**Introduction**

The iprPy framework consists of tools created to help researchers design and run scientific calculations in a better and smarter manner. In particular, there is built-in functionality and guiding principles that support the development of calculations to be reproducible, reusable, adaptable, and sharable. All data produced by the calculations is in a format that can be easily understood and interpreted by both humans and computers. Additional tools are included that make it possible to perform any implemented calculation in a high-throughput manner allowing for comparative studies of methods and models, parameter investigations, and full statistical verifications.

Why is all of this important? A simple way to outline this is by walking through the typical steps that a computational scientist goes through during a molecular dynamics (MD) research project.

1. The researcher begins by selecting appropriate interatomic potentials.
2. Atomic configurations of interest are constructed.
3. Simulations are performed using the potentials and configurations of interest.
4. The data is extracted from the simulation results and analyzed.
5. Models are developed to explain the data and a publication is produced.
6. The researcher moves onto a new project and all files for this work are archived.

The ultimate question is does this produce high-quality, meaningful results? Answering this question requires asking a number of other questions. Why were the particular potentials used selected? Will the results change with a different potential? Were enough configurations used to identify true trends? Are the results sensitive to other parameters not explored? Since MD is deterministic, are the behaviors observed statistically reliable? Can this work be easily validated and reproduced by other researchers?

Other questions also emerge that are of particular interest during subsequent research projects. Can the original researcher locate and understand the scripts and data files that they had created? Are those files and the knowledge for using them easily transferable to other researchers, or will they have to reconstruct the methods themselves? How easy is it to adapt the old calculation methods to new studies? Is the data open and available, or does it need to be reproduced as well?

The structure and tools of iprPy are meant to address most if not all of these questions. Calculation methods are implemented in Python allowing for single scripts to fully describe steps 2-4 described above. This makes the calculation scripts a complete representation of the methodology. Each calculation reads in a structured input file, and produces data in a format that is both human and machine readable. Built-in tools are used that allow for the interatomic potential to be easily swapped, making comparison simulations trivial. The Python scripts can be copied into Jupyter Notebooks allowing for full documentation of the underlying functions such that the process can be easily learned or relearned after step 6. Finally, having the ability to run high-throughput calculations makes it possible to investigate if the data is meaningful and reproducible.

**Overview**

To accomplish the goals outlined in the Introduction, iprPy consists of many components that work together. Not all of these components are code, and in fact some are little more than design guidelines or suggested best practices. This section gives an overview of the different components. More detailed information can be found in the later sections relating to each part of iprPy.

* The calculations themselves consist of Python scripts and any supporting files. Each calculation is designed to be a complete and independent unit of work, which reads all input parameters from a structured input file and produces XML records of the calculation’s metadata and processed data.
* High-throughput scripts are provided for preparing and running multiple instances of each calculation. In this framework, the XML records and raw simulation data are automatically added to a specified database.
* The included iprPy package provides functions and tools supporting rapid design of calculations and high-throughput scripts by allowing for the sharing of common code and parameter definitions. The iprPy package also treats the calculation scripts, XML record formats, and types of databases modularly allowing for new instances of each to be easily added.
* A library of JSON/XML reference data is also included that collects together meaningful parameter sets. For example, the parameters for different crystal prototypes, defect configurations, and the metadata associated with interatomic potentials.
* Primary documentation for the functions and methods of the iprPy package, as well as descriptions of the calculation methods and the meaning of the different parameters is included as Jupyter Notebooks.

The final component is not a direct part of iprPy, but is currently being used for most of the implemented calculations.

* atomman is a Python package designed for supporting MD simulations. In particular, it allows for the creation, and manipulation of atomic systems, and provides a wrapper around the LAMMPS MD software. This makes it possible to design calculations as single Python scripts that setup, perform and analyze one or more LAMMPS simulations. More information can be found at the atomman GitHub page: <https://github.com/usnistgov/atomman>

**Setup**

1. Install Python 2.7
   1. Using an Anaconda Python distribution is preferred as the scipy family of tools is heavily used.
   2. The package xmltodict needs to be manually installed beforehand. This can easily be done with the terminal command ‘pip install xmltodict’.
   3. All other requirements should come with Anaconda or will automatically install during step 4 (let me know if not true).
2. All of the files for iprPy can be found at:

* <https://github.com/usnistgov/iprPy> for the stable release.
* <https://github.com/lmhale99/iprPy> for the development version(s).

1. Download or clone the whole project to a local directory.
2. In a terminal, go into the iprPy root directory and add the iprPy package to Python with the command ‘python setup.py develop’.

**High-Throughput Scripts**

A number of scripts for running calculations in a high-throughput manner can be found in the ‘iprPy/high-throughput-scripts’ directory. This directory has subdirectories

* ‘prepare’ contains scripts that create multiple instances of different calculations
* ‘runner’ contains a script for automatically running the calculations, and other associated files.

prepare scripts

Each calculation currently has its own prepare script associated with it. Initially, one master prepare script was planned, but it ended up having too much overhead and complexity to effectively work.

All the prepare scripts are contained in subdirectories named after the calculations themselves. They can be executed by calling the script as a command with an input file as an argument. While each prepare script has its own unique combinatorial logic and input parameters, numerous utility functions are built into iprPy to allow for common feels and behaviors.

prepare input files

The input files are basic text files which define values for parameters associated with creating and running the calculations. The simple rules for the format are

1. Each line is read separately, and divided into whitespace delimited terms.
2. Blank lines are allowed.
3. Comments are allowed by starting terms with #. The # term and any subsequent terms on the line are ignored.
4. The first term in each line is a variable name.
5. All remaining terms are collected together as a complete value that is assigned to that variable name.
6. If a variable name is given, then a value must also be given.
7. Multiple lines for the same variable name are allowed, in which case the values are appended as a list.

Here is an example:

#This is a comment line

run\_directory /users/mydir/runtime

strain\_range 1e-5

strain\_range 1e-6

When read, only two variables will be created:

* run\_directory = ‘/users/mydir/runtime’
* strain\_range = [‘1e-5’, ‘1e-6’]

**High-Throughput Scripts, cont.**

Common input file variables

While the specific variables for each prepare script are different and can be interpreted in different ways, there are a few terms that are common.

**run\_directory** – path to the directory where the calculations that are being prepared are placed.

**lib\_directory** – path to the directory containing the library of calculation records.

**lammps\_command** – path to (or command for) the particular LAMMPS executable to use.

**mpi\_command –** full MPI command to use for running LAMMPS on multiple processors.

**length\_unit** – unit for length dimensions to use for input/output. Default value is angstrom.

**pressure\_unit** – unit for length dimensions to use for input/output. Default value is GPa.

**energy\_unit** – unit for length dimensions to use for input/output. Default value is eV.

**force\_unit** – unit for length dimensions to use for input/output. Default value is eV/angstrom.

More information on the terms used by each calculation can be found with the prepare script documentation.

runner

Once calculations have been prepared, they can be executed individually or through the use of the runner.py script. All runner.py needs is an input file listing run\_directory and lib\_directory variables and values. Here are some features of runner:

1. Each instance of runner picks a calculation in run\_directory at random and begins to execute it if no other runner has claimed it.
2. Automatically checks if results from another calculation are needed to perform the current calculation. If so, the runner tries to execute the required calculation first. NOTE: the current prepare scripts are NOT supporting this (yet).
3. If a calculation fails, the error message is collected without stopping the runner.
4. Results or errors are collected and saved into an xml record and saved to the lib\_directory record library.
5. The simulation directory is archived as a tar.gz file and moved to the lib\_directory.
6. Multiple runners can simultaneously work on calculations in the same run\_directory. They can also work on calculations in different run\_directories and feed results to the same lib\_directory.
7. When submitting on a cluster queue, each runner is assigned a particular number of cores to use. Therefore, running the calculations optimally can be done by collecting the jobs into run\_directories based on the number of processors used by the calculation.

**Calculation Scripts**

All calculation scripts are built with certain shared design characteristics. This is to help facilitate the development of new calculations and the associated prepare scripts.

The calculations are located in ‘iprPy/iprPy/calculations’. The calculations are placed here allowing them to be included when the iprPy package is imported in a Python script.

Calculation files

Each calculation is given its own directory, which contains the following files

* \_\_init\_\_.py – The Python init file that specifies that the directory is a submodule.
* calc\_\*.py – The Python calculation script.
* calc\_\*.template – A template version of the input file that the calculation script reads.
* Any other files that the calculation script needs to run properly.

Calculation script

The calculation scripts all follow the same basic design.

1. The calculation receives all variable input values from a specified input file. In other words, the calculation is executed with a command like “./calc\_\*.py input\_file.in”.
2. The function read\_input(f, UUID=None) reads the input file and interprets the values.
3. Calculation functions are called to perform the calculations, including functions that run LAMMPS simulations. Each calculation function should stand on its own (i.e. can be copied or imported to be used outside the calculation script.
4. Results are processed and passed to the function data\_model(input\_dict, results\_dict=None), which constructs a DataModelDict calculation record, then saved as a json/xml structured results file.

Common calculation functions

There are a few common functions defined for each calculation. These are accessible by importing the iprPy package, and support the construction of the prepare scripts.

* data\_model(input\_dict, results\_dict=None) – allows for complete and incomplete records to be constructed outside of the calculation.
* read\_input(f, UUID=None) – allows for input files to be interpreted outside the calculation script.
* template() – returns an open File object of the calc\_\*.template. Useful for prepare scripts that fill in this template.
* files() – returns a list of the absolute paths to the calc\_\*.py script and the other files required for properly running the calculation script. Useful for prepare scripts that copy these files.

Coming soon…