thermo Documentation

Alexander Gabourie

Sep 25, 2020

Contents

The thermo, or thermo, package is a set of Python tools to interface with the molecular dynamics (MD) simulator GPUMD (and LAMMPS for comparison) for the purpose of thermodynamic simulations (i.e. thermal conductivity).

Currently, the functionality is limited as it serves specific research purposes at this time; however, the long-term plan is to make this package to primarily serve GPUMD with only minor supporting functions for LAMMPS for the purpose of checking force-consistency between the simulators.

The documenation is produced and maintained by Alex Gabourie at Stanford University. It outlines the structure and usage of the thermo package.

CHAPTER 1

Documentation

This package contains four subpackages:
1. **gpumd** : Python interface specific to GPUMD.
2. **lammps** : Python interface specific to LAMMPS.
3. **shared** : Used strictly to compare GPUMD and LAMMPS.
4. **tools** : Extra support for more general MD related content.

1.1 gpumd

1.1.1 Calculations

thermo.gpumd.calc.**get_gkma_kappa**(*data*, *nbins*, *nsamples*, *dt*, *sample_interval*, *T=300*, *vol=1*, *max_tau=None*, *directions='xyz'*, *outputfile='heatmode.npy'*, *save=False*, *directory=None*, *return_data=True*)

Calculate the Green-Kubo thermal conductivity from modal heat current data from 'load_heatmode'

Args:

data (dict): Dictionary with heat currents loaded by 'load_heatmode'

nbins (int): Number of bins used during the GPUMD simulation

nsamples (int): Number of times heat flux was sampled with GKMA during GPUMD simulation

dt (float): Time step during data collection in fs

sample_interval (int): Number of time steps per sample of modal heat flux

T (float): Temperature of system during data collection

vol (float): Volume of system in angstroms^3

max_tau (float): Correlation time to calculate up to. Units of ns

directions (str): Directions to gather data from. Any order of 'xyz' is accepted. Excluding directions also allowed (i.e. 'xz' is accepted)

outputfile (str): File name to save read data to. Output file is a binary dictionary. Loading from a binary file is much faster than re-reading data files and saving is recommended

save (bool): Toggle saving data to binary dictionary. Loading from save file is much faster and recommended

directory (str): Name of directory storing the input file to read

return_data (bool): Toggle returning the loaded modal heat flux data. If this is False, the user should ensure that save is True

Returns: dict: Input data dict but with correlation, thermal conductivity, and lag time data included

Table 1: Output dictionary (new entries)

key	tau	kmxi	kmxo	kmyi	kmyo	kmz	jmxijx	jmx-ojx	jmyijy	jmy-ojy	jmzjz
units	ns	$Wm^{-1} K^{-1} x^{-1}$	$Wm^{-1} K^{-1} x^{-1}$	$Wm^{-1} K^{-1} x^{-1}$	$Wm^{-1} K^{-1} x^{-1}$	$Wm^{-1} K^{-1} x^{-1}$	$eV^3 amu^{-1} x^{-1}$	$eV^3 amu^{-1} x^{-1}$	$eV^3 amu^{-1} x^{-1}$	$eV^3 amu^{-1} x^{-1}$	$eV^3 amu^{-1} x^{-1}$

Here x is the size of the bins in THz. For example, if there are 4 bins per THz, x = 0.25 THz.

thermo.gpumd.calc.**hnemd_spectral_kappa**(*shc*, *Fe*, *T*, *V*)

Spectral thermal conductivity calculation from an SHC run

Args:

shc (dict): The data from a single SHC run as output by thermo.gpumd.data.load_shc

Fe (float): HNEMD force in (1/A)

T (float): HNEMD run temperature (K)

V (float): Volume (A^3) during HNEMD run

Returns: dict: Same as shc argument, but with spectral thermal conductivity included

Table 2: Output dictionary (new entries)

key	kwi	kwo
units	$Wm^{-1} K^{-1} THz^{-1}$	$Wm^{-1} K^{-1} THz^{-1}$

thermo.gpumd.calc.**running_ave**(*kappa*, *time*)

Gets running average. Reads and returns the structure input file from GPUMD.

Args: kappa (ndarray): Raw thermal conductivity time (ndarray): Time vector that kappa was sampled at

Returns: ndarray: Running average of kappa input

1.1.2 Data Loaders

thermo.gpumd.data.**get_frequency_info**(*bin_f_size*, *eigfile='eigenvector.out'*, *directory=None*)

Gathers eigen-frequency information from the eigenvector file and sorts it appropriately based on the selected frequency bins (identical to internal GPUMD representation).

Args:

bin_f_size (float): The frequency-based bin size (in THz)

eigfile (str): The filename of the eigenvector output/input file created by GPUMD phonon package

directory (str): Directory eigfile is stored

Returns: dict: Dictionary with the system eigen-freqeuency information along with binning information

Table 3: Output dictionary

key	fq	fmax	fmin	shift	nbins	bin_count	bin_f_size
units	THz	THz	THz	N/A	N/A	N/A	THz

thermo.gpumd.data.**load_compute**(*quantities=None*, *directory=None*, *filename='compute.out'*)

Loads data from compute.out GPUMD output file.

Currently supports loading a single run.

Args:

quantities (str or list(str)): Quantities to extract from compute.out Accepted quantities are:

['T', 'U', 'F', 'W', 'jp', 'jk'].

Other quantity will be ignored.

T=temperature, U=potential, F=force, W=virial, jp=heat current (potential), jk=heat current (kinetic)

directory (str): Directory to load compute file from

filename (str): file to load compute from

Returns: Dictionary containing the data from compute.out

Table 4: Output dictionary

key	T	U	F	W	jp	jk	Ein	Eout
units	K	eV	eVA^{-1}	eV	$eV^{3/2}$ $amu^{-1/2}$	$eV^{3/2}$ $amu^{-1/2}$	eV	eV

thermo.gpumd.data.**load_dos**(*num_dos_points*, *directory=None*, *filename='dos.out'*)

Loads data from dos.out GPUMD output file.

Args:

num_dos_points (int or list(int)): Number of frequency points the DOS is computed for.

directory (str): Directory to load 'dos.out' file from (dir. of simulation)

filename (str): File to load DOS from.

Returns: dict(dict)): Dictonary with DOS data. The outermost dictionary stores each individual run.

Table 5: Output dictionary

key	nu	DOSx	DOSy	DOSz
units	THz	THz^{-1}	THz^{-1}	THz^{-1}

thermo.gpumd.data.**load_force**(*n*, *directory=None*, *filename='force.out'*)

Loads data from force.out GPUMD output file.

Currently supports loading a single run.

Args:

n (int): Number of atoms force is output for

directory (str): Directory to load force file from

filename (str): Name of force data file

Returns: Numpy array of shape (n,3,-1) containing all forces (ev/A) from filename

thermo.gpumd.data.**load_hac**(*Nc*, *output_interval*, *directory=None*, *filename='hac.out'*)

Loads data from hac.out GPUMD output file.

Args:

Nc (int or list(int)): Number of correlation steps

output_interval (int or list(int)): Output interval for HAC and RTC data

directory (str): Directory containing hac data file

filename (str): The hac data file

Returns: dict: A dictionary containing the data from hac runs

Table 6: Output dictionary

key	t	kxi	kxo	kyi	kyo	kz	jxijx	jxojx	jyijy	jyojy	jzjz
units	ps	$Wm^{-1}K^{-1}$	$Wm^{-1}K^{-1}$	$Wm^{-1}K^{-1}$	$Wm^{-1}K^{-1}$	$Wm^{-1}K^{-1}$	eV^3amu^{-1}	eV^3amu^{-1}	eV^3amu^{-1}	eV^3amu^{-1}	eV^3amu^{-1}

thermo.gpumd.data.**load_heatmode**(*nbins*, *nsamples*, *directory=None*, *inputfile='heatmode.out'*, *directions='xyz'*, *outputfile='heatmode.npy'*, *ndiv=None*, *save=False*, *multiprocessing=False*, *ncore=None*, *block_size=65536*, *return_data=True*)

Loads data from heatmode.out GPUMD file. Option to save as binary file for fast re-load later. WARNING: If using multiprocessing, memory usage may be significantly larger than file size

Args:

nbins (int): Number of bins used during the GPUMD simulation

nsamples (int): Number of times heat flux was sampled with GKMA during GPUMD simulation

directory (str): Name of directory storing the input file to read

inputfile (str): Modal heat flux file output by GPUMD

directions (str): Directions to gather data from. Any order of 'xyz' is accepted. Excluding directions also allowed (i.e. 'xz' is accepted)

outputfile (str): File name to save read data to. Output file is a binary dictionary. Loading from a binary file is much faster than re-reading data files and saving is recommended

ndiv (int): Integer used to shrink number of bins output. If originally have 10 bins, but want 5, ndiv=2. nbins/ndiv need not be an integer

save (bool): Toggle saving data to binary dictionary. Loading from save file is much faster and recommended

multiprocessing (bool): Toggle using multi-core processing for conversion of text file

ncore (bool): Number of cores to use for multiprocessing. Ignored if multiprocessing is False

block_size (int): Size of block (in bytes) to be read per read operation. File reading performance depend on this parameter and file size

return_data (bool): Toggle returning the loaded modal heat flux data. If this is False, the user should ensure that save is True

Returns: dict: Dictionary with all modal heat fluxes requested

Table 7: Output dictionary

key	nbins	nsamples	jmxi	jmxo	jmyi	jmyo	jmz
units	N/A	N/A	$eV^{3/2}$ $amu^{-1/2}$ x^{-1}	$eV^{3/2}$ $amu^{-1/2}$ x^{-1}	$eV^{3/2}$ $amu^{-1/2}$ x^{-1}	$eV^{3/2}$ $amu^{-1/2}$ x^{-1}	$eV^{3/2}$ $amu^{-1/2}$ x^{-1}

Here x is the size of the bins in THz. For example, if there are 4 bins per THz, x = 0.25 THz.

thermo.gpumd.data.**load_kappa**(*directory=None*, *filename='kappa.out'*)

Loads data from kappa.out GPUMD output file which contains HNEMD kappa.

Args:

directory (str): Directory containing kappa data file

filename (str): The kappa data file

Returns: dict: A dictionary with keys corresponding to the columns in 'kappa.out'

Table 8: Output dictionary

key	kxi	kxo	kyi	kyo	kz
units	Wm^{-1} K^{-1}	Wm^{-1} K^{-1}	Wm^{-1} K^{-1}	Wm^{-1} K^{-1}	Wm^{-1} K^{-1}

thermo.gpumd.data.**load_kappamode**(*nbins*, *nsamples*, *directory=None*, *inputfile='kappamode.out'*, *directions='xyz'*, *outputfile='kappamode.npy'*, *ndiv=None*, *save=False*, *multiprocessing=False*, *ncore=None*, *block_size=65536*, *return_data=True*)

Loads data from kappamode.out GPUMD file. Option to save as binary file for fast re-load later. WARNING: If using multiprocessing, memory useage may be significantly larger than file size

Args:

nbins (int): Number of bins used during the GPUMD simulation

nsamples (int): Number of times thermal conductivity was sampled with HNEMA during GPUMD simulation

directory (str): Name of directory storing the input file to read

inputfile (str): Modal thermal conductivity file output by GPUMD

directions (str): Directions to gather data from. Any order of 'xyz' is accepted. Excluding directions also allowed (i.e. 'xz' is accepted)

outputfile (str): File name to save read data to. Output file is a binary dictionary. Loading from a binary file is much faster than re-reading data files and saving is recommended

ndiv (int): Integer used to shrink number of bins output. If originally have 10 bins, but want 5, ndiv=2. nbins/ndiv need not be an integer

save (bool): Toggle saving data to binary dictionary. Loading from save file is much faster and recommended

multiprocessing (bool): Toggle using multi-core processing for conversion of text file

ncore (bool): Number of cores to use for multiprocessing. Ignored if multiprocessing is False

block_size (int): Size of block (in bytes) to be read per read operation. File reading performance depend on this parameter and file size

return_data (bool): Toggle returning the loaded modal thermal conductivity data. If this is False, the user should ensure that save is True

Returns: dict: Dictionary with all modal thermal conductivities requested

Table 9: Output dictionary

key	nbins	nsamples	kmxi	kmxo	kmyi	kmyo	kmz
units	N/A	N/A	$Wm^{-1} K^{-1} x^{-1}$	$Wm^{-1} K^{-1} x^{-1}$	$Wm^{-1} K^{-1} x^{-1}$	$Wm^{-1} K^{-1} x^{-1}$	$Wm^{-1} K^{-1} x^{-1}$

Here x is the size of the bins in THz. For example, if there are 4 bins per THz, x = 0.25 THz.

thermo.gpumd.data.**load_omega2**(*directory=None*, *filename='omega2.out'*)
Loads data from omega2.out GPUMD output file.

Args:

directory (str): Directory to load force file from

filename (str): Name of force data file

Returns: Numpy array of shape (N_kpoints,3*N_basis) in units of THz. N_kpoints is number of k points in kpoint.in and N_basis is the number of basis atoms defined in basis.in

thermo.gpumd.data.**load_saved_heatmode**(*filename='heatmode.npy'*, *directory=None*)
Loads data saved by the 'load_heatmode' or 'get_gkma_kappa' function and returns the original dictionary.

Args:

filename (str): Name of the file to load

directory (str): Directory the data file is located in

Returns: dict: Dictionary with all modal heat flux previously requested

thermo.gpumd.data.**load_saved_kappamode**(*filename='kappamode.npy'*, *directory=None*)
Loads data saved by the 'load_kappamode' function and returns the original dictionary.

Args:

filename (str): Name of the file to load

directory (str): Directory the data file is located in

Returns: dict: Dictionary with all modal thermal conductivities previously requested

thermo.gpumd.data.**load_sdc**(*Nc*, *directory=None*, *filename='sdc.out'*)
Loads data from sdc.out GPUMD output file.

Args:

Nc (int or list(int)): Number of time correlation points the VAC/SDC is computed for

directory (str): Directory to load 'sdc.out' file from (dir. of simulation)

filename (str): File to load SDC from

Returns:

dict(dict): Dictonary with SDC/VAC data. The outermost dictionary stores each individual run

Table 10: Output dictionary

key	t	VACx	VACy	VACz	SDCx	SDCy	SDCz
units	ps	$A^2 ps^{-2}$	$A^2 ps^{-2}$	$A^2 ps^{-2}$	$A^2 ps^{-1}$	$A^2 ps^{-1}$	$A^2 ps^{-1}$

thermo.gpumd.data.**load_shc**(*Nc*, *num_omega*, *directory=None*, *filename='shc.out'*)
Loads the data from shc.out GPUMD output file.

Args:

Nc (int or list(int)): Maximum number of correlation steps. If multiple shc runs, can provide a list of Nc.

num_omega (int or list(int)): Number of frequency points. If multiple shc runs, can provide a list of num_omega.

directory (str): Directory to load 'shc.out' file from (dir. of simulation)

filename (str): File to load SHC from.

Returns: dict: Dictionary of in- and out-of-plane shc results (average)

Table 11: Output dictionary

key	t	Ki	Ko	nu	jwi	jwo
units	ps	A eV ps^{-1}	A eV ps^{-1}	THz	A eV ps^{-1} THz^{-1}	A eV ps^{-1} THz^{-1}

thermo.gpumd.data.**load_thermo**(*directory=None*, *filename='thermo.out'*)
Loads data from thermo.out GPUMD output file.

Args:

directory (str): Directory to load thermal data file from

filename (str): Name of thermal data file

Returns: 'output' dictionary containing the data from thermo.out

Table 12: Output dictionary

key	T	K	U	Px	Py	Pz	Lx	Ly	Lz	ax	ay	az	bx	by	bz	cx	cy	cz
units	K	eV	eV	GPa	GPa	GPa	A	A	A	A	A	A	A	A	A	A	A	A

thermo.gpumd.data.**load_vac**(*Nc*, *directory=None*, *filename='mvac.out'*)
Loads data from mvac.out GPUMD output file.

Args:

Nc (int or list(int)): Number of time correlation points the VAC is computed for

directory (str): Directory to load 'mvac.out' file from

filename (str): File to load VAC from

Returns:

dict(dict): Dictonary with VAC data. The outermost dictionary stores each individual run

Table 13: Output dictionary

key	t	VACx	VACy	VACz
units	ps	A^2 ps^{-2}	A^2 ps^{-2}	A^2 ps^{-2}

thermo.gpumd.data.**load_velocity**(*n*, *directory=None*, *filename='velocity.out'*)
Loads data from velocity.out GPUMD output file.

Currently supports loading a single run.

Args:

n (int): Number of atoms velocity is output for

directory (str): Directory to load velocity file from

filename (str): Name of velocity data file

Returns: Numpy array of shape (n,3,-1) containing all forces (A/ps) from filename

thermo.gpumd.data.**reduce_frequency_info**(*freq*, *ndiv=1*)

Recalculates frequency binning information based on how many times larger bins are wanted.

Args: freq (dict): Dictionary with frequency binning information from the get_frequency_info function output

ndiv (int): Integer used to shrink number of bins output. If originally have 10 bins, but want 5, ndiv=2. nbins/ndiv need not be an integer

Returns: dict: Dictionary with the system eigen freqeuency information along with binning information

thermo.gpumd.data.**tail**(*f*, *nlines*, *BLOCK_SIZE=32768*)

Reads the last nlines of a file.

Args:

f (filehandle): File handle of file to be read

nlines (int): Number of lines to be read from end of file

BLOCK_SIZE (int): Size of block (in bytes) to be read per read operation. Performance depend on this parameter and file size.

Returns: list: List of ordered final nlines of file

Additional Information: Since GPUMD output files are mostly append-only, this becomes useful when a simulation prematurely ends (i.e. cluster preempts run, but simulation restarts elsewhere). In this case, it is not necessary to clean the directory before re-running. File outputs will be too long (so there still is a storage concern), but the proper data can be extracted from the end of file. This may also be useful if you want to only grab data from the final m number of runs of the simulation

1.1.3 Input/Output

thermo.gpumd.io.**ase_atoms_to_gpumd**(*atoms*, *M*, *cutoff*, *gpumd_file='xyz.in'*, *sort_key=None*, *order=None*, *group_index=None*)

Converts ASE atoms to GPUMD compatible position file.

Args:

atoms (ase.Atoms): Atoms to write to gpumd file

M (int): Maximum number of neighbors for one atom

cutoff (float): Initial cutoff distance for building the neighbor list

gpumd_file (str): File to save the structure data to

sort_key (str): How to sort atoms ('group', 'type').

order (list(type)): List to sort by. Provide str for 'type', and int for 'group'

group_index (int): Selects the group to sort in the output.

thermo.gpumd.io.**convert_gpumd_atoms**(*in_file='xyz.in'*, *out_filename='in.xyz'*, *format='xyz'*, *atom_types=None*)

Converts the GPUMD input structure file to any compatible ASE output structure file. **Warning: Info dictionary may not be preserved**.

Args:

in_file (str): GPUMD position file to get structure from

out_filename (str): Name of output file after conversion

format (str): ASE supported output format

atom_types (list(str)): List of atom types (elements).

thermo.gpumd.io.**create_basis**(*atoms*)

Creates the basis.in file. Atoms passed to this must already have the basis of every atom defined.

Related: preproc.add_basis, preproc.repeat

Args:

atoms (ase.Atoms): Atoms of unit cell used to generate basis.in

thermo.gpumd.io.**create_kpoints**(*atoms*, *path='G'*, *npoints=1*, *special_points=None*)

Creates the file "kpoints.in", which specifies the kpoints needed for src/phonon

Args:

atoms (ase.Atoms): Unit cell to use for phonon calculation

path (str): String of special point names defining the path, e.g. 'GXL'

npoints (int): Number of points in total. Note that at least one point is added for each special point in the path

special_points (dict): Dictionary mapping special points to scaled kpoint coordinates. For example `{'G': [0, 0, 0], 'X': [1, 0, 0]}`

Returns: tuple: First element is the kpoints converted to x-coordinates, second the x-coordinates of the high symmetry points, and third the labels of those points.

thermo.gpumd.io.**import_trajectory**(*filename='movie.xyz'*, *in_file=None*, *atom_types=None*)

Reads the trajectory from GPUMD run and creates a list of ASE atoms.

Args:

filename (str): Name of the file that holds the GPUMD trajectory.

in_file (str): Name of the original structure input file. Not required, but can help load extra information not included in trajectory output.

atom_types (list(str)): List of atom types (elements).

Returns: list(ase.Atoms): A list of ASE atoms objects.

thermo.gpumd.io.**lammps_atoms_to_gpumd**(*filename*, *M*, *cutoff*, *style='atomic'*, *gpumd_file='xyz.in'*)

Converts a lammps data file to GPUMD compatible position file.

Args:

filename (str): LAMMPS data file name

M (int): Maximum number of neighbors for one atom

cutoff (float): Initial cutoff distance for building the neighbor list

style (str): Atom style used in LAMMPS data file

gpumd_file (str): File to save the structure data to

thermo.gpumd.io.**load_xyz**(*filename='xyz.in'*, *atom_types=None*)
Reads and returns the structure input file from GPUMD.

Args:

filename (str): Name of structure file

atom_types (list(str)): List of atom types (elements).

Returns: tuple: atoms, M, cutoff

atoms (ase.Atoms): ASE atoms object with x,y,z, mass, group, type, cell, and PBCs from input file. group is stored in tag, atom type may not correspond to correct atomic symbol

M (int): Max number of neighbor atoms

cutoff (float): Initial cutoff for neighbor list build

1.1.4 Preprocessing

thermo.gpumd.preproc.**add_basis**(*atoms*, *index=None*, *mapping=None*)
Assigns a basis index for each atom in atoms. Updates atoms.

Args:

atoms (ase.Atoms): Atoms to assign basis to.

index (list(int)): Atom indices of those in the unit cell. Order is important.

mapping (list(int)): Mapping of all atoms to the relevant basis positions

thermo.gpumd.preproc.**add_group_by_position**(*split*, *atoms*, *direction*)
Assigns groups to all atoms based on its position. Only works in one direction as it is used for NEMD. Returns a bookkeeping parameter, but atoms will be udated in-place.

Args:

split (list(float)): List of boundaries. First element should be lower boundary of sim. box in specified direction and the last the upper.

atoms (ase.Atoms): Atoms to group

direction (str): Which direction the split will work.

Returns: int: A list of number of atoms in each group.

thermo.gpumd.preproc.**add_group_by_type**(*atoms*, *types*)
Assigns groups to all atoms based on atom types. Returns a bookkeeping parameter, but atoms will be udated in-place.

Args:

atoms (ase.Atoms): Atoms to group

types (dict): Dictionary with types for keys and group as a value. Only one group allowed per atom. Assumed groups are integers starting at 0 and increasing in steps of 1. Ex. range(0,10).

Returns: int: A list of number of atoms in each group.

thermo.gpumd.preproc.**repeat**(*atoms*, *rep*)
A wrapper of ase.Atoms.repeat that is aware of GPUMD's basis information.

Args:

atoms (ase.Atoms): Atoms to assign velocities to.

rep (int | list(3 ints)): List of three positive integers or a single integer

thermo.gpumd.preproc.**set_velocities**(*atoms*, *custom=None*)

Sets the 'velocity' part of the atoms to be used in GPUMD. Custom velocities must be provided. They must also be in the units of eV^(1/2) amu^(-1/2).

Args:

atoms (ase.Atoms): Atoms to assign velocities to.

custom (list(list)): list of len(atoms) with each element made from a 3-element list for [vx, vy, vz]

1.2 lammps

1.2.1 Calculations

thermo.lammps.calc.**get_GKTC**(*directory='.'*, *T=300*, *vol=1*, *dt=None*, *rate=None*, *tau=None*, *heat-flux_file='heat_out.heatflux'*, *mat_file='heat_flux.mat'*)

Gets the thermal conductivity vs. time profile using the Green-Kubo formalism. thermal conductivity vector and time vector. Assumptions with no info given by user: dt = 1 fs, vol = 1, T=300, rate=dt, tau=total time

Args:

directory (string): This is the directory in which the simulation results are located. If not provided, the current directory is used.

T (float): This is the temperature at which the equlibrium simulation was run at. If not provided, T=300 is used. Units are in [K]

vol (float): This is the volume of the simulation system. If not provided, vol=1 is used. Units are [angstroms^3].

dt (float): This is the timestep of the green-kubo part of the simulation. If not provided, dt=1 fs is used. units are in [fs]

rate (int): This is the rate at which the heat flux is sampled. This is in number of timesteps. If not provided, we assume we sample once per timestep so, rate=dt

tau (int): max lag time to integrate over. This is in units of [ns]

heatflux_file (str): Filename of heatflux output. If not provided 'heat_out.heatflux' is used.

mat_file (str): MATLAB file to load, if exists. If not provided, 'heat_flux.mat' will be used. Also used as filename for saved MATLAB file.

Returns: dict: kx, ky, kz, t, directory, dt, tot_time, tau, T, vol, srate, jxjx, jyjy, jzjz

Output keys:

- kx (ndarray): x-direction thermal conductivity [W/m/K]
- ky (ndarray): y-direction thermal conductivity [W/m/K]
- kz (ndarray): z-direction thermal conductivity [W/m/K]
- t (ndarray): time [ns]
- directory (str): directory of results
- dt (float): timestep [fs]
- tot_time (float): total simulated time [ns]
- tau (int): Lag time [ns]

- T (float): [K]
- vol (float): Volume of simulation cell [angstroms^3]
- srate (float): See above
- jxjx (ndarray): x-direction heat flux autocorrelation
- jyjy (ndarray): y-direction heat flux autocorrelation
- jzjz (ndarray): z-direction heat flux autocorrelation

thermo.lammps.calc.**get_heat_flux**(*directory='.'*, *heatflux_file='heat_out.heatflux'*, *mat_file='heat_flux.mat'*)

Gets the heat flux from a LAMMPS EMD simulation. Creates a compressed .mat file if only in text form. Loads .mat form if exists.

Args:

directory (str): This is the directory in which the simulation results are located. If not provided, the current directory is used.

heatflux_file (str): Filename of heatflux output. If not provided 'heat_out.heatflux' is used.

mat_file (str): MATLAB file to load, if exists. If not provided, 'heat_flux.mat' will be used. Also used as filename for saved MATLAB file.

Returns: dict:Jx (list), Jy (list), Jz (list), rate (float)

1.2.2 Data Loaders

thermo.lammps.data.**extract_dt**(*log_file*)

Finds all time steps given in the lammps output log

Args:

log_file (str): LAMMPS log file to examine

Returns: list(float): The timesteps found in log_file in [ps]

thermo.lammps.data.**get_dimensions**(*filename*, *directory=None*)

Gets the dimensions of a 3D simulation from a LAMMPS trajectory.

Args:

filename (str): LAMMPS trajectory file to extract dimensions from

directory (str): The directory the trajectory file is found in

Returns: dict: Dictionary with keys given in the table below

Table 14: Output dictionary

key	x	y	z	A	V	xy	xz	yz
units	distance	distance	distance	distance^2	distance^3	distance	distance	distance

1.2.3 Input/Output

thermo.lammps.io.**ase_atoms_to_lammps**(*atoms*, *out_file='atoms.data'*, *add_masses=True*)

Converts ASE atoms to a lammps data file.

Args:

atoms (ase.Atoms): Atoms to write to lammps data file

gpumd_file (str): File to save the structure data to

1.3 shared

1.3.1 Force Comparison

thermo.shared.force.**compare_forces**(*f1_dict*, *f2_dict*)

Compares the LAMMPS and GPUMD forces and returns dictionary of comparison Forces are dict2 - dict1 values.

Args:

arg1 (dict): f1_dict dictionary containing extracted forces from a GPUMD or LAMMPS simulation

arg2 (dict): f2_dict dictionary containing extracted forces from a GPUMD or LAMMPS simulation

Returns: dict: comparison dictionary

thermo.shared.force.**load_forces**(*force_file*, *sim*)

Loads the forces from either GPUMD or LAMMPS output to facilitate a comparison between techniques.

Args:

arg1 (str) [force_file] Filename with forces

arg2 (str) [sim] If type == 'LAMMPS': The file path should be for the LAMMPS output forces LAMMPS file should be in the format given by the following LAMMPS input command: force all custom 1 <file> id fx fy fz If type == 'GPUMD': the force output file (f.out) path when GPUMD is compiled with the force flag

Returns: dict: dictionary containing sorted force vectors

1.4 tools

1.4.1 Lennard Jones

class thermo.tools.lj.**LJ**(*symbols=None*, *ignore_pairs=None*, *cut_scale=2.5*)

Bases: object

Stores all atoms for a simulation with their LJ parameters.

A special dictionary with atom symbols for keys and the epsilon and sigma LJ parameters for the values. This object interfaces with the UFF LJ potential parameters but can also accept arbitrary parameters.

Args:

symbols (str or list(str)): Optional input. A single symbol or a list of symbols to add to the initial LJ list.

ignore_pairs (list(sets)): List of sets where each set has two elements. Each element is a string for the symbol of the atom to ignore in that pair. Order in set is not important.

cut_scale (float): Specifies the multiplicative factor to use on the sigma parameter to define the cutoffs. Default is 2.5.

acknowledge_pair(*pair*)

Removes the pair from the ignore list and acknowledges it during the output.

Args:

pair (set): A two-element set where each entry is a string of the symbol in the pair to un-ignore.

acknowledge_pairs(*pairs*)
Removes pairs from the ignore list.

Args:

pairs (list(set)): A list of two-elements sets where each entry in each set is a string of the symbol in the pair to un-ignore.

add_UFF_params(*symbols*, *replace=False*)
Adds UFF parameters to the LJ object. Will replace existing parameters if 'replace' is set to True. UFF parameters are loaded from the package.

Args:

symbols (str or list(str)): A single symbol or a list of symbols to add to the initial LJ list.

replace (bool): Whether or not to replace existing symbols

add_param(*symbol*, *data*, *replace=True*)
Adds a custom parameter to the LJ object.

Args:

symbol (str): Symbol of atom type to add.

data (tuple(float)): A two-element tuple of numbers to represent the epsilon and sigma LJ values.

replace (bool): Whether or not to replace the item.

create_file(*filename='ljparams.txt'*, *atom_order=None*)
Outputs a GPUMD style LJ parameters file using the atoms defined in atom_order in the order defined in atom_order.

Args:

filename (str): The filename or full path with filename of the output.

atom_order (list(str)): List of atom symbols to include LJ params output file. The order will determine the order in the output file. *Required* Ex. ['a', 'b', 'c'] = pairs => 'aa', 'ab', 'ac', 'ba', 'bb', 'bc', 'ca', 'cb' 'cc' in this order.

custom_cutoff(*pair*, *cutoff*)
Sets a custom cutoff for a specific pair of atoms.

Args:

pair (set): A two-element set where each entry is a string of the symbol in the pair.

cutoff (float): Custom cutoff to use. In Angstroms.

ignore_pair(*pair*)
Adds a pair to the list of pairs that will be ignored when output to file.

Args:

pair (set): A two-element set where each entry is a string of the symbol in the pair to ignore.

ignore_pairs(*pairs*)
Adds a list of pairs that will be ignored when output to file.

Args:

pairs (list(set)): A list of two-element sets where each entry of each set is a string of the symbol in the pair to ignore.

remove_custom_cutoff(*pair*)
Removes a custom cutoff for a pair of atoms.

Args:

pair (set): A two-element set where each entry is a string of the symbol in the pair.

remove_param(*symbol*)
Removes an element from the LJ object. If item does not exist, nothing happens.

Args:

symbol (str): Symbol of atom type to remove.

replace_UFF_params(*symbols*, *add=False*)
Replaces current LJ parameters with UFF values. Will add new entries if 'add' is set to True. UFF parameters are loaded from the package.

Args:

symbols (str or list(str)): A single symbol or a list of symbols to add to the initial LJ list.

add (bool): Whether or not to replace existing symbols

set_cut_scale(*cut_scale*)
Sets the amount to scale the sigma values of each pair by to set the cutoff. Warning: setting this will remove any global cutoff, but leave custom cutoffs.

Args:

cut_scale (float): Scaling factor to be used on sigma

set_global_cutoff(*cutoff*)
Sets a global cutoff for all pairs. Warning: setting this will remove all other cutoff parameters.

Args:

cutoff (float): Custom cutoff to use. In Angstroms.

thermo.tools.lj.**lb_mixing**(*a1*, *a2*)
Applies Lorentz-Berthelot mixing rules on two atoms.

Args:

a1 (tuple): Tuple of (epsilon, sigma)

a2 (tuple): Tuple of (epsilon, sigma)

thermo.tools.lj.**load_UFF**()
Loads dictionary that stores relevant LJ from UFF.

Returns:

dict: Dictionary with atom symbols as the key and a tuple of epsilon and sigma in units of eV and Angstroms, respectively.

- genindex

Python Module Index

t

Index

R

S

T