Phonon-SED

Phonon spectral energy density from LAMMPS simulations

Ty Sterling
ty.sterling@colorado.edu
Department of Physics
University of Colorado Boulder

This is a code to calculate the phonon spectral energy density [1] from molecular dynamics. It is a beta version as some features don't work perfectly yet. It is designed to work with LAMMPS, and as such, includes some sample input files and a discussion of how to interface it with LAMMPS. It will also work straightforwardly with other MD engines if you modify it.

While it's not necessary to cite anything, I would appreciate an acknowledgement if you use this for any production work. A simple 'Thanks to Ty Sterling at the University of Colorado' and a link to the code on github will be fine!

Also, you should cite the papers relevant to the actual SED method. In particular, you should cite ref. [1] and the relevant refs. therein (go and look them up!)

Table of Contents:

Disclaimer:

As with any codes used in science, I suggest you carefully check any results for validity. In particular, the relaxation times calculated with the Lorentzian fitting are not quantitatively accurate. There are 2 reasons for this:

1. I didn't take care to properly scale all the FFT's throughout the code, as I didn't expect to use the actual *numbers* for anything. Comparing relaxation times to one-another by looking at the fit was good enough. I planned to go back and do this later and I even left notes in the code about this so that it can be easily modified in the future. I may do this eventually in a future release or you can do it yourself. Nonetheless, the dispersion produced by the code is qualitatively correct (assuming your simulation is good), and the relaxation times produced by the fit are qualitatively correct; i.e. you can compare them to one another. Just don't expect the number it outputs to be e.g. picoseconds. I don't know what it is!
2. Fitting the peaks in a BZ where the bands are close together can be difficult. In particular, the code might accidentally identify two closely spaced peaks as one and fit them as a single peak, producing bad relaxation times. In the literature, the usual work around for this is to project the velocities onto the eigenvectors (calculated separately, e.g. with PHONOPY), so that only peaks corresponding to a particular mode is found. This is fine and I was working on implementing it. I didn't get it to work quite right in my code and as such, I won't discuss it in detail in this document. In fact, I have manually hardcoded this feature to be disabled in the pSED script. Anyway, the actual calculation is a separate method in the 'Phonons.py' module that can be worked on later if you (or I) ever want to. Another note is that, at high temperature, there is no reason to expect the strongly anharmonic eigenvectors to be the same as the harmonic ones, so the validity of this projection is restricted to rather low temperatures. See ref [2].

Additionally, the results produced by the SED code depend strongly on the MD simulation. I won't discuss this any further. Just note that the included MD input files are examples only!!! In no way are the results converged. Make sure you know how to do MD simulations before using the results for production!

Lastly, if any feature of the code doesn't work or seems wrong, feel free to let me know and I will fix it if I have time. Otherwise, I am glad to discuss the code with you if you are interested in modifying it yourself. It hasn't been extensively tested or used to produce any results published anywhere… so it needs thoroughly debugged! I wrote this to learn the SED method and learned a lot. I am glad that it works so well, but as with any code, it has bugs! If you do make modification or improvements, I'd be glad to give you push access to the directory so that we can continue to improve the code.

The code is presented as is, with no warranty what-so-ever, nor is there any claim to the validity of the output or quality of results produced used with this code. Use at your own risk!

Phonon-SED

Phonon spectral energy density from LAMMPS simulations

Introduction:

The code calculates the following (see ref [1] and refs therein):

$$\Phi(\boldsymbol{q},\omega) = \frac{1}{4\pi\tau_0 N_T} \sum_{\alpha}^{\{x,y,z\}} \sum_{b}^{B} m_b \left| \int_0^{\tau_0} \sum_{n}^{N_T} \dot{u}_\alpha(n,b;t) \times \exp(\mathrm{i}\mathbf{q}\cdot\mathbf{r}(\mathrm{n},0;\mathrm{t}) - \mathrm{i}\omega\mathrm{t})\, dt \right|^2$$

In the above, **q** is a phonon wavevector, ω is the angular frequency, τ_0 is the simulation time, N_T is the number of unit cells in the crystal, α is the cartesian direction, b is the atom label in a given unit cell, m_b is the mass of atom b in the unit cell, $n_{\{x,y,z\}}$ is a unit cell, $\dot{u}_\alpha(n,b;t)$ is the velocity of atom b in unit cell n at time t in the direction α, and $\boldsymbol{r}(n,0;t)$ is the equilibrium position of unit cell n. In some words: the above formula calculates the normal mode velocity as a discrete Fourier transform of the atomic velocities. The normal mode velocity is then Fourier transformed to get its spectrum in the frequency domain. The phonon spectral energy density is seen as the intensity of the kinetic energy distributed in **q** and ω space. This is basically the dispersion weighted by the kinetic energy carried by the phonon. See ref [1] and refs. therein for the details! The line shapes of phonons at particular **q** points can be fit to determine the phonon relaxation times. My code does this fitting, but as discussed in the disclaimer, use this with caution. The data are qualitatively correct and can be compared, but their absolute value is dubious (because I am lazy). The theory of the relaxation times from the line shape is discussed in [1] as well. See the refs. in that paper for more info.

The spectral energy density code (pSED) takes as input:

- The velocities and positions of the atoms as a function of time. This is a file automatically output by LAMMPS. It is preprocessed into an hdf5 database (I think LAMMPS can write out hdf5 automatically, but I didn't mess with that) for easy reading by pSED. You don't need to know about the format of the velocity or position files or hdf5 database unless you want to modify the code. You can figure all this out pretty easily by looking at the LAMMPS output and my code.
- A file containing the unit cell label, mass, etc. of all the atoms. All the things in the above equation that are needed. I will discuss the format of this input file later. It is called 'lattice.dat

Overview of the code:

Now is probably a good time for an overview of the code. There are 3 main steps:

1. Perform an MD simulation to produce velocities.
2. Compute the SED from the velocities
3. Plot or fit the SED results

- The MD simulation can be performed with any package, as long as you modify the velocity file or my code to read the output appropriately. I designed it to work with LAMMPS and won't discuss this anymore. Run a simulation and produce the velocities **and** positions at the same time step spacing. Note that the max frequency you can calculate is determined by the time step in the MD simulation and the resolution of the frequency is determined by how long you run it. Do some calculations for appropriate sampling before running the simulation!
- For whatever structure you are simulating, you need to tell my code the masses of the atoms, which unit cell they belong to, and which position in that unit cell they are in. This is a file named 'lattice.dat' If you use my structure_maker_API, this is produced for you. Otherwise, you need to prepare a file as follows (for e.g. a 2 atom unitcell of carbon):

1	1	1	12.0
2	1	2	12.0
3	2	1	12.0
4	2	2	12.0
5	3	1	12.0
6	3	2	12.0
…	…	…	…
Natoms	Unitcell label	Basis index	Mass

- With 'lattice.dat' and the velocities and positions in hand, you are ready to use the pSED code. First, compress the velocities and positions into and hdf5 file. You only have to do this once, then can read it for all future pSED calculations from that simulation. See the input section for more details.
- Once the velocities are compressed, you have to enter a q-path and number of q-points in the input file ('INPUT'). You can enter several at once, but keep in mind the number of q-points that you actually will get results for depends on the number of unit cells in the simulation (i.e. Born-Von Karmon boundary conditions quantize the wave vectors!).
- Then you can run the code.
- The code will output a bunch of files containing the SED.
- You can use the same input file to plot the SED across the segment of the BZ you calculated, or at a particular q-point.
- For a particular q-point, you can also tell the code to fit the peaks. Just tell the code the q-point, how many peaks there are, and a guess of the location of the peak. You can guess the locations by first plotting the q-point and then writing down the positions of the peaks, then entering them in the input file. (note that there are no dimensions on the x-axis: just give it the number of the data point that you see.)
- This will write out relaxation times and a goodness of fit.

With some practice, this will all make more sense! Just play with it. Once you have the SED results computed, fitting peaks and plotting is easy and can be done many times.

Setup

You can get the code from https://github.com/tyst3273/phonon-sed, either by cloning the repo or downloading it as a zip file. You then need to 'setup' the code. Some discussion is in order:

- I tried to write this code 'object oriented' so that it is easy to modify. This was my first experience with object oriented programming (OOP) so I didn't adhere to best practices. Parts of the code are a bit messy, but overall, it is modular and can easily be adapted/modified.
- Anyway, setting it up is as simple as copying all the modules in the './modules' directory somewhere, or leaving them where they are, and specifying the path to the directory at the very beginning of the pSED script. If you use the 'structure_maker.py' file in the './structure_maker_API' directory to make structures, which I recommend since it will produce the LAMMPS and 'lattice.dat' input you need, then you should add the path to the modules there as well.
- TLDR: copy the .py files from './modules' somewhere, then open pSED and change the modulepath variable to be the path to the directory containing all the modules. You should also open './structure_maker_API/structure_maker.py' and change the variable there too.

NOTE: you should also have numpy, scipy, matplotlib, and hdf5, and yaml in your python installation. I recommend using Anaconda to get these if you don't have them.

LAMMPS simulations:

I have included a basic LAMMPS input file in the './lammps' directory. It is assumed that you have a working LAMMPS installation and know how to use it. NOTE: this file will NOT produce converged or even good results. This is merely an example! You will need to create an input file appropriate for the crystal you are working on!

Also consider the crystal input file to LAMMPS. You have to provide this and it has to correspond to the 'lattice.dat' file described in the introduction. To that end, I recommend using my 'structure_maker.py' file. It will make the LAMMPS input file and 'lattice.dat' file in 1 go and you don't have to do anything else. I won't document its use here: you can find it in './structure_maker_API' and its input is self explanatory.

Take a look at the input file now, in particular the variables 'timestep' and 'dt_dump', as well as the command run. Also note that velocities and positions are recorded at the same timesteps.

'timestep' is the timestep used to integrate Newton's equations. This will determine the validity of your simulations: if it is too large, your trajectories will be wrong and energy won't be conserved.

'dt_dump' is the frequency that data are printed. This is included since, very often, you need a very small timestep to get good trajectories, but not so much sampling is needed to get good SED results.

Note that 'timestep'*'dt_dump' is the effective timestep of how often velocities are recorded. The maximum frequency you can calculate in the SED is determined by this product. Think about this and make sure you understand it! (Hint: frequency is proportional to 1/time…)

run' is the number of timesteps. This will give you 'run'/'dt_dump' data points. This can still give you quite a lot of data, so be careful. Note that this variable determines the frequency resolution (think about it!) and, if you are fitting for lifetimes, the simulation time should be several times larger than the phonon lifetime (see ref [1] and refs therein).

Processing the LAMMPS data:

Once you have performed the LAMMPS simulation and you have the velocity and positions files, it’s time to use pSED. So now I will walk you through the input file.

It is hardcoded to be named ‘INPUT’, but you can name it anything you want and give the name as an argument to pSED (e.g. ./pSED your_file).

Within INPUT, there are a number of variables. Some of them can be changed to change the behavior, some are related to the simulation and shouldn’t be changed except for other simulation. First, I will talk about the ones that shouldn’t be changed on the fly. You should set these accordingly before using pSED!

- NUM_ATOMS: (integer) the number of atoms in your crystal. You can get this from the LAMMPS crystal input file.
- NUM_STEPS: (integer) the number of steps in your simulation. (i.e. ‘run’, see the LAMMPS section of this doc.)
- TIME_STEP: (real) the MD timestep, see the LAMMPS section. It should match ‘timestep’ in the LAMMPS input file. Note that my units are fs, where as LAMMPS uses ps (at least my input file does!).
- STRIDE: (integer) The sampling frequency in the simulation. See the LAMMPS sec. This is the ‘dt_dump’ variable.
- NUM_SPLITS: (integer) this is used to block average your trajectory for better statistics. E.g. if you set it to 4, it will divide your velocity file into 4 chunks, calculate the SED separately for each, then average them together.
- LAT_PARMAMS: (3 x real): these are the lattice constants in Angstrom
- PRIM_VECS (9 x real): these are the primitive lattice vectors. It should be 9 real numbers according to: b_{1x} b_{1y} b_{1z} b_{2x} b_{2y} b_{2z} b_{3x} b_{3y} b_{3z}. where e.g. b_{1x} is the x-component of the the 1st lattice vector (denoted by b)
- OUT_PREFIX: (string) just a prefix to be added to all the output files
- LATTICE_FILE (string) should just be lattice.dat
- VELS_FILE: (string) name of the LAMMPS velocity file
- POS_FILE: (string) name of the LAMMPS pos file
- FILE_FORMAT: (string): just leave it as LAMMPS unless you know what you are doing

All these should be set before doing anything with the pSED code. I recommend setting all this up once (using the provided file as a template) and just copying it over whenever you need a new one. That way, you only have to tweak a few parameters.

Once you have your velocities, you should compress them. You do this using the same input file and just changing the following option:

- COMPRESS: (bool 0/1). Running pSED with this set to 1 will cause the code to open the VELS_FILE and POS_FILE files and save them as a hdf5 data base. This actually takes a while, and the code will warn you of this! You have to do this before you can calculate the SED. Once it finishes, the code will exit, so you should set this back to 0 and run ./pSED again to continue.

Once you are done compressing, you should enter the q-points you want to calculate on. This is done thanks to:

- NUM_QPATHS: (integer) the number of q-paths to calculate on. Can be 1 or larger. e.g. if going from 0.0 0.0 0.0 to 0.5 0.0 0.0 then to 0.5 0.5 0.0, this should be set to 2.
- NUM_QPOINTS: (integer) the number of q-points to subdivide the path into. See the introduction as this should match the number of unit cells in your supercell (Born-Von Karmon boundary conditions!)
- QSYM_POINTS: (6*NUM_QPATHS real numbers) this should be an array of real numbers: there should be 3 specifying the beginning and endpoints of each q-path. e.g. for NUM_QPATHS=1, this could be 0.0 0.0 0.0 0.5 0.0 0.0 to go from Γ to X.

You should also make sure num splits is what you want it set to. Alternatively, you can run in debug mode just to get a quick survey of the output using DEBUG:

- DEBUG: (bool, 0/1) this will force the code to skip calculating the SED on any other chunks determined by NUM_SPLITS, only on the first one. e.g. if you want to do a quick test of your results, set NUM_SPLITS to something large and DEBUG = 1 so that only a small chunk of data is used.

Set it how you want and run ./pSED now. It will take a while depending on the size of your velocity data files! The code will print some status updates and tell you when it finishes.

Once the code is done, it will write some output files to the work directory with the extensions .pSED, .Qpts, and .THz. The prefix will be whatever you set OUT_PREFIX to. These contain the spectral-energy-density, the q-point list, and the list of frequencies in the .pSED file. The .Qpts and .THz files are self explanatory, but will tell you about .pSED. It contains an NxM matrix of numbers corresponding to the N frequencies in .THz and the M q-points in .Qpts. That's it. If you want to increase precision or change the format of this file, you can easily edit FileIO.py in the modules directory. It uses numpy's fancy savetxt() method and is easy to change.

Now you can plot and post process the results. If you want to plot the dispersion, just use:

- PLOT_BANDS (bool, 0/1): this will read the .pSED file and plot the dispersion using matplotlib's imshow() module. Look in Plot.py if you want to change this behavior. Note that this will cause the code to create the plot and then exit. If the .pSED file isn't available, it will complain!

If you want to plot the pSED at a particular q-slice, use:

- PLOT_SLICE: (bool, 0/1) this will cause the code to plot the SED only at a single q-point in a traditional plot instead of a heatmap (e.g. in PLOT_BANDS). Note that this will cause the code to create the plot and then exit. If the .pSED file isn't available, it will complain!
- Q_SLICE_INDEX: (integer in range 0, NUM_QPOINTS-1) this specifies which q-point to plot. The index corresponds to the the q-points in the .Qpts file.

If you want to fit peaks, use:

- LORENTZ (bool, 0/1): set to 1 to fit a Lorentz function to a peak. You have to specify which q-point, how many peaks to fit, and a guess for each peak location. Specify the q-point with Q_SLICE_INDEX, how many peaks with NUM_GUESSES, and guesses for the peaks with PEAK_GUESSES. Note that this will cause the code to create the plot and then exit. If the .pSED file isn't available, it will complain! Also, PLOT_SLICE overrides this, so make sure that is set to 0 before using this.
- NUM_GUESSES (integer, greater than 0): the number of peaks differ peaks to attempt to fit a Lorentz function to.
- PEAK_GUESSES: (integers, NUM_GUESSES of them): these should be indices for the peak guesses in NUM_GUESSES. You can look these up by plotting a q-slice of the SED thanks to PLOT_SLICE. Just write down the approximate center for all the peaks by looking at the number on the bottom of the plot, then give them to PEAK_GUESSES… I don't remember if the order matters.

How you should use these is by first creating a plot using PLOT_SLICE, looking at the peaks you want to plot (write down their index as seen on the x-axis!), then telling LORENTZ about these peaks using NUM_GUESSES and PEAK_GUESSES. This will produce some output files with the extensions .error and .params.

The columns in the params file are the centers (indices: you might have to convert these to frequency by looking at the .THz file!), the peak height, and the half-width-half max (HWHM). The HWHM can be interpreted as being proportional to the relaxation time (see ref. [1]). I say 'proportional to' because the units aren't fixed and the scaling is off. Still, the relative peak widths tell you about the relative scattering rates. If you need the times in a particular unit system to e.g. calculate thermal conductivity, you will gave to modify the code a little (particularly the subroutines in Phonons.py to fix the scaling and convert the LAMMPS velocities to the units system you want!). The .errors file contains the respective variance of the fit for the parameters in .params. The files have 1 row per peak fit in the order the peaks were given. The prefix is OUT_PREFIX plus '_LORENZT-Q_SLICE_INDEX'

Example:

I will go over a quick example for a small, 512 atom supercell of silicon. I have provided the LAMMPS input file and a Tersoff potential file. You should do the intermediate steps, e.g. run my structure_maker_API, run the LAMMPS simulations, compress the data etc. I will go over these steps.

Note about space-time: on my 16 core desktop, the LAMMPS simulation takes about 8 minutes and produces 2 ~1.5 GB files (the pos.dat and vels.dat files). The compression takes ~5 minutes and produces a smaller 1 GB file with the pos.dat and vels.dat files combined. You could probably delete those files now if you want, but you may want to save them as backups for real calculations. Calculating the SED on 8 q-points across 4 splits takes ~1 minute using my code. For larger systems, it takes a lot longer, but still isn't too bad. Plotting takes only a few seconds.

Unzip or clone the files from my github if you haven't already. Also make sure you have a working LAMMPS installation (you can easily get it with 'sudo apt install lammps' on Ubunut) and all the python dependencies (numpy, scipy, matplotlib, hdf5, and yaml). Now go into the 'pSED' directory.

A quick review of the 'Processing LAMMPS data' section is in order before continuing: The overview of the procedure is to first perform an MD simulation to get the trajectory, then compress the output files by modifying the INPUT file and using the pSED code. Then modify the INPUT file again and use the pSED code to compute the SED from the MD output. Then you can plot the bands and fit peaks by further modifying the INPUT file and running the pSED code.

Here are the step-by-step instructions for the example:

1. Creating the structure and lattice.dat files
    a. Go into the 'structure_maker_API' directory.
    b. Look at the 'structure_maker.py' file: it's pretty easy to see what it is doing.
    c. Run it ('python structure_maker.py').
    d. You should now have 2 new files: 'lattice.dat', and 'si.lammps'.
    e. Move 'lattice.dat' into your work directory. It should be in the same directory as the velocity and positions files as well as the 'INPUT' file for pSED.
    f. Move 'si.lammps' to the 'lammps' directory. This is the input file for LAMMPS.

2. Running the LAMMPS simulation
    a. Go to the 'lammps' directory. You should have 'in.vels', 'SiCGe.tersoff', and 'si.lammps' in this directory.
    b. You can open 'in.vels' and look at it now if you want. Otherwise, run the LAMMPS simulation. For me, this took 8 minutes on 16 cores.

c. If all goes well, you should have ‘log.lammps’, ‘pos.dat’, and ‘vels.dat’ in the directory now. Move ‘pos.dat’ and ‘vels.dat’ to your work directory (the one with ‘lattice.dat’ in it.)
d. Now go to the work directory.

3. Compressing the data
    a. Copy the ‘pSED’ executable and the ‘INPUT’ file into the work directory. You should have ‘lattice.dat’, ‘pos.dat’, ‘vels.dat’, ‘pSED’, and ‘INPUT’ in your directory.
    b. Open ‘INPUT’ (the pSED input file)
    c. Make sure PLOT_BANDS, PLOT_SLICE, and LORENTZ are all 0. If not, these routines will look for the appropriate pSED output files and complain if they don’t find them (you haven’t produced them yet!)
    d. You should also make sure all the other parameters are set properly for your simulation. See the ‘Processing LAMMPS data’ and ‘Dictionary’ sections for more info. They are set properly for this example, but you should check if you use this code for your own structures.
    e. Now set COMPRESS = 1
    f. Save the ‘INPUT’ file and close it.
    g. Run the ‘pSED’ executable. On linux using bash, this is as simple as ‘./pSED’
    h. It should tell you it is compressing the data. This only takes a few minutes for these small files, but for larger crystals and longer trajectories, it can take a while!
    i. If all goes well, the code should tell you it was successful. You should now have ‘dat.hdf5’ in your directory.
    j. Now open ‘INPUT’ again and set COMPRESS = 0. Save it and close it. If you don’t do this before running ‘pSED’ again, the code will delete ‘dat.hdf5’ and start compressing the data again!
    k. The ‘pSED’ code won’t need the ‘pos.dat’ and ‘vels.dat’ anymore, so you can put them somewhere else if you want.

4. Computing the SED
    a. Now you can compute the SED.
    b. The given ‘INPUT’ file is setup to calculate it on 8 q-points on the path from Γ to X in the conventional 8-atom silicon first BZ. There are 16 unitcells along the x-direction in the supercell, so this is a good choice.
    c. Also note that NUM_SPLITS = 2. The code will divide the data into 2 chunks, compute the SED separately for each, then average them.
    d. You can now run the code (‘./pSeD’).
    e. It will tell you the q-points you are using, then will start on the first split. It prints status updates along the way so you know how long it will take.
    f. Once it is done, it will print ‘ALL DONE’ and exit.
    g. You should now have ‘si-example.pSED’, ‘si-example.Qpts’, and ‘si-example.THz’ in your directory. See the ‘Processing LAMMPS data’ section for more info about these files.

5. Plotting the band-structure
    a. Now open 'INPUT' once more.
    b. Set PLOT_BANDS = 1, save and close it.
    c. Now run pSED ('./pSED').
    d. It should only take a second. If everything works properly, you should see a rough looking silicon dispersion (fig.1).
    e. If you are done plotting the bands, open 'INPUT' and set PLOT_BANDS = 0, or the code will keep plotting it and exit whenever you run 'pSED'.

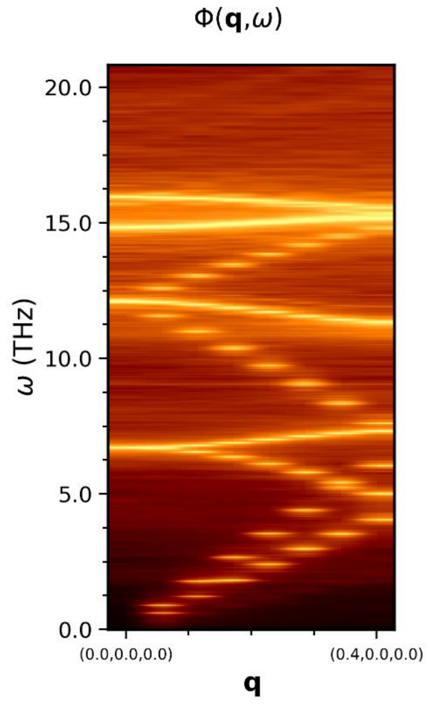

Fig: 1 Phonon Spectral Energy Density

This is the phonon spectral energy density computed in the example, step 5.

6. Plotting a q-slice
    a. To plot a particular q-slice, open 'INPUT' again.
    b. Set PLOT_SLICE = 1.
    c. In this example, I have set Q_SLICE_INDEX = 2. This is the 3rd q-point in the list (python indices start at 0 …)
    d. You can pick others, but the settings for LORENTZ are already appropriate for this one, so don't change it right now,
    e. Save 'INPUT' and close it.
    f. Now run pSED ('./pSED')
    g. You should see a log-scale plot of the SED at **q**=(0.125,0.0,0.0) in reduced coordinates (fig.2)
    h. Now is a good time to look for which peaks to fit. You can zoom in to get an accurate guess of the peak position. However, I have already set these for you in 'INPUT' for this example. I picked the first 2 peaks. You can zoom in and see

that their centers are roughly at 956 and1400. I suggest finishing the example, then coming back and plotting different peaks later.

i. If you are done plotting q-slices, open 'INPUT' and set PLOT_SLICE = 0, or the code will keep plotting and exiting when you run 'pSED'.

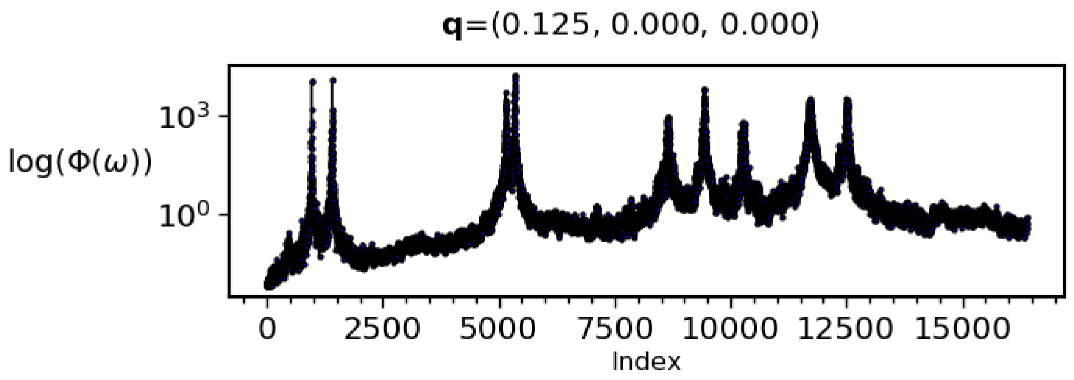

Fig: 2 A Q-slice of the Phonon Spectral Energy Density

This is the phonon spectral energy density plotted at q-index 2 as in step 6.

7. Fitting Lorentz functions to the peaks
   a. To fit Lorentz functions to the peaks, open 'INPUT' again and set LORENTZ = 1.
   b. You can look at NUM_GUESSES and see that I have picked 2: the first 2 peaks in the q-slice plot at **q**=(0.125,0.0,0.0).
   c. You can also look at PEAK_GUESSES and see that I have set them to 956 and 1400. You might have checked these numbers when you plotted the q-slice.
   d. Now save and close 'INPUT'
   e. Run the pSED code ('./pSED')
   f. You should see the same plot as fig.2 but with peaks fit to it. (fig. 3 is an example with the peaks included). The two red lines are the Lorentz functions fit to the peaks. The blue line is the superposition of Lorentz functions. If you fit **all** the peaks correctly, the blue line should be close to the SED line calculated from MD.
   g. You should also have some output files in the directory name 'si-example_LORENTZ-2.params' and 'si-example_LORENTZ-2.error'. These contain the fit parameters and the 2 in the name is the q-slice index you chose.
   h. You can find out what's in these files by reading the 'Processing LAMMPS data' section.

i. You can fit other peaks or other q-points by repeating steps 6 and 7. This is quick and easy and should all make sense with a little practice.

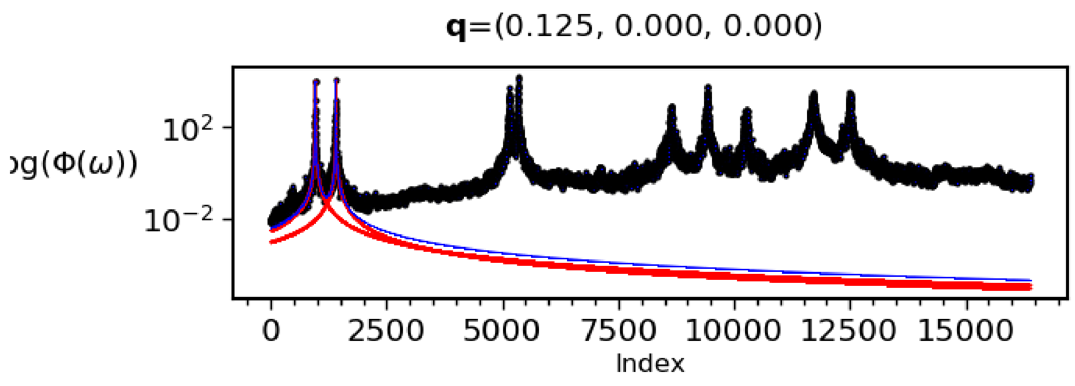

Fig: 3 Lorentz fit to the Q-slice of the Phonon Spectral Energy Density

This is the phonon spectral energy density plotted at q-index 2 as in step 6, with the peaks fit as done in step 7

You should now be able to use the code! You may have to tweak parameters and play with the code to get it to work for other systems, but this should be straightforward. Just keep in mind to always check convergence of your calculations and carefully consider the validity of all results before making any claims about the physics! Anyway, (see the disclaimer), the code is presented as is and I don't offer any guarantees about the quality of the results!

That being said, fig.4 and fig.5 are examples of the SED calculated for a larger silicon super cell at numerous temperatures. The bands show obvious broadening and softening of phonons as temperature increases. This is pretty cool! 😊

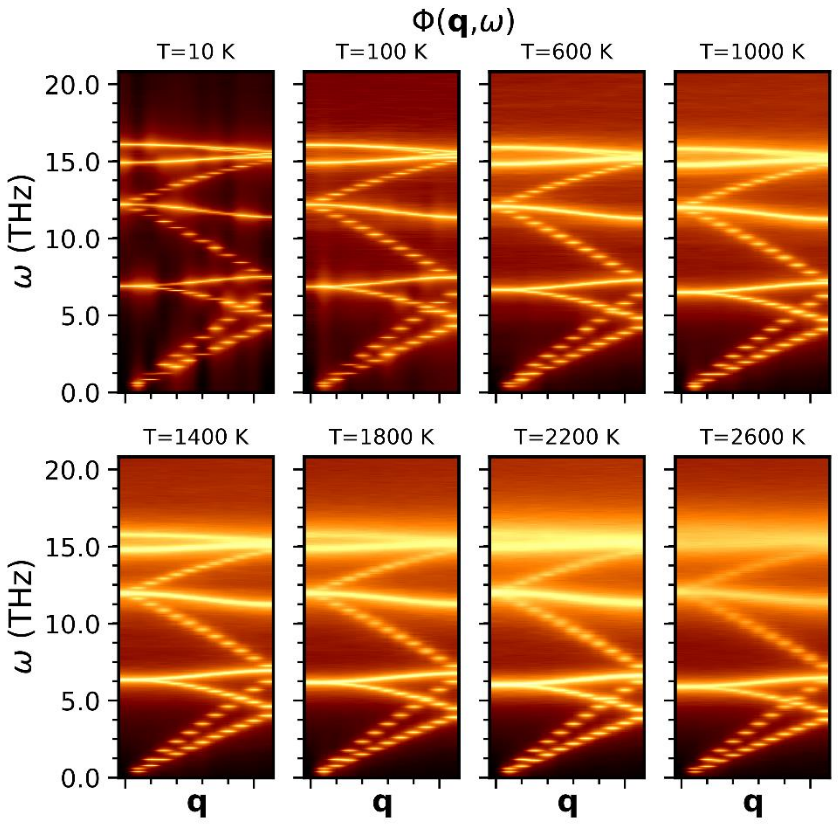

Fig: 4 Temperature dependent spectral energy density

This is the phonon spectral energy density of an 8-atom silicon supercell at different temperatures.

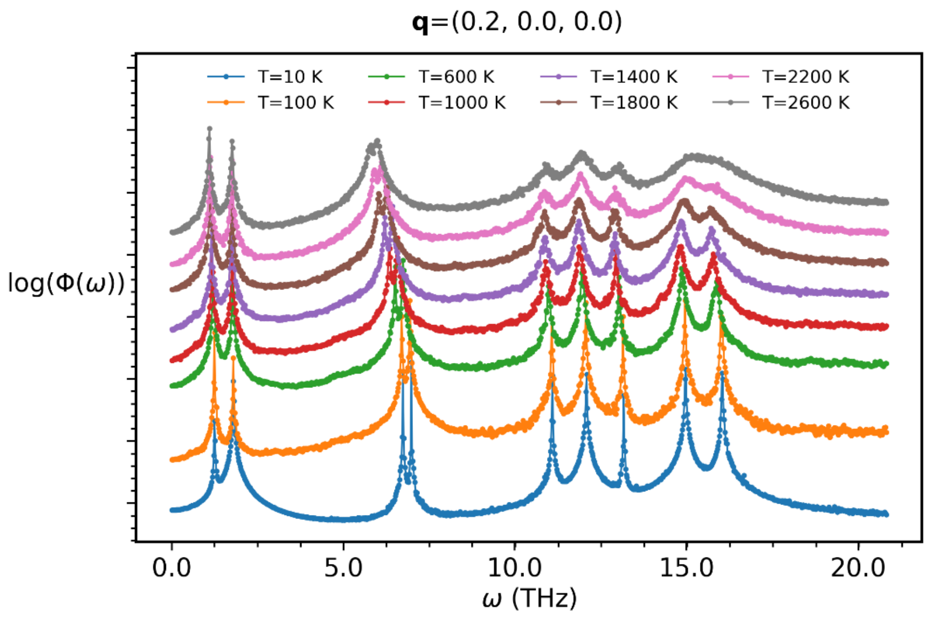

Fig: 5 Temperature dependent spectral energy density q-slice

This is the data in fig. 4 plotted at a particular q-slice. Note to broadening of the optical phonons and softening of acoustic modes as temperature increases

Phonon-SED

Phonon spectral energy density from LAMMPS simulations

Dictionary:

- NUM_ATOMS: (integer) the number of atoms in your crystal. You can get this from the LAMMPS crystal input file.

- NUM_STEPS: (integer) the number of steps in your simulation. (i.e. 'run', see the LAMMPS section of this doc.)

- TIME_STEP: (real) the MD timestep, see the LAMMPS section. It should match 'timestep' in the LAMMPS input file. Note that my units are fs, where as LAMMPS uses ps (at least my input file does!).

- STRIDE: (integer) The sampling frequency in the simulation. See the LAMMPS sec. This is the 'dt_dump' variable.

- NUM_SPLITS: (integer) this is used to block average your trajectory for better statistics. E.g. if you set it to 4, it will divide your velocity file into 4 chunks, calculate the SED separately for each, then average them together.

- LAT_PARMAMS: (3 x real): these are the lattice constants in Angstrom

- PRIM_VECS (9 x real): these are the primitive lattice vectors. It should be 9 real numbers according to: b_{1x} b_{1y} b_{1z} b_{2x} b_{2y} b_{2z} b_{3x} b_{3y} b_{3z}. where e.g. b_{1x} is the x-component of the the 1st lattice vector (denoted by b)

- OUT_PREFIX: (string) just a prefix to be added to all the output files

- LATTICE_FILE (string) should just be lattice.dat

- VELS_FILE: (string) name of the LAMMPS velocity file

- POS_FILE: (string) name of the LAMMPS pos file

- FILE_FORMAT: (string): just leave it as LAMMPS unless you know what you are doing

- COMPRESS: (bool 0/1). Running pSED with this set to 1 will cause the code to open the VELS_FILE and POS_FILE files and save them as a hdf5 data base. This actually takes a while, and the code will warn you of this! You have to do this before you can calculate the SED. Once it finishes, the code will exit, so you should set this back to 0 and run ./pSED again to continue.

- NUM_QPATHS: (integer) the number of q-paths to calculate on. Can be 1 or larger. e.g. if going from 0.0 0.0 0.0 to 0.5 0.0 0.0 then to 0.5 0.5 0.0, this should be set to 2.

- NUM_QPOINTS: (integer) the number of q-points to subdivide the path into. See the introduction as this should match the number of unit cells in your supercell (Born-Von Karmon boundary conditions!)

- QSYM_POINTS: (6*NUM_QPATHS real numbers) this should be an array of real numbers: there should be 3 specifying the beginning and endpoints of each q-path. e.g. for NUM_QPATHS=1, this could be 0.0 0.0 0.0 0.5 0.0 0.0 to go from Γ to X.

- DEBUG: (bool, 0/1) this will force the code to skip calculating the SED on any other chunks determined by NUM_SPLITS, only on the first one. e.g. if you want to do a quick test of your results, set NUM_SPLITS to something large and DEBUG = 1 so that only a small chunk of data is used.

- PLOT_BANDS (bool, 0/1): this will read the .pSED file and plot the dispersion using matplotlib's imshow() module. Look in Plot.py if you want to change this behavior. Note that this will cause the code to create the plot and then exit. If the .pSED file isn't available, it will complain!

- PLOT_SLICE: (bool, 0/1) this will cause the code to plot the SED only at a single q-point in a traditional plot instead of a heatmap (e.g. in PLOT_BANDS). Note that this will cause the code to create the plot and then exit. If the .pSED file isn't available, it will complain!

- Q_SLICE_INDEX: (integer in range 0, NUM_QPOINTS-1) this specifies which q-point to plot. The index corresponds to the the q-points in the .Qpts file.

- LORENTZ (bool, 0/1): set to 1 to fit a Lorentz function to a peak. You have to specify which q-point, how many peaks to fit, and a guess for each peak location. Specify the q-point with Q_SLICE_INDEX, how many peaks with NUM_GUESSES, and guesses for the peaks with PEAK_GUESSES. Note that this will cause the code to create the plot and then exit. If the .pSED file isn't available, it will complain! Also, PLOT_SLICE overrides this, so make sure that is set to 0 before using this.

- NUM_GUESSES (integer, greater than 0): the number of peaks differ peaks to attempt to fit a Lorentz function to.

- PEAK_GUESSES: (integers, NUM_GUESSES of them): these should be indices for the peak guesses in NUM_GUESSES. You can look these up by plotting a q-slice of the SED thanks to PLOT_SLICE. Just write down the approximate center for all the peaks by looking at the number on the bottom of the plot, then give them to PEAK_GUESSES… I don't remember if the order matters.

References:

1: Phys. Rev. B **81**, 081411 (R) 2010
2: Jour. Appl. Phys. **117,** 195102 (2015)