contrib/Travis for more information. |
| packmol | Compile packmol initial configuration generator. See the packmol website for more information |

* In serial mode, the lsq and md codes can be run with:   
  /path/to/executable < input\_file.name
* In MPI mode, the md code can be run with:   
  srun –n x /path/to/executable < input\_file.name  
  on LLNL HPC (using slurm, and submitted to the queue). Note that the “mpiicc” option should be used for this compilation  
  mpirun –np x –stdin all /path/to/executable < input\_file.name  
  on machines that do not have slurm. The “-stdin all” flag gives all processors access to stdin. For this application, the “mpicxx” option is recommended.

Note that a few special compiler options can be specified for the lsq and md codes. For example, by adding “-D VERBOSITY=1,” to the “MYFLAGS” argument list given in the Makefile, greater detail will be printed to output files. Including “-D FORCECHECK=1” will print out the “3b\_results.dat” file at each time step.

**Running the test suites/generating “correct” test suite reference files**

* Note that the suite is intended to use lsq-new-md-fmt.py version.

Immediately after pulling code (i.e. before making any modifications):

* Run from test\_suite-xx with ./generate\_test\_suite.sh

Once edits have been made:

* Run from test\_suite-xx with ./run\_test\_suite.sh

**Notes for Reading the Makefile**

**Flags and options:**

* PESFORMAT: (Currently, only gnuplot style is supported)  
  1 gnuplot style  
  2 matlab style  
  3 python style
* FORCECHECK:  
  1 Print out 3b forces on each atom at each time step (generally for debugging)  
  0 Don't print out the aforementioned (default)
* VERBOSITY:  
  1 Verbose (default)  
  0 Not verbose (not recommended)
* FPENALTY\_POWER  
  (Any double) The power on the fpenalty function – 3.0 is typically default
* CHECK\_CHEBY\_RANGE (i.e. check for proper behavior of transformed distance at end of cheby interval)  
  0 Do not check  
  1 Check (default)

**Compiler options**

|  |  |
| --- | --- |
| CXX=mpiicc -g -D USE\_MPI | Use for LLNL HPC; SLURM compatible; MPI |
| CXX=mpicc -g -D USE\_MPI | Use for GNU MPI |
| CXX=mpicxx -g -D USE\_MPI | Use for Debian (OSX) MPI |
| CXX=g++ -g | Serial with debugging information |
| CXX=g++ -O3 | Serial, extreme optimizations |
| CXX=g++ -g -Wall -Wextra -Wuninitialized -O1 | Serial Specialty debugging |
| CXX=g++ -g # -Wall –Wextra | Serial Specialty debugging |
| CXX=icpc -fast -Wall | Serial Specialty debugging |

**Makefile 101**

|  |  |
| --- | --- |
| “Target” | The name of a compiled object (i.e. the "\*" of "\*.o", or name of an exectuable) |
| “Rule” | How to compile something, for example, g++ -o my\_exec file1.cpp file2.cpp is a rule |
| “Prerequisite” | The files that are required to make a target (the "file1.cpp and file2.cpp from above) |
| $@ | Name of target of a rule |
| $< | Name of the first prerequisite of a rule |
| $^ | Name of all prerequisites of a rule, with spaces between them |
| -c | Compile file directly into an object... i.e. g++ -c myfile.cpp produces myfile.o |

**Quick Navigation:**

**LSQ C++ Code**

* [Main Control Variables](#_LSQ_CODE:_Main)
* [Topology Variables](#_LSQ_CODE:_Topology)
* [Extras](#_Extras)

**LSQ Python Codes**

* [Original script](#_LSQ_Python_Codes_7)
* [Original MKL script](#_LSQ_Python_Codes_6)
* [New single-threaded script](#_LSQ_Python_Codes_5)
* [New MKL script](#_LSQ_Python_Codes_4)

**MD Code**

* [Main Input File](#_MD_CODE:_Main)
* [Parameter file](#_MD_CODE:_Parameter) (This section of the manual is under construction)

**Utilities**

* [PES Scan generator](#_PES_Scan_generator)
* [Small utilities](#_Small_Utilities)

**Advanced Functionality**

* [Self-consistent Force Field Fitting](#_Self-consistent_force_field)
* [Fit Refitting/Extrapolation](#_Fit_refitting/extrapolation)

**Digressions**

* [Determination of “allowed” 3-body Power Sets](allowed#_Determination_of_)
* [Notes on Units](#_Important_note_on)

# Determination of “Allowed” 3-body Power Sets

Determination of the allowed powers and force field types for 3-body interactions are determined through the following process:

1. Use combinatorics to determine the number of possible unique atom triplets that can be constructed from a given set of atoms
2. Build a list of all possible non-unique combinations of atom pairs (i.e. list may have an entry for {OO,OH,OH}, {OH,OO,OH}, and {OH,OH, OO})
3. Affiliate each listed combination from above (2.) with a specifically-ordered name (i.e. {OO,OH,OH}, {OH,OO,OH}, and {OH,OH, OO} will all be associated with the name “OOOHOH”)
4. Begin determination of allowed powers for each unique triplet found in step 3, according to the following rules:
   1. Powers start from zero, so if order is specified to be 2, polynomial powers range from 0 to n-1, NOT 1 to n
   2. At least two pairs must have non-zero powers for the interaction to truly correspond to 3-body interactions
   3. Non-uniqueness upon power sorting must be taken into consideration. For example, for the type “OOOHOH”, powers of (1,1,0) are identical to (1,0,1)

The process for achieving item 4c from above:

1. For each triplet of atoms, determine all possible sets of 3B powers, by simply running a triple loop over the requested 3B order
2. Re-order each set of powers to reflect the nature of the 3 atom pairs describing the triplet. For example, if all three pairs are unique, powers do not need to be re-ordered. If two pairs are identical, re-order powers as if the pair types were arranged A != B == C. Otherwise, if all 3 pairs are identical, sort each set of powers in ascending order.
3. If this is the first set of powers, use as the first entry in a new list of “allowed” powers. Otherwise, only append to the list if the current set of powers has not already been added.
4. Finally, set a force field index for each set of allowed parameters, and associated it with the character map for the 3b type (i.e. “OOOHOH”).

# Important Note on Units:

* For all pair types *except DFTBPOLY*, the LSQ C++ code expects the .xyzf file to have **coordinates in Angstr**, but **forces in Hartree/Bohr**
* For all pair types *except DFTBPOLY*, the LSQ C++/Python codes output forces/potential parameters in terms of kcal/mol/Angstr and Kcal/mol

# LSQ CODE: Main Control Variables

|  |  |  |
| --- | --- | --- |
| Keyword | Allowed Values | Usage |
| # TRJFILE # | any string | Provides the name of the simulation trajectory file. Files use a .xyzf format, which is like the standard .xyz format, with two exceptions: (1) after the line containing the number of atoms, the x, y, and z box lengths are given, and (2) each coordinate line has x, y, and z forces on the corresponding atom appended. |
| # WRAPTRJ # | “true” or “false” | Defines whether coordinates should be wrapped (i.e. application of post-simulation periodic boundary conditions). |
| # NFRAMES # | Any integer > 0 | Number of frames in the .xyzf file |
| # NLAYERS # | Any integer > 0 | Number of supercells to create from simulation box (i.e. replicate images). |
| # FITCOUL # | “true” or “false” | Defines whether charges should be fit, or held fixed at user-defined values. Note that currently, functionality is only supported when # FITCOUL # is true, or when  # FITCOUL # is false and all charges are zero.  If # FITCOUL # is false, but charges are non-zero, program will attempt to subtract charge contributions from forces. |
| # FITPOVR # | “true” or “false” | Defines whether a ReaxFF type over-coordination force should be applied. Note that currently, functionality is only supported when # FITPOVR # is true, or when  # FITPOVR # is false and all # USEOVRP # are set to false.  If # FITPOVR # is set to false and at least one # USEOVRP # is set to true, program will attempt to subtract over-coordination contributions from forces. |
| # PAIRTYP # | “SPLINE”, “CHEBYSHEV”, “DFTBPOLY”, or “INVRSE\_R” | The form of the inter-molecular potential to be used for fitting.  If the # PAIRTYP # is **DFTBPOLY**, an integer must also be given to indicate the polynomial order. A value of n corresponds to monomials that range from order 1 to n.  If the # PAIRTYP # is **CHEBYSHEV**, two integers must be provided, where the first provides the order for 2-body interactions, and follows the same convention as for DFTBPOLY types, while the second integer provides the order for 3-body interactions, and a value of n indicates monomials ranging in order from 0 to n-1. Two additional numbers can be added, which specify the range for the Chebyshev distance transformation. If no values are specified, the program assumes -1 and 1. Values must fall between -1 and 1.  If the # PAIRTYP # is **INVRSE\_R**, an integer must also be given to indicate the number of terms. |
| # CHBTYPE # | “DEFAULT”, “INVRSE\_R”, “MORSE” | Determines how distance is transformed for a Chebyshev-type fit. Default means no transformation will be applied. Only used if # PAIRTYP # is CHEBYSHEV. |

**­­­­**

# LSQ CODE: Topology Variables

|  |  |  |
| --- | --- | --- |
| Keyword | Allowed Values | Usage |
| # NATMTYP # | Any integer > 0 | Number of unique atom types in trajectory |
| # TYPEIDX # | Integers > 0 | Ascending integers, ranging from 1 to # NATMTYP#. Index for each unique atom type. Values form a column below  # TYPEIDX #. |
| # ATM\_TYP # | Any chemical symbol  (i.e. C or Si) | Chemical symbol for each unique atom type. Values form a column below # ATM\_TYP #. |
| # ATMCHRG # | Any float OR “+” or “-“ | If # FITCOUL # is false: user-defined partial atomic charge for each unique atom type. If # FITCOUL # is true: either a positive or negative sign, to indicate how pair charge signs should be assigned. |
| # ATMMASS # | Any float > 0 | User-defined atomic mass for each unique atom type. Values form a column below # ATMMASS #. Only used to set masses for MD in the resulting parameter file. |
| # PAIRIDX # | Integers > 0 | Ascending integers, ranging from 1 to the total number of unique atom pair types. Values form a column below  # PAIRIDX #. |
| # ATM\_TYX # | Any chemical symbol  (i.e. C or Si) | # ATM\_TY1 # or # ATM\_TY2 #. Used to define interaction pair types. Order does not matter. |
| # S\_MINIM # | Any float > 0 | Inner cutoff for pair potential in Angstroms. |
| # S\_MAXIM # | Any float > 0 | Outer cutoff for pair potential in Angstroms. |
| # S\_DELTA # | Any float > 0 | Grid spacing for splines. Number of splines for a given pair type is (# S\_MAXIM # – # S\_MINIM #)/# S\_DELTA #. Only used if # PAIRTYP # is SPLINE, but regardless, something (such as “NA”) still needs to be entered in this field. |
| # MORSE\_LAMBDA # | Any float > 0 | Morse-type lambda to use when a Morse transformation is used on pair distance for a CHEBYSHEV type pair. Only used if # PAIRTYP # is CHEBYSHEV. |
| # USEOVRP # | “true” or “false” | Determines whether overbonding should be included for pair type. |
| # TO\_ATOM # | Any chemical symbol  (i.e. C or Si) | The atom that overbonding is considered for. For example, “O” for water-type systems. Only used if # USEOVRP # is true. |
| # P\_OVERB # | Any float > 0 | *p*over from the ReaxFF overbonding equation of order 1. Only used if # USEOVRP # is true. |
| # R\_0\_VAL # | Any float > 0 | *r*0 from the ReaxFF overbonding equation of order 1. Only used if # USEOVRP # is true. |
| # P\_1\_VAL # | Any float > 0 | *p*bX,1 from the ReaxFF overbonding equation of order 1. “X” refers to the “TO\_ATOM.” Only used if # USEOVRP # is true. |
| # P\_2\_VAL # | Any float > 0 | *p*bX,2 from the ReaxFF overbonding equation of order 1. “X” refers to the “TO\_ATOM.” Only used if # USEOVRP # is true. |
| # LAMBDA6 # | Any float > 0 | *λ*6 from the ReaxFF overbonding equation of order 1. Only used if # USEOVRP # is true. |

# 

# Extras

**Modifying the Penalty Function**

# A cubic penalty function, “fpenalty” is included in most potential fits which discourages close distances according to: fpenalty = cubic\_scaling(1-rlen/rmax)^3. By default, the value of cubic\_scaling is set to 1, however the user can specify a different value by adding a line before the # ENDFILE # tag in the input file:

PAIR CHEBYSHEV CUBIC SCALING: 0.4

# ENDFILE #

**Adding Charge Constraints**

If the user desires to fit charges during the force matching process, they can either do so with no constraints (the default option), or by specifying n\_atom\_pairs -1 constraints. These constraints are added to the end of the input file, as discussed in the “Modifying the Penalty Function” section above. Take water, as an example; here we have 3 pair types, OO, OH, and HH. We want to enforce that the sum of all charges is zero, and that H has half the charge of O, and we can do so by adding the following lines:

CHARGE CONSTRAINTS:

OO HH OH 1000.0 -4000.0 0.0 0.0

OO HH OH 1000.0 4000.0 4000.0 0.0

The first line can be re-written as the equation: 1000\*qOqO - 4000\*qHqH = 0, and enforces the relationship that |qO| = 2|qH|.

The second line can be re-written as the equation: 1000\*qOqO + 4000\*qHqH + 4000\*qOqH = 0, and enforces that the sign of qOH needs to be opposite of qHH and qOH (i.e. negative), and the relationship |qOH| = 2|qHH|. Note that each line needs entries for all atom pairs.

**Separating 2-body and 3-body outer cutoffs**

Currently, 3-body outer cutoffs are taken to be equivalent to the constituent 2-body outer cutoffs, by default. One has the option of setting all outer cutoffs to an equivalent value by adding the following line to the end of the fm\_setup,in file:

SPECIAL 3B S\_MAXIM: ALL 4.0

Otherwise, if one wants to specify each cutoff separately, syntax similar to the following should be used:

SPECIAL 3B S\_MAXIM: SPECIFIC 3

OOOOOO 4.0

OOOHOH 3.2

HHHHHH 2.5

Where the “3” is the number of cutoffs to be listed. Any 3-body type for which a line is not provided will use the same S\_MAXIM as the 2-body interactions, where constituent pairs determine the cutoff.

# LSQ Python Codes

|  |  |  |
| --- | --- | --- |
| Name | Usage | Notes |
| lsq.py | python lsq.py A.txt b.txt params.header  > params.txt | Produce output compatible with old version of MD code. **Ancillary support.** |
| lsq-new-md-fmt.py | python lsq-new-md-fmt.py lsq.py A.txt b.txt params.header ff\_groups.map > params.txt | Produce output compatible with new version of MD code. |
| lsq-new-md-fmt-weighted.py | python lsq-new-md-fmt.py lsq.py A.txt b.txt params.header ff\_groups.map WEIGHTFILE my\_weightfile.dat > params.txt | Located in the contrib/hist\_forces directory. Allows weights to be specified for forces. The input weight file should have one line for each in b.txt that specifies the weight to be applied. **Ancillary support.** |

# MD CODE: Main Input File

|  |  |  |
| --- | --- | --- |
| Keyword | Allowed Values | Usage |
| # PLOTPES # | See PES section | If true, this is the only keyword needed in the file. Defaults to false. |
| # RNDSEED # | Any integer > 0 | Seed for random number generator. |
| # TEMPERA # | Any float > 0 | Target simulation temperature, in Kelvin. |
| # CONVCUT # | float between 0 and 1 | Maximum allowed fraction deviation from input temperature. Defaults to 0.10 (10%). |
| # CMPRFRC # | “true <file name>”  or “false” | If true, forces are computed for a single time step and compared to the forces in the given file. |
| # SLFCNST # | “true” or “false” | If true, prints a VASP POSCAR file – Assumes a cell with all angles orthogonal. Defaults to false. |
| # TIMESTP # | Any float > 0 | MD time step in femtoseconds (fs). |
| # N\_MDSTP # | Any integer > 0 | Number of MD steps. |
| # NLAYERS # | Any integer > 0 | Number of supercells to create (i.e. replicate images). |
| # PRMFILE # | A file name | Name of the input parameter file. |
| # CRDFILE # | A file name | Name of file containing initial coordinates. Should be a .xyz or .xyzf file where the comment line specifies box dimensions, and each coordinate line optionally has the has the x, y, and z velocity or force components listed. |
| # VELINIT # | “true” or “false” | Initialize velocities using Box Muller, or read from coordinate file? |
| # THERMOST # | “HOOVER <Hoover time>” or “VELSCALE <scaling freq>” | Thermostat using Nose-Hoover approach, or velocity scaling. |
| # PRESSUR # | “ANALYTICAL” or “NUMERICAL” | Compute pressures analytically or numerically? |
| # WRPCRDS # | true or false | Apply PBC to coordinates? If not specified, simulation defaults to true. Use false if post processing for properties such as mean squared displacements. Defaults to true. |
| # FRQDFTB # | Any integer > 0 | How frequently should the DFTB .gen file be printed? **Note: All other “FRQ” and “PRNT” type variables must be specified below this one.** |
| # FRQENER # | Any integer > 0 | How frequently should the energies, etc be generated? |
| # PRNTVEL # | “false” or  “true” and  “FRQDFTB” or an integer > 0 | Should the velocities be printed durng the simulation? If so, specify the frequency for output. “FRQDFTB” means to print with the same frequency as the .gen file. Defaults to false. |
| # PRNTFRC # | “false” or  “true” and  “FRQDFTB” or an integer > 0 | Should the forces be printed during the simulation? Same as for # PRNTVEL # |

# 

# MD CODE: Parameter File

Note: This file is automatically generated by the LSQ C++ and python scripts. Parameter files can be generated by hand as well, but formatting is very important. If building by hand, it is easiest to start from a reference file, like those found in the test suites.

Features:

* In general, vertical spacing should NOT be modified
* In general, horizontal spacing should NOT be modified when in line with heading text
* In general, section ordering should NOT be modified. Exceptions are:
  + Lines with atom types info can be in any order
  + Lines with pair type info can be in any order
  + Note that subsequent ordering of related sections should be consistent, as these sections are used to determine type indices. For example, pair and triplet maps relate parameter type index to the pair/triplet type
* The heading section, prepended by exclamation points can contain whatever text the user wants. It is ignored by the MD code.
* Ordering of the Boolean variables in the next section is very important, as is the formatting/casing
* The PAIRTYPE line uses the same conventions as in the LSQ CODE: Main Control Variables section
* Charges are given in e, atom masses in amu

Special Controls:

A penalty function is added to the molecular mechanics force field to manage cases where a distance may fall below the rmin cutoff, and has the functional form: Penalty = penalty\_scaling(smin + penalty\_dist – rlen)^3, where parameters penalty scaling and penalty\_dist can be user-specified by adding the following lines above the “ATOM PAIR TRIPLETS” line, for example:

!# PAIRIDX # # ATM\_TY1 # # ATM\_TY1 # # S\_MINIM # # S\_MAXIM # . . .

0 C C 1 4.8 . . .

PAIR CHEBYSHEV PENALTY DIST: 0.05

PAIR CHEBYSHEV PENALTY SCALING: 2E8

PAIR CHEBYSHEV CUBIC SCALING: 1.0

ATOM PAIR TRIPLETS: 0

The “PAIR CHEBYSHEV CUBIC SCALING:” also controls a penalty function (“fpenalty” in the code), which helps to discourage small pair distances. See the LSQ C++ section for more details.

# PES Scan generator

This utility, which is built into the house\_md generates potential energy surface scans of Chebyshev-type force fields, based on an input parameter file. Below are a few examples of usage. As a word of caution, S\_MINIM, S\_MAXIM, and S\_DELTA are used to define the range and spacing between scan data points.

Example 1: Scan of only 2-body pair interactions (3-body energies not included in reported energy)

1. # PLOTPES #
2. true 3 params.txt-scan-2b
3. PAIRTYPE PARAMS: 0
4. PAIRTYPE PARAMS: 1
5. PAIRTYPE PARAMS: 2

Line 2 tells the program that 3 scans will be performed, and that parameters should be read from the provided parameter file. Lines 3,4, and 5 are taken **directly** from the parameter file, and specify the specific interaction types scan should be run for.

Example 2: Scan of only 3-body pair interactions (2-body energies not included in reported energy)

1. # PLOTPES #
2. true 4 params.txt-scan-3b
3. TRIPLETTYPE PARAMS: 0 SCAN 1
4. FIX IJ 3.0000 IK 3.0000 SCAN JK
5. TRIPLETTYPE PARAMS: 1 SCAN 2
6. FIX IJ 3.0000 IK 0.9572 SCAN JK
7. FIX IK 0.9572 JK 3.0000 SCAN IJ
8. TRIPLETTYPE PARAMS: 2 SCAN 2
9. FIX IJ 0.9572 IK 0.9572 SCAN JK
10. FIX IJ 0.9572 JK 1.5130 SCAN IK
11. TRIPLETTYPE PARAMS: 3 SCAN 1
12. FIX IJ 1.6238 IK 1.6238 SCAN JK

This example has similar syntax to the previous example, where now the first 3 “words” of lines 3, 5, 8, and 11 are taken directly from the parameter file. The following words are used to specify how many pairs within the 3 that define a 3-body interaction should be scanned. Lines 4, 6, 7, 9, 10, and 12 specify how those scans should be run. Scans for 3-body potentials are similar to an angle scan where bonds are held rigid. In other words, two distances are held fixed while a third is scanned. Words “IJ, IK, and JK” refer to a specific pair from the 3 atoms and correspond to the types given on lines containing “TRIPLETTYPE PARAMS:” in the parameter file.

Example 3: Scan of only 3-body pair interactions (2-body energies not included in reported 3-body energies), and 2-body energies (3-body energies not included in reported 3-body energies)

1. # PLOTPES #
2. true 4 params.txt-scan-3b
3. PAIRTYPE PARAMS: 0
4. PAIRTYPE PARAMS: 1
5. PAIRTYPE PARAMS: 2
6. TRIPLETTYPE PARAMS: 0 SCAN 1
7. FIX IJ 3.0000 IK 3.0000 SCAN JK
8. TRIPLETTYPE PARAMS: 1 SCAN 2
9. FIX IJ 3.0000 IK 0.9572 SCAN JK
10. FIX IK 0.9572 JK 3.0000 SCAN IJ
11. TRIPLETTYPE PARAMS: 2 SCAN 2
12. FIX IJ 0.9572 IK 0.9572 SCAN JK
13. FIX IJ 0.9572 JK 1.5130 SCAN IK
14. TRIPLETTYPE PARAMS: 3 SCAN 1
15. FIX IJ 1.6238 IK 1.6238 SCAN JK

This example simply shows that Examples 1 and 2 can be achieved with a single input file

Example 4: Scan of only 3-body pair interactions, with 2-body energies included

1. # PLOTPES #
2. true 4 params.txt-scan-3b
3. TRIPLETTYPE PARAMS: 0 SCAN 1 INCLUDE 2B PAIRTYPE PARAMS IJ 0 IK 0 JK 0
4. FIX IJ 3.0000 IK 3.0000 SCAN JK
5. TRIPLETTYPE PARAMS: 1 SCAN 2 INCLUDE 2B PAIRTYPE PARAMS IJ 0 IK 2 JK 2
6. FIX IJ 3.0000 IK 0.9572 SCAN JK
7. FIX IK 0.9572 JK 3.0000 SCAN IJ
8. TRIPLETTYPE PARAMS: 2 SCAN 2 INCLUDE 2B PAIRTYPE PARAMS IJ 2 IK 2 JK 1
9. FIX IJ 0.9572 IK 0.9572 SCAN JK
10. FIX IJ 0.9572 JK 1.5130 SCAN IK
11. TRIPLETTYPE PARAMS: 3 SCAN 1 INCLUDE 2B PAIRTYPE PARAMS IJ 1 IK 1 JK 1
12. FIX IJ 1.6238 IK 1.6238 SCAN JK

This example has syntax similar to Example 2. Do not combine this type of calculation with the type given in Example 1. The extra words on lines 3, 5, 8, and 11 specify that 2-body contributions should be included in the reported energies, and specifies what 2-body type each of the three pairs should be, where type index is taken from the parameter file.

# Small Utilities

These utilities can be found in the contrib (and sub-) folder(s).

|  |  |  |
| --- | --- | --- |
| Utility | Summary | Usage |
| combine\_xyz\_force.py | Take a b.txt or forceout.txt type file, a .xyz file, and a file with the box lengths for each frame and create a .xyzf file | Python combine\_xyz\_force.py <frames> <xyz file>  <force file> <box file> |
| dftbgen\_to\_xyz.py | Convert a dftbgen .gen file to a .xyz file | Python dftbgen\_to\_xyz.py <frames> <xyz file> |
| subtract\_forces.sh | Subtract forces from force field specified by parameter file from a .xyzf file’s forces | ./subtract\_forces.sh <frames> <input.xyzf file>  <input md file> |
| break\_apart\_xyz.py | Breaks a .xyzf trajectory file into its constituent frames | ./break\_apart\_xyz.py <frames> <input.xyzf file> |

# Self-Consistent Force Field Fitting

This functionality is provided by: **/contrib/submit\_self\_consist.sh**, and requires # SLFCNST # to be set true in the template run\_md.in file.

Self-consistent fitting is driven by a shell script that takes as input a QMMD trajectory and template files for the LSQ C++ fitting code, the MD code, and for VASP. See the shell script for more details on the required files.

In addition, the user specifies:

1. The number of self-consistent iterations to take
2. The number of frames from the original trajectory to always use in each iteration’s fitting trajectory
3. The number of frames from the first iteration’s trajectory to always use in each subsequent iteration’s fitting trajectory
4. The number of frames from all remaining trajectories to cycle into the fitting trajectory
5. The maximum number of frames to add to the total fitting trajectory – Once this number is exceeded, frames from iteration 2+ are cycled out and replaced with frames from newer iterations. This controls the total number of frames in the fitting trajectory at any given time.

To run a self-consistent fit, make sure the variables/paths in the driver script are set correctly, and call it from a queue-submission script.

This method will work best when combined with the fit refitting/extrapolation approach, but the combined use of these two capabilities is still being tested. See the next section for more details on this approach.

# Fit Refitting/Extrapolation

(Currently only tested for 2-body only fits)

This functionality is provided by: **/contrib/cheby\_refit/cheby\_fitting.py**

The standard fitting method had three potential pitfalls, which become more problematic when one attempts to perform a self-consistent fit “black box” type fit, where the user has limited ability to “tweak” parameters. Three such examples include:

1. Choice of penalty function parameters
2. Choice of Chebyshev order
3. Inadequate frames provided to ensure reasonable potential shape at small distance

The first of the above issues can come into play when running MD to generate the new fitting trajectory. Choice of penalty function parameters (in conjunction with selected time step) can influence the stability of the simulation; if the penalty functions are too steep, atoms can begin moving too fast and eventually lead to overlaps. The latter two issues are somewhat related. Occasionally, when sampling of small pair distances is poor, the fit potential can exhibit a decreasingly steep repulsive slope or an inflection point (more common when the Chebyshev order is too large). While the hope is that self-consistent fitting can remedy this type of behavior, if the shape in this region is too far off, the self-consistent fit will fail.

The refitting script allows the user many options to remedy these potential issues. The user specifies a PES scan and a threshold below which the script searches for a change in slope that indicates problematic behavior. The script then uses the three data points occurring immediately after the problematic region to perform an exponential fit, and replaces all points of smaller distance with the extrapolated values. The code then re-fits the Chebyshev parameters to the new PES.

The user can control many aspects of this fit including:

1. The range over which the extrapolation/refitting occurs. For example, the user may choose to only re-fit between *r*min and *r*max, and still use penalty functions to control behavior below *r*min. Alternatively, the user can have the exponential extrapolation continue to some product of kB*T* (a new *r*min will be output), or even to zero.
2. The Chebyshev power to use for fitting. The user can either specify a specific power to use, or allow the script to find the minimum required order to obtain a good fit.

NOTE: Compile the code with **FPENALTY\_POWER=0.0** when outputting PES for use with this feature; otherwise, the cubic fcut function will double counted when the refit potential is run through the MD code.