NE451
Simulation Methods

LAMMPs Tutorial 2: Introduction to calculating and plotting the radial distribution function with LAMMPs on the c1 cluster

Agnes Katai
University of Waterloo
Canada

Training Overview

Description:

These slides are designed for users new to LAMMPs, a C++ based classical molecular dynamics (MD) code. This is a step-by-step guide on how to calculate and plot the radial distribution function (RDF) of LiMnO2 using LAMMPs.

Topics Covered:

- Mathematical background and theory on RDF
- Defining simulation settings in the lammps input script
- Running lammps on c1 cluster using a Slurm job script

Logistics:

- **Type:** Learn AT your own pace
- **Audience:** Beginner-Intermediate
- **Prerequisite:** Review **Intro to HPC** and **LAMMPs Tutorial 1** training sessions

RDF - Theory

What is the radial distribution function (RDF)?

- The RDF describes how **the particle density varies with distance, r, from a reference particle.**
- To obtain the RDF, the **local particle density is normalized against the average (bulk) density**. The RDF can be thought of as the factor needed to multiply the bulk density, ρ, to obtain the local density, $\rho(r)$ about a reference atom. The RDF can also be referred to as $g(r)$.
  - g(r) = 1 $\rightarrow$ random distribution of particles; ideal gas
  - g(r) > 1 $\rightarrow$ more likely to find a particle
  - g(r) < 1 $\rightarrow$ less likely to find a particle
- Used to infer the **structure** and **phase** of a system. For example, **periodic structures (i.e., crystals) have sharp, repeating peaks** that appear at integer multiples of interatomic distance. **Amorphous materials and liquids lack long-range order** (i.e., g(r) approaches 1 at large distances).

RDF - Theory (Cont.)

How do you calculate the radial distribution function (RDF)?

$$g(r) = \frac{1}{4\pi r^2 \Delta r \, \rho N} \sum_{i=1}^{N} \sum_{j \neq i} \delta(r - r_{ij}) \qquad (1)$$

where dn_r is the average number of particles within a shell of infinitesimal distance, dr, dV_r is the spherical shell volume approximated for infinitesimal shell thickness, and ρ is the average (bulk) density of particles.

Hence, local density is related to bulk density as follows:

$$g(r) = \frac{\langle \rho(r) \rangle}{\rho} \qquad (2)$$

where $\langle \rho(r) \rangle$ is the average density of particles at a distance, r, from the reference particle (origin coordinate).

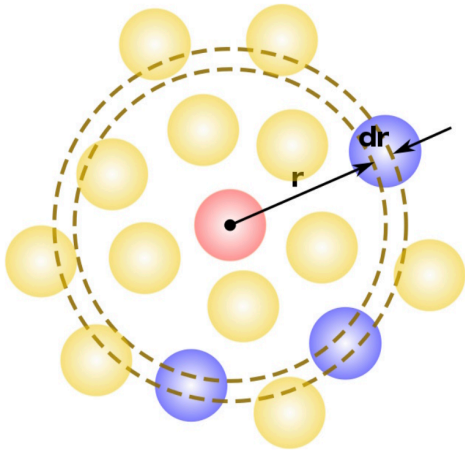


Figure: Schematic diagram for calculating the radial distribution function. The RDF describes the average number of particles found at a distance, r from a reference atom.

RDF - Theory (Cont.)

Additional notes about the RDF:

- Since dn_r is an **average**, you can average the local number of particles over several time frames to improve statistics

RDF - Algorithm (Cont.)

How does the RDF algorithm work?

Let's say we have a distribution of Li atoms within a system.

1. Select a central Li atom as the **"reference atom"**. This atom is designated as the origin coordinate ([0, 0, 0]).
2. Iterate over every other Li atom in the system (excluding the reference, of course) and calculate the **distance, *r*, with respect to the reference atom**.
3. Create a raw **histogram** of Li atom count vs distance from the reference atom, *r*. **Bin the calculated distances from Step 2. into the corresponding bin intervals that range from [r to r+dr]** (i.e., bin intervals may be 0.00 - 0.05 Å, 0.05 - 0.10 Å, 0.10 - 0.15 Å, etc.).

RDF - Algorithm (Cont.)

3. Now we must **normalize the number of Li atoms against the spherical shell volume,** dV_r. Naturally, the greater the distance from the reference atom, the larger the dV_r (note that the **spherical shell thickness,** dr**, does not change**). Therefore, the Li atom count needs to be divided by dV_r.
4. Update the histogram with the **normalized Li atom count in each RDF bin**. Recall that the bin width corresponds to the shell thickness, dr. The smaller the dr, the greater the number of bins in the histogram, and therefore, the higher the resolution.

RDF - LAMMPs Workflow

Workflow for calculating RDF using lammps:

1. Calculate the radial distribution function for each desired atom pair in the system (i.e., Li-Li, Li-Mn, Li-O, etc.).
2. Store the RDF data calculated at custom time intervals in a global array. Average the RDF over a custom number of frames and write the data to a file.
3. Open file in program of choice. Plot RDF.

The following guide will describe all lammps commands required to compute the RDF **that were not discussed in LAMMPs Tutorial 1**. This document presumes that you are already familiar with the basic lammps commands discussed in **LAMMPs Tutorial 1**.

RDF - LAMMPs Input Script


```
# --- Group Atoms ---
group              Li type 1
group              Mn type 2
group              O  type 3
```

- The `group` command allows the user to assign a custom name to an atom type. The user-assigned name for the atom group can then be used with other commands such as `fix`, `compute`, etc.

  Note: this command may be useful if your system consists of multiple elements.

RDF - LAMMPs Input Script (Cont.)

- Before setting a thermostat and/or barostat, it is necessary to define a timestep for the simulation and initialize the velocities of the atoms.

```
# ---------- Equilibration step  ------------
timestep         ${dt}
velocity         all create 670.0 123
```

Syntax: `velocity group-ID style args keyword value ...`

Description:

- The `velocity` command assigns velocities to atoms in a group in one of several styles. The `create` style is the most common, which generates an ensemble of velocities using a random number generator with a random seed. If you use the same seed, you should get the same random velocities each time. The assigned velocities must follow the specified distribution scaled to the desired temperature - by default, the distribution is `uniform`. However, you can request that the generated velocities adhere to a `gaussian` distribution instead.

  In this case, `all` atoms in the simulation are assigned a random velocity according to a `uniform` distribution (default) scaled to `670.0` K. The random seed is `123`.

Self-diffusivity - LAMMPs Input Script (Cont.)


```
# -------- Equilibration run  ---------
fix             integrate1 all npt temp 670.0 670.0 10.0 aniso 0.0 0.0 10.0

run             50000
unfix           integrate1
```

Syntax: `fix ID group-ID style_name keyword value ...`

Description:

- The `fix` command sets a thermostat or an operation that is applied to a group of atoms at each timestep or minimization step. In this case, a time integration is performed on Nose-Hoover style equations of motion to update positions and velocities for `all` atoms in the simulation box while maintaining constant temperature and pressure (`fix` style `npt`) via a Nose–Hoover thermostat and barostat. Thermostating and barostating is achieved by adding a dynamic variable to the particle velocities (thermostating) and the dimensions of the simulation box (barostating).

  When setting a thermostat, it is necessary to set the following parameters: the start and stop temperature in the temperature ramp (T_{start} and T_{stop}), as well as the characteristic time interval for rescaling particle velocities (T_{damp}). As a general rule, T_{damp} should be approximately 100 time steps. However, T_{damp} can be tuned accordingly for the system of interest - in this case, T_{damp} is 10.0.

  When setting a barostat, it is necessary to set the following parameters: the start and stop pressure in the pressure ramp (P_{start} and P_{stop}), as well as the characteristic timescale for rescaling the simulation box dimensions (P_{damp}). As a general rule, P_{damp} should be approximately 1000 time steps.

- The `run` command **runs or continues** the molecular dynamics simulation for a custom number of timesteps, *N*. The `run` command is critical - **without the `run` command, your simulation is static and there is no time integration.**
- The `unfix` command removes a fix that was previously defined.

  In this case, `unfix integrate1` stops the dynamics in the NPT ensemble. **It is generally necessary to remove a thermostat before you define a new thermostat (NVT, etc.)**.

RDF - LAMMPs Input Script (Cont.)


```
# --------- Define Compute ------------
compute           RDF all rdf 20 1 1 1 2 1 3
```

Syntax: `compute ID group-ID rdf Nbin itype1 jtype1 itype2 jtype2 ... keyword/value ...`

Description:

- The `compute rdf` command defines the computation for the radial distribution function, g(r), and the coordination number, coord(r), for all atoms in a group. For the `compute` style `rdf`, it is necessary to set the number of RDF bins (`Nbins`), as well as the atom pair(s), `i_type_N` and `j_type_N`, from 1 ... N_types, for which you wish to calculate the RDF. Specifically, i_type_N is the central atom and j_type_N is the distribution atom.

  In this case, the bin number is set to `20`. Recall that for a smoother curve (at the cost of computational resources), you can increase the bin number. The rdf is computed between atoms of type `1` and `1`, `1` and `2`, and `1` and `3`. Since Li maps to type 1, Mn to type 2 and O to type 3, it really means the RDF between Li-Li, Li-Mn and Li-O atoms.

RDF - LAMMPs Input Script (Cont.)

Important notes about this `compute`:

- The RDF is calculated using the **histogram** method discussed earlier in the slides by binning pairwise distances into `Nbins` from 0.0 to the cutoff force defined in the potential file. You can also set your own custom cutoff distance with `Rcut`.
- Once this computation is invoked, it produces a global array of size `Ncols` x `Nbins` where `Ncols` varies with number of atom pairs. You can extract the column of bin coordinates (center of the bins) by index 1 (i.e., `c_RDF[1]`). The next two successive indices, correspond to the g(r) and coord(r) (i.e., `c_RDF[2]`, `c_RDF[3]`, respectively). These values are stored separately for each atom pair, and the sequence repeats for every pair.

RDF - LAMMPs Input Script (Cont.)


```
# -------- Customize Output  ---------
thermo          100
dump myDump1 all custom 100 Dumpfiles/_*.dump id type element x y z vx vy vz fx fy fz
dump_modify myDump1 element Li Mn O
```

Syntax: `dump ID group-ID style N file attribute1 attribute2 ...`

Description:

- The `dump` command dictates what per-atom attributes are written to one or more dump file(s) once every N timesteps.

  In this case, a dump file named `_*.dump` is written to the `Dumpfiles/` directory every `100` timesteps. The dump files list the atom `id`, `type` and `element` name. The dump files also store the positions, velocities and forces associated with these atoms. All these dump files can be visualized with OVITO.

- The `dump_modify` command allows further customization to the dump files.

  In this case, the list of elements is mapped to the atom types from 1 to N_types. This allows for the atoms in the dump files to be labeled with their respective element names. That way, when you open the dump files with **OVITO**, the atoms will be colour-coded based on their element and labelled accordingly. Without this customization, all atoms may appear as C atoms which is not helpful.

RDF - LAMMPs Input Script (Cont.)


```
# ---------- Production step  ------------
fix              integrate2 all nvt temp 670.0 670.0 10.0
fix    1 all ave/time 10 10 100 c_RDF[*] file RDF.data mode vector
```

Syntax: `fix ID group-ID ave/time Nevery Nrepeat Nfreq value1 value2 ... keyword args ...`

Description:

- The `fix` style `nvt` sets a second thermostat for the production run during which data is stored and analyzed.

  The `fix` style `ave/time` averages over a set of inputs stored every custom number of timesteps. Specifically, the inputs are collected every N_{every} timesteps. The average is computed over these N_{repeat} values at every N_{freq} timesteps. The input used to compute the averages are stored in `c_RDF[*]` which is a global array for RDF data. The `file` keyword dictates that the data is written to the output file named `RDF.data`.

  For example, the following command: `fix 1 all ave/time 2 6 100 c_RDF[*]`, calculates averages the RDF values over the following frames: 90, 92, 94, 96, 98, 100, on timestep 100.

  In this case, the average is computed over the following frames: 10, 20, 30, 40, 50, 60, 70, 80, 90, 100, on timestep 100.

Step 1: Prepare RDF Calculation Input Files (Local Computer)

- Download the folder `rdf` from LEARN to your `Downloads` directory.
- Navigate to the folder:

```
cd ~/Downloads/lammps/rdf
```

- Inspect the structure file with ovito : Introduction to Ovito
  Inspect the structure file with nano

```
nano structure.lmp
```

- Check potential files (`library.meam`, `LiMnO.meam`) and input script (`in.meam`) before copying.

LAMMPS Input Script: Initialization

Setup

```
# ------------------------ INITIALIZATION ---------------------
units             metal
boundary          p p p
atom_style        atomic

# ---------- Read_data     ---------------------
read_data         structure.lmp
```

Explanation

- Use `metal` units (time in ps, energy in eV).
- Periodic boundary conditions in x, y, z.
- Atom style is `atomic`.
- Read the initial structure from `structure.lmp`.

LAMMPS Input Script: Groups and Force Field

Groups

```
# ----------------------- VARIABLES ---------------------
variable dt equal 0.001

# --------- Group Atoms -----------
group Li type 1
group Mn type 2
group O  type 3
```

Force Field

```
# ------------------------ FORCE FIELDS ----------------------
pair_style  meam
pair_coeff * * library.meam Li Mn O LiMnO.meam Li Mn O
```

Explanation

- Define timestep size (`0.001 ps`).
- Group atoms by type (Li, Mn, O).
- Use MEAM potential with parameter files.

LAMMPS Input Script: Equilibration

Commands

```
# ---------- EQUILIBRATION ------------
thermo          100
thermo_style    custom step temp pe ke etotal press lx ly lz

timestep        ${dt}
velocity        all create 670.0 123

fix integrate1 all npt temp 670.0 670.0 10.0 aniso 0.0 0.0 10.0

run             50000
unfix           integrate1
```

Explanation

- Print thermodynamic info every 100 steps.
- Initialize velocities at 670 K.
- Run 50,000 steps with NPT (constant pressure and temperature).

LAMMPS Input Script: Production + RDF

Commands

```
# ---------- PRODUCTION  ------------
compute RDF all rdf 20 1 1 1 2 1 3

thermo        100
thermo_style custom step temp pe ke etotal vol

dump myDump1 all custom 100 Dumpfiles/_*.dump id type element x y
dump_modify  myDump1 element Li Mn O

fix 1 all ave/time 10 10 100 c_RDF[*] file RDF.data mode vector

fix integrate2 all nvt temp 670.0 670.0 10.0
run 50000
unfix integrate2
```

Explanation

- Compute RDF with 20 bins (Li–Li, Li–Mn, Li–O).
- Save RDF data to `RDF.data`.
- Save trajectory dumps for visualization.
- Run production with NVT (constant volume and temperature).

Step 2: Copy RDF Folder to Cluster (Local Computer)

- From your local terminal, copy the folder to the cluster:

```
cd ~/Downloads/lammps/
scp -r rdf/ userid@c1.sci.uwaterloo.ca:~
```

- This transfers the entire `rdf` directory to your home folder on the cluster.

Step 3: Connect to the Cluster

- Start an SSH session:

```
ssh userid@c1.sci.uwaterloo.ca
```

- Navigate into the RDF directory:

```
cd rdf
```

RDF - Running lammps on c1 cluster

- Now we prepare our Slurm file for running lammps and we submit the job to the c1 cluster with the following command: `sbatch lammps_run.sh`
- We increase the number of cpus here for efficiency, since this calculation is more computationally expensive

```
#!/bin/bash
# this example is auto generated by the "guide" commar
#SBATCH --job-name=Diffusion     # Job Name
#SBATCH --time=0-01:00         # Max time (DD-HH:MM)
#SBATCH --cpus-per-task=16    # Number of CPUs
#SBATCH --mem=8G              # Memory per cpu
module load class-simulations

srun /scic/app/cpu/lammps/2025-04-02/bin/lmp < in.mean
```

- If you set up and ran lammps correctly, you should have all the dump files in `Dumpfiles/` folder as well as the `RDF.data` file in the current working directory.

Step 4: Run RDF Job on the Cluster

- Submit the job script to SLURM:

```
sbatch lammps_run.sh
```

- Monitor progress:

```
squeue -u userid
```

- SLURM output will be written to files such as:

```
slurm-84.out
slurm-172.out
```

Step 5: Inspect RDF Results on the Cluster

- Open the LAMMPS log file:

```
nano log.lammps
# or
less log.lammps
```

- The file `RDF.data` contains the calculated radial distribution function.
- You can quickly preview it with:

```
head RDF.data
```

Step 6: Copy RDF Results Back to Local Computer

- Download RDF output to your computer:

```
scp -r userid@c1.sci.uwaterloo.ca:~/rdf .
```

- Open `RDF.data` locally for analysis:

```
nano RDF.data
```

- open the .dump file with ovito

RDF - Data Post Processing

We can use various tools to plot our RDF from the RDF.data file. The data values are separated by a delimiter (space), so the data can be easily plotted with:

- Excel
- Python (or Jupyter lab)

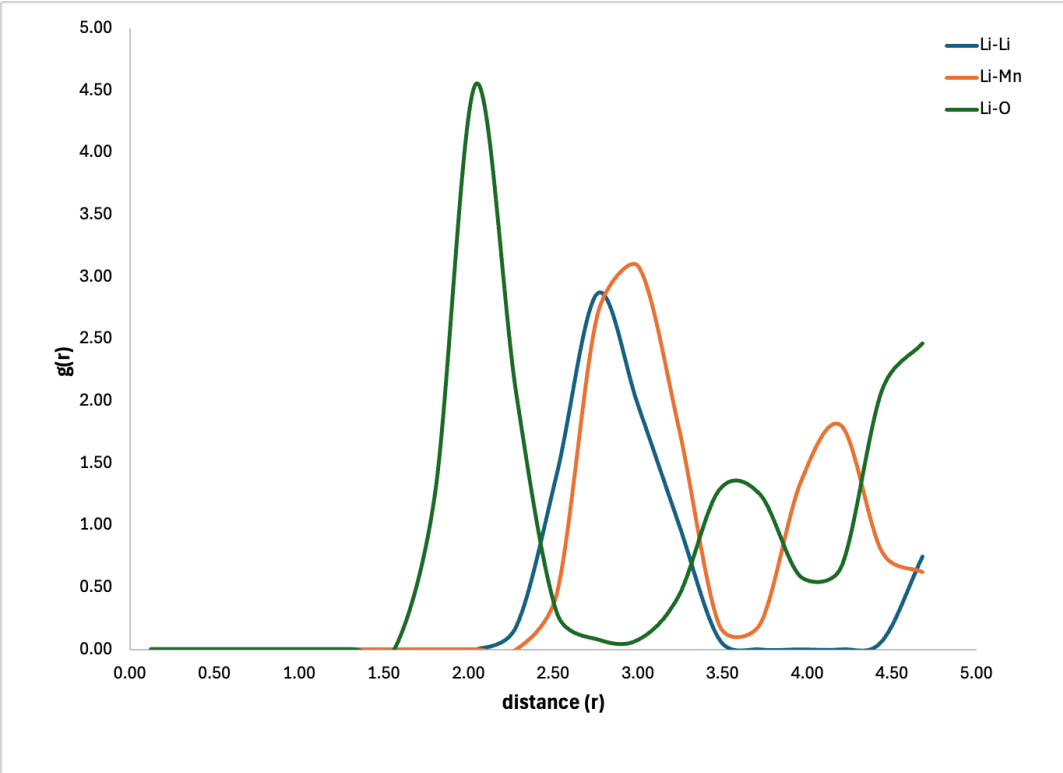
RDF - Data Post Processing (Cont.)

- Generally, you'd want to plot the RDF data stored in the **last frame**.
- The columns are labeled as follows:
  - Column 1 = RDF bin number. In this case, there should be 20 bins (or rows) per frame.
  - Column 2 = Bin coordinate (center of bin) $\rightarrow$ **x-value; remains the same for all pair types**
  - Column 3 = g(r) for Li-Li $\rightarrow$ **y-value**
  - Column 4 = coord(r) for Li-Li
  - Column 5 = g(r) for Li-Mn $\rightarrow$ **y-value**
  - Column 6 = coord(r) for Li-Mn
  - Column 7 = g(r) for Li-O $\rightarrow$ **y-value**
  - Column 8 = coord(r) for Li-O

  Recall that the two successive columns following the bin coordinate column repeat for each pair of atom types. Remember the element that corresponds to each atom type number.

RDF - Data Post Processing (Cont.)

- If you plotted the RDF correctly, you should have 3 data sets: Li-Li, Li-Mn and Li-O



Exercise: RDF Analysis

Tasks

- Increase RDF bins from 20 to 100
- Compare RDF at 300 K vs 670 K

Hints

```
# Open the LAMMPS input script
nano in.meam

# Look for the "compute rdf" command
# Example: compute myRDF all rdf 20
# Change it to:
compute myRDF all rdf 100

# To change temperature, edit fix nvt command:
fix 1 all nvt temp 300.0 300.0 100.0
# or
fix 1 all nvt temp 670.0 670.0 100.0
```