Routines for basic tests of atomistic potentials with universal interface

Bohumir Jelinek, Sergio D. Felicelli

Center for Advanced Vehicular Systems & Mechanical Engineering Department

Mississippi State University

John F. Peters

U.S. Army Engineer Research and Development Center

Vicksburg, Mississippi

Kiran Solanki

Arizona State University

Overview

- Testing of classical molecular dynamics (MD) potentials
- Atomistic Simulation Environment (ASE)
- ASE universality: electronic structure codes + LAMMPS
- ASE examples
- Tests of defect energies, heats of formation, elastic constants
- Summary

Testing of classical MD potentials for alloys

Task at hand

- validate *Modified Embedded Atom Method* (MEAM) potentials for Al, Si, Mg, Cu, Fe, and their alloys

Method

1. calculate basic structural properties of single crystals, formation energies of defects, and structural and elastic properties of simple compounds using MEAM
2. compare with other interatomic potentials and *ab-initio* methods

Issues

Need to learn formats of input parameter files, atomic configuration files, and output files of

- classical MD code implementing MEAM (LAMMPS)
- *ab-initio* code (VASP)

Need to create atomic configurations for

- single crystal structures, crystallic compounds, point defects (vacancies, interstitials, substitutions), planar defects (varying surfaces, stacking faults), and strained structures

What would help

A tool applicable to quickly evaluate basic properties from classical MD potentials and $ab\text{-}initio$ methods.

Ideally, a single universal tool would be able to

- create basic atomic configurations and manipulate them
- serve these atomistic configurations as inputs to a variety of methods/simulation codes and obtain energies

Anything like that available?

Atomistic Simulation Environment (ASE)[1]

- universal Python interface to many DFT codes (calculators), with visualization, simple GUI, documentation, and tutorials
- creates molecules, crystal structures, surfaces, nanotubes, analyzes symmetry and spacegroups
- provides support for Equation of state, structure optimization, dissociation, diffusion, constrains, NEB, vibration analysis, phonon calculations, infrared intensities, MD in NVE, NVT, and NPT ensembles, STM, and electron transport
- recent support for LAMMPS by Jörg Meyer (TU München)

[1] S. R. Bahn and K. W. Jacobsen, An object-oriented scripting interface to a legacy electronic structure code, Comput. Sci. Eng., Vol. 4, 56-66, 2002, https://wiki.fysik.dtu.dk/ase/

Calculators[1] working with ASE

Code	Description	Type
vasp	Planewave PAW code	DFT
abinit	A planewave pseudopotential code	DFT
siesta	LCAO pseudopotential code	DFT
exciting	Full Potential LAPW code	DFT, LAPW
jacapo	ASE interface to Dacapo, planewave ultra-soft pseudopotentials	DFT
dftb	DftbPlus DFT based tight binding	DFT
turbomole	Fast atom orbital code Turbomole	DFT, HF
FHI-aims	Numeric Atomic Orbital, full potential code	DFT, HF
fleur	Full Potential LAPW code	DFT, LAPW
emt	Effective Medium Theory calculator	EMT
Asap	Highly efficient EMT code (written in C++)	EMT
GPAW	Grid-based real-space PAW code	DFT, HF
Dacapo	Old interface to Dacapo. Requires Numeric python and ASE2	DFT
lammps	Classical molecular dynamics code	CMD

[1] https://wiki.fysik.dtu.dk/ase/ase/calculators/calculators.html

ASE documentation example - Calculators

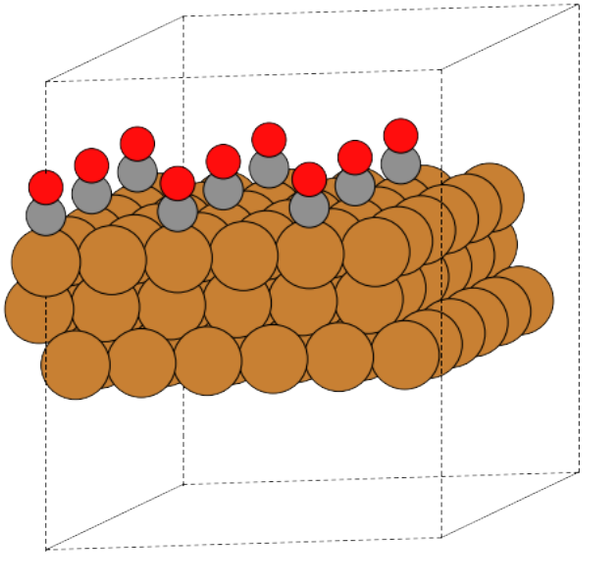
```
>>> a = read('molecule.xyz')
>>> e = a.get_potential_energy()
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
  File "/home/jjmo/ase/ase/atoms.py", line 547, in \
    get_potential_energy
    raise RuntimeError('Atoms object has no calculator.')
RuntimeError: Atoms object has no calculator.
>>> from ase.calculators.abinit import Abinit
>>> calc = Abinit(ecut=20*Ry)
>>> a.set_calculator(calc)
>>> e = a.get_potential_energy()
>>> print e
```

ASE documentation example - File i/o

```
from ase.lattice.surface import *

adsorbate = Atoms('CO')
adsorbate[1].z = 1.1

a = 3.61
slab = fcc111('Cu', (2, 2, 3), a=a, vacuum=7.0)
add_adsorbate(slab, adsorbate, 1.8, 'ontop')
```



```
from ase.io import *
write('slab.png', slab  * (3, 3, 1), rotation='10z,-80x',
      show_unit_cell=2)
write('slab.xyz', slab)
b = read('slab.xyz')
```

Transformation of general simulation cell into LAMMPS specific coordinate system

Issue:

- LAMMPS supports non-orthogonal (triclinic) simulation boxes, but triclinic box vectors cannot be arbitrarily oriented
- transformation is needed from ASE general coordinate system to LAMMPS specific coordinate system and back
- implemented in LAMMPS calculator (Jörg Meyer)

Basic tests of atomistic potentials (our work)

Elastic constants: C_{44} and (C_{11}-C_{12})/2

Heats of formation for binary compounds:

B1, B2, B3, C1, C15, A15, $D0_3$, $L1_2$

Planar defects - surfaces:

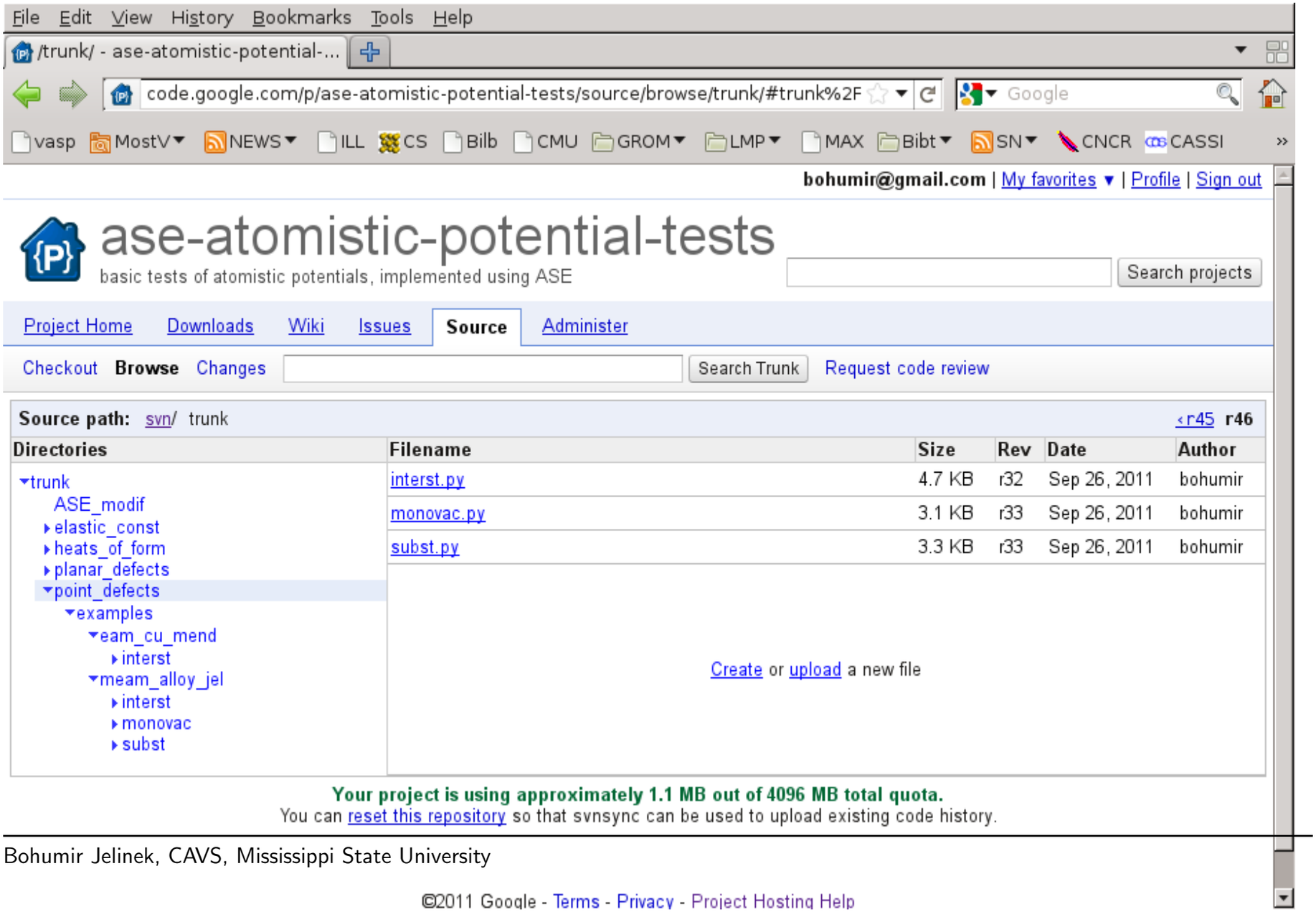
fcc (111), (110), (100)

bcc (111), (110), (100)

diamond (111), (100)

hcp (0001), $(10\bar{1}0)$

Point defects: vacancies, interstitials, substitutions


File Edit View History Bookmarks Tools Help
/trunk/ - ase-atomistic-potential-...
code.google.com/p/ase-atomistic-potential-tests/source/browse/trunk/#trunk%2F
Google
vasp MostV NEWS ILL CS Bilb CMU GROM LMP MAX Bibt SN CNCR CASSI
bohumir@gmail.com | My favorites | Profile | Sign out
ase-atomistic-potential-tests
basic tests of atomistic potentials, implemented using ASE
Search projects
Project Home Downloads Wiki Issues Source Administer
Checkout Browse Changes
Search Trunk
Request code review
Source path: svn/ trunk
‹r45 r46
Directories
trunk
ASE_modif
elastic_const
heats_of_form
planar_defects
point_defects
examples
eam_cu_mend
interst
meam_alloy_jel
interst
monovac
subst
Filename Size Rev Date Author
interst.py 4.7 KB r32 Sep 26, 2011 bohumir
monovac.py 3.1 KB r33 Sep 26, 2011 bohumir
subst.py 3.3 KB r33 Sep 26, 2011 bohumir
Create or upload a new file
Your project is using approximately 1.1 MB out of 4096 MB total quota.
You can reset this repository so that svnsync can be used to upload existing code history.
©2011 Google - Terms - Privacy - Project Hosting Help

Example - elastic constants

```
#!/usr/bin/env python

from ase.units import kJ, _e

# obtain species, structure, and lattice parameter from command line
#
import sys
argc = len(sys.argv)
if argc < 5:
    print 'usage:', sys.argv[0], 'Al Mg nacl lp'
    sys.exit(1)

el1 = sys.argv[1]
el2 = sys.argv[2]
struct = sys.argv[3]
lp = float(sys.argv[4])
print "el1:", el1, "el2:", el2, "str:", struct, "lp:", lp

# read model specification from ./model.py file
# pick elements from the model
#
```

```
import model
from model import pick_elements
species = [el1, el2]
pick_elements(model, species)

# initialize LAMMPS calculator
#
from ase.calculators.lammps import LAMMPS
calc = LAMMPS(parameters=model.parameters, files=model.files,
              specorder=species)

# setup structure
#
import numpy as np
if struct in ["fcc", "nacl", "cu2mg", "mgcu2",
              "zns", "caf2", "f2ca", "alfe3", "fe3al"]:
    # reference cell
    refcell = np.array([[0.0, 0.5, 0.5],
                        [0.5, 0.0, 0.5],
                        [0.5, 0.5, 0.0]])
    if struct == "fcc":
        elems = [el1]
        poss = [(0, 0, 0)]
```

```
    elif struct == "nacl":
        elems = [el1, el2]
        poss = [(0, 0, 0),
                (0.5, 0.5, 0.5)]
    elif struct == "cu2mg":
        elems = [el1, el1, el1, el1, el2, el2]
        poss = [(0.5, 0.5, 0.5),
                (0.0, 0.5, 0.5),
                (0.5, 0.0, 0.5),
                (0.5, 0.5, 0.0),
                (0.125, 0.125, 0.125),
                (0.875, 0.875, 0.875)]
    elif struct == "zns":
        elems = [el1, el2]
        poss = [(0.0, 0.0, 0.0),
                (0.25, 0.25, 0.25)]
    elif struct == "caf2":
        elems = [el1, el2, el2]
        poss = [(0.0, 0.0, 0.0),
                (0.25, 0.25, 0.25),
                (-0.25, -0.25, -0.25)]
    elif struct == "f2ca":
        elems = [el1, el1, el2]
```

```
        poss = [(0.25, 0.25, 0.25),
                (-0.25, -0.25, -0.25),
                (0.0, 0.0, 0.0)]
    elif struct == "alfe3":
        elems = [el1, el2, el2, el2]
        poss = [(0.0, 0.0, 0.0),
                (-0.5, 0.5, 0.5),
                (-0.25, -0.25, -0.25),
                (0.25, 0.25, 0.25)]
    elif struct == "fe3al":
        elems = [el1, el1, el1, el2]
        poss = [(-0.5, 0.5, 0.5),
                (-0.25, -0.25, -0.25),
                (0.25, 0.25, 0.25),
                (0.0, 0.0, 0.0)]
    elif struct == "cu2mg":
        elems = [el1, el1, el1, el1, el2, el2]
        poss = [(0.5, 0.5, 0.5),
                (0.0, 0.5, 0.5),
                (0.5, 0.0, 0.5),
                (0.5, 0.5, 0.0),
                (0.125, 0.125, 0.125),
                (0.875, 0.875, 0.875)]
```

```
    elif struct == "mgcu2":
        elems = [el1, el1, el2, el2, el2, el2]
        poss = [(0.125, 0.125, 0.125),
                (0.875, 0.875, 0.875),
                (0.5, 0.5, 0.5),
                (0.0, 0.5, 0.5),
                (0.5, 0.0, 0.5),
                (0.5, 0.5, 0.0)]

elif struct in ["sc", "aucu3", "cu3au", "cscl", "cr3si", "sicr3"]:
    # reference cell
    refcell = np.array([[1.0, 0.0, 0.0],
                        [0.0, 1.0, 0.0],
                        [0.0, 0.0, 1.0]])
    if struct == "sc":
        elems = [el1]
        poss = [(0.0, 0.0, 0.0)]
    elif struct == "aucu3":
        elems = [el1, el2, el2, el2]
        poss = [(0.0, 0.0, 0.0),
                (0.0, 0.5, 0.5),
                (0.5, 0.0, 0.5),
                (0.5, 0.5, 0.0)]
```

```
    elif struct == "cu3au":
        elems = [el1, el1, el1, el2]
        poss = [(0.0, 0.5, 0.5),
                (0.5, 0.0, 0.5),
                (0.5, 0.5, 0.0),
                (0.0, 0.0, 0.0)]
    elif struct == "cscl":
        elems = [el1, el2]
        poss = [(0.0, 0.0, 0.0),
                (0.5, 0.5, 0.5)]
    elif struct == "sicr3":
        elems = [el1, el1, el2, el2, el2, el2, el2, el2]
        poss = [(0.0, 0.0, 0.0),
                (0.5, 0.5, 0.5),
                (0.25, 0.50, 0.00),
                (0.75, 0.50, 0.00),
                (0.00, 0.25, 0.50),
                (0.00, 0.75, 0.50),
                (0.50, 0.00, 0.25),
                (0.50, 0.00, 0.75)]
    elif struct == "cr3si":
        elems = [el1, el1, el1, el1, el1, el1, el2, el2]
        poss = [(0.25, 0.50, 0.00),
```

```
                (0.75, 0.50, 0.00),
                (0.00, 0.25, 0.50),
                (0.00, 0.75, 0.50),
                (0.50, 0.00, 0.25),
                (0.50, 0.00, 0.75),
                (0.0, 0.0, 0.0),
                (0.5, 0.5, 0.5)]

# create atomic system from elements, positions, and initial cell
#
from ase.lattice.spacegroup import crystal
init_cell = lp * refcell
atoms = crystal(elems, poss, cell=init_cell)

# assign calculator, get energy and volume per atom
#
atoms.set_calculator(calc)
epa0 = atoms.get_potential_energy() / atoms.get_number_of_atoms()
vpa0 = atoms.get_volume() / atoms.get_number_of_atoms()
print "epa0:", epa0
print "vpa0:", vpa0, "\n"

# reoptimize/check volume
```

```
#
volumes = []
energies = []
for x in np.linspace(0.98, 1.02, 5):
    atoms.set_cell(init_cell * x, scale_atoms=True)
    volumes.append(atoms.get_volume() / atoms.get_number_of_atoms())
    energies.append(atoms.get_potential_energy() / atoms.get_number_of_atoms())
print "per atom volumes:", volumes
print "per atom energies:", energies

# fit EOS
#
from ase.utils.eos import EquationOfState
eos = EquationOfState(volumes, energies)
vpaf, epaf, B1 = eos.fit()
print "vpaf:", vpaf, "A^3"
print "epaf:", epaf, "eV"
print "B1:", B1 / kJ * 1.0e24, "GPa"

# get optimal lattice parameter from optimal volume
#
volrc = abs(np.linalg.det(refcell))
optlp = pow(vpaf * atoms.get_number_of_atoms() / volrc, 1. / 3.)
```

```
print "optlp:", optlp, "\n"

# get actual energy at optimal volume
#
opt_cell = optlp * refcell
atoms.set_cell(opt_cell, scale_atoms=True)
epao = atoms.get_potential_energy() / atoms.get_number_of_atoms()

strain = 0.001
diag = ([1, 0, 0],
        [0, 1, 0],
        [0, 0, 1])

# c44
#
defm1 = np.array([[0, strain, strain],
                  [strain, 0, strain],
                  [strain, strain, 0]])
cell1 = np.dot(opt_cell, defm1 + diag)
atoms.set_cell(cell1, scale_atoms=True)

ene1 = atoms.get_potential_energy() / atoms.get_number_of_atoms()
dele = (ene1 - epao) / vpaf * _e / 1.0e-30  # eV/angstrom^3
```

```
c44 = dele / 6 / strain / strain / 1e9  # GPa
print "epao:", epao
print "ene1:", ene1
print "del:", ene1 - epao
print "c44:", c44, "GPa\n"

# c44 other way
#
defm2 = np.array([[0, strain, 0],
                  [strain, 0, 0],
                  [0, 0, 0]])
cell2 = np.dot(opt_cell, defm2 + diag)
atoms.set_cell(cell2, scale_atoms=True)

ene2 = atoms.get_potential_energy() / atoms.get_number_of_atoms()
dele = (ene2 - epao) / vpaf * _e / 1.0e-30  # eV/angstrom^3
c44o = dele / 2 / strain / strain / 1e9  # GPa
print "epao:", epao
print "ene2:", ene2
print "del:", ene2 - epao
print "c44o:", c44o, "GPa\n"

# (c11-c12)/2
```

```
#
defm3 = np.array([[strain, 0, 0],
                  [0, 1 / (1 + strain) - 1, 0],
                  [0, 0, 0]])
cell3 = np.dot(opt_cell, defm3 + diag)
atoms.set_cell(cell3, scale_atoms=True)

ene3 = atoms.get_potential_energy() / atoms.get_number_of_atoms()
dele = (ene3 - epao) / vpaf * _e / 1.0e-30  # eV/angstrom^3
gammap = dele / 2 / strain / strain / 1e9  # GPa
print "epao:", epao
print "ene3:", ene3
print "del:", ene3 - epao
print "(c11-c12)/2:", gammap, "GPa\n"
```

Source path: svn/ trunk/ elastic_const/ examples/ meam_alloy_jel/ command.sh Edit file

```
PATH=$PATH:../..
PYTHONPATH=$PYTHONPATH:.

# directory with optimal lattice paramters for each compound
#
lpd="../../../hofs/examples/meam_alloy_jel/results"

function structures {
    pair=$1
    if   [ $pair == "Al:Si" ]; then echo "nacl cu3au f2ca cscl"
    elif [ $pair == "Al:Mg" ]; then echo "nacl cu3au aucu3 cscl"
    elif [ $pair == "Al:Cu" ]; then echo "nacl f2ca aucu3 alfe3 sicr3 cu3au cscl"
    elif [ $pair == "Al:Fe" ]; then echo "nacl cscl alfe3 aucu3 sicr3 mgcu2 cr3si cu3au fe3al
f2ca"
    elif [ $pair == "Si:Mg" ]; then echo "nacl caf2 aucu3 sicr3"
    elif [ $pair == "Si:Cu" ]; then echo "nacl aucu3 caf2"
    elif [ $pair == "Si:Fe" ]; then echo "nacl cscl alfe3 sicr3 aucu3 f2ca caf2"
    elif [ $pair == "Mg:Cu" ]; then echo "nacl mgcu2 cscl aucu3 alfe3"
    elif [ $pair == "Mg:Fe" ]; then echo "nacl mgcu2 aucu3 cu3au cscl"
    elif [ $pair == "Cu:Fe" ]; then echo "nacl aucu3 sicr3 fe3al cu3au cscl"
    fi
}

pairs="Al:Si Al:Mg Al:Cu Al:Fe Si:Mg Si:Cu Si:Fe Mg:Cu Mg:Fe Cu:Fe"

for ii in $pairs; do
    pair=`echo $ii | sed s/:/\ /`
    nosp=`echo $ii | sed s/://`
    strs=`structures $ii`
    for str in $strs; do
        lp=`awk '$1=="lpopt1:" {print $2}' ${lpd}/${nosp}-${str}.log | tail -1`
        echo $pair $str $lp
        elast.py $pair $str $lp &> results/${nosp}-${str}.log
    done
done
```

Source path: svn/ trunk/ elastic_const/ examples/ meam_alloy_jel/ results/ AlCu-alfe3.log

```
el1: Al el2: Cu str: alfe3 lp: 5.82243673988
epa0: -3.39968125
vpa0: 12.3365678982

per atom volumes: [11.611079013199491, 11.97015949501106, 12.336567898154142, 12.710378242036114, 13.091664546064356]
per atom energies: [-3.38374325, -3.39582275, -3.39968125, -3.39606125, -3.38565375]
vpaf: 12.3365667482 A^3
epaf: -3.39968060427 eV
B1: 107.814645207 GPa
optlp: 5.82243655897

epao: -3.39968125
ene1: -3.39964025
del: 4.1e-05
c44: 88.7460180651 GPa

epao: -3.39968125
ene2: -3.3996675
del: 1.37499999999e-05
c44o: 89.2871523212 GPa

epao: -3.39968125
ene3: -3.3996805
del: 7.49999999883e-07
(c11-c12)/2: 4.8702083077 GPa

```

Conclusions

- ASE provides a universal interface to many electronic-structure codes and LAMMPS
- ASE interface for LAMMPS and VASP was utilized in testing Al-Si-Mg-Cu-Fe MEAM alloy potentials
- Following the LAMMPS example, ASE can provide support to other classical MD codes
- ASE simplifies and increases efficiency of atomistic simulation research