**Implementation of Local Density Potentials in LAMMPS**

Tanmoy Sanyal and M. Scott Shell, UC Santa Barbara

We are trying to extend LAMMPS with a new potential style that is, in some sense, a generalization of embedded atom models. We call it a “local density potential” because it assigns an energy to an atom depending on the number of neighboring atoms of given type around it within a predefined spherical volume (i.e., within a cutoff). Our initial work suggests that such potentials could be widely useful for (1) implicit solvation approaches and (2) capturing effective multibody forces in a computationally efficient manner so as to improve the quality of coarse grained models.

Our general approach is to implement the potential in LAMMPS so that it can be used as a hybrid style with other explicit pair interaction terms (e.g., table spline, Lennard Jones, etc.). Because the local density potential is not a pair potential per se, our approach to the implementation is to simply supply a single auxiliary file with all specifications for the local density potentials that will be read upon initialization.

The following sections describe the potential itself and how we wish to implement it in LAMMPS.

**Illustration of the potential in a system with a single atom type**

A system of a single atom type (e.g., LJ argon) with a single local density (LD) potential would have an energy that follows

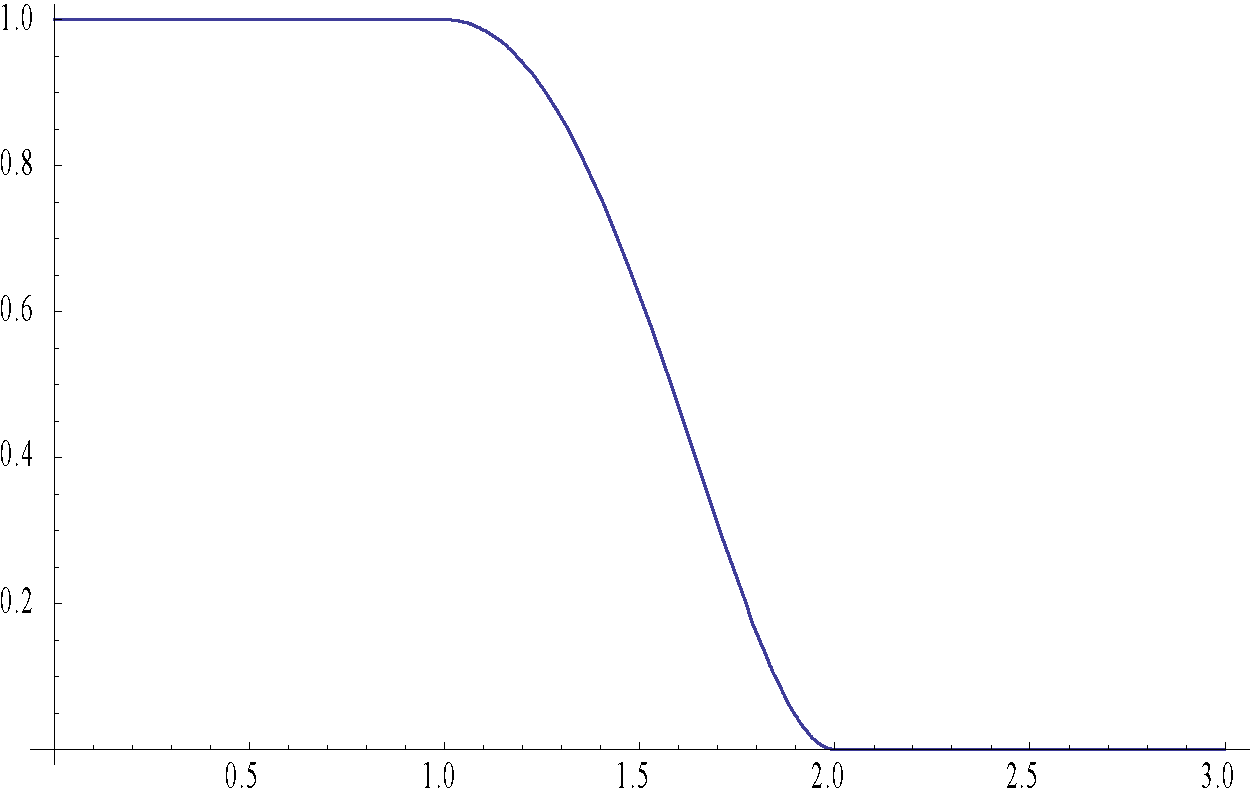
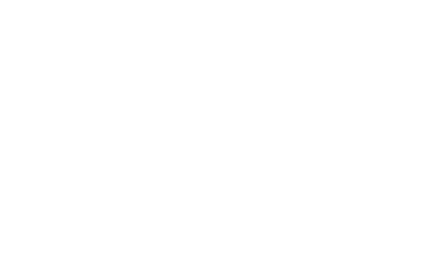
where is the local density at atom *i* and is similar in spirit to the embedding function used in EAM potentials. The local density at atom *i* is given by the sum

where is an **indicator function** that is one at and zero beyond a cutoff distance. The choice of is somewhat arbitrary, but the following piecewise cubic function seems sufficiently general:

with

The constants above are chosen so that the indicator function smoothly interpolates between 1 and 0 between the distances and , which we call the **inner** and **outer** cutoffs. It satisfies the properties

The figure below illustrates for and .



The “embedding function” may or may not have a closed-form expression. To maintain generality, we represent it with a spline-interpolated table over a predetermined range of . Outside of that range it simply adopts values at the endpoints.

**Systems with arbitrary numbers of atom types**

The potential is easily generalized to systems involving multiple atom types:

with

where gives the type of atom , the type of atom , and the coefficients and filter for atom types as specified by the user. We call the **central atom filter** as it determines to which atoms the potential applies:

On the other hand, we call the **neighbor atom filter** because it specifies which atom types to use in the calculation of the local density potential:

Note that the potentials need not be symmetric with respect to atom types, which is the reason for two distinct sets of coefficients and . An atom type may contribute to the local density but not the potential, or to the potential but not the local density. Such decisions are made by the user and should (ideally) be motivated on physical grounds for the problem at hand.

**Expression for the force**

As usual, the force extends directly from the potential derivative:

This has the form of a pairwise force. The force on central atom due to neighbor is

As indicated above, an equal and opposite force applies to atom . However, a *second* force arises when is the central atom and is the neighbor:

The total pair force that must be added to and subtracted from is therefore

Note that the local densities must be computed in a separate, previous pair loop.

**General form for implementation in LAMMPS**

Of course, a system with many atom types may have many different possible local density potentials, each with their own atom type filters, cutoffs, and embedding functions. The most general form of this potential, which we are implementing in LAMMPS, is

where is an index that spans the (arbitrary) number of applied local density potentials . Each local density is calculated as before with

The superscript on the indicator function simply indicates that it is associated with specific values of the cutoff distances, and .

To summarize, there may be distinct local density potentials. With each potential, one must specify:

* the inner and outer cutoffs and
* the central type filter where
* the neighbor type filter where
* the local density potential , typically as a table that is later spline-interpolated

**Approach to implementation in LAMMPS**

Even though the local density potential isn’t actually a pair potential, it seems most natural to implement it as a pair style based on examples in the MANYBODY package. Our starting point has been the code for the pair style eam/alloy. Our current implementation strategy is presented below:

Name of the pair style in LAMMPS input script: localdensity

How input parameters are taken: pair\_style localdensity <input file>

(no pair\_coeff command)

Input file format:

Line 1: comment or blank (this line is ignored)

Line 2: comment or blank (ignored)

Line 3: (number of LD potentials)

Line 4: blank (ignored)

Line 5: (lower and upper cutoffs)

Line 6: centraltypes (central atom types separated by spaces)

Line 7: neighbortypes (neighbor atom types separated by spaces)

Line 8: (number of values of and tabulated)

Line 9:

Line 10:

Line 11:

Line 11:

…

Line 9+:

Line 10+: blank (ignored)

Block 2

Block 3

…

Block

Thus the input file is separated into blocks each representing a separate local density potential and each specifying its own upper and lower cutoffs, central and neighbor atoms, and potential. In general, blank lines anywhere are ignored.

**Other specifications within the code:**

From a thorough study of several pair style source codes, we point out some specifications that we think are useful. Since, we are not very proficient in C++, these might need modification while actually writing the code.

1. Since, this is a local environment dependent potential, the use of full rather than half neighbor lists is preferred.
2. The coeff method of the class would only contain an error checking. If the pair\_coeff statement is used in the input script it will simply print an error message.

**List of variables, structures, functions etc (to be placed in the header file):**

double \*\*frho // 2D array for holding all the values read from file, one row for each potential.

double \*\*rho // 2D array for holding the values read from file, one row for each potential.

int npotentials // Number of LD potentials

int \*nrho // 1-D array, nrho[k] is the number of tabulated values for the LD potential.

double \*uppercut // 1-D array, uppercut[k] contains uppercut for the LD potential.

double \*lowercut //1-D array, lowercut[k] contains lowercut for the LD potential.

int \*\*centermask

/\*1-D array mapping the central atoms listed in file to the atom types for every potential block. Thus for the LD potential centermask[k][i] = 1, if atomtype i is present as a central atom for this potential, else centermask[k][i] = 0. \*/

int \*\*neighmask

/\*1-D array mapping the neighbor atoms listed in file to the atom types for every potential block. Thus for the LD potential neighmask[k][i] = 1, if atomtype i is present as a central atom for this potential, else neighmask[k][i] = 0. \*/

Constructor, Destructor

void settings(int narg, char \*\*arg)

void coeff (int narg, char \*arg)

void allocate()

double init\_one(int i, int j)

void init\_style()

void compute( {…})

void read\_file(char \*filename)

void spline\_interpolate()

double get\_phi(double rsq)

**Pseudo-code**

This is rough draft of the pseudo-code for the functions described above.

Constructor

Create all global arrays and declare all of them to be NULL.

Set npotentials = 0

Destructor

Destroy all created arrays

init\_style()

Request a full neighbor list

init\_one(int i int j)

cutmax = max(uppercut)

return cutmax

allocate()

make setflag[i][j] = 1 for all (i,j)

set allocated = 1

settings(int narg, char \*\*arg)

If (narg != 1)

print error message

else

read\_file(arg[0])

coeff(int narg, char \*arg)

print error message

read\_file (char \*filename)

Find npotentials from file.

Set up the following arrays:

memory->create (nrho, npotentials, “pair:nrho”)

memory->create(uppercut, npotentials, “pair:uppercut”)

memory->create(lowercut, npotentials, “pair:lowercut”)

memory->create(centermask, npotentials, atom->ntypes, “pair:centermask”)

memory->create(neighmask, npotentials, atom->ntypes, “pair:neigmask”)

memory->create(frho, npotentials+1, max(nrho), “pair:frho”)

memory->create(rho, npotentials+1, max(nrho), “pair:rho”)

Parse input file to populate all these arrays. Finally, populate frho[npotentials+1][] and rho[npotentials+1][] to zero so that atom types that are not mapped to any central atom types can point to these, (to be used for pair hybrid/overlay).

Call allocate()

get\_phi(double , k)

take as input the value of and calculate using Equation (1). Coefficients are all calculated inside this function. The uppercut and lowercut values are extracted as:

= lowercut[k]

= uppercut[k]

The reason for using a separate function for calculating is that, this can be modified to include other different type of local densities, if required.

spline\_interpolate(double LD, k)

Performs the spline interpolation using cubic B-splines. The index k is essential so that it knows which frho[k][] and rho[k][] to use.

compute

type = atom->type

inum = list->inum;

ilist = list->ilist;

numneigh = list->numneigh;

firstneigh = list->firstneigh;

// calculate local density for each atom

for i = 0 to inum {

jlist = firstneigh[i];

jnum = numneigh[i];

for j = 0 to jnum {

if cutsq[i][j]>=max(uppercut)^2 continue

Calculate

for k = 1 to npotentials {

if centermask[k][i] and neighmask[k][i]

if < uppercut[k]^2

LD[i] += get\_phi( k)

elseif centermask[k][j] and neighmask[k][i]

if <uppercut[k]

LD[j] += get\_phi(, k)

}

}

}

// calculate local density potential for each atom

for i = 0 to inum {

for k = 1 to npotentials {

if centermask[k][i]

(energy, force) = spline\_interpolate(LD[i], k)

Add this energy and force to the total accumulated

energy and force.

}

}

Note: The array LD[] contains the local density of each atom whose atom-type is specified as a central atom in the input file. It was not specified in the declaration or the dynamic creation of arrays, since we cannot figure out exactly how to do this.