Table of Contents

Table of Contents

Table of Contents

LAMMPS Documentation

(12 Apr 2006 version of LAMMPS)

LAMMPS stands for Large−scale Atomic/Molecular Massively Parallel Simulator.

LAMMPS is a classical molecular dynamics simulation code designed to run efficiently on parallel computers. It was developed at Sandia National Laboratories, a US Department of Energy facility, with funding from the DOE. It is an open−source code, distributed freely under the terms of the GNU Public License (GPL).

The primary author of the code is Steve Plimpton, who can be contacted at sjplimp@sandia.gov. The LAMMPS WWW Site at www.cs.sandia.gov/~sjplimp/lammps.html has more information about the code and its uses.

The LAMMPS documentation is organized into the following sections. If you find errors or omissions in this manual or have suggestions for useful information to add, please send us an email so we can improve the LAMMPS documentation.

PDF file of the entire manual, generated by htmldoc

1. Introduction

These sections provide an overview of what LAMMPS can and can't do, describe what it means for LAMMPS to be an open−source code, and acknowledge the funding and people who have contributed to LAMMPS over the years.

1.1 What is LAMMPS

LAMMPS is a classical molecular dynamics code that models an ensemble of particles in a liquid, solid, or gaseous state. It can model atomic, polymeric, biological, metallic, or granular systems using a variety of force fields and boundary conditions.

For examples of LAMMPS simulations, see the Publications page of the LAMMPS WWW Site.

LAMMPS runs efficiently on single−processor desktop or laptop machines, but is designed for parallel computers. It will run on any parallel machine that compiles C++ and supports the MPI message−passing library. This includes distributed− or shared−memory parallel machines and Beowulf−style clusters.

LAMMPS can model systems with only a few particles up to millions or billions. See this section for information on LAMMPS performance and scalability, or the Benchmarks section of the LAMMPS WWW Site.

LAMMPS is a freely−available open−source code, distributed under the terms of the GNU Public License, which means you can use or modify the code however you wish. See this section for a brief discussion of the open−source philosophy.

LAMMPS is designed to be easy to modify or extend with new capabilities, such as new force fields, atom types, boundary conditions, or diagnostics. See this section for more details.

The current version of LAMMPS is written in C++. Earlier versions were written in F77 and F90. See this section for more information on different versions. All versions can be downloaded from the LAMMPS WWW Site.

LAMMPS was originally developed under a US Department of Energy CRADA (Cooperative Research and Development Agreement) between two DOE labs and 3 companies. It is distributed by Sandia National Labs. See this section for more information on LAMMPS funding and individuals who have contributed to LAMMPS.

In the most general sense, LAMMPS integrates Newton's equations of motion for collections of atoms, molecules, or macroscopic particles that interact via short− or long−range forces with a variety of initial and/or boundary conditions. For computational efficiency LAMMPS uses neighbor lists to keep track of nearby particles. The lists are optimized for systems with particles that are repulsive at short distances, so that the local density of particles never becomes too large. On parallel machines, LAMMPS uses spatial−decomposition techniques to partition the simulation domain into small 3d sub−domains, one of which is assigned to each processor. Processors communicate and store "ghost" atom information for atoms that border their sub−domain. LAMMPS is most efficient (in a parallel sense) for systems whose particles fill a 3d rectangular box with roughly uniform density. Papers with technical details of the algorithms used in LAMMPS are listed in this section.

1.2 LAMMPS features

This section highlights LAMMPS features, with pointers to specific commands which give more details. If LAMMPS doesn't have your favorite interatomic potential, boundary condition, or atom type, see this section, which describes how you can add it to LAMMPS.

Kinds of systems LAMMPS can simulate:

(atom style command)

- atomic (e.g. box of Lennard−Jonesium)
- bead−spring polymers
- united−atom polymers or organic molecules
- all−atom polymers, organic molecules, proteins, DNA
- metals
- granular materials
- hybrid systems

Force fields:

(pair style, bond style, angle style, dihedral style, improper style, kspace style commands)

- pairwise vanderWaals, Coulombic, Buckingham, Morse, Yukawa potentials
- long−range Coulombics via Ewald summation or particle−particle particle−mesh (PPPM)
- CHARMM, AMBER, class 2 (COMPASS) force fields
- bond potentials (harmonic, FENE, nonlinear, Morse, bond−breaking)
- angle potentials (harmonic, cosine)
- dihedral potentials (harmonic, multi−harmonic)
- out−of−plane potentials (harmonic, cvff)
- embedded atom method (EAM) for metals and metal alloys

- frictional force fields for granular materials
- tabulated pairwise force fields
- hybrid pairwise force fields (e.g. combinations of pairwise potentials)
- hybrid bond force fields (e.g. combinations of bond potentials)

Creation of atoms:

(read_data, lattice, create_atoms, delete_atoms, displace_atoms commands)

- read in atom coords from files
- create atoms on one or more lattices (e.g. grain boundaries)
- delete geometric or logical groups of atoms (e.g. voids)
- displace atoms

Ensembles, constraints, and boundary conditions:

(fix command)

- constant NVE, NVT, NPT ensembles
- temperature control via rescaling, Nose/Hoover, or Langevin thermostatting
- pressure control via Nose/Hoover barostatting in 1 to 3 dimensions
- volume rescaling
- altered motion via velocity and force constraints
- harmonic (umbrella) constraint forces
- dragging of atoms to new positions
- SHAKE bond &angle constraints on small clusters of atoms
- rigid body motion of one or more groups of atoms
- wall constraints of various kinds
- targeted molecular dynamics (TMD) constraints
- gravity

Integrators:

(run, run_style, temper commands)

- velocity−Verlet integrator
- rRESPA hierarchical time integrator
- parallel tempering (replica exchange) across multiple simulations
- multiple independent simulations simultaneously

Output:

(dump, restart commands)

- binary restart files
- text dump files of atom coords, velocities, other per−atom attributes
- per−atom energy, stress, centro−symmetry parameter

Pre− and post−processing:

Our group has also written and released a separate toolkit called Pizza.py which provides tools for doing setup, analysis, plotting, and visualization for LAMMPS simulations. Pizza.py is written in Python and is available for download from the Pizza.py WWW site.

1.3 LAMMPS non−features

LAMMPS is designed to efficiently compute Newton's equations of motion for a system of interacting particles. Many of the tools needed to pre− and post−process the data for such simulations are not included in the LAMMPS kernel for several reasons:

- the desire to keep LAMMPS simple
- they are not parallel operations
- other codes already do them
- limited development resources

Specifically, LAMMPS itself does not:

- run thru a GUI
- build molecular systems
- assign force−field coefficients automagically
- perform sophisticated analyses of your MD simulation
- visualize your MD simulation
- plot your output data

A few tools for pre− and post−processing tasks are provided as part of the LAMMPS package; they are described in this section. However, many people use other codes or write their own tools for these tasks.

As noted above, our group has also written and released a separate toolkit called Pizza.py which addresses some of the listed bullets. It provides tools for doing setup, analysis, plotting, and visualization for LAMMPS simulations. Pizza.py is written in Python and is available for download from the Pizza.py WWW site.

LAMMPS requires as input a list of initial atom coordinates and types, molecular topology information, and force−field coefficients assigned to all atoms and bonds. LAMMPS will not build molecular systems and assign force−field parameters for you.

For atomic systems LAMMPS provides a create_atoms command which places atoms on solid−state lattices (fcc, bcc, etc). Assigning small numbers of force field coefficients can be done via the pair coeff, bond coeff, angle coeff, etc commands. For molecular systems or more complicated simulation geometries, users typically use another code as a builder and convert its output to LAMMPS input format, or write their own code to generate atom coordinate and molecular topology for LAMMPS to read in.

For complicated molecular systems (e.g. a protein), a multitude of topology information and hundreds of force−field coefficients must typically be specified. We suggest you use a program like CHARMM or AMBER or other molecular builders to setup such problems and dump its information to a file. You can then reformat the file as LAMMPS input. Some of the tools in this section can assist in this process.

Similarly, LAMMPS creates output files in a simple format. Most users post−process these files with their own analysis tools or re−format them for input into other programs, including visualization packages. If you are convinced you need to compute something on−the−fly as LAMMPS runs, see this section for a discussion of how you can use the dump and fix commands to print out data of your choosing. Keep in mind that complicated computations can slow down the molecular dynamics timestepping, particularly if the computations are not parallel, so it is often better to leave such analysis to post−processing codes.

A very simple (yet fast) visualizer is provided with the LAMMPS package − see the xmovie tool in this section. It creates xyz projection views of atomic coordinates and animates them. We find it very useful for debugging purposes. For high−quality visualization we recommend the following packages:

- Raster3d
- RasMol
- VMD
- AtomEye

Other features that LAMMPS does not yet (and may never) support are discussed in this section.

Finally, these are freely−available molecular dynamics codes, most of them parallel, which may be well−suited to the problems you want to model. They can also be used in conjunction with LAMMPS to perform complementary modeling tasks.

- CHARMM
- AMBER
- NAMD
- NWCHEM
- DL_POLY
- Tinker

CHARMM, AMBER, NAMD, NWCHEM, and Tinker are designed primarily for modeling biological molecules. CHARMM and AMBER use atom−decomposition (replicated−data) strategies for parallelism; NAMD and NWCHEM use spatial−decomposition approaches, similar to LAMMPS. Tinker is a serial code. DL_POLY includes potentials for a variety of biological and non−biological materials; both a replicated−data and spatial−decomposition version exist.

1.4 Open source distribution

LAMMPS comes with no warranty of any kind. As each source file states in its header, it is a copyrighted code that is distributed free−of− charge, under the terms of the GNU Public License (GPL). This is often referred to as open−source distribution − see www.gnu.org or www.opensource.org for more details. The legal text of the GPL is in the LICENSE file that is included in the LAMMPS distribution.

Here is a summary of what the GPL means for LAMMPS users:

(1) Anyone is free to use, modify, or extend LAMMPS in any way they choose, including for commercial purposes.

(2) If you distribute a modified version of LAMMPS, it must remain open−source, meaning you distribute it under the terms of the GPL. You should clearly annotate such a code as a derivative version of LAMMPS.

(3) If you release any code that includes LAMMPS source code, then it must also be open−sourced, meaning you distribute it under the terms of the GPL.

(4) If you give LAMMPS files to someone else, the GPL LICENSE file and source file headers (including the copyright and GPL notices) should remain part of the code.

In the spirit of an open−source code, these are various ways you can contribute to making LAMMPS better. You can send email on any of these items.

- Point prospective users to the LAMMPS WWW Site. Mention it in talks or link to it from your WWW site.
- If you find an error or omission in this manual or on the LAMMPS WWW Site, or have a suggestion for something to clarify or include, send an email.
- If you find a bug, this section describes how to report it.
- If you publish a paper using LAMMPS results, send the citation (and any cool pictures or movies if you like) to add to the Publications, Pictures, and Movies pages of the LAMMPS WWW Site, with links and attributions back to you.
- Create a new Makefile.machine that can be added to the src/MAKE directory.
- The tools sub−directory of the LAMMPS distribution has various stand−alone codes for pre− and post−processing of LAMMPS data. More details are given in this section. If you write a new tool that users will find useful, it can be added to the LAMMPS distribution.
- LAMMPS is designed to be easy to extend with new code for features like potentials, boundary conditions, diagnostic computations, etc. This section gives details. If you add a feature of general interest, it can be added to the LAMMPS distribution.
- The Benchmark page of the LAMMPS WWW Site lists LAMMPS performance on various platforms. The files needed to run the benchmarks are part of the LAMMPS distribution. If your machine is sufficiently different from those listed, your timing data can be added to the page.
- You can send feedback for the User Comments page of the LAMMPS WWW Site. It might be added to the page. No promises.
- Cash. Small denominations, unmarked bills preferred. Paper sack OK. Leave on desk. VISA also accepted. Chocolate chip cookies encouraged.

1.5 Acknowledgments and citations

LAMMPS development has been funded by the US Department of Energy (DOE), through its CRADA, LDRD, ASCI, and Genomes−to−Life programs and its OASCR and OBER offices.

Specifically, work on the latest version was funded in part by the US Department of Energy's Genomics:GTL program (www.doegenomestolife.org) under the project, "Carbon Sequestration in Synechococcus Sp.: From Molecular Machines to Hierarchical Modeling".

The following papers describe the parallel algorithms used in LAMMPS.

S. J. Plimpton, **Fast Parallel Algorithms for Short−Range Molecular Dynamics**, J Comp Phys, 117, 1−19 (1995).

S. J. Plimpton, R. Pollock, M. Stevens, **Particle−Mesh Ewald and rRESPA for Parallel Molecular Dynamics Simulations**, in Proc of the Eighth SIAM Conference on Parallel Processing for Scientific Computing, Minneapolis, MN (March 1997).

If you use LAMMPS results in your published work, please cite the J Comp Phys reference and include a pointer to the LAMMPS WWW Site (www.cs.sandia.gov/~sjplimp/lammps.html). A paper describing the latest version of LAMMPS is in the works; when it appears in print, you can check the LAMMPS WWW Site for a more current citation.

If you send me information about your publication, I'll be pleased to add it to the Publications page of the LAMMPS WWW Site. Ditto for a picture or movie for the Pictures or Movies pages.

The primary author of LAMMPS is Steve Plimpton at Sandia National Labs. Others have made significant contributions to the code:

Ewald and PPPM solvers	Roy Pollock (LLNL)
rRESPA	Mark Stevens &Paul Crozier (Sandia)
NVT/NPT integrators	Mark Stevens (Sandia)
class 2 force fields	Eric Simon (Cray)
HTFN energy minimizer	Todd Plantenga (Sandia)
msi2lmp tool	Steve Lustig (Dupont), Mike Peachey &John Carpenter (Cray)
CHARMM force fields	Paul Crozier (Sandia)
2d Ewald/PPPM	Paul Crozier (Sandia)
granular force fields and BC	Leo Silbert &Gary Grest (Sandia)
multi−harmonic dihedral potential	Mathias Putz (Sandia)
EAM potentials	Stephen Foiles (Sandia)
parallel tempering	Mark Sears (Sandia)
lmp2cfg and lmp2traj tools	Ara Kooser, Jeff Greathouse, Andrey Kalinichev (Sandia)
FFT support for SGI SCLS (Altix)	Jim Shepherd (Ga Tech)
targeted molecular dynamics (TMD)	Paul Crozier (Sandia), Christian Burisch (Bochum Univeristy, Germany)
force tables for	Paul Crozier

long−range Coulombics	(Sandia)
radial distribution functions	Paul Crozier &Jeff Greathouse (Sandia)
Morse bond potential	Jeff Greathouse (Sandia)
CHARMM LAMMPS tool	Pieter in't Veld and Paul Crozier (Sandia)
AMBER LAMMPS tool	Keir Novik (Univ College London) and Vikas Varshney (U Akron)
electric field fix	Christina Payne (Vanderbilt U)
cylindrical indenter fix	Ravi Agrawal (Northwestern U)
compressed dump files	Erik Luijten (U Illinois)
thermodynamics enhanced by fix quantities	Aidan Thompson (Sandia)
uniaxial strain fix	Carsten Svaneborg (Max Planck Institute)
TIP4P potential (4−site water)	Ahmed Ismail and Amalie Frischknecht (Sandia)
dissipative particle dynamics (DPD) potentials	Kurt Smith (U Pitt) and Frank van Swol (Sandia)
Finnis/Sinclair EAM	Tim Lau (MIT)
helix dihedral potential	Naveen Michaud−Agrawal (Johns Hopkins U) and Mark Stevens (Sandia)
restrain bond potential	Naveen Michaud−Agrawal (Johns Hopkins U)
cosine/squared	Naveen

angle potential	Michaud−Agrawal (Johns Hopkins U)

Other CRADA partners involved in the design and testing of LAMMPS were

- John Carpenter (Mayo Clinic, formerly at Cray Research)
- Terry Stouch (Lexicon Pharmaceuticals, formerly at Bristol Myers Squibb)
- Steve Lustig (Dupont)
- Jim Belak (LLNL)

Previous Section − LAMMPS WWW Site − LAMMPS Documentation − LAMMPS Commands − Next Section

2. Getting Started

This section describes how to unpack, make, and run LAMMPS, for both new and experienced users.

2.1 What's in the LAMMPS distribution

When you download LAMMPS you will need to unzip and untar the downloaded file with the following commands, after placing the file in an appropriate directory.

```
gunzip lammps*.tar.gz
tar xvf lammps*.tar
```

This will create a LAMMPS directory containing two files and several sub−directories:

README	text file
LICENSE	the GNU General Public License (GPL)
bench	benchmark problems
doc	documentation
examples	simple test problems
potentials	embedded atom method (EAM) potential files
src	source files
tools	pre− and post−processing tools

2.2 Making LAMMPS

Read this first:

Building LAMMPS can be non−trivial. You will likely need to edit a makefile, there are compiler options, additional libraries can be used (MPI, FFT), etc. Please read this section carefully. If you are not comfortable with makefiles, or building codes on a Unix platform, or running an MPI job on your machine, please find a local expert to help you. Many of the emails I get about build and run problems are not really about LAMMPS − they are peculiar to the user's system, compilers, libraries, etc. Such questions are better answered by a local expert.

If you have a build problem that you are convinced is a LAMMPS issue (e.g. the compiler complains about a line of LAMMPS source code), then please send an email. Note that doesn't include linking problems − that's a question for a local expert!

Also, if you succeed in building LAMMPS on a new kind of machine (which there isn't a similar Makefile for in the distribution), send it to sjplimp@sandia.gov and we'll include it in future LAMMPS releases.

Building a LAMMPS executable:

The src directory contains the C++ source and header files for LAMMPS. It also contains a top−level Makefile and a MAKE directory with low−level Makefile.* files for several machines. From within the src directory, type "make" or "gmake". You should see a list of available choices. If one of those is the machine and options you want, you can type a command like:

```
make linux
gmake mac
```

If you get no errors and an executable like lmp_linux or lmp_mac is produced, you're done; it's your lucky day. The remainder of this section addressed the following topics: errors that occur when making LAMMPS, editing a new low−level Makefile.foo, how to make LAMMPS with and without packages, and additional build tips.

Errors that occur when making LAMMPS:

(1) If the make command breaks immediately with errors that indicate it can't find files with a "*" in their names, this can be because your machine's make doesn't support wildcard expansion in a makefile. Try gmake instead of make. If that doesn't work, try using a −f switch with your make command to use Makefile.list which explicitly lists all the needed files, e.g.

```
make makelist
make −f Makefile.list linux
gmake −f Makefile.list mac
```

The first "make" command will create a current Makefile.list with all the file names in your src dir. The 2nd "make" command (make or gmake) will use it to build LAMMPS.

(2) Other errors typically occur because the low−level Makefile isn't setup correctly for your machine. If your platform is named "foo", you need to create a Makefile.foo in the MAKE directory. Use whatever existing file is closest to your platform as a starting point. See the next section for more instructions.

Editing a new low−level Makefile.foo:

These are the issues you need to address when editing a low−level Makefile for your machine. With a couple exceptions, the only portion of the file you should need to edit is the "System−specific Settings" section.

(1) Change the first line of Makefile.foo to include the word "foo" and whatever other options you set. This is the line you will see if you just type "make".

(2) Set the paths and flags for your C++ compiler, including optimization flags. You can use g++, the open−source GNU compiler, which is available on all Unix systems. Vendor compilers often produce faster code. On boxes with Intel CPUs, I use the free Intel icc compiler, which you can download from Intel's compiler site.

(3) If you want LAMMPS to run in parallel, you must have an MPI library installed on your platform. Makefile.foo needs to specify where the mpi.h file (−I switch) and the libmpi.a library (−L switch) is found. On my Linux box, I use Argonne's MPICH 1.2 which can be downloaded from the Argonne MPI site. LAM MPI should also work. If you are running on a big parallel platform, your system people or the vendor should have already installed a version of MPI, which will be faster than MPICH or LAM, so find out how to link against it. If you use MPICH or LAM, you will have to configure and build it for your platform. The MPI configure script should have compiler options to enable you to use the same compiler you are using for the LAMMPS build, which can avoid problems that may arise when linking LAMMPS to the MPI library.

(4) If you just want LAMMPS to run on a single processor, you can use the STUBS library in place of MPI, since you don't need an MPI library installed on your system. See the Makefile.serial file for how to specify the −I and −L switches. You will also need to build the STUBS library for your platform before making LAMMPS itself. From the STUBS dir, type "make" and it will hopefully create a libmpi.a suitable for linking to LAMMPS. If the build fails, you will need to edit the STUBS/Makefile for your platform.

The file STUBS/mpi.cpp has a CPU timer function MPI_Wtime() that calls gettimeofday() . If your system doesn't support gettimeofday() , you'll need to insert code to call another timer. Note that the ANSI−standard function clock() rolls over after an hour or so, and is therefore insufficient for timing long LAMMPS runs.

(5) If you want to use the particle−particle particle−mesh (PPPM) option in LAMMPS for long−range Coulombics, you must have a 1d FFT library installed on your platform. This is specified by a switch of the form −DFFT_XXX where XXX = INTEL, DEC, SGI, SCSL, or FFTW. All but the last one are native vendor−provided libraries. FFTW is a fast, portable library that should work on any platform. You can download it from www.fftw.org. Use version 2.1.X, not the newer 3.0.X. Building FFTW for my box was as simple as ./configure; make. Whichever FFT library you have on your platform, you'll need to set the appropriate −I and −L switches in Makefile.foo.

If you examine fft3d.c and fft3d.h you'll see it's possible to add other vendor FFT libraries via #ifdef statements in the appropriate places. If you successfully add a new FFT option, like −DFFT_IBM, please send me an email; I'd like to add it to LAMMPS.

(6) If you don't plan to use PPPM, you don't need an FFT library. Use a −DFFT_NONE switch in the CCFLAGS setting of Makefile.foo, or exclude the KSPACE package (see below).

(7) There are a few other −D compiler switches you can set as part of CCFLAGS. The read_data and dump commands will read/write gzipped files if you compile with −DGZIP. It requires that your Unix support the "popen" command. Using one of the −DPACK_ARRAY, −DPACK_POINTER, and −DPACK_MEMCPY options can make for faster parallel FFTs (in the PPPM solver) on some platforms. The −DPACK_ARRAY

setting is the default.

(8) The DEPFLAGS setting is how the C++ compiler creates a dependency file for each source file. This speeds re−compilation when source (*.cpp) or header (*.h) files are edited. Some compilers do not support dependency file creation, or may use a different switch than −D. GNU g++ works with −D. If your compiler can't create dependency files (a long list of errors involving *.d files), then you'll need to create a Makefile.foo patterned after Makefile.tflop, which uses different rules that do not involve dependency files.

That's it. Once you have a correct Makefile.foo and you have pre−built the MPI and FFT libraries it will use, all you need to do from the src directory is type one of these 2 commands:

```
make foo
gmake foo
```

You should get the executable lmp_foo when the build is complete.

How to make LAMMPS with and without packages:

The source code for LAMMPS is structured as a large set of core files that are always used plus additional packages, which are groups of files that enable a specific set of features. For example, force fields for molecular systems or granular systems are in packages. You can see the list of packages by typing "make package". The current list of packages is as follows:

class2	class 2 force fields
dpd	dissipative particle dynamics (DPD) force field
granular	force fields and boundary conditions for granular systems
kspace	long−range Ewald and particle−mesh (PPPM) solvers
molecule	force fields for molecular systems

Any or all of these packages can be included or excluded when LAMMPS is built. The default is to include only the kspace and molecule packages. You may wish to exclude certain packages if you will never run certain kinds of simulations. This will produce a smaller executable which in some cases will also run a bit faster.

Packages are included or excluded by typing "make yes−name" or "make no−name", where "name" is the name of the package. You can also type "make yes−all" or "make no−all" to include/exclude all optional packages. These commands work by simply moving files back and forth between the main src directory and sub−directories with the package name, so that the files are not seen when LAMMPS is built. After you have included or excluded a package, you must re−make LAMMPS.

Additional make options exist to help manage LAMMPS files that exist in both the src directory and in package sub−directories. Typing "make package−update" will overwrite src files with files from the package directories if the package has been included. Typing "make package−overwrite" will overwrite files in the package directories with src files. Typing "make package−check" will list differences between src and package versions of the same files.

Building LAMMPS as a library:

LAMMPS can be built as a library, which can then be called from another application or a scripting language. This is done by typing

```
make makelib
make −f Makefile.lib foo
```

where foo is the machine name. The first "make" command will create a current Makefile.lib with all the file names in your src dir. The 2nd "make" command will use it to build LAMMPS as a library. This requires that Makefile.foo have a library target (lib) and system−specific settings for ARCHIVE and ARFLAGS. See Makefile.linux for an example. This will create the file liblmp_foo.a which another application can link to. The library has 3 callable functions:

```
void lammps_open(int, char **);
void lammps_close();
int lammps_command(char *);
```

The lammps_open() function is used to initialize LAMMPS, passing in a list of strings as if they were command−line arguments when LAMMPS is run from the command line. The lammps_close() function is used to shut down LAMMPS and free all its memory. The lammps_command() function is used to pass a string to LAMMPS as if it were an input command read from an input script. See the library.cpp file for more information about the arguments and return values for these 3 functions.

Additional build tips:

(1) Building LAMMPS for multiple platforms.

You can make LAMMPS for multiple platforms from the same src directory. Each target creates its own object sub−dir called Obj_name where it stores the system−specific *.o files.

(2) Cleaning up.

Typing "make clean" will delete all *.o object files created when LAMMPS is built.

(3) On some 64−bit machines, compiling with −O3 appears to break the Coulombic tabling option used by the pair_style *lj/cut/coul/long* and *lj/charmm/coul/long* styles. By default, tabling is used by these styles since it can offer a 2x speed−up. It can be disabled via the pair_modify command. Alternatively, the associated files (e.g. pair_lj_cut_coul_long.cpp) can be compiled with −O2, or with the compiler flag *−fno−strict−aliasing*. Either of those build changes seems to fix the problem.

(4) Building for a Macintosh.

OS X is BSD Unix, so it already works. See the Makefile.mac file.

(5) Building for MicroSoft Windows.

I've never done this, but LAMMPS is just standard C++ with MPI and FFT calls. You should be able to use cygwin to build LAMMPS with a Unix make. Or you should be able to pull all the source files into Visual C++ (ugh) or some similar development environment and build it. In the src/MAKE/Windows directory are some notes from users on how they built LAMMPS under Windows, so you can look at their instructions for tips. Good luck − I can't help you on this one.

2.3 Running LAMMPS

By default, LAMMPS runs by reading commands from stdin; e.g. lmp_linux < in.file. This means you first create an input script (e.g. in.file) containing the desired commands. This section describes how input scripts are structured and what commands they contain.

You can test LAMMPS on any of the sample inputs provided in the examples directory. Input scripts are named in.* and sample outputs are named log.*.name.P where name is a machine and P is the number of processors it was run on.

Here is how you might run one of the Lennard−Jones tests on a Linux box, using mpirun to launch a parallel job:

```
cd src
make linux
cp lmp_linux ../examples/lj
cd ../examples/lj
mpirun −np 4 lmp_linux <in.lj.nve
```

The screen output from LAMMPS is described in the next section. As it runs, LAMMPS also writes a log.lammps file with the same information. Note that this sequence of commands copied the LAMMPS executable (lmp_linux) to the directory with the input files. If you don't do this, LAMMPS may look for input files or create output files in the directory where the executable is, rather than where you run it from.

If LAMMPS encounters errors in the input script or while running a simulation it will print an ERROR message and stop or a WARNING message and continue. See this section for a discussion of the various kinds of errors LAMMPS can or can't detect, a list of all ERROR and WARNING messages, and what to do about them.

LAMMPS can run a problem on any number of processors, including a single processor. In theory you should get identical answers on any number of processors and on any machine. In practice, numerical round−off can cause slight differences and eventual divergence of molecular dynamics phase space trajectories.

LAMMPS can run as large a problem as will fit in the physical memory of one or more processors. If you run out of memory, you must run on more processors or setup a smaller problem.

2.4 Command−line options

At run time, LAMMPS recognizes several optional command−line switches which may be used in any order. For example, lmp_ibm might be launched as follows:

```
mpirun −np 16 lmp_ibm −var f tmp.out −log my.log −screen none <in.alloy
```

These are the command−line options:

```
−echo style
```

Set the style of command echoing. The style can be *none* or *screen* or *log* or *both*. Depending on the style, each command read from the input script will be echoed to the screen and/or logfile. This can be useful to figure out which line of your script is causing an input error. The default value is *log*. The echo style can also be set by using the echo command in the input script itself.

```
−partition 8x2 4 5 ...
```

Invoke LAMMPS in multi−partition mode. When LAMMPS is run on P processors and this switch is not used, LAMMPS runs in one partition, i.e. all P processors run a single simulation. If this switch is used, the P processors are split into separate partitions and each partition runs its own simulation. The arguments to the switch specify the number of processors in each partition. Arguments of the form MxN mean M partitions, each with N processors. Arguments of the form N mean a single partition with N processors. The sum of processors in all partitions must equal P. Thus the command "−partition 8x2 4 5" has 10 partitions and runs on a total of 25 processors.

The input script specifies what simulation is run on which partition; see the variable and next commands. Simulations running on different partitions can also communicate with each other; see the temper command.

```
−in file
```

Specify a file to use as an input script. This is an optional switch when running LAMMPS in one−partition mode. If it is not specified, LAMMPS reads its input script from stdin − e.g. lmp_linux < in.run. This is a required switch when running LAMMPS in multi−partition mode, since multiple processors cannot all read from stdin.

```
−log file
```

Specify a log file for LAMMPS to write status information to. In one−partition mode, if the switch is not used, LAMMPS writes to the file log.lammps. If this switch is used, LAMMPS writes to the specified file. In multi−partition mode, if the switch is not used, a log.lammps file is created with hi−level status information. Each partition also writes to a log.lammps.N file where N is the partition ID. If the switch is specified in multi−partition mode, the hi−level logfile is named "file" and each partition also logs information to a file.N. For both one−partition and multi−partition mode, if the specified file is "none", then no log files are created. Using a log command in the input script will override this setting.

```
−screen file
```

Specify a file for LAMMPS to write it's screen information to. In one−partition mode, if the switch is not used, LAMMPS writes to the screen. If this switch is used, LAMMPS writes to the specified file instead and you will see no screen output. In multi−partition mode, if the switch is not used, hi−level status information is written to the screen. Each partition also writes to a screen.N file where N is the partition ID. If the switch is specified in multi−partition mode, the hi−level screen dump is named "file" and each partition also writes screen information to a file.N. For both one−partition and multi−partition mode, if the specified file is "none", then no screen output is performed.

```
−var X value
```

Specify a variable that will be defined for substitution purposes when the input script is read. X should be a single lower−case character from 'a' to 'z'. The value can be any string. Using this command−line option is equivalent to putting the line "variable X index value" at the beginning of the input script. See the variable command for more information.

2.5 LAMMPS screen output

As LAMMPS reads an input script, it prints information to both the screen and a log file about significant actions it takes to setup a simulation. When the simulation is ready to begin, LAMMPS performs various initializations and prints the amount of memory (in MBytes per processor) that the simulation requires. It also prints details of the initial thermodynamic state of the system. During the run itself, thermodynamic information is printed periodically, every few timesteps. When the run concludes, LAMMPS prints the final thermodynamic state and a total run time for the simulation. It then appends statistics about the CPU time and storage requirements for the simulation. An example set of statistics is shown here:

```
Loop time of 49.002 on 2 procs for 2004 atoms

Pair   time (%) = 35.0495 (71.5267)
Bond   time (%) = 0.092046 (0.187841)
Kspce  time (%) = 6.42073 (13.103)
Neigh  time (%) = 2.73485 (5.5811)
Comm   time (%) = 1.50291 (3.06703)
Outpt  time (%) = 0.013799 (0.0281601)
Other  time (%) = 2.13669 (4.36041)

Nlocal:    1002 ave, 1015 max, 989 min
Histogram: 1 0 0 0 0 0 0 0 0 1
Nghost:    8720 ave, 8724 max, 8716 min
Histogram: 1 0 0 0 0 0 0 0 0 1
Neighs:    354141 ave, 361422 max, 346860 min
Histogram: 1 0 0 0 0 0 0 0 0 1

Total # of neighbors = 708282
Ave neighs/atom = 353.434
Ave special neighs/atom = 2.34032
Number of reneighborings = 42
Dangerous reneighborings = 2
```

The first section gives the breakdown of the CPU run time (in seconds) into major categories. The second section lists the number of owned atoms (Nlocal), ghost atoms (Nghost), and pair−wise neighbors stored per processor. The max and min values give the spread of these values across processors with a 10−bin histogram showing the distribution. The total number of histogram counts is equal to the number of processors.

The last section gives aggregate statistics for pair−wise neighbors and special neighbors that LAMMPS keeps track of (see the special_bonds command). The number of times neighbor lists were rebuilt during the run is given as well as the number of potentially "dangerous" rebuilds. If atom movement triggered neighbor list rebuilding (see the neigh_modify command), then dangerous reneighborings are those that were triggered on the first timestep atom movement was checked for. If this count is non−zero you may wish to reduce the delay factor to insure no force interactions are missed by atoms moving beyond the neighbor skin distance before a rebuild takes place.

If an energy minimization was performed, additional information is printed that includes the energy change and convergence criteria.

2.6 Tips for users of previous LAMMPS versions

LAMMPS 2003 is a complete C++ rewrite of LAMMPS 2001, which was written in F90. Features of earlier versions of LAMMPS are listed in this section. The F90 and F77 versions (2001 and 99) are also freely distributed as open−source codes; check the LAMMPS WWW Site for distribution information if you prefer those versions. The 99 and 2001 versions are no longer under active development; they do not have all the features of LAMMPS 2003.

If you are a previous user of LAMMPS 2001, these are the most significant changes you will notice in LAMMPS 2003:

(1) The names and arguments of many input script commands have changed. All commands are now a single word (e.g. read_data instead of read data).

(2) All the functionality of LAMMPS 2001 is included in LAMMPS 2003, but you may need to specify the relevant commands in different ways.

(3) The format of the data file can be streamlined for some problems. See the read_data command for details. The data file section "Nonbond Coeff" has been renamed to "Pair Coeff" in LAMMPS 2003.

(4) Binary restart files written by LAMMPS 2001 cannot be read by LAMMPS 2003 with a read_restart command. This is because they were output by F90 which writes in a different binary format than C or C++ writes or reads. Use the *restart2data* tool provided with LAMMPS 2001 to convert the 2001 restart file to a text data file. Then edit the data file as necessary before using the LAMMPS 2003 read_data command to read it in.

(5) There are numerous small numerical changes in LAMMPS 2003 that mean you will not get identical answers when comparing to a 2001 run. However, your initial thermodynamic energy and MD trajectory should be close if you have setup the problem for both codes the same.

Previous Section − LAMMPS WWW Site − LAMMPS Documentation − LAMMPS Commands − Next Section

3. Commands

This section describes how a LAMMPS input script is formatted and what commands are used to define a LAMMPS simulation.

3.1 LAMMPS input script

LAMMPS executes by reading commands from a input script (text file), one line at a time. When the input script ends, LAMMPS exits. Each command causes LAMMPS to take some action. It may set an internal variable, read in a file, or run a simulation. Most commands have default settings, which means you only need to use the command if you wish to change the default.

In many cases, the ordering of commands in an input script is not important. However the following rules apply:

(1) LAMMPS does not read your entire input script and then perform a simulation with all the settings. Rather, the input script is read one line at a time and each command takes effect when it is read. Thus this sequence of commands:

```
timestep 0.5
run      100
run      100
```

does something different than this sequence:

```
run      100
timestep 0.5
run      100
```

In the first case, the specified timestep (0.5 fmsec) is used for two simulations of 100 timesteps each. In the 2nd case, the default timestep (1.0 fmsec) is used for the 1st 100 step simulation and a 0.5 fmsec timestep is used for the 2nd one.

(2) Some commands are only valid when they follow other commands. For example you cannot set the temperature of a group of atoms until atoms have been defined and a group command is used to define which atoms belong to the group.

(3) Sometimes command B will use values that can be set by command A. This means command A must precede command B in the input script if it is to have the desired effect. For example, the read_data command initializes the system by setting up the simulation box and assigning atoms to processors. If default values are not desired, the processors and boundary commands need to be used before read_data to tell LAMMPS how to map processors to the simulation box.

Many input script errors are detected by LAMMPS and an ERROR or WARNING message is printed. This section gives more information on what errors mean. The documentation for each command lists restrictions on how the command can be used.

3.2 Parsing rules

Each non−blank line in the input script is treated as a command. LAMMPS commands are case sensitive. Command names are lower−case, as are specified command arguments. Upper case letters may be used in file names or user−chosen ID strings.

Here is how each line in the input script is parsed by LAMMPS:

(1) If the line ends with a ""character (with no trailing whitespace), the command is assumed to continue on the next line. The next line is concatenated to the previous line by removing the ""character and newline. This allows long commands to be continued across two or more lines.

(2) All characters from the first "#" character onward are treated as comment and discarded.

(3) The line is searched repeatedly for $ characters. If the character following the $ is "a" to "z", the two−characters sequence (e.g. $x) is replaced with the corresponding variable text. See the variable command for details.

(4) The line is broken into "words" separated by whitespace (tabs, spaces). Note that words can thus contain letters, digits, underscores, or punctuation characters.

(5) The first word is the command name. All successive words in the line are arguments.

(6) Text with spaces can be enclosed in double quotes so it will be treated as a single argument. See the dump modify or fix print commands for examples. A '#' or '$' charater in text between double quotes will also not be treated as a comment or substituted for as a variable.

3.3 Input script structure

This section describes the structure of a typical LAMMPS input script. The "examples" directory in the LAMMPS distribution contains many sample input scripts; the corresponding problems are discussed in this section, and animated on the LAMMPS WWW Site.

A LAMMPS input script typically has 4 parts:

1. Initialization
2. Atom definition
3. Settings
4. Run a simulation

The last 2 parts can be repeated as many times as desired. I.e. run a simulation, change some settings, run some more, etc. Each of the 4 parts is now described in more detail. Remember that almost all the commands need only be used if a non−default value is desired.

(1) Initialization

Set parameters that need to be defined before atoms are created or read−in from a file.

The relevant commands are units, dimension, newton, processors, boundary, atom_style, atom_modify.

If force−field parameters appear in the files that will be read, these commands tell LAMMPS what kinds of force fields are being used: pair_style, bond_style, angle_style, dihedral_style, improper_style.

(2) Atom definition

There are 3 ways to define atoms in LAMMPS. Read them in from a data or restart file via the read_data or read_restart commands. These files can contain molecular topology information. Or create atoms on a lattice (with no molecular topology), using these commands: lattice, orient, origin, region, create_box, create_atoms. The entire set of atoms can be duplicated to make a larger simulation using the replicate command.

(3) Settings

Once atoms and molecular topology are defined, a variety of settings can be specified: force field coefficients, simulation parameters, output options, etc.

Force field coefficients are set by these commands (they can also be set in the read−in files): pair_coeff, bond_coeff, angle_coeff, dihedral_coeff, improper_coeff, kspace_style, dielectric, special_bonds.

Various simulation parameters are set by these commands: temperature, temp_modify, neighbor, neigh_modify, group, timestep, reset_timestep, run_style, min_style, min_modify.

Fixes impose a variety of boundary conditions, time integration, and diagnostic options. The fix command comes in many flavors.

Output options are set by these commands: thermo, dump, restart.

(4) Run a simulation

A molecular dynamics simulation is run using the run command. Energy minimizationn (molecular statics) is performed using the minimize command. A parallel tempering (replica−exchange) simulation can be run using the temper command.

3.4 Commands listed by category

This section lists all LAMMPS commands, grouped by category. The next section lists the same commands alphabetically. Note that some commands and their style options are part of specific LAMMPS packages. All packages are included in a LAMMPS build by default, but if you excluded a specific package when building LAMMPS, you cannot use the associated commands or styles. These dependencies are listed as Restrictions in the command's documentation.

Initialization:

atom_modify, atom_style, boundary, dimension, newton, processors, units

Atom definition:

create_atoms, create_box, lattice, orient, origin, read_data, read_restart, region, replicate

Force fields:

angle_coeff, angle_style, bond_coeff, bond_style, dielectric, dihedral_coeff, dihedral_style, improper_coeff, improper_style, kspace_modify, kspace_style, pair_coeff, pair_modify, pair_style, pair_write, special_bonds

Settings:

dipole, group, mass, min_modify, min_style, neigh_modify, neighbor, reset_timestep, run_style, set, temp_modify, temperature, timestep, velocity

Fixes:

fix, fix_modify, unfix

Output:

dump, dump_modify, restart, thermo, thermo_modify, thermo_style, undump, write_restart

Actions:

delete_atoms, delete_bonds, displace_atoms, minimize, run, temper

Miscellaneous:

cd, clear, echo, include, jump, label, log, next, print, variable

---

3.5 Individual commands

This section lists all LAMMPS commands alphabetically. The previous section lists the same commands, grouped by category. Note that some commands and their style options are part of specific LAMMPS packages. All packages are included in a LAMMPS build by default, but if you excluded a specific package when building LAMMPS, you cannot use the associated commands or styles. These dependencies are listed as Restrictions in the command's documentation.

angle_coeff	angle_style	atom_modify	atom_style	bond_coeff	bond_style
boundary	cd	clear	create_atoms	create_box	delete_atoms
delete_bonds	dielectric	dihedral_coeff	dihedral_style	dimension	dipole
displace_atoms	dump	dump_modify	echo	fix	fix_modify
group	improper_coeff	improper_style	include	jump	kspace_modify
kspace_style	label	lattice	log	mass	minimize
min_modify	min_style	neigh_modify	neighbor	newton	next
orient	origin	pair_coeff	pair_modify	pair_style	pair_write
print	processors	read_data	read_restart	region	replicate
reset_timestep	restart	run	run_style	set	special_bonds
temp_modify	temper	temperature	thermo	thermo_modify	thermo_style
timestep	undump	unfix	units	variable	velocity
write_restart					

Previous Section − LAMMPS WWW Site − LAMMPS Documentation − LAMMPS Commands − Next Section

---

4. How−to discussions

The following sections describe what commands can be used to perform certain kinds of LAMMPS simulations.

The example input scripts included in the LAMMPS distribution and highlighted in this section also show

how to setup and run various kinds of problems.

4.1 Restarting a simulation

There are 3 ways to continue a long LAMMPS simulation. Multiple run commands can be used in the same input script. Each run will continue from where the previous run left off. Or binary restart files can be saved to disk using the restart command. At a later time, these binary files can be read via a read_restart command in a new script. Or they can be converted to text data files and read by a read_data command in a new script. This section discusses the *restart2data* tool that is used to perform the conversion.

Here we give examples of 2 scripts that read either a binary restart file or a converted data file and then issue a new run command to continue where the previous run left off. They illustrate what settings must be made in the new script. Details are discussed in the documentation for the read_restart and read_data commands.

Look at the *in.chain* input script provided in the *bench* directory of the LAMMPS distribution to see the original script that these 2 scripts are based on. If that script had the line

```
restart         50 tmp.restart
```

added to it, it would produce 2 binary restart files (tmp.restart.50 and tmp.restart.100) as it ran.

This script could be used to read the 1st restart file and re−run the last 50 timesteps:

```
read_restart    tmp.restart.50

neighbor        0.4 bin
neigh_modify    every 1 delay 1

fix             1 all nve
fix             2 all langevin 1.0 1.0 10.0 904297

timestep        0.012

run             50
```

Note that the following commands do not need to be repeated because their settings are included in the restart file: *units, atom_style, special_bonds, pair_style, bond_style*. However these commands do need to be used, since their settings are not in the restart file: *neighbor, fix, timestep*.

If you actually use this script to perform a restarted run, you will notice that the thermodynamic data match at step 50 (if you also put a "thermo 50" command in the original script), but do not match at step 100. This is because the fix langevin command uses random numbers in a way that does not allow for perfect restarts.

As an alternate approach, the restart file could be converted to a data file using this tool:

```
restart2data tmp.restart.50 tmp.restart.data
```

Then, this script could be used to re−run the last 50 steps:

```
units           lj
atom_style      bond
pair_style      lj/cut 1.12
```

```
pair_modify     shift yes
bond_style      fene
special_bonds   0.0 1.0 1.0

read_data       tmp.restart.data

neighbor        0.4 bin
neigh_modify    every 1 delay 1

fix             1 all nve
fix             2 all langevin 1.0 1.0 10.0 904297

timestep        0.012

reset_timestep  50
run             50
```

Note that nearly all the settings specified in the original *in.chain* script must be repeated, except the *pair_coeff* and *bond_coeff* commands since the new data file lists the force field coefficients. Also, the reset_timestep command is used to tell LAMMPS the current timestep. This value is stored in restart files, but not in data files.

4.2 2d simulations

Use the dimension command to specify a 2d simulation.

Make the simulation box periodic in z via the boundary command. This is the default.

If using the create box command to define a simulation box, set the z dimensions narrow, but finite, so that the create_atoms command will tile the 3d simulation box with a single z plane of atoms − e.g.

```
create box 1 −10 10 −10 10 −0.25 0.25
```

If using the read data command to read in a file of atom coordinates, set the "zlo zhi" values to be finite but narrow, similar to the create_box command settings just described. For each atom in the file, assign a z coordinate so it falls inside the z−boundaries of the box − e.g. 0.0.

Use the fix enforce2d command as the last defined fix to insure that the z−components of velocities and forces are zeroed out every timestep. The reason to make it the last fix is so that any forces induced by other fixes will be zeroed out.

Many of the example input scripts included in the LAMMPS distribution are for 2d models.

4.3 CHARMM and AMBER force fields

There are many different ways to compute forces in the CHARMM and AMBER molecular dynamics codes, only some of which are available as options in LAMMPS. A force field has 2 parts: the formulas that define it and the coefficients used for a particular system. Here we only discuss formulas implemented in LAMMPS. Setting coefficients is done in the input data file via the read_data command or in the input script with commands like pair_coeff or bond_coeff. See this section for additional tools that can use CHARMM or AMBER to assign force field coefficients and convert their output into LAMMPS input.

These style choices compute force field formulas that are consistent with common options in CHARMM or AMBER. See each command's documentation for the formula it computes.

- bond_style harmonic
- angle_style charmm
- dihedral_style charmm
- pair_style lj/charmm/coul/charmm
- pair_style lj/charmm/coul/charmm/implicit
- pair_style lj/charmm/coul/long

- special_bonds charmm
- special_bonds amber

4.4 Running multiple simulations from one input script

This can be done in several ways. See the documentation for individual commands for more details on how these examples work.

If "multiple simulations" means continue a previous simulation for more timesteps, then you simply use the run command multiple times. For example, this script

```
units lj
atom_style atomic
read_data data.lj
run 10000
run 10000
run 10000
run 10000
run 10000
```

would run 5 successive simulations of the same system for a total of 50,000 timesteps.

If you wish to run totally different simulations, one after the other, the clear command can be used in between them to re−initialize LAMMPS. For example, this script

```
units lj
atom_style atomic
read_data data.lj
run 10000
clear
units lj
atom_style atomic
read_data data.lj.new
run 10000
```

would run 2 independent simulations, one after the other.

For large numbers of independent simulations, you can use variables and the next and jump commands to loop over the same input script multiple times with different settings. For example, this script, named in.polymer

```
variable d index run1 run2 run3 run4 run5 run6 run7 run8
cd $d
```

```
read_data data.polymer
run 10000
cd ..
clear
next d
jump in.polymer
```

would run 8 simulations in different directories, using a data.polymer file in each directory. The same concept could be used to run the same system at 8 different temperatures, using a temperature variable and storing the output in different log and dump files, for example

```
variable a loop 8
variable t index 0.8 0.85 0.9 0.95 1.0 1.05 1.1 1.15
log log.$a
read data.polymer
velocity all create $t 352839
fix 1 all nvt $t $t 100.0
dump 1 all atom 1000 dump.$a
run 100000
next t
next a
jump in.polymer
```

All of the above examples work whether you are running on 1 or multiple processors, but assumed you are running LAMMPS on a single partition of processors. LAMMPS can be run on multiple partitions via the "−partition" command−line switch as described in this section of the manual.

In the last 2 examples, if LAMMPS were run on 3 partitions, the same scripts could be used if the "index" and "loop" variables were replaced with *universe*−style variables, as described in the variable command. Also, the "next t" and "next a" commands would need to be replaced with a single "next a t" command. With these modifications, the 8 simulations of each script would run on the 3 partitions one after the other until all were finished. Initially, 3 simulations would be started simultaneously, one on each partition. When one finished, that partition would then start the 4th simulation, and so forth, until all 8 were completed.

4.5 Parallel tempering

The temper command can be used to perform a parallel tempering or replica−exchange simulation where multiple copies of a simulation are run at different temperatures on different sets of processors, and Monte Carlo temperature swaps are performed between pairs of copies.

Use the −procs and −in command−line switches to launch LAMMPS on multiple partitions.

In your input script, define a set of temperatures, one for each processor partition, using the variable command:

```
variable t proc 300.0 310.0 320.0 330.0
```

Define a fix of style nvt or langevin to control the temperature of each simulation:

```
fix myfix all nvt $t $t 100.0
```

Use the temper command in place of a run command to perform a simulation where tempering exchanges will take place:

```
temper 100000 100 $t myfix 3847 58382
```

4.6 Granular models

To run a simulation of a granular model, you will want to use the following commands:

- atom_style granular
- fix nve/gran
- fix gravity
- thermo_style gran

Use one of these 3 pair potentials:

- pair_style gran/history
- pair_style gran/no_history
- pair_style gran/hertzian

These commands implement fix options specific to granular systems:

- fix freeze
- fix gran/diag
- fix insert
- fix viscous
- fix wall/gran

The fix style *freeze* zeroes both the force and torque of frozen atoms, and should be used for granular system instead of the fix style *setforce*.

For computational efficiency, you can eliminate needless pairwise computations between frozen atoms by using this command:

- neigh_modify exclude

4.7 TIP3P water model

The TIP3P water model as implemented in CHARMM (MacKerell) specifies a 3−site rigid water molecule with charges and Lennard−Jones parameters assigned to each of the 3 atoms. In LAMMPS the fix shake command can be used to hold the two O−H bonds and the H−O−H angle rigid. A bond style of *harmonic* and an angle style of *harmonic* or *charmm* should also be used. These are the additional parameters (in real units) to set for O and H atoms and the water molecule to run a TIP3P model:

O charge = −0.834
H charge = 0.417

O mass = 15.9994
H mass = 1.008

LJ epsilon of O = 0.1521
LJ sigma of O = 3.15057

LJ epsilon of H = 0.046
LJ sigma of H = 0.400014

K of O−H bond = 450
r0 of O−H bond = 0.9572

K of H−O−H angle = 55
theta of H−O−H angle = 104.52

Note: If the LJ epsilon and sigma for H are set to 0.0, it corresponds to the original 1983 TIP3P model (Jorgensen).

4.7 TIP4P water model

The four−point TIP4P water model extends the traditional three−point TIP3P model by adding an additional site, usually massless, where the charge associated with the oxygen atom is placed. This M site is located at a fixed distance away from the oxygen along the bisector of the HOH bond angle.

Two different four−point models can be implemented using the pair styles with *tip4p* in their style name. For both these models, the bond lengths and bond angles should be held fixed using the fix shake command. The following parameters, in real units, should be used to specify the other parameters that match the original TIP4P model (Jorgensen).

O mass = 15.9994
H mass = 1.008

O charge = −1.040
H charge = 0.520

OH bond length = 0.9572
HOH bond angle = 104.52

OM distance = 0.15

LJ epsilon of O−O = 0.1550
LJ sigma of O−O = 3.1526

LJ epsilon, sigma of O−H, H−H = 0.0

An alternate four−point model, the TIP4P−Ew model (Horn), is identical, except for the following parameters:

O charge = −1.084
H charge = 0.5242

OM distance = 0.1250

LJ epsilon of O−O = 0.16275
LJ sigma of O−O = 3.16435

Note that the OM distance is specified in the pair_style command, not as part of the pair coefficients.

(Horn) Horn, Swope, Pitera, Madura, Dick, Hura, and Head−Gordon, J Chem Phys, 120, 9665 (2004).

(MacKerell) MacKerell, Bashford, Bellott, Dunbrack, Evanseck, Field, Fischer, Gao, Guo, Ha, et al, J Phys Chem, 102, 3586 (1998).

(Jorgensen) Jorgensen, Chandrasekhar, Madura, Impey, Klein, J Chem Phys, 79, 926 (1983).

Previous Section − LAMMPS WWW Site − LAMMPS Documentation − LAMMPS Commands − Next Section

5. Example problems

The LAMMPS distribution includes an examples sub−directory with several sample problems. Each problem is in a sub−directory of its own. Most are 2d models so that they run quickly, requiring at most a couple of minutes to run on a desktop machine. Each problem has an input script (in.*) and produces a log file (log.*) and dump file (dump.*) when it runs. Some use a data file (data.*) of initial coordinates as additional input. A few sample log file outputs on different machines and different numbers of processors are included in the directories to compare your answers to. E.g. a log file like log.crack.foo.P means it ran on P processors of machine "foo".

The dump files produced by the example runs can be animated using the xmovie tool described in the Tools section. MPEG versions of most of the xmovie animations are also viewable from the Examples page of the LAMMPS WWW Site.

These are the sample problems in the examples sub−directories:

flow	Couette and Poisseuille flow in a 2d channel
indent	spherical indenter into a 2d solid
micelle	self−assembly of small lipid−like molecules into 2d bilayers
obstacle	flow around two voids in a 2d channel
pour	pouring of granular particles into a 3d box, then

	chute flow
crack	crack propagation in a 2d solid
friction	frictional contact of spherical asperities between 2d surfaces
melt	rapid melt of 3d LJ system
peptide	dynamics of a small solvated peptide chain (5−mer)
shear	sideways shear applied to 2d solid, with and without a void

Here is how you might run and visualize one of the sample problems:

```
cd indent
cp ../../src/lmp_linux .           # copy LAMMPS executable to this dir
lmp_linux <in.indent              # run the problem
```

Running the simulation produces the files *dump.indent* and *log.lammps*. You can visualize the dump file as follows:

```
../../tools/xmovie/xmovie −scale dump.indent
```

Previous Section − LAMMPS WWW Site − LAMMPS Documentation − LAMMPS Commands − Next Section

---

6. Performance &scalability

LAMMPS performance on several prototypical benchmarks and machines is discussed on the Benchmarks page of the LAMMPS WWW Site where CPU timings and parallel efficiencies are listed. Here, the benchmarks are described briefly and some useful rules of thumb about their performance are highlighted.

These are the 5 benchmark problems:

1. LJ = atomic fluid, Lennard−Jones potential with 2.5 sigma cutoff (55 neighbors per atom), NVE integration
2. Chain = bead−spring polymer melt of 100−mer chains, FENE bonds and LJ pairwise interactions with a 2^(1/6) sigma cutoff (5 neighbors per atom), NVE integration
3. EAM = metallic solid, Cu EAM potential with 4.95 Angstrom cutoff (45 neighbors per atom), NVE integration
4. Chute = granular chute flow, frictional history potential with 1.1 sigma cutoff (7 neighbors per atom), NVE integration

5. Rhodo = rhodopsin protein in solvated lipid bilayer, CHARMM force field with a 10 Angstrom LJ cutoff (440 neighbors per atom), particle−particle particle−mesh (PPPM) for long−range Coulombics, NPT integration

The input files for running the benchmarks are included in the LAMMPS distribution, as are sample output files. Each of the 5 problems has 32,000 atoms and runs for 100 timesteps. Each can be run as a serial benchmarks (on one processor) or in parallel. In parallel, each benchmark can be run as a fixed−size or scaled−size problem. For fixed−size benchmarking, the same 32K atom problem is run on various numbers of processors. For scaled−size benchmarking, the model size is increased with the number of processors. E.g. on 8 processors, a 256K−atom problem is run; on 1024 processors, a 32−million atom problem is run, etc.

A useful metric from the benchmarks is the CPU cost per atom per timestep. Since LAMMPS performance scales roughly linearly with problem size and timesteps, the run time of any problem using the same model (atom style, force field, cutoff, etc) can then be estimated. For example, on a 1.7 GHz Pentium desktop machine (Intel icc compiler under Red Hat Linux), the CPU run−time in seconds/atom/timestep for the 5 problems is

Problem:	LJ	Chain	EAM	Chute	Rhodopsin
CPU/atom/step:	4.55E−6	2.18E−6	9.38E−6	2.18E−6	1.11E−4
Ratio to LJ:	1.0	0.48	2.06	0.48	24.5

The ratios mean that if the atomic LJ system has a normalized cost of 1.0, the bead−spring chains and granular systems run 2x faster, while the EAM metal and solvated protein models run 2x and 25x slower respectively. The bulk of these cost differences is due to the expense of computing a particular pairwise force field for a given number of neighbors per atom.

Performance on a parallel machine can also be predicted from the one−processor timings if the parallel efficiency can be estimated. The communication bandwidth and latency of a particular parallel machine affects the efficiency. On most machines LAMMPS will give fixed−size parallel efficiencies on these benchmarks above 50% so long as the atoms/processor count is a few 100 or greater − i.e. on 64 to 128 processors. Likewise, scaled−size parallel efficiencies will typically be 80% or greater up to very large processor counts. The benchmark data on the LAMMPS WWW Site gives specific examples on some different machines, including a run of 3/4 of a billion LJ atoms on 1500 processors that ran at 85% parallel efficiency.

Previous Section − LAMMPS WWW Site − LAMMPS Documentation − LAMMPS Commands − Next Section

---

7. Additional tools

LAMMPS is designed to be a computational kernel for performing molecular dynamics computations. Additional pre− and post−processing steps are often necessary to setup and analyze a simulation. A few additional tools are provided with the LAMMPS distribution and are described in this section.

Our group has also written and released a separate toolkit called Pizza.py which provides tools for doing setup, analysis, plotting, and visualization for LAMMPS simulations. Pizza.py is written in Python and is available for download from the Pizza.py WWW site.

Note that many users write their own setup or analysis tools or use other existing codes and convert their output to a LAMMPS input format or vice versa. The tools listed here are included in the LAMMPS distribution as examples of auxiliary tools. Some of them are not actively supported by Sandia, as they were contributed by LAMMPS users. If you have problems using them, we can direct you to the authors.

The source code for each of these codes is in the tools sub−directory of the LAMMPS distribution. There is a Makefile (which you may need to edit for your platform) which will build several of the tools which reside in that directory. Some of them are larger packages in their own sub−directories with their own Makefiles.

- replicate
- restart2data
- binary2txt
- data2xmovie
- chain
- micelle2d
- xmovie
- ch2lmp
- msi2lmp
- amber2lammps
- lmp2arc
- lmp2cfg
- lmp2traj

---

replicate tool

The file replicate.c takes a LAMMPS data file and replicates it into a larger system. The syntax for running the tool is

```
replicate options <infile > outfile
```

See the top of the replicate.c file for a discussion of the options. This tool is used by some of the LAMMPS benchmarks for creating larger systems to run scaled−size problems on multiple processors.

---

restart2data tool

The file restart2data.cpp converts a binary LAMMPS restart file into an ASCII data file. The syntax for running the tool is

```
restart2data restart−file data−file
```

This tool must be compiled on a platform that can read the binary file created by a LAMMPS run, since binary files are not compatible across all platforms.

Note that a text data file has less precision than a binary restart file. Hence, continuing a run from a converted data file will typically not conform as closely to a previous run as will restarting from a binary restart file.

---

binary2txt tool

The file binary2txt.cpp converts one or more binary LAMMPS dump file into ASCII text files. The syntax for running the tool is

```
binary2txt file1 file2 ...
```

which creates file1.txt, file2.txt, etc. This tool must be compiled on a platform that can read the binary file created by a LAMMPS run, since binary files are not compatible across all platforms.

---

data2xmovie tool

The file data2xmovie.c converts a LAMMPS data file into a snapshot suitable for visualizing with the xmovie tool, as it it had been output with a dump command from LAMMPS itself. The syntax for running the tool is

```
data2xmovie options <infile > outfile
```

See the top of the data2xmovie.c file for a discussion of the options.

---

chain tool

The file chain.f creates a LAMMPS data file containing bead−spring polymer chains and/or monomer solvent atoms. It uses a text file containing chain definition parameters as an input. The created chains and solvent atoms can strongly overlap, so LAMMPS needs to run the system initially with a "soft" pair potential to un−overlap it. The syntax for running the tool is

```
chain <def.chain > data.file
```

See the def.chain or def.chain.ab files in the tools directory for examples of definition files. This tool was used to create the system for the chain benchmark.

---

micelle2d tool

The file micelle2d.f creates a LAMMPS data file containing short lipid chains in a monomer solution. It uses a text file containing lipid definition parameters as an input. The created molecules and solvent atoms can strongly overlap, so LAMMPS needs to run the system initially with a "soft" pair potential to un−overlap it. The syntax for running the tool is

```
micelle2d <def.micelle2d > data.file
```

See the def.micelle2d file in the tools directory for an example of a definition file. This tool was used to create the system for the micelle example.

---

xmovie tool

The xmovie tool is an X−based visualization package that can read LAMMPS dump files and animate them. It is in its own sub−directory with the tools directory. You may need to modify its Makefile so that it can find the appropriate X libraries to link against.

The syntax for running xmovie is

```
xmovie options dump.file1 dump.file2 ...
```

If you just type "xmovie" you will see a list of options. Note that by default, LAMMPS dump files are in scaled coordinates, so you typically need to use the −scale option with xmovie. When xmovie runs it opens a visualization window and a control window. The control options are straightforward to use.

Xmovie was mostly written by Mike Uttormark (U Wisconsin) while he spent a summer at Sandia. It displays 2d projections of a 3d domain. While simple in design, it is an amazingly fast program that can render large numbers of atoms very quickly. It's a useful tool for debugging LAMMPS input and output and making sure your simulation is doing what you think it should. The animations on the Examples page of the LAMMPS WWW site were created with xmovie.

I've lost contact with Mike, so I hope he's comfortable with us distributing his great tool!

---

ch2lmp tool

The ch2lmp sub−directory contains tools for converting files back−and−forth between the CHARMM MD code and LAMMPS.

They are intended to make it easy to use CHARMM as a builder and as a post−processor for LAMMPS. Using charmm2lammps.pl, you can convert an ensemble built in CHARMM into its LAMMPS equivalent. Using lammps2pdb.pl you can convert LAMMPS atom dumps into pdb files.

See the README file in the ch2lmp sub−directory for more information.

These tools were created by Pieter in't Veld (pjintve@sandia.gov) and Paul Crozier (pscrozi@sandia.gov) at Sandia.

---

msi2lmp tool

The msi2lmp sub−directory contains a tool for creating LAMMPS input data files from Accelrys's Insight MD code (formerly MSI/Biosysm and its Discover MD code). See the README file for more information.

This tool was written by John Carpenter (Cray), Michael Peachey (Cray), and Steve Lustig (Dupont). John is now at the Mayo Clinic (jec@mayo.edu), but still fields questions about the tool.

This tool may be out−of−date with respect to the current LAMMPS and Insight versions. Since we don't use it at Sandia, you'll need to experiment with it yourself.

---

amber2lmp tool

The amber2lmp sub−directory contain two Python scripts for converting files back−and−forth between the AMBER MD code and LAMMPS. See the README file in amber2lmp for more information.

These tools were written by Keir Novik while he was at Queen Mary University of London. Keir is no longer there and cannot support these tools which are out−of−date with respect to the current LAMMPS version (and maybe with respect to AMBER as well). Since we don't use these tools at Sandia, you'll need to experiment with them and make necessary modifications yourself.

---

lmp2arc tool

The lmp2arc sub−directory contains a tool for converting LAMMPS output files to the format for Accelrys's Insight MD code (formerly MSI/Biosysm and its Discover MD code). See the README file for more information.

This tool was written by John Carpenter (Cray), Michael Peachey (Cray), and Steve Lustig (Dupont). John is now at the Mayo Clinic (jec@mayo.edu), but still fields questions about the tool.

This tool was updated for the current LAMMPS C++ version by Jeff Greathouse at Sandia (jagreat@sandia.gov).

---

lmp2cfg tool

The lmp2cfg sub−directory contains a tool for converting LAMMPS output files into a series of *.cfg files which can be read into the AtomEye visualizer. See the README file for more information.

This tool was written by Ara Kooser at Sandia (askoose@sandia.gov).

---

lmp2traj tool

The lmp2traj sub−directory contains a tool for converting LAMMPS output files into 3 analysis files. One file can be used to create contour maps of the atom positions over the course of the simulation. The other two files provide density profiles and dipole moments. See the README file for more information.

This tool was written by Ara Kooser at Sandia (askoose@sandia.gov).

Previous Section − LAMMPS WWW Site − LAMMPS Documentation − LAMMPS Commands − Next Section

---

8. Modifying &extending LAMMPS

LAMMPS is designed in a modular fashion so as to be easy to modify and extend with new functionality. In this section, changes and additions users can make are listed along with some minimal instructions. Realistically, the best way to add a new feature is to find a similar feature in LAMMPS and look at the corresponding source and header files to figure out what it does. You will need some knowledge of C++ to be able to understand the hi−level structure of LAMMPS and its class organization, but functions (class methods) that do actual computations are written in vanilla C−style code and operate on simple C−style data structures (vectors and arrays).

Most of the new features described in this section require you to write a new C++ class (except for dump, thermo, and variable options, described below, where you can make small edits to existing files). Creating a new class requires 2 files, a source code file (*.cpp) and a header file (*.h). Their contents are briefly discussed below. Enabling LAMMPS to invoke the new class is as simple as adding two definition lines to the style_user.h file, in the same syntax as the existing LAMMPS classes are defined in the style.h file.

The power of C++ and its object−orientation is that usually, all the code and variables needed to define the new feature are contained in the 2 files you write, and thus shouldn't make the rest of the code more complex or cause side−effect bugs.

Here is a concrete example. Suppose you write 2 files pair_foo.cpp and pair_foo.h that define a new class PairFoo that computes pairwise potentials described in the classic 1997 paper by Foo, et. al. If you wish to invoke those potentials in a LAMMPS input script with a command like

```
pair_style foo 0.1 3.5
```

you simply need to put your 2 files in the LAMMPS src directory, add 2 lines to the style_user.h file, and re−make the code.

The first line added to style_user.h would be

```
PairStyle(foo,PairFoo)
```

in the #ifdef PairClass section, where "foo" is the style keyword in the pair_style command, and PairFoo is the class name in your C++ files.

The 2nd line added to style_user.h would be

```
#include "pair_foo.h"
```

in the #ifdef PairInclude section, where pair_foo.h is the name of your new include file.

When you re−make LAMMPS, your new pairwise potential becomes part of the executable and can be invoked with a pair_style command like the example above. Arguments like 0.1 and 3.5 can be defined and processed by your new class.

Note that if you are using Makefile.list instead of Makefile to build LAMMPS, you will need to explicitly add the names of your new .cpp and .h file to Makefile.list.

Here is a list of the kinds of new features that can be added in this way. The dump and thermo options do not typically require new styles; LAMMPS can simply be recompiled after new code is added to dump_custom.cpp or thermo_custom.cpp.

- Pairwise potentials
- Bond, angle, dihedral, improper potentials
- Dump options
- Thermodynamic output options
- Temperature computation options
- Region geometry options
- Fix options which include integrators, temperature and pressure control, force constraints, boundary conditions, diagnostic output, etc
- Atom options
- Variable options
- New top−level commands

As illustrated by the pairwise example, these options are referred to in the LAMMPS documentation as the "style" of a particular command.

The instructions below for each category will list the header file for the parent class that these styles are sub−classes of. Public variables in that file are ones used and set by the sub−classes which are also used by the parent class. Sometimes they are also used by the rest of LAMMPS. Virtual functions in the header file which are set = 0 are ones you must define in your new class to give it the functionality LAMMPS expects. Virtual functions that are not set to 0 are functions you can optionally define.

Here are some additional guidelines for modifying LAMMPS and adding new functionality:

Think about whether what you want to do would be better as a pre− or post−processing step. Many computations are more easily and more quickly done that way.

Don't do anything within the timestepping of a run that isn't parallel. E.g. don't accumulate a bunch of data on a single processor and analyze it. You run the risk of seriously degrading the parallel efficiency.

If your new feature reads arguments or writes output, make sure you follow the unit conventions discussed by the units command.

If you add something you think is truly useful and doesn't impact LAMMPS performance when it isn't used, send me an email. We might be interested in adding it to the LAMMPS distribution.

Pairwise potentials

All classes that compute pairwise interactions are sub−classes of the Pair class. See the pair.h file for a list of methods this class defines.

Pair_lj_cut.cpp and pair_lj_cut.h are the simplest example of a Pair class. They implement the *lj/cut* style of the pair_style command.

Here is a brief description of the class methods in pair.h:

compute	the workhorse routine that computes the pairwise interactions
settings	reads the input script line with any arguments you define
coeff	set coefficients for one i,j type pair
init_one	perform initialization for one i,j type pair
write &read_restart	write/read i,j pair coeffs to restart files
write &read_restart_settings	write/read global settings to restart files

single	force and energy of a single pairwise interaction between 2 atoms
compute_inner/middle/outer	versions of compute used by rRESPA

The inner/middle/outer routines are optional. Only a few of the pairwise potentials use these in conjunction with rRESPA as set by the run_style command.

Bond, angle, dihedral, improper potentials

All classes that compute molecular interactions are sub−classes of the Bond, Angle, Dihedral, and Improper classes. See the bond.h, angle.h, dihedral.h, and improper.h file for a list of methods these classes defines.

Bond_harmonic.cpp and bond_harmonic.h are the simplest example of a Bond class. Ditto for the harmonic forms of the angle, dihedral, and improper style commands. The bond_harmonic files implement the *harmonic* style of the bond_style command.

Here is a brief description of the class methods in bond.h, angle.h, etc:

compute	the workhorse routine that computes the molecular interactions
coeff	set coefficients for one bond type
equilibrium_distance	length of bond, used by SHAKE
write &read_restart	writes/reads coeffs to restart files
single	force and energy of a single bond

Dump options

There are several classes that print dump files (snapshots of atoms) that are sub−classes of the Dump class. These include the dump_atom.cpp, dump_bond.cpp, and dump_custom.cpp files.

New dump classes can be added, but it is typically simpler to modify the DumpCustom class contained in the dump_custom.cpp file. See the dump command and its *custom* style for a list of what atom information can already be dumped by DumpCustom. If the attribute you want to dump is not in the list, or if you define a new atom style with new attributes (e.g. atoms that store their own magnetic moment), here is how to dump it out in a snapshot file:

Search the dump_custom.cpp and dump_custom.h files for the word "customize". It appears in roughly half a dozen locations. In each of the locations you can add a bit of code that will extend the DumpCustom class to enable it to dump a new quantity. E.g. you will add a keyword, add an if test, add a new small method that packs the requested data into a buffer, etc. For the latter, you can perform a modest amount of computation in this method; see the pack_xs() function for an example.

If desired, a dump custom option can also compute more complicated quantities by invoking a fix that computed quantities at the end of a timestep (should be the same timestep the dump is invoked on). See the ENERGY, CENTRO, and stress options (SXX, SYY, etc) in dump_custom.cpp for examples.

When you re−make LAMMPS, your new option should now be useable via the dump custom command.

Thermodynamic output options

There is only one class that computes and prints thermodynamic information to the screen and log file, although the thermo_style command treats its options as styles.

There are several styles defined in thermo.cpp: "one", "multi", and "granular". There is also a flexible "custom" style which allows you to specify what quantities will be printed each timestep where thermodynamics is computed. See the thermo_style command for a list of pre−defined quantities.

Here is how you can extend the thermo output capabilities. Search the thermo.cpp and thermo.h files for the word "customize" which will tell you where to make these additions. Note that fixes can also print−out thermodynamic quantities via the fix_modify command, so you do not need to modify thermo.cpp to print fix information.

If you want to create a new style (like "one" or "granular") that prints a collection of pre−defined quantities, you add a few lines that define the new style to thermo.cpp. First, add a #DEFINE line at the top of the file which lists the quantities to print. Then add the style name you have chosen to the if test in the constructor to copy the defined string to the line variable.

You can also add new quantities to the custom list. Add your new keyword to the if test in the parse_fields() function where the call to addfield() specifies the text string (8 character max) that will be printed with the quantity, the function that will compute it, and the data type (INT,FLOAT) of the quantity. Then at the bottom of the file, add a function compute_*() which computes the quantity you wish to print. The function assigns the quantity to the variable "dvalue" if it is a floating−point quantity, or to "ivalue" if it is an integer. See the other compute_*() functions for examples of how various quantities can be accessed, computed, summed across processors, normalized as per−atom values, etc. Also, if it makes sense to allow the quantity to be stored in a variable in the input script, add a couple of lines to the compute_value() function that is called when a variable is evaluated. Finally, add a prototype for your new compute method to thermo.h.

Temperature computation options

All classes that compute the temperature of the system are sub−classes of the Temperature class. See the temperature.h file for a list of methods these classes defines. Temperatures are computed by LAMMPS when velocities are set, when thermodynamics are computed, and when temperature is controlled by various thermostats like the fix nvt of fix langevin commands.

Temp_full.cpp and temp_full.h are the simplest example of a Temperature class. They implement the *full* style of the temperature command.

Here is a brief description of the class methods in temperature.h:

init	setup the temperature computation
compute	compute and return temperature

Region geometry options

All classes that define geometric regions are sub−classes of the Region class. See the region.h file for a list of methods these classes defines. Regions are used elsewhere in LAMMPS to group atoms, delete atoms to create a void, insert atoms in a specified region, etc.

Region_sphere.cpp and region_sphere.h are the simplest example of a Region class. They implement the *sphere* style of the region command.

Here is a brief description of the single class method required:

match	determine whether a point is in the region

Fix options

In LAMMPS, a "fix" is any operation that is computed during timestepping that alters some property of the system. Essentially everything that happens during a simulation besides force computation, neighbor list manipulation, and output, is a "fix". This includes time integration (update of velocity and coordinates), force constraints (SHAKE or walls), and diagnostics (compute a diffusion coefficient). See the fix.h file for a list of methods these classes defines.

There are dozens of fix options in LAMMPS; choose one as a template that is similar to what you want to implement. They can be as simple as zeroing out forces (see fix enforce2d which corresponds to the *enforce2d* style) or as complicated as applying SHAKE constraints on bonds and angles (see fix shake which corresponds to the *shake* style) which involves many extra computations.

Here is a brief description of the class methods in fix.h:

setmask	determines when the fix is called during the timestep
init	initialization before a run
setup	called immediately before the 1st timestep
initial_integrate	called at very beginning of each timestep
pre_exchange	called before atom exchange on re−neighboring steps
pre_neighbor	called before neighbor list build
post_force	called after pair &molecular forces are computed
final_integrate	called at end of each timestep
end_of_step	called at very end of timestep
write_restart	dumps fix info to restart file
restart	uses info from restart file to re−initialize the fix
grow_arrays	allocate memory for atom−based arrays used by fix
copy_arrays	copy atom info when an atom migrates to a new processor

memory_usage	report memory used by fix
pack_exchange	store atom's data in a buffer
unpack_exchange	retrieve atom's data from a buffer
pack_restart	store atom's data for writing to restart file
unpack_restart	retrieve atom's data from a restart file buffer
size_restart	size of atom's data
maxsize_restart	max size of atom's data
initial_integrate_respa	same as initial_integrate, but for rRESPA
post_force_respa	same as post_force, but for rRESPA
final_integrate_respa	same as final_integrate, but for rRESPA
pack_comm	pack a buffer to communicate a per−atom quantity
unpack_comm	unpack a buffer to communicate a per−atom quantity
pack_reverse_comm	pack a buffer to reverse communicate a per−atom quantity
unpack_reverse_comm	unpack a buffer to reverse communicate a per−atom quantity
thermo_fields	define quantities for thermodynamic

	output
thermo_compute	compute thermodynamic quantities

Typically, only a small fraction of these methods are defined for a particular fix. Setmask is mandatory, as it determines when the fix will be invoked during the timestep. Fixes that perform time integration (*nve*, *nvt*, *npt*) implement initial_integrate and final_integrate to perform velocity Verlet updates. Fixes that constrain forces implement post_force. Fixes that perform diagnostics typically implement end_of_step.

If the fix needs to store information for each atom that persists from timestep to timestep, it can manage that memory and migrate it with the atoms as they move from processors to processor by implementing the grow_arrays, copy_arrays, pack_exchange, and unpack_exchange methods. Similary, the pack_restart and unpack_restart methods can be implemented to store information about the fix in restart files. If you wish a integrator or force constraint fix to work with rRESPA (see the run_style command), the initial_integrate, post_force_integrate, and final_integrate_respa methods can be implemented. The thermo_fields and thermo_compute methods enable a fix to contribute values to thermodynamic output, as printed quantities and/or to be summed to the potential energy of the system.

Atom options

All classes that define an atom style are sub−classes of the Atom class. See the atom.h file for a list of methods these classes defines. The atom style determines what quantities are associated with an atom in a LAMMPS simulation. If one of the existing atom styles does not define all the arrays you need to store with an atom, then a new atom class can be created.

Atom_atomic.cpp and atom_atomic.h are the simplest example of an Atom class. They implement the *atomic* style of the atom_style command.

Here is a brief description of the class methods in atom.h:

copy	copy info for one atom to another atom's array location
pack_comm	store an atom's info in a buffer communicated every timestep
unpack_comm	retrieve an atom's info from the buffer
pack_reverse	store an atom's info in a buffer communicating partial forces
unpack_reverse	retrieve an atom's info from the buffer

pack_border	store an atom's info in a buffer communicated on neighbor re−builds
unpack_border	retrieve an atom's info from the buffer
pack_exchange	store all an atom's info to migrate to another processor
unpack_exchange	retrieve an atom's info from the buffer

There are also several methods in atom.cpp you will need to augment with information about your new atom class, following the patterns of the other atom styles. These routines are so similar for all classes, that it was simpler to just have one master routine for all classes.

constructor	create style variable and atom array ptrs to NULL
destructor	free memory for atom arrays
set_style	set style variable
check_style	check for pure style vs hybrid style
style2arg	convert style variables to keywords
grow	re−allocate atom arrays to longer lengths
unpack_data	parse atom lines from data file
create_one	create an individual atom of this

	style
size_restart	number of restart quantities associated with proc's atoms
pack_restart	pack atom quantities into a buffer
unpack_restart	unpack atom quantities from a buffer
memory_usage	memory allocated by atom arrays

Variable options

The variable class stores and evaluates input script variables $a, $b, ... $z, as described in this section. *Equal*−style variables are defined by an equation that is evaulated each time the variable is used. The equation can include functions, vectors, keywords, and numbers as described in the variable command. The list of valid functions, vectors, and keywords, can be extended by adding a few lines of code to the evaluate() method at the end of the variable.cpp file. Search for the word "customize" to find the correct locations for adding code.

A new function (e.g. foo(arg1,arg2,...)) can be added in the section that starts with the comment

```
// customize by adding function to this list and to if statement
```

A new vector (e.g. q) can be added in the section that starts with the comment

```
// customize by adding vector to this list and to if statement
```

A new keyword (e.g. mysum) can be added in the section that starts with the comment

```
// customize by adding keyword to this list and to if statement
```

Note that keywords supported by the thermo_style custom command are evaluated by the thermo routines, so do not need to be added to variable.cpp.

New top−level commands

It is possible to add a new command to a LAMMPS input script as opposed to adding a new style to an existing command (atom_style, pair_style, fix, etc). For example the create_atoms, read_data, velocity, and run commands are all top−level LAMMPS commands that are listed in the Command section of style.h. When such a command is encountered in the LAMMPS input script, the topmost level of LAMMPS (lammps.cpp) simply creates a class with the corresponding name, invokes the "command" method of the class, and passes it the arguments from the input script. The command method can perform whatever operations it wishes on the LAMMPS data structures.

Thus to add a new command, you simply need to add a *.cpp and *.h file containing a single class:

command	operations performed by the new command

Of course, the new class can define other methods and variables that it uses internally.

(Foo) Foo, Morefoo, and Maxfoo, J of Classic Potentials, 75, 345 (1997).

Previous Section − LAMMPS WWW Site − LAMMPS Documentation − LAMMPS Commands − Next Section

---

9. Errors

This section describes the various kinds of errors you can encounter when using LAMMPS.

9.1 Common problems

If two LAMMPS runs do not produce the same answer on different machines or different numbers of processors, this is typically not a bug. In theory you should get identical answers on any number of processors and on any machine. In practice, numerical round−off can cause slight differences and eventual divergence of molecular dynamics phase space trajectories within a few 100s or few 1000s of timesteps. However, the statistical properties of the two runs (e.g. average energy or temperature) should still be the same.

If the velocity command is used to set initial atom velocities, a particular atom can be assigned a different velocity when the problem on different machines. Obviously, this means the phase space trajectories of the two simulations will rapidly diverge. See the discussion of the *loop* option in the velocity command for details.

A LAMMPS simulation typically has two stages, setup and run. Most LAMMPS errors are detected at setup time; others like a bond stretching too far may not occur until the middle of a run.

LAMMPS tries to flag errors and print informative error messages so you can fix the problem. Of course LAMMPS cannot figure out your physics mistakes, like choosing too big a timestep, specifying invalid force field coefficients, or putting 2 atoms on top of each other! If you find errors that LAMMPS doesn't catch that you think it should flag, please send us an email.

If you get an error message about an invalid command in your input script, you can determine what command is causing the problem by looking in the log.lammps file or using the echo command to see it on the screen. For a given command, LAMMPS expects certain arguments in a specified order. If you mess this up, LAMMPS will often flag the error, but it may read a bogus argument and assign a value that is valid, but not what you wanted. E.g. trying to read the string "abc" as an integer value and assigning the associated variable a value of 0.

Generally, LAMMPS will print a message to the screen and exit gracefully when it encounters a fatal error. Sometimes it will print a WARNING and continue on; you can decide if the WARNING is important or not. If LAMMPS crashes or hangs without spitting out an error message first then it could be a bug (see this section) or one of the following cases:

LAMMPS runs in the available memory a processor allows to be allocated. Most reasonable MD runs are compute limited, not memory limited, so this shouldn't be a bottleneck on most platforms. Almost all large memory allocations in the code are done via C−style malloc's which will generate an error message if you run out of memory. Smaller chunks of memory are allocated via C++ "new" statements. If you are unlucky you could run out of memory just when one of these small requests is made, in which case the code will crash or hang (in parallel), since LAMMPS doesn't trap on those errors.

Illegal arithmetic can cause LAMMPS to run slow or crash. This is typically due to invalid physics and numerics that your simulation is computing. If you see wild thermodynamic values or NaN values in your LAMMPS output, something is wrong with your simulation.

In parallel, one way LAMMPS can hang is due to how different MPI implementations handle buffering of messages. If the code hangs without an error message, it may be that you need to specify an MPI setting or two (usually via an environment variable) to enable buffering or boost the sizes of messages that can be buffered.

9.2 Reporting bugs

If you are confident that you have found a bug in LAMMPS, we'd like to know about it via email.

First, check the "New features and bug fixes" section of the LAMMPS WWW site to see if the bug has already been reported or fixed.

If not, the most useful thing you can do for us is to isolate the problem. Run it on the smallest number of atoms and fewest number of processors and with the simplest input script that reproduces the bug.

Send an email that describes the problem and any ideas you have as to what is causing it or where in the code the problem might be. We'll request your input script and data files if necessary.

9.3 Error &warning Messages

These are two alphabetic lists of the ERROR and WARNING messages LAMMPS prints out and the reason why. If the explanation here is not sufficient, the documentation for the offending command may help. Grepping the source files for the text of the error message and staring at the source code and comments is also not a bad idea! Note that sometimes the same message can be printed from multiple places in the code.

Errors:

1−3 bond count is inconsistent
: An inconsistency was detected when computing the number of 1−3 neighbors for each atom. This likely means something is wrong with the bond topologies you have defined.

1−4 bond count is inconsistent
: An inconsistency was detected when computing the number of 1−4 neighbors for each atom. This likely means something is wrong with the bond topologies you have defined.

All angle coeffs are not set
: All angle coefficients must be set in the data file or by the angle_coeff command before running a simulation.

All bond coeffs are not set
: All bond coefficients must be set in the data file or by the bond_coeff command before running a simulation.

All EAM pair coeffs are not set
: All EAM pair coefficients must be set in the data file or by the pair_coeff command before running a simulation.

All dihedral coeffs are not set
: All dihedral coefficients must be set in the data file or by the dihedral_coeff command before running a simulation.

All dipole moments are not set
: For atom styles that define dipole moments for each atom type, all moments must be set in the data file or by the dipole command before running a simulation.

All improper coeffs are not set
: All improper coefficients must be set in the data file or by the improper_coeff command before running a simulation.

All masses are not set
: For atom styles that define masses for each atom type, all masses must be set in the data file or by the mass command before running a simulation. They must also be set before using the velocity command.

All pair coeffs are not set
: All pair coefficients must be set in the data file or by the pair_coeff command before running a simulation.

All universe variables must have same # of values
: Self−explanatory.

All variables in next command must be same style
: Self−explanatory.

Angle atoms %d %d %d missing on proc %d at step %d
: One or more of 3 atoms needed to compute a particular angle are missing on this processor. Typically this is because the pairwise cutoff is set too short or the angle has blown apart and an atom is too far away.

Angle atom missing in delete_bonds
: The delete_bonds command cannot find one or more atoms in a particular angle on a particular processor. The pairwise cutoff is too short or the atoms are too far apart to make a valid angle.

Angle atom missing in set command
: The set command cannot find one or more atoms in a particular angle on a particular processor. The pairwise cutoff is too short or the atoms are too far apart to make a valid angle.

Angle coeffs are not set
: No angle coefficients have been assigned in the data file or via the angle_coeff command.

Angle_coeff command before angle_style is defined

Coefficients cannot be set in the data file or via the angle_coeff command until an angle_style has been assigned.

Angle_coeff command before simulation box is defined

The angle_coeff command cannot be used before a read_data, read_restart, or create_box command.

Angle_coeff command when no angles allowed

The chosen atom style does not allow for angles to be defined.

Angles assigned incorrectly

Angles read in from the data file were not assigned correctly to atoms. This means there is something invalid about the topology definitions.

Angles defined but no angle types

The data file header lists angles but no angle types.

Atom count is inconsistent, cannot write restart file

Sum of atoms across processors does not equal initial total count. This is probably because you have lost some atoms.

Atom_modify command after simulation box is defined

The atom_modify command cannot be used after a read_data, read_restart, or create_box command.

Atom_modify command before atom_style command

The atom_modify command cannot be used before an atom style has been defined.

Atom style granular and dpd cannot be used together

Self−explanatory.

Atom style granular must perform 3d simulations

Atom style granular cannot be used with 2d simulations, because the pairwise potentials are inherently 3d.

Atom style hybrid cannot have hybrid as an argument

Self−explanatory. Check the input script.

Atom_style command after simulation box is defined

The atom_style command cannot be used after a read_data, read_restart, or create_box command.

Attempting to rescale a 0.0 temperature

Cannot rescale a temperature that is already 0.0.

Bad FENE bond

Two atoms in a FENE bond have become so far apart that the bond cannot be computed.

Bad grid of processors

The 3d grid of processors defined by the processors command does not match the number of processors LAMMPS is being run on.

Bad principal moments

Fix rigid did not compute the principal moments of inertia of a rigid group of atoms correctly.

Bad slab parameter

Kspace_modify value for the slab/volume keyword must be >= 2.0.

Bitmapped lookup tables require int/float be same size

Cannot use pair tables on this machine, because of word sizes. Use the pair_modify command with table 0 instead.

Bitmapped table is incorrect length in table file

Number of table entries is not a correct power of 2.

Bitmapped table in file does not match requested table

Setting for bitmapped table in pair_coeff command must match table in file exactly.

Bond atom missing in delete_bonds

The delete_bonds command cannot find one or more atoms in a particular bond on a particular processor. The pairwise cutoff is too short or the atoms are too far apart to make a valid bond.

Bond atom missing in set command

The set command cannot find one or more atoms in a particular bond on a particular processor. The pairwise cutoff is too short or the atoms are too far apart to make a valid bond.

Bond atoms %d %d missing on proc %d at step %d
: One or more of 2 atoms needed to compute a particular bond are missing on this processor. Typically this is because the pairwise cutoff is set too short or the bond has blown apart and an atom is too far away.

Bond coeffs are not set
: No bond coefficients have been assigned in the data file or via the bond_coeff command.

Bond coeff for hybrid has invalid style
: Bond style hybrid uses another bond style as one of its coefficients. The bond style used in the bond_coeff command or read from a restart file is not recognized.

Bond_coeff command before bond_style is defined
: Coefficients cannot be set in the data file or via the bond_coeff command until an bond_style has been assigned.

Bond_coeff command before simulation box is defined
: The bond_coeff command cannot be used before a read_data, read_restart, or create_box command.

Bond_coeff command when no bonds allowed
: The chosen atom style does not allow for bonds to be defined.

Bonds assigned incorrectly
: Bonds read in from the data file were not assigned correctly to atoms. This means there is something invalid about the topology definitions.

Bonds defined but no bond types
: The data file header lists bonds but no bond types.

Bond style hybrid cannot have hybrid as an argument
: Self−explanatory. Check the input script.

Bond style hybrid cannot use same bond style twice
: The sub−style arguments of bond_style hybrid cannot be duplicated. Check the input script.

Both sides of boundary must be periodic
: Cannot specify a boundary as periodic only on the lo or hi side. Must be periodic on both sides.

Boundary command after simulation box is defined
The boundary command cannot be used after a read_data, read_restart, or create_box command.
: *Box bounds are invalid*
: The box boundaries specified in the read_data file are invalid. The lo value must be less than the hi value for all 3 dimensions.

Can only wiggle zcylinder wall in z dim
: The Self−explanatory.

Cannot compute PPPM G
: LAMMPS failed to compute a valid approximation for the PPPM g_ewald factor that partitions the computation between real space and k−space.

Cannot compute PPPM X grid spacing
: LAMMPS failed to compute a valid PPPM grid spacing in the x dimension.

Cannot compute PPPM Y grid spacing
: LAMMPS failed to compute a valid PPPM grid spacing in the y dimension.

Cannot compute PPPM Z grid spacing
: LAMMPS failed to compute a valid PPPM grid spacing in the z dimension.

Cannot create atoms with undefined lattice
: Must use the lattice command before using the create_atoms command.

Cannot create_box after simulation box is defined
: The create_box command cannot be used after a read_data, read_restart, or create_box command.

Cannot create_box until atom_style is defined
: Self−explanatory.

Cannot create vels with loop all for non−contiguous atom IDs
: You cannot use the loop all option if you atom IDs do not span 1 to natoms

Cannot evaluate variable equal command

Syntax or keyword names in mathematical expression are not recognized.

Cannot find delete_bonds group ID

Group ID used in the delete_bonds command does not exist.

Cannot find set command group ID

Group ID used in the set command does not exist.

Cannot fix nph on a non−periodic dimension

Pressure can only be controlled on a dimension that is periodic.

Cannot fix npt on a non−periodic dimension

Pressure can only be controlled on a dimension that is periodic.

Cannot fix volume/rescale on a non−periodic boundary

Volume can only be rescaled on a dimension that is periodic.

Cannot fix uniaxial on non−periodic system

Volume can only be rescaled uniaxially if system is periodic in all 3 dimensions.

Cannot open EAM potential file %s

The specified EAM potential file cannot be opened. Check that the path and name are correct.

Cannot open file %s

The specified file cannot be opened. Check that the path and name are correct.

Cannot open fix com file %s

The output file for the fix com command cannot be opened. Check that the path and name are correct.

Cannot open fix gran/diag file %s

The output file for the fix gran/diag command cannot be opened. Check that the path and name are correct.

Cannot open fix msd file %s

The output file for the fix msd command cannot be opened. Check that the path and name are correct.

Cannot open fix rdf file %s

The output file for the fix rdf command cannot be opened. Check that the path and name are correct.

Cannot open fix tmd file %s

The output file for the fix tmd command cannot be opened. Check that the path and name are correct.

Cannot open gzipped file

LAMMPS is attempting to open a gzipped version of the specified file but was unsuccessful. Check that the path and name are correct.

Cannot open pair_write file

The specified output file for pair energies and forces cannot be opened. Check that the path and name are correct.

Cannot open restart file %s

The output restart file cannot be opened. Check that the path and name are correct and that disk space is available.

Cannot read_data after simulation box is defined

The read_data command cannot be used after a read_data, read_restart, or create_box command.

Cannot read_data until atom_style is defined

Self−explanatory.

Cannot read_restart after simulation box is defined

The read_restart command cannot be used after a read_data, read_restart, or create_box command.

Cannot redefine variable as a different style

The variable was already used as a different style variable.

Cannot replicate 2d simulation in z dimension

The replicate command cannot replicate a 2d simulation in the z dimension.

Cannot replicate with fixes that store atom quantities

Either fixes are defined that create and store atom−based vectors or a restart file was read which included atom−based vectors for fixes. The replicate command cannot duplicate that information for new atoms. You should use the replicate command before fixes are applied to the system.

Cannot run 2d simulation with nonperiodic Z dimension
: Use the boundary command to make the z dimension periodic in order to run a 2d simulation.

Cannot set both respa pair and inner/middle/outer
: In the rRESPA integrator, you must compute pairwise potentials either all together (pair), or in pieces (inner/middle/outer). You can't do both.

Cannot set dipole for this atom style
: This atom style does not support dipole settings for each atom type.

Cannot set mass for this atom style
: This atom style does not support mass settings for each atom type. Instead they are defined on a per−atom basis in the data file.

Cannot set respa middle without inner/outer
: In the rRESPA integrator, you must define both a inner and outer setting in order to use a middle setting.

Cannot set these values with this atom style
: Choice of set style does not match attribute of atom style.

Cannot use atom style granular with chosen thermo settings
: Cannot output temperature or pressure with atom style granular.

Cannot use delete_bonds with non−molecular system
: Your choice of atom style does not have bonds.

Cannot use dump bond with non−molecular system
: Your choice of atom style does not have bonds.

Cannot use Ewald with 2d simulation
: The kspace style ewald cannot be used in 2d simulations. You can use 2d Ewald in a 3d simulation; see the kspace_modify command.

Cannot use fix gravity vector with atom style granular
: Self−explanatory.

Cannot use fix rigid with atom style granular
: This fix is not yet enabled for this atom style.

Cannot use fix shake with non−molecular system
: Your choice of atom style does not have bonds.

Cannot use granular potential with pair hybrid
: Granular potentials cannot currently be used in a pair hybrid simulation.

Cannot use multiple long−range potentials with pair hybrid
: Only one sub−style potential with a long−range component can be used with pair_style hybrid.

Cannot use nonperiodic boundaries with Ewald
: For kspace style ewald, all 3 dimensions must have periodic boundaries unless you use the kspace_modify command to define a 2d slab with a non−periodic z dimension.

Cannot use nonperiodic boundaries with PPPM
: For kspace style pppm, All 3 dimensions must have periodic boundaries unless you use the kspace_modify command to define a 2d slab with a non−periodic z dimension.

Cannot use PPPM with 2d simulation
: The kspace style pppm cannot be used in 2d simulations. You can use 2d PPPM in a 3d simulation; see the kspace_modify command.

Cannot use region INF when box does not exist
: Regions that extend to the box boundaries can only be used after the create_box command has been used.

Cannot use vectors in variables unless atom map exists
: Vectors require an atom map to be able to lookup the vector index. Only atom styles with molecular information creat a global map.

Cannot zero momentum for less than 2 atoms

Velocity command is being used with momentum−zeroing options on a group with 0 or 1 atoms.

Command−line variable already exists
: Cannot use the −var command−line option to define the same variable more than once.

Could not create 3d FFT plan
: The FFT setup in pppm failed.

Could not create 3d remap plan
: The FFT setup in pppm failed.

Could not find delete_atoms group ID
: A group ID used in the delete_atoms command does not exist.

Could not find displace_atoms group ID
: A group ID used in the displace_atoms command does not exist.

Cound not find dump_modify ID
: A dump ID used in the dump_modify command does not exist.

Could not find dump group ID
: A group ID used in the dump command does not exist.

Could not find fix group ID
: A group ID used in the fix command does not exist.

Could not find fix rigid group ID
: A group ID used in the fix rigid command does not exist.

Could not find fix_modify ID
: A fix ID used in the fix_modify command does not exist.

Could not find fix_modify temp ID
: A temperature ID used in the fix_modify command does not exist.

Could not find fix spring vector group ID
: Group ID used with fix spring command does not exist.

Could not find temp_modify ID
: A temperature ID used in the temp_modify command does not exist.

Could not find temperature group ID
: A group ID used in the temperature command does not exist.

Could not find thermo_modify temp ID
: A temperature ID used in the thermo_modify command does not exist.

Could not find undump ID
: A dump ID used in the undump command does not exist.

Could not find unfix ID
: A fix ID used in the unfix command does not exist.

Could not find velocity group ID
: A group ID used in the velocity command does not exist.

Could not find velocity temperature ID
: A temperature ID used in the velocity command does not exist.

Could not open dump file
: The output file for the dump command cannot be opened. Check that the path and name are correct.

Could not open input script
: The input script file named in a command−line argument could not be opened.

Could not open log.lammps
: The default LAMMPS log file cannot be opened. Check that the directory you are running in allows for files to be created.

Could not open logfile
: The LAMMPS log file named in a command−line argument cannot be opened. Check that the path and name are correct.

Could not open logfile %s

The LAMMPS log file specified in the input script cannot be opened. Check that the path and name are correct.

Could not open new input file %s

The input script file named in an include or jump command could not be opened. Check that the path and name are correct.

Could not open screen file

The screen file specified as a command−line argument cannot be opened. Check that the directory you are running in allows for files to be created.

Could not open universe log file

For a multi−partition run, the master log file cannot be opened. Check that the directory you are running in allows for files to be created.

Could not open universe screen file

For a multi−partition run, the master screen file cannot be opened. Check that the directory you are running in allows for files to be created.

Create_atoms command before simulation box is defined

The create_atoms command cannot be used before a read_data, read_restart, or create_box command.

Create_atoms region ID does not exist

A region ID used in the create_atoms command does not exist.

Create_box region must be of type inside

The region used in the create_box command must not be an "outside" region. See the region command for details.

Create_box region ID does not exist

A region ID used in the create_box command does not exist.

Delete_atoms command before simulation box is defined

The delete_atoms command cannot be used before a read_data, read_restart, or create_box command.

Delete_atoms overlap not yet implemented

This option is not yet implemented in LAMMPS.

Delete_atoms region ID does not exist

A region ID used in the delete_atoms command does not exist.

Delete_bonds command before simulation box is defined

The delete_bonds command cannot be used before a read_data, read_restart, or create_box command.

Delete_bonds command with no atoms existing

No atoms are yet defined so the delete_bonds command cannot be used.

Did not assign all atoms correctly

Atoms read in from a data file were not assigned correctly to processors. This is likely due to some atom coordinates being outside a non−periodic simulation box.

Did not find keyword in table file

Keyword used in pair_coeff command was not found in table file.

Did not find fix shake partner info

Could not find bond partners implied by fix shake command. This error can be triggered if the delete_bonds command was used before fix shake, and it removed bonds without resetting the 1−2, 1−3, 1−4 weighting list via the special keyword.

Dihedral atoms %d %d %d %d missing on proc %d at step %d

One or more of 4 atoms needed to compute a particular dihedral are missing on this processor. Typically this is because the pairwise cutoff is set too short or the dihedral has blown apart and an atom is too far away.

Dihedral atom missing in delete_bonds

The delete_bonds command cannot find one or more atoms in a particular dihedral on a particular processor. The pairwise cutoff is too short or the atoms are too far apart to make a valid dihedral.

Dihedral atom missing in set command

The set command cannot find one or more atoms in a particular dihedral on a particular processor. The pairwise cutoff is too short or the atoms are too far apart to make a valid dihedral.

Dihedral coeffs are not set

No dihedral coefficients have been assigned in the data file or via the dihedral_coeff command.

Dihedral_coeff command before dihedral_style is defined

Coefficients cannot be set in the data file or via the dihedral_coeff command until an dihedral_style has been assigned.

Dihedral_coeff command before simulation box is defined

The dihedral_coeff command cannot be used before a read_data, read_restart, or create_box command.

Dihedral_coeff command when no dihedrals allowed

The chosen atom style does not allow for dihedrals to be defined.

Dihedrals assigned incorrectly

Dihedrals read in from the data file were not assigned correctly to atoms. This means there is something invalid about the topology definitions.

Dihedrals defined but no dihedral types

The data file header lists dihedrals but no dihedral types.

Dimension command after simulation box is defined

The dimension command cannot be used after a read_data, read_restart, or create_box command.

Dipole command before simulation box is defined

The dipole command cannot be used before a read_data, read_restart, or create_box command.

Displace_atoms command before simulation box is defined

The displace_atoms command cannot be used before a read_data, read_restart, or create_box command.

Dump_modify region ID does not exist

Self−explanatory.

Failed to allocate %d bytes for array %s

Your LAMMPS simulation has run out of memory. You need to run a smaller simulation or on more processors.

Failed to reallocate %d bytes for array %s

Your LAMMPS simulation has run out of memory. You need to run a smaller simulation or on more processors.

Fix command before simulation box is defined

The fix command cannot be used before a read_data, read_restart, or create_box command.

Fix insert region ID does not exist

A region ID used in the fix insert command does not exist.

Fix langevin period must be > 0.0

The time window for temperature relaxation must be > 0

Fix nph periods must be > 0.0

The time window for pressure relaxation must be > 0

Fix npt periods must be > 0.0

The time window for temperature or pressure relaxation must be > 0

Fix nvt period must be > 0.0

The time window for temperature relaxation must be > 0

Fix rdf requires a pair style be defined

Cannot use the rdf fix unless a pair style with a cutoff has been defined.

Fix tmd must come after integration fixes

Any fix tmd command must appear in the input script after all time integration fixes (nve, nvt, npt). See the fix tmd documentation for details.

Fix wall/gran can only be used with granular pair style

Self−explanatory.

Granular pair styles do not use pair_coeff settings
: The pair_coeff command cannot be used with granular force fields.

Gravity must point in −z to use with fix insert
: The fix insert command assumes the theta angle for gravity is 180.0.

Group command before simulation box is defined
: The group command cannot be used before a read_data, read_restart, or create_box command.

Group ID does not exist
: A group ID used in the group command does not exist.

Group region ID does not exist
: A region ID used in the group command does not exist.

Illegal ... command
: Self−explanatory. Check the input script syntax and compare to the documentation for the command.

Invalid $ variable
: The character following a $ in the input script is not between "a" and "z".

Invalid angle style
: The choice of angle style is unknown.

Invalid angle type index for fix shake
: Self−explanatory.

Invalid atom mass for fix shake
: Mass specified in fix shake command must be > 0.0.

Invalid atom style
: The choice of atom style is unknown.

Invalid atom type in neighbor exclusion list
: Atom types must range from 1 to Ntypes inclusive.

Invalid atom type index for fix shake
: Atom types must range from 1 to Ntypes inclusive.

Invalid atom types in fix rdf command
: Atom types must range from 1 to Ntypes inclusive.

Invalid atom types in pair_write command
: Atom types must range from 1 to Ntypes inclusive.

Invalid bond style
: The choice of bond style is unknown.

Invalid bond type index for fix shake
: Self−explanatory. Check the fix shake command in the input script.

Invalid coeffs for this angle style
: Cannot set class 2 coeffs in data file for this angle style.

Invalid coeffs for this dihedral style
: Cannot set class 2 coeffs in data file for this dihedral style.

Invalid coeffs for this improper style
: Cannot set class 2 coeffs in data file for this improper style.

Invalid command−line argument
: One or more command−line arguments is invalid. Check the syntax of the command you are using to launch LAMMPS.

Invalid cutoffs in pair_write command
: Inner cutoff must be larger than 0.0 and less than outer cutoff.

Invalid data file section: Angle Coeffs
: Atom style does not allow angles.

Invalid data file section: AngleAngle Coeffs
: Atom style does not allow impropers.

Invalid data file section: AngleAngleTorsion Coeffs
: Atom style does not allow dihedrals.

Invalid data file section: AngleTorsion Coeffs
: Atom style does not allow dihedrals.

Invalid data file section: Angles
: Atom style does not allow angles.

Invalid data file section: Bond Coeffs
: Atom style does not allow bonds.

Invalid data file section: BondAngle Coeffs
: Atom style does not allow angles.

Invalid data file section: BondBond Coeffs
: Atom style does not allow angles.

Invalid data file section: BondBond13 Coeffs
: Atom style does not allow dihedrals.

Invalid data file section: Bonds
: Atom style does not allow bonds.

Invalid data file section: Dihedral Coeffs
: Atom style does not allow dihedrals.

Invalid data file section: Dihedrals
: Atom style does not allow dihedrals.

Invalid data file section: EndBondTorsion Coeffs
: Atom style does not allow dihedrals.

Invalid data file section: Improper Coeffs
: Atom style does not allow impropers.

Invalid data file section: Impropers
: Atom style does not allow impropers.

Invalid data file section: MiddleBondTorsion Coeffs
: Atom style does not allow dihedrals.

Invalid dihedral style
: The choice of dihedral style is unknown.

Invalid dump frequency
: Dumps frequency must be 1 or greater.

Invalid dump style
: The choice of dump style is unknown.

Invalid group ID in neigh_modify command
: A group ID used in the neigh_modify command does not exist.

Invalid fix style
: The choice of fix style is unknown.

Invalid improper style
: The choice of improper style is unknown.

Invalid keyword in dump custom command
: One or more attribute keywords are not recognized.

Invalid keyword in thermo_style command
: One or more attribute keywords are not recognized.

Invalid keyword in variable equal command
: One or more attribute keywords are not recognized.

Invalid kspace style
: The choice of kspace style is unknown.

Invalid order of forces within respa levels
: For respa, ordering of force computations within respa levels must obey certain rules. E.g. bonds cannot be compute less frequently than angles, pairwise forces cannot be computed less frequently than kspace, etc.

Invalid random number seed in set command

Random number seed must be > 0.

Invalid region style

The choice of region style is unknown.

Invalid style in pair_write command

Self−explanatory. Check the input script.

Invalid temperature style

The choice of temperature style is unknown.

Invalid type for dipole set

Dipole command must set a type from 1−N where N is the number of atom types.

Invalid type for mass set

Mass command must set a type from 1−N where N is the number of atom types.

Invalid type in set command

Type used in set command must be from 1−N where N is the number of atom types (bond types, angle types, etc).

Invalid variable in command−line argument

Command−line arg −var must set a variable from "a" to "z".

Invalid variable in next command

Next command in input script must set variables from "a" to "z".

Invalid variable in variable command

Variable command in input script must set a variable from "a" to "z".

Invalid variable style with next command

Variable styles *equal* and *world* cannot be used in a next command.

Invalid vector in variable equal command

One or more vector names are not recognized.

Improper atoms %d %d %d %d missing on proc %d at step %d

One or more of 4 atoms needed to compute a particular improper are missing on this processor. Typically this is because the pairwise cutoff is set too short or the improper has blown apart and an atom is too far away.

Improper atom missing in delete_bonds

The delete_bonds command cannot find one or more atoms in a particular improper on a particular processor. The pairwise cutoff is too short or the atoms are too far apart to make a valid improper.

Improper atom missing in set command

The set command cannot find one or more atoms in a particular improper on a particular processor. The pairwise cutoff is too short or the atoms are too far apart to make a valid improper.

Improper coeffs are not set

No improper coefficients have been assigned in the data file or via the improper_coeff command.

Improper_coeff command before improper_style is defined

Coefficients cannot be set in the data file or via the improper_coeff command until an improper_style has been assigned.

Improper_coeff command before simulation box is defined

The improper_coeff command cannot be used before a read_data, read_restart, or create_box command.

Improper_coeff command when no impropers allowed

The chosen atom style does not allow for impropers to be defined.

Impropers assigned incorrectly

Impropers read in from the data file were not assigned correctly to atoms. This means there is something invalid about the topology definitions.

Impropers defined but no improper types

The data file header lists improper but no improper types.

Inconsistent dipole settings for some atoms

Dipole moment must be 0 for non−dipole type atoms. Dipole moment must be set for dipole type atoms.

Incorrect args for angle coefficients
: Self−explanatory. Check the input script or data file.

Incorrect args for bond coefficients
: Self−explanatory. Check the input script or data file.

Incorrect args for dihedral coefficients
: Self−explanatory. Check the input script or data file.

Incorrect args for improper coefficients
: Self−explanatory. Check the input script or data file.

Incorrect args for pair coefficients
: Self−explanatory. Check the input script or data file.

Incorrect atom format in data file
: Number of values per atom line in the data file is not consistent with the atom style.

Incorrect boundaries with slab Ewald
: Must have periodic x,y dimensions and non−periodic z dimension to use 2d slab option with Ewald.

Incorrect boundaries with slab PPPM
: Must have periodic x,y dimensions and non−periodic z dimension to use 2d slab option with PPPM.

Incorrect format in TMD target file
: Format of file read by fix tmd command is incorrect.

Incorrect multiplicity arg for dihedral coefficients
: Self−explanatory. Check the input script or data file.

Incorrect sign arg for dihedral coefficients
: Self−explanatory. Check the input script or data file.

Incorrect weight arg for dihedral coefficients
: Self−explanatory. Check the input script or data file.

Insertion region extends outside simulation box
: Region specified with fix insert command extends outside the global simulation box.

Insufficient Jacobi rotations for rigid body
: Eigensolve for rigid body was not sufficiently accurate.

Invalid dump_modify threshhold operator
: Operator keyword used for threshhold specification in not recognized.

Invalid flag in header of restart file
: Value read from beginning of restart file is not recognized.

Invalid keyword in pair table parameters
: Keyword used in list of table parameters is not recognized.

Invalid pair style
: The choice of pair style is unknown.

Invalid pair table cutoff
: Cutoffs in pair_coeff command are not valid with read−in pair table.

Invalid pair table length
: Length of read−in pair table is invalid
: Value read from beginning of restart file is not recognized.

KSpace style has not yet been set
: Cannot use kspace_modify command until a kspace style is set.

KSpace style is incompatible with Pair style
: Setting a kspace style requires that a pair style with a long−range Coulombic component be selected.

Label wasn't found in input script
: Self−explanatory.

Lattice style incompatible with dimension
: 2d simulation can use sq, sq2, or hex lattice. 3d simulation can use sc, bcc, or fcc lattice.

Lost atoms via displacement: original %.15g current %.15g
: Moving atoms via the displace_atoms command lost one or more atoms.

Lost atoms: original %.15g current %.15g
: A thermodynamic computation has detected lost atoms.

Marsaglia RNG cannot use 0 seed
: The random number generator use for the fix langevin command cannot use 0 as an initial seed.

Mass command before simulation box is defined
: The mass command cannot be used before a read_data, read_restart, or create_box command.

Minimize command before simulation box is defined
: The minimize command cannot be used before a read_data, read_restart, or create_box command.

Min_style command before simulation box is defined
: The min_style command cannot be used before a read_data, read_restart, or create_box command.

More than one freeze fix
: You can only define one freeze fix.

More than one shake fix
: You can only define one SHAKE fix.

Must define angle_style before Angle Coeffs
: Must use an angle_style command before reading a data file that defines Angle Coeffs.

Must define angle_style before BondAngle Coeffs
: Must use an angle_style command before reading a data file that defines Angle Coeffs.

Must define angle_style before BondBond Coeffs
: Must use an angle_style command before reading a data file that defines Angle Coeffs.

Must define bond_style before Bond Coeffs
: Must use a bond_style command before reading a data file that defines Bond Coeffs.

Must define dihedral_style before AngleAngleTorsion Coeffs
: Must use a dihedral_style command before reading a data file that defines AngleAngleTorsion Coeffs.

Must define dihedral_style before AngleTorsion Coeffs
: Must use a dihedral_style command before reading a data file that defines AngleTorsion Coeffs.

Must define dihedral_style before BondBond13 Coeffs
: Must use a dihedral_style command before reading a data file that defines BondBond13 Coeffs.

Must define dihedral_style before Dihedral Coeffs
: Must use a dihedral_style command before reading a data file that defines Dihedral Coeffs.

Must define dihedral_style before EndBondTorsion Coeffs
: Must use a dihedral_style command before reading a data file that defines EndBondTorsion Coeffs.

Must define dihedral_style before MiddleBondTorsion Coeffs
: Must use a dihedral_style command before reading a data file that defines MiddleBondTorsion Coeffs.

Must define improper_style before AngleAngle Coeffs
: Must use an improper_style command before reading a data file that defines AngleAngle Coeffs.

Must define improper_style before Improper Coeffs
: Must use an improper_style command before reading a data file that defines Improper Coeffs.

Must define pair_style before Pair Coeffs
: Must use a pair_style command before reading a data file that defines Pair Coeffs.

Must have more than one processor partition to temper
: Cannot use the temper command with only one processor partition. Use the −partition command−line option.

Must read Atoms before Angles
: The Atoms section of a data file must come before an Angles section.

Must read Atoms before Bonds
: The Atoms section of a data file must come before a Bonds section.

Must read Atoms before Dihedrals

The Atoms section of a data file must come before a Dihedrals section.

Must read Atoms before Impropers

The Atoms section of a data file must come before an Impropers section.

Must read Atoms before Velocities

The Atoms section of a data file must come before a Velocities section.

Must set both respa inner and outer

Cannot use just the inner or outer option with repsa without using the other.

Must specify a region in fix insert

Self−explanatory.

Must use −in switch with multiple partitions

A multi−partition simulation cannot read the input script from stdin. The −in command−line option must be used to specify a file.

Must use a block or cylinder region with fix insert

Self−explanatory.

Must use a molecular atom style with fix rigid molecule

Self−explanatory.

Must use molecular atom style with neigh_modify exclude molecule

Self−explanatory.

Must use a z−axis cylinder with fix insert

The axis of the cylinder region used with the fix insert command must be oriented along the z dimension.

Must use atom style dpd with pair style dpd

Self−explanatory.

Must use atom style granular with lj units

Self−explanatory.

Must use atom style granular with pair style granular

Self−explanatory.

Must use atom style granular with chosen thermo settings

If granular thermo info is to be output, must use atom style granular.

Must use atom style granular with granular thermo output

If atom style is granular, must use thermo style granular or custom.

Must use charged atom style with fix efield

The atom style being used does not allow atoms to have assigned charges. Hence it will not work with this fix which generates a force due to an E−field acting on charge.

Must use charged atom style with this pair style

The atom style being used does not allow atoms to have assigned charges. Hence it will not work with this choice of pair style.

Must use fix freeze with atom style granular

Self−explanatory.

Must use fix gran/diag with atom style granular

Self−explanatory.

Must use fix gran/diag with granular pair style

Self−explanatory.

Must use fix gravity chute with atom style granular

Self−explanatory.

Must use fix gravity spherical with atom style granular

Self−explanatory.

Must use fix gravity gradient with atom style granular

Self−explanatory.

Must use fix gravity with fix insert

Insertion of granular particles must be done under the influence of gravity.

Must use fix insert with atom style granular
: Self−explanatory.

Must use fix nve/gran with atom style granular
: Self−explanatory.

Must use fix wall/gran with atom style granular
: Self−explanatory.

Must use region with side = in with fix insert
: Self−explanatory.

Needed topology not in data file
: The header of the data file indicated that bonds or angles or dihedrals or impropers would be included, but they were not present.

Neighbor delay must be 0 or multiple of every setting
: The delay and every parameters set via the neigh_modify command are inconsistent. If the delay setting is non−zero, then it must be a multiple of the every setting.

Neighbor list overflow, boost neigh_modify one or page
: There are too many neighbors of a single atom. Use the neigh_modify command to increase the neighbor page size and the max number of neighbors allowed for one atom.

Newton bond change after simulation box is defined
: The newton command cannot be used to change the newton bond value after a read_data, read_restart, or create_box command.

No angles allowed with this atom style
: Self−explanatory. Check data file.

No atoms in data file
: The header of the data file indicated that atoms would be included, but they were not present.

No atoms to compute diffusion for
: The fix msd command has no atoms to compute on.

No bonds allowed with this atom style
: Self−explanatory. Check data file.

No dihedrals allowed with this atom style
: Self−explanatory. Check data file.

No dump custom arguments specified
: The dump custom command requires that atom quantities be specified to output to dump file.

No impropers allowed with this atom style
: Self−explanatory. Check data file.

No rigid bodies defined by fix rigid
: Self−explanatory.

Non integer # of swaps in temper command
: Swap frequency in temper command must evenly divide the total # of timesteps.

Non−orthogonal lattice vectors
: Self−explanatory.

One or zero atoms in rigid body
: Any rigid body defined by the fix rigid command must contain 2 or more atoms.

One or more atoms belong to multiple rigid bodies
: Two or more rigid bodies defined by the fix rigid command cannot contain the same atom.

Orientation vectors are not right−handed
: The 3 vectors defined by the orient command must form a right−handed coordinate system.

Out of range atoms − cannot compute PPPM
: One or more atoms are attempting to map their charge to a PPPM grid point that is not owned by a processor. This is usually because an atom has moved to far in a single timestep.

Pair distance < table inner cutoff
: Two atoms are closer together than the pairwise table allows.

Pair distance > table outer cutoff
: Two atoms are further apart than the pairwise table allows.

Pair table parameters did not set N
: List of pair table parameters must include N setting.

PPPM order cannot be greater than %d
: Self−explanatory.

PPPM stencil extends too far, reduce PPPM order
: The grid points that atom charge are mapped to cannot extend further than one neighbor processor away. Reducing the PPPM order via the kspace_modify command will reduce the stencil distance.

Pair coeff for hybrid has invalid style
: Style in pair coeff must have been listed in pair_style command.

Pair cutoff < Respa interior cutoff
: One or more pairwise cutoffs are too short to use with the specified rRESPA cutoffs.

Pair style hybrid cannot have hybrid as an argument
: Self−explanatory. Check the input script.

Pair style hybrid cannot use same pair style twice
: The sub−style arguments of pair_style hybrid cannot be duplicated. Check the input script.

Pair inner cutoff < Respa interior cutoff
: One or more pairwise cutoffs are too short to use with the specified rRESPA cutoffs.

Pair inner cutoff >= Pair outer cutoff
: The specified cutoffs for the pair style are inconsistent.

Pair style is incompatible with DihedralCharmm
: When using a dihedral style charmm, a pair style with a CHARMM component must also be selected, so that 1−4 pairwise coefficients are specified.

Pair style is incompatible with KSpace style
: If a pair style with a long−range Coulombic component is selected, then a kspace style must also be used.

Pair table cutoffs must all be equal to use with KSpace
: When using pair style table with a long−range KSpace solver, the cutoffs for all atom type pairs must all be the same, since the long−range solver starts at that cutoff.

Pair_coeff command before pair_style is defined
: Self−explanatory.

Pair_coeff command before simulation box is defined
: The pair_coeff command cannot be used before a read_data, read_restart, or create_box command.

Pair_modify command before pair_style is defined
: Self−explanatory.

Pair_write command before pair_style is defined
: Self−explanatory.

Potential with shear history requires newton pair off
: Granular potentials that include shear history effects can only be run with a newton setting where pairwise newton is "off".

Proc grid in z != 1 for 2d simulation
: There cannot be more than 1 processor in the z dimension of a 2d simulation.

Processor partitions are inconsistent
: The total number of processors in all partitions must match the number of processors LAMMPS is running on.

Processors command after simulation box is defined
: The processors command cannot be used after a read_data, read_restart, or create_box command.

Quaternion creation numeric error
: A numeric error occurred in the creation of a rigid body by the fix rigid command.

Quotes in a single arg

A single word should not be quoted in the input script; only a set of words with intervening spaces should be quoted.

R0 < 0 for fix spring command
: Equilibrium spring length is invalid.

Region union region ID does not exist
: One or more of the region IDs specified by the region union command does not exist.

Replacing a fix, but new style != old style
: A fix ID can be used a 2nd time, but only if the style matches the previous fix. In this case it is assumed you with to reset a fix's parameters. This error may mean you are mistakenly re−using a fix ID when you do not intend to.

Replicate command before simulation box is defined
: The replicate command cannot be used before a read_data, read_restart, or create_box command.

Replicate did not assign all atoms correctly
: Atoms replicated by the replicate command were not assigned correctly to processors. This is likely due to some atom coordinates being outside a non−periodic simulation box.

Requested atom types in EAM setfl file do not exist
: Atom type specified in pair_style eam command does not match number of types in setfl potential file.

Respa inner cutoffs are invalid
: The first cutoff must be <= the second cutoff.

Respa inner/middle/outer used with invalid pair style
: Only a few pair potentials support the use of respa inner, middle, outer options.

Respa levels must be >= 1
: Self−explanatory.

Respa middle cutoffs are invalid
: The first cutoff must be <= the second cutoff.

Respa not allowed with atom style granular
: Respa cannot be used with the granular atom style.

Reuse of dump ID
: A dump ID cannot be used twice.

Reuse of region ID
: A region ID cannot be used twice.

Reuse of temperature ID
: A temperature ID cannot be used twice.

Rigid fix must come before NPT/NPH fix
: NPT fix must be defined in input script after all rigid fixes, else the rigid fix contribution to the pressure virial is incorrect.

Run command before simulation box is defined
: The run command cannot be used before a read_data, read_restart, or create_box command.

Run_style command before simulation box is defined
: The run_style command cannot be used before a read_data, read_restart, or create_box command.

Set command before simulation box is defined
: The set command cannot be used before a read_data, read_restart, or create_box command.

Set command with no atoms existing
: No atoms are yet defined so the set command cannot be used.

Shake angles have different bond types
: All 3−atom angle−constrained SHAKE clusters specified by the fix shake command that are the same angle type, must also have the same bond types for the 2 bonds in the angle.

Shake atoms %d %d %d %d missing on proc %d at step %d
: The 4 atoms in a single shake cluster specified by the fix shake command are not all accessible to a processor. This probably means an atom has moved too far.

Shake atoms %d %d %d missing on proc %d at step %d
: The 3 atoms in a single shake cluster specified by the fix shake command are not all accessible to a processor. This probably means an atom has moved too far.

Shake atoms %d %d missing on proc %d at step %d
: The 2 atoms in a single shake cluster specified by the fix shake command are not all accessible to a processor. This probably means an atom has moved too far.

Shake cluster of more than 4 atoms
: A single cluster specified by the fix shake command can have no more than 4 atoms.

Shake clusters are connected
: A single cluster specified by the fix shake command must have a single central atom with up to 3 other atoms bonded to it.

Shake determinant = 0.0
: The determinant of the matrix being solved for a single cluster specified by the fix shake command is numerically invalid.

Shake fix must come before NPT/NPH fix
: NPT fix must be defined in input script after SHAKE fix, else the SHAKE fix contribution to the pressure virial is incorrect.

Substitution for undefined variable
: The variable specified with a $ symbol in an input script command has not been previously defined with a variable command.

Temper command before simulation box is defined
: The temper command cannot be used before a read_data, read_restart, or create_box command.

Tempering fix ID is not defined
: The fix ID specified by the temper command does not exist.

Tempering fix is not valid
: The fix specified by the temper command is not one that controls temperature (nvt or langevin).

Thermodynamics not computed on tempering swap steps
: The thermo command must insure that thermodynamics (including energy) is computed on the timesteps that tempering swaps are attempted.

Thermodynamics must compute PE for temper
: The thermo style must insure that thermodynamics computations include potential energy when tempering is performed.

Thermo_style command before simulation box is defined
: The thermo_style command cannot be used before a read_data, read_restart, or create_box command.

TMD target file did not list all group atoms
: The target file for the fix tmd command did not list all atoms in the fix group.

Too big a problem to run with a molecular atom style
: Cannot run a problem with > 2^31 atoms with molecular attributes.

Too few bits for lookup table
: Table size specified via pair_modify command does not work with your machine's floating point representation.

Too large an atom type in create_atoms command
: The atoms to be created by the create_atoms command must have a valid type.

Too many atoms in data file
: A data file cannot contain more than 2^31 atoms.

Too many atoms to use delete atoms command
: Cannot use delete_atoms command if number of atoms is greater than 2^31.

Too many atoms to use velocity create with loop all
: Cannot use velocity create command with loop all setting if number of atoms is greater than 2^31. Switch to local or geom setting.

Too many exponent bits for lookup table

Table size specified via pair_modify command does not work with your machine's floating point representation.

Too many mantissa bits for lookup table

Table size specified via pair_modify command does not work with your machine's floating point representation.

Too many groups

The maximum number of atom groups (including the "all" group) is given by MAX_GROUP in group.cpp and is 32.

Too many masses for fix shake

The fix shake command cannot list more masses than there are atom types.

Too many total bits for bitmapped lookup table

Table size specified via pair_modify command is too large. Note that a value of N generates a 2^N size table.

Too many touching neighbors − boost MAXTOUCH

A granular simulation has too many neighbors touching one atom. The MAXTOUCH parameter in fix_shear_history.cpp must be set larger and LAMMPS must be re−built.

Unbalanced quotes in input line

No matching end double quote was found following a leading double quote.

Unexpected end of data file

LAMMPS hit the end of the data file while attempting to read a section. Something is wrong with the format of the data file.

Units command after simulation box is defined

The units command cannot be used after a read_data, read_restart, or create_box command.

Unknown atom style in restart file

The atom style stored in the restart file is not recognized by LAMMPS.

Unknown command: %s

The command is not known to LAMMPS. Check the input script.

Unknown identifier in data file: %s

A section of the data file cannot be read by LAMMPS.

Unknown section in data file: %s

The keyword for a section of the data file is not recognized by LAMMPS.

Unknown table style in pair_style command

Style of table is invalid for use with pair_style table command.

Universe variable count < # of partitions

A world−style variable must specify a number of values >= to the number of processor partitions.

Use of displace_atoms with undefined lattice

Must use lattice command with displace_atoms command if units option is set to lattice.

Use of fix indent with undefined lattice

The lattice command must be used to define a lattice before using the fix indent command.

Use of region with undefined lattice

If scale = lattice (the default) for the region command, then a lattice must first be defined via the lattice command.

Use of temperature ramp with undefined lattice

If scale = lattice (the default) for the temperature ramp command, then a lattice must first be defined via the lattice command.

Use of velocity with undefined lattice

If scale = lattice (the default) for the velocity set or velocity ramp command, then a lattice must first be defined via the lattice command.

Using variable equal keyword before simulation box is defined

Cannot use simulation domain keywords in a equal style variable definition until the simulation box has been defined.

Using variable equal keyword before initial run
: Cannot use thermodynamic keywords in a equal style variable definition until a simulation run has been performed.

Velocity command before simulation box is defined
: The velocity command cannot be used before a read_data, read_restart, or create_box command.

Velocity command with no atoms existing
: A velocity command has been used, but no atoms yet exist.

Velocity ramp in z for a 2d problem
: Self−explanatory.

World variable count doesn't match # of partitions
: A world−style variable must specify a number of values equal to the number of processor partitions.

Write_restart command before simulation box is defined
: The write_restart command cannot be used before a read_data, read_restart, or create_box command.

Warnings:

FENE bond too long: %d %g
: A FENE bond has stretched dangerously far. It's interaction strength will be truncated to attempt to prevent the bond from blowing up.

FENE bond too long: %d %d %d %g
: A FENE bond has stretched dangerously far. It's interaction strength will be truncated to attempt to prevent the bond from blowing up.

Group for fix_modify temp != fix group
: The fix_modify command is specifying a temperature computation that computes a temperature on a different group of atoms than the fix itself operates on. This is probably not what you want to do.

Less insertions than requested
: Less atom insertions occurred on this timestep due to the fix insert command than were scheduled. This is probably because there were too many overlaps detected.

Lost atoms: original %.15g current %.15g
: A thermodynamic computation has detected lost atoms.

Mismatch between velocity and temperature groups
: The temperature computation used by the velocity command will not be on the same group of atoms that velocities are being set for. This is probably not what you want.

More than one dump custom with a centro attribute
: Each dump custom command that uses a per−atom centro attribute will cause a full neighbor list to be built and looped over. Thus it may be inefficient to use this attribute in multiple dump custom commands.

More than one dump custom with a stress attribute
: Each dump custom command that uses a per−atom stress tensor attribute will cause the neighbor list to be looped over and inter−processor communication to be performed. Thus it may be inefficient to use these attributes in multiple dump custom commands.

More than one dump custom with an energy attribute
: Each dump custom command that uses a per−atom energy attribute will cause the neighbor list to be looped over and inter−processor communication to be performed. Thus it may be inefficient to use this attribute in multiple dump custom commands.

More than one msd fix
: This will be computationally inefficient.

More than one rigid fix
: This will be computationally inefficient.

No fixes defined, atoms won't move

If you are not using a fix like nve, nvt, npt then atom velocities and coordinates will not be updated during timestepping.

One or more respa levels compute no forces

This is computationally inefficient.

Replacing a fix, but new group != old group

The ID and style of a fix match for a fix you are changing with a fix command, but the new group you are specifying does not match the old group.

Replicating in a non−periodic dimension

The parameters for a replicate command will cause a non−periodic dimension to be replicated; this may cause unwanted behavior.

Resetting angle_style to restart file value

The angle style defined in the LAMMPS input script does not match that of the restart file.

Resetting bond_style to restart file value

The bond style defined in the LAMMPS input script does not match that of the restart file.

Resetting boundary settings to restart file values

The boundary settings defined in the LAMMPS input script do not match that of the restart file.

Resetting dihedral_style to restart file value

The dihedral style defined in the LAMMPS input script does not match that of the restart file.

Resetting dimension to restart file value

The dimension value defined in the LAMMPS input script does not match that of the restart file.

Resetting improper_style to restart file value

The improper style defined in the LAMMPS input script does not match that of the restart file.

Resetting newton bond to restart file value

The value of the newton setting for bonds defined in the LAMMPS input script does not match that of the restart file.

Resetting pair_style to restart file value

The pair style defined in the LAMMPS input script does not match that of the restart file.

Resetting reneighboring criteria during minimization

Minimization requires that neigh_modify settings be delay = 0, every = 1, check = yes. Since these settings were not in place, LAMMPS changed them and will restore them to their original values after the minimization.

Resetting unit_style to restart file value

The unit style defined in the LAMMPS input script does not match that of the restart file.

Restart file used different # of processors

The restart file was written out by a LAMMPS simulation running on a different number of processors. Due to round−off, the trajectories of your restarted simulation may diverge a little more quickly than if you ran on the same # of processors.

Restart file used different 3d processor grid

The restart file was written out by a LAMMPS simulation running on a different 3d grid of processors. Due to round−off, the trajectories of your restarted simulation may diverge a little more quickly than if you ran on the same # of processors.

Restart file used different newton pair setting

The restart file was written out by a LAMMPS simulation running with a different value of the newton pair setting. The new simulation will use the value from the input script.

Restart file version does not match LAMMPS version

The version of LAMMPS that wrote the restart file does not match the version of LAMMPS that is reading the restart file. Generally this shouldn't be a problem, since restart file formats won't change very often if at all. But if they do, the code will probably crash trying to read the file. Versions of LAMMPS are specified by a date.

Shake determinant < 0.0

The determinant of the quadratic equation being solved for a single cluster specified by the fix shake command is numerically suspect. LAMMPS will set it to 0.0 and continue.

System is not charge neutral, net charge = %g

The total charge on all atoms on the system is not 0.0, which is not valid for Ewald or PPPM.

Table inner cutoff >= outer cutoff

You specified an inner cutoff for a Coulombic table that is longer than the global cutoff. Probably not what you wanted.

Temperature for NPH pressure is not for group all

User−assigned temperature to NPH fix does not compute temperature for all atoms. Since NPH computes a global pressure, the kinetic energy contribution from the temperature is assumed to also be for all atoms. Thus the pressure used by NPH could be inaccurate.

Temperature for NPT pressure is not for group all

User−assigned temperature to NPT fix does not compute temperature for all atoms. Since NPT computes a global pressure, the kinetic energy contribution from the temperature is assumed to also be for all atoms. Thus the pressure used by NPT could be inaccurate.

Using variable equal keyword with non−current thermo

The variable expression is being evaluated with a thermodynamic quantity on a timestep when thermodynamic information may not be current.

Previous Section − LAMMPS WWW Site − LAMMPS Documentation − LAMMPS Commands − Next Section

10. Future and history

This section lists features we are planning to add to LAMMPS, features of previous versions of LAMMPS, and features of other parallel molecular dynamics codes I've distributed.

10.1 Coming attractions

The current version of LAMMPS incorporates nearly all the features from previous parallel MD codes I developed. These include earlier versions of LAMMPS itself, Warp and ParaDyn for metals, and GranFlow for granular materials.

These are new features I'd like to eventually add to LAMMPS. Some are being worked on; some haven't been implemented because of lack of time or interest; others are just a lot of work!

- Monte Carlo bond−swapping for polymers (was in Fortran LAMMPS)
- torsional shear boundary conditions and temperature calculation
- deletion of created atoms that overlap
- bond breaking and creation potentials
- point dipole force fields
- 3−body force fields for materials like Si or silica
- modified EAM (MEAM) potentials for metals
- Brownian dynamics
- pressure and energy tail corrections for pairwise interactions
- Parinello−Rahman non−rectilinear simulation box

10.2 Past versions

LAMMPS development began in the mid 1990s under a cooperative research &development agreement (CRADA) between two DOE labs (Sandia and LLNL) and 3 companies (Cray, Bristol Myers Squibb, and Dupont). Soon after the CRADA ended, a final F77 version of the code, LAMMPS 99, was released. As development of LAMMPS continued at Sandia, the memory management in the code was converted to F90; a final F90 version was released as LAMMPS 2001.

The current LAMMPS is a rewrite in C++ and was first publicly released in 2004. It includes many new features, including features from other parallel molecular dynamics codes written at Sandia, namely ParaDyn, Warp, and GranFlow. ParaDyn is a parallel implementation of the popular serial DYNAMO code developed by Stephen Foiles and Murray Daw for their embedded atom method (EAM) metal potentials. ParaDyn uses atom− and force−decomposition algorithms to run in parallel. Warp is also a parallel implementation of the EAM potentials designed for large problems, with boundary conditions specific to shearing solids in varying geometries. GranFlow is a granular materials code with potentials and boundary conditions peculiar to granular systems. All of these codes (except ParaDyn) use spatial−decomposition techniques for their parallelism.

These older codes are available for download from the LAMMPS WWW site, except for Warp &GranFlow which were primarily used internally. A brief listing of their features is given here.

LAMMPS 2001

- F90 + MPI
- dynamic memory
- spatial−decomposition parallelism
- NVE, NVT, NPT, NPH, rRESPA integrators
- LJ and Coulombic pairwise force fields
- all−atom, united−atom, bead−spring polymer force fields
- CHARMM−compatible force fields
- class 2 force fields
- 3d/2d Ewald &PPPM
- various force and temperature constraints
- SHAKE
- Hessian−free truncated−Newton minimizer
- user−defined diagnostics

LAMMPS 99

- F77 + MPI
- static memory allocation
- spatial−decomposition parallelism
- most of the LAMMPS 2001 features with a few exceptions
- no 2d Ewald &PPPM
- molecular force fields are missing a few CHARMM terms
- no SHAKE

Warp

- F90 + MPI
- spatial−decomposition parallelism

- embedded atom method (EAM) metal potentials + LJ
- lattice and grain−boundary atom creation
- NVE, NVT integrators
- boundary conditions for applying shear stresses
- temperature controls for actively sheared systems
- per−atom energy and centro−symmetry computation and output

ParaDyn

- F77 + MPI
- atom− and force−decomposition parallelism
- embedded atom method (EAM) metal potentials
- lattice atom creation
- NVE, NVT, NPT integrators
- all serial DYNAMO features for controls and constraints

GranFlow

- F90 + MPI
- spatial−decomposition parallelism
- frictional granular potentials
- NVE integrator
- boundary conditions for granular flow and packing and walls
- particle insertion

LAMMPS WWW Site − LAMMPS Documentation − LAMMPS Commands

---

angle_coeff command

Syntax:

```
angle_coeff N args
```

- N = angle type (see asterik form below)
- args = coefficients for one or more angle types

Examples:

```
angle_coeff 1 300.0 107.0
angle_coeff * 5.0
angle_coeff 2*10 5.0
```

Description:

Specify the force field coefficients for one or more angle types. The number and meaning of the coefficients depends on the angle style. As described below, angle coefficients can also be set in the data file read by the read_data command or in a restart file.

N can be specified in one of two ways. An explicit numeric value can be used, as in the 1st example above. Or a wild−card asterik can be used to set the coefficients for multiple angle types. This takes the form "*" or "*n" or "n*" or "m*n". If N = the number of angle types, then an asterik with no numeric values means all types from 1 to N. A leading asterik means all types from 1 to n (inclusive). A trailing asterik means all types from n to N (inclusive). A middle asterik means all types from m to n (inclusive).

Note that using 2 angle_coeff commands for the same angle type is perfectly valid. For example, these commands set the coeffs for all angle types, then overwrite the coeffs for just angle type 2:

```
angle_coeff * 200.0 107.0 1.2
angle_coeff 2 50.0 107.0
```

A line in a data file that specifies angle coefficients uses the exact same format as the arguments of the angle_coeff command in an input script, except that wild−card asteriks should not be used since coefficients for all N types are listed in the file. For example, under the "Angle Coeffs" section of a data file, the line that corresponds to the 1st example above would be listed as

```
1 300.0 107.0
```

The units of each coefficient are shown in parenthesis.

$$E = K(\theta - \theta_0)^2 + K_{UB}(r - r_{UB})^2$$

For style *charmm*, specify 4 coefficients:

- K (energy/radian^2)
- theta0 (degrees)
- K_ub (energy/distance^2)
- r_ub (distance)

Theta0 is specified in degrees, but LAMMPS converts it to radians internally; hence the units of K are in energy/radian^2.

$$\begin{aligned} E &= E_a + E_{bb} + E_{ba} \\ E_a &= K_2(\theta - \theta_0)^2 + K_3(\theta - \theta_0)^3 + K_4(\theta - \theta_0)^4 \\ E_{bb} &= M(r_{ij} - r_1)(r_{jk} - r_2) \\ E_{ba} &= N_1(r_{ij} - r_1)(\theta - \theta_0) + N_2(r_{jk} - r_2)(\theta - \theta_0) \end{aligned}$$

For style *class2*, only coefficients for the Ea formula can be specified in the input script. These are the 4 coefficients:

- theta0 (degrees)
- K2 (energy/radian^2)
- K3 (energy/radian^2)
- K4 (energy/radian^2)

Theta0 is specified in degrees, but LAMMPS converts it to radians internally; hence the units of K are in energy/radian^2.

Coefficients for the Ebb and Eba formulas must be specified in the data file.

For the Ebb formula, the coefficients are listed under a "BondBond Coeffs" heading and each line lists 3 coefficients:

- M (energy/distance^2)
- r1 (distance)
- r2 (distance)

For the Eba formula, the coefficients are listed under a "BondAngle Coeffs" heading and each line lists 4 coefficients:

- N1 (energy/distance^2)
- N2 (energy/distance^2)
- r1 (distance)
- r2 (distance)

The theta0 value in the Eba formula is not specified, since it is the same value from the Ea formula.

$$E = K[1 + \cos(\theta)]$$

For style *cosine*, specify 1 coefficient:

- K (energy)

$$E = K[\cos^2(\theta) - \cos^2(\theta_0)]$$

For style *cosine/squared*, specify 2 coefficients:

- K (energy)
- theta0 (degrees)

Theta0 is specified in degrees, but LAMMPS converts it to radians internally.

$$E = K(\theta - \theta_0)^2$$

For style *harmonic,* specify 2 coefficients:

- K (energy/radian^2)
- theta0 (degrees)

Theta0 is specified in degrees, but LAMMPS converts it to radians internally; hence the units of K are in energy/radian^2.

Restrictions:

This command must come after the simulation box is defined by a read_data, read_restart, or create_box command.

An angle style must be defined before any angle coefficients are set, either in the input script or in a data file.

Related commands:

angle_style

Default: none

LAMMPS WWW Site − LAMMPS Documentation − LAMMPS Commands

---

angle_style command

Syntax:

```
angle_style style
```

- style = *none* or *charmm* or *class2* or *cosine* or *cosine/squared* or *harmonic*

Examples:

```
angle_style harmonic
angle_style charmm
```

Description:

Set the formula LAMMPS will use to compute angle interactions between triplets of atoms. The list of atom triplets is specified in the data or restart file and is read in by a read_data or read_restart command. The coefficients for the formula for each angle type can also be specified in those files or via the angle_coeff command. In all the formulas to follow, *theta* is the angle defined by the triplet of aotms.

A style of *none* means angle forces are not computed, even if angles are defined.

The *charmm* style uses the potential

$$E = K(\theta - \theta_0)^2 + K_{UB}(r - r_{UB})^2$$

with an additional Urey_Bradley term based on the distance *r* between the 1st and 3rd atoms in the angle. K, theta0, Kub, and Rub are coefficients defined for each angle type.

The *class2* style uses the potential

$$\begin{aligned} E &= E_a + E_{bb} + E_{ba} \\ E_a &= K_2(\theta - \theta_0)^2 + K_3(\theta - \theta_0)^3 + K_4(\theta - \theta_0)^4 \\ E_{bb} &= M(r_{ij} - r_1)(r_{jk} - r_2) \\ E_{ba} &= N_1(r_{ij} - r_1)(\theta - \theta_0) + N_2(r_{jk} - r_2)(\theta - \theta_0) \end{aligned}$$

where Ea is the angle term, Ebb is a bond−bond term, and Eba is a bond−angle term. Theta0 is the equilibrium angle and r1 and r2 are the equilibrium bond lengths. Kn, M, Nn, theta0, r1, r2 are coefficients defined for each angle type.

The *cosine* style uses the potential

$$E = K[1 + \cos(\theta)]$$

where K is defined for each angle type.

The *cosine/squared* style is a potential commonly used in the Gromacs MD code and is of the form

$$E = K[\cos^2(\theta) - \cos^2(\theta_0)]$$

where theta0 is the equilibrium value of the angle, and K is a prefactor. Note that the usual 1/2 factor is included in K. K and theta0 are coefficients defined for each angle type.

The *harmonic* style uses the potential

$$E = K(\theta - \theta_0)^2$$

where theta0 is the equilibrium value of the angle, and K is a prefactor. Note that the usual 1/2 factor is included in K. K and theta0 are coefficients defined for each angle type.

Restrictions:

Angle styles can only be set for atom_styles that allow angles to be defined.

Angle styles are part of the "molecular" package. They are only enabled if LAMMPS was built with that package. See the Making LAMMPS section for more info.

Related commands:

angle_coeff

Default:

```
angle_style none
```

LAMMPS WWW Site − LAMMPS Documentation − LAMMPS Commands

---

atom_modify command

Syntax:

```
atom_modify keyword value ...
```

- one or more keyword/value pairs may be appended
- keyword = *map*

```
map value = array or hash
```

Examples:

```
atom_modify map hash
```

Description:

Modify properties of the atom style selected within LAMMPS.

The *map* keyword determines how atom ID lookup is done for molecular problems. Lookups are performed by bond (angle, etc) routines in LAMMPS to find the local atom index associated with a global atom ID. When the *array* value is used, each processor stores a lookup table of length N, where N is the total # of atoms in the system. This is the fastest method for most simulations, but a processor can run out of memory to store the table for very large simulations. The *hash* value uses a hash table to perform the lookups. This method can be slightly slower than the *array* method, but its memory cost is proportional to N/P on each processor, where P is the total number of processors running the simulation.

Restrictions:

This command must be used before the simulation box is defined by a read_data or create_box command.

Related commands: none

Default:

By default, atomic (non−molecular) problems do not allocate maps. For molecular problems, the option default is map = array.

LAMMPS WWW Site − LAMMPS Documentation − LAMMPS Commands

---

atom_style command

Syntax:

```
atom_style style args
```

- style = *angle* or *atomic* or *bond* or *charge* or *dipole* or *dpd* or *full* or *granular* or *molecular* or *hybrid*

```
  args = none for any style except hybrid
  hybrid args = list of one or more styles
```

Examples:

```
atom_style bond
atom_style full
atom_style hybrid charge bond
```

Description:

Define what style of atoms to use in a simulation. This determines what attributes are associated with the atoms. This command must be used before a simulation is setup via a read_data, read_restart, or create_box command.

Once a style is assigned, it cannot be changed, so use a style general enough to encompass all attributes. E.g. with style *bond*, angular terms cannot be used or added later to the model. It is OK to use a style more general than needed, though it may be slightly inefficient.

The choice of style affects what quantities are stored by each atom, what quantities are communicated between processors to enable forces to be computed, and what quantities are listed in the data file read by the read_data command.

These are the attributes of each style. All styles store coordinates, velocities, atom IDs and types.

- *angle* = bonds and angles − e.g. bead−spring polymers with stiffness
- *atomic* = only the default values
- *bond* = bonds − e.g. bead−spring polymers
- *charge* = charge
- *dipole* = charge + dipole moments
- *dpd* = default values, also communicates velocities
- *molecular* = bonds, angles, dihedrals, impropers − e.g. all−atom polymers
- *full* = molecular + charge − e.g. biomolecules, charged polymers
- *granular* = granular material with rotational properties

Typical simulations with a single pair potential will use only one of these styles. For cases where multiple pair potentials will be used (see the pair_style *hybrid* command), it may be necessary to use multiple atom styles. In these cases the *hybrid* style can be used to list multiple atom styles. Atoms will then store and communicate the union of all quantities implied by the individual styles.

LAMMPS can be extended with new atom styles; see this section.

Restrictions:

This command cannot be used after the simulation box is defined by a read_data or create_box command.

The *angle*, *bond*, *full*, and *molecular* styles are part of the "molecular" package. The *granular* style is part of the "granular" package. The *dpd* style is part of the "dpd" package. They are only enabled if LAMMPS was built with that package. See the Making LAMMPS section for more info.

Related commands:

read_data, pair_style

Default: none

LAMMPS WWW Site − LAMMPS Documentation − LAMMPS Commands

bond_coeff command

Syntax:

```
bond_coeff N args
```

- N = bond type (see asterik form below)
- args = coefficients for one or more bond types

Examples:

```
bond_coeff 5 80.0 1.2
bond_coeff * 30.0 1.5 1.0 1.0
bond_coeff 1*4 30.0 1.5 1.0 1.0
bond_coeff 1 harmonic 200.0 1.0
```

Description:

Specify the force field coefficients for one or more bond types. The number and meaning of the coefficients depends on the bond style. As described below, bond coefficients can also be set in the data file read by the read_data command or in a restart file.

N can be specified in one of two ways. An explicit numeric value can be used, as in the 1st example above. Or a wild−card asterik can be used to set the coefficients for multiple bond types. This takes the form "*" or "*n" or "n*" or "m*n". If N = the number of bond types, then an asterik with no numeric values means all types from 1 to N. A leading asterik means all types from 1 to n (inclusive). A trailing asterik means all types from n to N (inclusive). A middle asterik means all types from m to n (inclusive).

Note that using 2 bond_coeff commands for the same bond type is perfectly valid. For example, these commands set the coeffs for all bond types, then overwrite the coeffs for just bond type 2:

```
bond_coeff * 100.0 1.2
bond_coeff 2 200.0 1.2
```

A line in a data file that specifies bond coefficients uses the exact same format as the arguments of the bond_coeff command in an input script, except that wild−card asteriks should not be used since coefficients for all N types must be listed in the file. For example, under the "Bond Coeffs" section of a data file, the line that corresponds to the 1st example above would be listed as

```
5 80.0 1.2
```

The units of each coefficient are shown in parenthesis.

$$E = K_2(r - r_0)^2 + K_3(r - r_0)^3 + K_4(r - r_0)^4$$

For style *class2*, specify 4 coefficients:

- R0 (distance)
- K2 (energy/distance^2)
- K3 (energy/distance^2)

- K4 (energy/distance^2)

$$E = -0.5 K R_0^2 \ln \left[1 - \left(\frac{r}{R_0} \right)^2 \right] + 4\epsilon \left[\left(\frac{\sigma}{r} \right)^{12} - \left(\frac{\sigma}{r} \right)^6 \right] + \epsilon$$

For style *fene*, specify 4 coefficients:

- K (energy/distance^2)
- R0 (distance)
- epsilon (energy)
- sigma (distance)

$$E = -0.5 K R_0 \ln \left[1 - \left(\frac{(r - \Delta)}{R_0} \right)^2 \right] + 4\epsilon \left[\left(\frac{\sigma}{(r - \Delta)} \right)^{12} - \left(\frac{\sigma}{(r - \Delta)} \right)^6 \right] + \epsilon$$

For style *fene/expand*, specify 5 coefficients:

- K (energy/distance^2)
- R0 (distance)
- epsilon (energy)
- sigma (distance)
- delta (distance)

$$E = K(r - r_0)^2$$

For style *harmonic*, specify 2 coefficients:

- K (energy/distance^2)
- r0 (distance)

$$E = D \left[1 - e^{-\alpha(r - r_0)} \right]^2$$

For style *morse*, specify 3 coefficients:

- D (energy)
- alpha (inverse distance)
- r0 (distance)

$$E = \frac{\epsilon (r - r_0)^2}{[\lambda^2 - (r - r_0)^2]}$$

For style *nonlinear*, specify 3 coefficients:

- epsilon (energy)
- r0 (distance)
- lamda (distance)

For style *restrain*, specify 4 coefficients:

- K (energy/distance^2)
- r0 (distance)
- r1 (distance)
- r2 (distance)

For style *hybrid*, the first coefficient sets the bond style and the remaining coefficients are those appropriate to that style. For example, these commands:

```
bond_coeff 1 fene 30.0 1.5 1.0 1.0
bond_coeff 2 harmonic 80.0 1.2
```

would set bonds of bond type 1 to be computed with a *fene* potential with coefficients 30.0, 1.5, 1.0, 1.0 for K, R0, epsilon, sigma. Likewise, bonds of bond type 2 would be computed with a *harmonic* potential with coefficients 80.0, 1.2 for K, r0.

Restrictions:

This command must come after the simulation box is defined by a read_data, read_restart, or create_box command.

A bond style must be defined before any bond coefficients are set, either in the input script or in a data file.

Related commands:

bond_style

Default: none

LAMMPS WWW Site − LAMMPS Documentation − LAMMPS Commands

---

bond_style command

Syntax:

```
bond_style style args
```

- style = *none* or *class2* or *fene* or *fene/expand* or *harmonic* or *morse* or *nonlinear* or *hybrid*

```
  args = none for any style except hybrid
  hybrid args = list of one or more styles
```

Examples:

```
bond_style harmonic
bond_style fene
bond_style hybrid harmonic fene
```

Description:

Set the formula(s) LAMMPS will use to compute bond interactions between pairs of atoms. The list of atom pairs is specified in the data or restart file and is read in by a read_data or read_restart command. The coefficients for the formula for each bond type can also be specified in those files or via the bond_coeff command. In all the formulas to follow, *r* is the distance between the 2 atoms in the bond.

A style of *none* means bond forces are not computed, even if bond are defined.

The *class2* style uses the potential

$$E = K_2(r - r_0)^2 + K_3(r - r_0)^3 + K_4(r - r_0)^4$$

where r0 is the equilibrium bond distance. Kn and r0 are coefficients defined for each bond type.

The *fene* style uses the potential

$$E = -0.5KR_0^2 \ln\left[1 - \left(\frac{r}{R_0}\right)^2\right] + 4\epsilon\left[\left(\frac{\sigma}{r}\right)^{12} - \left(\frac{\sigma}{r}\right)^6\right] + \epsilon$$

to define a finite extensible nonlinear elastic (FENE) potential (Kremer), used for bead−spring polymer models. The first term is attractive, the 2nd Lennard−Jones term is repulsive. The first term extends to R0, the maximum extent of the bond. The 2nd term is cutoff at 2^(1/6) sigma, the minimum of the LJ potential. K, R0, epsilon, and sigma are coefficients defined for each bond type.

The *fene/expand* style is similar to *fene* except that an extra shift factor of delta (positive or negative) is added to *r* to effectively change the bead size of the bonded atoms. The corresponding potential is

$$E = -0.5KR_0 \ln\left[1 - \left(\frac{(r-\Delta)}{R_0}\right)^2\right] + 4\epsilon\left[\left(\frac{\sigma}{(r-\Delta)}\right)^{12} - \left(\frac{\sigma}{(r-\Delta)}\right)^6\right] + \epsilon$$

The first term now extends to R0 + delta and the 2nd term is cutoff at 2^(1/6) sigma + delta. K, R0, epsilon, sigma, and delta are coefficients defined for each bond type.

The *harmonic* style uses the potential

$$E = K(r - r_0)^2$$

where r0 is the equilibrium bond distance. Note that the usual 1/2 factor is included in K. K and r0 are coefficients defined for each bond type.

The *morse* style uses the potential

$$E = D \left[1 - e^{-\alpha(r - r_0)} \right]^2$$

where r0 is the equilibrium bond distance, alpha is a stiffness parameter, and D determines the depth of the potential well. D, alpha, and r0 are coefficients defined for each bond type.

The *nonlinear* style uses the potential

$$E = \frac{\epsilon (r - r_0)^2}{\left[\lambda^2 - (r - r_0)^2 \right]}$$

to define an anharmonic spring (Rector) of equilibrium length r0 and maximum extension lamda. Epsilon, r0, and lamda are coefficients defined for each bond type.

The *hybrid* style enables the use of multiple bond styles in one simulation. A bond style is assigned to each bond type. For example, bonds in a polymer flow (of bond type 1) could be computed with a *fene* potential and bonds in the wall boundary (of bond type 2) could be computed with a *harmonic* potential. The assignment of bond type to style is made via the bond_coeff command or in the data file.

Restrictions:

Bond styles can only be set for atom styles that allow bonds to be defined.

Bond styles are part of the "molecular" package. They are only enabled if LAMMPS was built with that package. See the Making LAMMPS section for more info.

Related commands:

bond_coeff, delete_bonds

Default:

bond_style none

(Kremer) Kremer, Grest, J Chem Phys, 92, 5057 (1990).

(Rector) Rector, Van Swol, Henderson, Molecular Physics, 82, 1009 (1994).

LAMMPS WWW Site − LAMMPS Documentation − LAMMPS Commands

boundary command

Syntax:

```
boundary x y z
```

- x,y,z = *p* or *s* or *f* or *m*, one or two letters

```
  p is periodic
    f is non−periodic and fixed
    s is non−periodic and shrink−wrapped
    m is non−periodic and shrink−wrapped with a minimum value
```

Examples:

```
boundary p p f
boundary p fs p
boundary s f fm
```

Description:

Set the style of boundaries for the global simulation box in each dimension. A single letter assigns the same style to both the lower and upper face of the box. Two letters assigns the first style to the lower face and the second style to the upper face. The initial size of the simulation box is set by the read_data, read_restart, or create_box commands.

The style *p* means the box is periodic, so that particles interact across the boundary, and they can exit one end of the box and re−enter the other end. A periodic dimension can change in size due to constant pressure boundary conditions or volume rescaling (see the fix npt and fix volume/rescale commands). The *p* style must be applied to both faces of a dimension.

The styles *f*, *s*, and *m* mean the box is non−periodic, so that particles do not interact across the boundary and do not move from one side of the box to the other. For style *f*, the position of the face is fixed. If an atom moves outside the face it may be lost. For style *s*, the position of the face is set so as to encompass the atoms in that dimension (shrink−wrapping), no matter how far they move. For style *m*, shrink−wrapping occurs, but is bounded by the value specified in the data or restart file or set by the create_box command. For example, if the upper z face has a value of 50.0 in the data file, the face will always be positioned at 50.0 or above, even if the maximum z−extent of all the atoms becomes less than 50.0.

Restrictions:

This command cannot be used after the simulation box is defined by a read_data or create_box command.

Related commands:

See the thermo_modify command for a discussion of lost atoms.

Default:

```
boundary p p p
```

LAMMPS WWW Site − LAMMPS Documentation − LAMMPS Commands

cd command

Syntax:

```
cd dir
```

- dir = directory to change to

Examples:

```
cd sub1
cd ../new2
cd ..
```

Description:

Change the working directory. All subsequent LAMMPS commands that access files for reading or writing will use the new directory.

Restrictions:

If the specified directory does not exist, LAMMPS will not detect the error.

Related commands: none

Default: none

LAMMPS WWW Site − LAMMPS Documentation − LAMMPS Commands

clear command

Syntax:

```
clear
```

Examples:

```
(commands for 1st simulation)
clear
(commands for 2nd simulation)
```

Description:

This command deletes all atoms, restores all settings to their default values, and frees all memory allocated by LAMMPS. Once a clear command has been executed, it is as if LAMMPS were starting over, with only the exceptions noted below. This command enables multiple jobs to be run sequentially from one input script.

These settings are not affected by a clear command: the working directory (cd command), log file status (log command), echo status (echo command), and input script variables (variable command).

Restrictions: none

Related commands: none

Default: none

create_atoms command

Syntax:

```
create_atoms type region−ID
```

- type = atom type (1−N) of atoms to create on a lattice
- region−ID = ID of region each atom will belong to (optional)

Examples:

```
create_atoms 1 regsphere
create_atoms 3
```

Description:

This command creates atoms on a lattice as an alternative to reading in their coordinates via a read_data or read_restart command. A simulation box must already exist, which is created with the create_box command.

Before using this command, a lattice must be defined using the lattice command. If a region is not specified, the create_atoms command fills the entire simulation box with atoms on the lattice. If a region is specified, then the geometric volume is filled that is inside the simulation box and is also consistent with the region volume.

The *create_atoms* command can be used multiple times with different lattice orientations to create grain boundaries. Used in conjunction with the delete_atoms command, reasonably complex geometries can be created. The *create_atoms* command can also be used to add atoms to a system previously read in from a data or restart file. In all these cases, care should be taken to insure that new atoms do not overlap existing atoms inappropriately.

Created atoms are assigned the specified atom type and a velocity of 0.0.

Restrictions:

An atom_style and lattice must be previously defined to use this command.

Related commands:

lattice, orient, origin, region, create_box, read_data, read_restart

Default: none

create_box command

Syntax:

```
create_box N region−ID
```

- N = # of atom types to use in this simulation
- region−ID = ID of region to use as simulation domain

Examples:

```
create_atoms 2 mybox
```

Description:

This command creates a simulation box that encloses the specified region. Thus a region command must first be used to define a geometric domain. If the region is not of style *block*, LAMMPS encloses it with a rectangular simulation box.

The argument N is the number of atom types that will be used in the simulation.

Restrictions:

An atom_style and region must have been previously defined to use this command.

Related commands:

create_atoms, region

Default: none

LAMMPS WWW Site − LAMMPS Documentation − LAMMPS Commands

delete_atoms command

Syntax:

```
delete_atoms style args
```

- style = *group* or *region* or *overlap*

```
group args = group−ID
  region args = region−ID
  overlap args = distance (distance units)
```

Examples:

```
delete_atoms group edge
delete_atoms region regsphere
delete_atoms overlap 0.3
```

Description:

Delete the specfied atoms. For style *group*, it is all atoms belonging to the group. For style *region*, it is any atom that is in the region volume. For style *overlap*, pairs of atoms within the specified distance are searched for, and one of the 2 atoms is deleted. See the units command for a discussion of distance units.

This command can be used to carve out voids from a block of material.

After atoms are deleted, if the system is not molecular (no bonds), then atom IDs are re−assigned so that they run from 1 to the number of atoms in the system. This is not done for molecular systems, since it would foul up the bond connectivity that has already been assigned.

Restrictions: none

Related commands:

create_atoms

Default: none

LAMMPS WWW Site − LAMMPS Documentation − LAMMPS Commands

---

delete_bonds command

Syntax:

```
delete_bonds group-ID style args keyword ...
```

- group−ID = group ID
- style = *multi* or *atom* or *bond* or *angle* or *dihedral* or *improper* or *stats*

```
multi args = none
  atom args = an atom type
  bond args = a bond type
  angle args = an angle type
  dihedral args = a dihedral type
  improper args = an improper type
  stats args = none
```

- zero or more keywords may be appended to the args
- keyword = *undo* or *remove* or *special*

Examples:

```
delete_bonds frozen multi remove
delete_bonds all atom 4 special
delete_bonds all stats
```

Description:

Turn off (or on) molecular topology interactions, i.e. bonds, angles, dihedrals, impropers. This command is useful for deleting interactions that have been previously turned off by bond−breaking potentials. It is also useful for turning off topology interactions between frozen or rigid atoms. Pairwise interactions can be turned off via the neigh_modify exclude command. The fix shake command also effectively turns off certain bond and angle interactions.

For all styles, an interaction is only turned off (or on) if all the atoms involved are in the specified group. For style *multi* this is the only criterion applied − all types of bonds, angles, dihedrals, impropers in the group turned off.

For style *atom*, one or more of the atoms involved must also be of the specified type. For style *bond*, only bonds are candidates for turn−off, and the bond must be of the specified type. Styles *angle*, *dihedral*, and *improper* are treated similarly.

For style *stats* no interactions are turned off (or on); the status of all interactions in the specified group is simply reported. This is useful for diagnostic purposes if bonds have been turned off by a bond−breaking potential during a previous run.

The default behavior of the delete_bonds command is to turn off interactions by toggling their type to a negative value. E.g. a bond_type of 2 is set to −2. The neighbor list creation routines will not include such an interaction in their interaction lists. The default is also to not alter the list of 1−2, 1−3, 1−4 neighbors computed by the special_bonds command and used to weight pairwise force and energy calculations. This means that pairwise computations will proceed as if the bond (or angle, etc) were still turned on.

The keywords listed above can be appended to the argument list to alter the default behavior.

The *undo* keyword inverts the delete_bonds command so that the specified bonds, angles, etc are turned on if they are currently turned off. This means any negative value is toggled to positive. Note that the fix shake command also sets bond and angle types negative, so this option should not be used on those interactions.

The *remove* keyword is invoked at the end of the delete_bonds operation. It causes turned−off bonds (angles, etc) to be removed from each atom's data structure and then adjusts the global bond (angle, etc) counts accordingly. Removal is a permanent change; removed bonds cannot be turned back on via the *undo* keyword. Removal does not alter the pairwise 1−2, 1−3, 1−4 weighting list.

The *special* keyword is invoked at the end of the delete_bonds operation, after (optional) removal. It re−computes the pairwise 1−2, 1−3, 1−4 weighting list. The weighting list computation treats turned−off bonds the same as turned−on. Thus, turned−off bonds must be removed if you wish to change the weighting list.

Note that the choice of *remove* and *special* options affects how 1−2, 1−3, 1−4 pairwise interactions will be computed across bonds that have been modified by the delete_bonds command.

Restrictions:

This command requires force fields (pair, bond, etc) be setup before using it, so that cutoff lengths are initialized and inter−processor communication can be performed to coordinate the deleting of bonds.

If deleted bonds (angles, etc) are removed but the 1−2, 1−3, 1−4 weighting list is not recomputed, this can cause a later fix shake command to fail due to an atom's bonds being inconsistent with the weighting list. This should only happen if the group used in the fix command includes both atoms in the bond, in which case you probably should be recomputing the weighting list.

Related commands:

neigh_modify exclude, special_bonds, fix shake

Default: none

dielectric command

Syntax:

```
dielectric value
```

- value = dielectric constant

Examples:

```
dielectric 2.0
```

Description:

Set the dielectric constant for Coulombic interactions (pairwise and long−range) to this value. The constant is unitless, since it is used to reduce the strength of the interactions. The value is used in the denominator of the formulas for Coulombic interations − e.g. a value of 4.0 reduces the Coulombic interactions to 25% of their default strength. See the pair_style command for more details.

Restrictions: none

Related commands:

pair_style

Default:

```
dielectric 1.0
```

dihedral_coeff command

Syntax:

```
dihedral_coeff N args
```

- N = dihedral type (see asterik form below)
- args = coefficients for one or more dihedral types

Examples:

```
dihedral_coeff 1 80.0 1 3
dihedral_coeff * 80.0 1 3 0.5
dihedral_coeff 2* 80.0 1 3 0.5
```

Description:

Specify the force field coefficients for one or more dihedral types. The number and meaning of the coefficients depends on the dihedral style. As described below, dihedral coefficients can also be set in the data file read by the read_data command or in a restart file.

N can be specified in one of two ways. An explicit numeric value can be used, as in the 1st example above. Or a wild−card asterik can be used to set the coefficients for multiple dihedral types. This takes the form "*" or "*n" or "n*" or "m*n". If N = the number of dihedral types, then an asterik with no numeric values means all types from 1 to N. A leading asterik means all types from 1 to n (inclusive). A trailing asterik means all types from n to N (inclusive). A middle asterik means all types from m to n (inclusive).

Note that using 2 dihedral_coeff commands for the same dihedral type is perfectly valid. For example, these commands set the coeffs for all dihedral types, then overwrite the coeffs for just dihedral type 2:

```
dihedral_coeff * 80.0 1 3
dihedral_coeff 2 200.0 1 3
```

A line in a data file that specifies dihedral coefficients uses the exact same format as the arguments of the dihedral_coeff command in an input script, except that wild−card asteriks should not be used since coefficients for all N types are listed in the file. For example, under the "Dihedral Coeffs" section of a data file, the line that corresponds to the 1st example above would be listed as

```
1 80.0 1 3
```

See the dihedral_style command for more discussion of the formulas that use these coefficients. The units of each coefficient are shown in parenthesis.

$$E = K[1 + \cos(n\phi + d)]$$

For style *charmm*, specify 4 coefficients:

- K (energy)
- n (integer >= 0)
- d (integer value of degrees)
- weighting factor (0.0 to 1.0)

The weighting factor is applied to pairwise interaction between the 1st and 4th atoms in the dihedral. Note that this weighting factor is unrelated to the weighting factor specified by the special bonds command which applies to all 1−4 interactions in the system. For CHARMM force fields, the latter should typically be set to 0.0, else the 1−4 interactions in a dihedral will be computed twice (once in the pair routine, and once by the charmm/dihedral routine).

$$
\begin{aligned}
E &= E_d + E_{mbt} + E_{ebt} + E_{at} + E_{aat} + E_{bb13} \\
E_d &= \sum_{n=1}^{3} K_n[1 - \cos(n\phi - \phi_n)] \\
E_{mbt} &= (r_{jk} - r_2)[A_1 \cos(\phi) + A_2 \cos(2\phi) + A_3 \cos(3\phi)] \\
E_{ebt} &= (r_{ij} - r_1)[B_1 \cos(\phi) + B_2 \cos(2\phi) + B_3 \cos(3\phi)] + \\
&\quad (r_{kl} - r_3)[C_1 \cos(\phi) + C_2 \cos(2\phi) + C_3 \cos(3\phi)] \\
E_{at} &= (\theta_{ijk} - \theta_1)[D_1 \cos(\phi) + D_2 \cos(2\phi) + D_3 \cos(3\phi)] + \\
&\quad (\theta_{jkl} - \theta_2)[E_1 \cos(\phi) + E_2 \cos(2\phi) + E_3 \cos(3\phi)] \\
E_{aat} &= M(\theta_{ijk} - \theta_1)(\theta_{jkl} - \theta_2)\cos(\phi) \\
E_{bb13} &= N(r_{ij} - r_1)(r_{kl} - r_3)
\end{aligned}
$$

For style *class2*, only coefficients for the Ed formula can be specified in the input script. These are the 6 coefficients:

- K1 (energy)
- phi1 (degrees)
- K2 (energy)
- phi2 (degrees)
- K3 (energy)
- phi3 (degrees)

Coefficients for all the other formulas must be specified in the data file.

For the Embt formula, the coefficients are listed under a "MiddleBondTorsion Coeffs" heading and each line lists 4 coefficients:

- A1 (energy/distance)
- A2 (energy/distance)
- A3 (energy/distance)
- r2 (distance)

For the Eebt formula, the coefficients are listed under a "EndBondTorsion Coeffs" heading and each line lists 8 coefficients:

- B1 (energy/distance)
- B2 (energy/distance)
- B3 (energy/distance)
- C1 (energy/distance)
- C2 (energy/distance)
- C3 (energy/distance)
- r1 (distance)
- r3 (distance)

For the Eat formula, the coefficients are listed under a "AngleTorsion Coeffs" heading and each line lists 8 coefficients:

- D1 (energy/radian)

- D2 (energy/radian)
- D3 (energy/radian)
- E1 (energy/radian)
- E2 (energy/radian)
- E3 (energy/radian)
- theta1 (degrees)
- theta2 (degrees)

Theta1 and theta2 are specified in degrees, but LAMMPS converts them to radians internally; hence the units of D and E are in energy/radian.

For the Eaat formula, the coefficients are listed under a "AngleAngleTorsion Coeffs" heading and each line lists 3 coefficients:

- M (energy/radian^2)
- theta1 (degrees)
- theta2 (degrees)

Theta1 and theta2 are specified in degrees, but LAMMPS converts them to radians internally; hence the units of M are in energy/radian^2.

For the Ebb13 formula, the coefficients are listed under a "BondBond13 Coeffs" heading and each line lists 3 coefficients:

- N (energy/distance^2)
- r1 (distance)
- r3 (distance)

$$E = K[1 + d\cos(n\phi)]$$

For style *harmonic*, specify 3 coefficients:

- K (energy)
- d (+1 or −1)
- n (integer >= 0)

$$E = A[1 - \cos(\theta)] + B[1 + \cos(3\theta)] + C[1 + \cos(\theta + \frac{\pi}{4})]$$

For style *helix*, specify 3 coefficients:

- A (energy)
- B (energy)
- C (energy)

$$E = \sum_{n=1,5} A_n \cos^{n-1}(\phi)$$

For style *multiharmonic*, specify 5 coefficients:

- A1 (energy)
- A2 (energy)
- A3 (energy)
- A4 (energy)
- A5 (energy)

Restrictions:

This command must come after the simulation box is defined by a read_data, read_restart, or create_box command.

An dihedral style must be defined before any dihedral coefficients are set, either in the input script or in a data file.

Related commands:

dihedral_style

Default: none

LAMMPS WWW Site − LAMMPS Documentation − LAMMPS Commands

---

dihedral_style command

Syntax:

```
dihedral_style style
```

- style = *none* or *charmm* or *class2* or *harmonic* or *helix* or *multi/harmonic*

Examples:

```
dihedral_style harmonic
dihedral_style multi/harmonic
```

Description:

Set the formula LAMMPS will use to compute dihedral interactions between quadruplets of atoms. The list of atom quadruplets is specified in the data or restart file and is read in by a read_data or read_restart command. The coefficients for the formula for each dihedral type can also be specified in those files or by the dihedral_coeff command. In all the formulas to follow, *phi* is the torsional angle defined by the quadruplet of atoms.

Here are some important points to take note of when defining the LAMMPS dihedral coefficients in the formulas that follow so that they are compatible with other force fields:

- The LAMMPS convention is that the trans position = 180 degrees, while in some force fields trans = 0 degrees.
- Some force fields reverse the sign convention on *d*.

- Some force fields divide/multiply *K* by the number of multiple torsions that contain the j−k bond in an i−j−k−l torsion.
- Some force fields let *n* be positive or negative which corresponds to *d* = 1 or −1 for the harmonic style.

A style of *none* means dihedral forces are not computed, even if dihedrals are defined.

The *charmm* style uses the potential

$$E = K[1 + \cos(n\phi + d)]$$

K, d, and n are coefficients defined for each dihedral type. Additionally, a weighting factor is defined (see the dihedral_coeff command) which is applied to the pairwise LJ and Coulombic interaction between the 1st and 4th atom in the dihedral quadruplet. Note that this weighting factor is unrelated to the weighting factor specified by the special bonds command which applies to all 1−4 interactions in the system. For CHARMM force fields, the latter should typically be set to 0.0, else the 1−4 interactions in a dihedral will be computed twice (once in the pair routine, and once by the charmm/dihedral routine).

The *class2* style uses the potential

$$\begin{aligned}
E &= E_d + E_{mbt} + E_{ebt} + E_{at} + E_{aat} + E_{bb13} \\
E_d &= \sum_{n=1}^{3} K_n[1 - \cos(n\phi - \phi_n)] \\
E_{mbt} &= (r_{jk} - r_2)[A_1 \cos(\phi) + A_2 \cos(2\phi) + A_3 \cos(3\phi)] \\
E_{ebt} &= (r_{ij} - r_1)[B_1 \cos(\phi) + B_2 \cos(2\phi) + B_3 \cos(3\phi)] + \\
&\quad (r_{kl} - r_3)[C_1 \cos(\phi) + C_2 \cos(2\phi) + C_3 \cos(3\phi)] \\
E_{at} &= (\theta_{ijk} - \theta_1)[D_1 \cos(\phi) + D_2 \cos(2\phi) + D_3 \cos(3\phi)] + \\
&\quad (\theta_{jkl} - \theta_2)[E_1 \cos(\phi) + E_2 \cos(2\phi) + E_3 \cos(3\phi)] \\
E_{aat} &= M(\theta_{ijk} - \theta_1)(\theta_{jkl} - \theta_2)\cos(\phi) \\
E_{bb13} &= N(r_{ij} - r_1)(r_{kl} - r_3)
\end{aligned}$$

where Ed is the dihedral term, Embt is a middle−bond−torsion term, Eebt is an end−bond−torsion term, Eat is an angle−torsion term, Eaat is an angle−angle−torsion term, and Ebb13 is a bond−bond−13 term.

Theta1 and theta2 are equilibrium angles and r1 r2 r3 are equilibrium bond lengths. Kn, An, Bn, Cn, Dn, En, M, N, Phi_n, theta1, theta2, r1, r2, r3 are coefficients defined for each dihedral type.

The *harmonic* style uses the potential

$$E = K[1 + d\cos(n\phi)]$$

K, d, and n are coefficients defined for each dihedral type.

The *helix* style uses the potential

$$E = A[1 - \cos(\theta)] + B[1 + \cos(3\theta)] + C[1 + \cos(\theta + \frac{\pi}{4})]$$

A, B, C are coefficients defined for each dihedral type.

This coarse−grain dihedral potential is described in (Guo). For dihedral angles in the helical region, the energy function is represented by a standard potential consisting of three minima, one corresponding to the trans (t) state and the other to gauche states (g+ and g−). The paper describes how the A,B,C parameters are chosen so as to balance secondary (largely driven by local interactions) and tertiary structure (driven by long−range interactions).

The *multi/harmonic* style uses the potential

$$E = \sum_{n=1,5} A_n \cos^{n-1}(\phi)$$

A1, A2, A3, A4, and A5 are coefficients defined for each dihedral type.

Restrictions:

Dihedral styles can only be set for atom styles that allow dihedrals to be defined.

Dihedral styles are part of the "molecular" package. They are only enabled if LAMMPS was built with that package. See the Making LAMMPS section for more info.

Related commands:

dihedral_coeff

Default:

dihedral_style none

(Guo) Guo and Thirumalai, Journal of Molecular Biology, 263, 323−43 (1996).

LAMMPS WWW Site − LAMMPS Documentation − LAMMPS Commands

---

dimension command

Syntax:

```
dimension N
```

- N = 2 or 3

Examples:

```
dimension 2
```

Description:

Set the dimensionality of the simulation. By default LAMMPS runs 3d simulations. To run a 2d simulation, this command should be used prior to setting up a simulation box via the create_box or read_data commands. Restart files also store this setting.

See the discussion in this section for additional instructions on how to run 2d simulations.

Restrictions:

This command must be used before the simulation box is defined by a read_data or create_box command.

Related commands:

fix enforce2d

Default:

```
dimension 3
```

dipole command

Syntax:

```
dipole I value
```

- I = atom type (see asterik form below)
- value = dipole

Examples:

```
dipole 1 1.0
dipole 3 2.0
dipole 3*5 0.0
```

Description:

Set the dipole moment for all atoms of one or more atom types. This command is only used for atom styles that require dipole moments (atom_style dipole). A value of 0.0 should be used if the atom type has no dipole moment. Dipole values can also be set in the read_data data file. See the units command for a discussion of dipole units.

I can be specified in one of two ways. An explicit numeric value can be used, as in the 1st example above. Or a wild−card asterik can be used to set the dipole moment for multiple atom types. This takes the form "*" or "*n" or "n*" or "m*n". If N = the number of atom types, then an asterik with no numeric values means all types from 1 to N. A leading asterik means all types from 1 to n (inclusive). A trailing asterik means all types from n to N (inclusive). A middle asterik means all types from m to n (inclusive).

A line in a data file that specifies a dipole moement uses the exact same format as the arguments of the dipole command in an input script, except that no wild−card asterik can be used. For example, under the "Dipoles" section of a data file, the line that corresponds to the 1st example above would be listed as

```
1 1.0
```

Restrictions:

This command must come after the simulation box is defined by a read_data, read_restart, or create_box command.

All dipoles moments must be defined before a simulation is run (if the atom style requires dipoles be set).

Related commands: none

Default: none

LAMMPS WWW Site − LAMMPS Documentation − LAMMPS Commands

---

displace_atoms command

Syntax:

```
displace_atoms group−ID style args keyword value ...
```

- group−ID = ID of group of atoms to displace
- style = *move* or *ramp*

```
move args = delx dely delz
    delx,dely,delz = distance to displace in each dimension (distance units)
  ramp args = ddim dlo dhi dim clo chi
    ddim = x or y or z
    dlo,dhi = displacement distance between dlo and dhi (distance units)
    dim = x or y or z
    clo,chi = lower and upper bound of domain to displace (distance units)
```

- zero or more keyword/value pairs may be appended to the args

```
  keyword = units
    value = box or lattice
```

Examples:

```
displace_atoms top move 0 −5 0 units box
displace_atoms flow ramp x 0.0 5.0 y 2.0 20.5
```

Description:

Displace a group of atoms. This can be useful to move atoms a large distance before beginning a simulation. For example, in a shear simulation, an initial strain can be imposed on the system. Or two groups of atoms can be brought into closer proximity.

The *move* style displaces the group of atoms by the specified 3d distance. The *ramp* style displaces atoms a variable amount in one dimension depending on the atom's coordinate in a (possibly) different dimension. For example, the second example command displaces atoms in the x−direction an amount between 0.0 and 5.0 distance units. Each atom's displacement depends on the fractional distance its y coordinate is between 2.0 and 20.5. Atoms with y−coordinates outside those bounds will be moved the minimum (0.0) or maximum (5.0) amount.

Distance units for the displacement are determined by the setting of *box* or *lattice* for the *units* keyword. *Box* means distance units as defined by the units command − e.g. Angstroms for *real* units. *Lattice* means to use lattice spacings as defined by the lattice command. The default is to use lattice units.

Care should be taken not to move atoms on top of other atoms. After the move, atoms are remapped to the periodic simulation box. In parallel, atoms should not be moved so far that they cross more than one processor's sub−domain, else they may be lost. If this is a problem, successive displace_atom commands can be used to move a greater distance.

Restrictions: none

Related commands: none

Default:

The option defaults are units = lattice.

LAMMPS WWW Site − LAMMPS Documentation − LAMMPS Commands

---

dump command

Syntax:

```
dump ID group−ID style N file args
```

- ID = user−assigned name for the dump
- group−ID = ID of the group of atoms to be dumped
- style = *atom* or *bond* or *custom*
- N = dump every this many timesteps
- file = name of file to write dump info to
- args = list of arguments for a particular style

```
  atom args = none
    bond args = none
    custom args = list of atom attributes
      possible attributes = tag, type, x, y, z, xs, ys, zs, ix, iy, iz,
                   vx, vy, vz, fx, fy, fz, q, mux, muy, muz, tqx, tqy, tqz,
                   centro, eng, sxx, syy, szz, sxy, sxz, syz
        tag = atom ID
        type = atom type
        x,y,z = unscaled atom coordinates
        xs,ys,zs = scaled atom coordinates
        ix,iy,iz = box image that the atom is in in
        vx,vy,vz = atom velocities
        fx,fy,fz = forces on atoms
        q = atom charge
        mux,muy,muz = orientation of dipolar atom
        tqx,tqy,tqz = torque on dipolar atoms
        centro = per−atom centro−symmetry parameter
```

```
            eng = per−atom pairwise energy
            sxx, syy, szz, sxy, sxz, syz = per−atom stress tensor components
```

Examples:

```
dump myDump all atom 100 dump.atom
dump 2 subgroup atom 50 dump.run.bin
dump 4a all custom 100 dump.myforce.* tag type x y vx fx
dump 4b all custom 100 dump.%.myforce tag type eng sxx syy szz
```

Description:

Dump a snapshot of atom quantities to one or more files every so many timesteps. As described below, the filename determines the kind of output (text or binary or gzipped, one big file or one per timestep, one big file or one per processor). Only information for atoms in the specified group is dumped. The dump_modify command can also alter what atoms are included. Because snapshot data is collected from multiple processors, the order of lines (typically one per atom) written into the dump file for a single snapshot is indeterminate.

Dumps are performed on timesteps that are a multiple of N, including timestep 0. If one run ends and another begins on a timestep that is a multiple of N, only one snapshot is written. N must be > 0. N can be changed between runs by using the dump_modify command.

The specified filename determines how the dump file(s) is written. The default is to write one large text file, which is opened when the dump command is invoked and closed when an undump command is used or when LAMMPS exits.

If a '*' character appears in the filename, then one file per snapshot is written and the '*' character is replaced with the timestep value. For example, tmp.dump.* becomes tmp.dump.0, tmp.dump.10000, tmp.dump.20000, etc.

If a '%' character appears in the filename, then one file is written for each processor and the '%' character is replaced with the processor ID from 0 to P−1. For example, tmp.dump.% becomes tmp.dump.0, tmp.dump.1, ... tmp.dump.P−1, etc. This can be a fast mode of output on parallel machines that support parallel I/O.

Note that the '*' and '%' characters can be used together to produce a large number of small dump files!

If the filename ends with ".bin", the dump file (or files if '*' or '%' is also used) is written in binary format. A binary dump file will be about the same size as a text version, but will typically write out much faster. Of course, when post−processing, you will need to convert it back to text format (see the binary2txt tool) or write your own code to read the binary file. The format of the binary file can be understood by looking at the tools/binary2txt.cpp file.

If the filename ends with ".gz", the dump file (or files if '*' or '%' is also used) is written in gzipped format. A gzipped dump file will be about 3x smaller than the text version, but will also take longer to write.

The *style* keyword determines what atom quantities are written to the file. Settings made via the dump_modify command can also alter the format of individual values and the file itself. Note that because periodic boundary conditions are enforced only on timesteps when neighbor lists are rebuilt, the coordinates of an atom can be slightly outside the simulation box.

For style *atom*, atom coordinates are written to the file, along with their tag (atom ID) and type (depending on dump_modify settings).

For style *bond*, the bond topology between atoms is written, in the same format specified in the data file read by the read_data command. Both atoms in the bond must be in the dump group for the bond to be written. Any bonds that have been broken (see the bond_style command) are not written. Bonds that have been turned off (see the fix shake or delete_bonds commands) are written into the file.

Style *custom* allows you to specify a list of atom attributes to be written to the dump file for each atom. Possible attributes are described above and will appear in the order specified. Be careful not to specify a quantity that is not defined for a particular simulation − such as *q* for atom style bond, since that atom style doesn't assign charges. Dumps occur at the very end of a timestep, so atom attributes will include any effects due to fixes that are applied during the timestep.

The *mux*, *muy*, *muz*, *tqy*, *tqx*, *tqy* attributes are specific to dipolar systems defined with an atom style of *dipole*.

The *centro* attribute causes the centro−symmetry parameter to be computed for each atom in the dump group using the following formula from (Kelchner)

$$P = \sum_{i=1}^{6} |\vec{R}_i + \vec{R}_{i+6}|^2$$

where the 12 nearest neighbors are found and Ri and Ri+6 are the vectors from the central atom to the opposite pair of nearest neighbors. In solid state systems this is a useful measure of the local lattice disorder around an atom and can be used to characterize whether the atom is part of a perfect lattice, a local defect (e.g. a dislocation or stacking fault), or at a surface. The neighbor list needed to compute this quantity is constructed each time the dump is performed. Thus it can be inefficient to dump this quantity too frequently or to have multiple dump commands, each with a *centro* attribute.

The *eng* attribute computes the pairwise energy for each atom. This is its pairwise interaction with all of its neighbors (divided by 2). Summed over all atoms, this should equal the pairwise energy of the entire system (Van der Waals + Coulombic). However, for force fields that include a contribution to the pairwise energy that is computed as part of dihedral terms (i.e. 1−4 interactions), this contribution is not included in the per−atom pairwise energy. Computation of the per−atom energy requires a loop thru the neighbor list and inter−processor communication, so it can be inefficient to dump this quantity too frequently or to have multiple dump commands, each with a *eng* attribute.

The *sxx*, *syy*, *szz*, *sxy*, *sxz*, *syz* attributes compute the pairwise stress tensor for each atom where the *ab* component of the stress on atom *i* is given by

$$S_{ab} = -\left[m v_a v_b + \frac{1}{2} \sum_{j=1}^{N} (a_i - a_j) F_{b_{ij}} \right]$$

where the first term is a kinetic energy component for atom *i*, *j* loops over the *N* neighbors of atom *i*, and *Fb* is one of 3 components of force on atom *i* due to atom *j*. Both *a* and *b* can take on values x,y,z to generate the 6 components of the symmetric tensor.

Note that this formula for stress does not include virial contributions from intra−molecular interactions (e.g. bonds, angles, torsions, etc). Also note that this quantity is the negative of the per−atom pressure tensor. It is also really a stress−volume formulation. It would need to be divided by a per−atom volume to have units of stress, but an individual atom's volume is not easy to compute in a deformed solid. Computation of stress tensor components requires a loop thru the neighbor list and inter−processor communication, so it can be inefficient to dump this quantity too frequently or to have multiple dump commands, each with stress tensor attributes.

See this section for information on how to modify LAMMPS to dump other kinds of per−atom quantities.

Restrictions:

To write gzipped dump files, you must compile LAMMPS with the −DGZIP option − see the Making LAMMPS section of the documentation.

The *bond* style is part of the "molecular" package. It is only enabled if LAMMPS was built with that package. See the Making LAMMPS section for more info.

Granular systems and pair potentials cannot be used to compute per−atom energy and stress. The fix gran/diag command should be used instead.

Related commands:

dump_modify, undump

Default: none

(Kelchner) Kelchner, Plimpton, Hamilton, Phys Rev B, 58, 11085 (1998).

LAMMPS WWW Site − LAMMPS Documentation − LAMMPS Commands

---

dump_modify command

Syntax:

```
dump_modify dump−ID keyword args ...
```

- dump−ID = ID of dump to modify
- one or more keyword/arg pairs may be appended
- keyword = *format* or *scale* or *image* or *header* or *flush* or *region* or *thresh*

```
format arg = C−style format string for one line of output
  scale arg = yes or no
  image arg = yes or no
  header arg = item or xyz
  flush arg = yes or no
  every arg = N
    N = dump every this many timesteps
  region arg = region−ID or "none"
  thresh args = attribute operation value
    attribute = same attributes (x,fy,eng,sxx,etc) used by dump custom style
    operation = <or  <= or > or >= or = or
    value = numeric value to compare to
    these 3 args can be replaced by the word "none" to turn off threshholding
```

Examples:

```
dump_modify 1 format "%d %d %20.15g %g %g" scale yes
dump_modify myDump image yes scale no flush yes
dump_modify 1 header xyz region mySphere thresh x <0.0 thresh energy >= 3.2
```

Description:

Modify the parameters of a previously defined dump command. Not all parameters are relevant to all dump styles.

Each dump style has a default C−style format string which simply specifies %d for integers and %g for real values. The *format* keyword can be used to override the default with a new C−style format string. Do not include a trailing "\n" newline character in the format string.

The *scale* and *image* keywords apply only to the dump atom style. A scale value of *yes* means atom coords are written in normalized units from 0.0 to 1.0 in each box dimension. A value of *no* means they are written in absolute distance units (e.g. Angstroms or sigma). If the image value is *yes*, 3 flags are appended to each atom's coords which are the absolute box image of the atom in each dimension. For example, an x image flag of −3 with a normalized coord of 0.5 means the atom is in the center of the box, but has passed thru the box boundary 3 times and is really 3 box lengths to the left of its current coordinate.

The *header* keyword determines the file format for the snapshots. A value of *item* means each keyword is prefaced by "ITEM:" which is the default LAMMPS format that works with its associated post−processing tools like the Pizza.py toolkit. A value of *xyz* writes the dump file in the XYZ format used by other molecular modeling codes and visualization tools. For dump atom commands, each line will have 4 quantities: the atom type and unscaled coordinates. For dump custom commands, each line will list all the quantities it specifies.

The *flush* option invokes a flush operation after a dump snapshot is written to the dump file. This insures the output in that file is current (no buffering by the OS), even if LAMMPS halts before the simulation completes.

The *every* option changes the dump frequency originally specified by the dump command to a new value which must be > 0.

The *region* keyword only applies to the dump custom style. If specified, only atoms in the region will be written to the dump file. Only one region can be applied as a filter (the last one specified). See the region command for more details. Note that a region can be defined as the "inside" or "outside" of a geometric shape, and it can be the "union" or "intersection" of a series of simpler regions.

The *thresh* keyword only applies to the dump custom style. Multiple threshholds can be specified. Specifying "none" turns off all threshhold criteria. If theshholds are specified, only atoms whose attributes meet all the threshhold criteria are written to the dump file. The possible attributes that can be tested for are the same as those that can be specified in the dump custom command. Note that different attributes can be output by the dump custom command than are used as threshhold criteria by the dump_modify command.

Restrictions: none

Related commands:

dump, undump

Default:

The option defaults are format = %d and %g for each integer or floating point value, scale = yes, image = no, header = item, flush = yes, region = none, and thresh = none.

LAMMPS WWW Site − LAMMPS Documentation − LAMMPS Commands

---

echo command

Syntax:

```
echo style
```

- style = *none* or *screen* or *log* or *both*

Examples:

```
echo both
echo log
```

Description:

This command determines whether LAMMPS echoes each input script command to the screen and/or log file as it is read and processed. If an input script has errors, it can be useful to look at echoed output to see the last command processed.

Restrictions: none

Related commands: none

Default:

```
echo log
```

LAMMPS WWW Site − LAMMPS Documentation − LAMMPS Commands

---

fix command

Syntax:

```
fix ID group−ID style args
```

- ID = user−assigned name for the fix
- group−ID = ID of the group of atoms to apply the fix to
- style = one of a long list of possible style names (see below)
- args = arguments used by a particular style

Examples:

```
fix 1 all nve
fix 3 all nvt 300.0 300.0 0.01
fix mine top setforce 0.0 NULL 0.0
```

Description:

Set a fix that will be applied to a group of atoms. In LAMMPS, a "fix" is any operation that is applied to the system during timestepping or minimization. Examples include updating of atom positions and velocities due to time integration, controlling temperature, applying constraint forces to atoms, enforcing boundary conditions, computing diagnostics, etc. There are dozens of fixes defined in LAMMPS and new ones can be added − see this section for a discussion.

Each fix style has its own documentation page which describes its arguments and what it does. For example, see the fix setforce page for information on style *setforce*.

Fixes perform their operations at different stages of the timestep. If 2 or more fixes both operate at the same stage of the timestep, they are invoked in the order they were specified in the input script.

Specifying a new fix with the same ID as an existing fix effectively replaces the old fix (and its parameters) with the new fix. This can only be done if the new fix has the same style as the existing fix.

Fixes can be deleted with the unfix command. Note that this is the only way to turn off a fix; simply specifying a new fix with a similar style will not turn off the first one. For example, using a "fix nve" command for a second run after using a "fix nvt" command for the first run, will not cancel out the NVT time integration invoked by the "fix nvt" command. Thus two time integrators would be in place!

Here is an alphabetic list of fix styles defined in LAMMPS:

- fix addforce − add a force to each atom
- fix aveforce − add an averaged force to each atom
- fix com − compute a center−of−mass
- fix drag − drag atoms towards a defined coordinate
- fix efield − impose electric field on system
- fix enforce2d − zero out z−dimension velocity and force
- fix freeze − freeze atoms in a granular simulation
- fix gran/diag − compute granular diagnostics
- fix gravity − add gravity to atoms in a granular simulation
- fix indent − impose force due to an indenter
- fix insert − add new atoms to a granular simulation
- fix langevin − Langevin temperature control
- fix lineforce − constrain atoms to move in a line
- fix msd − compute mean−squared displacement (i.e. diffusion coefficient)
- fix nph − constant NPH time integration via Nose/Hoover
- fix npt − constant NPT time integration via Nose/Hoover
- fix nve − constant NVE time integration
- fix nve/gran − NVE time integration for granular systems
- fix nvt − constant NVT time integration via Nose/Hoover
- fix orient/fcc − add grain boundary mobility force
- fix planeforce − constrain atoms to move in a plane
- fix print − print text and variables during a simulation
- fix rdf − compute radial distribution functions
- fix rigid − constrain one or more clusters of atoms to move as a rigid body
- fix setforce − set the force on each atom

- fix shake − SHAKE constraints on bonds and/or angles
- fix spring − apply harmonic spring force to atoms
- fix temp/rescale − temperature control by velocity rescaling
- fix tmd − guide a group of atoms to a new configuration
- fix uniaxial − uniaxial straining of system while preserving total volume
- fix viscous − viscous damping for granular simulations
- fix volume/rescale − density control by volume rescaling
- fix wall/gran − frictional walls for granular simulations
- fix wall/lj93 − Lennard−Jones 9/3 walls
- fix wiggle − oscillate walls and frozen atoms

Restrictions:

Some fix styles are part of specific packages. They are only enabled if LAMMPS was built with that package. See the Making LAMMPS section for more info.

The *freeze*, *gran/diag*, *gravity*, *insert*, *nve/gran*, and *wall/gran* styles are part of the "granular" package.

Related commands:

unfix, fix_modify

Default: none

fix addforce command

Syntax:

```
fix ID group−ID addforce fx fy fz
```

- ID, group−ID are documented in fix command
- addforce = style name of this fix command
- fx,fy,fz = force component values (force units)

Examples:

```
fix kick flow addforce 1.0 0.0 0.0
```

Description:

Add fx,fy,fz to the corresponding component of force for each atom in the group. This command can be used to give an additional push to atoms in a simulation, such as for a simulation of Poiseuille flow in a channel.

Restrictions: none

Related commands:

fix setforce, fix aveforce

Default: none

LAMMPS WWW Site − LAMMPS Documentation − LAMMPS Commands

fix aveforce command

Syntax:

```
fix ID group−ID aveforce fx fy fz
```

- ID, group−ID are documented in fix command
- aveforce = style name of this fix command
- fx,fy,fz = force component values (force units)

Examples:

```
fix pressdown topwall aveforce 0.0 −1.0 0.0
fix 2 bottomwall aveforce NULL −1.0 0.0
```

Description:

Apply an additional external force to a group of atoms in such a way that every atom experiences the same force. This is useful for pushing on wall or boundary atoms so that the structure of the wall does not change over time.

The existing force is averaged for the group of atoms, component by component. The actual force on each atom is then set to the average value plus the component specified in this command. This means each atom in the group receives the same force.

If any of the arguments is specified as NULL then the forces in that dimension are not changed. Note that this is not the same as specifying a 0.0 value, since that sets all forces to the same average value without adding in any additional force.

Restrictions: none

Related commands:

fix setforce, fix addforce

Default: none

LAMMPS WWW Site − LAMMPS Documentation − LAMMPS Commands

fix com command

Syntax:

```
fix ID group−ID com N file
```

- ID, group−ID are documented in fix command
- com = style name of this fix command

- N = compute center−of−mass every this many timesteps
- file = filename to write center−of−mass info to

Examples:

```
fix 1 all com 100 com.out
```

Description:

Compute the center−of−mass of the group of atoms every N steps, including all effects due to atoms passing thru periodic boundaries. Write the results to the specified file.

Restrictions: none

Related commands: none

Default: none

LAMMPS WWW Site − LAMMPS Documentation − LAMMPS Commands

---

fix drag command

Syntax:

```
fix ID group−ID drag x y z fmag delta
```

- ID, group−ID are documented in fix command
- drag = style name of this fix command
- x,y,z = coord to drag atoms towards
- fmag = magnitude of force to apply to each atom (force units)
- delta = cutoff distance inside of which force is not applied (distance units)

Examples:

```
fix center small−molecule drag 0.0 10.0 0.0 5.0 2.0
```

Description:

Apply a force to each atom in a group to drag it towards the point (x,y,z). The magnitude of the force is specified by fmag. If an atom is closer than a distance delta to the point, then the force is not applied.

Any of the x,y,z values can be specified as NULL which means do not include that dimension in the distance calculation or force application.

This command can be used to steer one or more atoms to a new location in the simulation.

Restrictions: none

Related commands:

fix spring

Default: none

LAMMPS WWW Site − LAMMPS Documentation − LAMMPS Commands

---

fix efield command

Syntax:

```
fix ID group-ID efield ex ey ez
```

- ID, group−ID are documented in fix command
- efield = style name of this fix command
- ex,ey,ez = E−field component values (electric field units)

Examples:

```
fix kick external-field efield 1.0 0.0 0.0
```

Description:

Add a force F = qE to each charged atom in the group due to an external electric field being applied to the system.

Restrictions: none

Related commands:

fix addforce

Default: none

LAMMPS WWW Site − LAMMPS Documentation − LAMMPS Commands

---

fix enforce2d command

Syntax:

```
fix ID group-ID enforce2d
```

- ID, group−ID are documented in fix command
- enforce2d = style name of this fix command

Examples:

```
fix 5 all enforce2d
```

Description:

Zero out the z−dimension velocity and force on each atom in the group. This is useful when running a 2d simulation to insure that atoms do not move from their initial z coordinate.

Restrictions: none

Related commands: none

Default: none

LAMMPS WWW Site − LAMMPS Documentation − LAMMPS Commands

---

fix freeze command

Syntax:

```
fix ID group−ID freeze
```

- ID, group−ID are documented in fix command
- freeze = style name of this fix command

Examples:

```
fix 2 bottom freeze
```

Description:

Zero out the force and torque on a granular particle. This is useful for preventing certain particles from moving in a simulation.

Restrictions:

Can only be used with atom_style granular.

There can only be a single freeze fix defined. This is because other parts of the code (pair potentials, thermodynamics, etc) treat frozen particles differently and need to be able to reference a single group to which this fix is applied.

Related commands: none

atom_style granular

Default: none

LAMMPS WWW Site − LAMMPS Documentation − LAMMPS Commands

---

fix gran/diag command

Syntax:

```
fix ID group−ID gran/diag nevery file zlayer
```

- ID, group−ID are documented in fix command
- gran/diag = style name of this fix command

- nevery = compute diagnostics every this many timesteps
- file = filename to store diagnostic info in
- zlayer = bin size in z dimension

Examples:

```
fix 1 all gran/diag 1000 tmp 0.9
```

Description:

Compute aggregate density, velocity, and stress diagnostics for a group of granular atoms as a function of z depth in the granular system. The results are written to 3 files named file.den, file.vel, and file.str. The z bins begin at the bottom of the system and extend upward with a thickness of zlayer for each bin. The quantities written to the file are averaged over all atoms in the bin.

Restrictions:

Can only be used with atom_style granular.

Related commands:

atom_style granular

Default: none

fix gravity command

Syntax:

```
fix ID group gravity style args
```

- ID, group are documented in fix command
- gravity = style name of this fix command
- style = *chute* or *spherical* or *gradient* or *vector*

```
chute args = angle
    angle = angle in +x away from −z axis (in degrees)
  spherical args = phi theta
    phi = azimuthal angle from +x axis (in degrees)
    theta = angle from +z axis (in degrees)
  gradient args = phi theta phi_grad theta_grad
    phi = azimuthal angle from +x axis (in degrees)
    theta = angle from +z axis (in degrees)
    phi_grad = rate of change of angle phi (full rotations per time unit)
    theta_grad = rate of change of angle theta
      (full rotations per time unit)
  vector args = magnitude x y z
    magnitude = size of acceleration (force/mass units)
    x y z = vector direction to apply the acceleration
```

Examples:

```
fix 1 all gravity chute 24.0
fix 1 all gravity spherical 0.0 −180.0
fix 1 all gravity gradient 0.0 −180.0 0.0 0.1
fix 1 all gravity vector 100.0 1 1 0
```

Description:

Impose an additional acceleration on each particle in the group. For granular systems the magnitude is chosen so as to be due to gravity. For non−granular systems the magnitude of the acceleration is specified, so it can be any kind of driving field desired (e.g. a pressure gradient inducing a Poisselle flow). Note that this is different from what the fix addforce command does, since it adds the same force to each atom, independent of its mass. This command adds the same acceleration to each atom (force/mass).

The first 3 styles apply to granular systems. Style *chute* is typically used for simulations of chute flow where the specified angle is the chute angle, with flow occurring in the +x direction. Style *spherical* allows an arbitrary 3d direction to be specified for the gravity vector. Style *gradient* allows the direction of the gravity vector to be time dependent. The units of the gradient arguments are in full rotations per time unit. E.g. a timestep of 0.001 and a gradient of 0.1 means the gravity vector would rotate thru 360 degrees every 10,000 timesteps. For the time−dependent case, the initial direction of the gravity vector is phi,theta at the time the fix is specified.

The strength of the acceleration due to gravity is 1.0 in LJ units, which are the only allowed units for granular systems.

Style *vector* is used for non−granular systems. An acceleration of the specified magnitude is applied to each atom in the group in the vector direction given by (x,y,z).

Restrictions:

Styles *chute*, *spherical*, and *gradient* can only be used with atom_style granular. Style *vector* can only be used with non−granular systems.

Related commands:

atom_style granular, fix addforce

Default: none

LAMMPS WWW Site − LAMMPS Documentation − LAMMPS Commands

---

fix indent command

Syntax:

```
fix ID group−ID indent k keyword args ...
```

- ID, group−ID are documented in fix command
- indent = style name of this fix command
- k = force constant for indenter surface (force/distance^2 units)
- one or more keyword/value pairs may be appended to the args
- keyword = *sphere* or *cylinder* or *vel* or *units*

```
      sphere args = x y z R
          x,y,z = initial position of center of indenter
          R = sphere radius of indenter (distance units)
        cylinder args = dim c1 c2 R
          dim = x or y or z = axis of cylinder
          c1,c2 = coords of cylinder axis in other 2 dimensions (distance units)
          R = cylinder radius of indenter (distance units)
        vel args = vx vy vz
          vx,vy,vz = velocity of center of indenter (velocity units)
        units value = lattice or box
          lattice = the geometry is defined in lattice units
          box = the geometry is defined in simulation box units
```

Examples:

```
fix 1 all indent 10.0 sphere 0.0 0.0 15.0 3.0 vel 0.0 0.0 -1.0
fix 2 flow indent 10.0 cylinder z 0.0 0.0 10.0 units box
```

Description:

Insert an indenter within a simulation box. The indenter repels all atoms that touch it, so it can be used to push into a material or as an obstacle in a flow.

The indenter can either be spherical or cylindrical. You must set one of those 2 keywords.

A spherical indenter exerts a force of magnitude

```
F(r) = - k (r - R)^2
```

on each atom where *k* is the specified force constant, *r* is the distance from the atom to the center of the indenter, and *R* is the radius of the indenter. The force is repulsive and F(r) = 0 for *r* > *R*.

A cylindrical indenter exerts the same force, except that *r* is the distance from the atom to the center axis of the cylinder. The cylinder extends infinitely along its axis.

If the *vel* keyword is specified, the center (or axis) of the spherical (or cylindrical) indenter will move during the simulation, based on its initial (x,y,z) position and the specified (vx,vy,vz).

The *units* keyword determines the meaning of the distance units used to define the indenter. A *box* value selects standard distance units as defined by the units command, e.g. Angstroms for units = real or metal. A *lattice* value means the distance units are in cubic lattice spacings. The lattice command must first be used to define a lattice. Note that the units choice affects not only the indenter's physical geometry, but also its velocity and force constant since they are defined in terms of distance as well.

Restrictions: none

Related commands: none

Default:

The option defaults are vel = 0,0,0 and units = lattice.

LAMMPS WWW Site - LAMMPS Documentation - LAMMPS Commands

fix insert command

Syntax:

```
fix ID group−ID insert N type seed keyword values ...
```

- ID, group−ID are documented in fix command
- insert = style name of this fix command
- N = # of atoms to insert
- type = atom type to assign to inserted atoms
- seed = random # seed
- one or more keyword/value pairs may be appended to args
- keyword = *region* or *diam* or *dens* or *vol* or *zrate* or *vel*

```
region value = region−ID
    region−ID = ID of region to use as insertion volume
  diam values = lo hi
    lo,hi = range of diameters for inserted particles (distance units)
  dens values = lo hi
    lo,hi = range of densities for inserted particles
  vol values = fraction Nattempt
    fraction = desired volume fraction for filling insertion volume
    Nattempt = max # of insertion attempts per atom
  zrate value = rate
    rate = z velocity at which insertion volume moves (velocity units)
  vel values = vxlo vxhi vylo vyhi vz
    vxlo,vxhi = range of x velocities for inserted particles (velocity units)
    vylo,vyhi = range of y velocities for inserted particles (velocity units)
    vz = z velocity assigned to inserted particles (velocity units)
```

Examples:

```
fix 3 all insert 1000 2 29494 region myblock
fix 2 all insert 10000 1 19985583 region disk vol 0.33 100 zrate 1.0 diam 0.9 1.1
```

Description:

Insert particles into a granular run every few timesteps within a specified region until N particles have been inserted. This is useful for simulating the pouring of particles into a container.

Inserted particles are assigned the specified atom type and are assigned to two groups: the default group "all" and the group specified in the fix insert command (which can also be "all").

This command must use the *region* keyword to define an insertion volume. The specified region must have been previously defined with a region command. It must be of type *block* or a z−axis *cylinder* and must be defined with side = *in*.

Each timestep particles are inserted, they are placed randomly inside the insertion volume so as to mimic a stream of poured particles. The larger the volume, the more particles that can be inserted at any one timestep. Particles are inserted again after enough time has elapsed that the previously inserted particles fall out of the insertion volume under the influence of gravity. Insertions continue every so many timesteps until the desired # of particles has been inserted.

All other keywords are optional with defaults as shown below. The *diam*, *dens*, and *vel* options enable inserted particles to have a range of diameters or densities or xy velocities. The specific values for a particular inserted particle will be chosen randomly and uniformly between the specified bounds. The *vz* value for option *vel* assigns a z−velocity to each inserted particle.

The *vol* option specifies what volume fraction of the insertion volume will be filled with particles. The higher the value, the more particles are inserted each timestep. Since inserted particles cannot overlap, the maximum volume fraction should be no higher than about 0.6. LAMMPS will make up to Nattempt tries to insert a new particle without overlaps. If it fails it prints a warning.

The *zrate* option allows the insertion volume to move in the z direction. This enables pouring particles from a successively higher height over time.

Restrictions:

Can only be used with atom_style granular. A gravity fix in the −z direction must be defined for use in conjunction with this fix.

Related commands:

fix_gravity, region

Default:

The option defaults are diam = 1.0 1.0, dens = 1.0 1.0, vol = 0.25 50, zrate = 0.0, vel = 0.0 0.0 0.0 0.0 0.0.

LAMMPS WWW Site − LAMMPS Documentation − LAMMPS Commands

---

fix langevin command

Syntax:

```
fix ID group−ID langevin Tstart Tstop damp seed xflag yflag zflag
```

- ID, group−ID are documented in fix command
- langevin = style name of this fix command
- Tstart,Tstop = desired temperature at start/end of run (temperature units)
- damp = Langevin damping parameter (time units)
- seed = random # seed to use for white noise (integer > 0 and < 900000000)
- xflag,yflag,zflag = 0/1 for whether to apply to each dimension (optional)

Examples:

```
fix 3 boundary langevin 1.0 1.0 1000.0 699483
fix 1 all langevin 1.0 1.1 100.0 48279 0 1 1
```

Description:

Apply a Langevin thermostat to a group of atoms. Uniform random numbers are used to generate a white−noise term that is added to the force of each atom to keep them at a specified temperature. The desired temperature at each timestep is a ramped value during the run from *Tstart* to *Tstop*. The *damp* parameter is specified in time units and determines how rapidly the temperature is relaxed. For example, a value of 100.0 means to relax the temperature in a timespan of (roughly) 100 time units (tau or fmsec or psec − see the units command).

The random # *seed* should be a non−zero integer with 1 to 8 digits. A Marsaglia random number generator is used. Each processor uses the input seed to generate its own unique seed and its own stream of random numbers. Thus the dynamics of the system will not be identical on two runs on different numbers of processors. Also, the state of the random number generator is not saved in a restart file. This means you cannot do exact restarts when a fix langevin command is used.

The last 3 arguments are flags that specify which dimensions to add langevin white noise to. A flag of 0 means do not add noise to that dimension. A flag of 1 means add noise. The default is 1 for all 3 dimensions. These flags are optional; use all 3 or none of them.

The way that temperature is computed by this fix can be changed by using the fix_modify command.

A langevin fix does not update the coordinates or velocities of its atoms. It is normally used with a fix of style *nve* that does that. A langevin fix should not normally be used on atoms that also have their temperature controlled by another fix − e.g. a nvt or temp/rescale fix.

Restrictions: none

Related commands:

fix nvt, fix temp/rescale, fix_modify

Default: none

LAMMPS WWW Site − LAMMPS Documentation − LAMMPS Commands

fix lineforce command

Syntax:

```
fix ID group−ID lineforce x y z
```

- ID, group−ID are documented in fix command
- lineforce = style name of this fix command
- x y z = direction of line as a 3−vector

Examples:

```
fix hold boundary lineforce 0.0 1.0 1.0
```

Description:

Adjust the forces on each atom in the group so that it's motion will be along the linear direction specified by the vector (x,y,z). This is done by subtracting out components of force perpendicular to the line.

If the initial velocity of the atom is 0.0 (or along the line), then it should continue to move along the line thereafter.

Restrictions: none

Related commands:

fix planeforce

Default: none

fix_modify command

Syntax:

```
fix_modify fix−ID keyword value ...
```

- fix−ID = ID of the fix to modify
- one or more keyword/value pairs may be appended
- keyword = *temp* or *thermo* or *energy*

```
temp value = temperature ID
  thermo value = yes or no
  energy value = yes or no
```

Examples:

```
fix_modify 3 temp myTemp
fix_modify 1 thermo yes energy no
```

Description:

Modify one or more parameters of a previously defined fix. Not all fix styles support all parameters.

The *temp* keyword is used to determine how a fix computes temperature. The specified temperature ID must have been previously defined by the user via the temperature command. The default setting for *temp* is temperature ID = *default*. Fix styles that use the *temp* setting are temp/rescale, nvt, and npt.

The *thermo* and *energy* keywords enable a fix to compute and print quantities as part of the thermodynamic output to the screen and log file every so many timesteps (if they are defined as part of the fix). See the thermo_style command for how this printing is formatted. When the *thermo* keyword is set to yes, the fix prints one or more values. When the *energy* keyword is set to yes, the fix computes a contribution to the PotEng quantity (total potential energy) printed during thermodynamic output. The fixes that use this option include: fix nvt, fix npt, fix nph, fix temp/rescale, and fix wall/lj93. See the individual fix commands for more info on what is printed.

Restrictions: none

Related commands:

fix, temperature, thermo_style

Default:

The option defaults are temp = default, thermo = no, energy = no.

LAMMPS WWW Site − LAMMPS Documentation − LAMMPS Commands

---

fix msd command

Syntax:

```
fix ID group−ID msd N file
```

- ID, group−ID are documented in fix command
- msd = style name of this fix command
- N = compute mean−squared displacement every this many timesteps
- file = filename to write mean−squared displacement info to

Examples:

```
fix 1 all msd 100 diff.out
```

Description:

Compute the mean−squared displacement of the group of atoms every N steps, including all effects due to atoms passing thru periodic boundaries. The slope of the mean−squared displacement versus time is proportional to the diffusion coefficient of the diffusing atoms. The "origin" of the displacements is the position of each atom at the time the fix command was issued. Write the results to the specified file.

Restrictions: none

Related commands: none

Default: none

LAMMPS WWW Site − LAMMPS Documentation − LAMMPS Commands

---

fix nph command

Syntax:

```
fix ID group−ID nph p−style args keyword value ...
```

- ID, group−ID are documented in fix command
- nph = style name of this fix command
- p−style = *xyz* or *xy* or *yz* or *xz* or *aniso*

```
  xyz args = Pstart Pstop Pdamp
      Pstart,Pstop = desired pressure at start/end of run (pressure units)
      Pdamp = pressure damping parameter (time units)
    xy or yz or xz args = Px0 Px1 Py0 Py1 Pz0 Pz1 Pdamp
      Px0,Px1,Py0,Py1,Pz0,Pz1 = desired pressure in x,y,z at
        start/end (0/1) of run (pressure units)
      Pdamp = pressure damping parameter (time units)
    aniso args = Px0 Px1 Py0 Py1 Pz0 Pz1 Pdamp
      Px0,Px1,Py0,Py1,Pz0,Pz1 = desired pressure in x,y,z at
        start/end (0/1) of run (pressure units)
      Pdamp = pressure damping parameter (time units)
```

- zero or more keyword/value pairs may be appended to the args
- keyword = *drag* or *dilate*

```
drag value = drag factor added to barostat (0.0 = no drag)
  dilate value = all or partial
```

Examples:

```
fix 1 all nph xyz 0.0 0.0 1000.0
fix 2 all nph xz 5.0 5.0 NULL NULL 5.0 5.0 1000.0 drag 1.0
fix 2 all nph aniso 0.0 0.0 0.0 0.0 NULL NULL 1000.0
```

Description:

Perform constant NPH integration to update positions and velocities each timestep for atoms in the group using a Nose/Hoover pressure barostat. P is pressure. This creates a system trajectory consistent with the isobaric ensemble. Unlike fix npt, temperature will not be controlled if no other fix is used. Temperature can be controlled independently by using "fix langevin or fix temp/rescale.

The atoms in the fix group are the only ones whose velocities and positions are updated by the velocity/position update portion of the NPT integration.

Regardless of what atoms are in the fix group, a global pressure is computed for all atoms. Similarly, when the size of the simulation box is changed, all atoms are re−scaled to new positions, unless the keyword *dilate* is specified with a value of *partial*, in which case only the atoms in the fix group are re−scaled. The latter can be useful for leaving the coordinates of atoms in a solid substrate unchanged and controlling the pressure of a surrounding fluid.

By default, the temperature is also computed for all atoms (used as part of the pressure calculation), regardless of what group is specified. This is because the pressure contains a kinetic energy term which is derived from temperature, and the kinetic energy should be consistent with the virial term computed using all atoms. This can be changed by assigning a different temperature method to the fix via the fix_modify command. LAMMPS will warn you if you choose to compute temperature on a subset of atoms.

The pressure can be controlled in one of several styles, as specified by the *p−style* argument. Style *xyz* means couple all 3 dimensions together when pressure is computed (isotropic pressure), and dilate/contract the 3 dimensions together.

Styles *xy* or *yz* or *xz* means that the 2 specified dimensions are coupled together, both for pressure computation and for dilation/contraction. The 3rd dimension dilates/contracts independently, using its pressure component as the driving force.

For style *aniso*, all 3 dimensions dilate/contract independently using their individual pressure components as the 3 driving forces.

For any of the styles except *xyz*, any of the independent pressure components (e.g. z in *xy*, or any dimension in *aniso*) can have their target pressures (both start and stop values) specified as NULL. This means that no pressure control is applied to that dimension so that the box dimension remains unchanged.

In some cases (e.g. for solids) the pressure (volume) and/or temperature of the system can oscillate undesirably when a Nose/Hoover barostat is applied. The optional *drag* keyword will damp these oscillations, although it alters the Nose/Hoover equations. A value of 0.0 (no drag) leaves the Nose/Hoover formalism unchanged. A non−zero value adds a drag term; the larger the value specified, the greater the damping effect. Performing a short run and monitoring the pressure is the best way to determine if the drag term is working. Typically a value between 0.2 to 2.0 is sufficient to damp oscillations after a few periods.

For all pressure styles, the simulation box stays rectangular in shape. Parinello−Rahman boundary conditions (tilted box) are not implemented in LAMMPS.

For all styles, the *Pdamp* parameter is specified in time units and determines how rapidly the pressure is relaxed. For example, a value of 1000.0 means to relax the temperature in a timespan of (roughly) 1000 time units (tau or fmsec or psec − see the units command).

This fix supports the "fix_modify" options for *thermo* and *energy*. The former will print the contribution the fix makes to the energy of the system when thermodynamics is printed. The latter will add this contribution to the total potential energy (PotEng) so that energy conservation can be monitored.

Restrictions:

Any dimension being adjusted by this fix must be periodic. A dimension whose target pressures are specified as NULL can be non−periodic or periodic.

You should not use fix nvt with this fix. Instead, use fix npt if you want to control both temperature and pressure via Nose/Hoover.

Related commands:

fix nve, fix npt, fix_modify

Default:

The keyword defaults are drag = 0.0 and dilate = all.

LAMMPS WWW Site − LAMMPS Documentation − LAMMPS Commands

---

fix npt command

Syntax:

```
fix ID group−ID npt Tstart Tstop Tdamp p−style args keyword value ...
```

- ID, group−ID are documented in fix command
- npt = style name of this fix command
- Tstart,Tstop = desired temperature at start/end of run
- Tdamp = temperature damping parameter (time units)
- p−style = *xyz* or *xy* or *yz* or *xz* or *aniso*

```
xyz args = Pstart Pstop Pdamp
    Pstart,Pstop = desired pressure at start/end of run (pressure units)
    Pdamp = pressure damping parameter (time units)
  xy or yz or xz or aniso args = Px_start Px_stop Py_start Py_stop Pz_start Pz_stop Pdamp
    Px_start,Px_stop,... = desired pressure in x,y,z at start/end of run (pressure units)
    Pdamp = pressure damping parameter (time units)
```

- zero or more keyword/value pairs may be appended to the args
- keyword = *drag* or *dilate*

```
  drag value = drag factor added to barostat/thermostat (0.0 = no drag)
    dilate value = all or partial
```

Examples:

```
fix 1 all npt 300.0 300.0 100.0 xyz 0.0 0.0 1000.0
fix 2 all npt 300.0 300.0 100.0 xz 5.0 5.0 NULL NULL 5.0 5.0 1000.0
fix 2 all npt 300.0 300.0 100.0 xz 5.0 5.0 NULL NULL 5.0 5.0 1000.0 drag 0.2
fix 2 water npt 300.0 300.0 100.0 aniso 0.0 0.0 0.0 0.0 NULL NULL 1000.0 dilate partial
```

Description:

Perform constant NPT integration to update positions and velocities each timestep for atoms in the group using a Nose/Hoover temperature thermostat and Nose/Hoover pressure barostat. P is pressure; T is temperature. This creates a system trajectory consistent with the isothermal−isobaric ensemble.

The desired temperature at each timestep is a ramped value during the run from *Tstart* to *Tstop*. The *Tdamp* parameter is specified in time units and determines how rapidly the temperature is relaxed. For example, a value of 100.0 means to relax the temperature in a timespan of (roughly) 100 time units (tau or fmsec or psec − see the units command).

The atoms in the fix group are the only ones whose velocities and positions are updated by the velocity/position update portion of the NPT integration.

Regardless of what atoms are in the fix group, a global pressure is computed for all atoms. Similarly, when the size of the simulation box is changed, all atoms are re−scaled to new positions, unless the keyword *dilate* is specified with a value of *partial*, in which case only the atoms in the fix group are re−scaled. The latter can be useful for leaving the coordinates of atoms in a solid substrate unchanged and controlling the pressure of a surrounding fluid.

By default, the temperature is also computed for all atoms, regardless of what group is specified. This is because the pressure contains a kinetic energy term which is derived from temperature, and the kinetic energy should be consistent with the virial term computed using all atoms. This can be changed by assigning a different temperature method to the fix via the fix_modify command. LAMMPS will warn you if you choose to compute temperature on a subset of atoms.

The pressure can be controlled in one of several styles, as specified by the *p−style* argument. Style *xyz* means couple all 3 dimensions together when pressure is computed (isotropic pressure), and dilate/contract the 3 dimensions together.

Styles *xy* or *yz* or *xz* means that the 2 specified dimensions are coupled together, both for pressure computation and for dilation/contraction. The 3rd dimension dilates/contracts independently, using its pressure component as the driving force.

For style *aniso*, all 3 dimensions dilate/contract independently using their individual pressure components as the 3 driving forces.

For any of the styles except *xyz*, any of the independent pressure components (e.g. z in *xy*, or any dimension in *aniso*) can have their target pressures (both start and stop values) specified as NULL. This means that no pressure control is applied to that dimension so that the box dimension remains unchanged.

In some cases (e.g. for solids) the pressure (volume) and/or temperature of the system can oscillate undesirably when a Nose/Hoover barostat and thermostat is applied. The optional *drag* keyword will damp these oscillations, although it alters the Nose/Hoover equations. A value of 0.0 (no drag) leaves the Nose/Hoover formalism unchanged. A non−zero value adds a drag term; the larger the value specified, the greater the damping effect. Performing a short run and monitoring the pressure and temperature is the best way to determine if the drag term is working. Typically a value between 0.2 to 2.0 is sufficient to damp oscillations after a few periods.

For all pressure styles, the simulation box stays rectangular in shape. Parinello−Rahman boundary conditions (tilted box) are not implemented in LAMMPS.

For all styles, the *Pdamp* parameter operates like the *Tdamp* parameter, determining the time scale on which pressure is relaxed.

This fix supports the fix_modify options for *thermo* and *energy*. The former will print the contribution the fix makes to the energy of the system when thermodynamics is printed. The latter will add this contribution to the total potential energy (PotEng) so that energy conservation can be monitored.

Restrictions:

Any dimension being adjusted by this fix must be periodic. A dimension whose target pressures are specified as NULL can be non−periodic or periodic.

Related commands:

fix nve, fix nvt, fix nph, fix_modify

Default:

The keyword defaults are drag = 0.0 and dilate = all.

LAMMPS WWW Site − LAMMPS Documentation − LAMMPS Commands

---

fix nve command

Syntax:

```
fix ID group−ID nve
```

- ID, group−ID are documented in fix command
- nve = style name of this fix command

Examples:

```
fix 1 all nve
```

Description:

Perform constant NVE updates of position and velocity for atoms in the group each timestep. V is volume; E is energy. This creates a system trajectory consistent with the microcanonical ensemble.

Restrictions: none

Related commands:

fix nvt, fix npt

Default: none

LAMMPS WWW Site − LAMMPS Documentation − LAMMPS Commands

fix nve/gran command

Syntax:

```
fix ID group−ID nve/gran
```

- ID, group−ID are documented in fix command
- nve/gran = style name of this fix command

Examples:

```
fix 1 all nve/gran
```

Description:

Perform constant NVE updates each timestep on a group of atoms of atom style granular. V is volume; E is energy. Granular atoms store rotational information as well as position and velocity, so this integrator updates translational and rotational degrees of freedom due to forces and torques.

Restrictions: none

Can only be used with atom_style granular.

Related commands:

atom_style granular

Default: none

LAMMPS WWW Site − LAMMPS Documentation − LAMMPS Commands

fix nvt command

Syntax:

```
fix ID group−ID nvt Tstart Tstop Tdamp keyword value ...
```

* ID, group−ID are documented in fix command
* nvt = style name of this fix command
* Tstart,Tstop = desired temperature at start/end of run
* Tdamp = temperature damping parameter (time units)
* zero or more keyword/value pairs may be appended to the args
* keyword = *drag*

```
drag value = drag factor added to thermostat (0.0 = no drag)
```

Examples:

```
fix 1 all nvt 300.0 300.0 100.0
fix 1 all nvt 300.0 300.0 100.0 drag 0.2
```

Description:

Perform constant NVT integration to update positions and velocities each timestep for atoms in the group using a Nose/Hoover temperature thermostat. V is volume; T is temperature. This creates a system trajectory consistent with the canonical ensemble.

The desired temperature at each timestep is a ramped value during the run from *Tstart* to *Tstop*. The *Tdamp* parameter is specified in time units and determines how rapidly the temperature is relaxed. For example, a value of 100.0 means to relax the temperature in a timespan of (roughly) 100 time units (tau or fmsec or psec − see the units command).

In some cases (e.g. for solids) the temperature of the system can oscillate undesirably when a Nose/Hoover thermostat is applied. The optional *drag* keyword will damp these oscillations, although it alters the Nose/Hoover equations. A value of 0.0 (no drag) leaves the Nose/Hoover formalism unchanged. A non−zero value adds a drag term; the larger the value specified, the greater the damping effect. Performing a short run and monitoring the temperature is the best way to determine if the drag term is working. Typically a value between 0.2 to 2.0 is sufficient to damp oscillations after a few periods.

By default the temperature is computed only on the atoms in the fix group using the default temperature style. This can be changed by assigning a different temperature style to the fix via the fix_modify command.

This fix supports the fix_modify options for *thermo* and *energy*. The former will print the contribution the fix makes to the energy of the system when thermodynamics is printed. The latter will add this contribution to the total potential energy (PotEng) so that energy conservation can be monitored.

Restrictions: none

Related commands:

fix nve, fix npt, fix temp/rescale, fix langevin, fix_modify

Default:

The keyword defaults are drag = 0.0.

LAMMPS WWW Site − LAMMPS Documentation − LAMMPS Commands

fix planeforce command

Syntax:

```
fix ID group-ID planeforce x y z
```

- ID, group−ID are documented in fix command
- lineforce = style name of this fix command
- x y z = 3−vector that is normal to the plane

Examples:

```
fix hold boundary planeforce 1.0 0.0 0.0
```

Description:

Adjust the forces on each atom in the group so that it's motion will be in the plane specified by the normal vector (x,y,z). This is done by subtracting out components of force perpendicular to the plane.

If the initial velocity of the atom is 0.0 (or in the plane), then it should continue to move in the plane thereafter.

Restrictions: none

Related commands:

fix lineforce

Default: none

LAMMPS WWW Site − LAMMPS Documentation − LAMMPS Commands

fix print command

Syntax:

```
fix ID group-ID print N string
```

- ID, group−ID are documented in fix command
- print = style name of this fix command
- N = print every N steps
- string = text string to print with optional variable names

Examples:

```
fix extra all print 100 "Coords of marker atom = $x $y $z"
```

Description:

Print a text string to the screen and logfile every N steps during a simulation run. This can be used for diagnostic purposes or even as a debugging tool to monitor some quantity during a run. The text string must be a single argument, so it should be enclosed in double quotes if it is more than one word. If it contains variables ($a thru $z) it must be enclosed in double quotes to insure they are not evaluated when the input script is read, but will instead be evaluated when the string is printed.

See the variable command for a description of *equal* style variables which are the most useful ones to use with the fix print command, since they are evaluated afresh each timestep that the fix print line is output.

Note that if *equal*−style variables are used in the print line which contain thermo_style custom keywords for potential energy such as pe, eng, evdwl, ebond, etc, they will only be up−to−date on timesteps where thermodynamics are computed. Hence, if you output thermodynamics every 100 steps, but issue a fix print command with N=2 that contains such a variable, the printed value will only be current on timesteps that are a multiple of 100.

Restrictions: none

Related commands:

variable, print

Default: none

LAMMPS WWW Site − LAMMPS Documentation − LAMMPS Commands

fix rdf command

Syntax:

```
fix ID group−ID rdf N file Nbin itype1 jtype1 itype2 jtype2 ...
```

- ID, group−ID are documented in fix command
- rdf = style name of this fix command
- N = compute radial distribution function (RDF) every this many timesteps
- file = filename to write radial distribution funtion info to
- Nbin = number of RDF bins
- itypeN = central atom type for RDF pair N
- jtypeN = distribution atom type for RDF pair N

Examples:

```
fix 1 all rdf 500 rdf.out 100 1 1
fix 1 fluid rdf 10000 rdf.out 100 1 1 1 2 2 1 2 2
```

Description:

Compute the radial distribution function (RDF), also known as g(r), and coordination number every N steps. The RDF for each specified atom type pair is histogrammed in Nbin bins from distance 0 to Rc, where Rc = the maximum force cutoff for any pair of atom types. An atom pair only contributes to the RDF if

- both atoms are in the fix group
- the distance between them is within the maximum force cutoff
- their interaction is stored in the neighbor list

The latter will not be the case for bonded atoms (1−2, 1−3, 1−4 interactions within a molecular topology) if the pairwise weighting factor set by the special_bonds command is 0.0 for the 2 atoms.

The RDF statistics for each timestep are written to the specified file, as are the RDF values averaged over all timesteps.

Restrictions:

The RDF is not computed for distances longer than the force cutoff, since processors (in parallel) don't know atom coordinates for atoms further away than that distance. If you want an RDF for larger *r*, you'll need to post−process a dump file.

Related commands:

pair_style

Default: none

LAMMPS WWW Site − LAMMPS Documentation − LAMMPS Commands

---

fix rigid

Syntax:

```
fix ID group−ID rigid keyword values
```

- ID, group−ID are documented in fix command
- rigid = style name of this fix command
- keyword = *single* or *molecule* or *group*

```
single values = none
  molecule values = none
  group values = list of group IDs
```

Examples:

```
fix 1 clump rigid single
fix 1 polychains rigid molecule
fix 2 fluid rigid group clump1 clump2 clump3
```

Description:

Treat one or more sets of atoms as a rigid body. This means that each timestep the total force and torque on the rigid body is computed and the coordinates and velocities of the atoms are updated so that they move as a rigid body. This can be useful for freezing one or more portions of a large biomolecule, or for simulating a system of colloidal particles.

This fix updates the positions and velocities of the rigid atoms with a constant−energy time integration, so you should not update the same atoms via other fixes (e.g. nve, nvt, npt, temp/rescale, langevin).

For *single* the entire group of atoms is treated as one rigid body.

For *molecule*, each set of atoms in the group with a different molecule ID is treated as a rigid body.

For *group*, each of the listed groups is treated as a separate rigid body. Note that only atoms that are also in the fix group are included in each rigid body.

For computational efficiency, you should also turn off pairwise and bond interactions within each rigid body, as they no longer contribute to the motion. The neigh_modify exclude and delete_bonds commands are used to do this.

For computational efficiency, you should ideally define one rigid fix which includes all the desired rigid bodies. LAMMPS will allow multiple rigid fixes to be defined, but it is more expensive.

The degrees−of−freedom removed by rigid bodies are accounted for in temperature and pressure computations. Similary, the rigid body contribution to the pressure virial is also accounted for.

Restrictions:

This fix performs an MPI_Allreduce each timestep that is proportional in length to the number of rigid bodies. Hence it will not scale well in parallel if large numbers of rigid bodies are simulated.

If the atoms in a single rigid body initially straddle a periodic boundary, the input data file must define the image flags for each atom correctly, so that LAMMPS can "unwrap" the atoms into a valid rigid body.

Because this fix uses constant−energy integration, it means you cannot easily control the temperature of an ensemble of rigid bodies. You can try to use other fixes (langevin, temp/rescale) for this purpose, but the effects are not always satisfactory. If you are simulating a system that also contains non−rigid atoms (e.g. solvent), then you can thermostat those atoms and hope they will couple to the rigid bodies. The right solution is probably to enhance this fix to allow for direct temperature control of the rigid bodies.

Related commands:

delete_bonds, neigh_modify exclude

Default: none

LAMMPS WWW Site − LAMMPS Documentation − LAMMPS Commands

---

fix setforce command

Syntax:

```
fix ID group−ID setforce fx fy fz
```

- ID, group−ID are documented in fix command
- setforce = style name of this fix command
- fx,fy,fz = force component values

Examples:

```
fix freeze indenter setforce 0.0 0.0 0.0
fix 2 edge setforce NULL 0.0 0.0
```

Description:

Set each component of force on each atom in the group to the specified values fx,fy,fz. This erases all previously computed forces on the atom, though additional fixes could add new forces. This command can be used to freeze certain atoms in the simulation by zeroing their force, assuming their initial velocity zero.

Any of the fx,fy,fz values can be specified as NULL which means do not alter the force component in that dimension.

Restrictions: none

Related commands:

fix addforce, fix aveforce

Default: none

LAMMPS WWW Site − LAMMPS Documentation − LAMMPS Commands

---

fix shake style

Syntax:

```
fix ID group−ID shake tol iter N keyword values ...
```

- ID, group−ID are documented in fix command
- shake = style name of this fix command
- tol = accuracy tolerance of SHAKE solution
- iter = max # of iterations in each SHAKE solution
- N = print SHAKE statistics every this many timesteps (0 = never)
- one or more keyword/value pairs are appended
- keyword = *b* or *a* or *t* or *m*

```
b values = one or more bond types
  a values = one or more angle types
  t values = one or more atom types
  m value = one or more mass values
```

Examples:

```
fix 1 sub shake 0.0001 20 10 b 4 19 a 3 5 2
fix 1 sub shake 0.0001 20 10 t 5 6 m 1.0 a 31
```

Description:

Apply bond and angle constraints to specified bonds and angles in the simulation. This typically enables a longer timestep.

Each timestep the specified bonds and angles are reset to their equilibrium lengths and angular values. This is done by applying an additional constraint force so that the new positions preserve the desired atom separations. The equations for the additional force are solved via an iterative method that typically converges to an accurate solution in a few iterations. The desired tolerance (e.g. 1.0e−4 = 1 part in 10000) and maximum

# of iterations are specified as arguments. Setting the N argument will print statistics to the screen and log file about regarding the lengths of bonds and angles that are being constrained. Small delta values mean SHAKE is doing a good job.

In LAMMPS, only small clusters of atoms can be constrained. This is so the constraint calculation for a cluster can be performed by a single processor, to enable good parallel performance. A cluster is defined as a central atom connected to others in the cluster by constrained bonds. LAMMPS allows for the following kinds of clusters to be constrained: one central atom bonded to 1 or 2 or 3 atoms, or one central atom bonded to 2 others and the angle between the 3 atoms also constained. This means water molecules or CH2 or CH3 groups may be constrained, but not all the C−C backbone bonds of a long polymer chain.

The *b* keyword lists bond types that will be constrained. The *t* keyword lists atom types. All bonds connected to an atom of the specified type will be constrained. The *m* keyword lists atom masses. All bonds connected to atoms of the specified masses will be constrained (within a fudge factor of MASSDELTA specified in fix_shake.cpp). The *a* keyword lists angle types. If both bonds in the angle are constrained then the angle will also be constrained if its type is in the list.

For all keywords, a particular bond is only constrained if both atoms in the bond are in the group specified with the SHAKE fix.

The degrees−of−freedom removed by SHAKE bonds and angles are accounted for in temperature and pressure computations. Similary, the SHAKE contribution to the pressure virial is also accounted for.

Restrictions:

For computational efficiency, there can only be one shake fix defined in a simulation.

If you use a tolerance that is too large or a max−iteration count that is too small, the constraints will not be enforced very strongly, which can lead to poor energy conservation. You can test for this in your system by running a constant NVE simulation with a particular set of SHAKE parameters and monitoring the energy versus time.

Related commands: none

Default: none

LAMMPS WWW Site − LAMMPS Documentation − LAMMPS Commands

---

fix spring command

Syntax:

```
fix ID group−ID spring keyword values
```

- ID, group−ID are documented in fix command
- spring = style name of this fix command
- keyword = *tether* or *couple*

```
tether values = K x y z R0
    K = spring constant (force/distance units)
    x,y,z = point to which spring is tethered
    R0 = equilibrium distance from tether point (distance units)
  couple values = group−ID1 group−ID2 K x y z R0
    group−ID1,group−ID2 = two groups to couple together with a spring
    K = spring constant (force/distance units)
    x,y,z = direction of spring
    R0 = equilibrium distance of spring (distance units)
```

Examples:

```
fix pull ligand spring tether 50.0 0.0 0.0 0.0 0.0
fix pull ligand spring tether 50.0 0.0 0.0 0.0 5.0
fix pull ligand spring tether 50.0 NULL NULL 2.0 3.0
fix 5 lipids spring couple bilayer1 bilayer2 100.0 NULL NULL 10.0 0.0
fix longitudinal all spring couple pore ion 100.0 NULL NULL −20.0 0.0
fix radial all spring couple pore ion 100.0 0.0 0.0 NULL 5.0
```

Description:

Apply a spring force to a group of atoms or between two groups of atoms. This is useful for applying an umbrella force to a small molecule or lightly tethering a large group of atoms (e.g. all the solvent or a large molecule) to the center of the simulation box so that it doesn't wander away over the course of a long simulation. It can also be used to hold the centers of mass of two groups of atoms at a given distance or orientation with respect to each other.

The *tether* style attaches a spring between a fixed point *x,y,z* and the center of mass of the fix group of atoms. The equilibrium position of the spring is R0. At each timestep the distance R from the center of mass of the group of atoms to the tethering point is computed, taking account of wrap−around in a periodic simulation box. A restoring force of magnitude K (R − R0) Mi / M is applied to each atom in the group where *K* is the spring constant, Mi is the mass of the atom, and M is the total mass of all atoms in the group. Note that *K* thus represents the total force on the group of atoms, not a per−atom force.

The *couple* style links two groups of atoms together. Only atoms that are also in the fix group are considered to be part of either of the groups. The groups are coupled together by a spring that is at equilibrium when the two groups are displaced by a vector *x,y,z* with respect to each other and at a distance R0 from that displacement. Note that *x,y,z* is the equilibrium displacement of group 2 relative to group 1. Thus (1,1,0) is a different spring than (−1,−1,0). When the relative positions and distance between the two groups are not in equilibrium, the same spring force described above is applied to atoms in each of the two groups.

For both the *tether* and *couple* styles, any of the x,y,z values can be specified as NULL which means do not include that dimension in the distance calculation or force application.

The first example above pulls the ligand towards the point (0,0,0). The second example holds the ligand near the surface of a sphere of radius 5 around the point (0,0,0). The third example holds the ligand a distance 3 away from the z=2 plane (on either side).

The fourth example holds 2 bilayers a distance 10 apart in z. For the last two examples, imagine a pore (a slab of atoms with a cylindrical hole cut out) oriented with the pore axis along z, and an ion moving within the pore. The fifth example holds the ion a distance of −20 below the z = 0 center plane of the pore (umbrella sampling). The last example holds the ion a distance 5 away from the pore axis (assuming the center−of−mass of the pore in x,y is the pore axis).

Restrictions: none

Related commands:

fix drag

Default: none

LAMMPS WWW Site − LAMMPS Documentation − LAMMPS Commands

---

fix temp/rescale command

Syntax:

```
fix ID group−ID temp/rescale N Tstart Tstop window fraction
```

- ID, group−ID are documented in fix command
- temp/rescale = style name of this fix command
- N = perform rescaling every N steps
- Tstart,Tstop = desired temperature at start/end of run (temperature units)
- window = only rescale if temperature is outside this window (temperature units)
- fraction = rescale to target temperature by this fraction

Examples:

```
fix 3 flow temp/rescale 100 1.0 1.1 0.02 0.5
```

Description:

Reset the temperature of a group of atoms by explicitly rescaling their velocities.

Rescaling is performed every N timesteps. The target temperature is a ramped value between the *Tstart* and *Tstop* temperatures at the beginning and end of the run. Rescaling is only performed if the difference between the current and desired temperatures is greater than the *window* value. The amount of rescaling that is applied is a *fraction* (from 0.0 to 1.0) of the difference between the actual and desired temperature. E.g. if *fraction* = 1.0, the temperature is reset to exactly the desired value.

The way that temperature is computed by this fix can be changed by using the fix_modify command.

A temp/rescale fix does not update the coordinates of its atoms. It is normally used with a fix of style *nve* that does that. A temp/rescale fix should not normally be used on atoms that also have their temperature controlled by another fix − e.g. a nvt or langevin fix.

This fix supports the fix_modify options for *thermo* and *energy*. The former will print the contribution the fix makes to the energy of the system when thermodynamics is printed. The latter will add this contribution to the total potential energy (PotEng) so that energy conservation can be monitored.

Restrictions: none

Related commands:

fix langevin, fix nvt, fix_modify

Default: none

LAMMPS WWW Site − LAMMPS Documentation − LAMMPS Commands

fix tmd command

Syntax:

```
fix ID group−ID tmd rho_final file1 N file2
```

- ID, group−ID are documented in fix command
- tmd = style name of this fix command
- rho_final = desired value of rho at the end of the run (distance units)
- file1 = filename to read target structure from
- N = dump TMD statistics every this many timesteps, 0 = no dump
- file2 = filename to write TMD statistics to (only needed if N > 0)

Examples:

```
fix 1 all nve
fix 2 tmdatoms tmd 1.0 target_file 100 tmd_dump_file
```

Description:

Perform targeted molecular dynamics (TMD) on a group of atoms. A holonomic constraint is used to force the atoms to move towards (or away from) the target configuration. The parameter "rho" is monotonically decreased (or increased) from its initial value to rho_final at the end of the run. Rho has distance units and is a measure of the root−mean−squared distance (RMSD) between the current configuration of the atoms in the group and the target coordinates listed in file1. Thus a value of rho_final = 0.0 means move the atoms all the way to the final structure during the course of the run.

The format of the target file1 is as follows:

```
     0.0 25.0 xlo xhi
     0.0 25.0 ylo yhi
     0.0 25.0 zlo zhi
     125     24.97311   1.69005     23.46956 0 0 −1
     126     1.94691    2.79640     1.92799  1 0 0
     127     0.15906    3.46099     0.79121  1 0 0
     ...
```

The first 3 lines may or may not be needed, depending on the format of the atoms to follow. If image flags are included with the atoms, the 1st 3 lo/hi lines must appear in the file. If image flags are not included, the 1st 3 lines should not appear. The 3 lines contain the simulation box dimensions for the atom coordinates, in the same format as in a LAMMPS data file (see the read_data command).

The remaining lines each contain an atom ID and its target x,y,z coordinates. The atom lines (all or none of them) can optionally be followed by 3 integer values: nx,ny,nz. For periodic dimensions, they specify which image of the box the atom is considered to be in, i.e. a value of N (positive or negative) means add N times the box length to the coordinate to get the true value.

The atom lines can be listed in any order, but every atom in the group must be listed in the file. Atoms not in the fix group may also be listed; they will be ignored.

TMD statistics are written to file2 every N timesteps, unless N is specified as 0, which means no statistics.

The atoms in the fix tmd group should be integrated (via a fix nve, nvt, npt) along with other atoms in the system.

Restarts can be used with a fix tmd command. For example, imagine a 10000 timestep run with a rho_initial = 11 and a rho_final = 1. If a restart file was written after 2000 time steps, then the configuration in the file would have a rho value of 9. A new 8000 time step run could be performed with the same rho_final = 1 to complete the conformational change at the same transition rate. Note that for restarted runs, the name of the TMD statistics file should be changed to prevent it being overwritten.

For more information about TMD, see (Schlitter1) and (Schlitter2).

Restrictions:

All TMD fixes must be listed in the input script after all integrator fixes (nve, nvt, npt) are applied. This ensures that atoms are moved before their positions are corrected to comply with the constraint.

Atoms that have a TMD fix applied should not be part of a group to which a SHAKE fix is applied. This is because LAMMPS assumes there are not multiple competing holonomic constraints applied to the same atoms.

Related commands: none

Default: none

(Schlitter1) Schlitter, Swegat, Mulders, "Distance−type reaction coordinates for modelling activated processes", J Molecular Modeling, 7, 171−177 (2001).

(Schlitter2) Schlitter and Klahn, "The free energy of a reaction coordinate at multiple constraints: a concise formulation", Molecular Physics, 101, 3439−3443 (2003).

LAMMPS WWW Site − LAMMPS Documentation − LAMMPS Commands

---

fix uniaxial command

Syntax:

```
fix ID group−ID uniaxial N keyword dim amount
```

- ID, group−ID are documented in fix command
- uniaxial = style name of this fix command
- N = perform uniaxial rescaling every this many timesteps
- dim = *x* or *y* or *z*
- strain = uniaxial strain in dim (2.0 = 2x larger)

Examples:

```
fix 1 all uniaxial 100 x 2.0
```

Description:

Enable a uniaxial dilation/contraction of the simulation box during a simulation. For example if the direction is X and the strain is 2, then the final box size is 2L, L/sqrt(2), L/sqrt(2), where L**3 is a cube with the same volume as the initial box, which need not be cubic.

The chosen direction is ramped linearly during the course of the simulation. If the two perpendicular box sizes are equal then the deformation pathway is uniaxial at each timestep. If the two perpendicular box length sizes differ, then their aspect ratio will be linearily ramped down to 1. Irregardless of the initial box shape the total volume is constant during the deformation. Additional details provided by Carsten Svaneborg (Max Planck Institute for Complex Systems, Dresden, Germany) who authored this fix, are at the bottom of this page.

The initial simulation box boundaries at the beginning of a run are specified by the create_box or read_data or read_restart command used to setup the simulation, or they are the values at the end of the previous run. Every Nth timestep during the run, the various dimensions are expanded or contracted. The coordinates of all atoms in the group are also scaled to the new box size.

Restrictions:

To use this fix, all dimensions of the system must be periodic.

Related commands: none

Default: none

Extra Notes:

The uniaxial deformation is performed as follows:

For notational simplicity the deformation is assumed to be in the X direction with final strain lambda. alpha denotes an arbitrary Cartesian direction.

The initial strain is obtained from box dimensions:

```
lambdai_alpha = Box(alpha)/power(Box(0)*Box(1)*Box(2),1/3)
```

The final strain lambda in dir is specified:

```
lambdaf_x = lambda
lambda_y = lambda_z = 1/sqrt(lambda)
```

Volume conservation implies lambda_x(t)*lambda_y(t)lambda_z(t) = 1.0 for all times. Rather than time, let delta is denotes reduced time in the interval from 0 to 1.

We want a linear ramp in the specified strain component, such that MD time steps and uniaxial strain are linearly related:

```
lambda_x(delta) = lambdai_x (1-delta) + lambdaf_x
```

The problem that remains is to choose a deformation pathway for lambda_y(delta) and lambda_z(delta) that agrees with the initial and final strains, and at all times conserves volume. Secondly the pathway should be symmetric if the box has yz symmetry.

In the case where the initial box is symmetric in yz, this follows from volume conservation:

```
lambda_y(d) = lambda_z(d) = 1/sqrt(lambda_x(d))
```

However, in general the initial box dimensions in the y and z directions need not be the same so assume a relation:

```
lambda_y(d) = alpha(d)lambda_z(d)
```

From volume conservation it follows that

```
lambda_y(d) = sqrt(alpha(d)/lambda_x(d))
lambda_z(d) = 1/sqrt(alpha(d)*lambda_x(d))
```

The asymmetry parameter has to fulfill the following boundary conditions:

```
initial alpha(d=0) = lambdai_y/lambdai_z
final alpha(d=1) = 1
```

Any interpolation that does this will by define a continuous volume conserving deformation from the initial to the desired final state. The freedom of choice here is e.g. to relax the asymetry of the box very fast, and then slowly elongate along x, or to do this more slowly.

The choice used in the code is:

```
alpha(d) = lambdai_y/lambdai_z (1-d) + d
```

Note in some cases like strain −> strain the perpendicular strains do not follow a monotonic curve.

---

fix viscous command

Syntax:

```
fix ID group-ID viscous coeff
```

- ID, group−ID are documented in fix command
- viscous = style name of this fix command
- coeff = damping coefficient (unitless)

Examples:

```
fix 1 flow viscous 0.1
```

Description:

Add a viscous damping force to atoms in the group that is proportional to the velocity of the atom. This is useful for draining the kinetic energy from the system in a controlled fasion. The damping force F is given by F = − coeff * velocity. The larger the coefficient, the faster the kinetic energy is reduced.

Restrictions: none

Related commands: none

Default: none

fix volume/rescale command

Syntax:

```
fix ID group−ID volume/rescale N keyword args ...
```

- ID, group−ID are documented in fix command

- volume/rescale = style name of this fix command
- N = perform volume rescaling every this many timesteps
- one or more keyword/value pairs may be appended to the args
- keyword = *x* or *y* or *z*

```
x, y, z args = lo,hi = desired simulation box boundaries
    at end of run
```

Examples:

```
fix 1 all volume/rescale 100 x −9.0 9.0 z −5.0 5.0
```

Description:

Enable a volume (density) change during a simulation. Each of the 3 box dimensions is controlled separately. Any dimension being varied by this command must be periodic − see the boundary command. Dimensions not varied by this command can be periodic or non−periodic. The volume associated with an unspecified dimension can also be controlled by a fix npt command.

The initial simulation box boundaries at the beginning of a run are specified by the create_box or read_data or read_restart command used to setup the simulation, or they are the values at the end of the previous run. The desired simulation box boundaries at the end of the run are given by the *lo* and *hi* arguments. Every Nth timestep during the run, the simulation box is expanded or contracted to a ramped value between the initial and final values. The coordinates of all atoms in the group are also scaled to the new box size.

Restrictions:

Any dimension being varied by this fix must be periodic.

Related commands: none

Default: none

fix wall/gran command

Syntax:

```
fix ID group−ID wall/gran wallstyle args keyword values ...
```

- ID, group−ID are documented in fix command
- wall/gran = style name of this fix command
- style = *xplane* or *yplane* or *zplane* or *zcylinder*
- args = list of arguments for a particular style

```
  xplane or yplane or zplane args = lo hi gamma xmu
      lo, hi = position of lower and upper plane (either can be NULL)
      gamman = damping coeff for normal direction collisions with wall
      xmu = friction coeff for the wall
    zcylinder args = radius gamma xmu
      radius = cylinder radius (distance units)
      gamman = damping coeff for normal direction collisions with wall
      xmu = friction coeff for the wall
```

- zero or more keyword/value pairs may be appended to args

```
    keyword = wiggle
      values = dim amplitude period
        dim = x or y or z
        amplitude = size of oscillation (distance units)
        period = time of oscillation (time units)
```

Examples:

```
fix 1 all wall/gran xplane −10.0 10.0 50.0 0.5
fix 2 all wall/gran zcylinder 15.0 50.0 0.5 wiggle z 3.0 2.0
fix 1 all wall/gran zplane 0.0 NULL 100.0 0.5
```

Description:

Bound the simulation domain of a granular system with a frictional wall. All particles in the group interact with the wall when they are close enough to touch it.

The *wallstyle* can be planar or cylindrical. The 3 planar options specify a pair of walls in a dimension. Wall positions are given by *lo* and *hi*. Either of the values can be specified as NULL if a single wall is desired. For a *zcylinder* wallstyle, the cylinder's axis is at x = y = 0.0, and the radius of the cylinder is specified. For all wallstyles, a damping and friction coefficient for particle−wall interactions are also specified.

Optionally, a wall can be oscillated, similar to the oscillations of frozen particles specified by the fix_wiggle command. This is useful in packing simulations of granular particles. If the keyword *wiggle* is appended to the argument list, then a dimension for the motion, as well as it's *amplitude* and *period* is specified. Each timestep, the position of the wall in the appropriate *dim* is set according to this equation:

```
position = pos0 + A − A cos (omega * delta)
```

where *pos0* is the position at the time the fix was specified, *A* is the *amplitude*, *omega* is 2 PI / *period*, and *delta* is the elapsed time since the fix was specified. The velocity of the wall is also set to the derivative of this expression.

Restrictions:

A zcylinder wall can only be oscillated in the z dimension. This fix can only be used with atom_style granular.

Related commands:

fix_wiggle

Default: none

fix wall/93 command

Syntax:

```
fix ID group−ID wall/lj93 style coord epsilon sigma cutoff
```

- ID, group−ID are documented in fix command
- wall/lj93 = style name of this fix command
- style = *xlo* or *xhi* or *ylo* or *yhi* or *zlo* or *zhi*
- coord = position of wall
- epsilon = Lennard−Jones epsilon for wall−particle interaction
- sigma = Lennard−Jones sigma for wall−particle interaction
- cutoff = distance from wall at which wall−particle interaction is cut off

Examples:

```
fix wallhi all wall/lj93 xhi 10.0 1.0 1.0 1.12
```

Description:

Bound the simulation domain with a Lennard−Jones wall that encloses the atoms. The energy E of a wall−particle interactions is given by the 9−3 potential

$$E = \epsilon \left[\frac{2}{15} \left(\frac{\sigma}{r} \right)^9 - \left(\frac{\sigma}{r} \right)^3 \right] \qquad r < r_c$$

where *r* is the distance from the particle to the wall *coord*, and epsilon and sigma are the usual LJ parameters. Rc is the cutoff value specified in the command. This interaction is derived by integrating over a 3d half−lattice of Lennard−Jones 12−6 particles.

This fix supports the fix_modify options for *thermo* and *energy*. The former will print the contribution the fix makes to the energy of the system when thermodynamics is printed. The latter will add this contribution to the total potential energy (PotEng).

Restrictions: none

Related commands: none

Default: none

LAMMPS WWW Site − LAMMPS Documentation − LAMMPS Commands

---

fix wiggle command

Syntax:

fix ID group−ID wiggle dim amplitude period

- ID, group−ID are documented in fix command
- wiggle = style name of this fix command
- dim = *x* or *y* or *z*
- amplitude = size of oscillation (distance units)
- period = time of oscillation (time units)

Examples:

```
fix 1 frozen wiggle 3.0 0.5
```

Description:

Move a group of atoms in a sinusoidal oscillation. This is useful in granular simulations when boundary atoms are wiggled to induce packing of the dynamic atoms. The dimension *dim* of movement is specified as is the *amplitude* and *period* of the oscillations. Each timestep the *dim* coordinate of each atom is set to

```
coord = coord0 + A − A cos (omega * delta)
```

where *coord0* is the coordinate at the time the fix was specified, *A* is the *amplitude*, *omega* is 2 PI / *period*, and *delta* is the elapsed time since the fix was specified. The velocity of the atom is set to the derivative of this expression.

Restrictions: none

Related commands: none

Default: none

LAMMPS WWW Site − LAMMPS Documentation − LAMMPS Commands

---

group command

Syntax:

```
group ID style args
```

- ID = user−defined name of the group
- style = *region* or *type* or *id* or *molecule* or *subtract* or *union* or *intersect*

```
region args = region−ID
  type or id or molecule
    args = one or more atom types, atom IDs, or molecule IDs
    args = logical value
      logical = "" or ">="
      value = an atom type or atom ID or molecule ID (depending on style)
    args = logical value1 value2
      logical = ""
      value1,value2 = atom types or atom IDs or molecule IDs
        (depending on      style)
  subtract args = two or more group IDs
  union args = one or more group IDs
  intersect args = two or more group IDs
```

Examples:

```
group edge region regstrip
group water type 3 4
group sub id <= 150
group polyA molecule  50 250
group boundary subtract all a2 a3
group boundary union lower upper
group boundary intersect upper flow
```

Description:

Identify a collection of atoms as belonging to a group. The group ID can then be used in other commands such as fix, velocity, dump, or temperature to act on the atoms together.

If the group ID already exists, the group command adds the specified atoms to the group.

The *region* style puts all atoms in the region volume into the group. Note that this is a static one−time assignment. The atoms remain assigned (or not assigned) to the group even in they later move out of the region volume.

The *type*, *id*, and *molecule* styles put all atoms with the specified atom types, atom IDs, or molecule IDs into the group. These 3 styles can have their arguments specified in one of two formats. The 1st format is a list of values (types or IDs). For example, the 2nd command in the examples above, puts all atoms of type 3 or 4 into the group named *water*. The 2nd format is a *logical* followed by one or two values (type or ID). The 5 valid logicals are listed above. All the logicals except take a single argument. The 3rd example above adds all atoms with IDs from 1 to 150 to the group named *sub*. The logical means "between" and takes 2 arguments. The 4th example above adds all atoms belonging to molecules with IDs from 50 to 250 (inclusive) to the group named polyA.

The *subtract* style takes a list of two or more existing group names as arguments. All atoms that belong to the

1st group, but not to any of the other groups are added to the specified group.

The *union* style takes a list of one or more existing group names as arguments. All atoms that belong to any of the listed groups are added to the specified group.

The *intersect* style takes a list of two or more existing group names as arguments. Atoms that belong to every one of the listed groups are added to the specified group.

A group with the ID *all* is predefined. All atoms belong to this group.

Restrictions:

There can be no more than 32 defined groups, including "all".

Related commands:

region, fix, velocity, dump, temperature

Default:

All atoms belong to the "all" group.

LAMMPS WWW Site − LAMMPS Documentation − LAMMPS Commands

---

improper_coeff command

Syntax:

```
improper_coeff N args
```

- N = improper type (see asterik form below)
- args = coefficients for one or more improper types

Examples:

```
improper_coeff 1 300.0 0.0
improper_coeff * 80.2 −1 2
improper_coeff *4 80.2 −1 2
```

Description:

Specify the force field coefficients for one or more improper types. The number and meaning of the coefficients depends on the improper style. As described below, improper coefficients can also be set in the data file read by the read_data command or in a restart file.

N can be specified in one of two ways. An explicit numeric value can be used, as in the 1st example above. Or a wild−card asterik can be used to set the coefficients for multiple improper types. This takes the form "*" or "*n" or "n*" or "m*n". If N = the number of improper types, then an asterik with no numeric values means all types from 1 to N. A leading asterik means all types from 1 to n (inclusive). A trailing asterik means all types from n to N (inclusive). A middle asterik means all types from m to n (inclusive).

Note that using 2 improper_coeff commands for the same improper type is perfectly valid. For example, these commands set the coeffs for all improper types, then overwrite the coeffs for just improper type 2:

```
improper_coeff * 300.0 0.0
improper_coeff 2 50.0 0.0
```

A line in a data file that specifies improper coefficients uses the exact same format as the arguments of the improper_coeff command in an input script, except that wild−card asteriks should not be used since coefficients for all N types are listed in the file. For example, under the "Improper Coeffs" section of a data file, the line that corresponds to the 1st example above would be listed as

```
1 300.0 0.0
```

See the improper_style command for more discussion of the formulas that use these coefficients. The units of each coefficient are shown in parenthesis.

$$\begin{aligned} E &= E_i + E_{aa} \\ E_i &= K\left[\frac{\chi_{ijkl} + \chi_{kjli} + \chi_{ljik}}{3} - \chi_0\right]^2 \\ E_{aa} &= M_1(\theta_{ijk} - \theta_1)(\theta_{kjl} - \theta_3) + \\ &\quad M_2(\theta_{ijk} - \theta_1)(\theta_{ijl} - \theta_2) + \\ &\quad M_3(\theta_{ijl} - \theta_2)(\theta_{kjl} - \theta_3) \end{aligned}$$

For style *class2,* only coefficients for the Ei formula can be specified in the input script. These are the 2 coefficients:

- K (energy/radian^2)
- X0 (degrees)

X0 is specified in degrees, but LAMMPS converts it to radians internally; hence the units of K are in energy/radian^2.

Coefficients for the Eaa formula must be specified in the data file. For the Eaa formula, the coefficients are listed under a "AngleAngle Coeffs" heading and each line lists 6 coefficients:

- M1 (energy/distance)
- M2 (energy/distance)
- M3 (energy/distance)
- theta1 (degrees)
- theta2 (degrees)
- theta3 (degrees)

The theta values are specified in degrees, but LAMMPS converts them to radians internally; hence the units of M are in energy/radian^2.

$$E = K[1 + d\cos(n\phi)]$$

For style *cvff*, specify 3 coefficients:

- K (energy)
- d (+1 or −1)
- n (0,1,2,3,4,6)

$$E = K(\chi - \chi_0)^2$$

For style *harmonic*, specify 2 coefficients:

- K (energy/radian^2)
- X0 (degrees)

X0 is specified in degrees, but LAMMPS converts it to radians internally; hence the units of K are in energy/radian^2.

Restrictions:

This command must come after the simulation box is defined by a read_data, read_restart, or create_box command.

An improper style must be defined before any improper coefficients are set, either in the input script or in a data file.

Related commands:

improper_style

Default: none

LAMMPS WWW Site − LAMMPS Documentation − LAMMPS Commands

---

improper_style command

Syntax:

```
improper_style style
```

- style = *none* or *class2* or *cvff* or *harmonic*

Examples:

```
improper_style harmonic
improper_style cvff
```

Description:

Set the formula LAMMPS will use to compute improper interactions between quadruplets (trigonal centers) of atoms. The list of atom quadruplets is specified in the data or restart file and is read in by a read_data or read_restart command. The coefficients for the formula for each improper type can also be specified in those files or by the improper_coeff command.

A style of *none* means improper forces are not computed, even if impropers are defined.

The *class2* style uses the potential

$$\begin{aligned} E &= E_i + E_{aa} \\ E_i &= K[\frac{\chi_{ijkl} + \chi_{kjli} + \chi_{ljik}}{3} - \chi_0]^2 \\ E_{aa} &= M_1(\theta_{ijk} - \theta_1)(\theta_{kjl} - \theta_3) + \\ & \quad M_2(\theta_{ijk} - \theta_1)(\theta_{ijl} - \theta_2) + \\ & \quad M_3(\theta_{ijl} - \theta_2)(\theta_{kjl} - \theta_3) \end{aligned}$$

where Ei is the improper term and Eaa is an angle−angle term. The chi used in Ei is an average over 3 possible chi orientations. The subscripts on the various theta's refer to different combinations of atoms i,j,k,l used to form the angle; theta1, theta2, theta3 are the equilibrium positions of those angles. K, Mn, chi0, theta1, theta2, and theta3 are coefficients defined for each improper type.

The *cvff* style uses the potential

$$E = K[1 + d\cos(n\phi)]$$

where phi is the Wilson out−of−plane angle. K, d, and n are coefficients defined for each improper type.

The *harmonic* style uses the potential

$$E = K(\chi - \chi_0)^2$$

where X is the improper angle, X0 is its equilibrium value, and K is a prefactor. Note that the usual 1/2 factor is included in K. K and X0 are coefficients defined for each improper type.

Restrictions:

Improper styles can only be set for atom_style choices that allow impropers to be defined.

Improper styles are part of the "molecular" package. They are only enabled if LAMMPS was built with that package. See the Making LAMMPS section for more info.

Related commands:

improper_coeff

Default:

```
improper_style none
```

LAMMPS WWW Site − LAMMPS Documentation − LAMMPS Commands

---

include command

Syntax:

```
include file
```

- file = filename of new input script to switch to

Examples:

```
include newfile
include in.run2
```

Description:

This command opens a new input script file and begins reading LAMMPS commands from that file. When the new file is finished, the original file is returned to. Include files can be nested as deeply as desired. If input script A includes script B, and B includes A, then LAMMPS could run for a long time.

If the filename is a variable (see the variable command), different processor partitions can run different input scripts.

Restrictions: none

Related commands:

variable, jump

Default: none

LAMMPS WWW Site − LAMMPS Documentation − LAMMPS Commands

---

jump command

Syntax:

```
jump file label
```

- file = filename of new input script to switch to
- label = optional label within file to jump to

Examples:

```
jump newfile
jump in.run2 runloop
```

Description:

This command closes the current input script file, opens the file with the specified name, and begins reading LAMMPS commands from that file. The original file is not returned to, although by using multiple jump commands it is possible to chain from file to file or back to the original file.

Optionally, if a 2nd argument is used, it is treated as a label and the new file is scanned (without executing commands) until the label is found, and commands are executed from that point forward. This can be used to loop over a portion of the input script, as in this example. These commands perform 10 runs, each of 10000 steps, and create 10 dump files named file.1, file.2, etc. The next command is used to exit the loop after 10 iterations. When the "a" variable has been incremented for the 10th time, it will cause the next jump command to be skipped.

```
variable a loop 10
label loop
dump 1 all atom 100 file.$a
run 10000
undump 1
next a
jump in.lj loop
```

If the jump *file* argument is a variable, the jump command can be used to cause different processor partitions to run different input scripts. In this example, LAMMPS is run on 40 processors, with 4 partions of 10 procs each. An in.file containing the example variable and jump command will cause each partition to run a different simulation.

```
mpirun -np 40 lmp_ibm -partition 4x10 -in in.file

variable f world script.1 script.2 script.3 script.4
jump $f
```

Restrictions:

If you jump to a file and it does not contain the specified label, LAMMPS will come to the end of the file and exit.

Related commands:

variable, include, label, next

Default: none

LAMMPS WWW Site − LAMMPS Documentation − LAMMPS Commands

kspace_modify command

Syntax:

```
kspace_modify keyword value ...
```

- one or more keyword/value pairs may be listed
- keyword = *mesh* or *order* or *gewald* or *slab*

```
    mesh value = x y z
        x,y,z = PPPM FFT grid size in each dimension
      order value = N
        N = grid extent of Gaussian for PPPM mapping of each charge
      gewald value = r
        r = PPPM G-ewald parameter
      slab value = volfactor
        volfactor = ratio of the total extended volume used in the
          2d approximation compared with the volume of the simulation domain
```

Examples:

```
kspace_modify mesh 24 24 30 order 6
kspace_modify slab 3.0
```

Description:

Set parameters used by the kspace solvers defined by the kspace_style command. Not all parameters are relevant to all kspace styles.

The *mesh* keyword sets the 3d FFT grid size for kspace style pppm. Each dimension must be factorizable into powers of 2, 3, and 5. When this option is not set, the PPPM solver chooses its own grid size, consistent with the user−specified accuracy and pairwise cutoff. Values for x,y,z of 0,0,0 unset the option.

The *order* keyword determines how many grid spacings an atom's charge extends when it is mapped to the FFT grid in kspace style pppm. The default for this parameter is 5, which means each charge spans 5 grid cells in each dimension.

The *gewald* keyword sets the value of the PPPM G−ewald parameter. Without this setting, LAMMPS chooses the parameter automatically as a function of cutoff, precision, grid spacing, etc. This means it can vary from one simulation to the next which may not be desirable for matching a KSpace solver to a pre−tabulated pairwise potential. This setting can also be useful if PPPM fails to choose a good grid spacing and G−ewald parameter automatically. If the value is set to 0.0, LAMMPS will choose the G−ewald parameter automatically.

The *slab* keyword allows an Ewald or PPPM solver to be used for a systems that are periodic in x,y but non−periodic in z − a boundary setting of "boundary p p f". This is done by treating the system as if it were periodic in z, but inserting empty volume between atom slabs and removing dipole inter−slab interactions so that slab−slab interactions are effectively turned off. The volfactor value sets the ratio of the extended dimension in z divided by the actual dimension in z. The recommended value is 3.0. A larger value is inefficient; a smaller value introduces unwanted slab−slab interactions. The use of fixed boundaries in z means that the user must prevent particle migration beyond the initial z−bounds, typically by providing a wall−style fix.

Restrictions: none

Related commands:

kspace_style, boundary

Default:

The option defaults are mesh = 0 0 0, order = 5, gewald = 0.0, and slab = 1.0.

kspace_style command

Syntax:

```
kspace_style style value
```

- style = *none* or *ewald* or *pppm*

```
none value = none
  ewald value = precision
    precision = desired accuracy
  pppm value = precision
    precision = desired accuracy
  pppm/tip4p value = precision
    precision = desired accuracy
```

Examples:

```
kspace_style pppm 1.0e−4
kspace_style none
```

Description:

Define a K−space solver for LAMMPS to use each timestep to compute long−range Coulombic interactions. When such a solver is used in conjunction with an appropriate pair style, the cutoff for Coulombic interactions is effectively infinite; each charge in the system interacts with charges in an infinite array of periodic images of the simulation domain.

The *ewald* style performs a standard Ewald summation as described in any solid−state physics text.

The *pppm* style invokes a particle−particle particle−mesh solver (Hockney) which maps atom charge to a 3d mesh, uses 3d FFTs to solve Poisson's equation on the mesh, then interpolates electric fields on the mesh points back to the atoms. It is closely related to the particle−mesh Ewald technique (PME) (Darden) used in AMBER and CHARMM. The cost of traditional Ewald summation scales as N^(3/2) where N is the number of atoms in the system. The PPPM solver scales as Nlog(N) due to the FFTs, so it is almost always a faster choice (Pollock).

The *pppm/tip4p* style is identical to the *pppm* style except that it adds a charge at the massless 4th site in each TIP4P water molecule.

When a kspace style is used, a pair style that includes the short−range correction to the pairwise Coulombic forces must also be selected. These styles are ones that have a *coul/long* in their style name.

A precision value of 1.0e−4 means one part in 10000. This setting is used in conjunction with the pairwise cutoff to determine the number of K−space vectors for style *ewald* or the FFT grid size for style *pppm*.

Restrictions:

A simulation must be 3d and periodic in all dimensions to use an Ewald or PPPM solver. The only exception is if the slab option is set with kspace_modify, in which case the xy dimensions must be periodic and the z dimension must be non−periodic.

Kspace styles are part of the "kspace" package. They are only enabled if LAMMPS was built with that package. See the Making LAMMPS section for more info.

When using a pairwise TIP4P potential, you must use kspace style *pppm/tip4p* and vice versa.

Related commands:

kspace_modify, pair_style lj/cut/coul/long, pair_style lj/charmm/coul/long

Default:

```
kspace_style none
```

(Darden) Darden, York, Pedersen, J Chem Phys, 98, 10089 (1993).

(Hockney) Hockney and Eastwood, Computer Simulation Using Particles, Adam Hilger, NY (1989).

(Pollock) Pollock and Glosli, Comp Phys Comm, 95, 93 (1996).

---

label command

Syntax:

```
label ID
```

- ID = string used as label name

Examples:

```
label xyz
label loop
```

Description:

Label this line of the input script with the chosen ID. Unless a jump command was used previously, this does nothing. But if a jump command was used with a label argument to begin invoking this script file, then all command lines in the script prior to this line will be ignored. I.e. execution of the script will begin at this line. This is useful for looping over a section of the input script as discussed in the jump commmand.

Restrictions: none

Related commands: none

Default: none

lattice command

Syntax:

```
lattice style value
```

- style = *none* or *sc* or *bcc* or *fcc* or *sq* or *sq2* or *hex*

```
none value = none
  for all other styles:
    value = reduced density (for LJ units)
    value = cubic lattice constant in Angstroms (for real or metal units)
```

Examples:

```
lattice fcc 3.52
lattice hex 0.85
lattice none
```

Description:

Define a lattice type and lattice constant. This is required before using a commands that (optionally) use the lattice, such as create_atoms or region. The lattice type must be consistent with the dimension of the simulation − see the dimension command. Styles *sc* or *bcc* or *fcc* are for 3d problems. Styles *sq* or *sq2* or *hex* are for 2d problems. Lattices of style *fcc*, *bcc*, *sc*, or *hex* are described in any solid−state physics text. A *sq* lattice is one with atoms at the corners of a square. A *sq2* lattice is a *sq* lattice with an additional atom at the center of the square.

For unit style *real* or *metal*, the specified value is the cubic lattice constant in Angstroms. For unit style *lj*, the value is the reduced density (rho*) which LAMMPS converts into a cubic lattice constant. For 3d problems, the relationship "rho* = rho sigma^3" is used for the conversion, where rho = N/V with V = the volume of the cubic cell and N = 4 for *fcc*, 2 for *bcc*, and 1 for *sc* (simple cubic) lattices. For 2d problems, the relationship "rho* = rho sigma^2" is used for the conversion, where N = 2 for *sq2* or *hex* and 1 for *sq*. In the hex case, the unit cell is actually rectangular; it is extended by a factor of sqrt(3) in the y−dimension.

The command "lattice none" can be used to turn off the lattice setting. Any command that attempts to use a lattice constant will then generate an error.

Restrictions: none

Related commands:

dimension, orient, origin, create_atoms, region

Default:

```
lattice none
```

log command

Syntax:

```
log file
```

- file = name of new logfile

Examples:

```
log log.equil
```

Description:

This command closes the current LAMMPS log file, opens a new file with the specified name, and begins logging information to it. If the specified file name is *none*, then no new log file is opened.

If multiple processor partitions are being used, the file name should be a variable, so that different processors do not attempt to write to the same log file.

The file "log.lammps" is the default log file for a LAMMPS run. The name of the initial log file can also be set by the command−line switch −log. See this section for details.

Restrictions: none

Related commands: none

Default:

The default LAMMPS log file is named log.lammps

LAMMPS WWW Site − LAMMPS Documentation − LAMMPS Commands

mass command

Syntax:

```
mass I value
```

- I = atom type (see asterik form below)
- value = mass

Examples:

```
mass 1 1.0
mass * 62.5
mass 2* 62.5
```

Description:

Set the mass for all atoms of one or more atom types. Mass values can also be set in the read_data data file. See the units command for what mass units to use.

I can be specified in one of two ways. An explicit numeric value can be used, as in the 1st example above. Or a wild−card asterik can be used to set the mass for multiple atom types. This takes the form "*" or "*n" or "n*" or "m*n". If N = the number of atom types, then an asterik with no numeric values means all types from 1 to N. A leading asterik means all types from 1 to n (inclusive). A trailing asterik means all types from n to N (inclusive). A middle asterik means all types from m to n (inclusive).

A line in a data file that specifies mass uses the exact same format as the arguments of the mass command in an input script, except that no wild−card asterik can be used. For example, under the "Masses" section of a data file, the line that corresponds to the 1st example above would be listed as

```
1 1.0
```

Restrictions:

This command must come after the simulation box is defined by a read_data, read_restart, or create_box command.

All masses must be defined before a simulation is run (if the atom style requires masses be set). They must also all be defined before a velocity or fix shake command is used.

Masses are not set for atom style granular. This is because each atom is assigned an individual mass in the data or restart file.

Related commands: none

Default: none

LAMMPS WWW Site − LAMMPS Documentation − LAMMPS Commands

---

min_modify command

Syntax:

```
min_modify keyword values ...
```

- one or more keyword/value pairs may be listed

```
keyword = linestyle or dmin or dmax or lineiter
  linestyle value = secant or scan
  dmin value = min
    min = minimum distance for line search to move (distance units)
  dmax value = max
    max = maximum distance for line search to move (distance units)
  lineiter value = N
    N = max number of iterations in a line search
```

Examples:

```
min_modify linestle scan dmin 0.001 dmax 0.2
min_modify lineiter 5
```

Description:

This command sets parameters that affect the minimization algorithms. The various settings may effect the convergence rate and overall number of force evaulations required by a minimization, so users can experiment with these parameters to tune their minimizations.

The *linestyle* sets the algorithm used for 1d line searches at each outer iteration of the minimizer. The *secant* style uses two successive force/energy evaluations to create a parabola and pick its minimum as an estimate of the next iteration's 1d minimum. The *scan* style starts its 1d search at *dmin* and doubles the distance along the line at which the energy is computed until the minimum is passed. It continues only as far as *dmax*. Normally, the *secant* method should find more accurate 1d minimums in less iterations, but the *scan* method can be more robust.

The *dmin* and *dmax* settings are both used by the *scan* line search as described above. For the *secant* line search, only the *dmin* value is used to pick an initial point to begin the secant approximation.

The *lineiter* setting is used by the *secant* algorithm to limit its iterations. The smaller the setting, the more inaccurate the line search becomes. Nonlinear conjugate gradient is not thought to require high−accuracy line searches in order to converge efficiently.

Restrictions: none

Related commands:

min_style, minimize

Default:

The option defaults are linestyle = secant, dmin = 1.0e−5, dmax = 0.1, and lineiter = 10.

LAMMPS WWW Page − LAMMPS Documentation − LAMMPS Commands

---

min_style command

Syntax:

```
min_style style
```

- style = *cg* or *cg/fr* or *sd*

Examples:

```
min_style cg
min_style sd
```

Description:

Choose a minimization algorithm to use when a minimize command is performed.

Style *cg* is the Polak−Ribiere (PR) version of the conjugate gradient (CG) algorithm. At each iteration the force gradient is combined with the previous iteration information to compute a new search direction perpendicular (conjugate) to previous search directions. The PR variant affects how the direction is chosen and how the CG method is restarted when it ceases to make progress. The PR variant is thought to be the most effective CG choice.

Style *cg/fr* is the Fletcher−Reeves version of the conjugate gradient algorithm.

Style *sd* is a steepest descent algorithm. At each iteration, the downhill direction corresponding to the force vector (negative gradient of energy) is searched along by a 1d line search. Typically, steepest descent will not converge as quickly as CG, but may be more robust in some situations.

Restrictions: none

Related commands:

min_modify, minimize

Default:

```
min_style cg
```

minimize command

Syntax:

```
minimize tolerance maxiter maxeval
```

- tolerance = stopping tolerance
- maxiter = max iterations of minimizer
- maxeval = max number of total force/energy evaluations

Examples:

```
minimze 1.0e−4 100 1000
```

Description:

Perform an energy minimization of the system, by adjusting each atom's atomic coordinates. The algorithm used is set by the min_style command. Minimize commands can be interspersed with run commands to alternate between relaxation and dynamics. The minimizers are implemented in a robust fashion that should allow for systems with highly overlapped atoms (large energies and forces) to still be minimized by pushing the atoms off of each other.

A minimization involves an outer iteration loop which sets the search direction along which coordinates are changed. An inner iteration is then performed using a line search algorithm. The line search typically evaluates forces and energies several times to set new coordinates. The minimization stops if any of several criteria are met:

- the change in energy between outer iterations is less than the tolerance
- the number of outer iterations exceeds maxiter

- the number of force evaluations exceeds maxeval
- the 3N dimensional force vector goes (nearly) to zero

For the first criterion, the specified tolerance is unitless; it is met when the ratio of the energy delta to the energy magnitude is equal to the tolerance (e.g. one part in 1.0e−4).

During a minimization, the outer iteration count is treated as a timestep. Output is triggered by this timestep, e.g. thermodynamic output or dump and restart files.

For optimal convergence, a pair style that goes smoothly to 0.0 at the cutoff distance for both energy and force should typically be used though this is not required. Examples include *pair/lj/charmm/coul/charmm* and *pair/lj/charmm/coul/long*. If a *soft* potential is used the Astop value is used for the prefactor (no time dependence).

Only fixes that apply force constraints are invoked during minimization. The list of the currently implemented ones include fix *addforce*, *aveforce*, *enforce2d*, *indent*, *lineforce*, *planeforce*, *setforce*, and *wall/lj93*.

Following the minimization a statistical summary is printed that includes the energy change and convergence criteria information.

Restrictions:

Two features that are not yet implemented listed here, in case someone knows how they could be coded:

Two fixes not invoked by a minimization are fix shake and fix rigid. The effect of a fix shake can be approximated during a minimization by using stiff spring constants for the bonds and/or angles that would normally be constrained by the SHAKE algorithm.

The volume of the simulation domain is not allowed to change during a minimzation. Ideally we would allow a fix such as *npt* to impose an external pressure that would be included in the minimization (i.e. allow the box dimensions to change).

Related commands:

min_modify, min_style, run_style

Default: none

LAMMPS WWW Site − LAMMPS Documentation − LAMMPS Commands

---

neigh_modify command

Syntax:

```
neigh_modify keyword values ...
```

- one or more keyword/value pairs may be listed

```
keyword = delay or every or check or exclude or page or one
  delay value = N
    N = delay building until this many steps since last build
  every value = M
    M = build neighbor list every this many steps
  check value = yes or no
    yes = only build if some atom has moved half the skin distance or more
    no = always build on 1st step that every and delay are satisfied
  exclude values:
    type M N
      M,N = exclude if one atom in pair is type M, other is type N
    group group1−ID group2−ID
      group1−ID,group2−ID = exclude if one atom is in 1st group, other in 2nd
    molecule group−ID
      groupname = exclude if both atoms are in the same molecule and in the same group
    none
      delete all exclude settings
  page value = N
    N = number of pairs stored in a single neighbor page
  one value = N
    N = max number of neighbors of one atom
```

Examples:

```
neigh_modify every 2 delay 10 check yes page 100000
neigh_modify exclude type 2 3
neigh_modify exclude group frozen frozen check no
neigh_modify exclude group residue1 chain3
neigh_modify exclude molecule rigid
```

Description:

This command sets parameters that affect the pairwise neighbor list.

The *every*, *delay*, and *check* options affect how often the list is built as a simulation runs. The *delay* setting means never build a new list until at least N steps after the previous build. The *every* setting means build the list every M steps (after the delay has passed). If the *check* setting is *no*, the list is built on the 1st step that satisfies the *delay* and *every* settings. If the *check* setting is *yes*, then the list is only built on a particular step if some atom has moved more than half the skin distance (specified in the neighbor command) since the last build.

When the rRESPA integrator is used (see the run_style command), the *every* and *delay* parameters refer to the longest (outermost) timestep.

The *exclude* option turns off pairwise interactions between certain pairs of atoms, by not including them in the neighbor list. These are sample scenarios where this is useful:

- In crack simulations, pairwise interactions can be shut off between 2 slabs of atoms to effectively create a crack.
- When a large collection of atoms is treated as frozen, interactions between those atoms can be turned off to save needless computation. E.g. Using the fix setforce command to freeze a wall or portion of a bio−molecule.
- When one or more rigid bodies are specified, interactions within each body can be turned off to save needless computation. See the fix rigid command for more details.

The *exclude type* option turns off the pairwise interaction if one atom is of type M and the other of type N. M can equal N. The *exclude group* option turns off the interaction if one atom is in the first group and the other is the second. Group1−ID can equal group2−ID. The *exclude molecule* option turns off the interaction if both atoms are in the specified group and in the same molecule, as determined by their molecule ID.

Each of the exclude options can be specified multiple times. The *exclude type* option is the most efficient option to use; it requries only a single check, no matter how many times it has been specified. The other exclude options are more expensive if specified multiple times; they require one check for each time they have been specified.

Note that the exclude options only affect pairwise interactions; see the delete_bonds command for information on turning off bond interactions.

The *page* and *one* options affect how memory is allocated for the neighbor lists. For most simulations the default settings for these options are fine, but if a very large problem is being run or a very long cutoff is being used, these parameters can be tuned. The indices of neighboring atoms are stored in "pages", which are allocated one after another as they fill up. The size of each page is set by the *page* value. A new page is allocated when the next atom's neighbors could potentially overflow the list. This threshhold is set by the *one* value which tells LAMMPS the maximum number of neighbor's one atom can have.

Restrictions:

If the "delay" setting is non−zero, then it must be a multiple of the "every" setting.

The exclude molecule option can only be used with atom styles that define molecule IDs.

Related commands:

neighbor, delete_bonds

Default:

The option defaults are delay = 10, every = 1, check = yes, exclude = none, page = 10000, and one = 2000.

neighbor command

Syntax:

```
neighbor skin style
```

- skin = extra distance beyond force cutoff (distance units)
- style = *bin* or *nsq*

Examples:

```
neighbor 0.3 bin
neighbor 2.0 nsq
```

Description:

This command sets parameters that affect the building of the pairwise neighbor list. All atom pairs within a cutoff distance equal to the their force cutoff plus the *skin* distance are stored in the list. Typically, the larger the skin distance, the less often neighbor lists need to be built, but more pairs must be checked for possible force interactions every timestep.

The *style* value chooses what algorithm is used to build the list. *Binning* is an operation that scales linearly with N, the number of atoms on a processor. It is almost always faster than the *nsq* style which scales as N^2. For unsolvated small molecules in a non−periodic box, the *nsq* choice can sometimes be faster. Either style should give the same answers.

The default values for *skin* and *style* depend on the choice of units for the simulation.

The neigh_modify command has additional options that control how often neighbor lists are built and which pairs are stored in the list.

When a run is finished, counts of the number of neighbors stored in the pairwise list and the number of times neighbor lists were built are printed to the screen and log file. See this section for details.

Restrictions: none

Related commands:

neigh_modify, units

Default:

```
0.3 bin      for lj units
2.0 bin      for real or metal units
```

newton command

Syntax:

```
newton flag
newton flag1 flag2
```

- flag = *on* or *off* for both pairwise and bonded interactions
- flag1 = *on* or *off* for pairwise interactions
- flag2 = *on* or *off* for bonded interactions

Examples:

```
newton off
newton on off
```

Description:

This command turns Newton's 3rd law *on* or *off* for pairwise and bonded interactions. For most problems, setting Newton's 3rd law to *on* means a modest savings in computation at the cost of two times more communication. Whether this is faster depends on problem size, force cutoff lengths, a machine's compute/communication ratio, and how many processors are being used.

Setting the pairwise newton flag to *off* means that if two interacting atoms are on different processors, both processors compute their interaction and the resulting force information is not communicated. Similarly, for bonded interactions, newton *off* means that if a bond, angle, dihedral, or improper interaction contains atoms on 2 or more processors, the interaction is computed by each processor.

LAMMPS should produce the same answers for any newton flag settings, except for round−off issues.

With run_style *respa* and only bonded interactions (bond, angle, etc) computed in the innermost timestep, it may be faster to turn newton *off* for bonded interactions, to avoid extra communication in the innermost loop.

Restrictions:

The newton bond setting cannot be changed after the simulation box is defined by a read_data or create_box command.

Related commands:

run_style respa

Default:

```
newton on
```

LAMMPS WWW Site − LAMMPS Documentation − LAMMPS Commands

---

next command

Syntax:

```
next variables
```

- variables = one or more lower−case single−character variable names

Examples:

```
next x
next a t x
```

Description:

This command is used with variables defined by the variable command. It assigns the next value to the variable from the variable's list, so that when $X is subsequently substituted for in an input script command, the new value is used. X is a single lower−case character from "a" to "z".

All variables in a single next command must be the same style: *index*, *loop*, or *universe*. *Equal*− or *world*−style variables cannot be incremented by a next command.

When any of the variables in the next command has no more values, a flag is set that causes the input script to skip the next jump command encountered. This enables a loop containing a next command to exit.

When the next command is used with *index*− or *loop*−style variables, the next value is assigned to the variable for all processors. When the next command is used with *universe*−style variables, the next value is assigned to whichever processor partition executes the command first. All processors in the partition are assigned the same value. Running LAMMPS on multiple partitions of processors via the "−partition" command−line switch is described in this section of the manual. *Universe*−style variables are incremented using the files "tmp.lammps.variable" and "tmp.lammps.variable.lock" which you will see in your directory during such a LAMMPS run.

Here is an example of running a series of simulations using the next command with an *index*−style variable. If this input script is named in.polymer, 8 simulations would be run using data files from directories run1 thru run8.

```
variable d index run1 run2 run3 run4 run5 run6 run7 run8
cd $d
read_data data.polymer
run 10000
cd ..
clear
next d
jump in.polymer
```

If the variable "d" were of style *universe*, and the same in.polymer input script were run on 3 partitions of processors, then the first 3 simulations would begin, one on each set of processors. Whichever partition finished first, it would assign variable "d" the 4th value and run another simulation, and so forth until all 8 simulations were finished.

Jump and next commands can also be nested to enable multi−level loops. For example, this script will run 15 simulations in a double loop.

variable i loop 3 variable j loop 5 clear ... read_data data.polymer.$i$j print Running simulation $i.$j run 10000 next j jump in.script next i jump in.script

Restrictions: none

Related commands:

variable, jump, include

Default: none

LAMMPS WWW Site − LAMMPS Documentation − LAMMPS Commands

orient command

Syntax:

```
orient dim i j k
```

- dim = *x* or *y* or *z*
- i,j,k = orientation of lattice that is along box direction dim

Examples:

```
orient x 1 1 0
orient y −1 1 0
orient z 0 0 1
```

Description:

Specify the orientation of a cubic lattice along simulation box directions *x* or *y* or *z*. These 3 basis vectors are used when the create_atoms command generates a lattice of atoms.

The 3 basis vectors B1, B2, B3 must be mutually orthogonal and form a right−handed system such that B1 cross B2 is in the direction of B3.

The basis vectors should be specified in an irreducible form (smallest possible integers), though LAMMPS does not check for this.

Restrictions: none

Related commands:

origin, create_atoms

Default:

```
orient x 1 0 0
orient y 0 1 0
orient z 0 0 1
```

LAMMPS WWW Site − LAMMPS Documentation − LAMMPS Commands

---

origin command

Syntax:

```
origin x y z
```

- x,y,z = origin of a lattice

Examples:

```
origin 0.0 0.5 0.5
```

Description:

Set the origin of the lattice defined by the lattice command. The lattice is used by the create_atoms command to create new atoms and by other commands that use a lattice spacing as a distance measure. This command offsets the origin of the lattice from the (0,0,0) coordinate of the simulation box by some fraction of a lattice spacing in each dimension.

The specified values are in lattice coordinates from 0.0 to 1.0, so that a value of 0.5 means the lattice is displaced 1/2 a cubic cell.

Restrictions: none

Related commands:

lattice, orient

Default:

```
origin 0 0 0
```

---

pair_coeff command

Syntax:

```
pair_coeff I J args
```

- I,J = atom types (see asterik form below)
- args = coefficients for one or more pairs of atom types

Examples:

```
pair_coeff 2 2 1.0 1.0 2.5
pair_coeff 2 * 1.0 1.0
pair_coeff 3* 1*2 1.0 1.0 2.5
pair_coeff * * 1.0 1.0
pair_coeff 2 2 niu3
pair_coeff * * nialhjea 1 1 2
pair_coeff * 3 morse.table ENTRY1
pair_coeff 1 2 lj/cut 1.0 1.0 2.5
```

Description:

Specify the pairwise force field coefficients for one or more pairs of atom types. The number and meaning of the coefficients depends on the pair style. Pair coefficients can also be set in the data file read read_data command or in a restart file.

I and J can be specified in one of two ways. Explicit numeric values can be used for each, as in the 1st example above. In this case, I <= J is required. LAMMPS sets the coefficients for the symmetric J,I interaction to the same values.

A wild−card asterik can be used with the I,J arguments to set the coefficients for multiple pairs of atom types. This takes the form "*" or "*n" or "n*" or "m*n". If N = the number of atom types, then an asterik with no numeric values means all types from 1 to N. A leading asterik means all types from 1 to n (inclusive). A trailing asterik means all types from n to N (inclusive). A middle asterik means all types from m to n (inclusive). Note that only type pairs with I <= J are set; if asteriks imply type pairs where J < I, they are ignored.

Note that using 2 pair_coeff commands for the same I,J pair is perfectly valid. For example, these commands set the coeffs for all I,J pairs, then overwrite the coeffs for just the I,J = 2,3 pair:

```
pair_coeff * * 1.0 1.0 2.5
pair_coeff 2 3 2.0 1.0 1.12
```

A line in a data file that specifies pair coefficients uses the exact same format as the arguments of the pair_coeff command in an input script, with the exception of the I,J type arguments. In each line of the "Pair Coeffs" section of a data file, only a single type I is specified, which sets the coefficients for type I interacting with type I. This is because the section has exactly N lines, where N = the number of atom types. For this reason, the wild−card asterik should also not be used as part of the I argument. Thus in a data file, the line corresponding to the 1st example above would be listed as

```
2 1.0 1.0 2.5
```

If coefficients for type pairs with I not equal J are not set explicity by a pair_coeff command, they are inferred from the I,I and J,J settings by mixing rules; see the pair_modify command for a discussion. Exceptions to the mixing rules of the pair_modify command are discussed below.

Here are the coefficients specified by the pair_coeff command for each pair style. Note that when allowed, cutoff arguments are optional; if they are specified for a particular I,J pair they override the global cutoff(s) specified by the pair_style command.

The units of each coefficient are shown in parenthesis.

$$E = Ae^{-r/\rho} - \frac{C}{r^6} \qquad r < r_c$$

For styles *buck*, *buck/coul/cut*, and *buck/coul/long*, specify 3, 4, or 5 coefficients:

- A (energy units)
- rho (distance units)
- C (energy−distance^6 units)
- cutoff (distance units)
- cutoff2 (distance units)

The latter 2 coefficients are optional. If not specified, the global LJ and Coulombic cutoffs are used. If only one cutoff is specified, it is used as the cutoff for both LJ and Coulombic interactions for this type pair. If both coefficients are specified, they are used as the LJ and Coulombic cutoffs for this type pair.

For *buck/coul/long* only the LJ cutoff can be specified since a Coulombic cutoff cannot be specified for an individual I,J type pair. All type pairs use the same global Coulombic cutoff specified in the pair_style command.

The *dipole* styles are not yet implemented in LAMMPS. They will enable a point dipole and charge to be assigned to each atom and the resulting charge−dipole and dipole−dipole interactions to be computed.

$$\begin{aligned}
\vec{f} &= (F^C + F^D + F^R)\hat{r_{ij}} \qquad\qquad r < r_c \\
F^C &= Aw(r) \\
F^D &= -\gamma w^2(r)(\hat{r_{ij}} \bullet \vec{v_{ij}}) \\
F^R &= \sigma w(r)\alpha(\Delta t)^{-1/2} \\
w(r) &= 1 - r/r_c
\end{aligned}$$

For style *dpd*, specify 3 or 4 coefficients:

- A (force units)
- gamma (force/velocity units)
- sigma (force*sqrt(time) units)
- cutoff (distance units)

The last coefficient is optional. If not specified, the global DPD cutoff is used.

$$E_i = F_\alpha \left(\sum_{j \neq i} \rho_\alpha(r_{ij}) \right) + \sum_{j \neq i} \phi_{\alpha\beta}(r_{ij})$$

For style *eam*, potential values are read from a file that is in the DYNAMO single−element *funcfl* format.

Note that unlike for other potentials, you do not set cutoffs for EAM potentials in the pair_style or pair_coeff command; they are specified in the EAM potential files.

For style *eam* you must assign a potential file to each I,I pair of atom types by using a single pair_coeff argument:

- filename

Thus the following command

```
pair_coeff *2 1*2 cuu3
```

will read the cuu3 potential file and use the tabulated Cu values for F, phi, rho that it contains for type pairs 1,1 and 2,2 (type pairs 1,2 and 2,1 are ignored). In effect, this makes atom types 1 and 2 in LAMMPS be Cu atoms. Different single−element files can be assigned to different atom types to model an alloy system. The mixing to create alloy potentials for type pairs with I != J is done automatically the same way that the serial DYANMO code originally did it; you do not need to specify coefficients for these type pairs.

There are several *funcl* files in the *potentials* directory of the LAMMPS distribution. A DYNAMO single−element *funcfl* file is formatted as follows:

- line 1: comment (ignored)
- line 2: atomic number, mass, lattice constant, lattice type (e.g. FCC)
- line 3: Nrho, drho, Nr, dr, cutoff

On line 2, all values but the mass are ignored by LAMMPS. The mass is in atomic mass units which is converted by LAMMPS to the appropriate internal mass units. On line 3, Nrho and Nr are the number of tabulated values in the subsequent arrays, drho and dr are the spacing in density and distance space for the values in those arrays, and the specified cutoff becomes the pairwise cutoff used by LAMMPS for the potential. The units of dr are Angstroms; I'm not sure of the units for drho − some measure of electron density.

Following the 3 header lines are 3 arrays of tabulated values:

- embedding function F (Nrho values)
- pair potential function phi (Nr values)

* density function rho (Nr values)

The values for each array can be listed as multiple values per line, so long as each array starts on a new line. The individual values are (for example) phi(r) for r = 0,dr,2*dr, ... (Nr−1)*dr.

$$E_i = F_\alpha \left(\sum_{j \neq i} \rho_\alpha (r_{ij}) \right) + \sum_{j \neq i} \phi_{\alpha\beta} (r_{ij})$$

For style *eam/alloy*, potential values are read from a file that is in the DYNAMO multi−element *setfl* format.

Only one pair_coeff command can be used (one file). DYNAMO *setfl* files contain information for M elements. These are mapped to LAMMPS atom types by specifying N additional arguments after the filename, where N is the number of LAMMPS atom types:

* filename
* N values from 0 to M = mapping of *setfl* elements to atom types

As an example, the nialhjea *setfl* file has tabulated EAM values for 3 elements and their alloy interactions: Ni, Al, and H. If your LAMMPS simulation has 4 atoms types and you want the 1st 3 to be Ni, and the 4th to be Al, you would use the following pair_coeff command:

```
pair_coeff * * nialhjea 1 1 1 2
```

The 1st 2 arguments must be * * so as to span all LAMMPS atom types. The first three "1" values map LAMMPS atom types 1,2,3 to the 1st element (Ni) in the *setfl* file. The final "2" value maps LAMMPS atom type 4 to the 2nd element = Al. If a mapping value is "0", the mapping is not performed. This is useful when EAM potentials are part of the *hybrid* pair style, to represent non−EAM atom types.

There is one *setfl* file (nialhjea) in the *potentials* directory of the LAMMPS distribution. A DYNAMO multi−element *setfl* file is formatted as follows:

* lines 1,2,3 = comments (ignored)
* line 4: Nelements = # of elements in the file
* line 5: Nrho, drho, Nr, dr, cutoff

The meaning of the values in line 5 is the same as for the *funcfl* file described above. Note that the cutoff is a global value, valid for all pairwise interactions for all element pairings.

Following the 5 header lines are Nelements sections, one for each element, each with the following format:

* line 1 = atomic number, mass, lattice constant, lattice type (e.g. FCC)
* embedding function F (Nrho values)
* density function rho (Nr values)

As with the *funcfl* files, only the mass is used by LAMMPS from the 1st line. The F and rho arrays are unique to a single element and are formatted the same as in a *funcfl* file.

Following the Nelements sections, values for the pair potential phi arrays are listed for all i,j element pairs in the same format as other arrays. Since these interactions are symmetric (i,j = j,i) only phi arrays with i >= j are listed, in the following order: i,j = (1,1), (2,1), (2,2), (3,1), (3,2), (3,3), (4,1), ..., (Nelements, Nelements). The tabulated values for each phi function are listed in *setfl* files as r*phi, rather than as phi (in *funcfl* files).

$$E_i = F_\alpha \left(\sum_{j \neq i} \rho_{\alpha\beta}(r_{ij}) \right) + \sum_{j \neq i} \phi_{\alpha\beta}(r_{ij})$$

For style *eam/fs*, the form of the pair_coeff command is exactly the same as for style *eam/alloy*, e.g.

```
pair_coeff * * filename 1 1 1 2
```

where there are N additional arguments after the filename, where N is the number of LAMMPS atom types. The N values determine the mapping of LAMMPS atom types to EAM elements in the file, as described in style *eam/alloy*.

The difference is that files read by *eam/fs* are in a more general format than the DYNAMO *setfl* format read by *eam/alloy*, so that the i,j density functionals for all pairs of elements are included as needed by the Finnis/Sinclair formulation of the EAM.

There is one FS file (nialhjea_FS) in the *potentials* directory of the LAMMPS distribution. It is formatted as follows:

- lines 1,2,3 = comments (ignored)
- line 4: Nelements = # of elements in the file
- line 5: Nrho, drho, Nr, dr, cutoff

The 5−line header section is identical to an EAM *setfl* file.

Following the header are Nelements sections, one for each element I, each with the following format:

- line 1 = atomic number, mass, lattice constant, lattice type (e.g. FCC)
- embedding function F (Nrho values)
- density function rho for element I at element 1 (Nr values)
- density function rho for element I at element 2
- ...
- density function rho for element I at element Nelement

Following the Nelements sections, values for the pair potential phi arrays are listed in the same manner (r*phi) as in EAM *setfl* files. Note that the rho arrays in Finnis/Sinclair can be asymmetric (i,j != j,i) so there are Nelements^2 of them listed in the file. But the phi arrays are still symmetric, so only phi arrays for i >= j are listed.

$$F = f\left(\delta/d\right)\left(k_n \delta \mathbf{n}_{ij} - m_{\text{eff}}\gamma_n \mathbf{v}_n\right) + f\left(\delta/d\right)\left(-k_t \delta \mathbf{\Delta s}_t - m_{\text{eff}}\gamma_t \mathbf{v}_t\right)$$

For styles *gran/hertzian*, *gran/history*, and *gran/no_history*, there are no individual atom type coefficients that can be set. All global settings are made via the pair_style command.

$$
\begin{aligned}
E &= LJ(r) && r < r_{\rm in} \\
&= S(r) * LJ(r) && r_{\rm in} < r < r_{\rm out} \\
&= 0 && r > r_{\rm out} \\
E &= C(r) && r < r_{\rm in} \\
&= S(r) * C(r) && r_{\rm in} < r < r_{\rm out} \\
&= 0 && r > r_{\rm out} \\
LJ(r) &= 4\epsilon \left[\left(\frac{\sigma}{r}\right)^{12} - \left(\frac{\sigma}{r}\right)^{6} \right] \\
C(r) &= \frac{C q_i q_j}{\epsilon r} \\
S(r) &= \frac{\left[r_{\rm out}^2 - r^2\right]^2 \left[r_{\rm out}^2 + 2r^2 - 3r_{\rm in}^2\right]}{\left[r_{\rm out}^2 - {r_{\rm in}}^2\right]^3}
\end{aligned}
$$

For styles *lj/charmm/coul/charmm*, *lj/charmm/coul/charmm/implicit*, and *lj/charmm/coul/long*, specify 2 or 4 coefficients:

- epsilon (energy units)
- sigma (distance units)
- epsilon_14 (energy units)
- sigma_14 (distance units)

Note that sigma is defined as in the LJ formula above as the zero−crossing distance for the potential, not as the energy minimum at 2^(1/6) sigma.

The latter 2 coefficients are optional. If they are specified, they are used in the Lennard−Jones formula between 2 atoms of these types which are also first and fourth atoms in any dihedral. No cutoffs are specified because this CHARMM force field does not allow varying cutoffs for individual atom pairs; all pairs use the global cutoff(s) specified in the pair_style command.

$$E = \epsilon \left[2 \left(\frac{\sigma}{r}\right)^{9} - 3 \left(\frac{\sigma}{r}\right)^{6} \right] \qquad r < r_c$$

For styles *lj/class2*, *lj/class2/coul/cut*, and *lj/class2/coul/long*, specify 2, 3, or 4 coefficients:

- epsilon (energy units)
- sigma (distance units)
- cutoff1 (distance units)
- cutoff2 (distance units)

The latter 2 coefficients are optional. If not specified, the global class 2 and Coulombic cutoffs are used. If only one cutoff is specified, it is used as the cutoff for both class 2 and Coulombic interactions for this type pair. If both coefficients are specified, they are used as the class 2 and Coulombic cutoffs for this type pair.

For *lj/class2/coul/long* only the class 2 cutoff can be specified since a Coulombic cutoff cannot be specified for an individual I,J type pair. All type pairs use the same global Coulombic cutoff specified in the pair_style command.

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

$$E = \frac{C q_i q_j}{\epsilon r} \qquad r < r_c$$

$$E = \frac{C q_i q_j}{\epsilon r} \exp(-\kappa r) \qquad r < r_c$$

For styles *lj/cut*, *lj/cut/coul/cut*, *lj/cut/coul/debye*, *lj/cut/coul/long*, and *lj/cut/coul/long/tip4p* specify 2, 3, or 4 coefficients:

- epsilon (energy units)
- sigma (distance units)
- cutoff1 (distance units)
- cutoff2 (distance units)

Note that sigma is defined as in the LJ formula above as the zero−crossing distance for the potential, not as the energy minimum at 2^(1/6) sigma.

The latter 2 coefficients are optional. If not specified, the global LJ and Coulombic cutoffs are used. If only one cutoff is specified, it is used as the cutoff for both LJ and Coulombic interactions for this type pair. If both coefficients are specified, they are used as the LJ and Coulombic cutoffs for this type pair.

For *lj/cut/coul/long* and *lj/cut/coul/long/tip4p* only the LJ cutoff can be specified since a Coulombic cutoff cannot be specified for an individual I,J type pair. All type pairs use the same global Coulombic cutoff specified in the pair_style command.

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

For style *lj/expand*, specify 3 or 4 coefficients:

- epsilon (energy units)
- sigma (distance units)
- delta (distance units)
- cutoff (distance units)

The delta values can be positive or negative. Note that the cutoff does not include the delta distance. I.e. the actual force cutoff is the sum of cutoff + delta.

The last coefficient is optional. If not specified, the global LJ cutoff is used.

$$E = D_0 \left[e^{-2\alpha(r-r_0)} - 2e^{-\alpha(r-r_0)} \right] \qquad r < r_c$$

For style *morse*, specify 3 or 4 coefficients:

- D0 (energy units)
- alpha (1/distance units)
- r0 (distance units)
- cutoff (distance units)

The last coefficient is optional. If not specified, the global morse cutoff is used.

$$E = A \left[1 + \cos\left(\frac{\pi r}{r_c}\right) \right] \qquad r < r_c$$

For style *soft*, specify 2 or 3 coefficients:

- Astart (energy units)
- Astop (energy units)
- cutoff (distance units)

Astart and Astop are the values of the prefactor at the start and end of the next run. At intermediate times the value of A will be ramped between these 2 values. Note that before performing a 2nd run, you will want to adjust the values of Astart and Astop for all type pairs, or switch to a new pair style.

The last coefficient is optional. If not specified, the global soft cutoff is used.

For style *table*, specify 2 or 3 coefficients:

- filename
- keyword
- cutoff (distance units)

The filename specifies a file containing tabulated energy and force values. The keyword specifies a section of the file. The cutoff is an optional coefficient. If not specified, the outer cutoff in the table itself (see below) will be used to build an interpolation table that extend to the largest tablulated distance. If specified, only file values up to the cutoff are used to create the interpolation table.

The format of a tabulated file is as follows (without the parenthesized comments):

```
# Morse potential for Fe   (one or more comment or blank lines)

MORSE_FE                   (keyword is first text on line)
N 500 R 1.0 10.0           (N, R, RSQ, BITMAP, FPRIME parameters)
                           (blank)
1 1.0 25.5 102.34          (index, r, energy, force)
2 1.02 23.4 98.5
...
500 10.0 0.001 0.003
```

A section begins with a non−blank line whose 1st character is not a "#"; blank lines or lines starting with "#" can be used as comments between sections. The first line begins with a keyword which identifies the section. The line can contain additional text, but the initial text must match the argument specified in the pair_coeff command. The next line lists (in any order) one or more parameters for the table. Each parameter is a keyword followed by one or more numeric values.

The parameter "N" is required; its value is the number of table entries that follow. All other parameters are optional. If "R" or "RSQ" or "BITMAP" does not appear, then the distances in each line of the table are used as−is to perform spline interpolation. In this case, the table values can be spaced in *r* uniformly or however you wish to position table values in regions of large gradients.

If used, the parameters "R" or "RSQ" are followed by 2 values *rlo* and *rhi*. If specified, the distance associated with each energy and force value is computed from these 2 values (at high accuracy), rather than using the (low−accuracy) value listed in each line of the table. For "R", distances uniformly spaced between *rlo* and *rhi* are computed; for "RSQ", squared distances uniformly spaced between *rlo*rlo* and *rhi*rhi* are computed.

If used, the parameter "BITMAP" is also followed by 2 values *rlo* and *rhi*. These values, along with the "N" value determine the ordering of the N lines that follow and what distance is associated with each. This ordering is complex, so it is not documented here, since this file is typically produced by the pair_write command with its *bitmap* option. When the table is in BITMAP format, the "N" parameter in the file must be equal to 2^M where M is the value specified in the pair_style command. Also, a cutoff parameter cannot be used as an optional 3rd argument in the pair_coeff command; the entire table extent as specified in the file must be used.

If used, the parameter "FPRIME" is followed by 2 values *fplo* and *fphi* which are the derivative of the force at the innermost and outermost distances listed in the table. These values are needed by the spline construction routines. If not specified by the "FPRIME" parameter, they are estimated (less accurately) by the first 2 and last 2 force values in the table. This parameter is not used by BITMAP tables.

Following a blank line, the next N lines list the tabulated values. On each line, the 1st value is the index from 1 to N, the 2nd value is r (in distance units), the 3rd value is the energy (in energy units), and the 4th is the force (in force units). The r values must increase from one line to the next (unless the BITMAP parameter is specified).

Note that one file can contain many sections, each with a tabulated potential. LAMMPS reads the file section by section until it finds one that matches the specified keyword.

$$E = A\frac{e^{-\kappa r}}{r} \qquad r < r_c$$

For style *yukawa*, specify 1 or 2 coefficients:

- A (energy units)
- cutoff (distance units)

The last coefficient is optional. If not specified, the global yukawa cutoff is used.

For style *hybrid*, each pair_coeff command assigns one of the sub−styles specified in the pair_style command to a set of atom type pairs. The arguments of the pair_coeff command are the same as they would be for the sub−style itself, except that an additional argument is added (after the type pairs) which specifies which sub−style is being used. For example, consider a simulation with 3 atom types: types 1 and 2 are Ni atoms, type 3 are LJ atoms with charges. The following commands would set up the hybrid simulation:

```
atom_style hybrid eam charge
pair_style hybrid eam lj/cut/coul/cut 10.0 lj/cut 8.0
pair_coeff 1*2 1*2 eam niu3
pair_coeff 3 3 lj/cut/coul/cut 1.0 1.0
pair_coeff 1*2 3 lj/cut 0.5 1.2
```

The atom_style hybrid command is needed because atoms in the simulation will have both EAM and charge attributes.

Restrictions:

This command must come after the simulation box is defined by a read_data, read_restart, or create_box command.

Related commands:

pair_style, pair_modify, read_data, read_restart, pair_write

Default: none

LAMMPS WWW Site − LAMMPS Documentation − LAMMPS Commands

---

pair_modify command

Syntax:

```
pair_modify keyword value ...
```

- one or more keyword/value pairs may be listed
- keyword = *shift* or *mix*

```
  shift value = yes or no
    mix value = geometric or arithmetic or sixthpower
    table value = N
      2^N = # of values in table
    tabinner value = cutoff
      cutoff = inner cutoff at which to begin table (distance units)
```

Examples:

```
pair_modify shift yes
pair_modify mix arithmetic
pair_modify table 12
```

Description:

Modify the parameters of the currently defined pair style. Not all parameters are relevant to all pair styles.

The *shift* keyword determines whether the Lennard−Jones potential is shifted at its cutoff to 0.0. If so, this adds an energy term to each pairwise interaction which will be printed in the thermodynamic output, but does not affect atom dynamics (forces). Pair styles that are already 0.0 at their cutoff such as *lj/charmm/coul/charmm* are not affected by this setting.

The *mix* keyword affects how Lennard−Jones coefficients for epsilon and sigma are generated for interactions between atoms of type I and J, when I != J. (I = J coefficients are set explicitly in the data file or input script.) The pair_coeff command can be used in the input script to specify epilon/sigma for a specific I,J pairing, which overrides the setting of the *mix* keyword. In each case, the LJ cutoff is mixed the same way as sigma.

These are the formulas used by the 3 *mix* options:

geometric

```
epsilon_ij = sqrt(epsilon_i * epsilon_j)
sigma_ij = sqrt(sigma_i * sigma_j)
```

arithmetic

```
epsilon_ij = sqrt(epsilon_i * epsilon_j)
sigma_ij = (sigma_i + sigma_j) / 2
```

sixthpower

```
epsilon_ij = (2 * sqrt(epsilon_i*epsilon_j) * sigma_i^3 * sigma_j^3) /
             (sigma_i^6 + sigma_j^6)
sigma_ij=  ((sigma_i**6 + sigma_j**6) / 2) ^ (1/6)
```

Style *soft* only uses a pre−factor coefficient, which is always mixed geometrically, regardless of the *mix* setting. The *charmm* styles are always mixed arithmetically, regardless of the *mix* setting. The *class2* styles are always mixed as a sixthpower, regardless of the *mix* setting, except that the cutoff is mixed according to the mix setting. Style *lj/expand* always mixes its delta coefficient using the rule

```
delta_ij = (delta_i + delta_j) / 2
```

The *table* keyword applies to pair styles with a long−range Coulombic term (lj/cut/coul/long and lj/charmm/coul/long). If N is non−zero, a table of length 2^N is pre−computed for forces and energies, which can shrink their computational cost by up to a factor of 2. The table is indexed via a bit−mapping technique (Wolff) and a linear interpolation is performed between adjacent table values. In our experiments with different table styles (lookup, linear, spline), this method typically gave the best performance in terms of speed and accuracy.

The choice of table length is a tradeoff in accuracy versus speed. A larger N yields more accurate force computations, but requires more memory which can slow down the computation due to cache misses. A reasonable value of N is between 8 and 16. The default value of 12 (table of length 4096) gives approximately the same accuracy as the no−table (N = 0) option. For N = 0, forces and energies are computed directly, using a polynomial fit for the needed erfc() function evaluation, which is what earlier versions of LAMMPS did. Values greater than 16 typically slow down the simulation and will not improve accuracy; values from 1 to 8 give unreliable results.

The *tabinner* keyword sets an inner cutoff above which the pairwise computation is done by table lookup (if tables are invoked). The smaller this value is set, the less accurate the table becomes (for a given number of table values), which can require use of larger tables. The default cutoff value is sqrt(2.0) distance units which means nearly all pairwise interactions are computed via table lookup for simulations with "real" units, but some close pairs may be computed directly (non−table) for simulations with "lj" units.

Restrictions: none

Related commands:

pair_style, pair_coeff

Default:

The option defaults are shift = no, mix = arithmetic (for lj/charmm pair styles), mix = geometric (for other pair styles), table = 12, and tabinner = sqrt(2.0).

(Wolff) Wolff and Rudd, Comp Phys Comm, 120, 200−32 (1999).

LAMMPS WWW Site − LAMMPS Documentation − LAMMPS Commands

---

pair_style command

Syntax:

```
pair_style style args
```

- style = one of the following
  - *none*
  - *buck* or *buck/coul/cut* or *buck/coul/long*
  - *dipole/cut* or *dipole/long*
  - *dpd*
  - *eam* or *eam/alloy* or *eam/fs*
  - *gran/hertzian* or *gran/history* or *gran/no_history*
  - *lj/charmm/coul/charmm* or *lj/charmm/coul/charmm/implicit* or *lj/charmm/coul/long*
  - *lj/class2* or *lj/class2/coul/cut* or *lj/class2/coul/long*
  - *lj/cut* or *lj/cut/coul/cut* or *lj/cut/coul/debye* or *lj/cut/coul/long* or *lj/cut/coul/long/tip4p*
  - *lj/expand*
  - *morse*
  - *soft*
  - *table*
  - *yukawa*
  - *hybrid*
- args = list of arguments for a particular style

```
none args = none
  buck args = cutoff
  buck/coul/cut args = cutoff (cutoff2)
  buck/coul/long args = cutoff (cutoff2)
    cutoff = cutoff for Buckingham (and Coulombic if only 1 arg)
    cutoff2 = cutoff for Coulombic (optional)
  dipole/cut args = cutoff (cutoff2)
  dipole/long args = cutoff (cutoff2)
```

```
    cutoff = cutoff for Lennard Jones (and Coulombic if only 1 arg)
    cutoff2 = cutoff for Coulombic (optional)
  dpd args = T cutoff seed
    T = temperature (temperature units)
    cutoff = cutoff for DPD interactions
    seed = random # seed (integer > 0 and <900000000)
  eam args = none
  eam/alloy args = none
  eam/fs args = none
  gran/hertzian args = Kn, gamma_n, xmu, dampflag
  gran/history args = Kn, gamma_n, xmu, dampflag
  gran/no_history args = Kn, gamma_n, xmu, dampflag
    Kn = spring constant for particle repulsion (mg/d units where
         m is mass, g is the gravitational constant,
         d is diameter of a particle)
    gamma_n = damping coefficient for normal direction collisions
              (sqrt(g/d) units)
    xmu = friction coefficient or static yield criterion
    dampflag = flag (0/1) for whether to (no/yes) include tangential damping
  lj/charmm/coul/charmm args = inner outer (inner2) (outer2)
  lj/charmm/coul/charmm/implicit args = inner outer (inner2) (outer2)
  lj/charmm/coul/long args = inner outer (cutoff)
    inner, outer = switching cutoffs for LJ (and Coulombic if only 2 args)
    inner2, outer2 = switching cutoffs for Coulombic (optional)
    cutoff = cutoff for Coulombic (optional, outer is Coulombic cutoff if only 2 args)
  lj/class2 args = cutoff
  lj/class2/coul/cut args = cutoff (cutoff2)
  lj/class2/coul/long args = cutoff (cutoff2)
    cutoff = cutoff for class2 (and Coulombic if only 1 arg)
    cutoff2 = cutoff for Coulombic (optional)
  lj/cut args = cutoff
  lj/cut/coul/cut args = cutoff (cutoff2)
  lj/cut/coul/debye args = kappa cutoff (cutoff2)
  lj/cut/coul/long args = cutoff (cutoff2)
  lj/cut/coul/long/tip4p args = cutoff cutoff2 bdist otype htype
    cutoff = cutoff for Lennard Jones (and Coulombic if only 1 arg)
    cutoff2 = cutoff for Coulombic (optional, except for tip4p)
    kappa = Debye length (inverse distance)
    bdist = distance between O and 4th TIP4P site, typical value is 0.15 Angs (distance un
    otype,htype = TIP4P O and H atom types
  lj/expand args = cutoff
    cutoff = cutoff for expanded Lennard-Jones interactions
  morse args = cutoff
    cutoff = cutoff for Morse interactions
  soft = cutoff
    cutoff = cutoff for soft interactions
  table args = style N
    style = lookup or linear or spline or bitmap = method of interpolation
    N = use N values in lookup, linear, spline tables
    N = use 2^N values in bitmap tables
  yukawa args = kappa cutoff
    kappa = screening length (inverse distance)
    cutoff = cutoff for Yukawa interactions
  hybrid args = list of one or more styles with their args
```

Examples:

```
pair_style none
pair_style eam/alloy
pair_style gran/history 200000.0 0.5 1.0 1
pair_style lj/charmm/coul/charmm 10.0 10.3
pair_style lj/charmm/coul/charmm/implicit 10.0
pair_style lj/charmm/coul/long 10.0
pair_style lj/cut 2.5
pair_style lj/cut/coul/cut 10.0 8.0
pair_style lj/cut/coul/debye 1.5 3.0
pair_style lj/cut/coul/long 12.0
pair_style lj/cut/coul/long/tip4p 12.0 12.0 0.15 7 8
pair_style lj/expand 2.5
pair_style class2 8.0
pair_style soft 2.0
pair_style dpd 1.0 1.0 48279
pair_style table linear 1000
pair_style table bitmap 12
pair_style hybrid lj/charmm/coul/long 10.0 eam
```

Description:

Set the formula(s) LAMMPS will use to computing pairwise interactions. In LAMMPS, a pairwise force field includes all pairwise interactions (Lennard Jones, Coulombic, etc), so there is a range of style choices that encompass combinations of multiple kinds of interactions.

The coefficients for the formulas for each atom type pair are set by the pair_coeff command or read from a file by the read_data or read_restart commands. Mixing and shifting of the interaction potentials is discussed is the documentation for the pair_modify command.

The cutoff arguments set global cutoffs for all atom type pairs. The global value can be overridden by the pair_coeff command for a specific pair. The pair style settings (including global cutoffs) can be changed by a subsequent pair_style command using the same style. This will reset the cutoffs for all atom type pairs, including those previously set explicitly by a pair_coeff command. The exceptions to this are that pair_style *table* and *hybrid* settings cannot be reset. A new pair_style command for these styles will wipe out all previously specified pair_coeff values.

All cutoff arguments are in distance units. The distance(s) can be smaller or larger than the dimensions of the simulation box.

In the formulas to follow, *E* is the energy of a pairwise interaction between two atoms separated by a distance *r*. The force between the atoms is the negative derivative of this expression.

Style *none* turns off pairwise interactions.

With this choice, the force cutoff is 0.0, which means that only atoms within the neighbor skin distance (see the neighbor command) are communicated between processors. You must insure the skin distance is large enough to acquire atoms needed for computing bonds, angles, etc.

A pair style of *none* will also prevent pairwise neighbor lists from being built. However if the neighbor style is *bin*, data structures for binning are still allocated. If the neighbor skin distance is small, then these data structues can consume a large amount of memory. So you should either set the neighbor style to *nsq* or set the skin distance to a larger value.

The *buck* styles compute a Buckingham potential (exp/6 instead of Lennard−Jones 12/6) given by

$$E = Ae^{-r/\rho} - \frac{C}{r^6} \qquad r < r_c$$

where A, rho, and C are coefficients defined for each pair of atom types. Rc is the cutoff.

The *buck/coul/cut* and *buck/coul/long* styles add a Coulombic term as described in the *lj/cut* styles.

The *dipole* styles are not yet implemented in LAMMPS. They will enable a point dipole and charge to be assigned to each atom and the resulting charge−dipole and dipole−dipole interactions to be computed.

Style *dpd* computes a force field for dissipative particle dynamics (DPD) following the exposition in (Groot). The force on atom I due to atom J is given as a sum of 3 terms

$$\begin{aligned}
\vec{f} &= (F^C + F^D + F^R)\hat{r_{ij}} \qquad\qquad r < r_c \\
F^C &= Aw(r) \\
F^D &= -\gamma w^2(r)(\hat{r_{ij}} \bullet \vec{v_{ij}}) \\
F^R &= \sigma w(r)\alpha(\Delta t)^{-1/2} \\
w(r) &= 1 - r/r_c
\end{aligned}$$

where FC is a conservative force, FD is a dissipative force, and FR is a random force. Rij is a unit vector in the direction Ri − Rj, Vij is the vector difference in velocities of the two atoms = Vi − Vj, alpha is a Gaussian random number with zero mean and unit variance, dt is the timestep size, and w(r) is a weighting factor that varies between 0 and 1. A, gamma, and sigma are coefficients defined for each pair of atom types. Rc is the cutoff.

Style *eam* computes pairwise interactions for metals and metal alloys using embedded−atom method (EAM) potentials (Daw). The total energy Ei of an atom I is given by

$$E_i = F_\alpha\left(\sum_{j\neq i} \rho_\alpha(r_{ij})\right) + \sum_{j\neq i}\phi_{\alpha\beta}(r_{ij})$$

where F is the embedding energy which is a function of the atomic electron density rho, phi is a pair potential interaction, and alpha and beta are the element types of atoms I and J. The multi−body nature of the EAM potential is a result of the embedding energy term. Both summations in the formula are over all neighbors J of atom I within the cutoff distance.

The cutoff distance and the tabulated values of the functionals F, rho, and phi are listed in one or more files which are specified by the pair_coeff command. These are ASCII text files in a DYNAMO−style format which is described in the documentation for the pair_coeff command. DYNAMO is a serial MD code Several DYNAMO potential files for different metals are included in the "potentials" directory of the LAMMPS distribution.

IMPORTANT NOTE: The *eam* style reads single−element EAM potentials in the DYNAMO *funcfl* format. Single element or alloy systems can be modeled with *funcfl* files and style *eam*; for the alloy case LAMMPS mixes the single−element potentials to produce alloy potentials the same way that DYNAMO does. Alternatively, DYNAMO *setfl* files can be used by LAMMPS to model alloy systems by invoking the *eam/alloy* style as described below. *Setfl* files require no mixing as they specify alloy interactions explicitly.

Style *eam/alloy* computes pairwise interactions using the same formula as style *eam*. However the associated pair_coeff command reads a DYNAMO *setfl* file instead of a *funcfl* file. *Setfl* files can be used to model a single−element or alloy system. In the alloy case, as explained above, *setfl* files contain explicit tabulated values for alloy interactions. Thus they allow more generality than *funcfl* files for modeling alloys.

Style *eam/fs* computes pairwise interactions for metals and metal alloys using a generalized form of EAM potentials due to Finnis and Sinclair (Finnis). The total energy Ei of an atom I is given by

$$E_i = F_\alpha \left(\sum_{j \neq i} \rho_{\alpha\beta}(r_{ij}) \right) + \sum_{j \neq i} \phi_{\alpha\beta}(r_{ij})$$

This has the same form as the EAM formula above, except that rho is now a functional specific to the atomic types of both atoms I and J, so that different elements can contribute differently to the total electron density at an atomic site depending on the identity of the element at that atomic site.

The associated pair_coeff command for style *eam/fs* reads a DYNAMO *setfl* file that has been extended to include additional rho_alpha_beta arrays of tabulated values. The details are given in the pair_coeff documentation.

A discussion of how FS EAM differs from conventional EAM alloy potentials is given in (Ackland1). An example of such a potential is the same author's Fe−P FS potential (Ackland2). Note that while FS potentials always specify the embedding energy with a square root dependence on the total density, the implementation in LAMMPS does not require that; the user can tabulate any functional form he desires in the FS potential files.

The *gran* styles use the following formula (Silbert) for frictional force between two granular particles that are a distance r apart when r is less than the contact distance d.

$$F = f\left(\delta/d\right)\left(k_n \delta \mathbf{n}_{ij} - m_{\mathrm{eff}} \gamma_n \mathbf{v}_n\right) + f\left(\delta/d\right)\left(-k_t \delta \mathbf{\Delta s}_t - m_{\mathrm{eff}} \gamma_t \mathbf{v}_t\right)$$

The 1st term is a normal force and the 2nd term is a tangential force. The other quantites are as follows:

- delta = d − r
- f(x) = 1 for Hookean contacts used in pair styles *history* and *no_history*
- f(x) = sqrt(x) for pair style *hertzian*
- Kn = elastic constant for normal contact (listed above)
- Kt = elastic constant for tangential contact = 2/7 of Kn
- gamma_n = viscoelastic constants for normal contact (listed above)
- gamma_t = viscoelastic constants for tangential contact = 1/2 of gamma_n
- m_eff = Mi Mj / (Mi + Mj) = effective mass of 2 particles of mass Mi and Mj
- Delta St = tangential displacement vector between the 2 spherical particles which is truncated to satisfy a frictional yield criterion
- n = a unit vector along the line connecting the centers of the 2 particles
- Vn = normal component of the relative velocity of the 2 particles
- Vt = tangential component of the relative velocity of the 2 particles

See the citation for more discussion of the granular potentials.

The *lj/charmm* styles compute LJ and Coulombic interactions with an additional switching function S(r) that ramps the energy and force smoothly to zero between an inner and outer cutoff. It is a widely used option in the CHARMM MD code.

$$
\begin{aligned}
E &= LJ(r) & r < r_{\rm in} \\
&= S(r) * LJ(r) & r_{\rm in} < r < r_{\rm out} \\
&= 0 & r > r_{\rm out} \\
E &= C(r) & r < r_{\rm in} \\
&= S(r) * C(r) & r_{\rm in} < r < r_{\rm out} \\
&= 0 & r > r_{\rm out} \\
LJ(r) &= 4\epsilon \left[\left(\frac{\sigma}{r}\right)^{12} - \left(\frac{\sigma}{r}\right)^{6} \right] \\
C(r) &= \frac{C q_i q_j}{\epsilon r} \\
S(r) &= \frac{\left[r_{\rm out}^2 - r^2\right]^2 \left[r_{\rm out}^2 + 2r^2 - 3r_{\rm in}^2\right]}{\left[r_{\rm out}^2 - {r_{\rm in}}^2\right]^3}
\end{aligned}
$$

Both the LJ and Coulombic terms require an inner and outer cutoff. They can be the same for both formulas or different depending on whether 2 or 4 arguments are used in the pair_style command. In each case, the inner cutoff distance must be less than the outer cutoff. It it typical to make the difference between the 2 cutoffs about 1.0 Angstrom.

Style *lj/charmm/coul/charmm/implicit* computes the same formulas as style *lj/charmm/coul/charmm* except that an additional 1/r term is included in the Coulombic formula. The Coulombic energy thus varies as 1/r^2. This is effectively a distance−dependent dielectric term which is a simple model for an implicit solvent with additional screening. It is designed for use in a simulation of an unsolvated biomolecule (no explicit water molecules).

Style *lj/charmm/coul/long* computes the same formulas as style *lj/charmm/coul/charmm* except that an additional damping factor is applied to the Coulombic term, as in the discussion for style *lj/cut/coul/long*. Only one Coulombic cutoff is specified for *lj/charmm/coul/long*; if only 2 arguments are used in the pair_style command, then the outer LJ cutoff is used as the single Coulombic cutoff.

The *lj/class2* styles compute a 6/9 Lennard−Jones potential given by

$$
E = \epsilon \left[2 \left(\frac{\sigma}{r}\right)^{9} - 3 \left(\frac{\sigma}{r}\right)^{6} \right] \qquad r < r_c
$$

where epsilon and sigma are coefficients defined for each pair of atom types. Rc is the cutoff.

The *lj/class2/coul/cut* and *lj/class2/coul/long* styles add a Coulombic term as described in the *lj/cut* styles.

The *lj/cut* styles compute the standard 6/12 Lennard−Jones potential, given by

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

where epsilon and sigma are coefficients defined for each pair of atom types. Rc is the cutoff.

Style *lj/cut/coul/cut* adds a Coulombic pairwise interaction given by

$$E = \frac{C q_i q_j}{\epsilon r} \qquad r < r_c$$

where C is an energy−conversion constant, Qi and Qj are the charges on the 2 atoms, and epsilon is the dielectric constant which can be set by the dielectric command. If one cutoff is specified in the pair_style command, it is used for both the LJ and Coulombic terms. If two cutoffs are specified, they are used as cutoffs for the LJ and Coulombic terms respectively.

Style *lj/cut/coul/debye* adds an additional exp() damping factor to the Coulombic term, given by

$$E = \frac{C q_i q_j}{\epsilon r} \exp(-\kappa r) \qquad r < r_c$$

where Kappa is the Debye length. This potential is another way to mimic the screening effect of a polar solvent.

Style *lj/cut/coul/long* computes the same Coulombic interactions as style *lj/cut/coul/cut* except that an additional damping factor is applied to the Coulombic term so it can be used in conjunction with the kspace_style command and its *ewald* or *pppm* option. The Coulombic cutoff specified for this style means that pairwise interactions within this distance are computed directly; interactions outside that distance are computed in K−space.

Style *lj/cut/coul/long/tip4p* is designed to implement the TIP4P model of water (Jorgensen), which introduces a massless site located a distance *bdist* away from the oxygen atom along the bisector of the HOH bond angle. *Bdist* and the atomic types of the oxygen and hydrogen atoms in TIP4P waters are specified as part of the pair_style command.

Style *lj/expand* computes a LJ interaction with a distance shifted by delta

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

The epsilon, sigma, and delta coefficients are defined for each pair of atom types. Rc is the cutoff.

Style *morse* computes pairwise interactions with the formula

$$E = D_0 \left[e^{-2\alpha(r-r_0)} - 2e^{-\alpha(r-r_0)} \right] \qquad r < r_c$$

where D0, alpha, and r0 are coefficients defined for each pair of atom types. Rc is the cutoff.

Style *soft* is useful for pushing apart overlapping atoms, since it does not blow up as r goes to 0. It computes a pairwise interaction as

$$E = A \left[1 + \cos\left(\frac{\pi r}{r_c} \right) \right] \qquad r < r_c$$

where A is a pre−factor that varies in time from the start to the end of the run. Starting and ending values for A are specified by the pair_coeff or read_data command. Rc is the cutoff.

Style *table* creates interpolation tables of length *N* from pair potential and force values listed in a file(s) as a function of distance. The files are read by the pair_coeff command which also describes the file format.

The interpolation tables are created by fitting cubic splines to the file values and interpolating energy and force values at each of *N* distances. During a simulation, these tables are used to interpolate energy and force values as needed. The interpolation is done in one of 4 styles: *lookup*, *linear*, *spline*, or *bitmap*.

For the *lookup* style, the distance between 2 atoms is used to find the nearest table entry, which is the energy or force.

For the *linear* style, the distance is used to find 2 surrounding table values from which an energy or force is computed by linear interpolation.

For the *spline* style, a cubic spline coefficients are computed and stored each of the *N* values in the table. The pair distance is used to find the appropriate set of coefficients which are used to evaluate a cubic polynomial which computes the energy or force.

For the *bitmap* style, the N means to create interpolation tables that are 2^N in length. The pair distance is used to index into the table via a fast bit−mapping technique (Wolff) and a linear interpolation is performed between adjacent table values.

Style *yukawa* computes pairwise interactions with the formula

$$E = A \frac{e^{-\kappa r}}{r} \qquad r < r_c$$

where A is a coefficient defined for each pair of atom types. Rc is the cutoff.

The *hybrid* style enables the use of multiple pair styles in one simulation. A pair style can be assigned to each pair of atom types via the pair_coeff command.

For example, a metal on a LJ surface could be computed where the metal atoms interact with each other via a *eam* potential, the surface atoms interact with each other via a *lj/cut* potential, and the metal/surface interaction is also via a *lj/cut* potential.

All pair styles that will be used must be listed in the pair_style hybrid command (in any order). Each sub−style is listed with its arguments, as illustrated in the last example above.

Restrictions:

This command must be used before any coefficients are set by the pair_coeff, read_data, or read_restart commands.

The hybrid style cannot include any of the *gran* styles in its list of styles to use. Only one *coul/long* style can be used in the list of hybrid styles.

Some pair styles are part of specific packages. They are only enabled if LAMMPS was built with that package. See the Making LAMMPS section for more info.

The *gran/hertzian*, *gran/history*, and *gran/no_history* styles are part of the "granular" package. The *lj/charmm/coul/charmm* and *lj/charmm/coul/charmm/implicit* styles are part of the "molecule" package. The *lj/cut/coul/long* and *lj/charmm/coul/long* styles are part of the "kspace" package. The *eam* styles are part of the "metal" package. The *dpd* style is part of the "dpd" package.

On some 64−bit machines, compiling with −O3 appears to break the Coulombic tabling option used by the *lj/cut/coul/long* and *lj/charmm/coul/long* styles. See the "Additional build tips" section of the Making LAMMPS documentation pages for workarounds on this issue.

Related commands:

pair_coeff, read_data, pair_modify, kspace_style, dielectric, pair_write

Default:

```
pair_style none
```

(Ackland1) Ackland, Condensed Matter (2005).

(Ackland2) Ackland, Mendelev, Srolovitz, Han and Barashev, Journal of Physics: Condensed Matter, 16, S2629 (2004).

(Daw) Daw, Baskes, Phys Rev Lett, 50, 1285 (1983). Daw, Baskes, Phys Rev B, 29, 6443 (1984).

(Finnis) Finnis, Sinclair, Philosophical Magazine A, 50, 45 (1984).

(Groot) Groot, Warren, J Chem Phys, 107, 4423 (1997).

(Jorgensen) Jorgensen, Chandrasekhar, Madura, Impey, Klein, J Chem Phys, 79, 926 (1983).

(Silbert) Silbert, Ertas, Grest, Halsey, Levine, Plimpton, Phys Rev E, 64, p 051302 (2001).

(Wolff) Wolff and Rudd, Comp Phys Comm, 120, 200−32 (1999).

pair_write command

Syntax:

```
pair_write itype jtype N style inner outer file keyword Qi Qj
```

- itype,jtype = 2 atom types
- N = # of values
- style = *r* or *rsq* or *bitmap*
- inner,outer = inner and outer cutoff (distance units)
- file = name of file to write values to
- keyword = section name in file for this set of tabulated values
- Qi,Qj = 2 atom charges (charge units) (optional)

Examples:

```
pair_write 1 3 500 r 1.0 10.0 table.txt LJ
pair_write 1 1 1000 rsq 2.0 8.0 table.txt Yukawa_1_1 −0.5 0.5
```

Description:

Write energy and force values to a file as a function of distance for the currently defined pair potential. This is useful for plotting the potential function or otherwise debugging its values. If the file already exists, the table of values is appended to the end of the file to allow multiple tables of energy and force to be included in one file.

The energy and force values are computed at distances from inner to outer for 2 interacting atoms of type itype and jtype, using the appropriate pair_coeff coefficients. If the style is *r*, then N distances are used, evenly spaced in r; if the style is *rsq*, N distances are used, evenly spaced in r^2.

For example, for N = 7, style = *r*, inner = 1.0, and outer = 4.0, values are computed at r = 1.0, 1.5, 2.0, 2.5, 3.0, 3.5, 4.0.

If the style is *bitmap*, then 2^N values are written to the file in a format and order consistent with how they are read in by the pair_coeff command for pair style *table*. For reasonable accuracy in a bitmapped table, choose N >= 12, an *inner* value that is smaller than the distance of closest approach of 2 atoms, and an *outer* value <= cutoff of the potential.

If the pair potential is computed between charged atoms, the charges of the pair of interacting atoms can optionally be specified. If not specified, values of Qi = Qj = 1.0 are used.

The file is written in the format used as input for the pair_style *table* option with *keyword* as the section name. Each line written to the file lists an index number (1−N), a distance (in distance units), an energy (in energy units), and a force (in force units).

Restrictions:

All force field coefficients for pair and other kinds of interactions must be set before this command can be invoked.

Due to how the pairwise force is computed, an inner value > 0.0 must be specified even if the potential has a finite value at r = 0.0.

Related commands:

pair_style, pair_coeff

Default: none

LAMMPS WWW Site − LAMMPS Documentation − LAMMPS Commands

---

print command

Syntax:

```
print args
```

- args = one or more text strings and variables names to print out

Examples:

```
print The system volume is now $v
```

Description:

Print the list of arguments as a line of text to the screen and/or logfile. If variables are included in the arguments, they will be evaluated and their values printed. Note that if variables are included, the print string should not be enclosed in double quotes, else it will prevent the variables from being evaluated.

By using the print command in a section of the input script that is looped over (see the jump command), and by including variables of the *equal* style (see the variable command), a string with changing values can be printed.

Restrictions: none

Related commands:

fix print, variable

Default: none

LAMMPS WWW Site − LAMMPS Documentation − LAMMPS Commands

---

processors command

Syntax:

```
processors Px Py Pz
```

- Px,Py,Pz = # of processors in each dimension of a 3d grid

Examples:

```
processors 2 4 4
```

Description:

Specify how processors are mapped as a 3d logical grid to the global simulation box.

When this command has not been specified, LAMMPS will choose Px, Py, Pz based on the dimensions of the global simulation box so as to minimize the surface/volume ratio of each processor's sub−domain.

Since LAMMPS does not load−balance by changing the grid of 3d processors on−the−fly, this command should be used to override the LAMMPS default if it is known to be sub−optimal for a particular problem. For example, a problem where the atom's extent will change dramatically over the course of the simulation.

The product of Px, Py, Pz must equal P, the total # of processors LAMMPS is running on. If multiple partitions are being used then P is the number of processors in this partition; see this section for an explanation of the −partition command−line switch.

If P is large and prime, a grid such as 1 x P x 1 will be required, which may incur extra communication costs.

Restrictions:

This command cannot be used after the simulation box is defined by a read_data or create_box command. It can be used before a restart file is read to change the 3d processor grid from what is specified in the restart file.

Related commands: none

Default:

LAMMPS chooses Px, Py, Pz

LAMMPS WWW Site − LAMMPS Documentation − LAMMPS Commands

---

read_data command

Syntax:

```
read_data file
```

- file = name of data file to read in

Examples:

```
read_data data.lj
read_data ../run7/data.polymer.gz
```

Description:

Read in a data file containing information LAMMPS needs to run a simulation. The file can be ASCII text or a gzipped text file (detected by a .gz suffix). This is one of 3 ways to specify initial atom coordinates; see the read_restart and create_atoms commands for alternative methods.

The structure of the data file is important, though many settings and sections are optional or can come in any order. See the examples directory for sample data files for different problems.

A data file has a header and a body. The header appears first. The first line of the header is always skipped; it typically contains a description of the file. Then lines are read one at a time. Lines can have a trailing comment starting with '#' that is ignored. If the line is blank (only whitespace after comment is deleted), it is skipped. If the line contains a header keyword, the corresponding value(s) is read from the line. If it doesn't contain a header keyword, the line begins the body of the file.

The body of the file contains zero or more sections. The first line of a section has only a keyword. The next line is skipped. The remaining lines of the section contain values. The number of lines depends on the section keyword as described below. Zero or more blank lines can be used between sections. Sections can appear in any order, with a few exceptions as noted below.

The formatting of individual lines in the data file (indentation, spacing between words and numbers) is not important except that header and section keywords (e.g. atoms, xlo xhi, Masses, Bond Coeffs) must be capitalized as shown and can't have extra white space between their words − e.g. two spaces or a tab between "Bond" and "Coeffs" is not valid.

These are the recognized header keywords. Header lines can come in any order. The value(s) is read from the beginning of the line. Thus the keyword *atoms* should be in a line like "1000 atoms" and the keyword *ylo yhi* should be in a line like "−10.0 10.0 ylo yhi". All these settings have a default value of 0, except the lo/hi box size defaults are −0.5 and 0.5. A line need only appear if the value is different than the default.

- *atoms* = # of atoms in system
- *bonds* = # of bonds in system
- *angles* = # of angles in system
- *dihedrals* = # of dihedrals in system
- *impropers* = # of impropers in system
- *atom types* = # of atom types in system
- *bond types* = # of bond types in system
- *angle types* = # of angle types in system
- *dihedral types* = # of dihedral types in system
- *improper types* = # of improper types in system
- *xlo xhi* = simulation box boundaries in x dimension
- *ylo yhi* = simulation box boundaries in y dimension
- *zlo zhi* = simulation box boundaries in z dimension

For 2d simulations, the *zlo zhi* values should be set to bound the z coords for atoms that appear in the file; the default of −0.5 0.5 is valid if all z coords are 0.0.

The initial simulation box size is determined by the lo/hi settings. In any dimension, the system may be periodic or non−periodic; see the boundary command.

If the system is non−periodic (in a dimension), then all atoms in the data file should have coordinates (in that dimension) between the lo and hi values. Furthermore, if running in parallel, the lo/hi values should be just a bit smaller/larger than the min/max extent of atoms. For example, if your atoms extend from 0 to 50, you should not specify the box bounds as −10000 and 10000. Since LAMMPS uses the specified box size to layout the 3d grid of processors, this will be sub−optimal and may cause a parallel simulation to lose atoms when LAMMPS shrink−wraps the box to the atoms.

If the system is periodic (in a dimension), then atom coordinates can be outside the bounds; they will be remapped (in a periodic sense) back inside the box.

These are the section keywords for the body of the file.

- *Atoms, Velocities, Masses, Dipoles* = atom−property sections
- *Bonds, Angles, Dihedrals, Impropers* = molecular topolgy sections
- *Pair Coeffs, Bond Coeffs, Angle Coeffs, Dihedral Coeffs, Improper Coeffs* = force field sections
- *BondBond Coeffs, BondAngle Coeffs, MiddleBondTorsion Coeffs, EndBondTorsion Coeffs, AngleTorsion Coeffs, AngleAngleTorsion Coeffs, BondBond13 Coeffs, AngleAngle Coeffs* = class 2 force field sections

Each section is now listed in alphabetic order. The format of each section is described including the number of lines it must contain and rules (if any) for where it can appear in the data file.

Any individual line in the various sections can have a trailing comment starting with "#" for annotation purposes. E.g. in the Atoms section:

```
10 1 17 −1.0 10.0 5.0 6.0   # salt ion
```

Angle Coeffs section:

- one line per angle type
- line syntax: ID coeffs

  ```
  ID = angle type (1−N)
  coeffs = list of coeffs
  ```
- example:

  ```
  6 70 108.5 0 0
  ```

The number and meaning of the coefficients are specific to the defined angle style. See the angle_style and angle_coeff commands for details. Coefficients can also be set via the angle_coeff command in the input script.

AngleAngle Coeffs section:

- one line per improper type
- line syntax: ID coeffs

  ```
  ID = improper type (1−N)
  ```

```
   coeffs = list of coeffs (see improper coeff)
```

AngleAngleTorsion Coeffs section:

- one line per dihedral type
- line syntax: ID coeffs

```
   ID = dihedral type (1-N)
   coeffs = list of coeffs (see dihedral coeff)
```

Angles section:

- one line per angle
- line syntax: ID type atom1 atom2 atom3

```
   ID = number of angle (1-Nangles)
   type = angle type (1-Nangletype)
   atom1,atom2,atom3 = IDs of 1st,2nd,3rd atoms in angle
```

example:

```
   2 2 17 29 430
```

The 3 atoms are ordered linearly within the angle. E.g H,O,H for a water molecule. The *Angles* section must appear after the *Atoms* section. All values in this section must be integers (1, not 1.0).

AngleTorsion Coeffs section:

- one line per dihedral type
- line syntax: ID coeffs

```
   ID = dihedral type (1-N)
   coeffs = list of coeffs (see dihedral coeff)
```

Atoms section:

- one line per atom
- line syntax: depends on atom style

This is the list of all possible quantities that can appear on each line of this section:

- atom−ID = unique integer ID of atom
- molecule−ID = integer ID of the molecule the atom belongs to
- type−ID = integer type ID of atom (1−Ntype)
- q = charge on atom
- diameter = diameter of atom
- density = density of atom
- x,y,z = coordinates of atom
- mux,muy,muz = components of dipole orientation of atom
- nx,ny,nz = image indices for atom

Which of these quantities are actually listed depends on the atom style. This is the list of which styles require each quantity:

- atom−ID = all styles
- molecule−ID = angle, bond, molecular, full styles
- type−ID = all styles
- q = charge, dipole, full styles
- diameter = granular style
- density = granular style
- x,y,z = all styles
- mux,muy,muz = dipole style
- nx,ny,nz = optional for all styles (see below)

Any quantity that is used by the atom style appears in the order listed above. Thus if the atom style is *atomic*, an atom line should have 5 quantities: atom−ID, type−ID, x, y, z. If the atom style is *hybrid eam dipole molecular*, then an atom line should have 10 quantites: atom−ID, molecule−ID, type−ID, q, x, y, z, mux, muy, muz.

The units for these quantities depend on the unit style; see the units command for details.

For 2d simulations specify z as 0.0, or whatever value is within the *zlo zhi* setting in the data file header.

The atom−ID is used to identify the atom throughout the simulation and in dump files. Normally, IDs should be values from 1 to Natoms. Values larger than Natoms can be used, but they will cause extra memory to be allocated on each processor, if an atom map array is used (see the atom_modify command).

The molecule ID is a 2nd identifier attached to an atom. It can be 0 if it is an unbonded atom or if you don't wish to assign it to a molecule numbered from 1−N.

An *Atoms* section must appear in the data file if natoms > 0 in the header section. The atoms can be listed in any order.

Atom lines (all or none of them) can optionally list 3 final integer values: nx,ny,nz. For periodic dimensions, they specify which image of the box the atom is considered to be in. An image of 0 means the box as defined. A value of 2 means add 2 box lengths to get the true value. A value of −1 means subtract 1 box length to get the true value. LAMMPS updates these flags as atoms cross periodic boundaries during the simulation. The flags can be output via the dump and dump_modify commands. If nx,ny,nz values are not set in the data file, LAMMPS initializes them to 0.

Atom velocities are set to 0.0 when the *Atoms* section is read. They may later be set by a *Velocities* section or by a velocity command in the input script.

Bond Coeffs section:

- one line per bond type
- line syntax: ID coeffs

```
ID = bond type (1−N)
coeffs = list of coeffs
```
- example:

```
4 250 1.49
```

The number and meaning of the coefficients are specific to the defined bond style. See the bond_style and bond_coeff commands for details. Coefficients can also be set via the bond_coeff command in the input script.

BondAngle Coeffs section:

- one line per angle type
- line syntax: ID coeffs

```
ID = angle type (1-N)
coeffs = list of coeffs (see class 2 section of angle_coeff)
```

BondBond Coeffs section:

- one line per angle type
- line syntax: ID coeffs

```
ID = angle type (1-N)
coeffs = list of coeffs (see class 2 section of angle_coeff)
```

BondBond13 Coeffs section:

- one line per dihedral type
- line syntax: ID coeffs

```
ID = dihedral type (1-N)
coeffs = list of coeffs (see class 2 section of dihedral_coeff)
```

Bonds section:

- one line per bond
- line syntax: ID type atom1 atom2

```
ID = bond number (1-Nbonds)
type = bond type (1-Nbondtype)
atom1,atom2 = IDs of 1st,2nd atoms in bond
```

- example:

```
12 3 17 29
```

The *Bonds* section must appear after the *Atoms* section. All values in this section must be integers (1, not 1.0).

Dihedral Coeffs section:

- one line per dihedral type
- line syntax: ID coeffs

```
ID = dihedral type (1-N)
coeffs = list of coeffs
```

- example:

```
3 0.6 1 0 1
```

The number and meaning of the coefficients are specific to the defined dihedral style. See the dihedral_style and dihedral_coeff commands for details. Coefficients can also be set via the dihedral_coeff command in the input script.

Dihedrals section:

- one line per dihedral
- line syntax: ID type atom1 atom2 atom3 atom4

```
ID = number of dihedral (1-Ndihedrals)
type = dihedral type (1-Ndihedraltype)
atom1,atom2,atom3,atom4 = IDs of 1st,2nd,3rd,4th atoms in dihedral
```

- example:

```
12 4 17 29 30 21
```

The 4 atoms are ordered linearly within the dihedral. The *Dihedrals* section must appear after the *Atoms* section. All values in this section must be integers (1, not 1.0).

Dipoles section:

- one line per atom type line syntax: ID dipole−moment

```
ID = atom type (1-N)
dipole-moment = value of dipole moment
```

- example:

```
2 0.5
```

This defines the dipole moment of each atom type (which can be 0.0 for some types). This can also be set via the dipole command in the input script.

EndBondTorsion Coeffs section:

- one line per dihedral type
- line syntax: ID coeffs

```
ID = dihedral type (1-N)
coeffs = list of coeffs (see class 2 section of dihedral_coeff)
```

Improper Coeffs section:

- one line per improper type
- line syntax: ID coeffs

```
ID = improper type (1-N)
coeffs = list of coeffs
```

- example:

```
2 20 0.0548311
```

The number and meaning of the coefficients are specific to the defined improper style. See the improper_style and improper_coeff commands for details. Coefficients can also be set via the improper_coeff command in the input script.

Impropers section:

- one line per improper
- line syntax: ID type atom1 atom2 atom3 atom4

  ```
  ID = number of improper (1-Nimpropers)
  type = improper type (1-Nimpropertype)
  atom1,atom2,atom3,atom4 = IDs of 1st,2nd,3rd,4th atoms in improper
  ```
- example:

  ```
  12 3 17 29 13 100
  ```

The *Impropers* section must appear after the *Atoms* section. All values in this section must be integers (1, not 1.0).

Masses section:

- one line per atom type
- line syntax: ID mass

  ```
  ID = atom type (1-N)
  mass = mass value
  ```
- example:

  ```
  3 1.01
  ```

This defines the mass of each atom type. This can also be set via the mass command in the input script. This section should not be used for atom styles that define a mass for individual atoms − e.g. atom style granular.

MiddleBondTorsion Coeffs section:

- one line per dihedral type
- line syntax: ID coeffs

  ```
  ID = dihedral type (1-N)
  coeffs = list of coeffs (see class 2 section of dihedral_coeff)
  ```

Pair Coeffs section:

- one line per atom type
- line syntax: ID coeffs

  ```
  ID = atom type (1-N)
  coeffs = list of coeffs
  ```
- example:

  ```
  3 0.022 2.35197 0.022 2.35197
  ```

The number and meaning of the coefficients are specific to the defined pair style. See the pair_style and pair_coeff commands for details. Coefficients can also be set via the pair_coeff command in the input script.

Velocities section:

- one line per atom
- line syntax: atom−ID vx vy vz

  ```
  atom−ID = atom ID (1−N)
  vx,vy,vz = components of velocity of the atom
  ```
- example:

  ```
  45 −3.4 0.05 1.25
  ```

The velocity lines can appear in any order. This section can only be used after an *Atoms* section. Velocities can also be set by the velocity command in the input script.

Restrictions:

To read gzipped data files, you must compile LAMMPS with the −DGZIP option − see the Making LAMMPS section of the documentation.

Related commands:

read_restart, create_atoms

Default: none

LAMMPS WWW Site − LAMMPS Documentation − LAMMPS Commands

---

read_restart command

Syntax:

```
read_restart file
```

- file = name of binary restart file to read in

Examples:

```
read_restart save.10000
```

Description:

Read in a previously saved problem from a restart file. This allows continuation of a previous run.

Restart files are saved in binary format to enable exact restarts, meaning that the trajectories of a restarted run will precisely match those produced by the original run had it continued on. Several things can prevent exact restarts due to round−off effects, in which case the trajectories in the 2 runs will slowly diverge. These include running on a different number of processors or changing certain settings such as those set by the newton or processors commands. LAMMPS will issue a WARNING in these cases. Certain fixes will also not restart exactly, though they should provide statistically similar results. These include the shake and langevin styles. If a restarted run is immediately different than the run which produced the restart file, it could be a LAMMPS bug, so consider reporting it if you think the behavior is wrong.

Because restart files are binary, they may not be portable to other machines. They can be converted to ASCII data files using the restart2data tool in the tools sub−directory of the LAMMPS distribution.

A restart file stores the units and atom style, simulation box attibutes, individual atoms and their attributes including molecular topology, force field styles and coefficients, special_bonds settings, and atom group definitions. This means that commands for these quantities do not need to be specified in your input script that reads the restart file. The exceptions to this are listed below in the Restrictions section.

Information about the kspace_style settings are not stored in the restart file. Hence if you wish to invoke an Ewald or PPPM solver, this command must be re−issued after the restart file is read.

The restart file also stores values for any fixes that require state information to enable restarting where they left off. These include the *nvt* and *npt* styles that have a global state, as well as the *msd* and *wall/gran* styles that store information about each atom.

Fix commands are not stored in the restart file which means they must be specified in the input script that reads the restart file. To re−enable a fix whose state was stored in the restart file, the fix command in the new input script must use the same fix−ID and group−ID as the input script that wrote the restart file. LAMMPS will print a message indicating that the fix is being re−enabled.

Note that no other information is stored in the restart file. This means that your new input script should specify settings for quantities like timestep size, thermodynamic and dump output, etc.

Bond interactions (angle, etc) that have been turned off by the fix shake or delete_bonds command will be written to a restart file as if they are turned on. This means they will need to be turned off again in a new run after the restart file is read.

Bonds that are broken (e.g. by a bond−breaking potential) are written to the restart file as broken bonds with a type of 0. Thus these bonds will still be broken when the restart file is read.

Restrictions:

The pair_style *eam*, *table,* and *hybrid* styles do not store coefficient data for individual atom type pairs in the restart file. Nor does the bond_style *hybrid* style. Thus you must use new pair_coeff and bond_coeff commands to read the appropriate tabulated files or reset the coefficients after the restart file is read.

Related commands:

read_data, write_restart, restart

Default: none

LAMMPS WWW Site − LAMMPS Documentation − LAMMPS Commands

---

region command

Syntax:

```
region ID style args keyword value ...
```

- ID = user−assigned name for the region
- style = *block* or *sphere* or *cylinder* or *union* or *intersect*

```
  block args = xlo xhi ylo yhi zlo zhi
        xlo,xhi,ylo,yhi,zlo,zhi = bounds of block in all
          dimensions (distance units)
    sphere args = x y z radius
        x,y,z = center of sphere (distance units)
        radius = radius of sphere (distance units)
    cylinder args = dim c1 c2 radius lo hi
      dim = x or y or z = axis of cylinder
      c1,c2 = coords of cylinder axis in other 2 dimensions (distance units)
      radius = cylinder radius (distance units)
      lo,hi = bounds of cylinder in dim (distance units)
    union args = N reg−ID1 reg−ID2 ...
      N = # of regions to follow, must be 2 or greater
      reg−ID1,reg−ID2, ... = IDs of regions to join together
    intersect args = N reg−ID1 reg−ID2 ...
      N = # of regions to follow, must be 2 or greater
      reg−ID1,reg−ID2, ... = IDs of regions to intersect
```

- zero or more keyword/value pairs may be appended to the args
- keyword = *side* or *units*

```
  side value = in or out
      in = the region is inside the specified geometry
      out = the region is outside the specified geometry
    units value = lattice or box
      lattice = the geometry is defined in lattice units
      box = the geometry is defined in simulation box units
```

Examples:

```
region 1 block −3.0 5.0 INF 10.0 INF INF
region 2 sphere 0.0 0.0 0.0 5 side out
region void cylinder y 2 3 5 −5.0 INF units box
region outside union 4 side1 side2 side3 side4
```

Description:

This command defines a geometric region of space. Various other commands use regions. For example, the region can be filled with atoms via the create_atoms command. Or the atoms in the region can be identified as a group via the group command, or deleted via the delete_atoms command.

The lo/hi values for *block* or *cylinder* styles can be specified as INF which means they extend all the way to the global simulation box boundary. If a region is defined before the simulation box has been created (via create_box or read_data or read_restart commands), then an INF parameter cannot be used.

For style *cylinder*, the c1,c2 params are coordinates in the 2 other dimensions besides the cylinder axis dimension. For dim = x, c1/c2 = y/z; for dim = y, c1/c2 = x/z; for dim = z, c1/c2 = x/y. Thus the third example above specifes a cylinder with its axis in the y−direction located at x = 2.0 and z = 3.0, with a radius of 5.0, and extending in the y−direction from −5.0 to the upper box boundary.

The *union* style creates a region consisting of the volume of all the listed regions combined. The *intesect* style creates a region consisting of the volume that is common to all the listed regions.

The *side* keyword determines whether the region is considered to be inside or outside of the specified geometry. Using this keyword in conjunction with *union* and *intersect* regions, complex geometries can be built up. For example, if the interior of two spheres were each defined as regions, and a *union* style with *side* = out was constructed listing the region−IDs of the 2 spheres, the resulting region would be all the volume in the simulation box that was outside both of the spheres.

The *units* keyword determines the meaning of the distance units used to define the region. A *box* value selects standard distance units as defined by the units command, e.g. Angstroms for units = real or metal. A *lattice* value means the distance units are in cubic lattice spacings. The lattice command must first be used to define a lattice.

Restrictions: none

Related commands:

lattice, create_atoms, delete_atoms, group

Default:

The option defaults are side = in and units = lattice.

LAMMPS WWW Site − LAMMPS Documentation − LAMMPS Commands

---

replicate command

Syntax:

```
replicate nx ny nz
```

- nx,ny,nz = replication factors in each dimension

Examples:

```
replicate 2 3 2
```

Description:

Replicate the current simulation one or more times in each dimension. For example, replication factors of 2,2,2 will create a simulation with 8x as many atoms by doubling the simulation domain in each dimension. A replication factor of 1 in a dimension leaves the simulation domain unchanged.

All properties of the atoms are replicated, including their velocities, which may or may not be desirable. New atom IDs (tags) are assigned to new atoms, as are molecule IDs. Bonds and other topology interactions are created between pairs of new atoms as well as between old and new atoms. This is done by using the image flag for each atom to "unwrap" it out of the periodic box before replicating it. This means that molecular bonds you specify in the orignal data file that span the periodic box should be between two atoms with image flags that differ by 1. This will allow them to be unwrapped appropriately.

This command operates similar to the replicate tool in the tools sub−directory of the LAMMPS distribution which creates new data files from old ones.

Restrictions:

A 2d simulation cannot be replicated in the z dimension.

If a simulation is non−periodic in a dimension, care should be used when replicating it in that dimension, as it may put atoms nearly on top of each other.

If the current simulation was read in from a restart file (before a run is performed), there can have been no fix information stored in the file for individual atoms. Similarly, no fixes can be defined at the time the replicate command is used that require vectors of atom information to be stored. This is because the replicate command does not know how to replicate that information for new atoms it creates.

Related commands: none

Default: none

LAMMPS WWW Site − LAMMPS Documentation − LAMMPS Commands

reset_timestep command

Syntax:

```
reset_timestep N
```

- N = timestep number

Examples:

```
reset_timestep 0
reset_timestep 4000000
```

Description:

Set the timestep counter to the specified value. This command normally comes after the timestep has been set by reading it in from a file or a previous simulation advanced the timestep.

The read_data and create_box commands set the timestep to 0; the read_restart command sets the timestep to the value it had when the restart file was written.

Restrictions: none

Related commands: none

Default: none

LAMMPS WWW Site − LAMMPS Documentation − LAMMPS Commands

restart command

Syntax:

```
restart 0
restart N root
restart N file1 file2
```

- N = write a restart file every this many timesteps
- root = filename to which timestep # is appended
- file1,file2 = two full filenames, toggle between them when writing file

Examples:

```
restart 0
restart 1000 poly.restart
restart 10000 poly.r.1 poly.r.2
```

Description:

Write out a binary restart file every so many timesteps as a run proceeds. A value of 0 means do not write out restart files. Using one filename as an argument will create a series of filenames with a timestep suffix, e.g. the 2nd example above will create poly.restart.1000, poly.restart.2000, poly.restart.3000, etc. Using two filenames will produce only 2 restart files. LAMMPS will toggle between the 2 names as it writes successive restart files.

See the read_restart command for information about what is stored in a restart file.

Restart files can be read by a read_restart command to restart a simulation from a particular state. Because the file is binary (to enable exact restarts), it may not be readable on another machine. In this case, the restart2data program in the tools directory can be used to convert a restart file to an ASCII data file.

Restrictions: none

Related commands:

write_restart, read_restart

Default:

```
restart 0
```

LAMMPS WWW Site − LAMMPS Documentation − LAMMPS Commands

run command

Syntax:

```
run N
```

* N = # of timesteps

Examples:

```
run 10000
```

Description:

Run or continue dynamics for the specified number of timesteps.

When the run style is *respa*, N refers to outer loop (largest) timesteps.

A value of N = 0 is acceptable; only the thermodynamics of the system are computed and printed without taking a timestep.

Restrictions: none

Related commands:

minimize, run_style, temper

Default: none

LAMMPS WWW Site − LAMMPS Documentation − LAMMPS Commands

run_style command

Syntax:

```
run_style style args
```

* style = *verlet* or *respa*

```
verlet args = none
  respa args = N n1 n2 ... keyword values ...
    N = # of levels of rRESPA
    n1, n2, ... = loop factor bewteen rRESPA levels (N−1 values)
    zero or more keyword/value pairings may be appended to the loop factors
    keyword = bond or angle or dihedral or improper or
      pair or inner or middle or outer or kspace
      bond value = M
        M = which level (1−N) to compute bond forces in
      angle value = M
        M = which level (1−N) to compute angle forces in
      dihedral value = M
        M = which level (1−N) to compute dihedral forces in
      improper value = M
        M = which level (1−N) to compute improper forces in
      pair value = M
        M = which level (1−N) to compute pair forces in
      inner values = M cut1 cut2
        M = which level (1−N) to compute pair inner forces in
        cut1 = inner cutoff between pair inner and
               pair middle or outer  (distance units)
        cut2 = outer cutoff between pair inner and
```

```
                  pair middle or outer  (distance units)
         middle values = M cut1 cut2
           M = which level (1−N) to compute pair middle forces in
           cut1 = inner cutoff between pair middle and pair outer (distance units)
           cut2 = outer cutoff between pair middle and pair outer (distance units)
         outer value = M
           M = which level (1−N) to compute pair outer forces in
         kspace value = M
           M = which level (1−N) to compute kspace forces in
```

Examples:

```
run_style verlet
run_style respa 4 2 2 2 bond 1 dihedral 2 pair 3 kspace 4
run_style respa 4 2 2 2 bond 1 dihedral 2 inner 3 5.0 6.0 outer 4 kspace 4
```

Description:

Choose the style of time integrator used for molecular dynamics simulations performed by LAMMPS.

The *verlet* style is a velocity−Verlet integrator.

The *respa* style implements the rRESPA multi−timescale integrator (Tuckerman) with N hierarchical levels, where level 1 is the innermost loop (shortest timestep) and level N is the outermost loop (largest timestep). The loop factor arguments specify what the looping factor is between levels. N1 specifies the number of iterations of level 1 for a single iteration of level 2, N2 is the iterations of level 2 per iteration of level 3, etc. N−1 looping parameters must be specified.

The timestep command sets the timestep for the outermost rRESPA level. Thus if the example command above for a 4−level rRESPA had an outer timestep of 4.0 fmsec, the inner timestep would be 8x smaller or 0.5 fmsec. All other LAMMPS commands that specify number of timesteps (e.g. neigh_modify parameters, dump every N timesteps, etc) refer to the outermost timesteps.

The rRESPA keywords enable you to specify at what level of the hierarchy various forces will be computed. If not specified, the defaults are that bond forces are computed at level 1 (innermost loop), angle forces are computed where bond forces are, dihedral forces are computed where angle forces are, improper forces are computed where dihedral forces are, pair forces are computed at the outermost level, and kspace forces are computed where pair forces are. The inner, middle, outer forces have no defaults.

The *inner* and *middle* keywords take additional arguments for cutoffs that are used by the force computations. If the 2 cutoffs for *inner* are 5.0 and 6.0, this means that all pairs up to 6.0 apart are computed by the inner force. Those between 5.0 and 6.0 have their force go ramped to 0.0 so the overlap with the next regime (middle or outer) is smooth. The next regime (middle or outer) will compute forces for all pairs from 5.0 outward, with those from 5.0 to 6.0 having their value ramped in an inverse manner.

When using rRESPA (or for any MD simulation) care must be taken to choose a timestep size(s) that insures the Hamiltonian for the chosen ensemble is conserved. For the constant NVE ensemble, total energy must be conserved. Unfortunately, it is difficult to know *a priori* how well energy will be conserved, and a fairly long test simulation (~10 ps) is usually necessary in order to verify that no long−term drift in energy occurs with the trial set of parameters.

With that caveat, a few rules−of−thumb may be useful in selecting *respa* settings. The following applies mostly to biomolecular simulations using the CHARMM or a similar all−atom force field, but the concepts are adaptable to other problems. Without SHAKE, bonds involving hydrogen atoms exhibit high−frequency vibrations and require a timestep on the order of 0.5 fmsec in order to conserve energy. The relatively inexpensive force computations for the bonds, angles, impropers, and dihedrals can be computed on this innermost 0.5 fmsec step. The outermost timestep cannot be greater than 4.0 fmsec without risking energy drift. Smooth switching of forces between the levels of the rRESPA hierarchy is also necessary to avoid drift, and a 1−2 angstrom "healing distance" (the distance between the outer and inner cutoffs) works reasonably well. We thus recommend the following settings for use of the *respa* style without SHAKE in biomolecular simulations:

```
timestep  4.0
run_style respa 4 2 2 2 inner 2 4.5 6.0 middle 3 8.0 10.0 outer 4
```

With these settings, users can expect good energy conservation and roughly a 2.5 fold speedup over the *verlet* style with a 0.5 fmsec timestep.

If SHAKE is used with the *respa* style, time reversibility is lost, but substantially longer time steps can be achieved. For biomolecular simulations using the CHARMM or similar all−atom force field, bonds involving hydrogen atoms exhibit high frequency vibrations and require a time step on the order of 0.5 fmsec in order to conserve energy. These high frequency modes also limit the outer time step sizes since the modes are coupled. It is therefore desireable to use SHAKE with respa in order to freeze out these high frequency motions and increase the size of the time steps in the respa hierarchy. The following settings can be used for biomolecular simulations with SHAKE and rRESPA:

```
fix             2 all shake 0.000001 500 0 m 1.0 a 1
timestep        4.0
run_style       respa 2 2 inner 1 4.0 5.0 outer 2
```

With these settings, users can expect good energy conservation and roughly a 1.5 fold speedup over the *verlet* style with SHAKE and a 2.0 fmsec timestep.

For non−biomolecular simulations, the *respa* style can be advantageous if there is a clear separation of time scales − fast and slow modes in the simulation. Even a LJ system can benefit from rRESPA if the interactions are divided by the inner, middle and outer keywords. A 2−fold or more speedup can be obtained while maintaining good energy conservation. In real units, for a pure LJ fluid at liquid density, with a sigma of 3.0 angstroms, and epsilon of 0.1 Kcal/mol, the following settings seem to work well:

```
timestep  36.0
run_style respa 3 3 4 inner 1 3.0 4.0 middle 2 6.0 7.0 outer 3
```

Restrictions: none

Whenever using rRESPA, the user should experiment with trade−offs in speed and accuracy for their system, and verify that they are conserving energy to adequate precision.

Related commands:

timestep, run

Default:

```
run_style verlet
```

(Tuckerman) Tuckerman, Berne and Martyna, J Chem Phys, 97, p 1990 (1992).

LAMMPS WWW Site − LAMMPS Documentation − LAMMPS Commands

set command

Syntax:

```
set group−ID style value
```

- group−ID = ID of group
- style = *atom* or *bond* or *angle* or *dihedral* or *improper* or *charge* or *dipole*
- value = value to set selected atoms to

Examples:

```
set solvent atom 2
set edge bond 4
set half charge 0.5
```

Description:

Set an attribute for atoms in the group. Since these attributes are assigned by the read_data, read_restart or create_atoms commands, this command changes those assignments. This can be useful for altering pairwise and molecular force interactions, since force−field coefficients are defined in terms of types. It can also be used to change the labeling of atoms when they are output in dump files.

For style *atom*, the atom type of all atoms in the group is changed to the specified value from 1 to ntypes. Note that ntypes must be within the range the simulation was initialized for. The maximum number of types is set by the create_box command or the *atom types* field in the header of the data file read by the read_data command.

For style *bond*, *angle*, *dihedral*, or *improper*, the bond type (angle type, etc) of all bonds (angles, etc) of atoms in the group is changed to the specified value from 1 to nbondtypes (angletypes, etc). All atoms in the bond (angle, etc) must be in the group in order for the change to be made. The maximum number of types is set by the *bond types* (*angle types*, etc) field in the header of the data file.

For style *charge*, the charge of each atom in the group is set to the specified value.

For style *dipole*, the specified value is used as a random number seed. The dipole moment of each atom in the group is set to a random orientation with a magnitude determined by the dipole command setting for that atom type.

Restrictions:

This command requires inter−processor communication to coordinate the setting of bond types (angle types, etc). This means that pairwise force cutoffs must be already be set before using this command, so that each processor can acquire the correct atoms. This is not necessary to reset atom types.

Related commands:

create_box, create_atoms, read_data

Default: none

LAMMPS WWW Site − LAMMPS Documentation − LAMMPS Commands

special_bonds command

Syntax:

```
special_bonds style
special_bonds c1 c2 c3
special_bonds c1 c2 c3 c4 c5 c6
```

- style = *charmm* or *amber*
- c1,c2,c3,c4,c5,c6 = numeric coefficients from 0.0 to 1.0

Examples:

```
special_bonds charmm
special_bonds amber
special_bonds 0.0 0.0 1.0
special_bonds 0.0 0.0 1.0 0.0 0.0 0.5
```

Description:

Set the weighting coefficients for the pairwise force and energy contributions from atom pairs that are also bonded to each other directly or indirectly. The 1st coefficient is the weighting factor on 1−2 atom pairs, which are those directly bonded to each other. The 2nd coefficient is the weighting factor on 1−3 atom pairs which are those separated by 2 bonds (e.g. the 2 H atoms in a water molecule). The 3rd coefficient is the weighting factor on 1−4 atom pairs which are separated by 3 bonds (e.g. the 1st and 4th atoms in a dihedral interaction).

1−3, and 1−4 interactions are not computed using the list of angles and dihedrals defined in the simulation. Rather, they are inferred by the set of defined bonds. This distinction is important to remember if bonds are removed at some point during a simulation. Also note that turning off a bond (as opposed to removing it) does not change the inference of 1−2, 1−3, and 1−4 neighbors. See the delete_bonds command for more details.

The *charmm* style sets all 3 coefficients to 0.0, which is the default for the CHARMM force field. In pair styles *lj/charmm/coul/charmm* and *lj/charmm/coul/long* the 1−4 coefficients are defined explicitly, and these pair−wise contributions are computed in the charmm dihedral style − see the pair_coeff and dihedral_style commands for more information.

The *amber* style sets the 3 coefficients to 0.0 0.0 0.5 for LJ interactions and to 0.0 0.0 0.833 for Coulombic interactions, which is the default for a particular version of the AMBER force field, where the last value is 5/6.

A special_bonds command with 3 coefficients sets the 1−2, 1−3, and 1−4 coefficients for both LJ and Coulombic terms to those values.

A special_bonds command with 6 coefficients sets the 1−2, 1−3, and 1−4 LJ coefficients to the first 3 values and the Coulombic coefficients to the last 3 values.

Restrictions: none

Related commands:

delete_bonds

Default:

```
special_bonds 0.0 0.0 0.0
```

LAMMPS WWW Site − LAMMPS Documentation − LAMMPS Commands

---

temp_modify command

Syntax:

```
temp_modify temp−ID keyword value ...
```

- temp−ID = ID of temperature to modify
- one or more keyword/value pairs may be listed
- keyword = *extra*

```
  extra value = N
      N = # of extra degrees of freedom to subtract
```

Examples:

```
temp_modify mine extra 0
```

Description:

Modify the parameters of a previously defined temperature command.

The *extra* keyword refers to how many degrees−of−freedom are subtracted from 3N as a normalizing factor in the temperature computation. The default is 3 which is a correction factor for an ensemble of velocities with zero total linear momentum.

Restrictions: none

Related commands:

temperature

Default:

The option defaults are extra = 3.

LAMMPS WWW Site − LAMMPS Documentation − LAMMPS Commands

---

temper command

Syntax:

```
temper N M temp fix−ID seed1 seed2 index
```

- N = total # of timesteps to run
- M = attempt a tempering swap every this many steps
- temp = initial temperature for this ensemble
- fix−ID = ID of the fix that will control temperature during the run
- seed1 = random # seed used to decide on adjacent temperature to partner with
- seed2 = random # seed for Boltzmann factor in Metropolis swap
- index = which temperature (0 to N−1) I am simulating (optional)

Examples:

```
temper 100000 100 $t tempfix 0 58728
temper 40000 100 $t tempfix 0 32285 $w
```

Description:

Run a parallel tempering (replica exchange) simulation of multiple ensembles of a system on multiple partitions of processors. The processor partitions are defined using the −partition command−line switch (see this section). Each ensemble's temperature is typically controlled at a different value by a fix with ID *fix−ID* that controls temperature. Possible fix styles are nvt, npt, and temp/rescale. The desired temperature is specified by *temp*, which is typically a variable previously set in the input script, so that each partition is assigned a different temperature. See the variable command for more details. For example,

```
variable t world 300.0 310.0 320.0 330.0
```

As the tempering simulation runs for *N* timesteps, a swap between adjacent ensembles will be attempted every *M* timesteps. If *seed1* is 0, then the swap attempts will alternate between odd and even pairings. If *seed1* is non−zero then it is used as a seed in a random number generator to randomly choose an odd or even pairing each time. Each attempted swap of temperatures is either accepted or rejected based on a Boltzmann−weighted Metropolis criterion which uses *seed2* in the random number generator.

The last argument *index* is optional and is used when restarting a tempering run from a set of restart files (one for each replica) which had previously swapped to new temperatures. The *index* value (from 0 to N−1, where N is the # of replicas) identifies which temperature the replica was simulating on the timestep the restart files were written. Obviously, this argument must be a variable so that each partition has the correct value. Set the variable to the *N* values listed in the log file for the previous run for the replica temperatures at that timestep. For example if the log file listed

```
500000 2 4 0 1 3
```

then a setting of

```
variable w proc 2 4 0 1 3
```

would be used to restart the run with a tempering command like the example above with $w as the last argument.

Restrictions: none

Related commands:

variable

Default: none

---

temperature command

Syntax:

```
temperature ID group−ID style args keyword value ...
```

- ID = user−assigned name for the temperature
- group−ID = ID of group of atoms to compute the temperature for
- style = *full* or *partial* or *ramp* or *region*

```
  full args = none
   partial args = x y z
      x,y,z = 0 or 1
   ramp args = vdim vlo vhi dim clo chi
      vdim = vx or vy or vz
      vlo,vhi = subtract velocities between vlo and vhi (velocity units)
      dim = x or y or z
      clo,chi = lower and upper bound of domain to
                subtract from (distance units)
    region args = region−ID
```

- zero or more keyword/value pairs may be appended to the args
- keyword = *units*

```
  units value = lattice or box
```

Examples:

```
temperature mine peptide full
temperature new flow partial 1 1 0
temperature 2 all region border
temperature 2nd middle ramp vx 0 8 y 2 12 units lattice
```

Description:

Define a method for computing the temperature of a group of atoms.

The ID of the temperature can be referred to in other commands which perform or modify temperature computations: thermo_modify, velocity, fix_modify, temp_modify.

The style determines how the temperature is computed. The *full* style means KE = dim/2 N k T, where KE = total kinetic energy of the group of atoms (sum of 1/2 m v^2), dim = 2 or 3 = dimensionality of the simulation, N = number of atoms in the group, k = Boltzmann constant, and T = temperature.

The *partial* style uses the same formula as *full*, except entire dimensions can be eliminated from the kinetic energy computation. This can be useful for systems where atoms are flowing, and only the thermal temperature in the non−flow directions is desired. A "0" means do not use the component of velocity in that dimension when computing KE. In the example above with arguments 1 1 0, only x and y velocities (not z) would be used in computing KE and temperature.

The *ramp* style can be used to eliminate an imposed velocity on a system before computing thermal KE. The meaning of these arguments is the same as for the velocity command which was presumably used to impose the velocity.

The *region* style will compute the temperature of the atoms that are both in the group and in the region volume of the specified region ID. This is useful for thermostatting on a varying set of atoms that fall within a geometric region of the simulation domain, where those atoms can change from one timestep to the next.

The *units* keyword determines the meaning of the distance units used for coordinates (c1,c2) and velocities (vlo,vhi) defined for the *ramp* style. A *box* value selects standard distance units as defined by the units command, e.g. Angstroms for units = real or metal. A *lattice* value means the distance units are in cubic lattice spacings; e.g. lattice spacings / tau. The lattice command must first be used to define a lattice.

A temperature with ID = *default* is pre−defined by LAMMPS and uses to a *full* style computation on the *all* group of atoms. All operations in LAMMPS that compute temperatures use the *default* ID unless the input script changes it.

Restrictions: none

Related commands:

thermo_modify, velocity, fix_modify, temp_modify

Default:

A temperature with ID = *default* is defined by LAMMPS, as if it had been specified as "temperature default all full". The option default is units = lattice.

LAMMPS WWW Site − LAMMPS Documentation − LAMMPS Commands

---

thermo command

Syntax:

```
thermo N
```

- N = output thermodynamics every N timesteps

Examples:

```
thermo 100
```

Description:

Compute and print thermodynamics (temperature, energy, pressure) every N timesteps. A value of 0 will only compute thermodynamics at the beginning and end of a simulation.

Restrictions: none

Related commands:

thermo_style, thermo_modify

Default:

```
thermo 0
```

thermo_modify command

Syntax:

```
thermo_modify keyword value ...
```

- one or more keyword/value pairs may be listed
- keyword = *temp* or *lost* or *norm* or *flush* or *line* or *format*

```
temp value = ID of temperature
  lost value = error or warn or ignore
  norm value = yes or no
  flush value = yes or no
  line value = one or multi
  format value = int string or float string or N string
    N = integer from 1 to # of quantities being printed
    string = C−style format string
  window value = N
    N = number of previous print−outs to average over
```

Examples:

```
thermo_modify temp mydef
thermo_modify lost no flush yes
thermo_modify line multi format float %g format 3 %15.8g
```

Description:

Set options for how thermodynamics are computed and printed by LAMMPS.

The *temp* option allows you to specify which temperature computation will be used when thermodynamic info that uses temperature is computed and displayed (temperature, kinetic energy, pressure). Different ways to compute temperature can be defined by the user via the temperature command.

The *lost* option determines whether LAMMPS checks for lost atoms each time it computes thermodynamics and what it does if atoms are lost. If the value is *ignore*, LAMMPS does not check for lost atoms. If the value is *error* or *warn*, LAMMPS checks and either issues an error or warning. The code will exit with an error and continue with a warning. This can be a useful debugging option.

The *norm* option determines whether the thermodynamic print−out is normalized by the number of atoms or is the total summed across all atoms. Different atom styles have different defaults for this setting.

The *flush* option invokes a flush operation after thermodynamic info is written to the log file. This insures the output in that file is current (no buffering by the OS), even if LAMMPS halts before the simulation completes.

The *line* option determines whether thermodynamics will be printed as a series of numeric values on one line or in a multi−line format with 3 quantities with text strings per line and a dashed−line header containing the timestep and CPU time. This modify option overrides the *one* and *multi* thermo_style settings.

The *format* option sets the numeric format of individual printed quantities. The *int* and *float* settings set the format for all integer or floating−point quantities printed. The setting with a numeric value (e.g. format 5 %10.4g) sets the format of the Nth value printed. If the format for a specific value has been set, it will take precedent over the *int* or *float* setting.

The *window* option sets the number of previous thermodynamic screen outputs over which thermo_style custom *ave* quantities are averaged when printed.

Restrictions: none

Related commands:

thermo, thermo_style, temperature

Default:

The option defaults are temp = default, lost = error, norm = no for unit style of *lj*, norm = yes for unit style of *real* and *metal*, flush = no, window = 10. The defaults for the line and format options depend on the thermo style. For styles "one", "granular", and "custom" the line and format defaults are "one", "%8d", and "%12.8g". For style "multi", the line and format defaults are "multi", "%8d", and "%14.4f".

LAMMPS WWW Site − LAMMPS Documentation − LAMMPS Commands

thermo_style command

Syntax:

```
thermo_style style args
```

- style = *one* or *multi* or *granular* or *custom*
- args = list of arguments for a particular style

```
one args = none
  multi args = none
  granular args = none
  custom args = list of attributes
    possible attributes = step, atoms, cpu, temp, press, pe, ke, eng,
             evdwl, ecoul, epair, ebond, eangle, edihed, eimp, emol, elong,
             vol, lx, ly, lz, pxx, pyy, pzz, pxy, pxz, pyz
             gke, grot, tave, pave, eave, peave
      step = timestep
      atoms = # of atoms
      cpu = elapsed CPU time
```

```
             temp = temperature
             press = pressure
             pe = total potential energy
             ke = kinetic energy
             eng = total energy (pe + ke)
             evdwl = VanderWaal pairwise energy
             ecoul = Coulombic pairwise energy
             epair = pairwise (evdwl + ecoul)
             ebond = bond energy
             eangle = angle energy
             edihed = dihedral energy
             eimp = improper energy
             emol = molecular energy (ebond + eangle + edihed + eimp)
             elong = long−range kspace energy
             vol = volume
             lx,ly,lz = box lengths in x,y,z
             pxx,pyy,pzz,pxy,pxz,pyz = 6 components of pressure tensor
             gke = granular translational kinetic energy (without frozen atoms)
             grot = granular rotational kinetic energy (without frozen atoms)
             tave, pave, eave, peave = time−averaged temp, press, eng, pe
```

Examples:

```
thermo_style multi
thermo_style custom step temp pe eng press vol
```

Description:

Set the style in which thermodynamic data is printed to the screen and log file.

Style *one* prints a one−line summary of thermodynamic info that is the equivalent of "thermo_style custom step temp epair emol eng press". The line contains only numeric values.

Style *multi* prints a multiple−line listing of thermodynamic info that is the equivalent of "thermo_style custom eng ke temp pe ebond eangle edihed eimp evdwl ecoul elong press". The listing contains numeric values and a string ID for each quantity.

Style *granular* is used with atom style granular and prints a one−line numeric summary that is the equivalent of "thermo_style custom step atoms gke grot".

Style *custom* is the most general setting and allows you to specify which of the quantities listed above you want printed on each thermodynamic timestep.

Also, all styles except *custom* have *vol* added to their list of outputs as a final printed quantity when the simulation box volume changes during the simulation.

The time−averaged quantities *tave*, *pave*, *eave*, *peave* are averaged over the last N thermodynamic outputs to the screen (not the last N timesteps), where N is the value set by the *window* option of the thermo_modify command (N = 10 by default).

Some fixes also generate quantities that can be appended to these lists each time thermodyanmic info prints out if enabled by the fix_modify command. See invididual fix commands for more details, e.g. the fix nvt and fix npt commands.

Options invoked by the thermo_modify command can be used to set the one− or multi−line format of the print−out, the normalization of energy quantities (total or per−atom), and the numeric precision of each printed value.

Restrictions:

Atom style granular cannot compute the usual temperature and pressure settings because it stores atom masses differently. The gke and grot settings should be used instead (or use thermo style granular).

This command must come after the simulation box is defined by a read_data, read_restart, or create_box command.

Related commands:

thermo, thermo_modify, fix_modify

Default:

```
thermo_style one
```

LAMMPS WWW Site − LAMMPS Documentation − LAMMPS Commands

timestep command

Syntax:

```
timestep dt
```

- dt = timestep size (time units)

Examples:

```
timestep 2.0
timestep 0.003
```

Description:

Set the timestep size for subsequent molecular dynamics simulations. See the units command for a discussion of time units. Note that using the units command also sets the timestep to its default value in the chosen units.

When the run style is *respa*, dt is the timestep for the outer loop (largest) timestep.

Restrictions: none

Related commands:

run, run_style respa, units

Default:

timestep = 0.005 tau for units = lj
timestep = 1.0 fmsec for units = real
timestep = 0.001 psec for units = metal

LAMMPS WWW Site − LAMMPS Documentation − LAMMPS Commands

undump command

Syntax:

```
undump dump−ID
```

- dump−ID = ID of previously defined dump

Examples:

```
undump mine
undump 2
```

Description:

Turn off a previously defined dump so that it is no longer active. This closes the file associated with the dump.

Restrictions: none

Related commands:

dump

Default: none

LAMMPS WWW Site − LAMMPS Documentation − LAMMPS Commands

unfix command

Syntax:

```
unfix fix−ID
```

- fix−ID = ID of a previously defined fix

Examples:

```
unfix 2
unfix lower−boundary
```

Description:

Turn off a fix that was previously defined with a fix command.

Restrictions: none

Related commands:

fix

Default: none

---

units command

Syntax:

```
units style
```

- style = *lj* or *real* or *metal*

Examples:

```
units metal
units lj
```

Description:

This command sets the style of units used for a simulation. It detemines the units of all quantities specified in the input script and data file, as well as quantities output to the screen, log file, and dump files. Typically, this command is used at the very beginning of an input script.

For style *lj*, all quantities are unitless:

- distance = sigma
- time = tau
- mass = one
- energy = epsilon
- velocity = sigma/tau
- force = epsilon/sigma
- temperature = reduced LJ temperature
- pressure = reduced LJ pressure
- charge = reduced LJ charge
- electric field = force/charge

For style *real*, these are the units:

- distance = Angstroms
- time = femtoseconds
- mass = grams/mole
- energy = Kcal/mole
- velocity = Angstroms/femtosecond
- force = Kcal/mole−Angstrom
- temperature = degrees K

- pressure = atmospheres
- charge = multiple of electron charge (+1.0 is a proton)
- electric field = volts/Angstrom

For style *metal,* these are the units:

- distance = Angstroms
- time = picoseconds
- mass = grams/mole
- energy = eV
- velocity = Angstroms/picosecond
- force = eV/Angstrom
- temperature = degrees K
- pressure = bars
- charge = multiple of electron charge (+1.0 is a proton)
- electric field = volts/Angstrom

This command also sets the timestep size and neighbor skin distance to default values for each style. For style *lj* these are dt = 0.005 tau and skin = 0.3 sigma. For style *real* these are dt = 1.0 fmsec and skin = 2.0 Angstroms. For style *metal* these are dt = 0.001 psec and skin = 2.0 Angstroms.

Restrictions:

This command cannot be used after the simulation box is defined by a read_data or create_box command.

Related commands: none

Default:

```
units lj
```

LAMMPS WWW Site − LAMMPS Documentation − LAMMPS Commands

variable command

Syntax:

```
variable name style args ...
```

- name = single lower−case character, 'a' thru 'z'
- style = *index* or *loop* or *equal* or *world* or *universe*

```
index args = one or more strings
  loop args = N = integer size of loop
  equal args = one string containing functions, vectors, keywords, numbers
    functions = add(x,y), sub(x,y), mult(x,y), div(x,y), neg(x), pow(x,y), exp(x), ln(x)
    vectors = x[5], y[12], z[17], vx[88], vy[19], vz[2], fx[1], fy[2005], fz[1]
    keywords = same keywords (mostly) as in thermo_style custom command
  world args = one string for each partition of processors
  universe args = one or more strings
```

Examples:

```
variable x index run1 run2 run3 run4 run5 run6 run7 run8
variable a loop 20
variable b equal div(temp,3.0)
variable b equal add(x[234],mult(0.5,lx))
variable t world 300.0 310.0 320.0 330.0
variable x universe 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15
```

Description:

This command assigns one or more values to a variable name so that the variable can be used in subsequent input script commands. In this context a "value" is a string which could be text or numbers, as in the examples above. As explained in this section, occurrences of $X in an input script line are replaced by the variable's value, where X is a single lower−case character from "a" to "z".

As described below, for variable styles *index*, *loop*, and *universe*, the value assigned to a variable can be incremented via the next command. When there are no more values to assign, the variable is "exhausted" and a flag is set that causes the next jump command encountered in the input script to be skipped. This enables the construction of simple loops in the input script that are iterated over and exited from.

When a variable command is encountered for a variable that has already been specified, the command is skipped. This is the case for all variable styles except *equal*, so that *equal*−style variable names can be re−used and re−defined anytime. Skipping allows you to loop over the same input script many times without re−defining your variables. When a variable is exhausted via the next command, it is then available to be re−defined in a subsequent variable command.

For the *index* style, one or more strings are specified. Initially, the 1st string is assigned to the variable. Each time a next command is used with the variable name, the next string is assigned. All processors assign the same string to the variable. *Index*−style variables can also be set (with a single value) by using the command−line switch −var; see this section for details.

The *loop* style is identical to the *index* style except that the strings are the integers from 1 to N. Initially, the string "1" is assigned to the variable. Each time a next command is used with the variable name, the next string ("2", "3", etc) is assigned. All processors assign the same string to the variable.

For the *equal* style, a single string is specified which represents an equation that will be evaluated afresh each time the variable is used. Thus the variable can take on different values at different stages of the input script. For example, if the variable is used in a fix print command, it could print different values each timestep it was invoked. The next command cannot be used with *equal*−style variables, since there is only one value. Note that, as with any other input script command, it is feasible to use another variable in the *equal* variable's string, e.g. variable y equal mult($x,2). However, $x will be replaced immediately by it's current value when the command is first parsed, not each time that $y is substituted for.

The syntax of the equation assigned to *equal* variables is simple. It can contain "functions", "vectors", "keywords", or "numbers" in any combination.

- Function = a keyword followed by parenthesis with one or two arguments
- Supported functions = add(x,y), sub(x,y), mult(x,y), div(x,y), neg(x), pow(x,y), exp(x), ln(x)
- Example function usage = div(1.0e20,3.0), neg(x[34]), pow(lx,3.0)
- Vector = a keyword followed by square brackets containing an atom ID
- Supported vectors = x, y, z, vx, vy, vz, fx, fy, fz
- Example vector usage = x[123], fz[1000]

- Keyword = keywords supported by the thermo_style custom command except cpu and pressure tensor components (pxx, pyy, etc)
- Supported keywords = step, atoms, temp, press, pe, ke, eng, evdwl, ecoul, epair, ebond, eangle, edihed, eimp, emol, elong, vol, lx, ly, lz, gke, grot
- Example keyword usage = atoms, pow(vol,0.333), mult(elong,0.5)
- Number = 0.2, 1.0e20, −15.4, etc

Keywords have restrictions on when they can be assigned to variables. For example, keywords that compute thermodynamic quantites can only be invoked after the first simulation has begun. A warning is issued if thermodyanmic keywords are invoked on timesteps when thermodynamic information is not being printed to the screen, since the values assigned to the variable may be out−of−date.

The variable *equal* equation can also be nested in that function arguments can be functions, vectors, keywords, or numbers. For example, this is a valid equation:

```
variable x equal div(add(pe,ke),pow(vol,div(1,3)))
```

For the *world* style, one or more strings are specified. There must be one string for each processor partition or "world". See this section of the manual for information on running LAMMPS with multiple partitions via the "−partition" command−line switch. This variable command assigns one string to each world. All processors in the world are assigned the same string. The next command cannot be used with *equal*−style variables, since there is only one value per world. This style of variable is useful when you wish to run different simulations on different partitions, or when performing a parallel tempering simulation (see the temper command), to assign different temperatures to different partitions.

For the *universe* style, one or more strings are specified. There must be at least as many strings as there are processor partitions or "worlds". See this page for information on running LAMMPS with multiple partitions via the "−partition" command−line switch. This variable command initially assigns one string to each world. When a next command is encountered using this variable, the first processor partition to encounter it, is assigned the next available value. This continues until all the variable values are consumed. Thus, this command can be used to run 50 simulations on 8 processor partitions. The simulations will be run one after the other on whatever partition becomes available, until they are all finished. *Universe*−style variables are incremented using the files "tmp.lammps.variable" and "tmp.lammps.variable.lock" which you will see in your directory during such a LAMMPS run.

If a variable command is encountered when the variable has already been defined, the command is ignored. Thss allows an input script with a variable command to be processed multiple times; see the jump or include commands. It also means that the use of the command−line switch −var will override a corresponding variable setting in the input script.

Restrictions:

The use of atom vectors in *equal* style variables requires the atom style to use a global mapping in order to look up the vector indices. Only atom styles with molecular information create global maps.

Related commands:

next, jump, include, temper, fix print, print

Default: none

velocity command

Syntax:

```
velocity group−ID style args keyword value ...
```

- group−ID = ID of group of atoms whose velocity will be changed
- style = *create* or *set* or *scale* or *ramp* or *zero*

```
create args = temp seed
    temp = temperature value (temperature units)
    seed = random # seed (8 digits or less)
  set args = vx vy vz
    vx,vy,vz = velocity value or NULL (velocity units)
  scale args = temp
    temp = temperature value (temperature units)
  ramp args = vdim vlo vhi dim clo chi
    vdim = vx or vy or vz
    vlo,vhi = lower and upper velocity value (velocity units)
    dim = x or y or z
    clo,chi = lower and upper coordinate bound (distance units)
  zero args = linear or angular
    linear = zero the linear momentum
    angular = zero the angular momentum
```

- zero or more keyword/value pairs may be appended to the args
- keyword = *dist* or *sum* or *mom* or *rot* or *temp* or *loop* or *units*

```
dist value = uniform or gaussian
  sum value = no or yes
  mom value = no or yes
  rot value = no or yes
  temp value = temperature ID
  loop value = all or local or geom
  units value = box or lattice
```

Examples:

```
velocity all create 300.0 4928459 rot yes dist gaussian
velocity border set NULL 4.0 3.0 sum yes units box
velocity flow scale 300.0
velocity flow ramp lattice vx 0.0 5.0 y 5 20 temp mytemp
velocity all zero linear
```

Description:

Set or change the velocities of a group of atoms in one of several styles. For each style, there are required arguments and optional keyword/value parameters. Not all options are used by each style. Each option has a default as listed below.

The *create* style generates an ensemble of velocities using a random number generator with the specified seed as the specified temperature.

The *set* style sets the velocities of all atoms in the group to the specified values. If any component is specified as NULL, then it is not set.

The *scale* style computes the current temperature of the group of atoms and then rescales the velocities to the specified temperature.

The *ramp* style is similar to that used by the temperature ramp command. Velocities ramped uniformly from vlo to vhi are applied to dimension vx, or vy, or vz. The value assigned to a particular atom depends on its relative coordinate value (in dim) from clo to chi. For the example above, an atom with y−coordinate of 10 (1/4 of the way from 5 to 20), would be assigned a x−velocity of 1.25 (1/4 of the way from 0.0 to 5.0). Atoms outside the coordinate bounds (less than 5 or greater than 20 in this case), are assigned velocities equal to vlo or vhi (0.0 or 5.0 in this case).

The *zero* style adjusts the velocities of the group of atoms so that the aggregate linear or angular momentum is zero. No other changes are made to the velocities of the atoms.

All temperatures specified in the velocity command are in temperature units; see the units command. The units of velocities and coordinates depend on whether the *units* keyword is set to *box* or *lattice*, as discussed below.

The keyword/value option pairs are used in the following ways by the various styles.

The *dist* option is used by *create*. The ensemble of generated velocities can be a *uniform* distribution from some minimum to maximum value, scaled to produce the requested temperature. Or it can be a *gaussian* distribution with a mean of 0.0 and a sigma scaled to produce the requested temperature.

The *sum* option is used by all styles, except *zero*. The new velocities will be added to the existing ones if sum = yes, or will replace them if sum = no.

The *mom* and *rot* options are used by *create*. If mom = yes, the linear momentum of the newly created ensemble of velocities is zeroed; if rot = yes, the angular momentum is zeroed.

The *temp* option is used by *create* and *scale* to specify a user−defined temperature computation. If this option is not used, the default temperature (which has style *full*) is computed for the group of atoms specified in the velocity command. If the temperature should have degrees−of−freedom removed due to SHAKE constraints, then the appropriate fix shake command must be specified before the velocity command is issued.

The *loop* option is used by *create*. If loop = all, then each processor loops over all atoms in the simulation to create velocities, but only stores velocities for atoms it owns. This can be a slow loop for a large simulation. It will produce the same set of velocities, independent of the number of processors, if atoms were read from a data file. It will not produce such independent velocities if atoms were created using the create_atoms command. If loop = local, then each processor loops over only its atoms to produce velocities. The random number seed is adjusted to give a different set of velocities on each processor. This is a fast loop, but will always produce different sets of velocities when a simulation is run on a different number of processors. If loop = geom, then each processor loops over only its atoms. For each atom a unique random number seed is created, based on the atom's xyz coordinates. A velocity is generated using that seed. This is a fast loop and will always give the same set of velocities, independent of how many processors are used. However, the generated velocities may be more correlated than if the *all* or *local* options are used. Note that the *loop geom* option will not necessarily assign identical velocities for two simulations run on different machines. This is because the computations based on xyz coordinates are sensitive to tiny differences in the double−precision value for a coordinate as stored on a particular machine.

The *units* option is used by *set* and *ramp*. If units = box, the velocities and coordinates specified in the velocity command are in the standard units described by the units command (e.g. Angstroms/fmsec for real units). If units = lattice, velocities are in units of lattice spacings per time (e.g. spacings/fmsec) and coordinates are in lattice spacings. The lattice command must have been previously used to define the lattice spacing.

For all styles, no atoms are assigned z−component velocities if the simulation is 2d; see the dimension command.

Restrictions: none

Related commands:

fix shake, lattice

Default:

The option defaults are dist = uniform, sum = no, mom = yes, rot = no, temp = default, loop = all, and units = lattice.

write_restart command

Syntax:

```
write_restart file
```

- file = name of file to write restart information to

Examples:

```
write_restart restart.equil
```

Description:

Write a binary restart file of the current state of the simulation. See the read_restart command for information about what is stored in a restart file.

During a long simulation, the restart command is typically used to dump restart files periodically. The write_restart command is useful after a minimization or whenever you wish to write out a single current restart file.

Restart files can be read by a read_restart command to restart a simulation from a particular state. Because the file is binary (to enable exact restarts), it may not be readable on another machine. In this case, the restart2data program in the tools directory can be used to convert a restart file to an ASCII data file.

Restrictions: none

Related commands:

restart, read_restart

Default: none