**IPR High-Throughput Scripts**

**Version iprp\_0.0030**

**10/30/2015**

**Lucas Hale**

**Zachary Trautt**

**NIST**

**Standard disclaimers:**

Don’t blame us if it doesn’t work properly, but please tell us about the issue so we can correct it.

If you use this code, please give us credit. Since this is a beta version contact us before publication, etc. so that we can give the proper, up-to-date information on how to cite.

**Development notes**

This software is currently a beta version and doesn’t yet have all features implemented. The known limitations are:

1. Currently, the script runs in serial. The next version should hopefully work with IPython cluster to distribute the calculations to multiple worker engines. Alternatively, it should be possible to run multiple versions of the current script simultaneously without conflicts.
2. Most post-analysis, such as creating plots, is left up to the user. Currently, the raw output is divided between simple text files and csv files. If one of these is much more favorable, let us know.
3. For the time being, master.py needs to be ran from the IPR directory. This is tied into treating the iprp package as a subdirectory instead of an installed package.

**Table of Contents**

1. Extracting and Testing the Code
2. Input Script Details
3. Calculation Functions
4. iprp Package Documentation
5. Data Models
6. **Extracting and Testing the Code**
   1. **Check files**

When you unzip the IPR.zip file you should have an IPR directory and a Notebooks directory. In the IPR directory should be:

* iprp/ Python package with calculations and tools
* ref/ directory containing instances of reference data models
* results/ directory where the simulation results are saved
* example.in input file with highly detailed comments
* test.in input file for quick initial testing that the calculations work
* master.py central script for executing the associated calculations

The Notebooks directory should contain a number of IPython Notebook files (.ipynb). Each one of these is associated with a specific materials property calculation.

* 1. **Initial test**

1. Open test.in. This is the input script for testing the various calculation tools.
   1. For the input scripts, any line that starts with a ‘#’ is treated as a comment line and ignored.
   2. All the other lines are associated with specific commands. More details can be found in Chapter 2 of this guide.
2. Replace “YOUR-LAMMPS-PATH” with the directory information of the LAMMPS executable that you want to use.
3. Save.
4. Run: “python master.py test.in”.
5. The code will run through the various calculations for a limited number of cases.
6. When done, results/json should contain a number of directories containing json files, and results/sim should contain directories for each potential tested.
7. **Input Script Details**

All of the calculations can be ran using the master.py script with an input scripts. This section provides a general overview of the different commands possible. Details and examples can also be found in the example.in file.

Setup commands:

* lammps\_exe is used to specify the full path to the LAMMPS executable to run.
* ipr\_dir is used to specify the path to the IPR directory. Default is the current working directory. (Don’t include and run Master.py from the IPR directory for the time being.)
* potential defines which potentials are to be included in the calculations. Running this command a second time should overwrite the potentials list allowing it to be changed within one run script.
  + potential all says to use all available potentials.
  + potential name allows you to list by name which potentials to include.
  + potential element uses all potentials that contain any of the elements listed.
* crystal is used to build a list of crystal structures that are to be investigated This crystal list is used by all of the run commands except for run\_0K\_structure.
  + crystal clear removes all structures currently in the list. This allows different crystal lists to be used for different calculations.
  + crystal add appends one crystal structure to the list. Each crystal add requires both
    - prototype which defines the crystal’s prototype and
    - elements which is the list of elements corresponding to the prototype’s unique sites.

Run commands:

* run\_0K\_structure executes the static calculations of lattice and elastic constants for a wide variety of crystal prototypes. Run options are:
  + prototypes
    - prototypes all means do all available prototypes. (This is the default).
    - prototypes name allows for specific prototypes to be listed.
  + r\_range specifies the range of nearest neighbor distances to use. (Default is 2.0 5.0 200).
* run\_cij\_vs\_P executes the pressure dependent elastic moduli calculation. This will iterate over all potential-crystal combinations that have 0K\_structure results. Run options are:
  + strain defines the total strain range to explore. (Default is 0.1).
  + steps defines how many strain steps are used for the plot. (Default is 200).
* run\_point\_defect executes the point defect formation energy calculation. This will iterate over all potential-crystal combinations that have 0K\_structure results and an associated point defect library for the crystal’s prototype. Run options are:
  + types
    - types all means do all available point defects. (This is the default).
    - types name means only do point defects included in this list.
  + size\_range defines the minimum and maximum system sizes to use in the analysis. Each size, s, means that the system will be a supercell of sXsXs. Only even sizes are allowed. (Default is 4 10).
* Run\_dislocation constructs a dislocation monopole system, relaxes it and creates differential displacement and Nye tensor maps of the core. Run options are:
  + size sets the x y and z dimension scalers used in generating the system. Each resulting dimension, i, will be at least size[i] \* alat[i] wide. (Default is 40 40 1).
  + temperature specifies the temperature at which to perform an NVT relaxation on the system. If temperature is 0, then only a static energy minimization is performed. (Default is 0).
  + xwidth sets the range of x coordinates to include in the plotting window. (Default is -10 10).
  + ywidth sets the range of y coordinates to include in the plotting window. (Default is -10 10).

1. **Calculation Functions**

The heart of this project is performing a variety of materials property evaluations on interatomic potentials. To accomplish this, a number of isolated and independent calculation methods were determined and implemented into Python. Each calculation function may depend on running one or more LAMMPS simulations, which in turn requires a LAMMPS script generation function.

The ultimate goal is that each calculation function (and all script generators) will be included both as part of the iprp package, and within a demonstration IPython Notebook. The Notebook will serve as a tool for transparently sharing and instructing how the calculation works. Including the calculation within the iprp package allows for the Master.py program to execute it as part of one of the run commands, and gives users the ability to design their own investigations around the basic root tools and scripts used.

The IPython notebooks can be found in the Notebooks directory.

The Python calculation methods can be found in IPR/iprp/calcs, and included in a program using include iprp.calcs.

The LAMMPS script generation functions directly used by the calculations can be found in IPR/iprp/calcs/scripts. (Note some basic tools also exist in iprp/lammps/script\_gens.py). The script generators can be called directly using include iprp.calcs.scripts.

**Note**: Ideally each calculation should be identical in both its Notebook and iprp versions. Currently, there likely are a few small discrepancies. Don’t worry about letting me know about these for the time being unless they are directly affecting the calculation for you.

**Note2**: The calculations included in the Notebooks tool-alat-full-relax, tool-T-vs-Cij, and tool-Vacancy-Mobility are not listed here as the best way to include them in the iprp package and the Master.py program are still being determined. It is also possible that these calculation functions may substantially change.

* 1. **e\_plot**

The function e\_plot evaluates the cohesive energy of a crystal structure as a function of the nearest neighbor radial distance. In doing the calculation, no atomic relaxations are performed and the ratio of the lattice parameters are held fixed. The function parameters are

* lammps\_exe: path to the lammps executable to use
* pot: iprp.lammps.Potential object representing an interatomic potential
* elements: list of the elements to use
* masses: list of the elemental masses to use
* proto: iprp.models.CrystalPrototype object representing a crystal prototype
* rmin=2.0: minimum value of r (nearest neighbor distance) to use
* rmax=5.0: maximum value of r to use
* rsteps=200: number of points between rmin and rmax to use

This function returns

* rvalues: a list of the values of r used
* avalues: a list of the lattice parameter, a, associated with each rvalue
* evalues: a list of the corresponding cohesive energy values computed

**Notebook version:**

This calculation function can be examined and ran using the tool-E-vs-r0 IPython Notebook.

**master.py version:**

This calculation is the first one performed during the run\_0K\_structure command. The results can be found in

* tabular form in .dat files within results/sim/POTNAME/struct/ directories
* archival form in struct--.json files within the results/json/struct directory
  1. **refine\_lat**

The function refine\_lat quickly computes the ideal lattice parameters, cohesive energies and elastic moduli matrix for a specific crystal structure. Starting with a supplied lattice parameter guess, the Virial pressures and elastic moduli for the system are computed. A new lattice parameter guess is then obtained by calculating what strain would give zero pressure, assuming linear elasticity. The process is iterated until the values converge.

In doing the calculation, no atomic relaxations are performed as only the system dimensions (lattice parameters) are altered. As such, this function does not guarantee a structure’s stability. The parameters are

* lammps\_exe: path to the lammps executable to use
* pot: iprp.lammps.Potential object representing an interatomic potential
* elements: list of the elements to use
* masses: list of the elemental masses to use
* proto: iprp.models.CrystalPrototype object representing a crystal prototype
* alat\_init three dimensional numpy array containing lattice parameter guesses

This function returns a dictionary containing

* alat: three dimensional numpy array containing the refined lattice parameters
* ecoh: the cohesive energy of the refined system
* cij: the 6x6 elastic constant matrix for the refined system

**Notebook version:**

This calculation function can be examined and ran using the tool-a-Cij-quick IPython Notebook.

**master.py version:**

This calculation is performed during the run\_0K\_structure command. alat\_init is taken as the lattice parameters associated with the minimum energy obtained from the E vs r0 plot. The results can be found in

* database form in .csv files within results/sim/POTNAME/ directories
* archival form in struct--.json files within the results/json/struct directory
  1. **bulk\_mod**

The function bulk\_mod evaluates the bulk modulus associated with a particular crystal structure by applying a small hydrostatic strain to the system. In doing the calculation, no atomic relaxations are performed. The parameters are

* lammps\_exe: path to the lammps executable to use
* pot: iprp.lammps.Potential object representing an interatomic potential
* elements: list of the elements to use
* masses: list of the elemental masses to use
* proto: iprp.models.CrystalPrototype object representing a crystal prototype
* alat: three dimensional numpy array containing the 3 lattice parameters
* size: three dimensional numpy array specifying the system size

This function returns

* b\_mod: bulk modulus associated with the system

**Notebook version:**

Not implemented yet.

**master.py version:**

This calculation is performed during the run\_0K\_structure command. alat is taken as the refined lattice parameters obtained from refine\_lat. The results can be found in

* database form in .csv files within results/sim/POTNAME/ directories
* archival form in struct--.json files within the results/json/struct directory
  1. **cij\_vs\_p**

The function cij\_vs\_p computes the elastic constants as a function of the hydrostatic pressure. This is accomplished by first applying a range of hydrostatic strains to the system and measuring the Virial hydrostatic pressure as a function of system dimensions (i.e. lattice parameters). The elastic moduli matrix is then computed for each of the system dimensions.

In doing the calculation, no atomic relaxations are performed as only the system dimensions are altered. For non-cubic systems, the hydrostatic pressures and strains may not correspond at high pressures. The parameters are

* lammps\_exe: path to the lammps executable to use
* pot: iprp.lammps.Potential object representing an interatomic potential
* elements: list of the elements to use
* masses: list of the elemental masses to use
* proto: iprp.models.CrystalPrototype object representing a crystal prototype
* alat three dimensional numpy array containing the 3 lattice parameters
* delta strain range to examine (from –delta/2 to +delta/2)
* steps number of data points to evaluate the pressure and elastic moduli

This function returns a dictionary containing

* p: list of pressure values
* C11, etc: list of elastic constant values for a given Cij component

**Notebook version:**

This calculation function can be examined and ran using the tool-Cij-vs-P IPython Notebook.

**master.py version:**

This calculation is performed during the run\_Cij\_vs\_P command. The results can be found in

* Tabular form in .dat files within results/sim/POTNAME/CijvsP/ directories
* archival form in CijvsP--.json files within the results/json/CijvsP directory
  1. **ptd\_energy**

The function ptd\_energy computes the formation energy for different point defects. The analysis evaluates the defect energy within systems of different sizes. A quick structural check is performed to identify if any structures have transformed to another structure. Finally, an infinite atom value is extrapolated from the non-transformed values. The required parameters are

* lammps\_exe: path to the lammps executable to use
* pot: iprp.lammps.Potential object representing an interatomic potential
* elements: list of the elements to use
* masses: list of the elemental masses to use
* proto: iprp.models.CrystalPrototype object representing a crystal prototype
* alat: three dimensional numpy array containing the 3 lattice parameters
* ptd\_params: list containing dictionaries defining defect generation parameters

Optional parameters are

* output\_dir: specifies an output directory for LAMMPS dump files
* ptd\_tag: gives the name of the defect type (used for the dump file name)
* min\_size=4 the minimum system size multiplier to use (must be even integer)
* max\_size=10 the maximum system size multiplier to use (must be even integer)

This function returns

* natoms list of the number of atoms
* energies list of formation energies
* centro list of 3D vectors associated with one of the structure analysis methods
* rcoord list of 3D vectors associated with one of the structure analysis methods

**Notebook version:**

This calculation function can be examined and ran using the tool-Cij-vs-P IPython Notebook.

**master.py version:**

This calculation is performed during the run\_Cij\_vs\_P command. The results can be found in

* Tabular form in .dat files within results/sim/POTNAME/CijvsP/ directories
* archival form in CijvsP--.json files within the results/json/CijvsP directory
  1. **stroh**

The stroh.py module is different in that it contains a collection of functions related to solving the Eshelby anisotropic elasticity solution for a perfectly straight line displacement using the Stroh method. The solution used here always adds the line defect parallel to the z-axes of the system at xy coordinates (0,0). It is also unique in that the code is independent both of LAMMPS and the rest of the iprp package as it is a continuum solution that does not rely on any atomistic properties.

**ax\_check** checks that a supplied set of axes is orthogonal. The parameters are

* axes: a 3x3 numpy array of the crystallographic x-, y-, and z-axes of the system.
* tol: optional tolerance used for rounding out precision errors.

ax\_check returns the normalized direction vectors and their magnitudes. The normalized direction vector matrix is equivalent to the transformation matrix, T, associated with transforming from a [100], [010], [001] orientation to the axes supplied.

**c\_transform** transforms the elastic constant matrix to correspond to the system’s crystallographic axes. The parameters are

* initC: the [100],[010],[001] oriented 3x3x3x3 elastic constant matrix.
* T: the transformation matrix.
* tol: optional tolerance used for rounding out precision errors.

c\_transform returns the transformed 3x3x3x3 elastic constant matrix.

**c\_mn\_to\_c\_ijkl** converts an elastic constant matrix from the 6x6 representation to the 3x3x3x3 representation.

**c\_ijlk\_to\_c\_mn** converts an elastic constant matrix from the 3x3x3x3 representation to the 6x6 representation.

**stroh\_setup** uses the Stroh method to find the solution for the anisotropic elasticity solution. The parameters are

* C: the 3x3x3x3 elastic constant matrix transformed to the system’s orientation.
* tol: optional tolerance used for rounding out precision errors.

This function returns a dictionary containing various solution parameters and terms that are used by the subsequent stroh functions.

**stroh\_preln** computes the pre-ln energy term for the dislocation. The parameters are

* b: the Burgers vector transformed to the system’s crystal orientation.
* sd: the dictionary of solution terms returned by stroh\_setup.
* tol: optional tolerance used for rounding out precision errors.

This function returns the pre-ln factor in eV for b in Angstroms and sd based on using C in GPa.

**stroh\_disp\_point** calculates the elastic solution of the displacement of a point due to the presence of the line defect. As the defect is oriented along the z-direction, the solution only depends on the x and y coordinates of the point. The parameters are

* x: x-coordinate of the point.
* y: y-coordinate of the point.
* b: the Burgers vector transformed to the system’s crystal orientation.
* sd: the dictionary of solution terms returned by stroh\_setup.
* tol: optional tolerance used for rounding out precision errors.

A 3D numpy vector is returned for the displacement solution at that point. The units of the displacement should be in whatever length units are used for x, y, and b for sd. This needs to be verified as only x, y, b in Angstroms with sd based on C in GPa has been fully tested.

**stroh\_stress\_point** calculates the elastic solution of the stress at a point due to the presence of the line defect. As the defect is oriented along the z-direction, the solution only depends on the x and y coordinates of the point. The parameters are

* x: x-coordinate of the point.
* y: y-coordinate of the point.
* b: the Burgers vector transformed to the system’s crystal orientation.
* C: the 3x3x3x3 elastic constant matrix transformed to the system’s orientation.
* sd: the dictionary of solution terms returned by stroh\_setup.
* tol: optional tolerance used for rounding out precision errors.

A 3x3 numpy matrix is returned for the stress solution at that point. The stresses should be in the same units as the C matrix (both as a direct parameter and indirectly through sd). This needs to be verified as only x, y, b in Angstroms with C in GPa­ has been fully tested.

**Notebook version:**

The Stroh calculations can be examined and ran using the tool-Dislocation-Creation IPython Notebook.

**master.py version:**

This calculation is performed during the run\_dislocation command. The results can be found in

* LAMMPS dump files (for per-atom info) within results/sim/POTNAME/disl/ directories
* archival form (for pre-ln) in disl--.json files within the results/json/disl directory
  1. **nye**

The calculation nye computes a per atom strain and Nye tensor for an atomic system. The strain is computed by looking at the positions of an atom’s neighbors and relating them to neighbor vectors in a perfect system. The Nye tensor is then obtained from the gradient of the strain and is a useful metric for identifying and analyzing dislocation core structures. Performing a spatial integration over a dislocation-containing region gives the Burgers vector of that dislocation. The parameters are

* sys: an iprp.System which already has a calculated neighbor list.
* p: a list of perfect neighbor vector arrays.
* tmax: a cutoff angle used in determining atom neighbor pairs for the strain calculation.

The resulting calculation values are added to sys as per-atom quantities. Writing sys using iprp.lammps.write\_dump() will create a LAMMPS dump file including all of the per-atom properties.

**Note**: as mentioned above, sys requires that a neighbor list be previously constructed. This can be done using the System method neighbors() (i.e. sys.neighbors(cutoff)).

**Note2**: the parameters p, tmax, and the cutoff used for the neighbors list depend on the crystal structure chosen. The suggested published values for bcc and fcc crystals are included in the dislocation monopole library data models.

**Note3**: currently this function is limited by the handling of p in that only one set of p vectors is used for all atoms. This prevents the examination of any crystals where the local atomic environment is not equivalent for all atoms.

**Notebook version:**

The Stroh calculations can be examined and ran using the tool-Dislocation-Nye-Tensor IPython Notebook.

**master.py version:**

This calculation is performed during the run\_dislocation command. The results can be found in

* LAMMPS dump files (for per-atom info) within results/sim/POTNAME/disl/ directories
* archival form (for estimated Burgers vector) in disl--.json files within the results/json/disl directory
* as figures in results/json/disl/DISLOCATIONTYPE
  1. **dd**

This function creates a differential displacement map for a dislocation containing system showing arrows representative of the change in neighbor vectors between a defect-containing system and a perfect defect-free system. The parameters of the dd function are

* base: iprp.System of a defect-free system with a neighbor list calculated.
* data: iprp.System containing dislocation(s) and ids corresponding to base.
* prange: a 3x2 matrix identifying the window of atoms to include in the plot.
* burgers: the Burgers vector oriented with respect to the crystallographic directions.
* scale: scaling multipliers for the dd vector arrows (default is [1]).
* save: a Boolean for if the figure will be saved to a file (default is False).
* show: a Boolean for if the figure will be shown (default is True).

This function will create a matplotlib figure of the dd plot where the arrows represent the magnitude of neighbor slip in the Burgers vector direction.

**Note**: the base system needs a nearest neighbors list which can be built using base.neighbors(cutoff).

**Note2**: the z-range to include in prange should correspond to one periodic width along that line. The ideal z-range for a given dislocation can be included in the dislocation monopole library data model files. The color of the atom positions is based on their z-coordinates and will change if the z-range is changed.

**Note3**: scale is read in as a list, which for most systems is typically taken as [1]. Body-centered cubic screw dislocations are special in that the arrows are typically scaled according to the Burgers vector magnitude, which corresponds to scale = [1.8856]. Also, for bcc screw dislocations the edge component can be plotted by adding a second number to scale. For some reason, there is a discrepancy between the ± sense of the edge component vectors obtained here and those shown in published works.

1. **iprp Package Documentation**

The calculations listed in Section 3 are all built around system handling and manipulation tools included in the main iprp module, and in the iprp.lammps module.

More info soon…

1. **Data Models**

In evaluating different interatomic potentials, crystal prototypes, and defects, it is important to realize that each is specified by a specific set of parameters. Various data models are developed to represent these parameter sets and make it trivial to switch between different ones.

More info soon…