Linear Response: Theory and Computation Lecture 2: Calculating thermal conductivity using molecular dynamics simulation

Zheyong Fan

Aalto University

zheyong.fan@aalto.fi

12th, April, 2018

Overview

References

- J. M. Haile, *Molecular Dynamics Simulation: Elementary Methods*, (Wiley, 1992), **Chapter 7.** Good discussion on MD simulation of other transport properties (diffusion coefficient and viscosity).
- Mark E. Tuckerman, *Statistical Mechanics: Theory and Molecular Simulation*, (Oxford University Press, 2010), **Chapters 3, 4, 5.** Authoritative presentation of integration methods in MD with NVE, NVT, and NPT ensembles.
- Other reading materials can be found at https://www.dropbox.com/home/ASP-part-2-reading-materials

Computer codes

- A 100-line MD code written in MATLAB: https://github.com/brucefan1983/simple-md-matlab
- EMD method for thermal conductivity: https://github.com/brucefan1983/heat-conductivity-emd
- HNEMD method for thermal conductivity: https://github.com/brucefan1983/heat-conductivity-hnemd
- GPUMD – A super fast ($\sim 10^8$ atom $\times$ step/second) MD code with many-body potentials: https://github.com/brucefan1983/GPUMD
- If all of the above are not enough to you, try LAMMPS: http://lammps.sandia.gov/

Flow chart of simple (typical) MD simulation

- **initialize** the positions, velocities, forces and model parameters
- loop over the time steps
    - possibly update the **neighbor list**
    - fully update **positions** and partly update **velocities**
    - update **forces** and optionally related quantities
    - complete the **velocity** update
    - adjust the **temperature** and/or **pressure** according to the ensemble
    - **sample** data at a given frequency for later post-processing
- **post-processing** the saved data and **output** useful data
- **finalize**

Code fragments for initializing the velocities

```
double momentum_average[3] = {0.0, 0.0, 0.0};
for (int n = 0; n < N; ++n)
{
    vx[n] = -1.0 + (rand() * 2.0) / RAND_MAX;
    vy[n] = -1.0 + (rand() * 2.0) / RAND_MAX;
    vz[n] = -1.0 + (rand() * 2.0) / RAND_MAX;
    momentum_average[0] += m[n] * vx[n] / N;
    momentum_average[1] += m[n] * vy[n] / N;
    momentum_average[2] += m[n] * vz[n] / N;
}
for (int n = 0; n < N; ++n)
{
    vx[n] -= momentum_average[0] / m[n];
    vy[n] -= momentum_average[1] / m[n];
    vz[n] -= momentum_average[2] / m[n];
}
```

Code fragments for scaling the velocities

```
double temperature = 0.0;
for (int n = 0; n < N; ++n)
{
    double v2 = vx[n]*vx[n] + vy[n]*vy[n] + vz[n]*vz[n];
    temperature += m[n] * v2;
}
temperature /= 3.0 * K_B * N;
double scale_factor = sqrt(T_0 / temperature);
for (int n = 0; n < N; ++n)
{
    vx[n] *= scale_factor;
    vy[n] *= scale_factor;
    vz[n] *= scale_factor;
}
```

Velocity-Verlet algorithm in MD

From the lecture by Miguel, you have learned the velocity-Verlet integration method in MD:

$$\vec{v}_i(t+\Delta t) \approx \vec{v}_i(t) + \frac{\vec{F}_i(t) + \vec{F}_i(t+\Delta t)}{2m_i}\Delta t; \tag{1}$$

$$\vec{r}_i(t+\Delta t) \approx \vec{r}_i(t) + \vec{v}_i(t)\Delta t + \frac{1}{2}\frac{\vec{F}_i(t)}{m_i}(\Delta t)^2. \tag{2}$$

There, the algorithm was derived in a heuristic way. One can also derive the algorithm in a more formal way, starting from the following expression (L is the Liouville operator):

$$\begin{pmatrix} \vec{r}_i(t+\Delta t) \\ \vec{p}_i(t+\Delta t) \end{pmatrix} = e^{iL\Delta t} \begin{pmatrix} \vec{r}_i(t) \\ \vec{p}_i(t) \end{pmatrix}. \tag{3}$$

For details, study Sections 3.8.1, 3.8.2, and 3.10 of Tuckerman.

Code fragments for velocity-Verlet

```
double time_step_half = time_step * 0.5;
for (int n = 0; n < N; ++n)
{
    double mass_inv = 1.0 / m[n];
    vx[n] += (fx[n] * mass_inv) * time_step_half;
    vy[n] += (fy[n] * mass_inv) * time_step_half;
    vz[n] += (fz[n] * mass_inv) * time_step_half;
    if (flag == 1)
    {
        x[n] += vx[n] * time_step;
        y[n] += vy[n] * time_step;
        z[n] += vz[n] * time_step;
    }
}
```

Empirical potential

- Total potential energy U:

$$U = \sum_{i=1}^{N} U_i = \sum_{i=1}^{N} \frac{1}{2} \sum_{j \neq i} U_{ij}(r_{ij}). \tag{4}$$

- For the Lennard-Jones potential,

$$U_{ij} = 4\epsilon \left(\frac{\sigma^{12}}{r_{ij}^{12}} - \frac{\sigma^{6}}{r_{ij}^{6}} \right). \tag{5}$$

- Position difference:

$$\boxed{\vec{r}_{ij} \equiv \vec{r}_j - \vec{r}_i}. \tag{6}$$

- Minimum image convention:
  - if $x_{ij} > L_x/2$ then $x_{ij} = x_{ij} - L_x$
  - if $x_{ij} < -L_x/2$ then $x_{ij} = x_{ij} + L_x$

Code fragments for the minimum image convention

```
void apply_mic
(
    double lx, double ly, double lz,
    double lxh, double lyh, double lzh,
    double *x12, double *y12, double *z12
)
{
    if       (*x12 < - lxh) {*x12 += lx;}
    else if  (*x12 > + lxh) {*x12 -= lx;}
    if       (*y12 < - lyh) {*y12 += ly;}
    else if  (*y12 > + lyh) {*y12 -= ly;}
    if       (*z12 < - lzh) {*z12 += lz;}
    else if  (*z12 > + lzh) {*z12 -= lz;}
}
```

Force, stress, and heat current

- Force
$$\vec{F}_i = \sum_{j\neq i} \vec{F}_{ij}; \quad \vec{F}_{ij} = \frac{\partial U_{ij}(r_{ij})}{\partial r_{ij}} \frac{\vec{r}_{ij}}{r_{ij}} = -\vec{F}_{ji} \tag{7}$$

- Stress
$$\sigma^{\alpha\beta} = -\frac{1}{2V} \sum_i \sum_{j\neq i} r_{ij}^{\alpha} F_{ij}^{\beta} + \frac{N k_B T}{V} \delta^{\alpha\beta}. \tag{8}$$

- Heat current for two-body potentials (recall Exercise 2 from Lecture 1):
$$\vec{J} = \sum_i \vec{v}_i E_i - \frac{1}{2} \sum_i \sum_{j\neq i} \left(\vec{r}_{ij} \otimes \vec{F}_{ij} \right) \cdot \vec{v}_i \equiv \vec{J}^{\mathrm{kin}} + \vec{J}^{\mathrm{pot}}. \tag{9}$$

Code fragments for building the neighbor list

```
for (int n = 0; n < N; n++) {NN[n] = 0;}
for (int n1 = 0; n1 < N - 1; n1++)
{
    for (int n2 = n1 + 1; n2 < N; n2++)
    {
        double x12 = x[n2]-x[n1];
        double y12 = y[n2]-y[n1];
        double z12 = z[n2]-z[n1];
        apply_mic(lx,ly,lz,lxh,lyh,lzh,&x12,&y12,&z12);
        double distance_square = x12*x12 +y12*y12+z12*z12;
        if (distance_square < cutoff_square)
        {
            NL[n1 * MN + NN[n1]++] = n2;
        }
    }
}
```

Code fragments for calculating force

```
for (int i = 0; i < N - 1; ++i) {
    for (int k = 0; k < NN[i]; k++) {
        int j = NL[i * MN + k];
        double x_ij=x[j]-x[i]; double y_ij=y[j]-y[i];
        double z_ij=z[j]-z[i];
        apply_mic(lx,ly,lz,lxh,lyh,lzh,&x_ij,&y_ij,&z_ij);
        double r2 = x_ij*x_ij + y_ij*y_ij + z_ij*z_ij;
        if (r2 > cutoff_square) { continue; }
        double r4=r2*r2; double r8=r4*r4;
        double r14=r2*r4*r8;
        double f_ij = factor_1 / r8 - factor_2 / r14;
        fx[i] += f_ij * x_ij; fx[j] -= f_ij * x_ij;
        fy[i] += f_ij * y_ij; fy[j] -= f_ij * y_ij;
        fz[i] += f_ij * z_ij; fz[j] -= f_ij * z_ij;
    }
}
```

Green-Kubo relation for thermal conductivity

- Recall from the last lecture that Green-Kubo relations can be derived from linear response theory and empirical transport laws.
- Using Fourier's law, one can derive the following Green-Kubo relation for lattice (or phonon) thermal conductivity:

$$\boxed{\kappa_{\mu\nu}(t) = \frac{1}{k_B T^2 V} \int_0^t dt' \langle J_\mu(0) J_\nu(t') \rangle_e}. \tag{10}$$

  Here, k_B is Boltzmann's constant, V is the volume of the simulated system, T is the absolute temperature, $\langle J_\mu(0) J_\nu(t') \rangle_e$ is the heat current autocorrelation, and $J_\mu(0)$ and $J_\nu(t')$ are the total heat current of the system at two time points separated by an interval of t'.
- The symbol $\langle\rangle_e$ means equilibrium ensemble average, which will be substituted by averaging over different time origins.

Code fragments for calculating the heat current

```
for (int i = 0; i < N - 1; ++i) {
    for (int k = 0; k < NN[i]; k++) {
        int j = NL[i * MN + k];
        // some lines related to force calculations
        fx[i] += f_ij * x_ij; fx[j] -= f_ij * x_ij;
        fy[i] += f_ij * y_ij; fy[j] -= f_ij * y_ij;
        fz[i] += f_ij * z_ij; fz[j] -= f_ij * z_ij;
        double f_dot_v = x_ij * (vx[i] + vx[j])
                       + y_ij * (vy[i] + vy[j])
                       + z_ij * (vz[i] + vz[j]);
        f_dot_v *= f_ij * 0.5;
        hc[0] -= x_ij * f_dot_v;
        hc[1] -= y_ij * f_dot_v;
        hc[2] -= z_ij * f_dot_v;
    }
}
```

Code fragments for calculating the heat current autocorrelation function

```
// loop over the correlation time points
for (int nc = 0; nc < Nc; nc++)
{
    // loop over the time origins
    for (int m = 0; m < M; m++)
    {
        hac_x[nc] += hx[m] * hx[m + nc];
        hac_y[nc] += hy[m] * hy[m + nc];
        hac_z[nc] += hz[m] * hz[m + nc];
    }
    hac_x[nc] /= M;
    hac_y[nc] /= M;
    hac_z[nc] /= M;
}
```

Flow chart of the EMD method

- **Equilibration** stage: Equilibrate the system, controlling the temperature and optionally the pressure.
- **Production** stage: Change to the NVE ensemble, recording the heat current data.
- **Post-processing** stage: Calculate the thermal conductivity from the saved heat current data.

The HNEMD method – Thermal conductivity expression

- Recall from Lecture 1 that the nonequilibrium ensemble average of a general vector physical quantity $\vec{A}(\{\vec{r}_i, \vec{p}_i\})$ at time t after switching on the external driving force $\vec{F}_e$ can be written as:

$$\langle \vec{A}(t) \rangle_{ne} = \frac{1}{k_B T} \int_0^t dt' \langle \vec{A}(t') \otimes \vec{J}_d(0) \rangle_e \cdot \vec{F}_e. \qquad (11)$$

  Here, $\vec{J}_d$ is the **dissipative flux** defined by $\frac{d}{dt} H(\{\vec{r}_i, \vec{p}_i\}) = \vec{J}_d \cdot \vec{F}_e$.
- Setting both $\vec{A}$ and $\vec{J}_d$ in Eq. (11) to the heat current operator $\vec{J}$ gives

$$\langle \vec{J}(t) \rangle_{ne} = \frac{1}{k_B T} \int_0^t dt' \langle \vec{J}(t') \otimes \vec{J}(0) \rangle_e \cdot \vec{F}_e, \qquad (12)$$

- Comparing with the Green-Kubo relation, we have

$$\frac{\langle J_q^{\mu}(t) \rangle_{ne}}{TV} = \sum_{\nu} \kappa^{\mu\nu}(t) F_e^{\nu}. \qquad (13)$$

The HNEMD method – Equations of motion

- The equations of motion in linear response theory:

$$\frac{d}{dt}\vec{r}_i = \frac{\vec{p}_i}{m_i} + \mathbf{C}_i(\{\vec{r}_i, \vec{p}_i\}) \cdot \vec{F}_e; \tag{14}$$

$$\frac{d}{dt}\vec{p}_i = \vec{F}_i + \mathbf{D}_i(\{\vec{r}_i, \vec{p}_i\}) \cdot \vec{F}_e. \tag{15}$$

- Following Evans, we choose $\mathbf{C}_i(\{\vec{r}_i, \vec{p}_i\}) = 0$. Then

$$\frac{d}{dt}H = \sum_i \frac{\vec{p}_i}{m_i} \cdot \left(\vec{F}_i + \mathbf{D}_i \cdot \vec{F}_e\right) - \vec{F}_i \cdot \frac{\vec{p}_i}{m_i} = \sum_i \frac{\vec{p}_i}{m_i} \cdot \left(\mathbf{D}_i \cdot \vec{F}_e\right). \tag{16}$$

- Comparing this with $\frac{d}{dt}H(\{\vec{r}_i, \vec{p}_i\}) = \vec{J} \cdot \vec{F}_e$ we have

$$\mathbf{D}_i \cdot \vec{F}_e = E_i \vec{F}_e - \frac{1}{2} \sum_{j \neq i} \left(\vec{F}_{ij} \otimes \vec{r}_{ij}\right) \cdot \vec{F}_e. \tag{17}$$

Code fragments for force evaluation in the HNEMD method

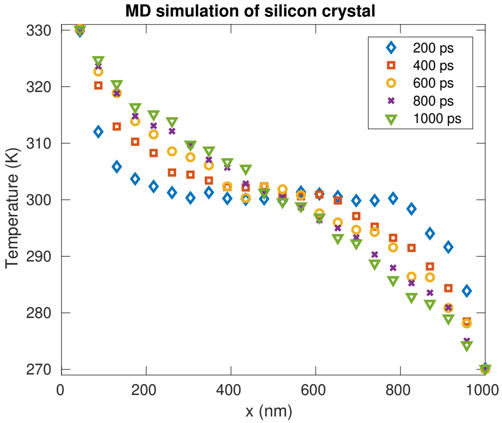
```
for (int i = 0; i < N - 1; ++i) {
    for (int k = 0; k < NN[i]; k++) {
        int j = NL[i * MN + k];
        // some lines related to force calculations
        double tmp = x_ij * Fe * 0.5;
        double internal = f_ij * x_ij;
        double external = internal * tmp; // driving force
        fx[i] += internal - external;
        fx[j] -= internal + external;
        // similar treatment for the y and z directions
    }
}
```

Flow chart of the HNEMD method

- **Equilibration** stage: Equilibrate the system, controlling the temperature and optionally the pressure.
- **Production** stage: Add the driving force, still controlling the temperature (otherwise the system will be heated up).
- **Post-processing** stage: Calculate the thermal conductivity from the saved heat current data.

NEMD method

A temperature gradient $|\nabla T|$ is generated using local thermostats or other methods. When steady-state is achieved, one calculates the thermal conductivity from $\kappa(L) = \frac{J}{A|\nabla T|}$.



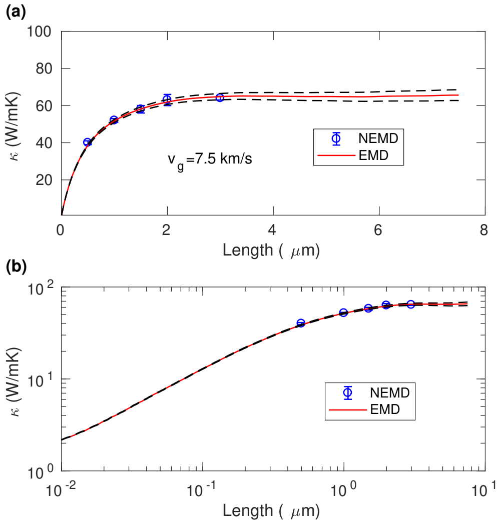
Flow chart of the NEMD method

- **Equilibration** stage: Equilibrate the system, controlling the temperature and optionally the pressure.
- **Production** stage: Generate the temperature gradient, measuring the temperature profile and the heat current.
- **Post-processing** stage: Calculate the thermal conductivity from the temperature gradient and the heat current.

Comparing the EMD and the NEMD methods

- The EMD method is a **homogeneous** method and has small finite-size effects. The thermal conductivity for an infinite-sized system can be obtained by using a relatively small simulation cell with periodic boundary conditions in the transport direction.
- The NEMD method is an **inhomogeneous** method and the simulation cell length has real meanings as in experiments. To get the thermal conductivity for an infinite-sized system, one usually needs to calculate the thermal conductivities of a few finite-sized systems and then extrapolate.
- The NEMD method can be used to study **thermal rectification**.
- The NEMD method can be used to calculate the **Kapitza thermal resistance**.

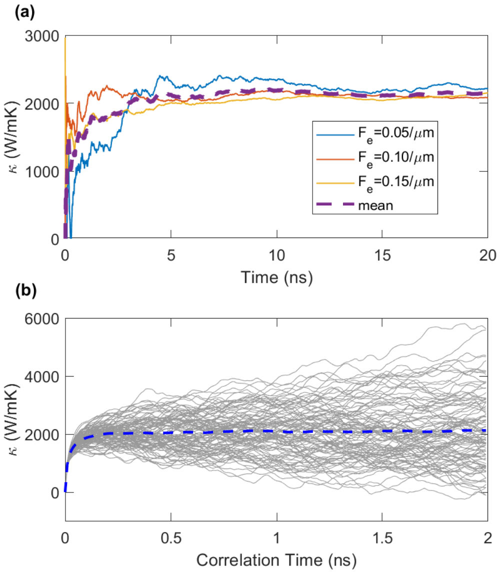
Case study: Silicon nanowire [Phys. Rev. B **97**, 094305 (2018)]



Comparing the EMD and the HNEMD methods

- Both are homogeneous method and therefore both have small finite-size effects.
- The HNEMD method is faster because it calculates the average of the heat current rather than the heat current autocorrelation function.
- The two methods give consistent results (see the next slide).
- The HNEMD method is rarely used because it is not easy to implement in an MD code, but this will be changed by the GPUMD code (on-going).

Case study: (10, 10)-Carbon nanotube [To be submitted]



Limitations and future directions of the MD-based methods for thermal conductivity calculations

- **Quantum effects** are not considered.
    - Quantify the quantum effects (on-going)
    - Develop an effective quantum correction method?
- **Empirical potentials** need to be improved
    - Better fitting methods (on-going)
    - Machine-learning potentials?
    - First-principles MD?

Project 1

- Using the codes provided in my github page (**heat-conductivity-emd** and **heat-conductivity-hnemd**) to study whether the EMD and the HNEMD methods give consistent results for Lennard-Jones argon from 20 K to 60 K. Compare your results to those in
  - A. J. H. McGaughey and M. Kaviany, "Thermal conductivity decomposition and analysis using molecular dynamics simulations. Part I. Lennard-Jones argon", International Journal of Heat and Mass Transfer **47**, 1783 (2004).
  - Z. Fan, T. Siro, and A. Harju, "Accelerated molecular dynamics force evaluation on graphics processing units for thermal conductivity calculations", Computer Physics Communications **184**, 1414 (2013).
- The lattice constants at 20, 30, 40, 50, 60 K are respectively 5.284, 5.305, 5.329, 5.356, 5.385 Å. Please make sure you use the correct lattice constant for each temperature, because the thermal conductivity is very sensitive to the pressure.
- Write a brief report (a two-page mini paper).