thermoMD Documentation

Alexander Gabourie

Jul 06, 2019

Contents:

The thermodynamic Molecular Dynamics, or thermoMD, package is a set of Python tools to interface with the molecular dynamics (MD) simulators GPUMD and LAMMPS 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 thermoMD package.

CHAPTER 1

Documentation

1.1 thermo

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.

Each subpackage has multiple corresponding modules. They will be listed as:
<short description> - <module name>

1.1.1 gpumd

Calculations - calc

thermo.gpumd.calc.**hnemd_spectral_decomp**(*dt*, *Nc*, *Fmax*, *Fe*, *T*, *A*, *Nc_conv=None*, *shc=None*, *directory=''*)

Computes the spectral decomposition from HNEMD between two groups of atoms.

Args:

dt (float): Sample period (in fs) of SHC method

Nc (int): Maximum number of correlation steps

Fmax (float): Maximum frequency (THz) to compute spectral decomposition to

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

T (float): HNEMD run temperature (in K)

A (float): Area (nm^2) that heat flows over

Nc_conv (int): Number of correlations steps to use for calculation

shc (dict): Dictionary from load_shc if already created

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

Returns:

out (dict): Dictionary with the spectral decomposition

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:

out (ndarray): Running average of kappa input

Data Loaders - data

thermo.gpumd.data.**load_dos**(*points_per_run*, *num_run=1*, *average=False*, *directory=''*, *filename='dos.out'*)

Loads data from dos.out GPUMD output file

Args:

points_per_run (int or list(int)): Number of frequency points the DOS is computed for. For num_run>1, a list can be provided to specify number of points in each run if they are different. Otherwise, it is assumed that the same number of points are used per run

num_run (int): Number of DOS runs in the dos.out file

average (bool): Averages all of the runs to a single output. Default is False. Only works if points_per_run is an int.

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

filename (str): File to load DOS from. Default is dos.out

Returns:

out (dict(dict)): Dictonary with DOS data. The outermost dictionary stores each individual run. Each run is a dictionary with keys:

- nu (THz)
- DOS_x (1/THz)
- DOS_y (1/THz)
- DOS_z (1/THz)

If average=True, this will also be stored as a run with the same run keys.

thermo.gpumd.data.**load_hac**(*directory=''*, *filename='hac.out'*)

Loads data from hac.out GPUMD output file which contains the heat-current autocorrelation and running thermal conductivity values

filename (str): File to load hac from. Default is hac.out

Created for GPUMD-v1.9

hacf - (ev^3/amu) k - (W/m/K) t - (ps)

out keys:

- hacf_xi
- hacf_xo
- hacf_x: ave. of i/o components
- hacf_yi
- hacf_yo
- hacf_y: ave of i/o components
- hacf_z
- k_xi
- k_xo
- k_x: ave of i/o components
- k_yi
- k_yo
- k_y: ave of i/o components
- k_z
- k_i: ave of x/y components
- k_o: ave of x/y components
- k: ave of all in-plane components
- t: correlation time

Args:

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

Returns:

out (dict): A dictionary with keys corresponding to the columns in 'hac.out' with some additional keys for aggregated values (see description)

thermo.gpumd.data.**load_kappa**(*directory=''*, *filename='kappa.out'*)

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

out keys:

- kx_in
- kx_out
- ky_in
- ky_out
- kz

Args:

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

filename (str): File to load kappa from. Default is kappa.out

Returns:

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

thermo.gpumd.data.**load_sdc**(*Nc*, *num_run=1*, *average=False*, *directory=''*, *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. For num_run>1, a list can be provided to specify number of points in each run if they are different. Otherwise, it is assumed that the same number of points are used per run

num_run (int): Number of SDC runs in the sdc.out file

average (bool): Averages all of the runs to a single output. Default is False. Only works if points_per_run is an int.

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

filename (str): File to load SDC from. Default is sdc.out

Returns:

sdc (dict(dict)): Dictionary with SDC data. The outermost dictionary stores each individual run. Each run is a dictionary with keys:

- t (ps)
- SDC_x (Angstrom^2/ps)
- SDC_y (Angstrom^2/ps)
- SDC_z (Angstrom^2/ps)

If average=True, this will also be stored as a run with the same run keys.

vac (dict(dict)): Dictonary with VAC data. The outermost dictionary stores each individual run. Each run is a dictionary with keys:

- t (ps)
- VAC_x (Angstrom^2/ps^2)
- VAC_y (Angstrom^2/ps^2)
- VAC_z (Angstrom^2/ps^2)

If average=True, this will also be stored as a run with the same run keys.

thermo.gpumd.data.**load_shc**(*Nc*, *directory=''*, *filename='shc.out'*)
Loads the data from shc.out GPUMD output file

Args:

Nc (int): Maximum number of correlation steps

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

filename (str): File to load SHC from. Default is shc.out

Returns:

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

thermo.gpumd.data.**load_vac**(*Nc*, *num_run=1*, *average=False*, *directory=''*, *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. For num_run>1, a list can be provided to specify number of points in each run if they are different. Otherwise, it is assumed that the same number of points are used per run

num_run (int): Number of VAC runs in the mvac.out file

average (bool): Averages all of the runs to a single output. Default is False. Only works if points_per_run is an int.

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

filename (str): File to load VAC from. Default is mvac.out

Returns:

out (dict(dict)): Dictonary with VAC data. The outermost dictionary stores each individual run. Each run is a dictionary with keys:

- t (ps)
- VAC_x (Angstrom^2/ps^2)
- VAC_y (Angstrom^2/ps^2)
- VAC_z (Angstrom^2/ps^2)

If average=True, this will also be stored as a run with the same run keys.

Input/Output - io

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', 'layer'). Default is None.

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

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.**convert_gpumd_trajectory**(*traj_file='xyz.out'*, *out_filename='out.xyz'*, *in_file='xyz.in'*, *format='xyz'*)

Converts GPUMD trajectory to any compatible ASE output. Default: xyz

Args:

traj_file (str): Trajetory from GPUMD

out_filename (str): File in which final trajectory should be saved

in_file (str): Original stucture input file to GPUMD. Needed to get atom numbers/types

format (str): ASE supported format

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:

traj (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:

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

Preprocessing - preproc

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:

counts (int) A list of number of atoms in each group.

thermo.gpumd.preproc.**add_group_by_type**(*atoms*, *groups*)

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:

counts (int) A list of number of atoms in each group.

thermo.gpumd.preproc.**assign_layer_by_position**(*split*, *atoms*, *direction*)

Assigns layers to all atoms based on its position. Only works in one direction. Similar to group but only one layer can be assigned to an atom. 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 assign layers to

direction (str): Which direction the split will work

Returns:

counts (int) A list of number of atoms in each layer

thermo.gpumd.preproc.**assign_layer_by_type**(*atoms*, *layers*)

Assigns a layer to all atoms based on atom types. Returns a bookkeeping parameter, but atoms will be udated in-place.

Args:

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

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

Returns:

counts (int) A list of number of atoms in each layer.

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]

1.1.2 lammps

Calculations - calc

thermo.lammps.calc.**autocorr**(*f*, *max_lag*)

Computes a fast autocorrelation function and returns up to max_lag

Args:

f (ndarray): Vector for autocorrelation

max_lag (float): Lag at which to calculate up to

Returns:

out (ndarray): Autocorrelation vector

thermo.lammps.calc.**get_GKTC**(***kwargs*)

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=tot_time

Keyword Arguments:

- **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]
- **log (string):** This is the path of the log file. This is only used if the *dt* keyword is not provided as it tries to extract the timestep from the logs
- **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 [ps]
- **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
- **srate (float):** This is related to rate, as it is the heat flux sampling rate in units of simulation time. This does not need to be provided if *rate* is already provided. Defaults are based on *rate* and *dt*. Units of [ns]
- **tau (int):** max lag time to integrate over. This is in units of [ps]

Args:

****kwargs (dict):** List of args above

Returns: out (dict):

- 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 (ndarra): time [ps]
- directory (str): directory of results
- log (str): name of log file
- dt (float): timestep [ps]
- tot_time (float): total simulated time [ps]
- tau (int): Lag time [ps]
- 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**(***kwargs*)

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

out keys:

Args: **kwargs (dict):

- **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: out (dict):

- Jx (list)
- Jy (list)
- Jz (list)
- rate (float)

Data Loaders - data

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:

dt (list(flaat)): The timesteps found in log_file in [ps]

thermo.lammps.data.**get_sim_dimensions**(*lammpstrj_file*)

Reads a LAMMPS trajectory file and extracts simulation dimensions.

Args:

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

Returns: out (dict): - x (list(float)): x-dimension [angstroms] - y (list(float)): y-dimension [angstroms] - z (list(float)): z-dimension [angstroms] - area (list(float)): [angstroms^2] - volume (list(float)): [anstroms^3]

Input/Output - io

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

1.1.3 shared

Force Comparison - force

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:

out (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:

out (dict) dictionary containing sorted force vectors

1.1.4 tools

Lennard Jones - lj

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 UFF parameters in the LJ object. 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:

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

CHAPTER 2

More Navigation:

- genindex

Python Module Index

t

Index

T