Simulation of Metals
Using the Embedded Atom Method (EAM) Potential

David Keffer
Department of Materials Science & Engineering
University of Tennessee, Knoxville
date begun: February 16, 2016
date last updated: February 16, 2016

Table of Contents

I. Purpose of Document

The purpose of this document is to provide a practical introduction to the simulation of metals using the embedded atom method (EAM) potential via LAMMPS. This document includes some brief history, a little theory and several practical examples.

II. Embedded-Atom Method

II.A. First Publications

The EAM potential is due to the work of Daw and Baskes. The first report is provided in the following letter (cited over 1,500 times as of Februrary, 2016):

Title: Semiempirical, Quantum Mechanical Calculation of Hydrogen Embrittlement in Metals
Authors: Murray S. Daw and M. I. Baskes
Journal: Physical Review Letters
Volume: 50 Issue: 17 Pages: 1285-1288 Published: April 25, 1983

A more complete description by Daw and Baskes followed shortly (cited over 5,500 times as of Februrary, 2016):

Title: Embedded-atom method: Derivation and application to impurities, surfaces, and other defects in metals
Authors: Murray S. Daw and M. I. Baskes
Journal: Physical Review B
Volume: 29 Issue: 12 Pages: 6443-6453 Published: June 15, 1984

II.B. Key Points of the Theory

From their 1984 paper in Phys. Rev. B, a few key points of the theory emerge. The student should read this paper.

- The energy can be decomposed into a local embedding energy, which each atoms feels based on the local electron density, and a longer-ranged lattice energy.
- The embedding energy is not pairwise additive. The lattice energy is pairwise additive.
- The embedding energy is a functional of the local electron density of the host material. It is speculated that this functional has a universal form, independent of host, but is "unknown and probably rather complicated".
- For pure components, the potential can be related to the lattice constant, elastic constants, vacancy-formation energy and sublimation energy. This allows reasonable parameterization of pure metals, based on readily available material properties.
- When impurities are formed, it is necessary to know the embedding energy potential for each atom type and pair potential for each type of pair.

- Mixed pair potentials assume a Coulombic form with effective charges.
- For FCC metals, the pair potential is cut off beyond the first neighbor.

III. EAM in LAMMPS

III.A. LAMMPS User Manual

EAM potentials appear as tabulated functions for the embedding energy, the electron density and pair potential. The LAMMPS user manual contains an entry on EAM potentials, located at the following URL: http://lammps.sandia.gov/doc/pair_eam.html. There is a discussion of formats for these files.

III.B. EAM Potential Files

Some EAM potential files are included in the potentials directory that can be downloaded with the source file. The LAMMPS manual also notes that there are other repositories of potentials:

- http://www.ctcms.nist.gov/potentials
- http://cst-www.nrl.navy.mil/ccm6/ap
- http://enpub.fulton.asu.edu/cms/potentials/main/main.htm

III.C. Examples

III.C.1: EAM Potential File

On the examples page of the course website, there are currently four example files for bcc Fe. These examples use the following input file, Fe_mm.eam.fs, taken from the LAMMPS directory of potentials. At the top of this file, the following source is indicated.

> Title: Development of new interatomic potentials appropriate for crystalline and liquid iron
> Authors: M. I. Mendelev , S. Han , D. J. Srolovitz , G. J. Ackland , D. Y. Sun & M. Asta
> Journal: Philosophical Magazine
> Volume: 83 Issue: 35 Pages: 3977-3994 Published: December 11, 2003

Appendix A of this reference provides functional forms and numerical values for the parameters in the potential. However, the EAM potential is rendered in a tabular format for use in LAMMPS.

III.C.2: Example 1. Single Point Energy Calculation

The first example calculates the energy of a static unit cell. Note that there is no minimization or simulation in this first example. It simply computes the energy of the initial configuration.

Also note that there are only two atoms in the unit cell, however there are 58 neighbors. When the cut-off is larger than the unit cell, LAMMPS includes images in the force and energy calculation.

III.C.3: Example 2. Single Point Energy Calculation w/ Input Lattice Constant

The second example performs the same task as the first example but does so by allowing the user to input the lattice parameter of the crystal structure via the command file rather than by hard-coding it into the LAMMPS input file.

III.C.4: Example 3. Series of Single Point Energy Calculations

The third example performs the same task as the first example over a range of lattice constants. There is again no minimization or simulation. In this example, there is a bash script that creates the input file on the fly for each calculation then removes it when the calculations are finished.

III.C.5: Example 4. Optimization of Lattice Parameter, followed by MD simulation

The fourth example performs both a minimization and a simulation. First, a one-dimensional optimization of the lattice parameter is performed. The objective function for this optimization is the potential energy of the crystal. This optimization returns the zero temperature lattice parameter. We can call this the zero temperature equilibrated result because at zero temperature, the entropic contribution to the free energy is zero, so the free energy is simply determined by the potential energy.

This optimization is performed using the Polak-Ribierre version of the Conjugate Gradient method. For more on this subject, see Chapter 10 of Numerical Recipes. The instructor has converted the FORTRAN code for the Polak-Ribierre version of the Conjugate Gradient found in Numerical Recipes to Matlab for those interested. It is posted in the library of subroutines section of the MSE 510 website, located at the following URL: http://utkstair.org/clausius/docs/mse510/index.html .

After the optimal lattice constant is obtained, a simulation at finite temperature is performed.

III.D. Elastic Constants

III.D.1. Basic Review

Because some of the students in this class may have little familiarity with elastic constants, we make a special point to define them here. These notes borrow liberally from the Wikipedia page on Hooke's Law, https://en.wikipedia.org/wiki/Hooke%27s_law . The stresses and strains of the material are often assumed to be linearly related to each other, in the "elastic" region, via an elastic constant. In a one-dimensional system, the strain, σ, is related to the stress, ε, via

$$\sigma = -c\epsilon,$$

In a three-dimensional Cartesian coordinate system, the stress and strain tensors can be represented by matrices

$$\epsilon = \begin{bmatrix} \epsilon_{1\,1} & \epsilon_{1\,2} & \epsilon_{1\,3} \\ \epsilon_{2\,1} & \epsilon_{2\,2} & \epsilon_{2\,3} \\ \epsilon_{3\,1} & \epsilon_{3\,2} & \epsilon_{3\,3} \end{bmatrix} \qquad \sigma = \begin{bmatrix} \sigma_{1\,1} & \sigma_{1\,2} & \sigma_{1\,3} \\ \sigma_{2\,1} & \sigma_{2\,2} & \sigma_{2\,3} \\ \sigma_{3\,1} & \sigma_{3\,2} & \sigma_{3\,3} \end{bmatrix}$$

Since the stress, $\sigma_{i\,j}$, and strain, $\epsilon_{k\,\ell}$, are now second order tensors, the elastic tensor (or stiffness tensor) is a fourth order tensor, $C_{i\,j\,k\,\ell}$, such that

$$\sigma_{i\,j} = -\sum_{k=1}^{3}\sum_{\ell=1}^{3} C_{i\,j\,k\,\ell}\epsilon_{k\,\ell}$$

There is an inherent symmetry in this problem that was notated by Voigt. Under properly posed conditions, there are only six independent elements of the stress and strain tensors. If we identify these as one-dimensional vectors, we have

$$[\boldsymbol{\sigma}] = \begin{bmatrix} \sigma_{11} \\ \sigma_{22} \\ \sigma_{33} \\ \sigma_{23} \\ \sigma_{13} \\ \sigma_{12} \end{bmatrix} \equiv \begin{bmatrix} \sigma_1 \\ \sigma_2 \\ \sigma_3 \\ \sigma_4 \\ \sigma_5 \\ \sigma_6 \end{bmatrix} \quad ; \quad [\boldsymbol{\epsilon}] = \begin{bmatrix} \epsilon_{11} \\ \epsilon_{22} \\ \epsilon_{33} \\ 2\epsilon_{23} \\ 2\epsilon_{13} \\ 2\epsilon_{12} \end{bmatrix} \equiv \begin{bmatrix} \epsilon_1 \\ \epsilon_2 \\ \epsilon_3 \\ \epsilon_4 \\ \epsilon_5 \\ \epsilon_6 \end{bmatrix}$$

Then the stiffness tensor can be expressed as

$$[\mathsf{C}] = \begin{bmatrix} c_{1111} & c_{1122} & c_{1133} & c_{1123} & c_{1131} & c_{1112} \\ c_{2211} & c_{2222} & c_{2233} & c_{2223} & c_{2231} & c_{2212} \\ c_{3311} & c_{3322} & c_{3333} & c_{3323} & c_{3331} & c_{3312} \\ c_{2311} & c_{2322} & c_{2333} & c_{2323} & c_{2331} & c_{2312} \\ c_{3111} & c_{3122} & c_{3133} & c_{3123} & c_{3131} & c_{3112} \\ c_{1211} & c_{1222} & c_{1233} & c_{1223} & c_{1231} & c_{1212} \end{bmatrix} \equiv \begin{bmatrix} C_{11} & C_{12} & C_{13} & C_{14} & C_{15} & C_{16} \\ C_{12} & C_{22} & C_{23} & C_{24} & C_{25} & C_{26} \\ C_{13} & C_{23} & C_{33} & C_{34} & C_{35} & C_{36} \\ C_{14} & C_{24} & C_{34} & C_{44} & C_{45} & C_{46} \\ C_{15} & C_{25} & C_{35} & C_{45} & C_{55} & C_{56} \\ C_{16} & C_{26} & C_{36} & C_{46} & C_{56} & C_{66} \end{bmatrix}$$

so that Hooke's Law can be written as

$$\sigma_i = C_{ij}\epsilon_j \ .$$

III.D.2. Elastic Constants from LAMMPS

Researchers are routinely calculating elastic constants in LAMMPS. There are two directories of examples for doing so in the examples folder on Newton. To access these examples, type `module avail` at the command line prompt. Find the most recent version of LAMMPS, which in our case is currently `lammps/7Dec15`. To discover the location of these files, type `module display lammps/7Dec15`. This command reveals that the path is `/data/apps/lammps/7Dec15`. If we change to that directory, there is a sub-directory names `examples`. This sub-directory includes many useful examples, including `ELASTIC` and `ELASTIC_T`. The first example computes elastic constants at zero temperature and the second at finite temperature. Here we will computer elastic constants at finite temperatures.

In the example contained within `ELASTIC_T`, there is a LAMMPS input file, `in.elastic`. This file calls three other files, `init.mod`, `potential.mod` and `displace.mod`. The first two files define the material system. The third file directs a series of simulations from which each component of the strain is varied and each component of the stress is recorded. From the strain and stress, one can compute the elastic constants.

In order to complete homework assignment two, the student must alter `init.mod` and `potential.mod`. No change in `displace.mod` or `in.elastic` is necessary. Of course, the student may certainly make modfications to any file or to start from scratch, should it suit her.

There are numerous features of LAMMPS input in this example, that we have not yet seen. A few of the most noticeable new functionalities are

- `include`: The include statement allows you make your LAMMPS input file "modular" (by splitting it into parts or modules). This allows you to split up a very large input file into manageable parts that have specific functionality. (http://lammps.sandia.gov/doc/include.html)
- `variable`: This command defines functional relationships to be evaluated later in the script or during the simulation. (http://lammps.sandia.gov/doc/variable.html)
- `write_restart`: This command allows you to save the configuration to resume the simulation from this point. (This is also useful for long runs that are killed by the job queue or a machine failure. The simulation can be rerun from the last restart saved.) (http://lammps.sandia.gov/doc/write_restart.html)
- `read_restart`: This command allows you to read the saved restart file. (http://lammps.sandia.gov/doc/read_restart.html)
- `change_box`: This command allows you to change some aspect of your simulation volume. (http://lammps.sandia.gov/doc/change_box.html)

We also observe in this example that there are numerous ways to access variables. They are summarized here.

- `${varname}`: This command allows us to access the value stored in varname.
- `c_ID` = global scalar, vector, or array calculated by a compute with ID

- `c_ID[I]` = Ith component of global vector or Ith column of global array calculated by a compute with ID
- `f_ID` = global scalar, vector, or array calculated by a fix with ID
- `f_ID[I]` = Ith component of global vector or Ith column of global array calculated by a fix with ID
- `v_name` = global value calculated by an equal-style variable with name

We also observe in this example that the fix command can be used to compute time averages, as in

- fix avp all ave/time ${nevery} ${nrepeat} ${nfreq} c_thermo_press mode vector

This fix command with id avp, calculates the temporal average of all components of the thermodynamic pressure tensor. This will come in handy for determining elastic constants.

III.D.3. Two Additional Linux Commands (`grep` and `diff`)

In unraveling the mysteries of the ELASTIC_T example, you may find yourself asking, where was "f_avp" defined? Since the input file is now split into four parts, it is not easy to find it. In the linux environment, you can use the `grep` function, with the syntax

```
grep string files
```

So for example, if you want to find all instances of "avp" in all files of the directory, you can type

```
grep avp *
```

If you want only to look in the *.mod files, you can type

```
grep avp *.mod
```

If you want to save this information to look at later, you can redirect the output to a file, for example, greptemp.txt, as follows

```
grep avp *.mod > greptemp.txt
```

For more information on grep, including how to change case sensitivity, check out the grep man file.

Since you have a working version of the code, there are two approaches to modifying it to do what you want. One method is make all your changes at once. This will work if you know what you are doing and you are very careful and no one interrupts you in the middle of the modification and it is your lucky day. If you suspect that any of these conditions may not be met, it is recommended for homework assignment 2 that you only make one or two modifications at a time and run the code, if possible, at intermediate points during the modification. This helps minimize frustration. Because if you make many changes at one time and the resulting code doesn't work, you don't know which of your changes has a problem.

Regardless, it may become useful to compare the original file and the new version that you have modified. In linux, you can compare the contents of two files using the `diff` command, as follows

```
diff file1 file2
```

This command will show you which lines are different between the two files. If you want to save this information to look at later, you can redirect the output to a file, for example, difftemp.txt, as follows

```
diff file1 file2 > greptemp.txt
```