pyiron Documentation

Release 0.3.23

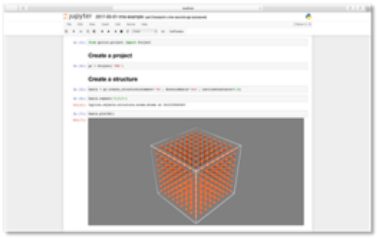
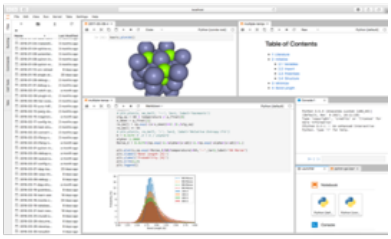
Max-Planck-Institut für Eisenforschung GmbH - Computational Ma

Jan 10, 2021

CONTENTS

pyiron - an integrated development environment (IDE) for computational materials science. It combines several tools in a common platform:

- Atomic structure objects – compatible to the Atomic Simulation Environment (ASE).
- Atomistic simulation codes – like LAMMPS and VASP.
- Feedback Loops – to construct dynamic simulation life cycles.
- Hierarchical data management – interfacing with storage resources like SQL and HDF5.
- Integrated visualization – based on NGLview.
- Interactive simulation protocols - based on Jupyter notebooks.
- Object oriented job management – for scaling complex simulation protocols from single jobs to high-throughput simulations.

pyiron (called pyron) is developed in the Computational Materials Design department of Joerg Neugebauer at the Max Planck Institut für Eisenforschung (Max Planck Institute for iron research). While its original focus was to provide a framework to develop and run complex simulation protocols as needed for ab initio thermodynamics it quickly evolved into a versatile tool to manage a wide variety of simulation tasks. In 2016 the Interdisciplinary Centre for Advanced Materials Simulation (ICAMS) joined the development of the framework with a specific focus on high throughput applications. In 2018 pyiron was released as open-source project.

Note: pyiron 0.X – Disclaimer: With the first open source release of pyiron under the BSD license we provide a fully functional core platform. We are currently working on finalizing various plugins, e.g. to enhance high throughput simulations, for Computational Phase Studies, and Electrochemistry and Corrosion. The code is published on Github.org, PyPi.org and Anaconda.org

---

CHAPTER

ONE

EXPLORE PYIRON

We provide various options to install, explore and run pyiron:

- Workstation Installation (recommeded): for Windows, Linux or Mac OS X workstations (interface for local VASP executable, support for the latest jupyterlab based GUI)
- Mybinder.org (beta): test pyiron directly in your browser (no VASP license, no visualization, only temporary data storage)
- Docker (for demonstration): requires Docker installation (no VASP license, only temporary data storage)

CHAPTER

TWO

JOIN THE DEVELOPMENT

Please contact us if you are interested in using pyiron:

- to interface your simulation code or method
- implementing high-throughput approaches based on atomistic codes
- to learn more about method development and Big Data in material science.

Please also check out the pyiron contributing guidelines

CHAPTER

THREE

CITING

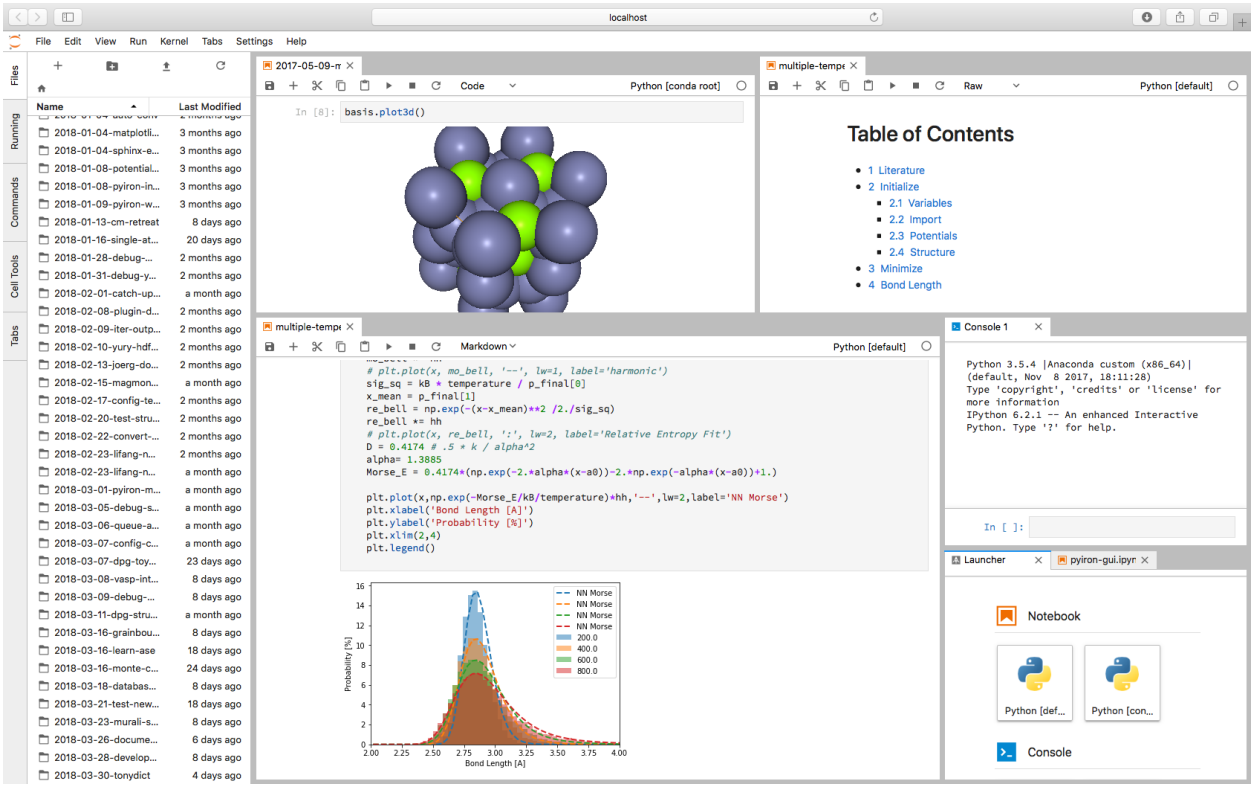
If you use pyiron in your research, please consider citing the following work:

```
@article{pyiron-paper,
  title = {pyiron: An integrated development environment for computational materials␣
↪science},
  journal = {Computational Materials Science},
  volume = {163},
  pages = {24 - 36},
  year = {2019},
  issn = {0927-0256},
  doi = {https://doi.org/10.1016/j.commatsci.2018.07.043},
  url = {http://www.sciencedirect.com/science/article/pii/S0927025618304786},
  author = {Jan Janssen and Sudarsan Surendralal and Yury Lysogorskiy and Mira␣
↪Todorova and Tilmann Hickel and Ralf Drautz and Jörg Neugebauer},
  keywords = {Modelling workflow, Integrated development environment, Complex␣
↪simulation protocols},
}
```

Read more about citing individual modules/ plugins of pyiron and the implemented simulation codes.

3.1 About

3.1.1 Introduction



pyiron is an integrated development environment for implementing, testing, and running simulations in computational materials science. It combines several tools in a common platform:

- Atomic structure objects – compatible to the Atomic Simulation Environment (ASE).
- Atomistic simulation codes – like LAMMPS and VASP.
- Feedback Loops – to construct dynamic simulation life cycles.
- Hierarchical data management – interfacing with storage resources like SQL and HDF5.
- Integrated visualization – based on NGLview.
- Interactive simulation protocols - based on Jupyter notebooks.
- Object oriented job management – for scaling complex simulation protocols from single jobs to high-throughput simulations.

pyiron (called pyron) is developed in the Computational Materials Design department of Joerg Neugebauer at the Max Planck Institut für Eisenforschung (Max Planck Institute for iron research). While its original focus was to provide a framework to develop and run complex simulation protocols as needed for ab initio thermodynamics it quickly evolved into a versatile tool to manage a wide variety of simulation tasks. In 2016 the Interdisciplinary Centre for Advanced Materials Simulation (ICAMS) joined the development of the framework with a specific focus on high throughput applications. In 2018 pyiron was released as open-source project.

3.1.2 Getting Help

Technical issues and bugs should be reported on Github all other questions can be asked on stackoverflow using the tag pyiron.

3.1.3 Release history

Release 0.2.0 (2018)

- Implement interactive interface to communicate with codes at runtime.

Release 0.1.0 (2018)

- opensource release - licensed under the BSD license.
- installation available on pip and anaconda.
- moved opensource repository to github.

Release 0.0.9 (2017)

- Name changed from PyIron to pyiron
- Fileoperations implemented (move, copy_to and remove).
- NGLview for visualisation.
- Atoms class speedup.
- Serial- and parallelmaster work with the cluster environment.
- Python 3.6 support added.

Release 0.0.8 (2016)

- Rewirte serial- and parallelmaster.
- Deprecated Qt environment in favor of jupyter.
- Python 3.5 support added.
- Use anaconda as recommended Python environment.
- Switch to Gitlab rather than subversion.

Release 0.0.5 (2015)

- Linux and Mac OS X support added.
- ASE compatible atom and atoms class.

Release 0.0.1 (2011)

- initial version named PyCMW

3.2 Installation

3.2.1 Conda Installation

The recommended way to install pyiron is via the conda package manager in a Linux environment. So if you are using Windows we recommend installing the Windows subsystem for Linux before you install pyiron and if you are on macOS X we recommend using a virtual machine/ virtual box. Native installations on both Windows and macOS X are possible but are restricted to molecular dynamics calculations with interatomic potentials and do not support density functional theory(DFT) codes. We collaborate with the open-source community at conda-forge to not only provide the pyiron package via their community channel, but also executables for compatible simulation codes like GPAW, LAMMPS and S/PHI/nX and their parameter files like pseudopotentials and interatomic potentials. To get started you can install pyiron using:

```
conda install -c conda-forge pyiron
```

Optional Dependencies

All the optional dependencies can also be installed via conda directly to simplify the setup of your simulation environment.

NGLview (Atomistic Structure Visualisation)

In pyiron we use the NGLview package to visualise atomistic structures directly in the jupyter notebook. To enable this feature, install NGLview:

```
conda install -c conda-forge nglview
```

In case you prefer jupyter lab over jupyter notebooks, you can also install NGLview for jupyter lab. This requires a few additional dependencies:

```
conda install -c conda-forge nodejs nglview
jupyter labextension install @jupyter-widgets/jupyterlab-manager --no-build
jupyter labextension install nglview-js-widgets
```

In addition to NGLview the first line also installs nodejs which is required to install your own jupyterlab plugins and rebuild jupyter lab. The following two lines install the jupyterlab extensions. Starting with the jupyterlab manager and followed by the NGLview javascript widget. During the installation of NGLview it is important to confirm that the NGLview version installed via conda is the same as the version of the NGLview javascript widget:

```
conda list nglview
jupyter labextension list
```

Supported simulation packages (quantum engines)

The following packages are supported to work out-of-the-box with pyiron, but must be installed independently either using conda or manual compilation. Manually compiled executables can be as much as 2-3x faster than conda-installed executables, and are therefore *strongly* recommended for high performance computing (HPC) usage. We discuss how to link any "homemade" executables to your pyiron installation in the advanced section.

LAMMPS (Molecular Dynamics with Interatomic Potentials)

LAMMPS stands for Large-scale Atomic/Molecular Massively Parallel Simulator and it is one of the most popular open-source molecular dynamics simulation codes for simulating solid-state materials (metals, semiconductors). As part of the pyiron project we maintain the conda package for LAMMPS to simplifiy its installation.

```
# serial + parallel, for linux and mac systems
conda install -c conda-forge lammps

# only serial (no python bindings), for native windows
conda install -c conda-forge -c pyiron lammps
```

On the conda-forge channel we provide LAMMPS executables for both serial and parallel (MPI) execution as well as their respective python bindings. The LAMMPS version on the pyiron channel is for native windows installations only and it is limited to serial execution with no Python bindings. We therefore highly recommend using the Linux subsystem for Windows rather than the native Windows installation.

S/PHI/nX (Density Functional Theory)

The S/PHI/nX DFT code is an open-source DFT code developed in close collaboration with the pyiron developers, therefore it is the recommended DFT code to be used with pyiron. The applications of S/PHI/nX range from constrained magnetic calculations to charged defects which makes it suitable for ab initio thermodynamics and beyond. The S/PHI/nX DFT code is only officially supported for Linux, so we recommend the use of a Linux subsystem (on Windows) or a virtual machine (on mac).

```
conda install -c conda-forge sphinxdft
```

GPAW (Density Functional Theory)

pyiron also supports GPAW, an open-source realspace DFT simulation code which is popular because of its Python bindings which allow accessing parameters of the DFT code during the run time. GPAW can be installed on Linux directly via conda:

```
conda install -c conda-forge gpaw
```

Additional simulation packages

SQSgenerator

The sqsgenerator is command line tool written in Python/Cython for finding optimized SQS structures. It is available as a separate conda package, once it is installed pyiron is able to use it inside pyiron simulation protocols without any additional imports:

```
conda install -c conda-forge sqsgenerator
```

3.2.2 Advanced Configuration

While the conda-based installation is usually sufficient for workstation installations to get started with pyiron, it can be extended to support your own executables, include your own parameter files, support commercial codes like VASP or updating the database performance by switching from SQLite to PostgreSQL.

Custom Executables and Parameter Files

pyiron can either be configured using a configuration file named `~/.pyiron` located in the user's home directory or by specifying environment variables. The options are similar either way, so we start with the configuration file. The default configuration file pyiron assumes if it does not find a configuration file is:

```
[DEFAULT]
PROJECT_CHECK_ENABLED = False
FILE = ~/pyiron.db
RESOURCE_PATHS = ${CONDA_PREFIX}/share/pyiron
```

The first line `[DEFAULT]` defines the current configuration to overwrite the default configuration. The second line `PROJECT_CHECK_ENABLED` disables the project check which enables pyiron to write to the whole file system. The third lines defines the object index to be stored in an SQLite database file `FILE` which is located in the home directory `~/pyiron.db`. It is important to copy the database in case you change the configuration otherwise existing calculation are lost. Finally the `RESOURCE_PATHS` provides the path to the parameter files. Inside pyiron you can check the current configuration using:

```
from pyiron_base import Settings
s = Settings()
s._configuration
```

Below, the individual options are explained one by one:

- the `[DEFAULT]` option defines the current `~/.pyiron` configuration to overwrite the default configuration.
- the `RESOURCE_PATHS` option defines the resource path is a list of `;` separated paths where pyiron checks for resource files. A template of such a resource directory is available on github and it can be downloaded as an archive from the release page. We recommend to create a folder `~/pyiron/resources` and store the parameter files and links to the executables there. The links are basically shell scripts which can be modified to load modules. By default the conda path is added, therefore there is no need to add it manually.
- the `PROJECT_PATHS` option is similar to the resource path but for storing simulation protocols rather than parameter files. When the `PROJECT_CHECK_ENABLED` option is set to `true` then the read and write access within pyiron is limited to the directories defined in the `PROJECT_PATHS`. Again multiple directories can be separated by `;`. An alternative but outdated name for this option is `TOP_LEVEL_DIRS`.

Besides the general variables in the ~/.pyiron configuration, the other settings are used to define the database connection. More detailed examples about the configuration can be found below; for now we continue with the configuration of the database. pyiron can use a database to build an index of the HDF5 files on the file system which accelerates job analysis. By default pyiron uses an SQLite database for this index, but the database can also be disabled or a PostgreSQL database can be used to improve performance.

- By default the database is defined by the FILE option which is equal to the DATABASE_FILE option and gives the path to the SQLite database file. As the SQLite database is a file-based database, it struggles with parallel access on a shared file system (common for HPC clusters).
- To address this limitation it is possible to disable the database on HPC clusters using the DISABLE_DATABASE option by setting it to true. This is commonly used when the calculations are only executed on the remote cluster but the analysis is done on a local workstation or a group server which supports an SQL-based database.
- The other database options, namely TYPE, HOST, NAME, USER, PASSWD and JOB_TABLE define the connection details to connect to a PostgreSQL database. Inside pyiron sqlalchemy is used to support different SQL-based databases, therefore it is also possible to provide the sqlalchemy connection string directly as CONNECTION.
- Finally some pyiron installations use a group management component which is currently in development. They might have additional options in their ~/.pyiron configuration to enable sharing calculations between different users. These options are VIEWERUSER, VIEWERPASSWD and VIEWER_TABLE. As this is a development feature it is not yet fully documented. Basically those are the access details for the global database viewer, which can read the database entries of all users. With this configuration it is possible to load jobs of other users.

In analogy to the ~/.pyiron configuration file pyiron also supports using environment variables to configure the pyiron installation. The available environment variables are:

- the PYIRONCONFIG environment variable defines the location of the .pyiron configuration file.
- the PYIRONRESOURCEPATHS environment variable defines the RESOURCE_PATHS option.
- the PYIRONPROJECTPATHS environment variable defines the PROJECT_PATHS option.
- the PYIRONPROJECTCHECKENABLED environment variable defines the PROJECT_CHECK_ENABLED option.
- the PYIRONDISABLE environment variable defines the DISABLE_DATABASE option.
- the PYIRONSQLTYPE, PYIRONSQLFILE, PYIRONSQHOST, PYIRONSQLDATABASE, PYIRONUSER and PYIRONSQLUSERKEY environment varaibles define the SQL database connection and can also be summarized in the PYIRONSQLCONNECTIONSTRING environment variable.
- the PYIRONSQLVIEWTABLENAME, PYIRONSQLVIEWUSER and PYIRONSQLVIEWUSERKEY environment variables define the SQL viewer connection and can also be summarized in the PYIRONSQLVIEWCONNECTIONSTRING environment variable.

To further explain the usage of the different parameters, we discuss common use cases in the following:

Use your own Executable for LAMMPS/ S/PHI/nX or GPAW

To add your own executables or parameter files it is necessary to initialise a user-defined configuration ~/.pyiron. You can start with a basic configuration like:

```
[DEFAULT]
FILE = ~/pyiron.db
PROJECT_PATHS = ~/pyiron/projects
RESOURCE_PATHS = ~/pyiron/resources
```

In this case pyiron can only execute calculations in the `~/pyiron/projects` directory. pyiron can't delete files outside this directory. Next to the projects directory `~/pyiron/projects` we create a resource directory `~/pyiron/resources` to store links to the executables and the corresponding parameter files. Both directories have to be created by the user and in case no `FILE` option is defined pyiron by default creates an SQLite database in the resource directory. Example resource directories are available on Github . Here we just discuss the LAMMPS resource directory as one example.

```
resources/
  lammps/
    bin/
      run_lammps_2020.03.03.sh
      run_lammps_2020.03.03_mpi.sh
    potentials/
      potentials_lammps.csv
```

The resource directory contains two sub folders `bin` which includes links to the executables and `potentials` which includes links to the interatomic potentials. The links to the executables are shell script which follow the naming convention `run_<code name>_<version>(_<tag>).sh` the `mpi` tag is used to indicate the MPI-enabled executables. If we take a look at the `run_lammps_2020.03.03_mpi.sh` shell script, it contains the following lines:

```
#!/bin/bash
mpiexec -n $1 --oversubscribe lmp_mpi -in control.inp;
```

Scripts with the `mpi` tag are called with two parameters the first being the number of cores the second the number of threads, while regular shell scripts do not get any input parameters. By using shell scripts it is easy to link existing executables which might require loading specific modules or setting environment variables. In the same way the parameter files for pyiron are stored in the csv format which makes them human editable. For shared installations we recommend storing the pyiron resources in a shared directory.

Configure VASP

The Vienna Ab initio Simulation Package is a popular commercial DFT code which is commonly used for large DFT calculations or high-throughput studies. pyiron implements a VASP wrapper but does not provide a VASP license. Therefore users have to compile their own VASP executable and provide their own VASP pseudopotentials (included with the VASP license). An example configuration for VASP in pyiron is available on Github:

```
resources/
  vasp/
    bin/
      run_vasp_5.4.4_default.sh
      run_vasp_5.4.4_default_mpi.sh
    potentials/
      potpaw/
      potpaw_PBE/
      potentials_vasp.csv
      potentials_vasp_lda_default.csv
      potentials_vasp_pbe_default.csv
```

Similar to the LAMMPS resource directory discussed above the VASP resource directory also contains a `bin` diirectory and a `potentials` directory. By adding the `default` tag we can set the default executable, in particular when compiling multiple variants of the same VASP version. Finally the directories `potpaw` and `potpaw_PBE` contain the VASP pseudopotentials, which are included with the VASP license and have to be added by the user.

PostgreSQL Database

To accelerate the pyiron installation it is recommended to use a PostgreSQL database rather than the default SQLite database. To configure the database server, the following options can be added to the `~/.pyiron`:

- `TYPE` the typ of the database, while sqlalchemy supports a wide range of differnet databases PostgreSQL is recommended and can be selected by setting the type to `Postgres`.
- `HOST` the database host where the database is running.
- `NAME` the name of the database.
- `USER` the database user, in contrast to many other software packages pyiron requires one database user per system user who is using pyiron. The database is only used to store an index of the calculations executed with pyiron, therefore knowledge gained from accessing the database is limited unless the user has also access to the file system.
- `PASSWD` the database user password. While it is a bad practice to store the database password in the configuration file, the database only contains the the job index. Still it is important that the user creates an pyiron specific password and should never store their system user password in the `.pyiron` configuration file.
- `JOB_TABLE` the name of the database table. pyiron is commonly using one table per user.

A typical `.pyiron` configuration with a PostgreSQL database might look like this:

```
[DEFAULT]
TYPE = Postgres
HOST = database.hpc-cluster.university.edu
NAME = pyiron
USER = janj
PASSWD = **********
JOB_TABLE = jobs_janj
PROJECT_PATHS = ~/pyiron/projects
RESOURCE_PATHS = ~/pyiron/resources
```

Be careful when updating the database configuration as pyiron does not transfer the content of the database automatically.

Remote HPC Cluster

While the previous section discussed the installation of pyiron on a local workstation, the following section discusses how to configure a remote HPC cluster to transfer jobs to the HPC cluser for execution and back for analysis. For setting up pyiron on an HPC cluster there are basically three different configurations available:

- Install pyiron on the HPC cluster, with jupyterhub running as a central service on the login node using the sudospawner to authorize users. In this configuration the user only needs a web browser and all simulation results will remain on the HPC cluster. The limitation of this approach is that both the global PostgreSQL database as well as the jupyterhub have to be running on the cluster with the PostgreSQL database being accessible from all compute nodes.
- The second configuration is running pyiron on the HPC without the jupyterhub or a database, and storing the simulation results on a group server. Servers in the research group are commonly less strictly governed, so installing the jupyterhub on the group server as well as the PostgreSQL database for faster data analysis should be possible in most cases. From the user perspective the setup still only requires a web browser on the user's end device, and leaves the task of backing up the simulation data on the group server side rather than the end-user.
- Finally the third configuration is the workstation installation, with a PostgreSQL database or even just a SQLite file based database with using the HPC cluster only to execute the calculation and copying the simulation results to local workstation after every calculation.

We start by explaining the first configuration and then build on top of this setup to add the remote transfer capabilities.

HPC Cluster with PostgreSQL Database and Jupyterhub

The `~/.pyiron` is structured just like a workstation installation with a PostgreSQL database as explained above. In addition to the previous resource directories we add another subfolder in the resource directory to configure the queuing system using pysqa as queuing system adapter. pysqa is based on the idea of using shell script based templates to configure the different queues as modern queuing sytem provide a wide range of settings but most users commonly submit their jobs with very similar settings. We discuss a sample configuration for SLURM sample configurations for other queuing systems are available on Github.

```
resources/
  queues/
    queue_1.sh
    queue_2.sh
    queue.yaml
```

The queues directory contains one `queue.yaml` configuration file and multiple jinja based shell script templates for submitting jobs. These templates define a commonly used set of parameters used to submit calculations, it can contain a restriction on a specific queue or partition but it does not have to. A typical queue template that might be used in `queue_1.sh` and `queue_2.sh` is shown below:

```
#!/bin/bash
#SBATCH --output=time.out
#SBATCH --job-name={{job_name}}
#SBATCH --workdir={{working_directory}}
#SBATCH --get-user-env=L
#SBATCH --partition=slurm
{%- if run_time_max %}
#SBATCH --time={{run_time_max // 60}}
{%- endif %}
{%- if memory_max %}
#SBATCH --mem={{memory_max}}
{%- endif %}
#SBATCH --cpus-per-task={{cores}}

{{command}}
```

Such a template contains the variables `{{job_name}}` which is used to identify the job on the queuing system. Typically, pyiron job names are constructed using the prefix `pi` followed by the pyiron job id. This allows pyiron to match the job on the queuing system with the job table. The second option is the `{{working_directory}}` which is the directory where the job is located and the simulation code is executed. For pyiron this is typically a subdirectory of the simulation protocol to simplify identifiying broken calculation on the filesystem. The third option is the `run_time` which specifies the run time in seconds, followed by the `memory_max` which specifies the memory requirement of a given calculation. Both parameters are optional. Finally the `cores` defines the number of CPU cores used for a calculation and the `command` parameter is set by pyiron to load a pyiron object during the execution. When a pyiron job is executed on a compute node, a python process is first called to reload the pyiron object and then the pyiron object calls the shell script just like a regular job executed on the login node. By initially calling a python process, pyiron is able to track the progress of the calculation.

Besides the queue templates, the queues directory also contains the queue configuration `queue.yaml`:

```
queue_type: SLURM
queue_primary: queue_one
queues:
```

(continues on next page)

(continued from previous page)

```
  queue_one: {cores_max: 40, cores_min: 1, run_time_max: 3600, script: queue_1.sh}
  queue_two: {cores_max: 1200, cores_min: 40, run_time_max: 345600, script: queue_2.
↪sh}
```

The queue configuration defines the limits of the individual queues which helps the user to select the appropriate queue for their simulation. The `queue_type` defines the type of the queuing system, the `queue_primary` defines the primary queue and finally `queues` defines the available queues. Typically each queue is associated with a shell script template, like in this case `queue_one` is associated with `queue_1.sh` and `queue_two` is associated with `queue_2.sh`. Additional queue configuration templates are available on Github.

Submit to Remote HPC

Submitting calculations to a remote HPC requires some light configuration. On the HPC, disable the database in the `.pyiron` with the following lines:

```
[DEFAULT]
DISABLE_DATABASE = True
PROJECT_PATHS = ~/pyiron/projects
RESOURCE_PATHS = ~/pyiron/resources
```

Then configure the remote HPC just like a regular HPC by adding the queuing system configuration as described above. It is recommended to test the submission on the remote HPC before configuring the datatransfer.

On the system that will be used to submit calculations to the remote HPC (e.g. your laptop or an in-between login machine), create the queues directory in the resource path, containing only the queue configuration:

```
resources/
  queues/
    queue.yaml
```

This queue configuration now includes additional options to handle the SSH connection to the remote cluster:

```
queue_type: REMOTE
queue_primary: queue_one
ssh_host: hpc-cluster.university.edu
ssh_username: janj
known_hosts: ~/.ssh/known_hosts
ssh_key: ~/.ssh/id_rsa
ssh_remote_config_dir: /u/share/pyiron/resources/queues/
ssh_remote_path: /u/janj/remote/
ssh_local_path: /home/jan/pyiron/projects/
ssh_continous_connection: True
queues:
  queue_one: {cores_max: 40, cores_min: 1, run_time_max: 3600}
  queue_two: {cores_max: 1200, cores_min: 40, run_time_max: 345600}
```

The `ssh_host` defines the name of the login node, with `ssh_username` the user on the remote machine and `known_hosts` and `ssh_key` the local configuration files to connect to the remote host. Currently pyiron only supports ssh key based authentification for remote calculation. By setting `ssh_continous_connection`, the same connection is reused for data transfers which is commonly more efficient than creating individual connections for each command. Still, this assumes that the connection between the workstation or group server and the remote HPC cluster is stable. If this is not the case - for example, when using a mobile connection - it is recommended to disable this option. The `ssh_remote_config_dir` defines the configuration of the queuing system on the remote cluster. Finally the calculations are copied from the local directory `ssh_local_path` to the remote directory

`ssh_remote_path`. In the above example, if a calculation is submitted in the directory `/home/jan/pyiron/projects/first/subproject` then the files are copied to `/u/janj/remote/first/subproject`. By retaining the path when transfering the files it is easier to debug failed calculations. Finally the queues are defined locally to have quick access to the queue configurations, but it is not necessary to define the submission templates as those are available on the remote machine. In addition the other resources have to be identical on both systems. The easiest way to achieve this is to copy the resource directory once the installation is working on the remote machine.

Submit to multiple Remote HPC Clusters

Finally pyiron also supports configuring multiple HPC clusters. In this case rather than creating a `queue.yaml` file in the queues resource directory we create a `clusters.yaml` file with the following content:

```
cluster_primary: cluster_one
cluster:
  cluster_one: cluster_1.yaml
  cluster_two: cluster_2.yaml
```

The `cluster_primary` defines the default cluster and the different clusters are each defined in their own `cluster_*.yaml` file. Those `cluster_*.yaml` have the same structure as the `queue.yaml` file discussed above, but they cannot be named `queue.yaml` as pyiron otherwise assumes that only one cluster is available.

3.2.3 Alternative Installation Options

So far we discussed the installation of pyiron on an individual workstation via conda or on a HPC cluster. In the following we focus on developer-specific setups to install pyiron directly from its source. It is recommended to start with a conda installation and then replace only the pyiron version so that conda can still automatically manage all dependencies/environment settings for you. In case this is not possible, e.g. if conda is not allowed on your HPC cluster, then pyiron can be installed directly from the source code.

Install from Source

For development, it is recommended to first create a conda environment containing all of pyiron's dependencies. The dependencies are available in pyiron's environment.yml file.

If conda is not available on your machine, the next best thing would be to install pyiron and its dependencies via pip.

Using pip

The default installation via pip installs the latest release version of pyiron. So in case your HPC cluster does not support installing pyiron via conda you can install this release version via pip and then continue with the setup of your remote HPC cluster as described above.

```
pip install pyiron
```

For those who want to test the nightly releases of pyiron which include the latest status of the master branch you can install those via pip as well:

```
pip install --pre pyiron
```

Using git

To get the latest pyiron version and access changes on development branches pyiron can also be installed via git. For example you can download the pyiron sourcecode to `~/pyiron/software` using:

```
git clone https://github.com/pyiron/pyiron.git ~/pyiron/software
```

Based on the previous workstation setup your `~/pyiron` directory should contain the following folders:

```
pyiron/
  projects/
  resources/
  software/
```

To include this version in your `PYTHONPATH` add the following line to your `~/.profile` or `~/.bashrc` configuration:

```
export PYTHONPATH=${HOME}/pyiron/software/:${PYTHONPATH}
```

When you import pyiron in any python shell or jupyter notebook it should load the version from `~/pyrion/software`. Finally you can switch to other branches using git:

```
git checkout -b master
```

In this case we switch to the master branch.

Download pyiron Parameter Files

For source code based installations it is also possible to download the pyiron resources directly from within pyiron. Simply open a python shell and import pyiron:

```
> import pyiron
> pyiron.install()
>>> It appears that pyiron is not yet configured, do you want to create a default␣
↪start configuration (recommended: yes). [yes/no]:
> yes
> exit()
```

This command does the following steps in the background:

- Create a `~/.pyiron` config file – with the default settings (for simple installations)
- Create a `~/pyiron/projects` directory – pyiron can only execute calculations within this project directory to prevent any interference with other tools or simulation management solutions.
- Create a `~/pyiron/resources` directory – this directory includes the link to the executables and potentials, sorted by code.

3.2.4 Demonstration and Training Environments

For workshops, tutorials, and lectures it is sometimes necessary to setup multiple computers with very similar configurations, and - depending on the conference location - internet access might be limited. For these cases pyiron provides setup instructions for demonstration and training environments.

Cloud Solutions

You can test pyiron on Mybinder.org (beta), without the need for a local installation. It is a flexible way to get a first impression of pyiron but it does not provide any permanent storage by default. Loading the pyiron environment on mybinder can take 5 to 15 minutes in case a new docker container needs to be built. Mybinder is a free service, so sessions on its servers are limited in duration and memory limits, and their stability is not guaranteed. We recommend having a backup plan when using mybinder for presentations/interactive tutorials, since the mybinder instance might be shutdown if it is idle for too long.

Docker Container

For demonstration purposes we provide Docker containers on Dockerhub these can be downloaded and executed locally once docker is installed. Again, these container images do not provide any permanent storage, so all information is lost once the docker container is shut down. To download the docker container use:

```
docker pull pyiron/pyiron:latest
```

After downloading the docker container you can use it either with jupyter notebook:

```
docker run -i -t -p 8888:8888 pyiron/pyiron /bin/bash -c "source /srv/conda/envs/
↪notebook/bin/activate; jupyter notebook --notebook-dir=/home/pyiron/ --ip='*' --
↪port=8888"
```

or with jupyter lab:

```
docker run -i -t -p 8888:8888 pyiron/pyiron /bin/bash -c "source /srv/conda/envs/
↪notebook/bin/activate; jupyter lab --notebook-dir=/home/pyiron/ --ip='*' --port=8888
↪"
```

After the run command the following line is displayed. Copy/paste this URL into your browser when you connect for the first time, to login with a token:

```
http://localhost:8888/?token=<your_token>
```

Open the link with your personal jupyter token `<your_token>` in the browser of your choice. Just like the Binder image, the Docker image comes with several pyiron examples preinstalled.

Install Utility

To setup a local lab with pyiron when the internet connection is limited, we provide a classical installer for Windows, macOS X and Linux which is based on the conda constructor. If you do not have anaconda installed you can download this installer and get started with just a single download.

3.2.5 Getting Started

Finally once you have installed pyiron you can quickly test your installation with the following minimalistic example. Many more examples are available in the Github repository.

First Calculation

After the successful configuration you can start your first pyiron calculation. Navigate to the the projects directory and start a jupyter notebook or jupyter lab session correspondingly:

```
cd ~/pyiron/projects
jupyter notebook
```

or

```
cd ~/pyiron/projects
jupyter lab
```

Open a new jupyter notebook and inside the notebook you can now validate your pyiron calculation by creating a test project, setting up an initial structure of bcc Fe, and visualising it using NGLview.

```
from pyiron import Project
pr = Project('test')
basis = pr.create_structure('Fe', 'bcc', 2.78)
basis.plot3d()
```

Finally a first lammps calculation can be executed by:

```
ham = pr.create_job(pr.job_type.Lammps, 'lammpstestjob')
ham.structure = basis
ham.potential = ham.list_potentials()[0]
ham.run()
```

Next Steps

To get a better overview of all the available functionality inside pyiron we recommend the examples provided in the examples section - *Tutorials*.

3.3 Tutorials

3.3.1 First steps through pyiron

This section gives a brief introduction about fundamental concepts of pyiron and how they can be used to setup, run and analyze atomic simulations. As a first step we import the libraries numpy for data analysis and matplotlib for visualization.

[1]:
```
import numpy as np
%matplotlib inline
import matplotlib.pylab as plt
```

To import pyiron simply use:

[2]:
```
from pyiron import Project
```

The Project object introduced below is central in pyiron. It allows to name the project as well as to derive all other objects such as structures, jobs etc. without having to import them. Thus, by code completion *Tab* the respective commands can be found easily.

We now create a pyiron Project named 'first_steps'.

[3]:
```
pr = Project(path='first_steps')
```

The project name also applies for the directory that is created for the project.

Perform a LAMMPS MD simulation

Having created an instance of the pyiron Project we now perform a LAMMPS molecular dynamics simulation.

For this basic simulation example we construct an fcc Al crystal in a cubic supercell (`cubic=True`). For more details on generating structures, please have a look at our *structures example*

[4]:
```
basis = pr.create_ase_bulk('Al', cubic=True)
supercell_3x3x3 = basis.repeat([3, 3, 3])
supercell_3x3x3.plot3d()
```

```
NGLWidget()
```

Here `create_ase_bulk` uses the ASE bulk module. The structure can be modified - here we extend the original cell to a 3x3x3 supercell (`repeat([3, 3, 3])`). Finally, we plot the structure using NGlview.

The project object allows to create various simulation job types. Here, we create a LAMMPS job.

[5]:
```
job = pr.create_job(job_type=pr.job_type.Lammps, job_name='Al_T800K')
```

Further, we specify a Molecular Dynamics simulation at $T = 800$ K using the supercell structure created above.

[6]:
```
job.structure = supercell_3x3x3
job.calc_md(temperature=800, pressure=0, n_ionic_steps=10000)
```

To see all available interatomic potentials which are compatible with the structure (for our example they must contain Al) and the job type (here LAMMPS) we call `job.list_potentials()`.

[7]:
```
job.list_potentials()
```

[7]:
```
['Al_Mg_Mendelev_eam', 'Zope_Ti_Al_2003_eam', 'Al_H_Ni_Angelo_eam']
```

From the above let us select the first potential in the list.

[8]:
```
pot = job.list_potentials()[0]
print ('Selected potential: ', pot)
job.potential = pot
```

```
Selected potential:  Al_Mg_Mendelev_eam
```

To run the LAMMPS simulation (locally) we now simply use:

[9]:
```
job.run()
```

Analyze the calculation

After the simulation has finished the information about the job can be accessed through the Project object.

[10]:
```
job = pr['Al_T800K']
job
```

[10]:
```
{'groups': ['input', 'output'], 'nodes': ['NAME', 'server', 'VERSION', 'TYPE']}
```

Printing the job object (note that in Jupyter we don't have to call a print statement if the variable/object is in the last line). The output lists the variables (nodes) and the directories (groups). To get a list of all variables stored in the generic output we type:

[11]:
```
job['output/generic']
```

[11]:
```
{'groups': [], 'nodes': ['temperatures', 'positions', 'steps', 'forces', 'energy_pot',
↪ 'energy_tot', 'volume', 'cells', 'pressures', 'unwrapped_positions', 'time']}
```

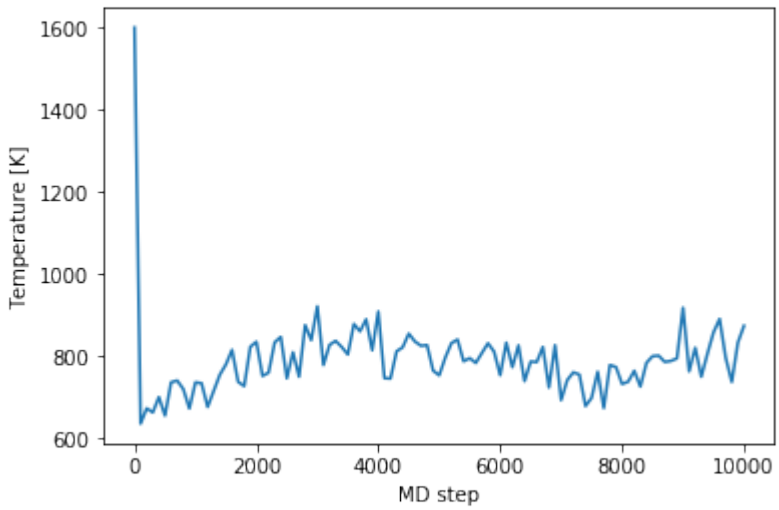
An animated 3d plot of the MD trajectories is created by:

[12]:
```
job.animate_structure()
```

```
NGLWidget(count=101)
```

To analyze the temperature evolution we plot it as function of the MD step.

[13]:
```
temperatures = job['output/generic/temperature']
steps = job['output/generic/steps']
plt.plot(steps, temperatures)
plt.xlabel('MD step')
plt.ylabel('Temperature [K]');
```



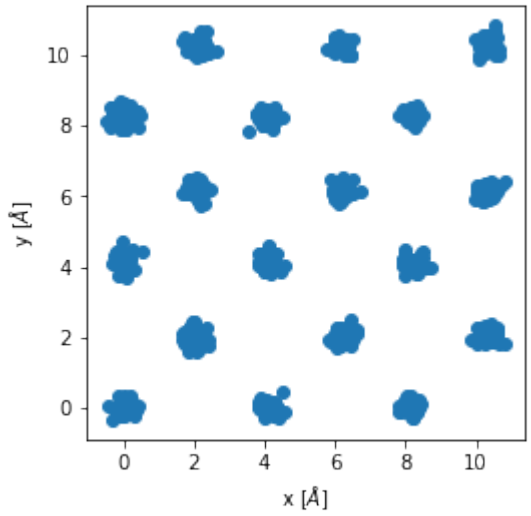
In the same way we can plot the trajectories.

[14]:
```
pos = job['output/generic/positions']
x, y, z = [pos[:, :, i] for i in range(3)]
sel = np.abs(z) < 0.1
fig, axs = plt.subplots(1,1)
axs.scatter(x[sel], y[sel])
axs.set_xlabel('x [$\AA$]')
```

(continues on next page)

(continued from previous page)

```
axs.set_ylabel('y [$\AA$]')
axs.set_aspect('equal', 'box');
```



Perform a series of jobs

To run the MD simulation for various temperatures we can simply loop over the desired temperature values.

[15]:
```
for temperature in np.arange(200, 1200, 200):
    job = pr.create_job(pr.job_type.Lammps,
                        'Al_T{}K'.format(int(temperature)))
    job.structure = supercell_3x3x3
    job.potential = pot
    job.calc_md(temperature=temperature,
                pressure=0,
                n_ionic_steps=10000)
    job.run()
```

To inspect the list of jobs in our current project we type (note that the existing job from the previous excercise at $T = 800$ K has been recognized and not run again):

[16]:
```
pr
```

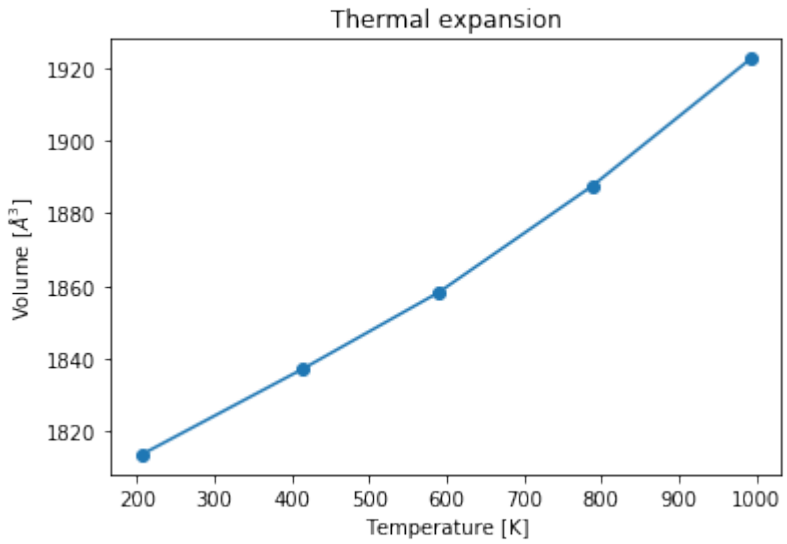
[16]:
```
['Al_T600K', 'Al_T800K', 'Al_T1000K', 'Al_T200K', 'Al_T400K']
```

We can now iterate over the jobs and extract volume and mean temperature.

[17]:
```
vol_lst, temp_lst = [], []
for job in pr.iter_jobs(convert_to_object=False):
    volumes = job['output/generic/volume']
    temperatures = job['output/generic/temperature']
    temp_lst.append(np.mean(temperatures[:-20]))
    vol_lst.append(np.mean(volumes[:-20]))
```

Then we can use the extracted information to plot the thermal expansion, calculated within the NPT ensemble. For plotting the temperature values in ascending order the volume list is mapped to the sorted temperature list.

[18]:
```
plt.figure()
vol_lst[:] = [vol_lst[np.argsort(temp_lst)[k]]
              for k in range(len(vol_lst))]
plt.plot(sorted(temp_lst), vol_lst,
         linestyle='-',marker='o',)
plt.title('Thermal expansion')
plt.xlabel('Temperature [K]')
plt.ylabel('Volume [$\AA^3$]');
```



Create a series of projects

We extend the previous example and compute the thermal expansion for three of the available aluminum potentials. First, let us create a new pyiron project named 'Al_potentials'. We can use the information of the previously run job 'Al_T200K' of the 'first_steps' project to find all the compatible potentials.

[19]:
```
pr = Project('Al_potentials')
pot_lst = pr['../first_steps/Al_T200K'].load_object().list_potentials()[:3]
```

[ ]:
```
```

[20]:
```
pot_lst
```

[20]:
```
['Al_Mg_Mendelev_eam', 'Zope_Ti_Al_2003_eam', 'Al_H_Ni_Angelo_eam']
```

Note again that `list_potentials()` automatically only returns the potentials that are compatible with the structure (chemical species) and the job type.

We can now loop over the selected potentials and run the MD simulation for the desired temperature values for any of the potentials.

[21]:
```
for pot in pot_lst:
    print ('Interatomic potential used: ',pot)
    pr_pot = pr.create_group(pot)
    for temperature in np.arange(200, 1200, 200):
        job = pr_pot.create_job(pr.job_type.Lammps,
                                'Al_T{}K'.format(int(temperature)))
```

(continues on next page)

(continued from previous page)

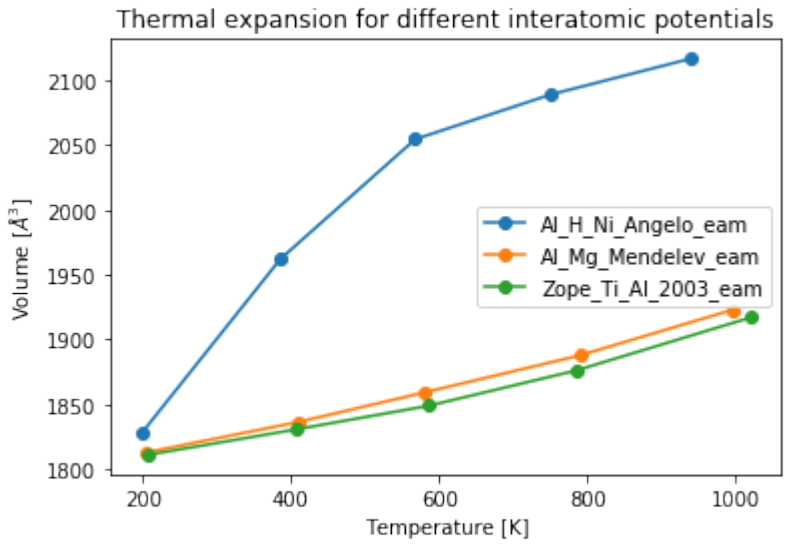
```
        job.structure = supercell_3x3x3
        job.potential = pot
        job.calc_md(temperature=temperature,
                    pressure=0,
                    n_ionic_steps=10000)
        job.run()
```

```
Interatomic potential used:  Al_Mg_Mendelev_eam
Interatomic potential used:  Zope_Ti_Al_2003_eam
Interatomic potential used:  Al_H_Ni_Angelo_eam
```

With the `pr.create_group()` command a new subproject (directory) is created named here by the name of the potential.

For any particular potential the thermal expansion data can be obtained again by looping over the jobs performed using that potential. To obtain the thermal expansion curves for all the potentials used we can simply iterate over the subprojects (directories) created above by using the `pr.iter_groups()` command.

[22]:
```
for p in pr.iter_groups():
    vol_lst, temp_lst = [], []
    for out in p.iter_jobs(path='output/generic'):
        volumes = out['volume']
        temperatures = out['temperature']
        temp_lst.append(np.mean(temperatures[:-20]))
        vol_lst.append(np.mean(volumes[:-20]))
    # Plot only if there is a job in that group
    if len(p.get_job_ids()) > 0:
        plt.plot(temp_lst, vol_lst,
                 linestyle='-',marker='o',
                 label=p.name)
plt.legend(loc='best')
plt.title('Thermal expansion for different interatomic potentials')
plt.xlabel('Temperature [K]')
plt.ylabel('Volume [$\AA^3$]');
```



[ ]:

3.3.2 Energy volume curve

Theory

Fitting the energy volume curve allows to calculate the equilibrium energy E_0, the equilirbium volume V_0, the equilibrium bulk modulus B_0 and its derivative $B_0^{'}$. These quantities can then be used as part of the Einstein model to get an initial prediction for the thermodynamik properties, the heat capacity C_v and the free energy F.

Initialisation

We start by importing matplotlib, numpy and the pyiron project class.

[1]:
```
%matplotlib inline
import matplotlib.pyplot as plt
import numpy as np
from pyiron import Project
```

In the next step we create a project, by specifying the name of the project.

[2]:
```
pr = Project(path='thermo')
```

Atomistic structure

To analyse the energy volume dependence a single super cell is sufficient, so we create an iron super cell as an example.

[3]:
```
basis = pr.create_structure(element='Fe', bravais_basis='bcc', lattice_constant=2.75)
basis.plot3d()
```

```
NGLWidget()
```

Calculation

Energy volume curves are commonly calculated with ab initio codes, so we use VASP in this example. But we focus on the generic commands so the same example works with any DFT code. We choose 'vasp' as job name prefix, select an energy cut off of $320eV$ and assign the basis to the job. Afterwards we apply the corresponding strain.

[4]:
```
for strain in np.linspace(0.95, 1.05, 7):
    strain_str = str(strain).replace('.', '_')
    job_vasp_strain = pr.create_job(job_type=pr.job_type.Gpaw, job_name='gpaw_' +␣
↪strain_str)
    job_vasp_strain.set_encut(320.0)
    job_vasp_strain.structure = basis.copy()
    job_vasp_strain.structure.set_cell(cell=basis.cell * strain ** (1/3), scale_
↪atoms=True)
    job_vasp_strain.run()
```

```
The job gpaw_0_95 was saved and received the ID: 1
The job gpaw_0_9666666666666667 was saved and received the ID: 2
The job gpaw_0_9833333333333333 was saved and received the ID: 3
The job gpaw_1_0 was saved and received the ID: 4
The job gpaw_1_0166666666666666 was saved and received the ID: 5
The job gpaw_1_0333333333333334 was saved and received the ID: 6
The job gpaw_1_05 was saved and received the ID: 7
```

As these are simple calculation, there is no need to submit them to the queuing sytem. We can confirm the status of the calculation with the job_table. If the status of each job is marked as finished, then we can continue with the next step.

[5]:
```
pr.job_table()
```

[5]:
```
   id    status chemicalformula                    job  \
0   1  finished            None               gpaw_0_95
1   2  finished            None  gpaw_0_9666666666666667
2   3  finished            None  gpaw_0_9833333333333333
3   4  finished            None                gpaw_1_0
4   5  finished            None  gpaw_1_0166666666666666
5   6  finished            None  gpaw_1_0333333333333334
6   7  finished            None               gpaw_1_05

                    subjob projectpath              project  \
0               /gpaw_0_95        None  /home/jovyan/thermo/
1  /gpaw_0_9666666666666667        None  /home/jovyan/thermo/
2  /gpaw_0_9833333333333333        None  /home/jovyan/thermo/
3                /gpaw_1_0        None  /home/jovyan/thermo/
4  /gpaw_1_0166666666666666        None  /home/jovyan/thermo/
5  /gpaw_1_0333333333333334        None  /home/jovyan/thermo/
6               /gpaw_1_05        None  /home/jovyan/thermo/

                   timestart timestop totalcputime  \
0 2020-10-02 17:24:24.200176     None         None
1 2020-10-02 17:26:45.417210     None         None
2 2020-10-02 17:28:37.112334     None         None
3 2020-10-02 17:30:26.714705     None         None
4 2020-10-02 17:31:58.800251     None         None
5 2020-10-02 17:34:47.304029     None         None
6 2020-10-02 17:36:23.322563     None         None

                                    computer hamilton hamversion parentid  \
0  pyiron@jupyter-pyiron-2dpyiron-2d996ovb6h#1  GpawJob       None     None
1  pyiron@jupyter-pyiron-2dpyiron-2d996ovb6h#1  GpawJob       None     None
2  pyiron@jupyter-pyiron-2dpyiron-2d996ovb6h#1  GpawJob       None     None
3  pyiron@jupyter-pyiron-2dpyiron-2d996ovb6h#1  GpawJob       None     None
4  pyiron@jupyter-pyiron-2dpyiron-2d996ovb6h#1  GpawJob       None     None
5  pyiron@jupyter-pyiron-2dpyiron-2d996ovb6h#1  GpawJob       None     None
6  pyiron@jupyter-pyiron-2dpyiron-2d996ovb6h#1  GpawJob       None     None

  masterid
0     None
1     None
2     None
3     None
4     None
5     None
6     None
```

Analysis

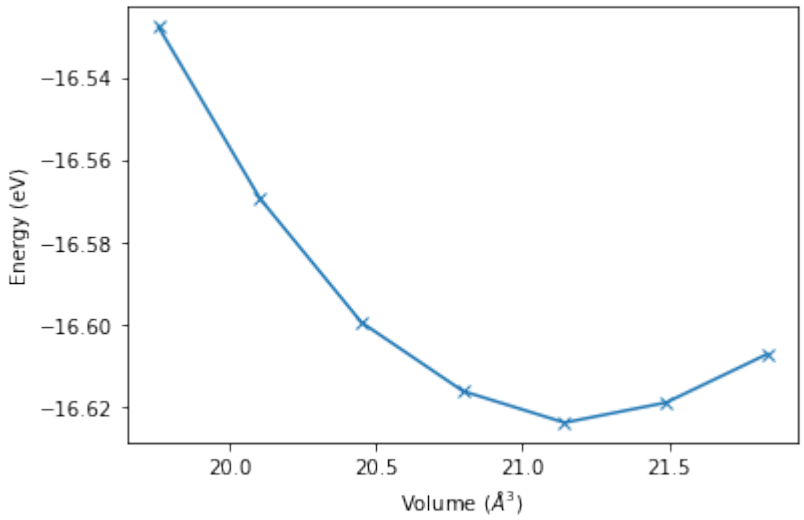
We aggregate the data for further processing in two separated lists, one for the volumes and one for the energies. To do so we iterate over the jobs within the project, filter the job names which contain the string 'vasp' and from those extract the final volume and the final energy.

[6]:
```
volume_lst, energy_lst = zip(*[[job['output/generic/volume'][-1], job['output/generic/
↪energy_pot'][-1]]
                               for job in pr.iter_jobs(convert_to_object=False) if
↪'gpaw' in job.job_name])
```

We plot the aggregated data using matplotlib.

[7]:
```
plt.plot(volume_lst, energy_lst, 'x-')
plt.xlabel('Volume ($\AA ^ 3$)')
plt.ylabel('Energy (eV)')
```

[7]:
```
Text(0, 0.5, 'Energy (eV)')
```



Encut Dependence

To extend the complexity of our simulation protocol we can not only iterate over different strains but also different energy cutoffs. For this we use multiple sub projects to structure the data. And we summarize the previous code in multiple functions to maintain a high level of readability. The first function calculates a specific strained configuration for an specifc energy cut off, while the second function analyses the different strained calculations for a specific energy cutoff and returns the list of energy volume pairs.

Functions

[8]:
```
def vasp_calculation_for_strain(pr, basis, strain, encut):
    strain_str = str(strain).replace('.', '_')
    job_vasp_strain = pr.create_job(job_type=pr.job_type.Gpaw, job_name='gpaw_' +␣
↪strain_str)
    job_vasp_strain.set_encut(encut)
    job_vasp_strain.structure = basis.copy()
    job_vasp_strain.structure.set_cell(cell=basis.cell * strain ** (1/3), scale_
↪atoms=True)
    job_vasp_strain.run()
```

[9]:
```
def energy_volume_pairs(pr):
    volume_lst, energy_lst = zip(*[[job['output/generic/volume'][-1], job['output/
↪generic/energy_pot'][-1]]
                                 for job in pr.iter_jobs(convert_to_object=False) if
↪'gpaw' in job.job_name])
    return volume_lst, energy_lst
```

Calculation

With these functions we can structure our code and implement the additional for loop to include multiple energy cutoffs.

[10]:
```
for encut in np.linspace(270, 320, 6):
    encut_str = 'encut_' + str(int(encut))
    pr_encut = pr.open(encut_str)
    for strain in np.linspace(0.95, 1.05, 7):
        vasp_calculation_for_strain(pr=pr_encut,
                                    basis=basis,
                                    strain=strain,
                                    encut=encut)
```

```
The job gpaw_0_95 was saved and received the ID: 8
The job gpaw_0_9666666666666667 was saved and received the ID: 9
The job gpaw_0_9833333333333333 was saved and received the ID: 10
The job gpaw_1_0 was saved and received the ID: 11
The job gpaw_1_0166666666666666 was saved and received the ID: 12
The job gpaw_1_0333333333333334 was saved and received the ID: 13
The job gpaw_1_05 was saved and received the ID: 14
The job gpaw_0_95 was saved and received the ID: 15
The job gpaw_0_9666666666666667 was saved and received the ID: 16
The job gpaw_0_9833333333333333 was saved and received the ID: 17
The job gpaw_1_0 was saved and received the ID: 18
The job gpaw_1_0166666666666666 was saved and received the ID: 19
The job gpaw_1_0333333333333334 was saved and received the ID: 20
The job gpaw_1_05 was saved and received the ID: 21
The job gpaw_0_95 was saved and received the ID: 22
The job gpaw_0_9666666666666667 was saved and received the ID: 23
The job gpaw_0_9833333333333333 was saved and received the ID: 24
The job gpaw_1_0 was saved and received the ID: 25
The job gpaw_1_0166666666666666 was saved and received the ID: 26
The job gpaw_1_0333333333333334 was saved and received the ID: 27
The job gpaw_1_05 was saved and received the ID: 28
The job gpaw_0_95 was saved and received the ID: 29
```

(continues on next page)

(continued from previous page)

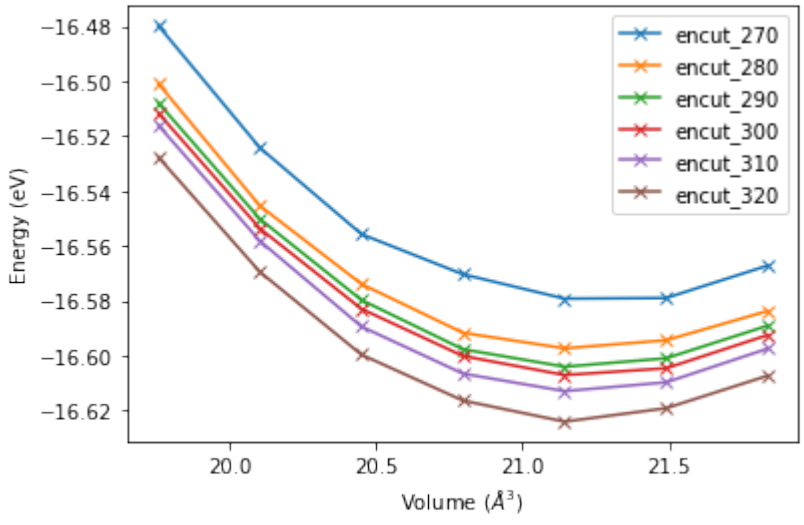
```
The job gpaw_0_9666666666666667 was saved and received the ID: 30
The job gpaw_0_9833333333333333 was saved and received the ID: 31
The job gpaw_1_0 was saved and received the ID: 32
The job gpaw_1_0166666666666666 was saved and received the ID: 33
The job gpaw_1_0333333333333334 was saved and received the ID: 34
The job gpaw_1_05 was saved and received the ID: 35
The job gpaw_0_95 was saved and received the ID: 36
The job gpaw_0_9666666666666667 was saved and received the ID: 37
The job gpaw_0_9833333333333333 was saved and received the ID: 38
The job gpaw_1_0 was saved and received the ID: 39
The job gpaw_1_0166666666666666 was saved and received the ID: 40
The job gpaw_1_0333333333333334 was saved and received the ID: 41
The job gpaw_1_05 was saved and received the ID: 42
The job gpaw_0_95 was saved and received the ID: 43
The job gpaw_0_9666666666666667 was saved and received the ID: 44
The job gpaw_0_9833333333333333 was saved and received the ID: 45
The job gpaw_1_0 was saved and received the ID: 46
The job gpaw_1_0166666666666666 was saved and received the ID: 47
The job gpaw_1_0333333333333334 was saved and received the ID: 48
The job gpaw_1_05 was saved and received the ID: 49
```

Analysis

The analysis is structured in a similar way. Here we use iter_groups() to iterate over the existing subprojects within our project and plot the individual energy volume curves using the functions defined above.

[11]:
```
for pr_encut in pr.iter_groups():
    volume_lst, energy_lst = energy_volume_pairs(pr_encut)
    plt.plot(volume_lst, energy_lst, 'x-', label=pr_encut.base_name)
plt.xlabel('Volume ($\AA ^ 3$)')
plt.ylabel('Energy (eV)')
plt.legend()
```

[11]:
```
<matplotlib.legend.Legend at 0x7f638828d6d0>
```



Fitting

After we created multiple datasets we can now start to fit the converged results. While it is possible to fit the results using a simple polynomial fit we prefer to use the phyiscally motivated birch murnaghan equation or the vinet equation. For this we create the Murnaghan object and use it is fitting functionality:

[12]:
```
murn = pr.create_job(job_type=pr.job_type.Murnaghan, job_name='murn')
```

Birch Marnaghan

[13]:
```
[e0, b0, bP, v0], [e0_error, b0_error, bP_error, v0_error] = murn._fit_leastsq(volume_
↪lst=volume_lst,
                                                                                energy_
↪lst=energy_lst,
                                                                                ␣
↪fittype='birchmurnaghan')
[e0, b0, bP, v0]
```

[13]:
```
[-16.623387215037408, 280.9875784436634, 4.060730693834813, 21.19199044120968]
```

Vinet

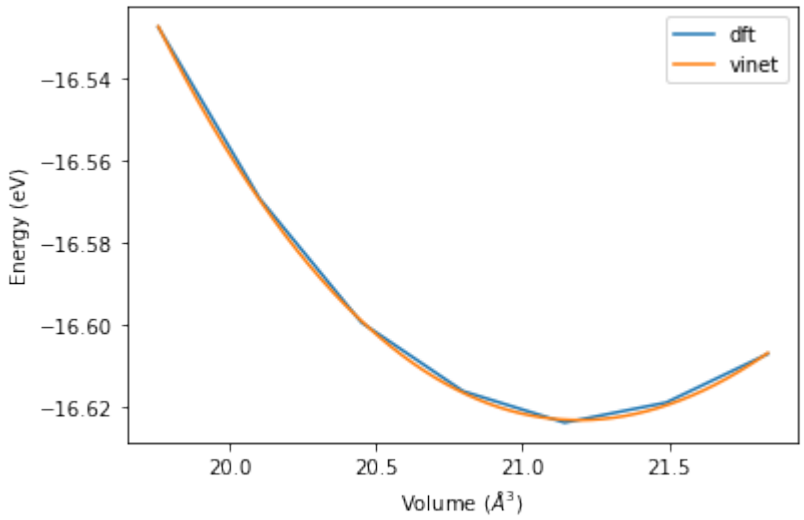
[14]:
```
[e0, b0, bP, v0], [e0_error, b0_error, bP_error, v0_error] = murn._fit_leastsq(volume_
↪lst=volume_lst,
                                                                                energy_
↪lst=energy_lst,
                                                                                ␣
↪fittype='vinet')
[e0, b0, bP, v0]
```

[14]:
```
[-16.623384845899427, 280.93805771100557, 4.105492272090299, 21.19185363600345]
```

We see that both equation of states give slightly different results, with overall good agreement. To validate the agreement we plot the with with the original data.

[15]:
```
vol_lst = np.linspace(np.min(volume_lst), np.max(volume_lst), 1000)
plt.plot(volume_lst, energy_lst, label='dft')
plt.plot(vol_lst, murn.fit_module.vinet_energy(vol_lst, e0, b0/ 160.21766208, bP, v0),
 ↪ label='vinet')
plt.xlabel('Volume ($\AA ^ 3$)')
plt.ylabel('Energy (eV)')
plt.legend()
```

[15]:
```
<matplotlib.legend.Legend at 0x7f638078d150>
```



Murnaghan Module

Besides the fitting capabilities the Murnaghan module can also be used to run a set of calculations. For this we define a reference job, which can be either a Vasp calculation or any other pyiron job type and then specify the input parameters for the Murnaghan job.

[16]:
```
job_vasp_strain = pr.create_job(job_type=pr.job_type.Gpaw, job_name='gpaw')
job_vasp_strain.set_encut(320)
job_vasp_strain.structure = basis.copy()
```

[17]:
```
murn = pr.create_job(job_type=pr.job_type.Murnaghan, job_name='murn')
murn.ref_job = job_vasp_strain
murn.input
```

[17]:
```
    Parameter       Value  \
0  num_points          11
1    fit_type  polynomial
2   fit_order           3
3   vol_range         0.1


                                                                  Comment
0                                                  number of sample points
1  ['polynomial', 'birch', 'birchmurnaghan', 'murnaghan', 'pouriertarantola', 'vinet']
2                                                  order of the fit polynom
3                          relative volume variation around volume defined by ref_ham
```

We modify the input parameters to agree with the settings used in the examples above and execute the simulation by calling the run command on the murnaghan job object.

[18]:
```
murn.input['num_points'] = 7
murn.input['vol_range'] = 0.05
```

[19]:
```
type(murn.structure)
```

[19]:
```
ase.atoms.Atoms
```

[20]:
```
pr.job_table()
```

[20]:

	id	status	chemicalformula	job	subjob	projectpath	project
0	1	finished	None	gpaw_0_95	/gpaw_0_95	None	/home/jovyan/thermo/
1	2	finished	None	gpaw_0_9666666666666667	/gpaw_0_9666666666666667	None	/home/jovyan/thermo/
2	3	finished	None	gpaw_0_9833333333333333	/gpaw_0_9833333333333333	None	/home/jovyan/thermo/
3	4	finished	None	gpaw_1_0	/gpaw_1_0	None	/home/jovyan/thermo/
4	5	finished	None	gpaw_1_0166666666666666			
5	6	finished	None	gpaw_1_0333333333333334			
6	7	finished	None	gpaw_1_05			
7	8	finished	None	gpaw_0_95			
8	9	finished	None	gpaw_0_9666666666666667			
9	10	finished	None	gpaw_0_9833333333333333			
10	11	finished	None	gpaw_1_0			
11	12	finished	None	gpaw_1_0166666666666666			
12	13	finished	None	gpaw_1_0333333333333334			
13	14	finished	None	gpaw_1_05			
14	15	finished	None	gpaw_0_95			
15	16	finished	None	gpaw_0_9666666666666667			
16	17	finished	None	gpaw_0_9833333333333333			
17	18	finished	None	gpaw_1_0			
18	19	finished	None	gpaw_1_0166666666666666			
19	20	finished	None	gpaw_1_0333333333333334			
20	21	finished	None	gpaw_1_05			
21	22	finished	None	gpaw_0_95			
22	23	finished	None	gpaw_0_9666666666666667			
23	24	finished	None	gpaw_0_9833333333333333			
24	25	finished	None	gpaw_1_0			
25	26	finished	None	gpaw_1_0166666666666666			
26	27	finished	None	gpaw_1_0333333333333334			
27	28	finished	None	gpaw_1_05			
28	29	finished	None	gpaw_0_95			
29	30	finished	None	gpaw_0_9666666666666667			
30	31	finished	None	gpaw_0_9833333333333333			
31	32	finished	None	gpaw_1_0			
32	33	finished	None	gpaw_1_0166666666666666			
33	34	finished	None	gpaw_1_0333333333333334			
34	35	finished	None	gpaw_1_05			
35	36	finished	None	gpaw_0_95			
36	37	finished	None	gpaw_0_9666666666666667			
37	38	finished	None	gpaw_0_9833333333333333			
38	39	finished	None	gpaw_1_0			
39	40	finished	None	gpaw_1_0166666666666666			
40	41	finished	None	gpaw_1_0333333333333334			
41	42	finished	None	gpaw_1_05			
42	43	finished	None	gpaw_0_95			
43	44	finished	None	gpaw_0_9666666666666667			
44	45	finished	None	gpaw_0_9833333333333333			
45	46	finished	None	gpaw_1_0			
46	47	finished	None	gpaw_1_0166666666666666			
47	48	finished	None	gpaw_1_0333333333333334			
48	49	finished	None	gpaw_1_05			

(continues on next page)

(continued from previous page)

```
4   /gpaw_1_0166666666666666        None            /home/jovyan/thermo/
5   /gpaw_1_0333333333333334        None            /home/jovyan/thermo/
6                /gpaw_1_05         None            /home/jovyan/thermo/
7                /gpaw_0_95         None  /home/jovyan/thermo/encut_270/
8   /gpaw_0_9666666666666667        None  /home/jovyan/thermo/encut_270/
9   /gpaw_0_9833333333333333        None  /home/jovyan/thermo/encut_270/
10               /gpaw_1_0          None  /home/jovyan/thermo/encut_270/
11  /gpaw_1_0166666666666666        None  /home/jovyan/thermo/encut_270/
12  /gpaw_1_0333333333333334        None  /home/jovyan/thermo/encut_270/
13               /gpaw_1_05         None  /home/jovyan/thermo/encut_270/
14               /gpaw_0_95         None  /home/jovyan/thermo/encut_280/
15  /gpaw_0_9666666666666667        None  /home/jovyan/thermo/encut_280/
16  /gpaw_0_9833333333333333        None  /home/jovyan/thermo/encut_280/
17               /gpaw_1_0          None  /home/jovyan/thermo/encut_280/
18  /gpaw_1_0166666666666666        None  /home/jovyan/thermo/encut_280/
19  /gpaw_1_0333333333333334        None  /home/jovyan/thermo/encut_280/
20               /gpaw_1_05         None  /home/jovyan/thermo/encut_280/
21               /gpaw_0_95         None  /home/jovyan/thermo/encut_290/
22  /gpaw_0_9666666666666667        None  /home/jovyan/thermo/encut_290/
23  /gpaw_0_9833333333333333        None  /home/jovyan/thermo/encut_290/
24               /gpaw_1_0          None  /home/jovyan/thermo/encut_290/
25  /gpaw_1_0166666666666666        None  /home/jovyan/thermo/encut_290/
26  /gpaw_1_0333333333333334        None  /home/jovyan/thermo/encut_290/
27               /gpaw_1_05         None  /home/jovyan/thermo/encut_290/
28               /gpaw_0_95         None  /home/jovyan/thermo/encut_300/
29  /gpaw_0_9666666666666667        None  /home/jovyan/thermo/encut_300/
30  /gpaw_0_9833333333333333        None  /home/jovyan/thermo/encut_300/
31               /gpaw_1_0          None  /home/jovyan/thermo/encut_300/
32  /gpaw_1_0166666666666666        None  /home/jovyan/thermo/encut_300/
33  /gpaw_1_0333333333333334        None  /home/jovyan/thermo/encut_300/
34               /gpaw_1_05         None  /home/jovyan/thermo/encut_300/
35               /gpaw_0_95         None  /home/jovyan/thermo/encut_310/
36  /gpaw_0_9666666666666667        None  /home/jovyan/thermo/encut_310/
37  /gpaw_0_9833333333333333        None  /home/jovyan/thermo/encut_310/
38               /gpaw_1_0          None  /home/jovyan/thermo/encut_310/
39  /gpaw_1_0166666666666666        None  /home/jovyan/thermo/encut_310/
40  /gpaw_1_0333333333333334        None  /home/jovyan/thermo/encut_310/
41               /gpaw_1_05         None  /home/jovyan/thermo/encut_310/
42               /gpaw_0_95         None  /home/jovyan/thermo/encut_320/
43  /gpaw_0_9666666666666667        None  /home/jovyan/thermo/encut_320/
44  /gpaw_0_9833333333333333        None  /home/jovyan/thermo/encut_320/
45               /gpaw_1_0          None  /home/jovyan/thermo/encut_320/
46  /gpaw_1_0166666666666666        None  /home/jovyan/thermo/encut_320/
47  /gpaw_1_0333333333333334        None  /home/jovyan/thermo/encut_320/
48               /gpaw_1_05         None  /home/jovyan/thermo/encut_320/

                   timestart timestop totalcputime  \
0  2020-10-02 17:24:24.200176     None         None
1  2020-10-02 17:26:45.417210     None         None
2  2020-10-02 17:28:37.112334     None         None
3  2020-10-02 17:30:26.714705     None         None
4  2020-10-02 17:31:58.800251     None         None
5  2020-10-02 17:34:47.304029     None         None
6  2020-10-02 17:36:23.322563     None         None
7  2020-10-02 17:38:00.805999     None         None
8  2020-10-02 17:40:27.023982     None         None
9  2020-10-02 17:42:55.820191     None         None
```

(continues on next page)

(continued from previous page)

```
10 2020-10-02 17:45:10.442772     None         None
11 2020-10-02 17:47:59.450726     None         None
12 2020-10-02 17:51:07.518608     None         None
13 2020-10-02 17:54:45.224784     None         None
14 2020-10-02 17:58:11.528057     None         None
15 2020-10-02 18:00:11.919363     None         None
16 2020-10-02 18:02:25.229474     None         None
17 2020-10-02 18:05:13.598633     None         None
18 2020-10-02 18:08:06.130672     None         None
19 2020-10-02 18:11:21.717226     None         None
20 2020-10-02 18:14:19.003564     None         None
21 2020-10-02 18:16:48.228097     None         None
22 2020-10-02 18:19:42.602848     None         None
23 2020-10-02 18:22:33.879253     None         None
24 2020-10-02 18:25:12.937586     None         None
25 2020-10-02 18:27:18.445423     None         None
26 2020-10-02 18:29:51.108935     None         None
27 2020-10-02 18:33:09.633165     None         None
28 2020-10-02 18:35:24.100573     None         None
29 2020-10-02 18:37:51.134146     None         None
30 2020-10-02 18:40:15.407176     None         None
31 2020-10-02 18:42:27.007123     None         None
32 2020-10-02 18:45:20.422390     None         None
33 2020-10-02 18:47:26.819490     None         None
34 2020-10-02 18:49:24.101232     None         None
35 2020-10-02 18:51:11.902579     None         None
36 2020-10-02 18:53:53.696423     None         None
37 2020-10-02 18:55:31.120613     None         None
38 2020-10-02 18:57:12.122217     None         None
39 2020-10-02 18:58:42.202686     None         None
40 2020-10-02 19:00:25.512077     None         None
41 2020-10-02 19:02:08.222775     None         None
42 2020-10-02 19:03:42.105391     None         None
43 2020-10-02 19:05:35.407201     None         None
44 2020-10-02 19:07:21.099215     None         None
45 2020-10-02 19:08:51.322535     None         None
46 2020-10-02 19:10:14.501550     None         None
47 2020-10-02 19:11:57.005028     None         None
48 2020-10-02 19:13:34.919295     None         None

                                    computer hamilton hamversion parentid  \
0   pyiron@jupyter-pyiron-2dpyiron-2d996ovb6h#1  GpawJob       None     None
1   pyiron@jupyter-pyiron-2dpyiron-2d996ovb6h#1  GpawJob       None     None
2   pyiron@jupyter-pyiron-2dpyiron-2d996ovb6h#1  GpawJob       None     None
3   pyiron@jupyter-pyiron-2dpyiron-2d996ovb6h#1  GpawJob       None     None
4   pyiron@jupyter-pyiron-2dpyiron-2d996ovb6h#1  GpawJob       None     None
5   pyiron@jupyter-pyiron-2dpyiron-2d996ovb6h#1  GpawJob       None     None
6   pyiron@jupyter-pyiron-2dpyiron-2d996ovb6h#1  GpawJob       None     None
7   pyiron@jupyter-pyiron-2dpyiron-2d996ovb6h#1  GpawJob       None     None
8   pyiron@jupyter-pyiron-2dpyiron-2d996ovb6h#1  GpawJob       None     None
9   pyiron@jupyter-pyiron-2dpyiron-2d996ovb6h#1  GpawJob       None     None
10  pyiron@jupyter-pyiron-2dpyiron-2d996ovb6h#1  GpawJob       None     None
11  pyiron@jupyter-pyiron-2dpyiron-2d996ovb6h#1  GpawJob       None     None
12  pyiron@jupyter-pyiron-2dpyiron-2d996ovb6h#1  GpawJob       None     None
13  pyiron@jupyter-pyiron-2dpyiron-2d996ovb6h#1  GpawJob       None     None
14  pyiron@jupyter-pyiron-2dpyiron-2d996ovb6h#1  GpawJob       None     None
15  pyiron@jupyter-pyiron-2dpyiron-2d996ovb6h#1  GpawJob       None     None
```

(continues on next page)

(continued from previous page)

```
16  pyiron@jupyter-pyiron-2dpyiron-2d996ovb6h#1  GpawJob       None     None
17  pyiron@jupyter-pyiron-2dpyiron-2d996ovb6h#1  GpawJob       None     None
18  pyiron@jupyter-pyiron-2dpyiron-2d996ovb6h#1  GpawJob       None     None
19  pyiron@jupyter-pyiron-2dpyiron-2d996ovb6h#1  GpawJob       None     None
20  pyiron@jupyter-pyiron-2dpyiron-2d996ovb6h#1  GpawJob       None     None
21  pyiron@jupyter-pyiron-2dpyiron-2d996ovb6h#1  GpawJob       None     None
22  pyiron@jupyter-pyiron-2dpyiron-2d996ovb6h#1  GpawJob       None     None
23  pyiron@jupyter-pyiron-2dpyiron-2d996ovb6h#1  GpawJob       None     None
24  pyiron@jupyter-pyiron-2dpyiron-2d996ovb6h#1  GpawJob       None     None
25  pyiron@jupyter-pyiron-2dpyiron-2d996ovb6h#1  GpawJob       None     None
26  pyiron@jupyter-pyiron-2dpyiron-2d996ovb6h#1  GpawJob       None     None
27  pyiron@jupyter-pyiron-2dpyiron-2d996ovb6h#1  GpawJob       None     None
28  pyiron@jupyter-pyiron-2dpyiron-2d996ovb6h#1  GpawJob       None     None
29  pyiron@jupyter-pyiron-2dpyiron-2d996ovb6h#1  GpawJob       None     None
30  pyiron@jupyter-pyiron-2dpyiron-2d996ovb6h#1  GpawJob       None     None
31  pyiron@jupyter-pyiron-2dpyiron-2d996ovb6h#1  GpawJob       None     None
32  pyiron@jupyter-pyiron-2dpyiron-2d996ovb6h#1  GpawJob       None     None
33  pyiron@jupyter-pyiron-2dpyiron-2d996ovb6h#1  GpawJob       None     None
34  pyiron@jupyter-pyiron-2dpyiron-2d996ovb6h#1  GpawJob       None     None
35  pyiron@jupyter-pyiron-2dpyiron-2d996ovb6h#1  GpawJob       None     None
36  pyiron@jupyter-pyiron-2dpyiron-2d996ovb6h#1  GpawJob       None     None
37  pyiron@jupyter-pyiron-2dpyiron-2d996ovb6h#1  GpawJob       None     None
38  pyiron@jupyter-pyiron-2dpyiron-2d996ovb6h#1  GpawJob       None     None
39  pyiron@jupyter-pyiron-2dpyiron-2d996ovb6h#1  GpawJob       None     None
40  pyiron@jupyter-pyiron-2dpyiron-2d996ovb6h#1  GpawJob       None     None
41  pyiron@jupyter-pyiron-2dpyiron-2d996ovb6h#1  GpawJob       None     None
42  pyiron@jupyter-pyiron-2dpyiron-2d996ovb6h#1  GpawJob       None     None
43  pyiron@jupyter-pyiron-2dpyiron-2d996ovb6h#1  GpawJob       None     None
44  pyiron@jupyter-pyiron-2dpyiron-2d996ovb6h#1  GpawJob       None     None
45  pyiron@jupyter-pyiron-2dpyiron-2d996ovb6h#1  GpawJob       None     None
46  pyiron@jupyter-pyiron-2dpyiron-2d996ovb6h#1  GpawJob       None     None
47  pyiron@jupyter-pyiron-2dpyiron-2d996ovb6h#1  GpawJob       None     None
48  pyiron@jupyter-pyiron-2dpyiron-2d996ovb6h#1  GpawJob       None     None

   masterid
0      None
1      None
2      None
3      None
4      None
5      None
6      None
7      None
8      None
9      None
10     None
11     None
12     None
13     None
14     None
15     None
16     None
17     None
18     None
19     None
20     None
21     None
```

(continues on next page)

(continued from previous page)

```
22      None
23      None
24      None
25      None
26      None
27      None
28      None
29      None
30      None
31      None
32      None
33      None
34      None
35      None
36      None
37      None
38      None
39      None
40      None
41      None
42      None
43      None
44      None
45      None
46      None
47      None
48      None
```

[21]:
```
murn.run()
```

```
The job murn was saved and received the ID: 50
```

```
/srv/conda/envs/notebook/lib/python3.7/site-packages/ase/cell.py:17: FutureWarning:␣
↪Cell object will no longer have pbc
  warnings.warn(deprecation_msg, FutureWarning)
```

```
The job strain_0_95 was saved and received the ID: 51
The job strain_0_9666667 was saved and received the ID: 52
The job strain_0_9833333 was saved and received the ID: 53
The job strain_1_0 was saved and received the ID: 54
The job strain_1_0166667 was saved and received the ID: 55
The job strain_1_0333333 was saved and received the ID: 56
The job strain_1_05 was saved and received the ID: 57
job_id:  51 finished
job_id:  52 finished
job_id:  53 finished
job_id:  54 finished
job_id:  55 finished
job_id:  56 finished
job_id:  57 finished
```

Afterwards we can use the build in capabilites to plot the resulting energy volume curve and fit different equations of state to the calculated energy volume pairs.

[22]:
```
murn.output_to_pandas()
```

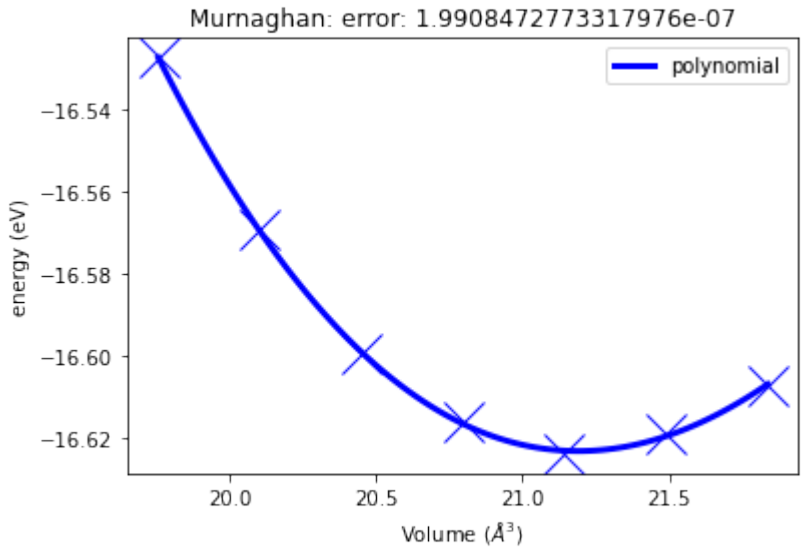
[22]:
```
      volume     energy  error  id  equilibrium_b_prime  \
0  19.757031 -16.527632    0.0  51             4.704621
```

(continues on next page)

(continued from previous page)

```
1  20.103646 -16.569446    0.0  52             4.704621
2  20.450260 -16.599599    0.0  53             4.704621
3  20.796875 -16.616336    0.0  54             4.704621
4  21.143490 -16.623997    0.0  55             4.704621
5  21.490104 -16.619111    0.0  56             4.704621
6  21.836719 -16.607260    0.0  57             4.704621

   equilibrium_bulk_modulus  equilibrium_energy  equilibrium_volume
0                 280.897787          -16.623369           21.189923
1                 280.897787          -16.623369           21.189923
2                 280.897787          -16.623369           21.189923
3                 280.897787          -16.623369           21.189923
4                 280.897787          -16.623369           21.189923
5                 280.897787          -16.623369           21.189923
6                 280.897787          -16.623369           21.189923
```

[23]:
```
murn.plot()
```



[24]:
```
murn.fit_vinet()
```

[24]:
```
{'fit_type': 'vinet',
 'volume_eq': 21.19185363600345,
 'energy_eq': -16.623384845899427,
 'bulkmodul_eq': 280.93805771100557,
 'b_prime_eq': 4.105492272090299,
 'least_square_error': array([4.97985872e-04, 1.01905536e+01, 1.63735940e+00, 8.
↪58104678e-03])}
```

Common mistakes

Not copying the basis

It is important to copy the basis before applying the strain, as the strain has to be applied on the initial structure, not the previous structure:

[25]:
```
volume_lst_with_copy = []
for strain in np.linspace(0.95, 1.05, 7):
    basis_copy = basis.copy()
    basis_copy.set_cell(cell=basis.cell * strain ** (1/3), scale_atoms=True)
    volume_lst_with_copy.append(basis_copy.get_volume())
```

[26]:
```
basis_copy = basis.copy()
volume_lst_without_copy = []
for strain in np.linspace(0.95, 1.05, 7):
    basis_copy.set_cell(cell=basis_copy.cell * strain ** (1/3), scale_atoms=True)
    volume_lst_without_copy.append(basis_copy.get_volume())
```

[27]:
```
volume_lst_with_copy, volume_lst_without_copy
```

[27]:
```
([19.757031250000004,
  20.10364583333333,
  20.450260416666666,
  20.796874999999996,
  21.143489583333338,
  21.490104166666654,
  21.83671875000001],
 [19.757031250000004,
  19.098463541666664,
  18.780155815972222,
  18.780155815972222,
  19.09315841290509,
  19.729597026668593,
  20.716076878002024])
```

Rescaling the cell

Another common issue is the rescaling of the supercell, there are multiple options to choose from. We used the option to scale the atoms with the supercell.

[28]:
```
basis_copy = basis.copy()
strain = 0.5
basis_copy.set_cell(cell=basis_copy.cell * strain ** (1/3), scale_atoms=True)
basis_copy.plot3d()
```

```
NGLWidget()
```

A nother typical case is rescaling the cell to increase the distance between the atoms or add vacuum. But that is not what we want to fit an energy volume curve.

[29]:
```
basis_copy = basis.copy()
strain = 0.5
basis_copy.set_cell(cell=basis_copy.cell * strain ** (1/3), scale_atoms=False)
basis_copy.plot3d()
```

```
NGLWidget()
```

The same can be achieved by setting the basis to relative coordinates.

[30]:
```
basis_copy = basis.copy()
strain = 0.5
basis_copy.set_relative()
basis_copy.cell *= strain ** (1/3)
basis_copy.plot3d()
```

```
NGLWidget()
```

[31]:
```
basis_copy = basis.copy()
strain = 0.5
basis_copy.cell *= strain ** (1/3)
basis_copy.plot3d()
```

```
NGLWidget()
```

[ ]:
```

```

3.3.3 Creating structures in pyiron

This section gives a brief introduction about some of the tools available in pyiron to construct atomic structures.

For the sake of compatibility, our structure class is written to be compatible with the popular Atomistic Simulation Environment package (ASE). This makes it possible to use routines from ASE to help set-up structures.

Furthermore, pyiron uses the NGLview package to visualize the structures and trajectories interactively in 3D using NGLview-widgets.

As preparation for the following discussion we import a few python libraries

[1]:
```
import numpy as np
%matplotlib inline
import matplotlib.pylab as plt
```

and create a pyiron project named ‘structures’:

[2]:
```
from pyiron import Project
pr = Project(path='structures')
```

Bulk crystals

In this section we discuss various possibilities to create bulk crystal structures.

Using `create_structure()`

The simplest way to generate simple crystal structures is using the inbuilt `create_structure()` function specifying the element symbol, Bravais basis and the lattice constant(s)

Note: The output gives a cubic cell rather than the smallest non-orthogonal unit cell.

[3]:
```
structure = pr.create_structure('Al',
                                bravais_basis='fcc',
                                lattice_constant=4.05)
```

To plot the structure interactively in 3D simply use:

[4]:
```
structure.plot3d()
```

```
NGLWidget()
```

Using `create_ase_bulk()`

Another convenient way to set up structures is using the `create_ase_bulk()` function which is built on top of the ASE build package for bulk crystals. This function returns an object which is of the pyiron structure object type.

Example: fcc bulk aluminum in a cubic cell

[5]:
```
structure = pr.create_ase_bulk('Al', cubic=True)
structure.plot3d()
```

```
NGLWidget()
```

Example: wurtzite GaN in a 3x3x3 repeated orthorhombic cell.

Note: - In contrast to new_structure = structure.repeat() which creates a new object, set_repeat() modifies the existing structure object. - Setting `spacefill=False` in the `plot3d()` method changes the atomic structure style to "ball and stick".

[6]:
```
structure = pr.create_ase_bulk('AlN',
                               crystalstructure='wurtzite',
                               a=3.5, orthorhombic=True)
structure.set_repeat([3,3,3])
structure.plot3d(spacefill=False)
```

```
NGLWidget()
```

Creating surfaces (using ASE)

Surfaces can be created using the `create_surface()` function which is also built on top of the ASE build package for surfaces

Example: Creating a 3x4 fcc Al(111) surface with 4 layers and a vacuum of 10 Ångström

[7]:
```
Al_111 = pr.create_surface("Al", surface_type="fcc111",
                           size=(3, 4, 4), vacuum=10, orthogonal=True)
Al_111.plot3d()
```

```
NGLWidget()
```

Creating structures without importing the project class

In all the examples shown above, the structures are create from the pyiron `Project` object. It is also possible to do this without importing/initializing this object. For this the appropriate imports must be made.

[8]:
```
from pyiron import create_ase_bulk, create_surface
```

[9]:
```
structure = create_ase_bulk('AlN',
                            crystalstructure='wurtzite',
                            a=3.5, orthorhombic=True)
structure.set_repeat([3,3,3])
structure.plot3d(spacefill=False)
```

```
NGLWidget()
```

[10]:
```
Al_111 = create_surface("Al", surface_type="fcc111",
                        size=(3, 4, 4), vacuum=10, orthogonal=True)
Al_111.plot3d()
```

```
NGLWidget()
```

Using the ASE spacegroup class

[11]:
```
from ase.spacegroup import crystal
from pyiron import ase_to_pyiron

a = 9.04
skutterudite = crystal(('Co', 'Sb'),
                       basis=[(0.25, 0.25, 0.25), (0.0, 0.335, 0.158)],
                       spacegroup=204,
                       cellpar=[a, a, a, 90, 90, 90])
skutterudite = ase_to_pyiron(skutterudite)
```

[12]:
```
skutterudite.plot3d()
```

```
NGLWidget()
```

Accessing the properties of the structure object

Using the bulk aluminum fcc example from before the structure object can be created by

[13]:
```
structure = pr.create_ase_bulk('Al', cubic=True)
```

A summary of the information about the structure is given by using

[14]:
```
print(structure)
```

```
Al: [0. 0. 0.]
Al: [0.    2.025 2.025]
Al: [2.025 0.    2.025]
Al: [2.025 2.025 0.   ]
pbc: [ True  True  True]
cell:
Cell([4.05, 4.05, 4.05])
```

The cell vectors of the structure object can be accessed and edited through

[15]:
```
structure.cell
```

[15]:
```
Cell([4.05, 4.05, 4.05])
```

The positions of the atoms in the structure object can be accessed and edited through

[16]:
```
structure.positions
```

[16]:
```
array([[0.   , 0.   , 0.   ],
       [0.   , 2.025, 2.025],
       [2.025, 0.   , 2.025],
       [2.025, 2.025, 0.   ]])
```

Point defects

Creating a single vacancy

We start by setting up a 4x4x4 supercell

[17]:
```
structure = pr.create_ase_bulk('Al', cubic=True)
structure.set_repeat([4,4,4])
```

To create the vacancy at position index "0" simply use:

[18]:
```
del structure[0]
```

To plot the structure that now contains a vacancy run:

[19]:
```
structure.plot3d()
```

```
NGLWidget()
```

Creating multiple vacancies

[20]:
```
# First create a 4x4x4 supercell
structure = pr.create_ase_bulk('Al', cubic=True)
structure.set_repeat([4,4,4])
print('Number of atoms in the repeat unit: ',structure.get_number_of_atoms())
```

```
Number of atoms in the repeat unit:  256
```

The `del` command works for passing a list of indices to the structure object. For example, a random set of n_{vac} vacancies can be created by using

[21]:
```
# Generate a list of indices for the vacancies
n_vac = 24
vac_ind_lst = np.random.permutation(len(structure))[:n_vac]

# Remove atoms according to the "vac_ind_lst"
del structure[vac_ind_lst]
```

[22]:
```
# Visualize the structure
print('Number of atoms in the repeat unit: ',structure.get_number_of_atoms())
structure.plot3d()
```

```
Number of atoms in the repeat unit:  232
```

```
NGLWidget()
```

Random substitutial alloys

[23]:
```
# Create a 4x4x4 supercell
structure = pr.create_ase_bulk('Al', cubic=True)
structure.set_repeat([4,4,4])
```

Substitutional atoms can be defined by changing the atomic species accessed through its position index.

Here, we set n_{sub} magnesium substitutional atoms at random positions

[24]:
```
n_sub = 24
structure[np.random.permutation(len(structure))[:n_sub]] = 'Mg'
```

[25]:
```
# Visualize the structure and print some additional information about the structure
print('Number of atoms in the repeat unit: ',structure.get_number_of_atoms())
print('Chemical formula: ',structure.get_chemical_formula())
structure.plot3d()
```

```
Number of atoms in the repeat unit:  256
Chemical formula:  Al232Mg24
```

```
NGLWidget()
```

Explicit definition of the structure

You can also set-up structures through the explicit input of the cell parameters and positions

[26]:
```
cell = 10.0 * np.eye(3) # Specifying the cell dimensions
positions = [[0.25, 0.25, 0.25], [0.75, 0.75, 0.75]]
elements = ['O', 'O']

# Now use the Atoms class to create the instance.
O_dimer = pr.create_atoms(elements=elements, scaled_positions=positions, cell=cell)

O_dimer.plot3d()
```

```
NGLWidget()
```

Importing from cif/other file formats

Parsers from ASE can be used to import structures from other formats. In this example, we will download and import a Nepheline structure from the Crystallography Open Database (COD)

[27]:
```
# The COD structures can be accessed through their unique COD identifier
cod = 1008753
filename = '{}.cif'.format(cod)
url = 'http://www.crystallography.net/cod/{}'.format(filename)
```

[28]:
```
cif_structure = """\
#------------------------------------------------------------------------------
#$Date: 2015-01-27 21:58:39 +0200 (Tue, 27 Jan 2015) $
#$Revision: 130149 $
#$URL: svn://www.crystallography.net/cod/cif/1/00/87/1008753.cif $
#------------------------------------------------------------------------------
#
# This file is available in the Crystallography Open Database (COD),
# http://www.crystallography.net/
#
# All data on this site have been placed in the public domain by the
# contributors.
#
data_1008753
loop_
_publ_author_name
'Buerger, M J'
'Klein, G E'
'Donnay, G'
_publ_section_title
;
Determination of the crystal structure of nepheline
;
_journal_coden_ASTM              AMMIAY
_journal_name_full               'American Mineralogist'
_journal_page_first              805
_journal_page_last               818
_journal_volume                  39
_journal_year                    1954
_chemical_formula_structural     'K Na3 Al4 Si4 O16'
_chemical_formula_sum            'Al4 K Na3 O16 Si4'
_chemical_name_mineral           Nepheline
_chemical_name_systematic        'Potassium trisodium tetraaluminium silicate'
_space_group_IT_number           173
_symmetry_cell_setting           hexagonal
_symmetry_Int_Tables_number      173
_symmetry_space_group_name_Hall  'P 6c'
_symmetry_space_group_name_H-M   'P 63'
_cell_angle_alpha                90
_cell_angle_beta                 90
_cell_angle_gamma                120
_cell_formula_units_Z            2
_cell_length_a                   10.01
_cell_length_b                   10.01
_cell_length_c                   8.405
_cell_volume                     729.4
_cod_database_code               1008753
loop_
_symmetry_equiv_pos_as_xyz
x,y,z
-y,x-y,z
y-x,-x,z
-x,-y,1/2+z
y,y-x,1/2+z
x-y,x,1/2+z
loop_
_atom_site_label
```

(continues on next page)

(continued from previous page)

```
_atom_site_type_symbol
_atom_site_symmetry_multiplicity
_atom_site_Wyckoff_symbol
_atom_site_fract_x
_atom_site_fract_y
_atom_site_fract_z
_atom_site_occupancy
_atom_site_attached_hydrogens
_atom_site_calc_flag
K1 K1+ 2 a 0. 0. 0. 1. 0 d
Al1 Al3+ 2 b 0.3333 0.6667 0.18 1. 0 d
Si1 Si4+ 2 b 0.3333 0.6667 0.82 1. 0 d
O1 O2- 2 b 0.3333 0.6667 0. 1. 0 d
Na1 Na1+ 6 c 0.008 0.432 0. 1. 0 d
Al2 Al3+ 6 c 0.092 0.33 0.67 1. 0 d
Si2 Si4+ 6 c 0.092 0.33 0.33 1. 0 d
O2 O2- 6 c 0.02 0.33 0.5 1. 0 d
O3 O2- 6 c 0.18 0.5 0.75 1. 0 d
O4 O2- 6 c 0.17 0.53 0.25 1. 0 d
O5 O2- 6 c 0.23 0.28 0.25 1. 0 d
O6 O2- 6 c 0.23 0.28 0.75 1. 0 d
loop_
_atom_type_symbol
_atom_type_oxidation_number
K1+ 1.000
Al3+ 3.000
Si4+ 4.000
O2- -2.000
Na1+ 1.000"""
```

[29]:
```
# Download and save the structure file locally
# import urllib
# urllib.request.urlretrieve(url=url, filename='strucs.'+filename);
with open('strucs.'+filename, "w") as f:
    f.writelines(cif_structure)
```

[30]:
```
# Using ase parsers to read the structure and then convert to a pyiron instance
import ase
from pyiron import ase_to_pyiron

structure = ase_to_pyiron(ase.io.read(filename='strucs.'+filename,
                                      format='cif'))
structure.info["cod"] = cod
```

```
/srv/conda/envs/notebook/lib/python3.7/site-packages/ase/io/cif.py:380: UserWarning:␣
↪crystal system 'hexagonal' is not interpreted for space group Spacegroup(173,␣
↪setting=1). This may result in wrong setting!
  setting_name, spacegroup))
```

[31]:
```
structure.plot3d()
```

```
NGLWidget()
```

Structures can be stored indepently from jobs in HDF5 by using the special `StructureContainer` job. To save to disk, call `run()`.

[32]:
```
container = pr.create_job(pr.job_type.StructureContainer, "nepheline")
container.structure = structure
container.run()
```

```
The job nepheline was saved and received the ID: 1
```

It's also possible to store multiple structures in one container and to store directly from a job. Let's use this here to store the equilibrated structures at finite temperatures.

[33]:
```
al_container = pr.create_job(pr.job_type.StructureContainer, "al_temp", delete_
↪existing_job=True)
for T in (400, 600, 800):
    j = pr.create_job(pr.job_type.Lammps, "T_{}".format(T))
    j.structure = pr.create_ase_bulk("Al", cubic = True)
    j.potential = j.list_potentials()[0]
    j.calc_md(temperature=T, n_ionic_steps=1000, pressure=0)
    j.run()
    structure = j.get_structure(-1)
    structure.info["T"] = T
    structure.info["P"] = 0
    al_container.append(structure)

al_container.run()
```

```
This group does not exist in the HDF5 file al_temp
The job T_400 was saved and received the ID: 2
The job T_600 was saved and received the ID: 3
The job T_800 was saved and received the ID: 4
The job al_temp was saved and received the ID: 5
```

[34]:
```
al_container.structure_lst[0].info
```

[34]:
```
{'T': 400, 'P': 0}
```

[35]:
```
al_container.structure_lst
```

[35]:
```
InputList([Al: [0.13389146 3.96541338 4.05893092]
Al: [3.99018226 2.0071096  1.95618182]
Al: [1.98560236 3.88778884 2.0465924 ]
Al: [2.04906472 2.05913422 0.09311447]
pbc: [ True  True  True]
cell:
Cell([[4.079370396328773, 2.497893949200251e-16, 2.497893949200251e-16], [0.0, 3.
↪973148678151775, 2.4328519056175543e-16], [0.0, 0.0, 4.0774098040140591]])
, Al: [0.0070279  4.03832899 0.08383998]
Al: [4.08339864 2.06533333 2.03444326]
Al: [2.20534808 4.07618808 1.94632881]
Al: [1.91118709 2.15964157 0.05514228]
pbc: [ True  True  True]
cell:
Cell([[4.103480856873612, 2.5126573483663535e-16, 2.5126573483663535e-16], [0.0, 4.
↪11316398781314, 2.5185865560217624e-16], [0.0, 0.0, 4.1197543283873851]])
, Al: [3.7382874  0.12171228 4.27645154]
Al: [0.05199557 1.91099383 2.20493355]
Al: [1.92074788 0.03592662 2.13915097]
Al: [1.89264518 1.93451826 0.04368514]
pbc: [ True  True  True]
cell:
```

(continues on next page)

(continued from previous page)

```
Cell([[3.8018380195130206, 2.3279543807366664e-16, 2.3279543807366664e-16], [0.0, 4.
↪003150985990483, 2.451223020748408e-16], [0.0, 0.0, 4.3321106023300721]])
])
```

[ ]:

3.3.4 Data mining using pyiron tables

In this example, the data mining capabilities of pyiron using the `PyironTables` class is demonstrated by computing and contrasting the ground state properties of fcc-Al using various force fields.

[1]:
```
from pyiron import Project
import numpy as np
```

[2]:
```
pr = Project("potential_scan")
```

Creating a dummy job to get list of potentials

In order to get the list of available LAMMPS potentials, a dummy job with an Al bulk structure is created

[3]:
```
dummy_job = pr.create_job(pr.job_type.Lammps, "dummy_job")
dummy_job.structure = pr.create_ase_bulk("Al")
# Chosing only select potentials to run (you can play with these valuess)
num_potentials = 5
potential_list = dummy_job.list_potentials()[:num_potentials]
```

Creating a Murnaghan job for each potential in their respective subprojects

A separate Murnaghan job (to compute equilibrium lattice constant and the bulk modulus) is created and run for every potential

[4]:
```
for pot in potential_list:
    pot_str = pot.replace("-", "_")
    # open a subproject within a project
    with pr.open(pot_str) as pr_sub:
        # no need for unique job name if in different subprojects
        job_name = "murn_Al"
        # Use the subproject to create the jobs
        murn = pr_sub.create_job(pr.job_type.Murnaghan, job_name)
        job_ref = pr_sub.create_job(pr.job_type.Lammps, "Al_ref")
        job_ref.structure = pr.create_ase_bulk("Al", cubic=True)
        job_ref.potential = pot
        job_ref.calc_minimize()
        murn.ref_job = job_ref
        # Some potentials may not work with certain LAMMPS compilations.
        # Therefore, we need to have a little exception handling
        try:
            murn.run()
        except RuntimeError:
            pass
```

```
The job murn_Al was saved and received the ID: 1
The job strain_0_9 was saved and received the ID: 2
The job strain_0_92 was saved and received the ID: 3
The job strain_0_94 was saved and received the ID: 4
The job strain_0_96 was saved and received the ID: 5
The job strain_0_98 was saved and received the ID: 6
The job strain_1_0 was saved and received the ID: 7
The job strain_1_02 was saved and received the ID: 8
The job strain_1_04 was saved and received the ID: 9
The job strain_1_06 was saved and received the ID: 10
The job strain_1_08 was saved and received the ID: 11
The job strain_1_1 was saved and received the ID: 12
job_id:  2 finished
job_id:  3 finished
job_id:  4 finished
job_id:  5 finished
job_id:  6 finished
job_id:  7 finished
job_id:  8 finished
job_id:  9 finished
job_id:  10 finished
job_id:  11 finished
job_id:  12 finished
The job murn_Al was saved and received the ID: 13
The job strain_0_9 was saved and received the ID: 14
The job strain_0_92 was saved and received the ID: 15
The job strain_0_94 was saved and received the ID: 16
The job strain_0_96 was saved and received the ID: 17
The job strain_0_98 was saved and received the ID: 18
The job strain_1_0 was saved and received the ID: 19
The job strain_1_02 was saved and received the ID: 20
The job strain_1_04 was saved and received the ID: 21
The job strain_1_06 was saved and received the ID: 22
The job strain_1_08 was saved and received the ID: 23
The job strain_1_1 was saved and received the ID: 24
job_id:  14 finished
job_id:  15 finished
job_id:  16 finished
job_id:  17 finished
job_id:  18 finished
job_id:  19 finished
job_id:  20 finished
job_id:  21 finished
job_id:  22 finished
job_id:  23 finished
job_id:  24 finished
The job murn_Al was saved and received the ID: 25
The job strain_0_9 was saved and received the ID: 26
The job strain_0_92 was saved and received the ID: 27
The job strain_0_94 was saved and received the ID: 28
The job strain_0_96 was saved and received the ID: 29
The job strain_0_98 was saved and received the ID: 30
The job strain_1_0 was saved and received the ID: 31
The job strain_1_02 was saved and received the ID: 32
The job strain_1_04 was saved and received the ID: 33
The job strain_1_06 was saved and received the ID: 34
The job strain_1_08 was saved and received the ID: 35
```

(continues on next page)

(continued from previous page)

```
The job strain_1_1 was saved and received the ID: 36
job_id:  26 finished
job_id:  27 finished
job_id:  28 finished
job_id:  29 finished
job_id:  30 finished
job_id:  31 finished
job_id:  32 finished
job_id:  33 finished
job_id:  34 finished
job_id:  35 finished
job_id:  36 finished
The job murn_Al was saved and received the ID: 37
The job strain_0_9 was saved and received the ID: 38
```

```
2020-05-01 14:22:19,979 - pyiron_log - WARNING - Job aborted
2020-05-01 14:22:19,982 - pyiron_log - WARNING - LAMMPS (3 Mar 2020)
Reading data file ...
  orthogonal box = (0 0 0) to (3.91023 3.91023 3.91023)
  1 by 1 by 1 MPI processor grid
  reading atoms ...
  4 atoms
  read_data CPU = 0.00191307 secs
ERROR: MEAM library error 3 (src/USER-MEAMC/pair_meamc.cpp:596)
Last command: pair_coeff * * MgAlZn.library.meam Mg Al MgAlZn.parameter.meam Mg Al Zn
```

```
The job murn_Al was saved and received the ID: 39
The job strain_0_9 was saved and received the ID: 40
The job strain_0_92 was saved and received the ID: 41
The job strain_0_94 was saved and received the ID: 42
The job strain_0_96 was saved and received the ID: 43
The job strain_0_98 was saved and received the ID: 44
The job strain_1_0 was saved and received the ID: 45
The job strain_1_02 was saved and received the ID: 46
The job strain_1_04 was saved and received the ID: 47
The job strain_1_06 was saved and received the ID: 48
The job strain_1_08 was saved and received the ID: 49
The job strain_1_1 was saved and received the ID: 50
job_id:  40 finished
job_id:  41 finished
job_id:  42 finished
job_id:  43 finished
job_id:  44 finished
job_id:  45 finished
job_id:  46 finished
job_id:  47 finished
job_id:  48 finished
job_id:  49 finished
job_id:  50 finished
```

If you inspect the job table, you would find that each Murnaghan job generates various small LAMMPS jobs (see column `hamilton`). Some of these jobs might have failed with status `aborted`.

[5]:
```
pr.job_table()
```

[5]:
```
    id    status chemicalformula          job        subjob  \
0    1  finished             Al4      murn_Al      /murn_Al
```

(continues on next page)

(continued from previous page)

```
1    2  finished            Al4   strain_0_9   /strain_0_9
2    3  finished            Al4  strain_0_92  /strain_0_92
3    4  finished            Al4  strain_0_94  /strain_0_94
4    5  finished            Al4  strain_0_96  /strain_0_96
5    6  finished            Al4  strain_0_98  /strain_0_98
6    7  finished            Al4   strain_1_0   /strain_1_0
7    8  finished            Al4  strain_1_02  /strain_1_02
8    9  finished            Al4  strain_1_04  /strain_1_04
9   10  finished            Al4  strain_1_06  /strain_1_06
10  11  finished            Al4  strain_1_08  /strain_1_08
11  12  finished            Al4   strain_1_1   /strain_1_1
12  13  finished            Al4      murn_Al      /murn_Al
13  14  finished            Al4   strain_0_9   /strain_0_9
14  15  finished            Al4  strain_0_92  /strain_0_92
15  16  finished            Al4  strain_0_94  /strain_0_94
16  17  finished            Al4  strain_0_96  /strain_0_96
17  18  finished            Al4  strain_0_98  /strain_0_98
18  19  finished            Al4   strain_1_0   /strain_1_0
19  20  finished            Al4  strain_1_02  /strain_1_02
20  21  finished            Al4  strain_1_04  /strain_1_04
21  22  finished            Al4  strain_1_06  /strain_1_06
22  23  finished            Al4  strain_1_08  /strain_1_08
23  24  finished            Al4   strain_1_1   /strain_1_1
24  25  finished            Al4      murn_Al      /murn_Al
25  26  finished            Al4   strain_0_9   /strain_0_9
26  27  finished            Al4  strain_0_92  /strain_0_92
27  28  finished            Al4  strain_0_94  /strain_0_94
28  29  finished            Al4  strain_0_96  /strain_0_96
29  30  finished            Al4  strain_0_98  /strain_0_98
30  31  finished            Al4   strain_1_0   /strain_1_0
31  32  finished            Al4  strain_1_02  /strain_1_02
32  33  finished            Al4  strain_1_04  /strain_1_04
33  34  finished            Al4  strain_1_06  /strain_1_06
34  35  finished            Al4  strain_1_08  /strain_1_08
35  36  finished            Al4   strain_1_1   /strain_1_1
36  37   aborted            Al4      murn_Al      /murn_Al
37  38   aborted            Al4   strain_0_9   /strain_0_9
38  39  finished            Al4      murn_Al      /murn_Al
39  40  finished            Al4   strain_0_9   /strain_0_9
40  41  finished            Al4  strain_0_92  /strain_0_92
41  42  finished            Al4  strain_0_94  /strain_0_94
42  43  finished            Al4  strain_0_96  /strain_0_96
43  44  finished            Al4  strain_0_98  /strain_0_98
44  45  finished            Al4   strain_1_0   /strain_1_0
45  46  finished            Al4  strain_1_02  /strain_1_02
46  47  finished            Al4  strain_1_04  /strain_1_04
47  48  finished            Al4  strain_1_06  /strain_1_06
48  49  finished            Al4  strain_1_08  /strain_1_08
49  50  finished            Al4   strain_1_1   /strain_1_1

           projectpath  \
0   /home/surendralal/
1   /home/surendralal/
2   /home/surendralal/
3   /home/surendralal/
4   /home/surendralal/
5   /home/surendralal/
```

(continues on next page)

(continued from previous page)

```
6   /home/surendralal/
7   /home/surendralal/
8   /home/surendralal/
9   /home/surendralal/
10  /home/surendralal/
11  /home/surendralal/
12  /home/surendralal/
13  /home/surendralal/
14  /home/surendralal/
15  /home/surendralal/
16  /home/surendralal/
17  /home/surendralal/
18  /home/surendralal/
19  /home/surendralal/
20  /home/surendralal/
21  /home/surendralal/
22  /home/surendralal/
23  /home/surendralal/
24  /home/surendralal/
25  /home/surendralal/
26  /home/surendralal/
27  /home/surendralal/
28  /home/surendralal/
29  /home/surendralal/
30  /home/surendralal/
31  /home/surendralal/
32  /home/surendralal/
33  /home/surendralal/
34  /home/surendralal/
35  /home/surendralal/
36  /home/surendralal/
37  /home/surendralal/
38  /home/surendralal/
39  /home/surendralal/
40  /home/surendralal/
41  /home/surendralal/
42  /home/surendralal/
43  /home/surendralal/
44  /home/surendralal/
45  /home/surendralal/
46  /home/surendralal/
47  /home/surendralal/
48  /home/surendralal/
49  /home/surendralal/


↪      project  \                                                                 ⌊
0                                        programs/pyiron/notebooks/potential_scan/Al_Mg_
↪Mendelev_eam/
1                          programs/pyiron/notebooks/potential_scan/Al_Mg_Mendelev_eam/
↪murn_Al_hdf5/
2                          programs/pyiron/notebooks/potential_scan/Al_Mg_Mendelev_eam/
↪murn_Al_hdf5/
3                          programs/pyiron/notebooks/potential_scan/Al_Mg_Mendelev_eam/
↪murn_Al_hdf5/
4                          programs/pyiron/notebooks/potential_scan/Al_Mg_Mendelev_eam/
↪murn_Al_hdf5/
```

(continues on next page)

(continued from previous page)

```
5                          programs/pyiron/notebooks/potential_scan/Al_Mg_Mendelev_eam/
↪murn_Al_hdf5/
6                          programs/pyiron/notebooks/potential_scan/Al_Mg_Mendelev_eam/
↪murn_Al_hdf5/
7                          programs/pyiron/notebooks/potential_scan/Al_Mg_Mendelev_eam/
↪murn_Al_hdf5/
8                          programs/pyiron/notebooks/potential_scan/Al_Mg_Mendelev_eam/
↪murn_Al_hdf5/
9                          programs/pyiron/notebooks/potential_scan/Al_Mg_Mendelev_eam/
↪murn_Al_hdf5/
10                         programs/pyiron/notebooks/potential_scan/Al_Mg_Mendelev_eam/
↪murn_Al_hdf5/
11                         programs/pyiron/notebooks/potential_scan/Al_Mg_Mendelev_eam/
↪murn_Al_hdf5/
12                                    programs/pyiron/notebooks/potential_scan/Zope_
↪Ti_Al_2003_eam/
13                        programs/pyiron/notebooks/potential_scan/Zope_Ti_Al_2003_eam/
↪murn_Al_hdf5/
14                        programs/pyiron/notebooks/potential_scan/Zope_Ti_Al_2003_eam/
↪murn_Al_hdf5/
15                        programs/pyiron/notebooks/potential_scan/Zope_Ti_Al_2003_eam/
↪murn_Al_hdf5/
16                        programs/pyiron/notebooks/potential_scan/Zope_Ti_Al_2003_eam/
↪murn_Al_hdf5/
17                        programs/pyiron/notebooks/potential_scan/Zope_Ti_Al_2003_eam/
↪murn_Al_hdf5/
18                        programs/pyiron/notebooks/potential_scan/Zope_Ti_Al_2003_eam/
↪murn_Al_hdf5/
19                        programs/pyiron/notebooks/potential_scan/Zope_Ti_Al_2003_eam/
↪murn_Al_hdf5/
20                        programs/pyiron/notebooks/potential_scan/Zope_Ti_Al_2003_eam/
↪murn_Al_hdf5/
21                        programs/pyiron/notebooks/potential_scan/Zope_Ti_Al_2003_eam/
↪murn_Al_hdf5/
22                        programs/pyiron/notebooks/potential_scan/Zope_Ti_Al_2003_eam/
↪murn_Al_hdf5/
23                        programs/pyiron/notebooks/potential_scan/Zope_Ti_Al_2003_eam/
↪murn_Al_hdf5/
24                                      programs/pyiron/notebooks/potential_scan/Al_H_
↪Ni_Angelo_eam/
25                         programs/pyiron/notebooks/potential_scan/Al_H_Ni_Angelo_eam/
↪murn_Al_hdf5/
26                         programs/pyiron/notebooks/potential_scan/Al_H_Ni_Angelo_eam/
↪murn_Al_hdf5/
27                         programs/pyiron/notebooks/potential_scan/Al_H_Ni_Angelo_eam/
↪murn_Al_hdf5/
28                         programs/pyiron/notebooks/potential_scan/Al_H_Ni_Angelo_eam/
↪murn_Al_hdf5/
29                         programs/pyiron/notebooks/potential_scan/Al_H_Ni_Angelo_eam/
↪murn_Al_hdf5/
30                         programs/pyiron/notebooks/potential_scan/Al_H_Ni_Angelo_eam/
↪murn_Al_hdf5/
31                         programs/pyiron/notebooks/potential_scan/Al_H_Ni_Angelo_eam/
↪murn_Al_hdf5/
32                         programs/pyiron/notebooks/potential_scan/Al_H_Ni_Angelo_eam/
↪murn_Al_hdf5/
33                         programs/pyiron/notebooks/potential_scan/Al_H_Ni_Angelo_eam/
↪murn_Al_hdf5/
```

(continues on next page)

(continued from previous page)

```
34                        programs/pyiron/notebooks/potential_scan/Al_H_Ni_Angelo_eam/
↪murn_Al_hdf5/
35                        programs/pyiron/notebooks/potential_scan/Al_H_Ni_Angelo_eam/
↪murn_Al_hdf5/
36                programs/pyiron/notebooks/potential_scan/2018__Dickel_D_E__Mg_Al_Zn__
↪LAMMPS__ipr1/
37  programs/pyiron/notebooks/potential_scan/2018__Dickel_D_E__Mg_Al_Zn__LAMMPS__ipr1/
↪murn_Al_hdf5/
38                      programs/pyiron/notebooks/potential_scan/2000__Landa_A__Al_Pb__
↪LAMMPS__ipr1/
39         programs/pyiron/notebooks/potential_scan/2000__Landa_A__Al_Pb__LAMMPS__ipr1/
↪murn_Al_hdf5/
40         programs/pyiron/notebooks/potential_scan/2000__Landa_A__Al_Pb__LAMMPS__ipr1/
↪murn_Al_hdf5/
41         programs/pyiron/notebooks/potential_scan/2000__Landa_A__Al_Pb__LAMMPS__ipr1/
↪murn_Al_hdf5/
42         programs/pyiron/notebooks/potential_scan/2000__Landa_A__Al_Pb__LAMMPS__ipr1/
↪murn_Al_hdf5/
43         programs/pyiron/notebooks/potential_scan/2000__Landa_A__Al_Pb__LAMMPS__ipr1/
↪murn_Al_hdf5/
44         programs/pyiron/notebooks/potential_scan/2000__Landa_A__Al_Pb__LAMMPS__ipr1/
↪murn_Al_hdf5/
45         programs/pyiron/notebooks/potential_scan/2000__Landa_A__Al_Pb__LAMMPS__ipr1/
↪murn_Al_hdf5/
46         programs/pyiron/notebooks/potential_scan/2000__Landa_A__Al_Pb__LAMMPS__ipr1/
↪murn_Al_hdf5/
47         programs/pyiron/notebooks/potential_scan/2000__Landa_A__Al_Pb__LAMMPS__ipr1/
↪murn_Al_hdf5/
48         programs/pyiron/notebooks/potential_scan/2000__Landa_A__Al_Pb__LAMMPS__ipr1/
↪murn_Al_hdf5/
49         programs/pyiron/notebooks/potential_scan/2000__Landa_A__Al_Pb__LAMMPS__ipr1/
↪murn_Al_hdf5/

                    timestart                   timestop  totalcputime  \
0  2020-05-01 14:20:15.185926 2020-05-01 14:20:52.212726          37.0
1  2020-05-01 14:20:16.872239 2020-05-01 14:20:18.199291           1.0
2  2020-05-01 14:20:20.376998 2020-05-01 14:20:21.474685           1.0
3  2020-05-01 14:20:23.410323 2020-05-01 14:20:24.454505           1.0
4  2020-05-01 14:20:26.407384 2020-05-01 14:20:27.448024           1.0
5  2020-05-01 14:20:29.389853 2020-05-01 14:20:30.457648           1.0
6  2020-05-01 14:20:32.440577 2020-05-01 14:20:33.587692           1.0
7  2020-05-01 14:20:35.659606 2020-05-01 14:20:36.717203           1.0
8  2020-05-01 14:20:39.247825 2020-05-01 14:20:40.631913           1.0
9  2020-05-01 14:20:43.093369 2020-05-01 14:20:44.365442           1.0
10 2020-05-01 14:20:46.700972 2020-05-01 14:20:47.809129           1.0
11 2020-05-01 14:20:49.872971 2020-05-01 14:20:51.002065           1.0
12 2020-05-01 14:20:52.854206 2020-05-01 14:21:40.211332          47.0
13 2020-05-01 14:20:54.595238 2020-05-01 14:20:55.863602           1.0
14 2020-05-01 14:20:58.465134 2020-05-01 14:20:59.616677           1.0
15 2020-05-01 14:21:02.323952 2020-05-01 14:21:03.842627           1.0
16 2020-05-01 14:21:07.120770 2020-05-01 14:21:08.247122           1.0
17 2020-05-01 14:21:10.867935 2020-05-01 14:21:12.084671           1.0
18 2020-05-01 14:21:14.859515 2020-05-01 14:21:15.890379           1.0
19 2020-05-01 14:21:18.333658 2020-05-01 14:21:19.773168           1.0
20 2020-05-01 14:21:23.134672 2020-05-01 14:21:24.701105           1.0
21 2020-05-01 14:21:28.160753 2020-05-01 14:21:29.635477           1.0
22 2020-05-01 14:21:32.177125 2020-05-01 14:21:33.407034           1.0
```

(continues on next page)

(continued from previous page)

```
23 2020-05-01 14:21:36.544373 2020-05-01 14:21:38.079025           1.0
24 2020-05-01 14:21:41.112811 2020-05-01 14:22:14.935040          33.0
25 2020-05-01 14:21:43.292578 2020-05-01 14:21:44.486249           1.0
26 2020-05-01 14:21:46.220651 2020-05-01 14:21:47.239424           1.0
27 2020-05-01 14:21:49.064622 2020-05-01 14:21:50.027115           0.0
28 2020-05-01 14:21:51.711371 2020-05-01 14:21:52.700248           0.0
29 2020-05-01 14:21:54.391263 2020-05-01 14:21:55.421046           1.0
30 2020-05-01 14:21:57.127116 2020-05-01 14:21:58.177664           1.0
31 2020-05-01 14:21:59.836684 2020-05-01 14:22:00.908548           1.0
32 2020-05-01 14:22:02.637902 2020-05-01 14:22:03.654759           1.0
33 2020-05-01 14:22:05.431956 2020-05-01 14:22:06.592121           1.0
34 2020-05-01 14:22:09.286335 2020-05-01 14:22:10.252819           0.0
35 2020-05-01 14:22:12.026812 2020-05-01 14:22:13.233506           1.0
36 2020-05-01 14:22:16.205392                        NaT           NaN
37 2020-05-01 14:22:19.500822                        NaT           NaN
38 2020-05-01 14:22:20.918639 2020-05-01 14:22:56.348776          35.0
39 2020-05-01 14:22:23.362886 2020-05-01 14:22:24.543279           1.0
40 2020-05-01 14:22:26.098609 2020-05-01 14:22:27.456331           1.0
41 2020-05-01 14:22:29.355607 2020-05-01 14:22:30.418893           1.0
42 2020-05-01 14:22:32.522105 2020-05-01 14:22:34.234605           1.0
43 2020-05-01 14:22:36.960119 2020-05-01 14:22:38.166629           1.0
44 2020-05-01 14:22:39.686173 2020-05-01 14:22:40.836256           1.0
45 2020-05-01 14:22:42.989847 2020-05-01 14:22:44.268105           1.0
46 2020-05-01 14:22:46.008623 2020-05-01 14:22:47.372670           1.0
47 2020-05-01 14:22:49.144214 2020-05-01 14:22:50.153294           1.0
48 2020-05-01 14:22:51.746560 2020-05-01 14:22:52.772483           1.0
49 2020-05-01 14:22:54.390591 2020-05-01 14:22:55.348395           0.0

                    computer   hamilton hamversion parentid  masterid
0   pyiron@cmdell17#1#11/11  Murnaghan      0.3.0     None       NaN
1         pyiron@cmdell17#1     Lammps        0.1     None       1.0
2         pyiron@cmdell17#1     Lammps        0.1     None       1.0
3         pyiron@cmdell17#1     Lammps        0.1     None       1.0
4         pyiron@cmdell17#1     Lammps        0.1     None       1.0
5         pyiron@cmdell17#1     Lammps        0.1     None       1.0
6         pyiron@cmdell17#1     Lammps        0.1     None       1.0
7         pyiron@cmdell17#1     Lammps        0.1     None       1.0
8         pyiron@cmdell17#1     Lammps        0.1     None       1.0
9         pyiron@cmdell17#1     Lammps        0.1     None       1.0
10        pyiron@cmdell17#1     Lammps        0.1     None       1.0
11        pyiron@cmdell17#1     Lammps        0.1     None       1.0
12  pyiron@cmdell17#1#11/11  Murnaghan      0.3.0     None       NaN
13        pyiron@cmdell17#1     Lammps        0.1     None      13.0
14        pyiron@cmdell17#1     Lammps        0.1     None      13.0
15        pyiron@cmdell17#1     Lammps        0.1     None      13.0
16        pyiron@cmdell17#1     Lammps        0.1     None      13.0
17        pyiron@cmdell17#1     Lammps        0.1     None      13.0
18        pyiron@cmdell17#1     Lammps        0.1     None      13.0
19        pyiron@cmdell17#1     Lammps        0.1     None      13.0
20        pyiron@cmdell17#1     Lammps        0.1     None      13.0
21        pyiron@cmdell17#1     Lammps        0.1     None      13.0
22        pyiron@cmdell17#1     Lammps        0.1     None      13.0
23        pyiron@cmdell17#1     Lammps        0.1     None      13.0
24  pyiron@cmdell17#1#11/11  Murnaghan      0.3.0     None       NaN
25        pyiron@cmdell17#1     Lammps        0.1     None      25.0
26        pyiron@cmdell17#1     Lammps        0.1     None      25.0
27        pyiron@cmdell17#1     Lammps        0.1     None      25.0
```

(continues on next page)

(continued from previous page)

```
28         pyiron@cmdell17#1      Lammps         0.1      None       25.0
29         pyiron@cmdell17#1      Lammps         0.1      None       25.0
30         pyiron@cmdell17#1      Lammps         0.1      None       25.0
31         pyiron@cmdell17#1      Lammps         0.1      None       25.0
32         pyiron@cmdell17#1      Lammps         0.1      None       25.0
33         pyiron@cmdell17#1      Lammps         0.1      None       25.0
34         pyiron@cmdell17#1      Lammps         0.1      None       25.0
35         pyiron@cmdell17#1      Lammps         0.1      None       25.0
36    pyiron@cmdell17#1#1/11   Murnaghan       0.3.0      None        NaN
37         pyiron@cmdell17#1      Lammps         0.1      None       37.0
38   pyiron@cmdell17#1#11/11   Murnaghan       0.3.0      None        NaN
39         pyiron@cmdell17#1      Lammps         0.1      None       39.0
40         pyiron@cmdell17#1      Lammps         0.1      None       39.0
41         pyiron@cmdell17#1      Lammps         0.1      None       39.0
42         pyiron@cmdell17#1      Lammps         0.1      None       39.0
43         pyiron@cmdell17#1      Lammps         0.1      None       39.0
44         pyiron@cmdell17#1      Lammps         0.1      None       39.0
45         pyiron@cmdell17#1      Lammps         0.1      None       39.0
46         pyiron@cmdell17#1      Lammps         0.1      None       39.0
47         pyiron@cmdell17#1      Lammps         0.1      None       39.0
48         pyiron@cmdell17#1      Lammps         0.1      None       39.0
49         pyiron@cmdell17#1      Lammps         0.1      None       39.0
```

Analysis using `PyironTables`

The idea now is to go over all finished Murnaghan jobs and extract the equilibrium lattice parameter and bulk modulus, and classify them based of the potential used.

Defining filter functions

Since a project can have thousands if not millions of jobs, it is necessary to "filter" the data and only apply the functions (some of which can be computationally expensive) to only this data. In this example, we need to filter jobs that are finished and are of type `Murnaghan`. This can be done in two ways: using the job table i.e. the entries in the database, or using the job itself i.e. using entries in the stored HDF5 file. Below are examples of filter functions acting on the job and the job table respectively.

[6]:
```
# Filtering using the database entries (which are obtained as a pandas Dataframe)
def db_filter_function(job_table):
    # Returns a pandas Series of boolean values (True for entries that have status␣
↪finished
    # and hamilton type Murnaghan.)
    return (job_table.status == "finished") & (job_table.hamilton == "Murnaghan")

# Filtering based on the job
def job_filter_function(job):
    # returns a boolean value if the status of the job
    #is finished and if "murn" is in it's job name
    return (job.status == "finished") & ("murn" in job.job_name)
```

Obviously, using the database is faster in this case but sometimes it might be necessary to filter based on some data that are stored in the HDF5 file of the job. The database filter is applied first followed by the job based filter.

Defining functions that act on jobs

Now we define a set of functions that will be applied on each job to return a certain value. The filtered jobs will be loaded and these functions will be applied on the loaded jobs. The advantage of such functions is that the jobs do not have to be loaded every time such operations are performed. The filtered jobs are loaded once, and then they are passed to these functions to construct the table.

[7]:
```
# Getting equilibrium lattice parameter from Murnaghan jobs
def get_lattice_parameter(job):
    return job["output/equilibrium_volume"] ** (1/3)

# Getting equilibrium bulk modulus from Murnaghan jobs
def get_bm(job):
    return job["output/equilibrium_bulk_modulus"]

# Getting the potential used in each Murnaghan job
def get_pot(job):
    child = job.project.inspect(job["output/id"][0])
    return child["input/potential/Name"]
```

Creating a pyiron table

Now that all the functions are defined, the pyiron table called “table” is created in the following way. This works like a job and can be reloaded at any time.

[8]:
```
%%time
# creating a pyiron table
table = pr.create_table("table")

# assigning a database filter function
table.db_filter_function = db_filter_function

# Alternatively/additionally, a job based filter function can be applied
# (it does the same thing in this case).

#table.filter_function = job_filter_function

# Adding the functions using the labels you like
table.add["a_eq"] = get_lattice_parameter
table.add["bulk_modulus"] = get_bm
table.add["potential"] = get_pot
# Running the table to generate the data
table.run(run_again=True)
```

```
  0%|          | 0/4 [00:00<?, ?it/s]
```

```
The job table was saved and received the ID: 51
```

```
100%|| 4/4 [00:00<00:00, 20.91it/s]
2020-05-01 14:22:57,257 - pyiron_log - WARNING - The job table is being loaded␣
↪instead of running. To re-run use the argument 'run_again=True'
```

```
CPU times: user 531 ms, sys: 156 ms, total: 688 ms
Wall time: 725 ms
```

The output can now be obtained as a pandas DataFrame

[9]:
```
table.get_dataframe()
```

[9]:

	job_id	a_eq	bulk_modulus	potential
0	1	4.045415	89.015487	Al_Mg_Mendelev_eam
1	13	4.049946	80.836779	Zope_Ti_Al_2003_eam
2	25	4.049954	81.040445	Al_H_Ni_Angelo_eam
3	39	4.031246	78.213776	2000--Landa-A--Al-Pb--LAMMPS--ipr1

You can now compare the computed equilibrium lattice constants for each potential to those computed in the NIST database for Al (fcc phase). https://www.ctcms.nist.gov/potentials/system/Al/#Al.

[ ]:
```
```

3.3.5 Phonopy in pyiron

We will use the quasi-harmonic approximation (via PyIron's implementation of the popular phonopy package) to evaluate look at thermal expansion and self-diffusion in Aluminium

[1]:
```
# Generic imports
from pyiron import Project
import numpy as np
%matplotlib  inline
import matplotlib.pylab as plt
import seaborn as sns
```

[2]:
```
pr = Project("PhonopyExample")
pot = '2009--Mendelev-M-I--Al-Mg--LAMMPS--ipr1'
```

Helper functions

Because repeating code is evil.

[3]:
```
def make_phonopy_job(template_job, name):
    """
    Create a phonopy job from a reference job.

    Args:
        template_job (pyiron job): The job to copy.
        name (str): What to call this new job.

    Returns:
        A new phonopy job.
    """
    project = template_job.project

    # What I want:
    # job_type =  template_job.job_type
    # What I have to do instead:
    job_type = pr.job_type.Lammps

    ref = project.create_job(job_type, name + "_ref")
    ref.structure = template_job.get_final_structure().copy()
    ref.potential = template_job.potential
```

(continues on next page)

(continued from previous page)

```
    phono = project.create_job(pr.job_type.PhonopyJob, name)
    phono.ref_job = ref
    return phono
```

[4]:
```
def scale_array(arr, scaler=None, new_range=1.):
    """
    Linearly transforms an array so that values equal to the minimum and maximum of␣
↪the
    `scaler` array are mapped to the range (0, `new_range`). Note that rescaled␣
↪values can
    still lie outside this range if the orignal values of `arr` are outside the␣
↪bounds of
    `scaler`.

    Args:
        arr (np.array): Array to rescale.
        scaler (np.array): Array by which to rescale. Default is `arr`.
        new_range (float): New value for data which was the size `np.amax(scaler)`.
          Default is 1.
    """
    if scaler is None:
        scaler = arr
    return new_range * (arr - np.amin(scaler)) / np.ptp(scaler)
```

Thermal Expansion

What does the QHA say the lattice constant is as a function of temperature?

[5]:
```
pr_te = pr.create_group("ThermalExpansion")
```

Relax the unit cell

If we were doing VASP instead it would be important to do the least computation as possible, so here we'll start by relaxing a simple unit cell to turn into a supercell later.

[6]:
```
job_unit = pr_te.create_job(pr.job_type.Lammps, "UnitCell")
```

[7]:
```
basis = pr_te.create_structure("Al", "fcc", 4.04)
```

[8]:
```
job_unit.structure = basis
job_unit.potential = pot
```

[9]:
```
job_unit.calc_minimize(pressure=0.0)
job_unit.run()
```

```
The job UnitCell was saved and received the ID: 8886147
```

[10]:
```
basis_rel = job_unit.get_final_structure()
```

Relax the bulk supercell

A relaxation which should take zero steps given our starting position!

[11]:
```
job_bulk_1 = pr_te.create_job(pr.job_type.Lammps, "Bulk_1")
# The _1 here refers to the fact that the volume has been rescaled by a factor of "1.0
↪"
# (i.e. it hasn't been rescaled)
```

[12]:
```
n_reps = 3
job_bulk_1.structure = basis_rel.repeat(rep=n_reps)
job_bulk_1.potential = pot
```

[13]:
```
job_bulk_1.structure.plot3d();
```

[14]:
```
job_bulk_1.calc_minimize(pressure=0.0)
job_bulk_1.run()
```

```
The job Bulk_1 was saved and received the ID: 8886148
```

Calculate phonons

Run phonopy on the bulk supercell

[15]:
```
phono_bulk_1 = make_phonopy_job(job_bulk_1, "PhonoBulk_1")
```

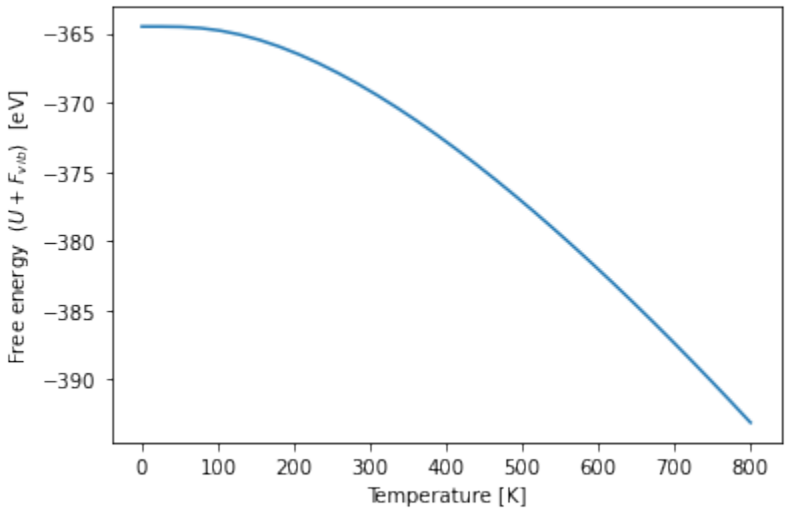
[16]:
```
phono_bulk_1.run()
# Run performs a whole bunch of child calculations
# Each one has the positions slightly deformed in the symmetry-appropriate ways needed
# to get the phonon properties
```

```
The job PhonoBulk_1 was saved and received the ID: 8886149
The job supercell_phonon_0 was saved and received the ID: 8886150
```

[17]:
```
# Let's see what we got...
T_min = 0
T_max = 800 # a bit below melting
T_step = 25
temperatures = np.linspace(T_min, T_max, int((T_max - T_min) / T_step))
tp_bulk_1 = phono_bulk_1.get_thermal_properties(temperatures=temperatures)
# `get_thermal_properties` uses the displacements and forces to generate phonon␣
↪information
```

[18]:
```
U_bulk_1 = job_bulk_1.output.energy_pot[-1]
Fvib_bulk_1 = tp_bulk_1.free_energies
plt.plot(temperatures, U_bulk_1 + Fvib_bulk_1)
plt.xlabel("Temperature [K]")
plt.ylabel("Free energy  ($U+F_{vib}$)  [eV]")
```

[18]:
```
Text(0, 0.5, 'Free energy  ($U+F_{vib}$)  [eV]')
```



Calculate thermal expansivity

Above we have the (QHA approximation to the) free energy as a function of temperature at a fixed volume. To evaluate the thermal expansivity, we need to create the entire F(V,T) surface. To get this, we just loop over jobs like the above, but scaled to have different lattice constants.

[19]:
```
# According to Wikipedia, the thermal expansivity is about 0.0023% / Kelvin
# So at our maximum temperature, we expect around 1.8% expansion
scale_min = -0.005
scale_max = 0.02
scale_step = 0.002
scales = np.linspace(scale_min, scale_max, int((scale_max - scale_min) / scale_step))
```

[20]:
```
# Let's keep things clean by making another sub-directory
pr_scales = pr_te.create_group("ScanScales")
```

[21]:
```
# Loop the phonon calculation over all the volumes
sc_bulk_rel = job_bulk_1.get_final_structure()
bulk_free_energies = np.zeros((len(scales), len(temperatures)))

for i, scale in enumerate(scales):
    name_tail = "_{}".format(str(scale).replace(".", "c").replace('-', 'm'))

    # Make a bulk job with the rescaled structure
    # (already relaxed, by symmetry won't change, calc static will be enough)
    job_bulk = pr_scales.create_job(pr.job_type.Lammps, "Bulk" + name_tail)
    job_bulk.potential = pot
    job_bulk.structure = sc_bulk_rel.apply_strain(epsilon=scale, return_box=True)
    job_bulk.calc_static()
    job_bulk.run()
    U = job_bulk.output.energy_tot[-1]

    # Use that job as a reference for a phonopy job
    phono_bulk = make_phonopy_job(job_bulk, "PhonoBulk" + name_tail)
    phono_bulk.run()
    tp_bulk = phono_bulk.get_thermal_properties(temperatures=temperatures)
```

(continues on next page)

(continued from previous page)

```
    Fvib = tp_bulk.free_energies

    # Fill in the row of free energies for this volume
    bulk_free_energies[i] = Fvib + U
```

```
The job Bulk_m0c005 was saved and received the ID: 8886151
The job PhonoBulk_m0c005 was saved and received the ID: 8886152
The job supercell_phonon_0 was saved and received the ID: 8886153
The job Bulk_m0c002727272727272727 was saved and received the ID: 8886154
The job PhonoBulk_m0c002727272727272727 was saved and received the ID: 8886155
The job supercell_phonon_0 was saved and received the ID: 8886156
The job Bulk_m0c000454545454545454 was saved and received the ID: 8886157
The job PhonoBulk_m0c000454545454545454 was saved and received the ID: 8886158
The job supercell_phonon_0 was saved and received the ID: 8886159
The job Bulk_0c0018181818181818186 was saved and received the ID: 8886160
The job PhonoBulk_0c0018181818181818186 was saved and received the ID: 8886161
The job supercell_phonon_0 was saved and received the ID: 8886162
The job Bulk_0c004090909090909092 was saved and received the ID: 8886163
The job PhonoBulk_0c004090909090909092 was saved and received the ID: 8886164
The job supercell_phonon_0 was saved and received the ID: 8886165
The job Bulk_0c006363636363636366 was saved and received the ID: 8886166
The job PhonoBulk_0c006363636363636366 was saved and received the ID: 8886167
The job supercell_phonon_0 was saved and received the ID: 8886168
The job Bulk_0c008636363636363636 was saved and received the ID: 8886169
The job PhonoBulk_0c008636363636363636 was saved and received the ID: 8886170
The job supercell_phonon_0 was saved and received the ID: 8886171
The job Bulk_0c01090909090909091 was saved and received the ID: 8886172
The job PhonoBulk_0c01090909090909091 was saved and received the ID: 8886173
The job supercell_phonon_0 was saved and received the ID: 8886174
The job Bulk_0c013181818181818183 was saved and received the ID: 8886175
The job PhonoBulk_0c013181818181818183 was saved and received the ID: 8886176
The job supercell_phonon_0 was saved and received the ID: 8886177
The job Bulk_0c015454545454545457 was saved and received the ID: 8886178
The job PhonoBulk_0c015454545454545457 was saved and received the ID: 8886179
The job supercell_phonon_0 was saved and received the ID: 8886180
The job Bulk_0c01772727272727273 was saved and received the ID: 8886182
The job PhonoBulk_0c01772727272727273 was saved and received the ID: 8886183
The job supercell_phonon_0 was saved and received the ID: 8886184
The job Bulk_0c02 was saved and received the ID: 8886186
The job PhonoBulk_0c02 was saved and received the ID: 8886187
The job supercell_phonon_0 was saved and received the ID: 8886188
```

[22]:
```
# The lattice constant is probably a more informative value than the 0K-relative␣
↪strain
latts = basis_rel.cell[0][0] * scales
```

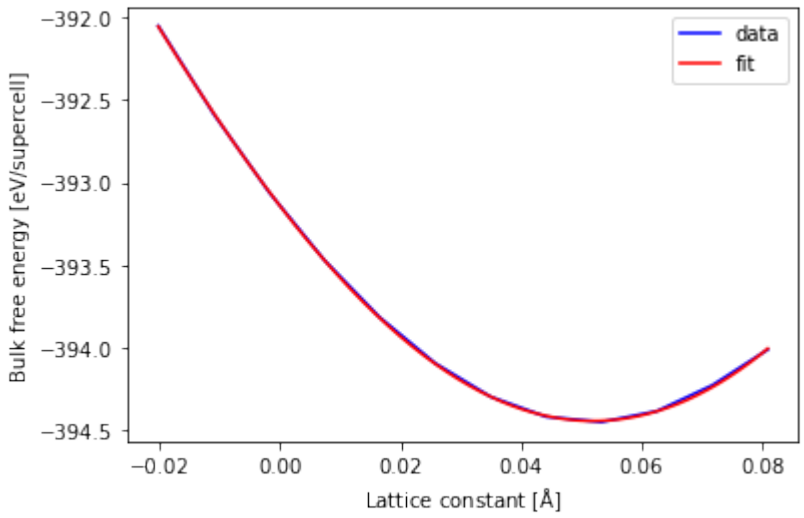
[23]:
```
# At each temperature, find the optimal volume by a simple quadratic fit
# ...Wait, which order fit will be good enough? Let's just spot-check
free_en = bulk_free_energies[:, -1]
plt.plot(latts, free_en, color='b', label='data')

# We'll plot the fit on a much denser mesh
fit_deg = 4
p = np.polyfit(latts, free_en, deg=fit_deg)
dense_latts = np.linspace(np.amin(latts), np.amax(latts), 1000)
#dense_latts = np.linspace(0, 10, 1000)
```

(continues on next page)

(continued from previous page)

```
plt.plot(dense_latts, np.polyval(p=p, x=dense_latts), color='r', label='fit')
plt.xlabel('Lattice constant [$\mathrm{\AA}$]')
plt.ylabel('Bulk free energy [eV/supercell]')
plt.legend()
# Ok, a fourth-order fit seems perfectly reasonable
```

[23]:
```
<matplotlib.legend.Legend at 0x2b151bd2ac10>
```



[24]:
```
# Now find optimal temperatures
best_latts = np.zeros(len(temperatures))
best_latt_guess = basis_rel.cell[0][0]
for i, T in enumerate(temperatures):
    free_en = bulk_free_energies[:, i]
    p = np.polyfit(latts, free_en, deg=fit_deg)
    extrema = np.roots(np.polyder(p, m=1)).real  # Find where first-derivative is zero
    best_latts[i] = extrema[np.argmin(np.abs(extrema - best_latt_guess))]
```

[25]:
```
# Check that they're resonable
print(best_latt_guess, '\n', best_latts)
```

```
4.045270475668763
 [0.11555233 0.11352406 0.10694882 0.10163624 0.09885196 1.43573459
 0.77014253 0.60322527 0.51918649 0.46683335 0.43047109 0.40346556
 0.38247104 0.36559688 0.35168556 0.33998477 0.32998232 0.32131628
 0.31372298 0.30700538 0.30101301 0.29562879 0.29076015 0.28633285
 0.28228657 0.2785718  0.27514744 0.27197909 0.2690377  0.26629857
 0.26374056 0.26134543]
```

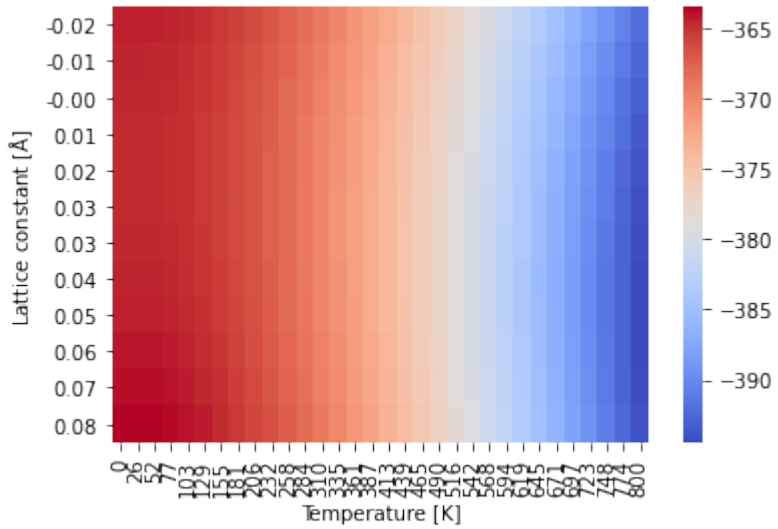
[26]:
```
# Let's look at the landscape
fig, ax = plt.subplots()
sns.heatmap(bulk_free_energies, ax=ax, cmap="coolwarm",
            xticklabels=['{:,.0f}'.format(T) for T in temperatures],
            yticklabels=['{:,.2f}'.format(a) for a in latts])
ax.set_xlabel("Temperature [K]")
ax.set_ylabel("Lattice constant [$\mathrm{\AA}$]")

# Overlaying the optimal path takes a couple changes of variables
# since the heatmap is plotting integer cells
```

(continues on next page)

(continued from previous page)

```
ax.plot(scale_array(temperatures, new_range=len(temperatures)),
        scale_array(best_latts, scaler=latts, new_range=len(latts)),
        color='k')
```

[26]:
```
[<matplotlib.lines.Line2D at 0x2b1519223a90>]
```



Vacancies and diffusion

Another common use of QHA is to calculate the pre-factor for migration in a diffusion event.

In particular, the diffusion jump barrier looks like $\omega_0 = \nu_0^\star \exp(-H_\mathrm{m}/k_\mathrm{B}T)$, where $\nu_0^\star = \prod_{i=1}^{3N-3} \nu_i^\mathrm{IS} / \prod_{i=1}^{3N-4} \nu_i^\mathrm{TS}$, with IS and TS indicating the initial and transition states, respectively. Note that the transition state is missing a single frequency, which is from the instability of the transition state. It's either an imaginary mode, which I think means a negative frequency. Meanwhile, H_m is the enthalpic barrier (difference between the initial and transition states) and $k_\mathrm{B}T$ is the usual thermal energy term.

Typically, these sorts of investigations use the nudged elastic band (NEB) to find the 0K transition state. You can do that with our new flexible jobs, but we'll save that for later. For now we'll just "approximate" the transition state with a simple linear interpolation.

Stable vacancy structures

Let's start by generating and relaxing the initial and final states

[27]:
```
pr_vac = pr.create_group("Vacancies")
```

[28]:
```
# Find two adjacent sites
print(job_bulk_1.structure.positions[0])
print(job_bulk_1.structure.positions[1])
# Yep, 1 and 2 will do
```

```
[0. 0. 0.]
[ 2.02263524e+00  2.02263524e+00 -7.63052415e-33]
```

[29]:
```
job_vac_i = pr_vac.create_job(pr.job_type.Lammps, "VacancyInitial")
job_vac_f = pr_vac.create_job(pr.job_type.Lammps, "VacancyFinal")

job_vac_i.potential = pot
job_vac_f.potential = pot
```

[30]:
```
sc_vac_i = sc_bulk_rel.copy()
sc_vac_i.pop(0)
job_vac_i.structure = sc_vac_i

sc_vac_f = sc_bulk_rel.copy()
sc_vac_f.pop(1)
job_vac_f.structure = sc_vac_f
```

[31]:
```
# Relax the new vacancy structures
job_vac_i.calc_minimize(pressure=0.0)
job_vac_i.run()

job_vac_f.calc_minimize(pressure=0.0)
job_vac_f.run()
```

```
The job VacancyInitial was saved and received the ID: 8886189
The job VacancyFinal was saved and received the ID: 8886190
```

DOS

The pyiron implementation of phonopy makes it very easy to look at the DOS. Let's see what the effect is of introducing a vacancy, and confirm that our two vacancies are equivalent.

[32]:
```
phon_vac_i = make_phonopy_job(job_vac_i, "PhonoVacInitial")
phon_vac_f = make_phonopy_job(job_vac_f, "PhonoVacFinal")
```

[33]:
```
phon_vac_i.run()
tp_vac_i = phon_vac_i.get_thermal_properties(temperatures=temperatures)

phon_vac_f.run()
tp_vac_f = phon_vac_i.get_thermal_properties(temperatures=temperatures)

# Note that the vacancy structures spawn many more child processes
# This is because the vacancy structure has lower symmetry
```

```
The job PhonoVacInitial was saved and received the ID: 8886191
The job supercell_phonon_0 was saved and received the ID: 8886192
The job supercell_phonon_1 was saved and received the ID: 8886193
The job supercell_phonon_2 was saved and received the ID: 8886194
The job supercell_phonon_3 was saved and received the ID: 8886195
The job supercell_phonon_4 was saved and received the ID: 8886196
The job supercell_phonon_5 was saved and received the ID: 8886197
The job supercell_phonon_6 was saved and received the ID: 8886198
The job supercell_phonon_7 was saved and received the ID: 8886199
The job supercell_phonon_8 was saved and received the ID: 8886200
The job supercell_phonon_9 was saved and received the ID: 8886201
The job supercell_phonon_10 was saved and received the ID: 8886202
The job supercell_phonon_11 was saved and received the ID: 8886203
The job supercell_phonon_12 was saved and received the ID: 8886204
```

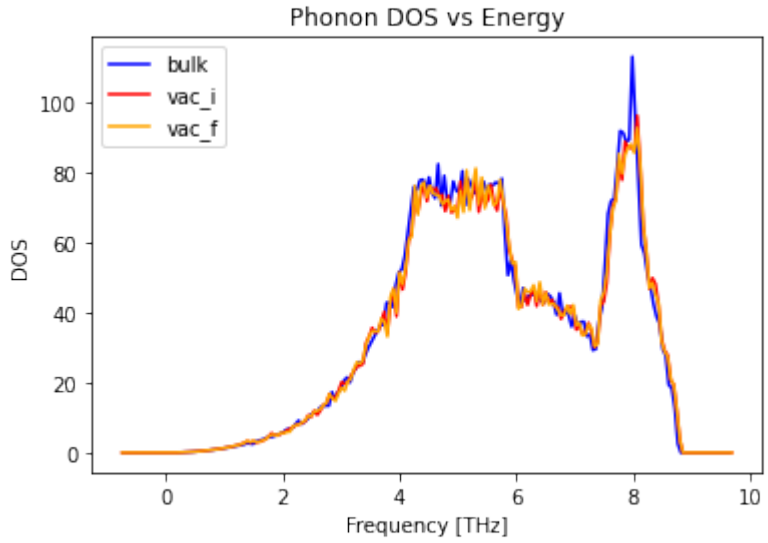
(continues on next page)

(continued from previous page)

```
The job supercell_phonon_13 was saved and received the ID: 8886205
The job supercell_phonon_14 was saved and received the ID: 8886207
The job supercell_phonon_15 was saved and received the ID: 8886208
The job supercell_phonon_16 was saved and received the ID: 8886209
The job supercell_phonon_17 was saved and received the ID: 8886210
The job supercell_phonon_18 was saved and received the ID: 8886211
The job supercell_phonon_19 was saved and received the ID: 8886212
The job supercell_phonon_20 was saved and received the ID: 8886213
The job PhonoVacFinal was saved and received the ID: 8886214
The job supercell_phonon_0 was saved and received the ID: 8886215
The job supercell_phonon_1 was saved and received the ID: 8886216
The job supercell_phonon_2 was saved and received the ID: 8886217
The job supercell_phonon_3 was saved and received the ID: 8886218
The job supercell_phonon_4 was saved and received the ID: 8886219
The job supercell_phonon_5 was saved and received the ID: 8886220
The job supercell_phonon_6 was saved and received the ID: 8886221
The job supercell_phonon_7 was saved and received the ID: 8886222
The job supercell_phonon_8 was saved and received the ID: 8886223
The job supercell_phonon_9 was saved and received the ID: 8886225
The job supercell_phonon_10 was saved and received the ID: 8886226
The job supercell_phonon_11 was saved and received the ID: 8886227
The job supercell_phonon_12 was saved and received the ID: 8886228
The job supercell_phonon_13 was saved and received the ID: 8886229
The job supercell_phonon_14 was saved and received the ID: 8886230
The job supercell_phonon_15 was saved and received the ID: 8886231
The job supercell_phonon_16 was saved and received the ID: 8886232
The job supercell_phonon_17 was saved and received the ID: 8886233
The job supercell_phonon_18 was saved and received the ID: 8886234
The job supercell_phonon_19 was saved and received the ID: 8886235
The job supercell_phonon_20 was saved and received the ID: 8886236
```

[34]:
```
fig, ax = plt.subplots()
phono_bulk_1.plot_dos(ax=ax, color='b', label='bulk')
phon_vac_i.plot_dos(ax=ax, color='r', label='vac_i')
phon_vac_f.plot_dos(ax=ax, color='orange', label='vac_f')
plt.legend()
```

[34]:
```
<matplotlib.legend.Legend at 0x2b157cd72890>
```



Attempt frequency

Now we get the attempt frequency by comparing the individual phonon spectra of initial and transition states

[35]:
```
# Interpolate initial and final positions to guesstimate the transition state
sc_vac_ts = sc_vac_i.copy()
sc_vac_ts.positions = 0.5 * (sc_vac_i.positions + sc_vac_f.positions)
```

[36]:
```
job_vac_ts = pr_vac.create_job(pr.job_type.Lammps, "VacancyTransition")
job_vac_ts.potential = pot
job_vac_ts.structure = sc_vac_ts
```

[37]:
```
# We _don't_ relax this job, or it would fall into the initial or final state!
job_vac_ts.calc_static()
job_vac_ts.run()
```

```
The job VacancyTransition was saved and received the ID: 8886237
```

[38]:
```
phon_vac_ts = make_phonopy_job(job_vac_ts, "PhonoVacTransition")
```

[39]:
```
phon_vac_ts.run()
tp_vac_ts = phon_vac_ts.get_thermal_properties(temperatures=temperatures)
```

```
The job PhonoVacTransition was saved and received the ID: 8886238
The job supercell_phonon_0 was saved and received the ID: 8886239
The job supercell_phonon_1 was saved and received the ID: 8886240
The job supercell_phonon_2 was saved and received the ID: 8886241
The job supercell_phonon_3 was saved and received the ID: 8886242
The job supercell_phonon_4 was saved and received the ID: 8886243
The job supercell_phonon_5 was saved and received the ID: 8886244
The job supercell_phonon_6 was saved and received the ID: 8886245
The job supercell_phonon_7 was saved and received the ID: 8886246
The job supercell_phonon_8 was saved and received the ID: 8886247
The job supercell_phonon_9 was saved and received the ID: 8886248
```

(continues on next page)

(continued from previous page)

```
The job supercell_phonon_10 was saved and received the ID: 8886249
The job supercell_phonon_11 was saved and received the ID: 8886250
The job supercell_phonon_12 was saved and received the ID: 8886251
The job supercell_phonon_13 was saved and received the ID: 8886252
The job supercell_phonon_14 was saved and received the ID: 8886253
The job supercell_phonon_15 was saved and received the ID: 8886254
The job supercell_phonon_16 was saved and received the ID: 8886255
The job supercell_phonon_17 was saved and received the ID: 8886256
The job supercell_phonon_18 was saved and received the ID: 8886257
The job supercell_phonon_19 was saved and received the ID: 8886258
The job supercell_phonon_20 was saved and received the ID: 8886259
The job supercell_phonon_21 was saved and received the ID: 8886260
The job supercell_phonon_22 was saved and received the ID: 8886261
The job supercell_phonon_23 was saved and received the ID: 8886262
The job supercell_phonon_24 was saved and received the ID: 8886263
The job supercell_phonon_25 was saved and received the ID: 8886264
The job supercell_phonon_26 was saved and received the ID: 8886265
The job supercell_phonon_27 was saved and received the ID: 8886267
The job supercell_phonon_28 was saved and received the ID: 8886269
The job supercell_phonon_29 was saved and received the ID: 8886270
The job supercell_phonon_30 was saved and received the ID: 8886271
The job supercell_phonon_31 was saved and received the ID: 8886272
The job supercell_phonon_32 was saved and received the ID: 8886273
The job supercell_phonon_33 was saved and received the ID: 8886274
The job supercell_phonon_34 was saved and received the ID: 8886275
The job supercell_phonon_35 was saved and received the ID: 8886276
The job supercell_phonon_36 was saved and received the ID: 8886277
The job supercell_phonon_37 was saved and received the ID: 8886278
The job supercell_phonon_38 was saved and received the ID: 8886279
The job supercell_phonon_39 was saved and received the ID: 8886280
The job supercell_phonon_40 was saved and received the ID: 8886281
The job supercell_phonon_41 was saved and received the ID: 8886282
The job supercell_phonon_42 was saved and received the ID: 8886283
The job supercell_phonon_43 was saved and received the ID: 8886284
The job supercell_phonon_44 was saved and received the ID: 8886285
The job supercell_phonon_45 was saved and received the ID: 8886286
The job supercell_phonon_46 was saved and received the ID: 8886287
The job supercell_phonon_47 was saved and received the ID: 8886288
The job supercell_phonon_48 was saved and received the ID: 8886289
The job supercell_phonon_49 was saved and received the ID: 8886290
The job supercell_phonon_50 was saved and received the ID: 8886291
The job supercell_phonon_51 was saved and received the ID: 8886292
The job supercell_phonon_52 was saved and received the ID: 8886293
The job supercell_phonon_53 was saved and received the ID: 8886294
The job supercell_phonon_54 was saved and received the ID: 8886295
The job supercell_phonon_55 was saved and received the ID: 8886296
The job supercell_phonon_56 was saved and received the ID: 8886297
The job supercell_phonon_57 was saved and received the ID: 8886298
The job supercell_phonon_58 was saved and received the ID: 8886299
The job supercell_phonon_59 was saved and received the ID: 8886300
The job supercell_phonon_60 was saved and received the ID: 8886301
The job supercell_phonon_61 was saved and received the ID: 8886302
The job supercell_phonon_62 was saved and received the ID: 8886303
The job supercell_phonon_63 was saved and received the ID: 8886304
The job supercell_phonon_64 was saved and received the ID: 8886305
The job supercell_phonon_65 was saved and received the ID: 8886306
The job supercell_phonon_66 was saved and received the ID: 8886307
```

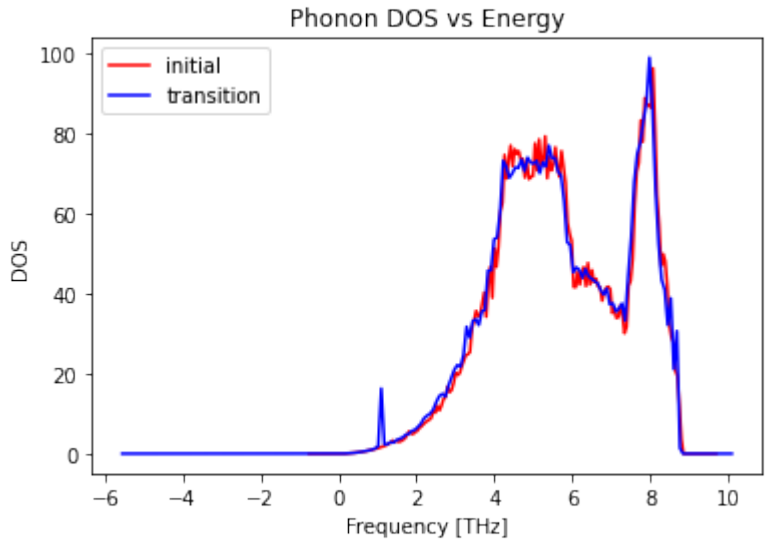
(continues on next page)

(continued from previous page)

```
The job supercell_phonon_67 was saved and received the ID: 8886308
The job supercell_phonon_68 was saved and received the ID: 8886309
The job supercell_phonon_69 was saved and received the ID: 8886310
The job supercell_phonon_70 was saved and received the ID: 8886311
The job supercell_phonon_71 was saved and received the ID: 8886312
The job supercell_phonon_72 was saved and received the ID: 8886313
The job supercell_phonon_73 was saved and received the ID: 8886314
The job supercell_phonon_74 was saved and received the ID: 8886315
The job supercell_phonon_75 was saved and received the ID: 8886316
The job supercell_phonon_76 was saved and received the ID: 8886317
The job supercell_phonon_77 was saved and received the ID: 8886318
The job supercell_phonon_78 was saved and received the ID: 8886319
The job supercell_phonon_79 was saved and received the ID: 8886320
The job supercell_phonon_80 was saved and received the ID: 8886321
The job supercell_phonon_81 was saved and received the ID: 8886322
The job supercell_phonon_82 was saved and received the ID: 8886323
The job supercell_phonon_83 was saved and received the ID: 8886324
The job supercell_phonon_84 was saved and received the ID: 8886325
The job supercell_phonon_85 was saved and received the ID: 8886326
The job supercell_phonon_86 was saved and received the ID: 8886327
The job supercell_phonon_87 was saved and received the ID: 8886328
The job supercell_phonon_88 was saved and received the ID: 8886329
The job supercell_phonon_89 was saved and received the ID: 8886330
The job supercell_phonon_90 was saved and received the ID: 8886331
The job supercell_phonon_91 was saved and received the ID: 8886332
The job supercell_phonon_92 was saved and received the ID: 8886333
The job supercell_phonon_93 was saved and received the ID: 8886334
The job supercell_phonon_94 was saved and received the ID: 8886335
```

[40]:
```
# The transition state has an imaginary mode (frequency < 0), let's see it
fig, ax = plt.subplots()
phon_vac_i.plot_dos(ax=ax, color='r', label='initial')
phon_vac_ts.plot_dos(ax=ax, color='b', label='transition')
plt.legend()
```

[40]:
```
<matplotlib.legend.Legend at 0x2b157c09cc90>
```



To calculate the attempt frequency, we'll ignore both the negative mode of the transition state (which we were warned about in the equation), as well as the three frequencies which correspond to rigid translation and are very near zero, and sometimes dip to be negative. Phonopy sorts the frequencies by magnitude, so we can just skip the first three and four for the initial and transition states, respectively. We take them at q=0.

[41]:
```
freq_i = phon_vac_i.phonopy.get_frequencies(0)[3:]
freq_ts = phon_vac_i.phonopy.get_frequencies(0)[4:]
```

[42]:
```
print(np.prod(freq_i))
```

```
6.870293244293476e+236
```

Recall: $\nu_0^\star = \prod_{i=1}^{3N-3} \nu_i^{\mathrm{IS}} / \prod_{i=1}^{3N-4} \nu_i^{\mathrm{TS}}$

[43]:
```
# Products are dangerous beasts, so we'll do a little numeric magic
nu = np.prod(freq_i[:-1] / freq_ts) * freq_i[-1]
print("Attempt frequency is ", nu, "THz (10^-12 s)")
```

```
Attempt frequency is  2.6826762430167848 THz (10^-12 s)
```

Mantina *et al.* (PRL 2008) report $\nu = 19.3$ THz using DFT and NEB, so our linearly-interpolated "transition state" with EAM is actually not doing so poorly.

There are many more things you can do with phonopy, including looking directly at the force constants, the Hessian matrix, etc. But hopefully this is a useful starting point.

[ ]:
```
```

3.3.6 Workfunction of hcp (0001) surfaces

In this notebook, we will show how to calculate the workfunction of selected hcp(0001) surfaces using VASP. Please keep in mind that the parameters used here give no converged results. They have been chosen to demonstrate the workflow using inexpensive calculations. For converged results, parameters such as lattice parameters, plane-wave energy cutoffs, reciprocal space sampling or the need to perform spin polarized calculations have to be carefully chosen

[1]:
```
import numpy as np
%matplotlib inline
import matplotlib.pylab as plt
import pandas as pd
import time
```

[2]:
```
from pyiron import Project
```

[3]:
```
pr = Project("hcp_workfunction")
```

Calculating the Workfunction of Mg(0001)

Structure creation

We use the `create_surface()` function which uses the ASE surface generator to build our surface slab structure

[4]:
```
 # Now we set-up the Mg (0001) surface
a = 3.1919
c = 5.1852

# Vacuum region to break the periodicity along the z-axis
vac = 10
size = (2, 2, 4)
Mg_0001 = pr.create_surface("Mg",
                            surface_type="hcp0001",
                            size=size,
                            a=a,
                            c=c,
                            orthogonal=True,
                            vacuum=vac)
Mg_0001.plot3d()
```

```
NGLWidget()
```

Using selective dynamics

We use selective dynamics to restrict relaxation to the surface atoms (first and last Mg layers). We use the advanced array indexing options available in the NumPy package (see here) to detect which atoms are at the surface and then freeze the rest

[5]:
```
# Initially freeze all the atoms
Mg_0001.add_tag(selective_dynamics=[False, False, False])

# Find which atoms are at the surface
# (based on the z-coordinate)
pos_z = Mg_0001.positions[:, 2]
z_min, z_max = np.min(pos_z), np.max(pos_z)
eps = 1e-4
relax_indices = np.argwhere(((pos_z - eps) > z_min)
                            & ((pos_z + eps) < z_max ))
relax_indices  = relax_indices.flatten()

# Now allow these atoms to relax

Mg_0001.selective_dynamics[relax_indices] = [True, True, True]
```

Setup and execution

To automate the calculation we define a function that has as input the project object, structure, job_name, Fermi smearing width, the type of k-point sampling and the plane-wave energy cutoff

[6]:
```
def get_ham(proj, basis, name, sigma=0.1, mesh="GP", encut=350):
    ham = proj.create_job(pr.job_type.Vasp, name)
    ham.set_convergence_precision(electronic_energy=1e-7,
                                  ionic_energy=1e-2)
    # Setting fermi-smearing
    ham.set_occupancy_smearing(smearing="fermi", width=sigma)
    # Ionic_minimization
    ham.calc_minimize(ionic_steps=100,
                      electronic_steps=60,
                      retain_electrostatic_potential=True,
                      pressure=None)
    ham.structure = basis
    ham.set_encut(encut=encut)
    if mesh == "GP":
        # Only the Gamma point
        ham.set_kpoints(scheme="GP")
    elif len(mesh) == 3:
        ham.set_kpoints(mesh=mesh)
    return ham
```

[7]:
```
ham_vasp = get_ham(proj=pr,
                   basis=Mg_0001,
                   name="Mg_0001",
                   sigma=0.1,
                   mesh="GP",
                   encut=350)
```

Submitting to the queue (optional)

If you use a cluster installation of pyiron, you can send the created jobs to the cluster by specifying the name of the queue and the number of cores

[8]:
```
# queue = ham_vasp.server.list_queues()[-1]
# ham_vasp.server.queue = queue
# ham_vasp.server.cores = 20
```

Choosing an appropriate executable

[9]:
```
ham_vasp.executable.available_versions
```

[9]:
```
['5.3',
 '5.3_col',
 '5.3_col_mpi',
 '5.3_mpi',
 '5.4',
 '5.4.4',
 '5.4.4_gam',
```

(continues on next page)

(continued from previous page)

```
 '5.4.4_gam_mpi',
 '5.4.4_mpi',
 '5.4.4_ncl',
 '5.4.4_ncl_mpi',
 '5.4.4_std',
 '5.4.4_std_mpi',
 '5.4_gamma',
 '5.4_gamma_mpi',
 '5.4_mpi']
```

Since this example uses the Γ point only, we can use the VASP Gamma-only version. If you use more k-points choose an appropriate executable

[10]:
```
ham_vasp.executable.version = "5.4_gamma"
```

Execution

The job is ready for execution

[11]:
```
ham_vasp.run()
```

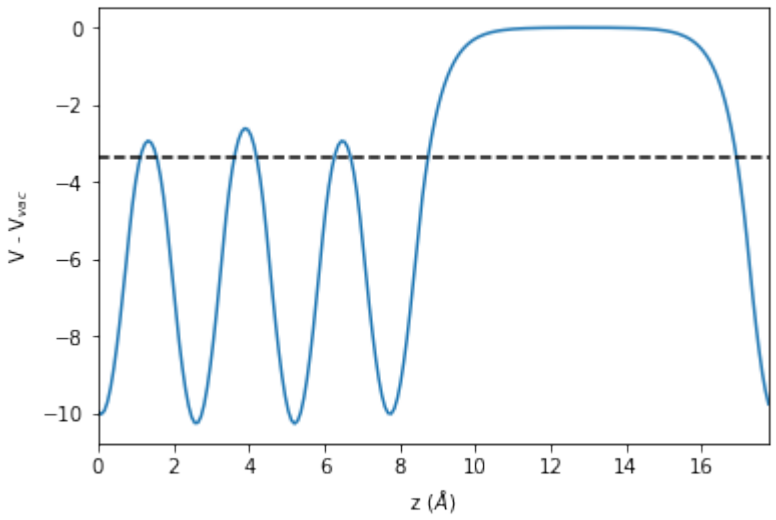
Post processing

To analyze the results we ensure that the job is finished (the `if` statement in the first line). We then compute the work function by subtracting the Fermi-level from the vacuum level

$\Phi = V_{vac} - \epsilon_F$

[12]:
```
if ham_vasp.status.finished:
    # Get the electrostatic potential
    epot = ham_vasp.get_electrostatic_potential()

    # Compute the lateral average along the z-axis (ind=2)
    epot_z = epot.get_average_along_axis(ind=2)

    # Get the final relaxed structure from the simulation
    struct = ham_vasp.get_structure(iteration_step=-1)
    r = np.linalg.norm(struct.cell[2])
    z = np.linspace(0, r, len(epot_z))

    # Computing the vacuum-level
    vac_level = np.max(epot_z)

    # Get the electronic structure
    es = ham_vasp.get_electronic_structure()
    print("wf:", vac_level - es.efermi)
    plt.plot(z, epot_z - vac_level)
    plt.xlim(0, r)
    plt.axhline(es.efermi - vac_level,
                color="black",
                linestyle="dashed")
    plt.xlabel("z ($\AA$)")
    plt.ylabel("V - V$_{vac}$");
```

```
wf: 3.37343565133
```



Looping over a series of hcp(0001) surfaces

We now repeat the workflow for a set of hcp metals (the chosen lattice parameters are approximate). Note that if you use the same naming convention, pyiron detects that a job with the same name exists ("Mg_0001") and loads the output from this calculation rather than launch a new job with the same name.

[13]:
```
hcp_dict = {"Zn": {"a":2.6649, "c": 4.9468},
            "Mg": {"a": 3.1919, "c": 5.1852},
            "Co": {"a": 2.5071 , "c": 4.0695},
            "Ru": {"a": 2.7059 , "c": 4.2815}}
```

[14]:
```
vac = 10
size = (2, 2, 4)
for element, lattice_parameters in hcp_dict.items():
    surf = pr.create_surface(element,
                             surface_type="hcp0001",
                             size=size,
                             a=lattice_parameters["a"],
                             c=lattice_parameters["c"],
                             orthogonal=True, vacuum=vac)
    surf.add_tag(selective_dynamics=[False, False, False])
    pos_z = surf.positions[:, 2]
    z_min, z_max = np.min(pos_z), np.max(pos_z)
    eps = 1e-4
    relax_indices = np.argwhere(((pos_z - eps) > z_min)
                                & ((pos_z + eps) < z_max ))
    relax_indices  = relax_indices.flatten()
    surf.selective_dynamics[relax_indices] = [True, True, True]
    job_name = "{}_0001".format(element)
    ham = get_ham(pr, surf,
                  name=job_name,
                  sigma=0.1,
                  mesh="GP",
                  encut=350)
```

(continues on next page)

(continued from previous page)

```
    #ham.server.cores = 20
    #ham.server.queue = queue
    ham.executable.version = '5.4_gamma'
    ham.run()
```

Loading and analyzing

Now we iterate over all jobs in this project and calculate the workfunction. We also time how long the cell takes to execute

[15]:
```
t1 = time.time()
for ham in pr.iter_jobs():
    if ham.status.finished:
            final_struct = ham.get_structure(iteration_step=-1)
            elec_structure = ham.get_electronic_structure()
            e_Fermi = elec_structure.efermi
            epot = ham.get_electrostatic_potential()
            epot_z = epot.get_average_along_axis(ind=2)
            vacuum_level = np.max(epot_z)
            wf = vacuum_level - e_Fermi
            element = final_struct.get_majority_species()[-1]
            hcp_dict[element]["work_func"] = wf
t2 = time.time()
print("time: {}s".format(t2-t1))
```

```
time: 9.250723838806152s
```

Compiling data in a table using pandas

[16]:
```
df = pd.DataFrame(hcp_dict).T
df = df.rename(columns={'a': 'a [A]',
                        'c': 'c [A]',
                        'work_func': 'wf [eV]'})
print(df.round(3))
```

```
    a [A]  c [A]  wf [eV]
Co  2.507  4.069    5.569
Mg  3.192  5.185    3.373
Ru  2.706  4.282    5.305
Zn  2.665  4.947    3.603
```

[ ]:

3.3.7 Molecular dynamics simulations of bulk water

In this example, we show how to perform molecular dynamics of bulk water using the popular interatomic TIP3P potential (W. L. Jorgensen et. al.) and LAMMPS.

[1]:
```
import numpy as np
%matplotlib inline
import matplotlib.pylab as plt
from pyiron import Project
import ase.units as units
import pandas
```

[2]:
```
pr = Project("tip3p_water")
```

Creating the initial structure

We will setup a cubic simulation box consisting of 27 water molecules density density is 1 g/cm^3. The target density is achieved by determining the required size of the simulation cell and repeating it in all three spatial dimensions

[3]:
```
density = 1.0e-24  # g/A^3
n_mols = 27
mol_mass_water = 18.015 # g/mol

# Determining the supercell size size
mass = mol_mass_water * n_mols / units.mol  # g
vol_h2o = mass / density # in A^3
a = vol_h2o ** (1./3.) # A

# Constructing the unitcell
n = int(round(n_mols ** (1. / 3.)))

dx = 0.7
r_O = [0, 0, 0]
r_H1 = [dx, dx, 0]
r_H2 = [-dx, dx, 0]
unit_cell = (a / n) * np.eye(3)
water = pr.create_atoms(elements=['H', 'H', 'O'],
                        positions=[r_H1, r_H2, r_O],
                        cell=unit_cell, pbc=True)
water.set_repeat([n, n, n])
water.plot3d()
```

```
NGLWidget()
```

[4]:
```
water.get_chemical_formula()
```

[4]:
```
'H54O27'
```

Equilibrate water structure

The initial water structure is obviously a poor starting point and requires equilibration (Due to the highly artificial structure a MD simulation with a standard time step of 1fs shows poor convergence). Molecular dynamics using a time step that is two orders of magnitude smaller allows us to generate an equilibrated water structure. We use the NVT ensemble for this calculation:

[5]:
```
water_potential = pandas.DataFrame({
    'Name': ['H2O_tip3p'],
    'Filename': [[]],
    'Model': ["TIP3P"],
    'Species': [['H', 'O']],
    'Config': [['# @potential_species H_O ### species in potential\n', '# W.L.␣
↪Jorgensen et.al., The Journal of Chemical Physics 79, 926 (1983); https://doi.org/
↪10.1063/1.445869\n', '#\n', '\n', 'units real\n', 'dimension 3\n', 'atom_style full\
↪n', '\n', '# create groups ###\n', 'group O type 2\n', 'group H type 1\n', '\n', '#
↪# set charges - beside manually ###\n', 'set group O charge -0.830\n', 'set group H␣
↪charge 0.415\n', '\n', '### TIP3P Potential Parameters ###\n', 'pair_style lj/cut/
↪coul/long 10.0\n', 'pair_coeff * * 0.0 0.0 \n', 'pair_coeff 2 2 0.102 3.188 \n',
↪'bond_style harmonic\n', 'bond_coeff 1 450 0.9572\n', 'angle_style harmonic\n',
↪'angle_coeff 1 55 104.52\n', 'kspace_style pppm 1.0e-5\n', '\n']]
})
```

[6]:
```
job_name = "water_slow"
ham = pr.create_job("Lammps", job_name)
ham.structure = water
ham.potential = water_potential
```

```
/srv/conda/envs/notebook/lib/python3.7/site-packages/pyiron/lammps/base.py:170:␣
↪UserWarning: WARNING: Non-'metal' units are not fully supported. Your calculation␣
↪should run OK, but results may not be saved in pyiron units.
  "WARNING: Non-'metal' units are not fully supported. Your calculation should run OK,
↪ but "
```

[7]:
```
ham.calc_md(temperature=300,
            n_ionic_steps=1e4,
            time_step=0.01)
ham.run()
```

```
The job water_slow was saved and received the ID: 1
```

[8]:
```
view = ham.animate_structure()
view
```

```
NGLWidget()
```

Full equilibration

At the end of this simulation, we have obtained a structure that approximately resembles water. Now we increase the time step to get a reasonably equilibrated structure

[9]:
```
# Get the final structure from the previous simulation
struct = ham.get_structure(iteration_step=-1)
job_name = "water_fast"
ham_eq = pr.create_job("Lammps", job_name)
ham_eq.structure = struct
```

(continues on next page)

(continued from previous page)

```
ham_eq.potential = water_potential
ham_eq.calc_md(temperature=300,
               n_ionic_steps=1e4,
               n_print=10,
               time_step=1)
ham_eq.run()
```

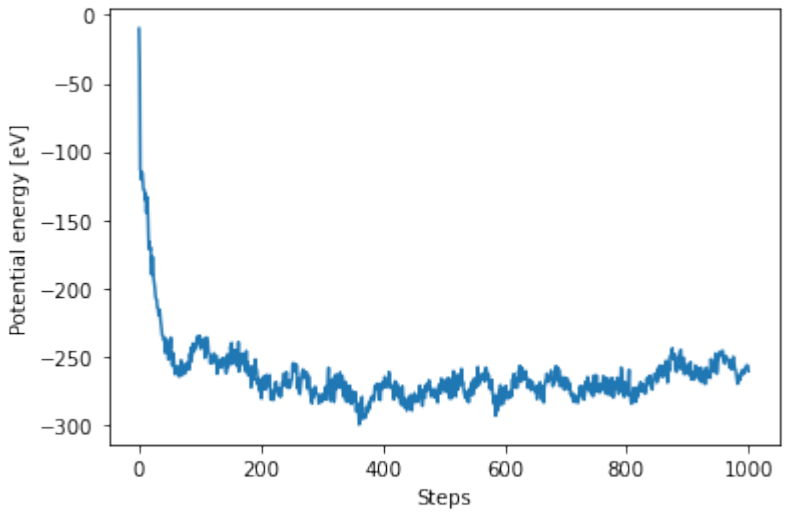
```
The job water_fast was saved and received the ID: 2
```

[10]:
```
view = ham_eq.animate_structure()
view
```

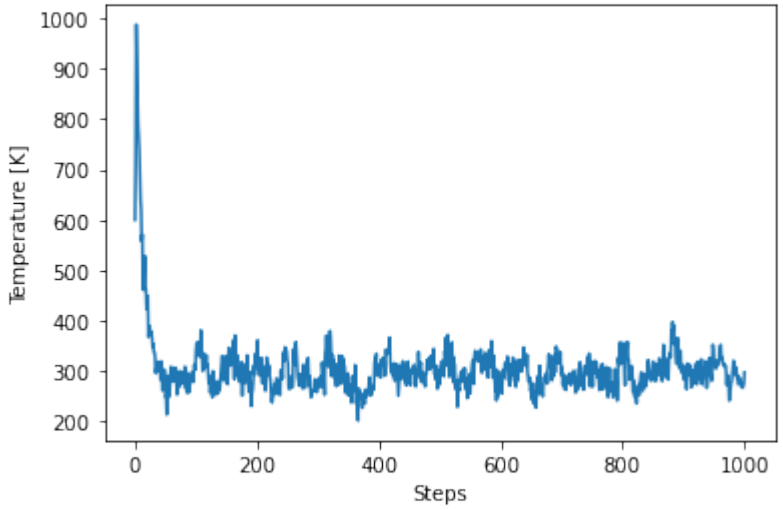
```
NGLWidget(max_frame=1000)
```

We can now plot the trajectories

[11]:
```
plt.plot(ham_eq["output/generic/energy_pot"])
plt.xlabel("Steps")
plt.ylabel("Potential energy [eV]");
```



[12]:
```
plt.plot(ham_eq["output/generic/temperature"])
plt.xlabel("Steps")
plt.ylabel("Temperature [K]");
```



Structure analysis

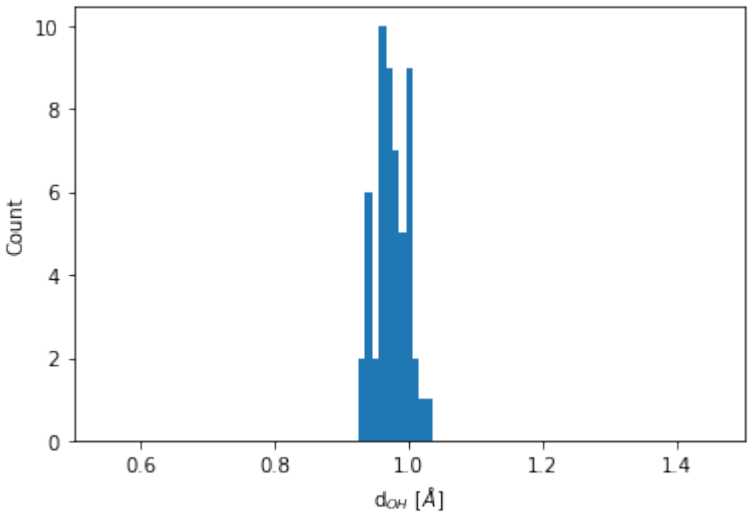
We will now use the `get_neighbors()` function to determine structural properties from the final structure of the simulation. We take advantage of the fact that the TIP3P water model is a rigid water model which means the nearest neighbors, i.e. the bound H atoms, of each O atom never change. Therefore they need to be indexed only once.

[13]:
```
final_struct = ham_eq.get_structure(iteration_step=-1)

# Get the indices based on species
O_indices = final_struct.select_index("O")
H_indices = final_struct.select_index("H")

# Getting only the first two neighbors
neighbors = final_struct.get_neighbors(num_neighbors=2)
```

Distribution of the O-H bond length

Every O atom has two H atoms as immediate neighbors. The distribution of this bond length is obtained by:

[14]:
```
bins = np.linspace(0.5, 1.5, 100)
plt.hist(neighbors.distances[O_indices].flatten(), bins=bins)
plt.xlim(0.5, 1.5)
plt.xlabel(r"d$_{OH}$ [$\AA$]")
plt.ylabel("Count");
```



Distribution of the O-O bond lengths

We need to extend the analysis to go beyond nearest neighbors. We do this by using the number of neighbors in a specified cutoff distance

[15]:
```
num_neighbors = final_struct.get_numbers_of_neighbors_in_sphere(cutoff_radius=9).max()
```

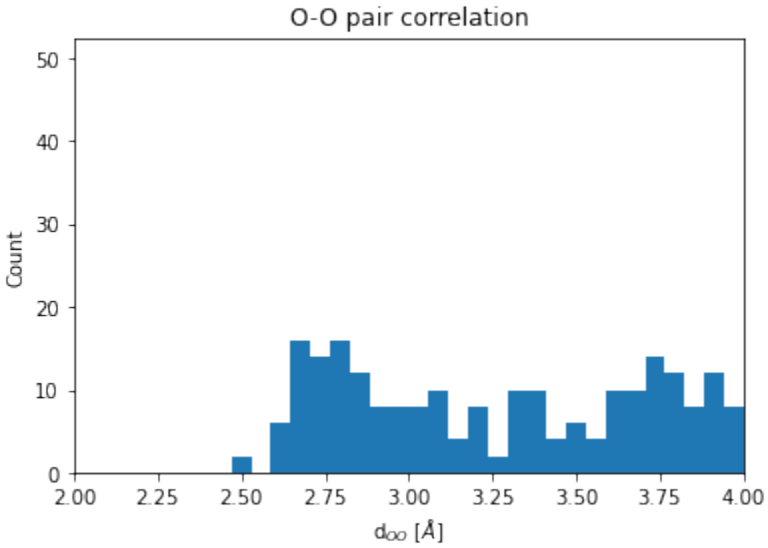
[16]:
```
neighbors = final_struct.get_neighbors(num_neighbors)
```

[17]:
```
neigh_indices = np.hstack(np.array(neighbors.indices)[O_indices])
neigh_distances = np.hstack(np.array(neighbors.distances)[O_indices])
```

One is often intended in an element specific pair correlation function. To obtain for example, the O-O coordination function, we do the following:

[18]:
```
# Getting the neighboring Oxyhen indices
O_neigh_indices  = np.in1d(neigh_indices, O_indices)
O_neigh_distances = neigh_distances[O_neigh_indices]
```

[19]:
```
bins = np.linspace(1, 8, 120)
count = plt.hist(O_neigh_distances, bins=bins)
plt.xlim(2, 4)
plt.title("O-O pair correlation")
plt.xlabel(r"d$_{OO}$ [$\AA$]")
plt.ylabel("Count");
```



We now extent our analysis to include statistically independent snapshots along the trajectory. This allows to obtain the radial pair distribution function of O-O distances in the NVT ensamble.

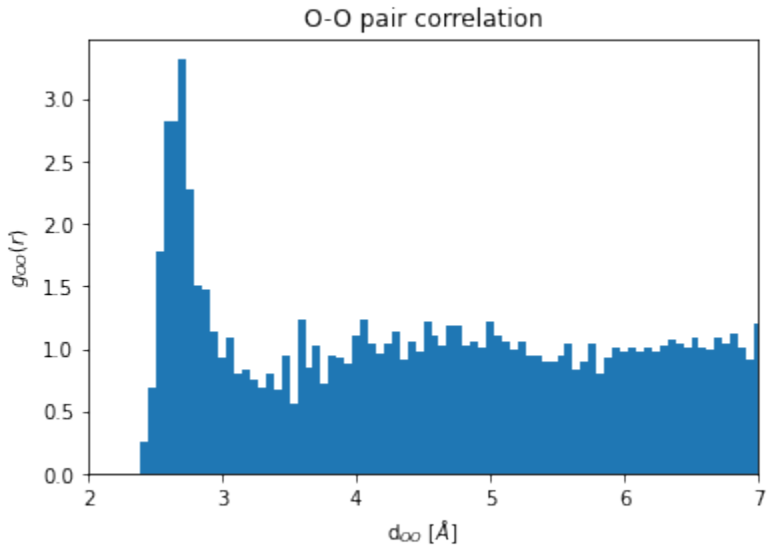
[20]:
```
traj=ham_eq["output/generic/positions"]
nsteps=len(traj)
stepincrement=int(nsteps/10)
# Start sampling snaphots after some equilibration time; do not double count last␣
↪step.
snapshots=range(stepincrement,nsteps-stepincrement,stepincrement)
```

[21]:
```
for i in snapshots:
    struct.positions = traj[i]
    neighbors = struct.get_neighbors(num_neighbors)
    neigh_indices = np.hstack(np.array(neighbors.indices)[O_indices])
    neigh_distances = np.hstack(np.array(neighbors.distances)[O_indices])
    O_neigh_indices  = np.in1d(neigh_indices, O_indices)
    #collect all distances in the same array
    O_neigh_distances = np.concatenate((O_neigh_distances,neigh_distances[O_neigh_
↪indices]))
```

To obtain a radial pair distribution function ($g(r)$), one has to normalize by the volume of the surfce increment of the sphere ($4\pi r^2 \Delta r$) and by the number of species, samples, and the number density.

[22]:
```
O_gr=np.histogram(O_neigh_distances,bins=bins)
dr=O_gr[1][1]-O_gr[1][0]
normfac=(n/a)**3*n**3*4*np.pi*dr*(len(snapshots)+1)
# (n/a)**3              number density
# n**3                  number of species
# (len(snapshots)+1)    number of samples (we also use final_struct)
```

[23]:
```
plt.bar(O_gr[1][0:-1],O_gr[0]/(normfac*O_gr[1][0:-1]**2),dr)
plt.xlim(2, 7)
plt.title("O-O pair correlation")
plt.xlabel(r"d$_{OO}$ [$\AA$]")
plt.ylabel("$g_{OO}(r)$");
```



[ ]:

3.3.8 Importing finished VASP calculations

Finished VASP calculations that were created outside of pyiron can be imported using the following script:

```
from pyiron.project import Project
pr = Project('imported_jobs')
# Searches and imports vasp jobs from 'vasp_directory'
path_to_import = "vasp_directory"
pr.import_from_path(path=path_to_import, recursive=True)
```

The calculations are imported into the project 'imported_jobs'. The recursive function imports vasp directories within each vasp directory if present.

Note: This functionality best works when both the vasprun.xml and OUTCAR files are present in the directories. The import would work only id the vasprun.xml file exists too. If the vasprun.xml file does not exist, the OUTCAR and CONTCAR files must be present

---

3.4 Command Line Interface

3.4.1 Usage Summary

There's a few command line tools shipped with pyiron to help administrating and keeping up with your pyiron project as well as some that are used internally. All of them are installed by default in the *pyiron* script that has a few sub commands.

pyiron install Installs the pyiron resources for the first time, if you don't get them via conda.

pyiron ls list the jobs inside a project and filter them with a few primitives

Print the run time of all finished jobs

pyiron ls -c job totalcputime -s finished

Print all jobs with iron

pyiron ls -e Fe

Print all jobs that successfully finished yesterday and a bit

pyiron ls -s finished -i 1d5h

Print all jobs that were aborted less than 5 hours ago and match "spx.*restart"

pyiron ls -n "spx.*restart" -i 5h -s aborted

pyiron rm Delete jobs and whole projects from the database and the file system. If you simply *rm* jobs and projects they are still in the database and can lead to confusion on pyiron's part.

pyiron wrapper Runs jobs from the database. pyiron uses this internally to start jobs on the remote cluster nodes, but you can also use it when you set the run mode to "manual" or to manually re-run jobs.

3.4.2 Developer Guide

Adding a new sub command is done by adding a new module to `pyiron.cli`. This module needs to define a `register` and a `main` function. The former is called with an `argparse.ArgumentParser` instance as sole argument and should define the command line interface in the usual way. The latter will be called with the parsed arguments and should just execute whatever it is that utility should be doing. Additionally if you need to control the `formatter_class` and `epilog` keyword arguments when creating the `argparse.ArgumentParser` instance you can set the `formatter` and `epilog` toplevel variables (see the *ls* sub command for an example). Finally you must add the module to the `pyiron.cli.cli_modules` dict.

3.5 Citing

The pyiron integrated development environment (IDE) for computational materials science - pyiron IDE - is based on a flexible plugin infrastructure. So depending on which modules are used please cite the corresponding papers.

3.5.1 pyiron paper (accepted)

```
@article{pyiron-paper,
  title = {pyiron: An integrated development environment for computational materials␣
↪science},
  journal = {Computational Materials Science},
  volume = {163},
  pages = {24 - 36},
  year = {2019},
  issn = {0927-0256},
  doi = {https://doi.org/10.1016/j.commatsci.2018.07.043},
  url = {http://www.sciencedirect.com/science/article/pii/S0927025618304786},
  author = {Jan Janssen and Sudarsan Surendralal and Yury Lysogorskiy and Mira␣
↪Todorova and Tilmann Hickel and Ralf Drautz and Jörg Neugebauer},
  keywords = {Modelling workflow, Integrated development environment, Complex␣
↪simulation protocols},
}
```

For all the other modules/ plugins in particular those hosted at https://gitlab.mpcdf.mpg.de/pyiron (MPIE internal) please ask the developers for the corrsponding references. We try to publish those under the open source license when the initial papers are published. Afterwards they are going to be added to the official Github repository.

3.5.2 external paper

Some of the features in pyiron rely on external codes which should be cited separatly. In alphabetical order:

ASE

pyiron is compatible with the Atomic Simulation Environment (ASE) structure classes, allowing the user to generate structures using the ASE framework and run the simulation within pyiron.

```
@article{ase-paper,
  author={Ask Hjorth Larsen and Jens Jørgen Mortensen and Jakob Blomqvist and Ivano E␣
↪Castelli and Rune Christensen and Marcin Dułak and Jesper Friis and Michael N␣
↪Groves and Bjørk Hammer and Cory Hargus and Eric D Hermes and Paul C Jennings and␣
↪Peter Bjerre Jensen and James Kermode and John R Kitchin and Esben Leonhard␣
↪Kolsbjerg and Joseph Kubal and Kristen Kaasbjerg and Steen Lysgaard and Jón␣
↪Bergmann Maronsson and Tristan Maxson and Thomas Olsen and Lars Pastewka and Andrew␣
↪Peterson and Carsten Rostgaard and Jakob Schiøtz and Ole Schütt and Mikkel Strange␣
↪and Kristian S Thygesen and Tejs Vegge and Lasse Vilhelmsen and Michael Walter and␣
↪Zhenhua Zeng and Karsten W Jacobsen},
  title={The atomic simulation environment--a Python library for working with atoms},
  journal={Journal of Physics: Condensed Matter},
  volume={29},
  number={27},
  pages={273002},
  url={http://stacks.iop.org/0953-8984/29/i=27/a=273002},
  year={2017}
}
```

LAMMPS

The LAMMPS molecular dynamics simulator is the default molecular dynamics code used by pyiron.

```
@article{lammps,
  title = {Fast Parallel Algorithms for Short-Range Molecular Dynamics},
  journal = {Journal of Computational Physics},
  volume = {117},
  number = {1},
  pages = {1-19},
  year = {1995},
  issn = {0021-9991},
  doi = {https://doi.org/10.1006/jcph.1995.1039},
  url = {http://www.sciencedirect.com/science/article/pii/S002199918571039X},
  author = {Steve Plimpton}
}
```

VASP

The Vienna Ab initio Simulation Package is the default ab initio used by pyiron.

```
@article{Kresse1993,
  title = {Ab initio molecular dynamics for liquid metals},
  author = {Kresse, G. and Hafner, J.},
  journal = {Phys. Rev. B},
  volume = {47},
  issue = {1},
  pages = {558--561},
  numpages = {0},
  month = {Jan},
  publisher = {American Physical Society},
  doi = {10.1103/PhysRevB.47.558},
  url = {https://link.aps.org/doi/10.1103/PhysRevB.47.558}
}
```

```
@article{Kresse1996a,
  title = {Efficiency of ab-initio total energy calculations for metals and␣
↪semiconductors using a plane-wave basis set},
  journal = {Computational Materials Science},
  volume = {6},
  number = {1},
  pages = {15-50},
  year = {1996},
  issn = {0927-0256},
  doi = {https://doi.org/10.1016/0927-0256(96)00008-0},
  url = {http://www.sciencedirect.com/science/article/pii/0927025696000080},
  author = {Kresse, G. and Furthm\"uller, J.}
}
```

```
@article{Kresse1996b,
  title = {Efficient iterative schemes for ab initio total-energy calculations using␣
↪a plane-wave basis set},
  author = {Kresse, G. and Furthm\"uller, J.},
  journal = {Phys. Rev. B},
  volume = {54},
  issue = {16},
  pages = {11169--11186},
  numpages = {0},
  year = {1996},
  month = {Oct},
  publisher = {American Physical Society},
  doi = {10.1103/PhysRevB.54.11169},
  url = {https://link.aps.org/doi/10.1103/PhysRevB.54.11169}
}
```

3.6 FAQ

3.6.1 How to cite pyiron?

To cite pyiron and the corresponding codes, please follow the instructions on the publication page.

3.6.2 What units does pyiron use?

- mass = atomic mass units
- distance = Angstroms
- time = femtoseconds
- energy = eV
- velocity = Angstroms/femtoseconds
- force = eV/Angstrom
- temperature = Kelvin
- pressure = GPa
- charge = multiple of electron charge (1.0 is a proton)

3.6.3 How to import existing calculation?

Importing existing calculations is currently only supported for VASP. A tutorial how to import existing calculations is available in the tutorial section.

3.6.4 How to import structures from files or existing databases?

To read structure formats you can use ASE and then convert the structure to a pyiron structure using:

```
from pyiron import ase_to_pyiron
pyiron_structure = ase_to_pyiron(ase_structure)
```

3.6.5 How to install pyiron?

pyiron is designed to be installed as centralized service on your local computer cluster, rather than a local installation on each individual workstation. To test pyiron online or with a local installation, please follow the instructions on the installation page.

3.6.6 How do I install additional codes for pyiron?

When installing pyiron via conda it is possible to install most opensource codes via conda as well:

Table 1: Install additional codes

code	job_type	How to install ?
GAUSSIAN	Gaussian	Compile on your own (commercial code)
Gpaw	Gpaw	*conda install -c conda-forge gpaw*
LAMMPS	Lammps	*conda install -c conda-forge lammps*
S/PHI/nX	Sphinx	*conda install -c conda-forge sphinxdft*
sqsgenerator	SQSJob	*conda install -c conda-forge sqsgenerator*
VASP	Vasp	Compile on your own (commercial code)

3.6.7 How to use a custom Pseudo potential in VASP?

There are two ways to assign custom potentials in VASP, either you can change the pseudo potential for all atoms of one species:

```
job_vasp.potential.Fe = "~/resources/vasp/potentials/potpaw_PBE/Fe/POTCAR"
```

Or alternatively you can change the pseudo potential of a single atom by creating a new element:

```
my_fe = pr.create_element(
    new_element_name="Fe",
    parent_element="Fe",
    potential_file="~/resources/vasp/potentials/potpaw_PBE/Fe/POTCAR"
)
job_vasp.structure[0] = my_fe
```

3.6.8 How to use VASP tags which are not supported by pyiron?

The underlying input of any simulation code in pyiron can be directly accessed. For VASP you can change the INCAR parameters using the VASP specific syntax:

```
job_vasp.input.incar["ENCUT"] = 320.0  # eV
```

3.6.9 How to use a custom potential in LAMMPS?

A custom empirical potential (here, a hybrid potential) can be defined in the following format:

```
custom_potential = pd.DataFrame({
  'Name': ['SrTiO3_Pedone'],
  'Filename': [[]],
  'Model': ['Custom'],
  'Species': [['O', 'Sr', 'Ti']],
  'Config': [['atom_style full\n',  # I use 'full' here as atom_style 'charge' gives␣
↪the same result
              '## create groups ###\n',
              'group O type 1\n',
              'group Sr type 2\n',
              'group Ti type 3\n',
```

(continues on next page)

(continued from previous page)

```
                '\n',
                '## set charges - beside manually ###\n',
                'set group O charge -1.2000\n',
                'set group Sr charge 1.2000\n',
                'set group Ti charge 2.4000\n',
                '\n',
                'pair_style hybrid/overlay morse 15.0 mie/cut 15.0 coul/long 15.0 beck␣
↪15.0\n',
                'pair_coeff * * coul/long\n',
                'pair_coeff 1 2 beck 3.0 0 0 0 0\n',
                'pair_coeff 1 3 beck 1.0 0 0 0 0\n',
                'pair_coeff 1 1 beck 22.0 0 0 0 0\n',
                'pair_coeff 1 2 mie/cut 3.0 1.0 12.0 0\n',
                'pair_coeff 1 3 mie/cut 1.0 1.0 12.0 0\n',
                'pair_coeff 1 1 mie/cut 22.0 1.0 12.0 0\n',
                'pair_coeff 1 2 morse 0.019623 1.8860 3.32833\n',
                'pair_coeff 1 3 morse 0.024235 2.2547 2.708943\n',
                'pair_coeff 1 1 morse 0.042395 1.3793 3.618701\n',
                'kspace_style ewald 1.0e-8\n']]
})
```

The lines in `Config` will be written to the LAMMPS `potential.inp` file. Make sure that the arrangement of the species in `Species` is the same as the group types `create groups` within `Config`. Otherwise, a mixup or the species may occur in the LAMMPS `structure.inp` file.

The potential can then be used by assigning `job.potential = custom_potential`.

3.6.10 How to extend the potential database inside pyiron?

By default pyiron provides access to the OpenKIM and NIST databases for interatomic potentials and individual potentials can be added as discussed above. While there was an option to extend the default database this option was disabled as it decreased the reproducibility of simulation protocols.

3.6.11 How to link your own executable?

The linking of executables is explained as part of the installation in the section of advanced configuarion options. By default pyiron links to the executables provided by conda but you can accelerate you calculation by compiling your own version of a given simulation code which is optimized for your hardware.

3.6.12 How to send a calculation to the background ?

While most examples execute calculations inline or in modal mode, it is also possible to send calculation in the background.

```
job.server.run_mode.non_modal = True
job.run()
print("execute other commands while the job is running.")
pr.wait_for_job(job)
```

In this example the job is executed in the background, while the print command is already executed. Afterwards the project object waits for the execution of the job in the background to be finished.

3.6.13 How to submit a calculation to the queuing system?

Just like executing calculation in the background it is also possible to submit calculation to the queuing system:

```
job.server.list_queues()  # returns a list of queues available on the system
job.server.view_queues()  # returns a DataFrame listing queues and their settings
job.server.queue = "my_queue"  # select a queue
job.server.cores = 80          # set the number of cores
job.server.run_time = 3600     # set the run time in seconds
job.run()
```

For the queuing system to be available in pyiron it is necessary to configure it. The configuration of different queuing systems is explained in the installation.

3.6.14 How to setup spin constraint calculation?

pyiron supports setting up spin constrained calculations for VASP using the generic spin_constraint property:

```
job_vasp.spin_constraints = 1
```

3.6.15 What is the meaning of the name - pyiron?

pyiron is the combination of **py** + **iron** connecting Python, the programming language with iron as pyiron was initially developed at the Max Planck Institut für Eisenforschung (iron research).

3.6.16 Which output quantities are stored in pyiron?

generic					
tag	dimension	description	VASP	SPHInX	LAMMPS
time	N_{step}	simulation time (fs)			x
steps	N_{step}	time steps			x
unwrapped_positions	N_{step} x N_{atom} x 3	unwrapped atom coordinates ( )	x	x	x
positions	N_{step} x N_{atom} x 3	wrapped atom coordinates ( )	x	x	x
velocities	N_{step} x N_{atom} x 3	velocity of each atom (/fs)			
forces	N_{step} x N_{atom} x 3	force on each atom (eV/)	x	x	x
cells	N_{step} x 3 x 3	cell dimensions (cf. VASP website) ( )	x	x	x
energy_tot	N_{step}	total energy of the system (eV)	x	x	x
energy_kin	N_{step}	kinetic energy of the system (eV)	x		
energy_pot	N_{step}	potential energy of the system (eV)	x		
pressures	N_{step} x 3 x 3	pressures (GPa)			x
temperature	N_{step}	temperature (K)	x		x
volume	N_{step} ?	supercell volume (^{3})	x	x	x
atom_voronoi	N_{step} x N_{atom}	Voronoi volume of each atom (^{3})			
atom_stress	N_{step} x N_{atom} x 3 x 3	stress per atom x atomic volume (eV)			x
atom_centro	N_{step} x N_{atom}	centro-symmetry parameter (^{2})			
atom_displace	N_{step} x N_{atom} x 3	displacement of each atom with respect to the initial position ( )			
computation_time	N_{step}	computation time of the simulation (s)		x	

dft					
tag	dimension	description	VASP	SPHInX	LAMMPS
(scf_)energy_int	N_{step}	internal energy (eV)		x	
(scf_)energy_free	N_{step}	free energy, same as energy_tot in generic (eV)	x	x	
(scf_)energy_zero	N_{step}	extrapolated energy, sigma 0 (eV)	x	x	
(scf_)energy_band	N_{step}	band gap energy (eV)		x	
(scf_)residue	N_{step} (x 2)	energy residue (eV)		x	
atoms_(scf_)spins	N_{step} x N_{atom}	spin moment of each atom (Bohr magneton)		x	
(scf_)magnetic_forces	N_{step} x N_{atom}	spin forces ? (eV/Bohr magneton)		x	
atom_spin_constraints	N_{step} x N_{atom}	spin constraints (Bohr magneton)		x	
bands_e_fermi	N_{step}	fermi energy (eV)		x	
bands_occ	N_{step} (x 2) x N_k x N_{states}	occupancy		x	
bands_k_weights	N_k	weight of each k point		x	
bands_eigen_values	N_{step} (x 2) x N_k x N_{states}	eigenspectrums (eV)		x	
scf_convergence	N_{step}	convergence of each ionic step		x	

- N_{step} refers to ionic steps and not electronic steps

- properties preceded by scf_ contain the values of each electronic step except for scf_convergence
- (x 2) refers to the additional column which appears only in magnetic calculations
- if the crosses under VASP, SPHInX or LAMMPS are missing, the corresponding properties are not implemented

3.7 Contributing to pyiron

The following is a set of guidelines for contributing to pyiron, which is hosted and maintained by the Max Planck Institut für Eisenforschung on GitHub. These are mostly guidelines to facilitate an efficient development workflow, and not necessarily rules. Use your best judgment, and feel free to propose changes even to this document in a pull request.

You can find all the pyiron packages at our github page . To create pull requests, you will need to become part of the pyiron organization. Please email us if you would like to join.

3.7.1 Wait I don't want to read this; I just have a quick question/bugfix!

1. Check out our FAQ page; your question might already be answered there.
2. If your question relates to a bug in pyiron, please briefly search the issues page and open a new labeled issue if you don't see anything related to your question there.
3. Please feel free just to send one of us a brief, descriptive email with your question, and we'll do our best to get back to you as ASAP as possible.

3.7.2 Table of Contents

- *My job does not run on the queue*

3.7.3 License

pyiron is released as an open-source project under the BSD 3-Clause License. Code contributions should also be considered open-source.

3.7.4 What should I know before I get started?

pyiron developer meetings

If you are interested in discussing pyiron's development, we encourage you to virtually participate in the weekly pyiron developer meeting at 14:00 german time (GMT+2). Check the discussion page for details.

3.7.5 How can I contribute?

Reporting bugs

> Note: If you find a closed issue that seems like it is the same thing that you're experiencing, open a new issue and include a link to the original issue in the body of your new one.

Before Submitting A Bug Report

Check if you can reproduce the problem in the latest version of pyiron. Check the FAQ page for a list of common questions and problems. Briefly search the issues page for bugs to see if the problem has already been reported. If it has and the issue is still open, add a comment to the existing issue instead of opening a new one.

How Do I Submit A (Good) Bug Report?

Bugs are tracked as GitHub issues. You can create an issue on the pyiron repository by including the following information:

- Use a clear and descriptive title for the issue to identify the problem.
- Describe the exact steps you took so we can reproduce the problem as closely as possible.
- Provide sample code that causes the problem. Include code snippets as markdown code blocks.
- Include information about the environment (OS, python version, how packages were installed) in which you were running pyiron.
- Explain what you expected to happen, and what happened instead.

Suggesting Enhancements

How Do I Submit A (Good) Enhancement Suggestion?

Enhancement suggestions are tracked as GitHub issues. You can create an issue on the pyiron repository by including the following information:

- Use a clear and descriptive title for the issue to identify the suggestion.
- Describe the exact behavior you would expect the suggested feature to produce.
- Provide sample code that you would use to access the feature. If possible, include code for how you think the feature could be built into pyiron's codebase. Include code snippets as markdown code blocks.

Your first code contribution

Unsure where to begin contributing to pyiron? You can start by looking through these good-first-issue and help-wanted issues:

- Good first issues - issues which should only require a few lines of code, and a test or two.
- Help wanted issues - issues which should be a bit more involved than beginner issues.

Local development

pyiron can be developed and tested locally. If you are using pyiron to run an external software package, e.g. VASP or LAMMPS, you might also need to install those packages locally to run certain integration tests in pyiron.

To get the developmental (git) version of pyiron,

```
git clone https://github.com/pyiron/pyiron.git
conda env update --name pyiron_dev --file pyiron/.ci_support/environment.yml
conda activate pyiron_dev
conda install conda-build
conda develop pyiron
```

Deploy development version to a managed environment

If you want to use a development version of pyiron in a managed environment where a version of pyiron is already installed outside of your control (e.g. on the cmti/cmfe cluster), you can still preload a local checkout of the repo, while using the dependencies already installed. Assuming pyiron and dependencies are already installed and setup, clone the repository to a location of your choice

```
mkdir -p ~/software
cd ~/software
git clone https://github.com/pyiron/pyiron.git
```

add this folder to your python path by adding this line to your *~/.profile*

```
export PYTHONPATH="$HOME/software/pyiron:$PYTHONPATH"
```

and finally restart any jupyter or jupyterhub session you might still have running. Within this folder you can then check out any local branchen, push your own dev branches, etc and python will automatically use this version over the system-wide installation. Check that it works by running the following cell

```
import pyiron
print(pyiron.__file__)
```

If it doesn't print the path of your checkout, check that you restarted all the relevant shell sessions and that the environment variables are correctly updated.

Local Testing

The full test suite is always run automatically when you open a new pull request. Still it sometimes nice to run all or only specific tests on your machine. To do that run from the repo root, e.g.

```
python -m unittest discover tests
python -m unittest discover tests/sphinx
python -m unittest tests/sphinx/test_base.py
```

Where the first line runs all tests, the second all the sphinx tests and the final line only the tests in that file. Keep in mind that to run the tests your repository needs to be inside your pyiron project folder and you need to have at least the basic resources installed from `tests/static`. A neat trick when testing/debugging is to combine the pdb and unittest modules like this

```
python -m pdb -m unittest ...
```

This allows you to re-use the sometimes complicated setups for your interactive debugging that might be otherwise difficult to replicate in a REPL.

Pull requests

The process described here has several goals:

- Maintain pyiron's quality
- Fix problems that are important to users
- Engage the community in working toward the best possible tools
- Enable a sustainable system for pyiron's maintainers to review contributions

Please follow these steps to have your contribution considered by the maintainers:

- Keep the changes in your pull request as focused as possible- only address one issue per pull request wherever possible.
- Follow the *Styleguides*
- Assign the appropriate label (see *Issue and pull request labels*) to your pull request. If you are fixing a specific Github issue, reference the issue directly in the pull request comments.
- If you are aware which maintainer is most closely related to the code you've edited, feel free to request their review.
- After you submit your pull request, verify that all status checks are passing.
- If a status check fails and it seems to be unrelated to your changes, explain why the failure is unrelated as a comment in your pull request.

While the prerequisites above must be satisfied prior to having your pull request reviewed, the reviewer(s) may ask you to complete additional design work, tests, or other changes before your pull request can be ultimately accepted.

3.7.6 Styleguides

Git commit messages

- Use the present tense ("Add feature" not "Added feature")
- Use the imperative mood ("Move cursor to…" not "Moves cursor to…")
- Limit the first line to 72 characters or less
- Reference issues and pull requests liberally after the first line
- When only changing documentation, include [ci skip] in the commit title
- Consider starting the commit message with an applicable emoji:

:art: (`:art:`) improves the format/structure of the code

:zap: (`:zap:`) improves performance

:memo: (`:memo:`) adds documentation

:bug: (`:bug:`) fixes a bug

:fire: (`:fire:`) removes code or files

:green_heart: (`:green_heart:`) fixes the CI build

:white_check_mark: (`:white_check_mark:`) adds tests

Managing git commits is much easier using an IDE (we recommend PyCharm).

Python styleguide

Please follow PEP8 conventions for all python code added to pyiron. Pull requests will be checked for PEP8 plus a few other security issues with Codacy, and will be rejected if they do not meet the specified formatting criteria.

Any new features should include coverage with a unit test, such that your pull request does not decrease pyiron's overall coverage. This will be automatically tested within the ci test suite and Coveralls.

Deprecation warning template

XXX is deprecated as of vers. A.B.C. It is not guaranteed to be in service in vers. D.E.F. Use YYY instead.

Documentation styleguide

All new/modified functions should include a docstring that follows the Google Python Docstring format.

Documentation is built automatically with Sphinx; any manually created documentation should be added as a restructured text (.rst) file under pyiron/docs/source.

Notebooks created to exemplify features in pyiron are very useful, and can even be used as integration tests. If you have added a major feature, consider creating a notebook to show its usage under pyiron/notebooks/. See the other examples that are already there.

3.7.7 Additional notes

Issue and pull request labels

We use the following tags to organize pyiron Github issues and pull requests:

- bug: something isn't working
- duplicate: this issue/pull request already existed
- enhancement: new feature or request
- good first issue: easy fix for beginners
- help wanted: extra attention is needed
- invalid: this doesn't seem right
- question: further information is requested
- wontfix: this will not be worked on
- stale: inactive after 2 weeks

Build status

The build status for pyiron and all sub packages are given below

pyiron releases

For the pyiron release management we use git tags:

```
https://git-scm.com/book/en/v2/Git-Basics-Tagging
```

The tag format consists of a tag_prefix (<package name>-) and the release version, for example:

```
pyiron-0.2.0
```

For the automated versioning we use:

```
https://github.com/warner/python-versioneer/
```

So the configuration of the release is included in setup.cfg:

```
https://github.com/pyiron/pyiron_base/blob/master/setup.cfg
```

As the pyiron packages are pure python packages – we use only the Linux Python 3.7 job to build the packages, as defined in the .travis.yml file:

```
https://github.com/pyiron/pyiron_base/blob/master/.travis.yml
```

The python 3.7 linux tests therefore takes more time, compared to the other tests on travis.

Just like each other commit to the master branch the tagged releases are pushed to pypi.org and anaconda.org:

```
https://pypi.org/project/pyiron-base/#history
https://anaconda.org/pyiron/pyiron_base
```

The major difference for pypi (pip) is that tagged releases are the default for pip while installing prerelease versions using pip requires the *–pre* flag. *pip install –pre pyiron*

Those pre-release versions are named <*version_number*>*.post0.dev*<*release number*>

```
0.2.0.post0.dev1
```

For anaconda the prereleases are pushed to the pyiron channel and can be installed using: conda install -c pyiron pyiron

On the other hand the tagged releases are available through conda-forge, as soon as the corresponding packages are merged:

```
https://github.com/conda-forge/pyiron-feedstock
conda install -c conda-forge pyiron
```

So for both conda and pip both the prereleases as well as the official releases are available.

3.7.8 Debugging

My job does not run on the queue

In case a job runs properly while executing it locally (or on the head node), but not when you submit it to a queue,

1. Check if the job class is available in the project:

In this example, we want a custom job class `ProtoMD` from the module `pyiron_contrib`:

```
from pyiron import Project
import pyiron_contrib  # only if importing a custom job class

pr = Project("debug")
dir(pr.job_type)
```

This should output:

```
>>> ['AtomisticExampleJob',
 'Atoms',
 'ConvEncutParallel',
 ...
 ...
 'ProtoMD']
```

If you see your job class in the list, proceed to step 3. If not,

2. Check if the job class in initialized in ``__init__.py`` of the module

Make sure that the `__init__.py` of your module (here, `pyiron_contrib`) initializes the job class in the following format:

```
from pyiron import Project
from pyiron.base.job.jobtype import JOB_CLASS_DICT

# Make classes available for new pyiron version
JOB_CLASS_DICT['ProtoMD'] = 'pyiron_contrib.protocol.compound.md'  # the path of your␣
↪job class
```

3. Confirm that the job class can be instantiatied

Create a new job, but instead of running it, save it:

```
job = pr.create_job(job_type = pr.job_type.ProtoMD, job_name = 'job')
...  # input parameters that the job requires
...
job.save()

>>> 98  # this is the job id of the saved job
```

Note down the job id, then run the following line:

```
job["TYPE"]
```

This should output an instance of the job class:

```
>>> "<class 'pyiron_contrib.protocol.compound.md.ProtoMD'>"
```

Now we know that the job class is indeed available in the project and can be instantiated.

4. Debug using a second notebook

Submitting and running a job on the queue, is essentially the same as saving a job in one notebook, but loading and executing it in another notebook.

In **a new notebook** , load the job that you just saved, using its job id. You may or may not import the module (here, `pyiron_conntirb`):

```
from pyiron import Project
# we do not import pyiron_contrib here, becasue it should not be necessary

pr = Project("second_notebook")
reloaded_job = pr.load(98)  # 98 is the job id of the previously saved job
reloaded_job.run(run_again=True)
```

If the job loads and runs properly, the job should also run properly on the queue. This also means that there may be a bug in your custom job class. Debug the job class, and repeat steps 3 and 4 till you no longer get an error in step 4.